

Lab Reference Guide

lab-reference-guide-2019.1-wkb-rev1-prelim

Lab Reference Guide 2019.1



© Copyright 2019 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, UltraScale, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. Cortex is a registered trademark of ARM in the EU and other countries. All other trademarks are the property of their respective owners.

DISCLAIMER

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>. All other trademarks are the property of their respective owners.

Table of Contents

Lab Reference Guide	3
Preliminary Version	3
Introduction	3
Understanding the Lab Environment	3
Vivado Design Suite Operations [Last Updated Version 2019.1]	5
Vivado Analyzer Operations [Last Updated Version 2017.1]	144
Vivado HLS Operations [Last Updated Version 2019.1]	158
System Generator Tool Operations [Last Updated Version 2018.3].....	180
SDK Tool Operations [Last Updated Version 2019.1]	181
QEMU Operations [Last Updated Version 2019.1]	293
SDx Tool Operations [Last Updated Version 2018.3].....	302
PetaLinux Operations [Last Updated Version 2019.1]	353
Board, OS, COM, and IP Address Tasks [Last Updated Version 2019.1]	363

Lab Reference Guide

Preliminary Version

Note: At the time of the release of the course from which you obtained this guide, updates were still being made to this *Lab Reference Guide*. As such, this guide is in a preliminary state.

For the final 2019.1 version, go to www.xilinx.com/downloads.htm.

Introduction

This *Lab Reference Guide* is a collection of *how to* topics for commonly performed tasks in FPGA and embedded development.

The guide is divided into the following sections:

- Vivado Design Suite Operations
- Vivado Analyzer Operations
- Vivado HLS Operations
- System Generator Tool Operations
- SDK Tool Operations
- QEMU Operations
- SDSoc Tool Operations
- PetaLinux Operations
- Board, OS, COM, and IP Address Tasks

Each section may contain sub-sections.

Understanding the Lab Environment

The labs and demos provided in this course are designed to run under the provided Linux platform. The provided Linux platform has a customized lab environment. These customizations have been made to simplify and enhance your learning experience. Many labs and demos can be successfully executed in the Windows environment as well. Topics including PetaLinux, Yocto, and QEMU are generally not supported under Windows.

The instructions, for the sake of brevity, are expressed using the Linux notation. This includes the forward slash ('/') as the hierarchy separator instead of the Windows backslash ('\'). For students who want to run under Windows, it is their responsibility to use the correct hierarchy separator.

Windows users can run Linux using the VirtualBox tool from Oracle. Details on how to acquire and install this tool can be found in the lab setup guide. The Xilinx Customer Education group offers a preconfigured virtual machine image and includes all the tools necessary to run the labs. Some tools and IP do require licenses, which are the responsibility of the student.

Several environment variables, described in detail in the lab setup guide, which also discusses how you can customize the environment for your situation, are provided for ease of use. Most notably, many environment variables have been defined to reduce the amount of typing you need to do when entering paths.

These environment variables are:

- XILINX_PATH - points to the installation directory for the Xilinx tools. Tools such as the Vivado Design Suite and SDK can be found here.
- PETALINUX_PATH - points to the installation directory for the Xilinx PetaLinux tools. Often this is within XILINX_PATH, but some choose to install it elsewhere.
- TRAINING_PATH - points to the space allocated for students to work through their labs. This directory includes prebuilt images and starting points for the labs and demos.
- <topic name> - points to the specific name of the topic within the TRAINING_PATH directory. For example, \$TRAINING_PATH/Open_AMP is shortened to just \$Open_AMP.

This lab uses \$<topic name> to indicate the directory for this lab, which is equivalent to \$TRAINING_PATH/<topic name>.

These environment variables work well in the Vivado Design Suite and scripts, but not in Eclipse-based tools such as SDK and the SDx™ environment. When using the Vivado tools, you can use \$<topic name> and the tools will automatically expand the \$<topic name> environment variable for you. However, when you see the notation \$<topic name>/workspace in SDK or the SDx environment, you should interpret it in its expanded form as /home/xilinx/training/<topic name>/workspace.

This lab assumes that you are starting with a Linux environment. If you are working on a Windows system, refer to the lab setup guide to see how to use the Oracle VirtualBox application to run the Xilinx Customer Education pre-configured Ubuntu Linux virtual machine.

Vivado Design Suite Operations [Last Updated Version 2019.1]

In This Section

Creating and Opening Operations	5
Vivado Add Sources Operations	26
Embedded Operations	34
Vivado IP Integrator Operations.....	43
Vivado Elaborated Design Operations.....	95
Vivado Simulation Operations	96
Vivado Synthesis Operations	99
Vivado Implementation Operations.....	107
Vivado Tcl Operations	116
Vivado Timing Miscellaneous Operations	124
Vivado Hardware Session Operations.....	131

Creating and Opening Operations

In This Section

Launching the Vivado Design Suite	6
Launching the Vivado Design Suite Using a Linux Terminal	7
Launching the Vivado Design Suite Tcl Shell	8
Creating a New Vivado Design Suite Project with Sources.....	8
Creating a Blank Vivado Design Suite Project.....	16
Opening a Vivado Design Suite Project	18
Opening Source Code in the Editor	19
Creating an HDL Source File	20
Opening a File in the Editor	22
Importing IP	23
Closing the Vivado Design Suite Project.....	24
Closing the Vivado Design Suite	25

Launching the Vivado Design Suite

There are a number of ways to launch the Vivado Design Suite. The two most popular mechanisms are shown here.

1-1. This can be done in two standard ways, use your preferred method.

- 1-1-1. [Windows 7 users]: Select **Start > All Programs > Xilinx Design Tools > Vivado 2019.1 > Vivado 2019.1**.

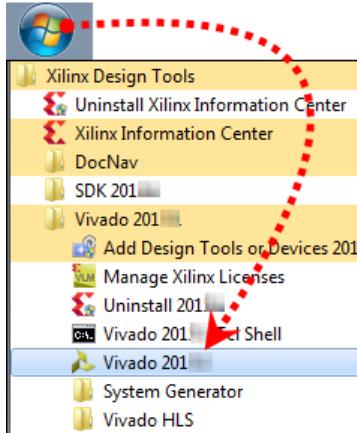


Figure 1: Launching the Vivado Design Suite from the Start Menu

- [Windows 10 users]: Select **Start > Xilinx Design Tools > Vivado 2019.1**.

You can double-click the **Vivado Design Suite** shortcut icon () from the Windows or Linux desktop or taskbar.

[Linux users]: If your system has not been configured with a shortcut on the desktop or taskbar, refer to the "Launching the Vivado IDE using a Linux Terminal" section in the *Lab Reference Guide*.

The Vivado Design Suite opens to the Welcome window. From the Welcome window you can create a new project, open an existing project, or enter Tcl commands directly into the Vivado Design Suite as well as access documentation and examples.

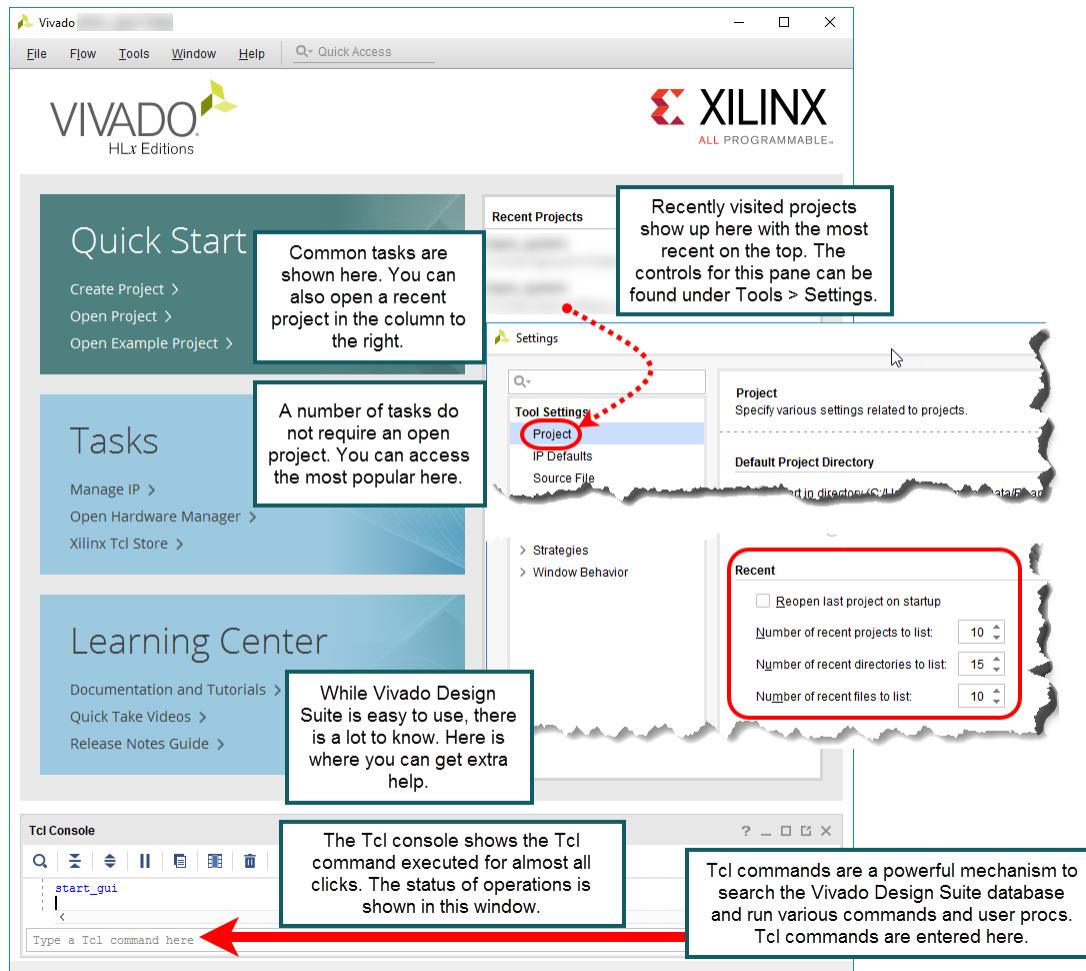


Figure 2: Vivado Design Suite Welcome Screen

Launching the Vivado Design Suite Using a Linux Terminal

- 1-1. [Linux users]: If your system has not been configured with a shortcut on the desktop or taskbar, then proceed with setting up and launching the Vivado Design Suite environment.**

- 1-1-1.** Press <**Ctrl + Alt + T**> to open a new terminal window.

- 1-1-2.** Change the directory to the topic cluster path:

```
[host] $ cd $<the topic cluster name>
```

- 1-1-3.** Enter the following command to set up the Vivado Design Suite environment appropriately:

```
[host] $ source $XILINX_PATH/Vivado/2019.1/settings.sh
```

- 1-1-4.** Type **vivado** and press <**Enter**> to launch the Vivado Design Suite environment.

Launching the Vivado Design Suite Tcl Shell

- 1-1. [Windows users]: Launch the Vivado Design Suite Tcl shell.**

- 1-1-1. [Windows 7 users]: Select Start > All Programs > Xilinx Design Tools > Vivado 2019.1 > Vivado 2019.1 Tcl Shell.**

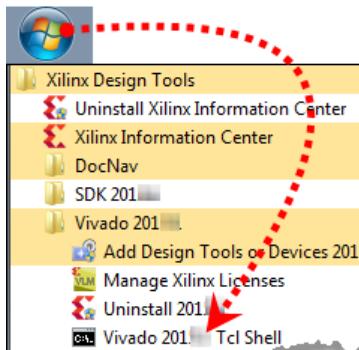


Figure 3: Vivado Design Suite Tcl Shell

[Windows 10 users]: Select Start > Xilinx Design Tools > Vivado 2019.1 Tcl Shell.

Creating a New Vivado Design Suite Project with Sources

Projects begin with the creation of a new project. The project contains sources, settings, graphics, IP, and other elements that are used to build a final bitstream.

1-1. Create a new Vivado Design Suite project.

1-1-1. Click **Create New Project** (1).

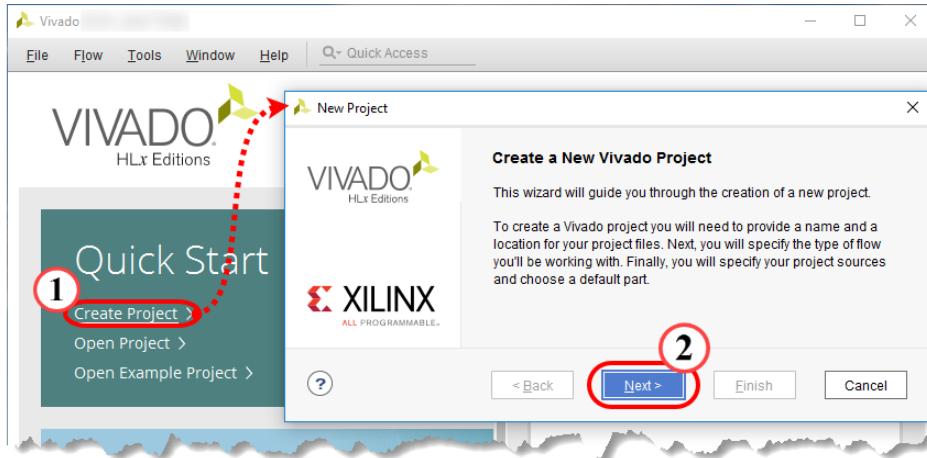


Figure 4: Creating a New Vivado Design Suite Project

This will launch the New Project Wizard.

1-1-2. Click **Next** to begin entering the specifics for this project (2).

1-2. You will now encounter a series of dialog boxes asking you to enter different pieces of information describing the project.

1-2-1. Enter **your project name** in the Project name field.

1-2-2. Enter **C:\training\<the topic cluster name>\lab\your board\<language> or your project name** in the Project location field.

Alternatively, you can use the browse feature to navigate to where you want the project to reside.

- 1-2-3.** If you are performing one of the Xilinx labs, deselect the **Create Project Subdirectory** option as leaving this checked will create an unnecessary level of hierarchy for the lab.



Figure 5: Entering the Project Name and Location

- 1-2-4.** Click **Next** to accept the selections and advance to selecting a type of project.

The Project Type dialog box invites you to choose between an RTL project or a post-synthesis project. Simply put, an RTL project enables you to add or create new HDL files and synthesize them, whereas a post-synthesis project requires pre-synthesized files.

- 1-2-5.** Select **RTL Project** since this project will be based on source code (rather than a netlist) (1).
- 1-2-6.** Since you will be adding RTL files in the next instruction, deselect the **Do not specify sources at this time** option (2).

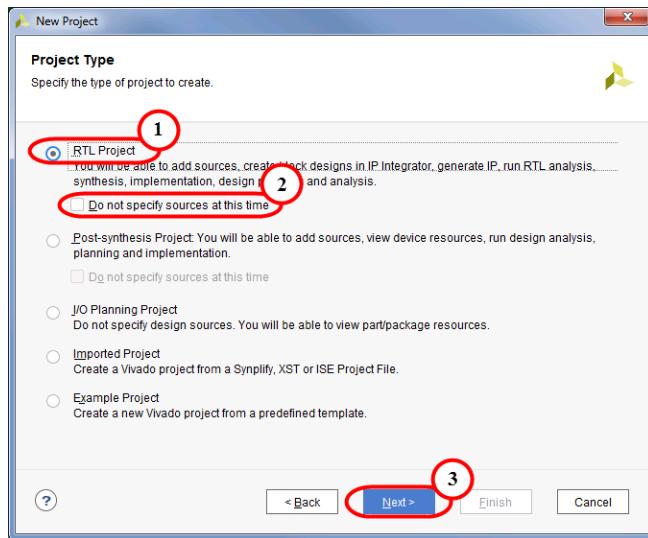


Figure 6: Setting the Project Type to RTL

- 1-2-7.** Click **Next** to accept the selection and advance to the adding sources stage (3).

- 1-3. Now that the project name, type, and location have been entered, the wizard invites you to add source files to the project. You can add entire directories where the sources are located, add specific files, or create new sources.**

Begin by adding the sources to your project.

- 1-3-1.** Click the **Plus** icon (+) to begin adding objects to the project.
- 1-3-2.** Select **Add Files** from the pop-up menu to begin adding your source files to the project.

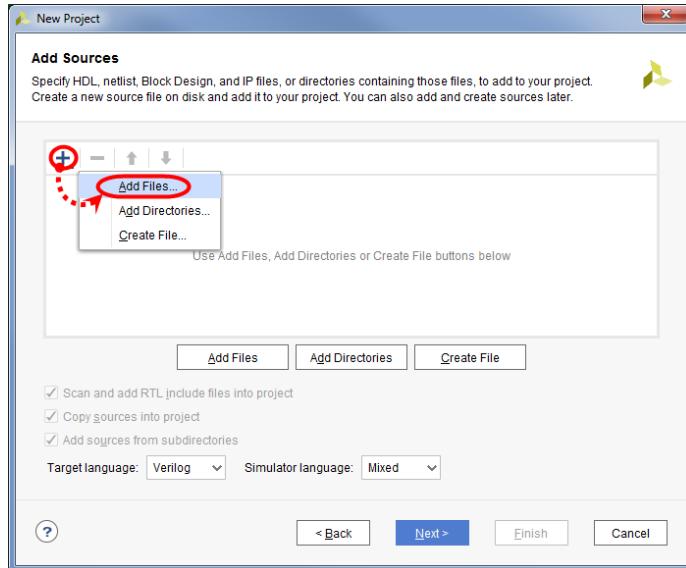


Figure 7: Adding Sources to the Project

After you add all the necessary files, the remainder of this dialog box will be addressed.
The Add Source Files dialog box opens.

- 1-3-3.** Browse to the **C:\training\<the topic cluster name>\lab\your board\<language> or your project name** directory (1).

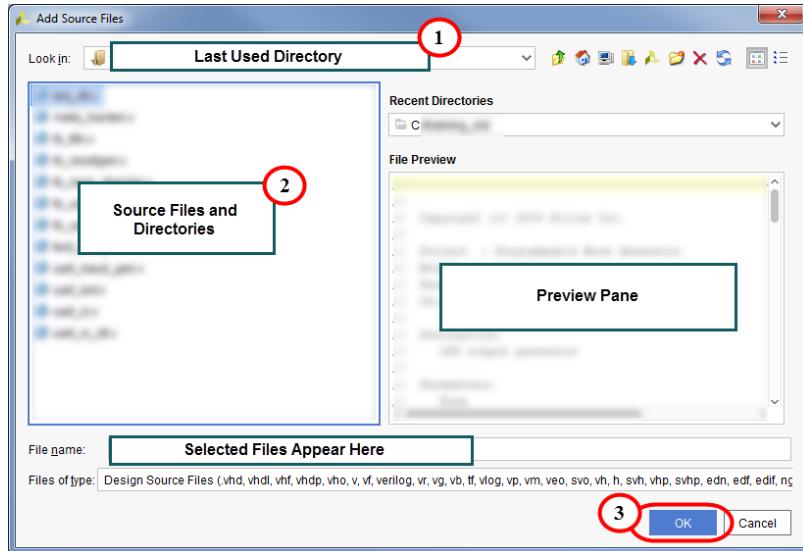


Figure 8: Adding Source Files

- 1-3-4.** Select or multi-select **your HDL sources** (2).

- 1-3-5.** Click **OK** to accept the selected files and add them as sources to the project (3).

The Add Source Files dialog box closes and returns you to the Add Sources dialog box.

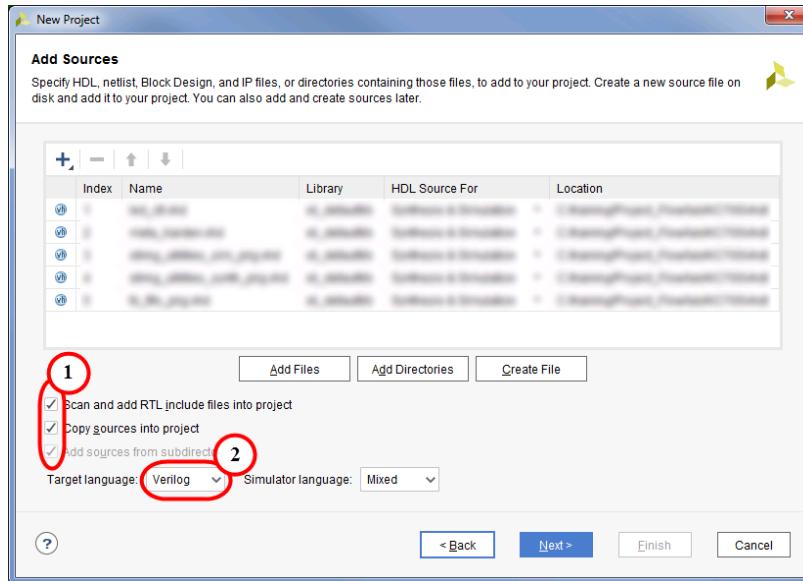


Figure 9: Setting Options in the Add Sources Window

1-4. Add any existing IP modules to the Vivado Design Suite project.

If you have existing IP modules, you will add them here.

- 1-4-1. Click the **Plus** icon (+) again to select the IP files.
- 1-4-2. Select **Add Files** to open the Add Source Files dialog box.

Since all Add Source Files dialog boxes are the same, you can refer to the previous Add Source Files dialog box illustration.

- 1-4-3. Select or multi-select **your IP modules**

- 1-4-4. Click **OK** to accept the selected files and add them as sources to the project.

If you have additional files located in other directories, you can repeat this instruction for each directory.

- 1-4-5. Confirm that the **Scan and Add RTL Include Files into Project** option is selected (used for Verilog, no effect for VHDL) in the Add Sources dialog box (1).

This will automatically pull in any include files used by Verilog sources.

- 1-4-6. Confirm that the **Copy Sources into Project** option is selected (1).

This will make a local copy of the source in the project space. Selecting this option enables you to make changes to the local copy without affecting the original source file.

- 1-4-7. Select **your preferred language** from the Target Language drop-down list (2).

This choice only affects which language is used for the generation of templates and wrappers. You can add files in any language regardless of which target language is selected. If you do not generate or use any templates or wrappers, this step is irrelevant and you can select any language.

- 1-4-8. Click **Next** to complete the adding of RTL sources and advance to adding any constraint files (3).

1-5. Add any existing constraint files to the Vivado Design Suite project.

If you have existing constraints, you will add them here.

- 1-5-1. Click the **Plus** icon (+) to select the type of object you want to import (1).

- 1-5-2. Select **Add Files** to open the Add Source Files dialog box.

The Add Source Files dialog box opens.

Since all Add Source Files dialog boxes are the same, you can refer to the previous Add Source Files dialog box illustration.

- 1-5-3. Browse to the **C:\training\<the topic cluster name>\lab\your board\<language> or your project name** directory.

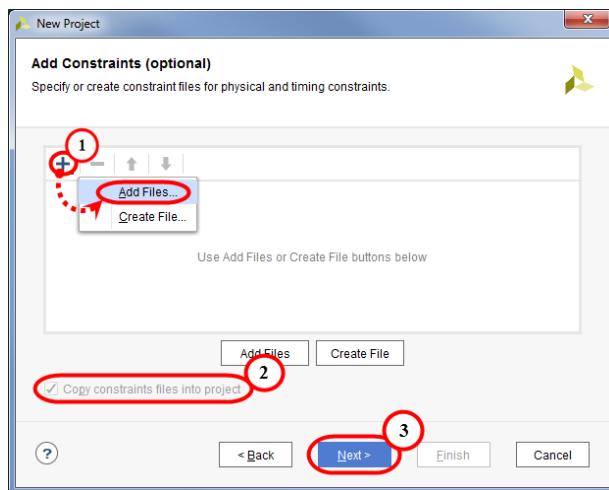


Figure 10: Adding Constraints

- 1-5-4. Select or multi-select **your constraints file**.

- 1-5-5. Click **OK** to accept the selected files and add them as sources to the project.

If you have additional files located in other directories you can repeat this instruction for each directory.

- 1-5-6. Confirm that the **Copy constraints files into Project** option is selected (2).

This will make a local copy of the source in the project space. Selecting this option enables you to make changes to the local copy without affecting the original source file.

- 1-5-7. Click **Next** to advance to selecting a target device (3).

1-6. Select the target part by first filtering by board and then by family. If you are not using a supported board, you will need to filter by part.

- 1-6-1.** Select **Boards** from the Select area (1).
- 1-6-2.** Select **the vendor of your board** from the Vendor drop-down list in the Filter area (2).

This filters the available boards to those that are populated with any member of the selected library.

- 1-6-3.** Select **your board** from the board list.

Alternatively you can select the board directly from the list at any time while in this dialog box.

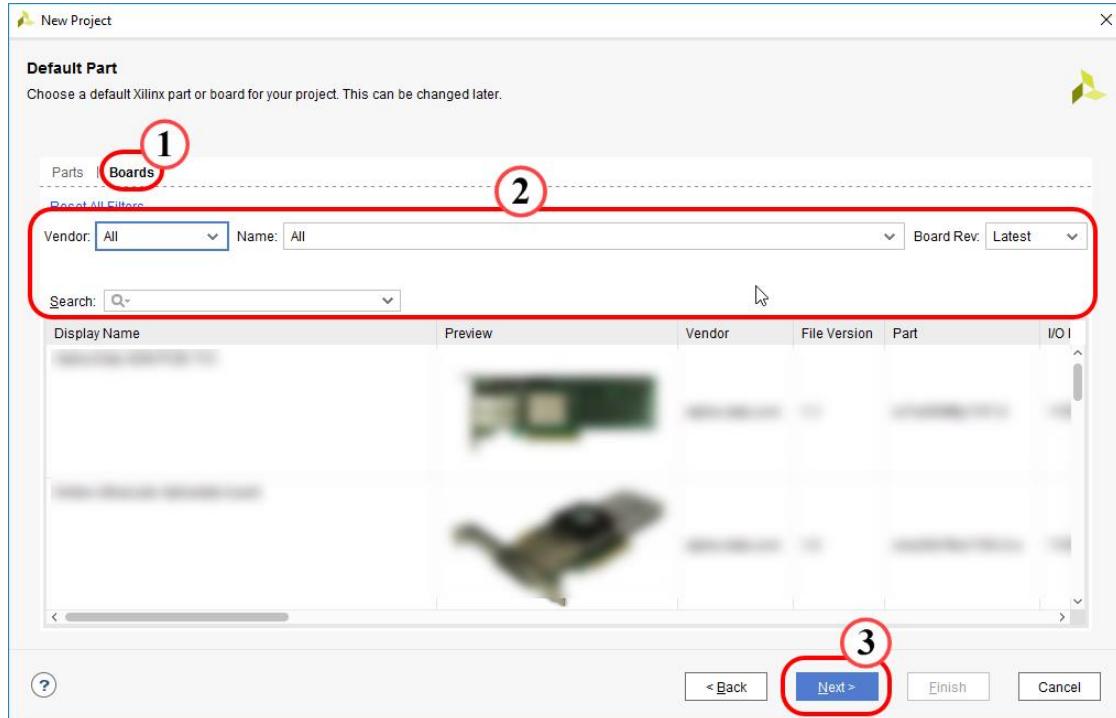


Figure 11: Selecting the Board for the Project

- 1-6-4.** Click **Next** to advance to the summary (3).

A summary of your project is displayed. If you want to change any of the information that you entered, you can do so now by clicking **Back** until you reach the correct dialog box and making the correction, or you can create the project now and edit the project properties, add or remove files, etc. later.

- 1-6-5.** Click **Finish**.

Your project is constructed and you are presented with the Vivado Design Suite main workspace environment.

Creating a Blank Vivado Design Suite Project

"Create Project" is the starting point for all designs. Projects contain sources, settings, graphics, IP, and other elements that are used to build a final bitstream and analyze a design. The Create New Project Wizard in the Vivado Design Suite allows you to specify HDL and other project resource files that will be included in the project.

1-1. Create a new blank Vivado Design Suite project.

- 1-1-1. Click **Create Project** to begin the process (1).

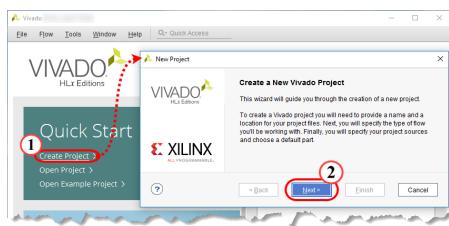


Figure 12: Creating a New Vivado Design Suite Project

This will launch the New Project Wizard.

- 1-1-2. Click **Next** to exit the introductory dialog box and begin entering in project-specific information (2).

1-2. You will now encounter a series of dialog boxes asking you to enter different pieces of information describing the project.

- 1-2-1. Enter **your project name** in the Project name field (1).
- 1-2-2. Enter the following location in the Project location field (2):

Enter project location here

Alternatively, you can use the browse feature to navigate to where you want the project to reside.

- 1-2-3. Deselect the **Create Project Subdirectory** option as leaving this checked will create an unnecessary level of hierarchy for this lab (3).

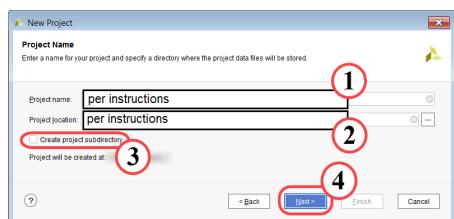


Figure 13: Entering the Project Name and Location

- 1-2-4.** Click **Next** to advance to the next dialog box (4).

Here you will specify your project type as either an RTL project or a post-synthesis project. Simply put, an RTL project enables you to add or create new HDL files and synthesize them, whereas the post-synthesis project requires pre-synthesized files. When an empty design is created, an RTL project is used.

- 1-2-5.** Select **RTL Project** (1).

- 1-2-6.** Select **Do not specify sources at this time** (2), which creates a blank project.

While existing sources could be entered at this time, you will enter them later so that you can move through this portion of the project creation process more quickly.

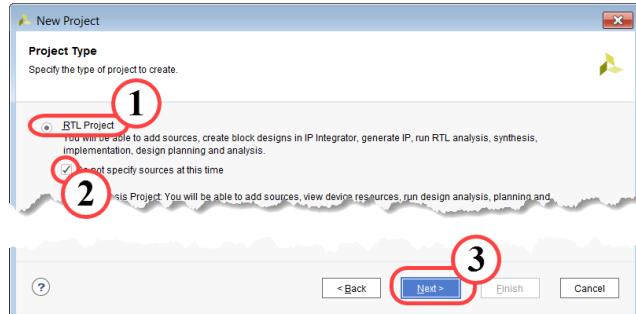


Figure 14: Selecting Project Type

- 1-2-7.** Click **Next** to advance to the target device/platform selection (3).

- 1-3.** **Select the target part by first filtering by board and then by family. If you are not using a supported board, you will need to filter by part.**

- 1-3-1.** Select **Boards** from the Select area to filter by board rather than by the specific part (1).

- 1-3-2.** Select **the vendor of your board** from the Vendor drop-down list in the Filter area (2).

This limits the number of boards seen to those manufactured by the specified vendor.

- 1-3-3.** Select **your board** from the board list.

Alternatively you can select the board directly from the list at any time while in this dialog box.

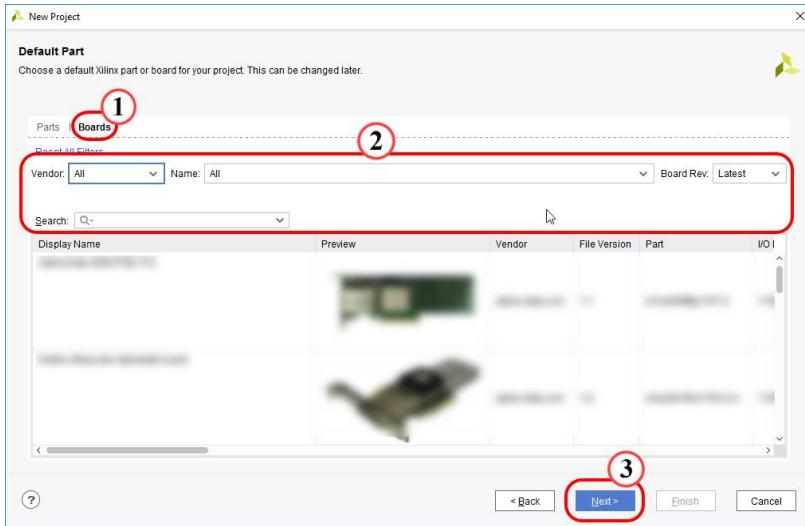


Figure 15: Selecting the Board for the Project

- 1-3-4.** Click **Next** to advance to the summary (3).

A summary of your project is displayed. If you want to change any of the information that you entered, you can do so now by clicking **Back** until you reach the correct dialog box and making the correction, or you can create the project now and edit the project properties, add or remove files, etc., later.

- 1-3-5.** Click **Finish** to accept these settings and build the project.

Your project is constructed and leaves you in the operational portion of the Vivado Design Suite GUI.

Opening a Vivado Design Suite Project

1-1. Open the existing Vivado Design Suite project *your project name*.

- 1-1-1. Click **Open Project** from the Quick Start section (1).

The Open Project dialog box opens (2).

- 1-1-2. Browse to the /home/xilinx/training/<the topic cluster name>/lab/your board/<language> or your project name directory in the **Look in** field (3).

Note: The drop-down arrow shows the directory hierarchy.

- 1-1-3. Select **your project name** (4).

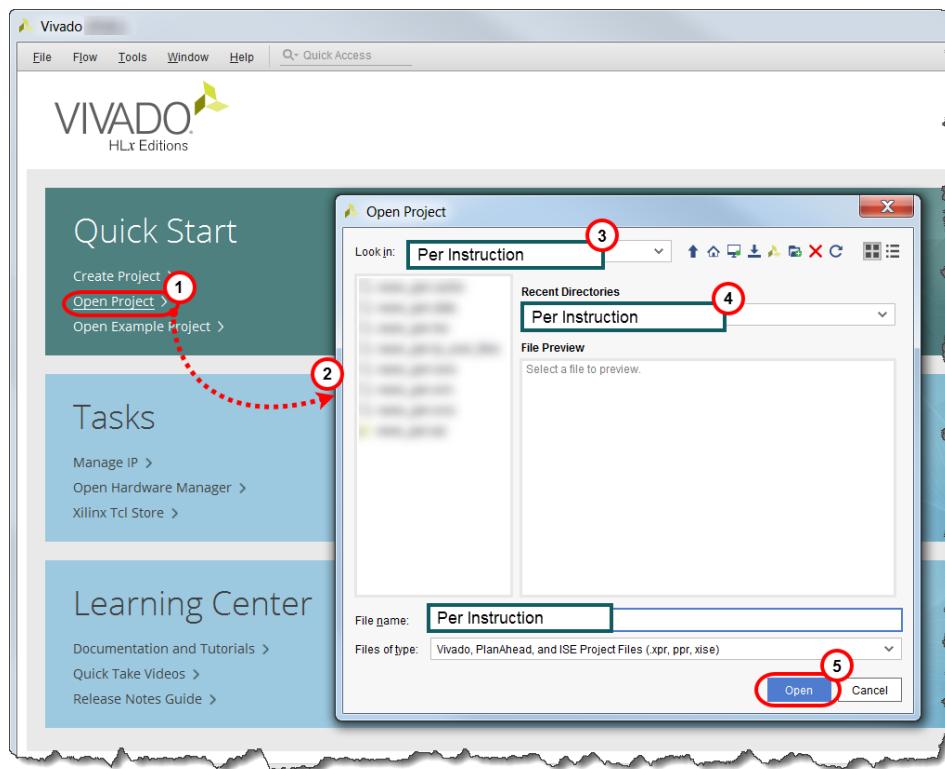


Figure 16: Opening an Existing Project

- 1-1-4. Click **OK** to open the selected project (5).

The project now opens in the Vivado Design Suite.

Opening Source Code in the Editor

1-1. Open ***the desired source file*** in the editor.

1-1-1. Under the Sources tab, locate ***the desired source file***.

Note: This process is valid for any of the sub-tabs under Sources. Typically the Hierarchy and Libraries sub-tabs are the most commonly used as these organize the user files in the most convenient fashion.

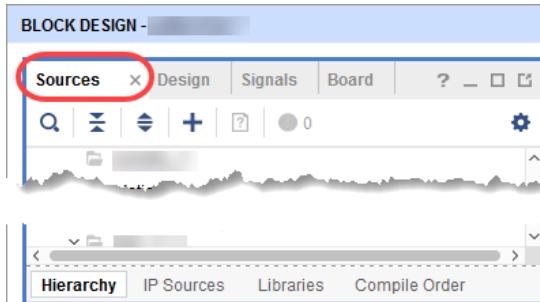


Figure 17: Sources Tab

1-1-2. Double-click ***the desired source file*** to open it in the editor.

Note: You can also right-click and select **Open File** to open the file in the editor.

Creating an HDL Source File

1-1. Create a new HDL source file called ***your HDL sources***.

1-1-1. Select **Add Sources** in the Flow Navigator under Project Manager.

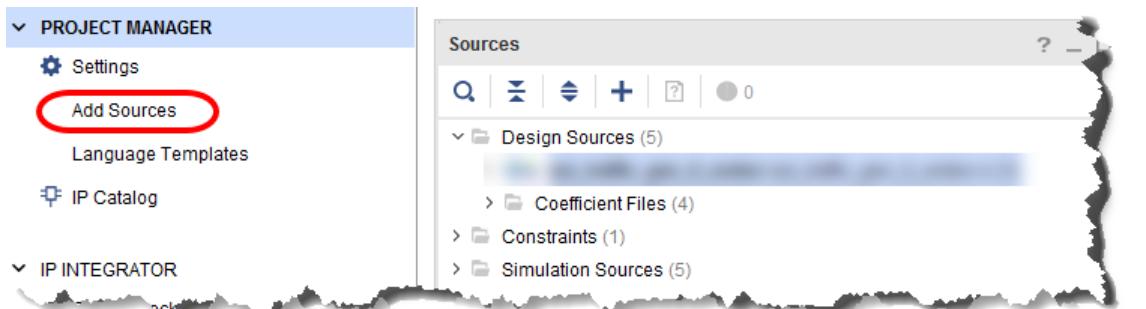


Figure 18: Selecting Add Sources

1-1-2. Select **Add or create design sources**.



Figure 19: Selecting Add or Create Design Sources

1-1-3. Click **Next**.

The Add or Create Design Sources dialog box opens.

1-1-4. Click the **Plus** (+) icon and select **Create File**.

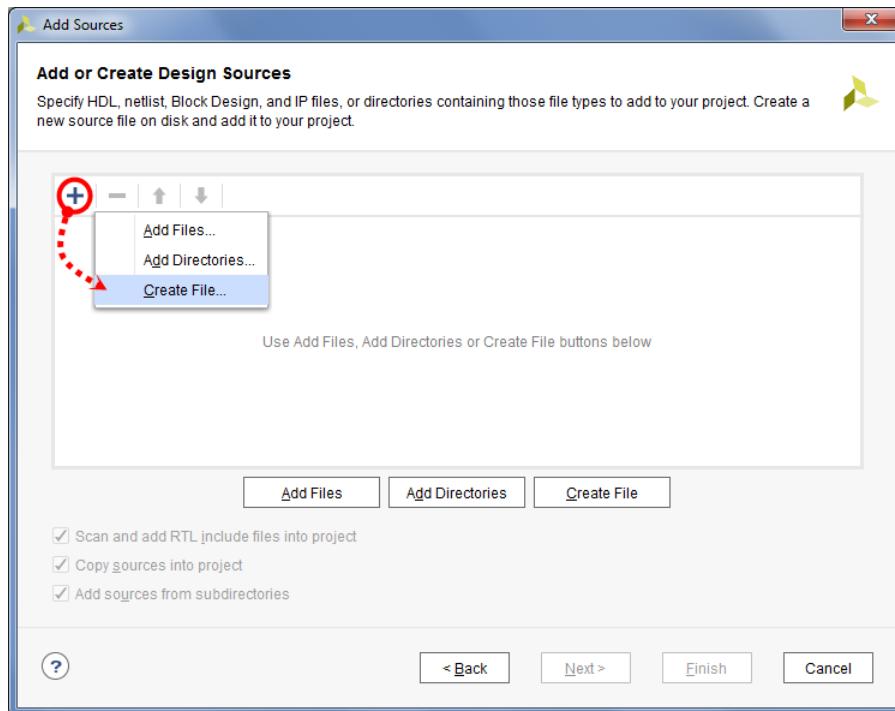


Figure 20: Selecting Create File

The Create Source File dialog box opens.

1-1-5. Select **your preferred language** from the File type drop-down list.

1-1-6. Enter **your HDL sources** as the file name.

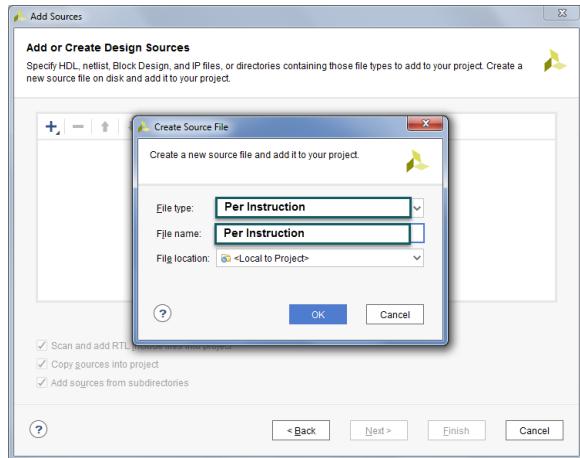


Figure 21: Entering File Name and Type

1-1-7. Click **OK** in the Create Source File dialog box.

1-1-8. Click **Finish** to add the new source file(s).

The Define Module dialog box opens.

Opening a File in the Editor

This process is used to look at any type of text-based file. It is suggested that if the file you want to open is part of the project, then you merely double-click the filename when in the Hierarchical view or the Library view.

1-1. Open the location of the text file to open\your file in the built-in editor.

1-1-1. Select **File > Text Editor > Open File**.

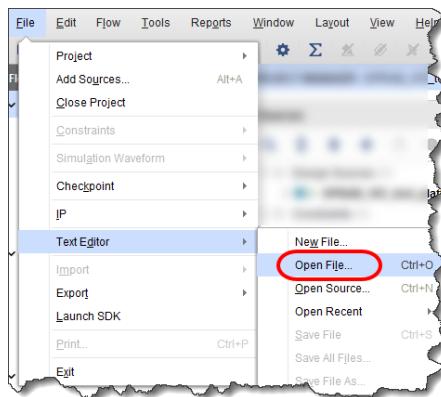


Figure 22: Opening a Text File from the File Menu

A file browser window opens.

1-1-2. Navigate to *the location of the text file to open.*

1-1-3. Select **your file**.

1-1-4. Click **OK**.

The text file opens in the workspace area.

Importing IP

IP can be imported from a number of tools provided that they adhere to the IP-XACT standard. Once created, the Vivado Design Suite must be made aware of the presence of the IP. This is typically performed from an open project. The instructions below describe the process of importing a single piece of IP; however, the steps can be repeated as necessary to collect different pieces of IP from various locations.

1-1. Import your IP cores into the open project.

1-1-1. Using the Flow Navigator, select **Project Manager > Project Settings** (1).

The Project Settings Dialog box opens with the General tab open.

1-1-2. Select the **IP** tab to access the IP settings (2).

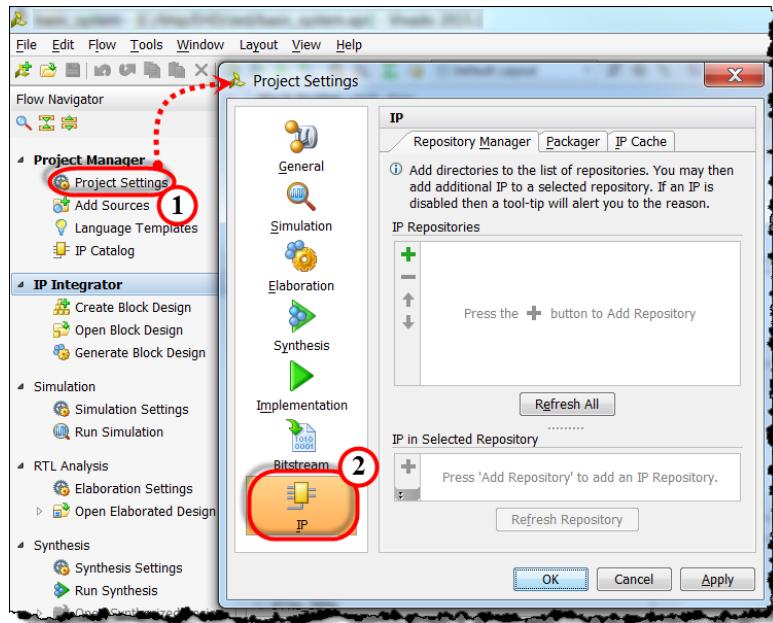


Figure 23: Accessing the Project's IP Settings

1-1-3. Click the green **Plus** icon (+) to open the file browser to point to the IP repository in which your IP is located (1).

1-1-4. Browse to *where your IP is located.*

1-1-5. Click Select.

IMPORTANT: If the IP repository that you have selected has the IP in zip format, then you will need to continue as described below. If, however, your IP has been expanded (for example, if you are revisiting this step or adding another piece of IP), then it will automatically appear in the IP in Selected Repository field. You can simply click **OK** to complete the repository inclusion process.

The selected IP repository is scanned for all IP that is listed in the Select IP To Add To Repository dialog box.

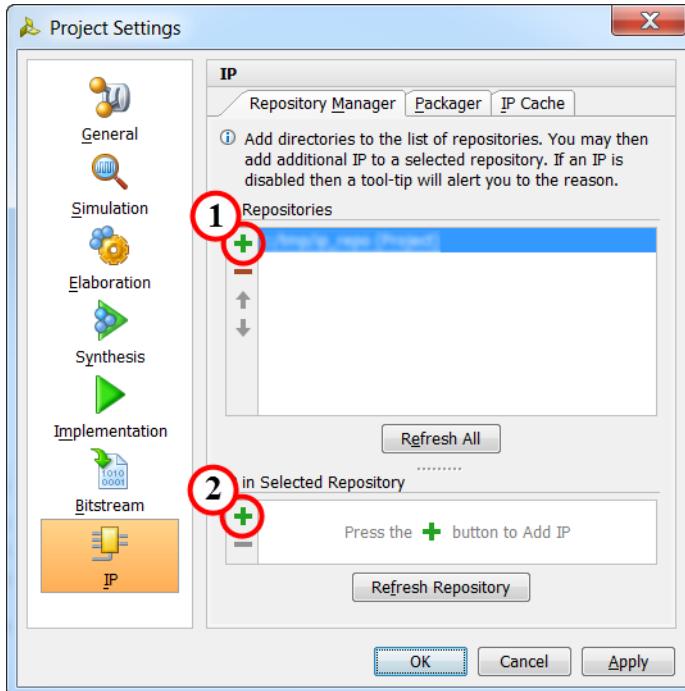


Figure 24: Adding the IP Repositories and the Specific IP

- 1-1-6. Click the green **Plus** icon (+) to open the Select IP To Add To Repository dialog box, which lists all the available IP in the selected repository (2).
- 1-1-7. Select one piece of IP that you would like to add from this directory.
- 1-1-8. Click **Open** to import the IP.
- 1-1-9. Click **OK** to expand the IP archive into the selected IP repository location.
- 1-1-10. Click **OK** to exit the project settings.

losing the Vivado Design Suite Project

1-1. Close the project.

- 1-1-1. Select **File > Close Project** to close the project.

The Close Project dialog box opens.

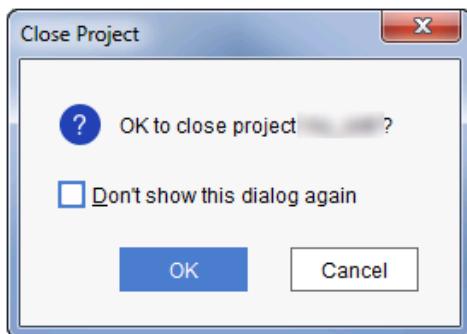


Figure 25: Close Project Dialog Box

- 1-1-2. Click **OK**.

Closing the Vivado Design Suite

1-1. Close the Vivado Design Suite.

- 1-1-1. Select **File > Exit**.

The Exit Vivado dialog box opens.



Figure 26: Exit Vivado Dialog Box

- 1-1-2. Click **OK**.

Vivado Add Sources Operations

In This Section

Adding an HDL Source File.....	26
Adding a Simulation Source File	27
Adding a Constraint File.....	29
Creating a SystemVerilog Simulation Source File.....	30
Adding Existing Constraints.....	32

Adding an HDL Source File

HDL source files can be added to the design at any time.

1-1. Add an HDL source file to the design.

- 1-1-1. Select **Add Sources** under the Flow Navigator tab in the Project Manager.

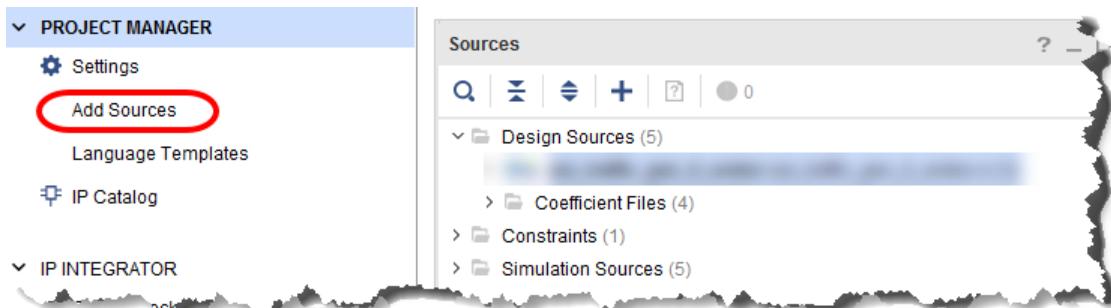


Figure 27: Selecting Add Sources

The Add Sources dialog box opens, allowing you to add HDL source files to the project.

- 1-1-2. Select **Add or create design sources** (1).



Figure 28: Selecting Add or Create Design Sources

- 1-1-3. Click **Next** to begin selecting source files (2).

The Add or Create Design Sources dialog box opens and prompts you to add existing HDL source files or to create new HDL sources files.

- 1-1-4. Click the **Plus (+)** icon and select **Add Files**.
- 1-1-5. Browse to *your source directory* if it is not open already.
- 1-1-6. Select **your HDL sources**.
- 1-1-7. Double-click the source file name in the Add Source Files dialog box to select the file(s) or click **OK**.

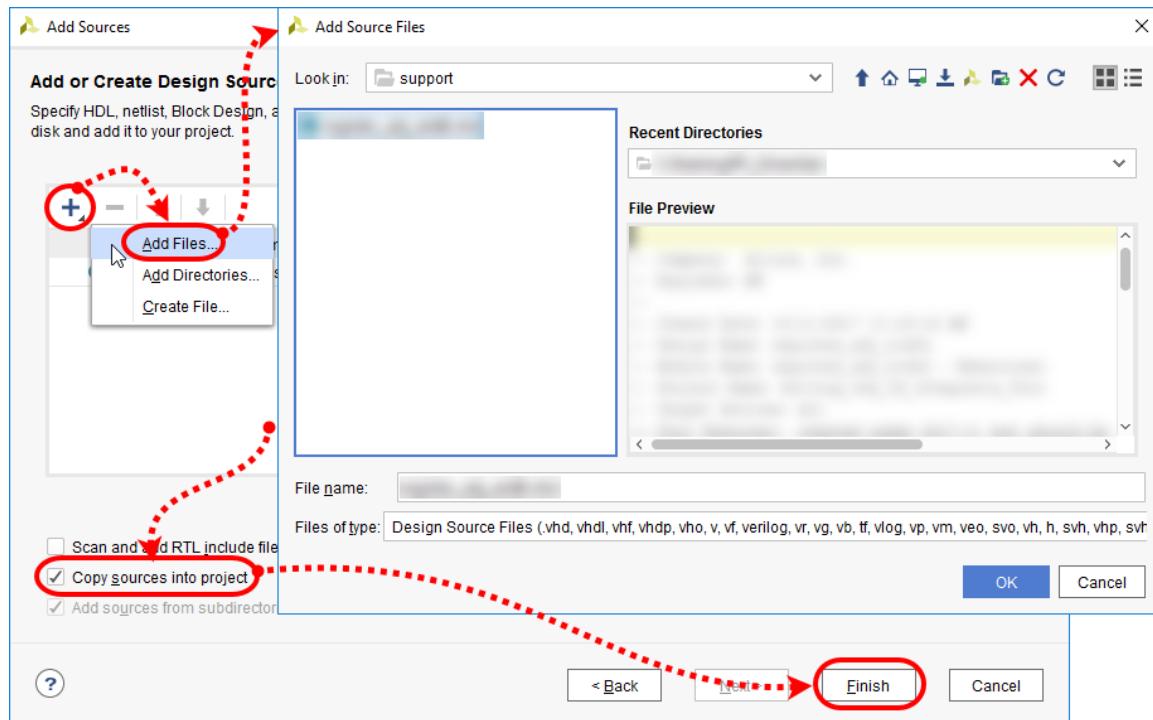


Figure 29: Selecting Add Files

- 1-1-8. Ensure that the **Copy sources into project** option is selected (when building IP this will be listed as **Copy sources into IP Directory**).
- 1-1-9. Click **Finish** in the Add or Create Design Sources dialog box to add the HDL sources to the project.

Adding a Simulation Source File

HDL simulation files can be added to the design at any time.

1-1. Add simulation files to the design.

- 1-1-1. Select **Add Sources** under Project Manager in the Flow Navigator.

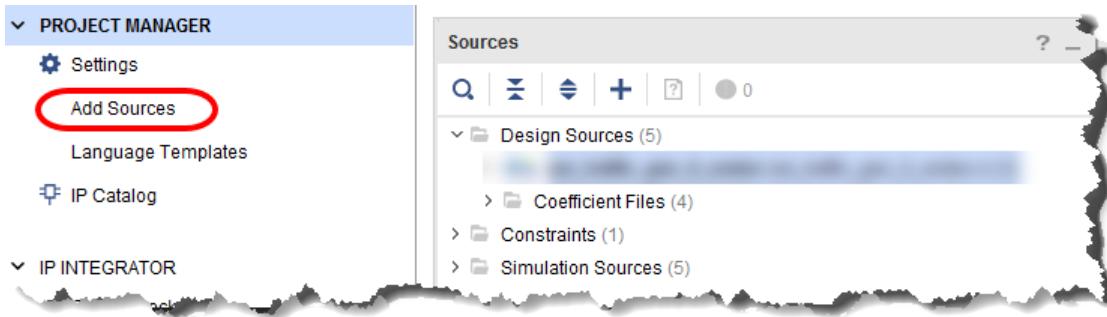


Figure 30: Selecting Add Sources

- 1-1-2. Select **Add or create simulation sources**.

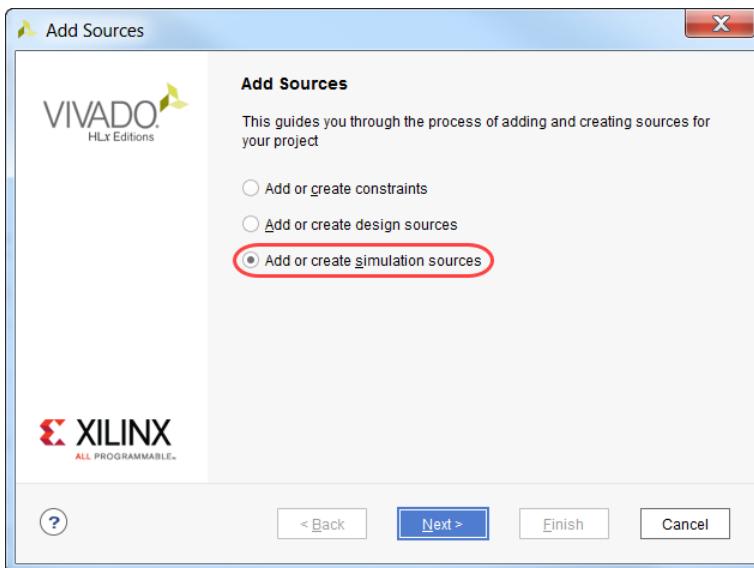


Figure 31: Add Sources Dialog Box

- 1-1-3. Click **Next**.

- 1-1-4. Click the **Plus** (+) icon to open the pop-up menu.

- 1-1-5. Select **Add Files** to open the Add Source Files dialog box which allows you to browse to the desired directory.

- 1-1-6. Browse to the `$::env(<the topic cluster name>)/support` directory if it is not open already.

- 1-1-7. Select **your HDL sources**.

- 1-1-8. Click **OK** in the Add Source Files dialog box.
- 1-1-9. Ensure that the **Copy sources into project** option is selected.
- 1-1-10. Click **Finish** to add the file(s) to the project.

Adding a Constraint File

A constraint file can be added to the design at any time.

1-1. Add a constraint file to the design.

- 1-1-1. In the Flow Manager, under Project Manager, select **Add Sources**.

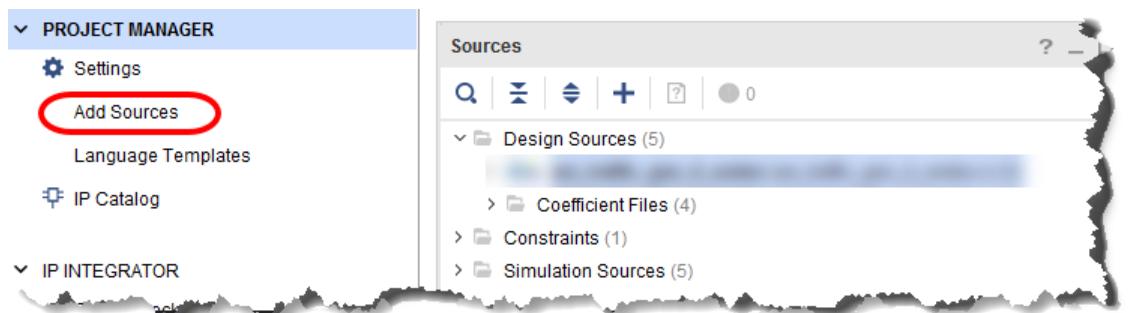


Figure 32: Selecting Add Sources

- 1-1-2. Select **Add or Create Constraints**.



Figure 33: Selecting Add or Create Constraints

1-1-3. Click Add Files.

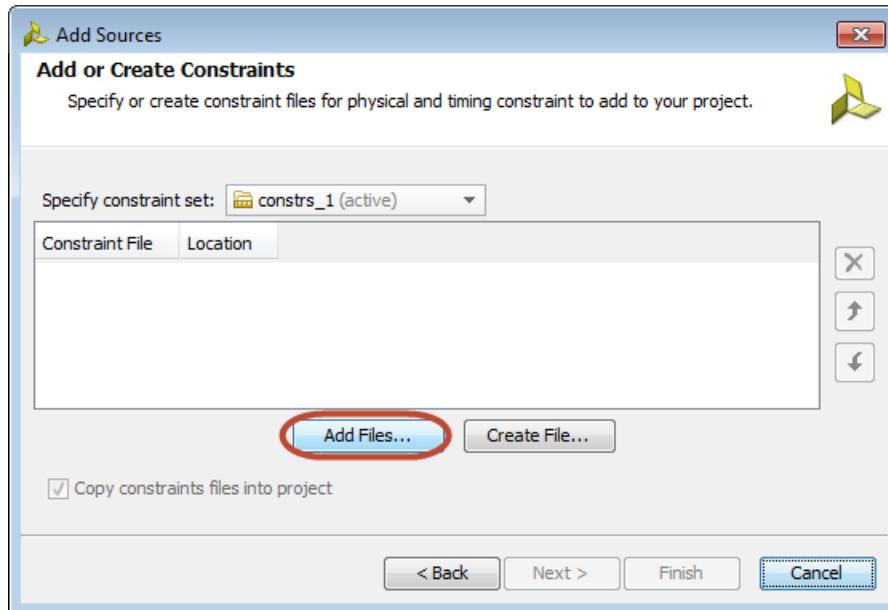


Figure 34: Selecting Add Files

- 1-1-4. Browse to `C:\training\<the topic cluster name>\lab\your board\<language>` or your project name.
- 1-1-5. Select **your constraints file** and click **OK**.
- 1-1-6. Click **Finish** in the Add or Create Design Sources dialog box to add the HDL sources to the project.

Creating a SystemVerilog Simulation Source File

1-1. Create a new SystemVerilog file called *your HDL sources*.

- 1-1-1. Select **Add Sources** in the Flow Navigator, under Project Manager.

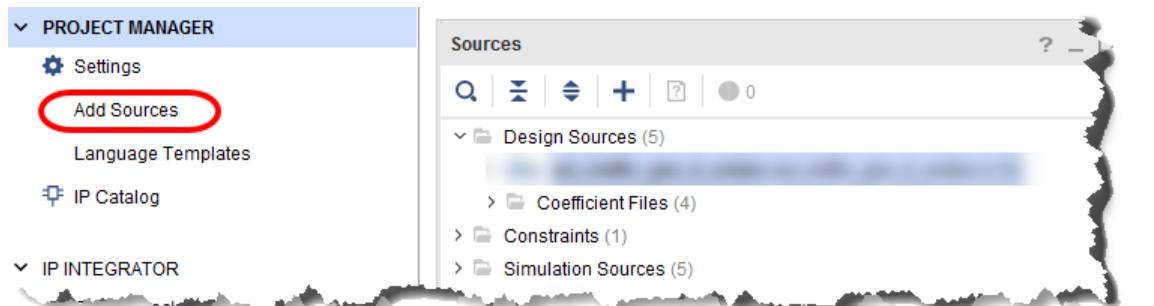


Figure 35: Selecting Add Sources

1-1-2. Select **Add or create simulation sources**.



Figure 36: Add Sources Dialog Box

Note that selecting **Add or create simulation sources** will designate the file for simulation only. When creating a design source file, you would select **Add or create design sources**, which will designate the file for both synthesis and simulation.

1-1-3. Click **Next**.

The Add or Create Simulation Sources dialog box opens.

1-1-4. Click the **Plus** (+) icon and select **Create File**.

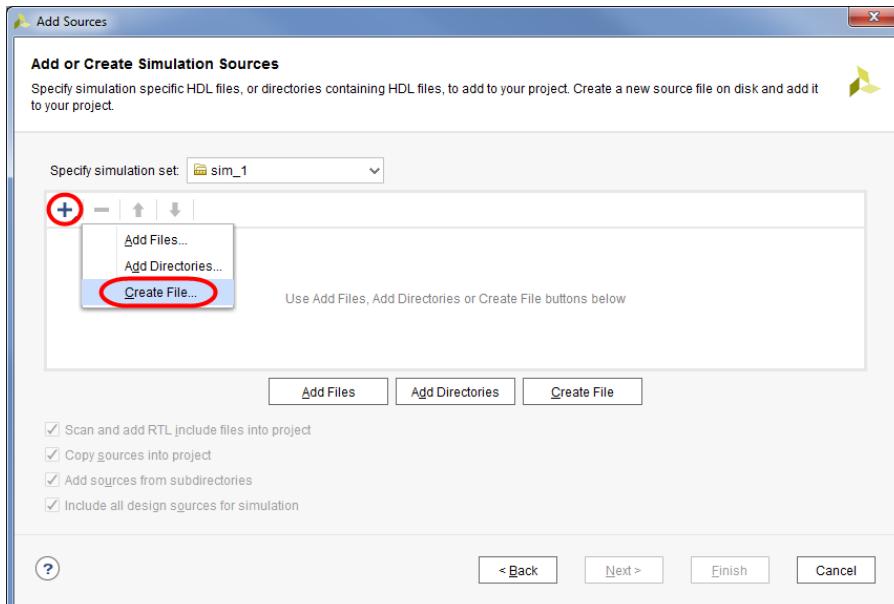


Figure 37: Selecting Create File

The Create Source File dialog box opens.

- 1-1-5. Enter **your HDL sources** as the file name.
- 1-1-6. Select **your preferred language** from the File type drop-down list.

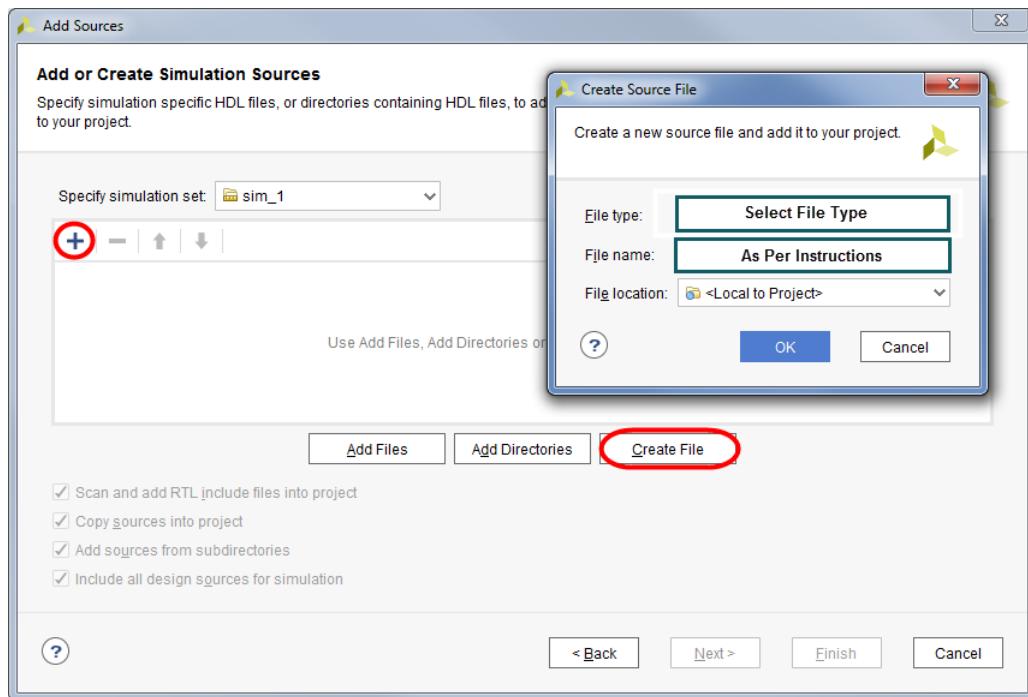


Figure 38: Entering Testbench Filename and Type

- 1-1-7. Click **OK** in the Create Source File dialog box.

- 1-1-8. Click **Finish**.

The Define Module dialog box opens.

- 1-1-9. Click **OK** to create the module without ports, since this module does not require them.

- 1-1-10. Click **Yes** to confirm that there the module definition has not been changed.

Adding Existing Constraints

1-1. Add your **constraints file** to the design.

- 1-1-1. Click **Add Sources** under Project Manager in the Flow Navigator.
- 1-1-2. Select **Add or Create Constraints**.



Figure 39: Adding a Constraint File to a Vivado Design Suite Project

- 1-1-3. Click **Next** to advance to the next dialog box.
- 1-1-4. Click the green **Plus** icon (+) and select **Add Files** to open the Add Constraint Files dialog box.
- 1-1-5. Browse to the *the location of your files* directory.
- 1-1-6. Select **your constraints file** as the desired constraint file.

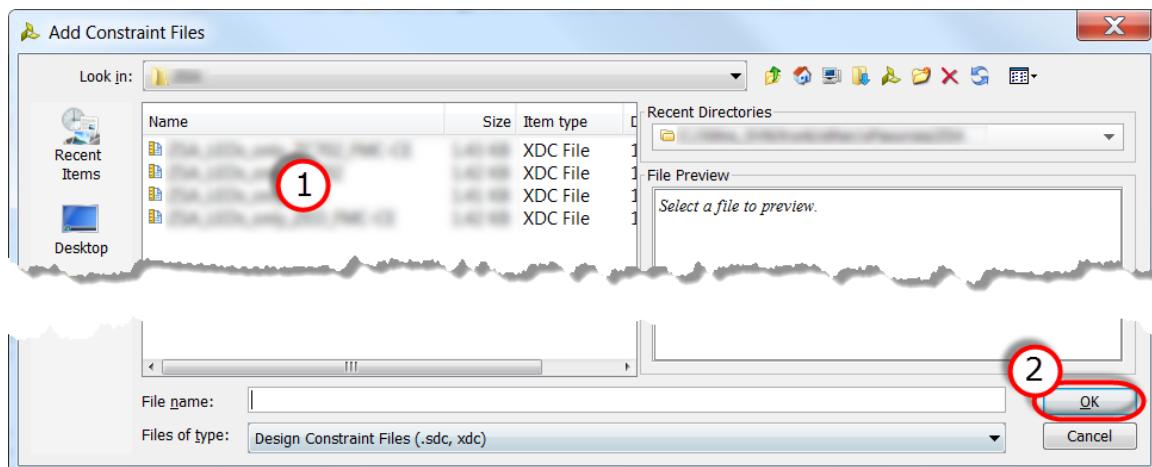


Figure 40: Selecting One or More Constraint Files

- 1-1-7. Click **OK** to select the file.
- 1-1-8. Click **Finish** to add the constraint file to the project.

Embedded Operations

In This Section

Exporting Only the Hardware Description for SDK	34
Exporting the Hardware Description and Bitstream for SDK.....	35
Exporting Hardware and Launching SDK.....	37
Launching SDK from within the Vivado Design Suite	39

Exporting Only the Hardware Description for SDK

Exporting only the hardware description for SDK rather than the completed bitstream is beneficial during the hardware development cycle in that the software engineers can begin writing code for the platform prior to needing the bitstream.

This engages the software developers earlier in the cycle and provides additional time for the hardware engineers to make final tweaks to the hardware portion of the design and ensure that the design meets timing.

1-1. Export only the hardware description for SDK.

1-1-1. Select **File (1) > **Export** (2) > **Export Hardware** (3).**

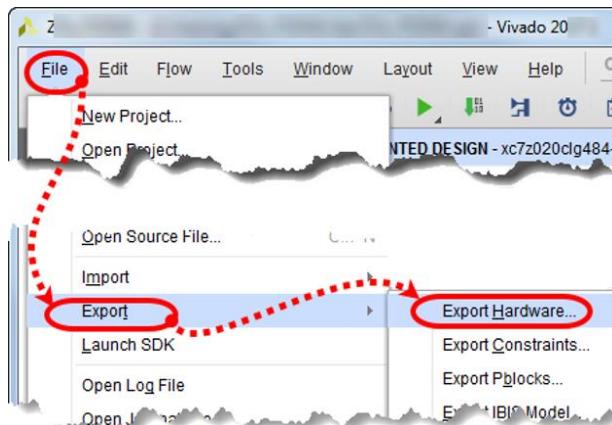


Figure 41: Exporting a Design to SDK

The Export Hardware dialog box opens.

1-1-2. Use the **Export to drop-down list to choose *the location of the files to be exported from the Vivado Design Suite* as the export location.**

This option enables you to specify a location for the exported hardware definition. Often it is beneficial to choose a directory separate from the hardware design files.

With the default <Local to Project> option, the Vivado Design Suite will export the hardware definition to a sub-directory of the current Vivado Design Suite project directory.

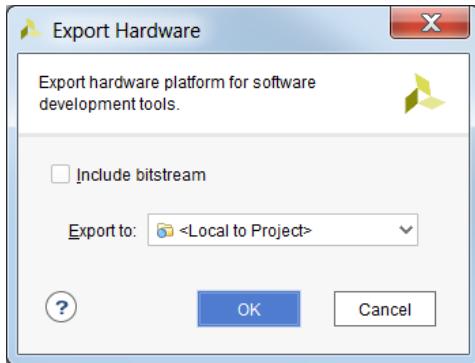


Figure 42: Export Hardware for SDK without Bitstream Generation

- 1-1-3. Click **OK** to proceed with the export and close the dialog box.

This will create a hardware description file (HDF) at the selected export path.

Exporting the Hardware Description and Bitstream for SDK

1-1. Export the design for SDK.

From the Vivado Design Suite you can export a description of the embedded system design in the form of a hardware description file (HDF) and launch SDK. When SDK is launched, the HDF file is automatically imported and a hardware platform specification generated. Note that exporting the hardware description and launching SDK are two separate actions.

- 1-1-1. Select **File > Export > Export Hardware**.

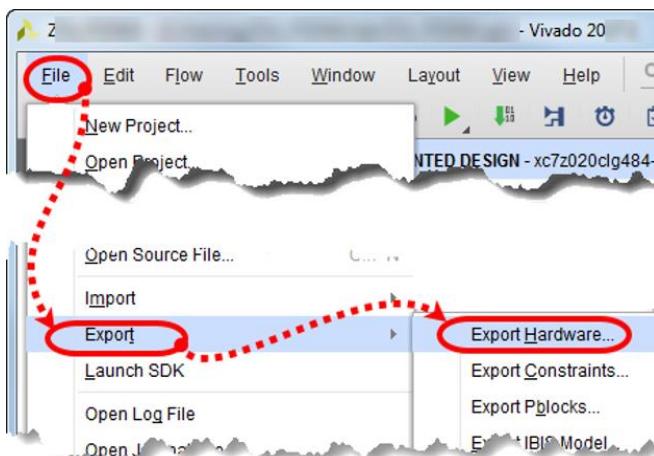


Figure 43: Exporting a Design to SDK

The Export dialog box opens.

The *Export to* field (1) allows you to specify a location to where the files should be exported.

- 1-1-2. Select **Choose Location** from the *Export to* drop-down list.
- 1-1-3. Navigate to **the location of the files to be exported from the Vivado Design Suite** as the export location.

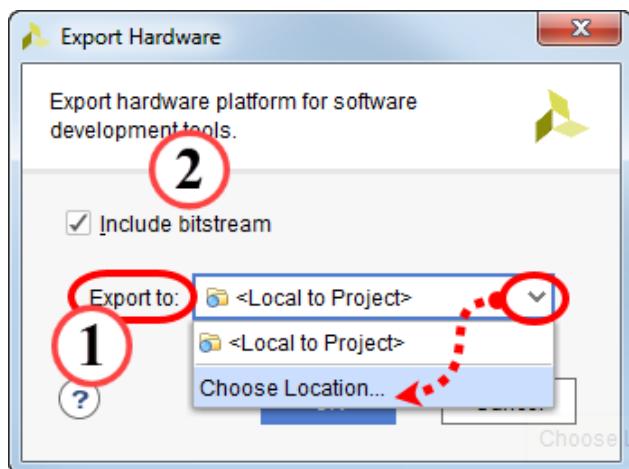


Figure 44: Using Choose Location to Browse to Destination Directory

Any time there is a part of the design in the PL or FPGA fabric, you would typically include the bitstream in the export (2). This is typically enabled so that all files required by the software designer are available.

The Choose Location option enables you to specify a location for the exported hardware definition. Often it is beneficial to choose a directory separate from hardware design files.

With the default <Local to Project> option, the Vivado Design Suite will export the hardware definition to a sub-directory of the current Vivado Design Suite project directory. This is typically done when the hardware designer is performing a quick check of the hardware. Often the files are exported to other locations when the expectation is to hand off to a software team.

Select the appropriate option from the drop-down list. If exporting to a directory that does not yet exist, create it.

- 1-1-4. Select the **Include bitstream** option to add the bitstream to the HDF so that all of the hardware information is available in SDK.
- 1-1-5. Click **OK** to export the hardware definition.

This will cause the HDF descriptor file to be generated and placed in the directory specified in the *Export to* field.

Exporting Hardware and Launching SDK

1-1. Export the design.

From the Vivado IDE you are able to export the hardware portion of the embedded system component as an HDF file and launch SDK.

1-1-1. Select **File** (1) > **Export** (2) > **Export Hardware** (3).

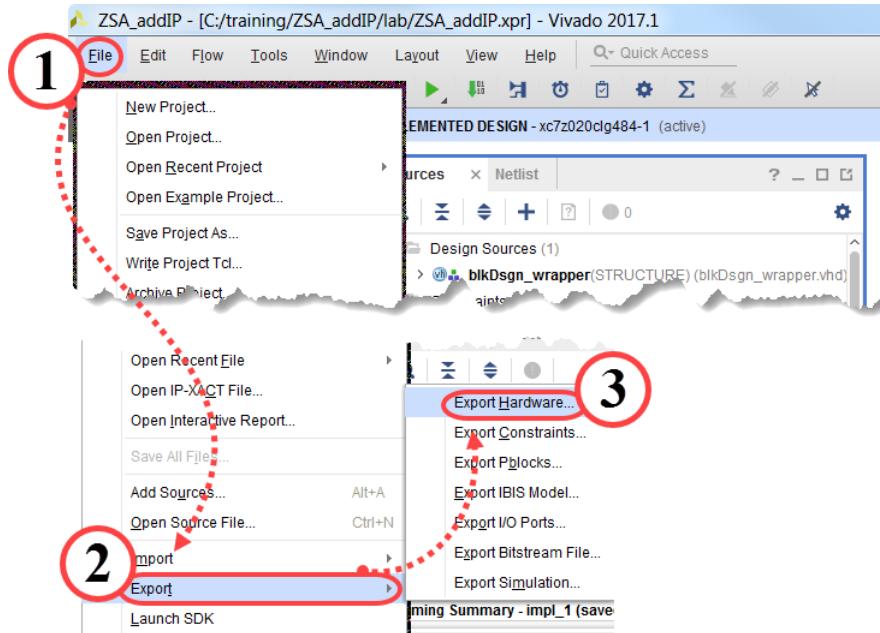


Figure 45: Exporting Hardware

The Export Hardware for SDK dialog box opens and asks you to select if the bitstream is to be exported. If you did not run the Generate Bitstream command, the Include Bitstream option is grayed out.

- 1-1-2. Select **Include bitstream** to include the bitstream in the HDF file.
- 1-1-3. Select **<Local to Project>** to export the hardware platform to the default project directory.
- 1-1-4. Click **OK** to close the dialog box and proceed with the export.

If an export file already exists at the specified location, click **Yes** to overwrite the existing file.

This will create a hardware description file (HDF) at the selected export path.

1-2. Launch SDK.

1-2-1. Select **File > Launch SDK**.

The Launch SDK dialog box opens.

Note: You could just as well open SDK from outside of the Vivado IDE, such as from the Windows Start menu, but for convenience, the Vivado Design Suite provides its own shortcut.

1-2-2. Select **the location of the files to be exported from the Vivado Design Suite** from the Exported location drop-down list in the dialog box.

1-2-3. Select **a workspace** from the Workspace drop-down list.

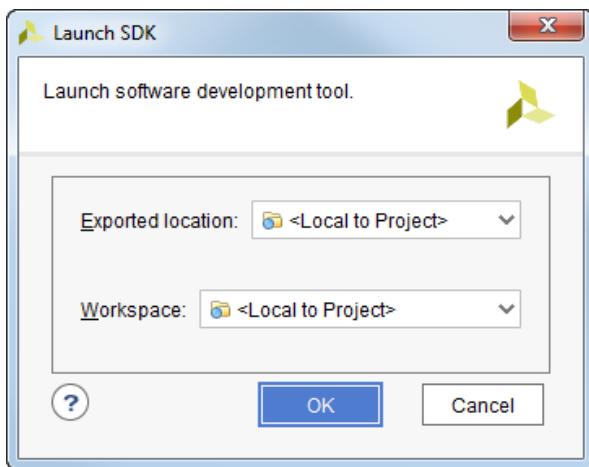


Figure 46: Launching SDK

1-2-4. Click **OK** to continue launching SDK.

The SDK tool will launch.

SDK takes the hardware description from the Vivado Design Suite and generates a model of the hardware from which other software is based. This hardware description project takes the name of *name of your block design_wrapper_hw_platform_0*. From here, you will construct a number of other projects based on this hardware platform description.

If the option to export to a local directory is used by default, then the SDK tool files will all be placed in a directory below the embedded project in *your project name.sdk*. The decision to not use the default directory is based on the *sharability* of the hardware files. Because you may be executing a number of software labs, it is convenient to keep all of the SDK files together.

Once set, you cannot change the location of this workspace. If it is necessary to move a software application to another location or computer, use the import and export features built into SDK. While not a requirement, it is a good idea to keep the SDK-related files together.

The actual exporting of the embedded processor hardware is transparent to the user.

1-2-5. Close the Welcome screen, if it appears, in order to view the underlying SDK perspective.

The Welcome screen is a quick and convenient way to access documents and tutorials. Although an excellent launching point to begin exploring SDK, it is not used in this lab. Once closed, the Welcome screen can always be re-opened from the main menu bar (**Help > Welcome**).

1-2-6. Maximize the SDK window so that all window panes are easier to view.

1-2-7. Navigate to the Address Map section of the open HDF.

Here you will see peripherals from your hardware design appear in the map with their assigned addresses.

The address map is just one example of information exported from the Vivado Design Suite to SDK.

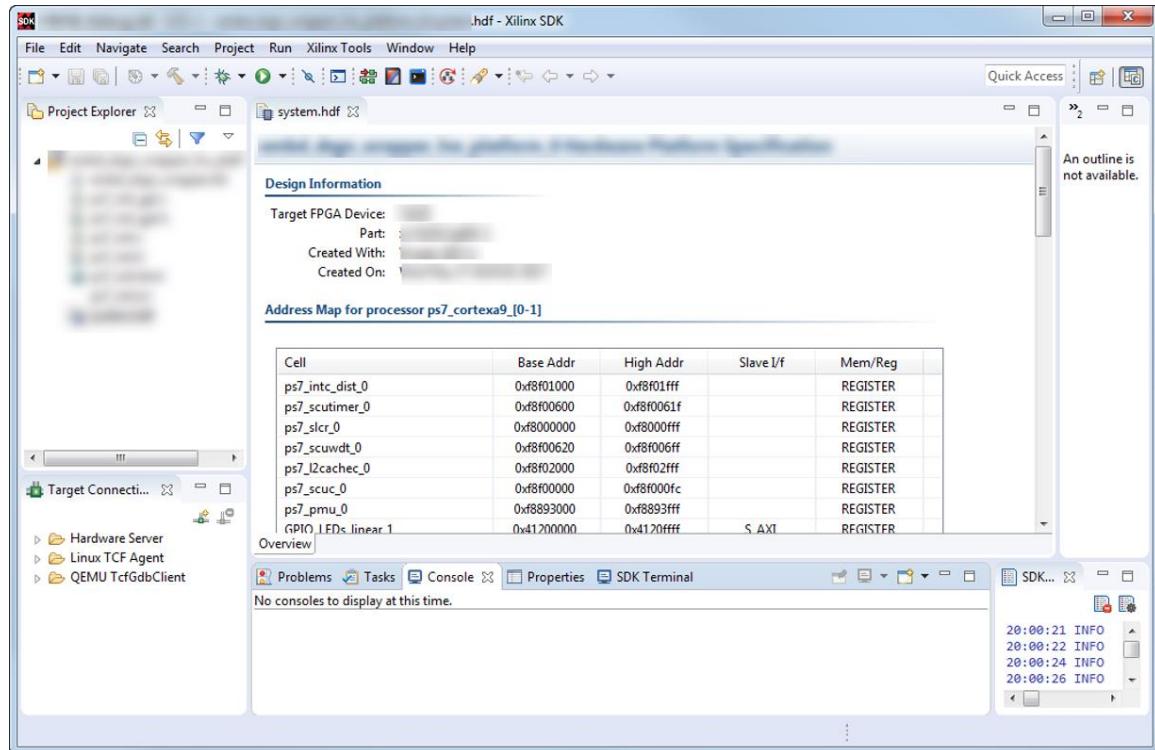


Figure 47: SDK Main Window

1-2-8. Select the **Console** tab in the bottom pane to bring the Console view to the foreground.

This view is useful for tracking the status of builds.

Launching SDK from within the Vivado Design Suite

Once the hardware design is complete and the exported files have been created, it is time to work on the software. If the software will be worked on by another team, then the Vivado Design Suite is exited and the hardware engineer's task is done (until the software team wants more capabilities).

Sometimes, however, it is desirable to immediately begin software development, such as when basic drivers need to be written to test the hardware. This situation calls for launching SDK from within the Vivado Design Suite.

1-1. Launch SDK with the exported design.

1-1-1. Select **File > Launch SDK**.

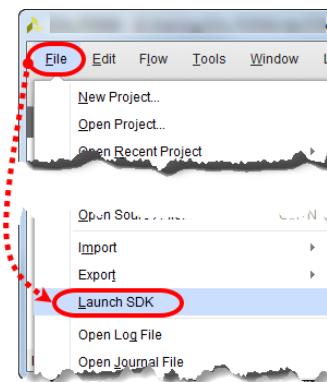


Figure 48: Launching SDK from within the Vivado Design Suite

1-1-2. Select **the location of the files to be exported from the Vivado Design Suite** (1) for the *Exported location* field and a **workspace** (2) for the *Workspace* field.

If selecting a specific directory that does not yet exist, create it.

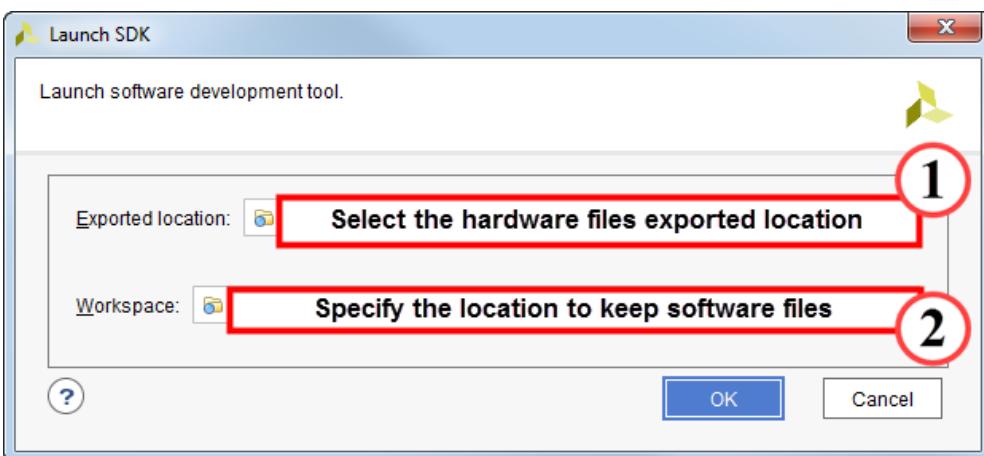


Figure 49: Launch SDK Dialog Box

The Workspace field enables you to specify a location where the software elements of the design will go. Often it is beneficial to create a separate directory in which to keep the software files separate from the hardware files.

1-1-3. Click **OK** to continue launching SDK.

The SDK tool will launch.

SDK takes the hardware description from the Vivado Design Suite and generates a model of the hardware from which other software is based. This hardware description project takes the name of *name of your block design_wrapper_hw_platform_0*. From here, you will construct a number of other projects based on this hardware platform description.

If the option to export to a local directory is used by default, then the SDK tool files will all be placed in a directory below the embedded project in *your project name.sdk*. The decision to not use the default directory is based on the *shareability* of the hardware files. Because you may be executing a number of software labs, it is convenient to keep all of the SDK files together.

Once set, you cannot change the location of this workspace. If it is necessary to move a software application to another location or computer, use the import and export features built into SDK. While not a requirement, it is a good idea to keep the SDK-related files together.

The actual exporting of the embedded processor hardware is transparent to the user.

1-1-4. Close the Welcome screen to view the underlying SDK perspective.

The Welcome screen is a quick and convenient way to access documents and tutorials. Although an excellent launch point to begin exploring SDK, it is not used in this lab. Once closed, the Welcome screen can always be re-opened from the main menu bar: *Help > Welcome*.

1-1-5. Maximize the SDK window so that all window panes are easier to view.

When SDK is launched from within the Vivado Design Suite, the hardware's HDF information is presented to you through the Hardware Platform Specification. This includes an Address Map section which lists out all of the various peripherals along with their addresses.

The address map is just one example of information exported from the Vivado Design Suite to SDK.

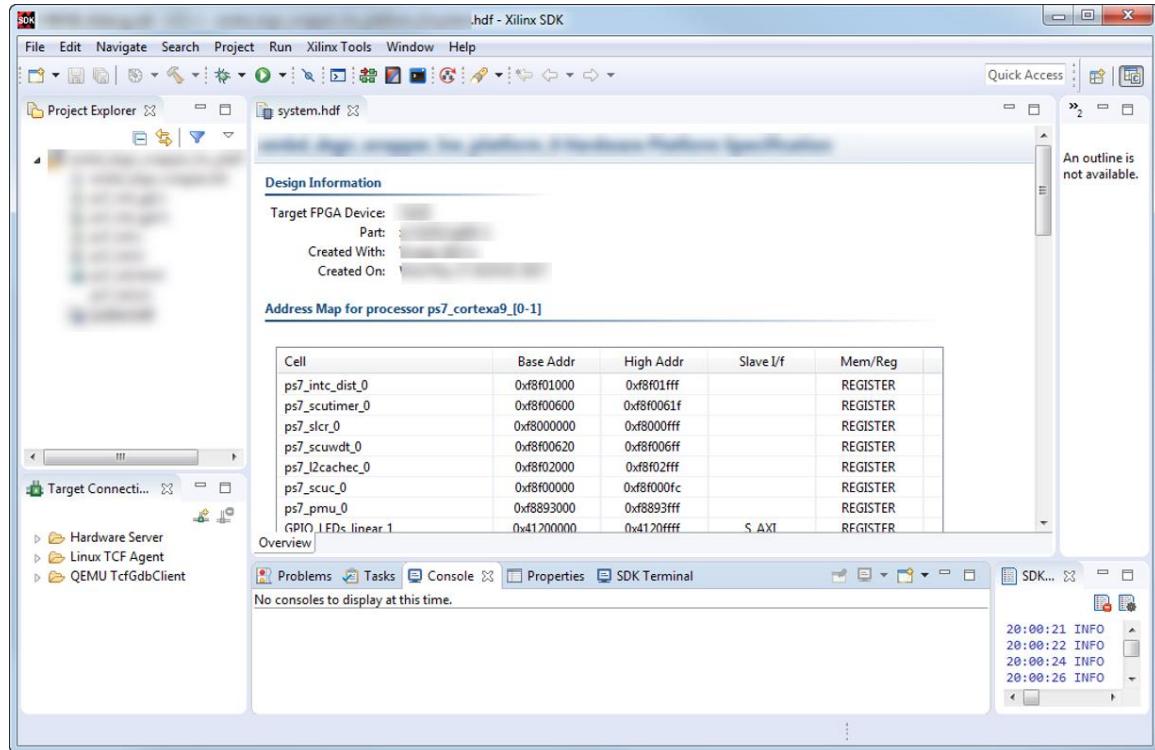


Figure 50: SDK Main Window

- 1-1-6.** Select the **Console** tab in the bottom pane to bring the Console view to the foreground.
This view is useful for tracking the status of builds.

Vivado IP Integrator Operations

In This Section

Creating an IP Integrator Block Design	44
Opening a Block Design	45
Zoom Operations in the IPI Block Diagram Editor	46
Adding a Processor to the Block Design.....	46
Adding a Zynq SoC Processor Block.....	50
Adding a Zynq UltraScale+ MPSoC Processor Block.....	52
Running the Zynq Designer Assistance	55
Customizing the Zynq Family's PS.....	56
Customizing the Zynq UltraScale+ MPSoC PS.....	60
Selecting a Zynq7 PS Preset Template.....	62
Customizing the MicroBlaze Processor	62
Adding IP to an IP Integrator Block Design	65
Customizing IP	67
Making Manual Connections Between Two Objects in the IP Integrator.....	67
Renaming an IP Block.....	69
Adding a Port to an IP Integrator Block Diagram.....	70
Adding an Interface Port to an IP Integrator Block Design.....	72
Making a Pin or Interface External.....	73
Running Connection Automation.....	74
Running Connection Automation for Multiple Modules	75
Running Block Automation	76
Locating Objects in the Board Design.....	77
Mapping Peripheral Addresses.....	79
Generating Output Products	81
Creating a Hierarchical Block in a Block Design.....	82
Adding IP to a Hierarchy Block	84
Expanding the Contents of a Hierarchical Block	84
Opening a Hierarchical Block in a New Tab.....	85
Marking Signals for Debugging	87
Updating IP.....	89
Regenerating the Layout.....	91
Running a Design Rule Check/Design Validation.....	91
Validating the Block Design.....	92
Saving the Block Design.....	92
Generating a Wrapper for a Block Design.....	93

Creating an IP Integrator Block Design

The Vivado IP integrator is a graphical tool that assists you in "stitching" together various pieces of IP. This tool can be used for both embedded and non-embedded designs.

1-1. Create an IP integrator block diagram.

- 1-1-1. Expand **IP INTEGRATOR** in the Flow Navigator if necessary (1).
- 1-1-2. Click **Create Block Design** to start creating a new IP subsystem (2).

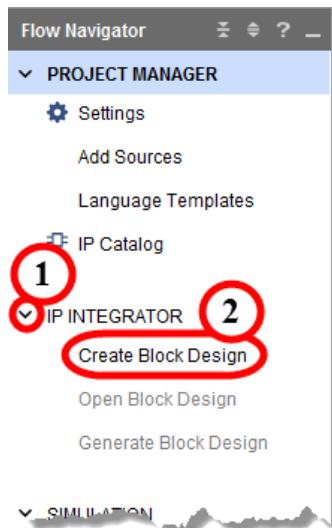


Figure 51: Launching the IP Integrator

- 1-1-3. Name the design **name of your block design** (1) when the Create Block Design dialog box opens.

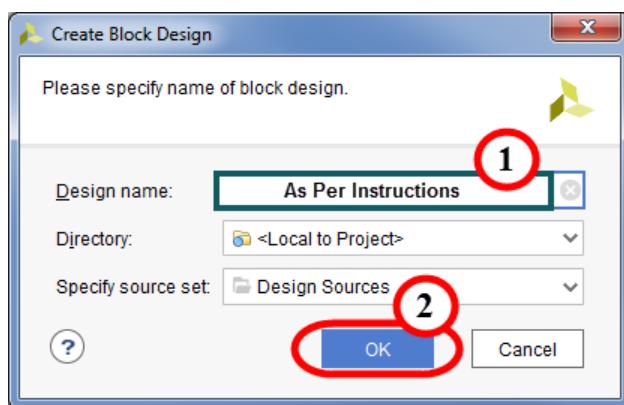


Figure 52: Creating an IP Integrator Block Design

- 1-1-4. Click **OK** to open a new, blank IP integrator canvas (2).

The IP integrator workspace opens with a note in the canvas area inviting you to begin adding IP.

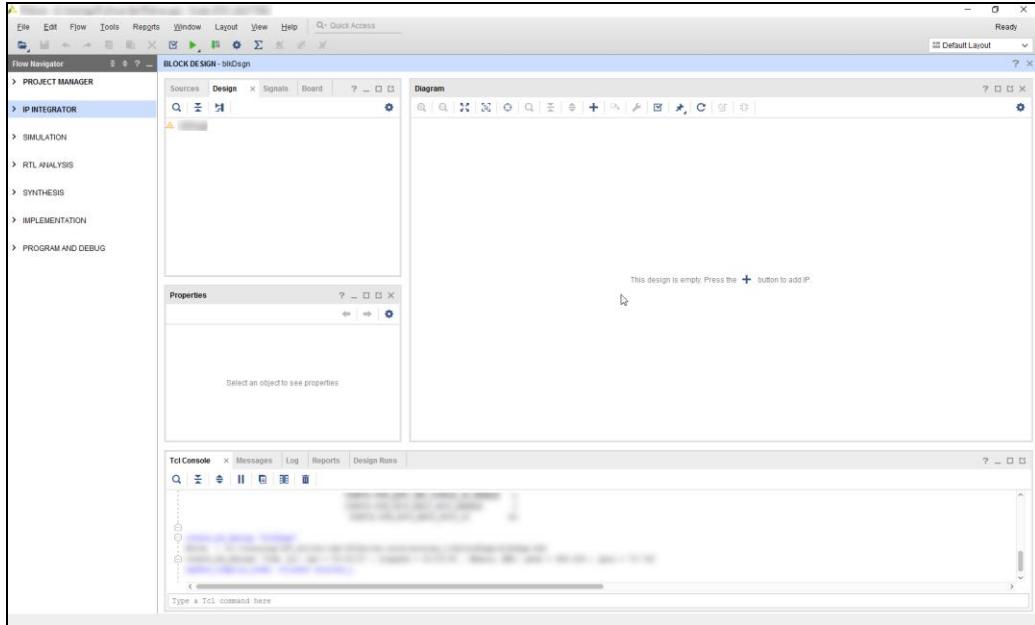


Figure 53: Initial View of the IP Integrator Tool Showing an Informational Message

Opening a Block Design

1-1. Open the *name of your block design* block design.

- 1-1-1. Access the **Sources** > **Hierarchy** view.
- 1-1-2. Locate the block design in the listing of the hierarchical sources.

If it is not readily viewable, click the **Expand all** icon (≡).

- 1-1-3. Double-click the block design entry.

The block design is indicated with a block design icon (█).

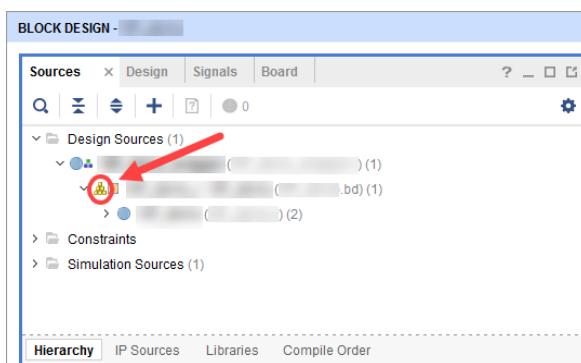
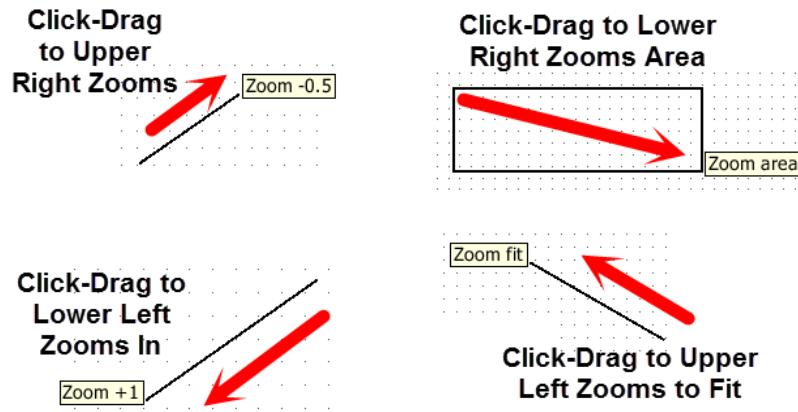


Figure 54: Opening an Existing Block Design

Zoom Operations in the IPI Block Diagram Editor

There are two methods to perform zoom operations in the block diagram editor.

- Method 1: Use the mouse gestures to zoom in for closer examination.



- Method 2: Use the horizontal toolbar to the top of the block design canvas.

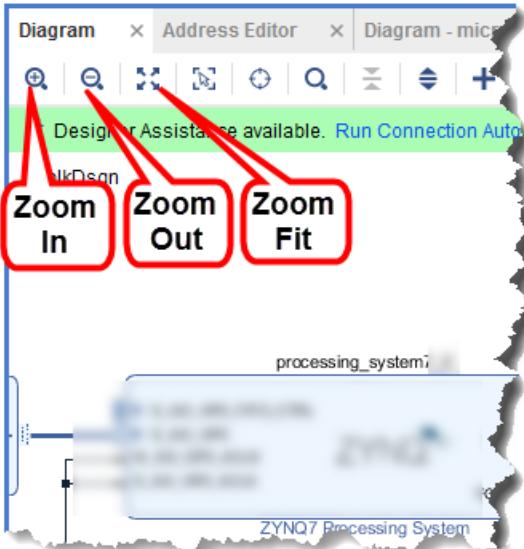


Figure 55: Block Diagram Zoom Toolbar

Adding a Processor to the Block Design

1-1. Add your processor to the IP integrator canvas.

- 1-1-1. Open the IP catalog in one of two ways:
- o Click the **Add IP** icon on the left border of the workspace.

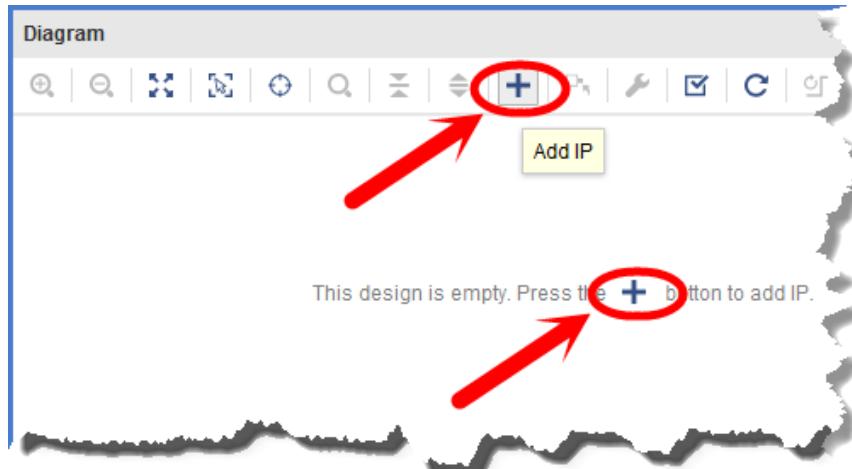


Figure 56: Opening the IP Catalog from the Add IP Icon

-- OR --

- o Right-click any background space in the workspace and select **Add IP**.

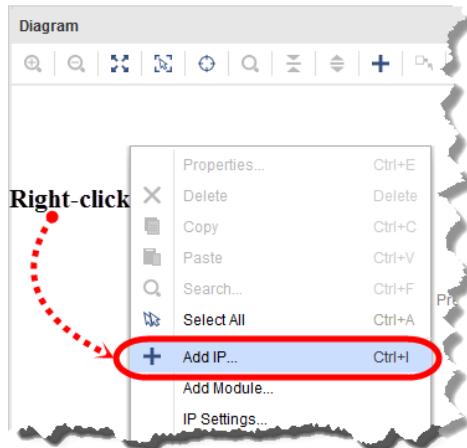


Figure 57: Opening the IP Catalog from Right-Clicking the Workspace

Important Note: Using Window > IP Catalog will add the new IP to the top level of hierarchy of the design—it will NOT add the IP to the diagram! You can, however, float the Window > IP Catalog and drag-and-drop from the catalog onto the canvas of the block design

Once the IP catalog opens, you can search for the processor block.

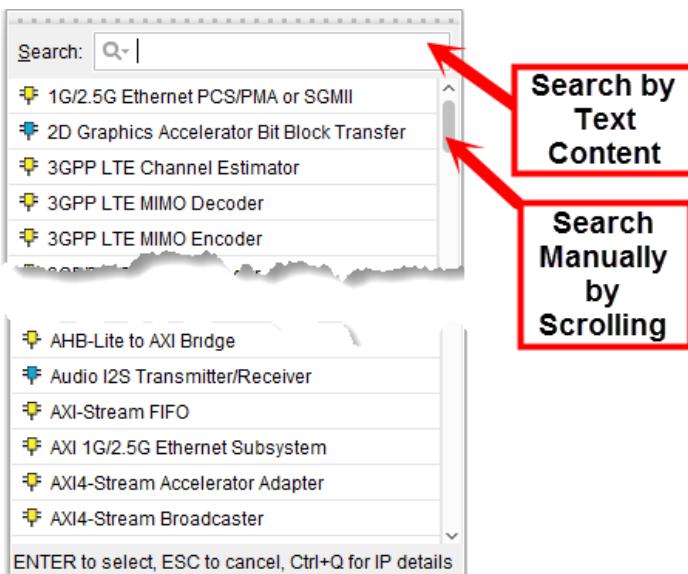


Figure 58: Two Mechanisms to Search for IP

- 1-1-2.** Enter part of the processor name into the Search field to narrow the search parameters (1).

For example, **zynq** will show the PS for the selected Zynq family or **microblaze** for the MicroBlaze processor. Note that the PS IP will only appear when a family containing this IP or a board with this part has been selected for the design.

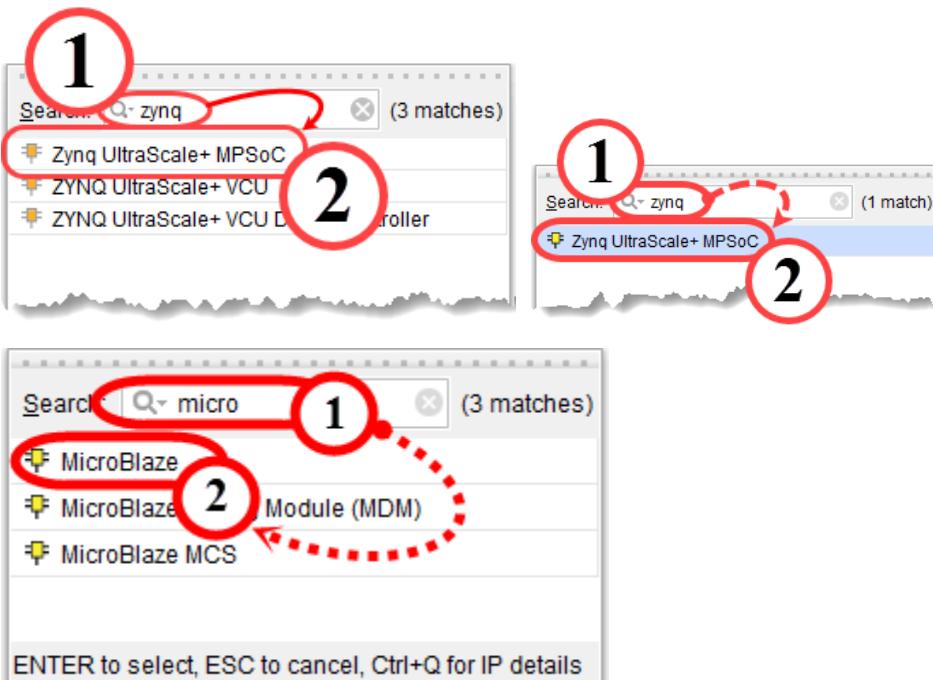


Figure 59: Narrowing Search Parameters (MicroBlaze Processor)

- 1-1-3. Double-click the processor name to add its IP block to the design (2).

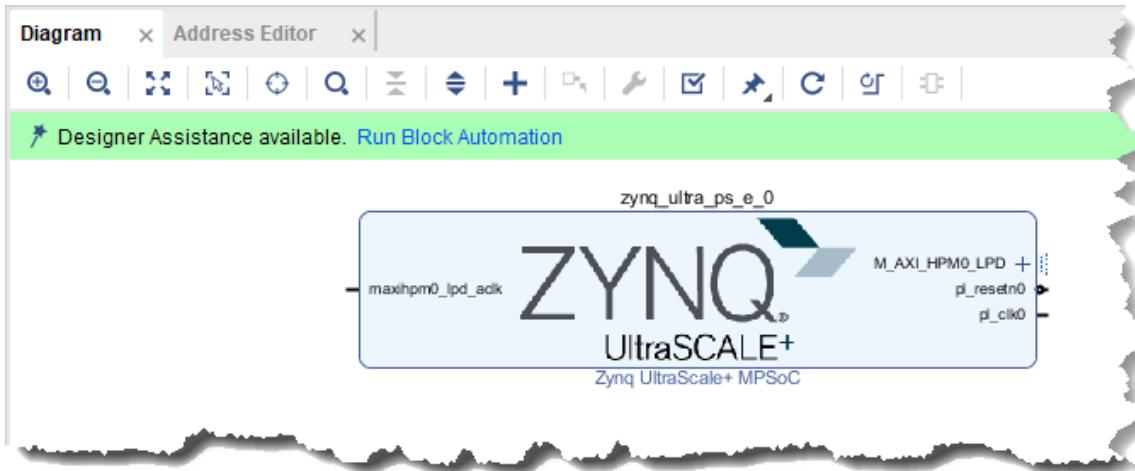


Figure 60: Zynq-7000 PS Block Symbol



Figure 61: Zynq UltraScale MPSoC Block Symbol

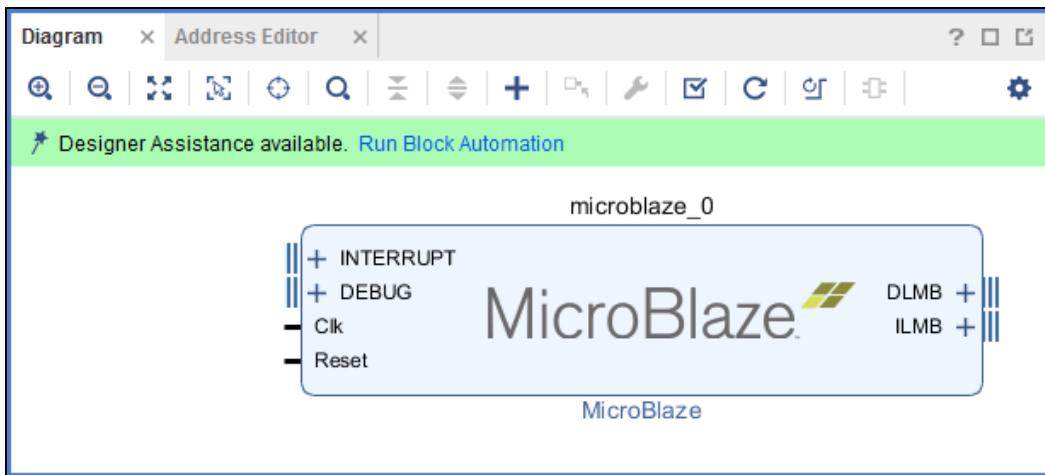


Figure 62: MicroBlaze Processor IP

Adding a Zynq SoC Processor Block

1-1. Add a Zynq PS processor block to the design. While the specifics of the processing systems found in the Zynq-7000, MPSoC/RFSoC, and Versal™ devices differ, the process of instantiating them is the same.

There are several ways to open the IP catalog. Select one of the following procedures to add IP to the canvas.

1-1-1. Click **Add IP to open the IP catalog.**

- Click the **Add IP** icon in either the middle of the canvas or the toolbar.

This icon can be found in the middle of the canvas only when you are starting with a blank page.

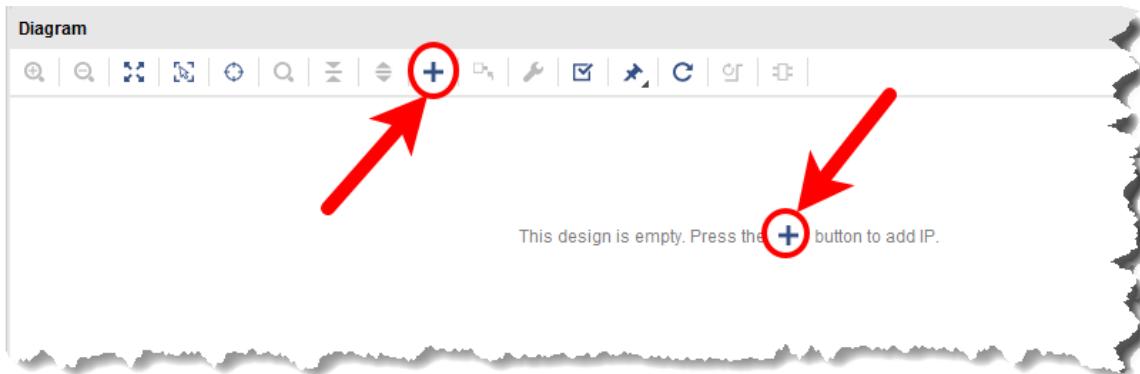


Figure 63: Opening the IP Catalog from the Initial Information Bar Display

-- OR --

- Right-click any background space in the workspace and select **Add IP**.

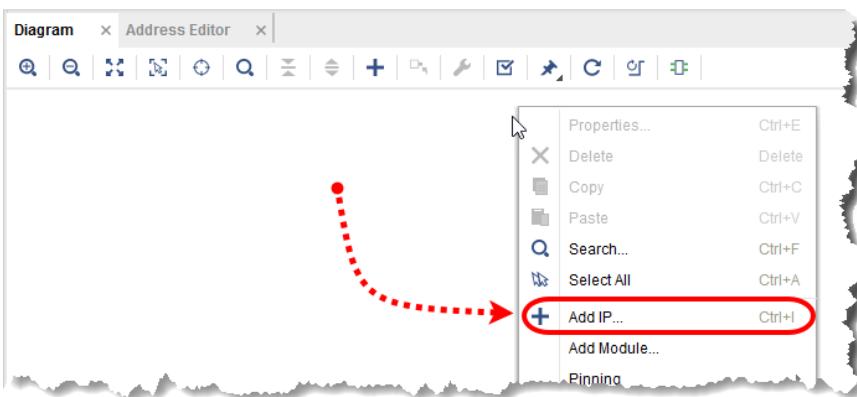


Figure 64: Opening the IP Catalog from Right-Clicking the Workspace

-- OR --

- Press <**Ctrl + I**> to access the IP catalog.

Note that although the convention is to show the capital letter, the key combination uses a lower case 'i'.

Once the IP catalog opens, you can search for the Zynq SoC processor block.

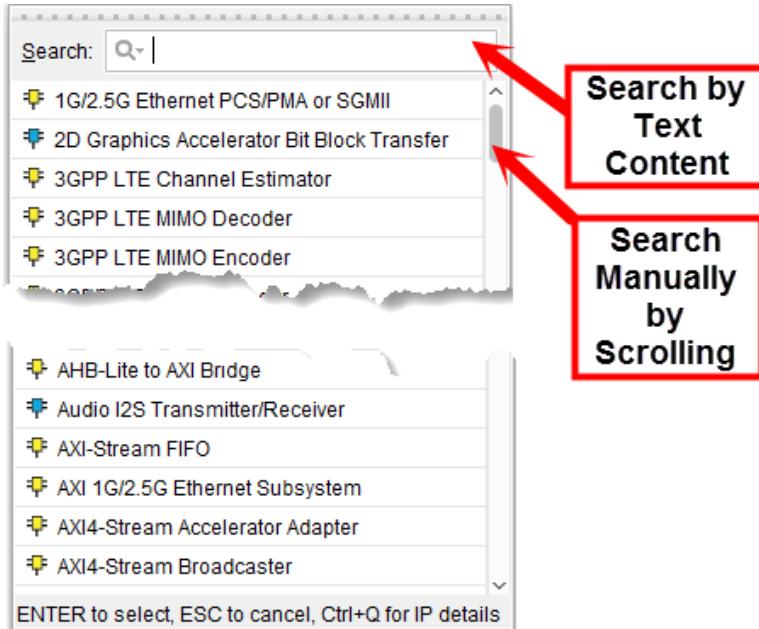


Figure 65: Two Mechanisms to Search for IP

- 1-1-2. Enter **zynq** into the Search field to narrow the search parameters (1).

The proper IP will appear depending on the board or device that you selected. The PS of the Zynq-7000 device will appear as ZYNQ7 while the MPSOC/RFSOCs will show as Zynq UltraScale+ MPSOC, and so forth. Only the IP supporting the device you selected will appear in the IP list.

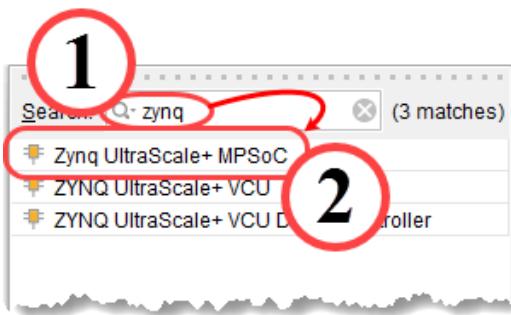


Figure 66: Narrowing the Search Parameters (Zynq-7000 PS)

This is the IP for the entire processing system (PS).

- 1-1-3. Double-click the IP name to add the processing system block to the design (2).

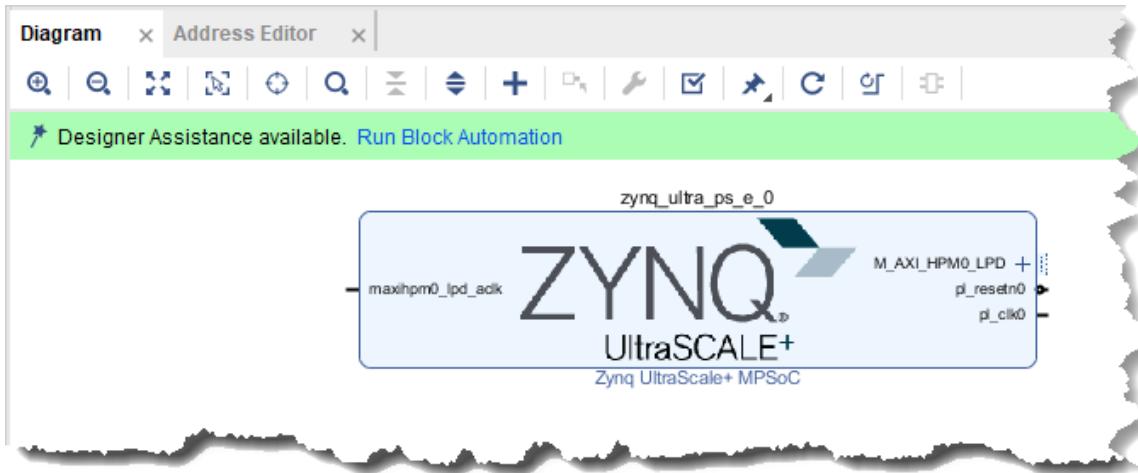


Figure 67: Zynq-7000 PS Block Symbol

Adding a Zynq UltraScale+ MPSoC Processor Block

- 1-1. Add a Zynq UltraScale+ MPSoC processor block to the design.

There are several ways to open the IP catalog. Since this is a blank canvas, you are invited to click the add IP icon in the middle of the canvas.

- 1-1-1. Click **Add IP** to open the IP catalog.

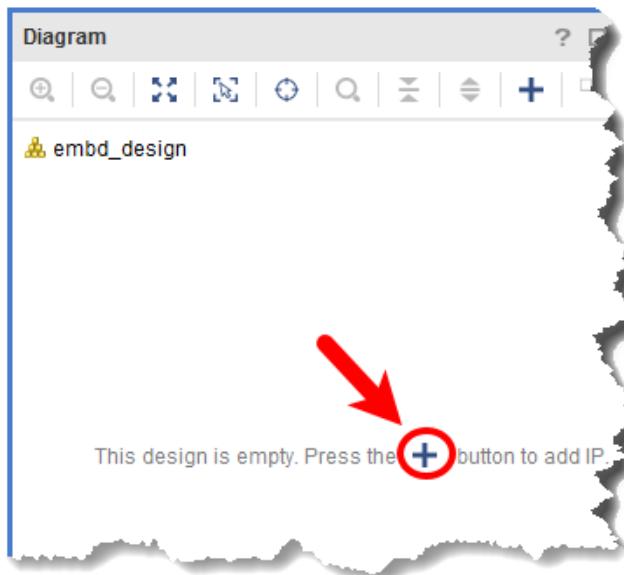


Figure 68: Opening the IP Catalog from the Initial Information Bar Display

Regardless of whether the design already has IP placed, you can always open the IP catalog in one of several ways:

- o Click the **Add IP** icon in the toolbar of the workspace.

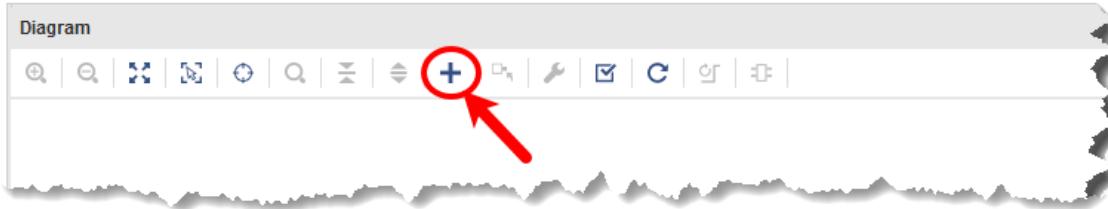


Figure 69: Opening the IP Catalog from the Add IP Icon

-- OR --

- o Right-click any background space in the workspace and select **Add IP**.

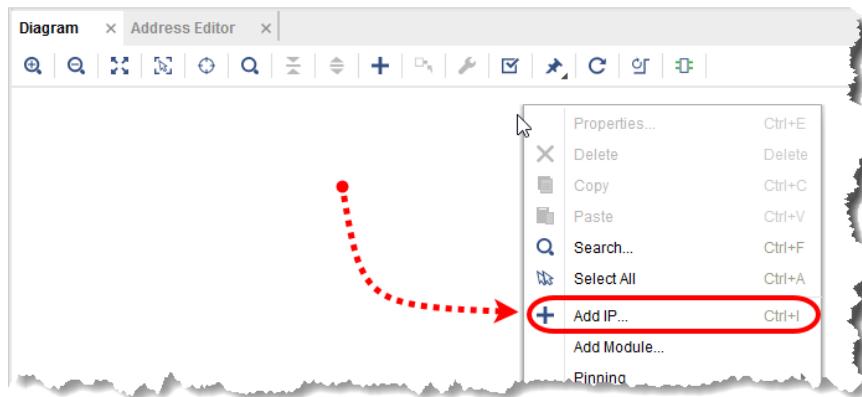


Figure 70: Opening the IP Catalog from Right-Clicking the Workspace

-- OR --

- o Press <Ctrl + I> to access the IP catalog.

Note: Using Windows > IP Catalog will add the new IP to the top level of hierarchy of the design — it will NOT add the IP to the diagram! You can, however, float the Windows > IP Catalog and drag-and-drop from the catalog onto the canvas of the block design.

Once the IP catalog opens, you can search for the Zynq UltraScale+ MPSoC processor block.

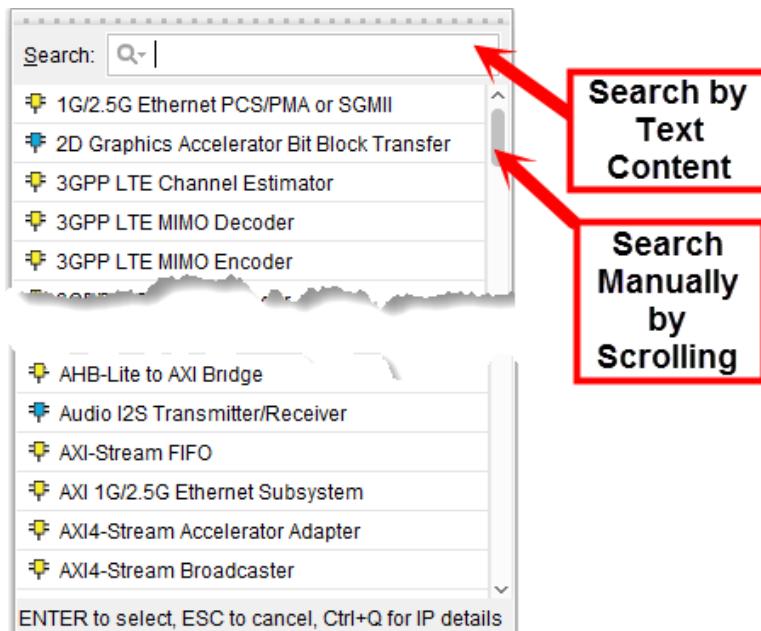


Figure 71: Two Mechanisms to Search for IP

- 1-1-2.** Enter **zynq** into the Search field to narrow the search parameters (1).

The Zynq UltraScale+ MPSoC PS IP will only appear when Zynq UltraScale+ MPSoCs or boards with these parts have been selected for the design.

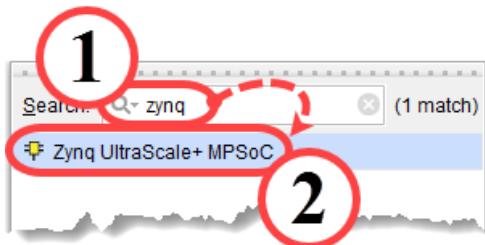


Figure 72: Selecting the Zynq UltraScale+ MPSoC IP

This is the IP for the entire processing system (PS).

- 1-1-3.** Double-click the **Zynq UltraScale+ MPSoC** IP entry to add the Zynq UltraScale+ MPSoC block to the design (2).



Figure 73: Zynq UltraScale+ MPSoC Block in the Block Diagram

Running the Zynq Designer Assistance

Many IP blocks, including the Zynq SoC IP block are supported by Designer Assistance. Designer Assistance automates many of the commonly used basic connections and/or configurations.

1-1. Use Designer Assistance to make preliminary connections to the Zynq SoC IP block.

- 1-1-1.** If the block design is not already open, select the **Diagram** tab in the Block Design pane to view the block design.

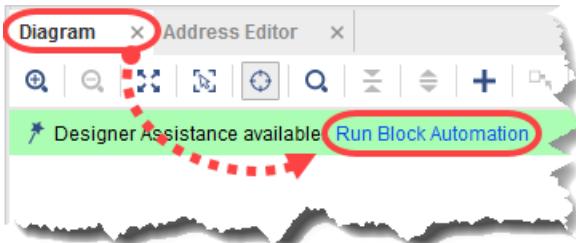


Figure 74: Selecting the Diagram Tab

- 1-1-2.** Click **Run Block Automation** to launch Designer Assistance.

The Run Block Automation dialog box opens.

- 1-1-3.** Deselect **Apply Board Preset** because you already set the board preset in the Re-customization wizard.

Having this enabled here will overwrite any work you may have done in re-customization.

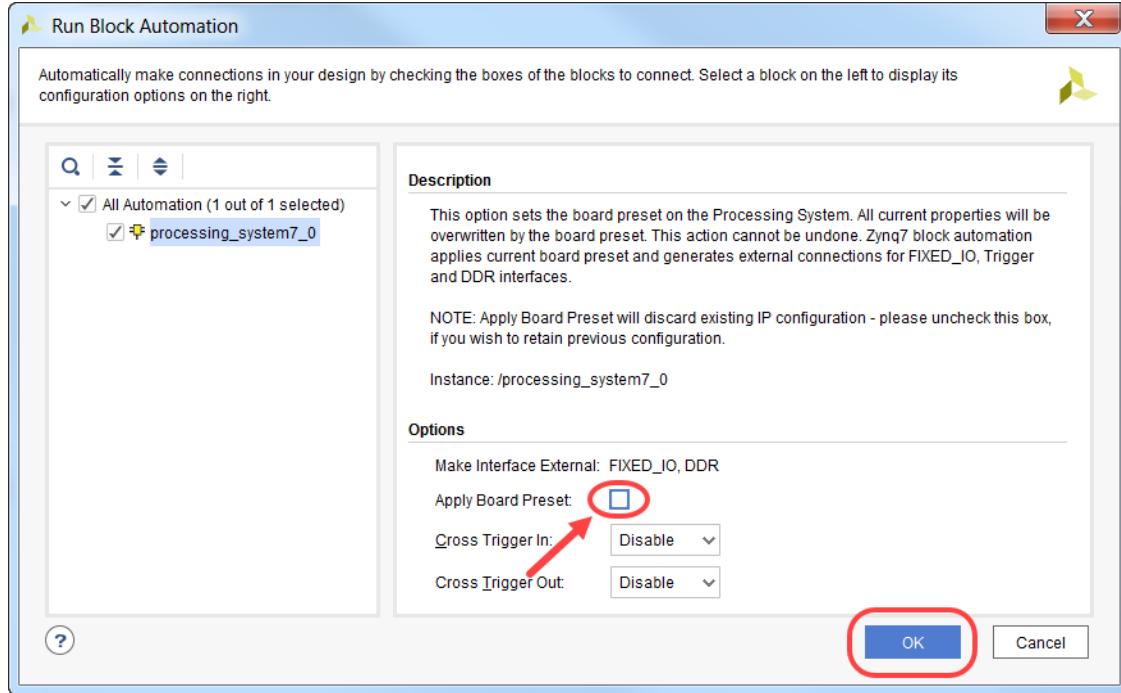


Figure 75: Running the Designer Automation Assistance on the Zynq PS

The dialog box lists the connections that will be created: the FIXED_IO (MIO connections) and the DDR interface connections.

- 1-1-4. Click **OK** to make these connections between the processing_system7 block and the package pins.

Customizing the Zynq Family's PS

The Zynq device's processing system (PS) contains a large number of customizable features. The following instructions will illustrate how to access various sections of the PS, not necessarily indicating which settings to configure. Note that the Re-customization Wizard differs between the different processing systems in the various Zynq devices.

1-1. Access the Re-customization dialog box.

- 1-1-1. Double-click the PS block.

This could be the ZYNQ7 or ZYNQ MPSOC block or any PS block depending on the device family.

-- OR --

Right-click the Zynq PS icon and select **Customize Block**.

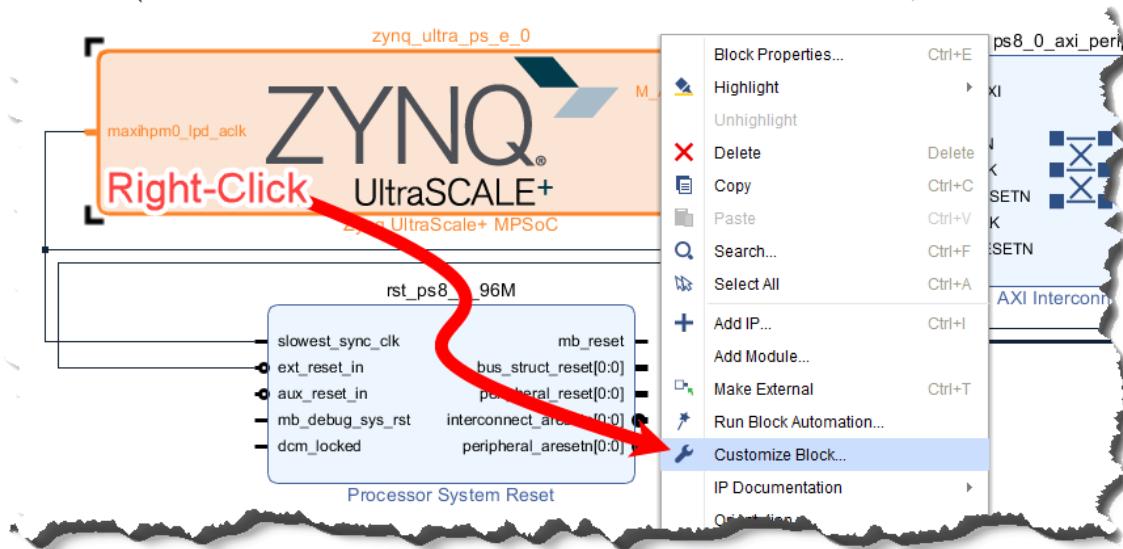


Figure 76: Opening the Re-customization Wizard for the Zynq SoC PS

The Re-customization dialog box opens to reveal the PS's block design.

1-2. Import board-specific settings.

If you are using an evaluation board, you may want to load the settings for this board as it will contain many of the parameters specific to that board (such as DDR memory timing parameters, enabled peripherals supported by that board, etc.)

You can also create a .bd file for your custom board if you want and import those settings.

The Zynq UltraScale+ MPSoC presets behave differently than the Zynq-7000 device presets as the Zynq-7000 device presets contain settings for the various evaluation boards and the Zynq UltraScale+ MPSoC presets enable you to create and recall your own specific settings.

- 1-2-1. Click **Presets** to access the pre-defined values associated with one of the supported boards.

Note: If you are building a board of your own, it is possible to create your own preset for that board. Presets are very convenient as they can describe the DDR settings as well as enable the peripherals and their pin mappings for the specific board. This saves a tremendous amount of effort for the designer.

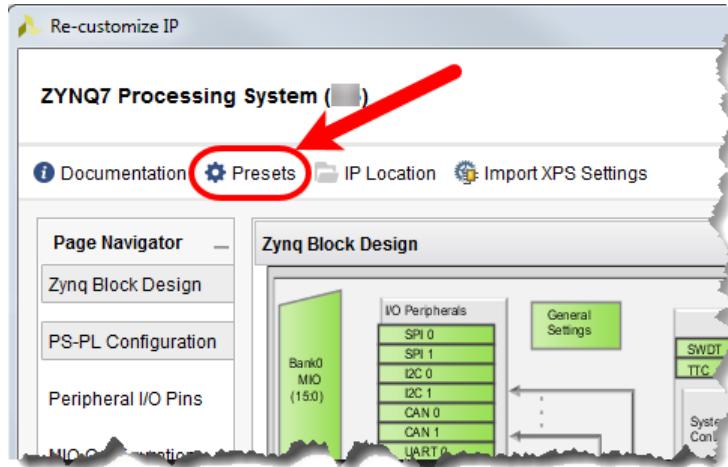


Figure 77: Accessing the Presets (Zynq PS)

1-2-2. [Zynq-7000 device users]: Select the template for the board you are targeting.

All the Xilinx boards that support the device that was selected during the project creation are shown under the System Template (Presets) drop-down list. Even if you selected the part by board, you have the opportunity to select a different board that has the same part on it.

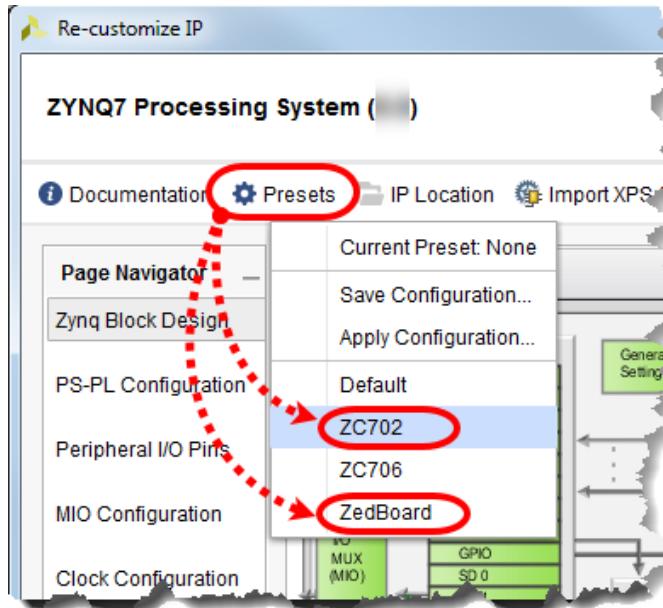


Figure 78: Selecting a Preset Template (ZedBoard and ZC702 Shown as Examples)

[Zynq MPSoC/RFSoc device users]: If you have a Tcl script that describes the configuration of the PS, you can load it by select **Presets > Apply Configuration**.



The next instruction provides you with general guidance as to how to customize various aspects of the PS. At the end of this instruction you will be provided with specific guidance regarding how you are to customize the PS.

1-3. Select the page or specific block to re-customize.

- 1-3-1. Click the topic in the Page Navigator or any green (active) box from the Zynq Block Diagram.

Along the left are the navigational tabs that you will use to access different groups of parameters.

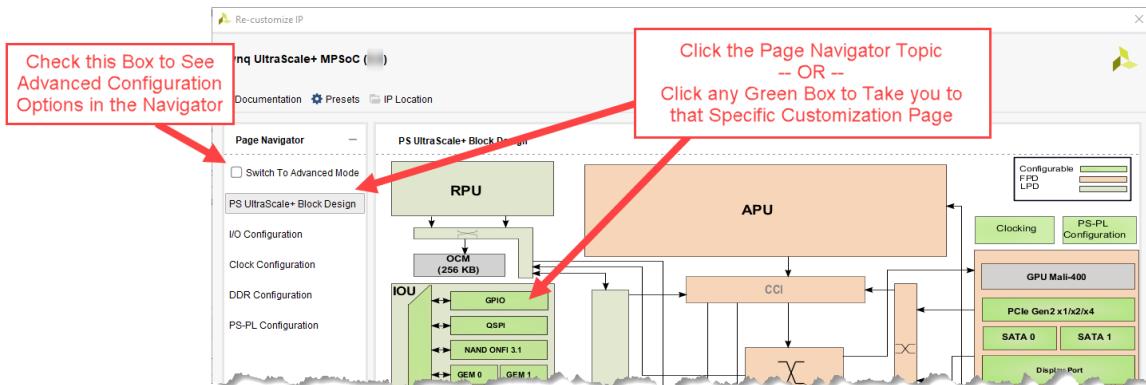


Figure 79: Navigating the Zynq PS Recustomization Dialog Box

- PS-PL Configuration: This page contains all the signals that cross the PS-PL boundary. For the Zynq-7000 devices, these signals are broken down further into:
 - General: Contains default baud rates for the UARTs, FTM trace buffer settings, PS-PL cross triggering enables, enables for the clock triggers, and reset.
 - DMA Controller: Contains enables for the four peripheral request interfaces.

- GP Master AXI Interface: Enables/disables access to the GP Master AXI Interfaces to the PL.
- GP Slave AXI Interface: Enables/disables access to the GP Slave AXI Interfaces from the PL.
- HP Slave AXI Interface: Enables/disables access to the GP Slave AXI Interfaces from the PL and sets the port width.
- ACP Slave AXI Interface: Enables/disables access to the ACP Slave AXI Interface from the PL.
- Peripheral I/O Pins
- MIO Configuration
- Clock Configuration
- DDR Configuration
- SMC Timing Calculation
- Interrupts (Zynq UltraScale+ has this setting under PS-PL Configuration)

Customizing the Zynq UltraScale+ MPSoC PS

The Zynq UltraScale+ MPSoC PS contains a large number of customizable features. The following instructions will illustrate how to access various sections of the Zynq UltraScale+ MPSoC PS, not necessarily indicating which settings to configure.

1-1. Access the Re-customization dialog box.

1-1-1. Double-click the **Zynq UltraScale+** icon.

-- OR --

Right-click the **Zynq UltraScale+** icon and select **Customize Block**.

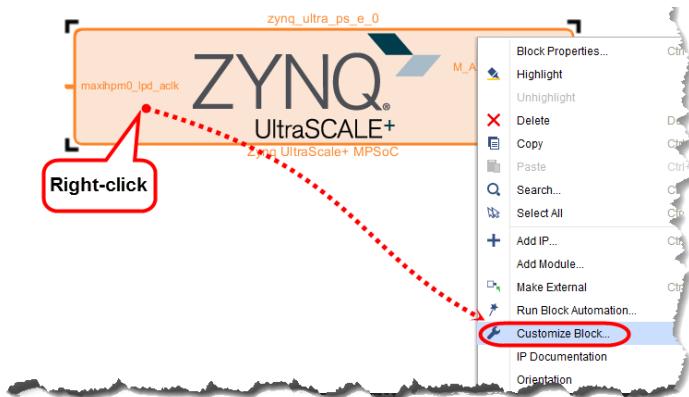


Figure 80: Customizing the Zynq UltraScale+ MPSoC

The Re-customization dialog box opens to reveal the Zynq block design.

1-2. Select the page or specific block to re-customize.

- 1-2-1. Click the topic in the Page Navigator or any green (active) box from the Zynq Block Diagram.

Along the left are the navigational tabs that you will use to access different groups of parameters.

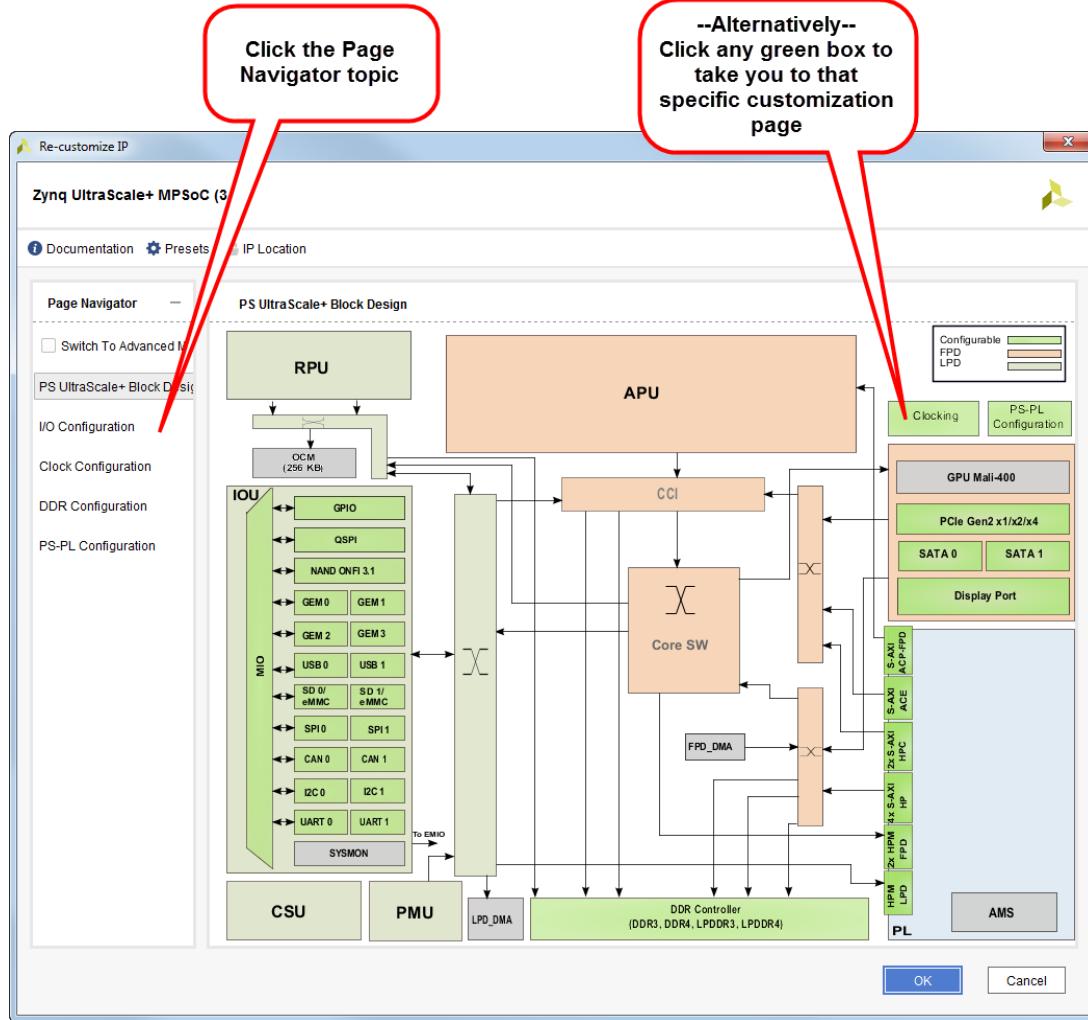


Figure 81: Re-customize IP Dialog Box

- PS-PL Configuration: This page contains all the signals that cross the PS-PL boundary. These signals are broken down further into:
 - General: Contains interrupts from PS to PL and PL to PS, Address Fragmentation parameters and others such as the RTC, DMA, and Live Audio and Video settings.
 - PS-PL Interfaces: Contains enables for all of the available AXI masters and slaves including the ACP and ACE slave AXI interfaces.
 - Debug: PS - PL cross-trigger inputs and outputs and general-purpose inputs and outputs.

- o I/O Configuration
- o Clock Configuration
- o DDR Configuration

Selecting a Zynq7 PS Preset Template

1-1. Configure the ZYNQ7 processing system using the Preset template.

1-1-1. Double-click the **ZYNQ7 Processing System** IP block to open the Re-customize IP dialog box.

1-1-2. Click **Presets** in the Re-customize IP dialog box.

This preset contains settings appropriate for your board, such as peripheral pin locations, DDR memory parameters, etc.

1-1-3. Select **the preset for your board**.

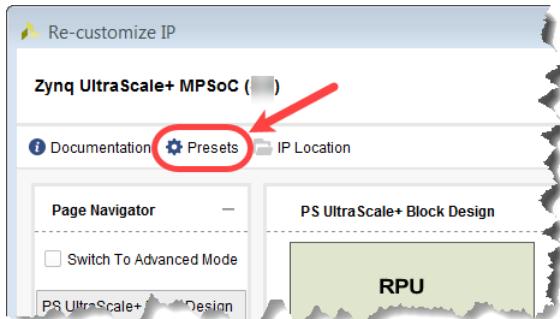


Figure 82: Selecting a Preset Template

1-1-4. Click **OK** to load the values associated with this preset.

Customizing the MicroBlaze Processor

1-1. Configure the MicroBlaze processor system with a Typical configuration.

The Typical configuration is one of several customizable configurations. It includes an MDM interface for debugging, instruction and data caches for use with DDR memory, and exception/interrupt handling. This configuration provides a balance between frequency, area, and overall performance.

- 1-1-1. Double-click the **MicroBlaze** IP block or right-click the **MicroBlaze** IP block and select **Customize IP**.

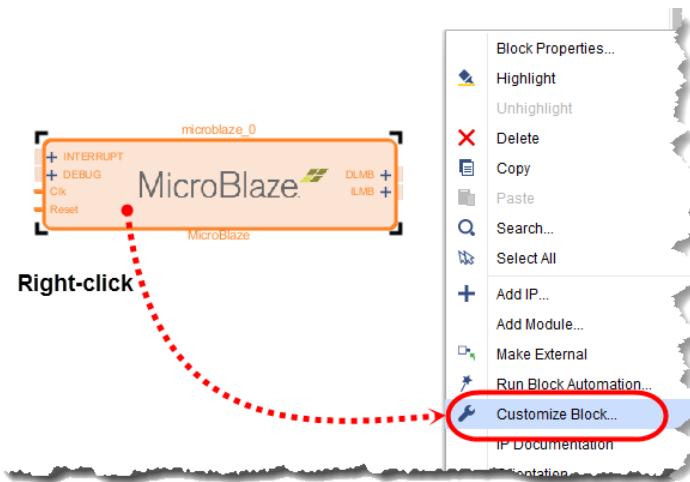


Figure 83: Starting Customization on the MicroBlaze Processor

The Re-customization IP dialog box opens.

- 1-1-2.** Select **Typical** from the Predefined Configurations > Select Configuration drop-down list.

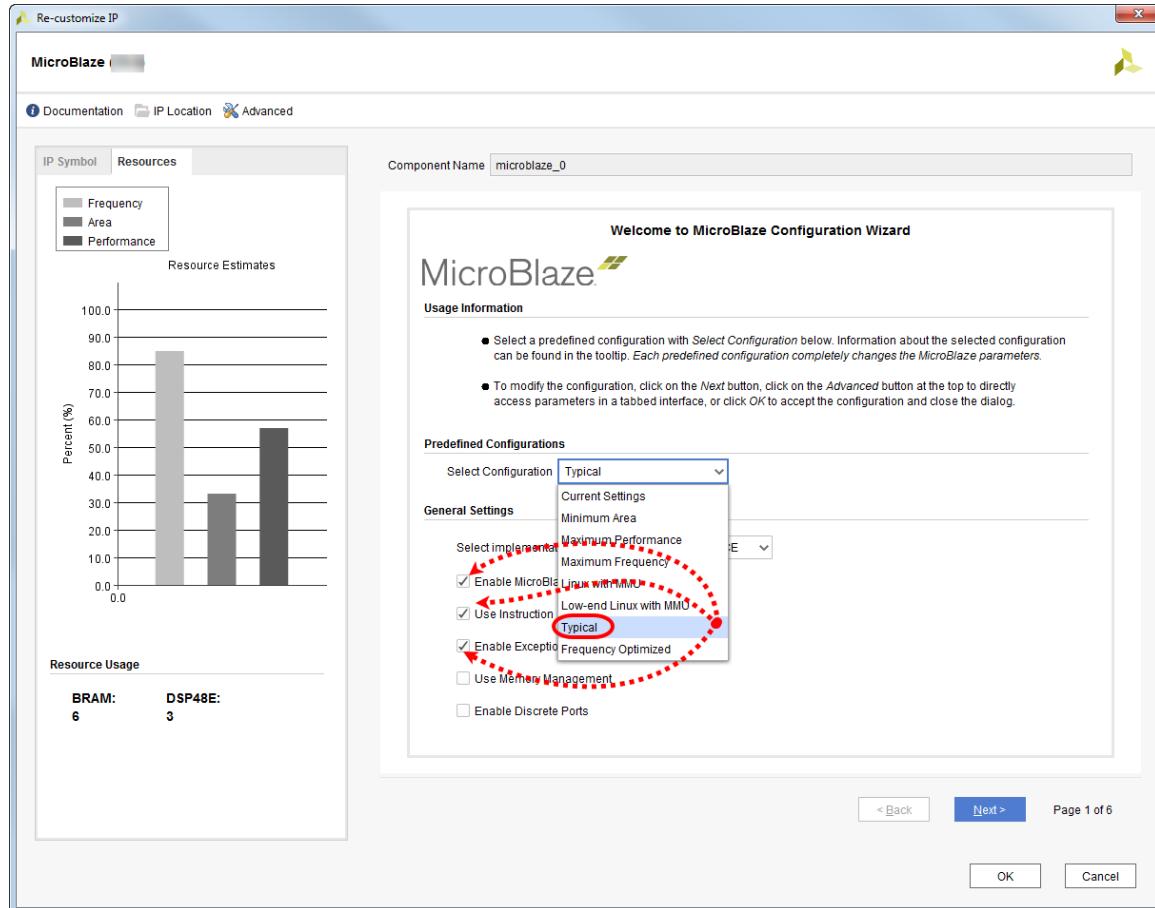


Figure 84: Selecting the Typical Configuration for the MicroBlaze Processor

Notice that the Enable Microprocessor Debug Module, Use Instruction and Data Caches, and Enable Exceptions options are automatically selected as part of the Typical configuration. Cache memory is good to have when dealing with slower memories such as DDR and Flash;

- 1-1-3.** Review the default settings for the Typical configuration.

The MicroBlaze processor can be built as either a 32-bit or 64-bit implementation.

- 1-1-4.** Select the 32-bit Implementation.

- 1-1-5.** Leave the General Settings pull-down option at **PERFORMANCE**.

While the predefined configurations and general settings (implementation optimization) choices define the general configuration and options of the MicroBlaze processor, other options can be selected and overloaded into the default settings made by these configurations. Generally, these options address supporting hardware for the core processor's capabilities and includes debugging capabilities through the MDM, caches, memory management units, etc.

- 1-1-6. If selected, deselect the **Use Instruction and Data Cache** option.
- 1-1-7. Ensure that the **Enable MicroBlaze Debug Module Interface** and **Enable Exceptions** options are selected.

This is done to illustrate how *preconfigured* settings can be overridden by the user. Cache is most useful when applied in conjunction with DDR memory. If the MicroBlaze processor is running entirely out of block RAM, the caches may actually degrade the performance of the system.

Note that there are several pages where modifications to this configuration can be made. You can explore these settings as time permits.

- 1-1-8. Click **OK** to save this configuration.

Adding IP to an IP Integrator Block Design

1-1. Open the IP catalog.

- 1-1-1. The IP catalog can be opened in one of several ways:
 - o Right-click any background space in the workspace and select **Add IP**.

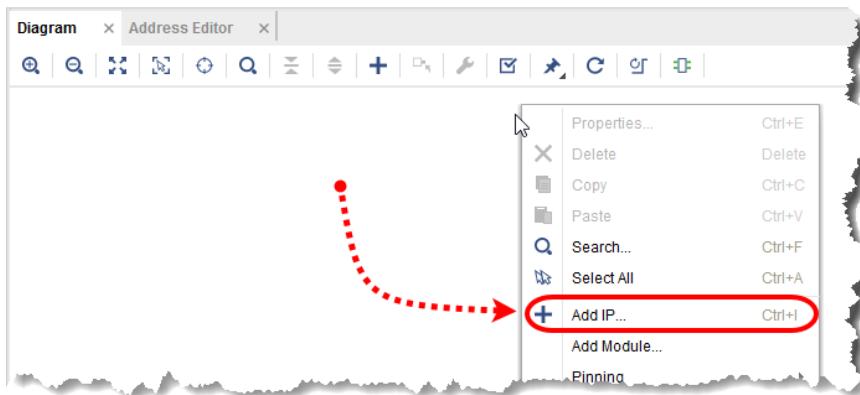


Figure 85: Opening the IP Catalog from Right-Clicking the Workspace

- o Press <**Ctrl + I**> to open the IP catalog.

- o If the design is empty, you will be invited to add IP by clicking either '+' icon.

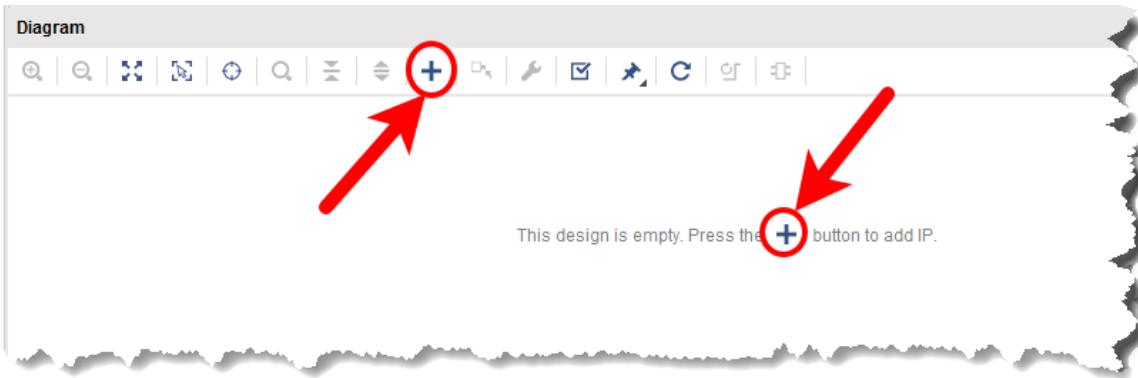


Figure 86: Opening the IP Catalog from the Initial Information Bar Display

Important Note: Using Windows > IP Catalog will add the new IP to the top level of hierarchy of the design—it will NOT add the IP to the diagram! It is, however, possible to float this window and drag-and-drop IP into the Block Design canvas.

1-2. Once the IP catalog opens, search for *the desired piece of IP*.

- 1-2-1. Type all or part of the IP name into the Search field.
- 1-2-2. Scroll to *the desired piece of IP*.

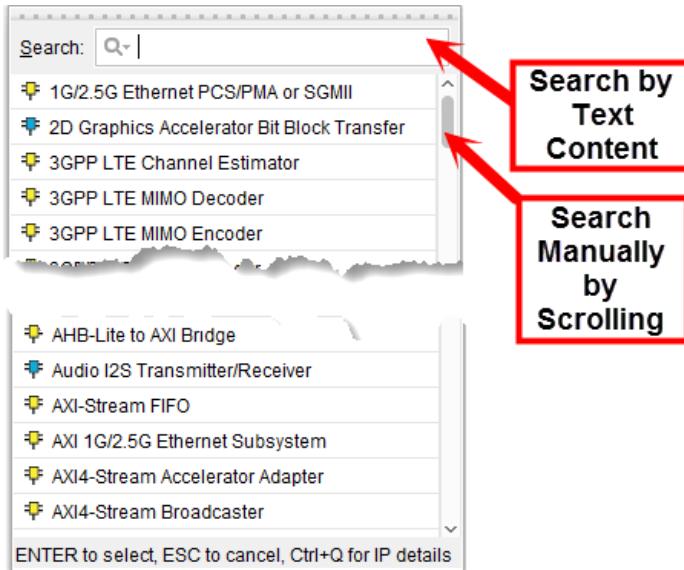


Figure 87: Two Mechanisms to Search for IP

- 1-2-3. Double-click the name of the IP to add it to the design (if you use this method, the IP catalog will close after the IP has been added to the design).

or

Drag-and-drop the IP into the workspace.

The IP catalog will remain open once the IP has been added to the design. This is a convenient way to quickly add multiple pieces of IP. Pressing the <Esc> key will close the IP catalog.

Customizing IP

Almost every piece of IP is customizable. This provides a great deal of flexibility to the designer to create a system that is tailored to specific needs.

1-1. Open the Re-customization Wizard for the desired piece of IP.

- 1-1-1. Double-click the IP block.

Alternatively, you can right-click the IP block and select **Customize Block** from the context menu.

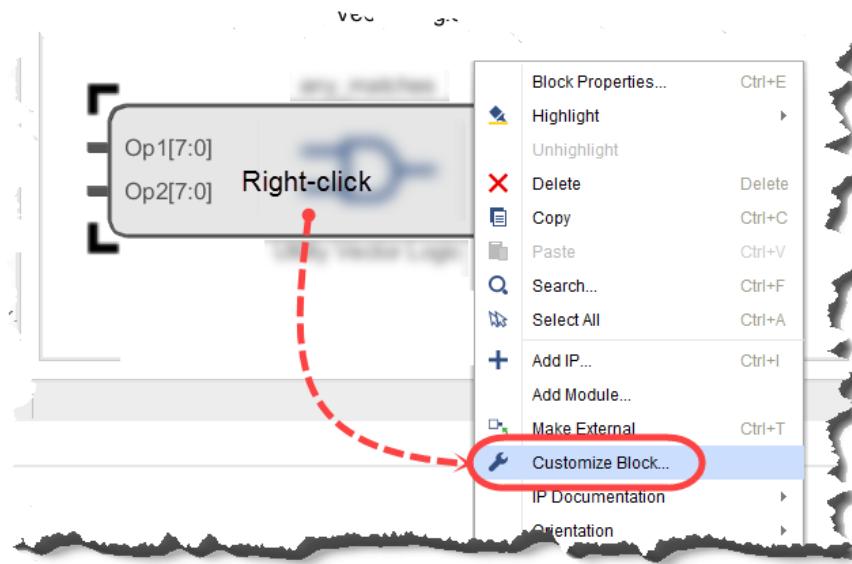


Figure 88: Opening the Recustomization Wizard for a Piece of IP

The Re-customization Wizard for that piece of IP opens.

Making Manual Connections Between Two Objects in the IP Integrator

Making connections is a simple process in the IP integrator tool. Hover over a port or interface on a block with your mouse (the cursor changes to a pencil icon from the arrow icon). You can then click-drag the connection to a legal port or interface. As the connecting wire is stretched to its destination, an assistant runs in the background and places a green check mark next to all the ports or interfaces that are allowed to be connected when the cursor approaches a legitimate connection point.

There are two mechanisms for making these connections: Using the Run Connection Automation feature, which is available for most IP, and manual connections. The latter is addressed here.

1-1. Connect two IP blocks using the manual connection technique.

- 1-1-1. Click **a port or interface** to begin the connection process.

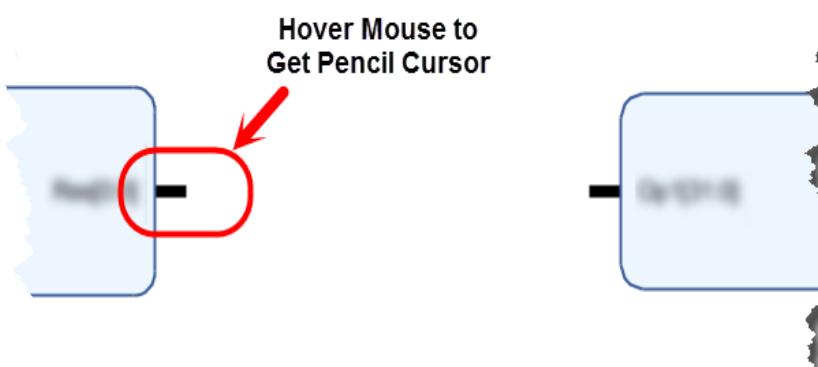


Figure 89: Hovering the Mouse Over a Port to Get the Pencil Icon

The port or interface is highlighted, showing that it has been selected and the cursor changes to a pencil, indicating that it is ready to draw/connect a wire.

- 1-1-2. Drag the pencil to **another port or interface**.

As the cursor approaches a valid port or interface, a green arrow appears next to legal connection points.



Figure 90: Finding Legal Connection Points

- 1-1-3. Release the mouse button to make the connection.



Figure 91: Finishing the Connection

Interfaces are connected in exactly the same way, except that a single wire or vector is not connected. Rather, a bundle of signals and/or buses is connected.

Note: If you want to exit this drawing mode, press <Esc>.

Renaming an IP Block

Renaming an IP block often helps the designer keep track of what that particular piece of IP does rather than what it *is*.

1-1. Rename the IP block *with the new name for this piece of IP*.

- 1-1-1.** Select the desired IP block in the Block Design window (1).

The current name of the selected IP block appears in the Block Properties window when the General tab is selected.

- 1-1-2.** Enter the new name for this specific piece of IP into the Name field (2).

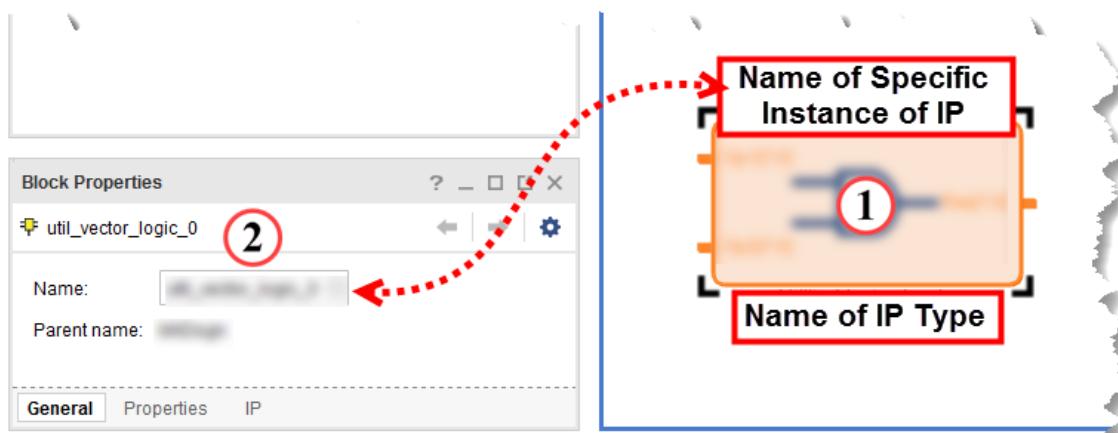


Figure 92: Renaming an IP Block

- 1-1-3.** Press <Enter> to save the change.

Adding a Port to an IP Integrator Block Diagram

Ports are constructs that allow signals to pass between hierarchical levels. Ports can be combined together to form interfaces. Here you will learn how to create a single port. Ports can take on a variety of special cases, all of which can be selected through the *type* parameter.

Port Type	Meaning
Clock	Clock signals: placed on clock networks; enter the frequency in the Frequency field
Reset	Reset network

Port Type	Meaning
Interrupt	For embedded systems: ports marked as interrupt are entered into the interrupt select table for the processor(s)
Data	General-purpose data signals
Clock Enable	Clock enable signals: used with clocks
Other	Undefined signals: cannot be connected to ports other than those marked as "other"

1-1. Create a port.

- 1-1-1. Right-click an unoccupied section of the Vivado IP integrator canvas (1).
- 1-1-2. Select **Create Port** (2).



Figure 93: Creating a New Port

The Create Port dialog box opens.

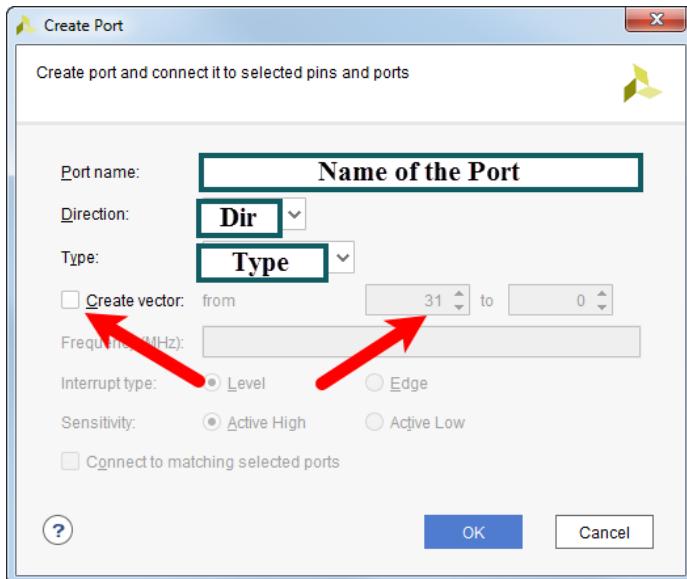


Figure 94: Create Port Dialog Box

- 1-1-3. Enter **the name of the port** in the Port name field.
- 1-1-4. Select the port direction as **input, output, or inout/bidirectional**.
- 1-1-5. Select the type to be **according to the table below**.
- 1-1-6. Ensure that the **Create vector** box is checked and If this port is more than one signal wide, select the Create vector option and indicate the range of the vector.
- 1-1-7. Click **OK**.

Adding an Interface Port to an IP Integrator Block Design

Interface ports are mechanisms that allow a grouping of signals to pass between hierarchy modules or hierarchical levels.

1-1. Create an interface port.

- 1-1-1. Right-click an unoccupied section of the IP integrator canvas (1).
- 1-1-2. Select **Create Interface Port** from the context menu (2).

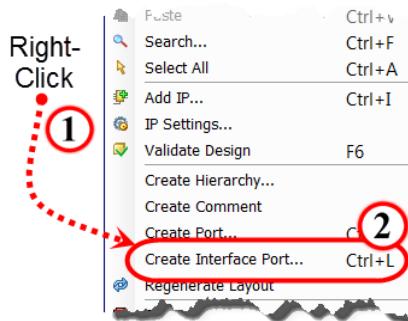


Figure 95: Creating an Interface Port

Note: You can also press <Ctrl + L>.

The Create Interface Port dialog box opens.

- 1-1-3. Enter the name of the interface port.
- 1-1-4. Enter or select the VLNV.
VLNV stands for vendor, library, name, and version. Typically this option is populated by the tool.
- 1-1-5. Select **if the interface port is a master, slave, or monitor**.

1-1-6. Select to connect to the specified port or just have the interface port floating.

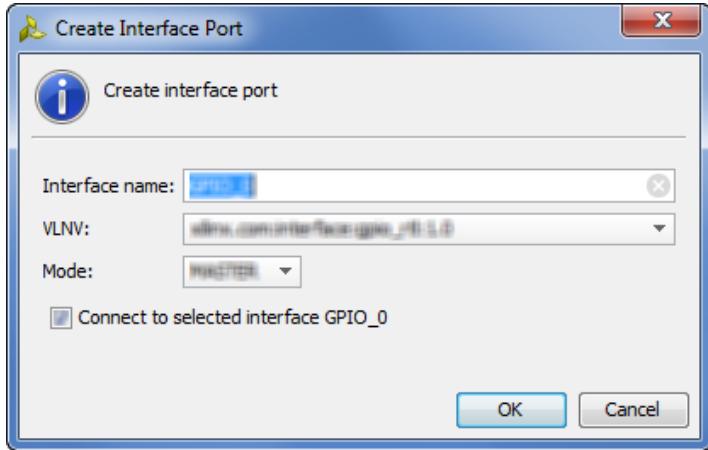


Figure 96: Create Interface Port Dialog Box

1-1-7. Click OK.

Making a Pin or Interface External

There are several mechanisms for making a pin or interface external. The easiest way is described here.

1-1. Make your ports and/or interfaces external to the block design.

1-1-1. Locate the IP block containing the pin or interface you want to make external.

Right-click the pin or interface, making sure that just the pin of interest is highlighted.

1-1-2. Select **Make External**.

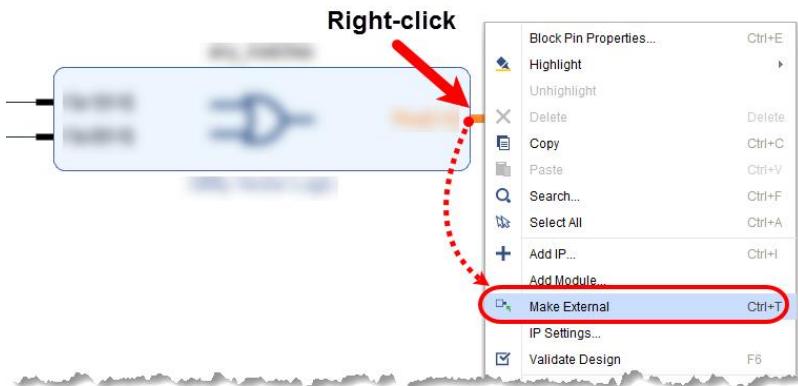


Figure 97: Making a Port or Interface External

A port is created and a net is generated to connect it to the pin or interface that you specified.

Running Connection Automation

1-1. Use the Connection Automation to make connections to the the desired piece of IP block.

- 1-1-1. Click **Run Connection Automation** in the Design tab information bar.

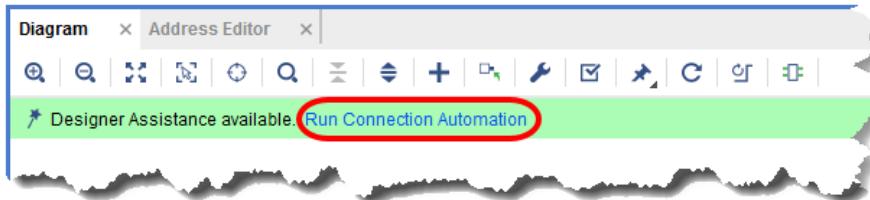


Figure 98: Running Connection Automation

This opens a Run Connection Automation dialog box listing all the IP currently in the design that can have Designer Assistance run on it. IPs are listed in a hierarchy on the left where each child node of an IP node represents an interface that is eligible for automation. Any options associated with a particular automation will be shown on the right whenever an interface node is selected on the left.

- 1-1-2. Click the **Expand All** icon (\bowtie) above the list of IPs in the left-hand panel so that you have a full view of all the IP available for connection automation.
- 1-1-3. Check **your interface ports**, on which Connection Automation will be run.

Notice how options appear on the right after the interface node is selected. In some cases, you want to leave all automation options at their defaults but, in other cases, it can be useful to customize the options here. For example, several automations allow you to select which clock or AXI master will be automatically connected to the IP.

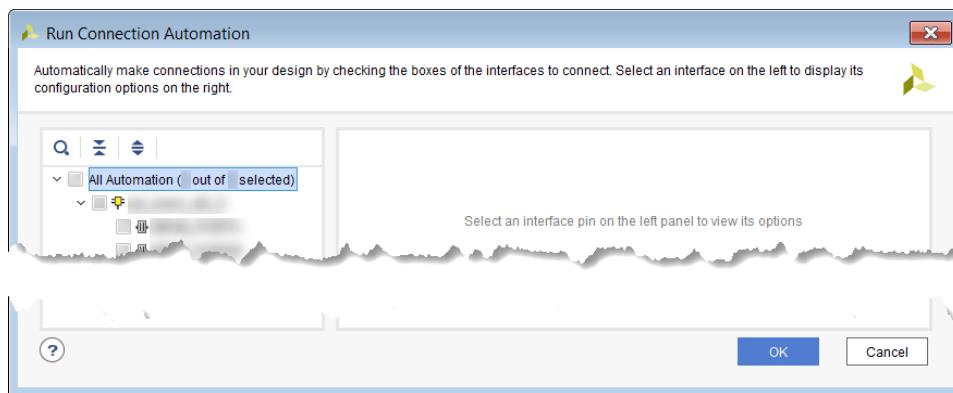


Figure 99: Generic Run Connection Automation Wizard

- 1-1-4. Click **OK** to run the selected automation.

Connection Automation is fully capable of performing multiple connections—here a single connection was illustrated to show the basics of the capability.

Running Connection Automation for Multiple Modules

The Connection Automation Wizard automates many of the commonly used basic connections between IP catalog components in the block diagram editor. Connections include AXI, clocks, reset, and external ports. Specific connections, such as interrupts, IP specific, and custom user connections are not processed by the wizard and must be completed manually. Many IP blocks are supported by the Connection Automation Wizard.

1-1. Use Designer Assistance to automate multiple connections.

- 1-1-1. Click **Run Connection Automation** in the Design tab information bar.

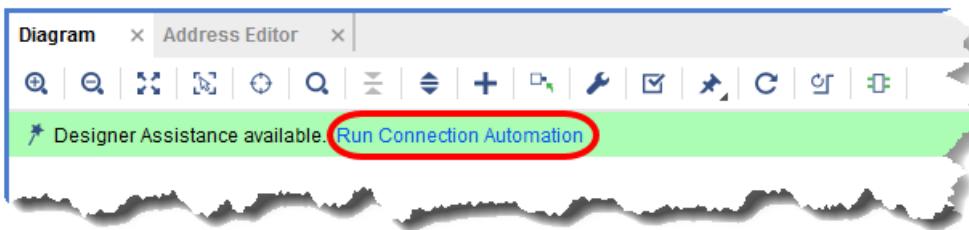


Figure 100: Running Connection Automation

This opens a Run Connection Automation dialog box listing all the IP currently in the design that Designer Assistance can run automations on. IPs are listed in a hierarchy on the left where each child node of an IP node represents an interface that is eligible for automation. Any options associated with a particular automation will be shown on the right whenever an interface node is selected on the left.

- 1-1-2. Click the **Expand All** icon (\oplus) to ensure that the list of available automations is fully visible.
- 1-1-3. Check the node that corresponds to automating connections for the following IPs and/or interface: **IP core**.

If asked to check *All*, select the top-level **All Automation** node. Similarly, selecting an IP node will automate connections to all interfaces available under that IP.

Notice how options appear on the right after the interface node is selected. In some cases, you want to leave all automation options at their defaults but, in other cases, it can be useful to customize the options here. For example, several automations allow you to select which clock or AXI master will be automatically connected to the IP.

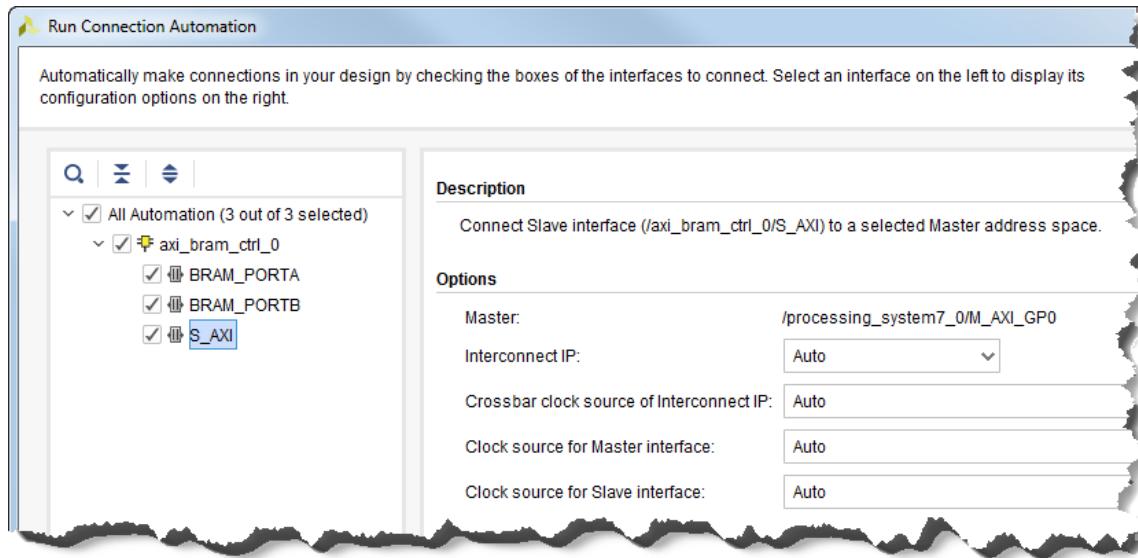


Figure 101: Run Connection Automation Connecting the PS to MicroBlaze Processor (Zynq SoC)

- 1-1-4. Click **OK** to run the selected automation.

Note that use of Designer Automation is not a required part of the design flow. Any automation done via Designer Assistance can also always be done manually.

Running Block Automation

1-1. Use Block Automation to automate the configuration of recently instantiated IP.

- 1-1-1. Click **Run Block Automation** from the Designer Assistance information bar.

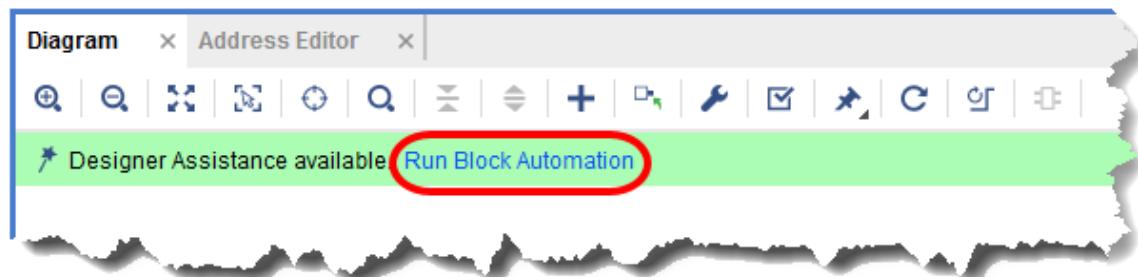


Figure 102: Designer Assistance - Block Automation (Run Connection Automation May be Absent)

This opens a Run Block Automation dialog box listing all the IP currently in the design eligible for block automation. IPs are listed in a hierarchy on the left and any options associated with a particular automation will be shown in the right pane whenever an IP instance is selected in the left pane.

- 1-1-2. Click the **Expand All** icon () to ensure that the list of available automations is fully visible (1).

- 1-1-3. Select either the top-level **All Automation** check box or individual blocks as listed below.

Unless otherwise indicated within the block list, maintain default automation options for all blocks.

- o Blocks: the desired piece of IP

- 1-1-4. Deselect **Apply Board Preset** when using Zynq devices, as this will undo any work that you have done in the processor's Re-customization Wizard (2).

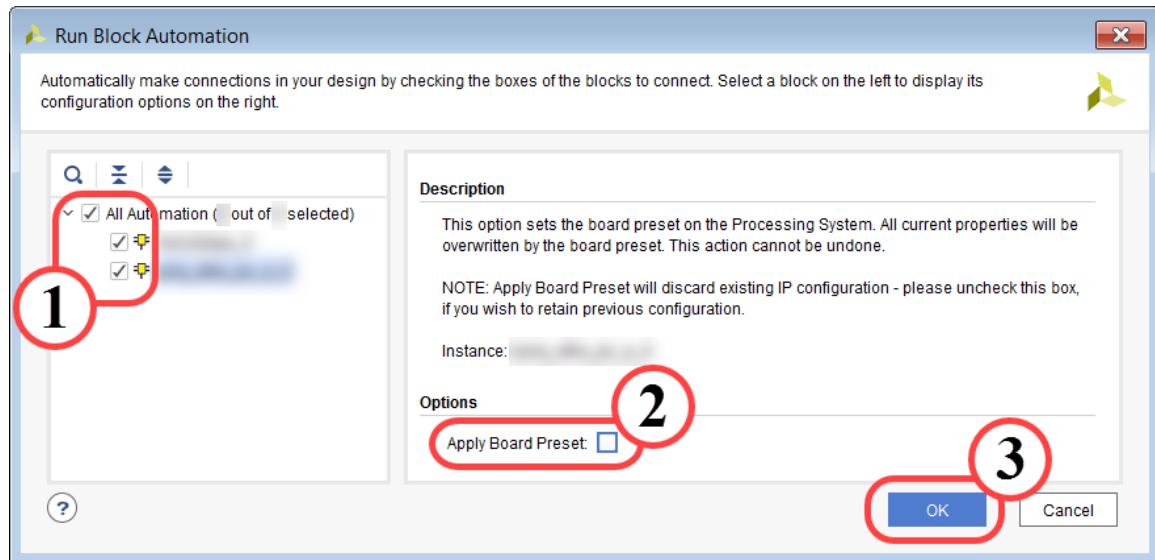


Figure 103: Run Block Automation Dialog Box

- 1-1-5. Click **OK** to run the selected automation (3).

Locating Objects in the Board Design

Some designs can become quite large and complex. When "Regenerate Layout" is run, how can you find various IP blocks, nets, or ports/interfaces? This instruction reviews the use of the design hierarchy in locating objects.

1-1. Locate the object.

1-1-1. Locate the Design Hierarchy window.

This is where all the elements in a design are located. Elements are organized into external interfaces, interface connections, ports, nets, hierarchical blocks, and IP.

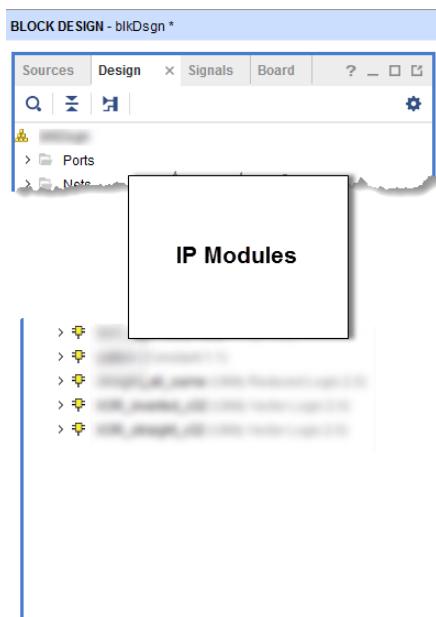


Figure 104: Locating the Design Hierarchy View

1-1-2. Expand the branch corresponding to the type of element you are looking for (hierarchy block, port, etc.)

If you are looking for a specific port or interface on an IP block, you can expand that IP block. Hierarchy blocks contain their own hierarchy of elements.

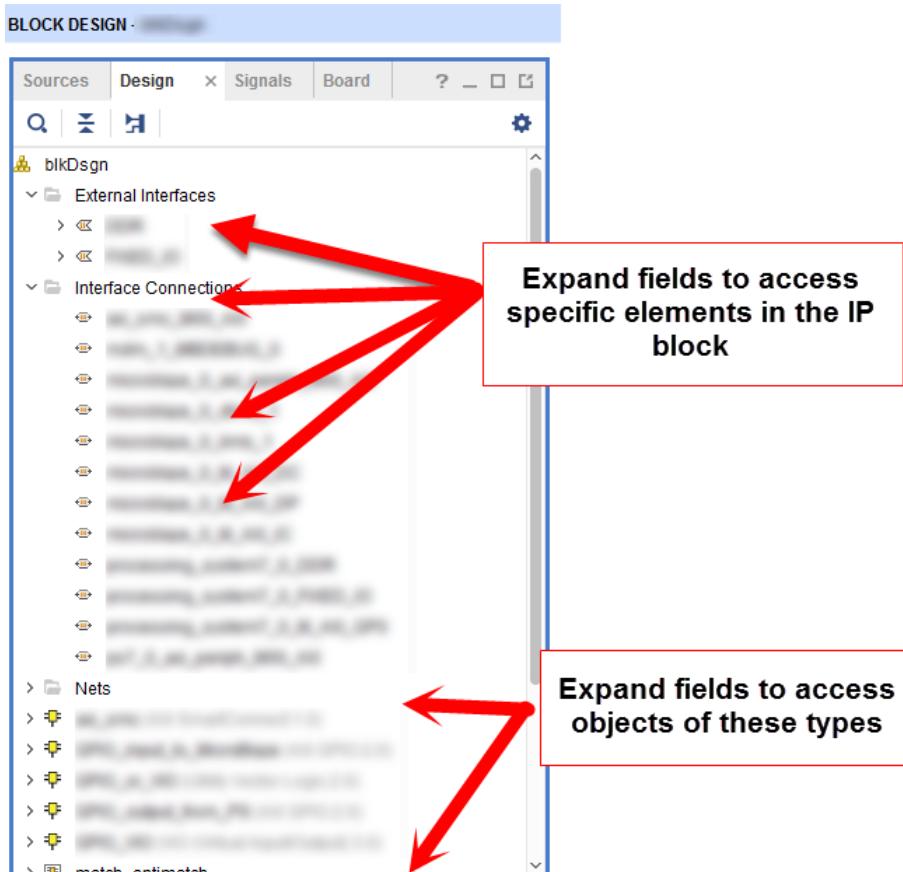


Figure 105: Expanding Branches in the Design Hierarchy Tree

- 1-1-3. Click the specific element or elements that you want to locate.

The Schematic view will highlight (select) that item.

Mapping Peripheral Addresses

Ensuring that each peripheral has its own memory location is a key aspect of getting the hardware and software to work together.

1-1. Ensure that all peripherals have an address assigned to them.

1-1-1. Select the **Address Editor** tab.

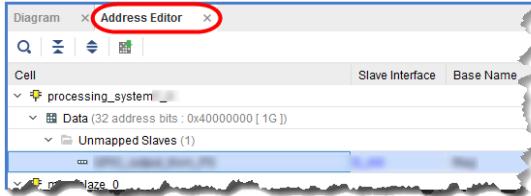


Figure 106: Locating the Address Editor Tab

1-1-2. Click the **Expand** icon () to expand all the entries in the window.

1-1-3. Search for any field containing the text "Unmapped Slaves".

This field will appear *only once for each AXI tree*.

If there are unmapped devices, continue with the next instruction; otherwise skip the next instruction.

1-2. Assign addresses for any peripheral that do not yet have an address assigned to them.

1-2-1. Right-click the peripheral that you want to assign an address to.

1-2-2. Select **Assign Address** to assign addresses peripheral by peripheral.

-- OR --

Select **Auto Assign Addresses** to assign addresses to all peripherals that do not have an address assigned to them.

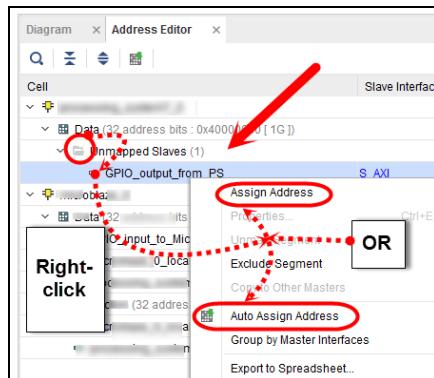


Figure 107: Opening the Dialog Box for Assigning an Address

Generating Output Products

Once a block design has been created, its output products can be generated. The synthesis task automatically runs this phase; however, many other operations require this step to be taken, such as exporting a block design for SDK.

1-1. Generate the output products for the items you want.

1-1-1. Select **the items you want** to generate the output products.

It may be necessary to expand the hierarchy in the Sources window to locate the object(s) of interest.

1-1-2. Right-click the items and select **Generate Output Products** from the context menu.

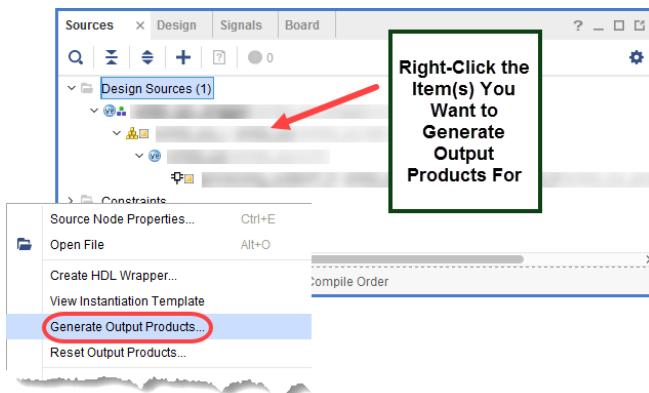


Figure 108: Selecting the Items to Generate Output Products For

The Generate Output Products dialog box opens, listing which output products will be generated.

How the design will be synthesized is up to the user. Global means that all of the IP in the design along with the user code will be synthesized. Out of context per IP will synthesize and generate a netlist for each piece of IP. Out of context per Block Design will synthesize only the contents of the selected block design.

- 1-1-3. Select the **Global** synthesis option so that the complete design—the block design, its IP, and the user code (including the wrapper)—will be generated.

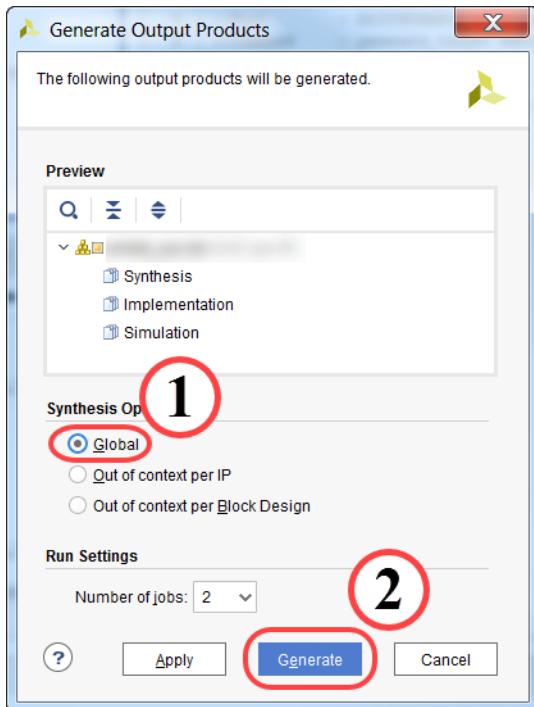


Figure 109: Managing the Output Products

The Run Settings specify the number of processors on your development machine that you would like to run the task on. Generally, this is set to the maximum number of processors.

- 1-1-4. Click **Generate** to start the generation process.

When generation completes, a success dialog box opens.

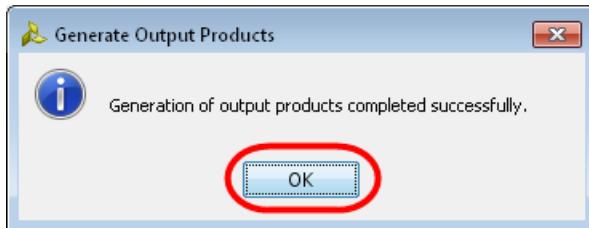


Figure 110: Successful Output Products Generation

- 1-1-5. Click **OK** to close the Generate Output Products dialog box.

Creating a Hierarchical Block in a Block Design

Hierarchy blocks are useful for grouping similar logic together to simplify cluttered block designs.

1-1. Create and name a hierarchy block.

There are two processes involved in creating a hierarchy block. One is the creation of a hierarchy block; the other is the addition of IP to the block.

These two tasks can be performed in either order:

- **Select the IP to be included in the hierarchy block and create a hierarchy block around that IP**
- **Create an empty hierarchy block and add IP to it.**

These processes are not mutually exclusive because once a hierarchy block is created, additional IP can be added to it regardless of how the hierarchy block was initially created.

The following describes how to create an empty hierarchy block.

- 1-1-1. Right-click in the block diagram.
- 1-1-2. Select **Create Hierarchy** from the context menu.

The Create Hierarchy dialog box allows you to specify the name for the hierarchy.

- 1-1-3. Enter **the name of your hierarchy** in the Cell name field.

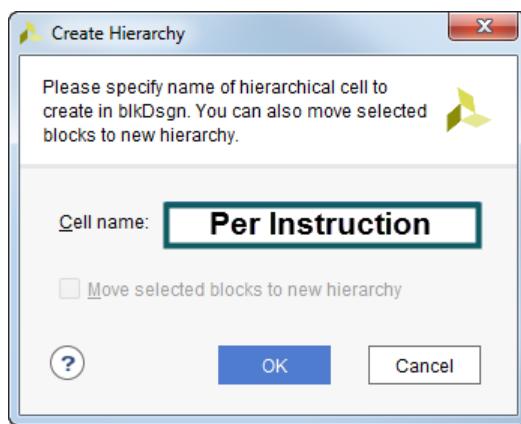


Figure 111: Create Hierarchy Dialog Box

- 1-1-4. Click **OK** to create an empty hierarchy.

Note that if you were to select the IP from the canvas and/or the Design view, then follow the above tasks—you will create the hierarchy block with the IP already included in it.

Adding IP to a Hierarchy Block

IP can be added to a hierarchy block before or after the hierarchy block is created.

1-1. Add your **IP modules** IP blocks to the hierachal block.

1-1-1. Select **your IP modules** IP blocks.

This can be done graphically by using <**Ctrl** + **click**> to select each piece of IP from the canvas as well as selecting the IP from the Design tab.

Note that when you select the IP blocks in the block diagram, they will also be highlighted in the Design window and vice versa.

1-1-2. Drag-and-drop the IP into the name of your hierarchy.

When moving the IP, you will notice the "plus-sign-in-a-diamond" appear in the IP's outline. This indicates that it will be added to the hierarchy block that it is being dragged over.

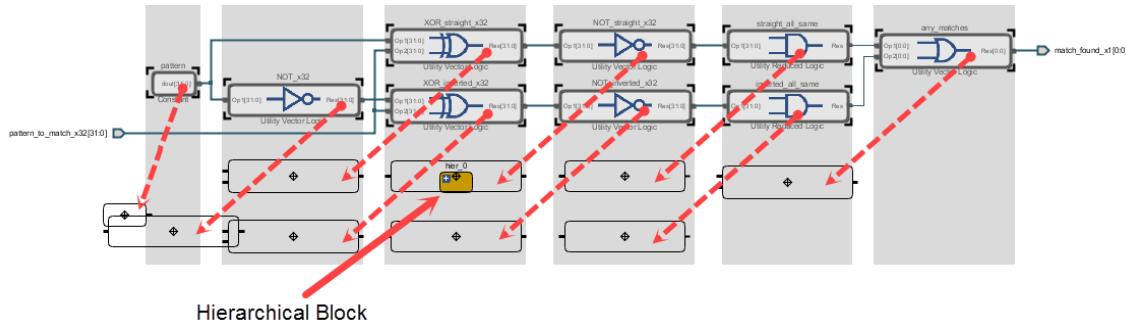


Figure 112: Example of Dragging IP into a Hierarchical Block

Alternatively, if the hierarchical block has not yet been created, you can select the IP from the canvas as well as the Design tab, then right-click the canvas to access the context menu and select **Create Hierarchy**.

Expanding the Contents of a Hierarchical Block

Sometimes it is easier to understand the diagram when the contents are displayed in another tab, but sometimes you want to see the context of how the contents of a hierarchical block interact with the current level of hierarchy.

1-1. Expand the name of your hierarchy.

1-1-1. Locate **the name of your hierarchy**.

This can be done by visually searching through the canvas, or using the alphabetical list of items in the Design tab.

1-1-2. Click the '+' sign in the upper-left corner of the hierarchy block.



Figure 113: Expanding a Hierarchy Block

Note: The hierarchy block can be collapsed by clicking the '-' button in the upper-left corner.

Opening a Hierarchical Block in a New Tab

Once a hierarchy exists, there are times when you might want to see what is happening inside the block. Here's how to access the contents of a hierarchy block.

1-1. Open the name of your hierarchy in a new tab.

As with many other features of the tool, there are several ways to gain entry into the hierarchy block. First, you must identify which block you want to enter.

1-1-1. Identify the block that you want to enter.

This can be done in one of two ways:

- Right-click the hierarchy block and select **Open in new tab** or press <F4>.

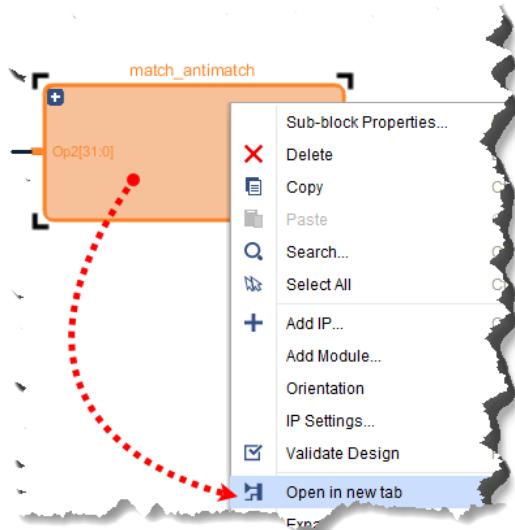


Figure 114: Opening the Hierarchy Block in a New Tab

- Alternatively, you can use the hierarchy browser, which is the name of the block diagram found immediately below the Diagram tab.

1-1-2. From the top left of the Diagram tab, use the Hierarchy Path tool to browse the hierarchies in the block design.

- 1-1-3. Select **the name of your hierarchy** from the listing of hierarchies to open the hierarchy in a new tab.

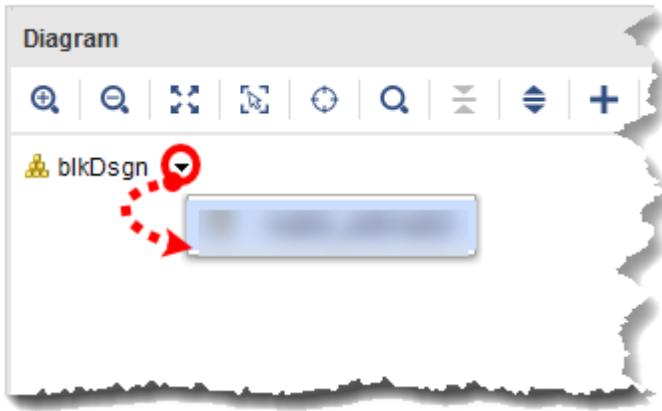


Figure 115: Access the Hierarchical Blocks in a Block Diagram

Note: If this is not available as shown in the figure, click the **Settings** symbol in the extreme top-right corner. Go to the General section and select the **Show Hierarchy** option.

the name of your hierarchy opens in a new tab with the name Diagram - the name of your hierarchy.

Marking Signals for Debugging

The Vivado IP integrator enables you to *mark* signals for debugging. These signals are collected and automatically tied into a special type of debugging core called the Integrated Logic Analyzer (ILA). This type of core, unlike the VIO, contains both simple and complex triggering mechanisms and memory for storing full-speed samples.

1-1. Mark the net or nets that you want to mark for debugging for debug.

1-1-1. Select the net or nets that you want to mark for debugging.

You can select them from the canvas or Design list window.

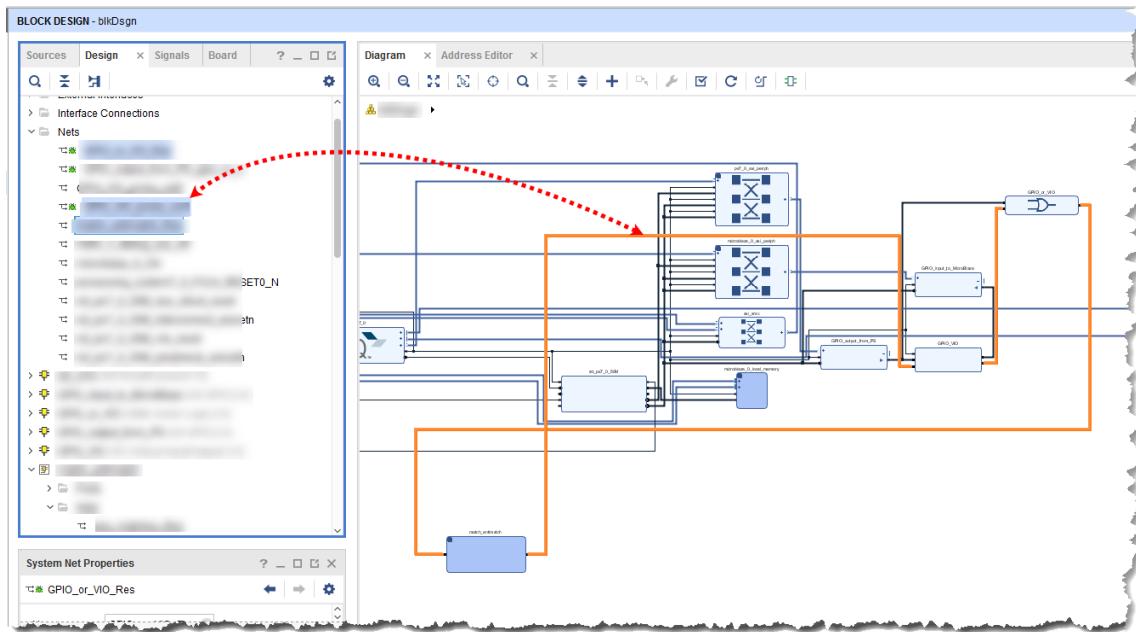


Figure 116: Selecting Nets

1-1-2. Right-click to open the context menu.

1-1-3. Select **Debug**.

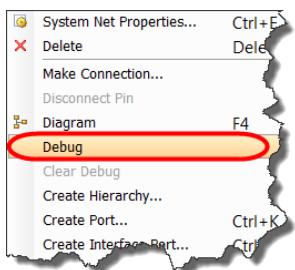


Figure 117: Marking the Nets for Debugging

Updating IP

The Vivado Design Suite continuously checks for possible problems during all phases of operation.

When older designs are brought into a newer version of the Vivado Design Suite, there is often a mismatch between the IP versions in the design and those that are available in the newer version of the tools. These discrepancies are reported in both a pop-up dialog box as well as the Report IP Status command.

While there are a few reasons for keeping the original version of the IP, it is usually preferable to update the IP by using the Report IP Status command.

1-1. Run the IP Status report to identify which IPs are locked (out of date) in the block design.

1-1-1. If it is not already available, open the block diagram.

When the Diagram tab is selected, the information bar indicates if any IP blocks are out of date and need to be upgraded. From here you can generate the IP Status report.

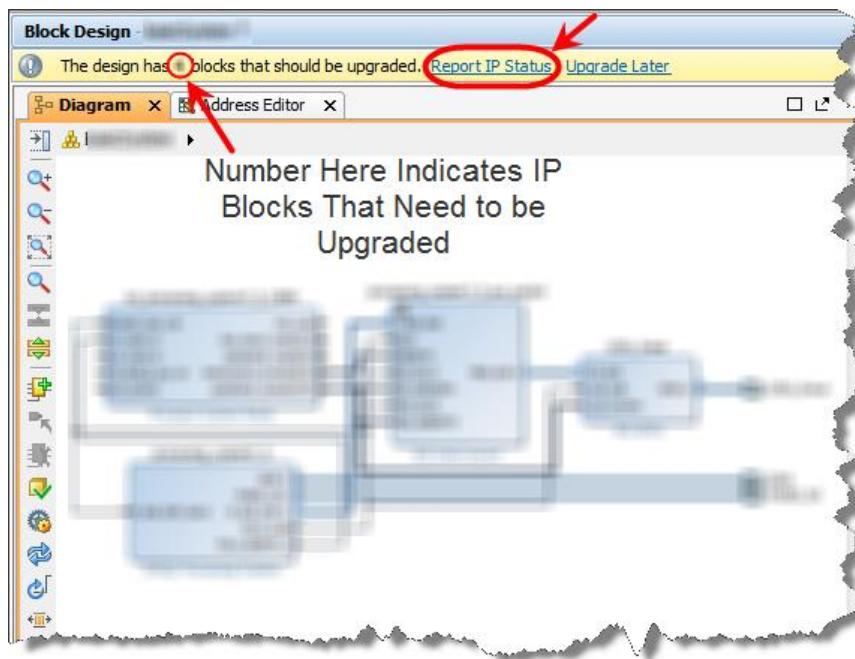


Figure 118: Report IP Status Command in the Block Design

1-1-2. Click **Report IP Status** in the information bar to generate the IP Status report.

Alternatively, you can run select **Tools > Report > Report IP Status** to generate the IP Status report.

1-1-3. Review the IP Status report.

The IP Status report gives you the status of all IP in the design, current and recommended versions of IP if the IP needs to be upgraded, the change log of IP revisions, etc.

The IP Status column provides you the reason for IP being locked.

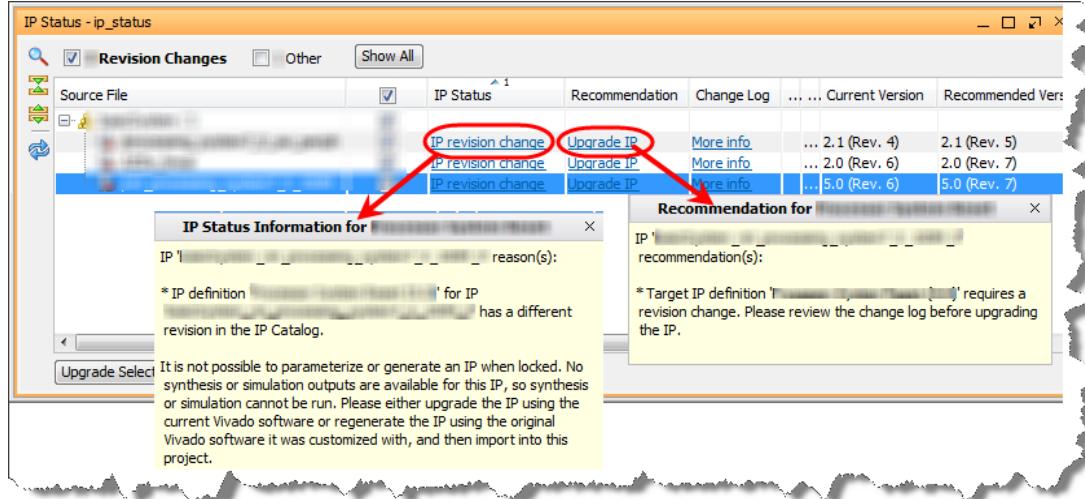


Figure 119: IP Status and Recommendation Columns IP Status Report

1-2. Upgrade your IP modules to the current version.

- 1-2-1. Place a check in the checkbox of the IP you want to upgrade to indicate that you will take further action on these cores (1).
- 1-2-2. Click **Upgrade Selected** in the IP Status window to begin the IP upgrade process (2).

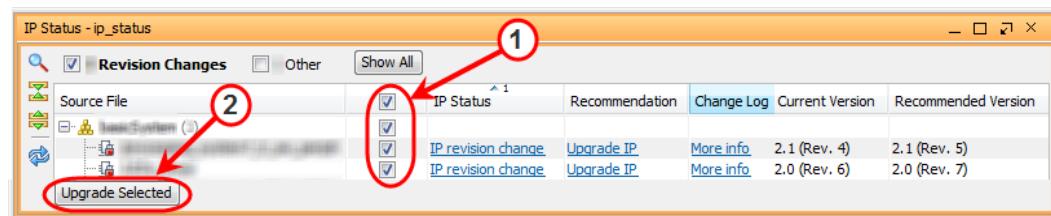


Figure 120: IP Status Report

- 1-2-3. Click **OK** in the Upgrade IP dialog box to acknowledge the upgrading of the selected IP to the current version.

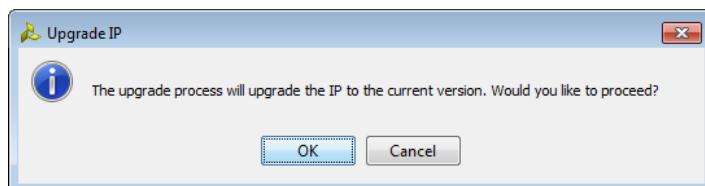


Figure 121: Upgrade IP Dialog Box

- 1-2-4. Rerun the IP Status report to confirm that all the IP have been upgraded.

Regenerating the Layout

The canvas can become disorganized after manual edits. The IP integrator tool can automatically reconfigure the layout to be aesthetically pleasing.

1-1. Regenerate the layout.

- 1-1-1. Right-click anywhere on the canvas and select **Regenerate Layout**.

Alternatively, you can click the **Regenerate Layout** icon.

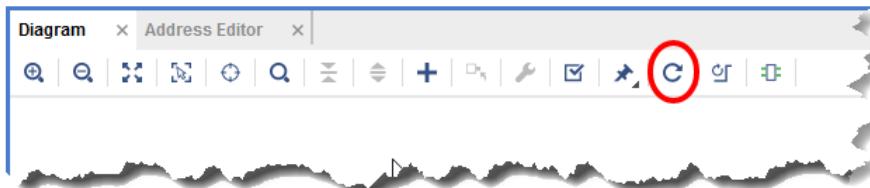


Figure 122: Regenerating the Layout

Running a Design Rule Check/Design Validation

Partial design rule checks are automatically run every time an IP is connected. Design validation is a more comprehensive (but not exhaustive) block design-wide verification and must be manually run.

1-1. Run a manual design validation.

- 1-1-1. Select **Tools > Validate Design**.

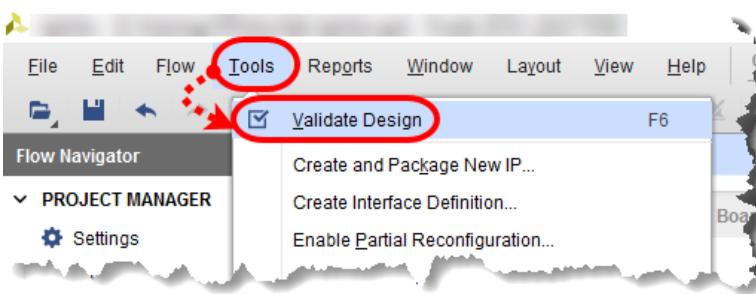


Figure 123: Validating the Design

When design validation is complete, results are reported via the Messages tab at the bottom of the Vivado IDE.

Because quite a number of messages may be displayed, you have the option of filtering what to view.

If no issues are found, then a dialog box will appear that reports that validation succeeded.

- 1-1-2. Click **OK** to close the dialog box.

If issues were located:

- 1-1-3. Select the **Messages** tab to view the items discovered by the validation process.

- 1-1-4. Filter the types of messages by placing a check mark next to the types of messages you want to see: errors, infos, or status.

The display is automatically updated.

Hyperlinks are provided to help you quickly locate the issue.

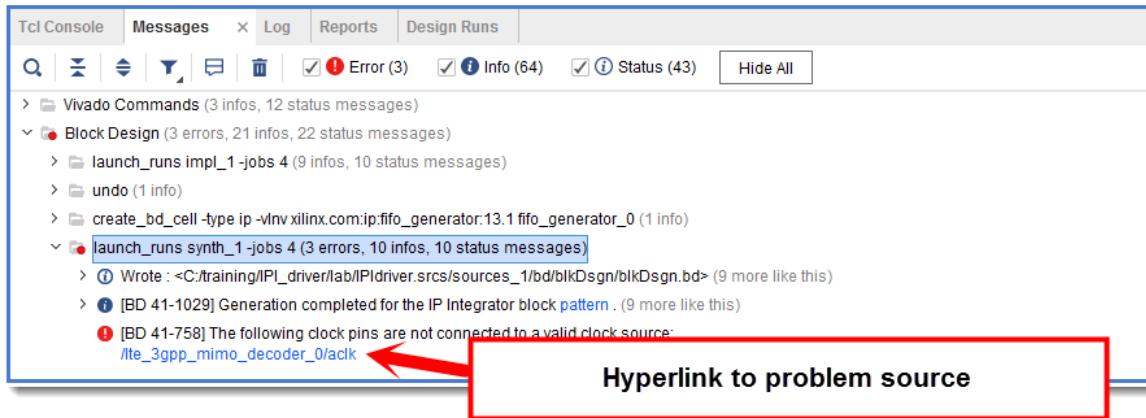


Figure 124: Viewing the Results of the Design Validation

- 1-1-5. Correct any issues and rerun validation until no more issues are found.

Validating the Block Design

- 1-1. Validate the design to catch any connection and address map errors.

- 1-1-1. Select **Tools > Validate Design**, or you may press **F6**.

Any issues are displayed in the Console window.

- 1-1-2. Click **OK** to close the window.

Saving the Block Design

1-1. Save the block design.

- 1-1-1. Select **File > Save Block Design** to save the block design.

Alternatively, you can press <**Ctrl + S**>.

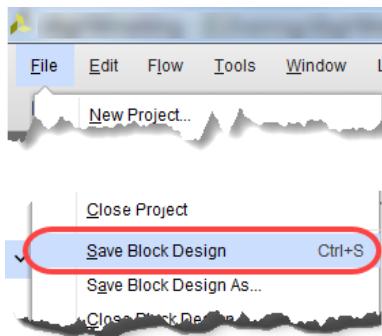


Figure 125: Saving the Block Design

Generating a Wrapper for a Block Design

While a block design can be directly converted into a netlist, block designs are most frequently used as structural modules within a larger design.

Many embedded designs often begin with the embedded block design then have either *related* logic (operates in concert with the embedded portion of the design) or *unrelated* logic (does not tie to any aspect of the embedded portion of the design) added.

Typically, once the embedded design has been created, an HDL wrapper is created that instantiates the embedded design and provides a platform for adding other logic.

1-1. Create a wrapper for the block design.

- 1-1-1. Ensure that the Sources tab is selected; if it is not, select it to open the Sources view (1).

While the block diagram can be found under any of the next level tabs (Hierarchy, IP Sources, Libraries, or Compile Order), it is shown here from the Hierarchy tab as this will be typical of the way that many designers use the tool.

- 1-1-2. Right-click the block design name which is under the Design Sources node (2).

Notice the icon (agini) which indicates that this entry is a block design.

This will open a pop-up window to actions that can be taken for this entry.

1-1-3. Select **Create HDL Wrapper** to begin the wrapper creation process (3).

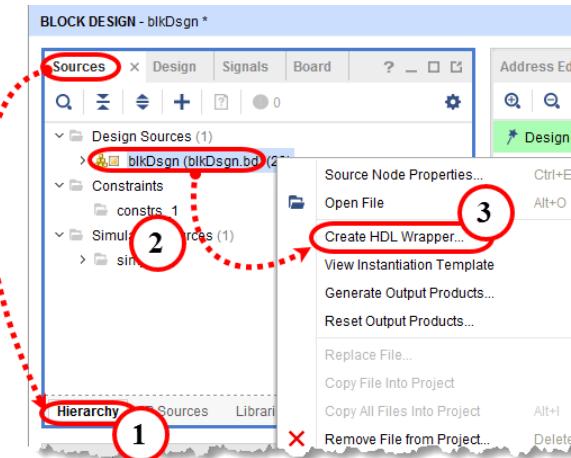


Figure 126: Selecting Create HDL Wrapper

The Create HDL Wrapper dialog box opens, showing a list of options relating to how the wrapper should be handled once generated.

Two choices are presented:

- Generate a wrapper that you can manually edit, which is useful when you have additional logic to add to the top level wrapper, or
- Generate a wrapper that is managed by the Vivado Design Suite, which is ideal for designs that are completely created with a block design.

1-1-4. Select the option which best matches your needs (typically this is the default option of letting the Vivado Design Suite manage the wrapper).

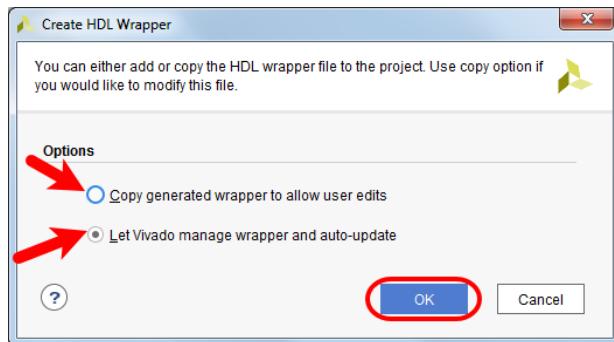


Figure 127: Generating the Wrapper and Copying the Generated File into the Project

This will produce a wrapper file in the selected template language. You can double-click it to open it in a new editor window. The project hierarchy will automatically be adjusted to reflect the addition of this new file.

1-1-5. Click **OK** to create the wrapper.

Vivado Elaborated Design Operations

In This Section

Opening the Elaborated Design	95
Creating a Schematic on an Elaborated Design.....	95
Running UltraFast DRC Checks on the Elaborated Design.....	95
Running DRC Checks on the Elaborated Design.....	96
Closing the Elaborated Design.....	96

Opening the Elaborated Design

1-1. Open the elaborated design.

- 1-1-1. Click **Open Elaborated Design** under RTL Analysis in the Flow Navigator to elaborate the design.

The elaborated RTL design enables various analysis views, including RTL Netlist, Schematic, and Graphical Hierarchy. These views have a cross-select feature, which allows you to debug and optimize the RTL.

The Elaborate Design dialog box opens.

- 1-1-2. Click **OK** in the dialog box to open the elaborated design.

Creating a Schematic on an Elaborated Design

1-1. Create a Schematic on an Elaborated Design.

- 1-1-1. In the Flow Navigator, under RTL Analysis > Elaborated Design, click **Schematic**.

The RTL Schematic opens in the main workspace area.

Running UltraFast DRC Checks on the Elaborated Design

1-1. Run UltraFast™ design methodology DRC checks on the elaborated design.

- 1-1-1. Click **Report Methodology** under RTL Analysis > Open Elaborated Design in the Flow Navigator.

The Report Methodology dialog box opens.

- 1-1-2. Click **OK** to run the design methodology checks and find errors or problems in the current design.

Running DRC Checks on the Elaborated Design

1-1. Run DRC checks on the elaborated design.

- 1-1-1. Click **Report DRC** under RTL Analysis > Open Elaborated Design in the Flow Navigator.

The Report DRC dialog box opens.

- 1-1-2. Select **default** in the Vivado Rule Decks section.

- 1-1-3. Observe the categories in the **Rules** section.

- 1-1-4. Click **OK** to generate the DRC report.

Closing the Elaborated Design

1-1. Close the elaborated design.

- 1-1-1. Select **File > Close Elaborated Design** to close the elaborated design.

The Confirm Close dialog box opens.

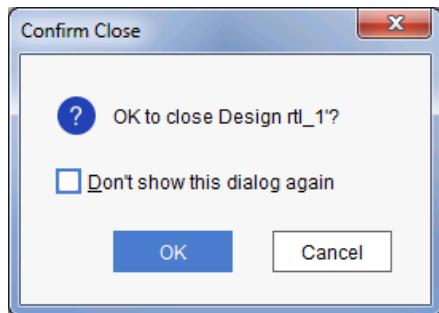


Figure 128: Confirm Close Dialog Box

- 1-1-2. Click **OK**.

Vivado Simulation Operations

In This Section

Running Behavioral Simulation.....	97
Running Timing Simulation.....	97
Accessing Simulation Settings	97
Closing Vivado Simulation.....	98

Running Behavioral Simulation

1-1. Run behavioral simulation using the Vivado simulator.

- 1-1-1. Select **Simulation > Run Simulation > Run Behavioral Simulation** from the Flow Navigator under Simulation.

The simulation window opens and simulation runs for the length of time specified in the Simulation Settings (1 us default).

You can alternatively start the the Vivado simulator by entering `launch_simulation` in the Tcl command line.

Running Timing Simulation

1-1. Run behavioral simulation using the Vivado simulator.

- 1-1-1. Select **Run Simulation > Run Post-Implementation Timing Simulation** from the Flow Navigator, under Simulation.

The simulation window opens and the simulation runs for the length of time specified in the Simulation Settings (1 us default).

Accessing Simulation Settings

1-1. Open the simulation settings.

1-1-1. Select **Settings** in the Flow Navigator, under Project Manager and go to **Simulation**.

The simulation settings opens and contains five sub-tabs: Compilation, Elaboration, Simulation, Netlist, and Advanced.

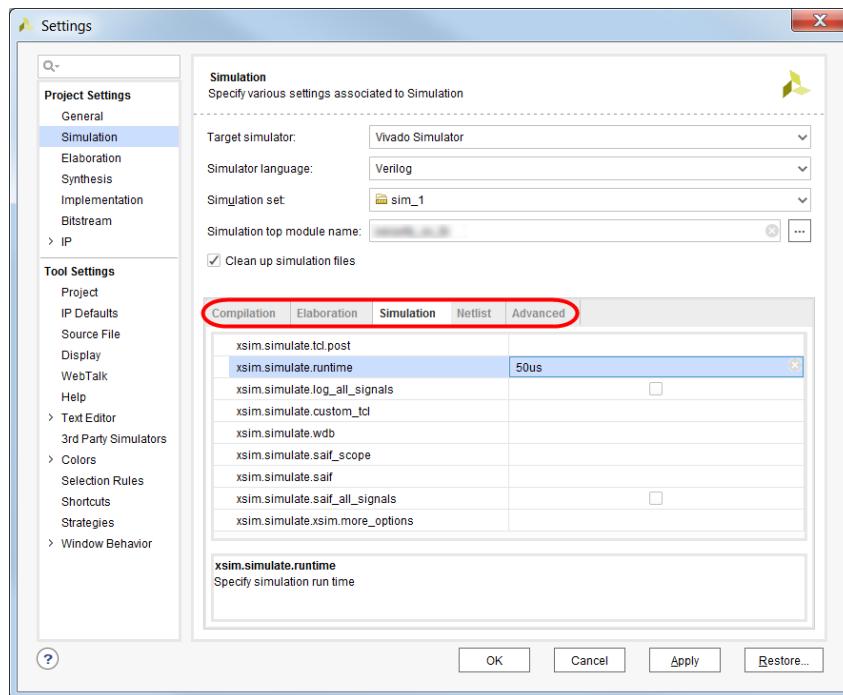


Figure 129: Vivado Simulator Settings

Closing Vivado Simulation

1-1. Close the simulation.

1-1-1. Select **Simulation** in the Flow Navigator, then right-click and select **Close Simulation**.

1-1-2. Click **OK** to close the simulation.

Vivado Synthesis Operations

In This Section

Setting Synthesis Options.....	99
Running Synthesis.....	100
Opening the Synthesized Design.....	102
Reloading the Synthesized Design.....	102
Generating a Utilization Report.....	103
Generating a Clocks Networks Report.....	103
Running a DRC Report on the Synthesized Design	104
Opening the Timing Constraints Window after Synthesis	104
Generating a Timing Summary Report on the Synthesized Design	104
Running Simultaneous Switching Noise (SSN) Analysis.....	106
Generating a Clock Interaction Report on the Synthesized Design	106
Closing the Synthesized Design	106

Setting Synthesis Options

1-1. Set the synthesis options.

1-1-1. Click **Synthesis Settings** under Synthesis in the Flow Navigator.

Alternatively, you can select **Flow > Synthesis Settings**.

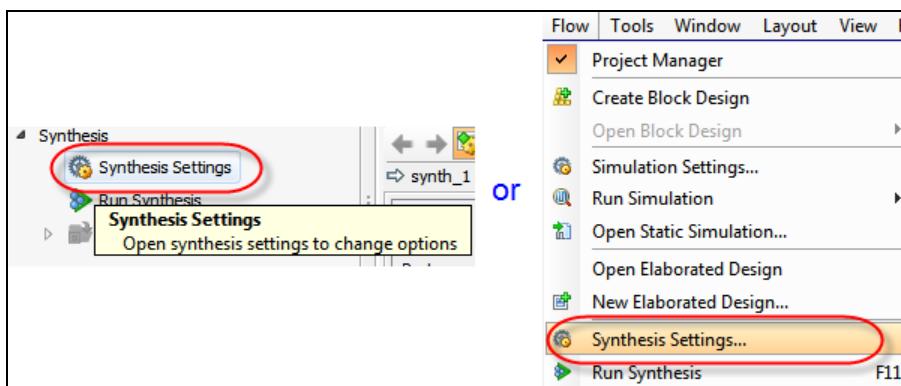


Figure 130: Selecting Synthesis Settings

The Synthesis Project Settings dialog box opens.

Note that the strategy is set to **Vivado Synthesis Defaults**.

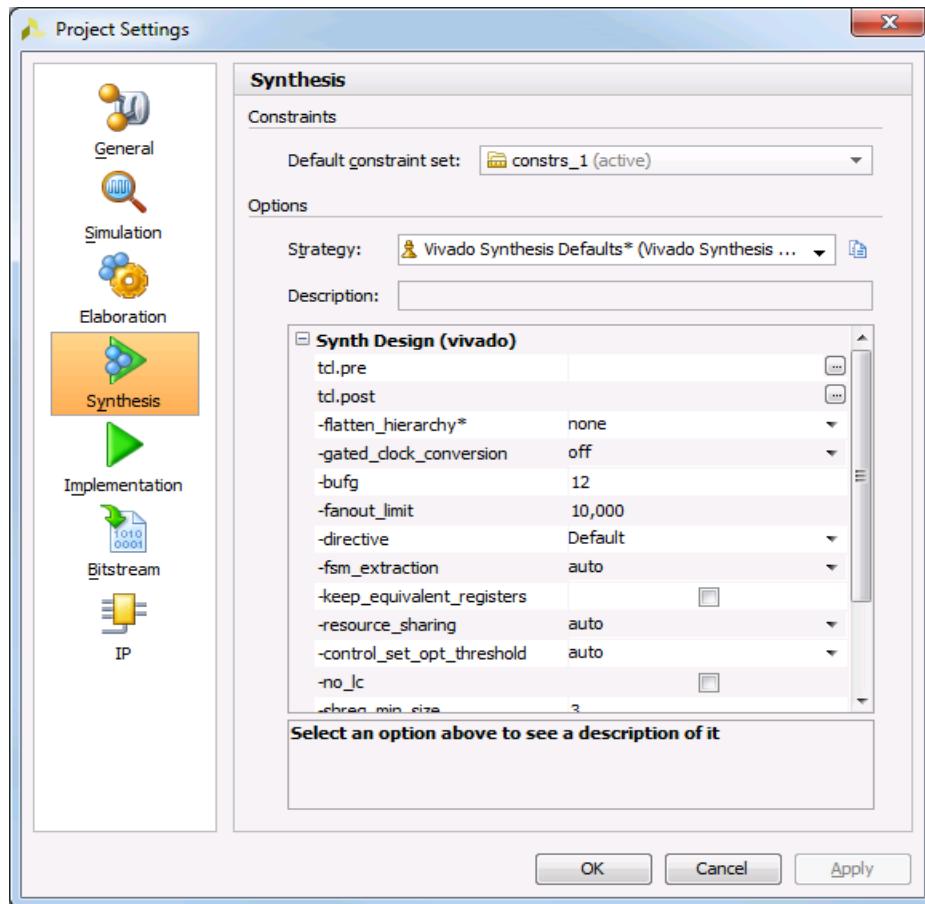


Figure 131: Synthesis Project Setting Dialog Box

- 1-1-2.** Click the field next to the "**-flatten_hierarchy**" option and select **rebuilt**, if it is not already selected.

This option allows the design hierarchy to be more useful for design analysis because many logical references will be maintained.

- 1-1-3.** Click **OK**.

Running Synthesis

1-1. Run synthesis.

1-1-1. Click **Run Synthesis** in the Flow Navigator under Synthesis.

1-1-2. Click **OK** to launch the runs.

Alternatively, you can also select **Flow > Run Synthesis** or press **<F11>**.

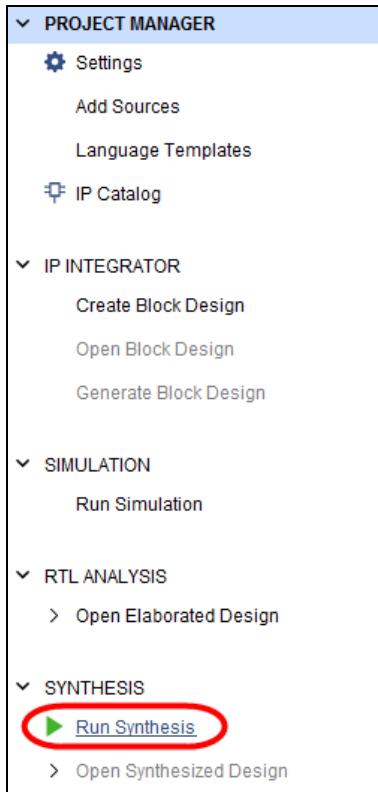


Figure 132: Selecting Run Synthesis

1-1-3. Click **Save** if you are asked to save your files.

After the synthesis process completes, the Synthesis Completed dialog box opens. The dialog box prompts you to run implementation, open the synthesized design, or view reports.

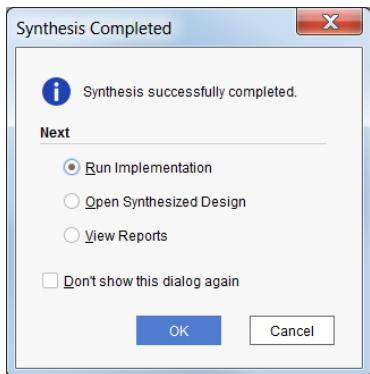


Figure 133: Synthesis Completed Dialog Box

- 1-1-4. Click **OK** to continue with your preferred choice or **Cancel** to simply close the dialog box and return to the normal view of the Vivado Design Suite.

Opening the Synthesized Design

1-1. Open the synthesized design.

- 1-1-1. Click **Open Synthesized Design** under Synthesis in the Flow Navigator.

Alternatively, you can select **Flow > Open Synthesized Design**.

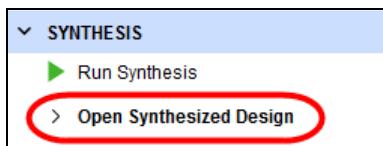


Figure 134: Opening the Synthesized Design

Reloading the Synthesized Design

When you make any changes to the XDC file, such as commenting, uncommenting, and adding constraints, then synthesis goes into an out-of-date state.

1-1. Reload the synthesized design.

- 1-1-1. Click the **more info** link at in the top-right corner status bar and then click the **Force up-to-date** link.

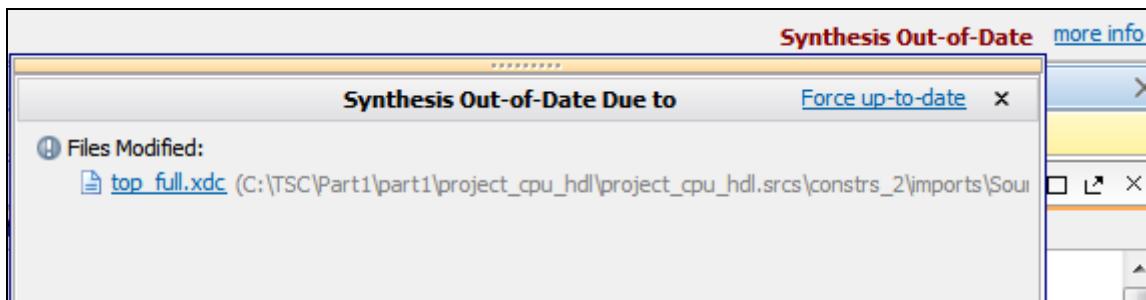


Figure 135: Force-up-to-date Link

- 1-1-2. Click the **Reload** link in the status bar to reload the synthesized design.

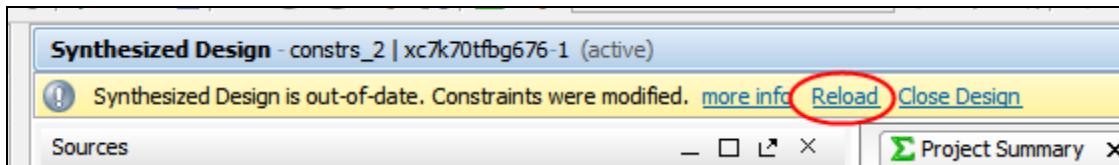


Figure 136: Reload Link

The synthesized design will open with any changes applied to it.

Generating a Utilization Report

1-1. Generate a Utilization report.

- 1-1-1. Click **Report Utilization** under Synthesized Design in the Flow Navigator.

Alternatively, you can select **Reports > Report Utilization**.

Generating a Clocks Networks Report

1-1. Generate a Clocks Networks report.

- 1-1-1. Click **Report Clock Networks** in the Flow Navigator under Synthesized Design.

Alternatively, you can select **Reports > Timing > Report Clock Networks**.

The Report Clock Networks dialog box opens.

- 1-1-2. Click **OK**.

Running a DRC Report on the Synthesized Design

1-1. Run a DRC report on the synthesized design to check for any violations.

- 1-1-1. Click **Report DRC** under Synthesis > Synthesized Design in the Flow Navigator.

The Report DRC dialog box opens.

- 1-1-2. Select **default** in the Rule Decks section of the Report DRC dialog box.

- 1-1-3. Click **OK** to start the default DRC checks on the synthesized design.

Opening the Timing Constraints Window after Synthesis

1-1. Open the Timing Constraints window.

- 1-1-1. Select **Window > Timing Constraints**.

This window can also be opened by clicking **Edit Timing Constraints** under Synthesized Design in the Flow Navigator.

The Timing Constraints window opens in the main workspace area.

Generating a Timing Summary Report on the Synthesized Design

The timing summary command is accessible in the Vivado Design Suite GUI once the synthesized design is opened in the Vivado IDE. If the synthesized design is not opened in the GUI and you do not remember how to open the design, refer to the "Opening the Synthesized Design" section in the *Lab Reference Guide*.

1-1. Generate a Timing Summary report.

- 1-1-1. Click **Report Timing Summary** under **Synthesis > Open Synthesized Design** in the Flow Navigator.

The Report Timing Summary dialog box opens.

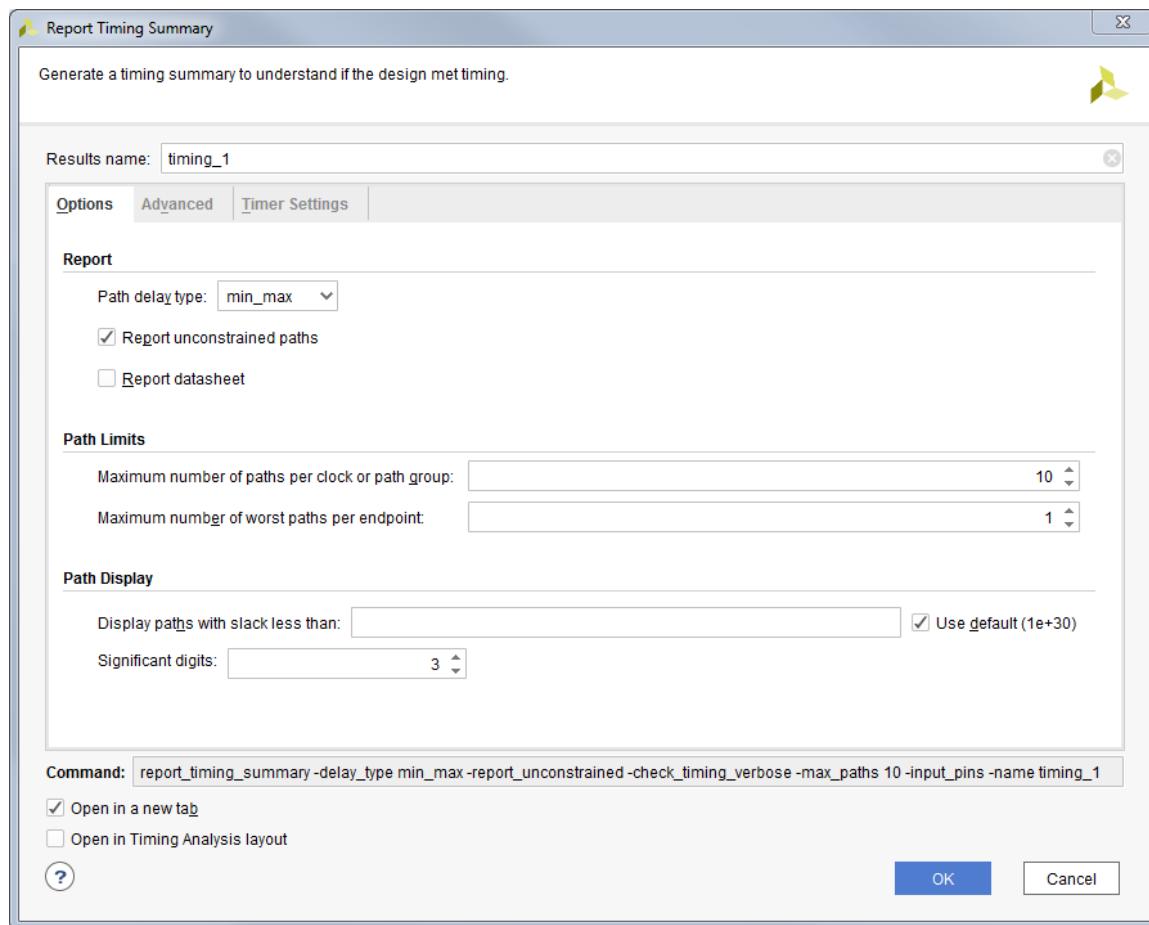


Figure 137: Report Timing Summary Dialog Box

- 1-1-2. Click **OK** to generate the Timing Summary report using the default settings.

The Design Timing Summary window opens at the bottom in the Timing tab.

Running Simultaneous Switching Noise (SSN) Analysis

Simultaneous switching noise (SSN) analysis can be performed to help identify potential signal integrity issues.

1-1. Run SSN on the design.

- 1-1-1.** From the Flow Navigator, click **Report Noise** under Synthesis > Synthesized Design.
- 1-1-2.** Click **OK** in the Report Noise dialog box.

The Noise Report window opens at the bottom in the Noise tab.

Generating a Clock Interaction Report on the Synthesized Design

The clock interaction command is accessible in the Vivado Design Suite GUI once the synthesized design is opened in the Vivado IDE. If the synthesized design is not opened in the GUI and you do not remember how to open the synthesized design, refer to the "Opening the Synthesized Design" section in the *Lab Reference Guide*.

1-1. Generate a Clock Interaction Report.

- 1-1-1.** In the Flow Navigator, under Synthesis > Synthesized Design, select **Report Clock Interaction**.

Alternatively, you can select **Tools > Timing > Report Clock Interaction**.

The Report Clock Interaction dialog box opens.

- 1-1-2.** Click **OK** with the default settings.

The Clock Interaction Report opens in GUI mode.

Closing the Synthesized Design

1-1. Close the synthesized design.

1-1-1. Select **File > Close Synthesized Design**.

The Confirm Close dialog box may open.

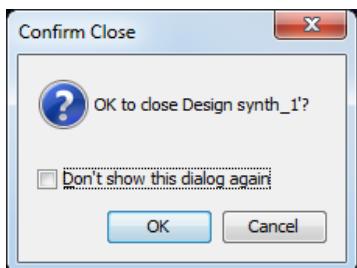


Figure 138: Confirm Close Dialog Box

1-1-2. Click **OK** to close the synthesized design if the Confirm Close dialog box opens.

Alternatively, you can also close the synthesized design by clicking the **X** in the Synthesized Design status bar at the top, or entering the Tcl command `close_design` in the Tcl Console.

Vivado Implementation Operations

In This Section

Setting Implementation Options	107
Running Implementation	109
Running All Tools – RTL to Bitstream	110
Generating a Utilization Report on the Implemented Design	112
Opening the Implemented Design.....	113
Generating a Timing Summary Report on the Implemented Design	113
Generating a Clock Interaction Report	114
Closing the Implemented Design	114
Generating Clock Networks	115
Generating a Power Report on an Implemented Design.....	115

Setting Implementation Options

1-1. Set the implementation options.

- 1-1-1. In the Flow Navigator, under Implementation, click **Implementation Settings**.

Alternatively, you can select **Flow > Implementation Settings**.

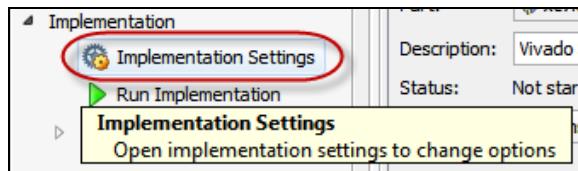


Figure 139: Clicking Implementation Settings

The Implementation Settings dialog box opens.

Note that the default strategy is set to **Vivado Implementation Defaults**.

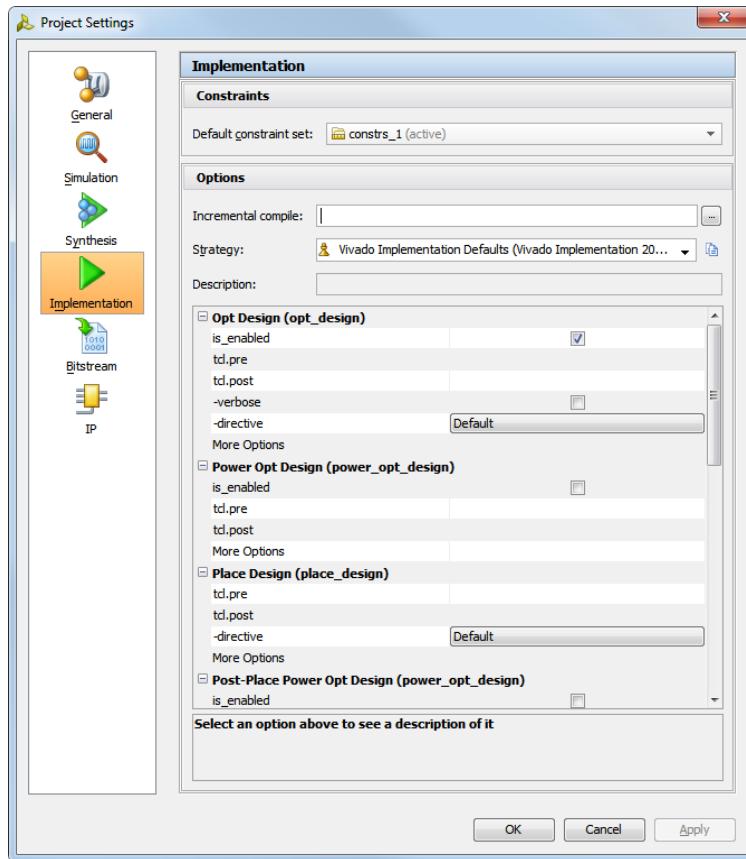


Figure 140: Implementation Settings Dialog Box

Note: You can select any option as needed, such as changing the **-directive** options from **Default** to **Explore**, for example.

- 1-1-2. Click **OK**.

Running Implementation

1-1. Implement the design.

- 1-1-1. Click **Run Implementation** in the Flow Navigator (under Implementation).

Alternatively, you can select **Flow > Run Implementation**.

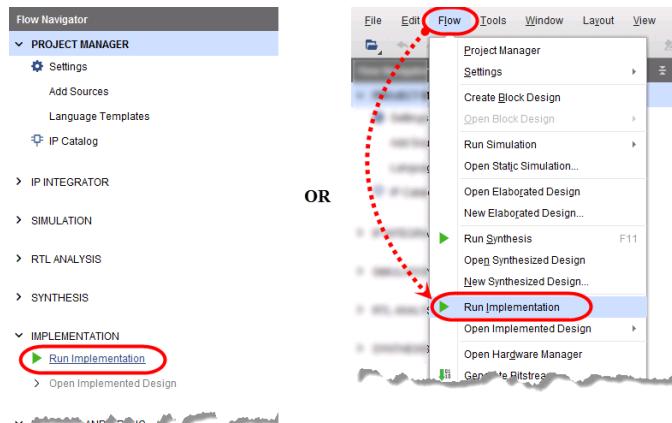


Figure 141: Selecting Run Implementation

Note: If needed, click **OK** to launch synthesis first if prompted.

Notice that the tools run all of the processes required to implement the design. This means that if the synthesized netlist is not available already, the Vivado Design Suite will run synthesis before running implementation.

After implementation completes, the Implementation Completed dialog box opens. The dialog box prompts you to open the implemented design, generate the bitstream, or view reports.

- 1-1-2. Click **OK** again to launch the selected runs.

- 1-1-3. Select **Open Implemented Design**.

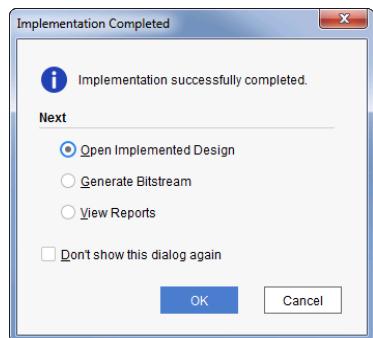


Figure 142: Implementation Completed Dialog Box

- 1-1-4. Click **OK**.

Running All Tools – RTL to Bitstream

At this point, it assumed that the design sources, including HDL files, IP catalog components, and other sources (if present), are instantiated and that the design is functionally completed. Further, all constraint files should have been added to the project by the time the bitstream is generated.

1-1. Synthesize and implement the design and create a bitstream.

These steps can be executed individually to aid in performance examination and debugging. If you are confident that the design source modules are error free and the tool properties are properly set, then all three of these steps can be completed by just engaging the Generate Bitstream tool.

This tool with check if the synthesis and implementation runs exist and have been completed error free. This option saves you from having to monitor intermediate tool completion and starting the next tool.

1-1-1. Using the Flow Navigator, under Program and Debug, click **Generate Bitstream**.

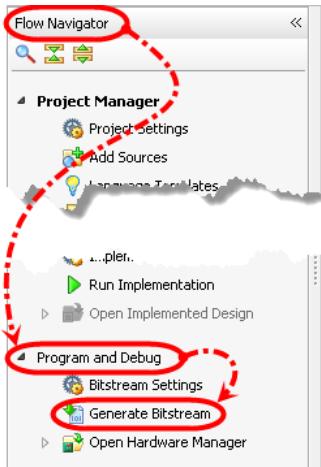


Figure 143: Generating the Bitstream

Alternatively, you can select **Flow > Generate Bitstream**.

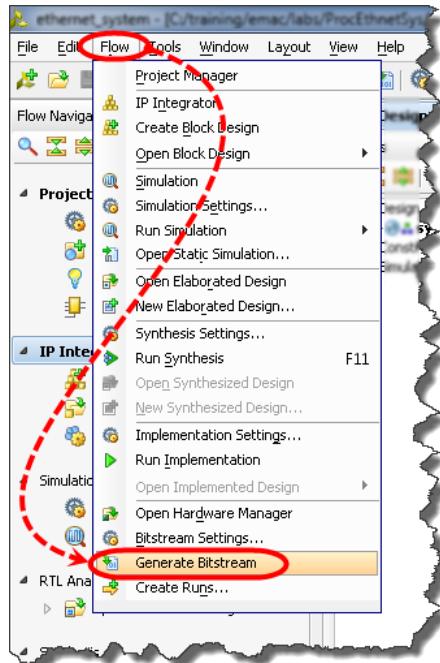


Figure 144: Generating the Bitstream

Notice that the tools run all of the processes required to generate a bitstream. This means that if the synthesized netlist is not available already, the Vivado Design Suite will run synthesis before running implementation and then generating the bitstream.

- 1-1-2. If the No Implementation Results Available dialog box appears, click **Yes** to proceed with launching the synthesis and implementation tools.

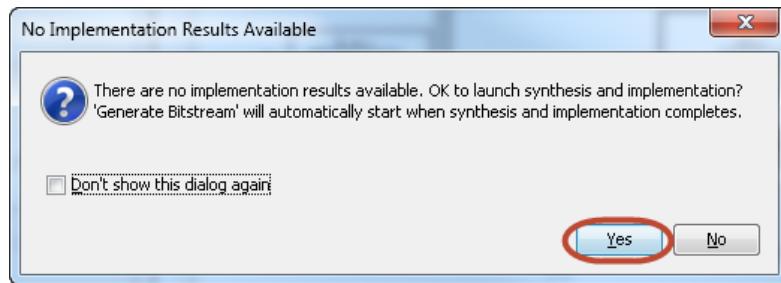


Figure 145: No Implementation Results Available Dialog Box

After bitstream generation completes, the Bitstream Generation Completed dialog box opens. The dialog box prompts you to open the implemented design, view reports, or open the hardware manager.

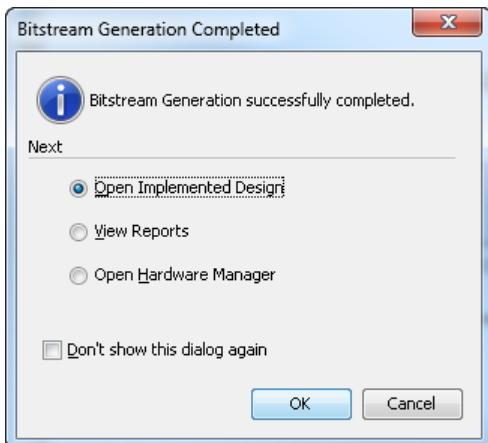


Figure 146: Bitstream Generation Completed Dialog Box

Generating a Utilization Report on the Implemented Design

1-1. Generate a Utilization report on the implemented design.

- 1-1-1. Click **Report Utilization** under Implemented Design in the Flow Navigator.

Alternatively, you can select **Reports > Report Utilization**.

The Report Utilization dialog box opens.

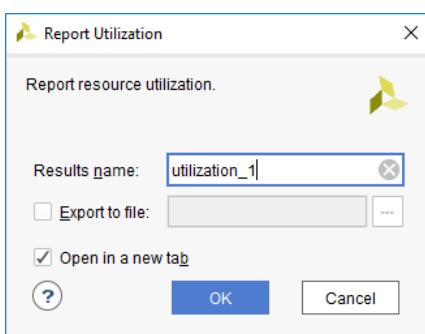


Figure 147: Report Utilization Dialog Box

- 1-1-2. Click **OK**.

The Design Utilization Summary window opens at the bottom in the Utilization tab.

Note that the FPGA resources are listed in a hierarchical format for each RTL design module.

Opening the Implemented Design

1-1. Open the implemented design.

- 1-1-1. Click **Open Implemented Design** under Implementation in the Flow Navigator.

Alternatively, you can select **Flow > Open Implemented Design**.



Figure 148: Opening the Implemented Design

Note that the Timing Summary report is automatically generated in read only mode when the implemented design is opened.

Generating a Timing Summary Report on the Implemented Design

The timing summary command is accessible in the Vivado Design Suite GUI once the implemented design is opened in the Vivado IDE. If the implemented design is not opened in the GUI and you do not remember how to open the design, refer to the "Opening the Implemented Design" section in the *Lab Reference Guide*.

1-1. Generate a Timing Summary report.

- 1-1-1. Click **Report Timing Summary** under **Implementation > Implemented Design** in the Flow Navigator.

The Report Timing Summary dialog box opens.

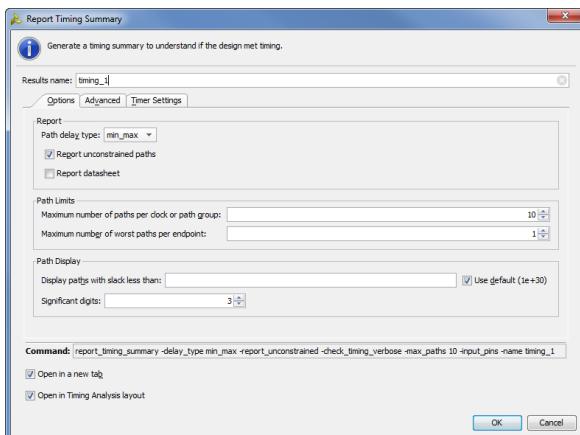


Figure 149: Report Timing Summary Dialog Box

- 1-1-2. Click **OK** to generate the report using the default settings.

The Design Timing Summary window opens at the bottom in the Timing tab.

Note that failing timing paths are indicated in red.

Generating a Clock Interaction Report

The clock interaction command is accessible in the Vivado Design Suite GUI once the implemented design is opened in the Vivado IDE. If the implemented design is not opened in the GUI and you do not remember how to open the implemented design, refer to "Opening the Implemented Design" section in the *Lab Reference Guide*.

1-1. Generate a Clock Interaction Report.

- 1-1-1.** In the Flow Navigator, under Implementation > Implemented Design, select **Report Clock Interaction**.

Alternatively, you select **Tools > Timing > Report Clock Interaction**.

The Report Clock Interaction dialog box opens

- 1-1-2.** Click **OK** with the default settings.

The Clock Interaction Report opens in GUI mode.

Closing the Implemented Design

1-1. Close the implemented design.

- 1-1-1.** Select **File > Close Implemented Design** to close the implemented design.

The Confirm Close dialog box opens.

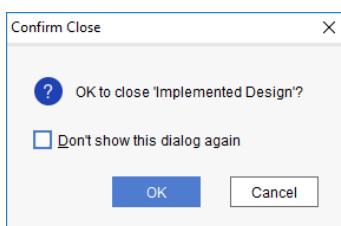


Figure 150: Confirm Close Dialog Box

- 1-1-2.** Click **OK** to close the implemented design.

Alternatively, you can also close the implemented design by selecting **Flow > Close Implemented Design** or clicking the **X** in the Implemented Design status bar at the top.

Generating Clock Networks

1-1. Generate a Clocks Networks report.

- 1-1-1. In the Flow Navigator, under Implemented Design, click **Report Clock Networks**.

Alternatively, you can select **Tools > Report > Report Clock Networks**.

Generating a Power Report on an Implemented Design

1-1. Generate a Power report.

- 1-1-1. In the Flow Navigator, under Implementation > Implemented Design, click **Report Power**.

The Report Power dialog box opens.

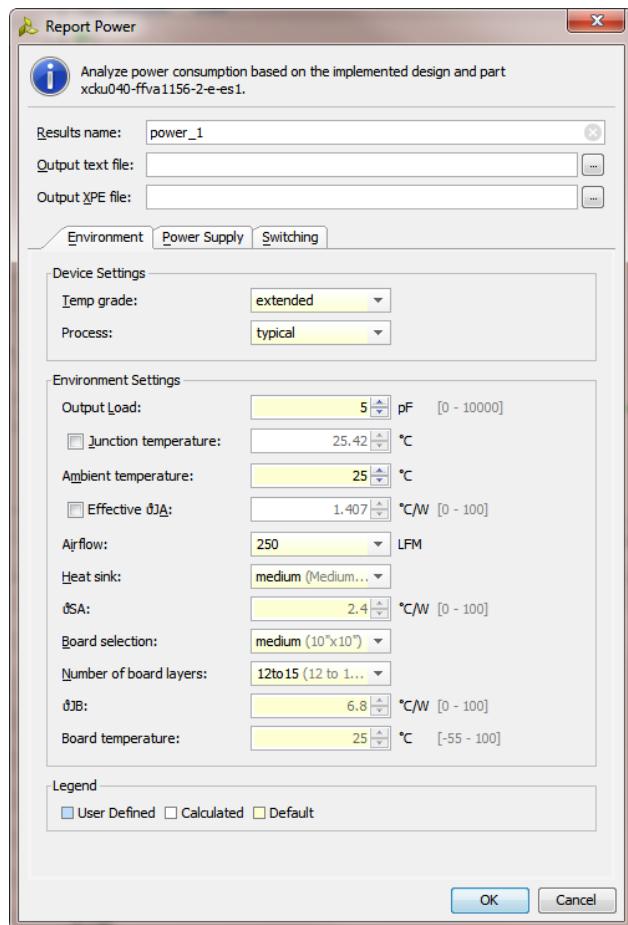


Figure 151: Report Power Dialog Box

- 1-1-2. Click **OK**.

Vivado Tcl Operations

In This Section

Running a Tcl Script from within the Vivado Design Suite	116
Clearing the Tcl Console.....	117
Generating a Timing Summary Report with the Tcl Interface.....	118
Using the llength Command to Obtain the Number of Paths Between Two Domains ...	119
Using the join Command to Join All Timing Paths Between Two Domains	120
Building a Custom Timing Report with the Tcl Interface	120
Generating a List of High Fanout Nets in the Design	121
Building a Custom Timing Report.....	123

Running a Tcl Script from within the Vivado Design Suite

The Vivado Design Suite offers both GUI and scripted control. Scripted control takes the form of Tcl commands. These Tcl commands can be entered directly into the tool one at a time, or an entire Tcl script can be loaded and executed.

1-1. Run a Tcl script.

1-1-1. Locate the Tcl command line entry.

The command line entry can be found either on the Welcome page prior to a project being opened, or once a project has been opened.

From the Welcome screen:

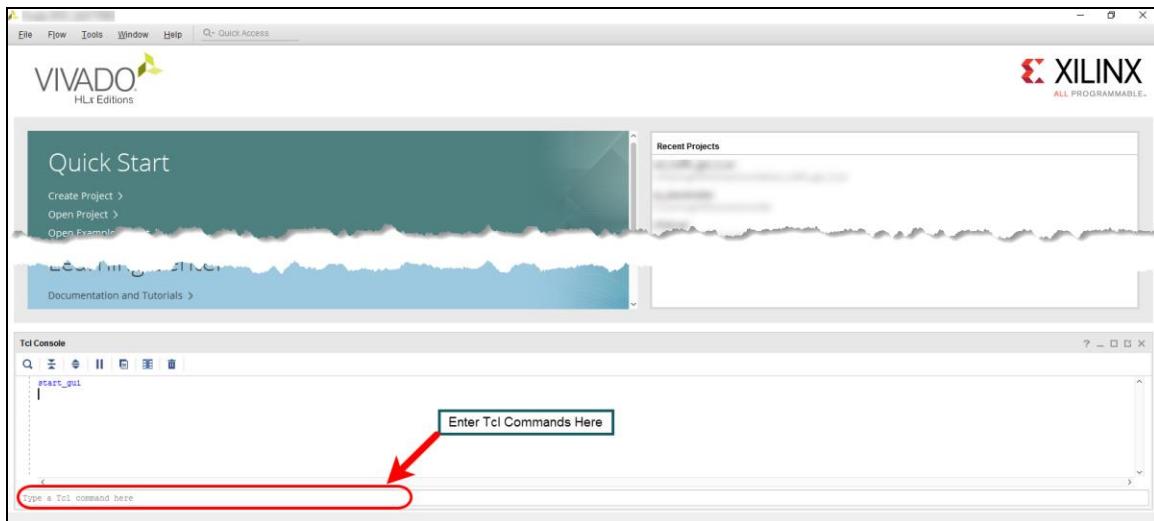


Figure 152: Accessing the Tcl Console from the Getting Started Page

From an opened project:

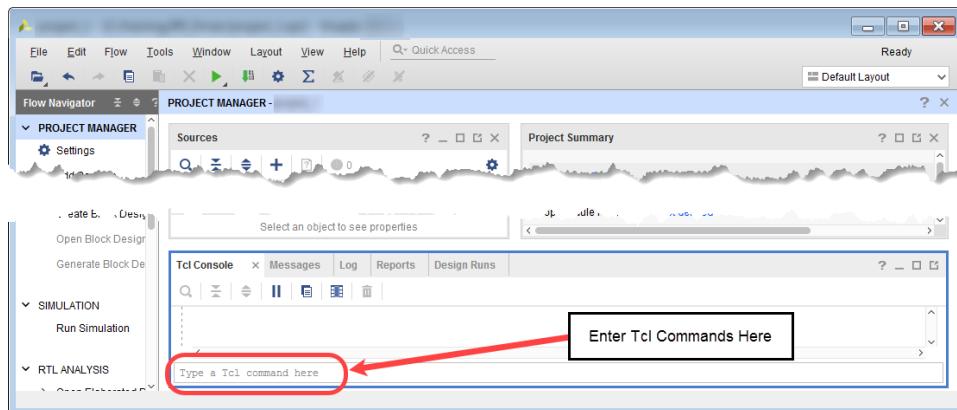


Figure 153: Entering Commands into the Tcl Console from an Open Project

The default directory for the Tcl environment is nested within the Xilinx installation directory. This placement, however, is often disadvantageous. In most cases, you will want to navigate to a more useful path. To do this, use the `cd` command to change directory to the user directory.

- 1-1-2. Change the current working directory to where the Tcl script is located by entering:

```
cd $::env(<the topic cluster name>) /support
```

- 1-1-3. Verify that you are now where you want to be by entering the following into the Tcl command line:

```
pwd
```

This should show that you are within the *<the topic cluster name>* directory within the training directory. Note that the training directory can be anywhere in the system. The import part is that you are currently working from the *support* directory under *<the topic cluster name>*.

- 1-1-4. Run the following Tcl command to run the helper Tcl script for this lab:

```
source your Tcl script.tcl
```

The Tcl script is run as though you typed each command included in the Tcl script into the Tcl command line. You can follow the execution of the script and monitor for any errors or warnings in the Tcl Console.

Clearing the Tcl Console

1-1. Clear the Tcl Console window.

This helps to make it clear how each command is treated.

- 1-1-1. Right-click in the Tcl Console (not the Tcl command line) and select **Clear All Output**.

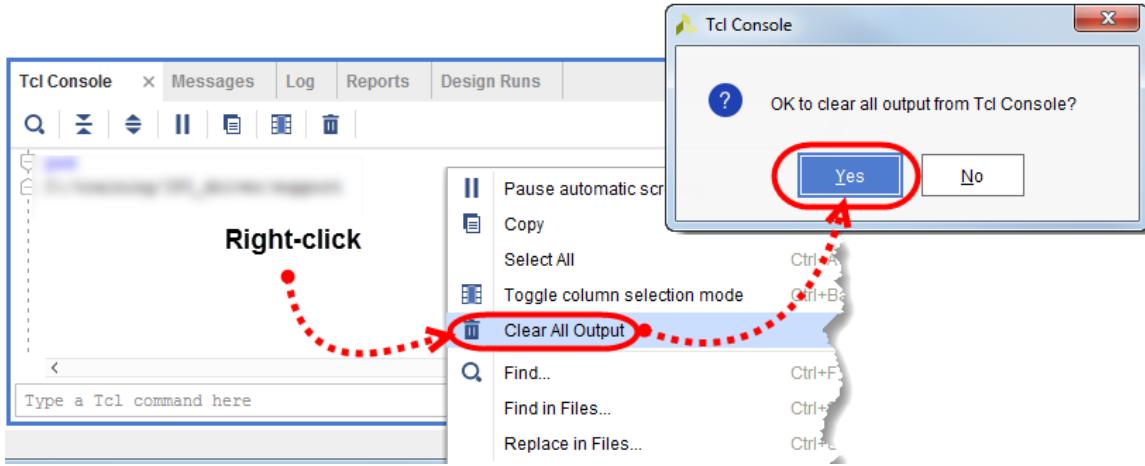


Figure 154: Clearing the Tcl Console

- 1-1-2. Click **Yes** to clear the Tcl Console.

Generating a Timing Summary Report with the Tcl Interface

The timing summary command is accessible in the Vivado Design Suite GUI once the synthesized design or implemented design is opened in the Vivado IDE. If the synthesized/implemented design is not opened in the GUI and you do not remember how to open the design, refer to the "Opening the Synthesized Design" or "Opening the Implemented Design" section in the *Lab Reference Guide*.

1-1. Generate a timing report with the Tcl interface.

- 1-1-1. In the Tcl Console at the bottom of the GUI, enter the following command:

```
report_timing_summary -delay_type max -report_unconstrained  
-check_timing_verbose -max_paths 10 -input_pins
```

This displays the 10 worst paths per clock or per group and reports the unconstrained paths as well.

- 1-1-2.** If you want to save the timing results to a text file, enter the following command:

```
report_timing_summary -delay_type max -report_unconstrained
-check_timing_verbose -max_paths 10 -input_pins -file
C:/<file_location>/<file_name>.txt
```

- 1-1-3.** If you want to see the timing results in the GUI use the `-name` option:

```
report_timing_summary -delay_type max -report_unconstrained
-check_timing_verbose -max_paths 10 -input_pins -name timing_1 -file
C:/<file location>/<file_name>.txt -name timing_1
```

- `-delay_type` – Values are min, max and min_max (sets the type of analysis to be run. For synthesized designs, only max delay analysis).
- `-report_unconstrained` – Reports the unconstrained paths in the report.
- `-check_timing_verbose` – Reports the missing timing constraints in the report.
- `-max_paths` – Number of worst paths to be displayed in the timing report.
- `-input_pins` – Shows the inputs pins the timing report.
- `-file` – Writes the output timing results to a file to the specified location.
- `-name` – To see the results in GUI mode.

Using the `llength` Command to Obtain the Number of Paths Between Two Domains

The `llength` Tcl command can be used to return the length of the list (elements).

For example:

```
set Vivado [list 1 2 3] --> returns the 1 2 3
length $Vivado --> returns 3
```

1-1. Use the `llength` command to obtain the number of paths that exists between two domains.

- 1-1-1.** In the Tcl Console, generate the list of timing paths that exist between two domains with `get_timing_paths`:

```
get_timing_paths -from [get_clocks <clock_name>] -to [get_clocks
<clock_name>] -max_paths 100
```

This command will display the timing paths that exist between the two clock domains.

- 1-1-2.** Use the `llength` command to return the number of paths that exist between the two clock domains:

```
llength [get_timing_paths -from [get_clocks <clock_name>] -to
[get_clocks <clock_name>] -max_paths 100]
```

Using the join Command to Join All Timing Paths Between Two Domains

The `join` Tcl command can be used to join two strings or characters with a new line or another character.

For example:

```
set Vivado [1 2 3]
```

returns --> 1 2 3

```
join $Vivado \n
```

returns --> 1

2

3

1-1. Use the `join` command to join the timing paths between two domains with a new line.

- 1-1-1. In the Tcl Console, generate the list of timing paths that exist between two domains with `get_timing_paths`:

```
get_timing_paths -from [get_clocks <clock_name>] -to [get_clocks <clock_name>] -max_paths 100
```

This command will display the timing paths that exist between the two clock domains.

- 1-1-2. Join the timing paths with a new line.

```
join [get_timing_paths -from [get_clocks <clock_name>] -to [get_clocks <clock_name>] -max_paths 100] \n
```

Note that `\n` adds a new line for each timing path.

Building a Custom Timing Report with the Tcl Interface

The `report_timing` command is accessible in the Vivado Design Suite GUI once the synthesized design or implemented design is opened in the Vivado IDE. If the synthesized/implemented design is not opened in the GUI and you do not remember how to open the design, refer to the "Opening the Synthesized Design" or "Opening the Implemented Design" section in the *Lab Reference Guide*.

The `report_timing` command displays specific timing paths only.

1-1. Generate a custom timing report with Tcl interface.

1-1-1. In the Tcl Console at the bottom of the GUI, enter the following command:

```
report_timing -from [startup points] -to [end points] -delay_type  
min_max -max_paths 10 -sort_by group -input_pins -name timing_1
```

- `-from` – Startup points such as sequential components, F/F pins, etc.
- `-to` – End points such as data inputs of sequential components, etc.
- `-delay_type` – Values are min, max and min_max (sets the type of analysis to be run. For synthesized designs, only max delay analysis).
- `-report_unconstrained` – Reports the unconstrained paths in the report.
- `-check_timing_verbose` – Reports the missing timing constraints in the report.
- `-max_paths` – Number of worst paths to be displayed in the timing report.
- `-input_pins` – Shows the inputs pins the timing report.
- `-file` – Writes the output timing results to a file to the specified location.
- `-name` – To see the results in GUI mode.

For more information about the options that can be used with `report_timing`, type `report_timing -help` in the Tcl Console.

Generating a List of High Fanout Nets in the Design

1-1. Generate a list of high fanout nets in the design.

1-1-1. Enter the following command in the Tcl Console:

```
report_high_fanout_nets
```

The above command generates with default settings a report identifying the high fanout nets.

The default settings are:

- -max_nets = 10 → Number of nets to report in the report.
- -min_fanout = 1 → Report nets that have fanout greater than or equal to the specified integer. This allows you to report nets with specific fanout loads of interest.
- -max_fanout = no maximum limit for default → Report nets that have fanout less than or equal to the specified integer. This allows you to report nets with specific fanout loads of interest.

The report shows the fanout, driver type and net name.

```
Tcl Console
report_high_fanout_nets
Copyright 1986-2014 Xilinx, Inc. All Rights Reserved.

Tool Version : Vivado v.2014.3.0 (win64) Build 982312 Mon Aug 11 19:45:43 MDT 2014
Date : Mon Aug 25 11:43:59 2014
Host : XHDYOGITHAR30 running 64-bit Service Pack 1 (build 7601)
Command : report_high_fanout_nets
Design : wave_gen
Device : xc7k70t

-----
High Fan-out Nets Information

1. Summary
-----

+-----+-----+
| Net Name | Fanout | Driver Type |
+-----+-----+
| rst_gen_i0/reset_bridge_clk_rx_i0/rst_clk_rx | 253 | FDPE |
| clk_gen_i0/clk_core_i0/CLK_OUT2 | 149 | BUFG |
| rst_gen_i0/reset_bridge_clk_tx_i0/rst_clk_tx | 129 | FDPE |
| resp_gen_i0/Q[0] | 43 | FDRE |
| resp_gen_i0/Q[1] | 39 | FDRE |
| cmd_parse_i0/O[4][1] | 35 | FDRE |
| clk_gen_i0/clk_div_i0/n_0_cnt[0]_i_2 | 32 | LUT5 |
| clk_gen_i0/clk_div_i0/n_0_cnt_reg[0] | 32 | FDRE |
| uart_rx_i0/uart_rx_ctl_i0/O8 | 32 | FDRE |
| rst_gen_i0/reset_bridge_clk Samp_i0/rst_clk Samp | 31 | FDPE |
+-----+-----+

INFO: [Vivado 12-3468] Generate high fanout nets report | CPU: 0.000000 secs
```

Figure 155: report_high_fanout_nets Report

Building a Custom Timing Report

The Report Timing options are identical to the Report Timing Summary options with the addition of a few more filtering options. Note that Report Timing does not cover Pulse Width reports.

1-1. Run the `report_timing` command to view specific timing paths.

1-1-1. Select **Tools > Timing > Report Timing**.

The Report Timing dialog box opens.

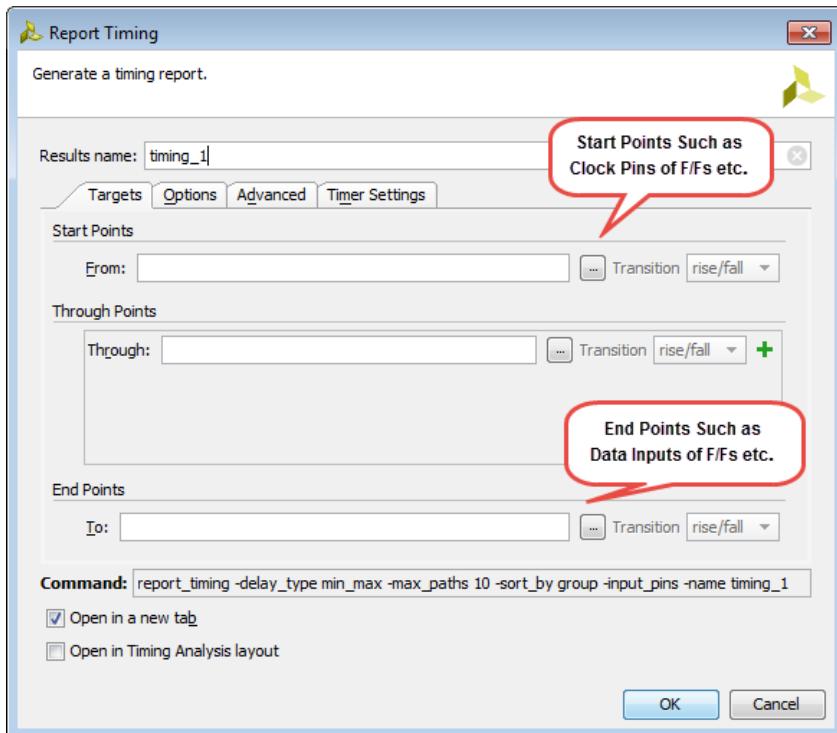


Figure 156: Report Timing Dialog Box: Targets Tab

Report Timing provides several filtering options that you must use in order to report a particular path or group of paths. The filters are based on the structure of a timing path.

- **Startpoints (From):** List of startpoints, such as sequential cell clock pins, sequential cells, input ports, bidirectional ports, or the source clock. If you combine several startpoints in a list, the reported paths will start from any of these netlist objects. The Rise/Fall filter selects a particular source clock edge.
- **Through Point Groups (Through):** List of pins, ports, combinational cells, or nets. You can combine several netlist objects in one list if you want to filter on paths that traverse any of them. You can also specify several Through options to refine your filters and report paths that traverse all groups of through points in the same order as they are listed in the command options.

- **Endpoints (To):** List of endpoints, such as input data pins of sequential cells, sequential cells, output ports, bidirectional ports or destination clock. If you combine several endpoints in a list, the reported paths will end with any of these netlist objects. In general, the Rise/Fall option selects a particular data edge. But if you specified a destination clock, it selects a particular clock edge.

The remaining three tabs in the Report Timing dialog box are similar to the Report Timing Summary dialog box. For more information, refer to the "Generating a Timing Summary Report with Description" section under in the *Lab Reference Guide*.

You also can customize the timing report by selecting the required startup points and end points and generating the custom timing report to view these specific timing paths only.

Vivado Timing Miscellaneous Operations

In This Section

Generating a Timing Summary Report with Description	124
Identifying the Failing Paths from a Timing Report.....	126
Understanding the Path Detail Info from a Timing Path Report	126
Generating a Custom Schematic to Display Selected Number of Failing Timing Paths..	129

Generating a Timing Summary Report with Description

The timing summary command is accessible in the Vivado Design Suite GUI once the synthesized design or implemented design is opened in the Vivado IDE. If the synthesized/implemented design is not opened in the GUI and you do not remember how to open the design, refer to the "Opening the Synthesized Design" or "Opening the Implemented Design" section in the *Lab Reference Guide*.

1-1. Generate a timing summary report.

- 1-1-1. In the Flow Navigator, under Synthesized Design or Implemented Design, click **Report Timing Summary**.

The Report Timing Summary dialog box opens.

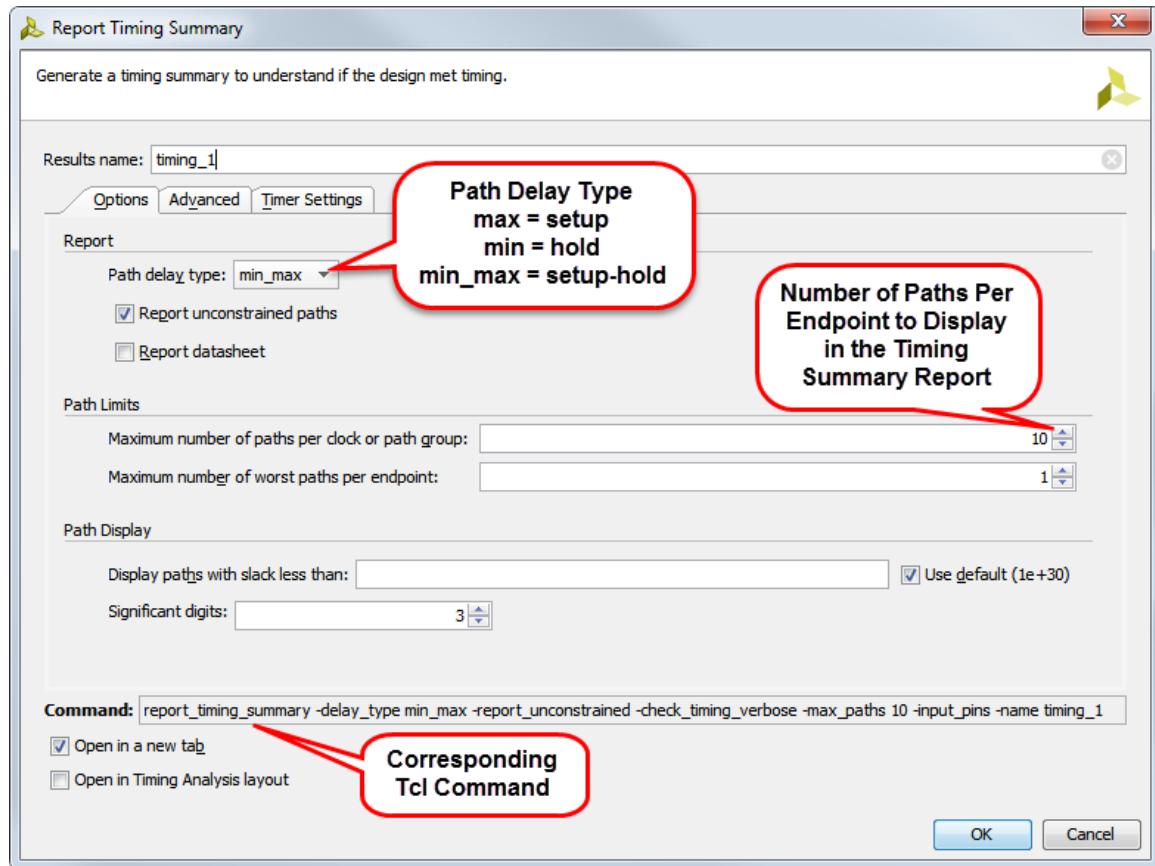


Figure 157: Timing Summary Report Dialog Box: Options Tab

- 1-1-2. Select the options in the Report Timing Summary dialog box and then click **OK**.

The Timing - Timing Summary -timing_1 window opens in the Timing tab at the bottom of the GUI.

Note that timing_1 is the default name of the timing report. You can change the name in the Report Timing Summary dialog box.

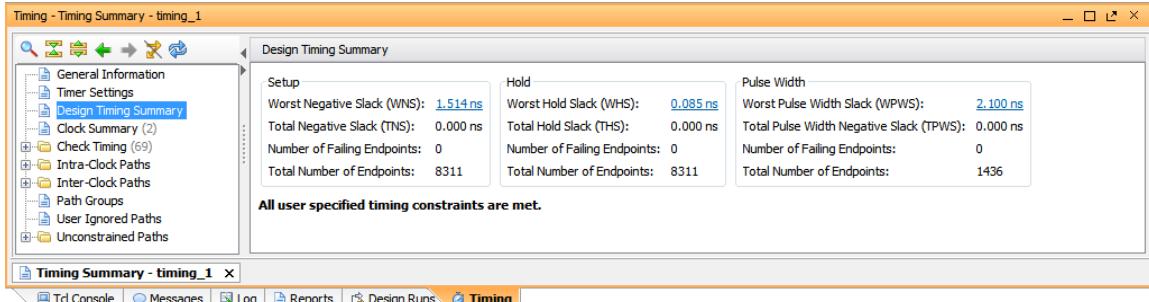


Figure 158: Design Timing Summary Window

Identifying the Failing Paths from a Timing Report

In the timing report, the failing timing paths are displayed in red. In other words, the timing report shows slack values of failing timing paths in red.

Understanding the Path Detail Info from a Timing Path Report

The Timing Path Summary displays important information from the timing path details. You can review it to determine the cause of a violation without having to analyze the details of the timing path. Information includes slack, path requirement, data path delay, cell delay, route delay, clock skew, and clock uncertainty.

1-1. Review and understand the path detail information from the timing path report.

- 1-1-1. Double-click a particular path to view its detailed timing path report.

This opens a Path <number> - report_name tab in the main workspace area.

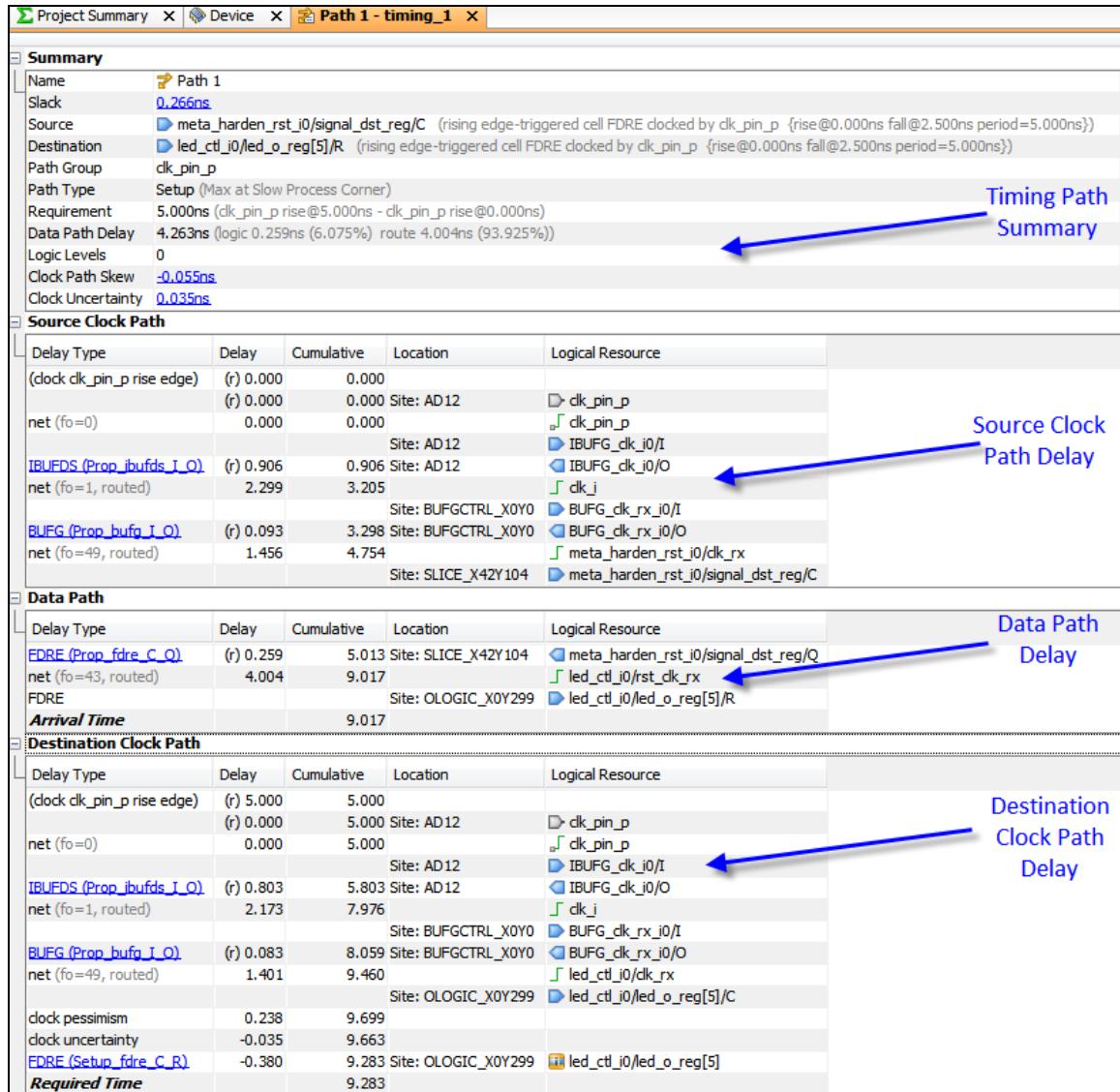


Figure 159: Path Detail Information of a Selected Path

The timing path properties report provides a detailed summary of the timing path covered by the specific clock path group. When you click the links in the report, the logic objects are selected in other views. The timing path report provides the detailed information of the logic objects in the path and their associated delays for the source clock path, data path and the destination clock path. The details of the timing path report is as follows.

Timing Path Summary: Provides brief information about the timing path and reports slack for the timing path endpoints. The slack is the difference between the data required time and the data arrival timing at the path endpoint.

- Slack: A positive slack indicates that the path meets the path requirement, which is derived from the timing constraints. The Slack equation depends on the analysis performed.
 - Max delay analysis (setup/recovery): slack = data required time – data arrival time
 - Min delay analysis (hold/removal): slack = data arrival time – data required time

Data required and arrival times are calculated and reported in the other sub-sections of the timing path report.

- Source: The path startpoint and the source clock that launches the data. The startpoint is usually the clock pin of a sequential cell or an input port. When applicable, the second line displays the primitive and the edge sensitivity of the clock pin. It also provides the clock name and the clock edges definition (waveform and period).
- Destination: The path endpoint and the destination clock that captures the data. The endpoint is usually the input data pin of the destination sequential cell or an output port. Whenever applicable, the second line displays the primitive and the edge sensitivity of the clock pin. It also provides the clock name and the clock edges definition (waveform and period).
- Path Group: The timing group that the path endpoint belongs to. This is usually the group defined by the destination clock, except for asynchronous timing checks (recovery/removal) which are grouped in the ****async_default**** timing group. User-defined groups can also appear here. They are convenient for reporting purpose.
- Path Type: The type of analysis performed on this path.
 - Max indicates that the maximum delay values are used to calculate the data path delay, which corresponds to setup and recovery analysis.
 - Min indicates that the minimum delay values are used to calculate the data path delay, which corresponds to hold and removal analysis.

This line also shows which corner was used for the report: Slow or Fast.

- Requirement: The timing path requirement, which is typically:
 - One clock period for setup/recovery analysis.
 - 0 ns for hold/removal analysis, when the startpoint and endpoint are controlled by the same clock, or by clocks with no phase-shift.

When the path is between two different clocks, the requirement corresponds to the smallest positive difference between any source and destination clock edges. This value is overridden by timing exception constraints such as multicycle path, max delay and min delay.

- Data Path Delay: Accumulated delay through the logic section of the path. The clock delay is excluded unless the clock is used as a data. The type of delay corresponds to what the Path Type line describes.

- Logic Levels: The number of each type of primitives included in the data section of the path, excluding the startpoint and the endpoint cells.
- Clock Path Skew: The insertion delay difference between the launch edge of the source clock and the launch edge of the source clock and the capture edge of the destination clock, plus clock pessimism correction (if any).
- Clock Uncertainty: The total amount of possible time variation between any pair of clock edges. The uncertainty comprises the computed clock jitter (system and discrete), the phase error introduced by certain hardware primitives and any clock uncertainty specified by the user in the design constraints (set_clock_uncertainty). The user clock uncertainty is additive to the uncertainty computed by the Vivado timing engine.

Source Clock Path: Provides the detailed information of the logic objects in the path and their associated delays for the source clock path. This source clock path is the path followed by the source clock from its source point to the clock pin of the launching flip-flop.

Data Path: Provides the detailed information of the logic objects in the path and their associated delays for the internal circuitry, between the launching and capturing flip-flops. The active clock pin of the launching flip-flop is called the path startpoint. The data input pin of the capturing flip-flop is called the path endpoint.

Destination Clock Path: Provides the detailed information of the logic objects in the path and their associated delays for the destination clock path. The destination clock path is the path followed by the destination clock from its source point, typically an input port, to the clock pin of the capturing flip-flop.

Generating a Custom Schematic to Display Selected Number of Failing Timing Paths

1-1. Generate a custom schematic to display a selected number of failing timing paths from the timing report.

- 1-1-1. In Design Timing summary window, select the **Show only failing checks** () icon in the top-left corner to show only failing timing paths.

- 1-1-2. Select the **N** number of failing paths using the the <**Shift**> or <**Ctrl**> key.

N = required number such as 10, 11 , 20, etc.

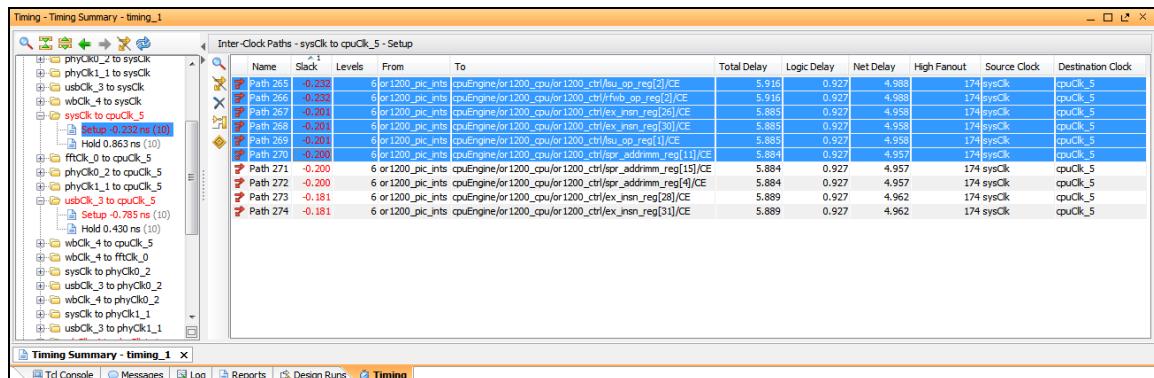


Figure 160: Selecting Multiple Failing Timing Paths

- 1-1-3. Right-click and select **Schematic**, or press **F4**.

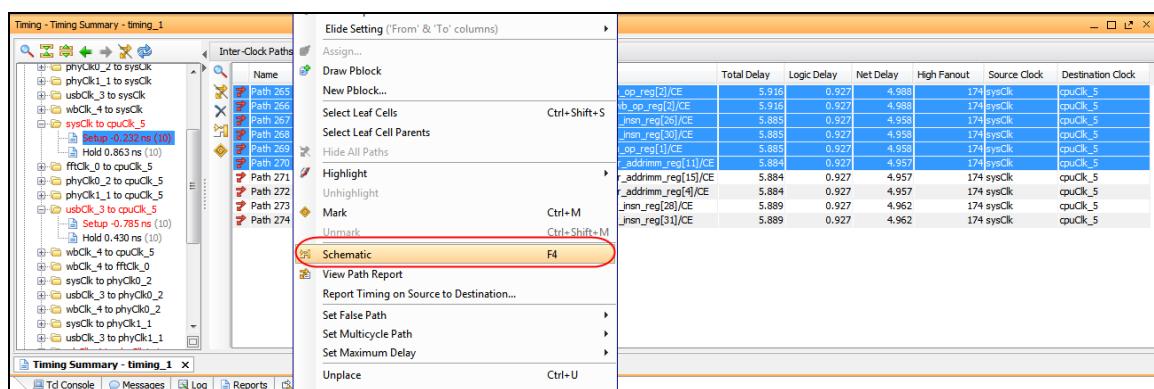


Figure 161: Selecting the Schematic Option for the Selected Paths

The Schematic tab opens in the main workspace area, showing the schematic for the selected paths.

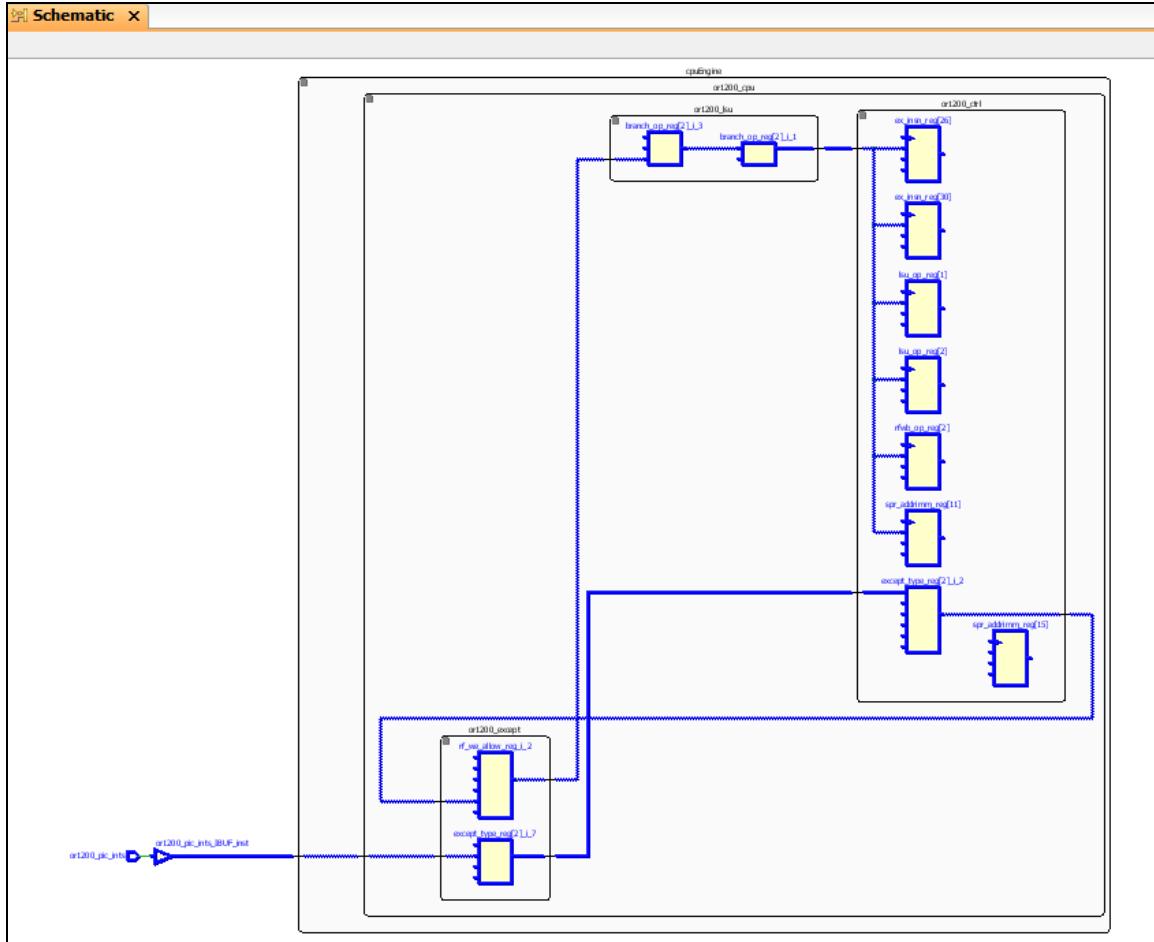


Figure 162: Schematic Tab in the Main Workspace Area

Likewise, you can generate any custom schematic to view associated logic in the selected timing path(s).

Vivado Hardware Session Operations

In This Section

Implementing a Design and Generating a Bitstream	132
Opening the Hardware Manager	133
Downloading a Bitstream Using the Hardware Manager.....	136
Generating the Bitstream and Downloading the Design.....	138
Generating the Bitstream.....	141
Launching and Configuring the Tera Term Terminal Program in Serial Port Mode.....	141

Implementing a Design and Generating a Bitstream

1-1. Generate the implemented design and bitstream.

The Vivado Design Suite runs the equivalent of a "make" file that understands dependencies. If synthesis or implementation has not been run, and bitstream generation is requested, then the tools are smart enough to synthesize whatever modules need to be synthesized and implement the design prior to generating the bitstream.

- 1-1-1. Click **Generate Bitstream** from the Flow Navigator under Program and Debug to run all operations necessary to generate the bitstream.

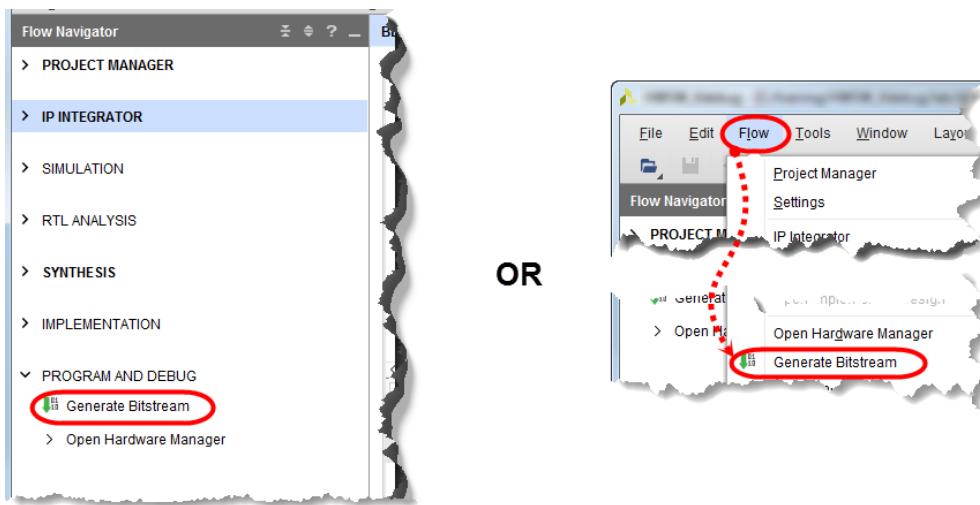


Figure 163: Launching the Generate Bitstream Process

- 1-1-2. Click **OK** if any critical warning messages dialog boxes appear and continue the bitstream generation operation.

You will need to determine if the critical warnings need to be addressed immediately or if they can be ignored.

When implementation is complete, the Implementation Complete dialog box opens.

The status indicator in the upper-right corner of the workspace will indicate when bitstream generation is complete as well as in the Design Runs tab in the bottom panel.

When generation is complete, the Bitstream Generation Completed dialog box opens.

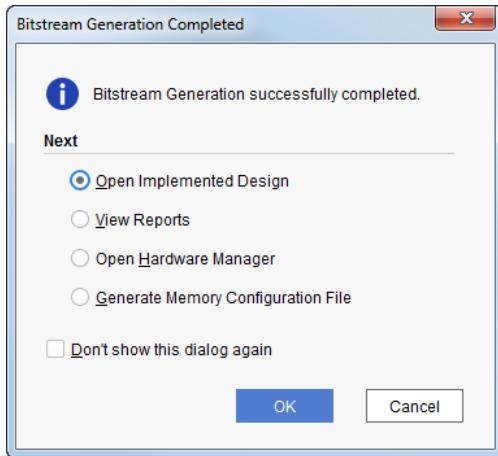


Figure 164: Bitstream Generation Completed Dialog Box

You can view reports to review what was created or open the hardware manager to program the device or view the information from the Vivado analyzer tool. You can also simply cancel to perform neither of these tasks. These tasks, and others, are still available through the Flow Navigator.

- 1-1-3. Click **Cancel** to perform none of these tasks and end the bitstream generation process, or, if you would like to quickly jump to one of these frequently used next-steps, merely select the option of your choice and click **OK**.

Opening the Hardware Manager

A hardware manager is the portion of the Vivado Design Suite that enables the monitoring of cores that were added to a design.

1-1. Open the hardware manager.

- 1-1-1. Click **Open Hardware Manager** in the Bitstream Completed dialog box and click **OK**.

The Hardware Manager window opens.

The hardware needs to be connected and the information bar invites you to open an existing or a new target.

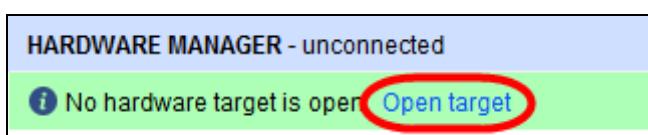


Figure 165: Opening a Hardware Target

1-2. Connect the target through the New Target Wizard to guide you through the process.

1-2-1. Click Open target > Open New Target.



Figure 166: Open Hardware Target Dialog Box

1-2-2. Click **Next to set the hardware server settings.**

1-2-3. Enter a name for the server.

Typically this is left at its default value.

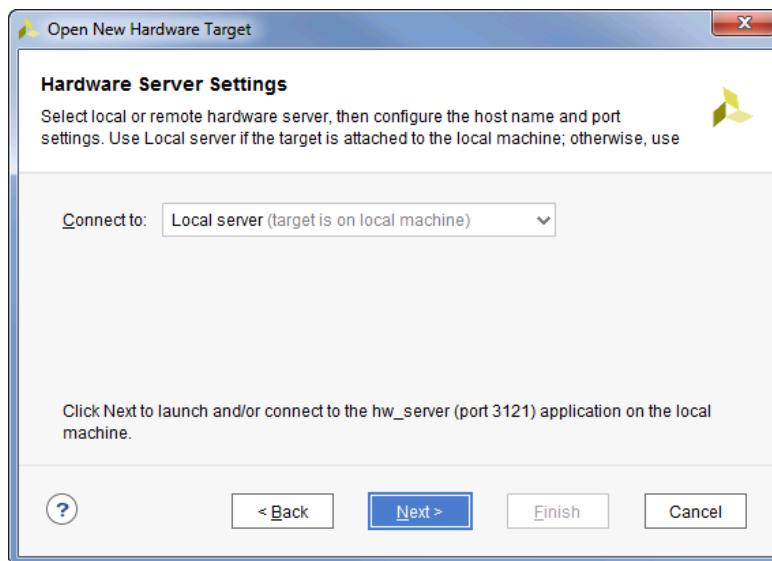


Figure 167: Setting the Server Name for the New Hardware Target

1-2-4. Click **Next to select the hardware target.**

1-2-5. Verify the hardware target.

This becomes important when there are multiple targets connected to the PC.

You can change the frequency of the JTAG cable if you are experiencing communications problems.

[Linux users]: If you receive an error saying that no active target is found, check the USB connections by going to **Devices > USB** in the VirtualBox toolbar at the top.

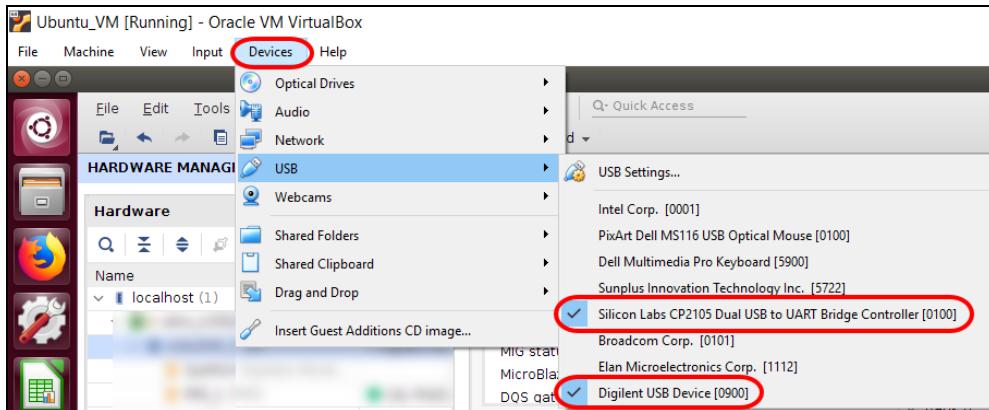


Figure 168: Selecting the Hardware Target

- 1-2-6. Leave the frequency at its default value.

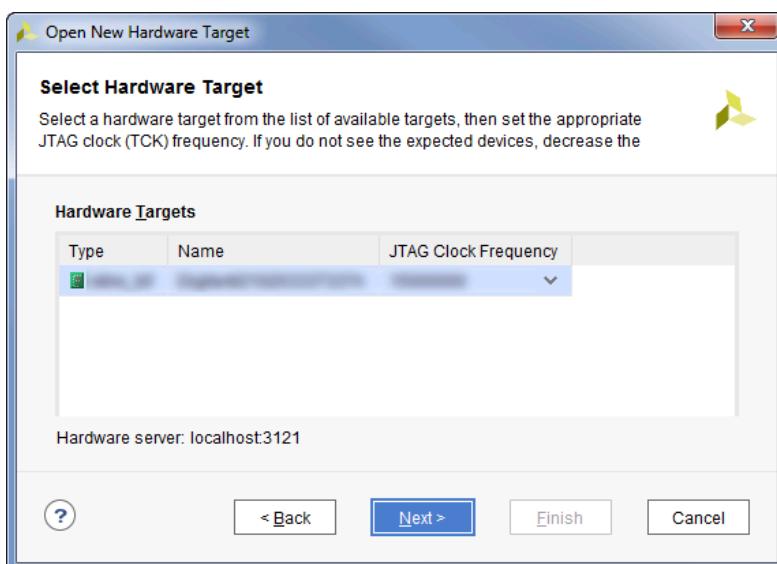


Figure 169: Selecting the Hardware Target

- 1-2-7. Click **Next** to view the hardware target summary.

A summary of the connection is displayed.

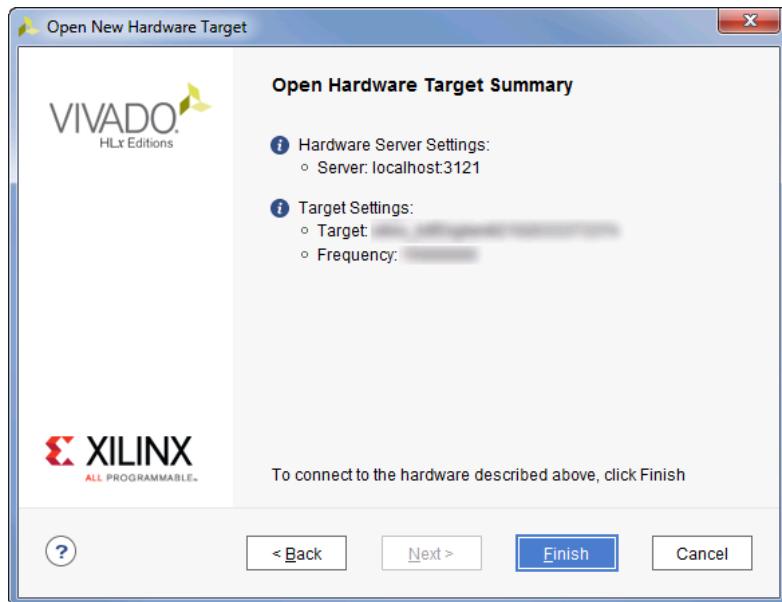


Figure 170: Summary of the Open Hardware Target Settings

- 1-2-8. Click **Finish** to connect to the new hardware target.

Downloading a Bitstream Using the Hardware Manager

Now that a hardware session is established, your task is to download a bitstream to your board.

1-1. Download a *bitstream* to the target board.

- 1-1-1. From the Hardware pane, identify the FPGA/SoC that you would like to program.
- 1-1-2. Right-click that entry and select **Program Device**.

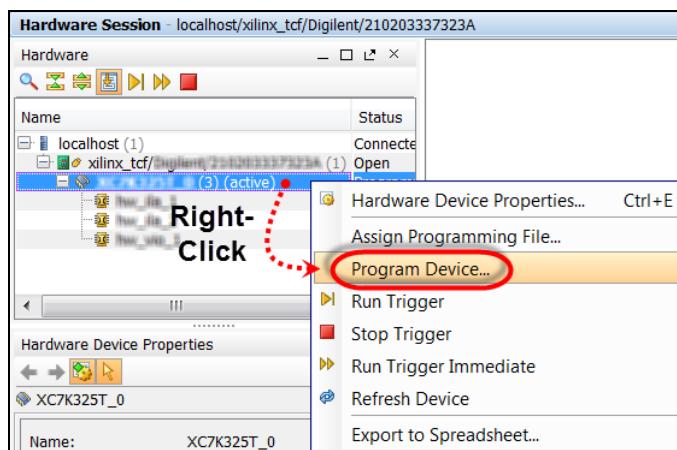


Figure 171: Selecting Program Device

- 1-1-3. In the Program Device dialog box that opens, you can either use the default project bitstream or navigate to another bitstream.

It is normal for the *Debug probes file* field to be blank if your design has no debug cores.

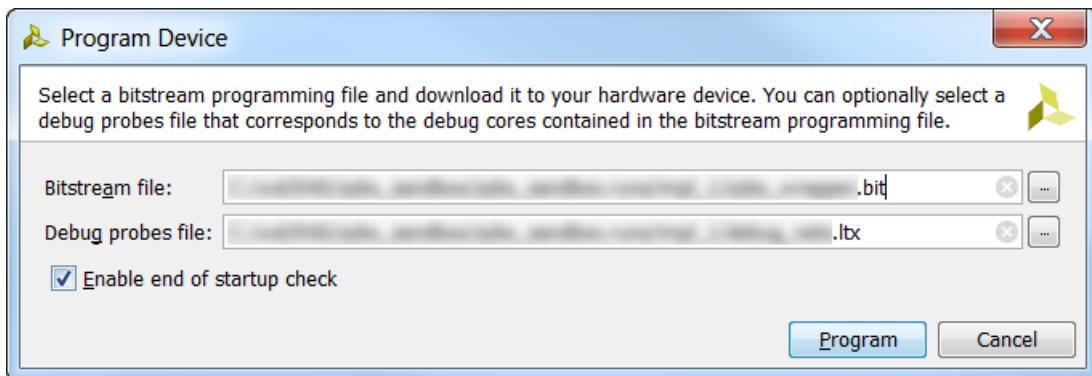


Figure 172: Selecting the Bitstream and Programming the Device

- 1-1-4. Click **Program**.

A progress bar appears and, when complete, the dialog box closes.

Generating the Bitstream and Downloading the Design

1-1. Generate the bitstream.

- 1-1-1. Click **Generate Bitstream** from the Flow Navigator, under Program and Debug.

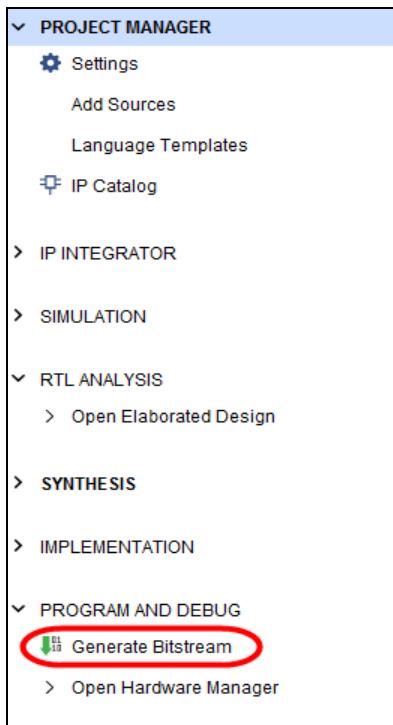


Figure 173: Generate Bitstream

- 1-1-2. Click **OK** to launch the runs.

Note that the Generate Bitstream process will try to resynthesize and implement the design if any process is out of date.

- 1-1-3. Click **Yes** if prompted to rerun any of the processes that are needed for bitstream generation.

- 1-1-4. Click **Cancel** in the Bitstream Generation Completed dialog box.

1-2. Power on the board.

- 1-2-1. Connect a USB cable from a USB port on your computer to the USB UART connector on the evaluation board.
- 1-2-2. Connect the USB cable to the Digilent USB JTAG interface.
- 1-2-3. Ensure that the power cord is plugged in and turn on the evaluation board.

1-2-4. Make sure that the board settings are proper.

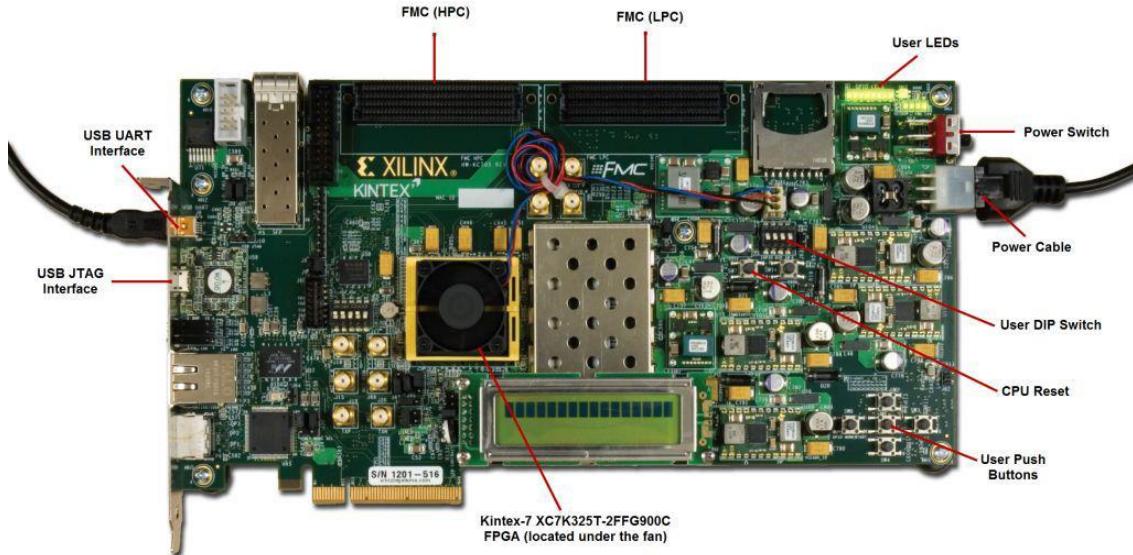


Figure 174: KC705 Evaluation Board

You may be prompted to install drivers when the board is first connected. Do not allow the driver installation to search the Web, but allow it to search for the drivers on your computer.

1-2-5. Ensure that all DIP switches (SW11) are in the off position.

1-3. Program the Kintex FPGA on the your board board using the hardware manager.

- 1-3-1.** Click **Open Hardware Manager** in the Flow Navigator, under Program and Debug.
- 1-3-2.** Click **Open Target > Open New Target**.
- 1-3-3.** Click **Next** in the Open New Hardware Target dialog box.
- 1-3-4.** Click **Next** in the Hardware Server Settings dialog box.
- 1-3-5.** Click **Next** in the Select Hardware Target dialog box.
- 1-3-6.** Click **Finish**.
- 1-3-7.** For the Kintex-7 board, right-click **xc7k325t_0** and select **Program Device**.

For the Kintex UltraScale board, right-click **xcku040_0** and select **Program Device**.

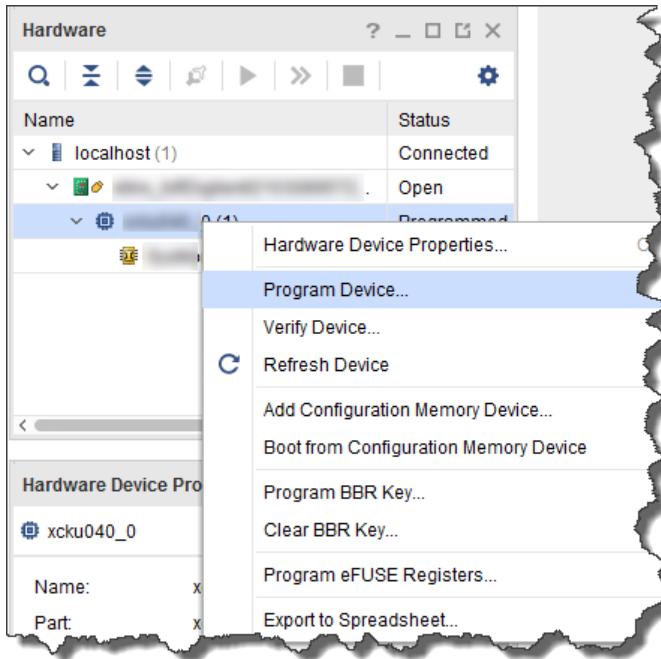


Figure 175: Programming the Device

The Program Device dialog box opens.

Observe that the bit file that will be used to program the device has been included in the Bitstream file field.

1-3-8. Click **Program.**

When programming the FPGA is complete, the status will show Programmed.

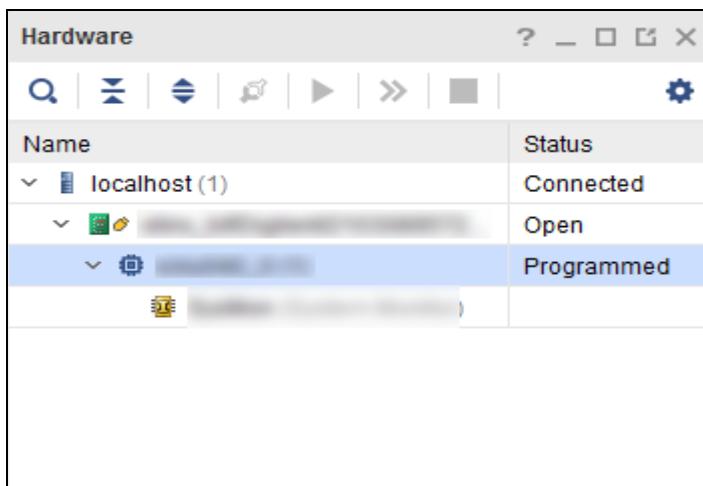


Figure 176: Programming Status of the FPGA

Generating the Bitstream

1-1. Generate the bitstream.

- 1-1-1. From the Flow Navigator, under Program and Debug, click **Generate Bitstream**.

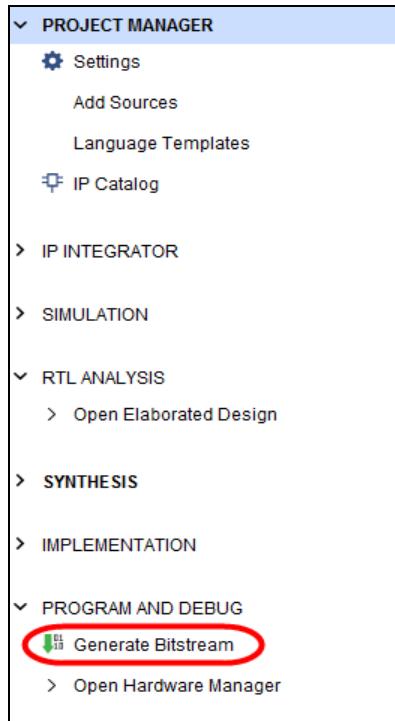


Figure 177: Generate Bitstream

Note that the generate bitstream process will try to resynthesize and implement the design if any process is out of date.

- 1-1-2. Click **Yes** if prompted to rerun any of the processes that are needed for bitstream generation.

Launching and Configuring the Tera Term Terminal Program in Serial Port Mode

Maintaining consistency in the instructions is an important aspect of the lab experience. The labs will use general and vague commands such as "Open the terminal emulator". Depending on the system that you are using, you must know which terminal emulator to configure and launch. Here are the instructions for opening a terminal emulator for both the Linux and Windows environments.

Linux

gtkterm is a simple GTK+ terminal used to communicate with the serial port. While other terminal emulators may be used, the instruction provided here are specific to Tera Term.

1-1. Launch gtkterm and set the port configuration.

- 1-1-1. Press <**Ctrl + Alt + T**> launch a Linux terminal window.
- 1-1-2. Enter the following to launch gtkterm from the terminal window:

```
sudo gtkterm
```

- 1-1-3. When prompted for the password, **xilinx**.

Note: While the application will run as a regular user, you must be a super user to access the ports.

When the GtkTerm window opens, perform the following.

- 1-1-4. Select **Configuration > Port** to open the Configuration dialog box.
- 1-1-5. Identify the port associated with your board and set the port as **/dev/ttyUSBx** (where x could be 0, 1, 2, etc.)
- 1-1-6. Set the baud rate to **115200**.

Leave the rest of the settings at their default.

Note: You can open multiple instances of /dev/USBx if you are unable to find out which port your UART is connected to.

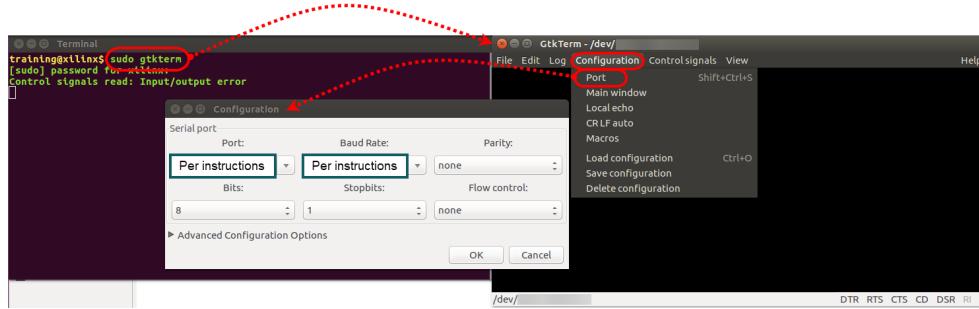


Figure 178: Opening gtkterm and Selecting the Port Configuration

- 1-1-7.** [Optional] You can save these settings so that you do not have to reconfigure GtkTerm each time you open it by clicking **Configuration > Save configuration**.

If you save the configuration as "default" this configuration will open when GtkTerm starts. Otherwise you can save the configuration by another name; however, you will then need to load the configuration each time you start GtkTerm.

- 1-1-8.** Click **OK** to save the settings and leave the terminal open.

Windows

Tera Term is a popular public domain terminal emulation program. It is capable of operating as a serial port terminal or as a telnet client. While other terminal emulators may be used, the instruction provided here are specific to Tera Term.

1-2. Launch the Tera Term terminal program.

- 1-2-1.** Double-click the **Tera Term** icon on the Windows desktop to launch Tera Term.

Alternatively, you can select **Start > All Programs > Tera Term > Tera Term**.

- 1-2-2.** Select **Serial** as the connection (1).

- 1-2-3.** Click the **Port** drop-down list to view the available COM ports (2).

Note: If your port is not listed, exit Tera Term, power cycle your board and re-start this step.

- 1-2-4.** Select the COM # discovered in the last step (3).

Note that if you cannot immediately identify the proper COM port, review "Determining the Active COM Port".

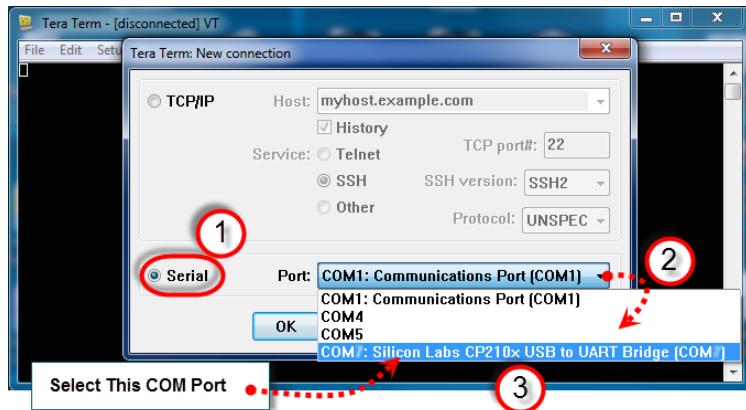


Figure 179: Selecting the COM Port

Note: The COM port setting is specific to the computer being used and may need to be different than shown. Use the COM port # that was discovered in the previous step.

- 1-2-5.** Click **OK**.

The terminal console window opens.

1-2-6. Select **Setup > Serial Port**.

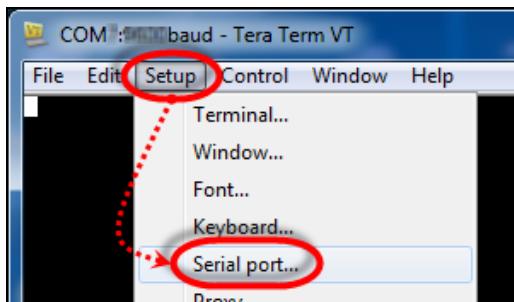


Figure 180: Opening the Tera Term Serial Port Setup Window

The Tera Term Serial Port Setup dialog box opens.

1-2-7. Confirm that the proper serial port has been selected (1).

1-2-8. Set the baud rate to **115200** (2).

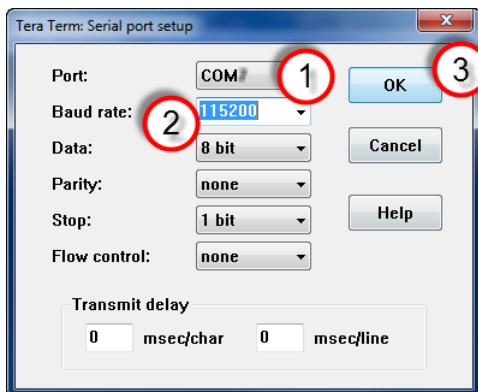


Figure 181: Setting the Parameters for the Serial Port

Note: The COM port setting is specific to the computer being used and may need to be different than shown. Use the COM port # that was discovered in the previous step.

1-2-9. Click **OK** (3).

Tera Term is now configured to receive and transmit serial information to/from the evaluation board.

Vivado Analyzer Operations [Last Updated Version 2017.1]

This sub-section covers many of the common activities associated with configuring IP cores and collecting data.

In This Section

Informative Information	145
Instructions	147

Informative Information

The following information is for background understanding of the materials discussed in the Instructions section.

Overview of the Vivado Analyzer Hardware Session

The following is a quick review of the Hardware Session screen usage.

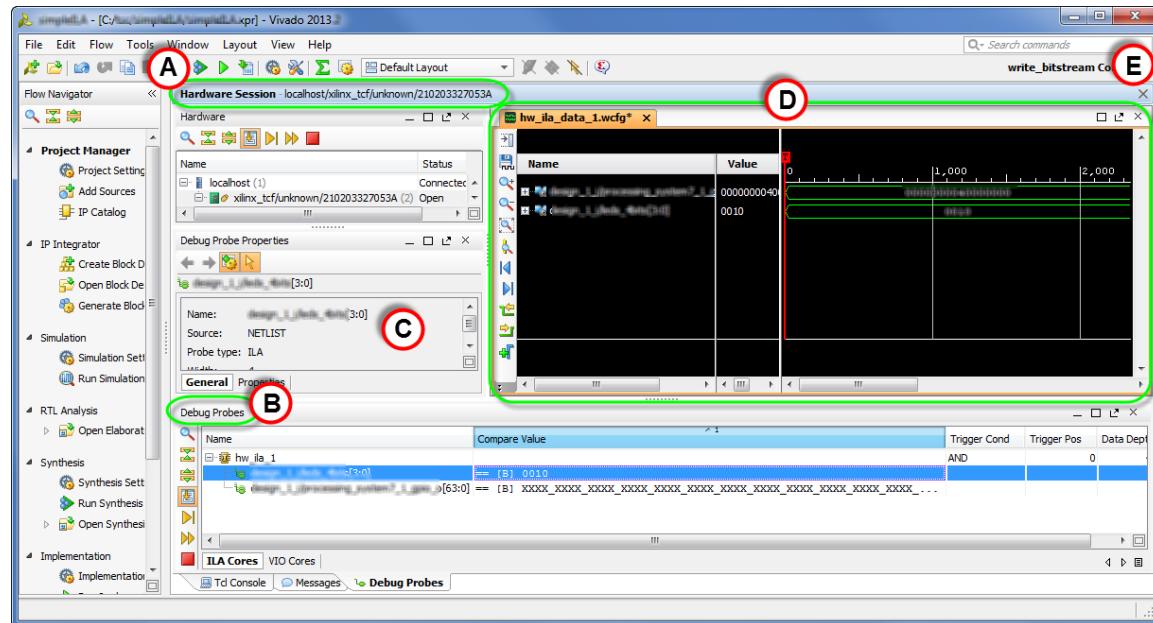


Figure 182: Vivado Analyzer - Hardware Session Screen Usage

- A: Name of the active function (hardware session) and its TCF connection – This shows that you have successfully opened the hardware session.
- B: Debug Probes – This section lists each ILA and VIO and the various probes within each.
- C: Debug Probe Properties – Provides relevant information about the selected probe (the type of device it is attached to, its width, etc.)
- D: Waveform Window – Each tab supports a different ILA or VIO.
- E: Exit Hardware Session – Click the 'X' to exit the hardware session and access other Vivado Design Suite capabilities.

Understanding Trigger Conditions

Trigger conditions describe the desired state of various trigger-capable signals. When these settings are found, the ILA will trigger and capture data.

The Vivado Analyzer recognizes five different signal conditions: Rising, Falling, 1, 0, and X (don't care). These signals conditions can be logically joined to describe more complex situations.

First, consider a single probe example:

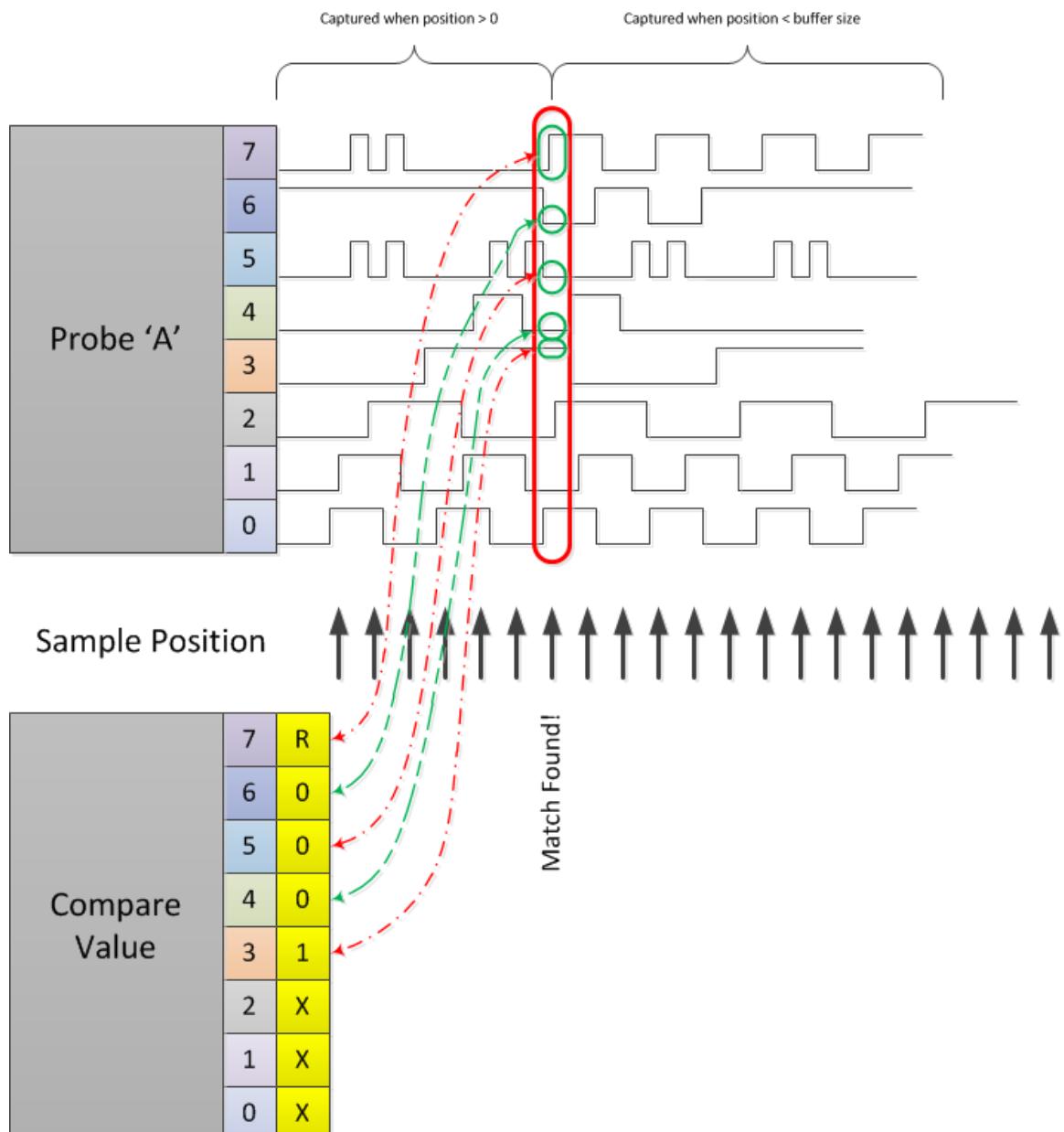


Figure 183: An Individual Probe's Compare Value

Each position or signal within a given probe corresponds to one of five possible match conditions (R,F,0,1,X) as specified in the Compare Value setting. The first time each signal matches its specified "compare value", the combination becomes "true". When all of the signals simultaneously matches all of the "compare values", the probe condition becomes true. If there is only one probe, then the trigger conditions are met and data is captured.

If there are multiple probes within the ILA/VIO, then the results of the probes can be ANDed or ORed together to cause a trigger. For example, if it does not matter which probe triggers in order to capture data, you would set the ILA trigger condition to "OR". If both probes must be simultaneously "true", then set the trigger condition to "AND", which will then cause the trigger.

Instructions

The following sections contain only the detailed instructions for the specified task.

In This Section

Adding IP from the IP Catalog to the Project.....	147
Opening the Vivado Analyzer Hardware Manager	149
Adding Probes (Signals) to the Waveform Viewer.....	153
Removing Probes (Signals) from the Waveform Viewer.....	153
Setting the Trigger Condition.....	154
Loading a Waveform Configuration	156
Arming the Vivado Analyzer Core	156
Triggering the ILA Immediately	157

Adding IP from the IP Catalog to the Project

1-1. Add the desired piece of IP to the project.

- 1-1-1. If necessary, expand **Project Manager** under the Flow Navigator to expose the IP Catalog entry.
- 1-1-2. Double-click the **IP Catalog** entry.

The IP Catalog tab opens in the Project Manager work area.

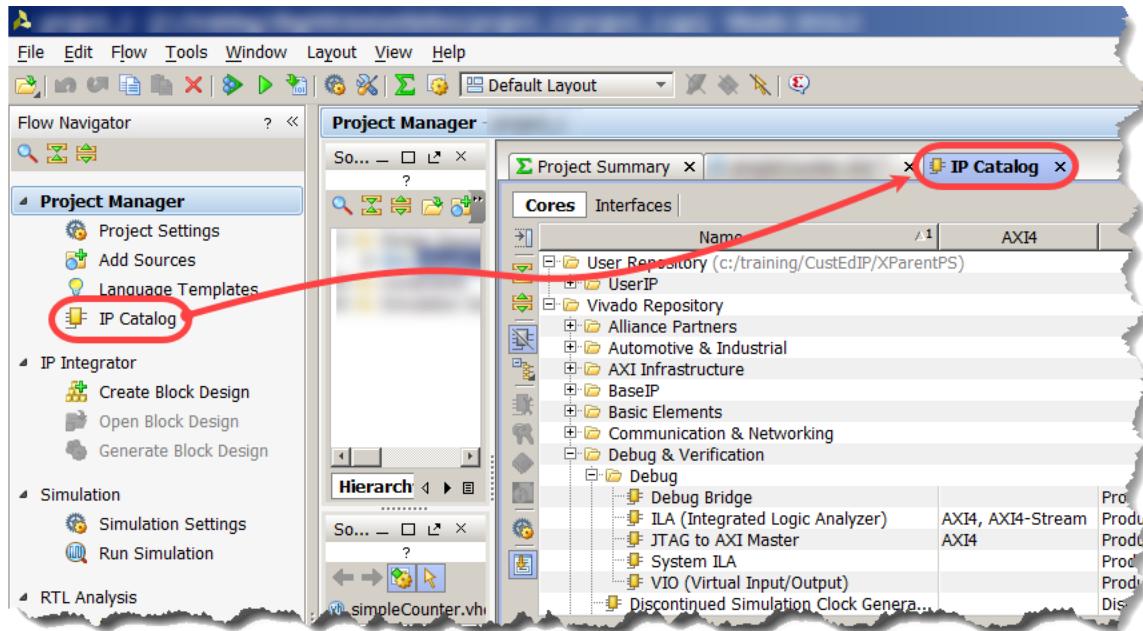


Figure 184: Opening the IP Catalog from the Project Manager

The IP catalog is organized into categories to help you quickly locate the IP you want.

- 1-1-3. Expand **the appropriate IP category** to expose the desired piece of IP.
- 1-1-4. Double-click **the desired piece of IP** to begin customization.

You can customize the IP here, or do so later.

- 1-1-5. Click **OK** to conclude the customization and advance to the output products generation.

Out-of-context IP generation will generate a netlist for the IP independent of other IP. This is the preferred way to generate outputs as any changes you make will only affect the IP that the changes were made to and accelerates the design process.

You can generate the netlist now, which takes a few moments, or defer generation until synthesis.

- 1-1-6. Click **Generate** to generate the output products for this core.

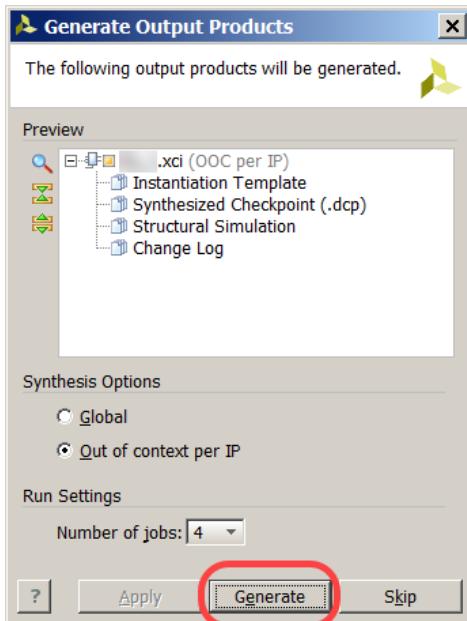


Figure 185: Generating the Out-of-Context IP

This will typically take a minute or so depending on the amount of code that must be synthesized.

- 1-1-7. When this process completes, click **OK** to close the summary dialog box.

Opening the Vivado Analyzer Hardware Manager

The Vivado logic analyzer requires a connection to the target hardware board. The actual connection is typically made using a USB-to-JTAG intelligent cable. Many of the Xilinx evaluation boards provide support for the intelligent cable as a module component on the board.

When you open a hardware session, a cable server program is launched that identifies the cable type and JTAG components on the board. A TCP/IP port is opened for a connection that may be to a logic analyzer or other application.

1-1. Open the hardware manager.

- 1-1-1. Expand **Program and Debug** from the Flow Navigator.
- 1-1-2. Double-click **Open Hardware Manager**.

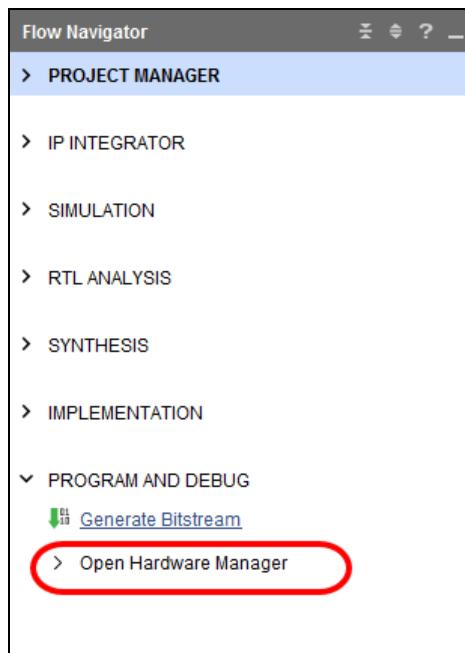


Figure 186: Opening the Hardware Manager from an Open Project

- 1-1-3.** Click **Open target** > **Open New Target** from the Hardware Session window to open a wizard that will facilitate making a connection to the USB platform JTAG download cable.

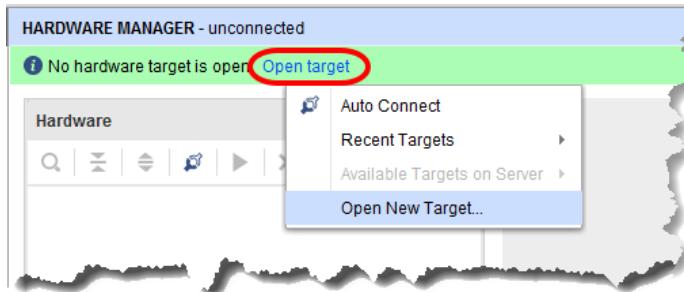


Figure 187: Selecting Open New Hardware Target

- 1-1-4.** Click **Next** to bypass the welcome dialog box.

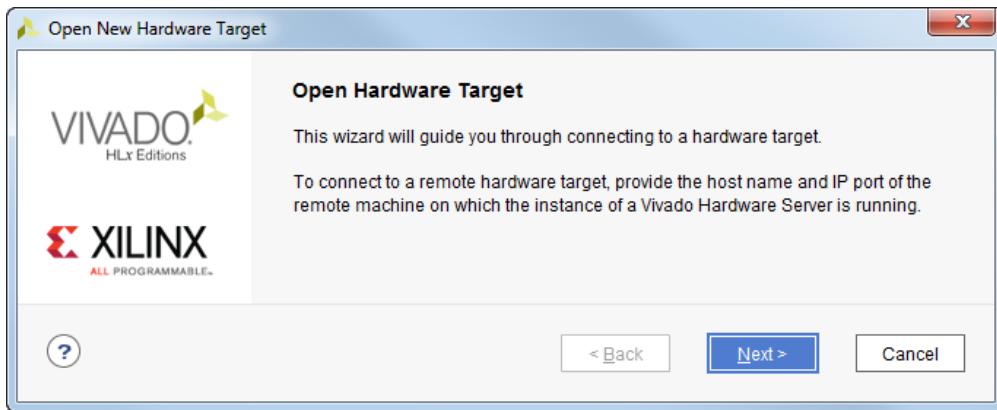


Figure 188: Opening a New Hardware Target

- 1-1-5.** Use the default local server name since your board is connected to the machine that you are running the Vivado Design Suite on.

If this is not the case, you will need to select the connection to the machine that is attached to the board.

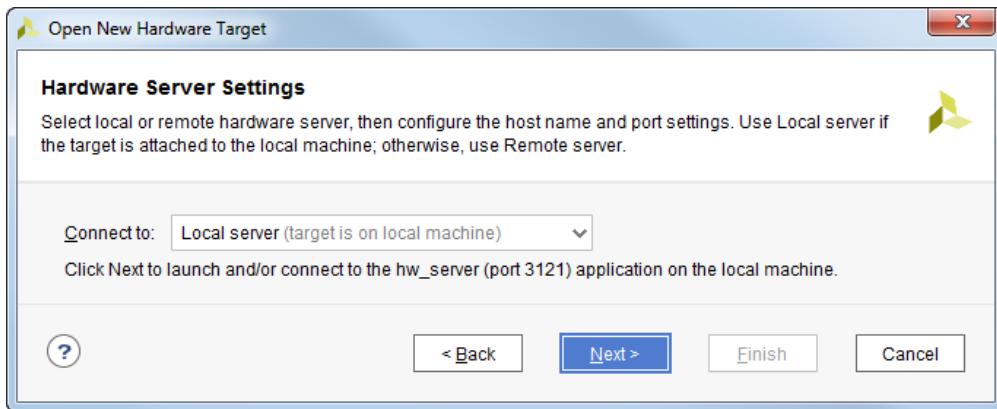


Figure 189: Using the Default Server Name

- 1-1-6. Click **Next** to scan the JTAG chain and view the nodes within the chain.

You should now see the xilinx_tcf hardware target and the hardware devices present in the JTAG chain. The JTAG clock speed is also shown.

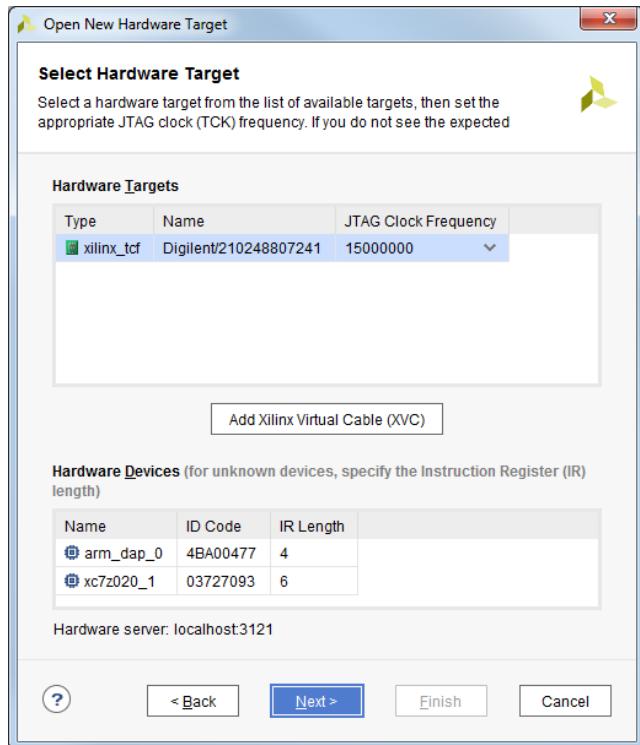


Figure 190: Viewing the Hardware Target and Devices

- 1-1-7. Click **Next** to advance to the summary.

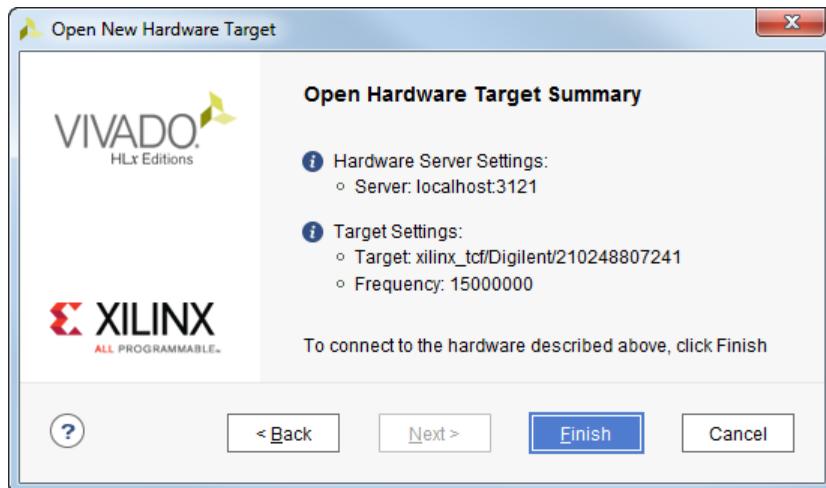


Figure 191: Summary Dialog Box

- 1-1-8. Click **Finish** to connect to the target.

Adding Probes (Signals) to the Waveform Viewer

When the hardware session opens, one or more probes (inputs into an ILA or VIO) are present.

1-1. Add a probe.

- 1-1-1. [Optional] Select one or more probes to add to the waveform in the Design Probes window.
- 1-1-2. Right-click the probe to add to the waveform from the Design Probes window.
 - o Select **Add Probes to Waveform** to add the selected probes to the waveform or
 - o Select **Add All Probes to Waveform** to add all the probes

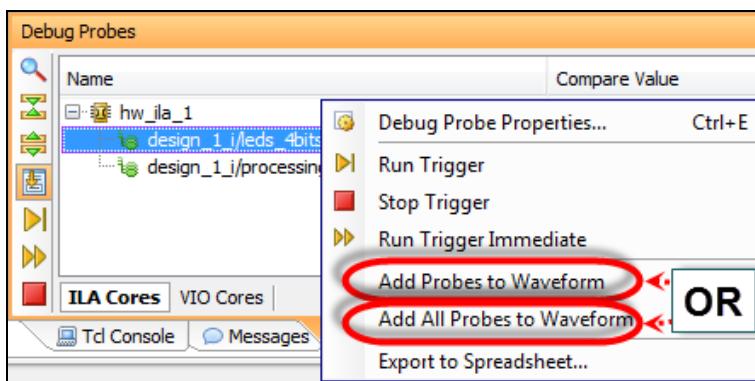


Figure 192: Adding Selected (or All) Probes to the Waveform

The probes will be added after any existing probes in the waveform.

Note that any given probe may be added more than once. This may be desirable as each probe entry in the waveform can have a different radix or color.

Removing Probes (Signals) from the Waveform Viewer

When the hardware session opens, one or more probes (inputs into an ILA or VIO) are present.

1-1. Remove one or more probes that you no longer want.

- 1-1-1. Select one or more probes from the Waveform window.
- 1-1-2. Press the **Del** key or right-click and select **Delete**.

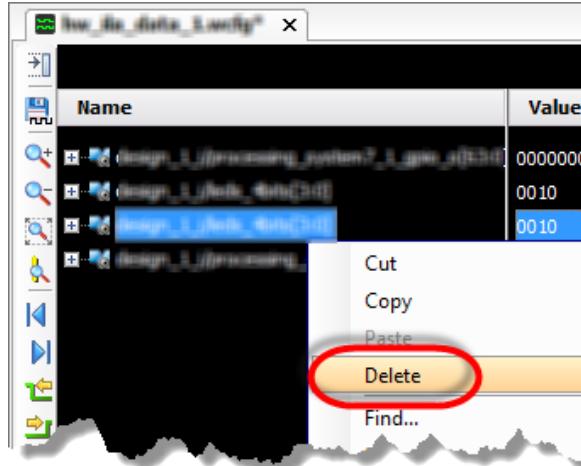


Figure 193: Deleting a Probe from the Waveform Viewer

Setting the Trigger Condition

1-1. Set the probe's trigger condition to the condition that you want to test for.

- 1-1-1. Select the **Debug Probes** tab in the console region of the workspace.

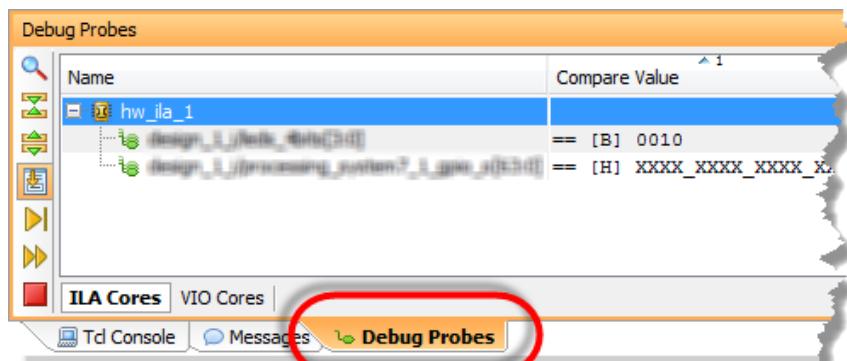


Figure 194: Locating the Debug Probes Tab

- 1-1-2. Click in the Compare Value column next to the probe that you want to configure.

A small dialog box opens.

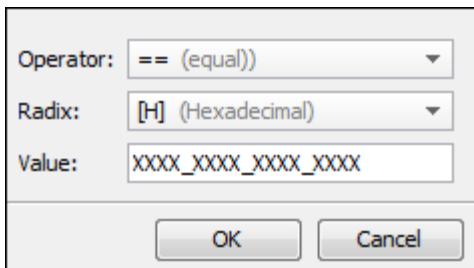


Figure 195: Compare Value Dialog Box

- 1-1-3. Set the operator to test against.

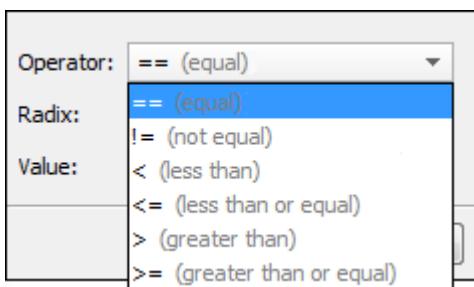


Figure 196: Setting the Comparison Operator

The == and != operators are typically used for testing individual signals within a probe. The other operators are generally used with testing numeric values using the entire probe as the compare value.

- 1-1-4. Set the radix to what is most conducive to the signal type.

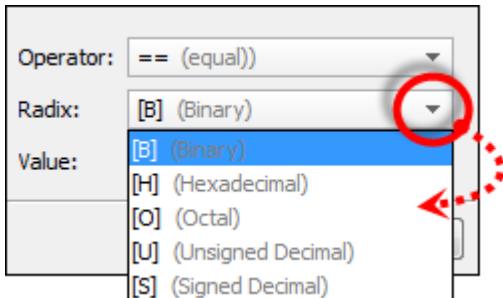


Figure 197: Selecting the Radix for the Compare Value

If you want to pick out individual signals, binary is a good choice. If you want identify data or addresses, then hexadecimal is a better choice. If you want to examine the results of a computation, then some form of decimal might be best.

- 1-1-5. Set the value (using the radix you just chose) to compare against.
- 1-1-6. Click **OK**.

Loading a Waveform Configuration

A waveform configuration is a file that contains information regarding how you want your waveform to appear: signals names, radices, colors, etc.

1-1. Load your waveform configuration file(s).

1-1-1. Select **File > Open Waveform Configuration**.

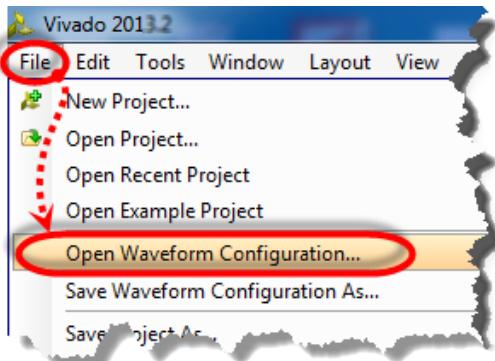


Figure 198: Opening a Waveform Configuration File (.wcfg)

A browser dialog box opens. Files must be entered one at a time.

- 1-1-2. Navigate to a waveform configuration file that you want to load.
- 1-1-3. Select the file.
- 1-1-4. Click **Open**.
- 1-1-5. Repeat for additional waveform files.

Arming the Vivado Analyzer Core

The Vivado analyzer cores are individually armed. Arming a core simply means that the core is made ready to collect incoming signals.

There are two basic modes of arming: Run Trigger and Run Trigger Immediate. Run Trigger Immediate ignores the trigger conditions and simply captures data from the time that you click Run Trigger Immediate until the core's memory fills. Run Trigger immediately captures data when the specified trigger condition is met.

Cores must be armed one at a time; therefore, it is good design methodology to use the Trigger In and Trigger Out ports of the ILA and arm the cores from the last core backwards to the first core. In this fashion, the sequence cannot be completed until the first ILA is armed. This prevents "earlier" ILAs from arming and having their trigger conditions met before you can arm the subsequent ILAs.

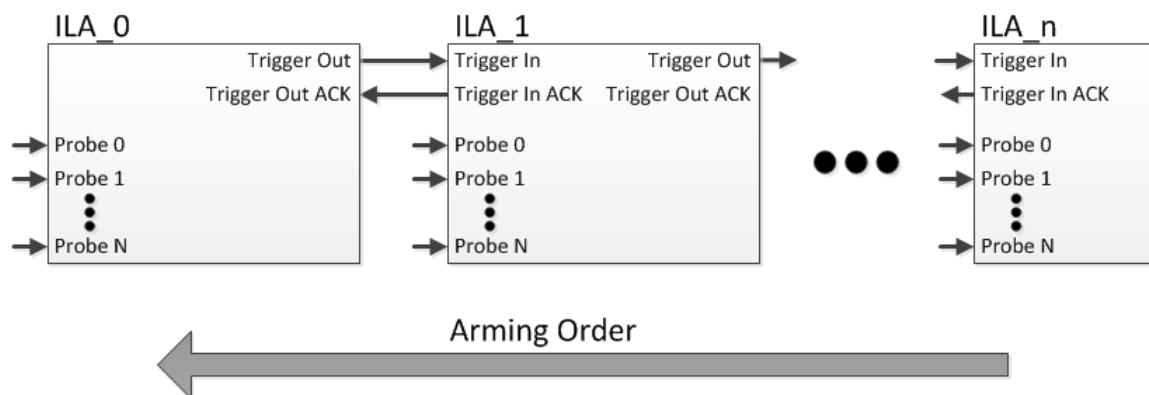


Figure 199: Connections and Arming Order for Multiple ILAs

1-1. Arm the ILA core.

- 1-1-1. Click the **Run Trigger** icon (▶).

Triggering the ILA Immediately

Sometimes it is beneficial to see the activity on an ILA or VIO without having to set or modify the trigger conditions.

If trigger conditions have not yet been set (all comparisons are set to 'X' - don't care), this is equivalent to Run Trigger Immediate.

1-1. Capture whatever data is currently presented to the specified analyzer core (trigger the ILA/VIO immediately).

- 1-1-1. Click the **Run Trigger Immediate** icon (»).

After a (typically) short delay, depending on the sample rate and size of the capture buffer, data will be uploaded from the device and displayed in the Waveform viewer.

Vivado HLS Operations [Last Updated Version 2019.1]

This section contains instructions for commonly performed tasks using the Vivado High-Level Synthesis (HLS) tool.

In This Section

Launching the Vivado HLS Tool.....	158
Launching the Vivado HLS Tool Using a Linux Terminal.....	160
Creating a Vivado HLS Project	160
Opening a Vivado HLS Project.....	165
Adding Synthesizable Source(s) to a Vivado HLS Project.....	165
Adding Simulation Source(s) to a Vivado HLS Project.....	167
Creating a New Source for an HLS Project.....	167
Simulating a Vivado HLS Design	168
Setting Synthesis Options.....	171
Synthesizing a Vivado HLS Design	172
Cosimulating a Vivado HLS Design.....	173
Exporting a Vivado HLS Design as IP.....	177
Analyzing an HLS Design	177

Launching the Vivado HLS Tool

There are a number of ways to launch the Vivado HLS tool. The two most popular mechanisms are shown here.

1-1. Launch the Vivado HLS tool.

- 1-1-1. [Windows 7 users:] Select **Start > All Programs > Xilinx Design Tools > Vivado 2019.1 > Vivado HLS 2019.1.**

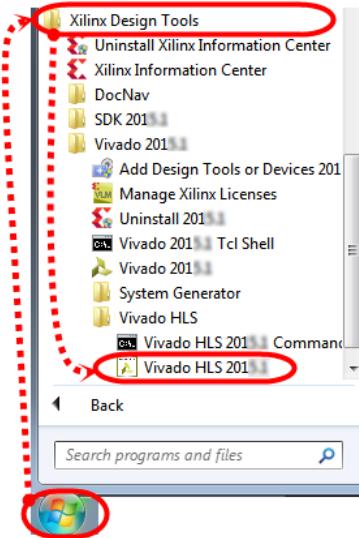


Figure 200: Launching the Vivado HLS Tool

- [Windows 10 users:] Select **Start > Xilinx Design Tools > Vivado HLS 2019.1.**

You can also double-click the **Vivado HLS** shortcut icon () from the Windows or Linux desktop or taskbar.

[Linux users]: If your system has not been configured with a shortcut on the desktop or taskbar, refer to the "Launching the Vivado HLS using a Linux Terminal" section in the *Lab Reference Guide*.

The Vivado HLS tool opens to the Welcome window. From the Welcome window you can create a new project, open examples, and access documentation and examples.



Figure 201: Vivado HLS Welcome Page

Launching the Vivado HLS Tool Using a Linux Terminal

- 1-1. [Linux users]: If your system has not been configured with a shortcut on the desktop or taskbar, then proceed with setting up and launching the Vivado HLS tool environment.**

1-1-1. Press <**Ctrl + Alt + T**> to open a new terminal window.

1-1-2. Change the directory to the topic cluster path:

```
[host] $ cd $<the topic cluster name>
```

1-1-3. Enter the following command to set up the Vivado Design Suite environment appropriately:

```
[host] $ source $XILINX_PATH/Vivado/2019.1/settings.sh
```

1-1-4. Type **vivado_hls** and press <**Enter**> to launch the Vivado Design Suite environment.

Creating a Vivado HLS Project

Here you will create a new Vivado HLS project from scratch.

1-1. Create a Vivado HLS project named *your HLS project name*.

- 1-1-1. Click **Create New Project** from the Welcome Page.



Figure 202: Creating a New Vivado HLS Tool Project

1-2. The Project Configuration dialog box asks for a project name and location.

- 1-2-1. Enter **your HLS project name** in the Project name field (1).
- 1-2-2. Enter **where your files are located** in the Location field (2).

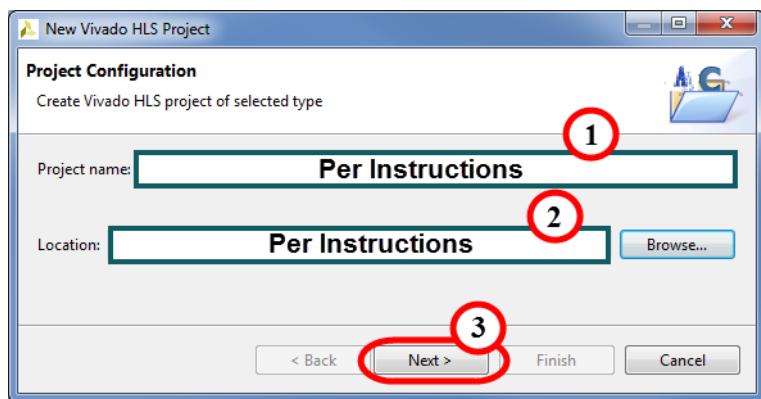


Figure 203: Configuring a New HLS Project

- 1-2-3. Click **Next** (3).

1-3. The Add/Remove Files dialog box opens. Here you will be invited to add existing files or create new sources.**1-3-1.** Click **Add Files**.**1-3-2.** Browse to **where your files are located**.**1-3-3.** Select **your HLS source files**.

The Vivado HLS tool automatically adds the working directory (project directory) and any directory that contains C files added to the project to the search path. Hence, header files that reside in these directories are automatically included in the project (no need to explicitly specify them). You must specify the path to all other header files (if any) by clicking the Edit CFLAGS button.

1-3-4. Click **OK** to add these files.

Note: If do not have existing files at this moment and you want to create new ones, click **New File**.

Note also that you can add compiler directives specific to each entry at this point.

1-3-5. Click **Browse** next to the Top Function field.

The Select Top function dialog box opens, which lists all the functions available from the specified source files.

1-3-6. Select **the name of the top function (the name of the top file)** from the list and click **OK**.

Note: You can also manually enter the name of the top function in the Top Function field.

In any C program, the top-level function is called main(). In the Vivado HLS design flow, you can specify any sub-function below main() as the top-level function for synthesis. You cannot synthesize the top-level function main().

The following are additional rules:

- Only one function is allowed as the top-level function for synthesis.
- Any sub-functions in the hierarchy under the top-level function for synthesis are also synthesized.

- o If you want to synthesize functions that are not in the hierarchy under the top-level function for synthesis, you must merge the functions into a single top-level function for synthesis.

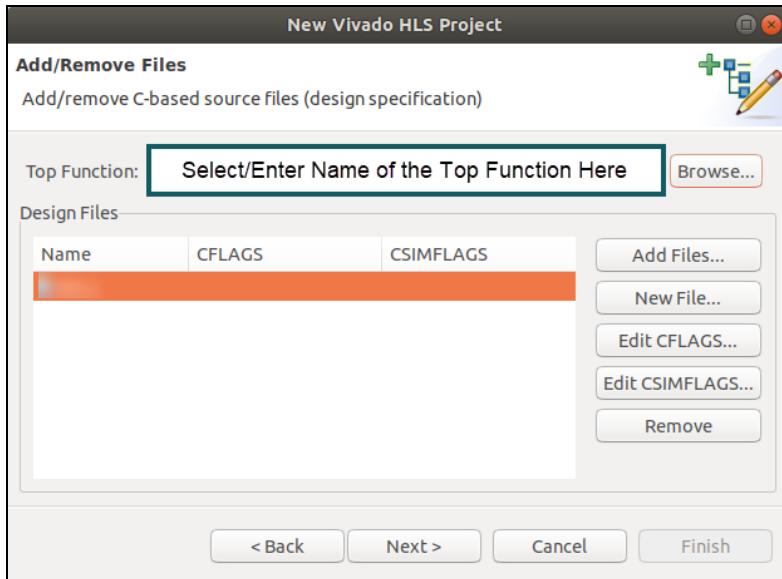


Figure 204: Adding Files to a New Vivado HLS Project

1-3-7. Click **Next**.

1-4. **Add any existing test bench files.**

If you have (or want) any test bench files, they can be entered here.
Sometimes the test bench is built into the synthesizable file.

1-4-1. Click **Add Files**.

1-4-2. Navigate to **where your files are located**.

1-4-3. Select **your HLS testbench files**.

1-4-4. Click **Open** to add these files.

1-4-5. Click **Next**.

1-5. Finally, it is time to specify some of the physical parameters of the design.

By default, your solution name is populated in the Solution Name field. No changes are required.

1-5-1. Set the clock period to **your clock period.**

You can leave the Uncertainty field blank.

1-5-2. Click the **Browse button to select a part or board.**

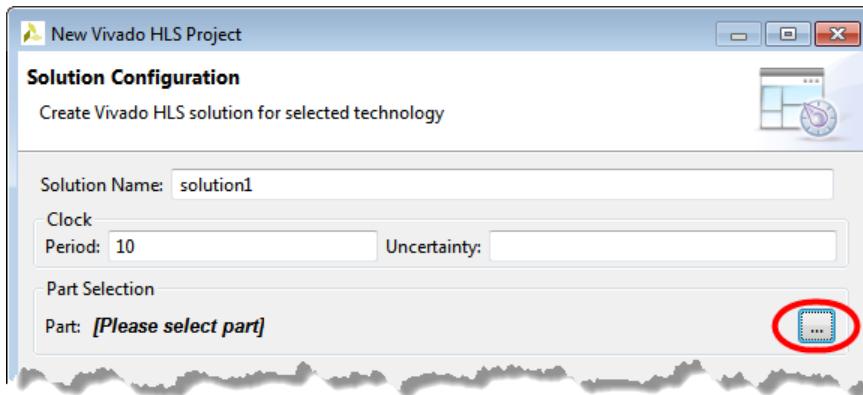


Figure 205: Locating the Board Browse Button

1-5-3. Click **Boards as shown below.**

1-5-4. Enter **the family name in the Search field.**



Figure 206: Filtering to Quickly Locate Target Platforms

1-5-5. Select **your board from the search list.**

1-5-6. Click **OK to select the board.**

1-5-7. Click **Finish.**

You will see the created project in the Explorer tab.

Opening a Vivado HLS Project

1-1. Open the existing Vivado Design Suite project *your HLS project name*.

- 1-1-1. Click **Open Project** from the Quick Start section (1).

The Open Project dialog box opens (2).

- 1-1-2. Click **[Browse ...]** (3).

- 1-1-3. Click **OK** (4).

The Browse For Folder dialog box opens (5).

- 1-1-4. Browse to the *the location at which the project should be placed/your HLS project name* directory (6).

Note: The drop-down arrow shows the directory hierarchy.

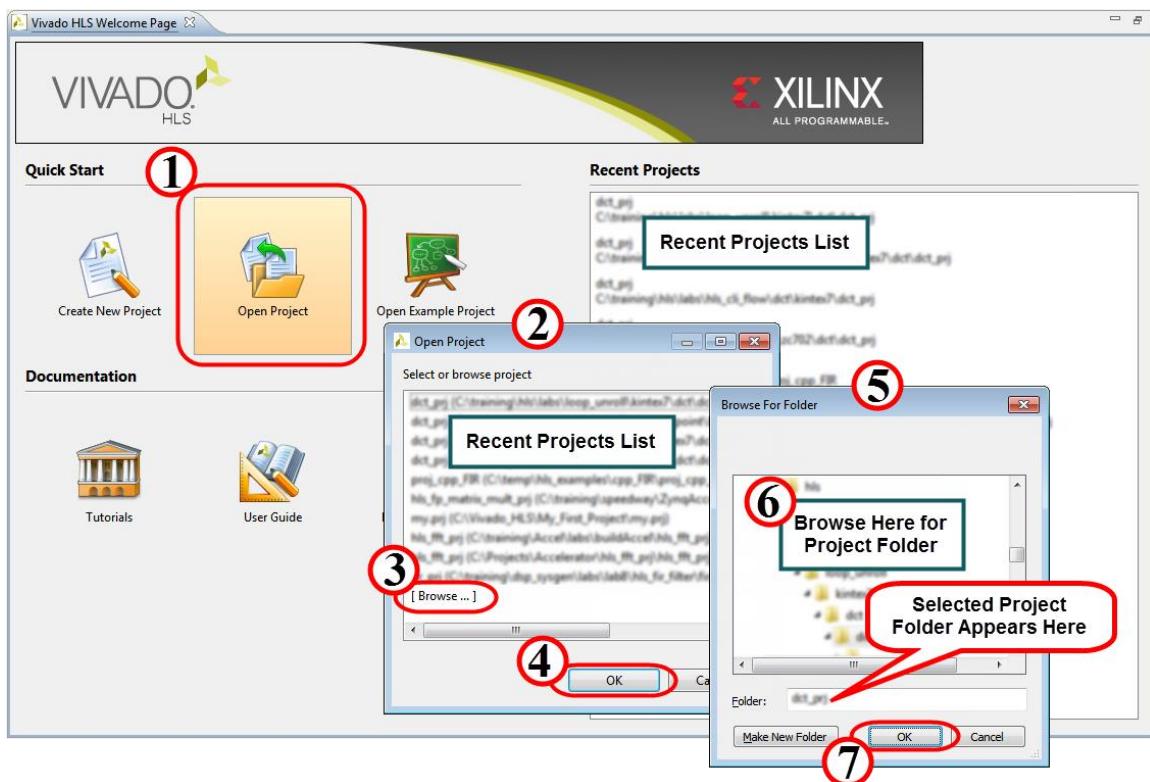


Figure 207: Opening a Vivado HLS Project

- 1-1-5. Click **OK** to open the selected project and close the dialog box (7).

The HLS project now opens in the Vivado HLS.

Adding Synthesizable Source(s) to a Vivado HLS Project

If you missed the opportunity to add files during the project creation phase, you can still add existing files or create new files for your HLS project.

1-1. Add your synthesizable source(s) to the Vivado HLS project.

- 1-1-1. Using the Explorer, expand the tree until the **Source** branch is seen.
- 1-1-2. Right-click **Source** and select **Add Files**.



Figure 208: Add Existing Files or Create New Files for the HLS Project

-- OR --

Select **Project > Add Source**.

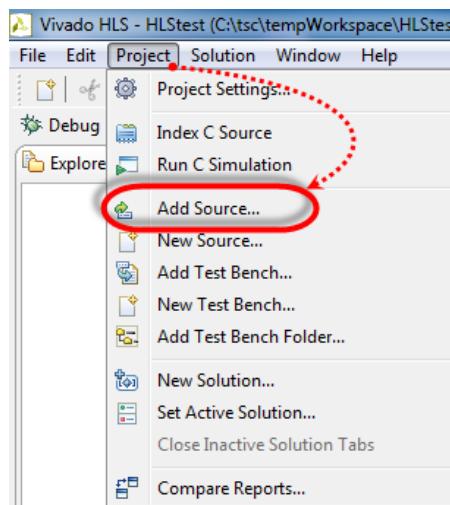


Figure 209: Adding a New Source from the Project Menu

- 1-1-3. Navigate to the directory where your files are located.
- 1-1-4. Select the file or files you want to import.
- 1-1-5. Click **Open**.

Adding Simulation Source(s) to a Vivado HLS Project

If you missed the opportunity to add simulation files during the project creation phase, you can still add existing files or create new simulation files for your HLS project.

1-1. Add your simulation sources to your Vivado HLS project.

- 1-1-1. Expand your project if necessary.
- 1-1-2. Right-click **Test Bench** and select **Add Files**.

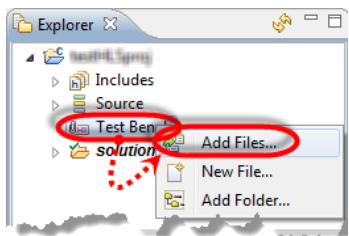


Figure 210: Adding Files to a Vivado HLS Testbench

A browser window opens.

- 1-1-3. Browse to your simulation sources.
- 1-1-4. Click **Open**.

Creating a New Source for an HLS Project

1-1. Create a new source for to the current project.

- 1-1-1. Using the Explorer, expand the tree until the **Source** branch is seen.
- 1-1-2. Right-click **Source** and select **New File**.

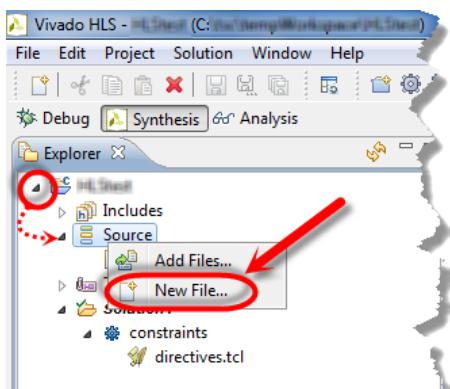


Figure 211: Creating a New HLS Source File

-- OR --

Select **Project > New Source**.

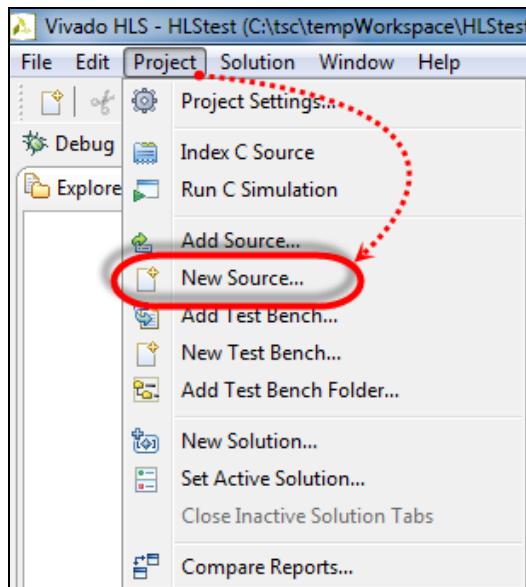


Figure 212: Creating a New Source from the Project Menu

1-1-3. Navigate to *the directory where your files are located*.

1-1-4. Enter the name for your new source

1-1-5. Click **Open**.

The text editor opens and you can begin entering your new code.

Simulating a Vivado HLS Design

1-1. Simulate the Vivado HLS tool design.

- 1-1-1. Select **Project > Run C Simulation** or click the **Run C Simulation** icon ().

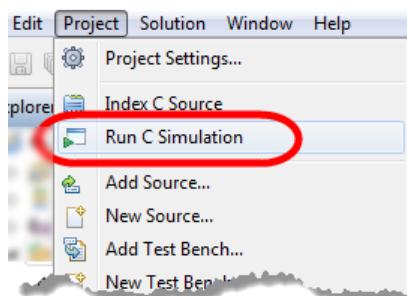


Figure 213: Launching the C Simulation

The Run C Simulation dialog box opens.

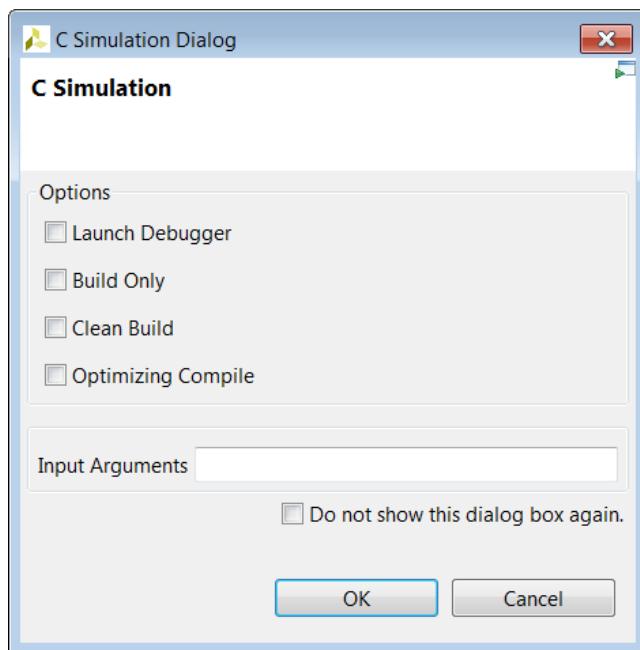


Figure 214: C Simulation Dialog Box

Each of the options controls how simulation is run:

- **Launch Debugger:** After compilation the debug perspective automatically opens for you to step through the code.
- **Build Only:** Compiles the C code, but the simulation does not run.
- **Clean Build:** Removes any existing executable and object files before compiling the code.

- Optimizing Compile: Uses the gcc/g++ -O option (no debug info and this is mutually exclusive with debug options; this may run faster, but the difference is not substantial).

1-1-2. Select **the options that you want.**

1-1-3. Click **OK.**

The simulation log will be displayed in the editor pane.

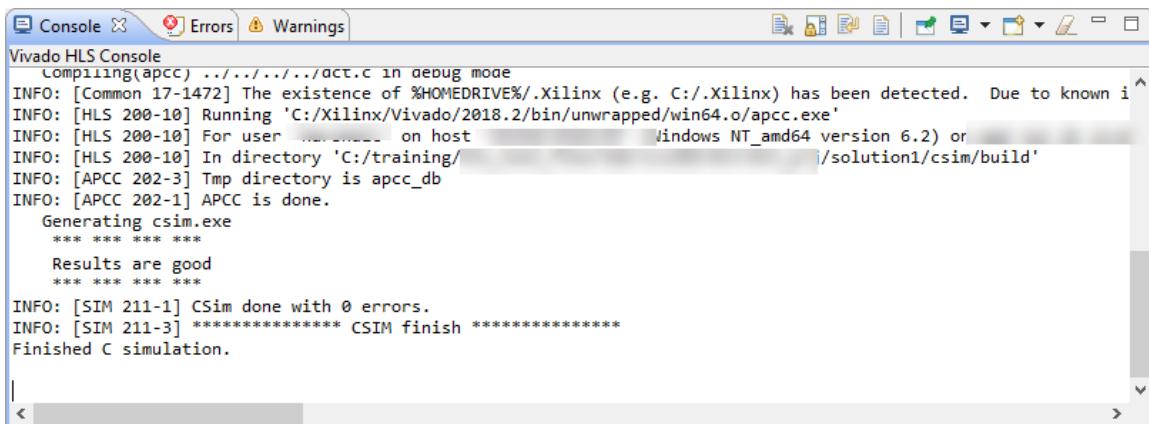
1-2. View the simulation report.

The information generated by the Vivado HLS tool can be found in two places, both described here.

The first is the Console window, which reports not only the output produced by the code being simulated, but all of the simulation engine messages as well. The simulation log provides only a few simulation engine messages and the simulated code output.

1-2-1. Select the **Console** tab in the lower portion of the tool's GUI.

You may need to scroll to view all the output produced by the simulation.



The screenshot shows the Vivado HLS Console window. The title bar says "Console". Below it, tabs for "Errors" and "Warnings" are visible. The main area displays the following text:

```
Vivado HLS Console
Compiling(apcc) ../../../../../../act.c in debug mode
INFO: [Common 17-1472] The existence of %HOMEDRIVE%/.Xilinx (e.g. C:/Xilinx) has been detected. Due to known i
INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2018.2/bin/unwrapped/win64.o/apcc.exe'
INFO: [HLS 200-10] For user _____ on host _____ Windows NT_amd64 version 6.2) or
INFO: [HLS 200-10] In directory 'C:/training/
INFO: [APCC 202-3] Tmp directory is apcc_db
INFO: [APCC 202-1] APCC is done.
Generating csim.exe
*** *** ***
Results are good
*** *** ***
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] ***** CSIM finish *****
Finished C simulation.
```

Figure 215: Example Output After Simulation

The other location, described below, provides only a few simulation engine messages and the simulated code output. Typically this is opened after the simulation completes; however, if you need to access it after closing the log pane, here's how to access the simulation report.

1-2-2. Expand **your HLS project name > solution1 > csim > report** in the Explorer pane.

- 1-2-3.** Double-click the log file name to open it in the editor pane.

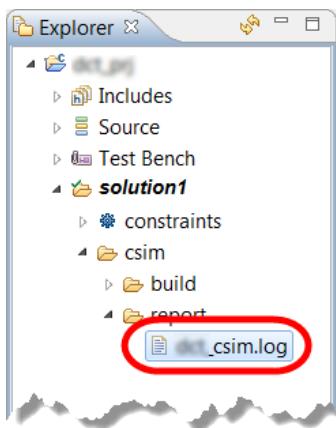


Figure 216: Locating the Simulation Log File

Setting Synthesis Options

Various information relevant to how the tools will synthesize a design can be entered into the Vivado HLS tool.

- 1-1. Set the synthesis options to the clock period that your design needs to meet and your clock uncertainty, or you can leave this field blank.**
- 1-1-1.** Select **Solution > Solution Settings** or click the **Open Synthesis Options** icon (⚙).

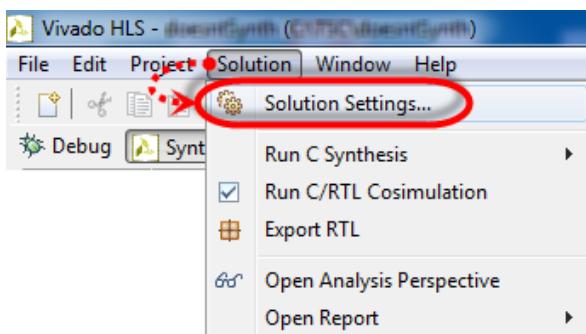


Figure 217: Accessing Synthesis Settings from the Menu Bar

The Solutions Settings dialog box opens.

- 1-1-2. Click the **Synthesis** option in the navigation panel.

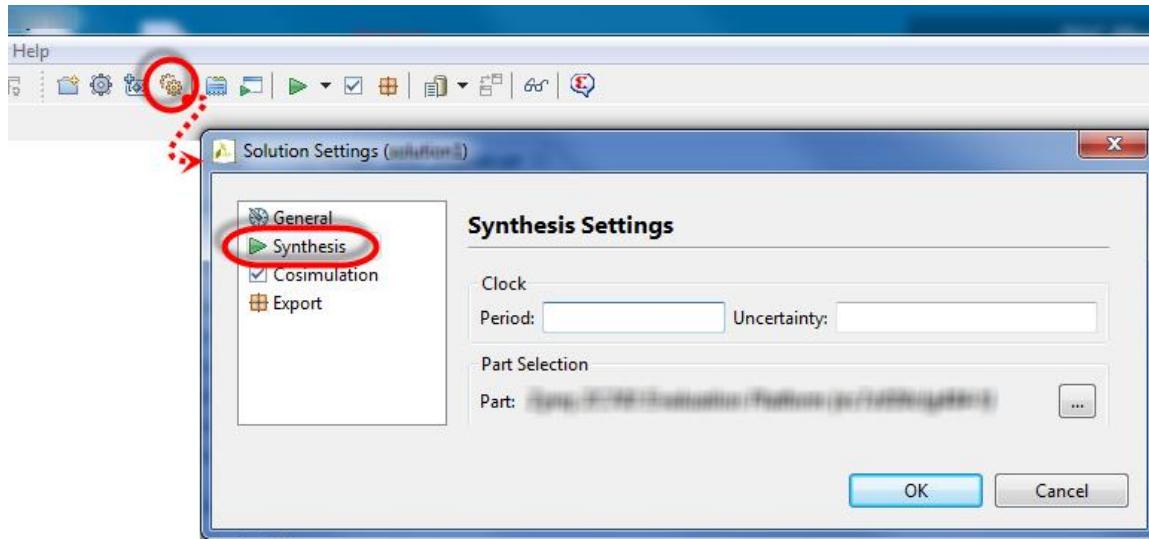


Figure 218: Accessing the Synthesis Options in the Solutions Setting Dialog Box

- 1-1-3. Set the period to the clock period that your design needs to meet.
1-1-4. Set the uncertainty to your clock uncertainty, or you can leave this field blank.
1-1-5. Click **OK**.

Synthesizing a Vivado HLS Design

1-1. Synthesize the design.

- 1-1-1. Select **Solution > Run C Synthesis > Active Solution** or click the **Run C Synthesis** icon in the menu bar.



Figure 219: Launching Synthesis

This option synthesizes the currently selected solution.

All solutions (or selected solutions) can be synthesized by using the drop-down menu next to the synthesis icon. You can synthesize all solutions or synthesize selected solutions in addition to the default.

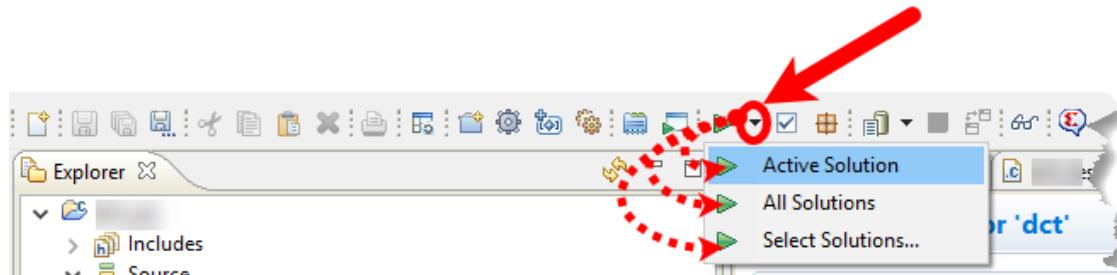


Figure 220: Options for What to Synthesize

Cosimulating a Vivado HLS Design

1-1. Cosimulate the Vivado HLS tool design.

- 1-1-1. Select **Solution > Run C/RTL Cosimulation** or click the **Run C/RTL Cosimulation** icon ().

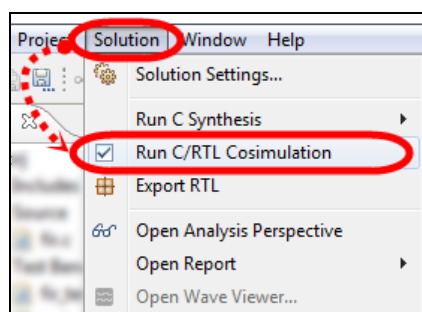


Figure 221: Launching from the Menu

The Run C/RTL Co-simulation dialog box opens.

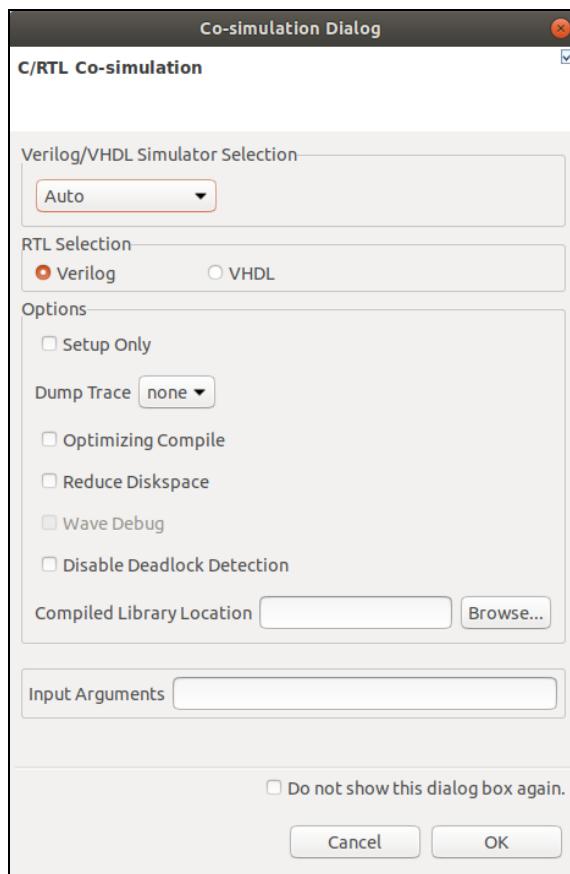


Figure 222: Co-simulation Dialog Box

Each of the options controls how C/RTL co-simulation is run:

- RTL Selection: Select the RTL that is simulated (Verilog/VHDL).
- Setup Only: This creates all the files (wrappers, adapters and scripts) required to run the simulation but does not execute the simulator.
- Dump Trace: During RTL verification, the trace files can be saved and viewed using an appropriate viewer. By selecting this option, the trace file will be saved to the `<solution>/sim/<RTL>` folder.
- Optimizing Compile: This ensures a high level of optimization is used to compile the C test bench. Using this option increases the compile time but the simulation executes faster.
- Reduce Diskspace: Saves the results for all transactions before executing RTL simulation. In some cases, this can result in large data files. This option can be used to execute one transaction at a time and reduce the amount of disk space required for the file. If the function is executed N times in the C test bench, the `reduce_diskspace` option ensures N separate RTL simulations are performed. This causes the simulation to run slower.

- Compiled Library Location: This specifies the location of the compiled library for a third-party RTL simulator.
- Input Arguments: This allows the specification of any arguments required by the test bench.

1-1-2. Select **the options that you want.**

1-1-3. Click **OK.**

The simulation log will be displayed in the editor pane.

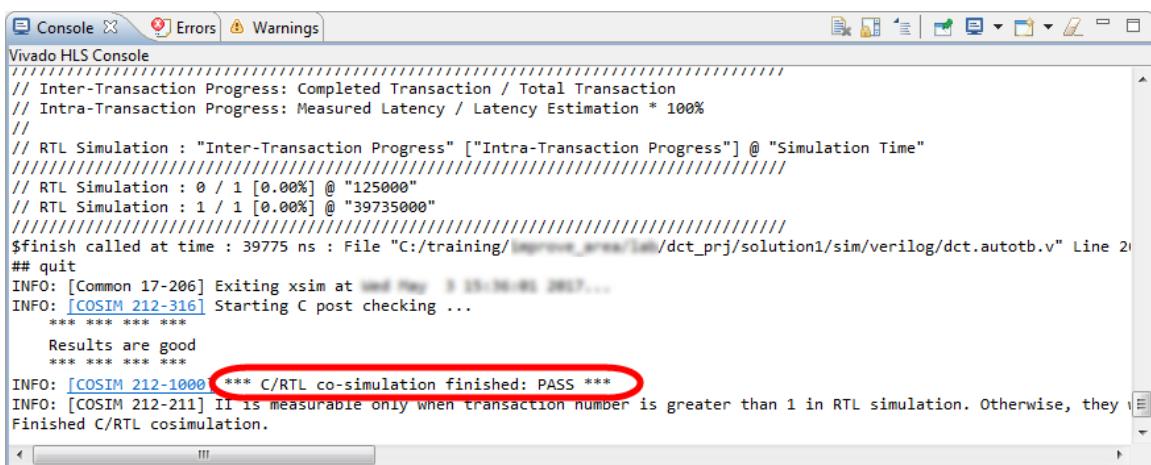
1-2. View the Cosimulation report.

The information generated by the Vivado HLS tool can be found in two places, both described here.

The first is the Console window, which reports not only the output produced by the code being simulated, but all of the simulation engine messages as well. The simulation log provides only a few simulation engine messages and the simulated code output.

1-2-1. Select the **Console tab in the lower portion of the tool's GUI.**

You may need to scroll to view all the output produced by the cosimulation.



```

Vivado HLS Console
=====
// Inter-Transaction Progress: Completed Transaction / Total Transaction
// Intra-Transaction Progress: Measured Latency / Latency Estimation * 100%
//
// RTL Simulation : "Inter-Transaction Progress" ["Intra-Transaction Progress"] @ "Simulation Time"
// RTL Simulation : 0 / 1 [0.00%] @ "125000"
// RTL Simulation : 1 / 1 [0.00%] @ "39735000"
$finish called at time : 39775 ns : File "C:/training/.../dct_prj/solution1/sim/verilog/dct.autob.v" Line 20
## quit
INFO: [Common 17-206] Exiting xsim at 10:45:38 AM 2017...
INFO: [COSIM 212-316] Starting C post checking ...
*** **** ***
Results are good
*** **** ***
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
INFO: [COSIM 212-211] It is measurable only when transaction number is greater than 1 in RTL simulation. Otherwise, they are not.
Finished C/RTL cosimulation.

```

Figure 223: Example Output After a C/RTL Co-simulation

The other location, described below, provides only a few simulation engine messages and the simulated code output. Typically this is opened after the simulation completes; however, if you need to access it after closing the log pane, here is how to access the simulation report.

1-2-2. Expand **your HLS project name > solution1 > sim > report in the Explorer pane.**

- 1-2-3. Double-click the log file name to open it in the editor pane.

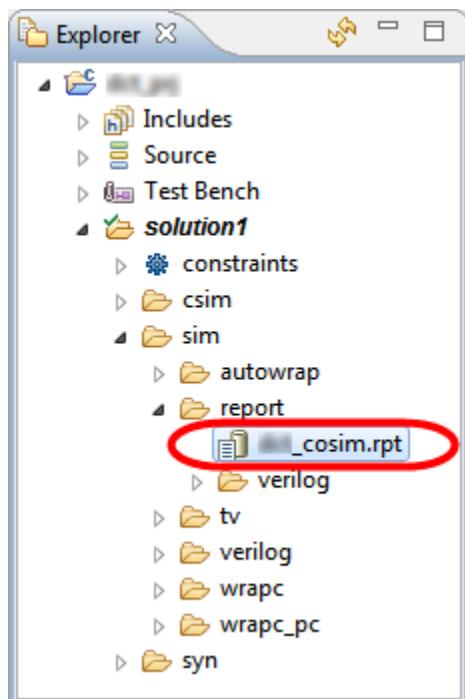
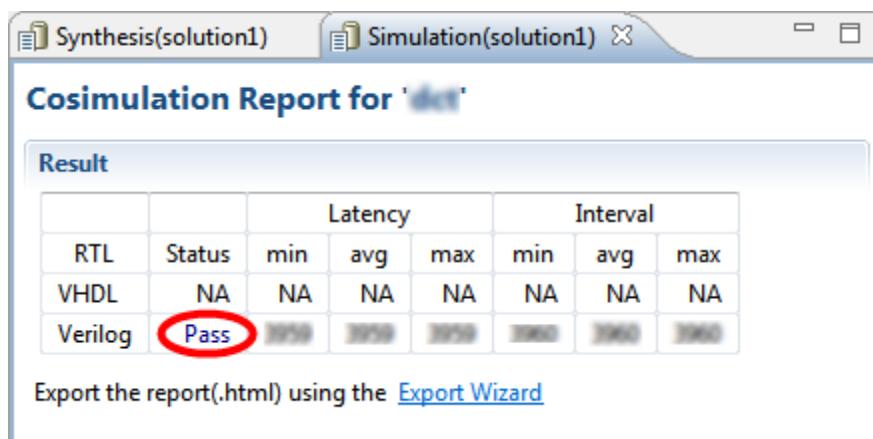


Figure 224: Locating the Co-simulation Log File

The Cosimulation Report in HTML format will be displayed in the main viewing area.



RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	
Verilog	Pass	3999	3999	3999	3999	3999	

Figure 225: Cosimulation Report – HTML

You can quickly verify the cosimulation status here.

Exporting a Vivado HLS Design as IP

1-1. Export the Vivado HLS design as a piece of IP.

1-1-1. Select **Solution > Export RTL**.

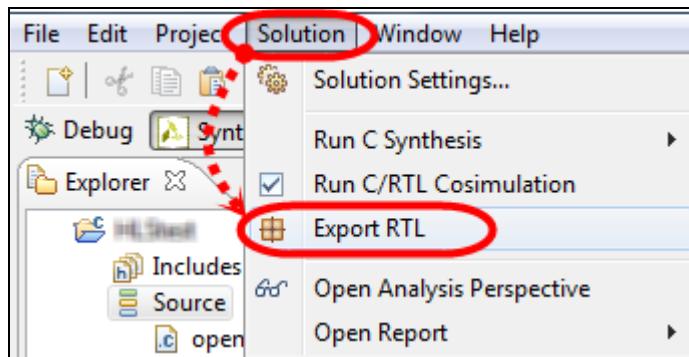


Figure 226: Exporting the RTL from Vivado HLS using the Menu Bar

The Export RTL dialog box allows you to select the style of the exported files.

1-1-2. Select **IP Catalog** for compatibility with the Vivado Design Suite.

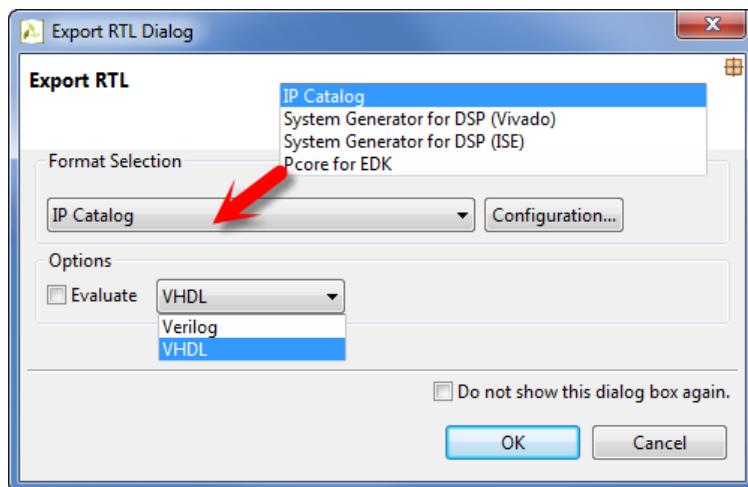


Figure 227: Selecting the Export Format for the IP

1-1-3. Click **OK**.

Analyzing an HLS Design

The Vivado HLS design tool provides various perspectives into the behavior of the design including data flow diagrams and textual reports.

1-1. Perform the analysis.

- 1-1-1. Click the **Analyze** button or the **Analyze** icon () to initiate the analysis for the active solution.

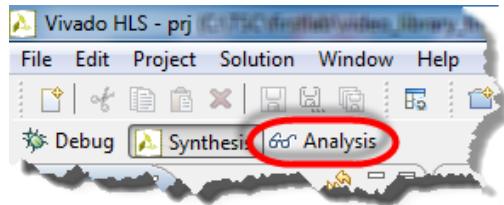


Figure 228: Locating the Analysis Button

The data-flow diagram opens.

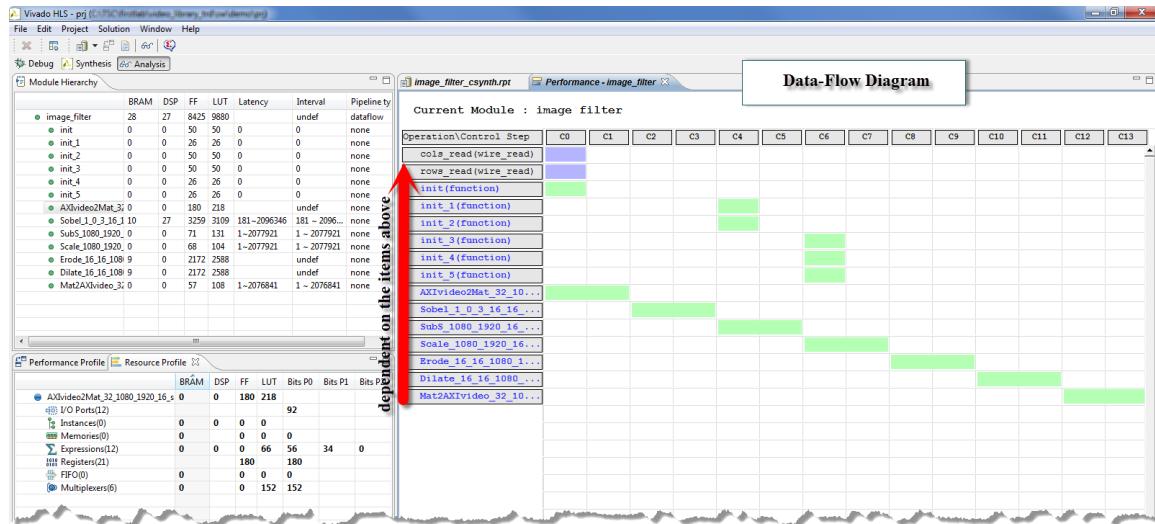


Figure 229: Example of Analysis View

For a full description of the analysis capability, refer to the *HLS User Guide*.

1-2. Determine the approximate performance for the overall design.

- 1-2-1. Expand the **active solution > syn > report**.
- 1-2-2. Double-click the file representing the top of the hierarchy.

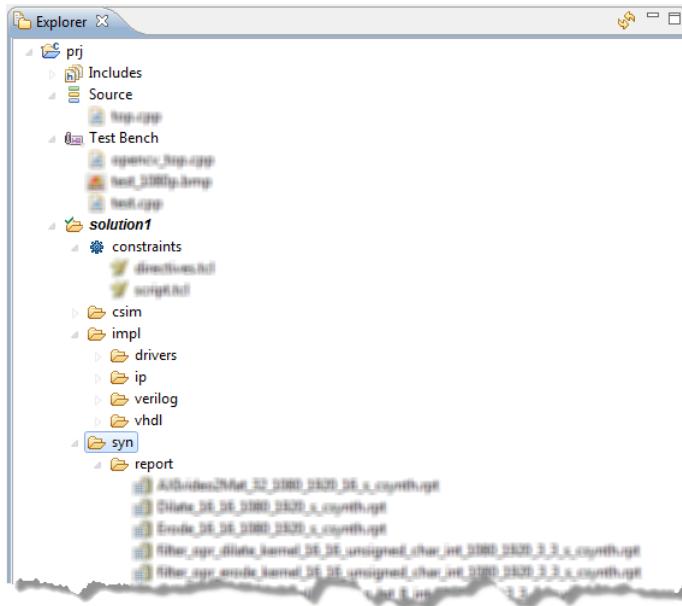


Figure 230: Accessing the Synthesis Reports

- 1-2-3. Locate the performance approximations in the report.

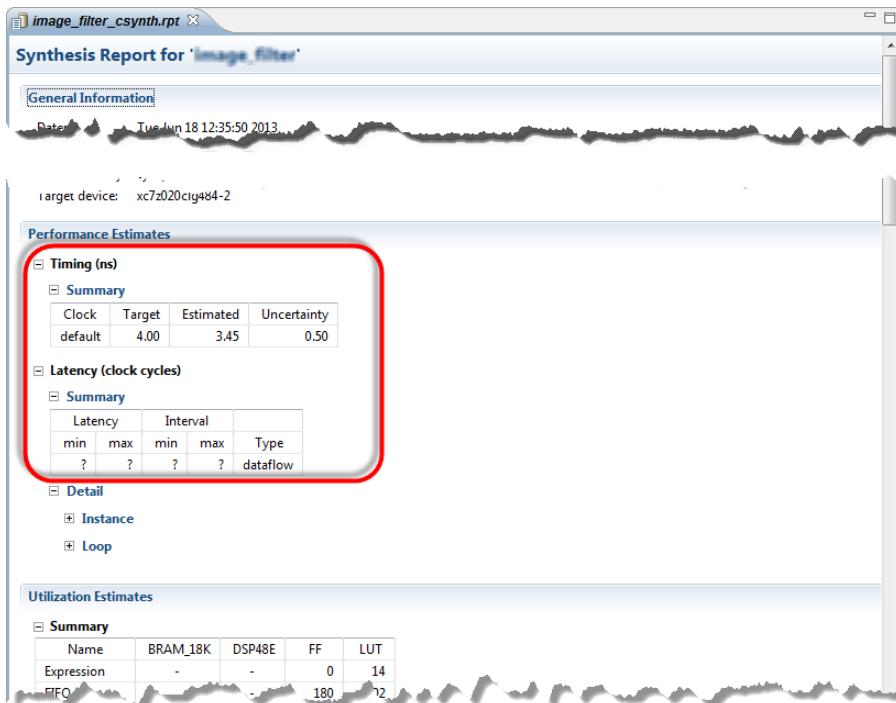


Figure 231: Locating the Performance Approximations in the Synthesis Report

System Generator Tool Operations [Last Updated Version 2018.3]

This section contains instructions for commonly performed tasks using the System Generator tool.

In This Section

Launching the Xilinx System Generator Tool	180
Simulating a Model in the Simulink Software	180

Launching the Xilinx System Generator Tool

1-1. Launch Xilinx System Generator.

- 1-1-1. **[Windows 7 users]:** Select **Start > All Programs > Xilinx Design Tools > Vivado 2019.1 > System Generator > System Generator 2019.1.**

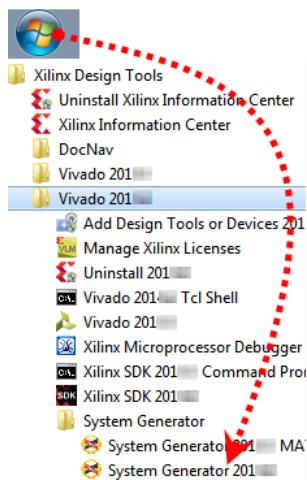


Figure 232: Launching Xilinx System Generator from the Start Menu

[Windows 10 users]: Select **Start > Xilinx Design Tools > System Generator 2019.1.**

[Linux users]: Press **<Ctrl + Alt + T>** to open a new terminal window and enter **sysgen** to launch DSP System Generator.

Simulating a Model in the Simulink Software

1-1. Simulate the model.

- 1-1-1. Select **Simulation > Run** in the Simulink Editor to run the simulation.

Alternatively, you can click the **Run** button  in the toolbar.

SDK Tool Operations [Last Updated Version 2019.1]

Also included in this section are any third-party solutions commonly used with SDK, such as terminal emulators, etc.

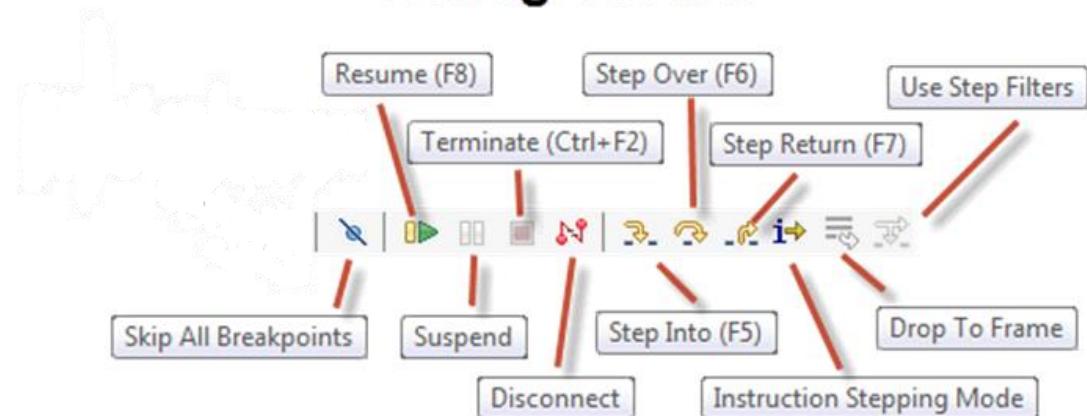
Informative Information

The following information is for background understanding of the materials discussed in the Instructions section.

GDB Debug Controls and Windows

The Software Debugger toolbar icons provide easy access to the various debugger features. Take a moment to familiarize yourself with the toolbar location, icons, and their functions. Note that some of the more popular icons have function keys associated with them.

Debug Toolbar



Instructions

The following sections contain only the detailed instructions for the specified task.

In This Section

Launching SDK via the SDK Launcher Tool.....	183
Launching the SDK or SDSoc Tool and Setting the Workspace.....	184
Launching the Xilinx Software Command Line Tool	185
Running a Tcl Script within the Xilinx Software Command Line Tool.....	186
Importing an Example Application from a Library.....	186
Creating a Hardware Platform Specification	187
Creating a Hardware Platform Specification and BSP.....	190
Updating a Hardware Platform Specification	190
Creating a Basic Board Support Package	192
Enabling C Libraries	194
Creating a C/C++ Application Project with a New BSP.....	195
Creating a C/C++ Application Project with an Existing BSP.....	198
Creating a C/C++ Application Project from the SDK Welcome Window	201
Importing Sources to an Application	203
Launching the SDK Tool and Setting the Workspace in a Linux Environment	205
Adding a Library to a BSP.....	206
Importing an Existing Project into the SDK or SDSoc Tool	206
Enabling and Disabling Automatic Compilations	209
Setting Compiler Options.....	209
Setting Compiler Options [No Macro Settings].....	212
Adding Symbols to Application Project Settings	213
Adding Symbols to Application Project Settings	215
Opening a Source File in the Editor	215
Finding Text in a Source File	216
Enabling Line Numbering in the SDK Text Editor	218
Saving and Compiling an Application.....	218
Compiling All Projects.....	219
Customizing the Linker Script.....	219
Programming the Device from SDK	220
Creating a Boot Image File	221
Configuring a Local Repository.....	224
Setting Up a Run Configuration	225
Setting Up a Run Configuration (System Debugger)	227
Running with a Default Configuration	229
Opening the XSCT Console	230
Debugging	232
Running the Application on Hardware	269
Linux and Remote System Explorer.....	269
Launching a Software Application on Hardware	287
Switching Perspectives	288
Configuring the Terminal	288
Configuring the SDK Terminal.....	290
Terminating a Running Application	292
Closing the Xilinx SDK Tool	293

Launching SDK via the SDK Launcher Tool

Several common and repetitive tasks have been automated with a custom tool called *SDK Launcher*. This tool will automatically create an SDK workspace, create a hardware platform specification targeting the desired development board and processor combination and, finally, launch the SDK tool in the open workspace.

The hardware platform specification will be named *your hardware platform description name*. If you are unsure how to do any of these tasks manually, reference the relevant sections in the *Lab Reference Guide*.

1-1. Open the SDK Launcher tool.

- 1-1-1. **[Windows users]:** Open File Explorer and browse to the `C:\training\tools` directory.

[Linux users]: Open File Explorer and browse to the
`/home/xilinx/training/tools` directory.

- 1-1-2. Double-click the **SDKLauncher.jar** file to launch the application.

[Linux users]: If double-clicking does not work, then enter `java -jar SDKLauncher.jar` in a terminal.

1-2. Configure the SDK Launcher.

- 1-2-1. Click the **Browse** icon to open a file browser window (1).

- 1-2-2. **[Windows users]:** Select `C:\training\<the topic cluster name>` to identify which lab to target.

[Linux users]: Select `/home/xilinx/training/<the topic cluster name>` to identify which lab to target.

- 1-2-3. Click **Open** to select the path and return to the tool's main GUI.

- 1-2-4. Click the Down arrow next to the platform list to expand the list (2).

- 1-2-5. Select the board and target processor that you want to use.

Note: Selecting PS will target the ARM® processor and MicroBlaze will target the MicroBlaze™ processor even in the presence of a PS.

The SDK version should be populated with the latest version in the SDK Version field. This can be changed—however, you do so at your own risk.

This instructs the SDK Launcher application to use a particular version of the SDK tool.

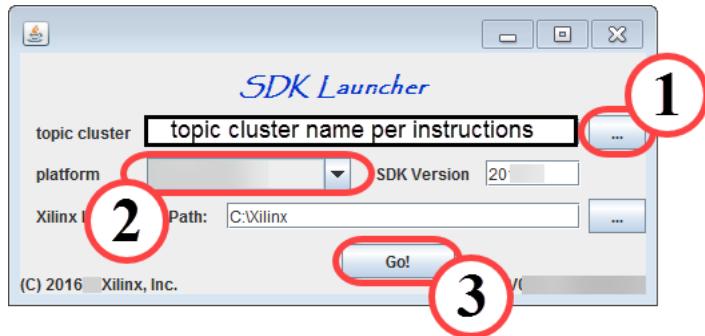


Figure 233: Configuring the SDK Launcher Tool

1-2-6. Click **Go** (3).

This automatically sets up the workspace, creates the hardware platform specification, and launches SDK.

1-2-7. Close the Welcome screen once SDK launches, if it is open.

1-2-8. Close the SDK Launcher by clicking the 'X' in the upper-right corner of the dialog box.

Launching the SDK or SDSoC Tool and Setting the Workspace

1-1. Launch the Xilinx SDK tool and set the workspace.

1-1-1. Click the SDK icon on the taskbar to launch the SDK tool.

The Workspace Launcher opens after a moment.

The SDK tool creates a workspace environment that initially only contains a thin structure that tracks tool settings and maintains the tool log file. As projects are added, this workspace will update to include hardware projects, BSPs, and your software applications. Workspaces can be switched from within the tool by selecting **File > Switch Workspace**.

If it becomes necessary to move a software application to another location or computer, use the import and export features. Manually copying files is not recommended as workspace files are set to use absolute path names and this will cause the tool to become unstable.

1-1-2. Enter the **\$<the topic cluster name>/lab** into the Workspace field or use the Browse button to browse to the location when the Workspace Launcher opens.

Note: The Eclipse-based tools do not support the expansion of environment variables and you will need to make this expansion yourself. You will need to replace **\$<the topic cluster name>** with the path to the training directory followed by **<the topic cluster name>/lab**. If you do not know what the path to the training directory is, open a Linux terminal (**Ctrl + Alt + T**) and enter **echo \$TRAINING_PATH**. The default path is

/home/xilinx/training; however, your machine may have been configured differently.

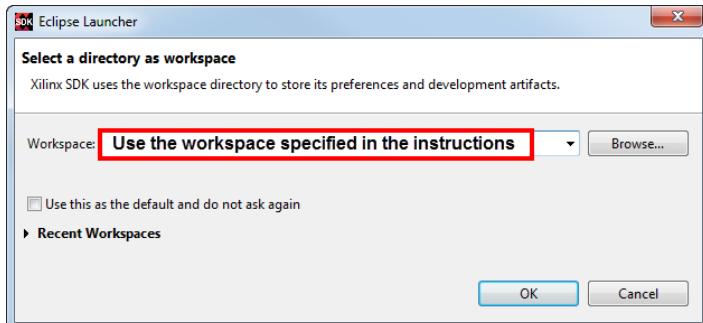


Figure 234: Setting Up the Workspace Environment Path

- 1-1-3. Click **OK** to close the Workspace Launcher dialog box and open the new workspace.

When the SDK tool is launched on its own, you must manually identify where you want the workspace and create or import the necessary hardware description to begin developing an application.

- 1-1-4. Close the **Welcome** tab if it appears.

This will give you more room to view your project. You may also want to maximize the SDK window, as there will be a lot to see.

Launching the Xilinx Software Command Line Tool

1-1. Launch the Xilinx Software Command Line Tool (XSCT).

- 1-1-1. [Windows users]: Select **Start > All Programs > Xilinx Design Tools > SDK 2019.1 > Xilinx Software Command Line Tool 2019.1** to launch the tool.

Alternatively, you can launch the tool from its desktop shortcut, if available.

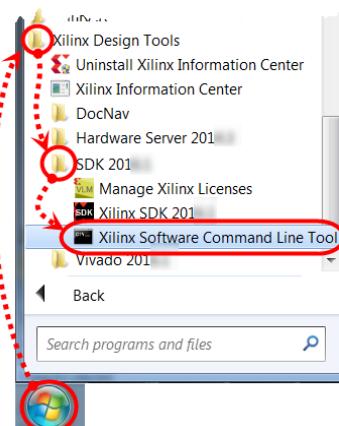


Figure 235: Launching XSCT

[Linux users]: Press <**Ctrl + Alt + T**> to launch a terminal, enter **xsct**, and press <**Enter**>.

The Xilinx Software Command Line Tool opens.



Figure 236: Xilinx Software Command Line Tool

You can now enter Tcl commands into the tool.

Running a Tcl Script within the Xilinx Software Command Line Tool

1-1. Run the provided Tcl script to build the SDK workspace

- 1-1-1.** Enter the following command to change the directory to the location of the Tcl script you want to run:

[Windows users]: cd the location of the Tcl file to open

[Linux users]: cd /home/xilinx/training/<the topic cluster name>/support

- 1-1-2.** Enter the following command to run the script:

source your Tcl script

This will load the environment with the Tcl *procs* contained in your Tcl script and run any Tcl code not included in the procs.

Importing an Example Application from a Library

1-1. Import an example application.

- 1-1-1. Scroll to the bottom of the *system.mss* file.

At the bottom of the page, there is a Libraries section that contains the your files library or libraries.

- 1-1-2. Click the **Import Examples** link next to the your library library under the libraries entry (1).

The Import Examples dialog box opens, displaying a list of all of the examples that are available for this library.

- 1-1-3. Select the **the example application** example (2).

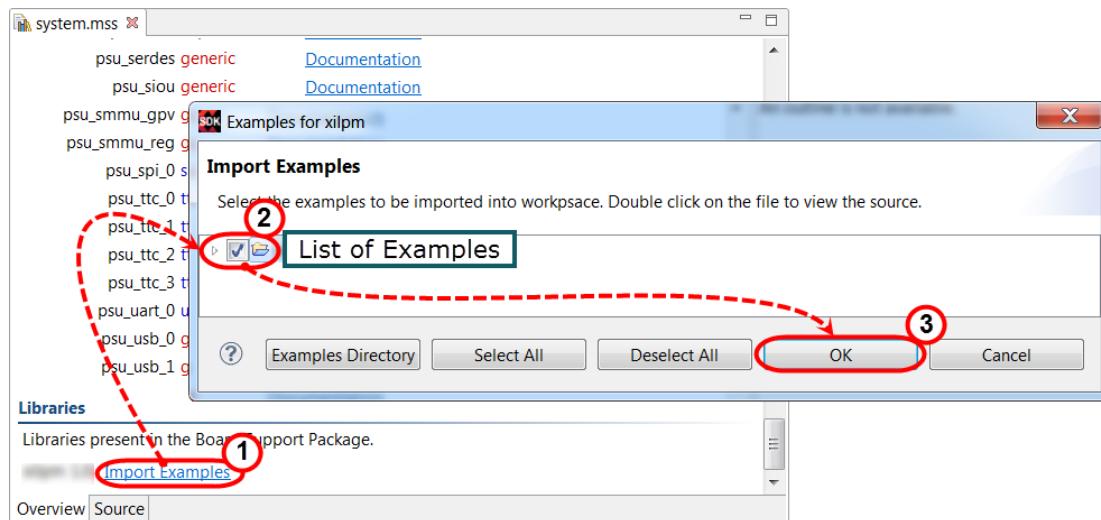


Figure 237: Importing Library Example

- 1-1-4. Click **OK** to close the dialog box and create a new application project for this example based on this BSP (3).

Creating a Hardware Platform Specification

The hardware platform specification contains a thorough description of the hardware design: what types of processors are present, active peripherals in the PS and PL for Zynq® SoC-based systems or a list of all peripherals for a non-Zynq SoC system, a full system memory map, etc. Based on this description, software such as the board support package (BSP) and application can be tailored to the hardware.

Often you will use your own hardware description file (typically generated by the Vivado Design Suite); however, there may be times when you just want to quickly test your code using the standard peripheral set in a Zynq device. Xilinx provides a collection of predefined hardware templates for a number of Xilinx and vendor-based boards.

1-1. Create the hardware platform specification.

- 1-1-1. Select **File** (1) > **New** (2) > **Other** (3) to see the Xilinx-specific options.

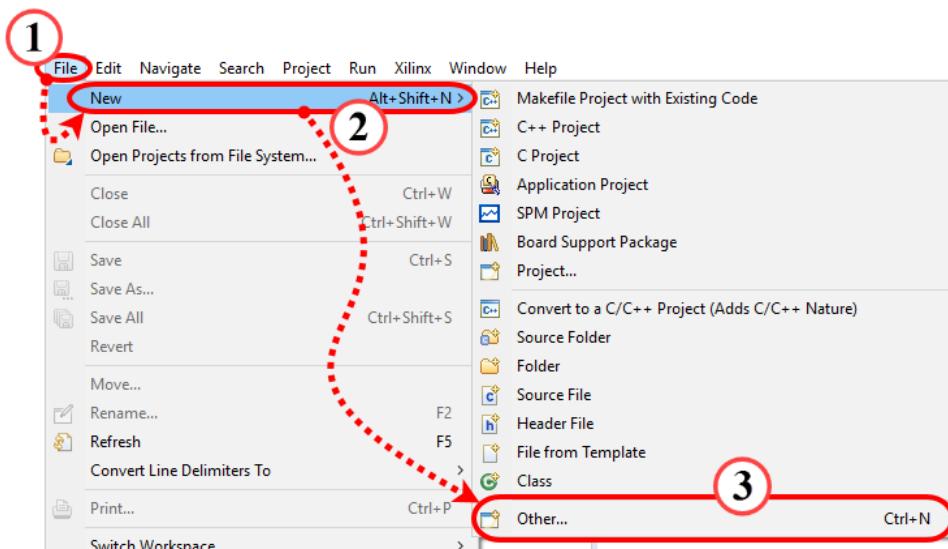


Figure 238: Accessing the New Wizards

The Select a Wizard dialog box opens. Here you can select one of many different wizards.

1-1-2. Expand the **Xilinx** folder (1).

1-1-3. Select **Hardware Platform Specification** (2).

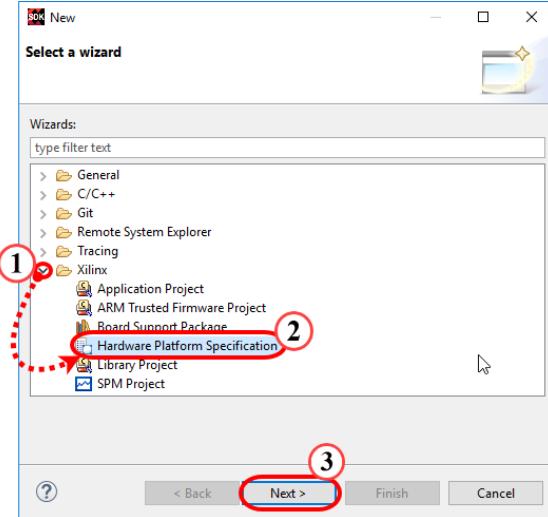


Figure 239: Selecting the Hardware Platform Specification Wizard

1-1-4. Click **Next** to open the New Hardware Project dialog box (3).

The New Hardware Project dialog box opens. Here you will be able to specify a project name and the hardware description file that was exported by the Vivado Design Suite.

1-1-5. Enter **your hardware platform description name** in the *Project name* field.

1-1-6. **[Windows users]:** Browse to the *the location of the files to be exported from the Vivado Design Suite* directory under the Target Hardware Specification region and select the **name of your block design** file.

[Linux users]: Browse to the */home/xilinx/training/tools* directory under the Target Hardware Specification region and select the **name of your block design** file.

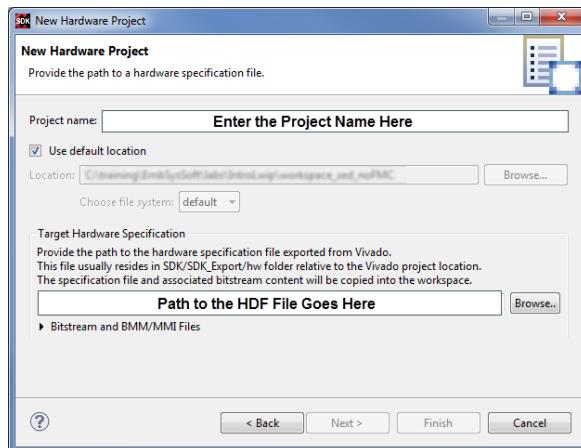


Figure 240: New Hardware Project Dialog Box

1-1-7. Click **Finish** to create the new hardware project.

Creating a Hardware Platform Specification and BSP

1-1. Create the hardware platform specification and board support package.

1-1-1. Select **File > New > Board Support Package**.

1-1-2. Enter the project name as **your BSP name**.

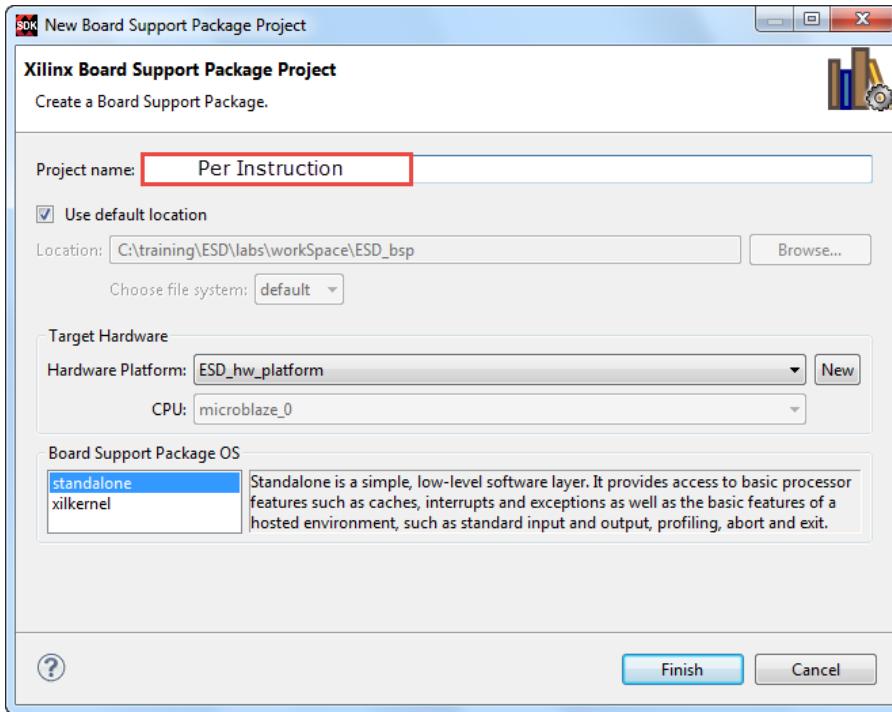


Figure 241: Newly Created Hardware Project

SDK supports multiple hardware platform projects in the same workspace.

Note: When you scroll lower in the *system.hdf* file, you can display information about the IP.

1-1-3. Click **Finish**.

1-1-4. Click **OK** in the Board Support Package Settings dialog box.

Updating a Hardware Platform Specification

Often the hardware will change during the development process. This could include changes to the hardware design itself or include a bitstream where one was not present before.

1-1. Update the hardware platform specification.

1-1-1. Right-click the hardware platform specification that you want to update to open the context menu (1).

1-1-2. Select **Change Hardware Platform Specification** (2).

Typically, a warning message appears alerting you to a change (3).

1-1-3. Click **Yes** to acknowledge that a change is imminent (4).

1-1-4. Navigate to *the path of your hardware platform description* (5).

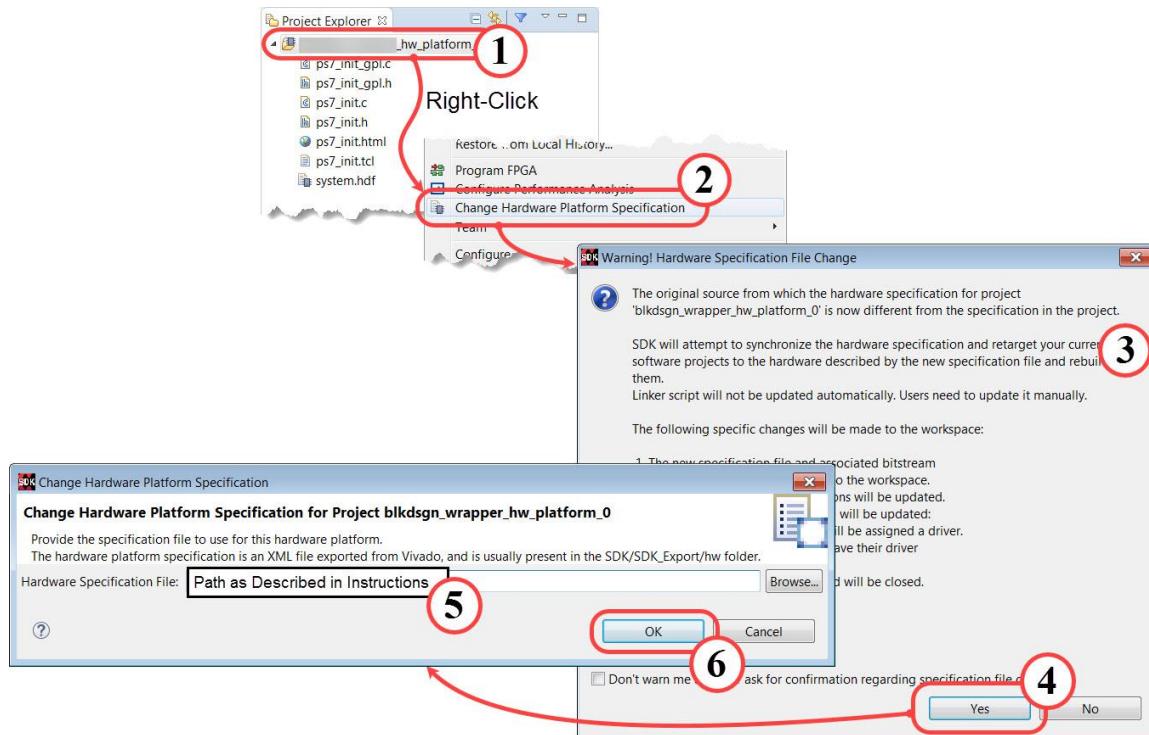


Figure 242: Changing the Hardware Platform Specification

1-1-5. Click **OK** to accept and load the new hardware platform specification (6).

Creating a Basic Board Support Package

1-1. Create a basic board support package.

1-1-1. Select **File > New > Board Support Package**.

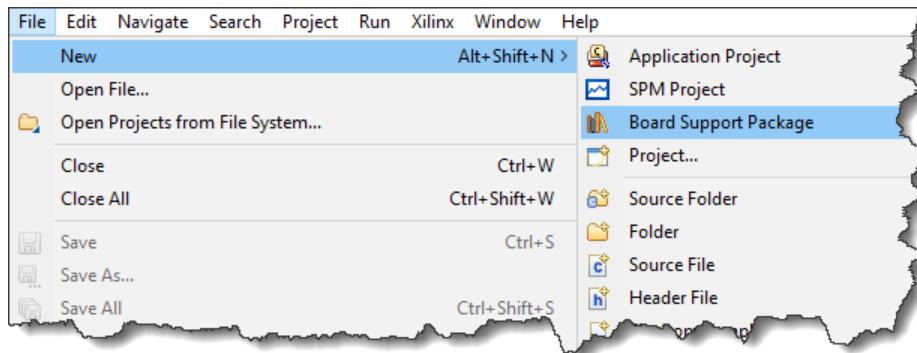


Figure 243: Launching the Board Support Package Wizard

The Xilinx Board Support Package Project dialog opens. Here you will give this BSP a name and tie it to both a design (hardware platform) and a particular CPU within that design.

1-1-2. Enter **your BSP name** in the Project name field.

Most workspaces use a single hardware platform.

If your design has multiple hardware platforms, ensure that **your hardware platform description name** is selected from the Hardware Platform drop-down list.

If your embedded design has more than one CPU within a hardware platform, ensure that **your processor** is selected from the CPU drop-down list.

1-1-3. Select **your desired operating system** from the Board Support Package OS section.

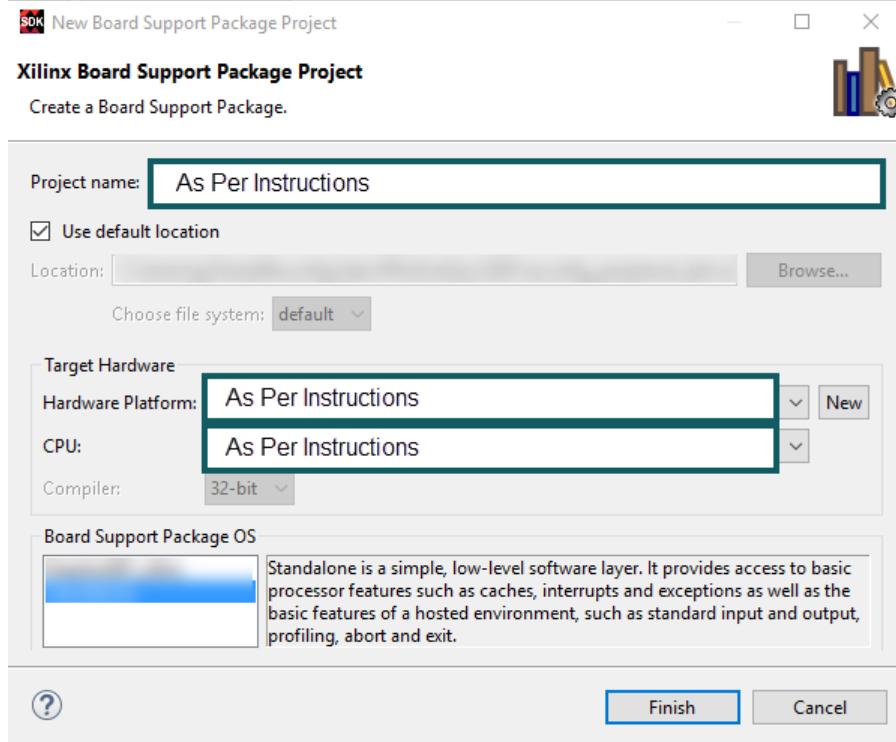


Figure 244: Board Support Package Project Dialog Box

1-1-4. Click **Finish**.

Now that the BSP has been created, it can be customized if so desired.

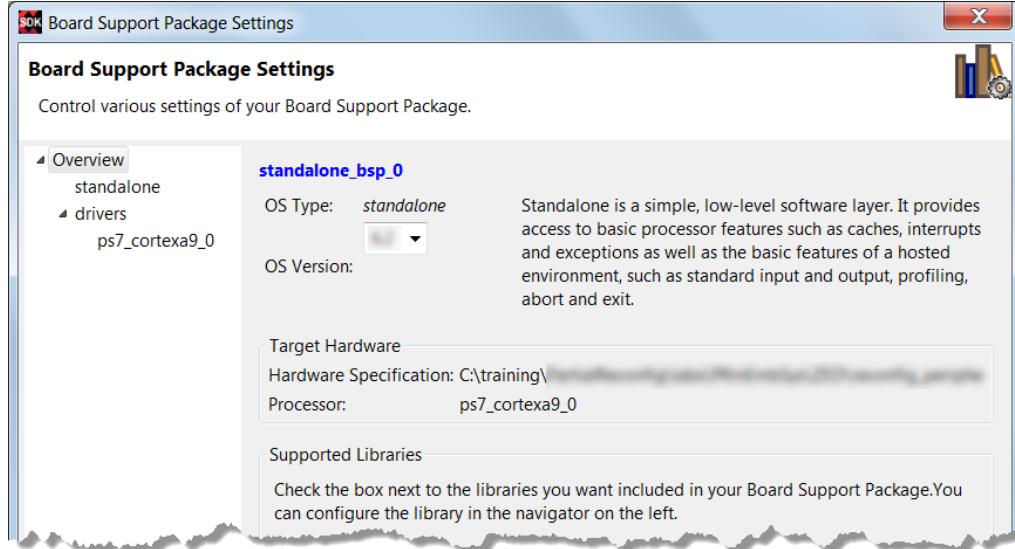


Figure 245: Board Support Package Settings Dialog Box

1-1-5. Click **OK** to use the default settings.

Enabling C Libraries

1-1. Enable libraries for *your application project name*.

1-1-1. From the Project Explorer tab, right-click **your application project name**.

1-1-2. Select **C/C++ Build Settings**.

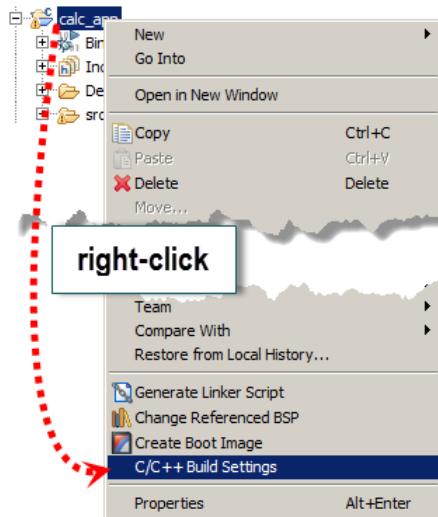


Figure 246: Accessing the C/C++ Build Options

The C/C++ Build Properties window opens.

1-1-3. If necessary, expand the processor (**ARM** or **Microblaze**) **gcc linker** under Tool Settings.

1-1-4. Select **Libraries** (2).

1-1-5. Click the **Add** icon (3).

The Enter Value dialog box opens so that you can enter additional libraries.

1-1-6. Enter "**m**" to include the math libraries (4).

Refer to the user guide for other library codes.

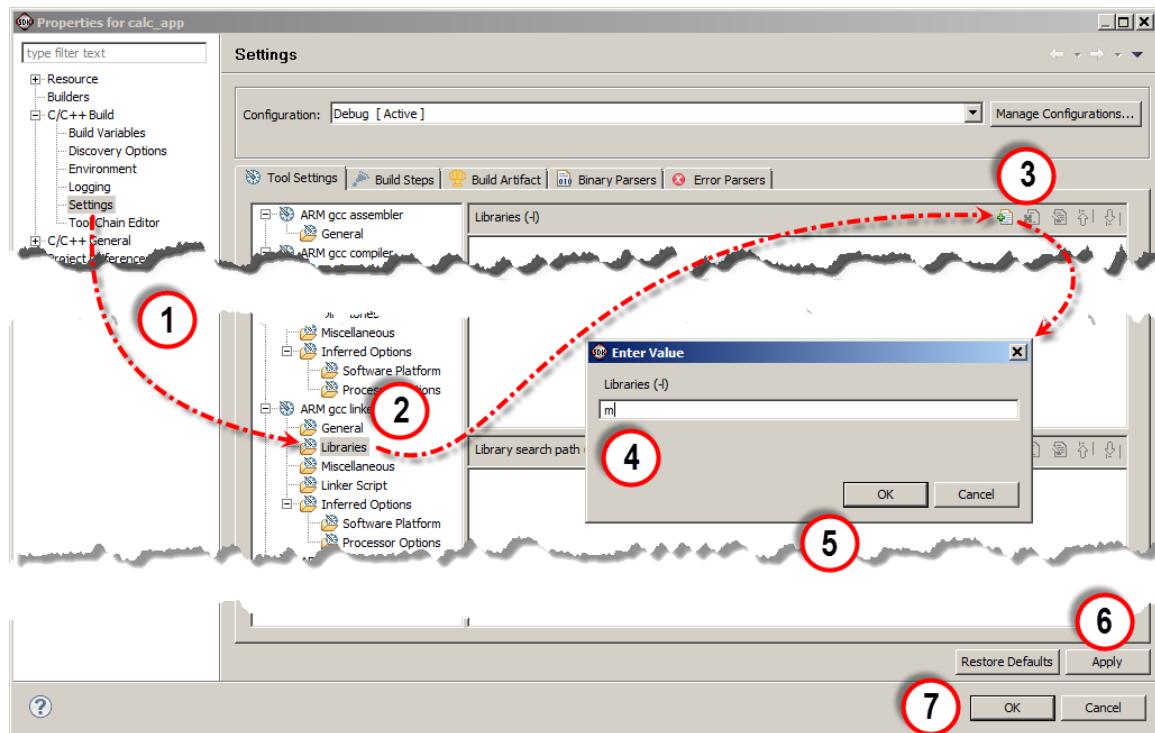


Figure 247: Adding the Math Library to the Linker

- 1-1-7.** Click **OK** (5).

You will see the math library (m) appear under the Libraries (-l) display pane.

- 1-1-8.** Click **OK**.

This will cause the project to be rebuilt with the new options.

Creating a C/C++ Application Project with a New BSP

Using the Application Project Wizard is a quick way to set up a C or C++ software application project that targets an existing processor and OS platform (Standalone, FreeRTOS, or Linux). You can automatically generate the board support package (BSP), as you will here, or select an existing BSP. Based on the dialog box choices, the appropriate tool chain is selected for pre-processing, compiling, assembling, and linking.

1-1. Create a new Standalone application targeted for the processor you wish to target.

- 1-1-1. Select **File > New > Application Project** to open a new project window.

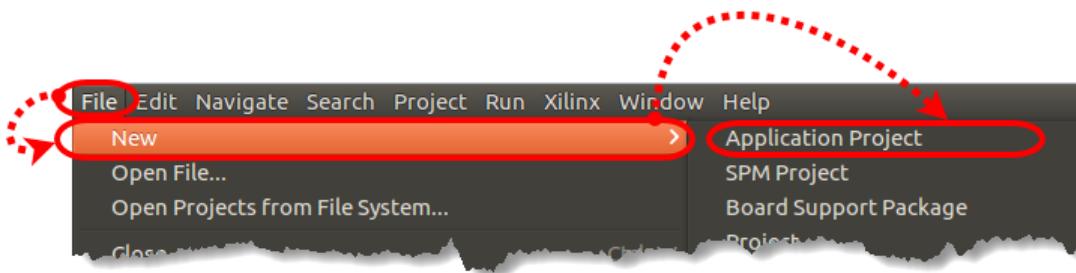


Figure 248: Navigating to the Application Project in SDK

- 1-1-2. Enter **your application_app** into the Project name field (1).

While the "_app" is not required, it does make identifying what kind of project the entry is much easier.

- 1-1-3. Ensure that **your desired operating system** is selected from the OS Platform drop-down list (2).

- 1-1-4. Ensure that you have **your hardware platform description name** selected from the Hardware Platform drop-down list as the SDK or SDSoc tool can manage multiple platforms within a single workspace (3).

This will populate the processor drop-down list based on the processors available in the indicated hardware platform description.

- 1-1-5. Ensure that **the processor you wish to target** is selected from the processor drop-down list (4).

This is a critical element in multi-processor designs as the application will be targeted to a specific processor. Additionally, the application templates that will be offered are based on the type of processor.

- 1-1-6. Select **your preferred language** as the compiler to use (5).

If active, leave the compiler and hypervisor guest options at their default.

By default, the New Project Wizard will attempt to create a new BSP for this application. Leave the option set to "Create New".

1-1-7. Enter the new BSP project name as **your application_bsp**.

By default, the tool will copy the project name to the BSP name and append "_bsp". While there is nothing wrong with this, these instructions will follow the philosophy of using only a single descriptor at the end of a project name: _app for applications, _bsp for BSPs, and _hw for hardware projects.

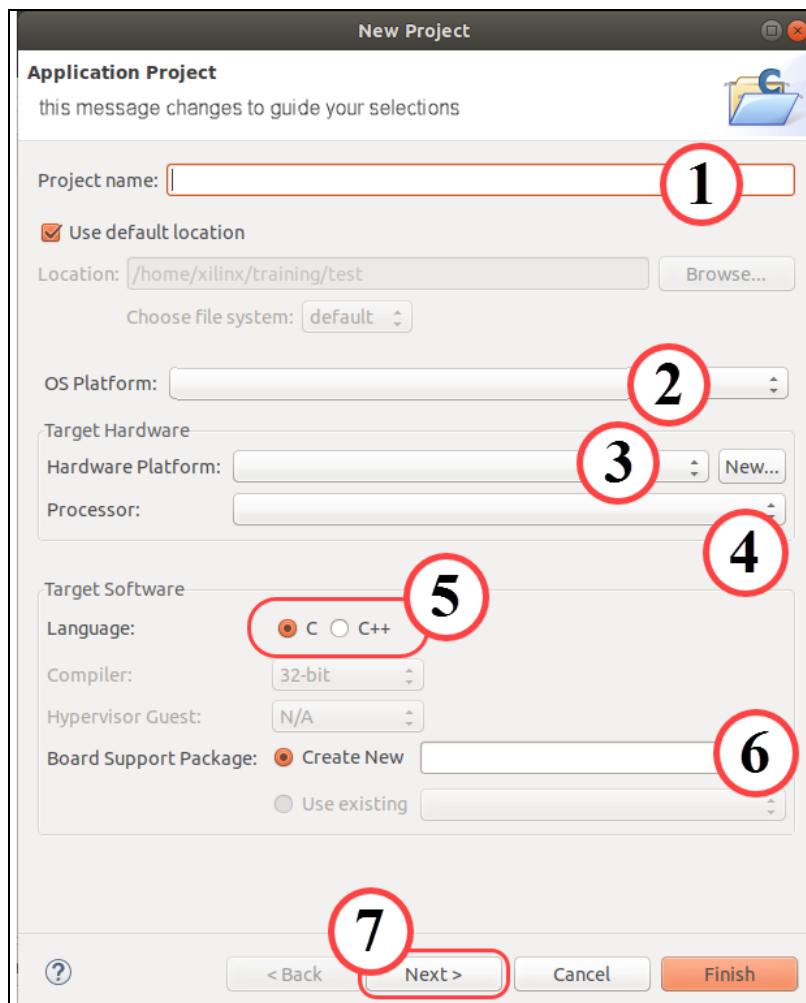


Figure 249: Creating a New Application and BSP Project

If you click Finish to create the project and bypass the selection of an application template, then the blank application template will be used.

1-1-8. Click **Next** to select an application template (7).

1-1-9. Select an application template (1).

The applications shown here represent the templates that are available for a specific processor. Depending on the processor selection, the options of templates will change.

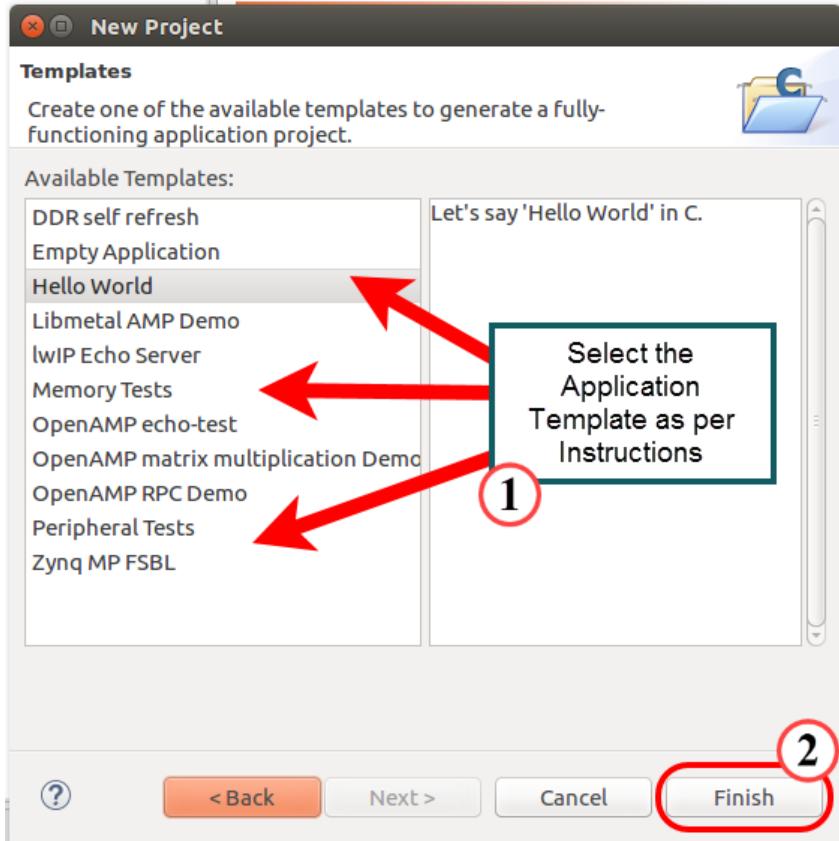


Figure 250: SDK Select Application Templates in Linux

1-1-10. Click **Finish to create the new application project and BSP (2).**

As the project is created, the newly generated BSP is compiled and the sources added as templates are also compiled. The status and compiler messages are visible in the Console tab. The creation of the new application project takes less than a minute.

Creating a C/C++ Application Project with an Existing BSP

Using the Application Project Wizard is a quick way to set up a C or C++ software application project that targets an existing processor and OS platform (Standalone or Linux). You can automatically generate the board support package (BSP) or select an existing one. Based on the dialog box choices, the appropriate tool chain is selected for pre-processing, compiling, assembling, and linking.

1-1. Create a new C/C++ application project named ***your application project name***. Use the board support package named ***your BSP name***.

- 1-1-1. Select **File** (1) > **New** (2) > **Application Project** (3) to open the New Project dialog box.

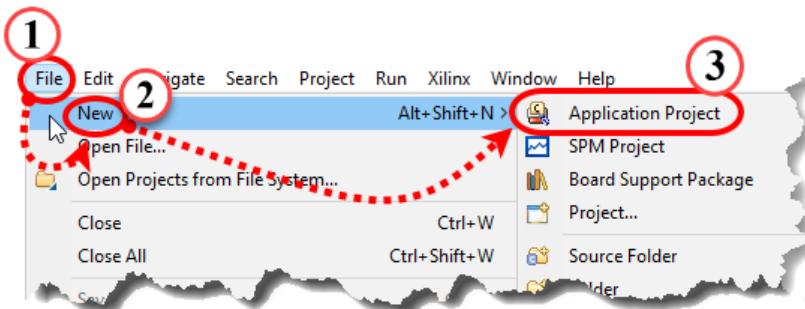


Figure 251: Creating an Application Project

- 1-1-2. Enter ***your application project name*** as the project name.
- 1-1-3. Ensure that you have ***your hardware platform description name*** selected from the Hardware Platform drop-down list as the SDK or SDSoc tool can manage multiple platforms within a single workspace.
This will populate the Processor drop-down list accordingly.
- 1-1-4. Ensure that ***the processor you wish to target*** is selected from the Processor drop-down list.
- 1-1-5. Ensure that ***your desired operating system*** is selected from the OS Platform drop-down list.
- 1-1-6. Select **Use Existing** in order to select one of the existing BSPs that have been created for this processor.
- 1-1-7. Choose ***your BSP name*** from the drop-down list.

Note that BSPs are based on the processor itself, not the type of processor. That is, if you create a BSP for processor 0 in the PS then that BSP is only available for applications targeting processor 0.

If an application is to run on processor 1, even if it is the same type of processor as processor 0, a different BSP must be created for that processor.

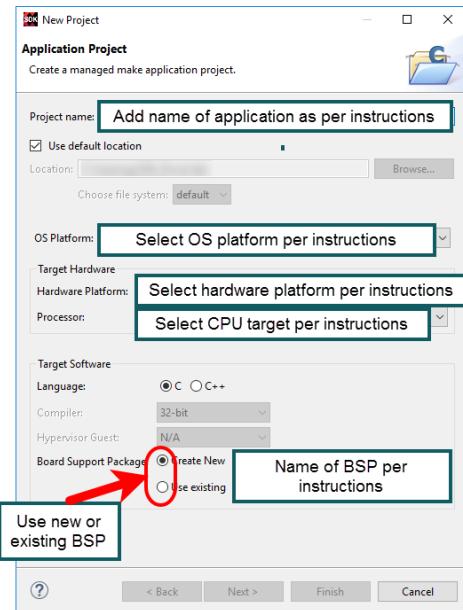


Figure 252: Entering Application Project Information

1-1-8. Click **Next** to select the template for this application.

1-1-9. Select **an application template** (1).

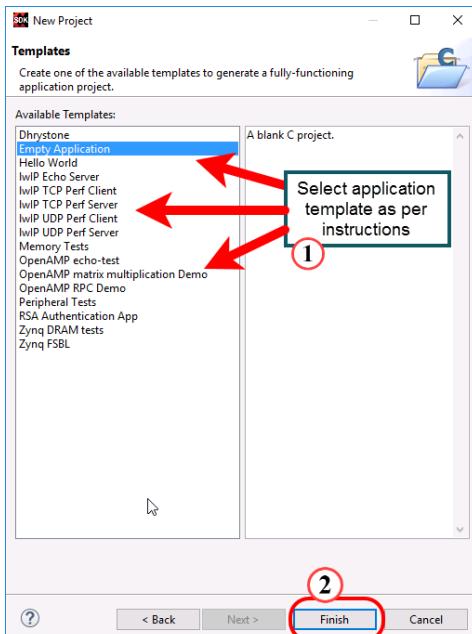


Figure 253: Selecting an Application Template (Selection in Figure May Not Match Instructions)

1-1-10. Click **Finish** to create the new project (2).

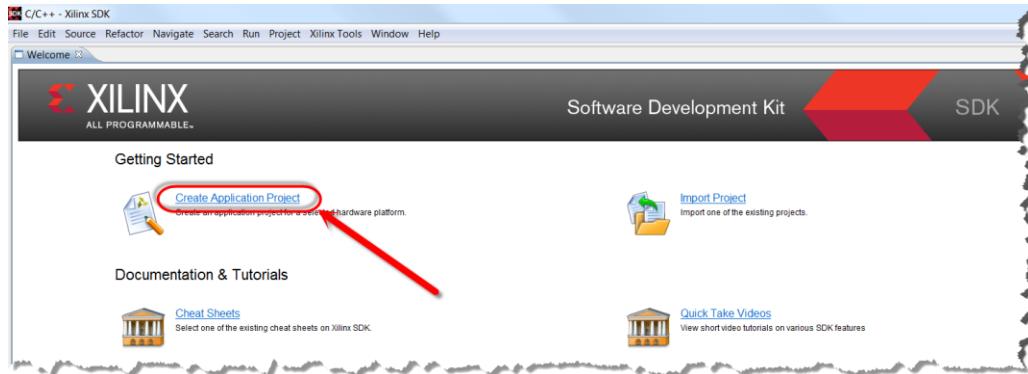
As the project is created, the new BSP is compiled and any sources from the templates are also compiled. The status and compiler messages are visible in the *Console* tab. The creation of the new project usually takes less than a minute.

Creating a C/C++ Application Project from the SDK Welcome Window

Using the Application Project wizard is a quick way to set up a C or C++ software application project that targets an existing processor and OS platform (standalone or Linux). You can automatically generate the board support package (BSP) or select an existing one. Based on the dialog box choices, the appropriate tool chain is selected for pre-processing, compiling, assembling, and linking.

- 1-1. Create a new C/C++ application from the welcome window. Name the project ***your application project name***. Use the board support package named ***your BSP name***.**

- 1-1-1. Click **Create Application Project****



- 1-1-2. Enter the *Project name* as ***your application project name***.**
- 1-1-3. Ensure that you have ***your hardware platform description name*** selected as SDK can manage multiple platforms within a single workspace.**

Notice that there are a number of pre-defined targets available. These targets are for Zynq boards from both Xilinx and Avnet. They contain the descriptions for the peripherals available in the PS. If your design contains any peripherals in the PL, you will need to load your board specific platform description.

- 1-1-4. Ensure that ***the processor you wish to target*** is selected.**

- 1-1-5.** If you have already created a BSP for this hardware platform you can select **Use Existing** and choose an existing BSP from the workspace; otherwise, select **Create New** and enter a name for the BSP, or you can use the default name provided.

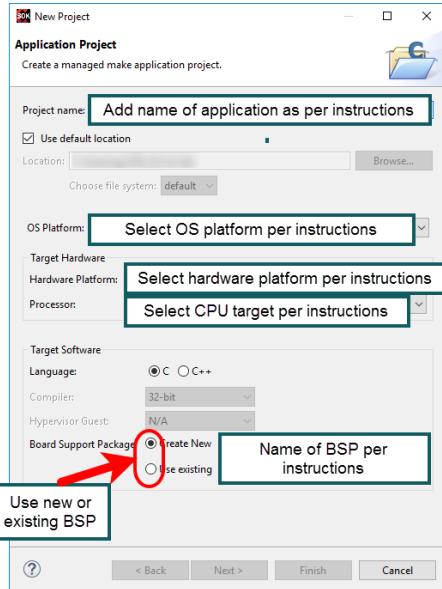


Figure 254: Entering Application Project Information

- 1-1-6.** Click **Next**.

- 1-1-7.** Select an application template.

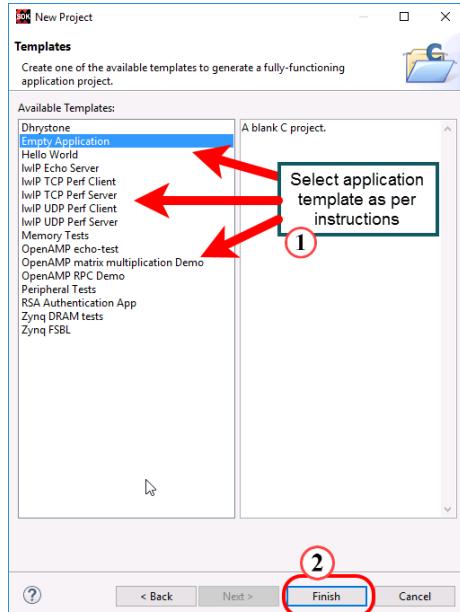


Figure 255: Selecting an Application Template (Selection in Figure May Not Match Instructions)

- 1-1-8.** Click **Finish**.

Importing Sources to an Application

It is common practice to add existing resource files (*.c, *.h, *.cpp, etc.) to a software project. The SDK tool requires this operation to be performed as an import function.

1-1. Add *your files* to the application.

The preferred method for importing sources is shown here.

- 1-1-1. Expand the project named **your application project name** > **src** using the Project Explorer.
- 1-1-2. Right-click the desired destination directory in the project that you want to place the resource files (1).

This is typically the **src** directory.

- 1-1-3. Select **Import** to open the Import Wizard (2).

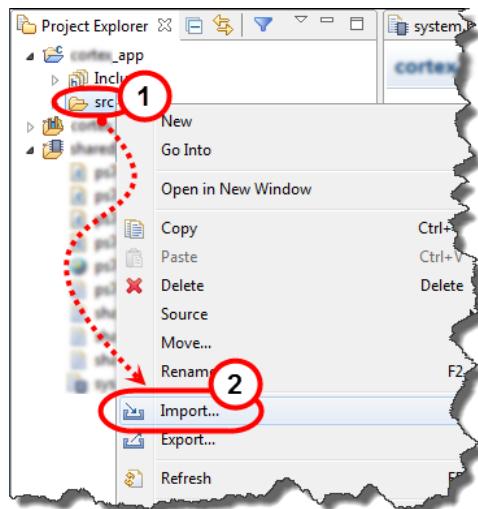


Figure 256: Importing a Resource File

The Import Wizard dialog box opens.

- 1-1-4. Expand the **General** branch (1).
- 1-1-5. Select **File System** as you will be selecting individual files directly from the file system (2).

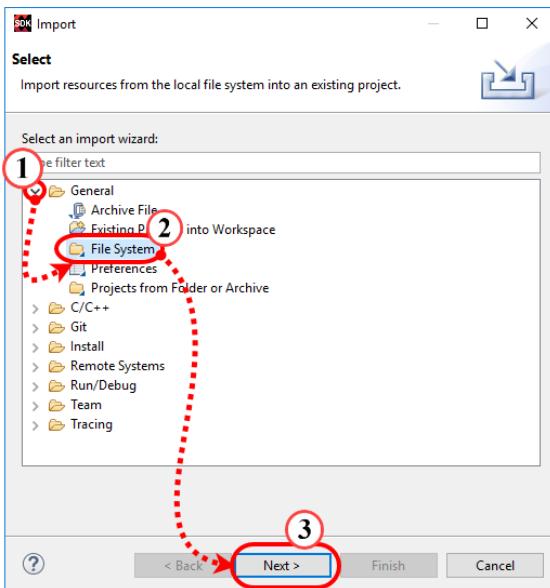


Figure 257: Selecting File System

- 1-1-6. Click **Next** to advance to specifying the files to import (3).
- 1-1-7. Browse to the directory where your files are located.
You are only selecting the directory at this time—the file selection comes next.
- 1-1-8. Select the file(s) by checking the box beside **your files**.

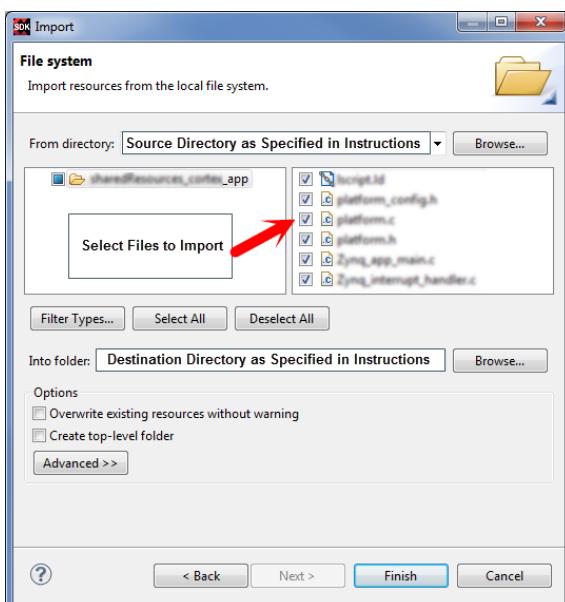


Figure 258: Selecting Resource Files

The *Into folder* directory will default to the location selected when you engaged the import function. If you wanted to change this, you could click **Browse** and select a new location.

- 1-1-9. Click **Finish** to import the selected files and close the wizard.

Note: If the workspace has the automatic build option enabled, the project will automatically build with the new resource files. The Console view at the bottom of the IDE will show the results of the build.

Launching the SDK Tool and Setting the Workspace in a Linux Environment

1-1. Launch the SDK tool and set the workspace.

- 1-1-1. Click the SDK icon on the taskbar, desktop, or quick launch bar to launch SDK.
- 1-1-2. Enter **your workspace location** into the Workspace field when the Workspace Launcher opens, or use the Browse button.

Hint: Remember to expand \$TRAINING_PATH as described in the lab setup guide..

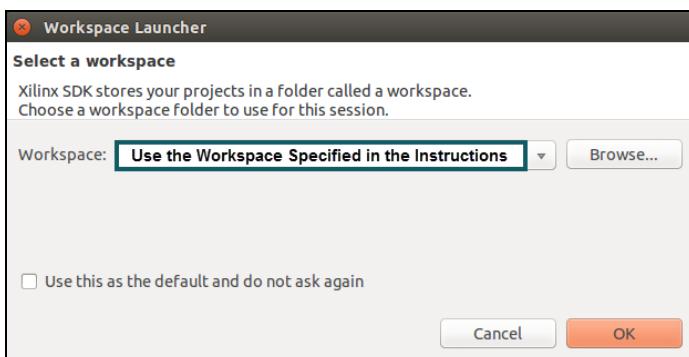


Figure 259: Setting Up the Workspace Environment Path

- 1-1-3. Click **OK** to close the Workspace Launcher dialog box and open the new workspace.
- 1-1-4. Close the **Welcome** tab if it appears by clicking the 'X' in the Welcome tab.



Figure 260: Closing XSDK's Welcome Tab

This will give you more room to view your project. You may also want to maximize the SDK or SDSoc window as there will be a lot to see.

Adding a Library to a BSP

1-1. Open the BSP settings.

- 1-1-1. Expand the **your BSP name** project so that the *system.mss* file is shown (1).

The *system.mss* file provides an overview of the BSP and supports modification and regeneration of the BSP and its sources.

- 1-1-2. Double-click the **system.mss** file to open the Board Support Package Project settings dialog box (2).

- 1-1-3. Click the **Modify this BSP's Settings** button to open a dialog box that contains the BSP Settings (3).

- 1-1-4. Click **Overview** in the left window pane (4) if the Overview view is not selected.

- 1-1-5. Select **your library** under the Supported Libraries section (5).

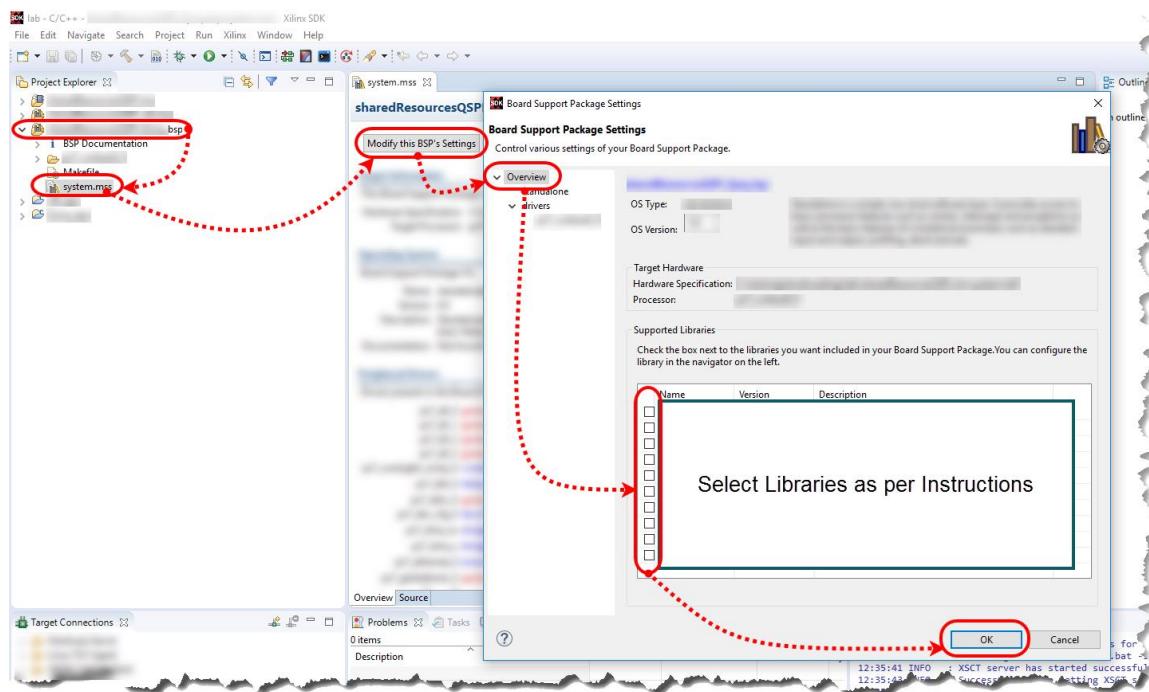


Figure 261: Accessing the BSP Customization Dialog Box

- 1-1-6. Click **OK** to modify the BSP and rebuild the BSP's sources and binaries (6).

Importing an Existing Project into the SDK or SDSoC Tool

One or more projects can be exported as a single zipped file. This is convenient when passing projects or parts of projects to teammates or clients. Once the recipient has received this archive, the zipped file must be processed and one or more projects can then be imported.

1-1. Import an existing project.

- 1-1-1. Select **File > Import** to open the Import Wizard.

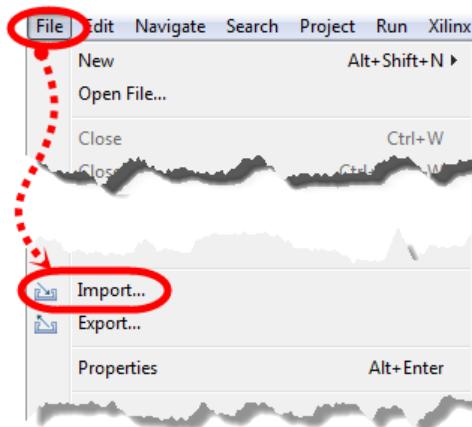


Figure 262: Accessing the Import Wizard

The Import dialog box opens.

- 1-1-2. Expand the **General** node to access the commonly used methods (1).
- 1-1-3. Select **Existing Projects into Workspace** as the goal is to import an existing project (2).

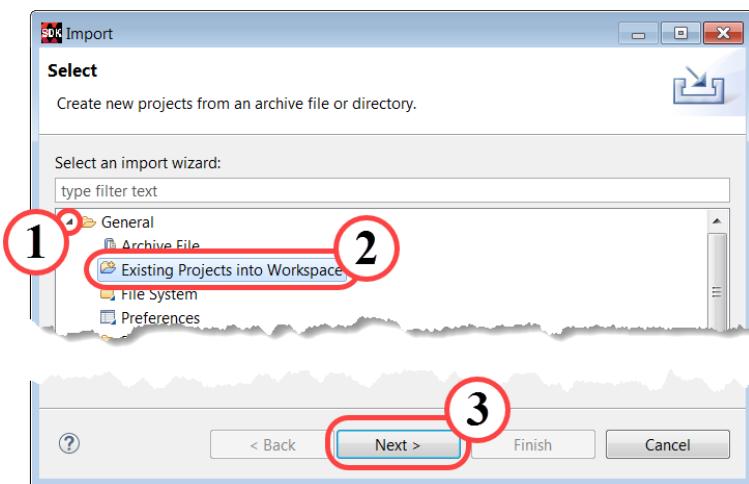


Figure 263: Choosing to Import an Existing Project into the Workspace

- 1-1-4. Click **Next** to enter project-specific data (3).

1-1-5. Select the **Select archive file** option (1).

Note that projects can be archived either as single zip files, or you can import from a directory. Typically, projects are preserved as archives as they are easier to move.

1-1-6. [Windows users]: Click **Browse** to navigate to *where your SDK project archive file is located* (2).

[Linux users]: Click **Browse** to navigate to /home/xilinx/training/<the topic cluster name>/support (2).

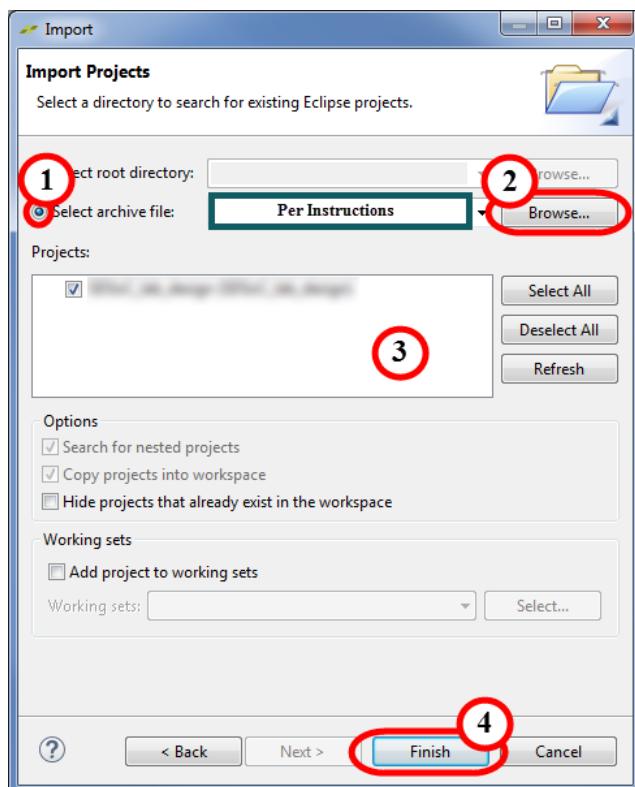
1-1-7. Select **your SDK project archive file**, which contains the archived projects.**1-1-8.** Click **Open** to open the archive and list the various projects in that archive.**1-1-9.** Select **the projects from within the archive** to import (3).

Figure 264: Import Settings for Archived Projects

1-1-10. Click **Finish** to perform the importing of the project (4).

Enabling and Disabling Automatic Compilations

1-1. Enable or disable the automatic compilation capability.

1-1-1. Select Project > Build Automatically.

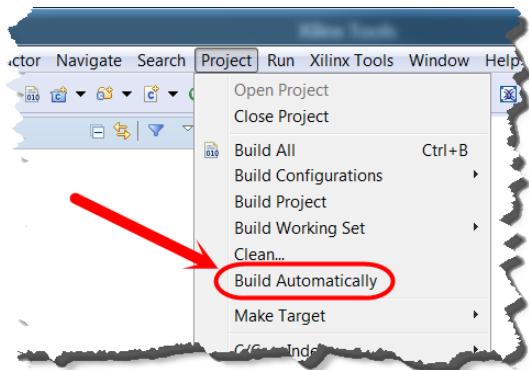


Figure 265: Selecting Build Automatically

This will toggle the Build Automatically option. When the Build Automatically option is active, a blue check mark will appear next to it.

Setting Compiler Options

Compiler options describe the designer's intentions and goals to the compiler. These options directly affect how the object file is constructed. Here you will visit some of the most commonly used options.

1-1. Access the compiler options for your application.

1-1-1. Right-click the application to change or verify compiler options for and select C/C++ Build Settings.

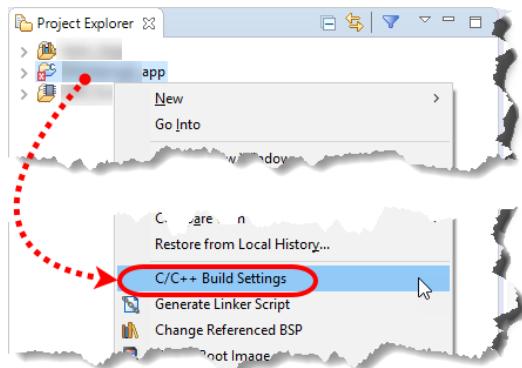


Figure 266: Accessing the Compiler Settings for a Specific Application

The C/C++ Build Settings dialog box opens.

- 1-1-2.** Click **Settings** to access the specific settings for the compiler.

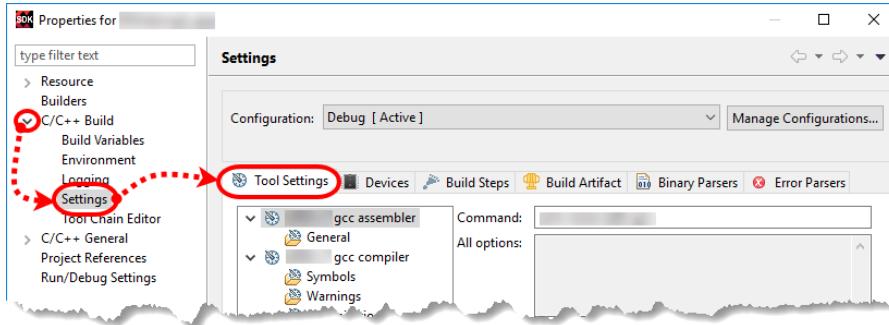


Figure 267: Accessing the Compiler Settings

- 1-2.** **Compiler symbols enable you to define symbols for the pre-processor. By defining symbols here, no changes need to be made to the source code. The Defined Symbols entry enables you to add, remove, edit, and reorder your symbols.**

Remember that #ifdef only tests to see if a preprocessor symbol has been defined or not, and #if can test against specific values (e.g., assigning a value to a symbol can be done like this: NEW_SYMBOL=123).

- 1-2-1.** Click **Symbols** under the compiler entry under the Tool Settings tab.
- 1-2-2.** Click the green + icon to create a new symbol.
- 1-2-3.** Enter **the symbol name (and optionally a symbol value)** into the Enter Value dialog box.
- 1-2-4.** Click **OK** to complete the creation of this new symbol.

The above steps can be repeated as necessary to create as many symbols as you need.

- 1-2-5.** Click **Apply**.

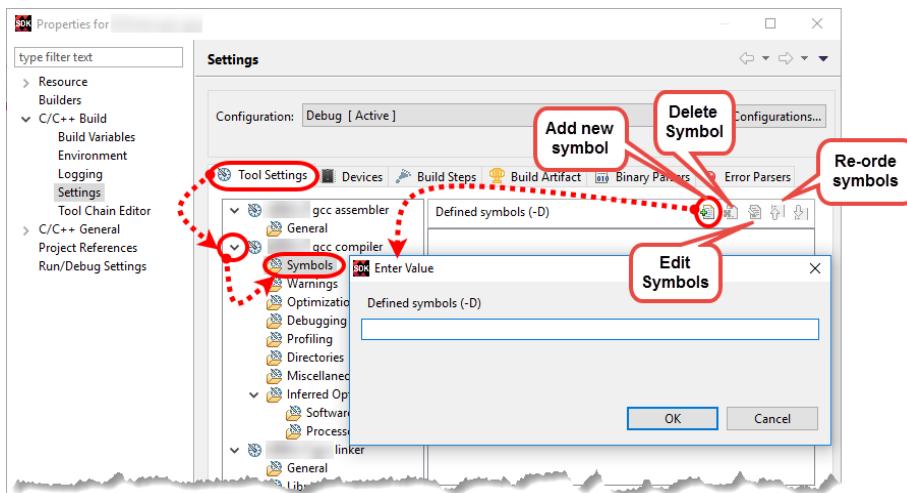


Figure 268: Creating a New Symbol

- 1-3.** The Optimization tab enables you to select the level of optimization needed to meet your timing requirements for this application. Note that the optimization should be set to Zero (-O0) for debugging; otherwise, the one-to-one correspondence between source code and executing code is lost.

Remember that the levels of optimization listed in the pull-down menu are actually collections of specific optimization flags. Refer to the manual for a listing of which flags are enabled for each optimization level. If you are not happy with the pre-defined optimization levels, you can add your own optimization flags to the *Other optimization flags* text field. Remember that optimizations can change the way your code operates!

- 1-3-1.** Select the **Optimization** tab.
- 1-3-2.** Select your default level of optimization from the Optimization Level drop-down list.
- 1-3-3.** Add any additional flags to the Other optimization flags field.
- 1-3-4.** Click **Apply**.

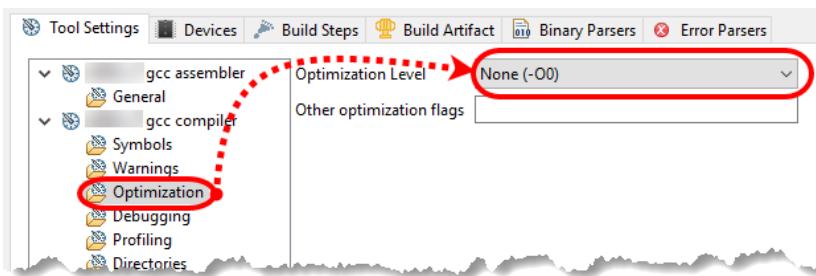


Figure 269: Setting Optimization Levels

- 1-4.** The Debugging tab enables you to select the level of debug information to debug the application. The debug levels are -g1 (minimal debug information), -g (default debug information) and -g3 (maximum debug information).

- 1-4-1.** Select the **Debugging** tab.
- 1-4-2.** Select **Maximum (-g3)** from the Debug level drop-down list.

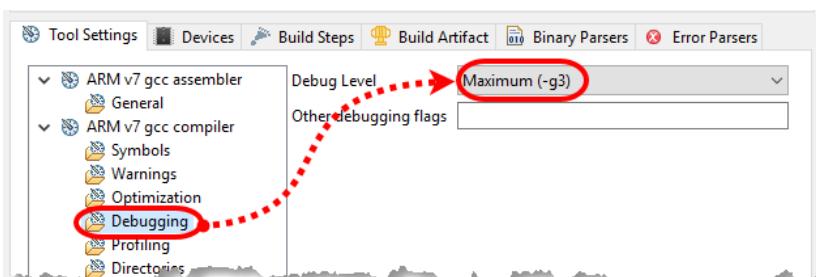


Figure 270: Setting Debug Level

1-4-3. Click **OK.**

The application is rebuilt. A successful build is indicated when the program size is returned.

Setting Compiler Options [No Macro Settings]

Compiler options describe the designer's intentions to the compiler. These options directly affect how the object file is constructed. Here you will visit some of the most commonly used options.

1-1. Access the compiler options for the application project.

- 1-1-1. Right-click the application that you want to change/verify compiler options for and select **C/C++ Build Settings**.

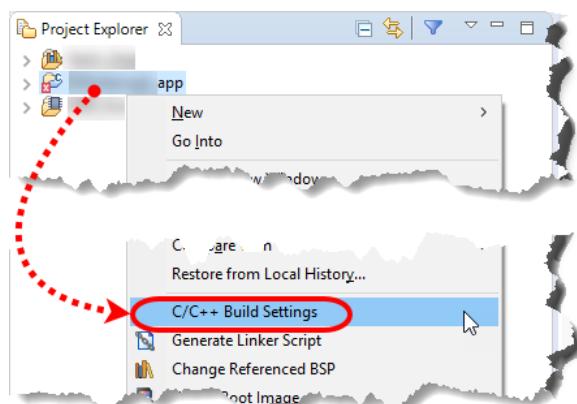


Figure 271: Accessing the Compiler Settings for a Specific Application

The C/C++ Build Settings dialog box opens.

- 1-1-2. Click **Settings** to access the specific settings for the compiler.

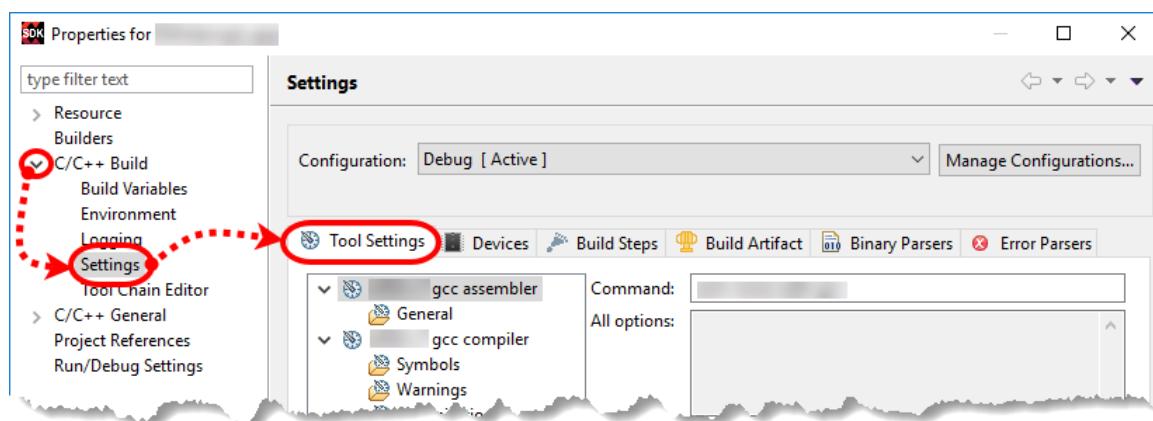


Figure 272: Accessing the Compiler Settings

- 1-2.** The Optimization tab enables you to select the level of optimization needed to meet your timing requirements for this application. Note that the optimization should be set to Zero (-O0) for debugging; otherwise, the one-to-one correspondence between source code and executing code is lost.

Remember that the levels of optimization listed in the pull-down menu are actually collections of specific optimization flags. Refer to the *GNU Compiler Manual* for a listing of which flags are enabled for each optimization level. If you are not happy with the pre-defined optimization levels, you can add your own optimization flags to the *Other optimization flags* text field. Remember that optimizations can change the way your code operates!

- 1-2-1.** Select the **Optimization** entry in the Tool Settings list to access the optimization settings.
- 1-2-2.** Select your desired level of optimization from the Optimization Level drop-down list.
If there are any additional flags to the Other optimization flags field, you can add them here.
- 1-2-3.** Click **Apply**.

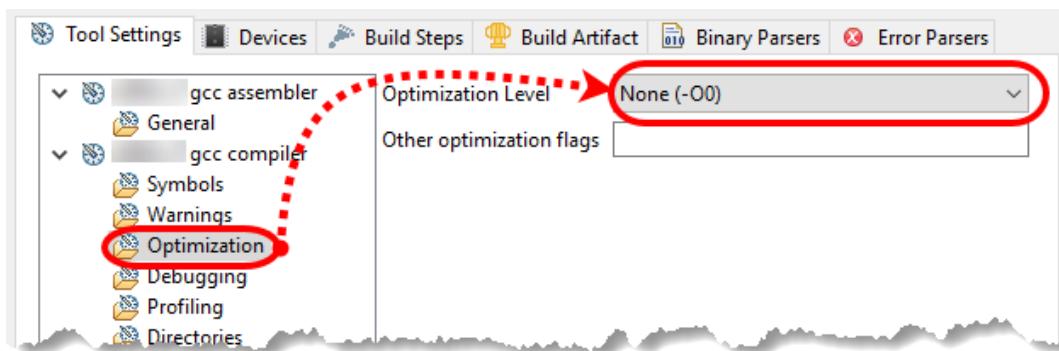


Figure 273: Setting Optimization Levels

- 1-2-4.** Click **OK** to save your settings and exit.

The application is rebuilt. A successful build is indicated when the program size is returned.

Adding Symbols to Application Project Settings

Adding a symbol to an application project is equivalent to adding a #DEFINE SYMBOL_NAME statement that is visible to the entire project. It is common practice to use symbols to conditionally compile code by guarding code with conditional pre-processor statements (e.g. #ifdef SYMBOL_NAME ... #endif).

1-1. Add symbol(s) to compiler settings.

- 1-1-1. Right-clicking the application project folder (**your application project name**) in the Project Explorer pane to open the context menu.
- 1-1-2. Select **C/C++ Build Settings** to open the edit window for this project.
- 1-1-3. From the left side of the properties window, select **Settings** to view the tool settings (1).
- 1-1-4. If it is not already selected, select the **Tool Settings** tab (2).
- 1-1-5. Under the appropriate compiler entry, select **Symbols**. (3)

Since SDK supports multiple compiler tool chains for a host of processors, select the compiler for the specific processor you are using. Generally this will be ARM gcc compiler for the Zynq-7000 family of devices, and GNU for the MicroBlaze processor (in any family of SoC or FPGAs). The important aspect is that you are in the compiler options settings.

- 1-1-6. Under the Defined symbols area, click the **Add** (+) icon to open a dialog box where you can enter a new symbol name (4).
- 1-1-7. Enter a symbol name as indicated below in the Enter Value dialog box (5).
- 1-1-8. Click **OK** to add the new symbols one by one (6).
- 1-1-9. Enter **application project symbol names as required** as the symbol as per the target platform.

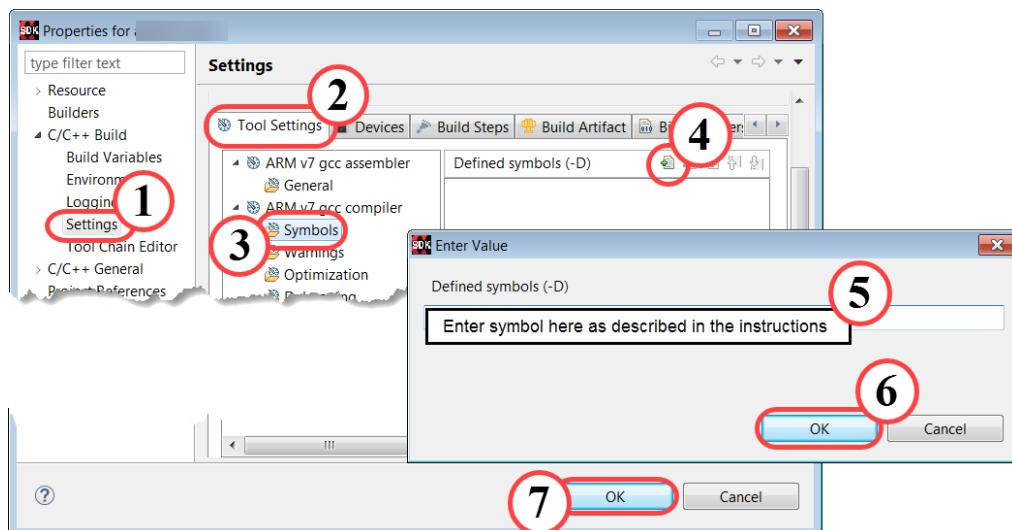


Figure 274: Defining a New Compiler Symbol

1-1-10. Once all symbols are added, click **OK** to apply the changes and close the Properties dialog box (7).

SDK will automatically rebuild your project with the new settings.

Adding Symbols to Application Project Settings

Adding a symbol to a BSP is equivalent to adding a "#DEFINE SYM_NAME" statement that is visible to all the source files that make up the BSP. BSP source files use symbols to conditionally compile code by guarding code with conditional pre-processor statements (for example, #IFDEF SYMBOL_NAME ... #ENDIF). By adding the appropriate symbol to a BSP you can add or remove various features.

1-1. Add compiler flag(s) to BSP settings.

1-1-1. Open the Board Support Package Settings dialog:

- Right-click the BSP project folder (e.g., **your BSP name**) in the Project Explorer pane.
- Select **Board Support Package Settings** from the context menu.

1-1-2. From the tree-view pane on the left, select **Overview > drivers > your processor**.

1-1-3. Under "Configuration for OS" on the right, select the Value field associated with the row for **extra_compiler_flags**.

The value field should become editable.

1-1-4. Append the following text to the existing value, using a space to separate this new string from the existing string.

- **-DUSEAMP=1**

Note that the flag -DSYMBOL is equivalent to adding a "#DEFINE SYMBOL" statement to BSP code with global scope. Similarly, the flag -DSYMBOL=VALUE is equivalent to the "#DEFINE SYMBOL VALUE" statement.

1-1-5. Once the value is updated, click OK to apply changes and close the settings dialog box.

SDK will automatically rebuild the BSP with the new settings as well as any dependent application projects.

Opening a Source File in the Editor

1-1. Open *the desired source file* in the editor.

- 1-1-1. Locate **the desired source file** using the Project Explorer pane.

Note: You may need to expand the branches of the tree (**project name > src**).

- 1-1-2. Double-click the source file to open it in the editor window.

Alternatively, you can right-click the source file name and select **Open**.

The **Open With** option provides access to other editors, including those outside the SDK or SDSoc tool environment.

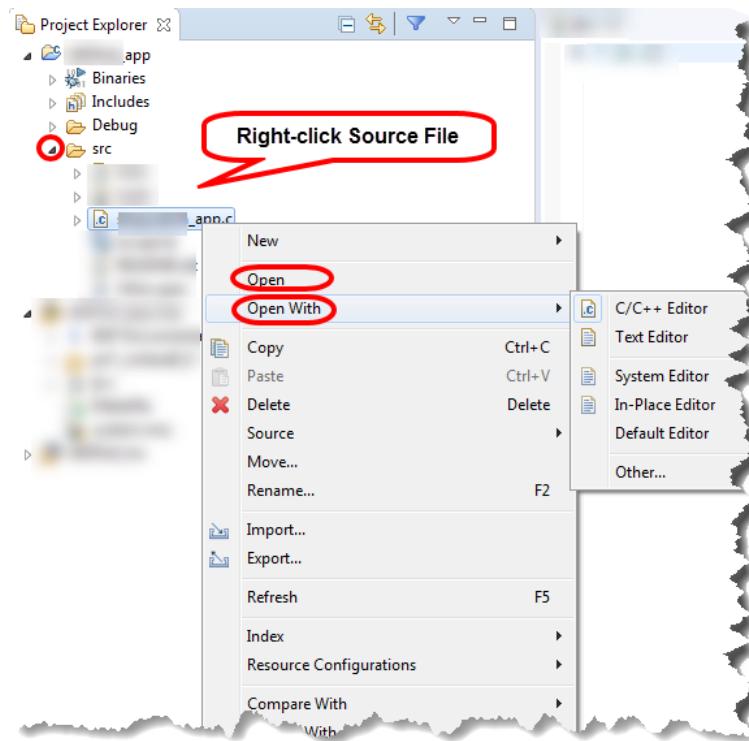


Figure 275: Opening a Source File via Right-Click

Finding Text in a Source File

1-1. Find *the text that you are looking for*.

The find/replace operation is accessible through both a click sequence and a keyboard shortcut.

- 1-1-1. Select **Edit > Find/Replace** or press **<Ctrl + F>**.

The Find/Replace dialog box opens.

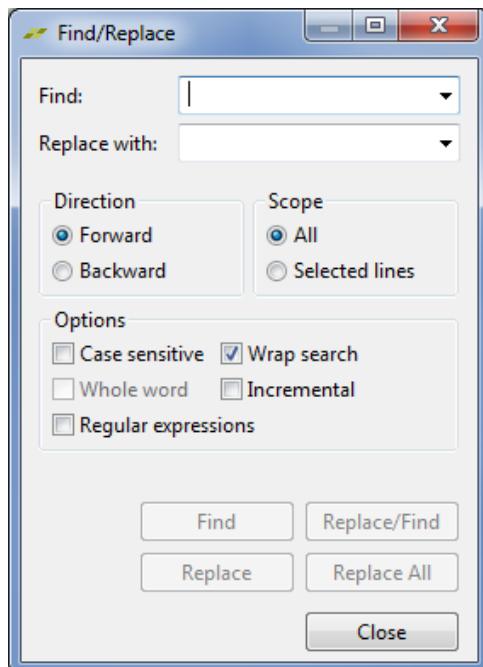


Figure 276: Default Find/Replace Dialog Box

- 1-1-2. Enter **the text that you are looking for** in the Find field.
- 1-1-3. Click **Find** or press **<Enter>** to find the next occurrence of *the text that you are looking for*.
- 1-1-4. Continue clicking **Find** or pressing **<Enter>** until you locate the specific instance that you are looking for.

With the **Wrap search** option enabled, the file is treated as a continuous loop and the find operation will jump to the next occurrence at the top of the file (when searching forwards) or at the bottom of the file (when searching backwards). A "ding" sound is made when the search wraps around.

If you are looking for text within a specific region of the code you would first highlight the region to perform the search in, then launch the Find/Replace function as described in this topic.

- 1-1-5. Click **Close** to close the Find/Replace dialog box.

Enabling Line Numbering in the SDK Text Editor

Line numbering is off by default, but it can be easily turned on.

1-1. Turn on line numbering in the text editor.

- 1-1-1. Open the source file in the editor if it is not already open.
- 1-1-2. Right-click in the left margin of the text editor.
- 1-1-3. Select **Show Line Numbers**.

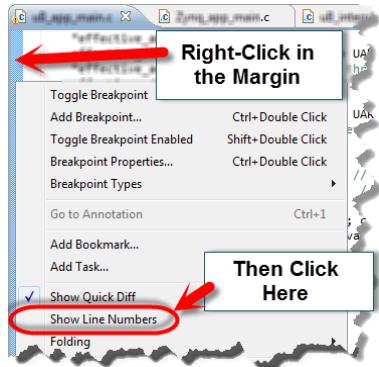


Figure 277: Enabling Line Numbering

Saving and Compiling an Application

1-1. Save and compile your open source file.

- 1-1-1. Press <Ctrl + S>, click the **Save** icon (💾) or the **Save All** icon (💾), or select **File > Save**.

Each time a file is saved, it is automatically rebuilt. You can monitor the behavior in the console view. Alternately, you can force all sources to be rebuilt by selecting **Project > Build All**.

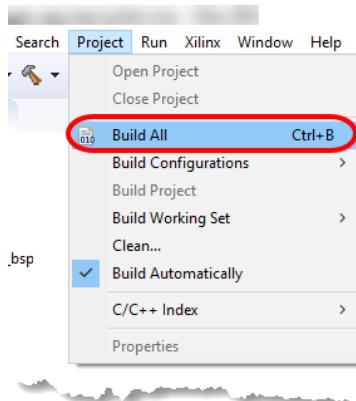


Figure 278: Selecting Build All

Compiling All Projects

1-1. Build all of the projects.

- 1-1-1. Select **Project > Build All** to force the building of all applications.

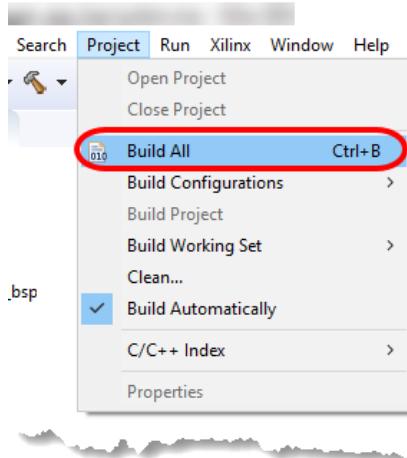


Figure 279: Selecting Build All

Customizing the Linker Script

The linker script controls where various pieces of the software reside in physical memory and how the physical memory is partitioned for use by the application.

1-1. Create a custom linker script for a C/C++ application.

- 1-1-1. Right-click your application project name.
- 1-1-2. Select **Generate Linker Script**.

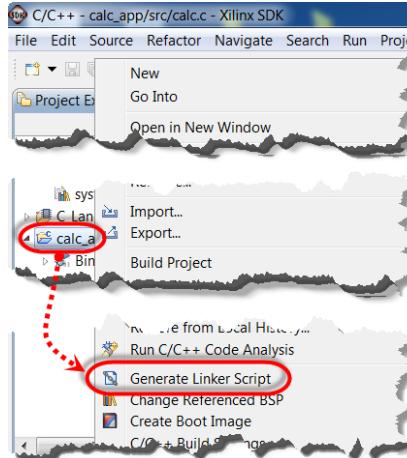


Figure 280: Accessing the Generate Linker Script Capability

The Generate Linker Script dialog box appears.

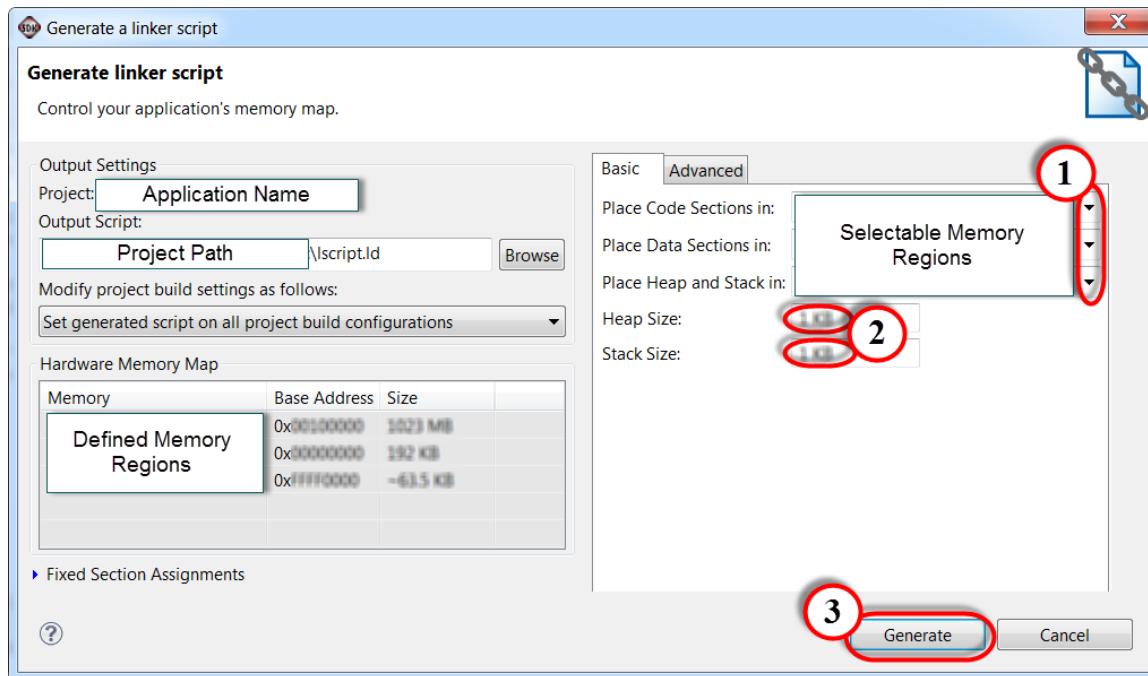


Figure 281: Common Sections of the Generate Linker Script Dialog Box

From here you can select where your code, data, and heap/stack sections reside in physical memory, as well as assigning how much space is allocated to your heap and stack.

Programming the Device from SDK

1-1. Program the device.

The bitstream resides within the SDK workspace as it was imported when the hardware platform was created. Before proceeding, make sure that the board is powered ON and the boot pins are configured in JTAG mode.

1-1-1. Select **Xilinx > Program FPGA** or click the icon.

The Program FPGA dialog box opens.

The default bitstream is the bitstream that is part of the collection of files that was exported from the Vivado Design Suite.

The BMM/MMI file is used to describe how block RAM memory clusters are loaded with instruction and/or data information that is contained as part of the bitstream. The BMM format is used in older versions of the tool, while the MMI format is used in newer versions of the tool. Typically, Zynq SoC-only designs do not require a BMM/MMI file, and MicroBlaze processor designs do require their use.

ELF files, under the Software Configuration section of the dialog box, are also related to BMM/MMI usage. If no BMM/MMI file is specified, then this section will remain empty.

- 1-1-2.** Keep all default settings and click **Program** to program the programmable logic portion of the device.

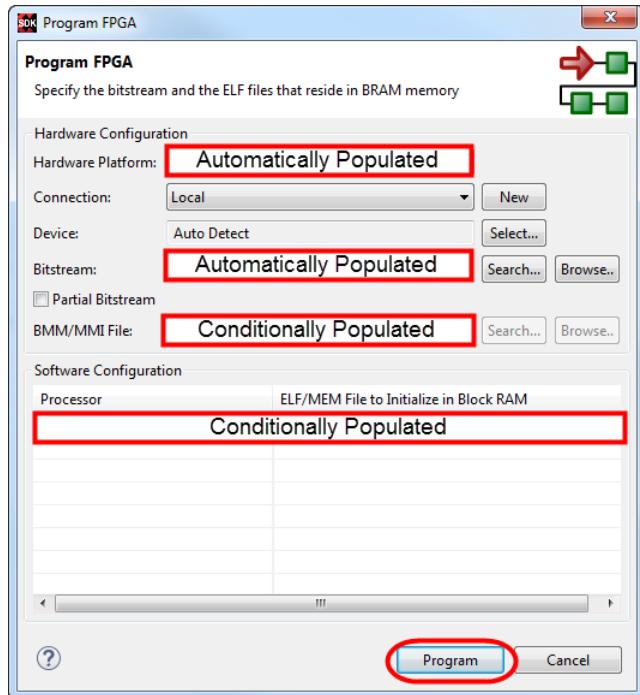


Figure 282: Programming the FPGA

It will take about a minute to configure the programmable logic. A progress bar will appear to show how far along the programming process is. Typically, the progress bar will pause near the halfway mark for a few seconds before completing the configuration. The result of the FPGA configuration can be viewed in the SDK Log tab at the bottom of the IDE.

Creating a Boot Image File

You will create a boot image file that contains the Zynq All Programmable SoC FSBL, the bit file for programmable logic (PL) configuration, and the application.

1-1. Create a boot image (MCS or BIN) for the *your application* application.

- 1-1-1. Right-click the software application in the Project Explorer tab (1).
- 1-1-2. Select **Create Boot Image** (2).

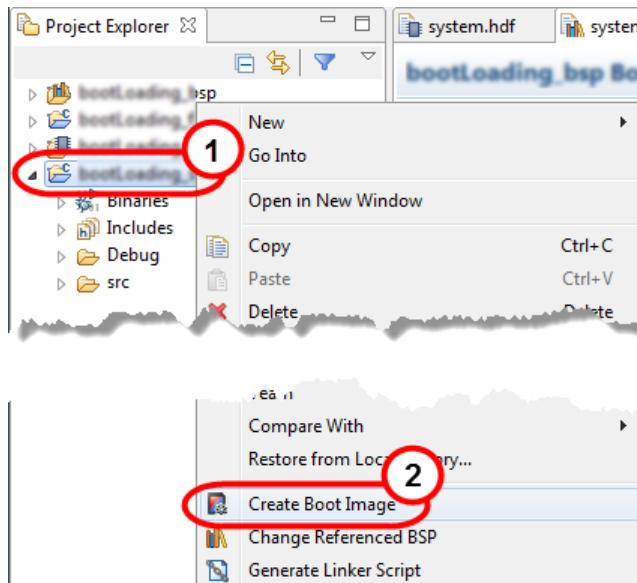


Figure 283: Selecting Create Boot Image

The Create Zynq Boot Image dialog box opens. You will find that the BIF file path has been selected automatically. Also in the Boot image partitions section, you will see the ***** files.

- 1-1-3. Make sure that the Output BIF file path field points to the *your workspace location\your application\bootimage* directory, where <TargetBoard> is zc702 for ZC702 board and zed for the ZedBoard.
- 1-1-4. Make sure that the *Output path* file name is **MCS or BIN**.

Note: To boot from Flash, the MCS file is required. If you want to boot from the SD card, the BIN file is required.

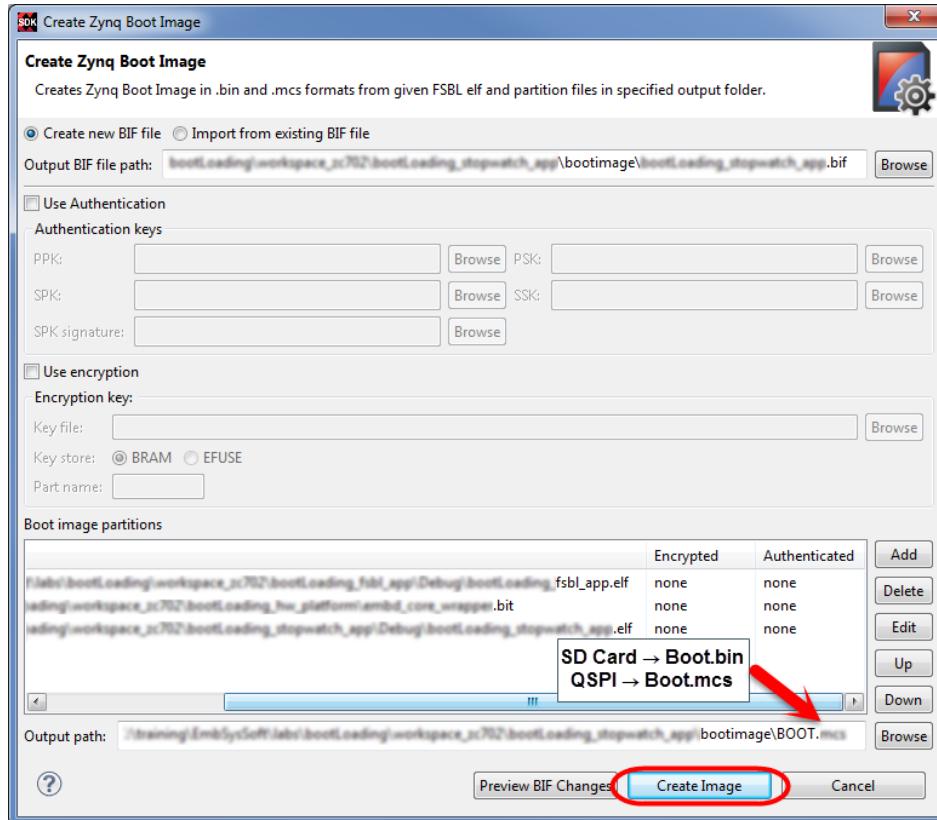


Figure 284: Create Zynq Boot Image Dialog Box

1-2. Create the boot image and view the generated file.

- 1-2-1. Click **Create Image**.
- 1-2-2. Expand the **your application** folder in the Project Explorer tab.

Note that the *bootimage* folder has now been created.

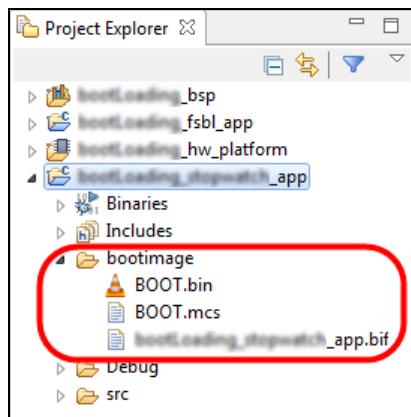


Figure 285: bootimage Folder Created

The *BOOT.mcs* file is used for programming the Flash.

The Intel MCS-86 Hexadecimal Object (.mcs) record format has a 9-character (4-field) prefix that defines the start of the record, byte count, load address, and record type, as well as a 2-character checksum suffix. This is File Format Code 88.

- 1-2-3. Double-click **your application.bif** in the *bootimage* folder.
- 1-2-4. Review the content.

Configuring a Local Repository

Local repositories are convenient mechanisms for [CD: fill in summary and add variables]

1-1. Add a local repository to the SDK environment.

- 1-1-1. Select **Xilinx Tools > Repositories**.

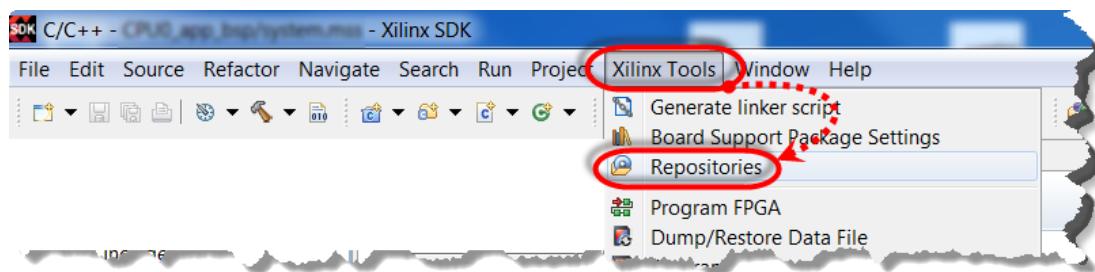


Figure 286: Selecting Repositories

The Repository Preferences dialog box opens.

- 1-1-2. Next to the Local repositories text box, click **New**.

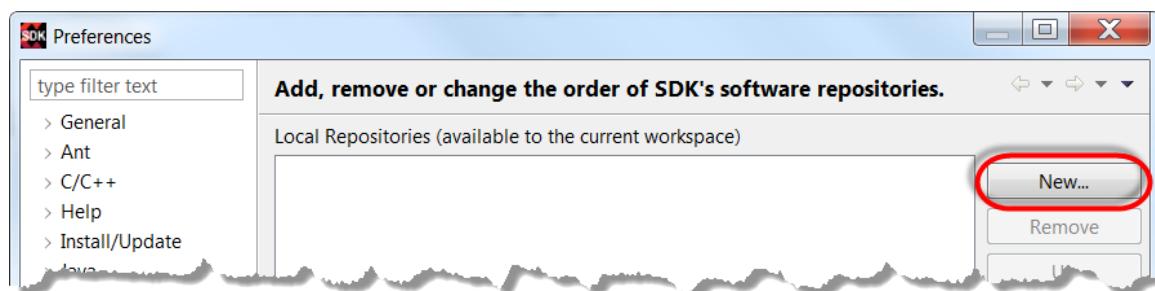


Figure 287: Creating a New Repository

- 1-1-3. Browse to the location of the local repository.
- 1-1-4. Select the directory for the repository.

1-1-5. Click **Apply**.

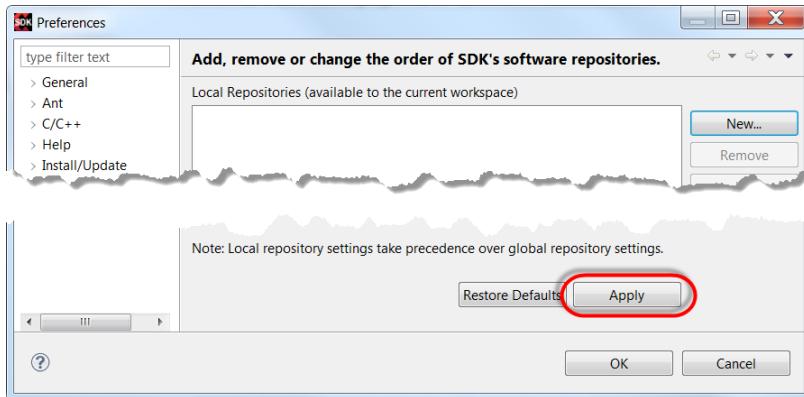


Figure 288: Applying Selections

1-1-6. Click **OK**.

Setting Up a Run Configuration

A Run configuration defines how you want the system to work when running an application. While there are a significant number of switches and options, the most common are shown below.

1-1. Set up a Run configuration for your *application project name*.

- 1-1-1. Right-click **your application project name** from the Project Explorer pane (1).
- 1-1-2. Select **Run As** from the context menu (2).
- 1-1-3. Select **Run Configurations** (3).

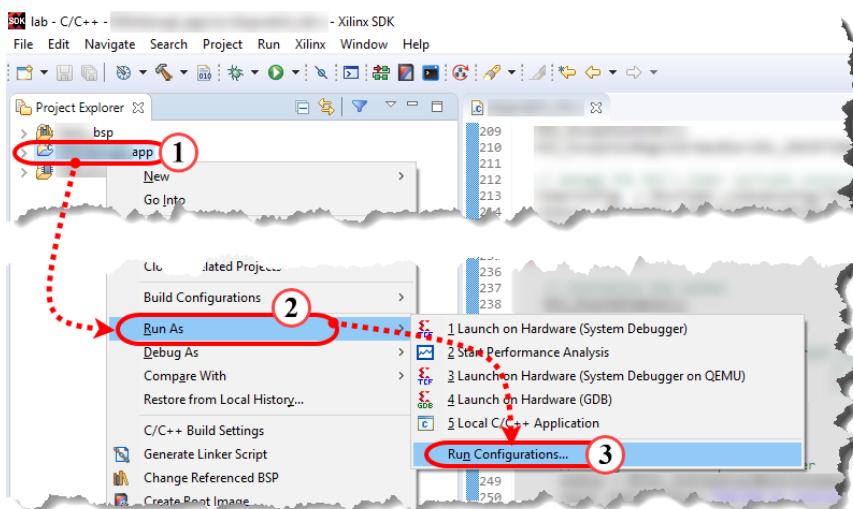


Figure 289: Creating a Run Configuration for an Application

The Run Configurations window opens.

- 1-1-4.** Double-click **Xilinx C/C++ application (System Debugger)** to create a launch configuration (1).

Alternatively, you can select **Xilinx C/C++ application (System Debugger)** and click the **New** configuration button.

If this is the first debug configuration being created for the project, a welcome type dialog opens. If one or more configurations exist, then the last open configuration will be displayed. In either case, a new configuration can always be added. Existing configurations are shown in the left pane and can be selected for debugging.

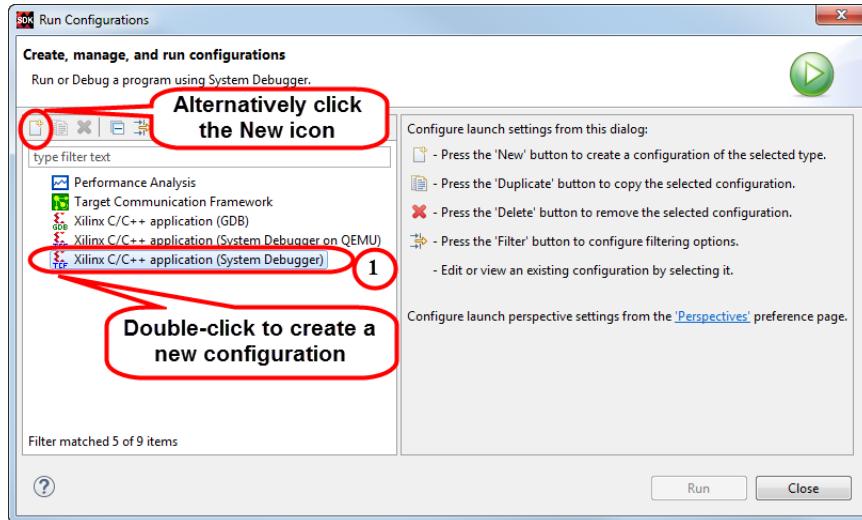


Figure 290: Creating a New System Run/Debug Configuration

A new Run configuration is created named *System Debugger using Debug_your application project name.elf on Local* (2).

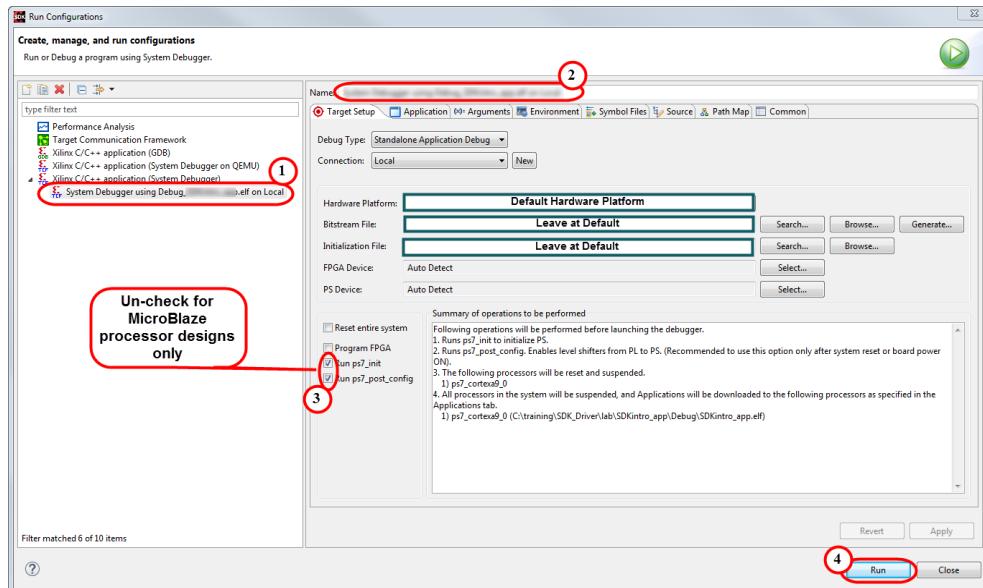


Figure 291: Run Configurations Dialog Box (Zynq SoC)

The new configuration will appear with other existing configurations and have the name of your application. You will also note that a number of fields are automatically filled in for you using the name of your application as the basis (that is, if your application is named "XYZ", then the Name field will be populated with XYZ Debug.) Most users will leave the other settings at their default.

- 1-1-5. Deselect the **Run ps7_init** and **Run ps7_post_config** options (3) if you are targeting a MicroBlaze processor design on the Zynq SoC.

This is to prevent the tools from automatically trying to configure the underlying Zynq SoC platform, even though a MicroBlaze processor is targeted.

- 1-1-6. Click **Run** to close the dialog box and launch the software application on the hardware evaluation board (4).

Setting Up a Run Configuration (System Debugger)

- 1-1. **Create a Run configuration. Run configurations associate an ELF object file to a target for execution. In this case, the target is a hardware board accessed over a network TCP/IP connection.**

- 1-1-1. Click the **C/C++** perspective (top right) to return to the original perspective.
- 1-1-2. Right-click **your application project name** from the Project Explorer pane.
- 1-1-3. Select **Run As > Run Configurations**.

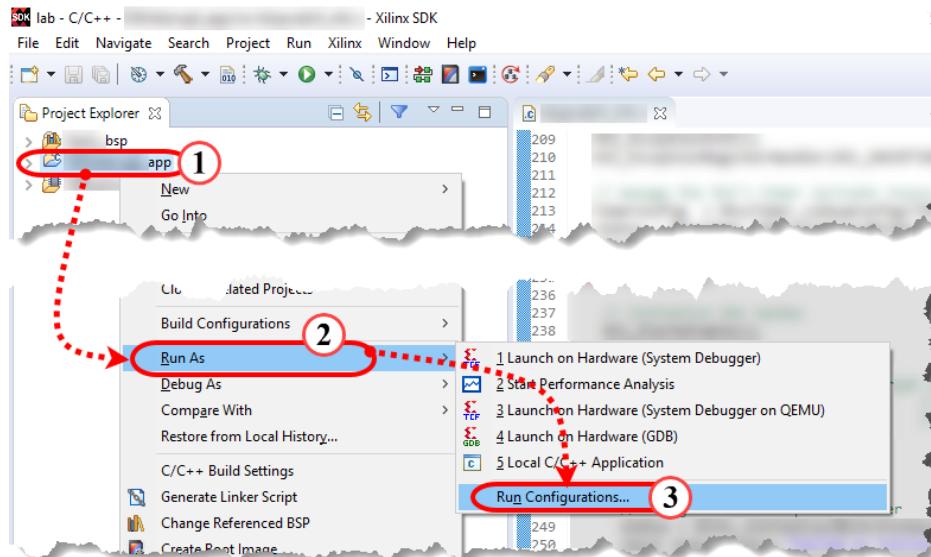


Figure 292: Selecting Run Configurations

1-2. Configure the debug type and host connection.

- 1-2-1. Double-click **Xilinx C/C++ application (System Debugger)** to create a launch configuration.
- 1-2-2. Select **Linux Application Debug** from the Debug Type drop-down list.
- 1-2-3. Click **New** next to the Connection drop-down list.
- 1-2-4. Enter **ZynqBoard** in the Target Name field.
- 1-2-5. Enter **the IP address of the board** in the Host field.
- 1-2-6. Enter **1534** in the Port field.

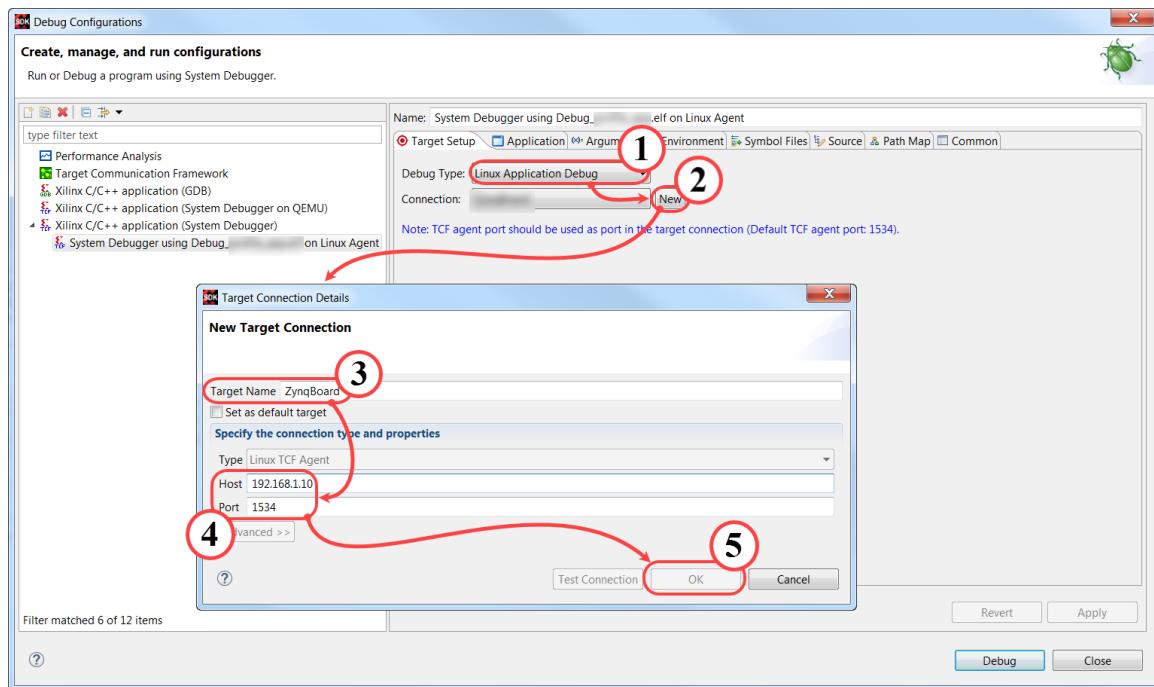


Figure 293: Selecting the Debug Type and Connection

- 1-2-7. Click **OK**.

1-3. Select the application and remote file path to copy the ELF file to the board.

1-3-1. Select the **Application** tab.

1-3-2. Click **Browse** next to the Local File Path field.

1-3-3. **[Windows users]:** Select the local file path to be `C:\training\<the topic cluster name>\lab\your application\Debug\your application.elf`.

[Linux users]: Select the local file path to be `/home/xilinx/training/<the topic cluster name>/lab/your application/Debug/your application.elf`.

1-3-4. Enter `/tmp/your application.elf` in the Remote File Path field.

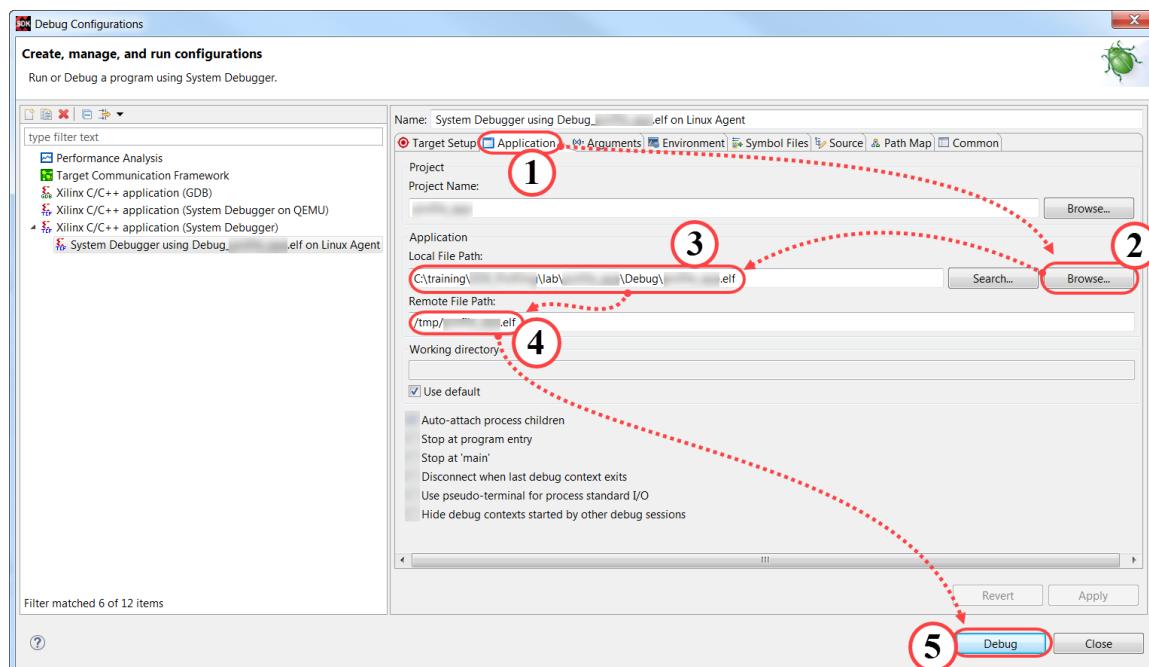


Figure 294: Selecting the Local File Path and Remote File Path

1-3-5. Click **Run**.

Running with a Default Configuration

Configurations are a very powerful ally in setting up a run for a very specific set of criteria. Often, however, the default configuration settings are adequate and there is no need to click through a number of dialogs just to run the program.

1-1. Run **your application** with the default configuration settings.

- 1-1-1. Right-click **your application** in the Project Explorer to open the context menu.
- 1-1-2. Select **Run As > Launch on Hardware (System Debugger)** to immediately launch the application on the hardware.

Note: Selecting **Run As > Launch on Hardware (GDB)** will also work; however, this is a deprecated flow.

If an application is already running, a warning message will appear asking you to terminate the previous session. If you receive this message, click **Yes** to terminate the running application and run the new scenario.

The application runs.

--- Alternate Method ---

Select **your application** in the Project Explorer tab.

Once the application project is selected, you can launch the run by:

- o Use the menu bar to select **Run > Run**.
 - Then select one of the options from the **Run As** window (this window will pop up only if there is no Run configuration).
- o Press <**Ctrl + F11**>.
- o Click the **Run** icon (

Opening the XSCT Console

The XSCT console allows you to enter Tcl commands directly into SDK. This can be particularly useful when building a design up to a particular point or performing repetitive processes.

1-1. Open the XSCT console so that you can enter Tcl-based commands directly into the Xilinx SDK tool.

1-1-1. Select the **Window** entry in the menu bar to access the pull-down menu (1).

1-1-2. Select **Show View** to see the list of commonly opened views (2).

SDK offers too many views to be contained in this list. Extensions and less frequently used views are pushed off to another layer of selections.

1-1-3. Select **Other** to open the selections pop-up menu (3).

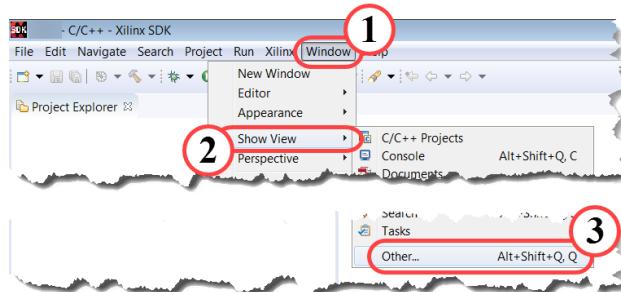


Figure 295: Accessing the Available Views

The Show View dialog box opens.

1-1-4. Scroll to the last entry in the menu: *Xilinx*.

1-1-5. Expand **Xilinx** (4).

1-1-6. Select **XSCT Console** (5).

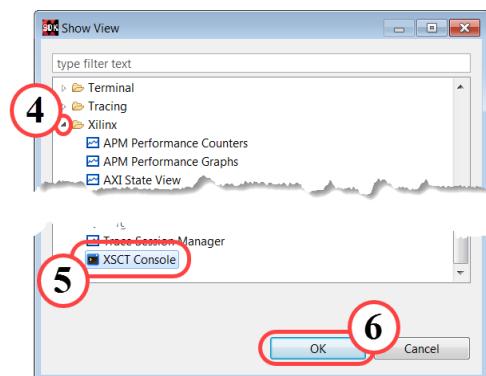


Figure 296: Selecting the XSCT Consolve View

1-1-7. Click **OK** to open the XSCT console (6).

Debugging

Debugging covers a broad collection of topics including:

- Controlling execution (single stepping, running to breakpoints, etc.)
- Breakpoint management
- Memory tracing

The following topics provide a brief introduction to these concepts and techniques.

In This Section

Configuring the Target Connection	232
Setting Up a GDB Debug Configuration	234
Enabling Instruction Stepping	238
Launching an Existing Debug Configuration.....	239
Launching an Existing Run Configuration.....	240
Setting Up a System Debugger Debug Configuration	240
Setting Up a Debug Configuration (System Debugger – Linux)	243
Creating and Running the Abbreviated Debug Configuration.....	248
Launching System Debug of a Software Application on Hardware.....	248
Enabling Cross Trinngering in a GDB Debug Configuration	250
Controlling Execution	254
Managing Breakpoints.....	256
Monitoring Variables	262
Managing Expressions	263
Changing the Value of a Variable	264
Using Expressions	266
Setting Up the Linux Console for Debugging	268

Configuring the Target Connection

1-1. Configure the target connection in the integrated development environment.

- 1-1-1. Return to the SDK or SDSoc tool.
- 1-1-2. Using the Target Connections view, expand **Linux TCF Agent**.

Optional: If the Target Connections view is not visible, then select **Window > Show View > Other > Xilinx > Target Connections** and click **OK**.

- 1-1-3. Select and right-click **Linux Agent [default]**.
- 1-1-4. Select **Edit** from the context menu.

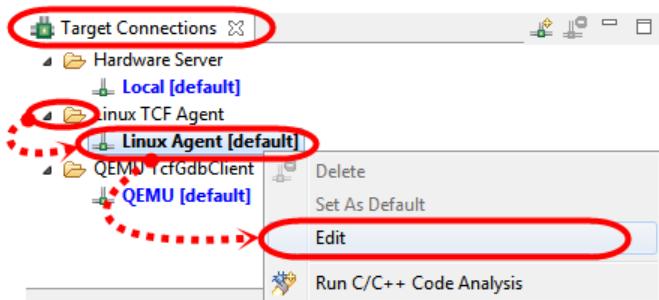


Figure 297: Accessing Linux Agent Target Connection

The Target Connection Details dialog box opens.

- 1-1-5. Change the host address to **the IP address of the board**.

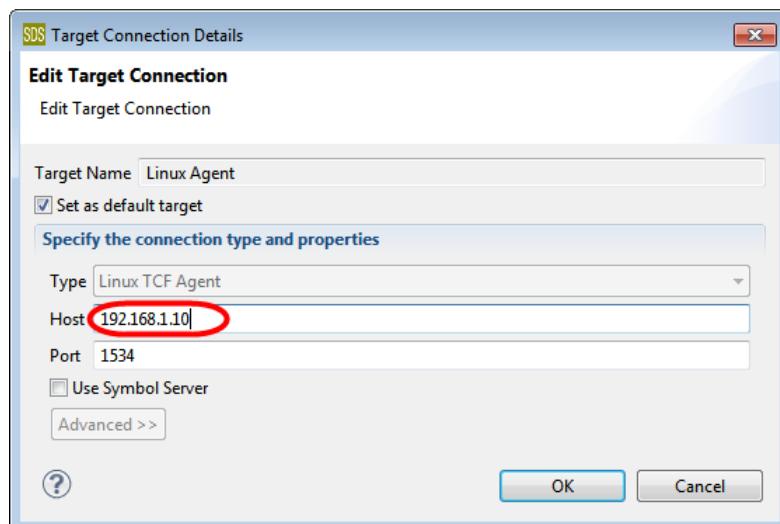


Figure 298: Example Host Connection with IP Address Set to 192.168.1.10

- 1-1-6. Click **OK** to set the connection.

Setting Up a GDB Debug Configuration

A Debug configuration defines how you want the system to work when performing a debug operation. Typically a debug operation switches to the debug perspective. While there are a significant number of switches and options, the most common are shown below.

1-1. Set up a debug configuration for a specific application project.

- 1-1-1. From the Project Explorer pane, right-click the application project that you wish to build the Debug Configuration for (1).
- 1-1-2. Select **Debug As** (2).
- 1-1-3. Select **Debug Configuration** (3).

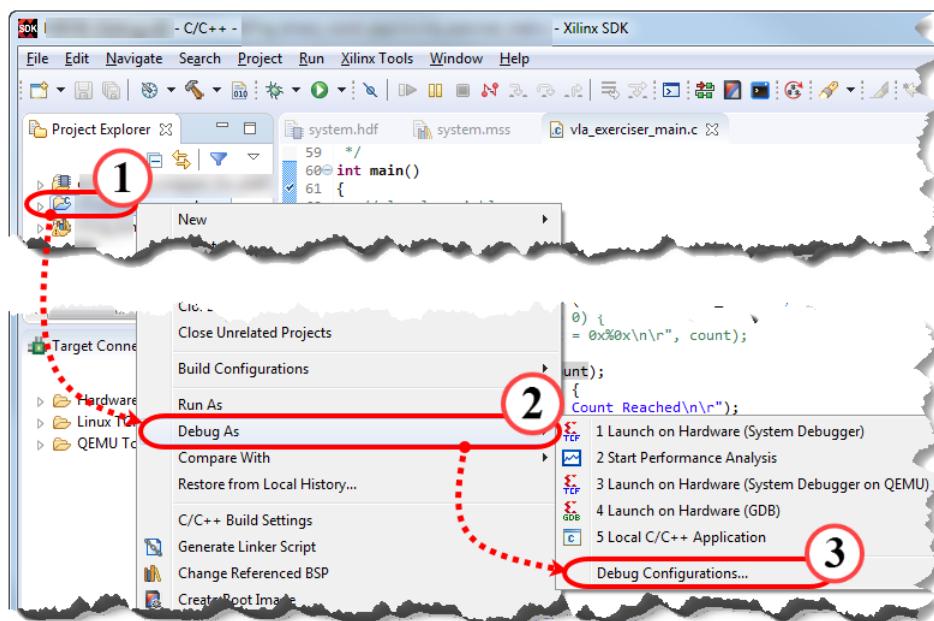


Figure 299: Accessing the Debug Configuration Dialog Box

The Debug Configurations dialog box opens.

- 1-1-4. Select **Xilinx C/C++ application (GDB)** since you will be debugging this type of system (1).
- 1-1-5. Click the **Create New Configuration** icon to create the new configuration for your application (2).

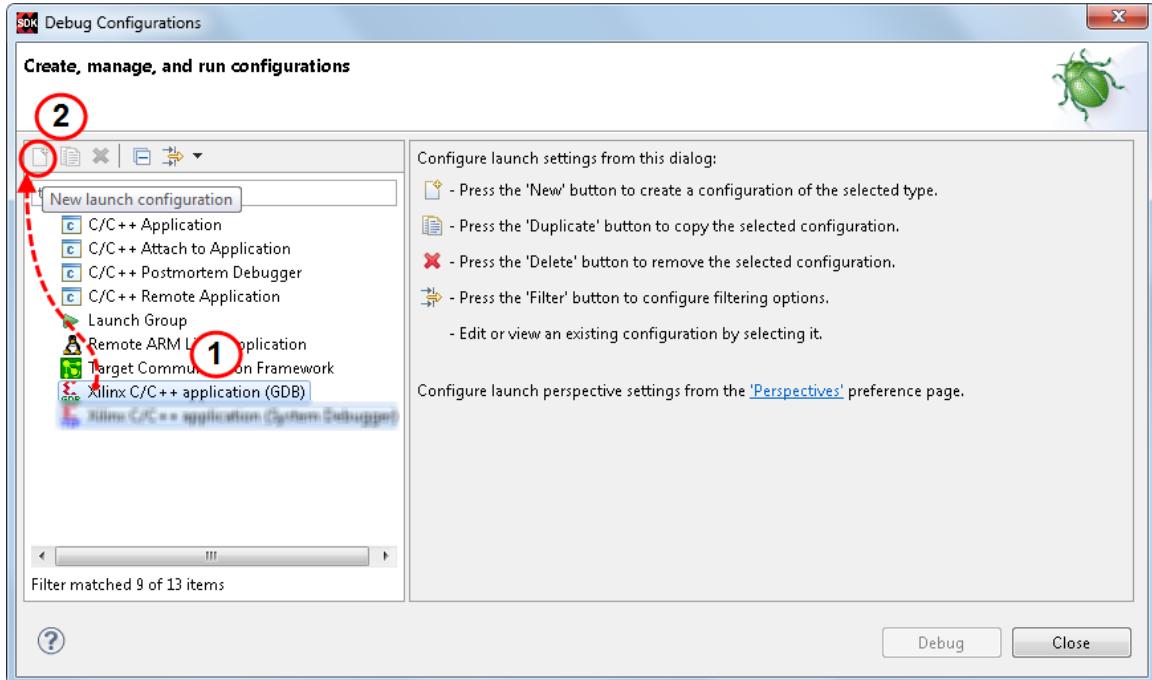


Figure 300: Creating a New Debug Configuration

A new Debug Configuration is created. The figure above depicts a SDK workspace that does not yet have any debug or run configurations defined. If other configurations did exist, or after the creation the first configuration, the configuration window will appear as below. Note the default configuration name and other parameters that are automatically filled in. In most cases, you just need to click the Debug button to begin the session. This Debug Configuration menu is useful when you want to later change debug parameters.

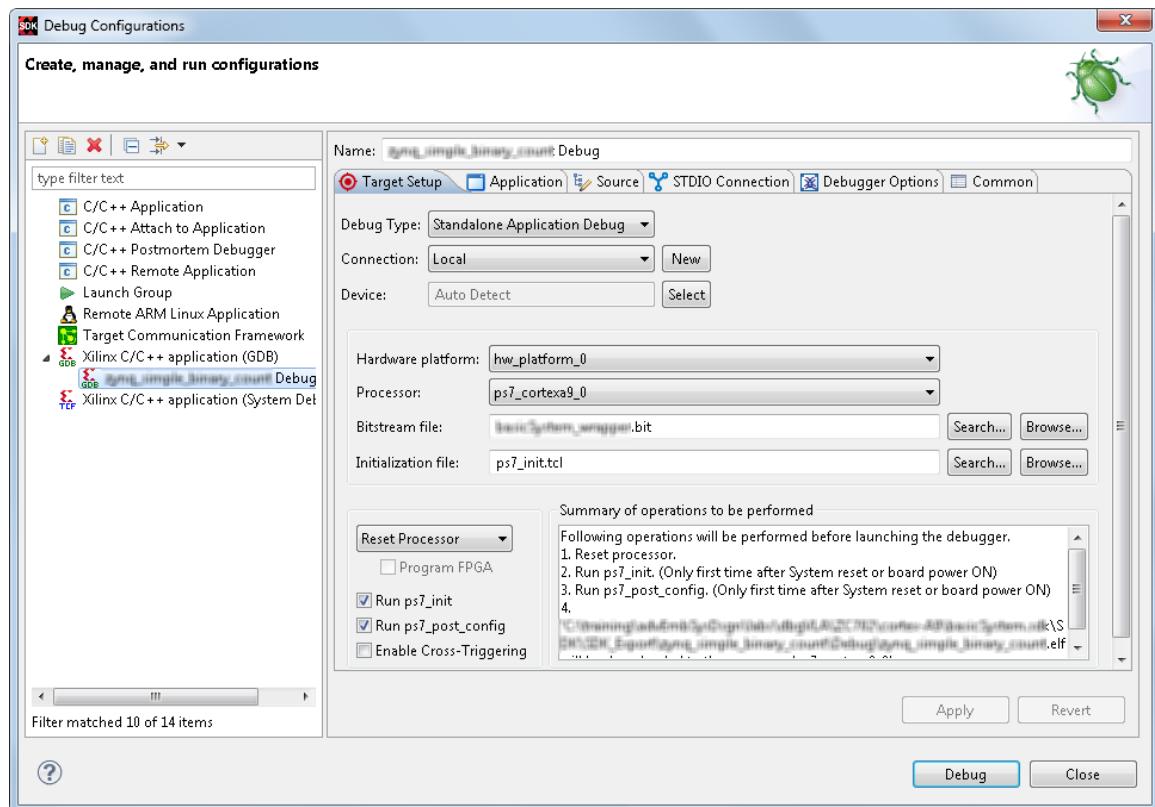


Figure 301: Creating the New Debug Configuration

The new configuration will appear with other existing configurations and have the name of your application. You will also note that a number of fields are automatically filled in for you using the name of your application as the basis (that is, if your application is named "XYZ", then the Name field will be populated with XYZ Debug, and the C/C++ Application field will be populated with Debug\XYZ.elf).

1-1-6. Verify in the **Target Setup** tab to configure how the device is to be initialized.

Typical systems that use "Reset Entire System" while multi-processor are:

- o Single Cortex A-9 processor systems
- o The Cortex A-9 processor designated as the master processor in a system

Typical systems that use "Reset Processor Only" are:

- o Any slave processor in a multi-processor system

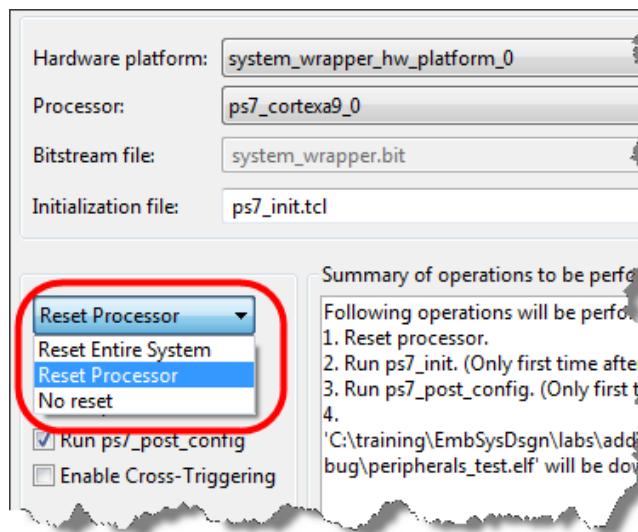


Figure 302: Selecting Reset Behavior

- 1-1-7.** Select the **STDIO Connection** tab to use the console as your serial port terminal.

Note that if you enable this, the serial communication that would normally be routed through an RS-232 UART is actually routed through a JTAG UART.

DO NOT ENABLE THIS CAPABILITY IF YOUR INTENTION IS TO TEST THE RS-232 UART. You can use the SDK Terminal tab for this purpose.

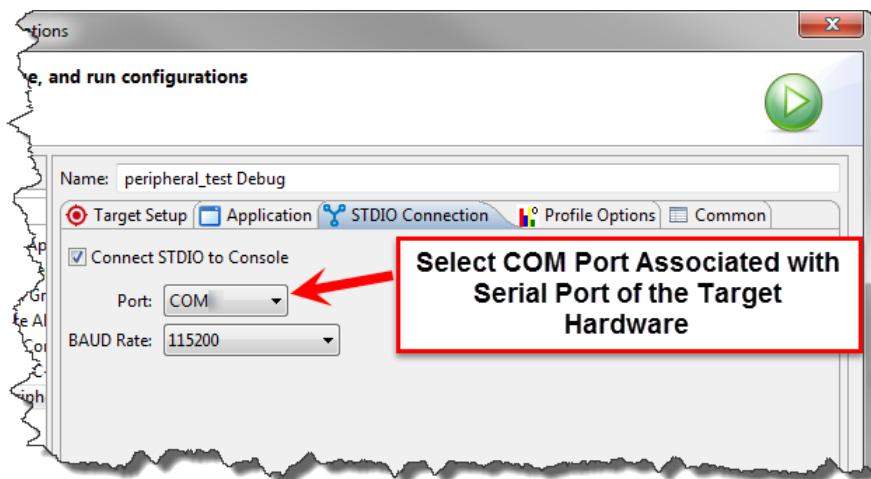


Figure 303: Directing the STDIO to the Console

If you are unsure on how to locate the proper COM port, refer to the "Configuring the SDK Terminal" section or "Configuring Tera Term" section.

Most users will leave the other settings at their default.

- 1-1-8. Click **Apply** to save the setting and leave the dialog box open.
- 1-1-9. Click **Debug** to close the window and launch the debugging session.

If the Confirm Perspective Switch dialog box appears, click **Yes**.

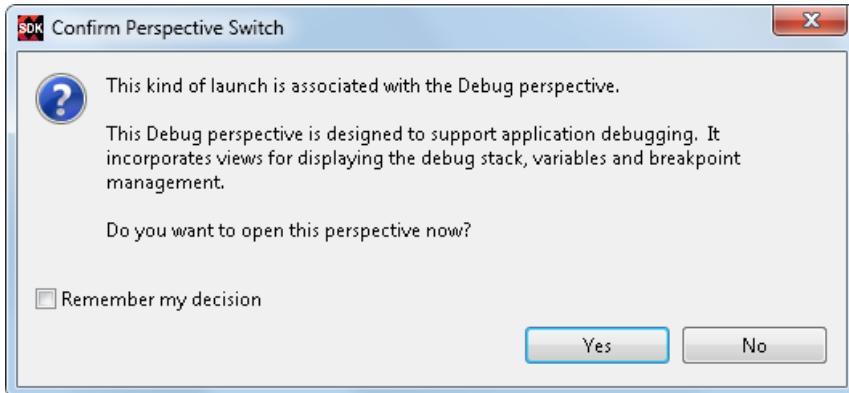


Figure 304: Confirming Switch of Perspective

The Debug Perspective view opens.

Enabling Instruction

Stepping through a high-level language such as C source code is an easier and more intuitive way to see the functionality of code while debugging. However, the high-level language is an abstraction of the actual instructions being executed on the hardware. Instruction stepping removes the abstraction and allows stepping through the actual assembly instructions being executed on the hardware. This instruction stepping functionality is necessary to have when extremely low-level debugging must be done.

Instruction stepping can be performed regardless of the level of compiler optimization. Note that the C-level instructions may not execute in the expected order as the underlying assembly code has been rearranged by the optimizer. Therefore, it is recommended that optimization level zero (no optimization) be used when C source code is debugged.

1-1. Enable instruction stepping.



Figure 305: Instruction Stepping

- 1-1-1.** Click the **Instruction Stepping Mode** (i[▶]) icon from the toolbar.

This will open the Disassembly window and the next instruction to be executed will be highlighted.

Launching an Existing Debug Configuration

1-1. Launch an existing debug configuration.

- 1-1-1.** Click the down arrow next to the Debug Configurations icon (⚙) to select from the drop-down list (1).
- 1-1-2.** Select the appropriate existing debug configuration to launch a new debug session (2).

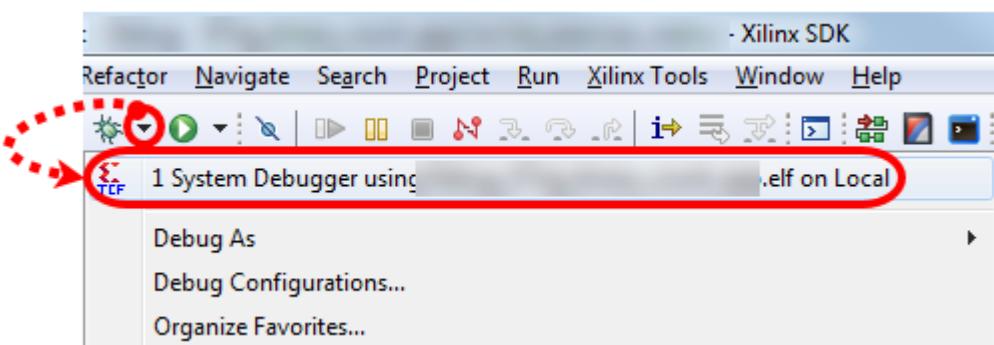


Figure 306: Launching an Existing Debug Configuration

Note: This will launch a debug session with the configuration that was created earlier. There may be several debug configurations that exist. Select the one that is appropriate for your lab.

- 1-1-3.** Click **Yes** if the Confirm Perspective Switch dialog opens and informs you that the perspective view is being changed (from C/C++ to Debug).

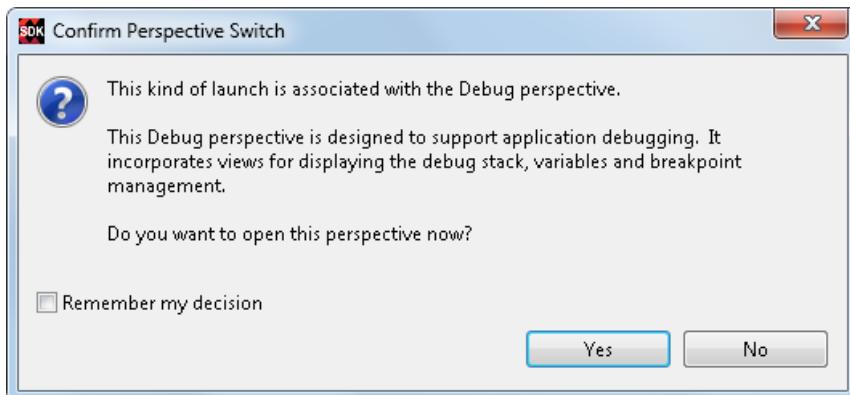


Figure 307: Confirming Switch of Perspective

Launching an Existing Run Configuration

1-1. Launch an existing Run configuration.

- 1-1-1. Click the down arrow next to the Run Configurations icon (1) to select from the drop-down list (1).
- 1-1-2. Select the appropriate existing run configuration to launch the application (2).

Note: This will launch the application with the configuration that was created earlier. There may be several run configurations that exist. Select the one that is appropriate for your lab.

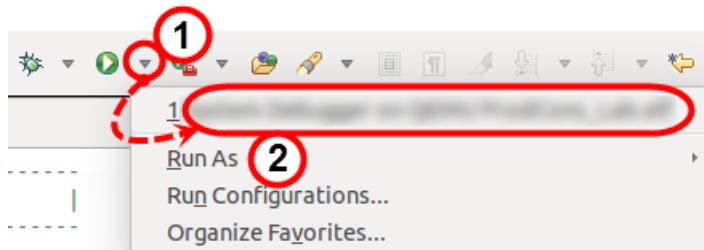


Figure 308: Launching an Existing Run Configuration

Setting Up a System Debugger Debug Configuration

A Debug configuration defines how you want the system to work when performing a debug operation and maps an ELF object file to a target for execution. Typically, a debug operation switches to the Debug perspective. While there are a significant number of switches and options, the most common are shown below.

1-1. Set up a debug configuration for a specific application project.

- 1-1-1. Right-click the application project that you want to build the Debug configuration for from the Project Explorer pane (1).
- 1-1-2. Select **Debug As** to open the menu of predefined configurations and the configuration manager (2).
- 1-1-3. Select **Debug Configurations** to view all the available debug options (3).

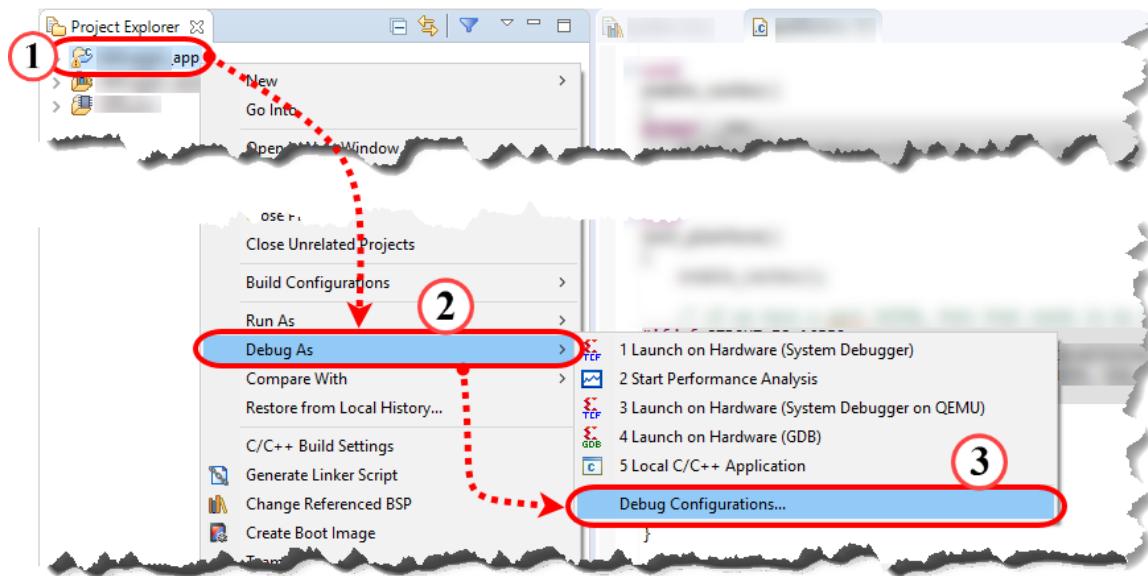


Figure 309: Creating a Debug Configuration

The Debug Configurations dialog box opens.

- 1-1-4. Select **Xilinx C/C++ application (System Debugger)** since you will be debugging this type of system (1).

GDB still works; however, it is considered deprecated for new designs.

- 1-1-5. Click the **Create New Configuration** icon to create the new configuration for your application (2).

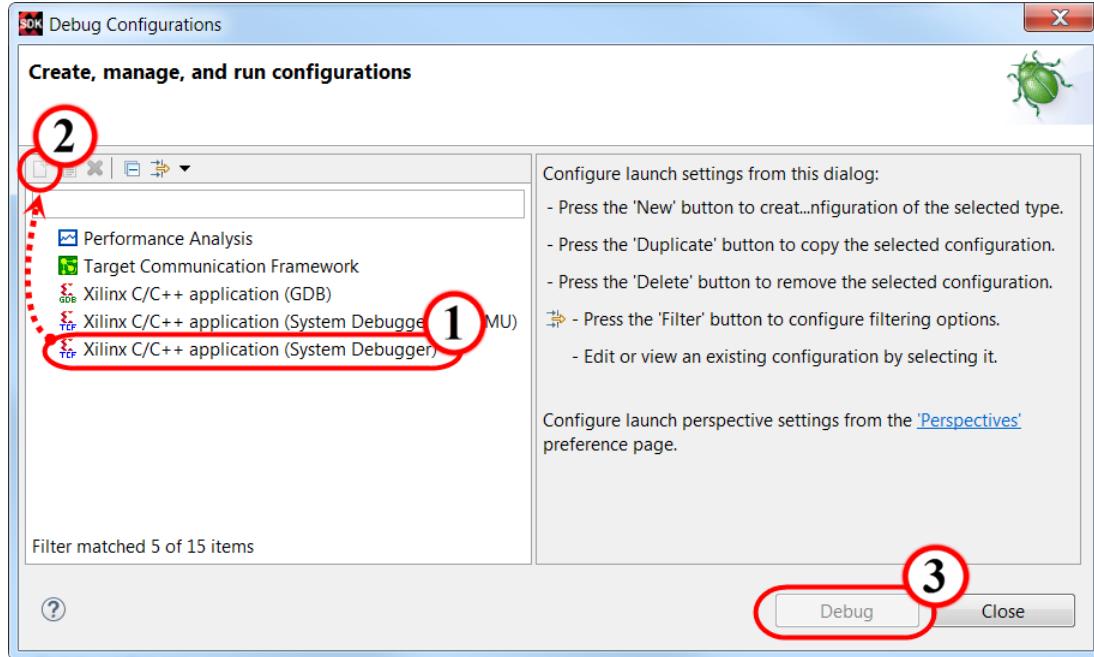


Figure 310: Creating a New Debug Configuration

A new Debug Configuration is created. The previous figure depicts an SDK workspace that does not yet have any debug or run configurations defined. If other configurations did exist, or after the creation of a first configuration, the new configuration will appear under the type of configuration.

Note the default configuration name and other parameters that are automatically filled in. In most cases, you just need to click the Debug button to begin the session. This Debug Configuration menu is useful when you want to later change debug parameters.

The new configuration will appear with other existing configurations and have the name of your application. You will also note that a number of fields are automatically filled in for you using the name of your application as the basis (that is, if your application is named "XYZ", then the Name field will be populated with XYZ Debug, and the C/C++ Application field will be populated with Debug\XYZ.elf).

- 1-1-6. Click **Debug** to close the window and launch the debugging session (3).

If the Confirm Perspective Switch dialog box appears, click **Yes**.

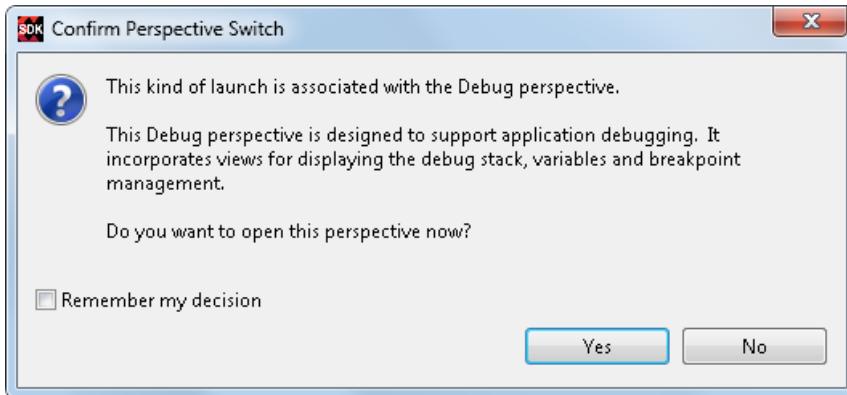


Figure 311: Confirming Switch of Perspective

The Debug Perspective view opens.

Setting Up a Debug Configuration (System Debugger – Linux)

Run and Debug configurations are application project objects that contain communication, hardware, and execution options for running an application on a hardware or emulation platform. The selections for *Run* and *Debug* are identical and only differ in that *Run* just executes the application and *Debug* opens a debug perspective and launches a debug program.

There are different types of Run and Debug configurations based on the operating system (or Standalone libraries) and the SDK download/debug tools that you want to use. Multiple configurations can be defined for any project. This facilitates enhanced development and debugging.

- 1-1. Create a new Linux Debug configuration. Debug and Run configurations associate an ELF object file to a target (typically a hardware board) for execution. In this case, the target is a hardware board accessed over the Ethernet TCP/IP connection.**

- 1-1-1. Click the C/C++ tab in the upper-right corner of the GUI to return to the C/C++ perspective, if not already in the C/C++ perspective.**

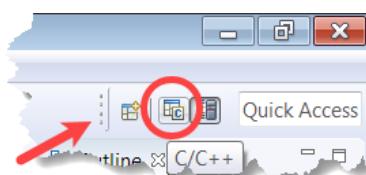


Figure 312: Changing Perspective

If the C/C++ perspective icon is not immediately visible, drag the vertical bar (pointed to by the arrow) to the left.

This brings back access to the software projects. It is accessed by first clicking the >> in the same location. If this tab is not available, you can also return to the perspective by selecting **Window > Open Perspective > Other > C/C++ (default)**.

- 1-1-2.** Right-click **your application** in the Project Explorer window (1).

- 1-1-3.** Select **Debug As** (2) > **Debug Configurations** (3).

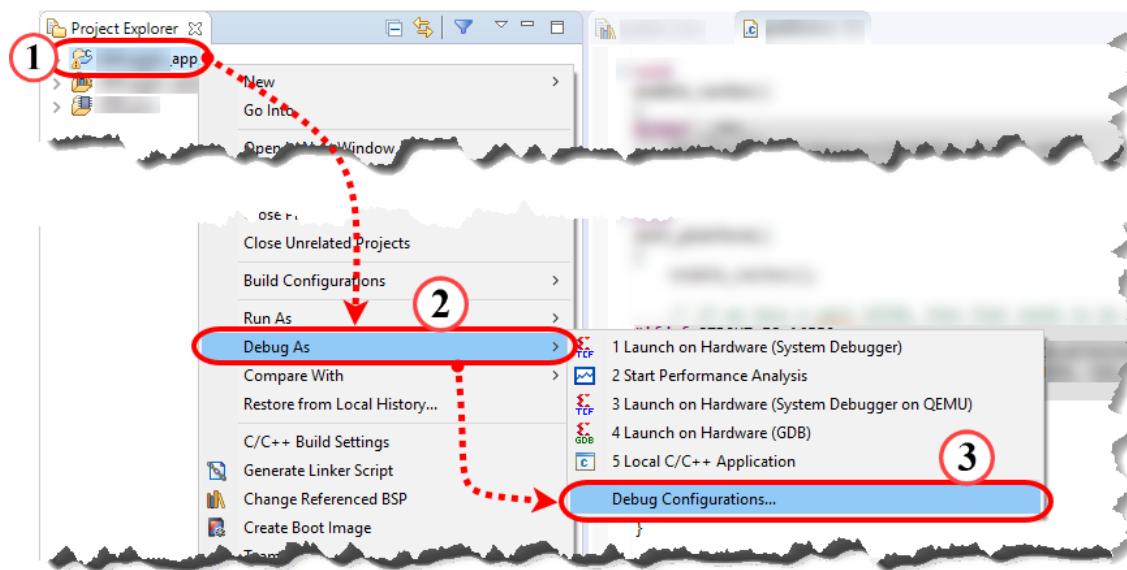


Figure 313: Selecting Debug Configurations

The Debug Configurations dialog box opens.

- 1-1-4.** Double-click **Xilinx C/C++ application (System Debugger)** to create a launch configuration (1).

Alternatively, you can select **Xilinx C/C++ application (System Debugger)** and click the **New** configuration button.

If this is the first debug configuration being created for the project, a welcome type dialog opens. If one or more configurations exist, then the last open configuration will be displayed. In either case, a new configuration can always be added. Existing configurations are shown in the left pane and can be selected for debugging.

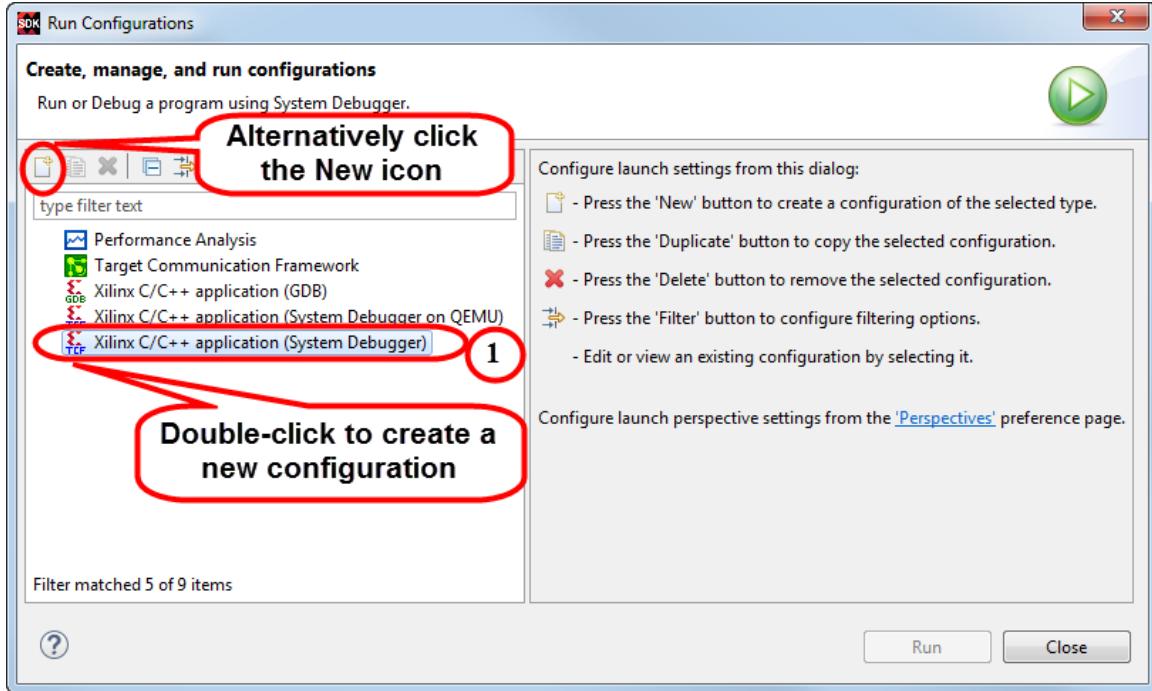


Figure 314: Creating a New System Run/Debug Configuration

1-2. Configure the debug type and host connection.

- 1-2-1. Select **Linux Application Debug** from the Debug Type drop-down list (1).

This will engage the proper debug tool to use.

- 1-2-2. Click **New** next to the Connection drop-down list to launch the Target Connection Details dialog box (2).

A new connection will be defined from the debugger to the hardware (or emulator) target.

- 1-2-3. Enter **ZynqBoard** in the Target Name field as the name of the connection (3).

- 1-2-4. Enter **the IP address of the board** in the Host field (4).

This is the IP address of the RSE connection that was set up earlier.

- 1-2-5. Confirm that the Port field is set to the default value of **1534**.

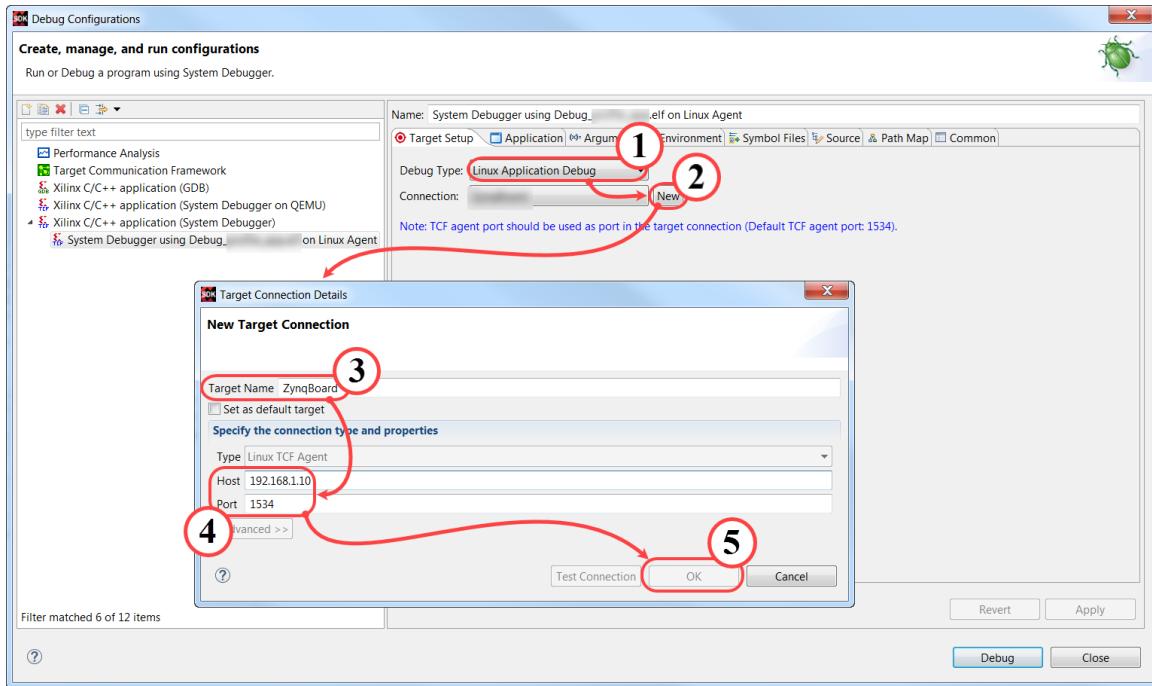
1-2-6. Click OK (5).

Figure 315: Selecting the Debug Type and Connection

1-3. Select the software application ELF file to debug and its file path on the remote target board where it is to be copied.

The actual software application debugging takes place on the Linux platform running on the target hardware, so it is necessary to put a copy of the ELF file on it.

1-3-1. Select the Application tab (1).

This is where the software application is selected and allows configuration of where the application is placed in the remote file system.

1-3-2. Click Browse next to the Local File Path field to select the software application ELF file (2).**1-3-3. [Windows users]: Navigate to and select the file C:\training\<the topic cluster name>\lab\your board\your processor\your application.elf (3).**

[Linux users]: Navigate to and select the file /home/xilinx/training/<the topic cluster name>/lab/your project name/Debug/your application.elf (3).

1-3-4. Enter /tmp/your application.elf in the Remote File Path field as the location on the target platform where the application ELF file will be copied (4).

This directory (/tmp) has user read/write/execute privileges whereas not all versions of Linux allow the user rwx from /mnt.

Another, and typically faster, way to fill in the fields is to select the project using the Browse button adjacent to the Project Name text field. This will allow you to quickly select any of the projects in the workspace. Once selected, the Local File Path field will be automatically filled in (if the ELF is available) as well as the remote file path.

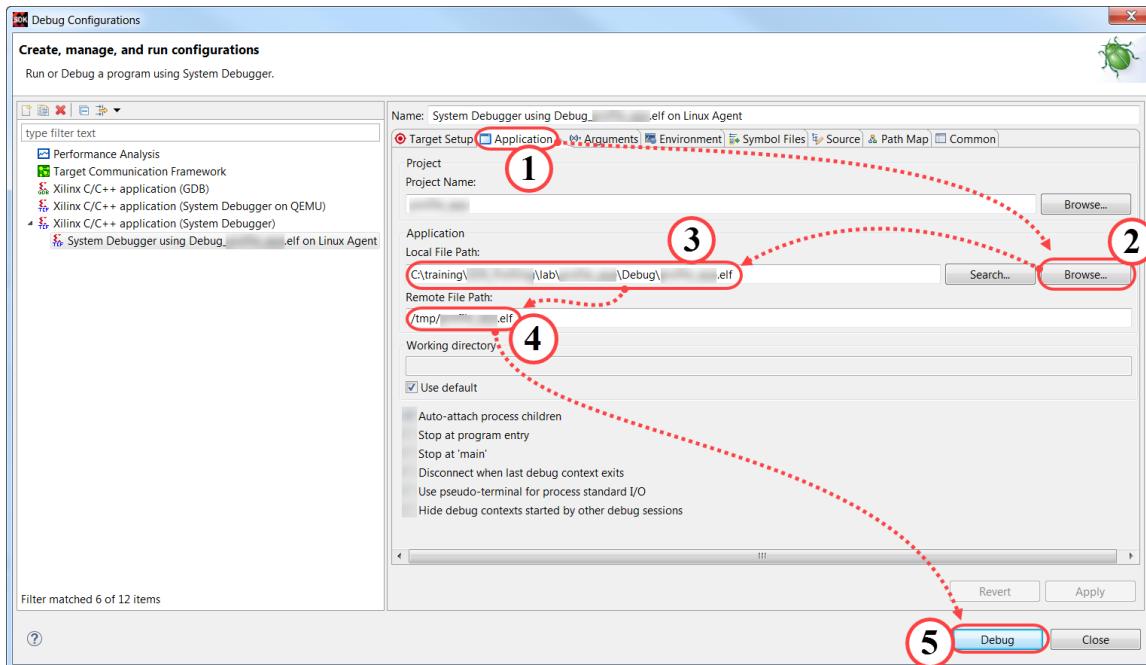


Figure 316: Selecting the Local File Path and Remote File Path

The remaining options will be accepted at their default values.

- 1-3-5. Click **Debug** (5).
- 1-3-6. Click **Yes** to confirm opening the Debug perspective view.

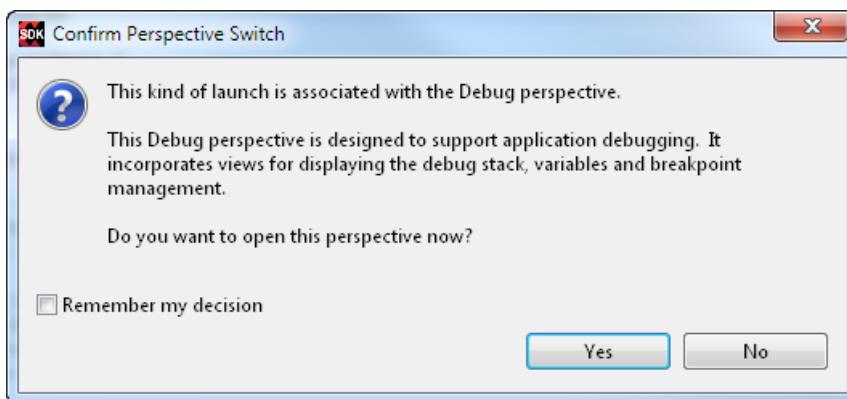


Figure 317: Confirming Perspective Switch

The Debug perspective opens.

Creating and Running the Abbreviated Debug Configuration

The process of setting up a full Debug configuration is very powerful and offers many options. Many times however the default options are sufficient and you just want to quickly run your application in debug mode.

1-1. Run the default Debug configuration for your application.

- 1-1-1. Right-click **your application** to open the context menu (1).
- 1-1-2. Select **Debug As** to see the various debugging options (2).
- 1-1-3. Select **Launch on Hardware (System Debugger)** to switch to the Debug perspective and begin the debugging session (3).

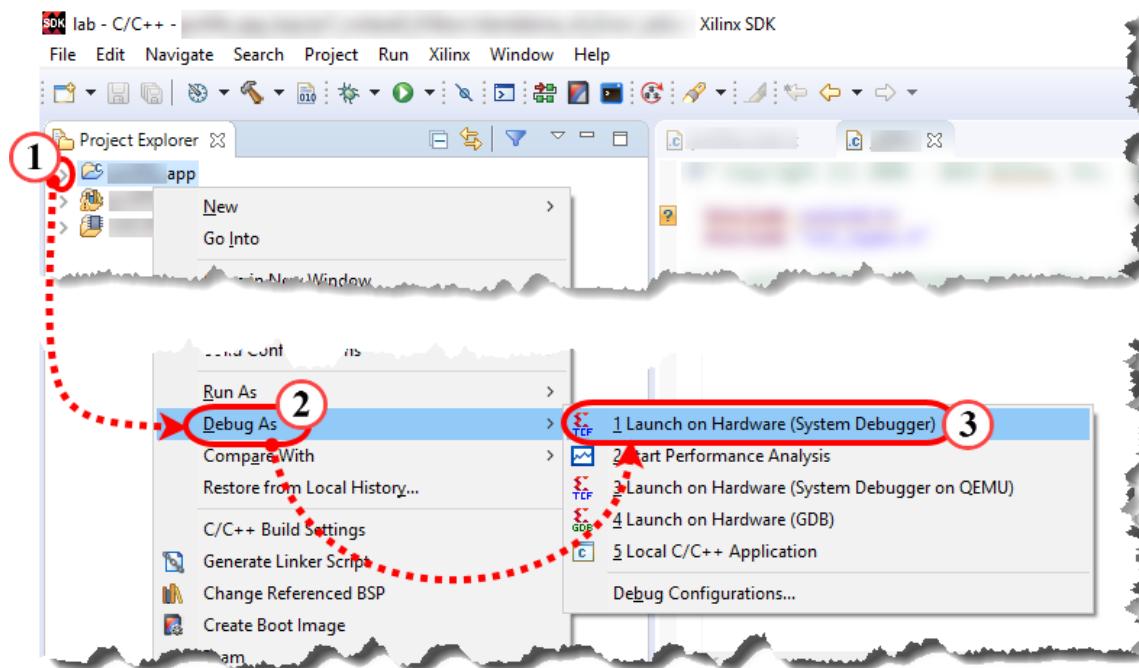


Figure 318: Launching the Debug Session with the Default Configuration

- 1-1-4. If a dialog box opens asking you to confirm the perspective switch, click **Yes**.

Launching System Debug of a Software Application on Hardware

A Debug configuration defines how you want the system to work when debugging an application. A Launch on Hardware (System Debugger) menu selection will start a debug session of the selected software application with a Debug configuration at default settings, saving you a few mouse clicks.

1-1. Launch and debug the software application on the hardware evaluation board.

- 1-1-1. Right-click the application project that you want to launch from the Project Explorer pane (1).
- 1-1-2. Select **Debug As** (2).
- 1-1-3. Select **Launch on Hardware (System Debugger)** (3).

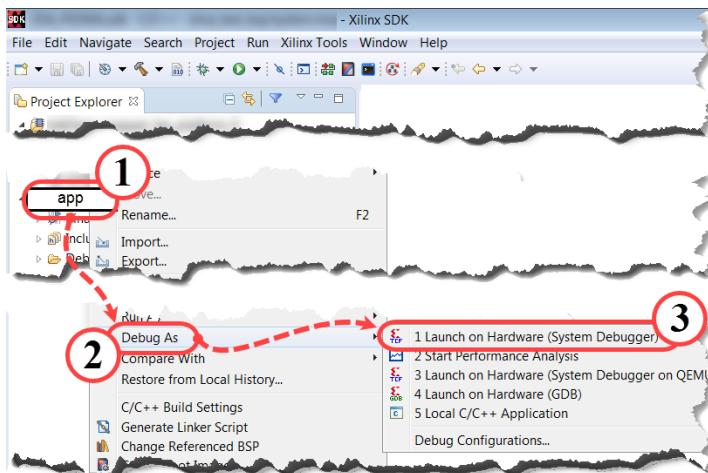


Figure 319: Selecting Launch on Hardware

The Confirm Perspective Switch dialog box opens.

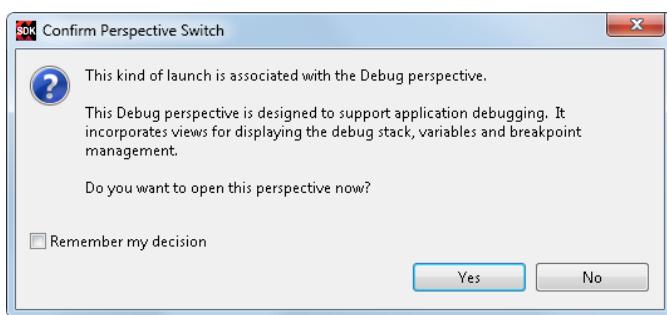


Figure 320: Confirming Switch of Perspective

- 1-1-4. Click **Yes**.

The Debug Perspective view opens.

Enabling Cross Trinngering in a GDB Debug Configuration

Cross-triggering between the Vivado logic analyzer and the SDK System Debugger must be enabled in the Xilinx C/C++ application's (System Debugger) debug configuration. This is typically performed from the C/C++ perspective.

1-1. Reconfigure an existing debug configuration for a specific application project to support hardware-software cross triggering.

- 1-1-1. Right-click the application project that you want to modify the Debug configuration for in the Project Explorer pane (1).
- 1-1-2. Select **Debug As** (2).
- 1-1-3. Select **Debug Configurations** (3) to open the configurations manager.

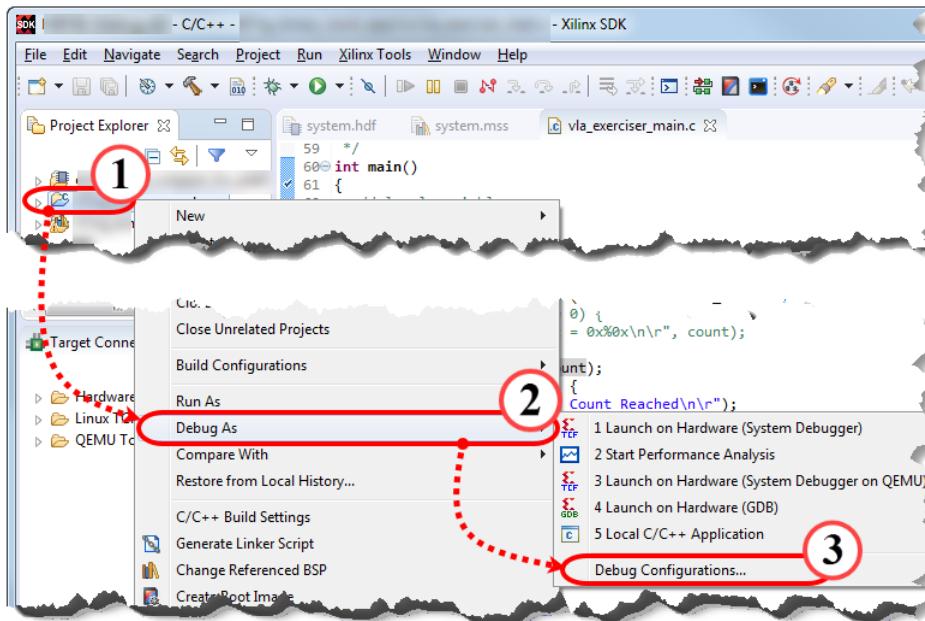


Figure 321: Accessing the Debug Configuration Dialog Box

The Debug Configurations dialog box opens.

- 1-1-4. Expand (>) **Xilinx C/C++ application (System Debugger)** to access the existing debug configurations.
- 1-1-5. Select **the Debug configuration**.
- 1-1-6. Select the **Target Setup** tab, which contains the enable box for cross-triggering.
- 1-1-7. Select the **Enable Cross-Triggering** option.

When this is done, the Cross-Triggering options button becomes active.

- 1-1-8. Click the **Cross-Triggering** options button to open the Cross Trigger Breakpoints window.

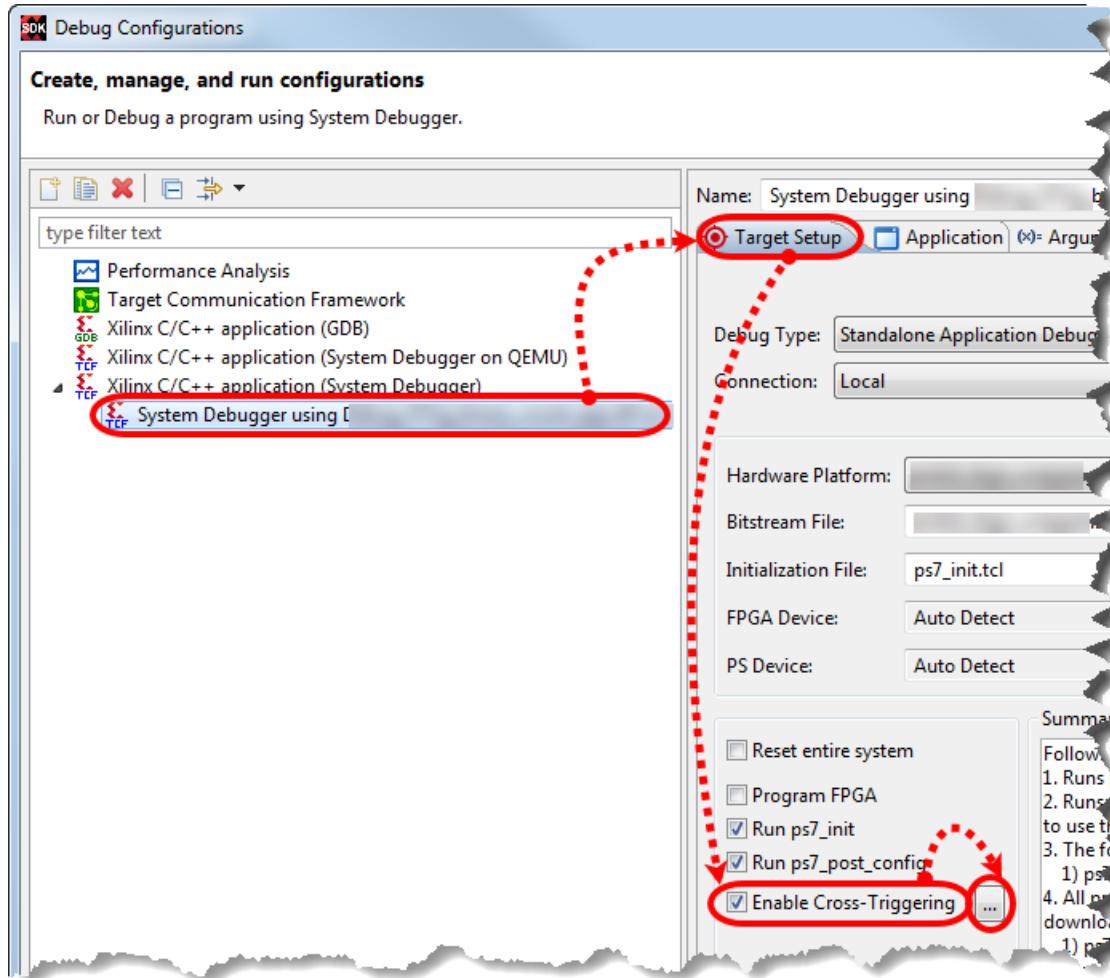


Figure 322: Enabling Cross-Triggering

The Cross Trigger Breakpoints window opens. Here you can see all of the breakpoints associated with cross-triggering.

Note: Two cross-trigger conditions will be created. One for the input signals to the PS and one for the output signals to the PL. Each cross-trigger condition created will result in the generation of a unique breakpoint that is visible in the breakpoints view during the debug session.

While it should be possible to combine the cross-triggering conditions into one single condition handling both directions, keeping them isolated makes sense as there is a breakpoint per direction and it is more obvious what is happening.

1-1-9. Click **Create** to create a new input breakpoint.

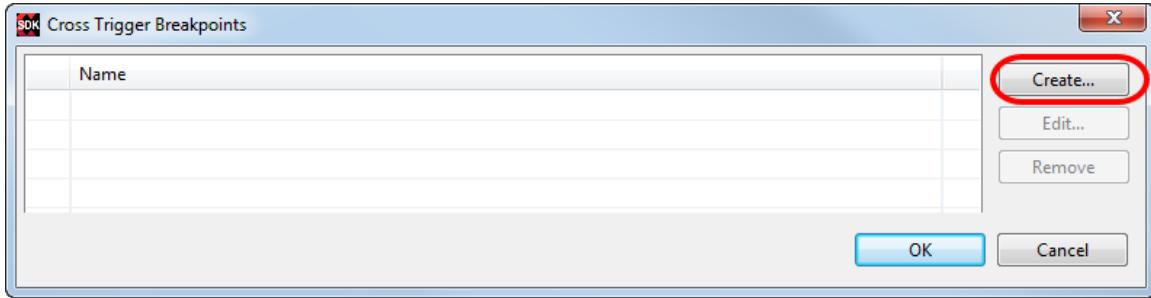


Figure 323: Creating a New Cross Trigger Breakpoint

The Edit Cross Trigger Breakpoint window opens. Here you have access to all of the input and output cross triggering signals and you will now select which ones you want to be active.

1-1-10. Enable the output triggers by placing a check mark in the following signals:

- o CPU-0 > CPU0 debug request (enables the PS request to the PL, output from PS) (1)
- o FTM > FTM triggers 0 through 3 (enable the triggers *into* the PS, input into PS) (2)

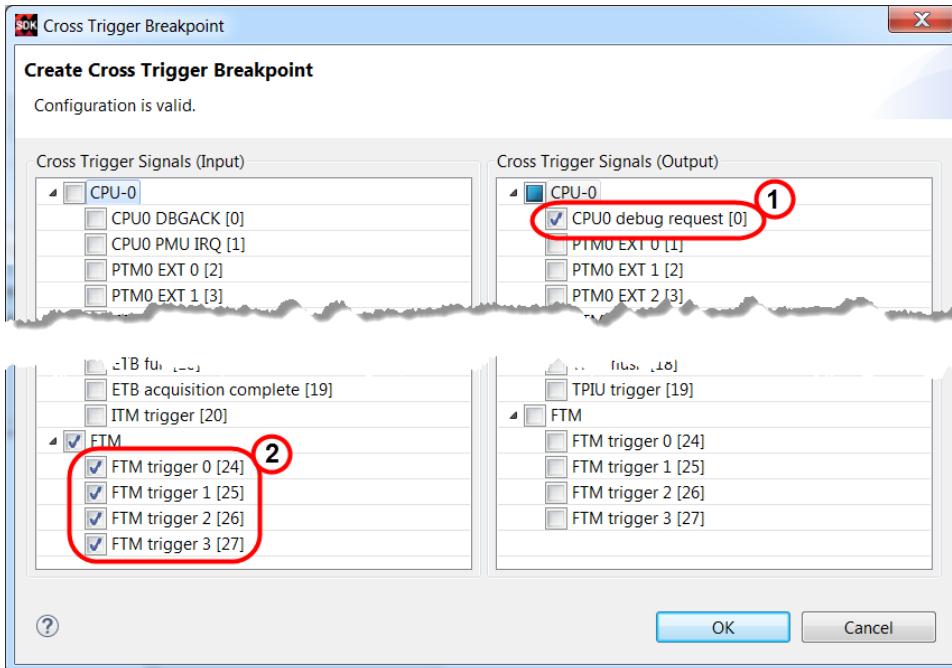


Figure 324: Cross Trigger Breakpoints - Input

1-1-11. Click **OK** to accept these breakpoints and return to the previous window.

1-1-12. Click **Create** to create the Output breakpoint.

The Create Cross Trigger Breakpoint editing window will appear again.

1-1-13. Enable the input triggers by placing a check mark in the following signals:

- CPU-0 > CPU0 DBGACK (debug acknowledge response from PL for interrupt) (1)
- FTM > FTM triggers 0 through 3 (enable the trigger from the PS into the PL) (2)

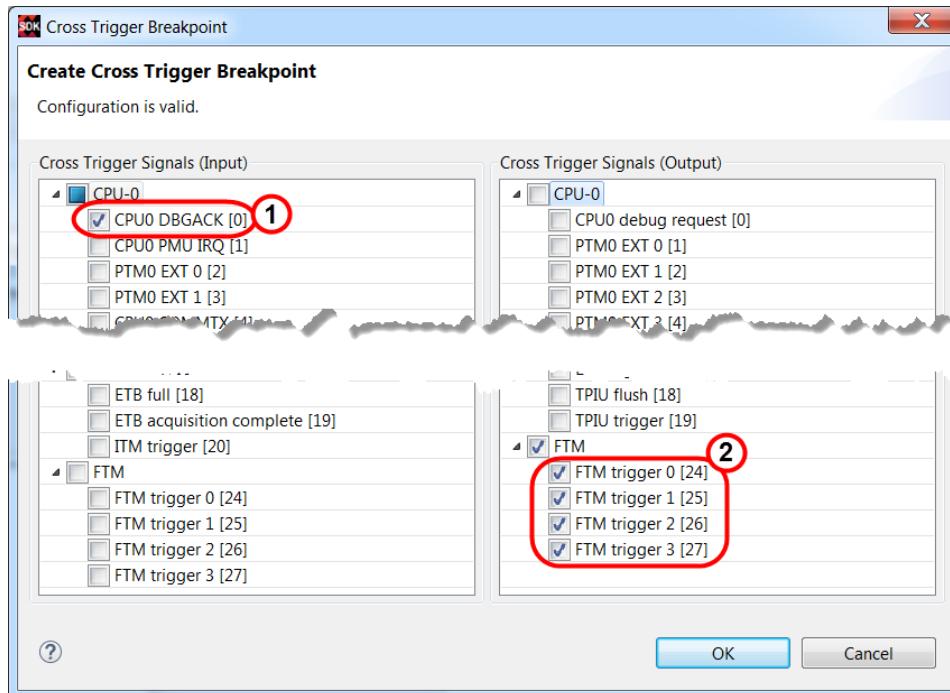


Figure 325: Cross Trigger Breakpoints - Output

1-1-14. Click **OK** to accept these breakpoints and return to the previous window.

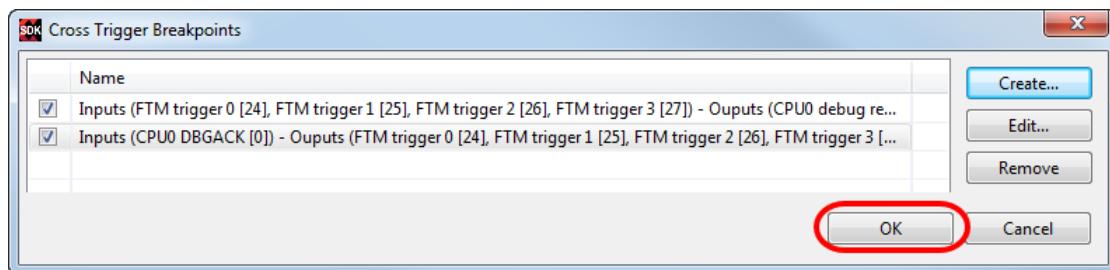


Figure 326: Reviewing the Newly Created Cross Trigger Breakpoints

1-1-15. Click **OK** to close this window and return to the Debug configuration.

1-1-16. Click **Apply** to use these debug configuration settings.

1-1-17. Click **Debug** to launch the debugger session.

1-1-18. Click **Yes** to terminate the previous launch if an SDK Warning dialog box appears.

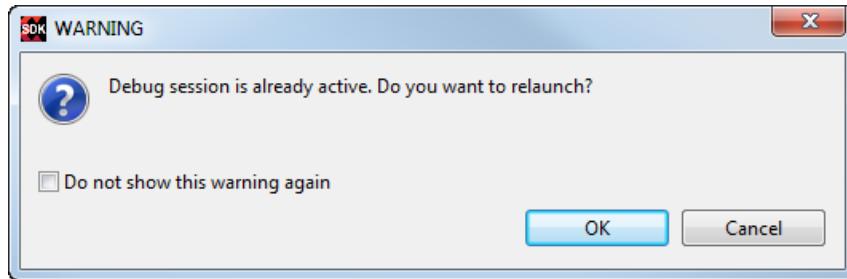


Figure 327: Terminating the Previous Launch

If the Confirm Perspective Switch dialog box appears, click **Yes**.

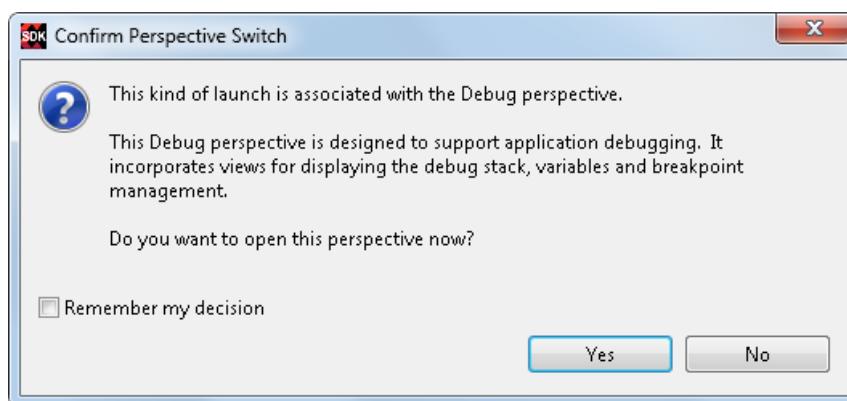


Figure 328: Confirming Switch of Perspective

The Debug perspective view opens.

Controlling Execution

Once the program is running there are a number of mechanisms available to control how you can proceed through the code. The following table lists many of the commonly used capabilities.

Icon	Shortcu t	Description
		Launches debugger (changes to Debug perspective)
	F8	Continues execution – runs until next breakpoint encountered or paused/terminated
		Pauses execution – temporarily stops at current instruction being executed
	^F2	Terminate (stop) execution – no further debugging possible unless relaunched
	F5	Step Into (descend into a function) – used to enter a function to debug
	F6	Step Over (execute function) – used when the function is known to be good
	F7	Step Return (return to calling location) – used to exit a function after checking

Managing Breakpoints

Breakpoints enable the designer to stop the code at almost any given point. Breakpoints can be added, deleted, toggled, enabled, and disabled.

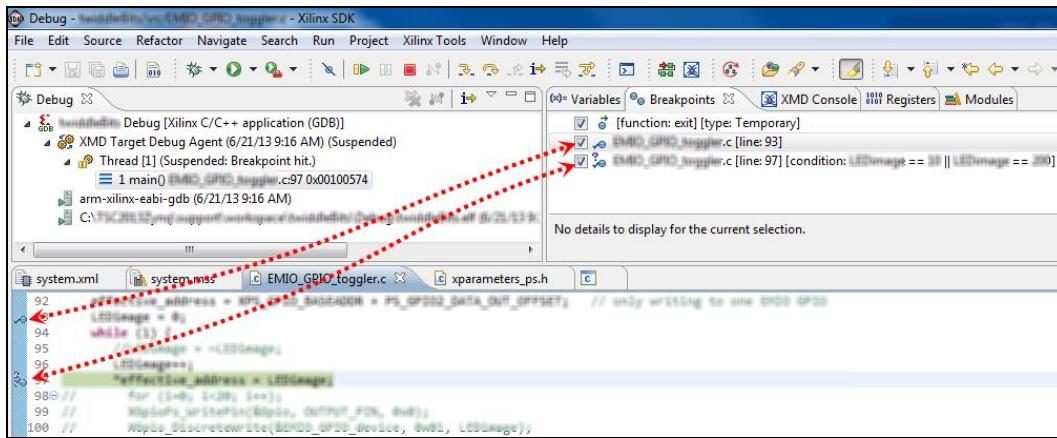


Figure 329: Relationship Between Breakpoints and the Breakpoint Window

Adding Breakpoints

1-1. Add a breakpoint at the current line.

1-1-1. Right-click the left margin in the editor XX.

1-1-2. Select **Add Breakpoint** or **Toggle Breakpoint**.

If you select Add Breakpoint, then the Breakpoint Properties dialog box opens. Here you can specify the details for this breakpoint if necessary to create a conditional breakpoint.

If you select Toggle Breakpoint, then a simple breakpoint will be created (or removed if one already existed) on the line you specified using default breakpoint properties.

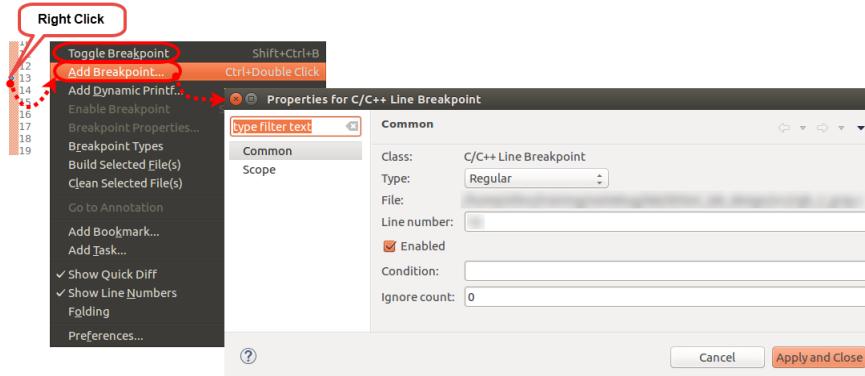


Figure 330: Adding a Simple or Conditional Breakpoint

1-1-3. Click **OK** to create the breakpoint if Add Breakpoint is selected.

Removing Breakpoints

Refer to the "Managing Breakpoints" section in the *Lab Reference Guide* for a more complete view of breakpoint management.

1-1. Remove a breakpoint from a line of code from the editor window.

- 1-1-1. Right-click the breakpoint that you want to remove.
- 1-1-2. Select **Toggle Breakpoint**.

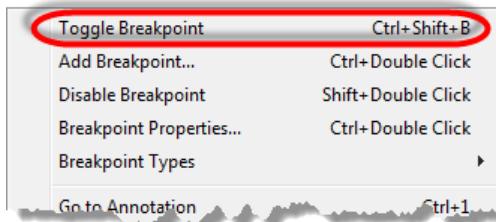


Figure 331: Toggling a Breakpoint Into and Out of Existence

-- OR --

Breakpoints can be removed from the Breakpoints window.

- 1-1-3. Right-click the breakpoint that you want to remove.
- 1-1-4. Select **Remove** from the context menu to delete the selected breakpoint.

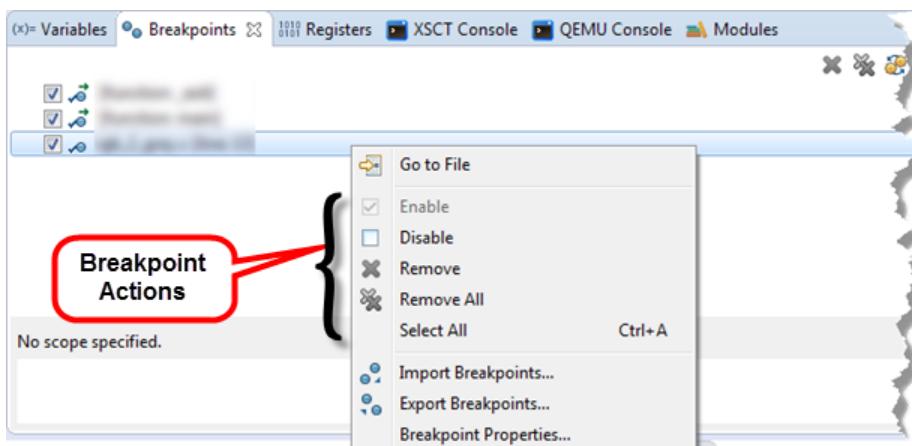


Figure 332: Actions from the Breakpoint Window

Accessing a Breakpoint's Properties

Refer to the "Managing Breakpoints" topic for a more complete view of breakpoint management.

1-1. Access the breakpoint's properties.

- 1-1-1. Right-click the breakpoint in the Editor window -- OR -- right-click the breakpoint in the Breakpoint window
- 1-1-2. Select **Breakpoint Properties**.

Creating a Conditional Breakpoint

Refer to the "Managing Breakpoints" topic for a more complete view of breakpoint management.

Breakpoints can be made conditional. This means that when the breakpoint is set at a particular line of code and that line of code is reached, then a C style expression is evaluated. If that expression evaluates "true", then execution pauses on that line; otherwise, execution continues as if there were no breakpoint present.

The most common way to do this is when you "Add Breakpoint" (see the above entry). Alternatively, you can access the properties for an existing breakpoint.

1-1. Access the breakpoint's properties.

- 1-1-1. Right-click the breakpoint in the Editor window -- OR -- right-click the breakpoint in the Breakpoint Window.
- 1-1-2. Select **Breakpoint Properties**.
- 1-1-3. Enter a legal C language conditional statement into the Condition. field.

Remember that variable scoping rules apply! That is, the variables that you are testing in this conditional must be "active" when this breakpoint is tested.

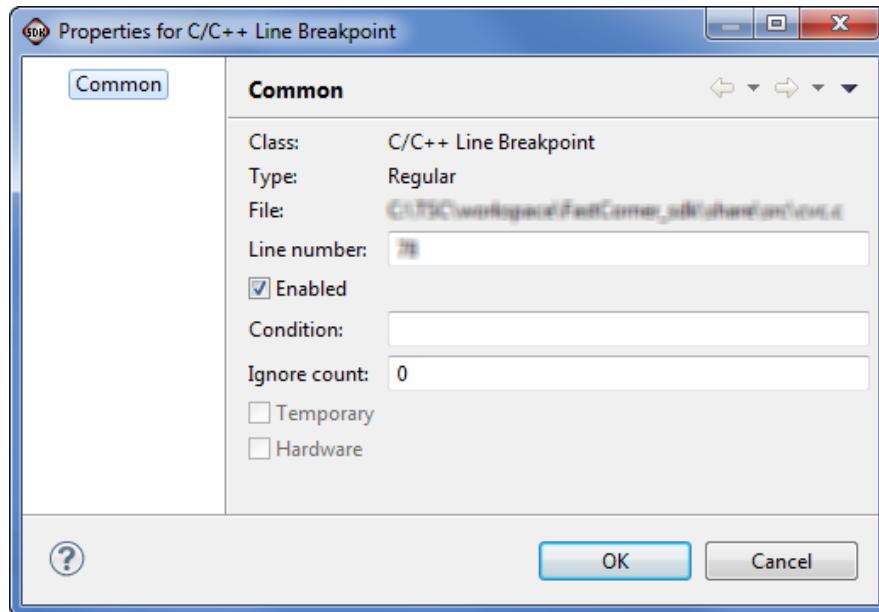


Figure 333: Breakpoint Properties Dialog Box

- 1-1-4. Click **OK**.

Enabling and Disabling a Breakpoint

Refer to the "Managing Breakpoints" topic for a more complete view of breakpoint management.

1-1. Enable a breakpoint from the Breakpoint window.

- 1-1-1. Locate the breakpoint to toggle.
- 1-1-2. Click in the checkbox next to that breakpoint name.

A check mark will show if the breakpoint is enabled.

1-2. Disable a breakpoint from the Breakpoint window.

- 1-2-1. Locate the breakpoint to toggle.
- 1-2-2. Click in the checkbox next to that breakpoint name.

A check mark will be absent when the breakpoint is disabled.

1-3. Enable a breakpoint from the Editor window.

- 1-3-1. Locate the breakpoint to toggle in the Editor window.
- 1-3-2. Right-click the breakpoint icon and select **Enable Breakpoint**.

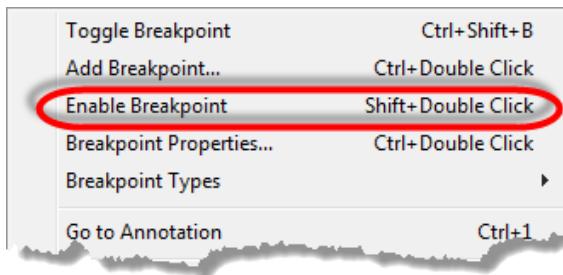


Figure 334: Enabling Breakpoint from the Pop-Up Menu

1-4. Disable a breakpoint from the Editor window.

- 1-4-1. Locate the breakpoint to toggle in the Editor window.
- 1-4-2. Right-click the breakpoint icon and select **Disable Breakpoint**.

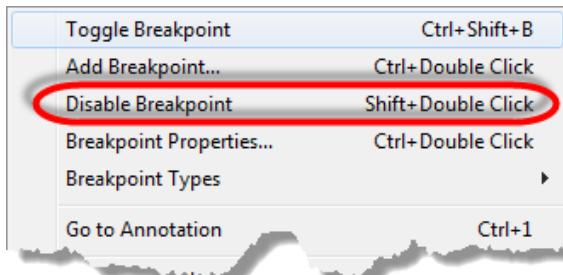


Figure 335: Disabling a Breakpoint from the Pop-Up Window

1-5. Enable and disable a breakpoint from the Breakpoint window.

- 1-5-1. Locate the breakpoint to toggle.
- 1-5-2. Click in the checkbox next to that breakpoint name.

A check mark will show if the breakpoint is enabled.

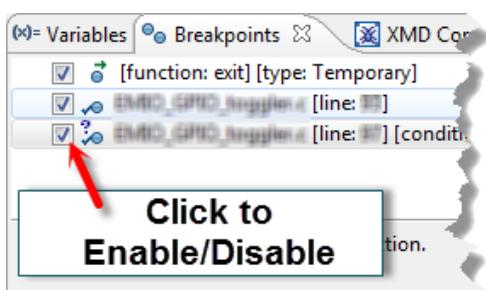


Figure 336: Enabling and Disabling Breakpoints

Running to a Breakpoint

1-1. Run to the next breakpoint.

1-1-1. Click the **Resume** icon (▶) or press <F8> to continue operation.

The application will run until:

- An unconditional breakpoint is met,
- A conditional breakpoint whose condition is satisfied is met, or
- The user halts (■) or pauses (▷) execution

Single-Stepping

Single-stepping is the process of executing a single line of code at a time. When assembly language is single-stepped, each instruction is atomic and only that one instruction is run per single-step.

Higher level languages (such as C and C++) typically have many assembly instructions per line. When these higher level languages are single-stepped, all of the assembly level instructions that make up the high-level line of code is executed.

Optimization must be set to 0 (off) for this to happen in a proper fashion. When optimization is enabled, the assembly language instructions are re-arranged and the relationship between a single line of high level code and its associated assembly instructions is broken. This results in single-stepping appearing to jump around in a non-linear fashion.

Single-stepping takes two basic forms: stepping over and stepping into.

Stepping over means that if the line of high-level (C/C++) code being executed is a function, then all of the instructions contained within that function is executed, basically treating the function call as a single instruction. Stepping into means that if the line of high-level (C/C++) code being executed is a function, then control passes to that function and one can single-step all of the instructions within that function.

Once you descend into the function you do not have to single-step or continue to the end of the function. You can execute the remainder of the function and return to the calling point by clicking the Step Out Of icon.

Stepping over is accomplished by pressing <F6> or clicking the ➔ icon.

Stepping into a function is accomplished by pressing <F5> or clicking the ➤ icon.

Once in a function, you can step-return by pressing <F7> or clicking the ➙ icon.

Monitoring Variables

Variables can be monitored in several ways. One of those ways is via the Variables tab.

1-1. Observe the values in the Variables tab.

- If the Variables tab is not visible, select **Window > Show View > Variables**.

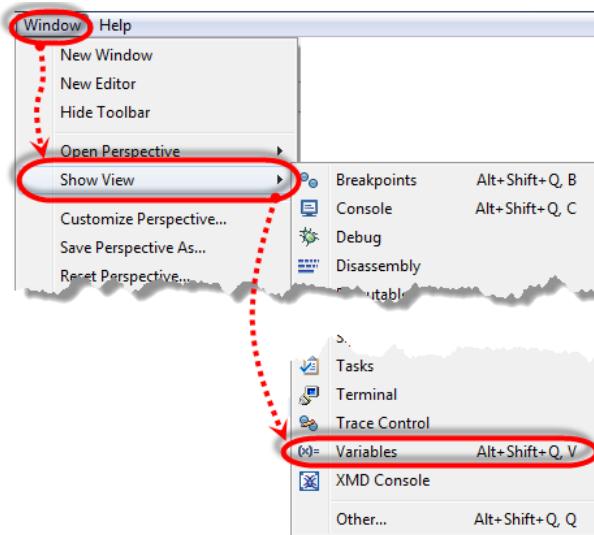


Figure 337: Making the Variables Tab Visible

This exposes the Variables tab.

Name	Type	Value
↳ Status	XStatus	0
↳ func_	const char [10]	"XPfw_Main\000"
↳ [1]	char	'X'
↳ [2]	char	'P'
↳ [3]	char	'f'
↳ [4]	char	'w'
↳ [5]	char	'.'
↳ [6]	char	'M'
↳ [7]	char	'a'
↳ [8]	char	'i'
↳ [9]	char	'n'
'\000'		
Hex: 00, Dec: 0, Oct: 0		
Bin: 0000,0000		
Size: 1 byte, Type: char		
Address: 0xffffdce6d1		

Figure 338: Variables Tab Overview

Note: The Variables tab can be hidden behind other tabs in the same pane as the Breakpoint tab. Simply select the Variables tab to bring it to the foreground.

Selecting an entry shows the Alternative Representations of that entry in the lower panel. In the above figure, the array element `_func_[9]` is selected, which is the null character. Its alternative representations are all 0 in hex, decimal, and octal. The number of bytes for the data type are shown along with its address in hex.

Managing Expressions

Expressions are C-style combinations of mathematical and/or conditional operations and variables. Typically expressions are used to evaluate portions of larger expressions in the code.

For example, if there is a more complex conditional statement in the code such as `"if (!(condition1) && (condition2 || condition3) { ... }`", a programmer might create three expressions, one for each condition. In this way, the programmer can quickly see which of the conditions is true or not and how it affects the larger "if" statement.

1-1. Access the Expressions window.

1-1-1. If the Expressions tab is not visible, select **Window > Show View > Expressions**.

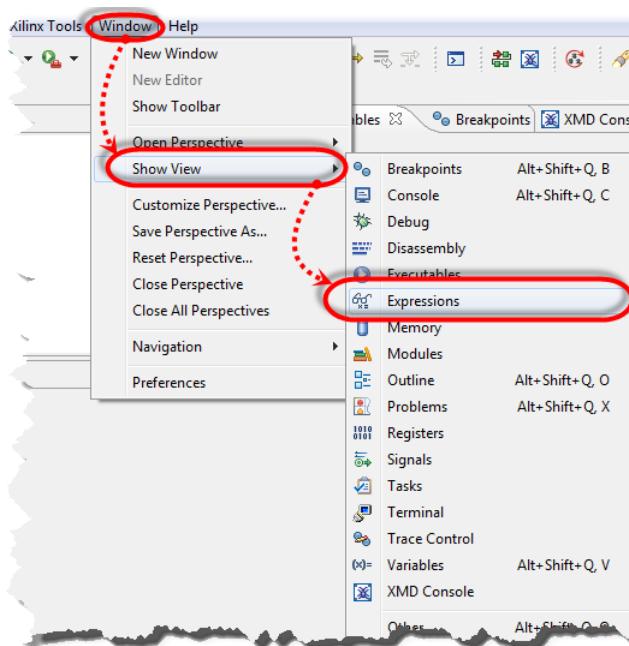


Figure 339: Accessing the Expressions Window from the Debug Perspective

1-2. Create (add) a new expression.

- 1-2-1. Click the green Plus icon (+) in the Expressions tab.

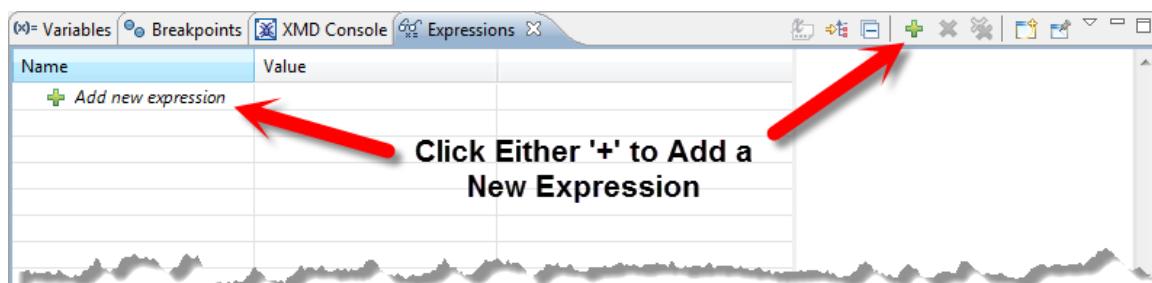


Figure 340: Adding a New Expression

- 1-2-2. Type in the new expression.

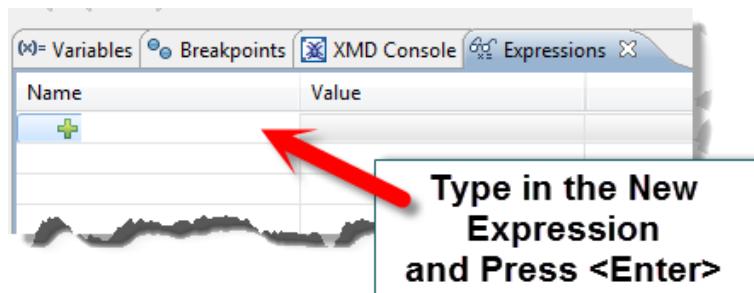


Figure 341: Entering the New Expression

Be aware that expressions are subject to scoping rules. That is, if a variable is not "active" in the region of code that is currently running, then any expression that uses that variable cannot be evaluated and will report an error. This error will go away when the variable becomes "active" as one steps through the code.

Changing the Value of a Variable

Sometimes it is beneficial to change the value of a variable during execution. This is often done to shorten wait loops or cause an event to happen that might not occur as the code is currently written.

1-1. Change the value of a variable.

- If the Variables tab is not visible, select **Window > Show View > Variables**.

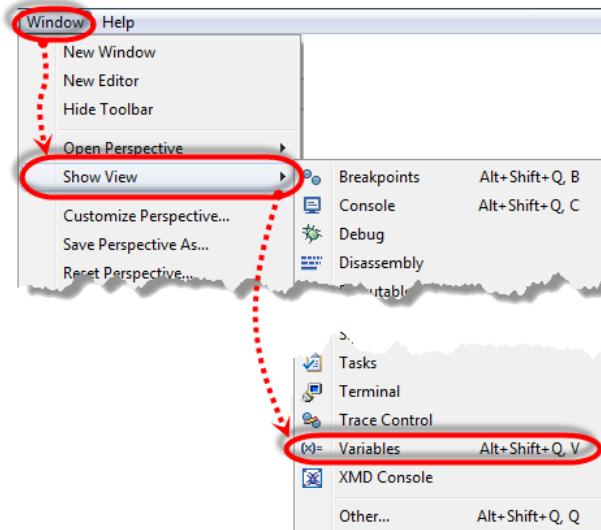


Figure 342: Making the Variables Tab Visible

This exposes the Variables tab.

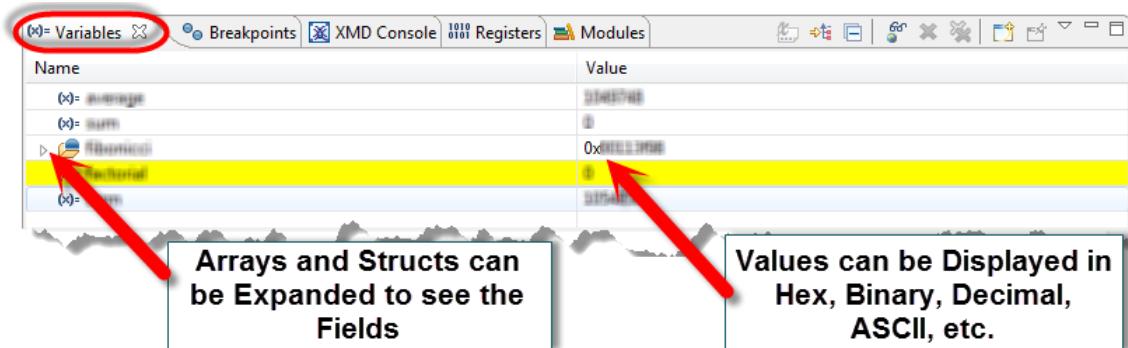


Figure 343: Example View of the Variables Tab

Variables are listed alphabetically.

- 1-1-2. Change the radix of a variable by right-clicking the variable name and selecting **Change Value**.

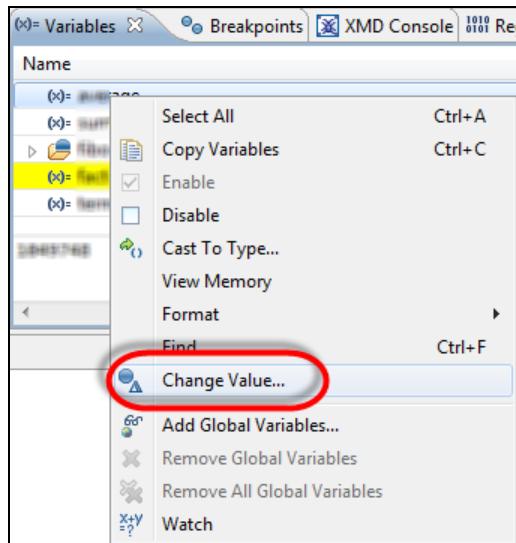


Figure 344: Changinge the Value of a Variable

- 1-1-3. Enter the new value of the variable.

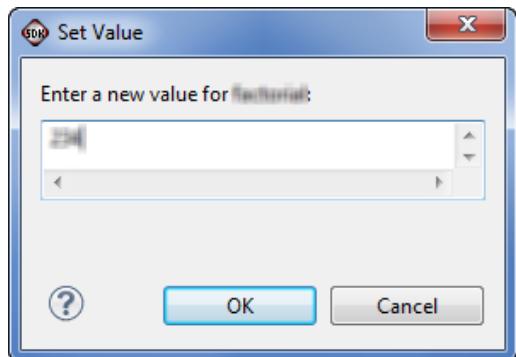


Figure 345: Setting the Value of a Variable

- 1-1-4. Click **OK**.

Using Expressions

Expressions are combinations of variables. Typically these are used to show intermediate results of various C expressions at various times during execution. An expression can be any legal C/C++ expression, including conditional tests (<, >, ==, &&, |, ...) or mathematical constructs (+, -, *, &, |,...).

1-1. Observe the values in the Expressions tab.

- 1-1-1. If the Expressions tab is not visible, select **Window > Show View > Expressions**.

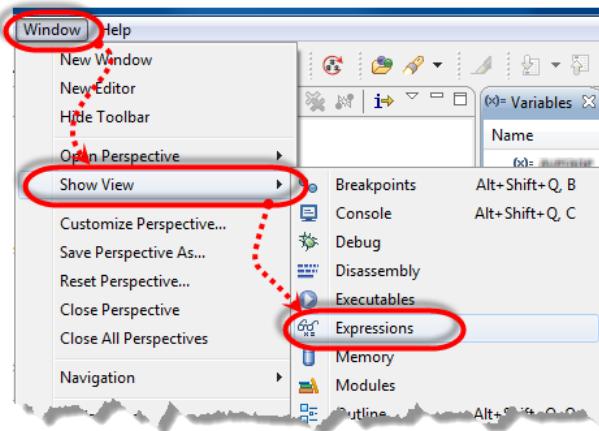


Figure 346: Making the Expressions Tab Visible

- 1-1-2. Select the **Expressions** tab.

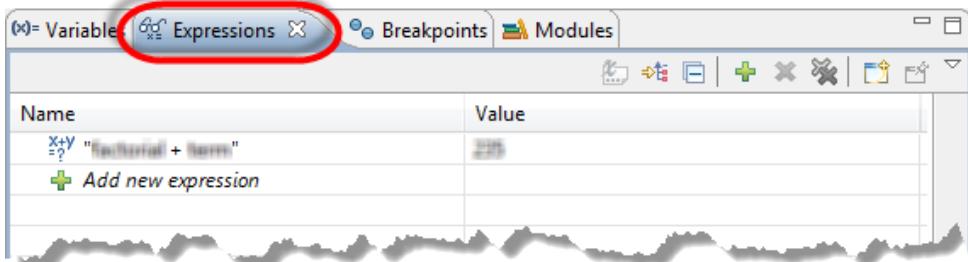


Figure 347: Expressions Tab

- 1-1-3. Click **Add new expressions**.

- 1-1-4. Type in the new expression.

Remember that each variable that you use must be in the current scope (that is, active). You can use global variables.

- 1-1-5. Press <**Enter**>.

Setting Up the Linux Console for Debugging

When you are debugging a Linux software application, the console is multi-use, displaying debugger messages, errors, and program output from various different sources. The instructions below show a typical console setup with minimal debug messages, concentrating on output messages from the program being debugged.

1-1. Perform several housekeeping tasks that will enable the proper display of STDOUT in the RSE console.

This console has different view modes that are not needed and hide the desired program execution terminal view. This is expected to change as the SDK RSE tools evolve.

- 1-1-1. Verify that the Console tab is selected.
- 1-1-2. In the upper right of the Console tab, click the **Verbose console mode** icon to turn off verbose mode.

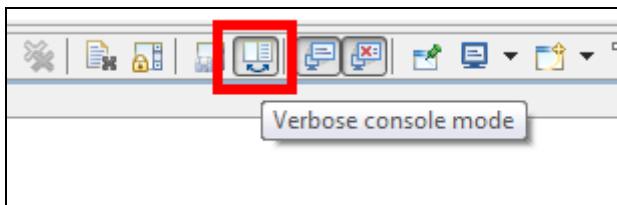


Figure 348: Verbose Console Mode

- 1-1-3. In the upper right of the Console tab, click the arrow next to the Display Selected Console icon and select **Remote ARM Linux Application** that is in the target execution location that was set in the Debug Configuration.

This will typically be in the **/tmp** directory and, typically, the last choice in the list.

This selects the terminal output of the application, so *printf* statements can be viewed.

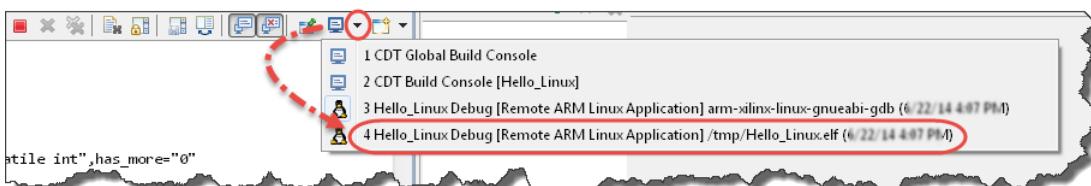


Figure 349: Display Selected Console

Running the Application on Hardware

1-1. Run *your application project name*.

1-1-1. Right-click **your application project name**.

1-1-2. Select **Run As > Launch on Hardware (System Debugger)**.

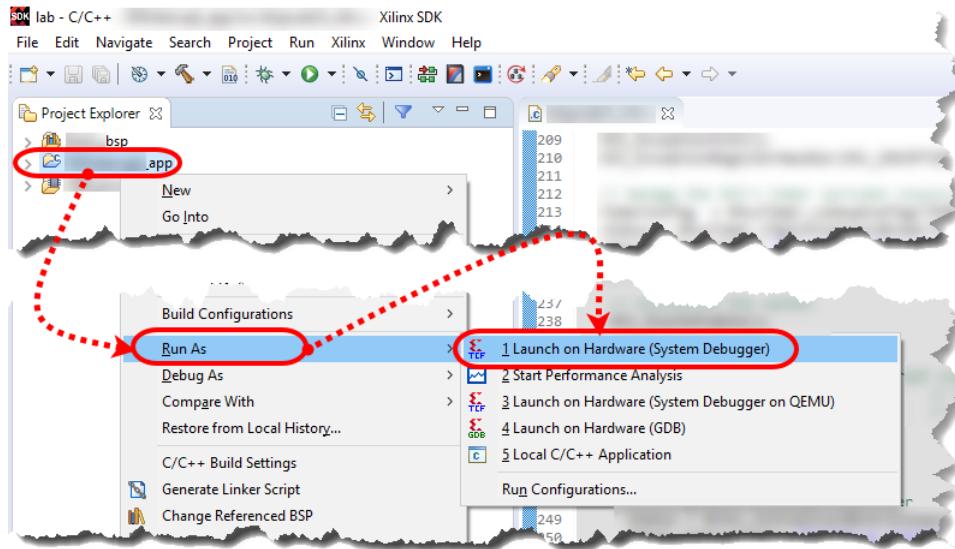


Figure 350: Launching a Run on Hardware

1-1-3. Click **Yes** if you are asked to terminate a previous run.

The program launches on the hardware.

Linux and Remote System Explorer

This section covers Linux and using the Remote System Explorer.

In This Section

Creating a Linux C/C++ Application Project.....	270
Creating a Linux Debug Configuration.....	272
Configuring the PC's Ethernet Port for Remote System Explorer.....	276
Opening the Remote System Explorer.....	276
Setting the IP Address of the Development Board	279
Verifying the Host's Static IP Ethernet Port from the Development Board	280
Downloading and Running the Linux Application Using RSE.....	280
Configuring the Ethernet Port for Use with the Remote System Explorer.....	286

Creating a Linux C/C++ Application Project

Using the Application Project Wizard is a quick way to set up a Linux C or C++ software application project that targets an existing processor and Linux version. Based on the dialog box choices, the appropriate toolchain is selected for pre-processing, compiling, assembling, and linking.

1-1. Create a new C/C++ application project named *your application project name*.

- 1-1-1. Select **File** (1) > **New** (2) > **Application Project** (3) to open the New Project dialog box.

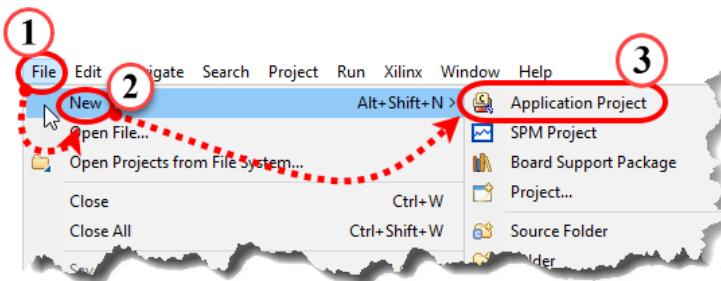


Figure 351: Creating an Application Project

- 1-1-2. Enter **your application project name** as the project name (1).
- 1-1-3. Ensure that **linux** is selected from the OS Platform drop-down list (2 and 3).
- 1-1-4. Ensure that **the processor you wish to target** is selected from the Processor drop-down list.
- 1-1-5. Select your preferred language: C or C++ (4).

This selects which tools will be used to compile your code.

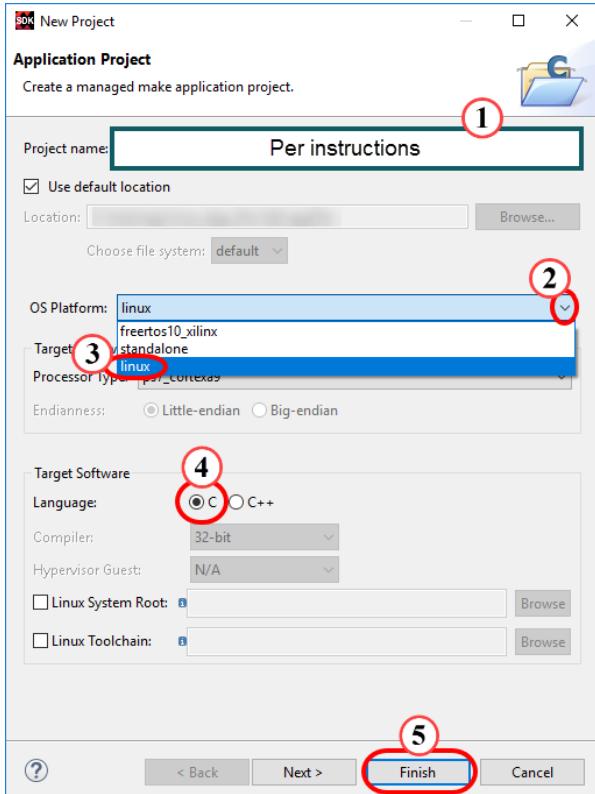


Figure 352: Selecting Linux for the New Application Project

- 1-1-6. Click **Next** to view the templates available for Linux applications (5).
- 1-1-7. Select **an application template** (1).

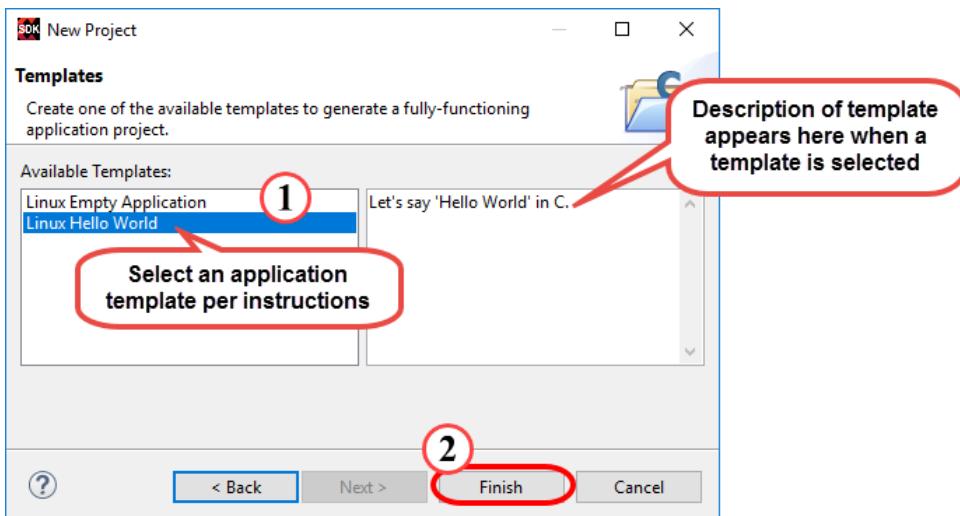


Figure 353: Selecting a Linux Template

- 1-1-8. Click **Finish** to close the dialog box and create the new project (2).

Creating a Linux Debug Configuration

Run and Debug configurations are application project objects that contain communication, hardware, and execution options for running an application on a hardware or emulation platform. The selections for Run and Debug are identical and only differ in that Run just executes the application and Debug opens a debug perspective and launches a debug program. There are different type of Run and Debug configurations based on the operating system (or Standalone libraries) and the SDK download/debug tools that you want to use.

1-1. Create a Linux Debug configuration. Debug and Run configurations associate an ELF object file to a target (typically a hardware board) for execution. In this case, the target is a hardware board accessed over the Ethernet TCP/IP connection that was set up when the RSE tool was engaged.

- 1-1-1.** Select the **C/C++** tab in the upper-right corner of the GUI to return to the C/C++ perspective.

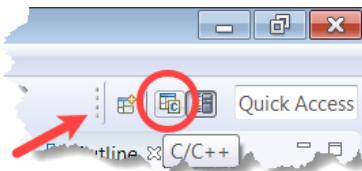


Figure 354: Changing Perspective

This brings back access to the software projects. It is accessed by first clicking the >> in the same location. If this tab is not available, you can also return to the perspective by selecting **Window > Open Perspective > Other > C/C++ (default)**.

- 1-1-2.** In the Project Explorer tab (1), right-click **your application project name** and select **Debug as** (2) > **Debug Configurations** (3).

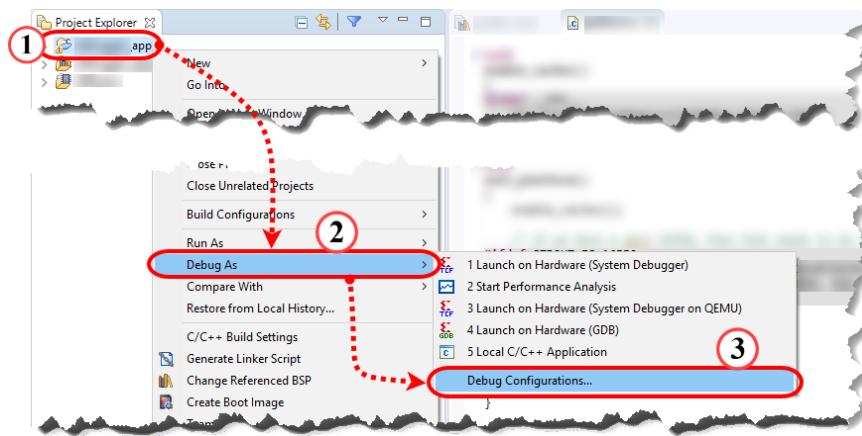


Figure 355: Selecting Debug Configurations

The Debug Configurations dialog box opens. A choice of debug configurations are displayed. For debugging Linux applications, the **Remote ARM Linux Application** configuration is specified to connect the Linux OS software application debugger (that is built into Linux) to the SDK debug perspective. All debug communications will take place via the Ethernet RSE connection that was set up earlier.

- 1-1-3. Double-click **Remote ARM Linux Application** to create the new debug configuration.

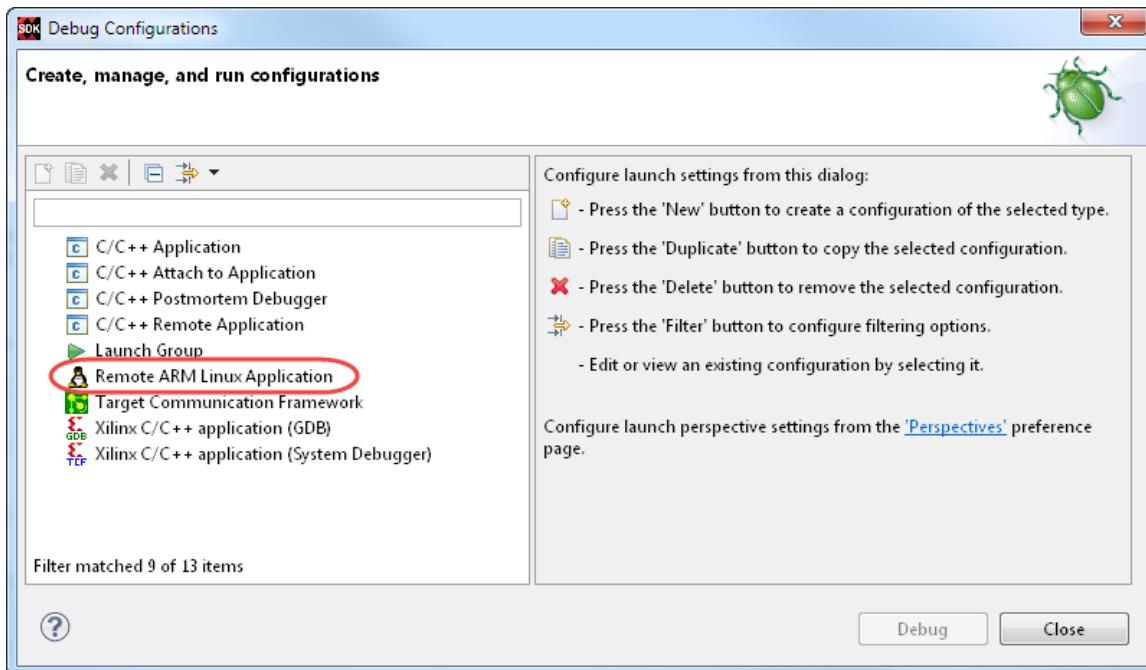


Figure 356: Selecting Remote ARM Linux Application

In the dialog box that follows, three fields auto populate using **your application project name** as the base naming for these fields.

- 1-1-4.** Select **192.168.1.10** from the Connection drop-down list (1).
- 1-1-5.** In the Remote Absolute File Path for C/C++ Application field, enter **/tmp/your application project name.elf** (2).

This is where the application to be debugged will reside on the Linux file platform, which in this case, is a memory file system.

Note the use of Linux "/" vs. Windows "\" in the path specification.

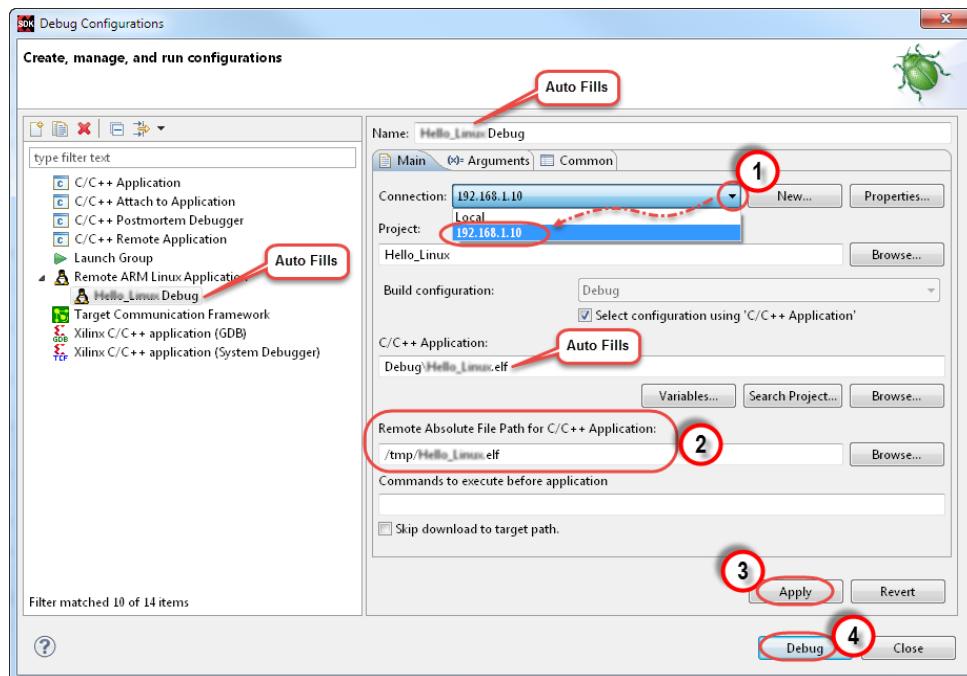


Figure 357: Linux RSE Configuration Dialog Box

- 1-1-6.** Click **Apply** (3) to store (remember) these settings.
- 1-1-7.** Click **Debug** (4) to begin debugging.
- 1-2.** Enter the user ID and password. If the connection is already established, this prompt will not appear and this step can be skipped.
- 1-2-1.** Enter **root** in the User ID and Password fields.

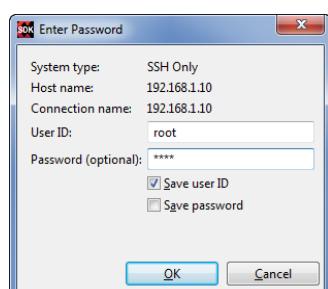


Figure 358: Entering the User ID and Password

1-2-2. Click **OK** to log in.

1-2-3. Click **Yes** to confirm opening the Debug perspective.

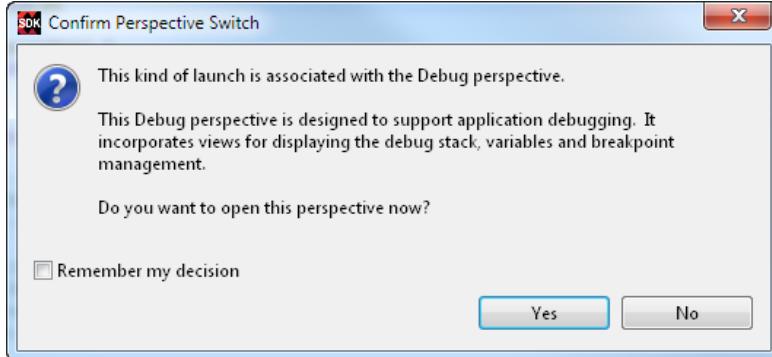


Figure 359: Confirming Perspective Switch

The Debug perspective opens.

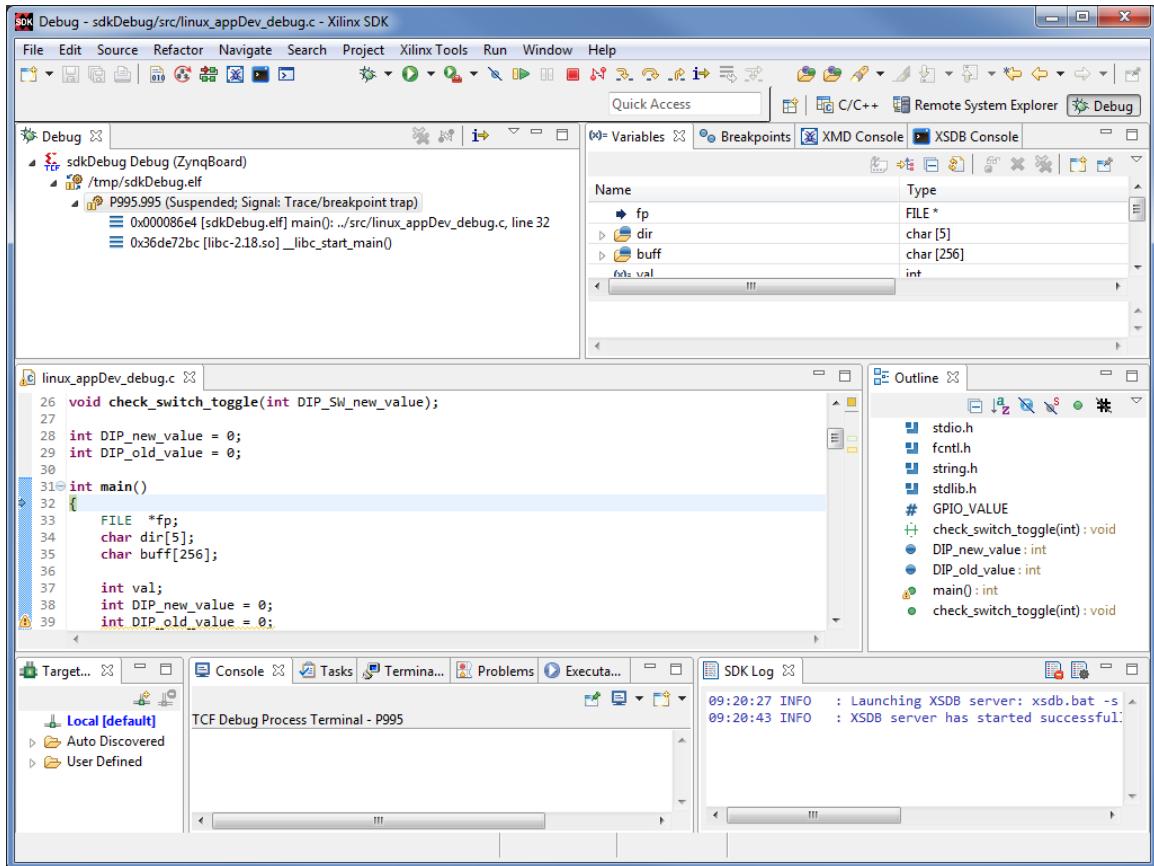


Figure 360: Debug Perspective

Program operation is suspended at the first executable statement in *main{} (not running)*.

Note that local variables for the current function are shown in the Variables tab.

Configuring the PC's Ethernet Port for Remote System Explorer

The PC's Ethernet port must be properly configured to handle traffic with the Linux platform.

1-1. Configure your PC to support the Remote System Explorer.

- 1-1-1.** Disconnect from the VPN if you are connected.
- 1-1-2.** Access your PC's Control Panel.
- 1-1-3.** Select **Network and Sharing**.
- 1-1-4.** Identify your cabled ethernet connection:
 - NOT the Bluetooth connection
 - NOT the VPN connection
 - NOT the wireless connection
- 1-1-5.** Right-click the cabled Ethernet connection.
- 1-1-6.** Ensure that the **Internet Protocol Version 6** option is unchecked.
- 1-1-7.** Select **Internet Protocol Version 4**.
- 1-1-8.** Select **Properties**.
- 1-1-9.** Assign an address other than 192.168.1.10 (which is the address of the device on Xilinx evaluation boards).
Typically, 192.168.1.11 is selected.
- 1-1-10.** Save and close all open windows

Opening the Remote System Explorer

The Remote System Explorer (RSE) tool uses the TCP/IP protocol to make a connection between the PC host computer and the hardware platform. This connection is similar to, and may be made in addition to the UART terminal connection. The major difference is that this communications channel is much faster than the UART and can be done remotely over any network that the development board is connected to.

1-1. Use the Remote System Explorer (RSE) tool to download the software application to the Linux platform.

- 1-1-1. Select **Window > Perspective > Open Perspective > Other** to open a new SDK perspective.

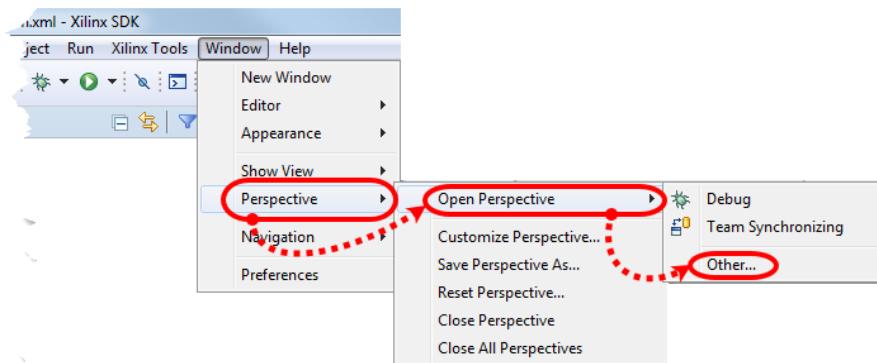


Figure 361: Selecting Open Perspective - Other

- 1-1-2. Select **Remote System Explorer** as it is the name of the perspective (a set of views that comprise the IDE desktop) that is being opened.
- 1-1-3. Click **OK** to open a new Remote System Explorer perspective.

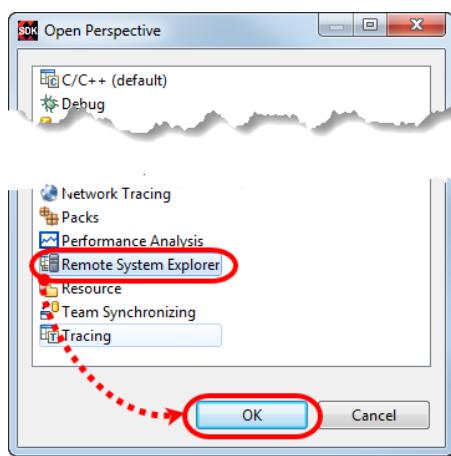


Figure 362: Selecting Remote System Explorer

1-2. Establish a new connection by using the industry-standard SSH protocol over the Ethernet port.

1-2-1. Right-click **Local** in the Remote Systems tab (top left).

This is where RSE connections are maintained, similar to projects in a workspace. It is possible to have multiple connections to different types of communication hardware ports.

1-2-2. Select **New > Connection**.

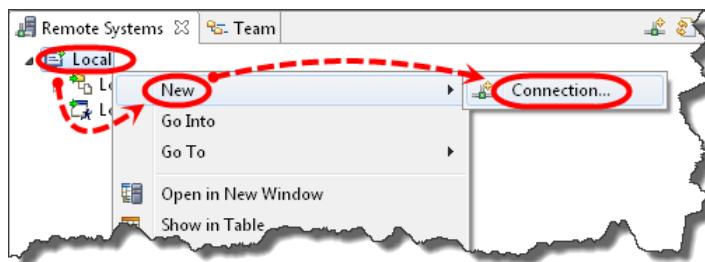


Figure 363: Selecting New Connection

1-2-3. Select **SSH Only** from the Select Remote System Type dialog box.

1-2-4. Click **Next** to advance to the next dialog box.

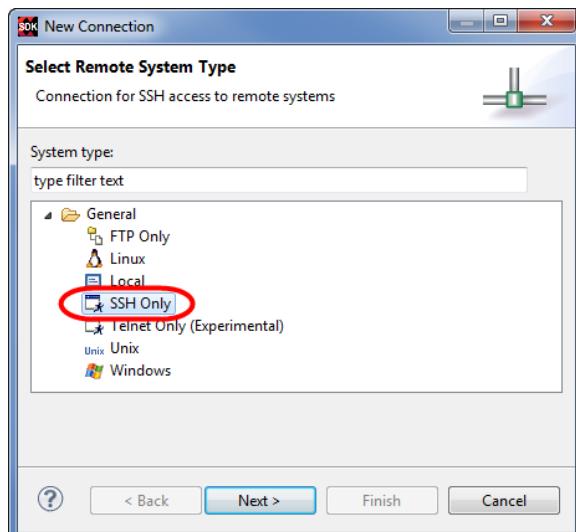


Figure 364: Creating a New SSH RSE Connection

A New Connection dialog box opens. You will now configure the new connection. This is the address of the hardware platform that Linux is running on. This address was pre-established (Linux default) or assigned over the serial terminal console after Linux booted.

1-2-5. Enter **the IP address of the board** into the *Host name* field.

1-2-6. Enter **the IP address of the board** into the *Connection name* fields.

The *Parent profile* field can stay at its default, which will be specific to your machine.

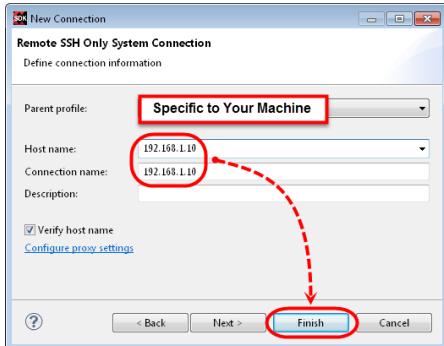


Figure 365: Entering the Host and Connection Names

- 1-2-7.** Click **Finish** to create the new connection.

The SDK tool switches to the Remote System Explorer perspective. A Perspective is a collection of views (tabs/windows) that are related to the type of task being performed, such as editing, debugging, etc.

Setting the IP Address of the Development Board

1-1. Set the IP address of the board.

- 1-1-1.** Enter the following at the Linux command prompt to change the IP address of the board to the IP address of the board:

```
ifconfig eth0 the IP address of the board
```

Note: This can be any address other than the one that the host is configured to.

Optional: You can verify the IP address by entering the following command:

```
ifconfig eth0
```

The response should be *similar* to the following:

```
eth0      Link encap:Ethernet  HWaddr 00:0A:35:00:01:22
          inet  addr:192.168.1.10  Bcast:192.168.1.255
          Mask:255.255.255.0
                  inet6 addr: fe80::20a:35ff:fe00:122/64 Scope:Link
                      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
                      RX packets:23 errors:0 dropped:0 overruns:0 frame:0
                      TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
                      collisions:0 txqueuelen:1000
                      RX bytes:2913 (2.8 KiB)  TX bytes:1446 (1.4 KiB)
                      Interrupt:54 Base address:0xb000
```

Verifying the Host's Static IP Ethernet Port from the Development Board

1-1. Verify the Ethernet connectivity between the host and the development board.

This will also indirectly verify that the host PC Ethernet port is set to an IP address of the host IP address.

1-1-1. Ping the host using the Linux terminal console to verify connectivity between the host and the target:

```
ping the host IP address -c 1
```

If the ping was successful (indicated by a 0% packet loss with roundtrip times), you can continue to the next section.

If it was not successful, there are several possibilities:

- The host IP address is not set properly.
 - See instructions for configuring the host's IP address in the *Lab Setup Guide*.
- The Ethernet cable may not be properly connected.
 - Unplug and replug the cable on both ends; verify that the Ethernet LEDs are flickering (if available).
- The Windows firewall is blocking the connection. Follow the procedure below to disable the Windows firewall.
 - Access the Windows Firewall controls through the Control Panel (select **Start > Control Panel > System and Security > Windows Firewall**).
 - From the options in the left sidebar, select **Turn Windows Firewall on or off**.
 - Select the **Turn off Windows Firewall** option for both network location settings.
- The sub-net address for either the board or host may not be entered correctly.
 - If you need to configure your PC's Ethernet port, refer to "Configuring the PC's Ethernet Port for Remote System Explorer" section under SDK or SDSoC Operations > Linux and Remote System Explorer in the *Lab Reference Guide*.

Downloading and Running the Linux Application Using RSE

The Remote System Explorer (RSE) tool uses the TCP/IP protocol to make a connection between the PC host computer and the hardware platform. This connection is similar to, and may be made in addition to the UART terminal connection. The major difference is that this communications channel is much faster than the UART and can be done remotely over any network that the development board is connected to.

1-1. Use the Remote System Explorer (RSE) tool to download the software application to the Linux platform.

- 1-1-1. Select **Window > Perspective > Open Perspective > Other** to open a new SDK perspective.

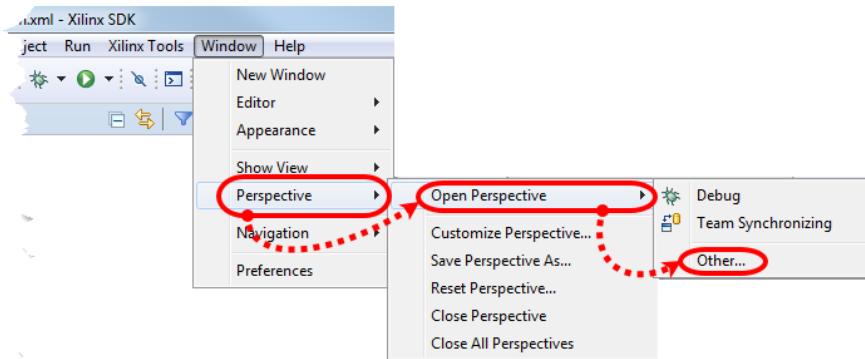


Figure 366: Selecting Open Perspective - Other

- 1-1-2. Select **Remote System Explorer** as it is the name of the perspective (a set of views that comprise the IDE desktop) that is being opened.
- 1-1-3. Click **OK** to open a new Remote System Explorer perspective.

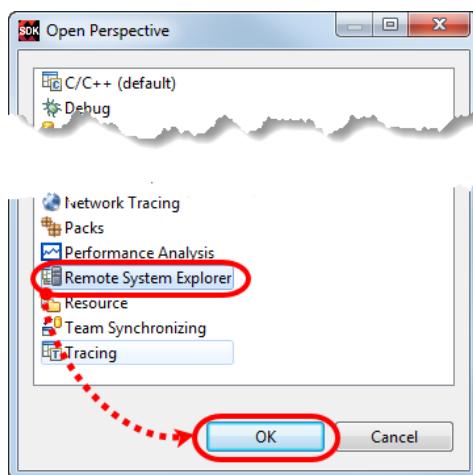


Figure 367: Selecting Remote System Explorer

1-2. Establish a new connection by using the industry-standard SSH protocol over the Ethernet port.

1-2-1. Right-click **Local** in the Remote Systems tab (top left).

This is where RSE connections are maintained, similar to projects in a workspace. It is possible to have multiple connections to different types of communication hardware ports.

1-2-2. Select **New > Connection**.

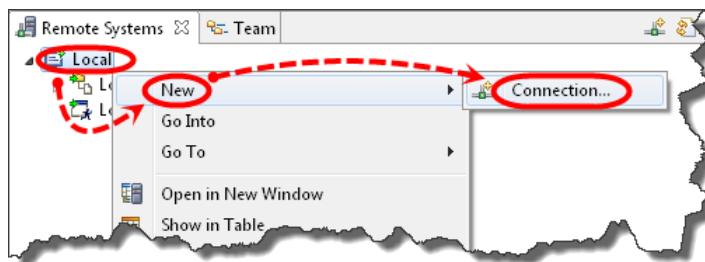


Figure 368: Selecting New Connection

1-2-3. Select **SSH Only** from the Select Remote System Type dialog box.

1-2-4. Click **Next** to advance to the next dialog box.

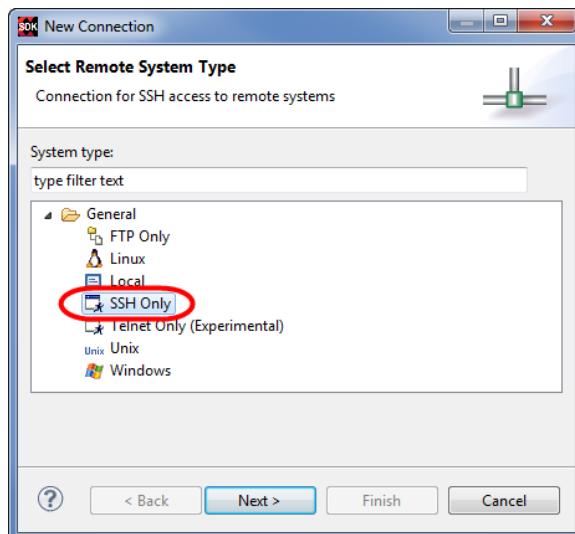


Figure 369: Creating a New SSH RSE Connection

A New Connection dialog box opens. You will now configure the new connection. This is the address of the hardware platform that Linux is running on. This address was pre-established (Linux default) or assigned over the serial terminal console after Linux booted.

1-2-5. Enter **the IP address of the board** into the *Host name* field.

1-2-6. Enter **the IP address of the board** into the *Connection name* fields.

The *Parent profile* field can stay at its default, which will be specific to your machine.

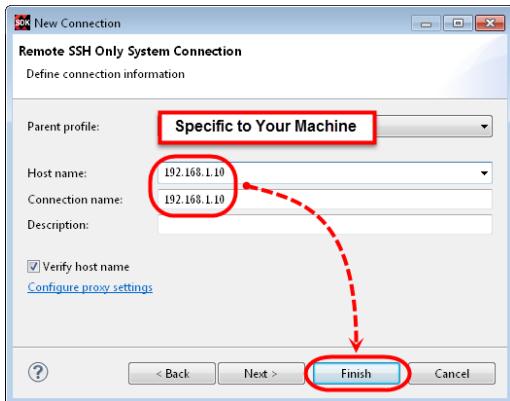


Figure 370: Entering the Host and Connection Names

- 1-2-7. Click **Finish** to create the new connection.

The SDK tool switches to the Remote System Explorer perspective. A Perspective is a collection of views (tabs/windows) that are related to the type of task being performed, such as editing, debugging, etc.

- 1-3. **Create a Run configuration. Run configurations associate an ELF object file to a target for execution. In this case, the target is a hardware board accessed over a network TCP/IP connection.**

- 1-3-1. Click the **C/C++** perspective (top right) to return to the original perspective.
- 1-3-2. In the Project Explorer tab, right-click **your project name**.
- 1-3-3. Select **Run as > Run Configurations**.

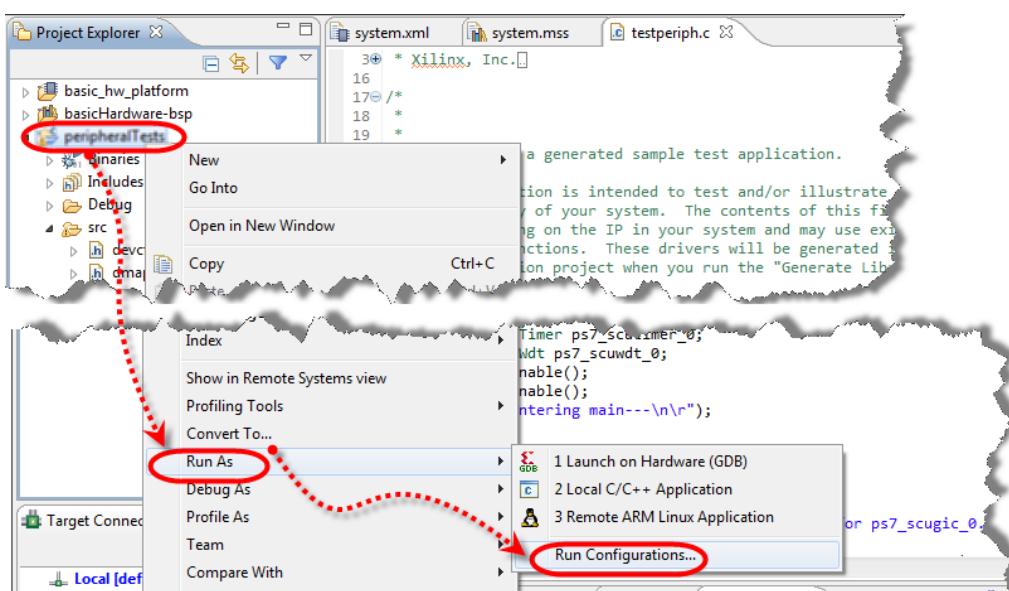


Figure 371: Selecting Run Configurations

1-4. Remember that this is a Linux application, not a Standalone/bare-metal application.

Perform the following operations from the Create, manage, and run configurations dialog box.

1-4-1. Double-click Remote ARM Linux Application.

1-4-2. Select 192.168.1.10 from the Connection drop-down list.

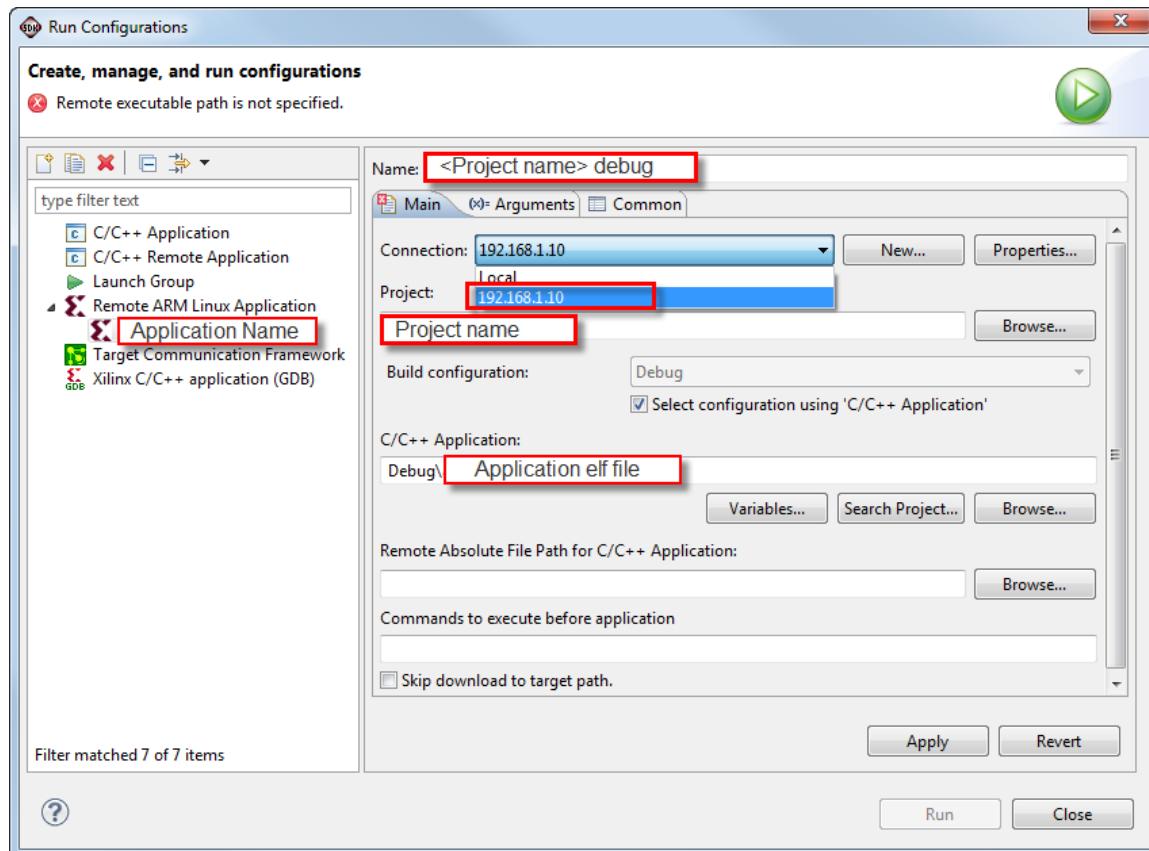


Figure 372: Selecting the IP Address

- 1-4-3.** In the Remote Absolute File Path for C/C++ Application field, enter **/tmp/your application.elf**.

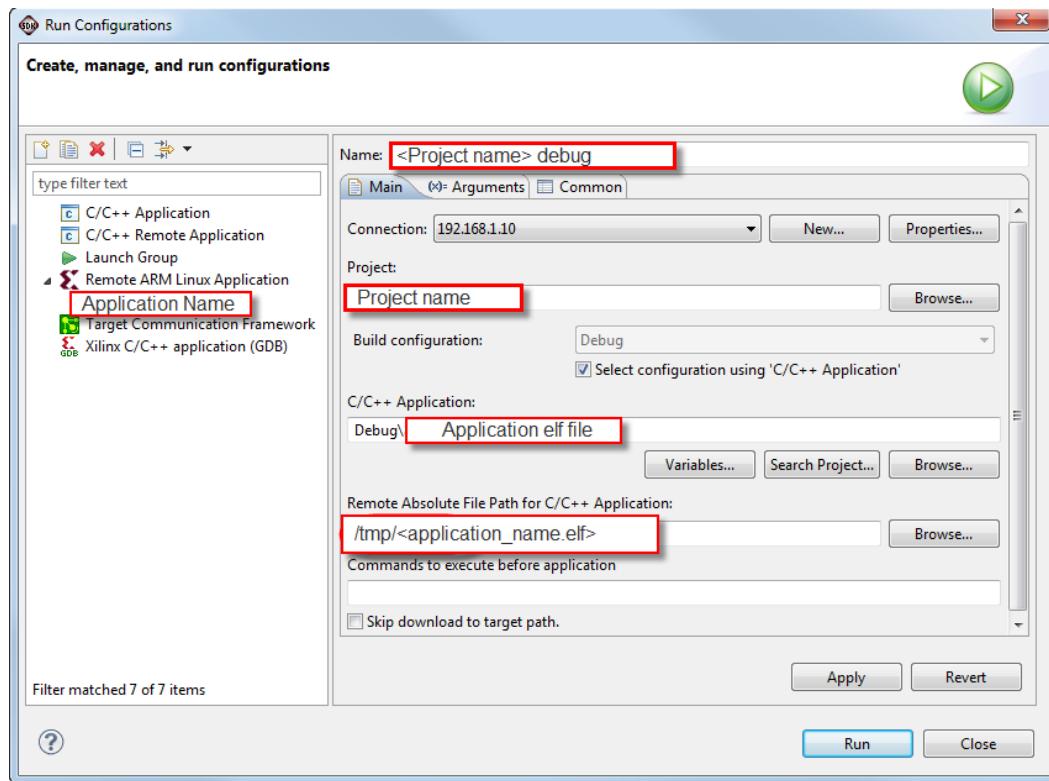


Figure 373: Setting the Remote File Path for the Application

1-5. Run the program.

- 1-5-1.** Click **Apply**.
- 1-5-2.** Click **Run**.
- 1-5-3.** Enter **root** in the User ID and Password fields.

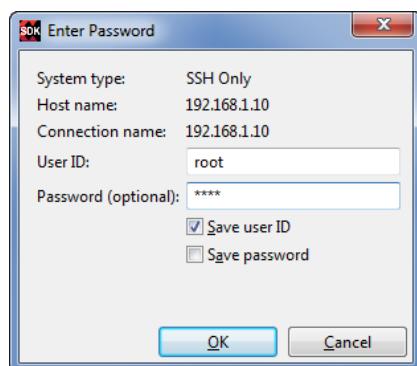


Figure 374: Entering the User ID and Password

- 1-5-4.** Click **OK**.
- 1-5-5.** Click **Run**.

Configuring the Ethernet Port for Use with the Remote System Explorer

1-1. Set your system IP to be in the range of 192.168.1.1 to 192.168.1.255.

- 1-1-1. Ensure that your system IP is not 192.168.1.10 (which is the board IP).
- 1-1-2. Select **Control Panel > Network and Sharing Center > Change adapter settings**.
- 1-1-3. Right-click the appropriate Local Area Connection and select **Properties** (1).
- 1-1-4. In the Local Area Connections Properties dialog box, double-click **Internet Protocol Version 4 (TCP/IPv4)** (2).
- 1-1-5. Select the **Use the following IP address** option (3).
- 1-1-6. Enter an IP address in the range **192.168.1.1** to **192.168.1.255** except 192.168.1.10 (which is the board IP) (3).

The Subnet mask should become updated after the IP address has been entered.

- 1-1-7. Click **OK** to close both dialog boxes.

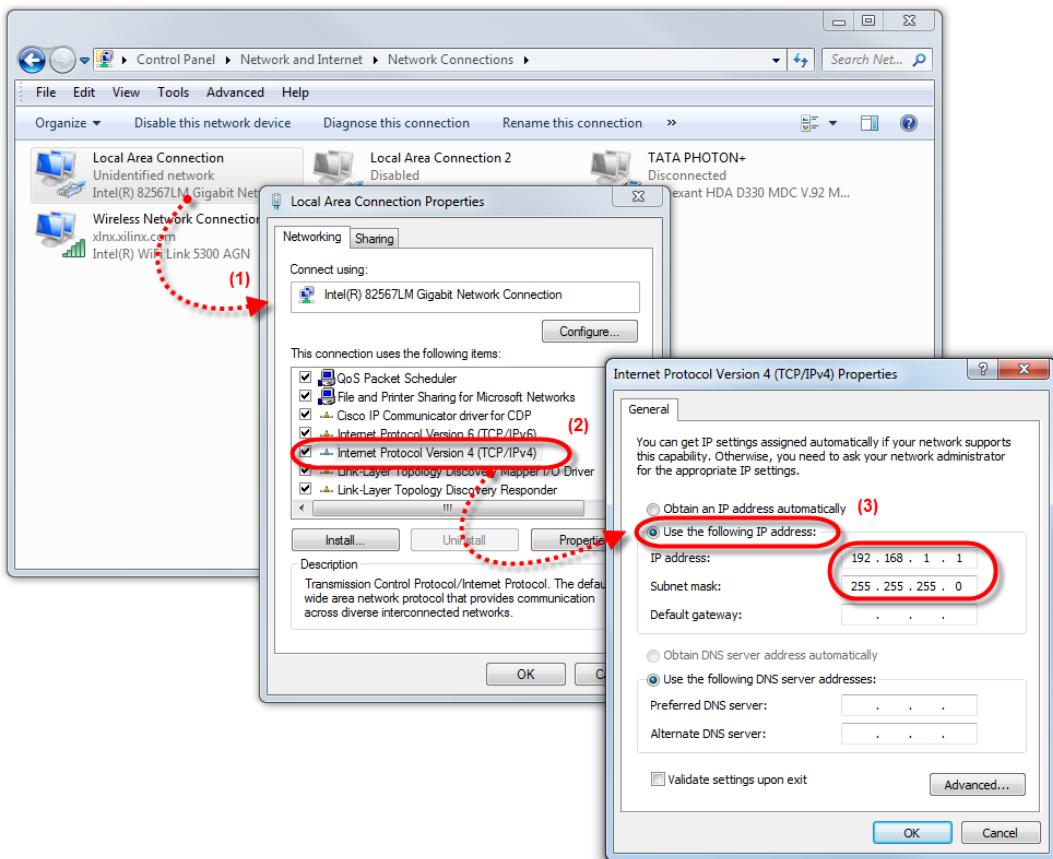


Figure 375: Configuring the Ethernet Port (RSE)

Note: You should disconnect from VPN if you are connected.

Launching a Software Application on Hardware

A Run configuration defines how you want the system to work when running an application. A Launch on Hardware menu selection will start the selected software application with a Run configuration at default settings, saving you a few mouse clicks.

1-1. Launch and execute the software application on the hardware evaluation board.

- 1-1-1. From the Project Explorer pane, right-click the application project that you want to launch (1).
- 1-1-2. Select **Run As** (2).
- 1-1-3. Select **Launch on Hardware** (3).

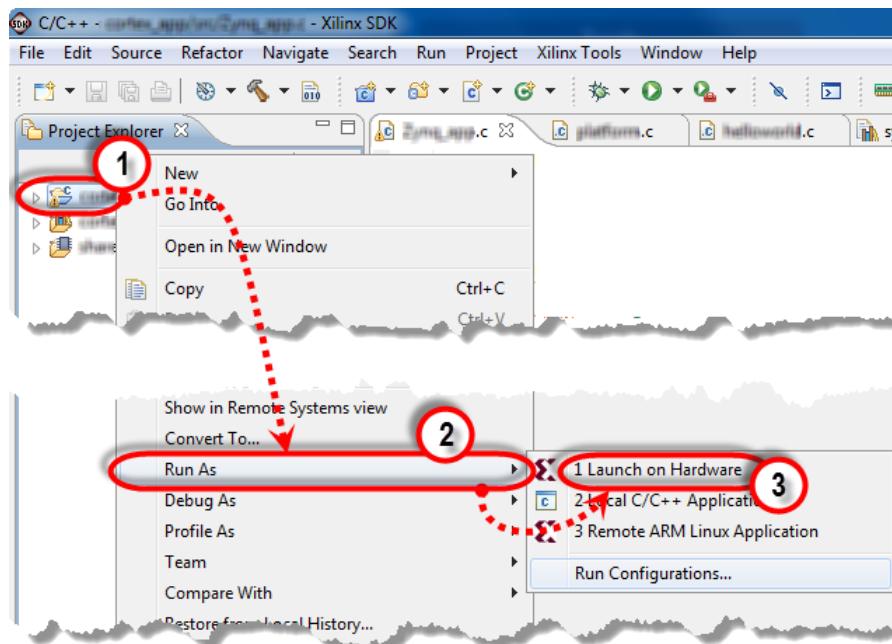


Figure 376: Selecting Launch on Hardware

The software application is downloaded into hardware and started. Details can be viewed in the Console window.

Switching Perspectives

1-1. A perspective is a collection of views that assist you in a specific task. There are several pre-defined views, including a C/C++ view for developing applications and a Debug view for debugging. You can create your own perspectives as well.

- 1-1-1.** Locate the icon for the perspective that you would like to switch to in the upper right-hand corner.
- 1-1-2.** Click the desired icon.

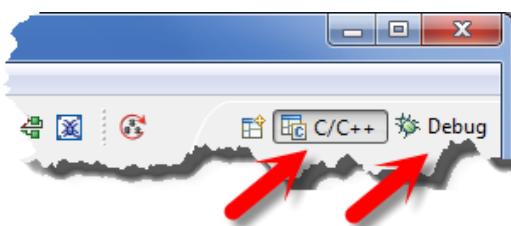


Figure 377: Clicking the Desired Icon

- 1-1-3.** Click the **Open Perspective** icon if the perspective is not readily available.
- 1-1-4.** Select the desired perspective from the list.

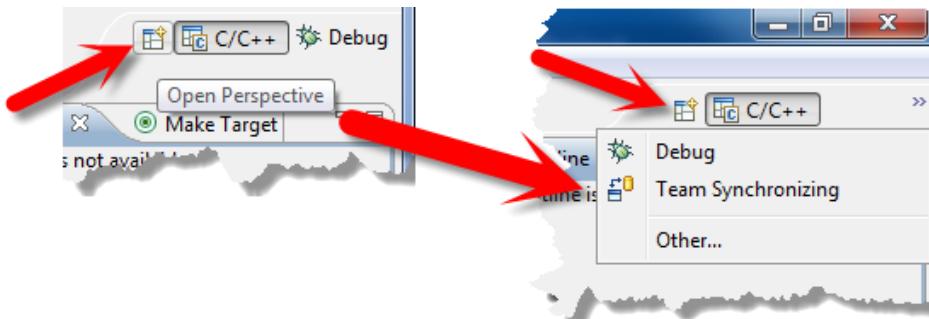


Figure 378: Selecting the Desired Perspective

Configuring the Terminal

The Terminal tab is used with a serial source (UART), SSH, or Telnet session. The following instructions will show you how to configure this tab to work with a serial UART.

1-1. Open the Terminal tab.

- 1-1-1. If the Terminal tab is not visible, select **Windows > Show View > Other > Terminal > Terminal** to open it or create a new Terminal tab.

This will open a Terminal tab in one of the panels. The tab can be moved to any other panel by click-dragging it to the new panel.

Note: More than one terminal can be enabled at a time.

- 1-1-2. Select the **Terminal** tab to access the terminal icons (1).

- 1-1-3. Click the **Settings** icon (2).

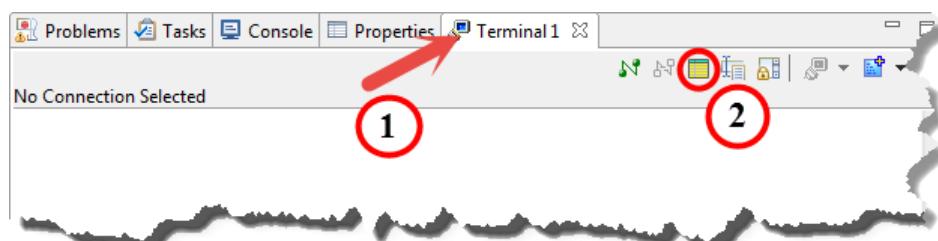


Figure 379: Accessing the Terminal Settings

Alternatively, you can also click the **Connect** icon () to open the Terminal Settings dialog box. If the terminal was previously configured, this action will not open the Terminal Settings dialog box, rather it will attempt to open the communication channel with the previously selected settings.

Changing the settings requires that you close the channel and use the Settings icon.

1-2. Configure the settings as shown in the following figure.

1-2-1. Select the connection type as **Serial**.

1-2-2. Select the proper port for the COM #.

Your board must be on and connected; otherwise the USB bridge is not enumerated and your COM port will not appear.

1-2-3. Set the baud rate to **115200**.

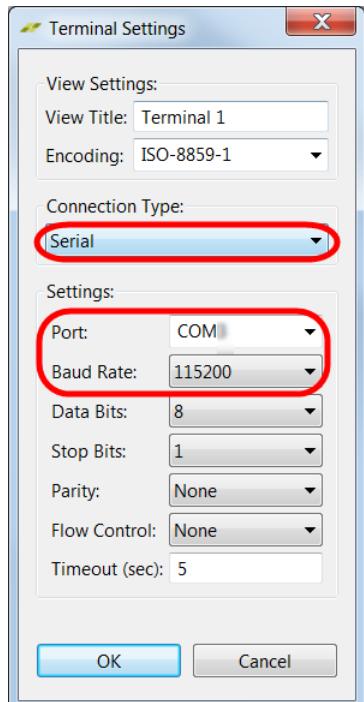


Figure 380: Configuring the Terminal Settings

1-2-4. Click **OK**.

The terminal session will be connected to the associated COM port on the PC.

Configuring the SDK Terminal

The SDK terminal is an interface that only supports serial port/UART communications. The more general Terminal tab is able to support other formats, such as SSH and Telnet.

1-1. Locate the SDK Terminal tab. Generally this tab is found in the same panel as the console.

- 1-1-1. If the SDK Terminal tab is not currently visible, select **Window > Show View > Other > Xilinx > SDK Terminal** to open it.

This tab typically opens in the same window as the console.

1-2. Configure the SDK Terminal.

- 1-2-1. Click the green '+' sign to open the Connect to Serial Port dialog box.

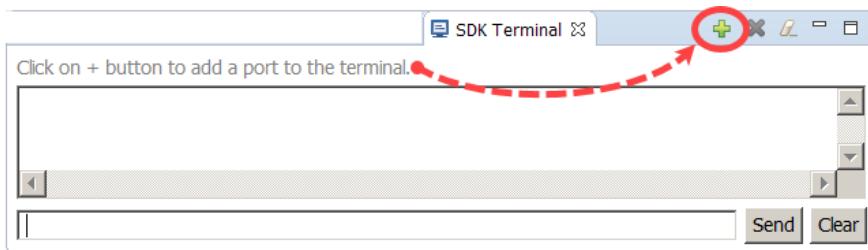


Figure 381: Adding/Associating a Port to the Terminal

A pop-up appears asking you to configure the settings for the serial port.

- 1-2-2. Select the serial port that is connected to the device you want to communicate with (1).

This is the port number associated with the Serial Port/USB connection from your board. This is often the highest-numbered com port, but not always. Your board must be powered on in order to see this port.

- 1-2-3. Set the baud rate to **115200** (2).

- 1-2-4. Leave the other settings at their defaults.

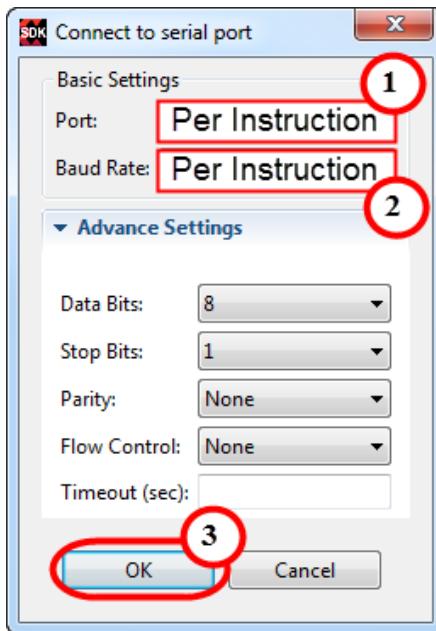


Figure 382: Configuring the SDK Terminal

- 1-2-5. Click **OK** to save these settings and begin the terminal session (3).

Terminating a Running Application

- 1-1. Terminate the running application from the SDK Console tab.

- 1-1-1. Select the **Console** tab.

- 1-1-2. Click the **Terminate** icon () to terminate the execution of the running application.

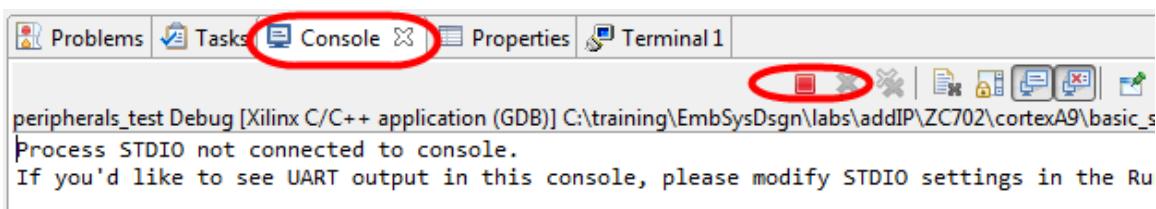


Figure 383: Terminating the Application

While not required, terminating application program execution will avoid a *running* message if an attempt is made to begin running the same or different application program while one is presently executing.

Closing the Xilinx SDK Tool

1-1. Close the Xilinx SDK tool.

- 1-1-1. Select **File > Exit** to save the SDK configuration and exit the tool.

If you have not configured your workspace to not request confirmation on close, a dialog box will appear confirming that you wish to close SDK.

You have the option to set SDK to never ask you for exit confirmation.

- 1-1-2. Click **Yes** if this dialog box appears to conclude the saving of the SDK configuration and exiting of the tool.

QEMU Operations [Last Updated Version 2019.1]

In This Section

Configuring QEMU from SDK.....	293
Opening the VirtualBox and Running an OS.....	295
Running an Application on QEMU from SDK.....	296
Selecting the QEMU Console Log.....	297
Setting Up a (QEMU) System Debugger Debug Configuration	298
Stopping QEMU.....	301
Starting the QEMU.....	301
Shutting Down a Running QEMU Virtual Machine Using the Command Prompt	302

Configuring QEMU from SDK

1-1. Configure QEMU.

- 1-1-1. Select **Xilinx > Configure QEMU Settings**.

The Configure QEMU Settings window opens.

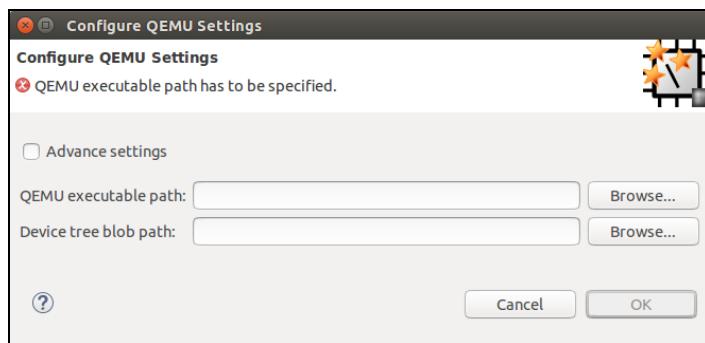


Figure 384: Configuring QEMU Settings

- 1-1-2.** Click **Browse** next to the *QEMU executable path* field to open a file selection dialog box.

Since QEMU is not part of the Xilinx package, you will need to move to the root of the directory structure, then burrow down to the location of the QEMU executable.

- 1-1-3.** Click **File System** from the left panel titled *Places*, or if there is a panel titled *Devices*, click the **Computer** icon to navigate to the root of the directory structure.

- 1-1-4.** Navigate to the `$XILINX_PATH/SDK/$VERSION/bin` directory.

This is easily done by double-clicking each directory in the path as it appears.

- 1-1-5.** Select the QEMU executable file **name of the QEMU executable**.

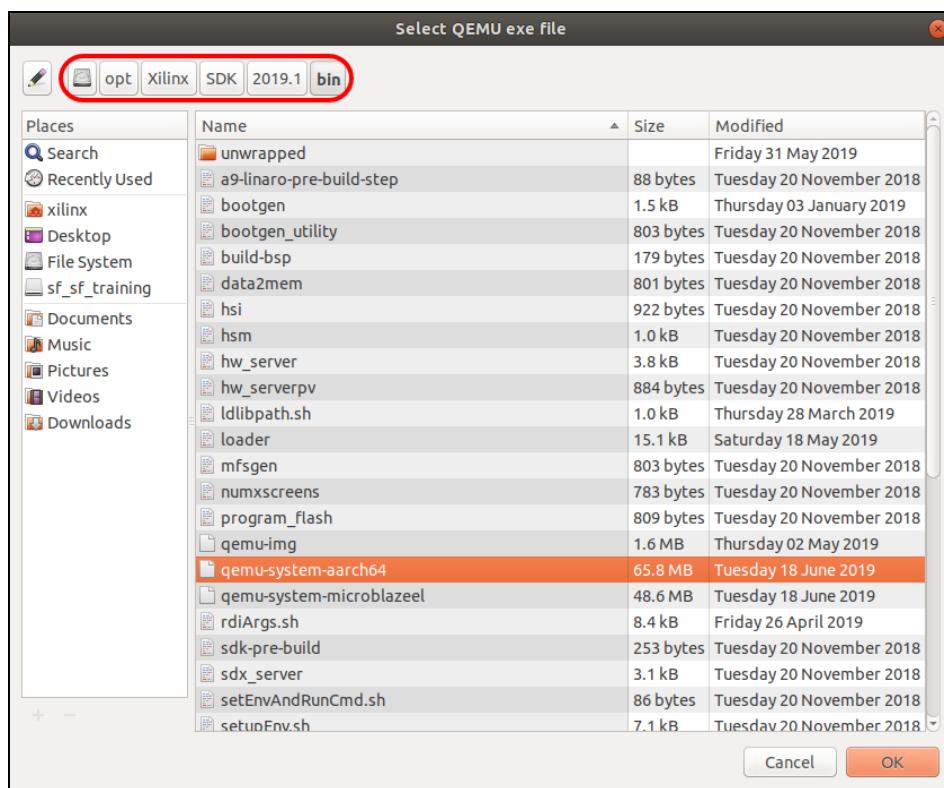


Figure 385: Selecting the QEMU EXE File

- 1-1-6.** Click **OK** to close the file selection dialog box.

- 1-1-7.** Click **Browse** next to the *Device tree blob path* field to open a file selection dialog box.

- 1-1-8.** Click the **xilinx** entry under the *Places* quick navigation area.

The xilinx directory is your home directory. You will be able to quickly navigate to the tools directory where the device tree blob is provided for your convenience.

- 1-1-9.** Navigate to the `$TOOLS` directory.

This directory is equivalent to `$TRAINING_PATH/tools`.

- 1-1-10.** Select the device tree blob **name of DTB**.

- 1-1-11.** Click **OK** to close the file selection dialog and return to the Configure QEMU Settings dialog box.
- 1-1-12.** Click **OK** to close the QEMU configuration window.

Opening the VirtualBox and Running an OS

VirtualBox is a software tool that enables users to run different operating systems under an existing host operating system.

1-1. Open VirtualBox to run the the hosted OS operating system on this host.

- 1-1-1.** Select **Start > All Programs > Oracle VM VirtualBox > Oracle VM VirtualBox** or double-click the **VirtualBox** icon ().

VirtualBox opens and lists operating systems available for launching on the left side of the pane.

If a pop-up window appears informing you that there is a newer version of VirtualBox available, click **Skip** to continue with the version provided to you.

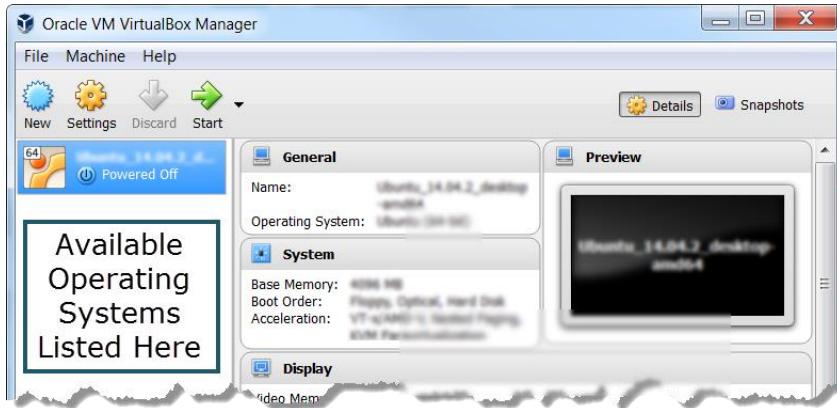


Figure 386: Opening VirtualBox

- 1-1-2.** Double-click the **the hosted OS** operating system to load it.

This will cause a new window to open showing the interface to the selected operating system.

- 1-1-3.** Review the notices (if any) that appear at the top of the window.

If these are not significant, close the messages.

An example of Ubuntu Linux is shown below.

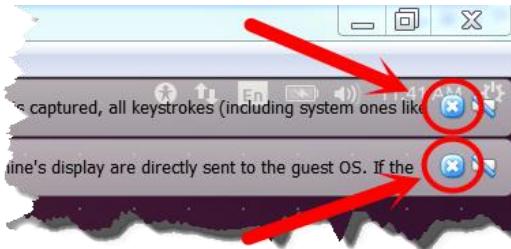


Figure 387: Closing Information Messages

Running an Application on QEMU from SDK

You will now configure the application to run on QEMU.

1-1. Run *your application* with the default configuration settings.

- 1-1-1. Right-click **your application** in the Project Explorer to open the context menu.
- 1-1-2. Select **Run As > Run Configurations**.
- 1-1-3. Select **Xilinx C/C++ application (System Debugger on QEMU)** to indicate which type of configuration to create (1).
- 1-1-4. Click the **New Launch Configuration** icon to create the configuration (2).

This will create the specific configuration (3).

You can review the automatically filled fields under the various tabs.

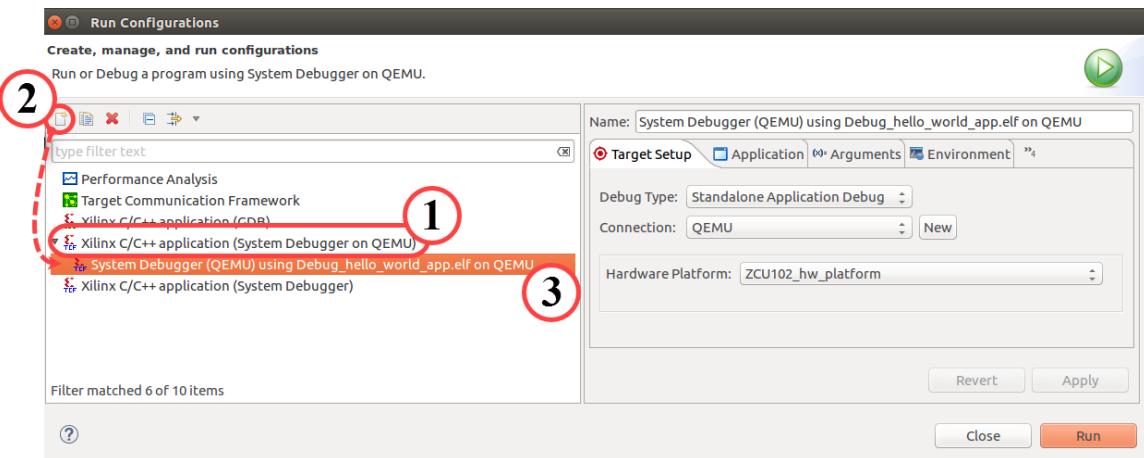


Figure 388: Verifying the Target Setup Tab (Example)

1-1-5. Select the **Application** tab to view the details for the application (1).

1-1-6. Select **the processor you wish to target** in the list box (2).

No changes are required for this project; however, it is worth noting how the various applications can be mapped to processors.

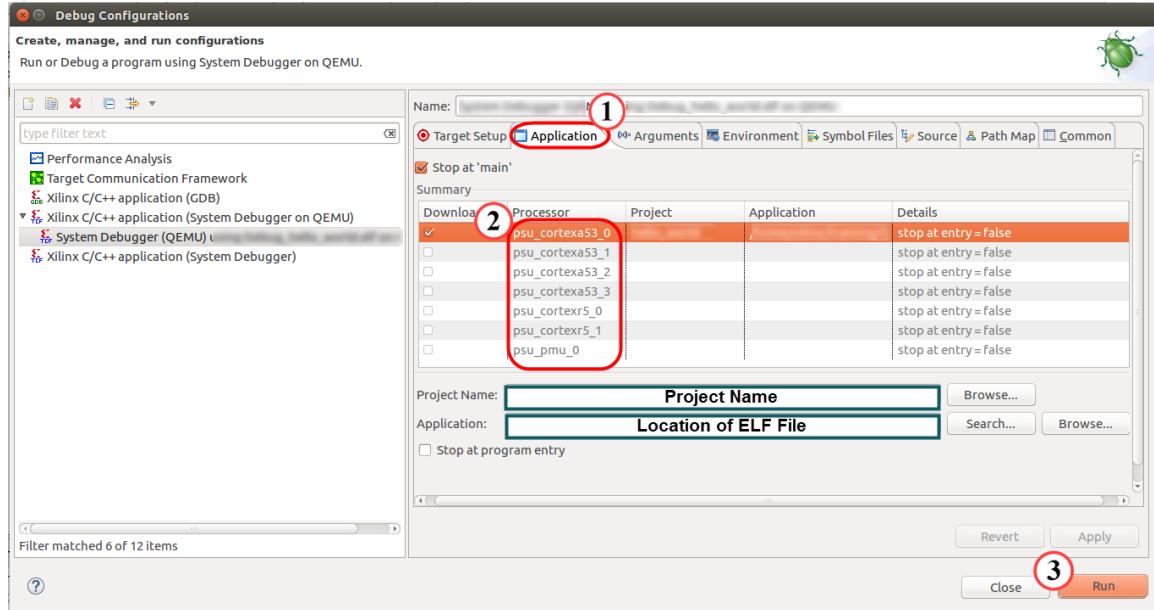


Figure 389: Verifying the Application Tab

Note: It is possible to retarget the processor that the application will execute on by selecting a different processor and configuring it accordingly.

1-1-7. Click **Run** to launch QEMU and run the applications listed in the QEMU emulator (3).

You should see a dialog box open indicating that QEMU is being configured and the results posted to the serial port will appear in the Emulation Console.

Selecting the QEMU Console Log

The QEMU Console, sometimes referred to as the emulation console, displays the information coming from the QEMU emulator. Here you will open this normally hidden view.

1-1. Select the emulation console.

- 1-1-1. Select **Window > Show View > Other** to open the list of views.

A dialog box will open with a list of available views.

- 1-1-2. Expand the **Xilinx** folder to see all of the Xilinx-specific options.

- 1-1-3. Select **Emulation Console** to open a view to the emulation console.

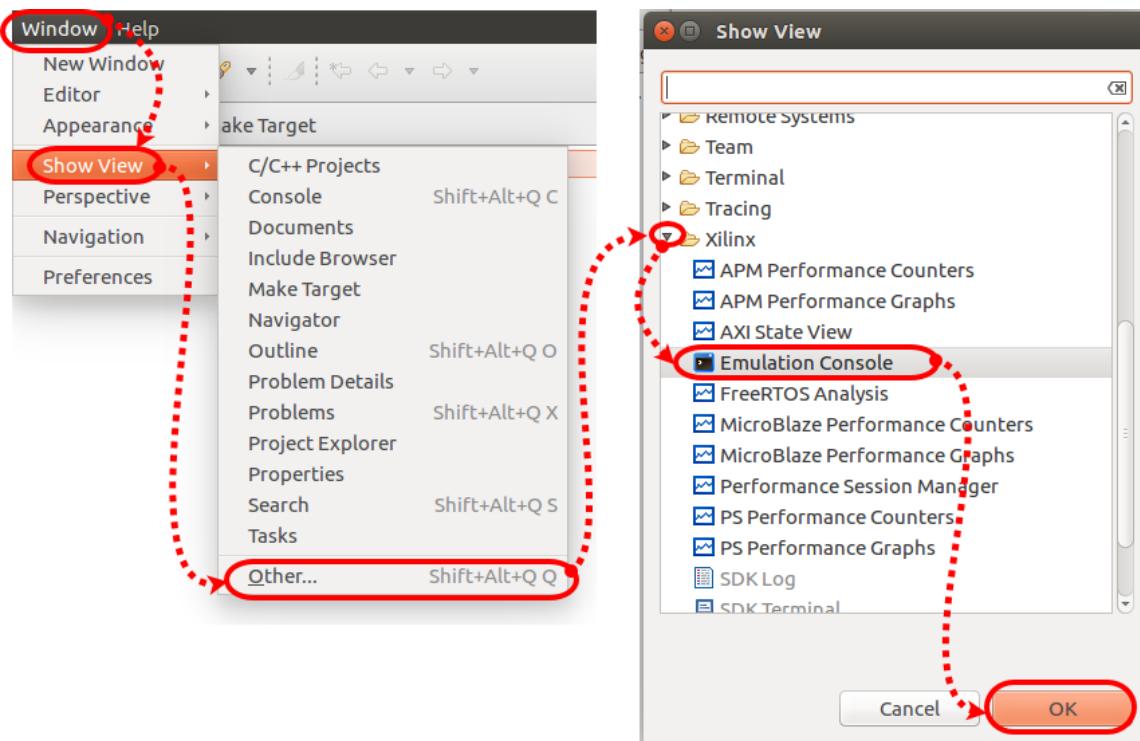


Figure 390: Selecting Emulation Console

- 1-1-4. Click **OK** to close the dialog box.

The QEMU Console log is now visible in the C/C++ perspective, usually in the lower-right corner.

Setting Up a (QEMU) System Debugger Debug Configuration

A Debug configuration defines how you want the system to work when performing a debug operation and maps an ELF object file to a target for execution. Typically, a debug operation switches to the Debug perspective. While there are a significant number of switches and options, the most common are shown below.

1-1. Set up a debug configuration for a specific application project.

- 1-1-1. Right-click **your project name** in the Project Explorer pane that you want to create the Debug configuration for (1).
- 1-1-2. Select **Debug As** to open the sub-menu of predefined configurations and the configuration manager (2).
- 1-1-3. Select **Debug Configurations** to view all the available debug options (3).

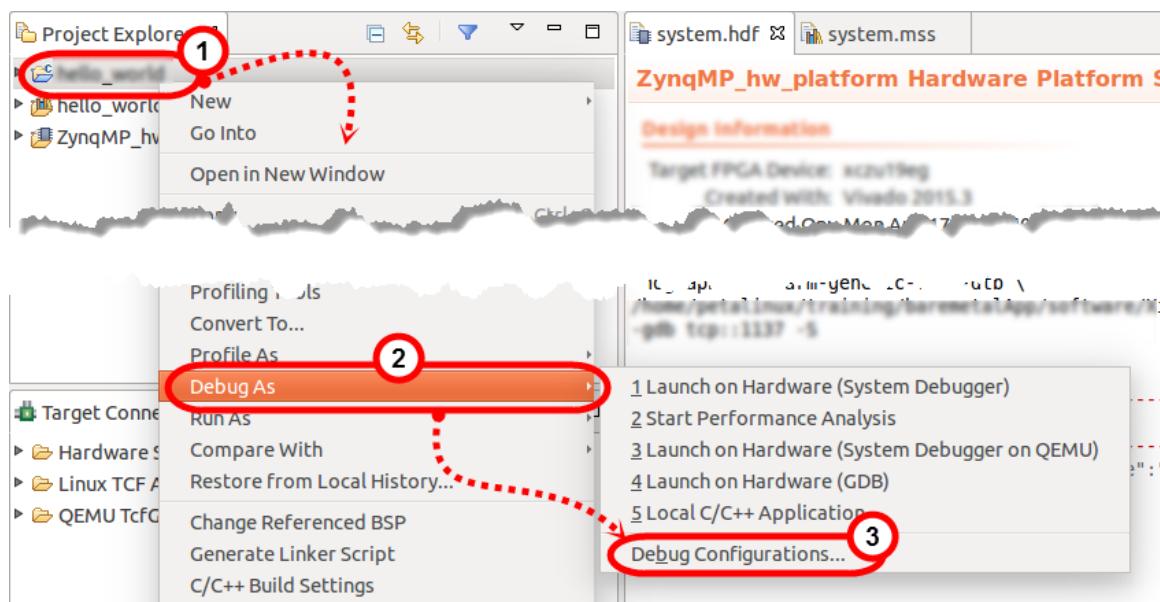


Figure 391: Creating a Debug Configuration (QEMU)

The Debug Configurations dialog box opens.

- 1-1-4. Double-click **Xilinx C/C++ application (System Debugger on QEMU)** to create a new configuration for your application.

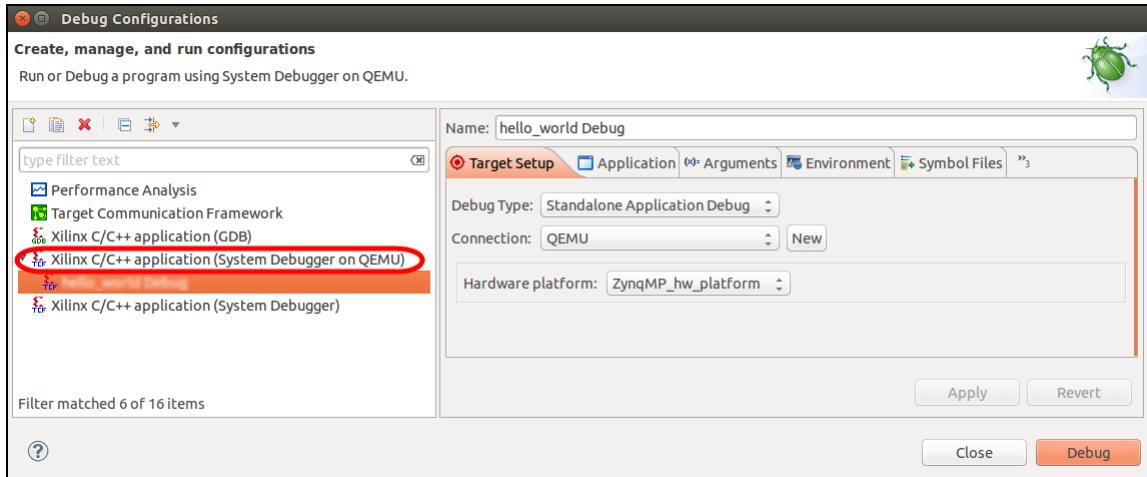


Figure 392: Creating a New Debug Configuration (QEMU)

Note: Some tool versions will list the hardware platform as ZynqMP_ZCU102_hw_platform. This will work as well.

Note: A new Debug configuration is created. Note the default configuration name and other parameters that are automatically filled in. In most cases, you just need to click the Debug button to begin the session. This Debug configuration menu is useful when you want to later change debug parameters. The new configuration will appear with other existing configurations and have the name of your application.

You should also note that a number of fields are automatically filled in for you using the name of your application as the basis (that is, if your application is named "XYZ", then the Name field will be populated with XYZ Debug, and the C/C++ Application field will be populated with Debug\XYZ.elf).

- 1-1-5. Click **Debug** to close the window and launch the debugging session.
1-1-6. Click **Yes** if the Confirm Perspective Switch dialog box appears.

Optional: Select the **Remember my decision** option to prevent this dialog box from opening when switching to the debug perspective.

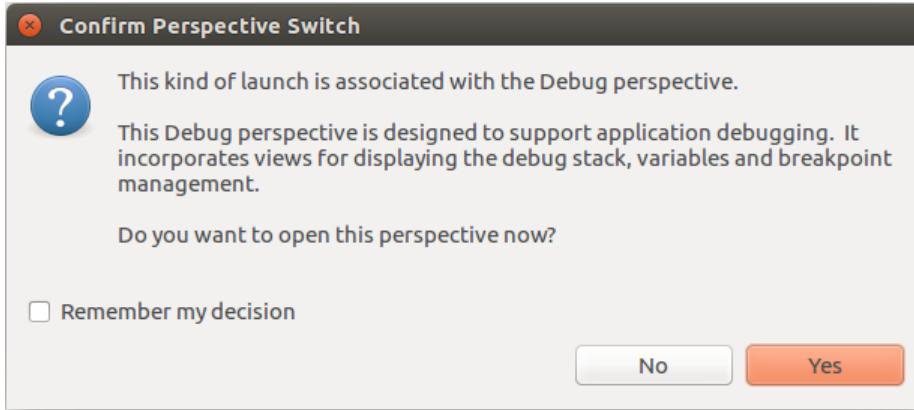


Figure 393: Confirm Switch Perspective

The Debug Perspective view opens.

Stopping QEMU

1-1. Stop the QEMU.

1-1-1. Select **Xilinx Tools > Configure QEMU Settings**.

1-1-2. Click **Stop QEMU**.

Note: The QEMU Status should change to Stopped.

1-1-3. Click **OK** to close the Configure QEMU Settings window.

Starting the QEMU

1-1. Start the QEMU.

1-1-1. Select **Xilinx Tools > Configure QEMU Settings**.

1-1-2. Click **Start QEMU**.

Note: The QEMU Status should change to Running.

1-1-3. Click **OK** to close the Configure QEMU Settings window.

Shutting Down a Running QEMU Virtual Machine Using the Command Prompt

1-1. Shut down the running QEMU virtual machine.

- 1-1-1. Enter the following from the command line of the virtual machine (only for Linux):

```
[host]$ poweroff
```

Several system messages will be displayed in the console as the Linux instance is shutting down.

- 1-1-2. Enter the key combination <Ctrl + A> followed by X to terminate the QEMU session.

SDx Tool Operations [Last Updated Version 2018.3]

In This Section

Launching the SDx Tool.....	303
Launching the SDx Tool and Creating an SDx Tool Project.....	304
Launching the SDx Tool and Creating an SDx Tool Project [Simplified].....	308
Importing Source Files to an Application	309
Importing an Existing Project into the SDx Tool	311
Importing an Existing Project into the SDx Tool	313
Highlighting a Process for Debug	314
Configuring the Board Settings.....	315
Configuring the Board Settings for Linux	320
Setting the Build Configuration to SDDebug.....	323
Launching a Debug Session.....	323
Setting Up a System Debugger Debug Configuration (Standalone)	325
Setting Up a System Debugger Debug Configuration (Linux).....	329
Setting the Build Configuration to SDDebug.....	337
Creating a C/C++ SDSoc Tool Project.....	337
Configuring the Terminal.....	339
Configuring the PC's Ethernet Port for Remote System Explorer.....	340
Generating the Software Estimate from the SDSoc Tool	341
Opening a Previously Run Performance Estimate	344
Removing Breakpoints.....	345
Adding a Watchpoint to a Global Variable	346
Selecting a Build Configuration.....	349
Marking a Function for Hardware (Dashboard Method)	349
Viewing the Generated System Hardware.....	350
Viewing the Generated Accelerator	351
Cleaning and Building a Project	351

Launching the SDx Tool

1-1. Launch the SDx design environment and select a workspace.

- 1-1-1.** Select **Start > All Programs > Xilinx Design Tools > SDx 2019.1 > SDx IDE 2019.1** to launch the SDx design environment.

Alternatively, you can launch the SDx tool from the desktop shortcut () , if available.

The Workspace Launcher opens after a moment.

The SDx tool creates a workspace environment that initially only contains a thin structure that tracks tool settings and maintains a log file. As projects are added, this workspace will update to include all types of supported projects. Workspaces can be switched from within the tool by selecting **File > Switch Workspace**.

If it becomes necessary to move a software application to another location or computer, use the import and export feature. Manually copying files is not recommended as workspace files are set to use absolute path names and may cause the tool to become unstable.

Remember to keep the path names short as long paths may cause problems on Windows-based machines. Place your project at the root level or one hierarchical level below to help keep the path name short is recommended.

- 1-1-2.** Enter **your workspace location** into the workspace field when the Workspace Launcher opens, or use the Browse button.

Note that when you use the Browse button, you will need to select the **your workspace location** directory and click **OK**.

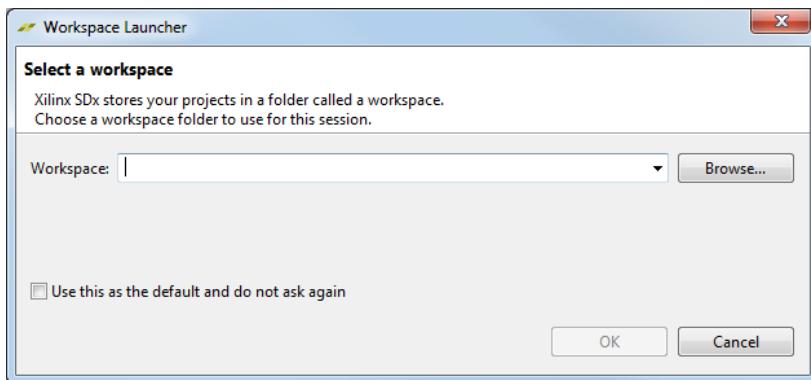


Figure 394: Selecting a Workspace

- 1-1-3.** Click **OK** to close the Workspace Launcher dialog box and open the new workspace.

- 1-1-4.** Close the Welcome tab if it appears.

This will give you more room to view your project. You may also wish to maximize the SDx window, there will be a lot to see.

Launching the SDx Tool and Creating an SDx Tool Project

1-1. Launch the SDx design environment and select a workspace.

- 1-1-1. [Windows 7 users]: Select **Start > All Programs > Xilinx Design Tools > SDx 2019.1 > SDx IDE 2019.1** to launch the SDx design environment.

[Windows 10 users]: Select **Start > Xilinx Design Tools > SDx IDE 2019.1** to launch the SDx design environment.

[Linux users]: Press **<Ctrl + Alt + T>** to launch a Terminal, type **sdx**, and press **<Enter>** to launch the SDx design environment.

- 1-1-2. [Windows users]: Enter **your workspace location** into the Workspace field when the Eclipse Launcher opens, or use the Browse button.

Note that when you use the Browse button, you will need to select the **your workspace location** directory and click **OK**.

[Linux users]: Enter **/home/xilinx/training/<the topic cluster name>/lab** into Workspace field when the Eclipse Launcher opens, or use the Browse button.

Note that when you use Browse button, you will need to select the **/home/xilinx/training/<the topic cluster name>/lab** directory and click **OK**.

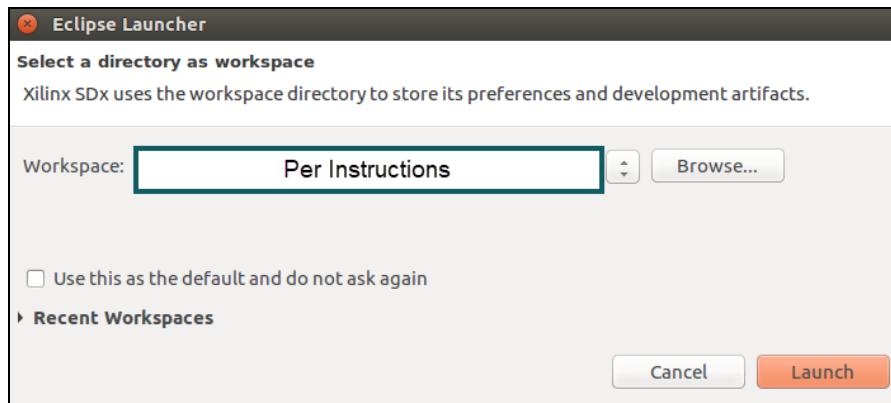


Figure 395: Selecting the Workspace

- 1-1-3. Click **Launch** to close the Eclipse Launcher dialog box and open the new workspace.

1-2. Create a new SDx tool application project.

1-2-1. Click **Create Application Project** (1).

1-2-2. Enter **your project name** in the Project name field.

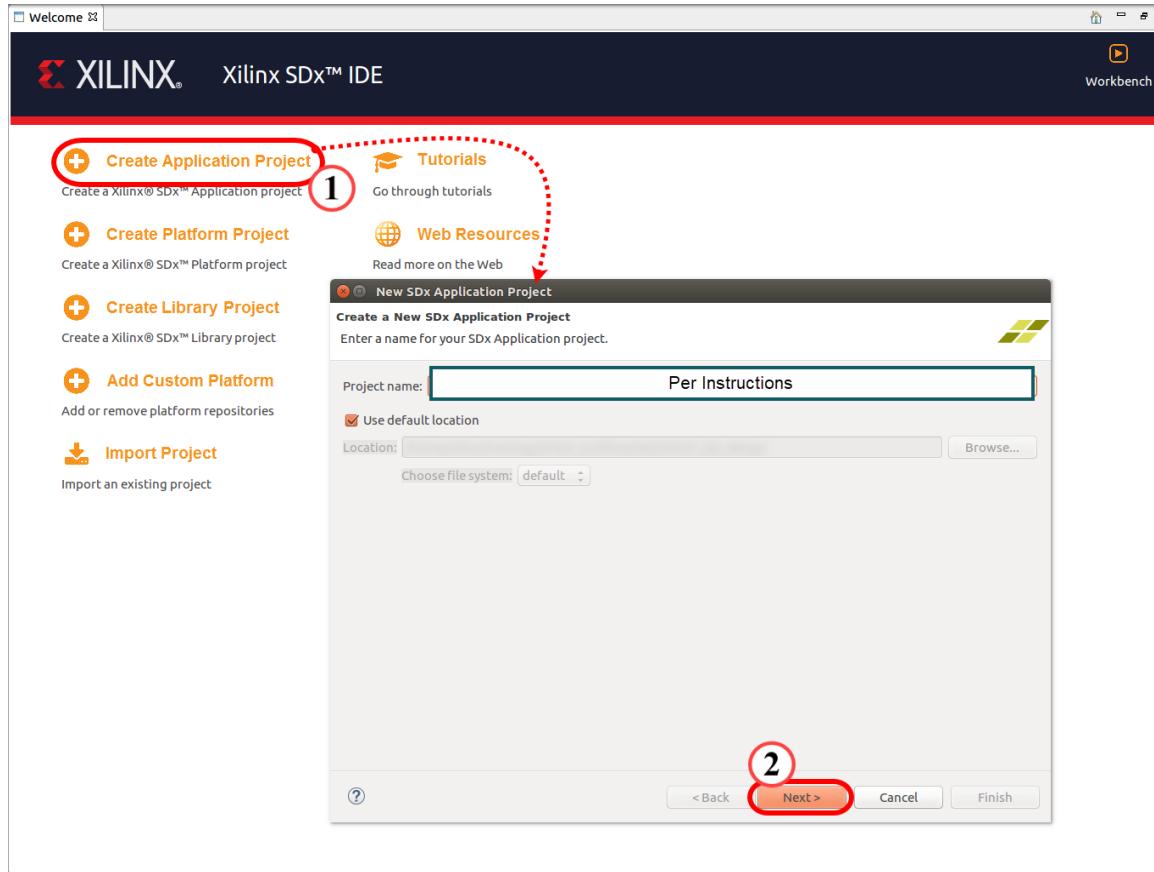


Figure 396: Creating an SDx Project

1-2-3. Click **Next** (2).

1-3. Choose the platform that will execute your application.

1-3-1. Select **your board** from the board list.

Alternatively, you can type the board name in the Find field to quickly select the board, which will be listed.

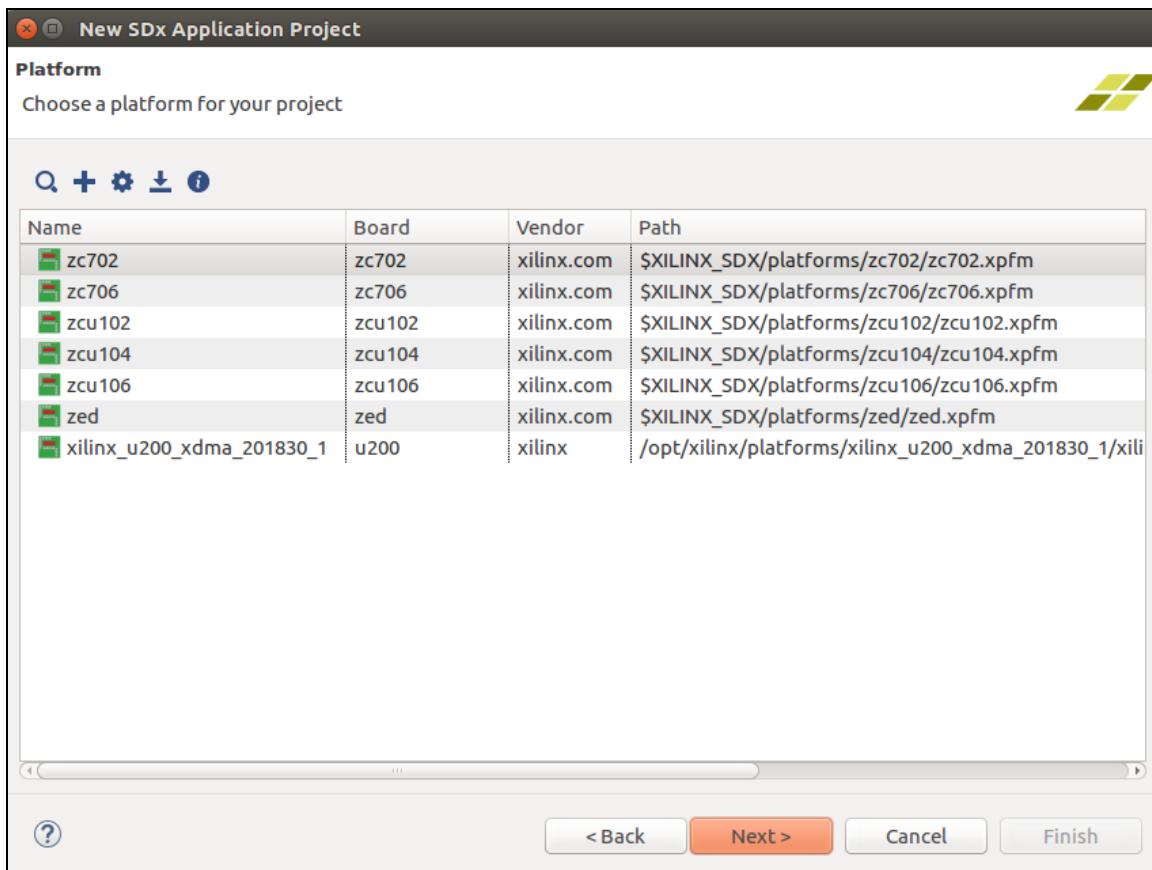


Figure 397: Selecting the Platform

1-3-2. Click **Next**.

1-4. Now provide the system configuration details for your project.

- 1-4-1. Select **your desired operating system** from the System configuration drop-down list based on your Software Platform preference (1).

Notice that the **Domain** and **Operating System** has been updated automatically based on your System Configuration selection (2).

- 1-4-2. Verify the processor that is mentioned in the CPU field under Domain.

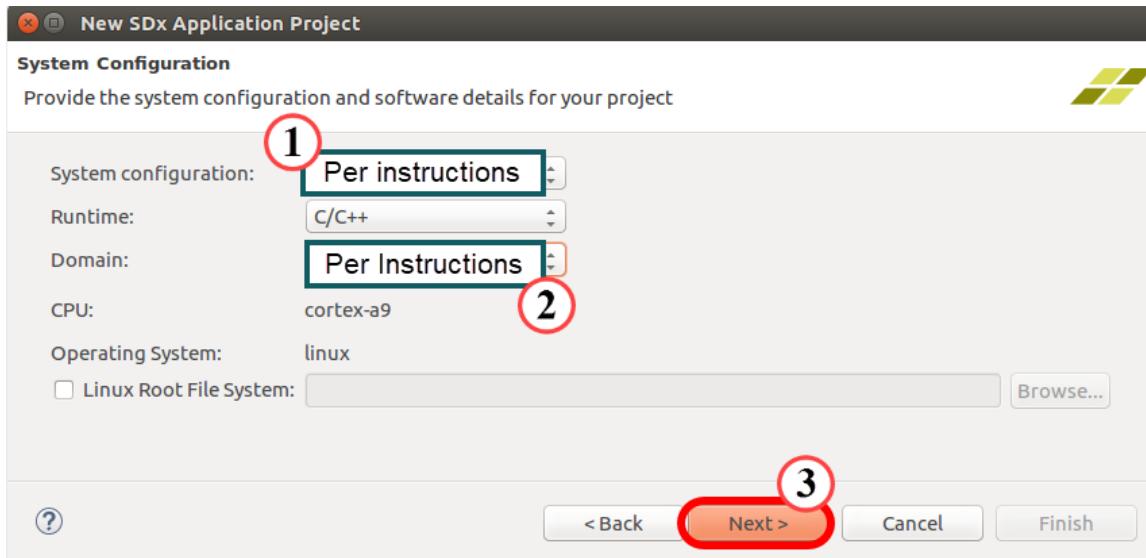


Figure 398: Selecting Target Platform and Target CPU (Standalone)

- 1-4-3. Click **Next** (3).

If you have selected Linux as the system configuration, leave the additional settings at their default.

1-5. Select an application template.

- 1-5-1. Select **an application template** from the Available Templates section.

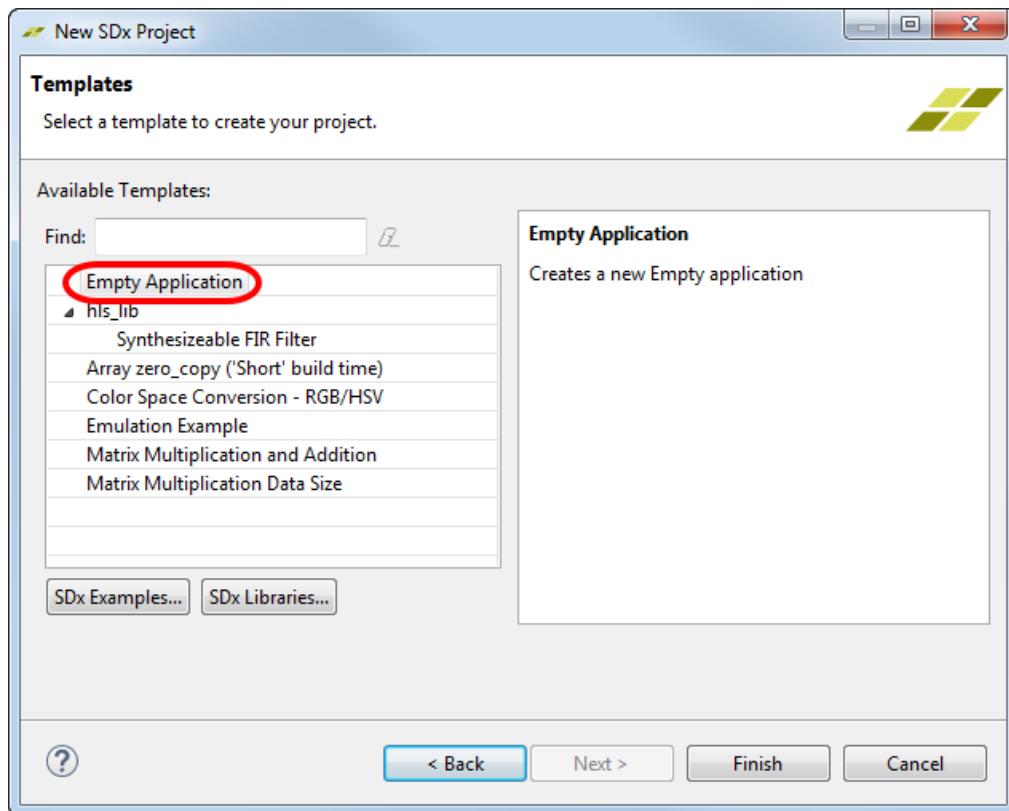


Figure 399: Selecting the Application from the Available Template

- 1-5-2. Click **Finish**.

The SDx IDE environment opens.

Launching the SDx Tool and Creating an SDx Tool Project [Simplified]

- 1-1. Launch the SDx tool and create a project, called **your project name**, using the Empty Application template and targeting the your board.
- 1-1-1. [Windows 7 users]: Select Start > All Programs > Xilinx Design Tools > SDx 2019.1 > SDx IDE 2019.1 to launch the SDx design environment.
[Windows 10 users]: Select Start > Xilinx Design Tools > SDx IDE 2019.1 to launch the SDx design environment.
[Linux users]: Press <**Ctrl + Alt + T**> to launch a Terminal, type **sdx**, and press <Enter> to launch the SDx design environment.
- 1-1-2. [Windows users]: Select **your workspace location** as the workspace and click **Launch**.
[Linux users]: Select **/home/xilinx/training/******/lab** as the workspace and click **Launch**.
- 1-1-3. Create a new SDx application project called **your project name** and click **Next**.
- 1-1-4. Select **your board** (depending on the board you are using) and click **Next**.
- 1-1-5. Select **your desired operating system** from the System configuration drop-down list and click **Next**.
- 1-1-6. Select **Empty Application** and click **Finish**.

This opens the SDx IDE environment.

Importing Source Files to an Application

1-1. Add the source files to the design.

- 1-1-1. Expand the project named **PMU_Extended_app** > **src** using the Project Explorer.
- 1-1-2. Right-click the **src** directory in the project to place the resource files (1).
- 1-1-3. Select **Import** to open the Import Wizard (2).

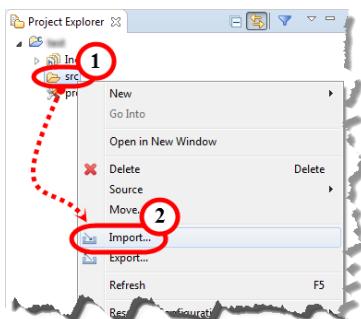


Figure 400: Importing a Resource File

The Import Wizard dialog box opens.

1-1-4. Expand **General** (1).

1-1-5. Select **File System** as you will be selecting individual files directly from the file system (2).

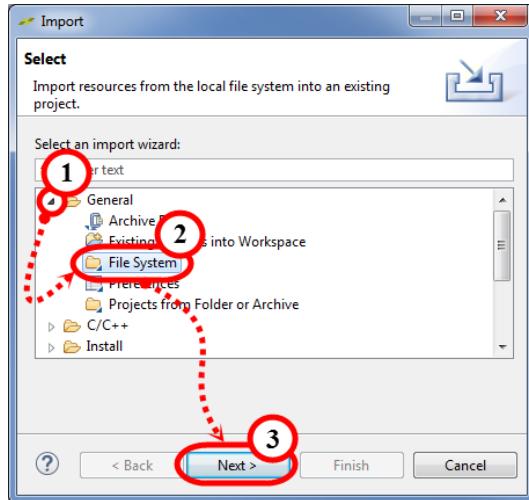


Figure 401: Selecting File System

1-1-6. Click **Next** to specifying the files to import (3).

1-1-7. **[Windows users]:** Browse to *PMU_Extended_app\support* in the *From directory* field.

[Linux users]: Browse to */home/xilinx/training/<the topic cluster name>/support* in the *From directory* field.

1-1-8. Select the file(s) by checking the box beside **ping_pong.c**.

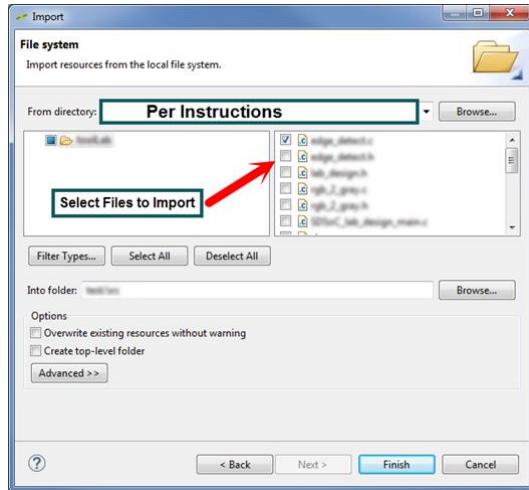


Figure 402: Selecting Resource Files

The *Into folder* directory will default to the location selected when you engaged the import function, but you can click **Browse** to change this location.

1-1-9. Click **Finish** to import the selected files and close the wizard.

Note: If the workspace has the automatic build option enabled, the project will automatically build with the new resource files. The Console view at the bottom of the IDE will show the results of the build.

Importing an Existing Project into the SDx Tool

One or more projects can be exported as a single zipped file. This is convenient when passing projects or parts of projects to teammates or clients. Once the recipient has received this archive, the zip file must be processed and one or more projects can then be imported.

1-1. Import an existing project.

1-1-1. Select **File > Import** to open the Import Wizard.

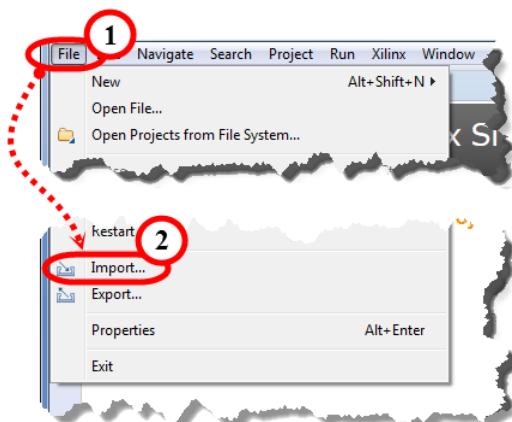


Figure 403: Accessing the Import Wizard

The Import dialog box opens.

1-1-2. Expand the **General** node to access the commonly used methods (1).

1-1-3. Select **Existing Projects into Workspace** as the goal is to import an existing project (2).

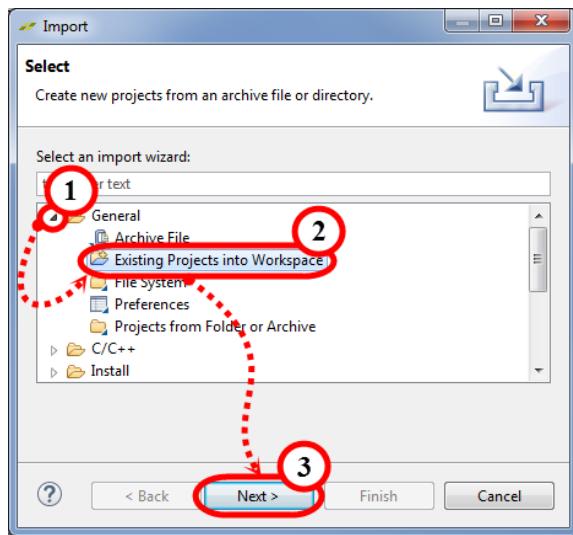


Figure 404: Choosing to Import an Existing Project into the Workspace

1-1-4. Click **Next** to enter project-specific data (3).

1-1-5. Select the **Select archive file** option (1).

Note that projects can be archived either as single zip files, or you can import from a directory. Typically, projects are preserved as archives as they are easier to move.

1-1-6. Click **Browse** to navigate to *where your SDK project archive file is located* (2).

1-1-7. Select **your SDK project archive file**, which contains the archived projects.

1-1-8. Click **Open** to open the archive and list the various projects in that archive.

1-1-9. Select **the projects from within the archive** to import (3).

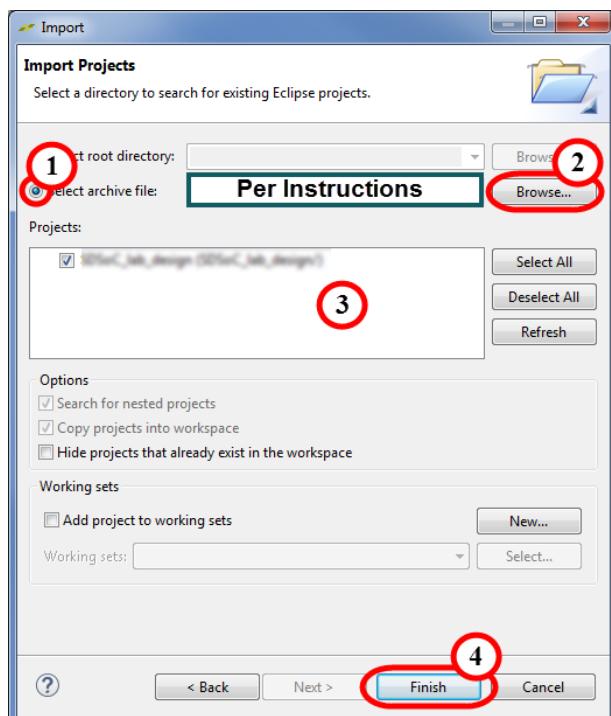


Figure 405: Selecting the Archive File

1-1-10. Click **Finish** to perform the importing of the project (4).

Importing an Existing Project into the SDx Tool

1-1. Import an existing project.

1-1-1. Select **File > Import** to open the Import Wizard.

1-1-2. Expand **General > Existing Projects into Workspace** and click **Next**.

1-1-3. Select the **Select archive file** option.

1-1-4. **[Windows users]:** Click **Browse** to navigate to *where your SDK project archive file is located*.

[Linux users]: Click **Browse** to navigate to the directory where your files are located.

- 1-1-5. Select **your SDK project archive file** and click **Open** to open the archive and list the various projects in that archive.
- 1-1-6. Select **the projects from within the archive** to import and click **Finish**.

Highlighting a Process for Debug

- 1-1. **Optional for Linux (System configuration for Board): Highlight the process for debug.**

Sometimes the process for debugging is not highlighted in the Debug view and the flow controls such as pause, resume, single step, etc. will be grayed out. Follow the procedure below to enable the controls.

- 1-1-1. Expand the Debug view, if it is not already expanded, to expand the top-level debug session (1).
- 1-1-2. Expand **/tmp/your project name.elf**, if it is not already expanded, to expand the next level showing the binary that is being executed (2).

Note: The location of the process on the remote device (/tmp/your project name.elf) is also visible.

- 1-1-3. Click the process that is currently suspended (3).

Note: This will tell the Debug tools which remote application/process to debug and enable the flow control tools.

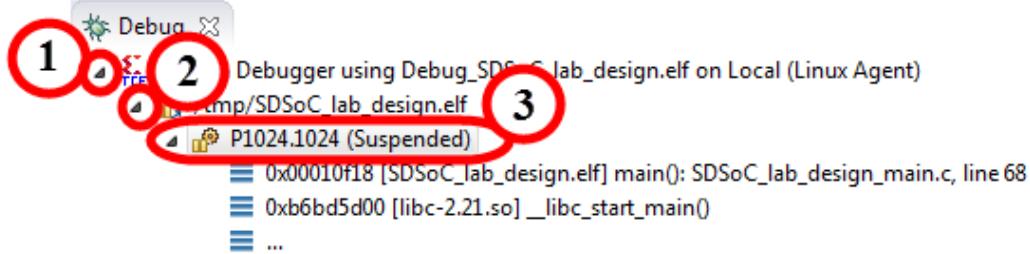


Figure 406: Highlighting the Debug Session Application

Tip: In the figure above and below, P1024 is the process ID (PID) of the application running under Linux; your PID may be different. Issuing the `ps` command through the terminal will show all of the system processes that are currently running. There will be a listing with the same PID that matches the number of your suspended application.

```

root@xilinx-zc702-2017_4:~# ps
  PID  USER      TIME  COMMAND
  1 root      0:01  init [S]
  2 root      0:00  [kthreadd]
  3 root      0:00  [ksoftirqd/0]
  4 root      0:00  [kworker/0:0]
  5 root      0:00  [kworker/0:80]
  6 root      0:00  [kworker/4:0]
  7 root      0:00  [rcu_preempt]
  8 root      0:00  [rcu_sched]
  9 root      0:00  [rcu_bh]
 10 root     0:00  [migration/0]
 11 root     0:00  [rcu_main]
 13 root      0:00  [bioset]
 41 root      0:00  [spio1]
545 root      0:00  [bioset]
550 root      0:00  [bioset]
555 root      0:00  [bioset]
560 root      0:00  [bioset]
695 root      0:00  [IRQ/38-nec81]
724 root      0:00  [ip6_addrconf]
725 root      0:00  [kworker/1:2]
737 root      0:00  [bioset]
738 root      0:00  [mcqd/0]
784 root      0:00  /sbin/udevd -d
1207 root      0:00  /usr/sbin/inetd
1250 root      0:00  udhcpc -R -b -p /var/run/udhcpc.eth0.pid -i eth0
1257 root      0:00  /usr/sbin/dropbear -r /etc/dropbear/dropbear_rsa_host_key
1265 root      0:00  /sbin/syslogd -n -0 /var/log/messages
1268 root      0:00  /sbin/klogd -n
1279 root      0:02  /usr/bin/tcf-agent -d -l - -18
1284 root      0:00  {start_getty} /bin/sh /bin/start_getty 115200 ttyPS0
1285 root      0:00  /sbin/getty 38400 ttys1
1386 root      0:00  /tmp/SDSoC_lab_design.elf
 141 root      0:00  ps
root@xilinx-zc702-2017_4:~#
  
```

Figure 407: Linux Terminal – `ps` Command

Configuring the Board Settings

1-1. Configure the board settings to connect to JTAG or SD card mode to run the application.

- 1-1-1. Connect a USB cable from a USB port on your computer to the USB UART connector on the evaluation board.
- 1-1-2. Connect the USB cable to the Digilent USB JTAG interface.
- 1-1-3. **[Linux (System Configuration for Board) users]:** Connect an Ethernet cable from the board to your computer.
- 1-1-4. Set the jumpers as shown below:

[Standalone (System Configuration for Board) users]: Set the boot mode pin to JTAG settings.

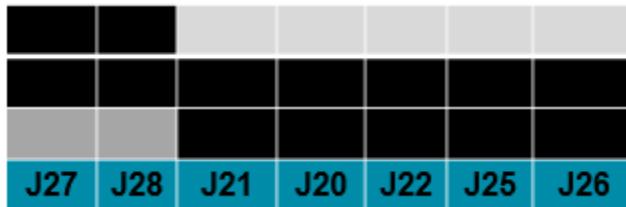


Figure 408: ZC702 Board - JTAG Settings

						ON/3.3V
						Off/Gnd
JP8/JP11 Mode 4	JP7/JP10 Mode 3	JP6/JP9 Mode 2	JP5/JP8 Mode 1	JP4/JP7 Mode 0		

Figure 409: ZedBoard - JTAG Settings

[Linux (System Configuration for Board) users]: Set the boot mode pin to SD card settings.



Figure 410: ZC702 Board - SD Card Settings

						ON/3.3V
						Off/Gnd
JP8/JP11 Mode 4	JP7/JP10 Mode 3	JP6/JP9 Mode 2	JP5/JP8 Mode 1	JP4/JP7 Mode 0		

Figure 411: ZedBoard - SD Card Settings

Note: Black indicates the jumper connection. If you need more information on jumper settings, refer to the *Lab Setup Guide*.

- 1-1-5. **[Linux (System Configuration for Board) users]:** Proceed to the step below that begins with "Copy the files to the SD card."
- 1-1-6. Power on the board and proceed to the step below that begins with "Launch Tera Term or a similar serial port emulator."

1-2. [Linux (System Configuration for Board) users]: Copy the files to the SD card.

- 1-2-1. Insert an SD card into the PC's SD card slot.
- 1-2-2. Browse to the SD card drive using File Explorer and erase or reformat the SD card.
- 1-2-3. [Windows users]: Browse to the image located at *the location of the files you want to copy to the SD card*.

[Linux users]: Browse to the image located at /home/xilinx/training/<the topic cluster name>/lab/your project name/Debug/sd_card.

- 1-2-4. Copy all the files from the source directory to the SD card.

Note: The files should go into the root of the SD card.

- 1-2-5. Remove the SD card from the PC card slot and insert the SD card into its slot on the board.
- 1-2-6. Power on the board.

1-3. [Windows users]: Launch Tera Term or a similar serial port emulator.

- 1-3-1. Make a new connection by selecting **Serial** as the connection type.
- 1-3-2. Identify the **COM** port associated with your board.
- 1-3-3. Set the serial port connection within the serial port terminal emulator to **115200** baud, **8** data bits, **No** parity, **1** stop bit.

1-4. [Linux users]: Verify whether the USB and JTAG cables are detected and enabled.

Note: If the cables are detected but not enabled (checked) then manually enable them by clicking them.

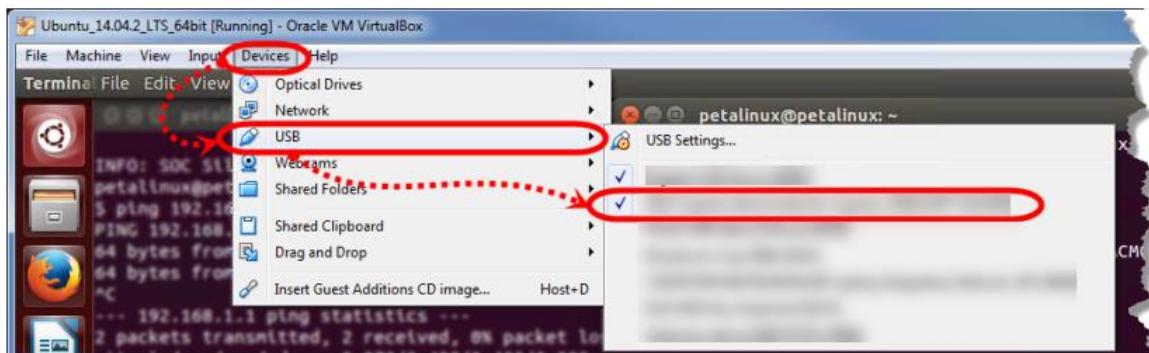


Figure 412: Detecting the USB and JTAG Cables

1-5. Launch gtkterm and set the port configuration.

- 1-5-1. Press <**Ctrl + Alt + T**> launch a Linux terminal window.

- 1-5-2.** Enter the following to launch gtkterm from the terminal window:

```
sudo gtkterm
```

- 1-5-3.** When prompted for the password, **xilinx**.

Note: While the application will run as a regular user, you must be a super user to access the ports.

When the GtkTerm window opens, perform the following.

- 1-5-4.** Select **Configuration > Port** to open the Configuration dialog box.

- 1-5-5.** Identify the port associated with your board and set the port as **/dev/ttyUSBx** (where x could be 0, 1, 2, etc.)

- 1-5-6.** Set the baud rate to **115200**.

Leave the rest of the settings at their default.

Note: You can open multiple instances of /dev/USBx if you are unable to find out which port your UART is connected to.

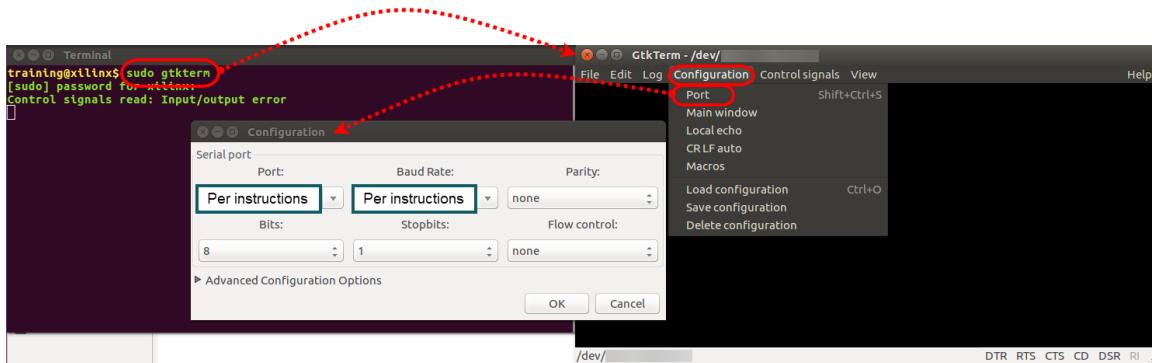


Figure 413: Opening gtkterm and Selecting the Port Configuration

- 1-5-7.** [Optional] You can save these settings so that you do not have to reconfigure GtkTerm each time you open it by clicking **Configuration > Save configuration**.

If you save the configuration as "default" this configuration will open when GtkTerm starts. Otherwise you can save the configuration by another name; however, you will then need to load the configuration each time you start GtkTerm.

- 1-5-8.** Click **OK** to save the settings and leave the terminal open.

1-6. [Linux (System Configuration for Board) users]: Set the IP address of the board.

- 1-6-1. Set the host's Ethernet address to a static IP (suggested: 192.168.1.10).

If you are unfamiliar with this process, refer to the *Lab Setup Guide*.

- 1-6-2. Enter the following at the serial port terminal to change the IP address of the board to the IP address of the board:

```
ifconfig eth0 the IP address of the board
```

Note: This can be any address other than the one that the host is configured to.

- 1-6-3. Verify the IP address by entering the following command:

```
ifconfig eth0
```

- 1-6-4. Ping the host using the Linux terminal console to verify connectivity between the host and the target:

```
ping the host IP address -c 1
```

This verifies the Ethernet connectivity between the host and the development board.

Ping should be successful indicated by 0% packet loss.

1-7. [Linux (System Configuration for Board) users]: Configure the target connection to communicate to the host PC.

- 1-7-1. Return to the SDx tool.

- 1-7-2. Expand **Linux TCF Agent** using the Target Connections view.

Optional: If the Target Connections view is not visible, then select **Window > Show View > Other > Xilinx > Target Connections** and click **OK**.

- 1-7-3. Select and right-click **Linux Agent [default]**.

- 1-7-4. Select **Edit** from the context menu.

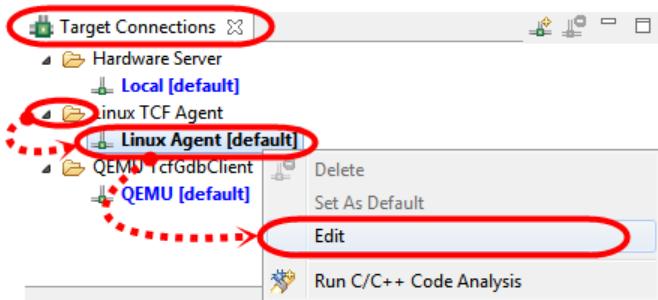


Figure 414: Accessing Linux Agent Target Connection

The Target Connection Details dialog box opens.

1-7-5. Change the host address to **the IP address of the board**.

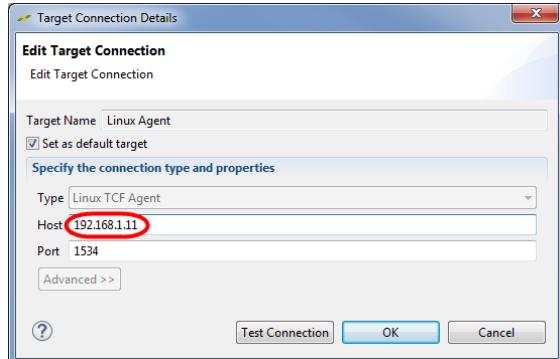


Figure 415: Changing Host Address for Target Connection

1-7-6. Click **OK** to set the connection.

Configuring the Board Settings for Linux

1-1. Configure the board settings to SD Card mode to run the application.

- 1-1-1. Connect a USB cable from a USB port on your computer to the USB UART connector on the evaluation board.
- 1-1-2. Connect the USB cable to the Digilent USB JTAG interface.
- 1-1-3. Connect an Ethernet cable from the board to your computer.
- 1-1-4. Set the jumpers as shown below.
- 1-1-5. Set the boot mode pin to SD card settings.



Figure 416: ZC702 Board - SD Card Settings

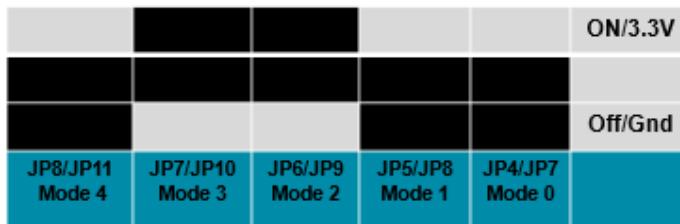


Figure 417: ZedBoard - SD Card Settings

Note: Black indicates the jumper connection. If you need more information on jumper settings, refer to the *Lab Setup Guide*.

1-2. Copy the files to the SD card.

- 1-2-1. Insert an SD card into the PC's SD card slot.
- 1-2-2. Browse to the SD card drive using Windows Explorer and erase or reformat the SD card.
- 1-2-3. Browse to the image located at *the location of the files you want to copy to the SD card*.
- 1-2-4. Copy all the files from the source directory to the SD card.

Note: The files should go into the root of the SD card.

- 1-2-5. Remove the SD card from the PC card slot and insert the SD card into its slot on the board.
- 1-2-6. Power on the board.

1-3. Launch Tera Term or a similar serial port emulator.

- 1-3-1. Make a new connection by selecting **Serial** as the connection type.
- 1-3-2. Identify the **COM** port associated with your board.
- 1-3-3. Set the serial port connection within the serial port terminal emulator to **115200** baud, **8** data bits, **No** parity, **1** stop bit.

1-4. Set the IP address of the board.

- 1-4-1. Set the host's Ethernet address to a static IP (suggested: 192.168.1.10).

If you are unfamiliar with this process, refer to the *Lab Setup Guide*.

- 1-4-2. Enter the following at the serial port terminal to change the IP address of the board to the IP address of the board:

```
ifconfig eth0 the IP address of the board
```

Note: This can be any address other than the one that the host is configured to.

- 1-4-3. Verify the IP address by entering the following command:

```
ifconfig eth0
```

- 1-4-4. Ping the host using the Linux terminal console to verify connectivity between the host and the target:

```
ping the host IP address -c 1
```

This verifies the Ethernet connectivity between the host and the development board.

Ping should be successful, indicated by 0% packet loss.

1-5. Configure the target connection to communicate to the host PC.

- 1-5-1. Return to the SDx tool.
- 1-5-2. Expand **Linux TCF Agent** using the Target Connections view.

Optional: If the Target Connections view is not visible, then select **Window > Show View > Other > Xilinx > Target Connections** and click **OK**.

- 1-5-3. Select and right-click **Linux Agent [default]**.
- 1-5-4. Select **Edit** from the context menu.

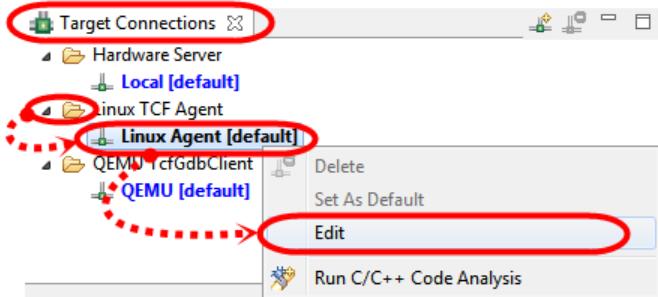


Figure 418: Accessing Linux Agent Target Connection

The Target Connection Details dialog box opens.

- 1-5-5. Change the host address to **the IP address of the board**.

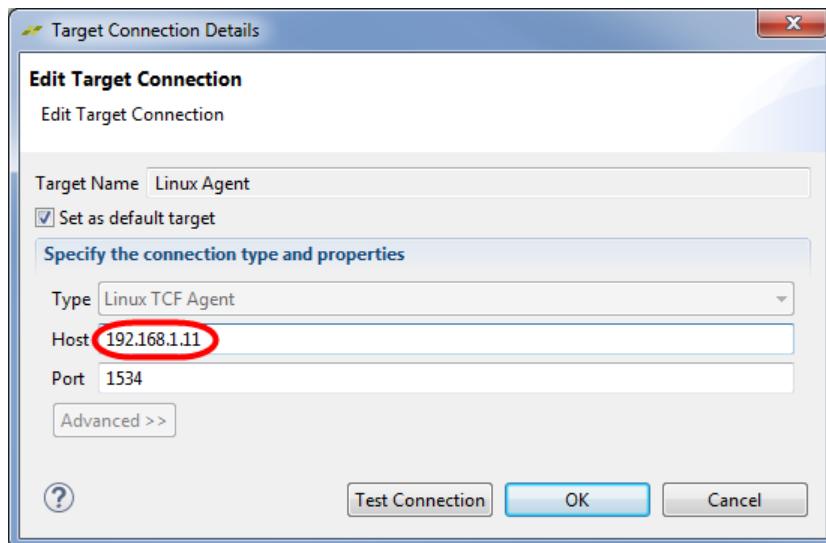


Figure 419: Changing Host Address for Target Connection

- 1-5-6. Click **OK** to set the connection.

Setting the Build Configuration to SDDebug

A Debug configuration defines how you want the system to work when performing a debug operation. Typically a debug operation switches to the Debug perspective. While there are a significant number of switches and options, the most common are shown below.

1-1. Set the build configuration to the needed build configuration (SDEstimate, SDDebug, or SDRelease).

- 1-1-1. Right-click the project name in the Project Explorer pane to open the context menu.
- 1-1-2. Select **Build Configurations > Set Active > the needed build configuration (SDEstimate, SDDebug, or SDRelease)** to build the desired configurations.

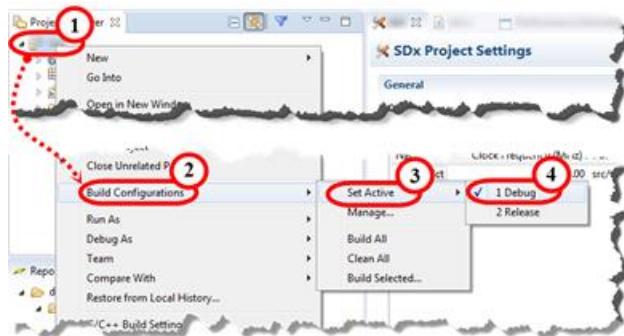


Figure 420: Selecting the Debug Build Configuration (Example)

- 1-1-3. Right-click the project name and select **Build Project**.

This will take a minute or two to complete building the project.

Launching a Debug Session

A Debug configuration defines how you want the system to work when performing a debug operation. Typically a debug operation switches to the Debug perspective. While there are a significant number of switches and options, the most common are shown below.

1-1. Set the build configuration to the needed build configuration (SDEstimate, SDDebug, or SDRelease).

- 1-1-1. Right-click the project name in the Project Explorer pane to open the context menu.
- 1-1-2. Select **Build Configurations > Set Active > the needed build configuration (SDEstimate, SDDebug, or SDRelease)** to build the desired configurations.

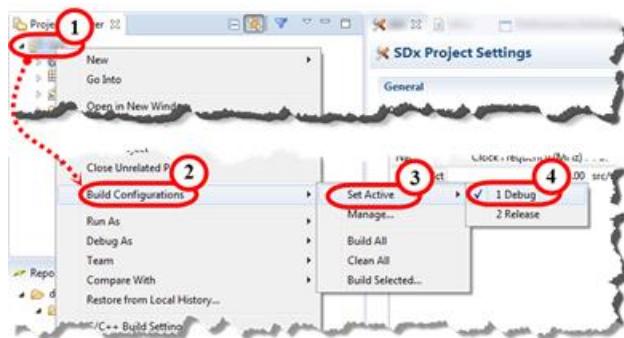


Figure 421: Selecting the Debug Build Configuration (Example)

1-1-3. Right-click the project name and select **Build Project**.

This will take a minute or two to complete building the project.

1-1-4. Right-click the project name and select **Debug As > Launch on Hardware (SDSoC Debugger)**.

1-1-5. Click **Yes** if you are asked to perform a perspective switch.

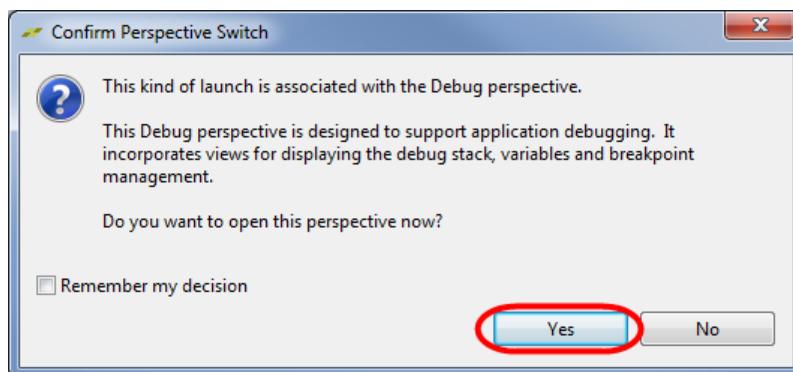


Figure 422: Confirm Perspective Switch to Debug

The SDSoc tool will now program the PS/PL and switch the perspective to the Debug view.

Program operation is suspended at the first executable statement in *main{}* (not running).

Note that local variables for the current function are shown in the Variables tab.

Setting Up a System Debugger Debug Configuration (Standalone)

A Debug configuration defines how you want the system to work when performing a debug operation and maps an ELF file to a target for execution. Typically a debug operation switches to the Debug perspective. While there are a significant number of switches and options, the most common are shown below.

If you are using the SDSoc tool, skip to the next instruction below that begins with "This setup is specific for the SDSoc tool."

This setup is specific for the SDK tool.

1-1. Set up a debug configuration for a specific application project.

- 1-1-1. From the Project Explorer pane, right-click the application project that you want to build the Debug configuration for (1).
- 1-1-2. Select **Debug As** to open the menu of predefined configurations and the configuration manager (2).
- 1-1-3. Select **Debug Configurations** to view all the available debug options (3).

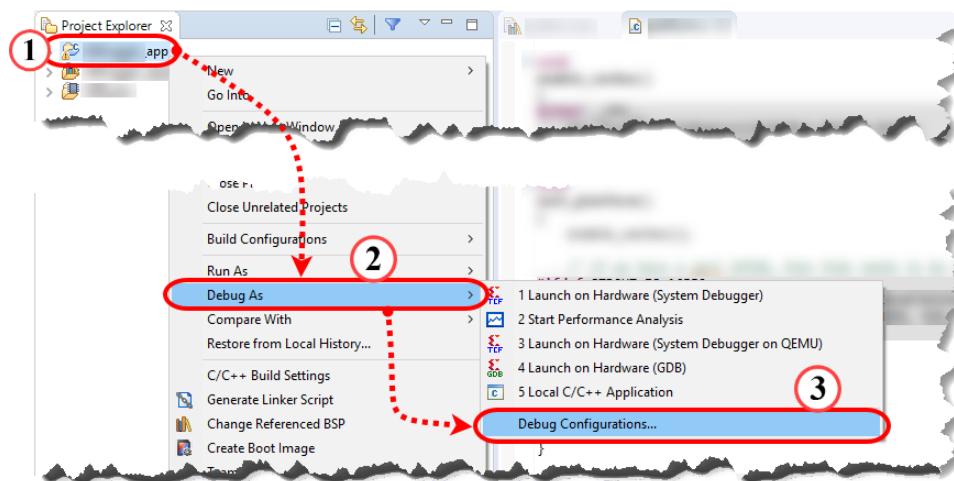


Figure 423: Creating a Debug Configuration

The Debug Configurations dialog box opens.

- 1-1-4.** Select **Xilinx C/C++ application (System Debugger)** since you will be debugging using this debugger (1).

GDB still works; however, it is considered deprecated for new designs.

- 1-1-5.** Click the **Create New Configuration** icon to create the new configuration for your application (2).

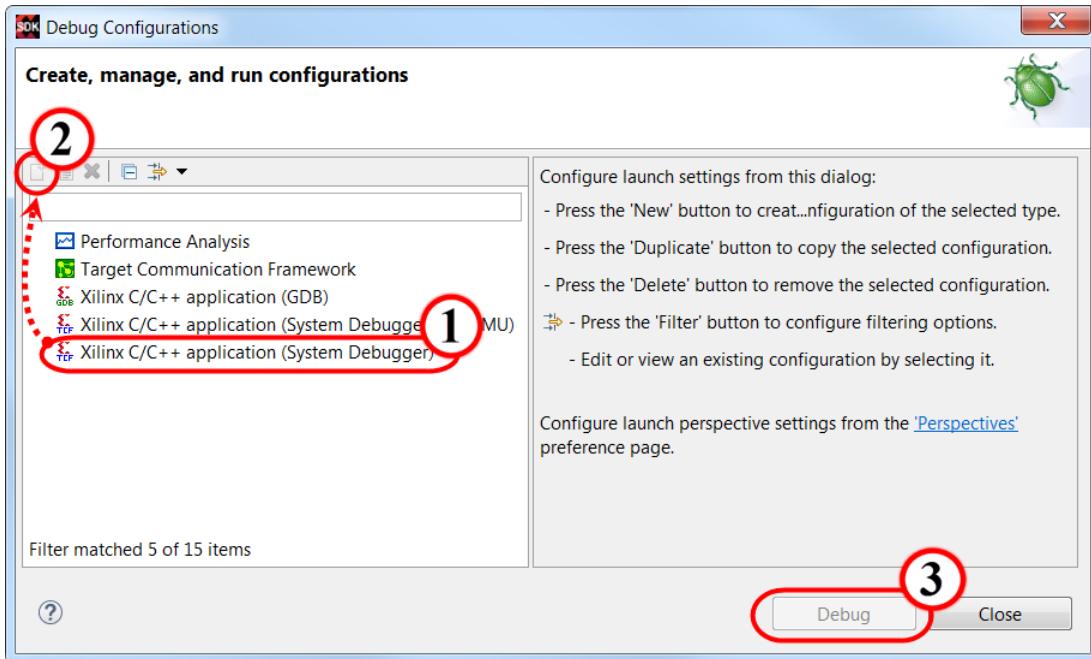


Figure 424: Creating a New Debug Configuration

A new Debug Configuration is created. The figure above depicts an SDK workspace that does not yet have any debug or run configurations defined. If other configurations do exist, or after creating the first configuration, the exiting configurations will appear below the configuration type. Note the default configuration name and other parameters that are automatically filled in. In most cases, you just need to click the Debug button to begin the session. This Debug Configuration menu is useful when you want to use different debug parameters.

The new configuration will appear under the type of configuration you selected, in this case "Xilinx C/C++ application (System Debugger)" and have the name of your project by default (you can change this as one of the many parameters in the debug). You will also note that a number of fields are automatically filled in for you using the name of your project somewhere in the debug configuration name.

- 1-1-6.** Click **Debug** to close the window and launch the debugging session.

If the Confirm Perspective Switch dialog box appears, click **Yes**.

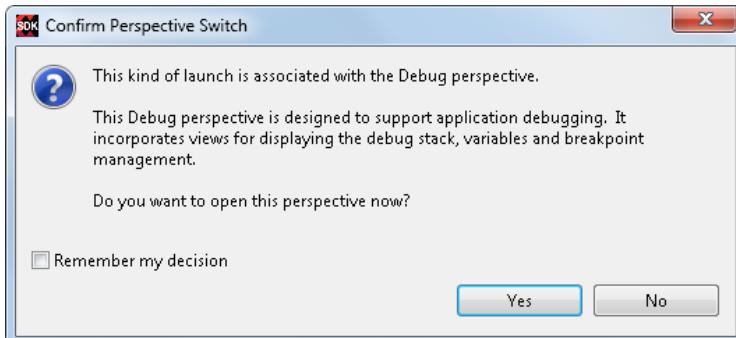


Figure 425: Confirming Switch of Perspective

The Debug Perspective view opens.

This setup is specific for the SDSoc tool. Skip this instruction if you are using SDK.

1-2. Set up a debug configuration for an SDSoc tool project.

- 1-2-1. Using the Project Explorer pane, ensure that the project (**your project name**) you want to build the Debug configuration for is expanded.
- 1-2-2. Right-click **project.sdsoc** to open the context menu (1).
- 1-2-3. Select **Open** to bring up the project overview page in the main workspace (2).

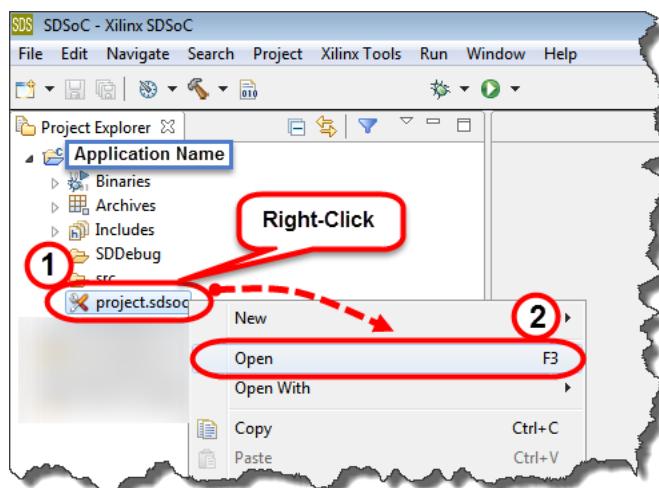


Figure 426: Opening the Project Overview

1-2-4. Click the **Debug application** link (1).

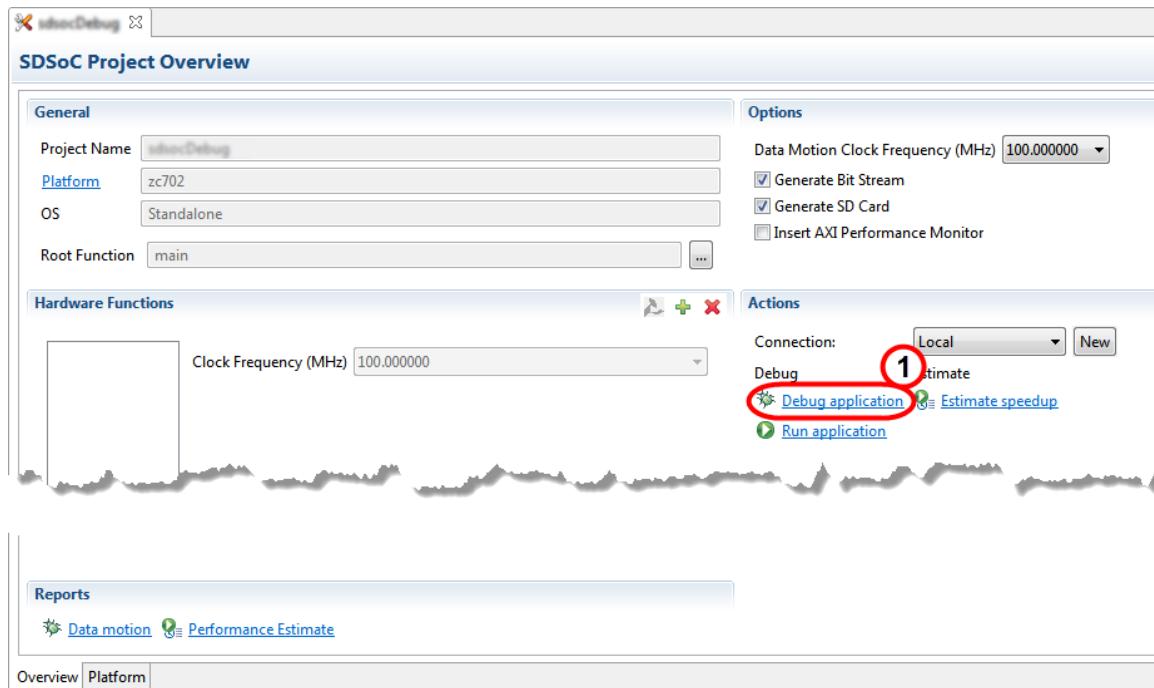


Figure 427: Clicking Debug Application

If the project has not yet been built using SDDebug, the SDSoC development environment will compile the project with the SDDebug build configuration.

The SDSoC tool will now create a Debug configuration.

1-2-5. If a dialog box asking to perform a perspective switch appears, click **Yes**.

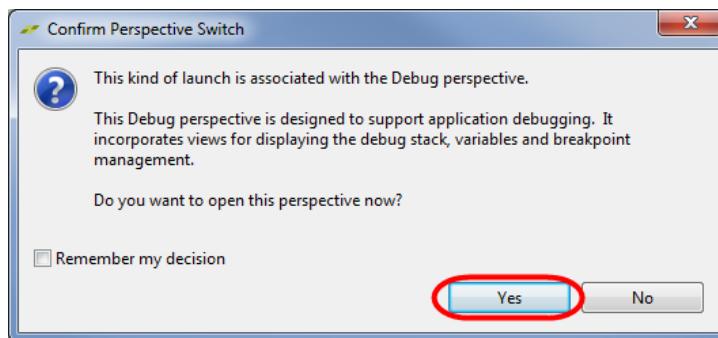


Figure 428: Confirm Perspective Switch to Debug

The SDSoC tool will now program the PS/PL and switch the perspective to the Debug view.

Program operation is suspended at the first executable statement in *main{}* (not running).

Note that local variables for the current function are shown in the Variables tab.

Setting Up a System Debugger Debug Configuration (Linux)

Run and Debug configurations are application project objects that contain communication, hardware, and execution options for running an application on a hardware or emulation platform. The selections for Run and Debug are identical and only differ in that Run just executes the application and Debug opens a debug perspective and launches a debug program.

There are different type of Run and Debug configurations based on the operating system (or Standalone libraries) and the SDK or SDSoc download/debug tools that you want to use. A project can have multiple configurations to save various setups of this information.

Note: The following is specific to the SDK development environment. If you are using the SDSoc tool, skip to the instruction that begins with "**Note:** The following is specific to the SDSoc development environment configuration."

- 1-1. Create a new Linux Debug configuration. Debug and Run configurations associate an ELF object file to a target (typically a hardware board) for execution. In this case, the target is a hardware board accessed over the Ethernet TCP/IP connection that was set up when the RSE tool was engaged.**
- 1-1-1.** Click the **C/C++** tab in the upper-right corner of the GUI to return to the C/C++ perspective.

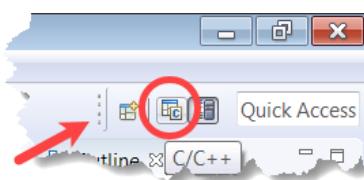


Figure 429: Changing Perspective

This brings back access to the software projects. It is accessed by first clicking the >> in the same location. If this tab is not available, you can also return to the perspective by selecting **Window > Open Perspective > Other > C/C++ (default)**.

1-1-2. Right-click **your application** in the Project Explorer window (1).

1-1-3. Select **Debug As** (2) > **Debug Configurations** (3).

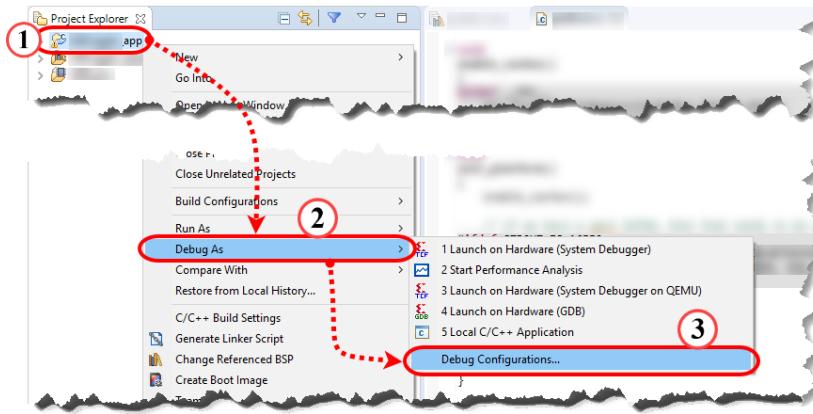


Figure 430: Selecting Debug Configurations

The Debug Configurations dialog box opens.

1-1-4. Double-click **Xilinx C/C++ application (System Debugger)** to create a launch configuration (1).

Alternatively, you can select **Xilinx C/C++ application (System Debugger)** and click the **New** configuration button.

If this is the first debug configuration being created for the project, a welcome type dialog opens. If one or more configurations exist, then the last open configuration will be displayed. In either case, a new configuration can always be added. Existing configurations are shown in the left pane and can be selected for debugging.

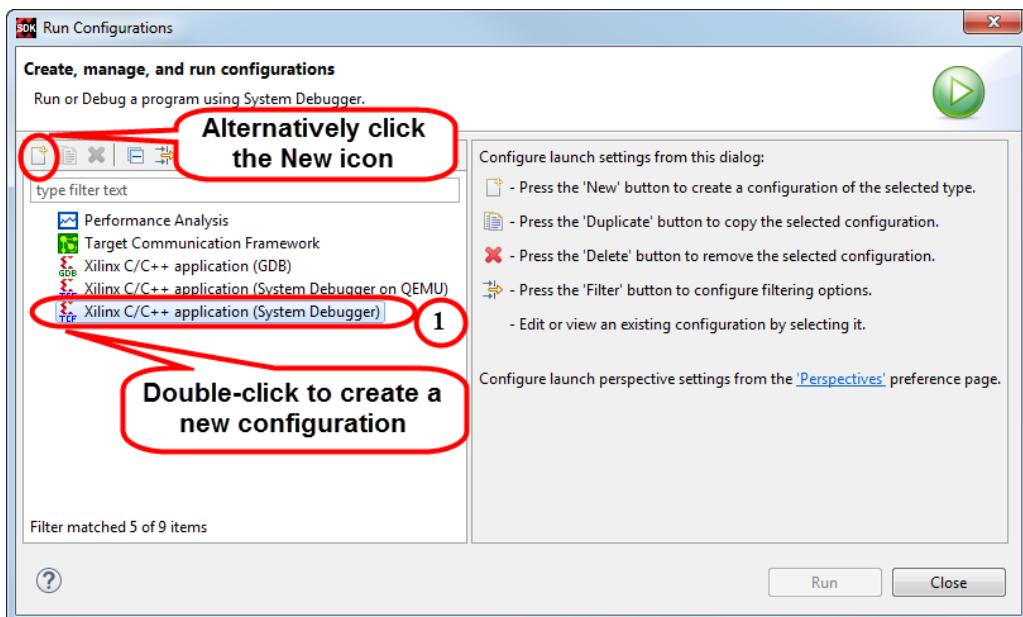


Figure 431: Creating a New System Run/Debug Configuration

1-2. Configure the debug type and host connection.

- 1-2-1.** Select **Linux Application Debug** from the Debug Type drop-down list (2).

This will engage the proper debug tool to use.

- 1-2-2.** Click **New** next to the Connection drop-down list to launch the Target Connection Details dialog box (3).

A new connection will be defined from the debugger to the hardware (or emulator) target.

- 1-2-3.** Enter **ZynqBoard** in the Target Name field as the name of the connection (4).

Note that the type of connection defaults to Hardware Server, which will be the RSE connection that was set up earlier.

- 1-2-4.** Enter **192.168.1.10** in the Host field (5).

This is the IP address of the RSE connection that was set up earlier.

- 1-2-5.** Enter **1534** in the Port field (5).

This is the default TCP/IP port number for the connection.

- 1-2-6.** Click **OK** (6).

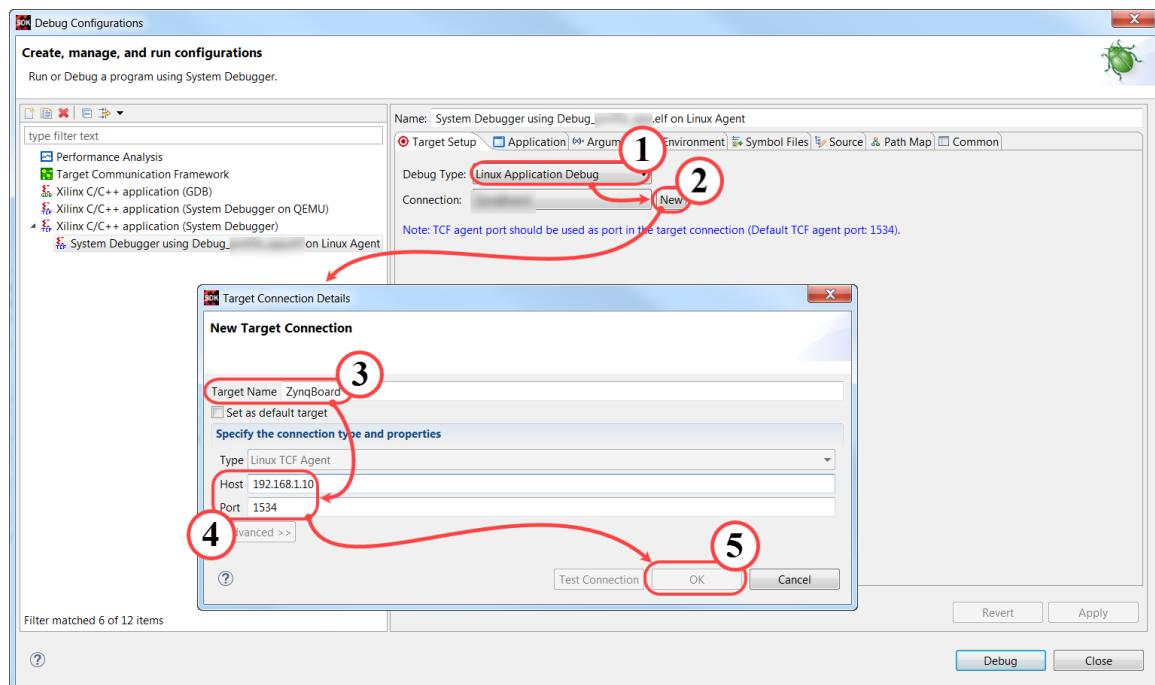


Figure 432: Selecting the Debug Type and Connection

1-3. Select the software application ELF file to debug and its file path on the remote target board where it is to be copied.

The actual software application debugging takes place on the Linux platform running on the target hardware, so it is necessary to put a copy of the ELF file on it.

1-3-1. Select the Application tab (1).

This is where the software application is selected and where to put it on the remote platform.

1-3-2. Click Browse next to the Local File Path field to select the software application ELF file (2).

1-3-3. Select the Local File Path to be C:\training\<the topic cluster name>\lab\SDDebug\your application.elf (3).

1-3-4. Enter /tmp/your application.elf in the Remote File Path field as the location on the target platform where the application ELF file will be copied (4).

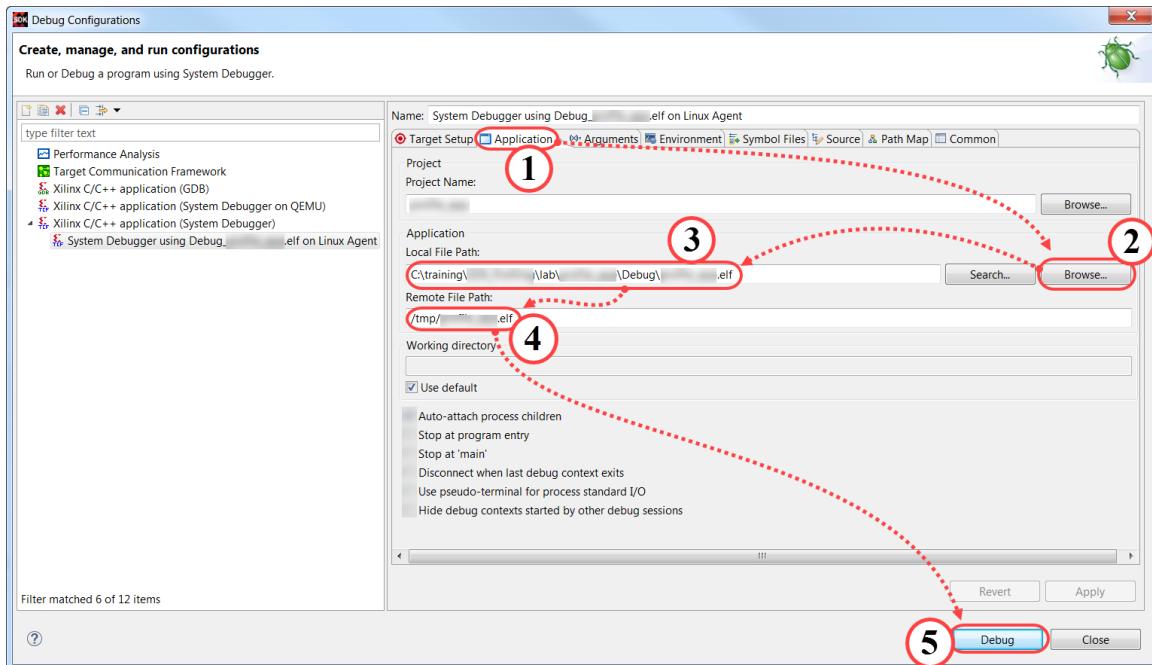


Figure 433: Selecting the Local File Path and Remote File Path

The remaining options will be accepted at their default values.

1-3-5. Click Debug (5).

- 1-3-6.** Click **Yes** to confirm opening the Debug perspective view.

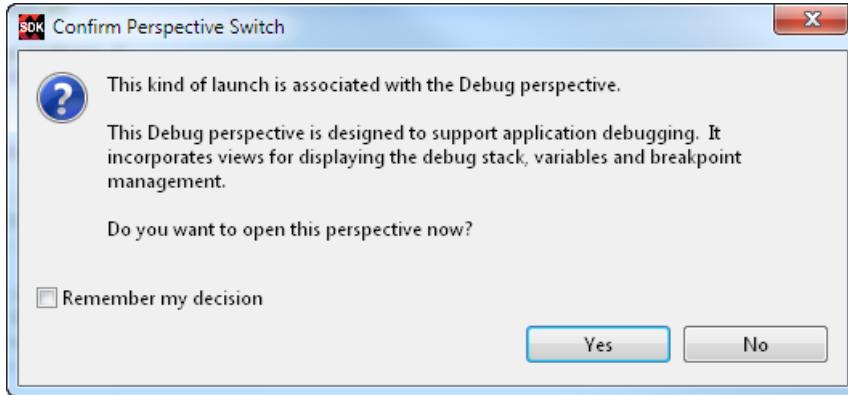


Figure 434: Confirming Perspective Switch

The Debug perspective opens.

The SDK debug configuration ends here.

Note: The following is specific to the SDSoc development environment configuration. Skip to the next step if using SDK.

1-4. Create a new Linux TCF connection to the target board.

- 1-4-1.** Using the Project Explorer pane, ensure that the project (**your project name**) you want to build the Debug configuration for is expanded.
- 1-4-2.** Right-click **project.sdsoc** to open the context menu (1).
- 1-4-3.** Select **Open** to bring up the project overview page in the main workspace (2).

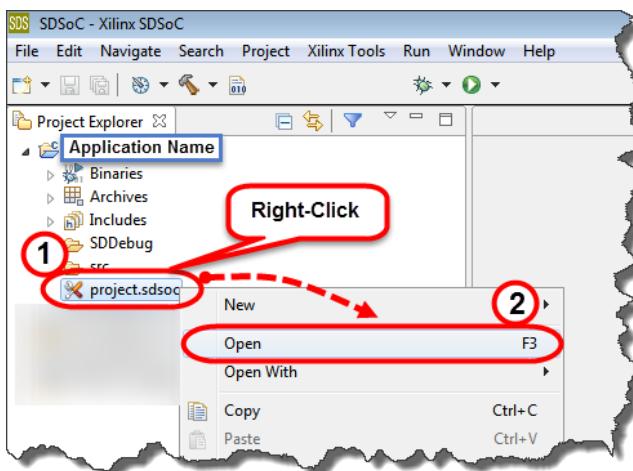


Figure 435: Opening the Project Overview

- 1-4-4.** Click **New** from the Actions area in the Project Overview window.

Note: This will open a dialog box asking for connection details.

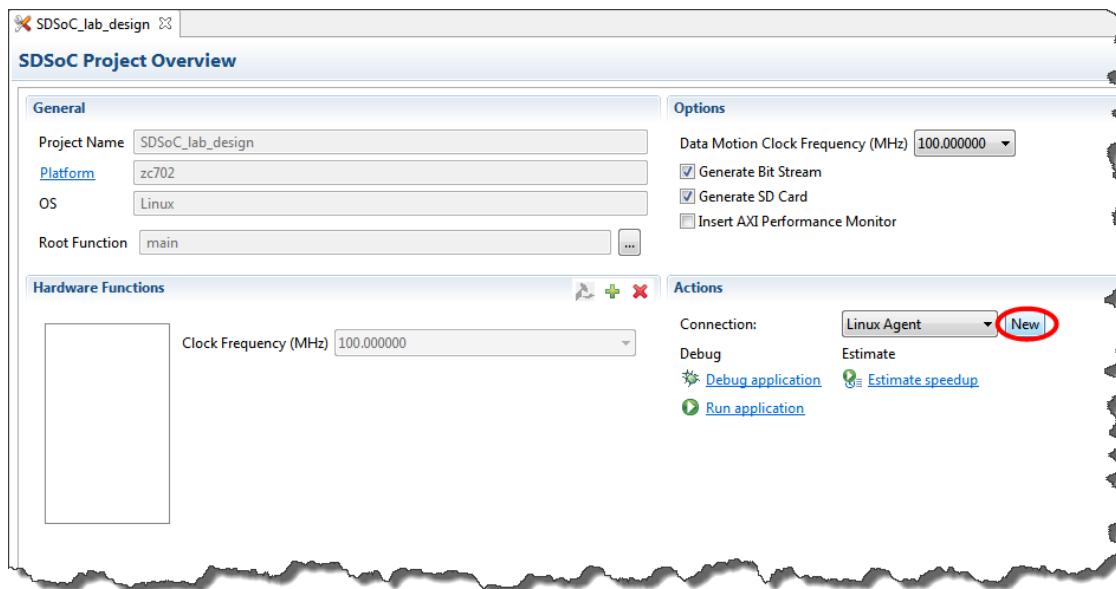


Figure 436: Creating a New Hardware Connection

- 1-4-5. Enter **remote_connection** in the Target Name field to set the target/connection name.
- 1-4-6. Enter **192.168.1.10** in the Host field to set the IP address of the development board that was configured earlier.

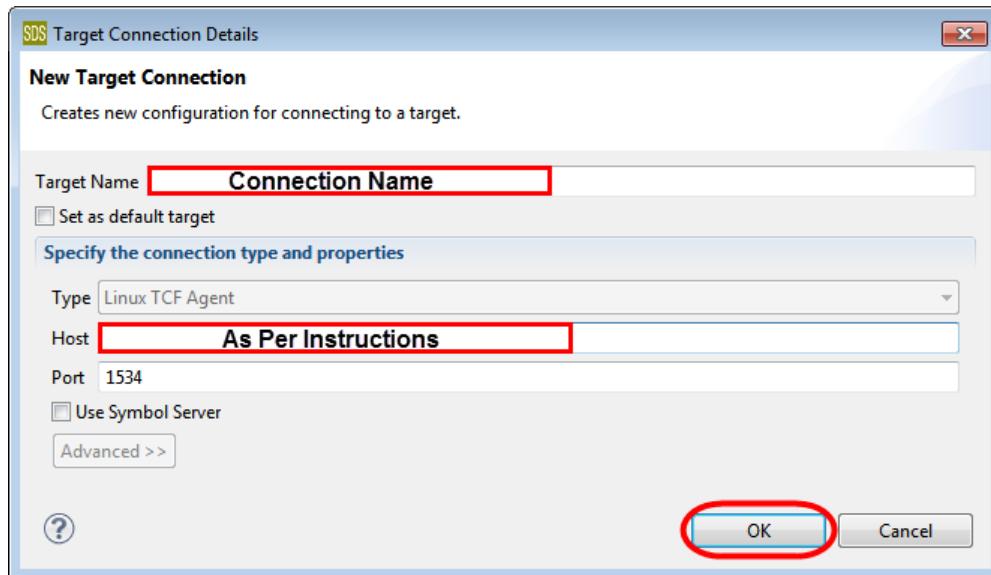


Figure 437: Target Connection Details

- 1-4-7. Select **OK** to close the dialog box and return to the Project Overview window.

1-5. Start the debugging process.

- 1-5-1. From the Actions area in the Project Overview window, click the **Connection** drop-down list down arrow to access all of the available connections (1).
- 1-5-2. Select **remote_connection** to choose the just created connection (2).
- 1-5-3. Click the **Debug application** link to create a debug configuration and open the debug perspective (3).

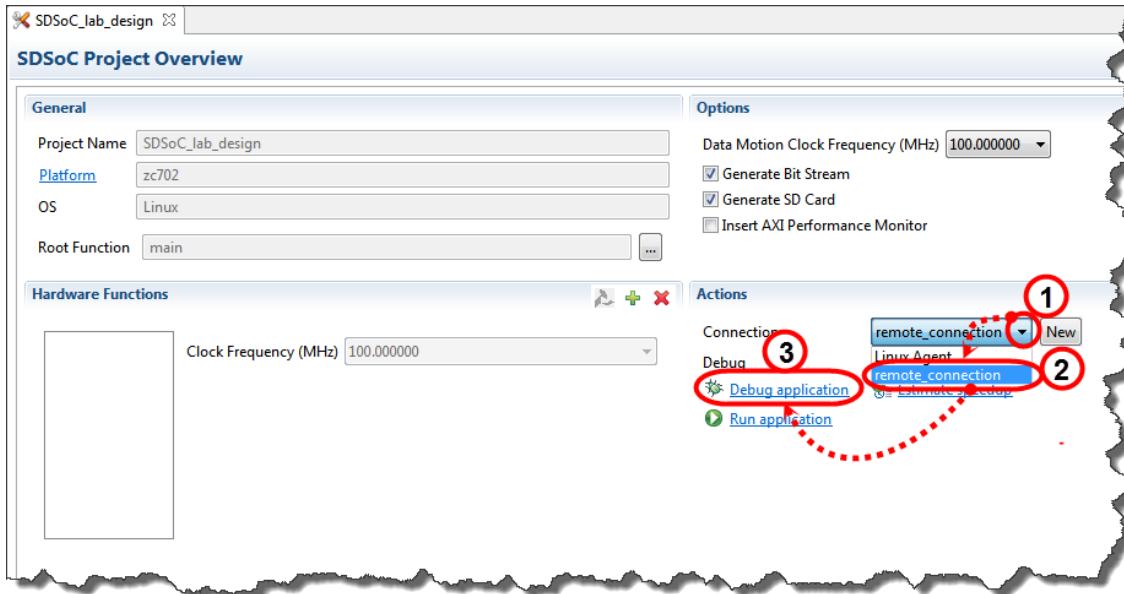


Figure 438: Selecting Connection and Debug

If the project has not yet been built using SDDebug, the SDSoc development environment will compile the project using the SDDebug build configuration.

The SDSoc tool will now create a Debug configuration.

- 1-5-4. If a dialog box asking to perform a perspective switch appears, click **Yes**.

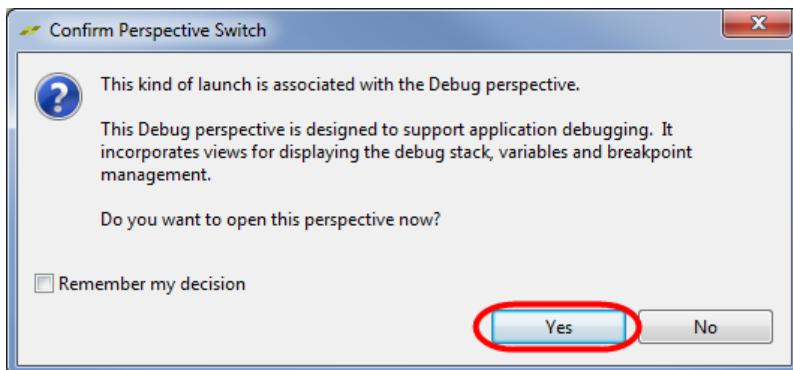


Figure 439: Confirm Perspective Switch to Debug

The SDSoc toll will now program the PS/PL and switch the perspective to the Debug view.

1-6. Optional for Linux (System configuration for Board): Highlight the process for debug.

Sometimes the process for debugging is not highlighted in the Debug view and the flow controls such as pause, resume, single step, etc. will be grayed out. Follow the procedure below to enable the controls.

- 1-6-1. Expand the Debug view, if it is not already expanded, to expand the top-level debug session (1).
- 1-6-2. Expand **/tmp/your project name.elf**, if it is not already expanded, to expand the next level showing the binary that is being executed (2).

Note: The location of the process on the remote device (**/tmp/your project name.elf**) is also visible.

- 1-6-3. Click the process that is currently suspended (3).

Note: This will tell the Debug tools which remote application/process to debug and enable the flow control tools.

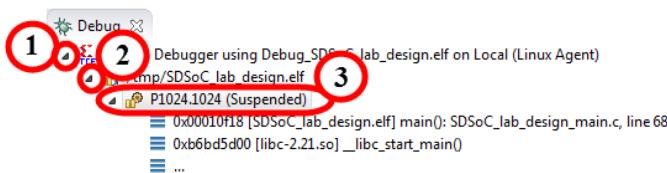


Figure 440: Highlighting the Debug Session Application

Tip: In the figure above and below, P1024 is the process ID (PID) of the application running under Linux; your PID may be different. Issuing the `ps` command through the terminal will show all of the system processes that are currently running. There will be a listing with the same PID that matches the number of your suspended application.

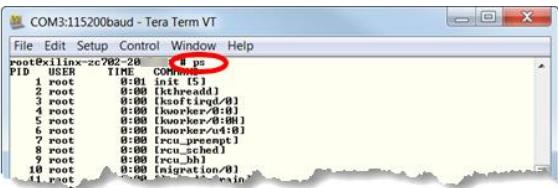


Figure 441: Linux Terminal – ps Command

Setting the Build Configuration to SDDebug

A Debug/Release configuration defines how you want the system to work. Typically, a debug operation switches to the Debug perspective. While there are a significant number of switches and options, the most common are shown below.

1-1. Set the build configuration to the needed build configuration (SDEstimate, SDDebug, or SDRelease).

- 1-1-1. Right-click the project name in the Project Explorer pane to open the context menu.
- 1-1-2. Select **Build Configurations > Set Active > the needed build configuration (SDEstimate, SDDebug, or SDRelease)** to build the desired configurations.

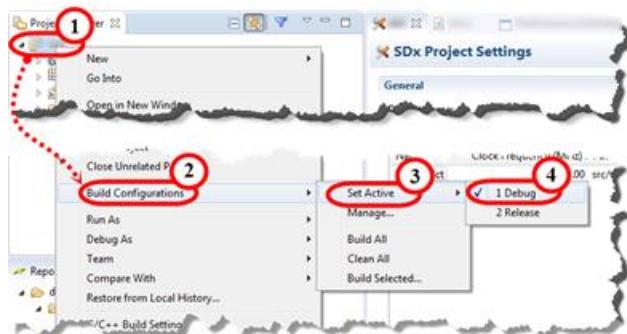


Figure 442: Selecting the Debug Build Configuration (Example)

1-1-3. Right-click the project name and select **Build Project**.

It will take about 5-10 minutes to finish building the project.

Creating a C/C++ SDSoc Tool Project

Using the SDSoc Project Wizard is a quick way to set up a C or C++ software application project that targets an existing processor and OS platform (Standalone or Linux). You can automatically generate the board support package (BSP) or select an existing one. Based on the dialog box choices, the appropriate tool chain is selected for pre-processing, compiling, assembling, and linking.

1-1. Create a new C/C++ application project named *your application project name*. Use the board support package named *your BSP name*.

- 1-1-1. Select **File** (1) > **New** (2) > **Application Project** (3) to open the New Project dialog box.

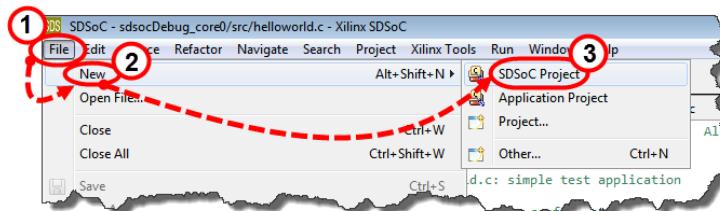


Figure 443: Creating an SDSoc Tool Project Using the Wizard

- 1-1-2. Enter **your application project name** as the project name.
- 1-1-3. Ensure that **Standalone** is selected from the OS Platform drop-down list.
- 1-1-4. Ensure that you have **your hardware platform description name** selected from the Hardware Platform drop-down list as the SDK or SDSoc tool can manage multiple platforms within a single workspace.

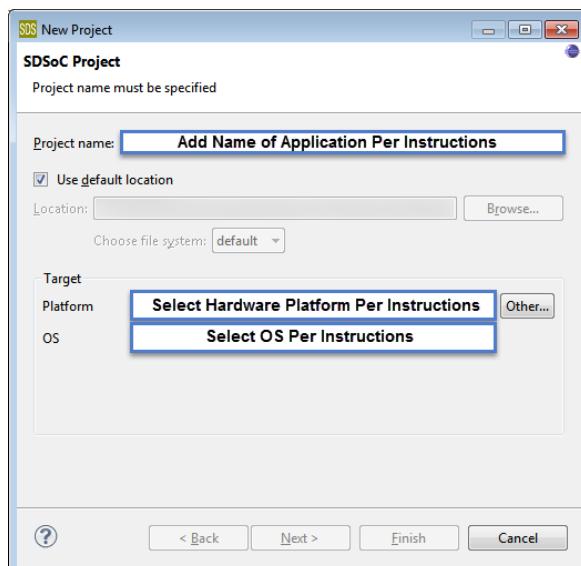


Figure 444: New SDSoc Project Dialog Box

1-1-5. Click **Next** to select the template for this application.

1-1-6. Select **an application template**.

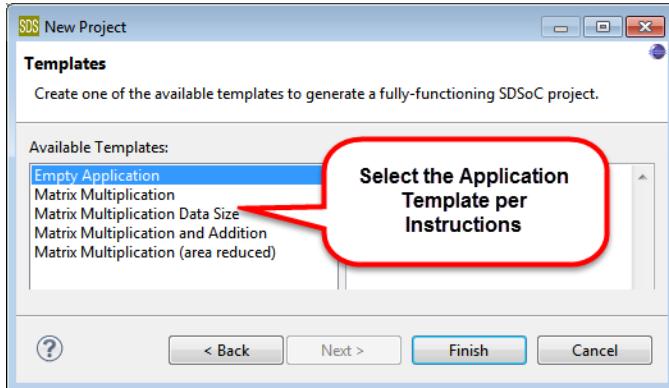


Figure 445: Selecting Application Template

1-1-7. Click **Finish** to create the new project.

Configuring the Terminal

1-1. Open the Terminal tab.

1-1-1. Select the **SDx Terminal** tab to access the terminal icons (1).

If the Terminal tab is not visible, select **Window > Show View > Other > Terminal**.

Note that more than one terminal can be enabled at a time.

1-1-2. Click the **Settings** icon (2).

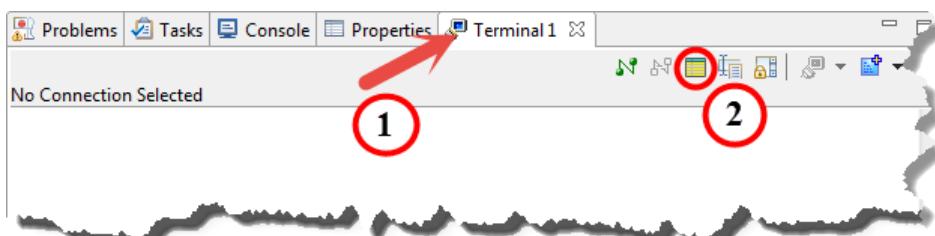


Figure 446: Accessing the Terminal Settings

Alternatively, you can also click the **Connect** icon () to open the Terminal Settings dialog box. If the terminal was previously configured, this action will not open the Terminal Settings dialog box; rather it will attempt to open the communication channel with the previously selected settings.

Changing the settings requires that you close the channel and use the Settings icon.

1-2. Configure the settings as shown in the following figure.

1-2-1. Select the connection type as **Serial**.

1-2-2. Select the proper port for the COM #.

Your board must be on and connected; otherwise the USB bridge is not enumerated and your COM port will not appear.

1-2-3. Set the baud rate to **115200**.

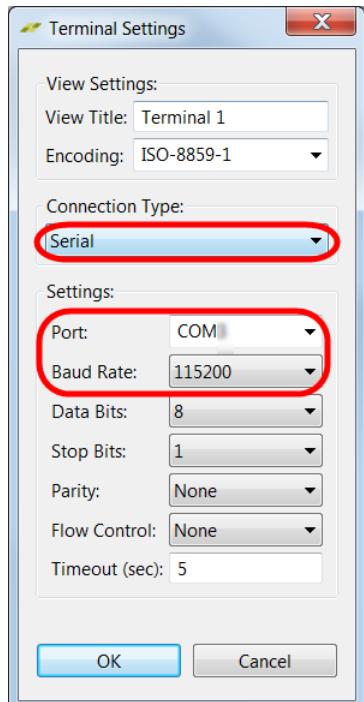


Figure 447: Configuring the Terminal Settings

1-2-4. Click **OK**.

The terminal session will be connected to the associated COM port on the PC.

Configuring the PC's Ethernet Port for Remote System Explorer

The PC's Ethernet port must be properly configured to handle traffic with the Linux platform.

1-1. Configure your PC to support the Remote System Explorer.

- 1-1-1. Disconnect from the VPN if you are connected.
- 1-1-2. Access your PC's Control Panel.
- 1-1-3. Select **Network and Sharing**.
- 1-1-4. Identify your cabled ethernet connection:
 - NOT the Bluetooth connection
 - NOT the VPN connection
 - NOT the wireless connection
- 1-1-5. Right-click the cabled Ethernet connection.
- 1-1-6. Ensure that the **Internet Protocol Version 6** option is unchecked.
- 1-1-7. Select **Internet Protocol Version 4**.
- 1-1-8. Select **Properties**.
- 1-1-9. Assign an address other than 192.168.1.10 (which is the address of the device on Xilinx evaluation boards).
Typically, 192.168.1.11 is selected.
- 1-1-10. Save and close all open windows

Generating the Software Estimate from the SDSoc Tool

1-1. Generate the software estimation.

- 1-1-1.** Click the **Click Here** link in the Performance Estimation Report to run only the "software-only application performance and speedup" test.

If the Performance Estimation Report is not open, you can open it by clicking **Performance Estimate Report** under the **SDSoC_lab_design > Debug** in the **Reports** panel.

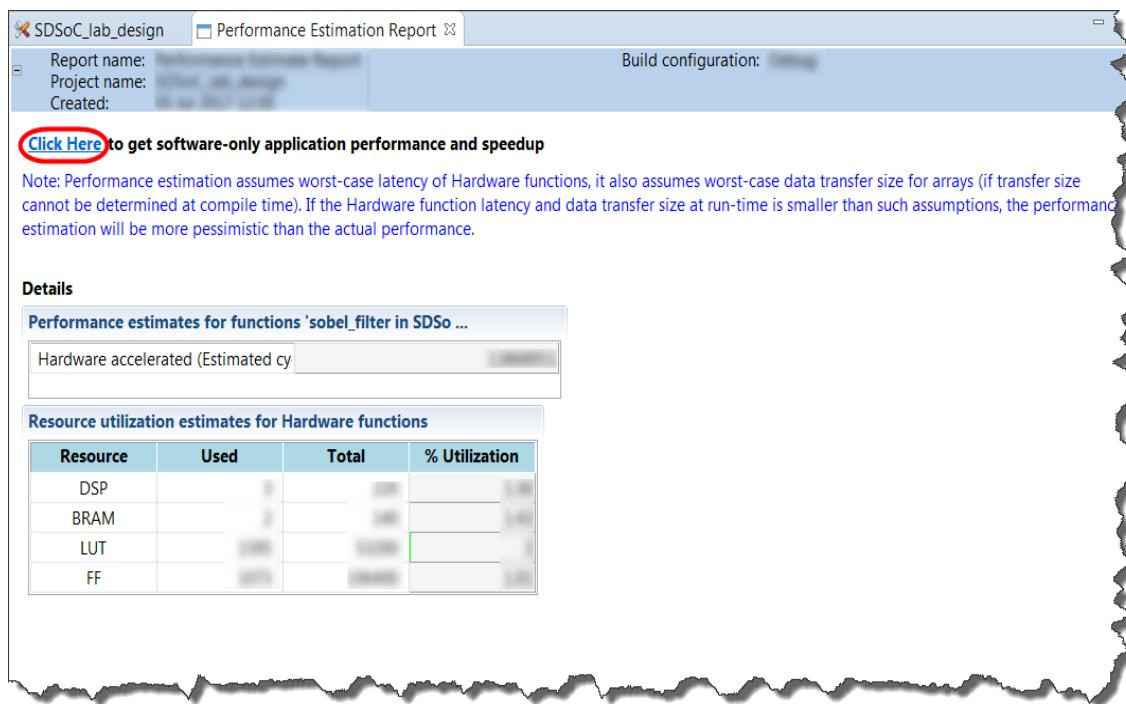


Figure 448: Gathering Software-only Performance Information

The Run Application To Get Its Performance dialog box opens.

- 1-1-2.** Establish a software connection between the host machine and the target board.

Standalone (System Configuration for Board) users: Use the default settings when your board is connected to your PC (local).

Linux (System Configuration for Board) users: Make sure that the Connection is set to Linux Agent or the New Connection set by you.

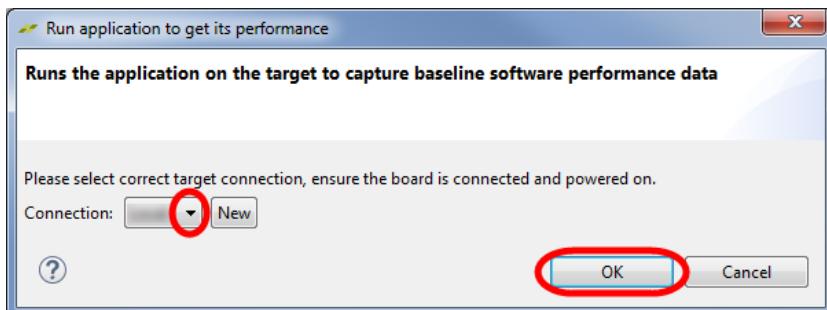


Figure 449: Run Application To Get Its Performance Dialog Box

- 1-1-3. Click **OK** to run the application and load the results into the tab.

The PS portion of the SoC will now be programmed and the estimation numbers for software obtained.

Note: It may take a minute or two for the performance estimation and should see the Console tab with the Performance Estimation results as shown below.

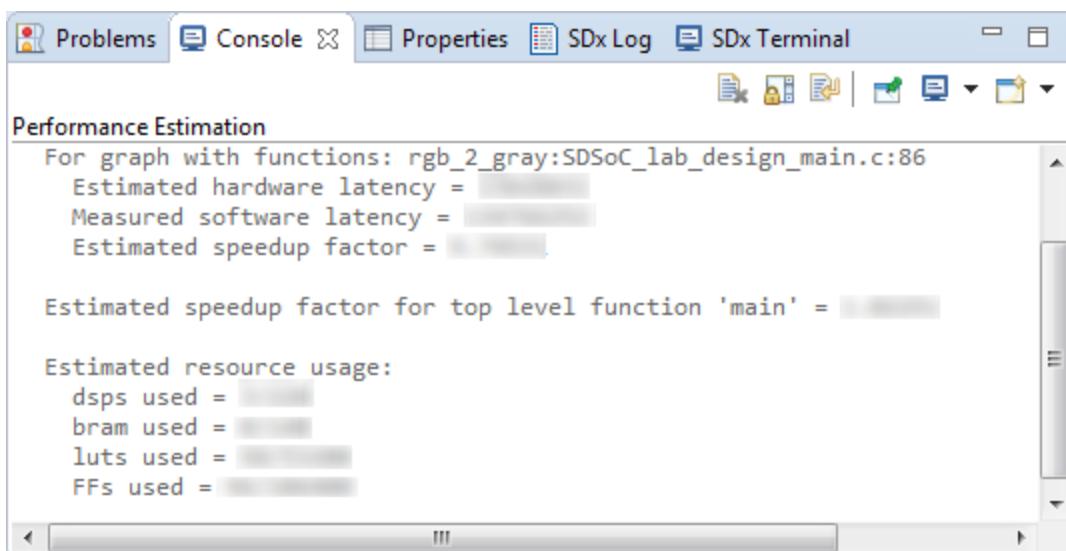


Figure 450: Performance Estimation in the Console Tab

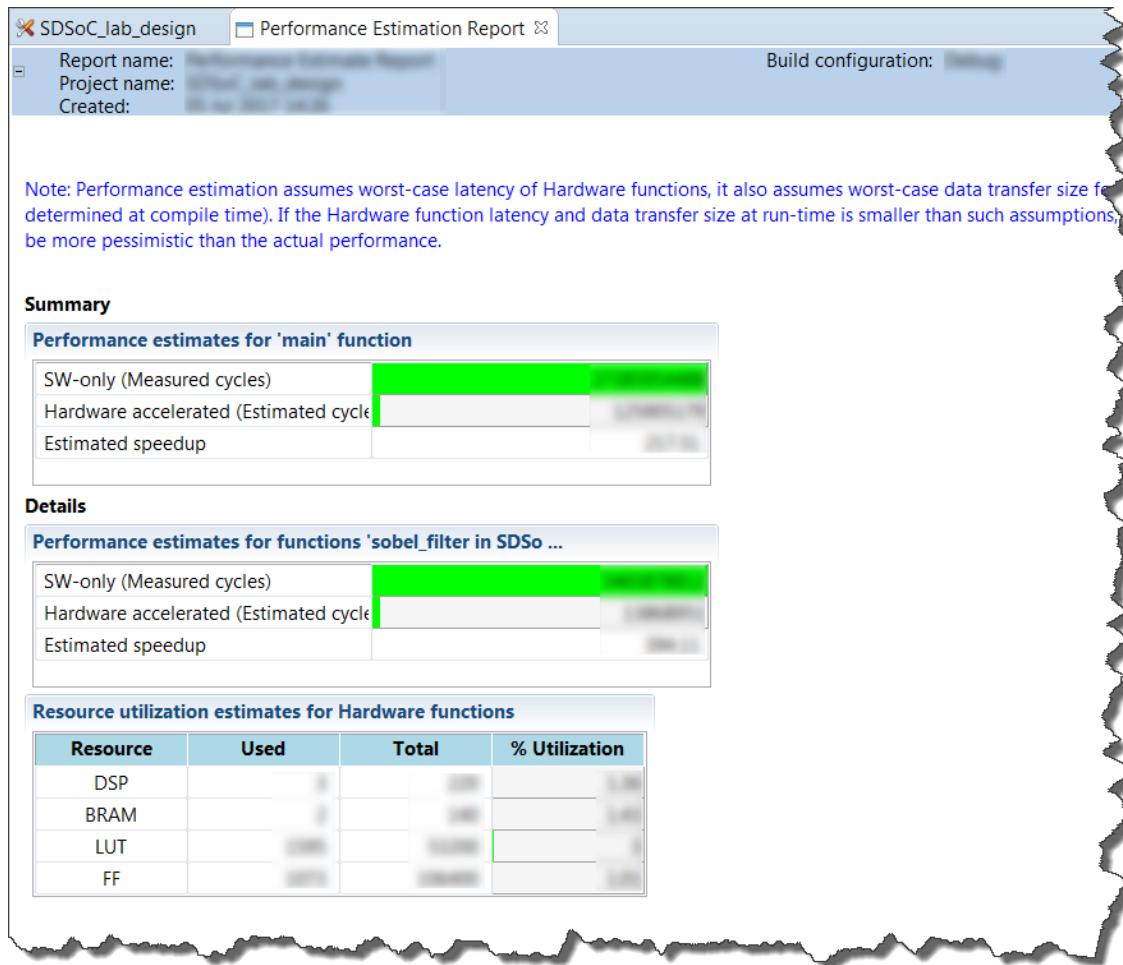


Figure 451: Full Estimate Report for Optimized Design

If the Performance Estimation Report tab closes, it can be reopened by double-clicking **Debug > _sds > est > perf.est**.

The Performance Estimation Report tab will now be displayed with details on the accelerated functions and system speed up.

Opening a Previously Run Performance Estimate

1-1. Open a previously run performance estimate.

- 1-1-1. Expand the desired project > **Debug** > **_sds** > **est** under the project listed in the Project Explorer.

Note: If the **est** folder is not visible, you will need to rebuild the Debug configuration by enabling the **Estimate Performance** option from the SDx Project Settings tab. The quickest way to perform this action is to click the pull-down menu next to the Build icon () and select **Debug**. Alternatively, you can select **Project > Build All**.

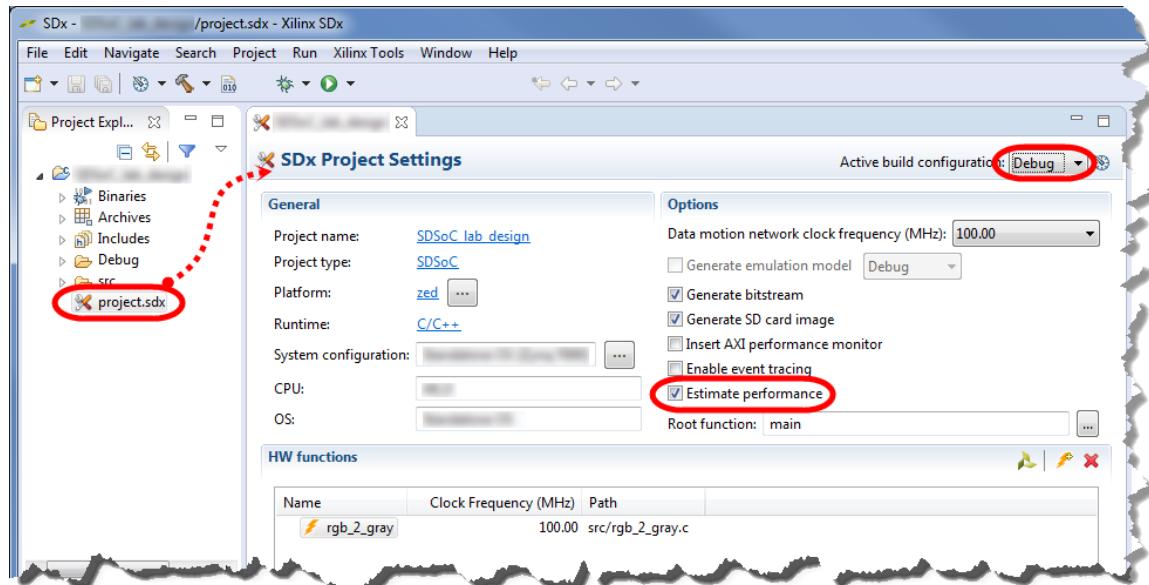


Figure 452: Enabling the Performance Estimate and Debug Configuration Build

- 1-1-2. Double-click the **perf.est** entry to open the report in a new tab.

Since the software-only application performance numbers are not preserved in the performance estimate, you will need to have your board configured, connected, and powered on. Make sure that your board is connected and configured to boot from the SD card (if the SD card is absent, the device will default to JTAG mode).

Removing Breakpoints

1-1. Remove a breakpoint from a line of code from the editor window.

1-1-1. Right-click the breakpoint that you want to remove.

1-1-2. Select **Toggle Breakpoint**.

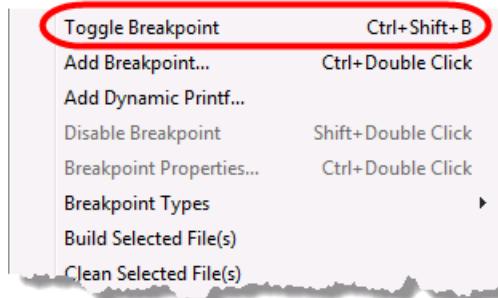


Figure 453: Toggling a Breakpoint Into and Out of Existence

-- OR --

Breakpoints can also be removed from the Breakpoints window.

1-1-3. Right-click the breakpoint that you want to remove.

1-1-4. Select **Remove** from the context menu to delete the selected breakpoint.

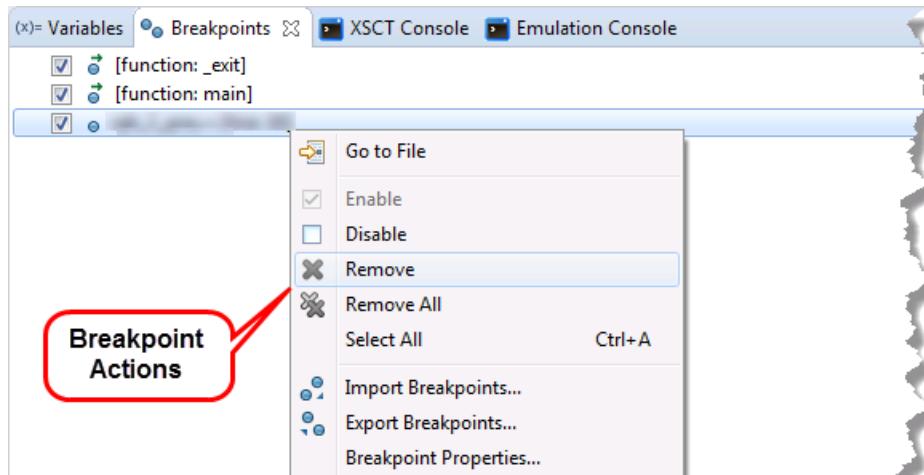


Figure 454: Actions from the Breakpoint Window

Adding a Watchpoint to a Global Variable

1-1. View the Outline tab.

- 1-1-1. Select the **Outline** tab to see the outline of the current source file.

Tip: If the outline is not visible, select **Window** (1) > **Show View** (2) > **Outline** (3).

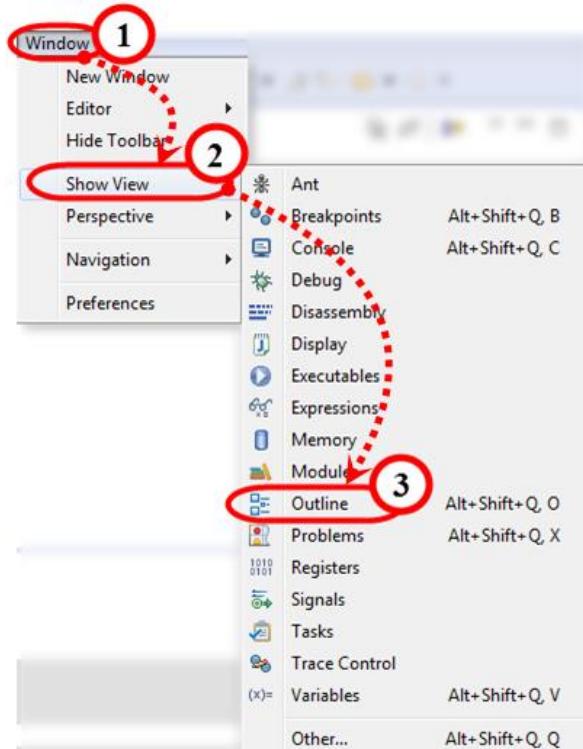


Figure 455: Opening the Outline View

1-2. Select the global variable to add a watchpoint to it.

- 1-2-1. Right-click the global variable **the symbol name (and optionally a symbol value)** (1).
- 1-2-2. Select **Toggle Watchpoint** from the context menu (2).

Tip: Each type of entry in the Outline view has a symbol that distinguishes its type. The symbol for variables with a global scope is (●).

- 1-2-3. Select the **Read** option so that if this variable is read, it will trigger a pause in execution (3).
- 1-2-4. Select the **Write** option if it is unchecked so that, like the Read option, if the variable is written to, it will trigger a pause in execution (4).

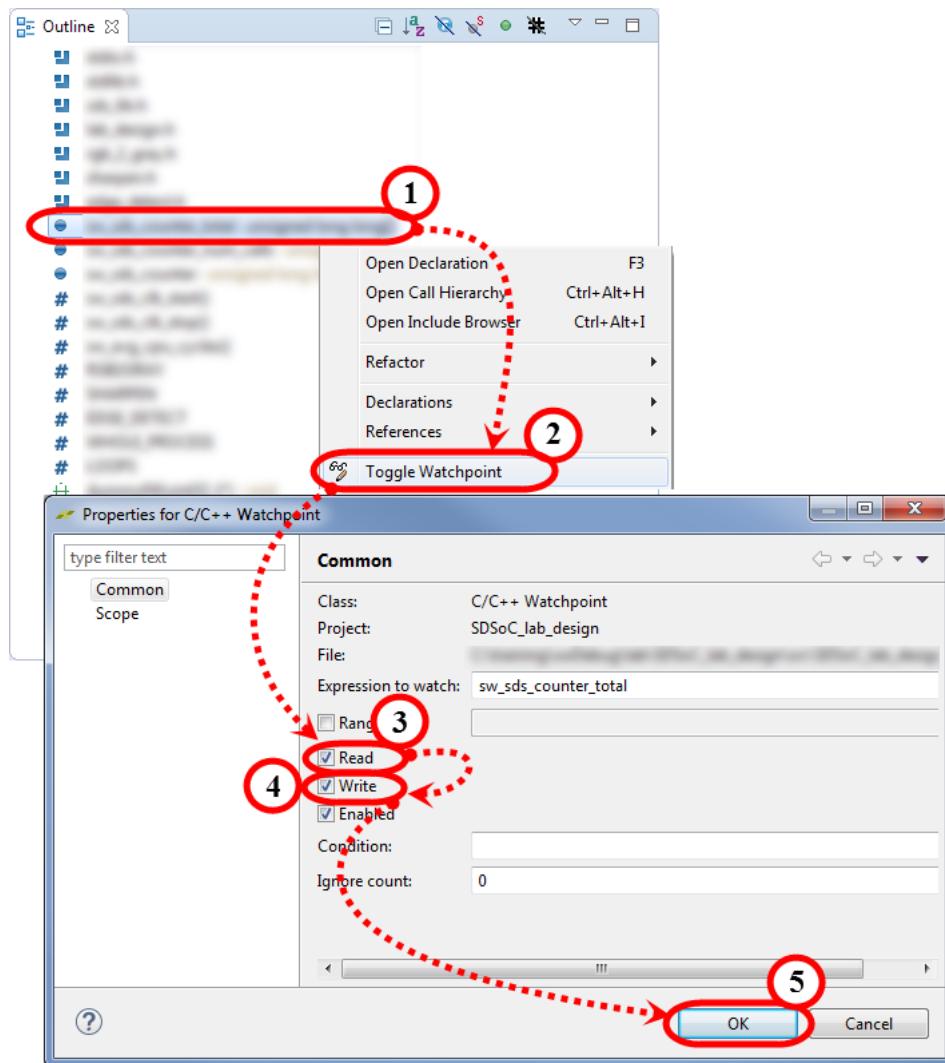


Figure 456: Toggling the Watchpoint

- 1-2-5. Click **OK** (5).

Note: The watchpoint is now visible in the Breakpoints tab under the Debug perspective.

Selecting a Build Configuration

1-1. Set the build configuration to the needed build configuration (SDEstimate, SDDebug, or SDRelease).

- 1-1-1. Right-click the project name in the Project Explorer pane to open the context menu.
- 1-1-2. Select **Build Configurations > Set Active > the needed build configuration (SDEstimate, SDDebug, or SDRelease)** to build the desired configurations.

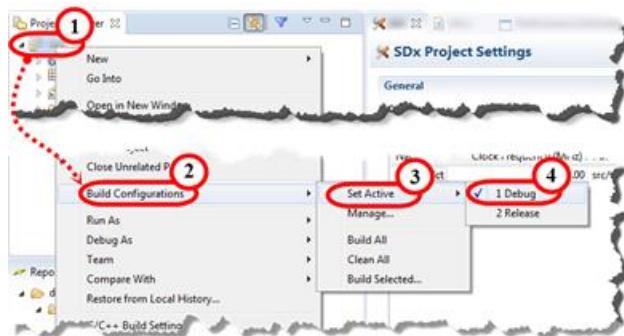


Figure 457: Selecting the Debug Build Configuration (Example)

Marking a Function for Hardware (Dashboard Method)

1-1. Mark *the functions you want accelerated* for hardware acceleration.

- 1-1-1. Expand the project for which you want to mark functions for hardware using the Project Explorer tab.
- 1-1-2. Double-click the **project.sdx** entry to open the main dashboard.
- 1-1-3. Select the tab named with the project (1) to access the Hardware Functions list (2) if it not already active.
- 1-1-4. Click the **Add Hardware Function** icon (⚡) to open the Function Selection Wizard (3).

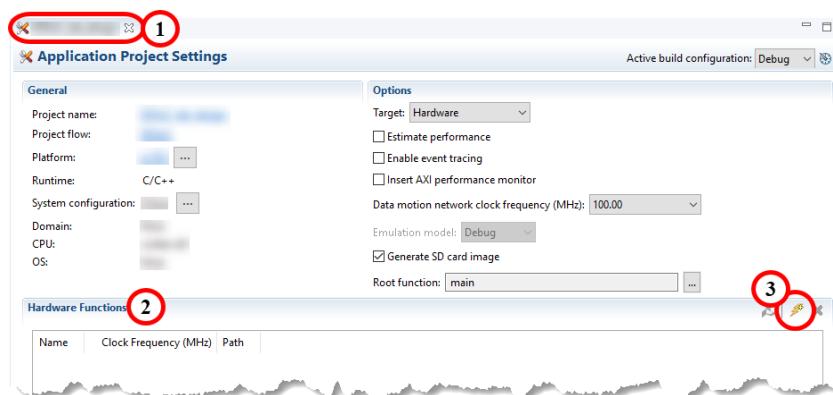


Figure 458: Targeting a Function to Hardware from the SDx Project Overview

The Select Functions for Hardware Acceleration dialog box opens.

- 1-1-5. Select the **the functions you want accelerated** functions to move to hardware (1).

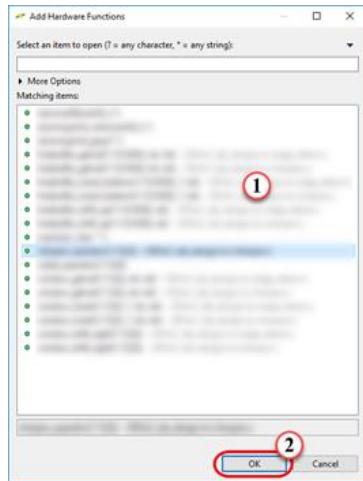


Figure 459: Choosing the Function to Move to Hardware

- 1-1-6. Click **OK** to accept (2).

Note that the selected functions appear in the Hardware Functions region in the dashboard.

Viewing the Generated System Hardware

- 1-1. **Examine the system hardware.**

The SDx tool calls a number of other tools under the hood that build a Vivado Design Suite project.

- 1-1-1. Use the Project Explorer window to navigate to your project > *Release* > *_sds* > *p0* > *vivado* > *prj*.

It is grayed out because it is tool generated. This project can still be opened and edited.

- 1-1-2. Double-click the **XPR** file to open the Vivado Design Suite project.

[Windows users]: If this process does not open the Vivado Design Suite project, you can open the Vivado Design Suite by selecting **Start** > **All Programs** > **Xilinx Design Tools** > **Vivado 2019.1** > **Vivado 2019.1**. Once the Vivado Design Suite opens, click the **Open Project** link and navigate to the XPR file.

[Linux users]: If this process does not open the Vivado Design Suite project, you can open the Vivado Design Suite by pressing **<Ctrl + Alt + T>** to launch a terminal and then entering **vivado** in the terminal. Once the Vivado Design Suite opens, click the **Open Project** link and navigate to the XPR file.

- 1-1-3. Click **Open Block Design** under the Flow Navigator to open the IP integrator view, which will show you the block design for this embedded system.
- 1-1-4. Close the Vivado Design Suite.

Viewing the Generated Accelerator

1-1. Open the Vivado HLS tool projects for an accelerator/hardware function.

- 1-1-1. Expand **SDSoC_lab_design > Debug > _sds > vhls**.

Each accelerator in the design produces a separate Vivado HLS tool project. Each project is stored in its own directory.

- 1-1-2. Expand the desired accelerator's folder.

- 1-1-3. Double-click the **vivado_hls.app** file.

The APP file is the Vivado HLS tool project file. Double-clicking the file will open the Vivado HLS tool project for the selected function.

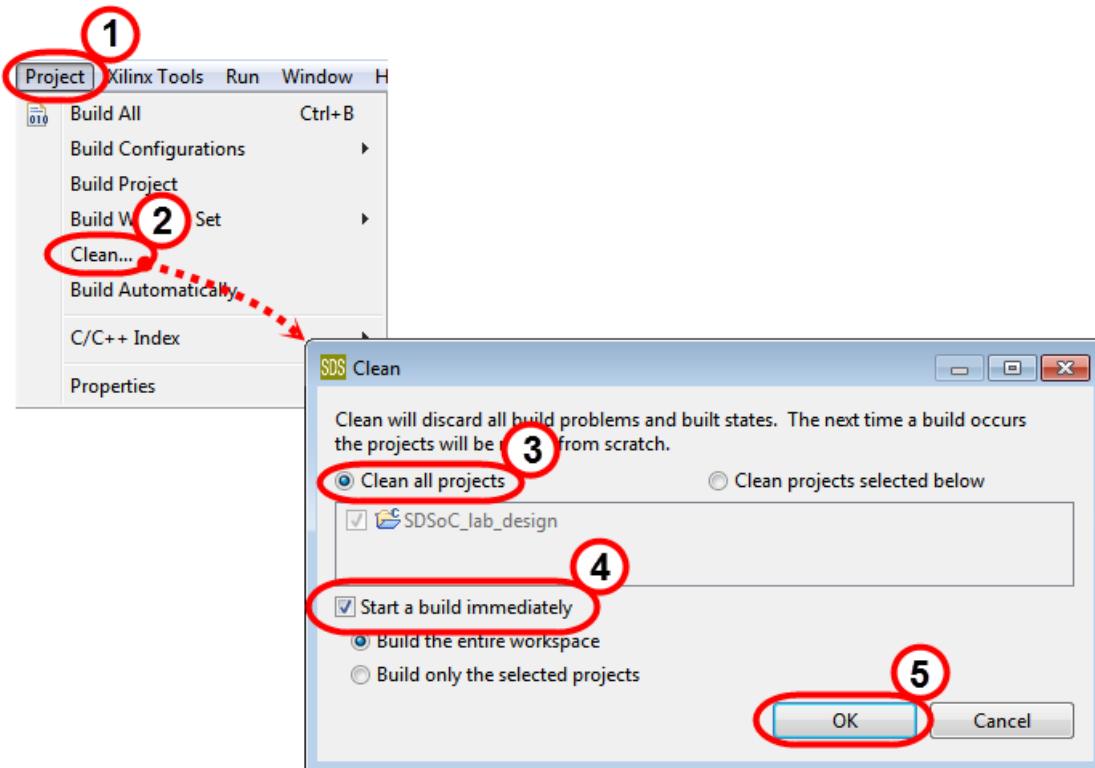
If the file does not open, you can alternatively open the file manually by launching the Vivado HLS tool and navigating to the proper directory under the SDSoC tool project.

Cleaning and Building a Project

1-1. Clean and build the project.

- 1-1-1. Select **Project** (1) > **Clean** (2) to open the Clean dialog box, which controls how many projects in the workspace will be cleaned.
- 1-1-2. Ensure that **Clean all projects** is selected (3).

Since there is only one project in the current workspace, you could have selected "Clean projects selected below" and selected only the project or projects that you wanted.

1-1-3. Ensure that **Start a build immediately** is selected (4).**Figure 460:** Project Clean and Build**1-1-4.** Click **OK** to clean and build the project with the selected functions for hardware acceleration (5).

The report that is generated is for the hardware acceleration only.

Note: If the project did not start building, try once again by selecting **Project > Clean**.

PetaLinux Operations [Last Updated Version 2019.1]

In This Section

Creating a New PetaLinux Project (Using Any BSP)	353
Creating a New PetaLinux Project (Using a BSP)	354
Creating a New Application	354
Creating a New Application (Enable App)	354
Creating a New Application (Enable App) (Using Any BSP).....	355
Selecting the New Application to be Included in the Build Process.....	356
Building the Image	357
Booting the Linux Image on the Board (Running Netboot)	358
Running the Application in QEMU	358
Configuring Network Settings.....	359

Creating a New PetaLinux Project (Using Any BSP)

1-1. Use the `petalinux-create` command to create a new embedded Linux for the chosen platform.

1-1-1. Change to the lab directory:

```
[host] $ cd $<the topic cluster name>/lab
```

This is done so that when the `petalinux-create` command is run, the project is created at this location.

1-1-2. Create a new PetaLinux project:

```
[host] $ petalinux-create -t project -s $tools/your BSP name -n your project name
```

The above command assumes that the board support package (BSP) is installed in the `$tools` directory. Modify the path if the BSP is in a different location.

Note: After the command is executed, information about the created project will be displayed.

Note: The PetaLinux project directory is `$<the topic cluster name>/lab/your project name` and this is where all work for this project must be performed.

1-1-3. Change the directory to the PetaLinux project:

```
[host] $ cd your project name
```

Creating a New PetaLinux Project (Using a BSP)

1-1. Use the petalinux-create command to create a new embedded Linux platform and choose the platform.

- 1-1-1. Enter the following command from the <the topic cluster name> directory to create a new PetaLinux project:

```
[host]$ petalinux-create -t project -s  
/media/xilinx/tools/avnet-digilent-zedboard-v2019.1-final.bsp  
INFO: Create project:  
INFO: Projects:  
INFO:      * avnet-digilent-zedboard-2019.1  
INFO: has been successfully installed to /media/xilinx/training/<the  
topic cluster name>/lab  
INFO: New project successfully created in  
/media/xilinx/training/<the topic cluster name>/lab
```

The above command assumes that the board support package (BSP) is installed in the /media/xilinx/tools directory. Modify the path if the BSP is in a different location.

You should see the project directory (/media/xilinx/training/<the topic cluster name>/lab): avnet-digilent-zedboard-2019.1.

Creating a New Application

1-1. Create a new application.

The PetaLinux tools allow you to create user application templates for either C or C++. These templates include application source code and makefiles so that you can easily configure and compile applications for the target and install them into the root file system.

- 1-1-1. Enter the following command to create a new user application inside a PetaLinux project:

```
[host]$ petalinux-create -t apps --name your application --template  
c
```

Creating a New Application (Enable App)

1-1. Create a new application and enable it in the root file system.

The PetaLinux tools allow you to create user application templates for either C or C++. These templates include application source code and makefiles so

that you can easily configure and compile applications for the target and install them into the root file system.

- 1-1-1. Make sure that you are in the PetaLinux project location.

- 1-1-2. Enter the following command to create a new user application inside a PetaLinux project:

```
[host]$ petalinux-create -t apps --name your_application --enable
```

The `--enable` option automatically enables it in the project's root file system. To create a C++ application template, pass the `--template c++` option.

Creating a New Application (Enable App) (Using Any BSP)

- 1-1. **Create a new application and enable it in the root file system.**

The PetaLinux tools allow you to create user application templates for either C or C++. These templates include application source code and makefiles so that you can easily configure and compile applications for the target and install them into the root file system.

- 1-1-1. Make sure that you are in the PetaLinux project location.

- 1-1-2. Enter the following command to create a new user application inside a PetaLinux project:

```
[host] $ petalinux-create -t apps --name your_application --enable
```

Note: After execution the following information will be printed.

INFO: Create apps: your application

INFO: New apps successfully created in /home/xilinx/training/<the topic cluster name>/lab/your project name/project-spec/meta-user/recipes-apps/your application

INFO: Enabling created component...

INFO: sourcing bitbake

INFO: oldconfig rootfs

INFO: your application has been enabled

Note: The new application that you have created can be found in the <project-root>/project-spec/meta-user/recipes-apps/your application directory, where <project-root> is \$<the topic cluster name>/lab/your project name.

The `--enable` option automatically enables the just created application in the project's root file system. To create a C++ application template, pass `--template c++` as another parameter to the just entered command.

Selecting the New Application to be Included in the Build Process

1-1. Select the new application to be included in the build process. The application is not enabled by default.

- 1-1-1. Ensure that your current working directory (project directory) is /media/xilinx/training/<the topic cluster name>/lab/your project name.
- 1-1-2. Enter the following command to launch the rootfs configuration menu:

```
[host]$ petalinux-config -c rootfs
```

Note: Make sure that the terminal window is at least 80 columns wide. If your terminal is not at least 80 columns wide, you will see "Error: Failed to config linux/rootfs! Your display is too small to run menuconfig."

The linux/rootfs configuration menu opens.

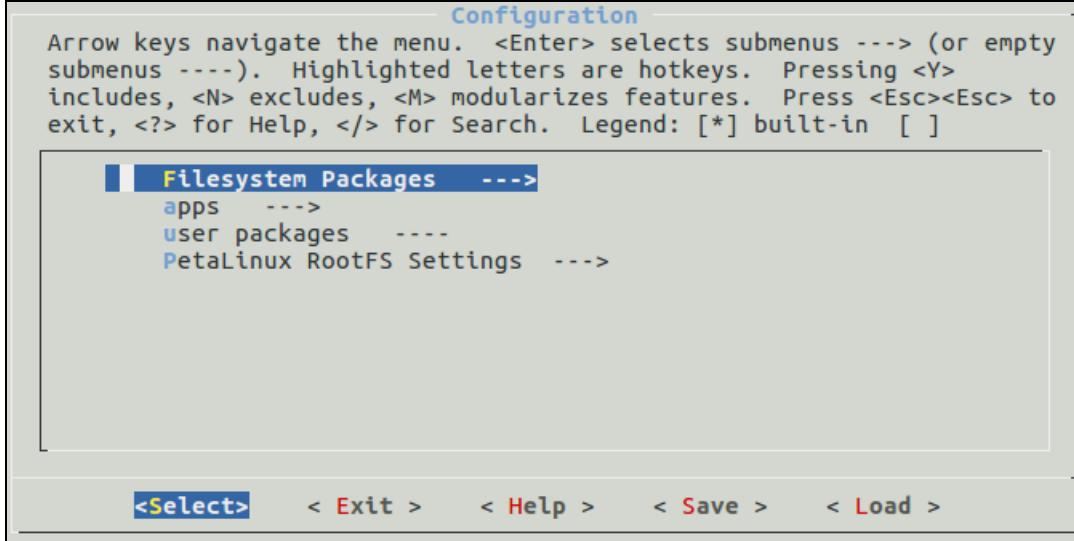


Figure 461: linux/rootfs Configuration Menu

- 1-1-3. Press the **Down Arrow** key (↓) to scroll down the menu to **apps**.
- 1-1-4. Press <Enter> to go into the apps submenu.

The new application **your application** is listed in the menu.

- 1-1-5. Press <Y> to select the application **your application**.

Selecting the application will be included in the build process.

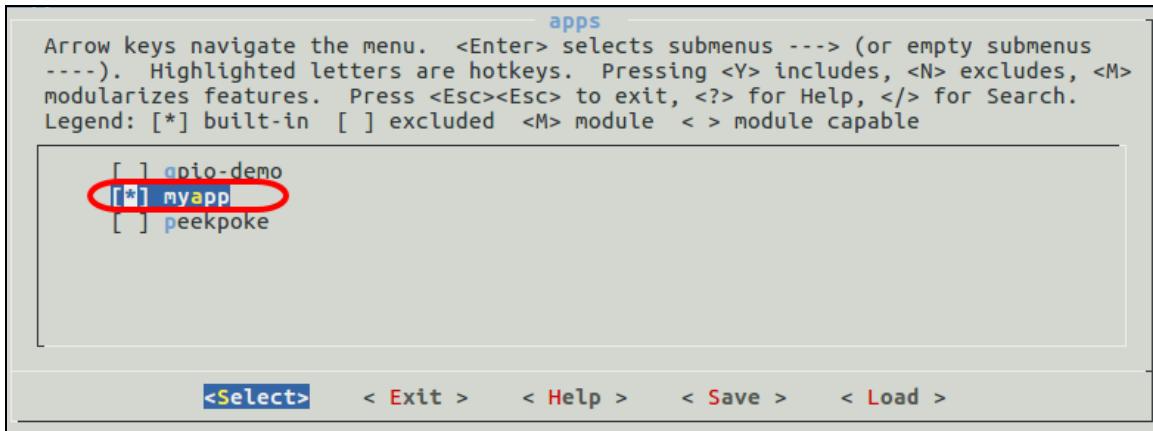


Figure 462: Selecting the Application

- 1-1-6. Select and press **Exit** to return to the Configuration menu.

Building the Image

1-1. Build the Linux image.

This process will take approximately 15-45 minutes based on your system configuration. If you have time you can proceed or else you can skip this step and proceed with using prebuilt images.

Extremely important! Before running the petalinux-build command, make sure that you are connected to the internet.

- 1-1-1. Make sure that you are in the root of the PetaLinux project directory (\$<the topic cluster name>/lab/ZCU102).
- 1-1-2. Enter the following command to build the image:

```
[host]$ petalinux-build
```

Note: This process will take approximately 15-45 minutes based on your system configuration.

Running `petalinux-build` in the project directory will build the system image, including the selected user application your application.

The full compilation log (`build.log`) is stored in the build subdirectory of the PetaLinux project. The Linux software images and the device tree are generated in the `<plnx_project_root>/images/linux` subdirectory of the PetaLinux project.

Booting the Linux Image on the Board (Running Netboot)

1-1. Boot the new Linux image on the board.

- 1-1-1. Press any key to stop auto-boot when you see messages similar to the following in the GtkTerm window:

```
U-Boot [REDACTED] (Feb 06 [REDACTED] - 15:22:46 +0530)

DRAM: ECC disabled 512 MiB
MMC: sdhci@e0100000: 0
SF: Detected S25FL256S_64K with page siize 64 KiB, total 32 MiB
In: serial
Out: serial
Err: serial
Net: ZYNQ GEM: e000b000Hit any key to stop autoboot: 0
Zynq>
```

Figure 463: Stopping the Autoboot (ZedBoard)

If the system has booted, reset the board (BTN7) again and stop auto-boot.

Note: If you have finished setting the IP address and server IP address and saved once, you can directly run the command `run netboot`.

- 1-1-2. Set the IP address for the target board so that it can communicate to the host and load the new image:

```
Zynq> setenv ipaddr 192.168.1.10
```

- 1-1-3. Set the TFTP server IP to the host IP:

```
Zynq> setenv serverip 192.168.1.1
```

- 1-1-4. Save the environment to the SPI Flash:

```
Zynq> saveenv
```

- 1-1-5. Download and boot the new image using TFTP:

```
Zynq> run netboot
```

This command will download the `image.ub` file from `/tftpboot` on the host to the main memory of the ARM Cortex-A9 MPCore system and boot the system with the image.

Running the Application in QEMU

1-1. Run the application in QEMU.

- 1-1-1. Enter the following command to boot the newly built PetaLinux image through QEMU:

```
# petalinux-boot --qemu --kernel
```

- 1-1-2. After the system boots, log into the system by entering `root` as both the login name and password.

Configuring Network Settings

1-1. Launch VirtualBox.

- 1-1-1. Select **Start Menu > All Programs > Oracle VM VirtualBox > Oracle VM VirtualBox** to start the VirtualBox Manager program.
- 1-1-2. Select the **MPSoC_Ubuntu** virtual machine titled from the left pane (1).
- 1-1-3. Double-click the virtual machine (1) or click the green right arrow icon to launch the selected virtual machine (2).

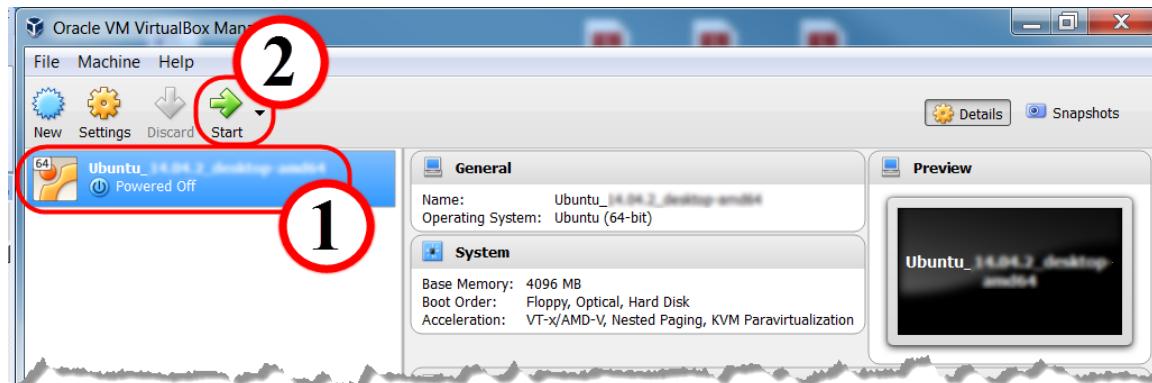


Figure 464: VirtualBox Manager Window

The MPSoC_Ubuntu virtual machine will now launch in a new window.

- 1-1-4. Click the **Maximize** (□) icon in the window title bar to make it full screen if the virtual machine window is not maximized.

1-2. Disable the Static IP settings.

1-2-1. Press <Ctrl + Alt + T> to open a new terminal window.

1-2-2. Enter the following command to open the interface file:

```
gedit /etc/network/interfaces
```

1-2-3. Comment all the lines in the interfaces file except the two lines as shown below:

```
#interfaces(5) file used by ifup(8) and ifdown(8)

auto lo

iface lo inet loopback

#auto eth0

#iface eth0 inet static
#address 192.168.1.1
#netmask 255.255.255.0
#broadcast 192.168.1.255
#gateway 192.168.1.
```

1-2-4. Press <Ctrl + S> to save the changes.

1-2-5. Click <X> in the upper-left corner of the gedit window to exit the editor.

1-2-6. Enter the following command in the terminal to shut down the OS:

```
sudo poweroff
```

1-2-7. Enter **xilinx** as password when asked to enter a password.

1-3. Change the VirtualBox network settings.

1-3-1. Select **Network** as shown in the figure below.

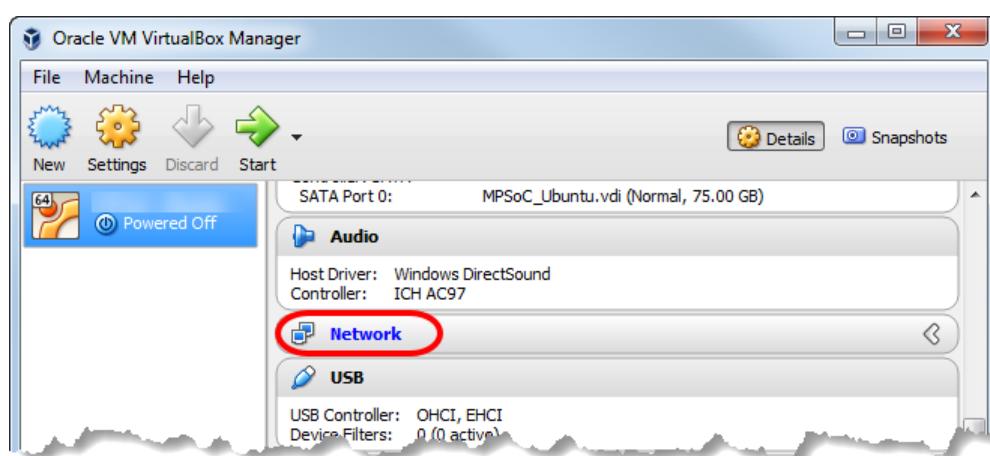


Figure 465: Selecting the Network Tab

- 1-3-2. Select the **Enable Network Adapter** option to enable the network adapter.

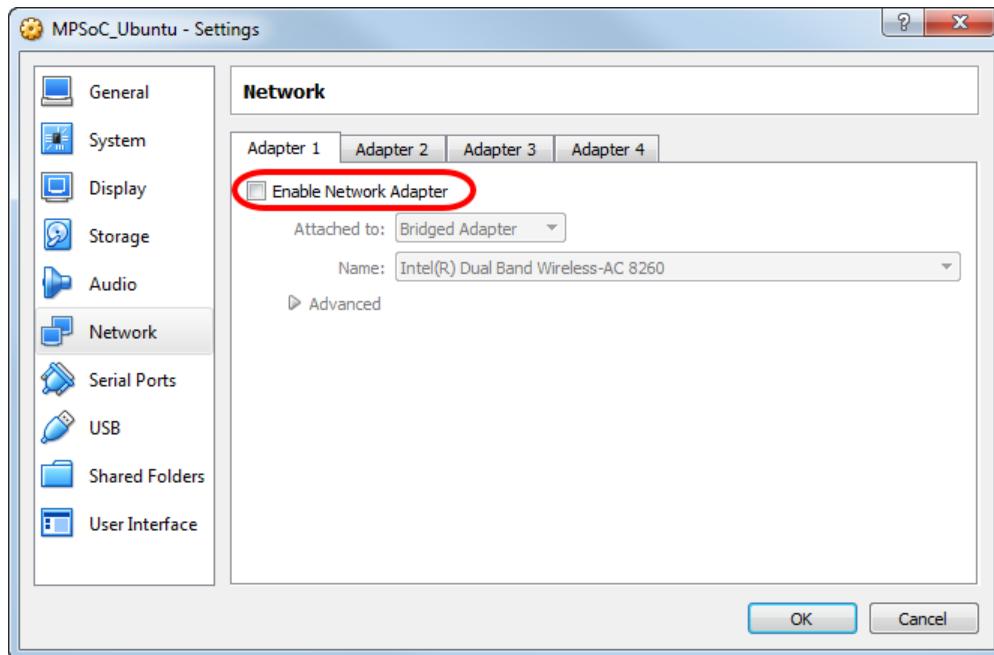


Figure 466: Enabling the Network Adapter

- 1-3-3. Select the appropriate name option from the drop-down list.

This will vary from system to system. The figures shown here are for illustration purposes only.

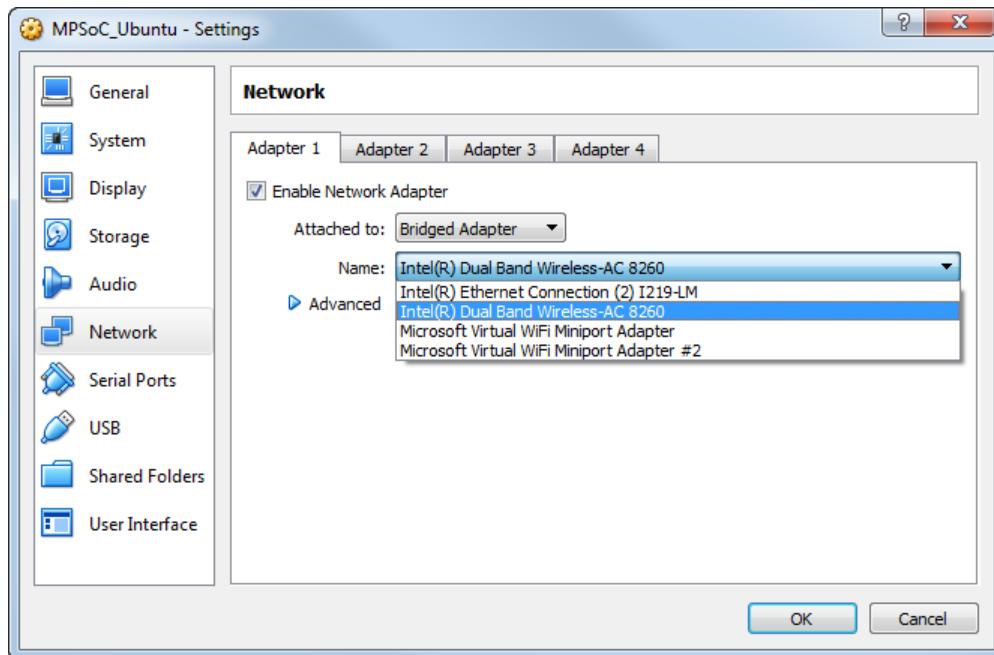


Figure 467: Selecting the Appropriate Network Connection

This example shows that **Intel(R) Dual Band Wireless-AC 8260** (or the wired network **Intel(R) Ethernet Connection (2) I219-LM**) is selected because it is the name of our network connection on the Windows machine running at the backend as shown below.

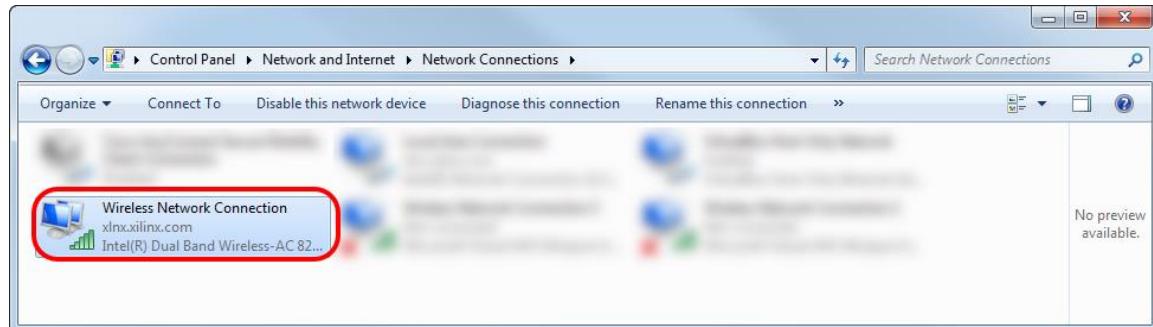


Figure 468: Network Connection on Windows Machine

You will have to check the network connection of your Windows machine and then select the appropriate name in the VirtualBox.

- 1-3-4. Select the **Advanced** option in the Adapter tab in the VirtualBox to show the extra configuration parameters.
- 1-3-5. Edit the configuration as shown below.

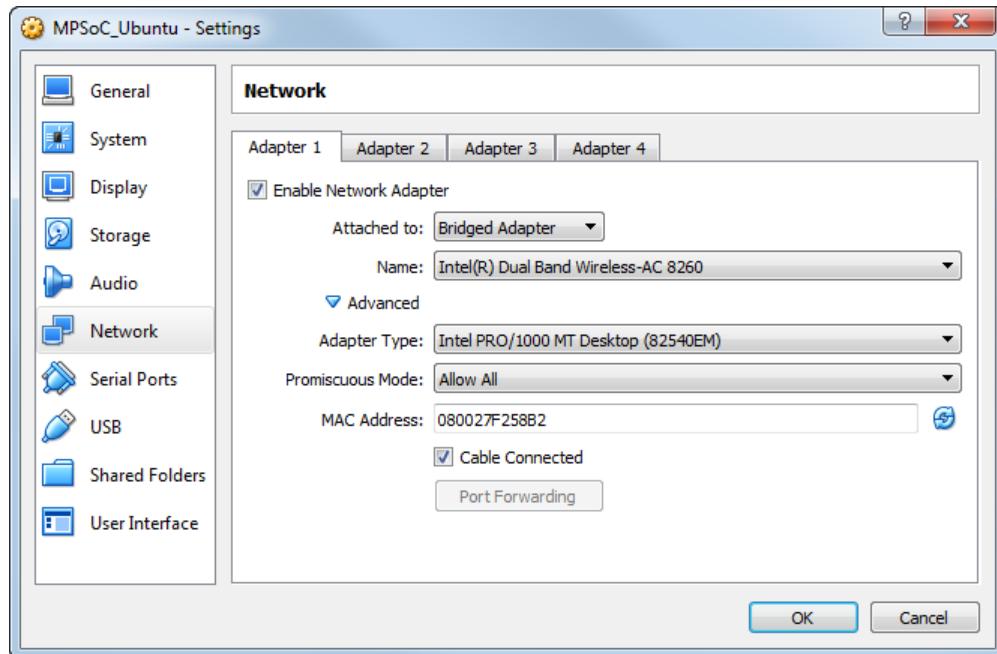


Figure 469: Configuring Advanced Options

Note: The network connection name may be different in your environment.

- 1-3-6. Click **OK** to bring the changes into effect.

Board, OS, COM, and IP Address Tasks [Last Updated Version 2019.1]

In This Section

Configuring the SDK or SDSoc Terminal for the ZC702 or Tera Term for the ZedBoard	363
Configuring the Terminal.....	366
Determining the COM Port Assigned by the USB Driver.....	368
Configuring the SDK Terminal	370
Linux Operations	372
Running a Virtual Machine Using VirtualBox.....	375
Cleaning Up the VirtualBox File System	376
Setting the Static Host IP Address on a PC (Windows 7).....	376
Setting Up the Hardware – ZC702 and ZedBoard	382
Verifying Board Jumper/Switch Settings Configured to Boot from the SD Card	384
Powering Up the Hardware Platform	385
Preparing an SD Card [Common for Windows and Linux].....	385
Booting Linux.....	386
Launching and Configuring the Tera Term Terminal Program in Serial Port Mode.....	387
Connecting the ZC702 Board	390
Connecting the ZedBoard.....	392
Connecting the ZedBoard without the FMC-CE Card	393
Setting the Jumpers on the ZC702 Board.....	394
Setting the Jumpers on the ZC706 Board.....	396
Setting the Jumpers on the ZCU102 Board.....	399
Setting the Jumpers on the ZedBoard.....	401
Setting Up the Hardware - KC705	402
Setting up the Hardware - KCU105.....	405

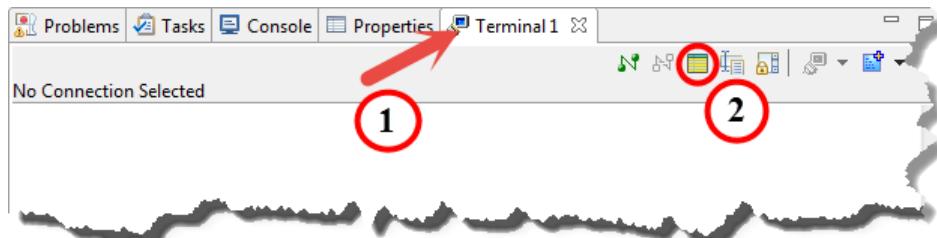
Configuring the SDK or SDSoc Terminal for the ZC702 or Tera Term for the ZedBoard

- 1-1. ZedBoard users: Skip to the instruction below that begins with "ZedBoard users: Open the Tera Term terminal program."**

ZC702 board users: Open the SDK or SDSoc terminal program.

- 1-1-1.** Select the **Terminal** tab to access the terminal icons.

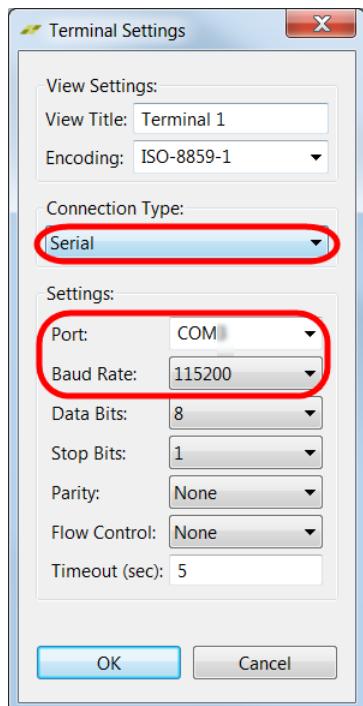
If the Terminal tab is not visible, select **Window > Show View > Terminal**. Note that more than one terminal can be enabled at a time.

1-1-2. Click the **Settings** icon ().**Figure 470:** Accessing the Terminal Settings

Alternatively, you can also click the **Connect** icon () to open the Terminal Settings dialog box. If the terminal was previously configured, this will open it with those settings and connect to the associated COM port.

1-1-3. Configure the settings as follows:

- o Select the connection type as **Serial**
- o Select the port as the COM # discovered previously (in another instruction)
- o Set the baud rate to **115200**

**Figure 471:** Configuring the Terminal Settings**1-1-4.** Click **OK**.

The terminal session will be connected to the associated COM port on the PC.

- 1-2. ZC702 board users: You have already associated a serial terminal with your board. Skip to the next step.**

ZedBoard users: Open the Tera Term terminal program.

- 1-2-1.** From the Windows desktop, double-click the **Tera Term** icon to launch Tera Term.

Alternatively, you select **Start > All Programs > Tera Term > Tera Term**.

- 1-2-2.** Select **File > New Connection**.

- 1-2-3.** Select **Serial** as the connection (1).

- 1-2-4.** Click the **Port** drop-down list to view the available COM ports.

Note: If your port is not listed, exit Tera Term, power cycle your board and restart this step.

- 1-2-5.** Select the COM # discovered previously (3).

Note that the ZC702 will show the Silicon Labs driver as shown in the figure below and the ZedBoard will show the Cypress driver.

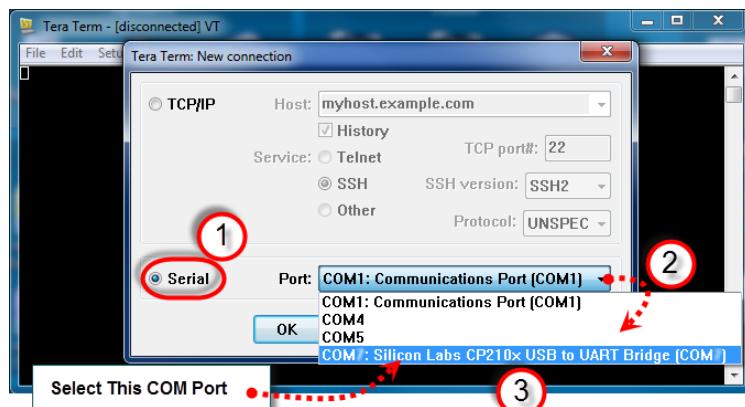
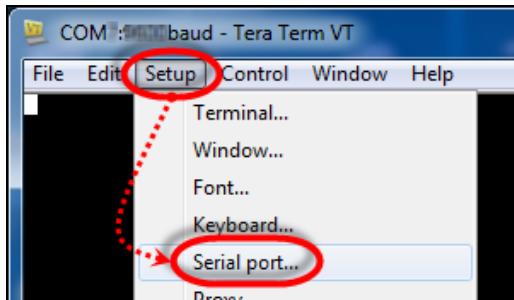


Figure 472: Selecting the COM Port

Note: The COM port setting is specific to the computer being used and may need to be different than shown. Use the COM port # that was discovered previously.

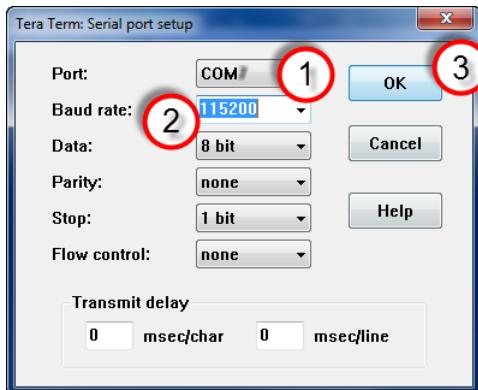
- 1-2-6.** Click **OK**.

The terminal console window opens.

1-2-7. Select **Setup > Serial Port**.**Figure 473:** Opening the Tera Term Serial Port Setup Window

The Tera Term Serial Port Setup dialog box opens.

- 1-2-8.** Confirm that the proper serial port has been selected (1).
- 1-2-9.** Set the baud rate to **115200** (2).

**Figure 474:** Setting the Parameters for the Serial Port

Note: The COM port setting is specific to the computer being used and may need to be different than shown. Use the COM port # that was discovered previously.

- 1-2-10.** Click **OK** (3).

Tera Term is now configured to receive and transmit serial information to/from the evaluation board.

Configuring the Terminal

The Terminal tab is used with a serial source (UART), SSH, or Telnet session. The following instructions will show you how to configure this tab to work with a serial UART.

1-1. Open the Terminal tab.

- 1-1-1. If the Terminal tab is not visible, select **Windows > Show View > Other > Terminal > Terminal** to open it or create a new Terminal tab.

This will open a Terminal tab in one of the panels. The tab can be moved to any other panel by click-dragging it to the new panel.

Note: More than one terminal can be enabled at a time.

- 1-1-2. Select the **Terminal** tab to access the terminal icons (1).

- 1-1-3. Click the **Settings** icon (2).

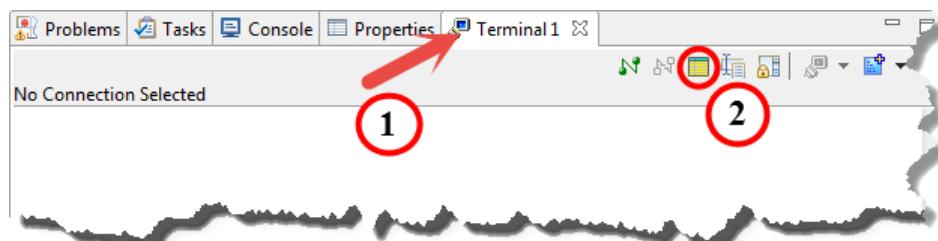


Figure 475: Accessing the Terminal Settings

Alternatively, you can also click the **Connect** icon () to open the Terminal Settings dialog box. If the terminal was previously configured, this action will not open the Terminal Settings dialog box, rather it will attempt to open the communication channel with the previously selected settings.

Changing the settings requires that you close the channel and use the Settings icon.

1-2. Configure the settings as shown in the following figure.

1-2-1. Select the connection type as **Serial**.

1-2-2. Select the proper port for the COM #.

Your board must be on and connected; otherwise the USB bridge is not enumerated and your COM port will not appear.

1-2-3. Set the baud rate to **115200**.

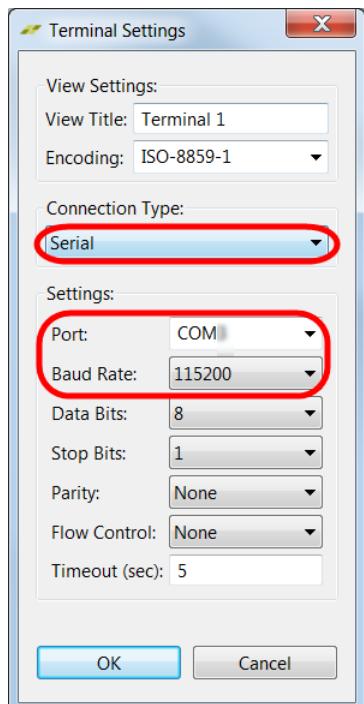


Figure 476: Configuring the Terminal Settings

1-2-4. Click **OK**.

The terminal session will be connected to the associated COM port on the PC.

Determining the COM Port Assigned by the USB Driver

A serial UART terminal emulator program provides a simple and straightforward way to collect and transmit serial information using the RS-232 protocols. Most modern PCs lack the traditional DB-9 connectors for RS-232 communication and instead use USB ports.

From the PC's perspective there is still a COMx port being used; however, the actual COM port number is not known until USB enumeration. The PC is capable of supporting multiple COM ports, so it is important to know which connection is the one to use when communicating with the development board.

When using a PC COM port with communications software (terminal emulator) on the PC such as the SDK Terminal tab, Tera Term, or other software, it is necessary to identify the COM port number associated with serial connection to the evaluation board. This is done by identifying the USB COM port driver and which COM port is being assigned.

The Xilinx MPSOC boards support three USB ports. Two are connected to the UARTs in the PS and one is available to the PL, assuming that there is a UART implemented in the PL.

1-1. Determine which COM port is connected to the USB serial port from the board.

- 1-1-1. Make sure that the development board is powered on and the serial UART device USB cable is in place.

This ensures that the USB-to-serial bridge will be enumerated by the PC host.

The Device Manager in Windows lists all of the hardware items in the system. While there are a number of routes to opening the device manager, the easiest is as follows.

- 1-1-2. Click **Start** (1).
- 1-1-3. Enter the following into the search bar (2):

device manager



Figure 477: Accessing the Windows Application Search

Note: You may be asked to confirm opening the Device Manager. If so, click **Yes**.

1-1-4. Expand **Ports (COM & LPT)** so that you can see all of the enumerated ports.

Various boards use different USB bridge chips. The name of the driver varies correspondingly.

- o **ZCU102/4 users:** Locate **Silicon Labs Quad CP210x USB to UART Bridge: Interface 0 (COM#)**.

- Note that there will be a group of (at least) four COM ports listed. This is because this board has a quad USB to UART bridge. Interface 0 and 1 are attached to stdout 0 and 1, respectively. Also note the COM port number assigned to *Silicon Labs Quad CP210x USB to UART Bridge: Interface 0 (COM#)*"as this is the COM port number you will need when establishing communications with the board, specifically the output from the PMU.

Note: The # indicates the port number for this serial connection.

Example:

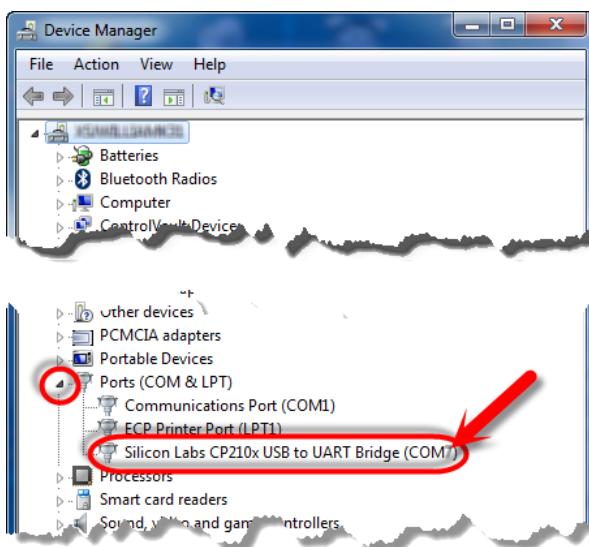


Figure 478: Locating the Silicon Labs Com Port Driver

1-1-5. Note the COM# corresponding to the USB bridge chip.

This is the port number you will use when connecting a terminal to the board.

1-1-6. Close the Device Manager by clicking the red 'X' in the upper-right corner of the panel.

Configuring the SDK Terminal

[Windows users]: The SDK terminal is an interface that only supports serial port/UART communications. The more general Terminal tab is able to support other formats, such as SSH and Telnet.

1-1. Locate the SDK Terminal tab. Generally this tab is found in the same panel as the console.

- 1-1-1.** If the SDK Terminal tab is not currently visible, select **Window > Show View > Other > Xilinx > SDK Terminal** to open it.

This tab typically opens in the same window as the console.

1-2. Configure the SDK Terminal.

- 1-2-1.** Click the green '+' sign to open the Connect to Serial Port dialog box.

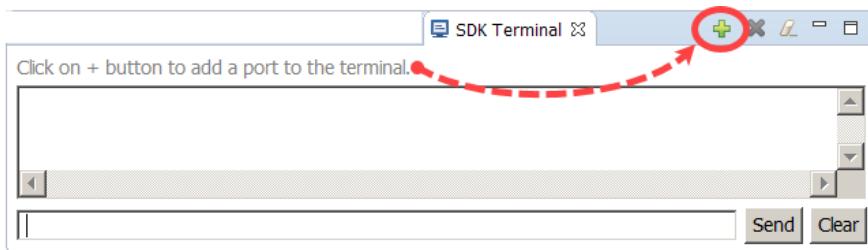


Figure 479: Adding/Associating a Port to the Terminal

A pop-up appears asking you to configure the settings for the serial port.

- 1-2-2.** Select the serial port that is connected to the device you want to communicate with (1).

This is the port number associated with the Serial Port/USB connection from your board.

This is often the highest-numbered com port, but not always. Your board must be powered on in order to see this port.

- 1-2-3.** Set the baud rate to **115200** (2).

1-2-4. Leave the other settings at their defaults.

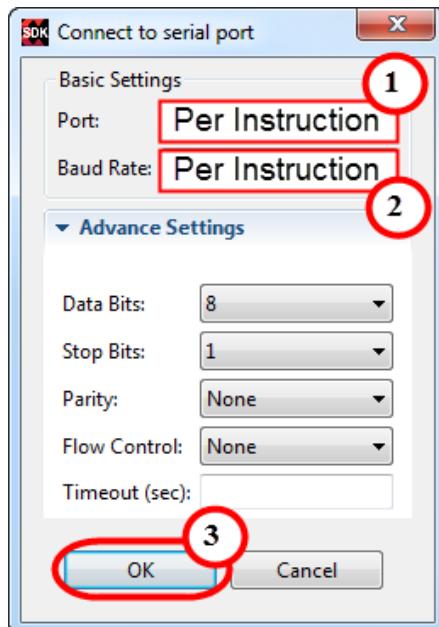


Figure 480: Configuring the SDK Terminal

1-2-5. Click **OK** to save these settings and begin the terminal session (3).

Linux Operations

Opening a Terminal – Linux Ubuntu

A Linux terminal allows instructions to be entered directly via the command line interface, rather than executed by the operating system through a GUI. A terminal can be used to completely configure and control a Linux system without the need for a graphical desktop.

Even when there is a graphical desktop available for use, it may sometimes be necessary or more convenient to use the terminal window to launch programs and configure settings.

1-1. Open a Linux terminal window.

- 1-1-1. Click the Linux terminal window icon in the taskbar.

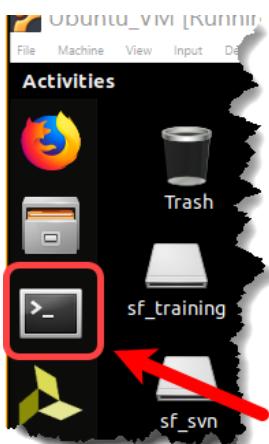


Figure 481: Ubuntu – Locating the Terminal

Alternatively, you can press <Ctrl + Alt + T>.

The terminal window will open, leaving you at your home directory.

Copying Lab Files to the /home/xilinx/training Directory

1-1. Copy the files and the directory structure from the shared Windows folder to your *training* directory in Linux.

1-1-1. Press <Ctrl + Alt + T> to open a new terminal window.

1-1-2. Enter the following command to run a shell script in the Linux terminal window:

```
[host]$ source /media/sf_training/setup_TopicCluster.sh  
MPSOC_HSV_Build
```

This script will copy the starting point of the lab from the Windows shared directory to your working directory. The starting point contains any files required by this lab.

Shutting Down Ubuntu Linux

1-1. [Windows users]: Shut down Ubuntu Linux.

1-1-1. Click the **System Settings** icon in the upper-right corner of the Ubuntu desktop (1).

1-1-2. Select **Shutdown** (2).

The Shutdown dialog box opens.

1-1-3. Click **Shutdown** (3).

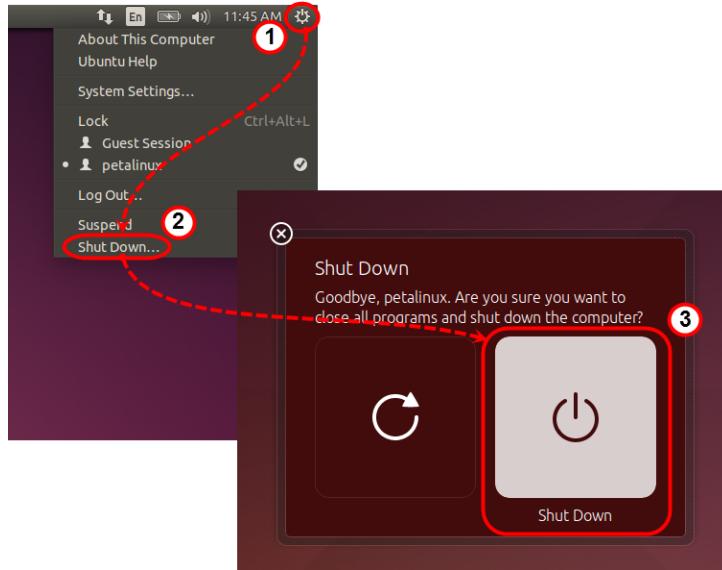


Figure 482: Shutting Down

Running a Virtual Machine Using VirtualBox

1-1. Launch the VirtualBox Manager application.

The instructions for this lab are provided for the Linux environment. While Microsoft Windows works for most development situations, there are some tools that are not yet supported under the Windows operating system. For the sake of consistency, all Xilinx Customer Education labs are delivered using the Linux environment.

Running Linux under Windows is easily achieved using a virtual machine that forms an isolated container for another OS. The Oracle VirtualBox application was selected as the framework for hosting virtual machines as it works well and is available without the hassles of licensing. The Xilinx Customer Education team provides a VM that is properly configured to run the labs and demos. If you need to install VirtualBox and the VM, consult the lab setup guide in which this procedure is described in detail.

- 1-1-1. Select **Start Menu > Oracle VM VirtualBox > Oracle VM VirtualBox** to launch the VirtualBox Manager program.

Alternately, if a desktop or taskbar icon is available, you can simply click it.

When the VirtualBox Manager opens, the left-hand pane will show all of the available virtual machines. If name of the virtual machine is absent, refer to the lab setup guide for instructions on how to download, install, and configure the VM.

- 1-1-2. Double-click the **name of the virtual machine** virtual machine (1).

Alternately, you can single-click the virtual machine of choice, then click the green right arrow icon to launch the selected virtual machine (2).

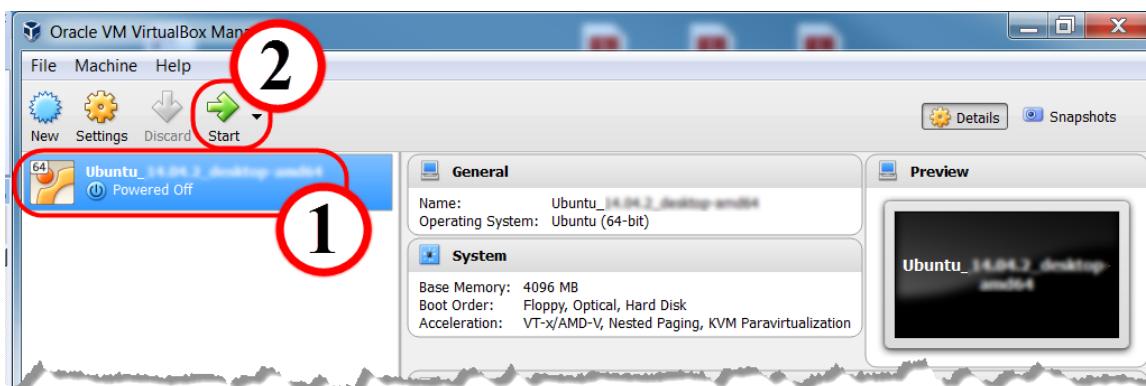


Figure 483: VirtualBox Manager Window

The name of the virtual machine virtual machine will now launch in a new window. It takes a few moments to boot. If any warning messages appears, such as anything regarding the mouse or keyboard, click the "x" on the line with the error and continue.

- 1-1-3. Click the **Maximize** (□) icon in the window title bar to expand the screen to its full size, if not already maximized.

Cleaning Up the VirtualBox File System

[Optional]: Clean up the file system.

Some systems may be memory constrained. Removing the workspace frees a portion of the disk space, allowing other labs to be performed. You can delete the directory containing the lab you just ran by using the graphical interface or the command line interface. Both processes will recursively delete all of the files in the \$<the topic cluster name> directory.

1-1. Clean up the VirtualBox file system via the GUI.

- 1-1-1. Using the graphical browser (Windows: press the <**Windows**> key + <**E**>; Linux: press <**Ctrl + N**>), navigate to \$<the topic cluster name>.
- 1-1-2. Select <**the topic cluster name**>.
- 1-1-3. Press <**Delete**>.

1-2. Clean up the VirtualBox file system via the command line.

- 1-2-1. Open a terminal window (Windows: press the <**Windows**> key + <**R**>, then enter **cmd**; Linux: press <**Ctrl + Alt + T**>)
- 1-2-2. Enter the following command to delete the contents of the workspace:

[Windows users]: `rd /s /q $<the topic cluster name>`

[Linux users]: `rm -rf $<the topic cluster name>`

Setting the Static Host IP Address on a PC (Windows 7)

These instructions illustrate how to change to or set up a *static* IP address on the host (laptop) computer. Follow the reverse of these instructions to change back to a DHCP-received address.

For labs that require an Ethernet connection between the laptop host computer, the hardwired Ethernet settings on your host machine must be set up properly to work with the hardware evaluation board. Typically, the PC's Ethernet port is set to automatically request an IP address from the network DHCP client server. Because the hardwire Ethernet connection will attach directly to the hardware evaluation board, there will not be a DHCP server to supply the needed IP address for the host computer. It is therefore necessary to reconfigure the TCP/IP client (laptop) with its own *static* IP address. After you are finished with this lab, you can revert this step and set TCP/IP properties to obtain an IP address automatically.

Most Xilinx software applications use an IP address of 192.168.1.11. The host computer is required to have an IP address of 192.168.1.**num**, where **num** is any number between 2 and 255 except 11 (which is reserved for the Xilinx hardware).

Windows

1-1. Set the static host IP address.

- 1-1-1. If wireless is on, it is suggested that you **turn if off** with the switch on your computer or disable in settings.

How to perform this will vary with different PC wireless hardware.

While this step should not be necessary, there has been reports that the wired Ethernet port does not work with static address (192.168.1.**num**) using the same base subnet address of the wireless adapter.

- 1-1-2. **[Windows 7 users]:** Click **Start** (1).

- 1-1-3. **[Windows 7 users]:** Enter the following into the search bar (2):

network and sharing center

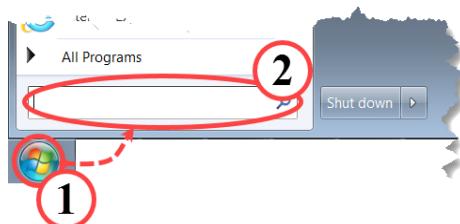


Figure 484: Accessing the Windows Application Search

Note: You may be asked to confirm opening the Device Manager. If so, click **Yes**.

[Windows 10 users]: Right-click the networking icon on the bottom-right corner of the taskbar (1).

[Windows 10 users]: Select **Open Network and Sharing Center** (2).

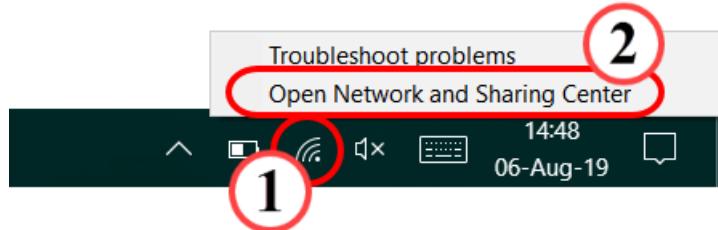


Figure 485: Selecting Open Network and Sharing Center

- 1-1-4.** From the Network and Sharing Center left pane, select **Change Adapter Settings**.

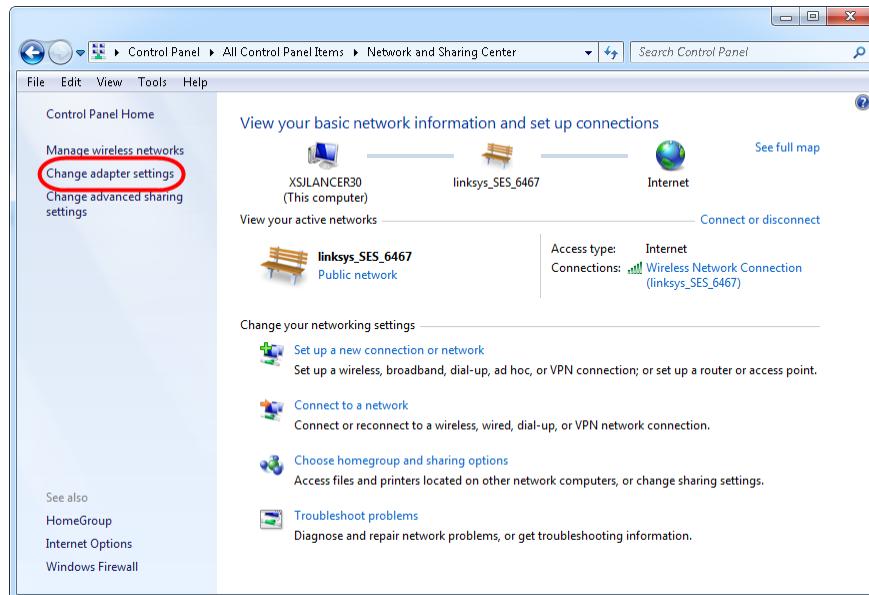


Figure 486: Control Panel Network and Sharing Center

- 1-1-5.** Select the Ethernet adapter to be used.

1-1-6. Right-click and select **Properties**.

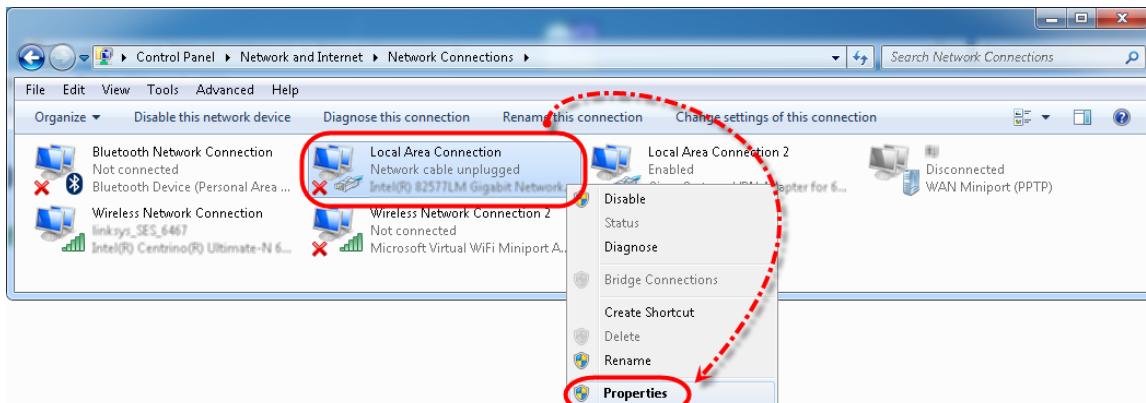


Figure 487: Network Connections

1-1-7. Click **Yes** to make changes (if necessary).

1-1-8. Deselect **Internet Protocol Version 6 (TCP/IPv6)** (1).

1-1-9. Select **Internet Protocol Version 4 (TCP/IPv4)** and click **Properties** (2).

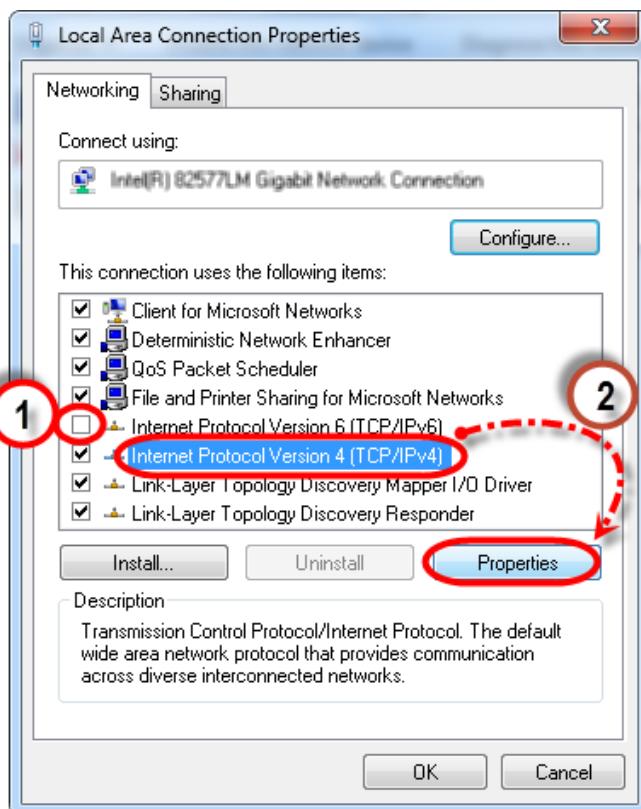


Figure 488: Local Area Connection Properties

1-1-10. Select **Use the following IP address** (1).

1-1-11. Enter **192.168.1.10** in the IP address field (2).

This value is fairly arbitrary, but it cannot be the same as the IP address designated for the board.

The Subnet mask field should fill in with **255.255.255.0** automatically after leaving the IP address field (3).

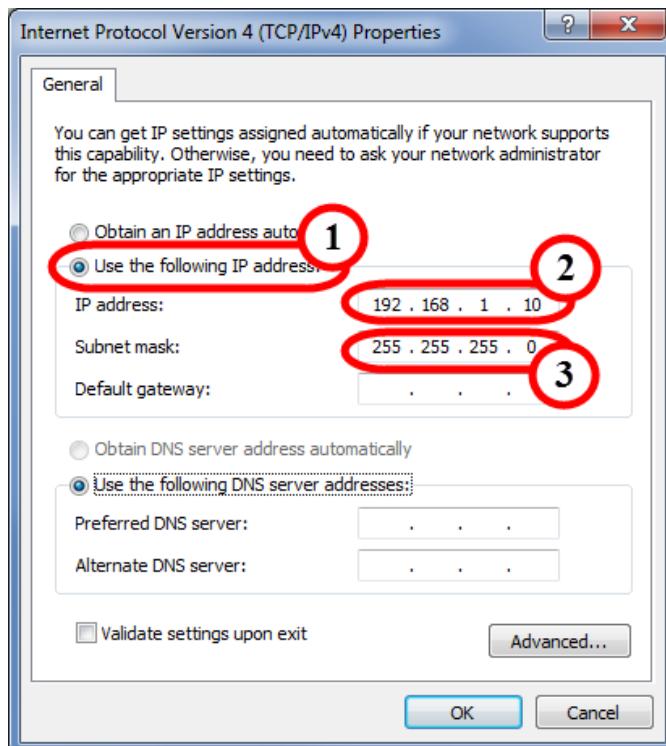


Figure 489: Set TCP-IPv4 Properties for Static Address

1-1-12. Click **OK**.

1-1-13. Click **Close**.

1-1-14. Close the Control Panel.

1-2. To return to a DHCP-obtained address, follow the same steps to revert the settings.

- 1-2-1. Select the **Internet Protocol Version 6 (TCP/IPv6)** option.
- 1-2-2. Select **Internet Protocol Version 4 (TCP/IPv4)** and click **Properties**.
- 1-2-3. Select **Obtain an IP address automatically**.

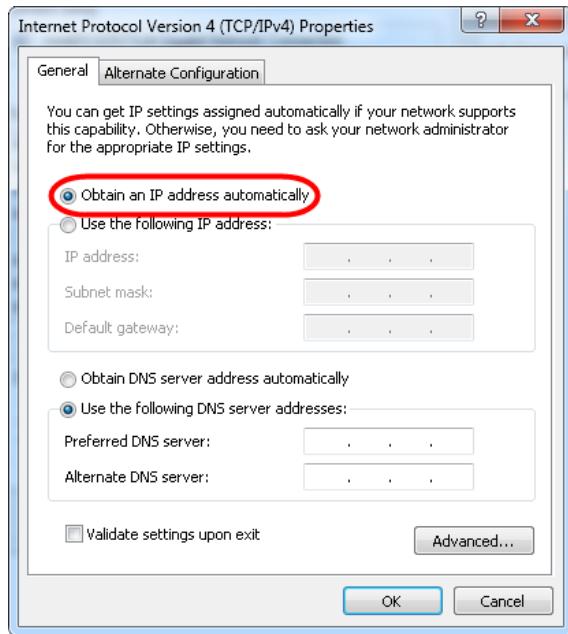


Figure 490: Set TCP-IPv4 Properties for DHCP Obtained Address

1-2-4. Click **OK**.

1-2-5. Click **Close**.

Linux

1-3. Set the static host IP address.

- 1-3-1. Press **<Ctrl + Alt + T>** to open a new terminal.
- 1-3-2. Enter the following command to set the static IP:

```
[host] $ ifconfig eth0 192.168.1.10
```

- 1-3-3. Exit the terminal:

```
[host] $ exit
```

1-4. To return to a DHCP-obtained address, follow the same steps to revert the settings.

- 1-4-1. Reboot the Linux machine or VM to revert the static IP settings that you set in the previous step.

Setting Up the Hardware – ZC702 and ZedBoard

Set up and connect the hardware evaluation board, or verify that this has properly been done before turning on the power. The instructions that follow apply to the ZC702 and ZedBoard. The figures will tell you which one you are using. In some cases, the customer education FMC daughter card may or may not be present, as it is not used in all lab exercises. Its presence in a lab that does not utilize it will not cause a problem.

1-1. Connect the board to your machine as shown below.

1-1-1. Make sure that the following cables are always present:

- Digilent USB port cable: Platform download cable function
 - Micro A male to Type A male
 - ZC702: Connector on U23 module (not shown in the ZC702 figure)
- Power supply
 - ZedBoard: J17
- USB UART port: Provides for COMx communication
 - ZC702: J17 – Mini A male to Type A male
 - ZedBoard: J14 – Micro A male to Type A male
- Power supply

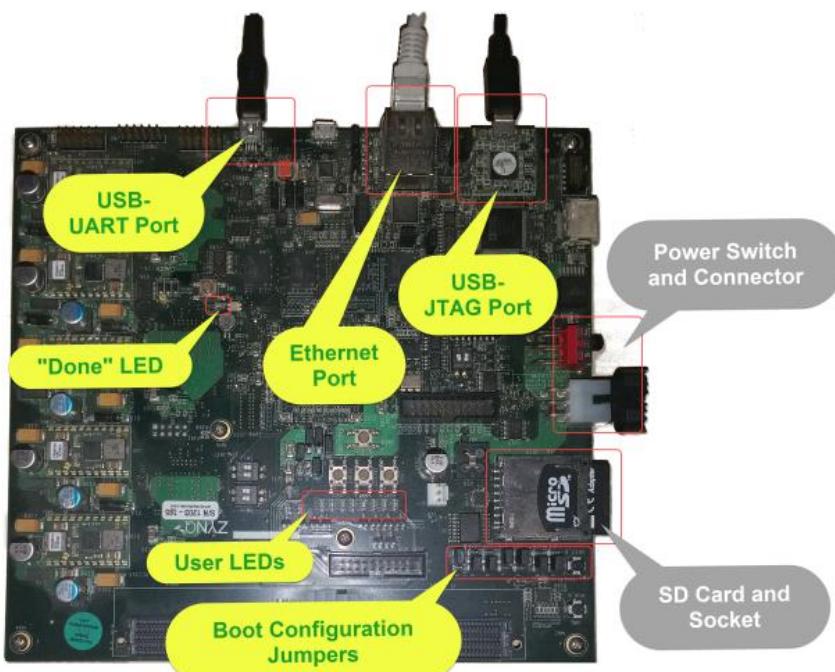


Figure 491: ZC702 Development Board

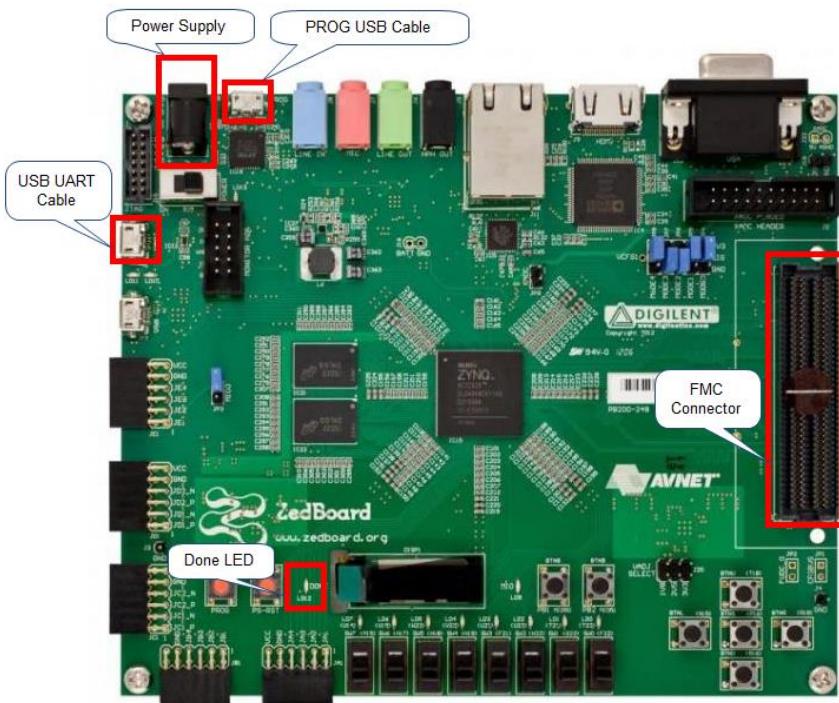


Figure 492: ZedBoard

ZC702 board users: Make sure that the FMC-CE card is plugged into the FMC1 connector.

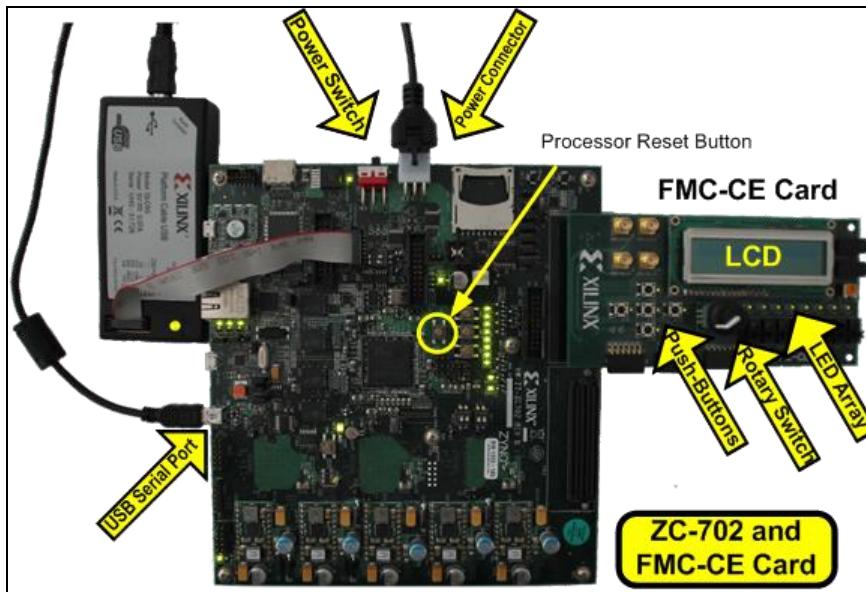


Figure 493: ZC702 and FMC-CE Card

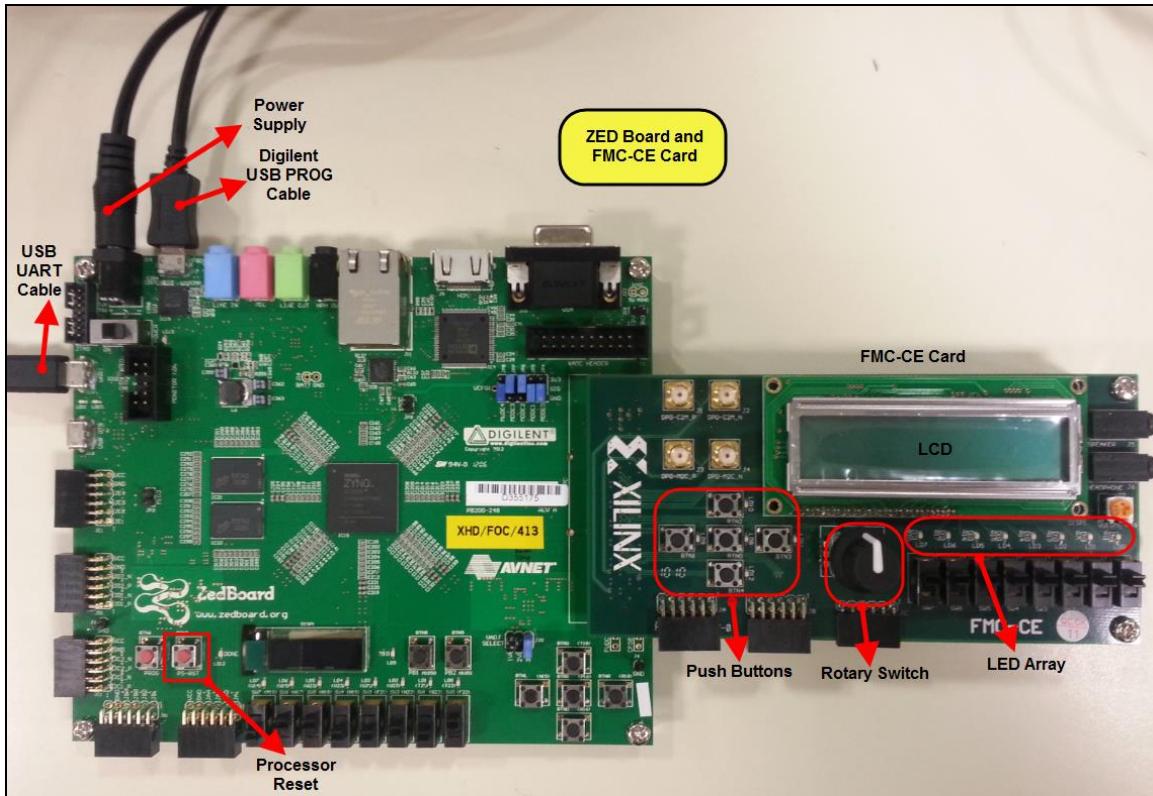


Figure 494: ZedBoard and FMC-CE Card

1-1-2. Power up the board using the mounted slide switch.

Verifying Board Jumper/Switch Settings Configured to Boot from the SD Card

The board evaluation hardware platform has jumpers (or switches) to select the boot mode for the processor.

1-1. Verify that your board hardware platform is properly connected and the jumpers (or switches) are configured to boot from the SD card.

- 1-1-1. Confirm that the power to the evaluation board is OFF.
- 1-1-2. Ensure that the connections between the host PC and the hardware evaluation board are correct (ask for assistance from the instructor if necessary).

The connections include power, serial bridge USB, Ethernet, and the USB Platform download cables.

- 1-1-3. Make sure that the SD card with a Linux image is inserted into the board card slot.

- 1-1-4. Verify the jumper settings on the board are set up to boot from the SD card.

Refer to the "Jumper/Switch Settings your board - Boot from SD Card " topic under the Hardware Requirements - your board Hardware Setup section in the *Lab Reference Guide*, or ask your instructor for assistance.

Powering Up the Hardware Platform

1-1. Apply power to (turn on) the your board hardware platform.

- 1-1-1. Make sure that AC power is connected to the power brick.
- 1-1-2. Ensure that the power brick is connected to the board.
- 1-1-3. Confirm the serial port USB connection between the board and the PC.
- 1-1-4. Confirm the USB-JTAG connection between the board and the PC.
- 1-1-5. Slide the power switch to the on position.

Some LEDs on the board will illuminate when the board is powered.

Preparing an SD Card [Common for Windows and Linux]

1-1. Prepare an SD card.

- 1-1-1. Insert an SD card into the PC's SD card slot.
- 1-1-2. **[Linux users]:** Make sure that you copy the SD card files from Linux to the C:\training directory by entering the following command:

```
[host]$ cp -rf <path_to_sd_card_files> /media/sf_training
```

- 1-1-3. Browse to the SD card drive using Windows Explorer.

Optional: You may want to erase or reformat the SD card at this time, which may prevent any unwanted interaction with other files that may be on the card.

- 1-1-4. Open a second Windows Explorer window to browse to *the location of the files you want to copy to the SD card*, which will be copied to the SD card.

Be aware that many SD card images are delivered as compressed files. You will need to ensure that these files are properly decompressed unless otherwise indicated.

- 1-1-5. Drag-and-drop all the files from the source directory to the SD card directory.

This will automatically copy the files onto the SD card.

Note: The files should go into the root of the SD card unless there are specific instructions to the contrary.

- 1-1-6. Close both Windows Explorer windows.

- 1-1-7. Remove the SD card from the PC card slot.

- 1-1-8. Turn off power to the hardware platform.

Note: The boot selection switches or jumpers must be properly set to boot from the SD card. See the appropriate section in the *Lab Setup Guide*.

- 1-1-9. Insert the SD card into its slot on the hardware platform.

Booting Linux

You must have an SD card with a properly constructed Linux boot image. This can be open-source Linux, PetaLinux, or a third-party image.

1-1. Set the jumpers/switches on your board to boot from the SD card.

If you do not recall how to perform this task, refer to the "Configuring the ZC702 Jumpers" section under SDK Operations in the *Lab Reference Guide*.

1-2. Insert an SD card with a properly constructed Linux boot image.

1-3. Apply power to the board.

1-3-1. Open a terminal window if you want to observe the Linux boot process.

This process typically takes several minutes to complete. The reason for apply power prior to setting up a terminal window is that the PC must first "see" the UART (via USB). This can only happen when the board is powered up. If you want to see all of the Linux boot-up messages, you will need to recycle power on the board.

If you do not recall how to perform this task, refer to the "Configuring the SDK Terminal for ZC702/6 and Tera Term for the Zed Board" section under SDK Operations in the *Lab Reference Guide*.

Launching and Configuring the Tera Term Terminal Program in Serial Port Mode

Maintaining consistency in the instructions is an important aspect of the lab experience. The labs will use general and vague commands such as "Open the terminal emulator". Depending on the system that you are using, you must know which terminal emulator to configure and launch. Here are the instructions for opening a terminal emulator for both the Linux and Windows environments.

Linux

gtkterm is a simple GTK+ terminal used to communicate with the serial port. While other terminal emulators may be used, the instruction provided here are specific to Tera Term.

1-1. Launch gtkterm and set the port configuration.

1-1-1. Press <Ctrl + Alt + T> launch a Linux terminal window.

- 1-1-2.** Enter the following to launch gtkterm from the terminal window:

```
sudo gtkterm
```

- 1-1-3.** When prompted for the password, **xilinx**.

Note: While the application will run as a regular user, you must be a super user to access the ports.

When the GtkTerm window opens, perform the following.

- 1-1-4.** Select **Configuration > Port** to open the Configuration dialog box.

- 1-1-5.** Identify the port associated with your board and set the port as **/dev/ttyUSBx** (where x could be 0, 1, 2, etc.)

- 1-1-6.** Set the baud rate to **115200**.

Leave the rest of the settings at their default.

Note: You can open multiple instances of /dev/USBx if you are unable to find out which port your UART is connected to.

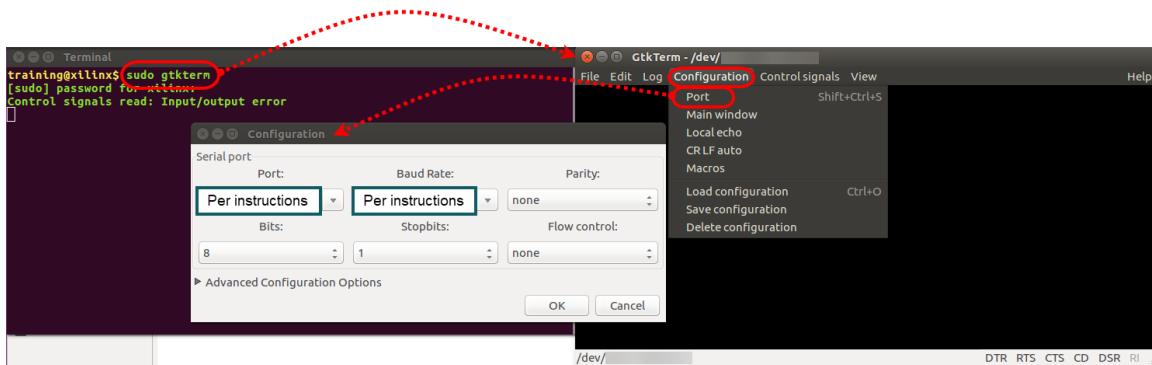


Figure 495: Opening gtkterm and Selecting the Port Configuration

- 1-1-7.** [Optional] You can save these settings so that you do not have to reconfigure GtkTerm each time you open it by clicking **Configuration > Save configuration**.

If you save the configuration as "default" this configuration will open when GtkTerm starts. Otherwise you can save the configuration by another name; however, you will then need to load the configuration each time you start GtkTerm.

- 1-1-8.** Click **OK** to save the settings and leave the terminal open.

Windows

Tera Term is a popular public domain terminal emulation program. It is capable of operating as a serial port terminal or as a telnet client. While other terminal emulators may be used, the instruction provided here are specific to Tera Term.

1-2. Launch the Tera Term terminal program.

- 1-2-1. Double-click the **Tera Term** icon on the Windows desktop to launch Tera Term.

Alternatively, you can select **Start > All Programs > Tera Term > Tera Term**.

- 1-2-2. Select **Serial** as the connection (1).

- 1-2-3. Click the **Port** drop-down list to view the available COM ports (2).

Note: If your port is not listed, exit Tera Term, power cycle your board and re-start this step.

- 1-2-4. Select the COM # discovered in the last step (3).

Note that if you cannot immediately identify the proper COM port, review "Determining the Active COM Port".

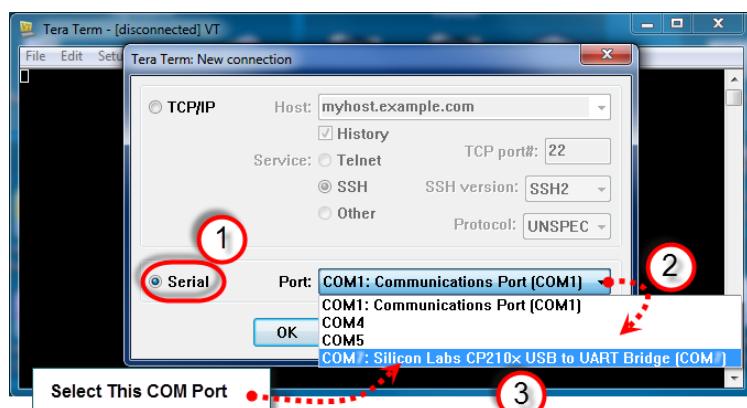
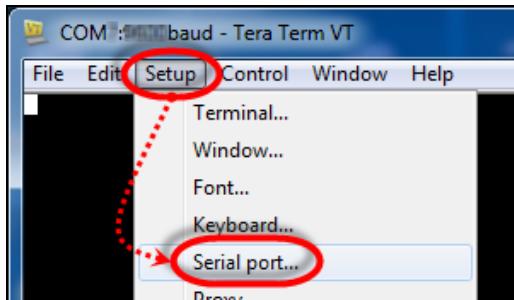


Figure 496: Selecting the COM Port

Note: The COM port setting is specific to the computer being used and may need to be different than shown. Use the COM port # that was discovered in the previous step.

- 1-2-5. Click **OK**.

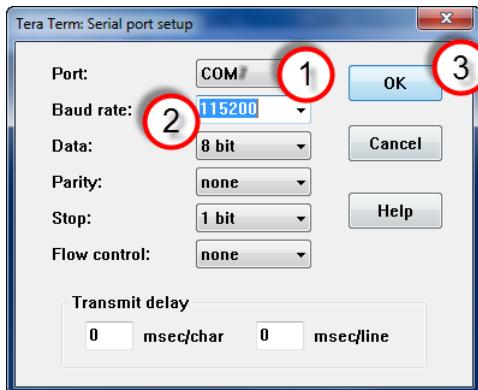
The terminal console window opens.

1-2-6. Select **Setup > Serial Port**.**Figure 497:** Opening the Tera Term Serial Port Setup Window

The Tera Term Serial Port Setup dialog box opens.

1-2-7. Confirm that the proper serial port has been selected (1).

1-2-8. Set the baud rate to **115200** (2).

**Figure 498:** Setting the Parameters for the Serial Port

Note: The COM port setting is specific to the computer being used and may need to be different than shown. Use the COM port # that was discovered in the previous step.

1-2-9. Click **OK** (3).

Tera Term is now configured to receive and transmit serial information to/from the evaluation board.

Connecting the ZC702 Board

Set up and connect the ZC702 board, or verify that this has properly been done before turning on the power. All the connections are made to the board itself.

1-1. Connect the board to your machine as shown below. Note the location and functions of the various connectors and buttons.

1-1-1. Make sure that the following cables are always present:

- Digilent USB port cable: Platform download cable function
 - U23 - Type A male to Micro B (PC USB A → ZC702 USB Micro B)
 - Connector on U23 module (highlighted in the illustration below, but not connected)
 - **Note:** The Platform Cable module may be used (shown in the figure for reference); however, the preferred method is using the USB port
- USB UART port: Provides for COMx communication
 - J17 – Type A male to Mini B (PC USB A → ZC702 USB Mini B (UART port) - J17)
- Most embedded Linux labs require an Ethernet (RJ45) cable
 - PC USB A → ZC702 USB Mini B (UART port) - J17
- Power supply

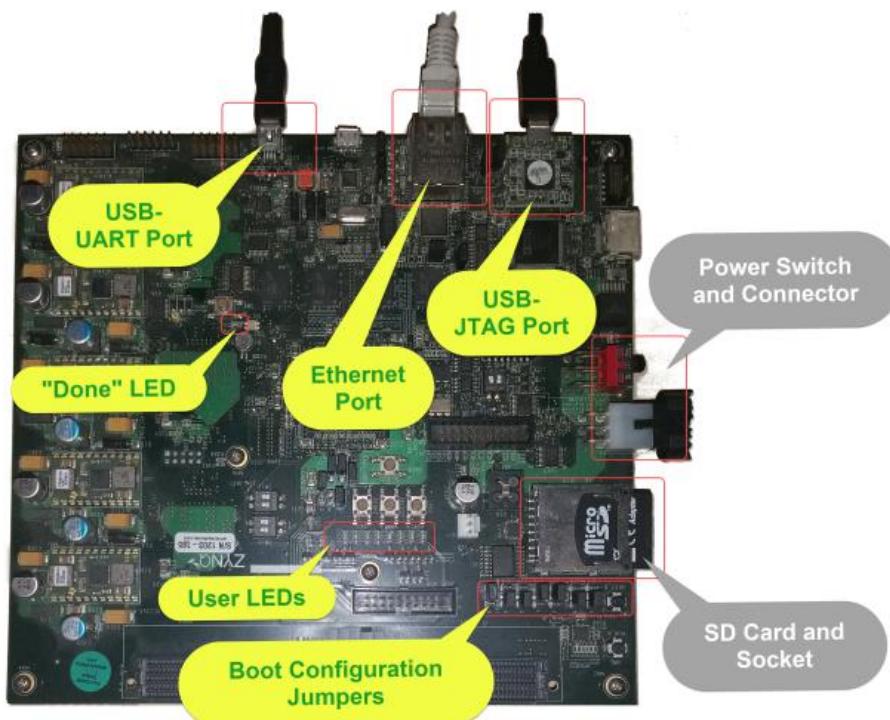


Figure 499: ZC702 Development Board

1-1-2. Power up the ZC702 board using the slide switch.

Connecting the ZedBoard

Set up and connect the ZedBoard, or verify that this has properly been done before turning on the power. All the connections are made to the board itself.

1-1. Connect the board to your machine as shown below.

1-1-1. Make sure that the following cables are always present:

- Digilent USB port cable: Platform download cable function
 - Type A male to Micro B (PC USB A → ZedBoard USB Micro B (JTAG port) - J17)
 - **Note:** Typically the Platform Cable module is not used; however, it is supported.
- USB UART port: Provides for COMx communication
 - J14 – Type A male to Micro B (PC USB A → ZedBoard USB Micro B (UART port) - J14)
- Most Linux labs require an Ethernet cable
 - PC RJ45 → ZedBoard RJ45 (straight-through cable) - J11
- Power supply

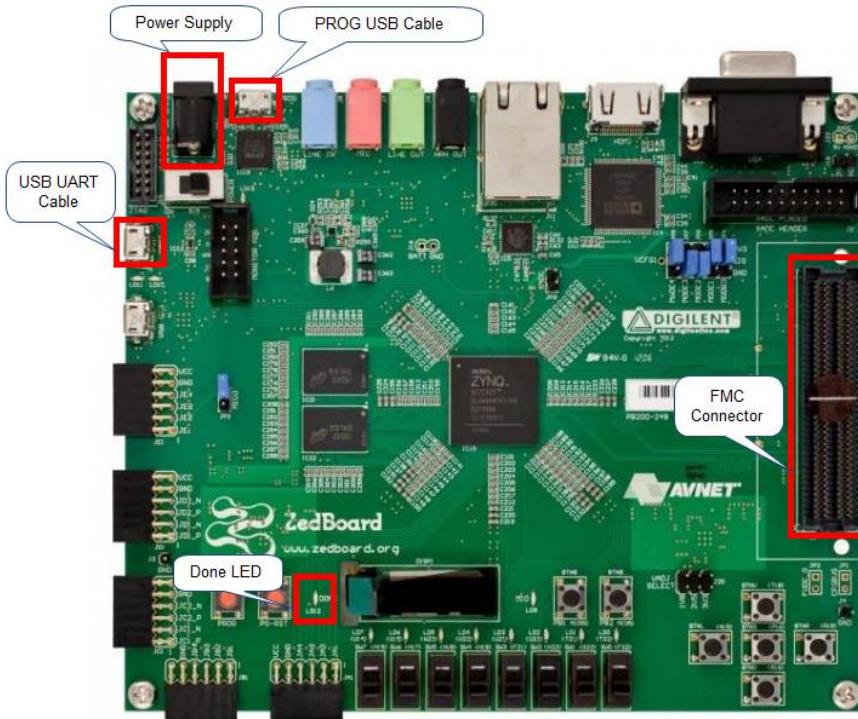


Figure 500: ZedBoard

1-1-2. Power up the ZedBoard using the mounted slide switch.

Connecting the ZedBoard without the FMC-CE Card

Set up and connect the ZedBoard, or verify that this has properly been done before turning on the power. All the connections are made to the board itself.

1-1. Connect the board to your machine as shown below.

1-1-1. Make sure that the following cables are always present:

- Digilent USB port cable: Platform download cable function
 - Type A male to Micro B
 - **Note:** Typically the Platform Cable module is not used; however, it is supported.
 - J17
- USB UART port: Provides for COMx communication
 - J14 – Type A male to Micro B
- Power supply

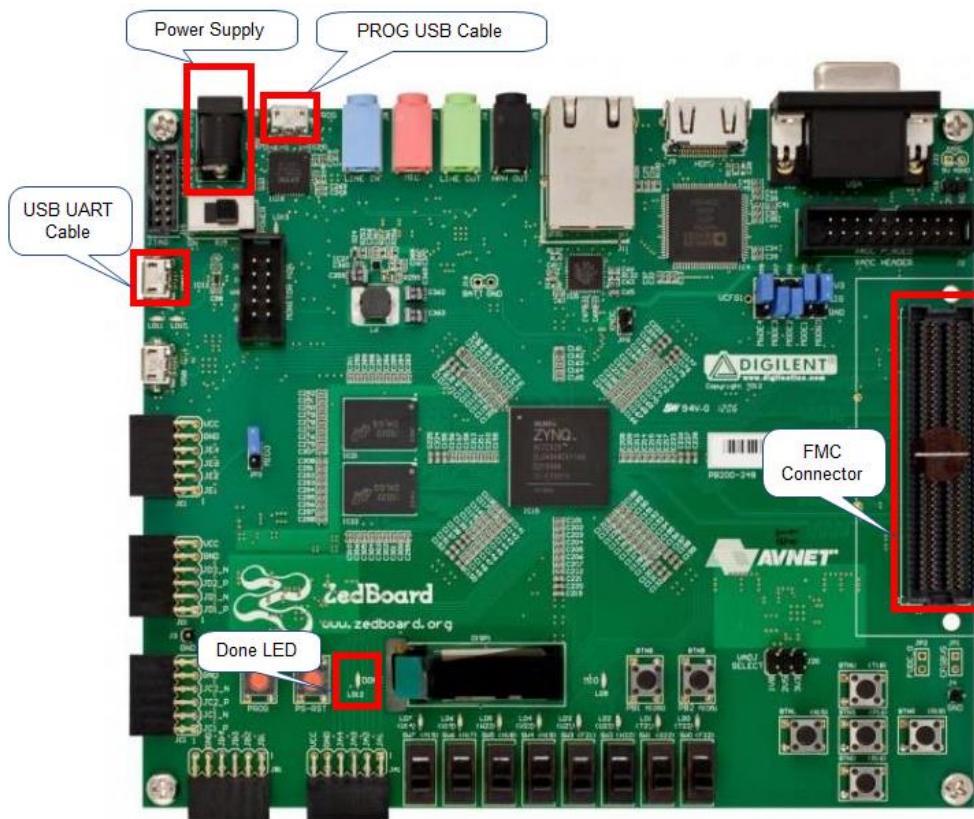


Figure 501: ZedBoard

1-1-2. Power up the ZedBoard using the mounted slide switch.

Setting the Jumpers on the ZC702 Board

The jumpers connected to the Zynq device are read by the Stage 0 bootloader and are used to determine where the Zynq PS boots from.

1-1. Set the jumpers on the ZC702 board based on your booting requirements.

- **Boot from SD card**
- **JTAG communications with PC (no boot)**
- **Boot from QSPI**

The three following topics cover these cases.

Jumper/Switch Settings ZC702 - Boot from SD Card

SD Card Boot:

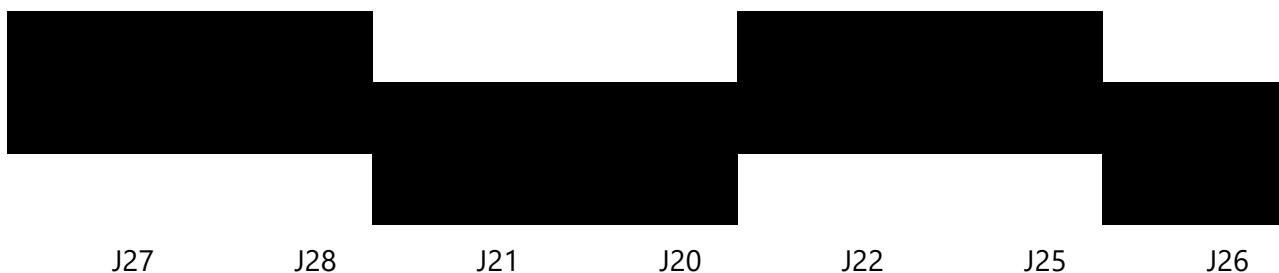


Figure 502: SD Card Boot Settings (ZC702 Board)

Jumper/Switch Settings ZC702 - Boot from JTAG

JTAG Mode:



J27 J28 J21 J20 J22 J25 J26

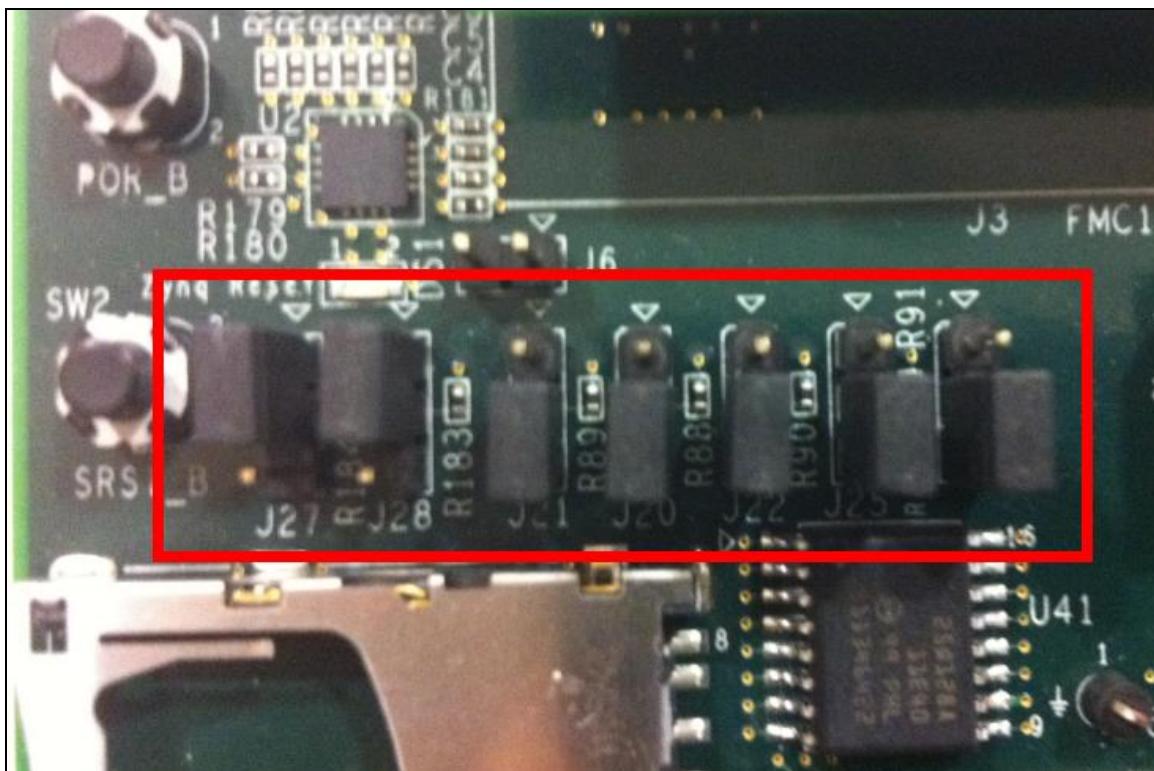


Figure 503: JTAG Boot Settings (ZC702 Board)

It is worth noting that if the SD card settings are used, but no SD card is present in the socket, then the boot mode will fall back to JTAG.

The ZC702 board has two JTAG cable options (most ATPs use option 1):

1. Digilent on-board platform cable USB interface using USB A-mini B cable: Set SW10 (i.e. JTAG Select dip switches) on Zynq board to "01"
2. Xilinx Platform USB: Set SW10 (i.e. JTAG Select dip switches) on Zynq board to "10".

Jumper/Switch Settings ZC702 - Boot from QSPI

Flash Boot:

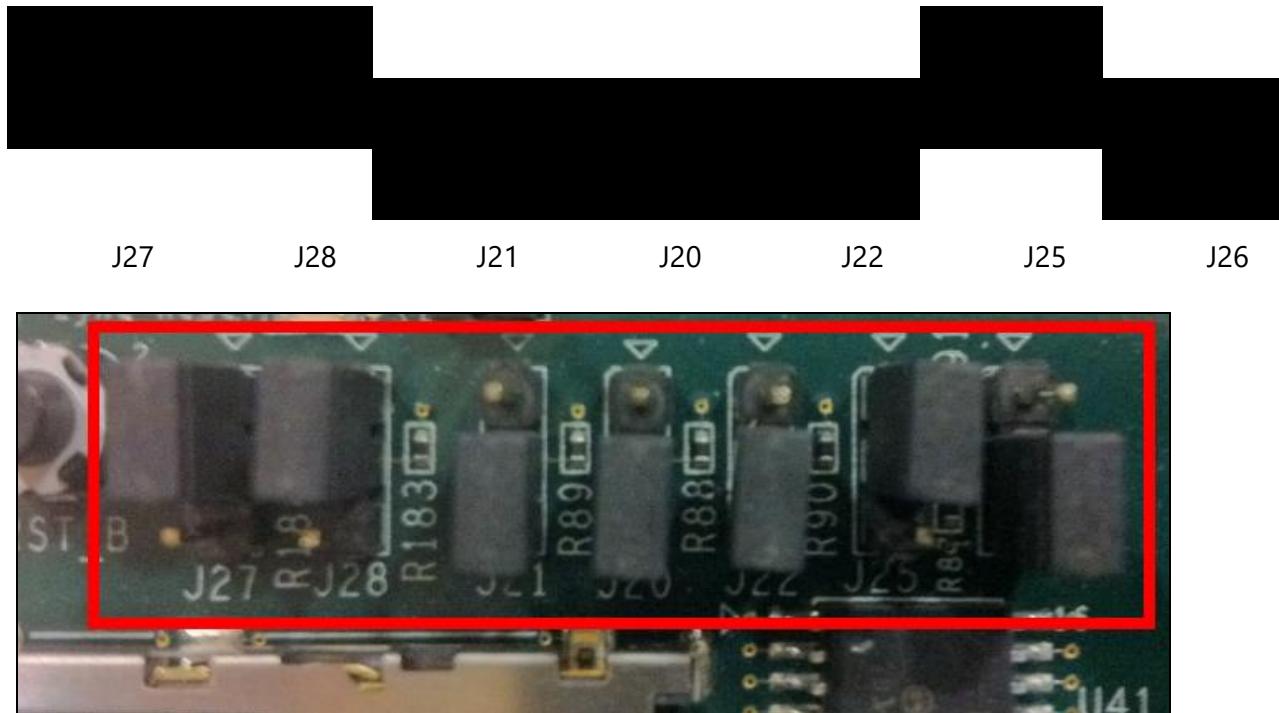


Figure 504: Flash Boot Settings (ZC702 Board)

Setting the Jumpers on the ZC706 Board

1-1. Set the jumpers on the ZC706 board based on your booting requirements.

- **Boot from SD card**
- **JTAG communications with PC (no boot)**
- **Boot from QSPI**

The three following topics cover these cases.

Jumper/Switch Settings ZC706 - Boot from SD Card

SD Card Boot:

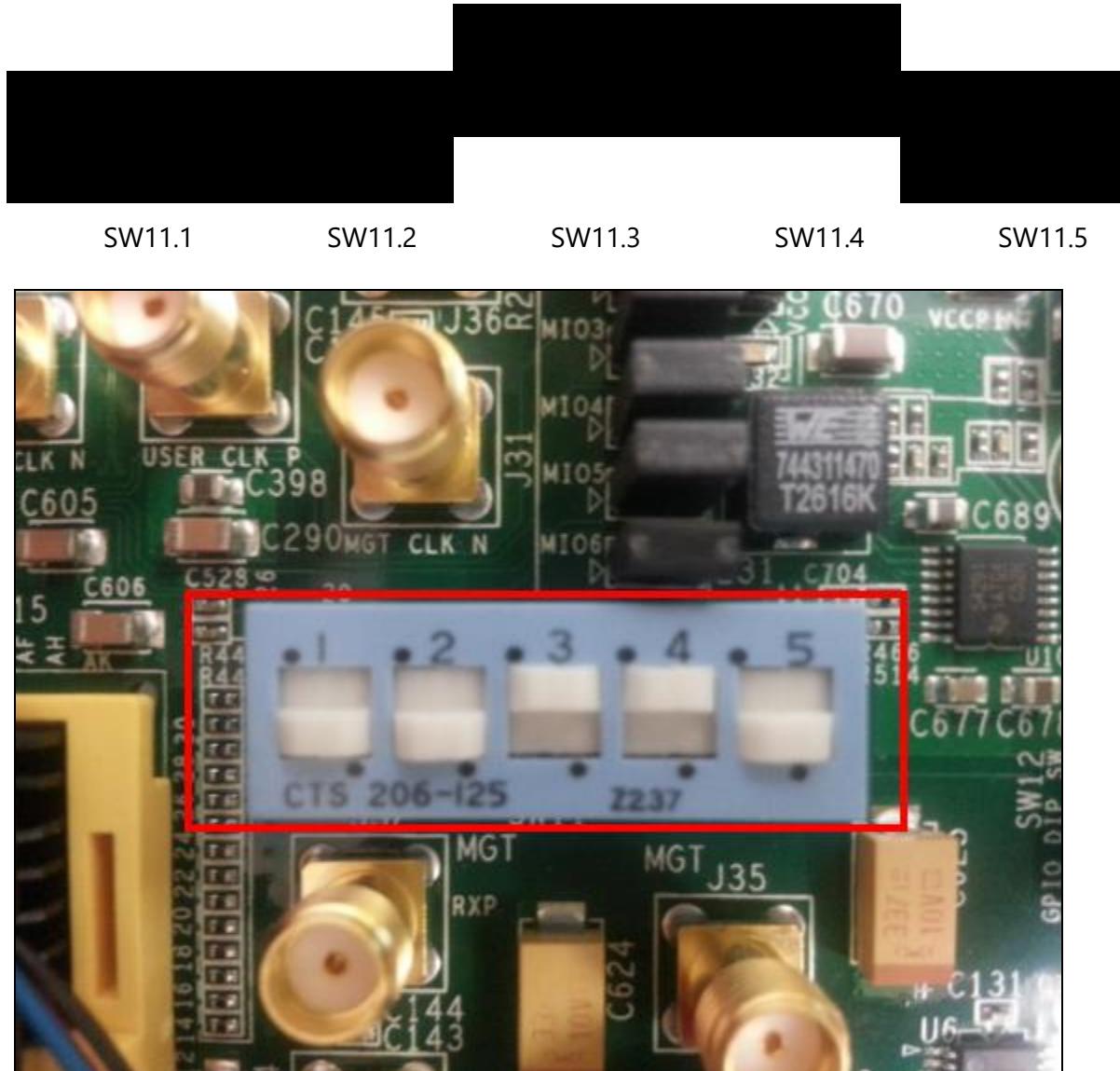
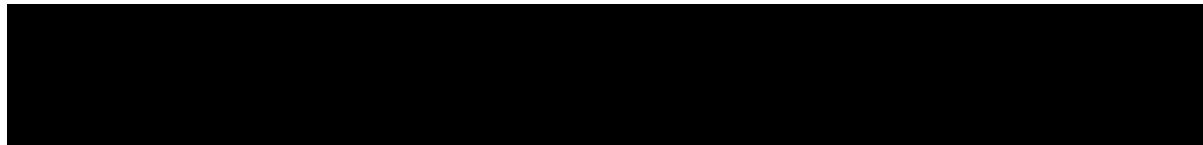


Figure 505: SD Card Boot Settings (ZC706 Board)

Jumper/Switch Settings ZC706 - Boot from JTAG

JTAG Mode:



SW11.1

SW11.2

SW11.3

SW11.4

SW11.5

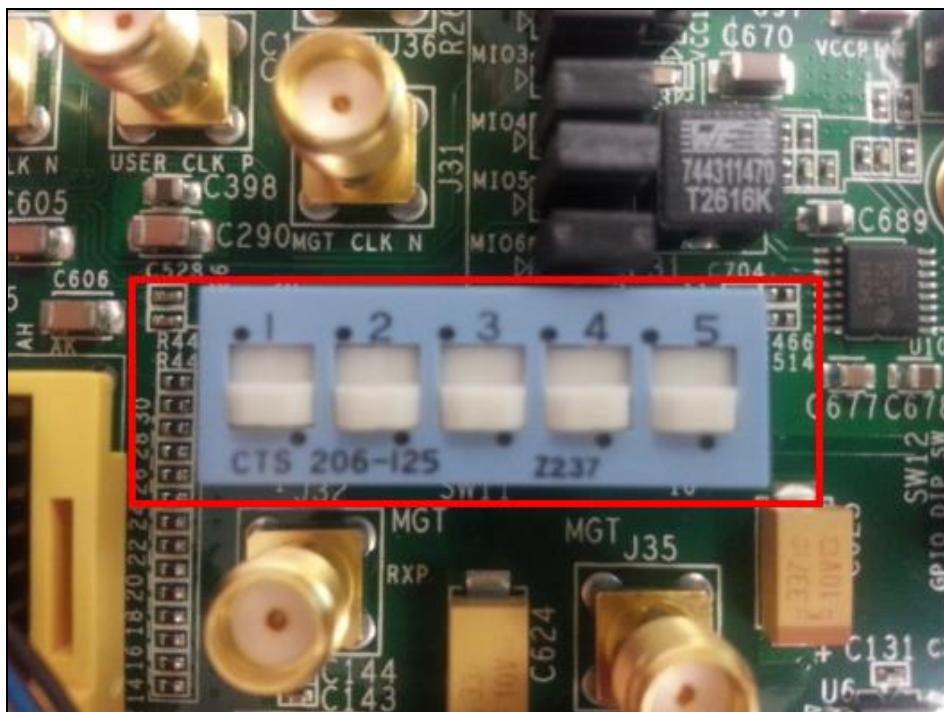


Figure 506: JTAG Boot Settings (ZC706 Board)

It is worth noting that if the SD card settings are used, but no SD card is present in the socket, then the boot mode will fall back to JTAG.

The ZC706 board has a Digilent interface option and you can use USB A-mini B cable.

To use this cable instead of Xilinx Platform USB make sure that jumper settings is in JTAG mode as mentioned above.

- Set SW4 (i.e. JTAG Select dip switches) on Zynq board to "01" which is to select Digilent interface. Whereas to select Xilinx Platform USB SW4 should be "10".

Jumper/Switch Settings ZC706 - Boot from QSPI

Flash Boot:

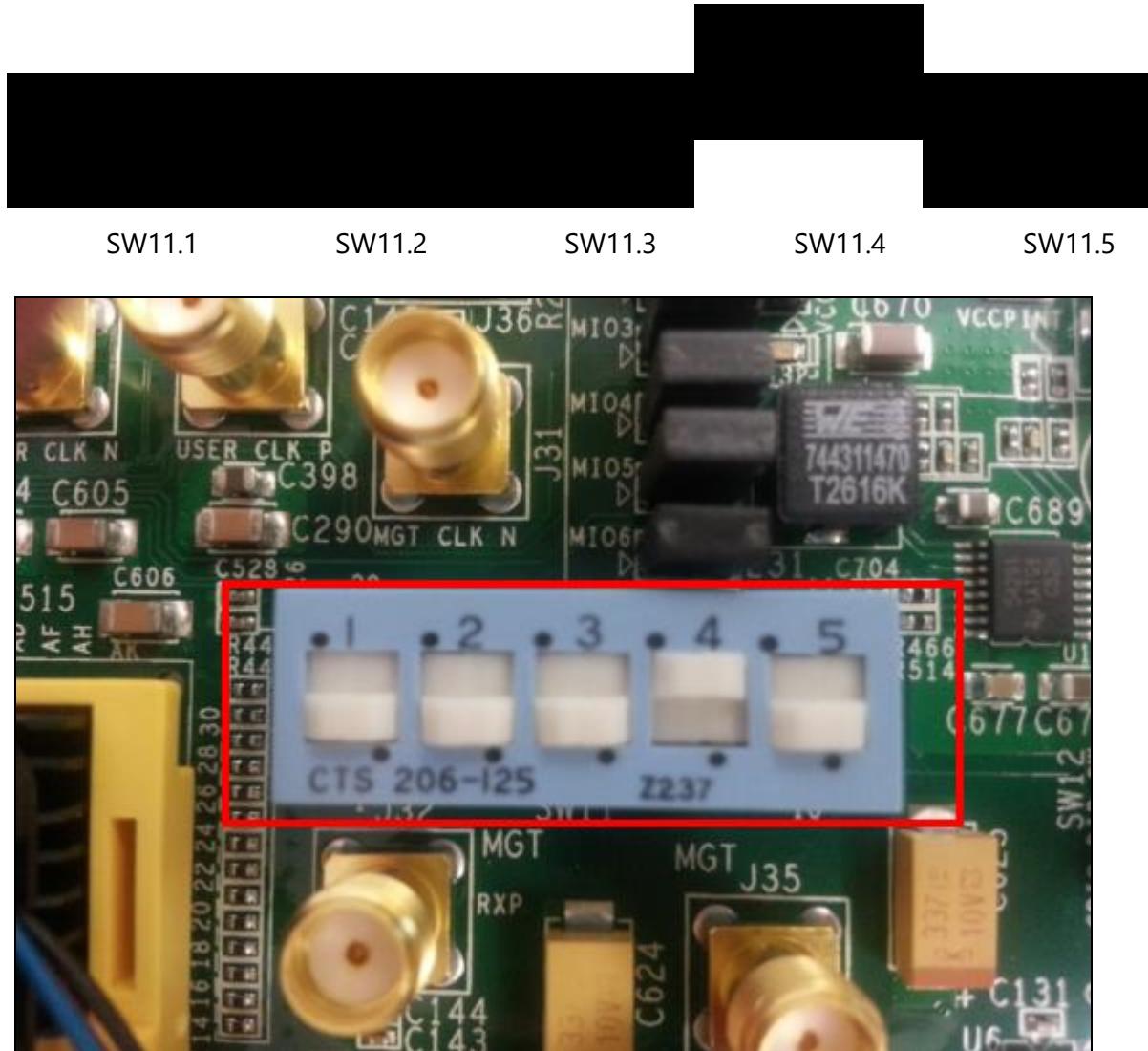


Figure 507: Flash Boot Settings (ZC706 Board)

Setting the Jumpers on the ZCU102 Board

1-1. Set the jumpers on the ZCU102 board based on your booting requirements.

- Boot from SD card
- JTAG communications with PC (no boot)

The following topics cover these cases.

Jumper/Switch Settings ZCU102 - Boot from SD Card

SD Card Mode:

Up				
Down				
	SW6.1	SW6.2	SW6.3	SW6.4



Figure 508: SD Card Boot Settings (ZCU102 Board)

Jumper-Switch Settings ZCU102 - Boot from JTAG

JTAG Mode:

Up	SW6.1	SW6.2	SW6.3	SW6.4
Down				

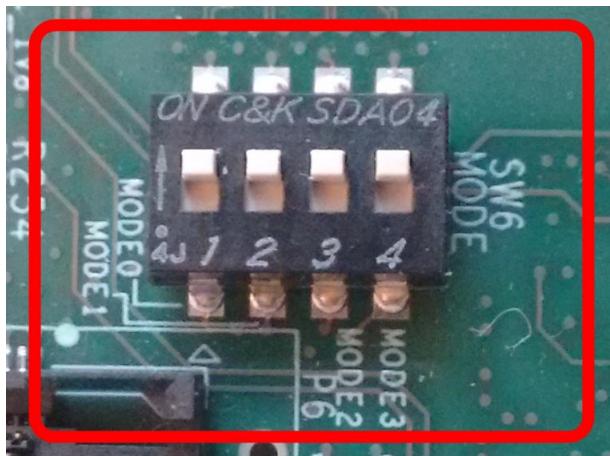


Figure 509: JTAG Boot Settings (ZCU102 Board)

Setting the Jumpers on the ZedBoard

1-1. Set the jumpers on the ZedBoard based on your booting requirements.

- **Boot from SD card**
- **JTAG communications with PC (no boot)**
- **Boot from QSPI**

Jumper Settings ZedBoard - Boot from SD Card

SD Card Boot:

JP8/JP11 - Mode 4	JP7/JP10 - Mode 3	JP6/JP9 - Mode 2	JP5/JP8 - Mode 1	JP4/JP7 - Mode 0	
					ON/3.3V
					Off/Gnd

Jumper - JP6 (shortened)

VADJ SELECT - 3V3

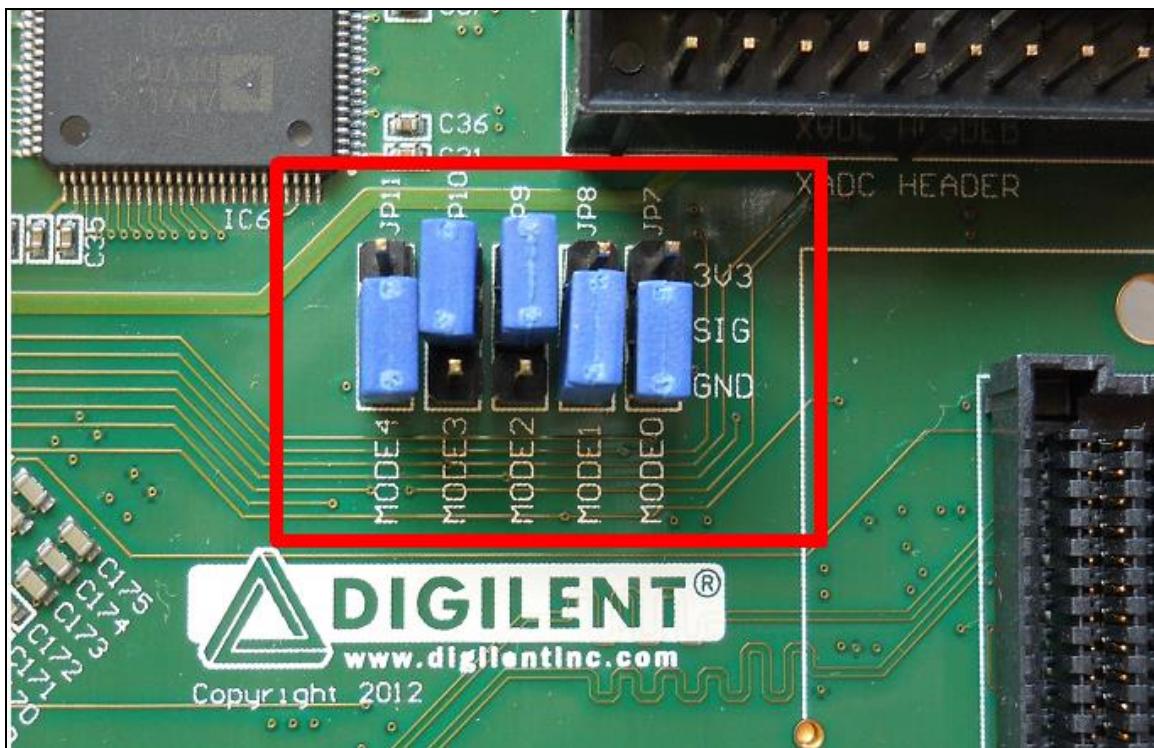


Figure 510: SD Card Boot Settings (ZedBoard)

Setting Up the Hardware - KC705

Set up and connect the hardware evaluation board, or verify that this has properly been done before turning on the power.

1-1. Connect the board to your machine as shown below.

- 1-1-1. Connect a USB cable from a USB port on your computer to the **USB UART** connector on the evaluation board.
- 1-1-2. Similarly, connect the USB cable to the **Digilent USB JTAG** interface.
- 1-1-3. Ensure that the power cord is plugged in and turn on the evaluation board.
- 1-1-4. Make sure that the board settings are proper.
- 1-1-5. Ensure that all DIP switches (SW11) are in the off position.

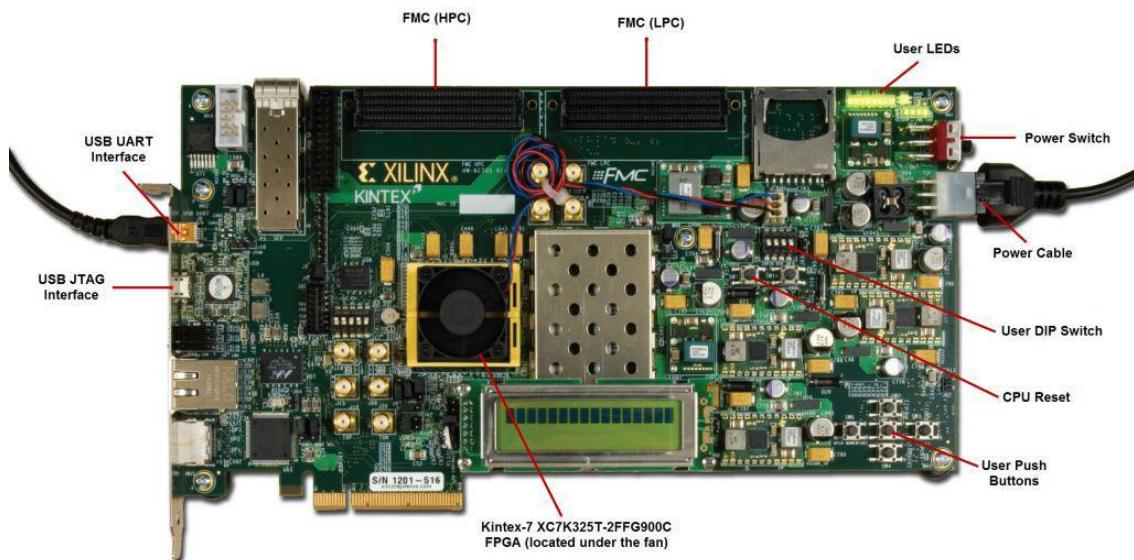


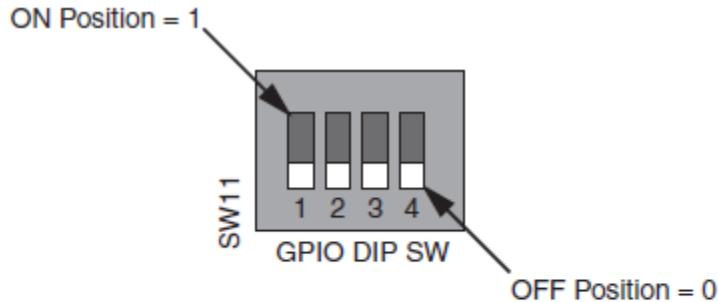
Figure 511: KC705 Evaluation Board

You may be prompted to install drivers when the board is first connected. Do not allow the driver installation to search the Web, but allow it to search for the drivers on your computer.

Default Switch Settings

1-1. Default DIP switch for SW11 user GPIO settings.

- 1-1-1. Set all the switch positions to the default settings.



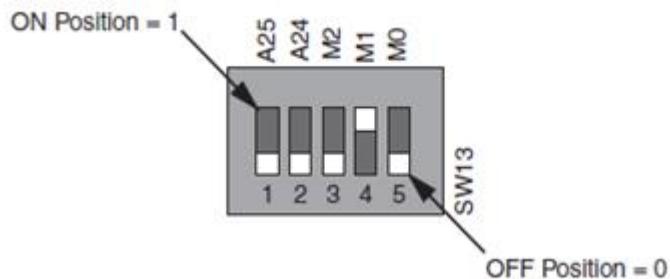
SW11 Default Switch Settings

Position	Function	Default
1	GPIO_DIP_SW3	Off
2	GPIO_DIP_SW2	Off
3	GPIO_DIP_SW1	Off
4	GPIO_DIP_SW0	Off

Figure 512: KC705: SW11 Default Switch Settings

1-2. Default DIP switch SW 13 mode and Flash address settings.

- 1-2-1. Set the switch positions to the default settings.



SW13 Default Switch Settings

Position	Function		Default
1	FLASH_A25	A25	Off
2	FLASH_A24	A24	Off
3	FPGA_M2	M0	Off
4	FPGA_M1	M1	On
5	FPGA_M0	M3	Off

Figure 513: KC705: SW13 Default Switch Settings

Setting up the Hardware - KCU105

Set up and connect the hardware evaluation board, or verify that this has been done properly before turning on the power.

1-1. Connect the board to your machine as shown below.

- 1-1-1. Connect a USB cable from a USB port on your computer to the USB UART connector on the evaluation board.
- 1-1-2. Similarly, connect the USB cable to the USB JTAG connector on the evaluation board.
- 1-1-3. Ensure that the power cord is plugged in and turn on the evaluation board.
- 1-1-4. Make sure that the board settings are correct.

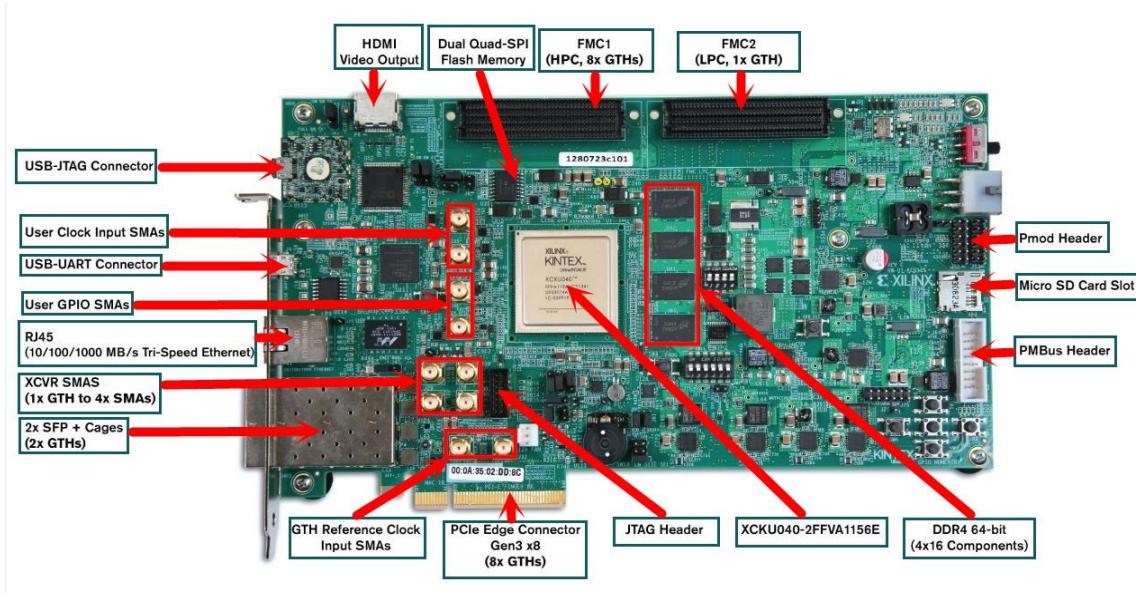


Figure 514: KCU105 Evaluation Board