

Lab Reference Guide

lab-reference-guide-2016.1-wkb-rev1-prelim

Lab Reference Guide 2016.1



© Copyright 2016 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
Vivado Design Suite Operations	4
Vivado Analyzer Operations.....	135
Vivado HLS Operations	146
SDK Tool Operations	165
QEMU Operations	266
SDSoC Tool Operations.....	275
PetaLinux Operations.....	309
Board, OS, COM, and IP Address Tasks.....	315

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 2016.1 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
- SDK Tool Operations
- QEMU Operations
- SDSoc Tool Operations
- PetaLinux Operations
- Board, OS, COM, and IP Address Tasks

Each section may contain sub-sections.

Vivado Design Suite Operations

In This Section

Creating and Opening Operations	4
Embedded Operations	24
Vivado IP Integrator Operations.....	33
Vivado Elaborated Design Operations.....	79
Vivado Simulation Operations	82
Vivado Synthesis Operations	84
Vivado Implementation Operations.....	92
Vivado Tcl Operations	102
Vivado Timing Miscellaneous Operations	110
Vivado Hardware Session Operations.....	116
Vivado Add Sources Operations	126

Creating and Opening Operations

In This Section

Launching the Vivado Design Suite	5
Launching the Vivado Design Suite Tcl Shell	6
Creating a New Vivado Design Suite Project with Sources.....	6
Creating a Blank Vivado Design Suite Project.....	14
Creating an HDL Source File	17
Opening a Vivado Design Suite Project	19
Opening a File in the Editor	20
Importing IP	21
Closing the Vivado Design Suite Project.....	23
Closing the Vivado Design Suite	23

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. Launch the Vivado Design Suite.

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

- 1-1-1. Select **Start > All Programs > Xilinx Design Tools > Vivado 2016.1 > Vivado 2016.1.**

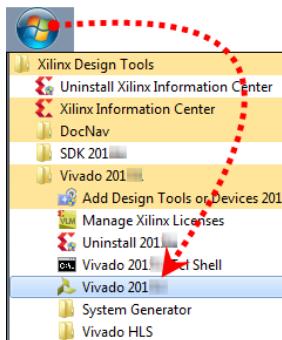


Figure 1: Launching the Vivado Design Suite from the Start Menu

-- OR --

Double-click the **Vivado Design Suite** shortcut icon (

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.

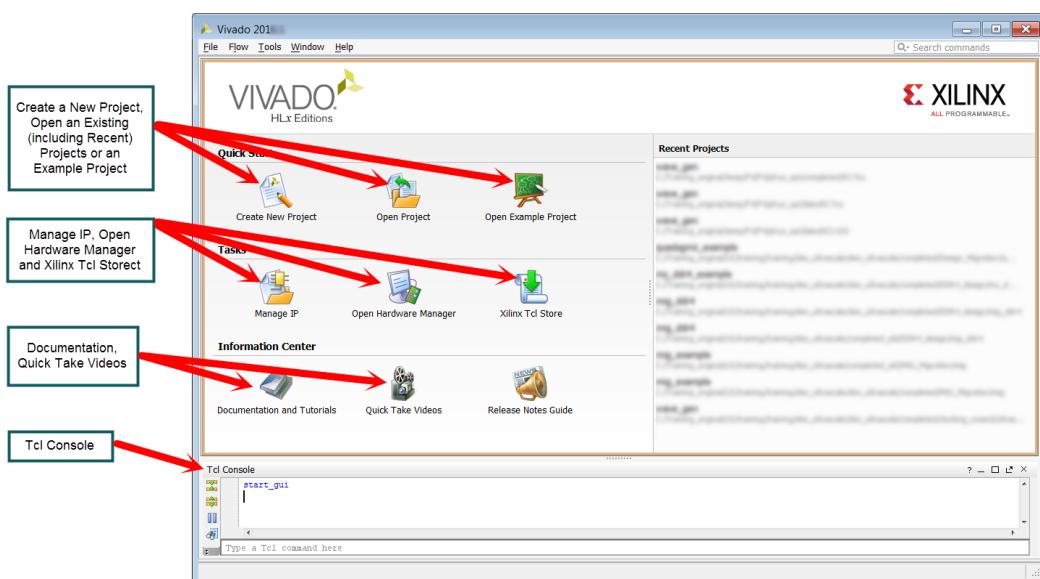


Figure 2: Vivado Design Suite Welcome Screen

Launching the Vivado Design Suite Tcl Shell

1-1. Launch the Vivado Design Suite Tcl shell.

- 1-1-1.** Select **Start > All Programs > Xilinx Design Tools > Vivado 2016.1 > Vivado 2016.1 Tcl Shell.**

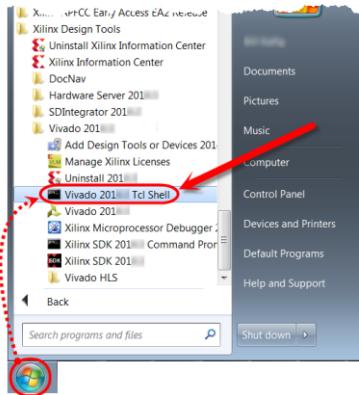


Figure 3: Vivado Design Suite Tcl Shell

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).

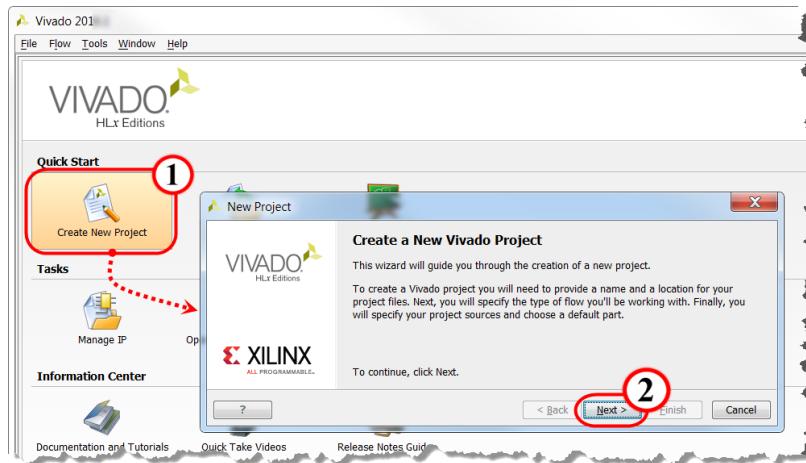


Figure 4: Creating a New Vivado Design Suite Project

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****\labs\lab name\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.

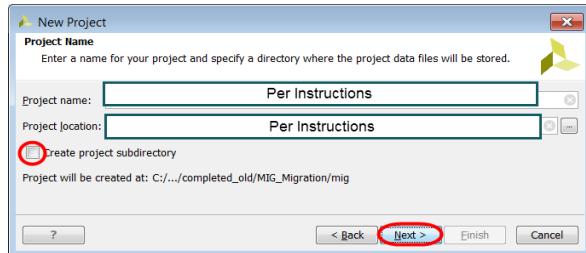


Figure 5: Entering the Project Name and Location

- 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).

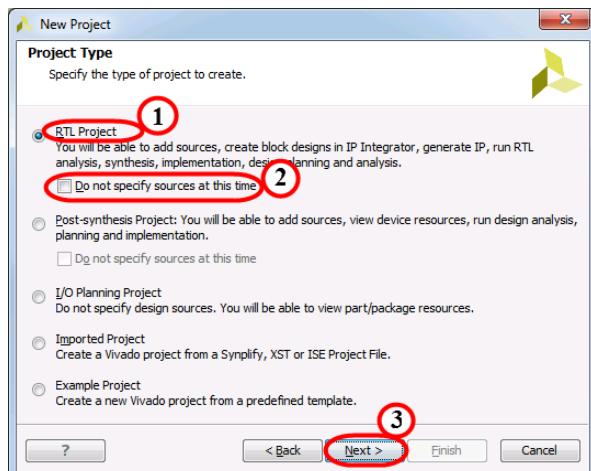


Figure 6: Setting the Project Type to RTL

- 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 green **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.

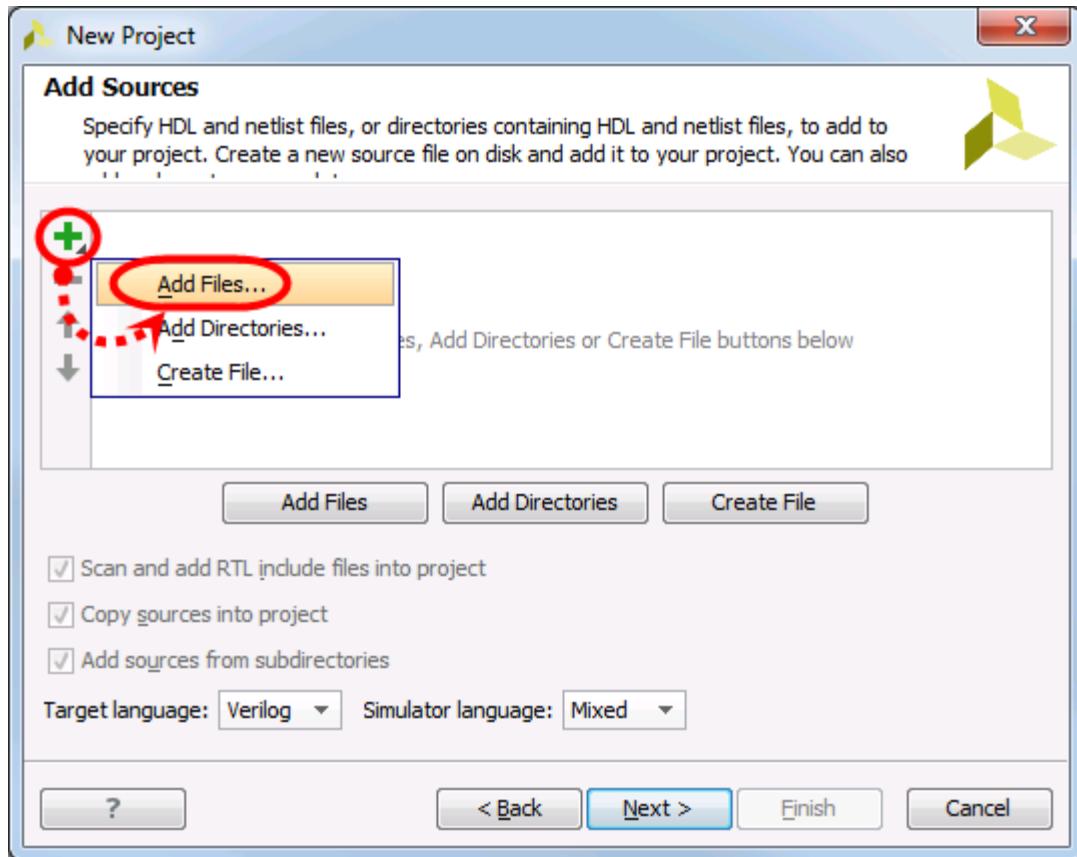


Figure 7: Adding Sources to the Project

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****\labs\lab name\your board\<language> or your project name** directory (1).

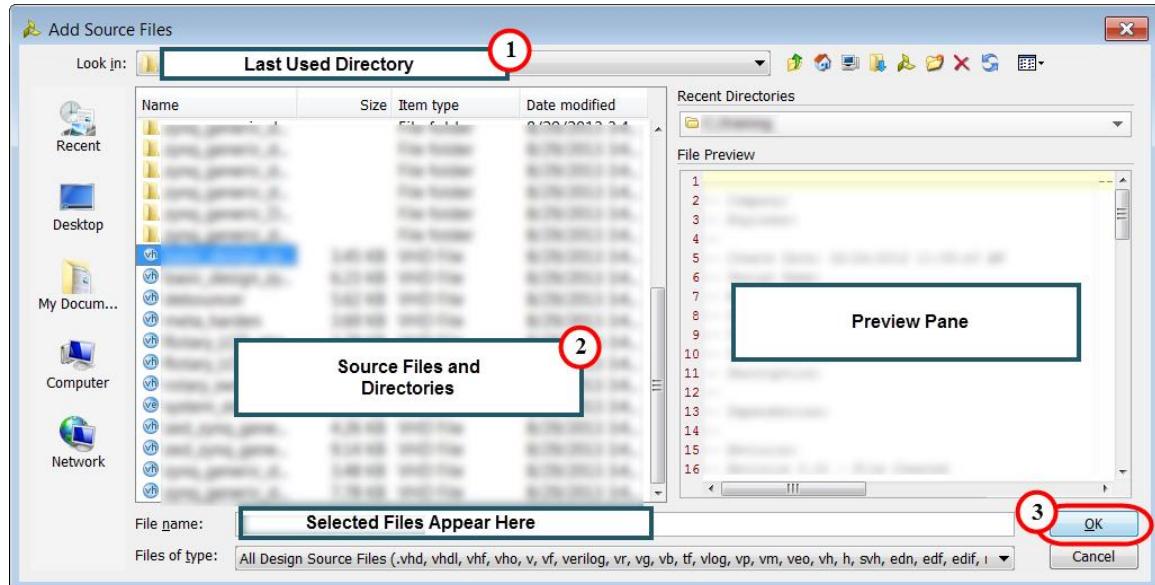


Figure 8: Adding Source Files

- 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.

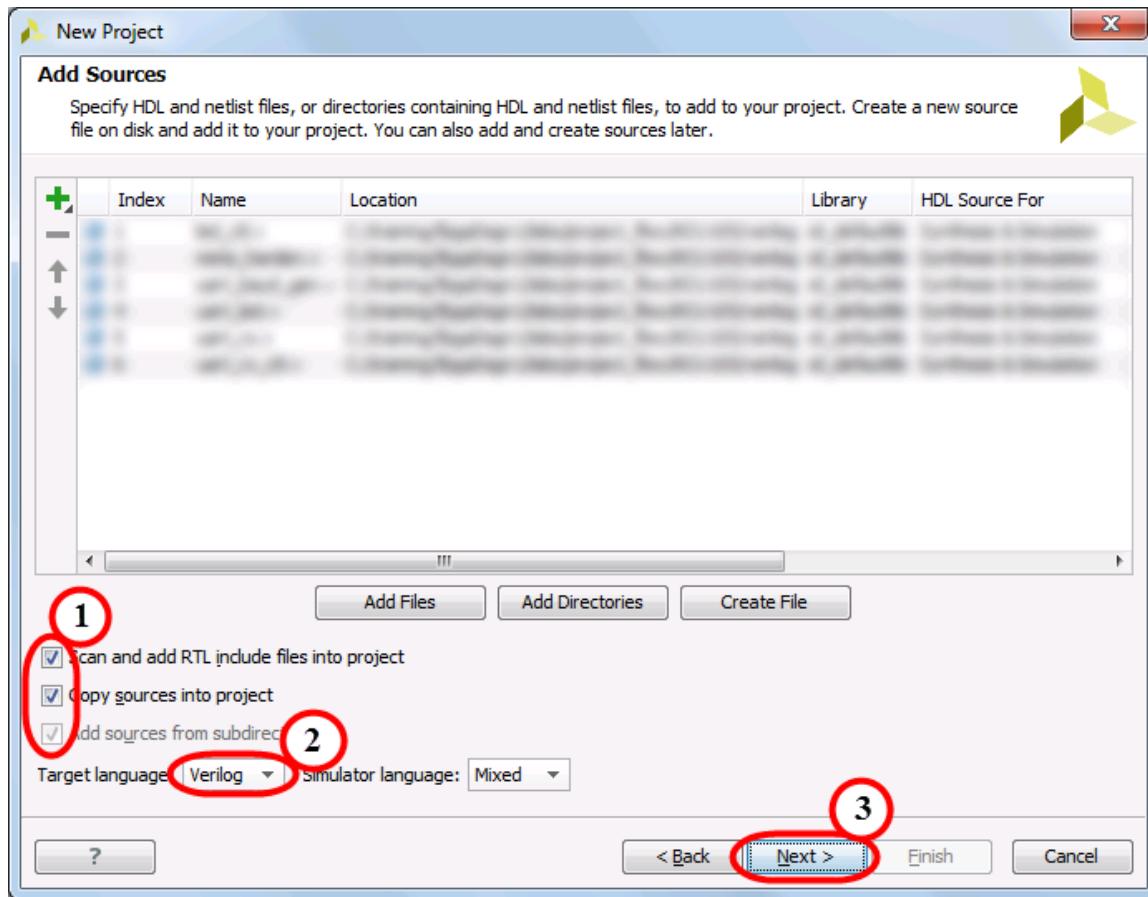


Figure 9: Setting Options in the Add Sources Window

- 1-3-6.** 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.

- 1-3-7.** 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.

- 1-3-8.** 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.

- 1-3-9.** Click **Next** to complete the adding of RTL sources and advance to adding any IP repositories (3).

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

- 1-4-2. 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.

- 1-4-3. Browse to the **C:\training****\labs\lab name\your board\<language> or your project name** directory (1).

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

- 1-4-5. 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.

- 1-4-6. Confirm that the **Copy Sources into Project** option is selected (2).

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

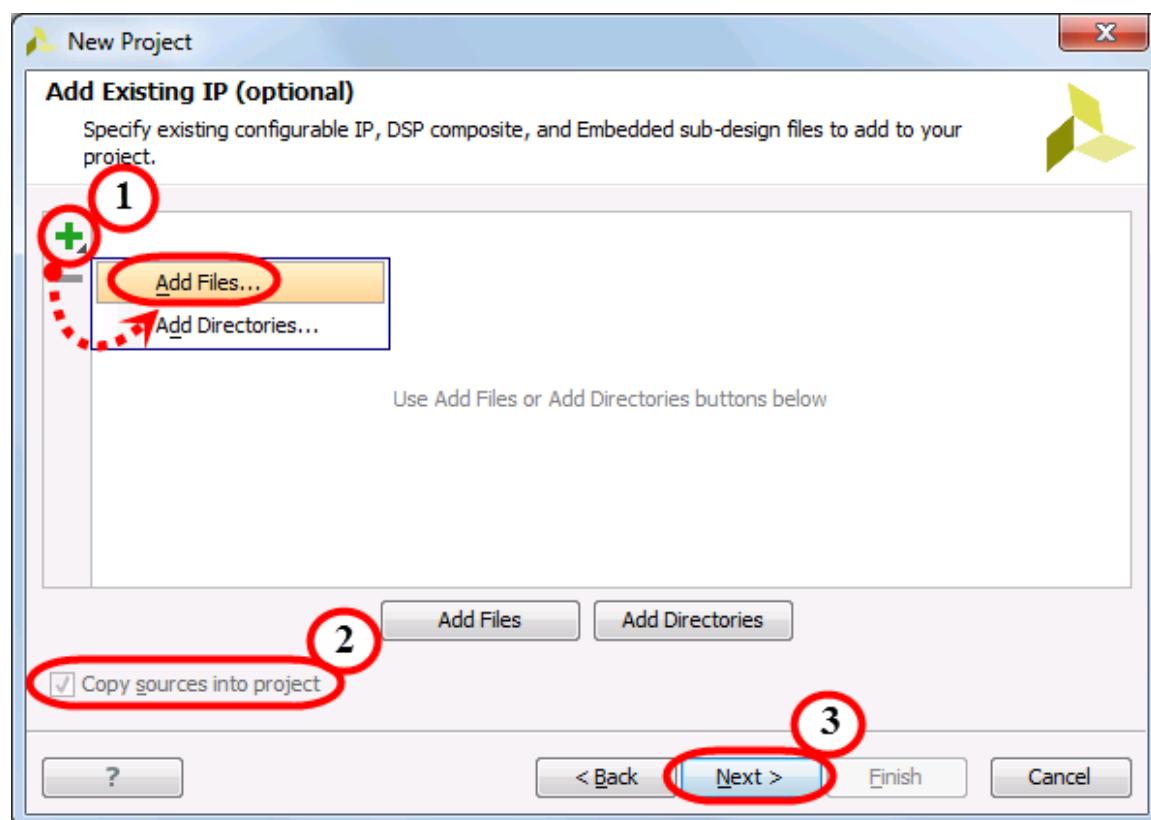


Figure 10: Adding Existing IP

- 1-4-7. Click **Next** (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 green **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****\labs\lab name\your board\<language> or your project name** directory.

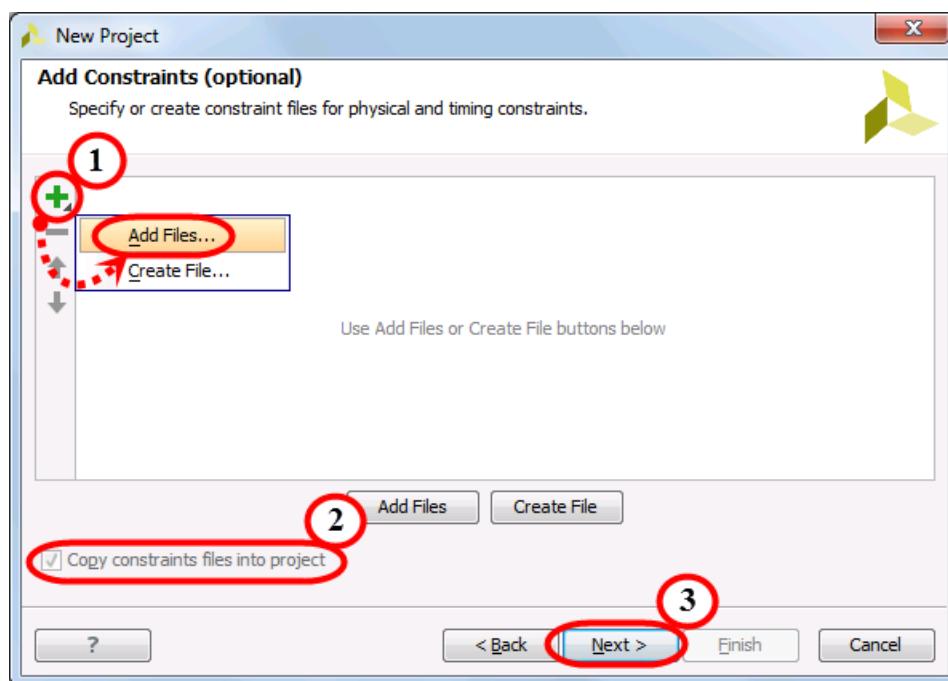


Figure 11: Adding Constraints

- 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.

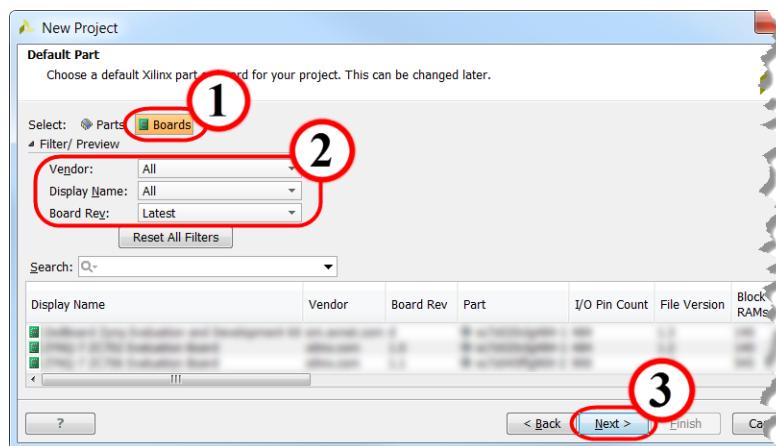


Figure 12: Selecting the Board for the Project

- 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 New Project" is the starting point for all designs. Projects contains 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 New Project** to begin the process (1).

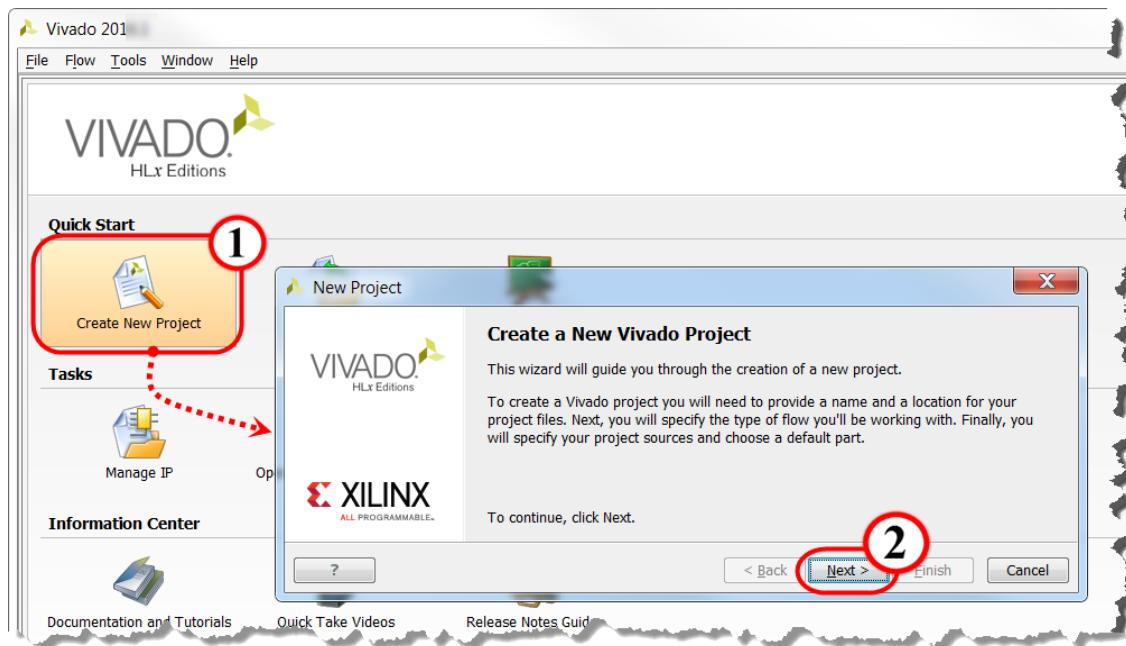


Figure 13: Creating a New Vivado Design Suite Project

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.
- 1-2-2.** Enter **C:\training****\labs\lab name\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.** Deselect the **Create Project Subdirectory** option as leaving this checked will create an unnecessary level of hierarchy for this lab.

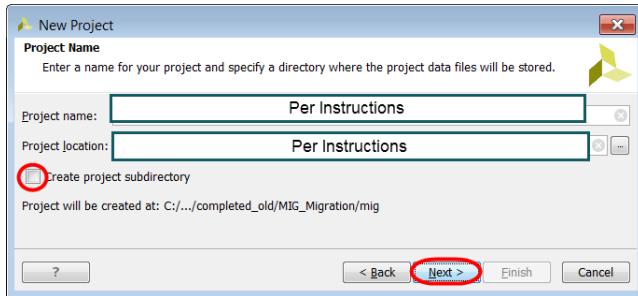


Figure 14: Entering the Project Name and Location

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

Here you will 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 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.

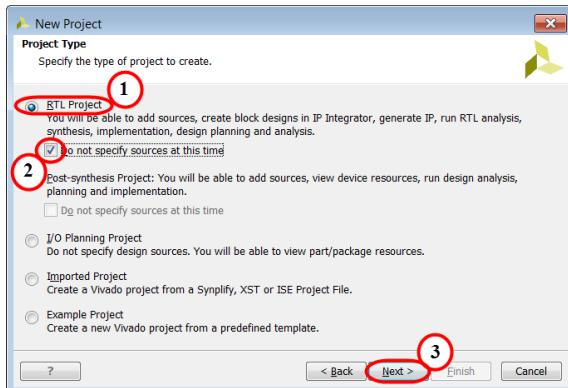


Figure 15: Selecting Project Type

- 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.

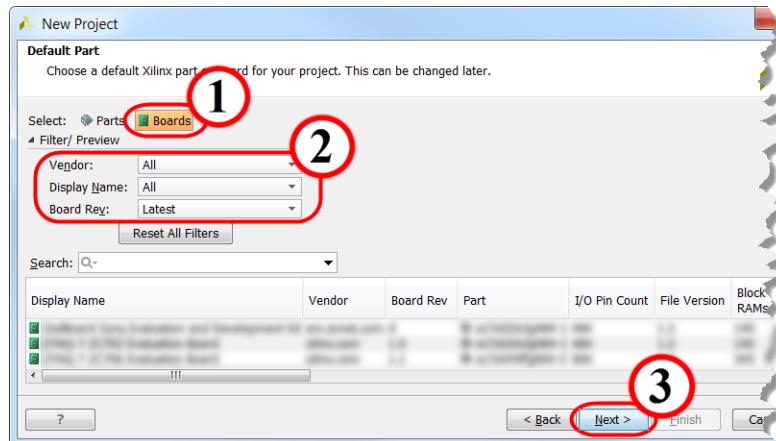


Figure 16: Selecting the Board for the Project

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.

Creating an HDL Source File

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

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

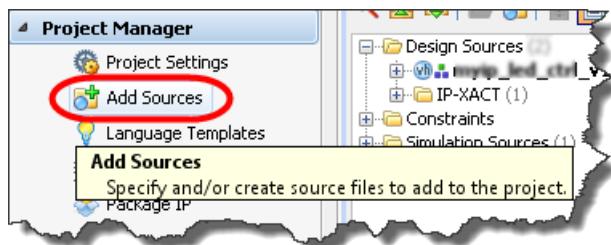


Figure 17: Selecting Add Sources

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



Figure 18: 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**.

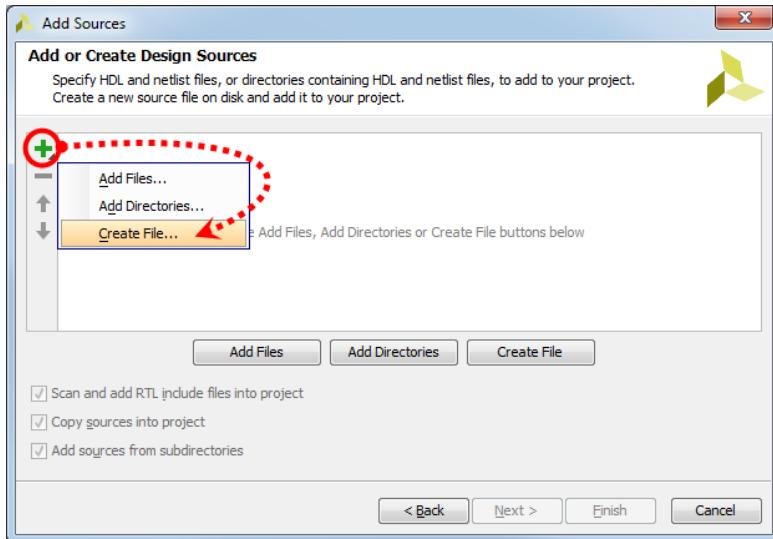


Figure 19: Selecting Create File

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.

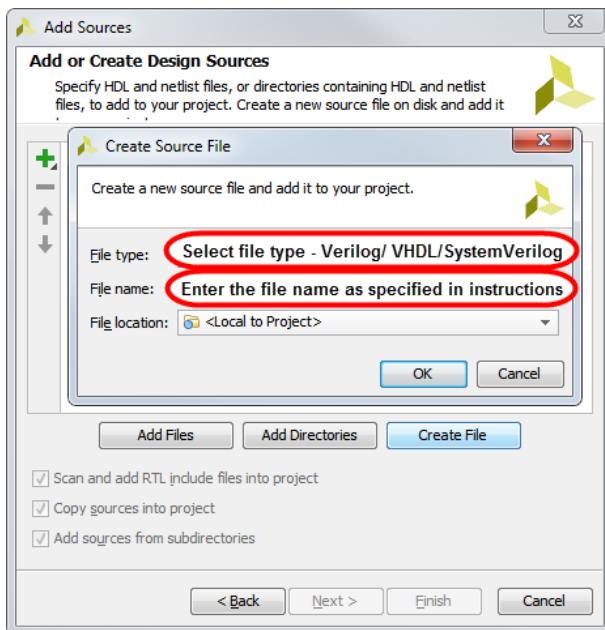


Figure 20: Entering File Name and Type

- 1-1-7.** Click **OK** in the Create Source File dialog box.
1-1-8. Click **Finish**.

The Define Module dialog box opens.

Opening a Vivado Design Suite Project

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

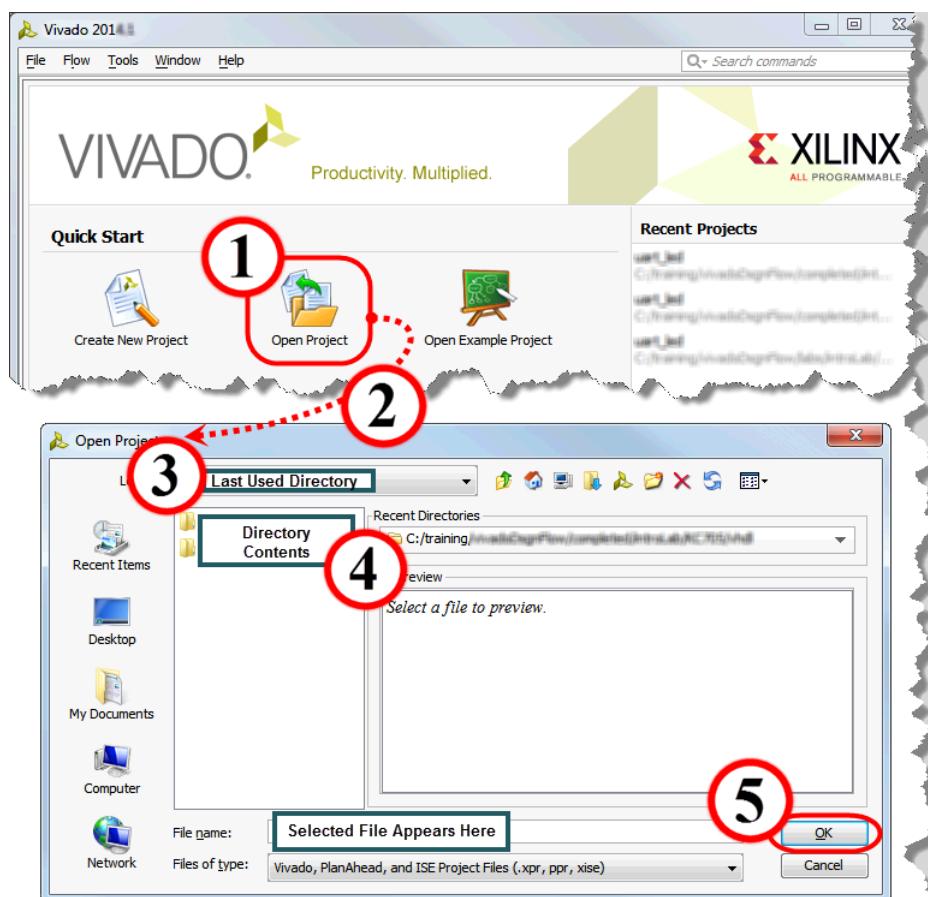
- 1-1-1. Click **Open Project** from the Quick Start section (1).

The Open Project dialog box opens (2).

- 1-1-2. Browse to the *C:\training****\labs\lab name\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).



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

The project now opens in the Vivado Design Suite.

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 > Open File**.

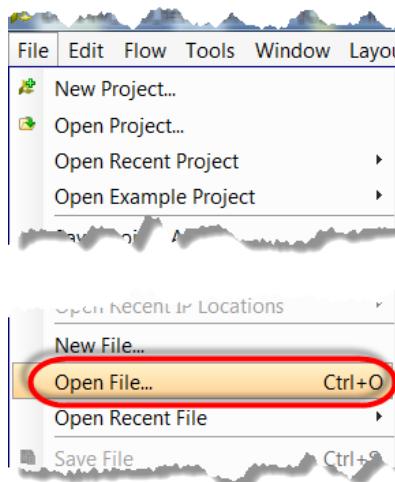


Figure 21: Opening a Text File from the File Menu

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).

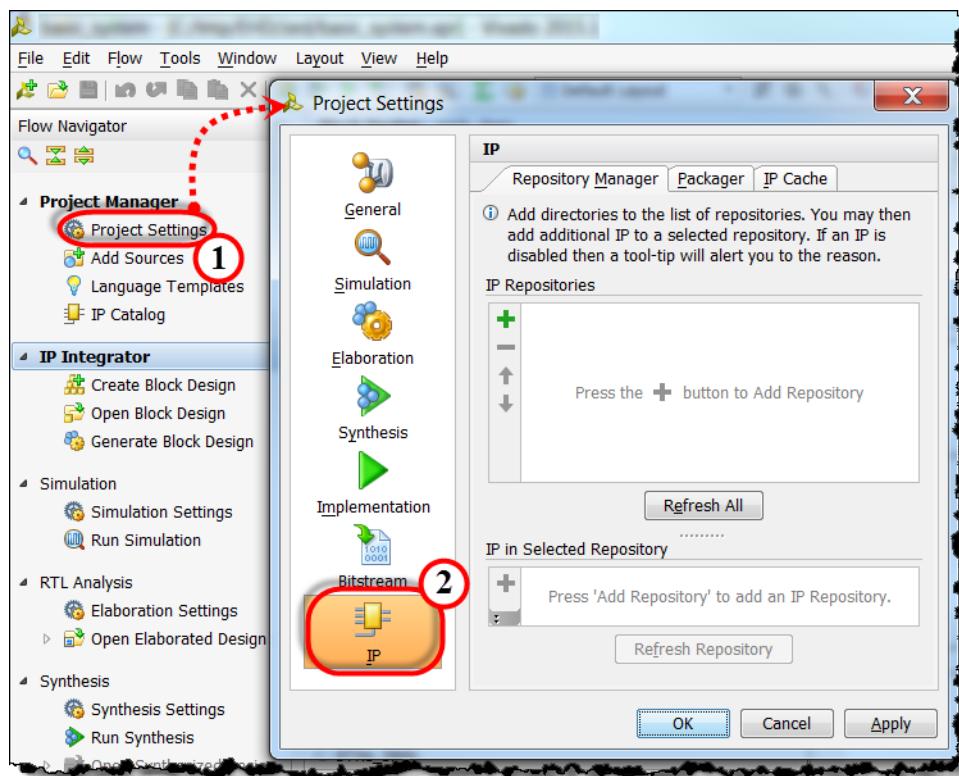


Figure 22: Accessing the Project's IP Settings

- 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.

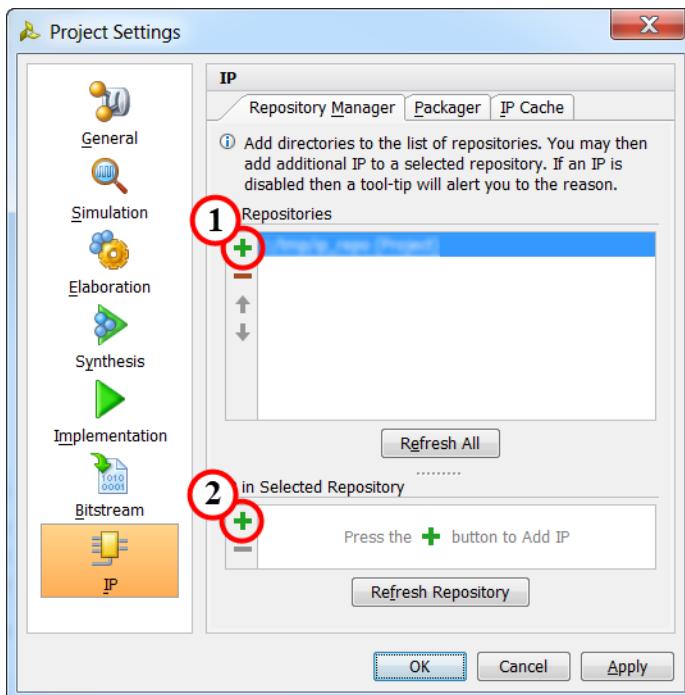


Figure 23: Adding the IP Repositories and the Specific IP

- 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.

Closing 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.

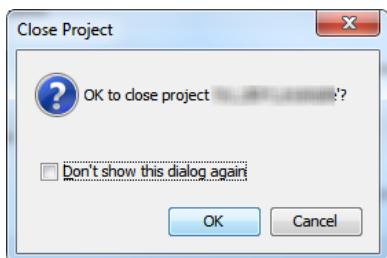


Figure 24: Close Project Dialog Box

- 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 25: Exit Vivado Dialog Box

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

Embedded Operations

In This Section

Exporting Only the Hardware Description for SDK	24
Exporting the Hardware Description and Bitstream for SDK.....	25
Exporting Hardware and Launching SDK.....	27
Launching SDK from within the Vivado Design Suite	30

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).

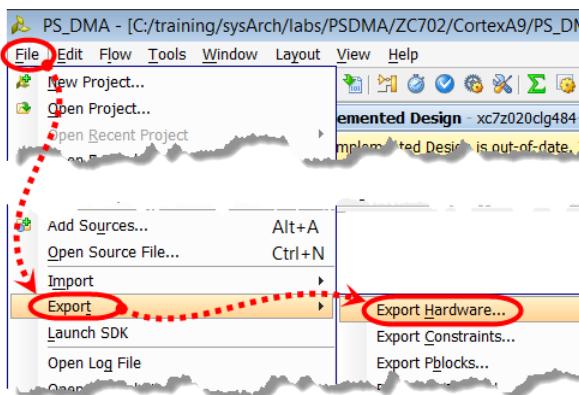


Figure 26: Exporting a Design to SDK

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.

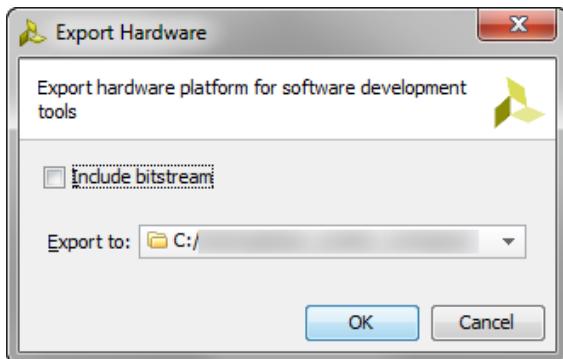


Figure 27: Export Hardware for SDK - Without Bitstream Generation

- 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**.

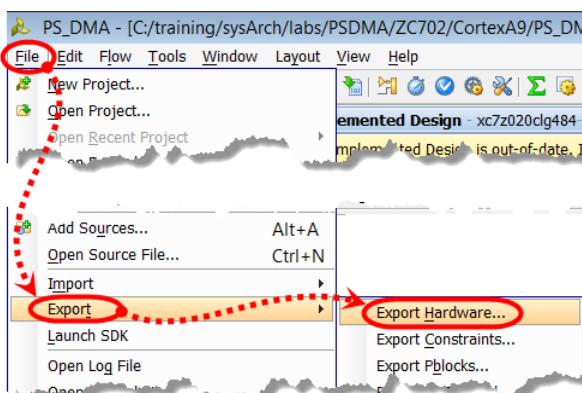


Figure 28: Exporting a Design to SDK

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.

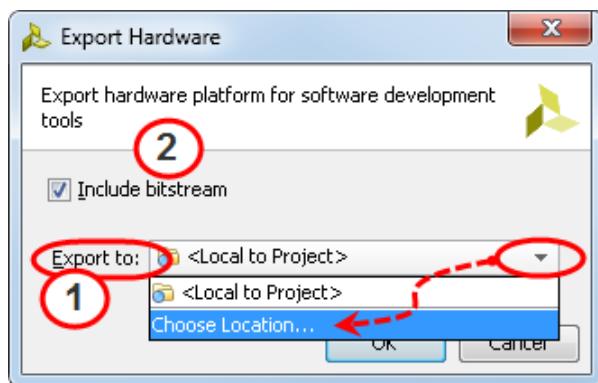


Figure 29: Using Choose Location to Browse to Destination Directory

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).

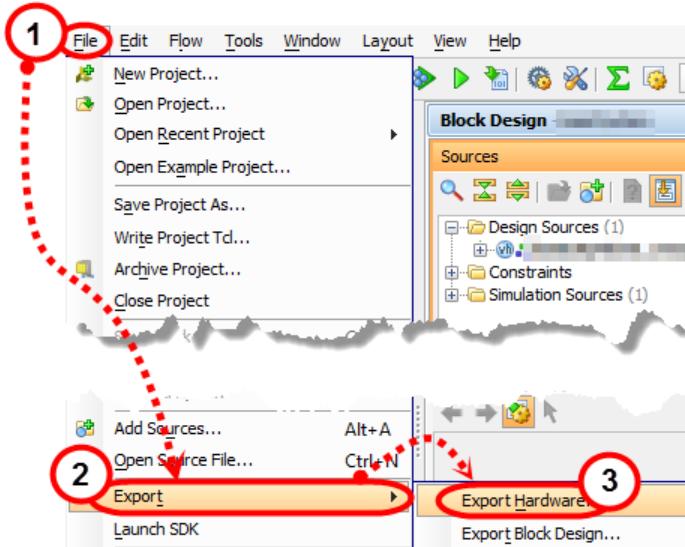


Figure 30: Exporting Hardware

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. From the dialog box, select **the location of the files to be exported from the Vivado Design Suite** from the Exported location drop-down list.
- 1-2-3. Select **a workspace** from the Workspace drop-down list.

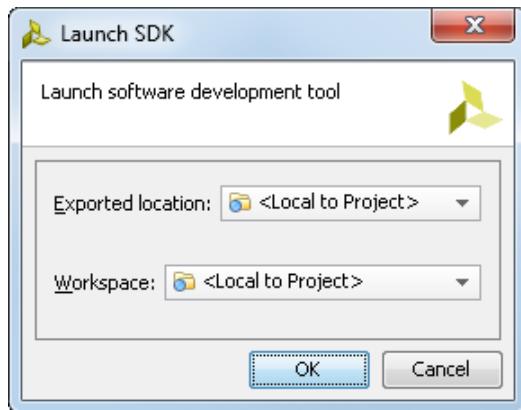


Figure 31: Launching SDK

- 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 and verify that any peripherals from your hardware design appear in the map with the correct address.

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

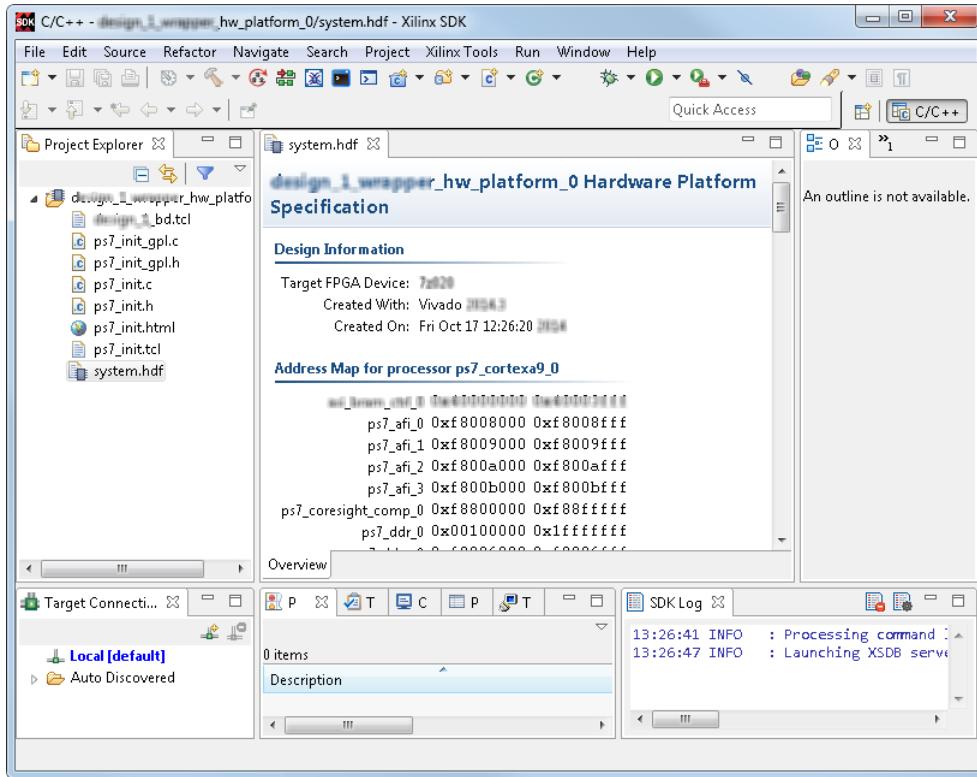


Figure 32: SDK Main Window

- 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 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**.

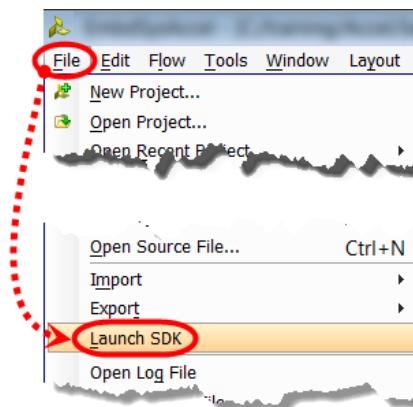


Figure 33: Launching SDK from within the Vivado Design Suite

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.

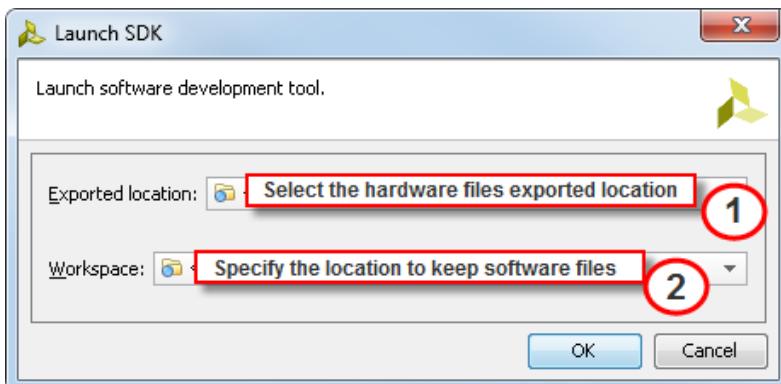


Figure 34: Launch SDK Dialog Box

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 *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.

- 1-1-4.** 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 launch point to begin exploring SDK, its 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.
1-1-6. Navigate to the Address Map section of the open hardware description file and verify that any peripherals from your hardware design appear in the map with the correct address.

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

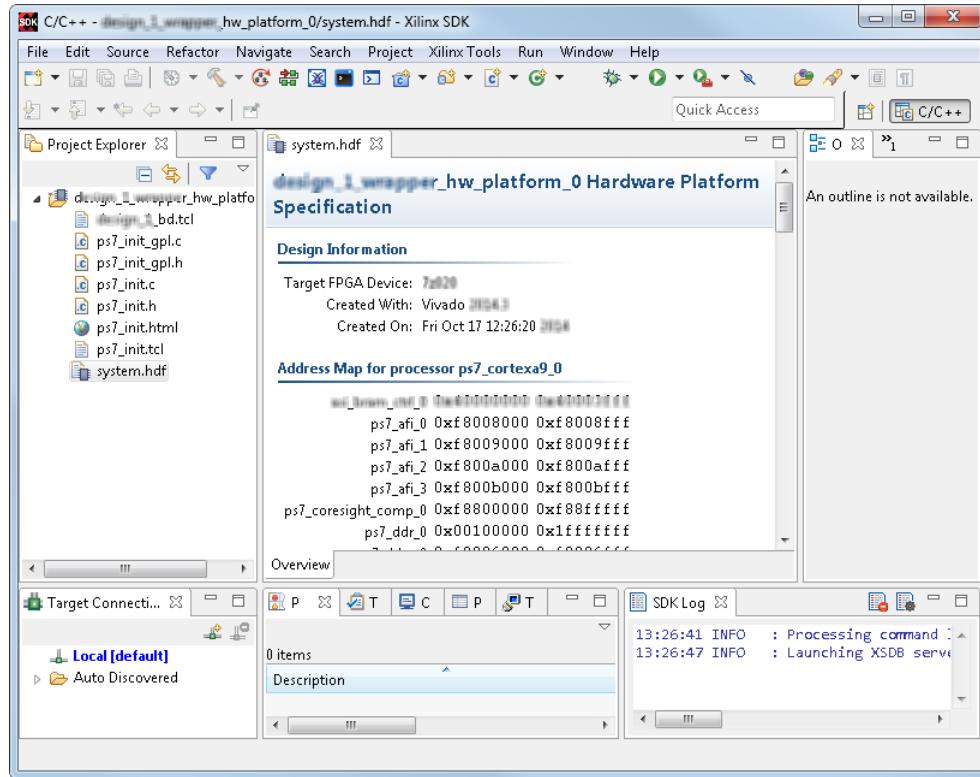


Figure 35: SDK Main Window

- 1-1-7.** 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	34
Opening a Block Design	35
Zoom Operations in the IPI Block Diagram Editor	36
Adding a Processor to the Block Design.....	37
Adding a Zynq AP SoC Processor Block.....	40
Adding a Zynq UltraScale+ MPSoC Processor Block.....	43
Running the Zynq Designer Assistance	46
Customizing the Zynq All Programmable SoC PS	47
Selecting a Zynq7 PS Preset Template.....	50
Customizing the MicroBlaze Processor	51
Adding IP to an IP Integrator Block Design	52
Making Connections Between IP Blocks in the IP Integrator.....	55
Renaming an IP Block.....	56
Customizing IP	57
Adding a Port to an IP Integrator Block Diagram.....	58
Adding an Interface Port to an IP Integrator Block Design.....	59
Making a Pin or Interface External.....	61
Running Connection Automation.....	61
Running Connection Automation for Multiple Modules	63
Running Block Automation	64
Locating Objects in the Board Design.....	65
Mapping Peripheral Addresses.....	67
Generating Output Products	68
Generating a Wrapper for a Block Design.....	69
Creating a Hierarchical Block in a Block Design.....	70
Adding IP to a Hierarchy Block	72
Expanding the Contents of a Hierarchical Block	73
Opening a Hierarchical Block in a New Tab.....	74
Marking Signals for Debugging	75
Updating IP.....	76
Running a Design Rule Check/Design Validation.....	78
Validating the Block Design.....	79

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. If necessary, expand the **IP Integrator** field under the Flow Navigator (1).
- 1-1-2. Click **Create Block Design** to start creating a new IP subsystem (2).

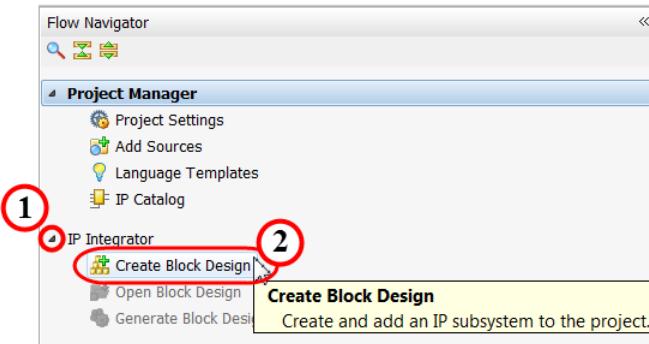


Figure 36: Launching the IP Integrator

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

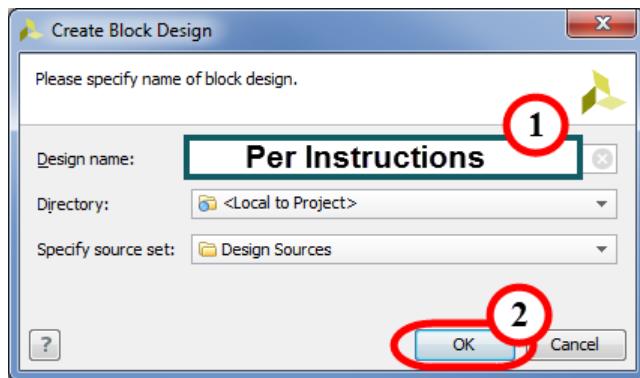


Figure 37: Creating an IP Integrator Block Design

- 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.

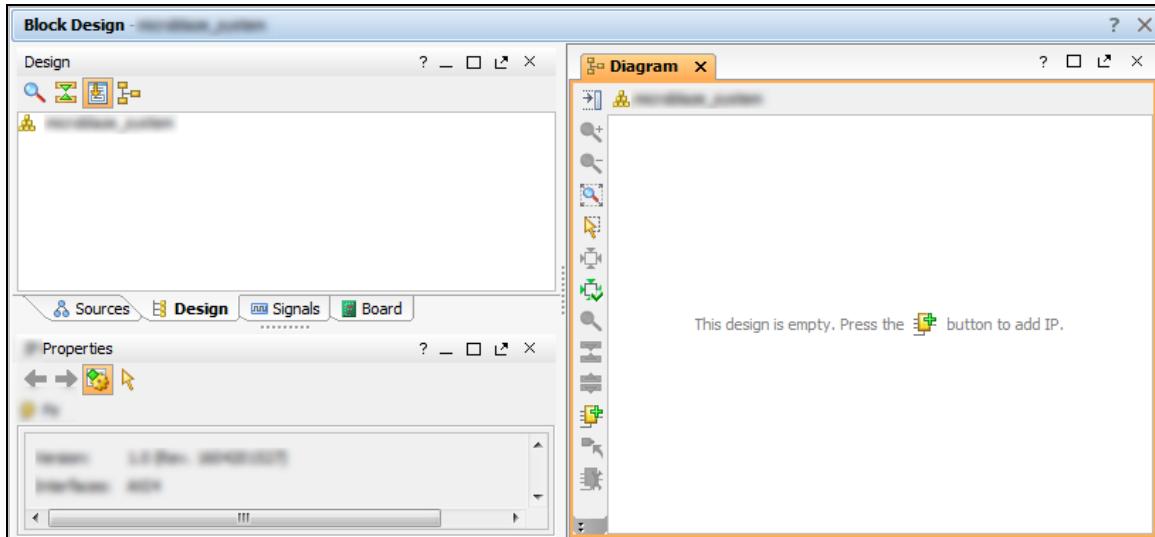


Figure 38: Initial View of the IP Integrator Tool Showing an Informational Message

Opening a Block Design

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

- 1-1-1. Access the **Hierarchical > Sources** 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 ().

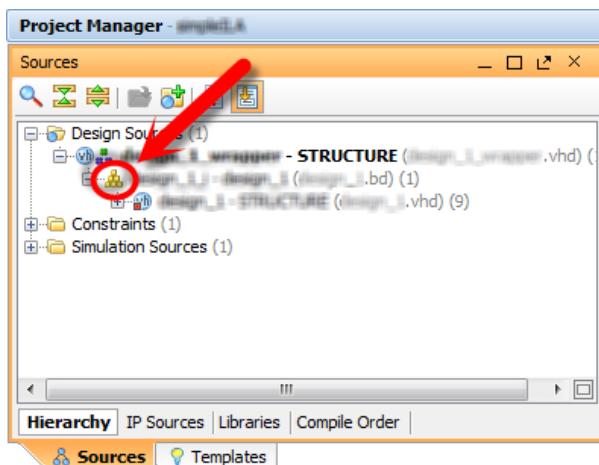
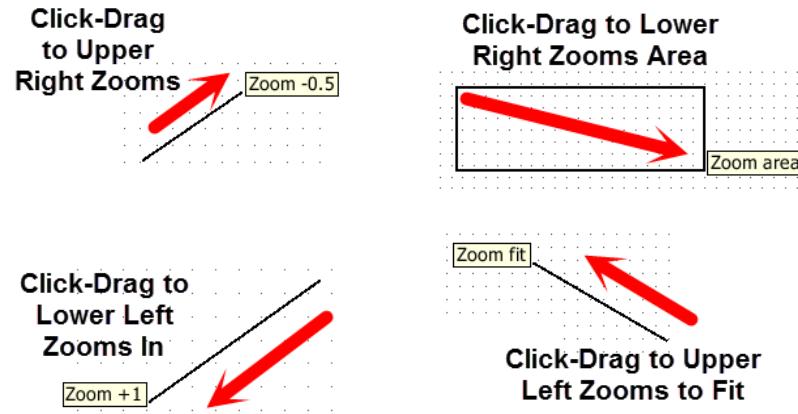


Figure 39: Opening an Existing Block Design

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 vertical toolbar to the left of the block design canvas.

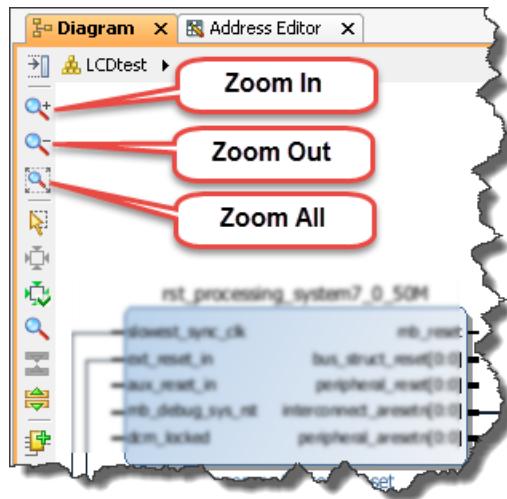


Figure 40: Block Diagram Zoom Toolbar

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:
- Click the **Add IP** icon on the left border of the workspace.

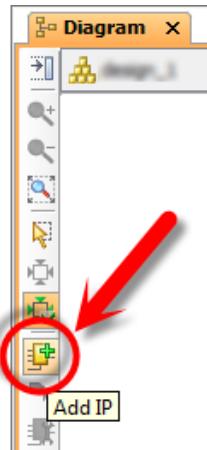


Figure 41: Opening the IP Catalog from the Add IP Icon

-- OR --

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

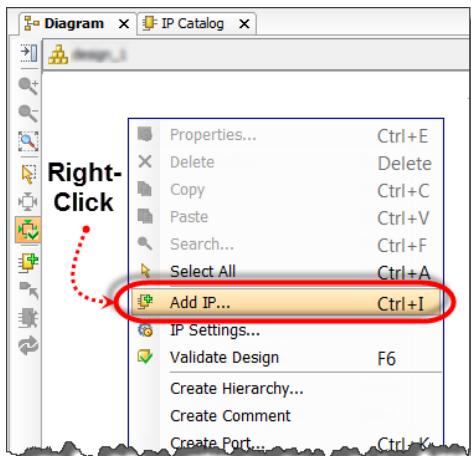


Figure 42: Opening the IP Catalog from Right-Clicking the Workspace

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! 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 processor block.

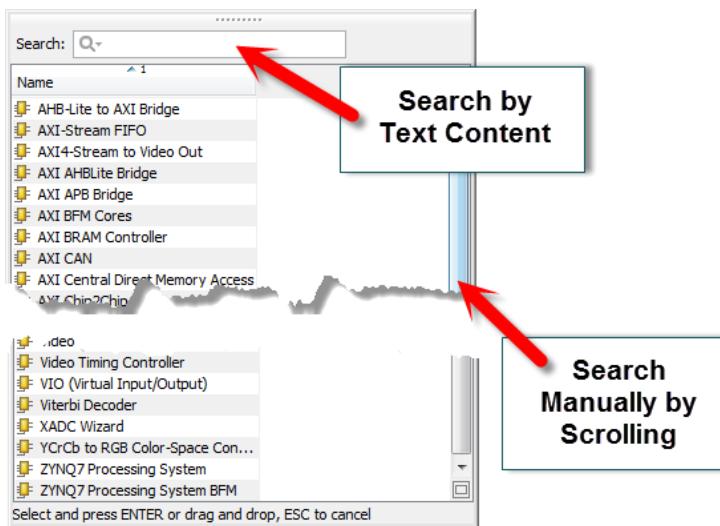


Figure 43: Two Mechanisms to Search for IP

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

For example, **zynq** for the Zynq All Programmable SoC or **microblaze** for the MicroBlaze processor. Note that the Zynq AP SoC PS IP will only appear when Zynq-7000 AP SoCs or boards with these parts have been selected for the design.

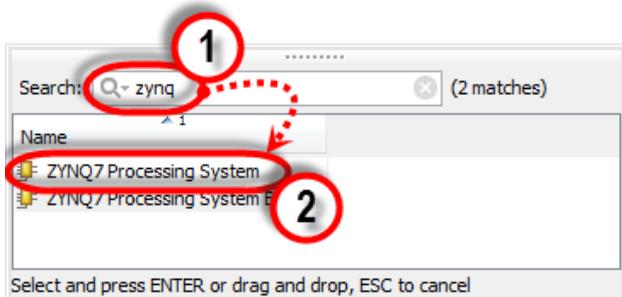


Figure 44: Narrowing Search Parameters (Zynq AP SoC)

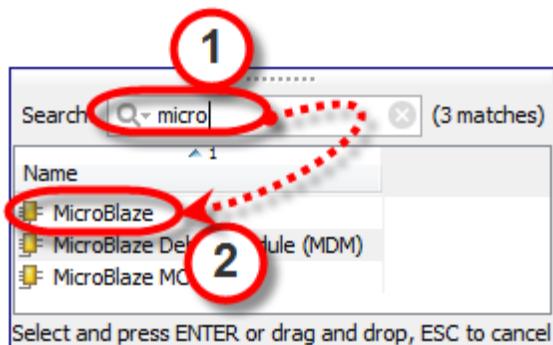


Figure 45: Narrowing Search Parameters (MicroBlaze Processor)

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

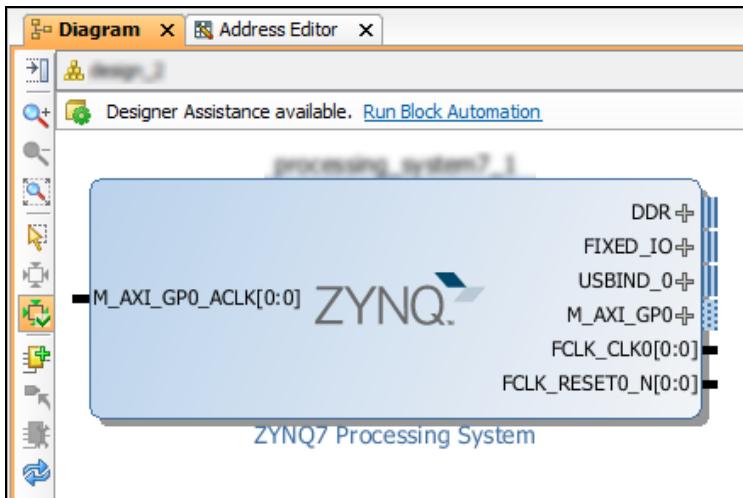


Figure 46: Zynq Processing System IP

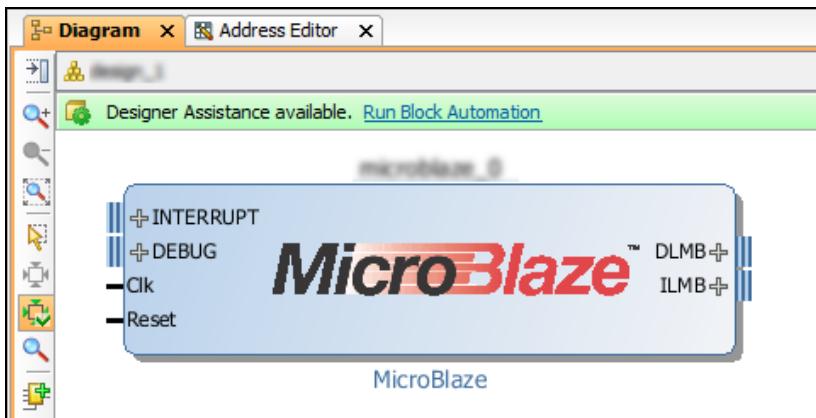


Figure 47: MicroBlaze Processor IP

Adding a Zynq AP SoC Processor Block

1-1. Add a Zynq All Programmable SoC 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.

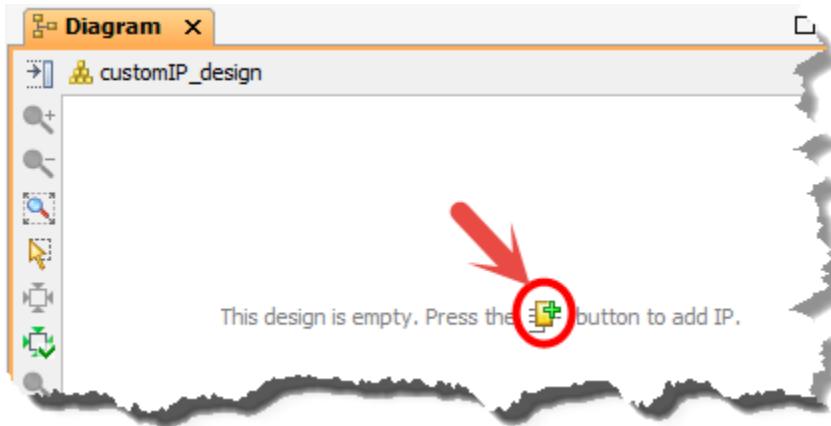


Figure 48: Opening the IP Catalog from the Initial Information Bar Display

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 left toolbar of the workspace.



Figure 49: Opening the IP Catalog from the Add IP Icon

-- OR --

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

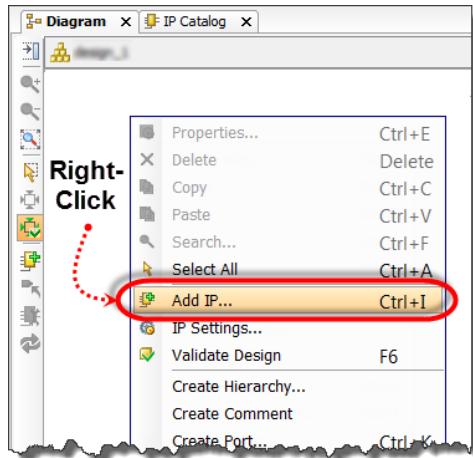


Figure 50: Opening the IP Catalog from Right-Clicking the Workspace

-- OR --

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

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! 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 All Programmable SoC processor block.

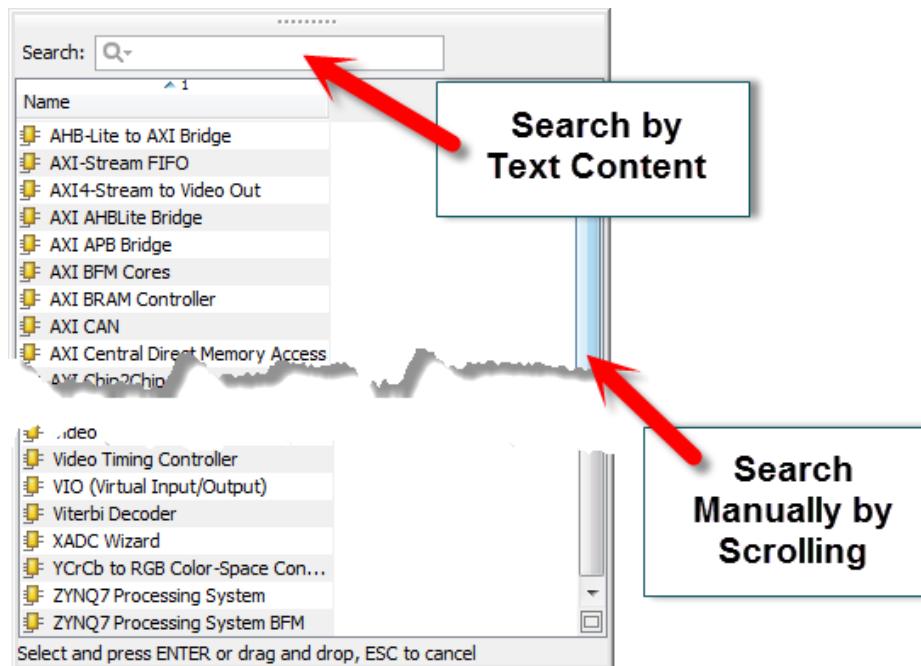


Figure 51: Two Mechanisms to Search for IP

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

The Zynq PS IP will only appear when Zynq-7000 SoCs or boards with these parts have been selected for the design.

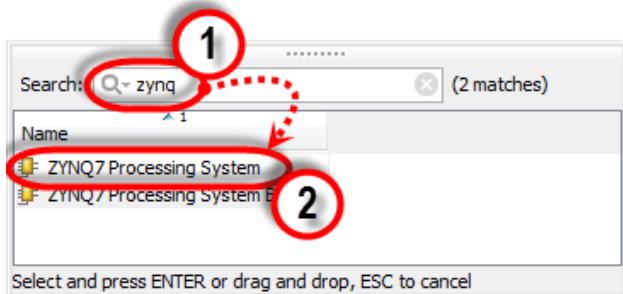


Figure 52: Narrowing Search Parameters (Zynq AP SoC)

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

- 1-1-3.** Double-click the **ZYNQ7 Processing System** IP entry to add the ZYNQ7 Processing System block to the design (2).

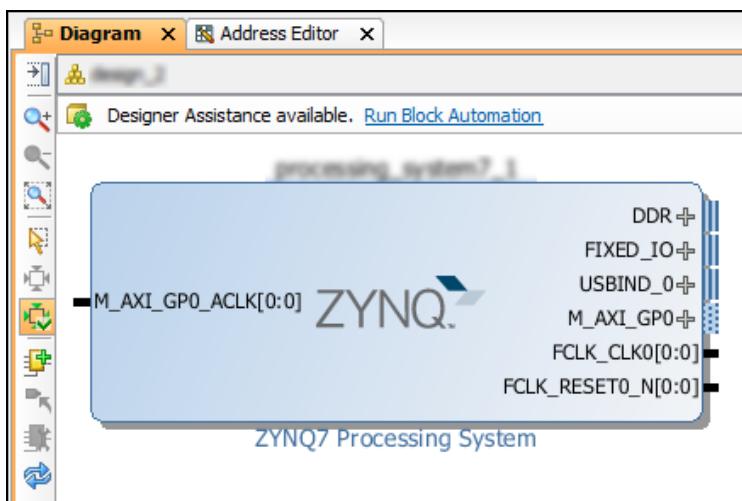


Figure 53: Zynq Processing System IP

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.

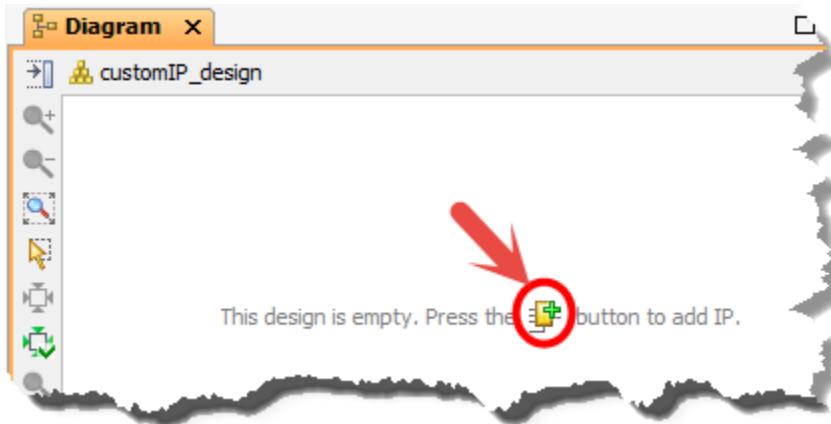


Figure 54: Opening the IP Catalog from the Initial Information Bar Display

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 left toolbar of the workspace.

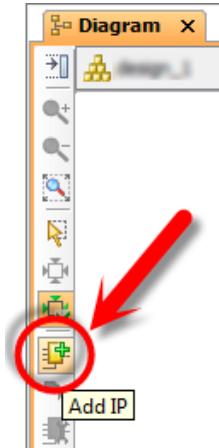


Figure 55: Opening the IP Catalog from the Add IP Icon

-- OR --

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

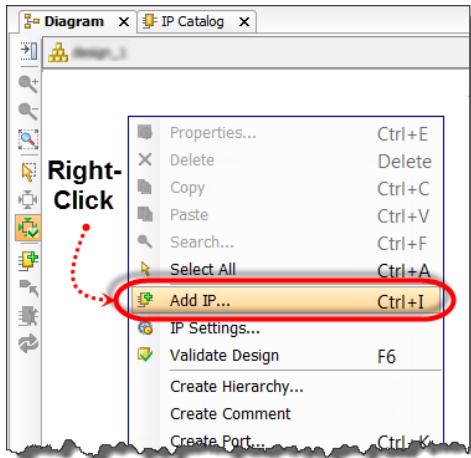


Figure 56: Opening the IP Catalog from Right-Clicking the Workspace

-- OR --

- 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.

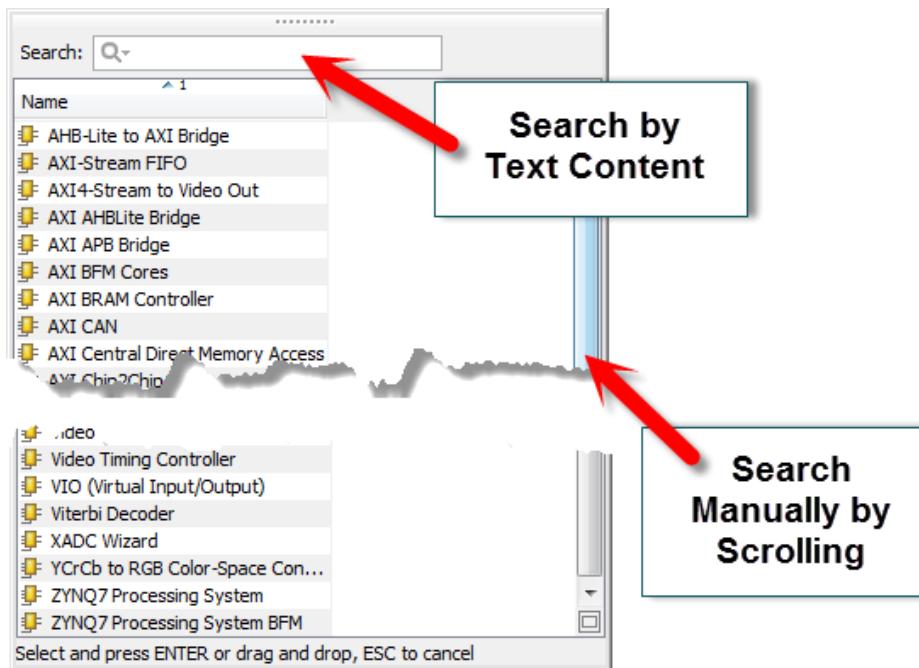


Figure 57: Two Mechanisms to Search for IP

- 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.

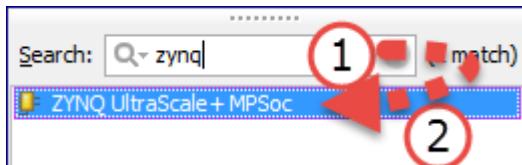


Figure 58: Selecting the Zynq UltraScale+ MPSoC IP

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).

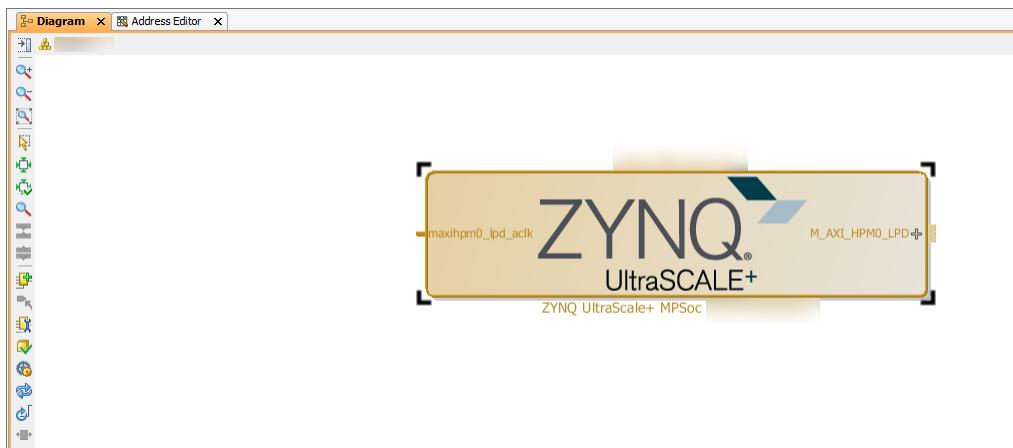


Figure 59: Zynq UltraScale+ MPSoC Block in the Block Diagram

Running the Zynq Designer Assistance

Many IP blocks, including the Zynq All Programmable 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 All Programmable 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 (1).



Figure 60: Selecting the Diagram Tab

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

The Run Block Automation dialog box opens.

- 1-1-3. Deselect **Apply Board Preset** because most designers will set the board preset in the Re-customization wizard.

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

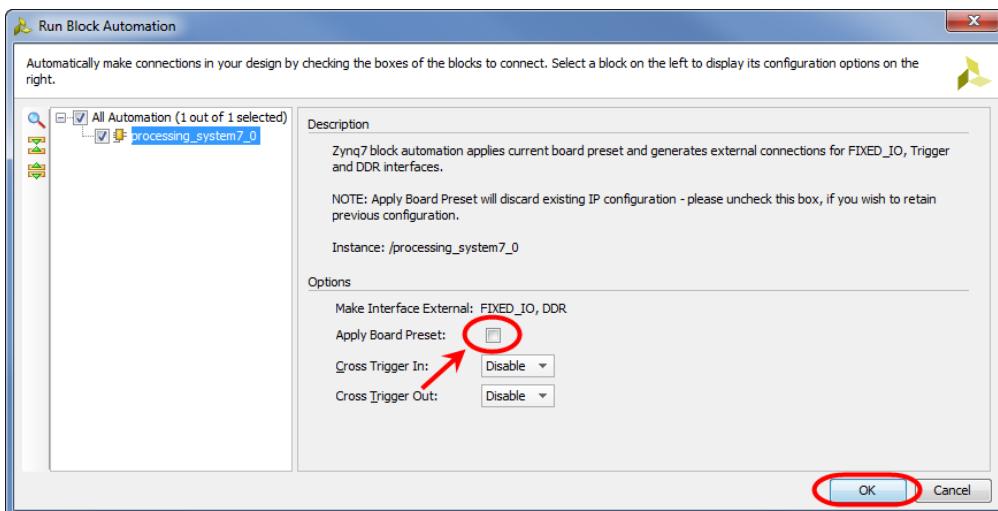


Figure 61: Running the Designer Automation Assistance on the Zynq PS

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 All Programmable SoC PS

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

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

- 1-1-1. Double-click the **Zynq PS** icon.

-- OR --

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

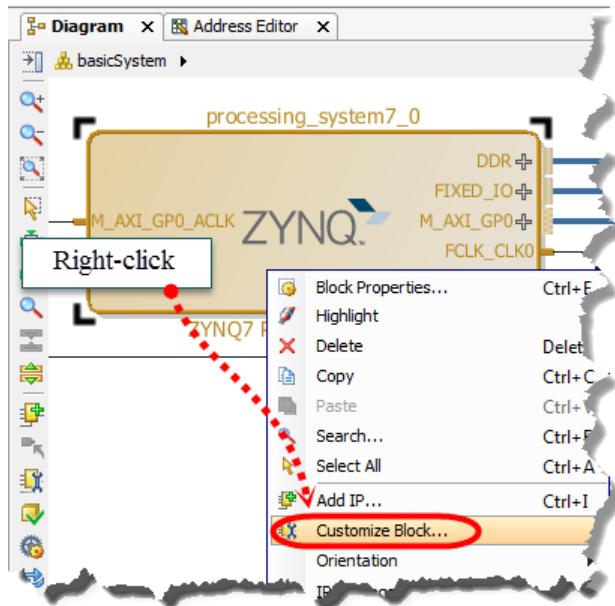


Figure 62: Opening the Re-customization Dialog Box for the Zynq All Programmable SoC PS

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

1-2-1. Click **Presets**.

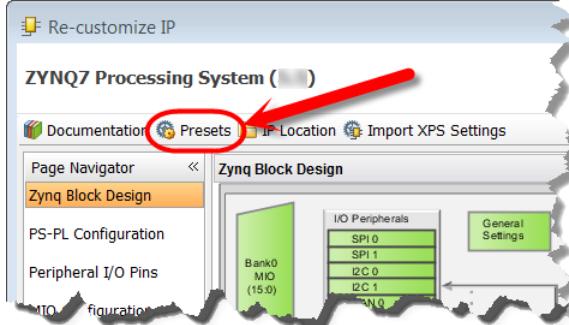


Figure 63: Accessing the Presets Button

1-2-2. Select the template for your board.

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.

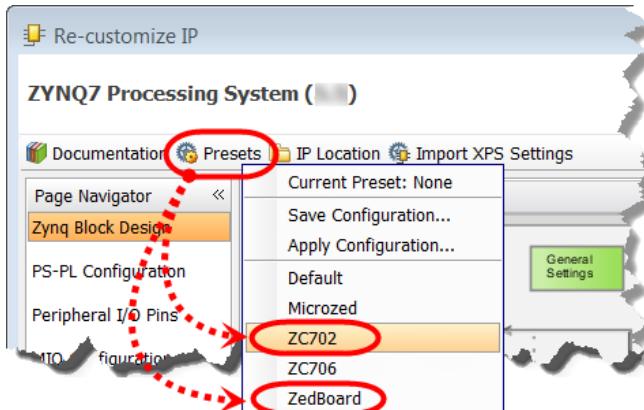


Figure 64: Selecting a Preset Template (ZedBoard and ZC702 shown as examples)

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.

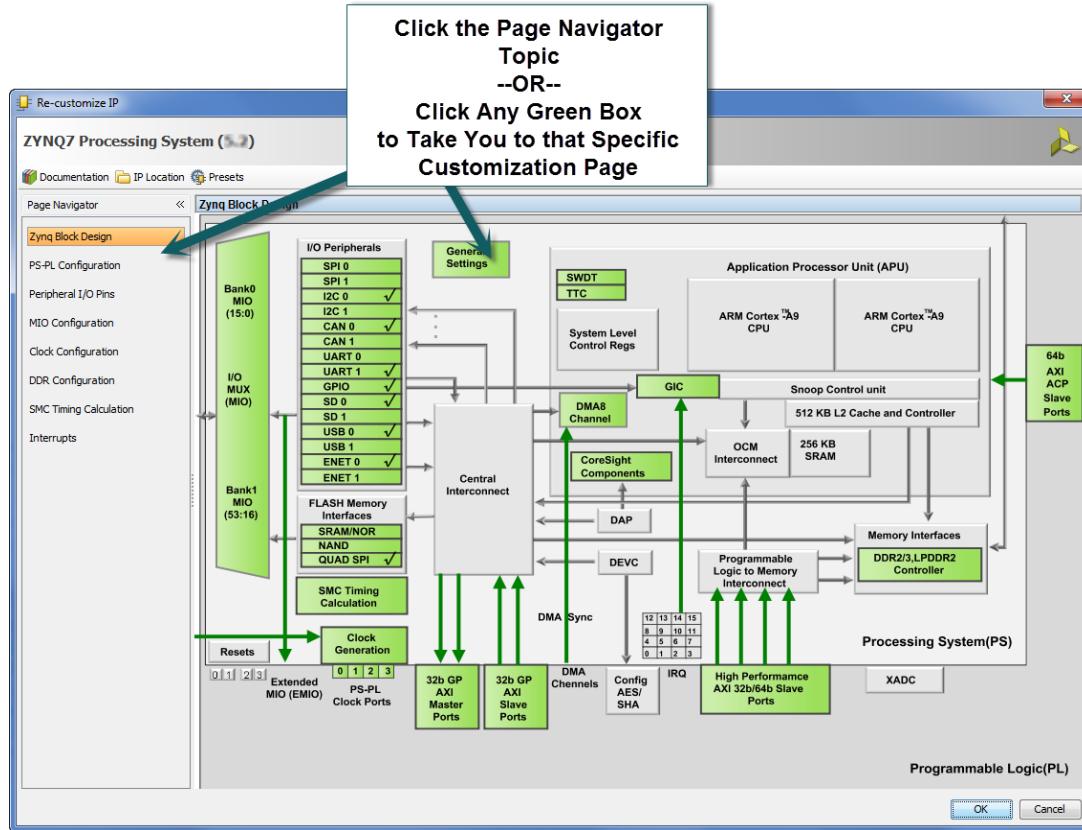


Figure 65: Navigating the Zynq PS Recustomization Dialog Box

- PS-PL Configuration: This page contains all the signals that cross the PS-PL boundary. These signals are broken down further into:
 - General: Contains 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 HP 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

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**.

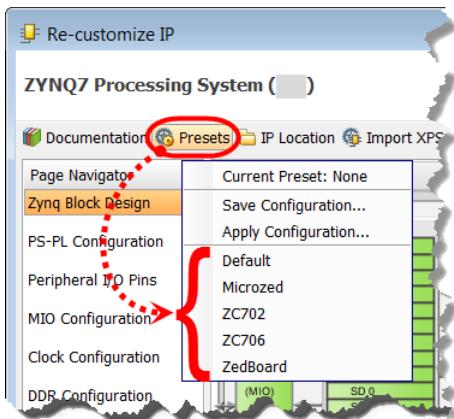


Figure 66: Selecting a Preset Template

- 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.

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

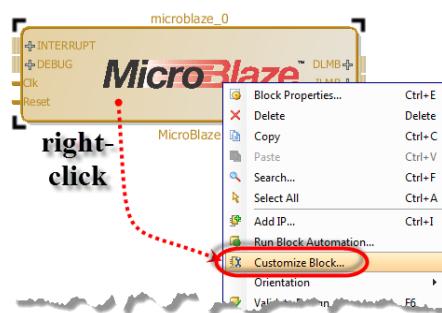


Figure 67: Starting Customization on the MicroBlaze Processor

The Re-customization IP dialog box opens.

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

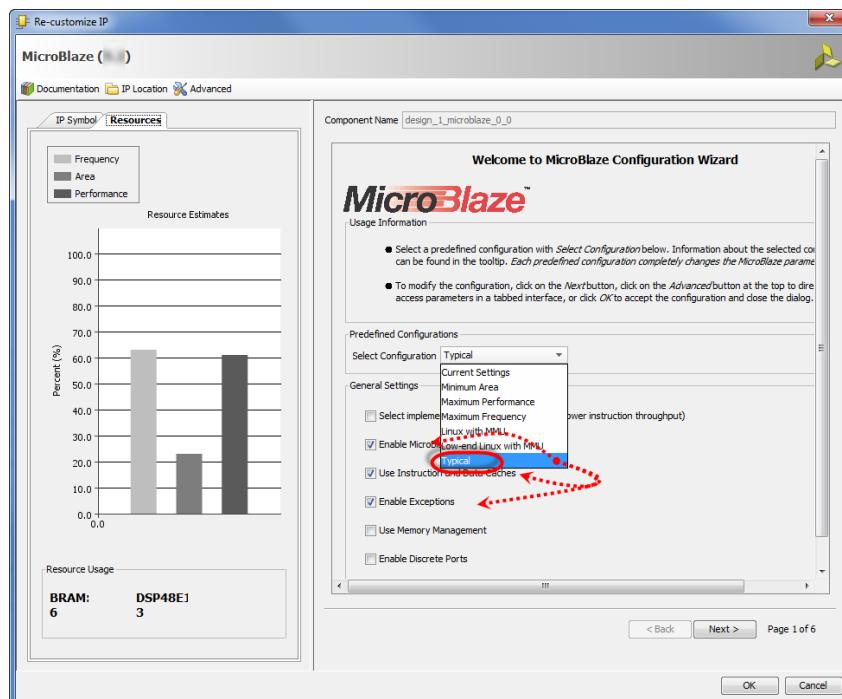


Figure 68: Selecting the Typical Configuration for the MicroBlaze Processor

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.

1-1-4. Click **OK** to save this configuration.

Adding IP to an IP Integrator Block Design

The IP integrator performs three fundamental tasks: collecting IP, connecting IP, and performing "real-time" design rule checks (DRCs). Additional checks are performed during the validate design and design generation processes. This step focuses on the first of these three tasks.

1-1. Open the IP catalog.

1-1-1. The IP catalog can be opened in one of several ways:

- o Click the **Add IP** icon on the left border of the workspace.

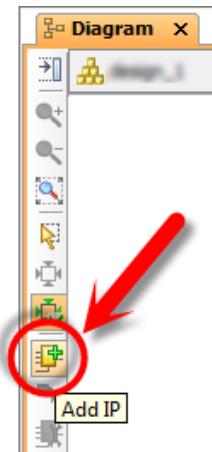


Figure 69: Opening the IP Catalog from the Add IP Icon

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

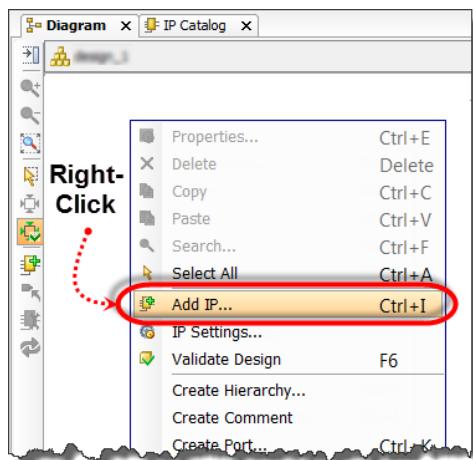


Figure 70: Opening the IP Catalog from Right-Clicking the Workspace

- Press <**Ctrl + I**> to open the IP catalog.
- If the design is empty, the informational bar offers to help get you started. You can click **Add IP** to open the IP catalog.

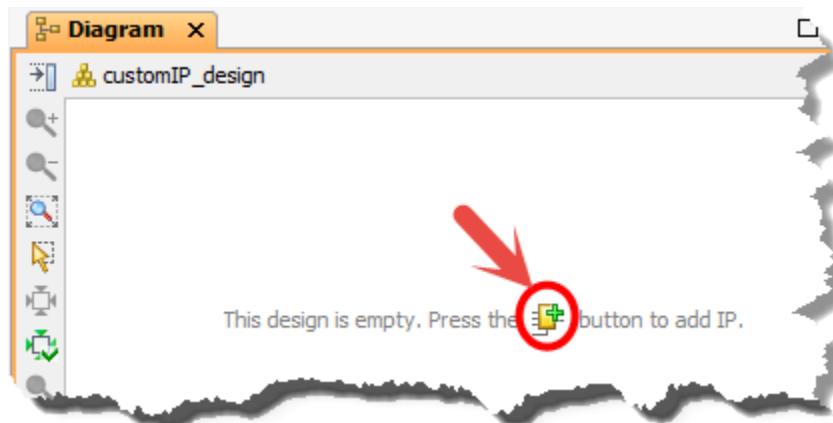


Figure 71: Opening the IP Catalog from the Initial Information Bar Display

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 the desired piece of IP.

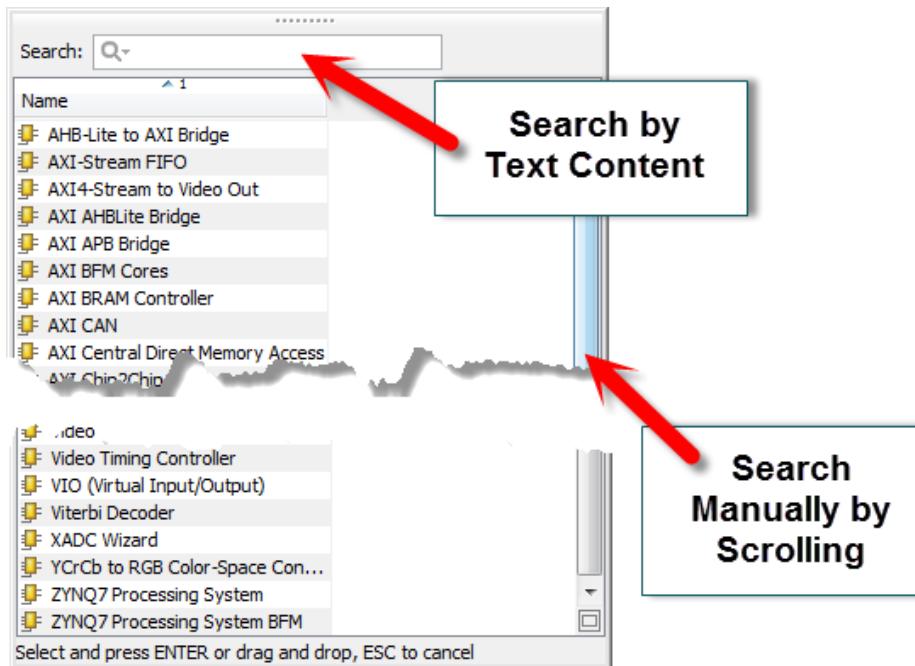


Figure 72: Two Mechanisms to Search for IP

1-2-3. Double-click the name of the IP to add it to the design.

or

Drag-and-drop the IP into the workspace.

The IP catalog will close once the IP has been added to the design.

Making Connections Between IP Blocks in the IP Integrator

Making connections is a simple process in the IP integrator tool. Each port or interface on a block is first selected. As the connecting wire is stretched to its destination, an assistant runs in the background and places a green check mark next to the port or interface that is allowed to be connected when the cursor approaches a legitimate connection point.

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

1-1. Connect two IP blocks.

- 1-1-1. Click a port or interface to connect.

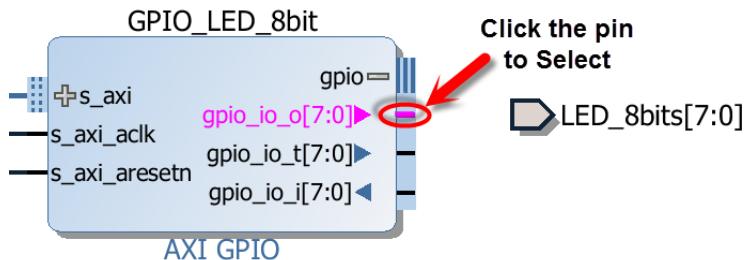


Figure 73: Example of a Selected Port

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 a port with a green arrow next to it.

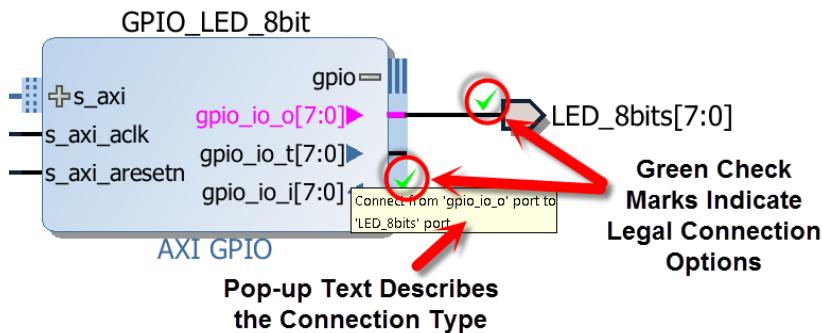


Figure 74: Making the Connection

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

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

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 under the General tab.

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

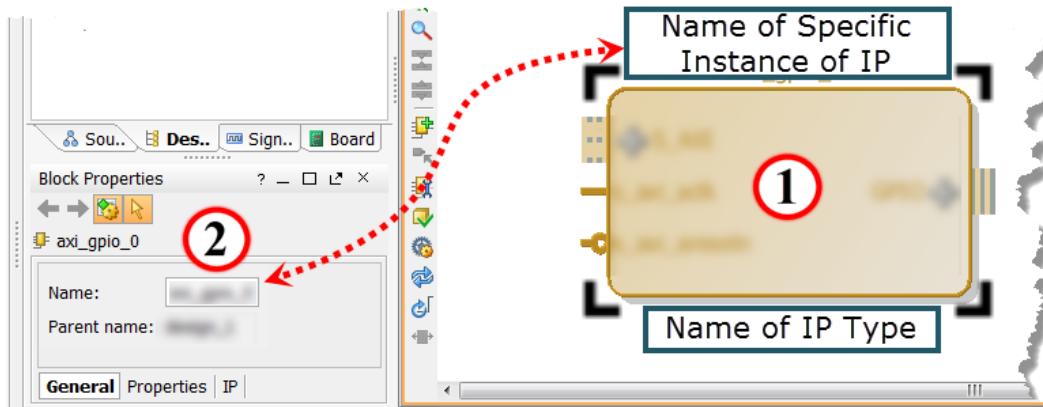


Figure 75: Renaming an IP Block

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

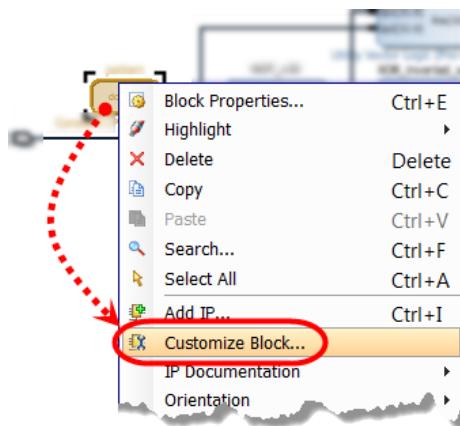
Customizing IP

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. Either:

- Double-click the IP block or
- Right-click the IP block and select **Customize Block** from the context menu.



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

1-1-2. Make the necessary customizations.

1-1-3. Click **OK** to save and exit the Re-customization Wizard.

Adding a Port to an IP Integrator Block Diagram

Ports are the mechanism that allow signals to pass between hierarchical levels. Ports can be combined together to form interfaces. This entry focuses on single port creation.

1-1. Create a port.

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

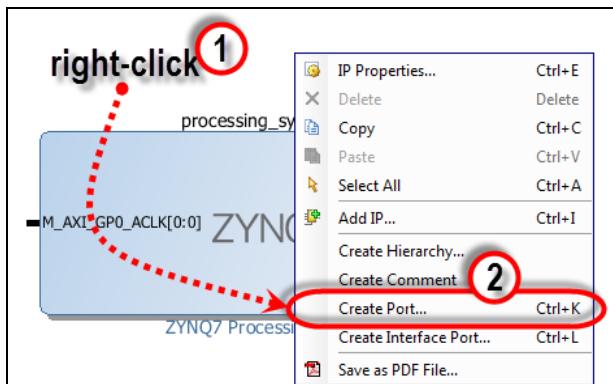


Figure 76: Creating a New Port

The Create Port dialog box opens.

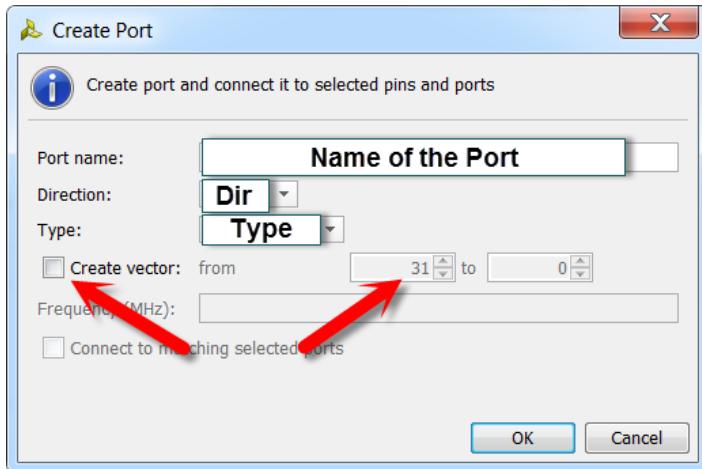


Figure 77: Create Port Dialog Box

- 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**.

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

1-1-6. 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).

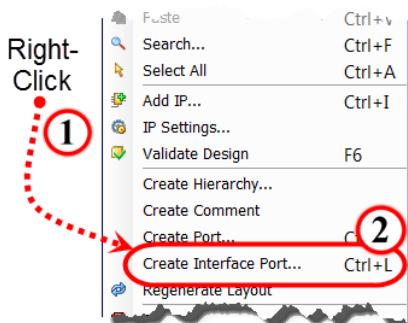


Figure 78: Creating an Interface Port

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

The Create Interface Port dialog box opens.

1-1-3. Enter <interface_port_name>.

1-1-4. Enter or select the VLVN.

Typically this is populated by the tool. VLVN stands for Vendor, Library, Name, and Version.

1-1-5. Select <interface_mode>

[CD: explain more about this]

1-1-6. Choose <interface_port_connection_type>

[CD: explain more about this]

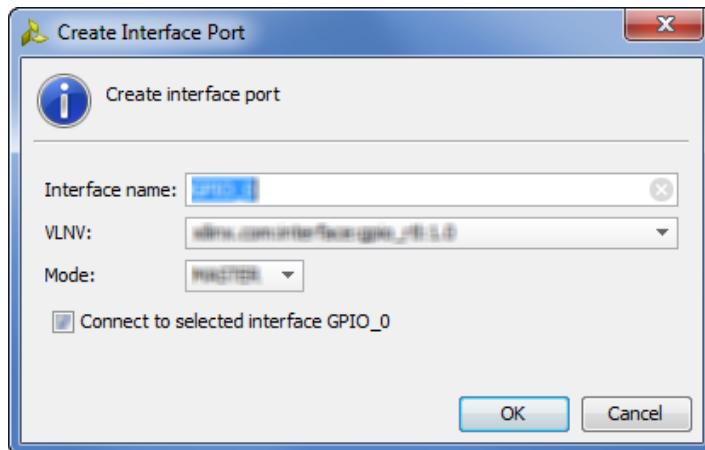


Figure 79: Create Interface Port Dialog Box

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 and select **Make External**.

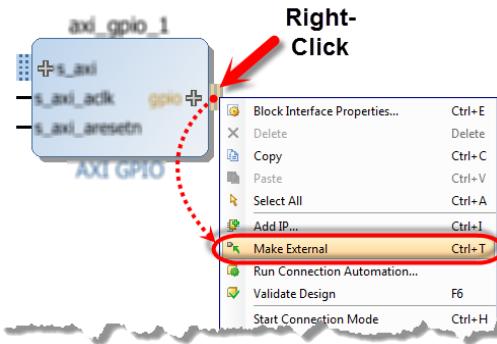


Figure 80: Making a Port or Interface External

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

Running Connection Automation

Many IP blocks are supported by the Connection Assistant. The Connection Assistant automates many of the commonly used basic connections, such as to AXI ports.

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.

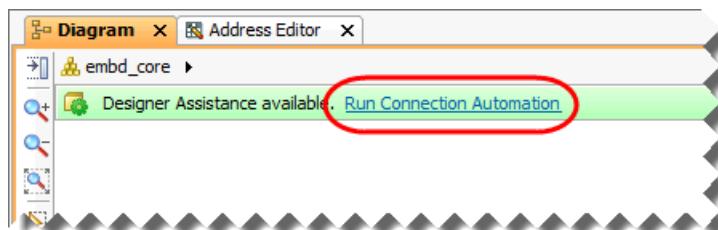


Figure 81: Running Connection Automation

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.** Clicking the **Expand All** icon () to ensure that the list of available automations is fully visible.
- 1-1-3.** Check the node that corresponds to the AXI input side of the **the desired piece of IP** block (i.e., S_AXI interface).

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.

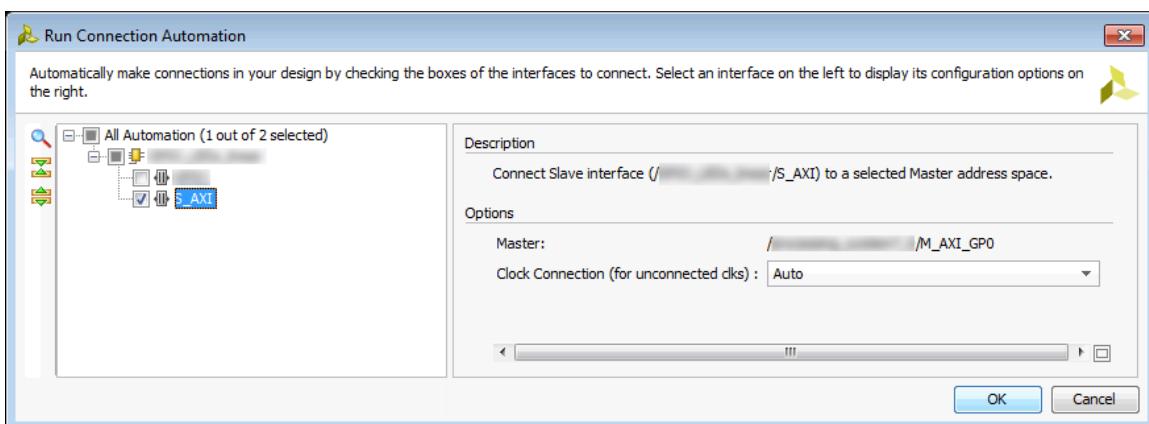


Figure 82: Run Connection Automation Dialog Box

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

These steps had you select only one automation so that you have the opportunity to make some connections manually. But if you already understand how to make the remaining connections manually, you are encouraged to select the other automations as well. Additionally, note that use of Designer Assistance is not a required part of the design flow. Any automation done via Designer Assistance can also always be done manually.

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.

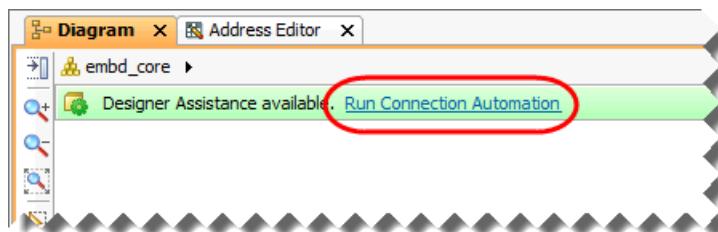


Figure 83: Running Connection Automation

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 (green arrow) 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.

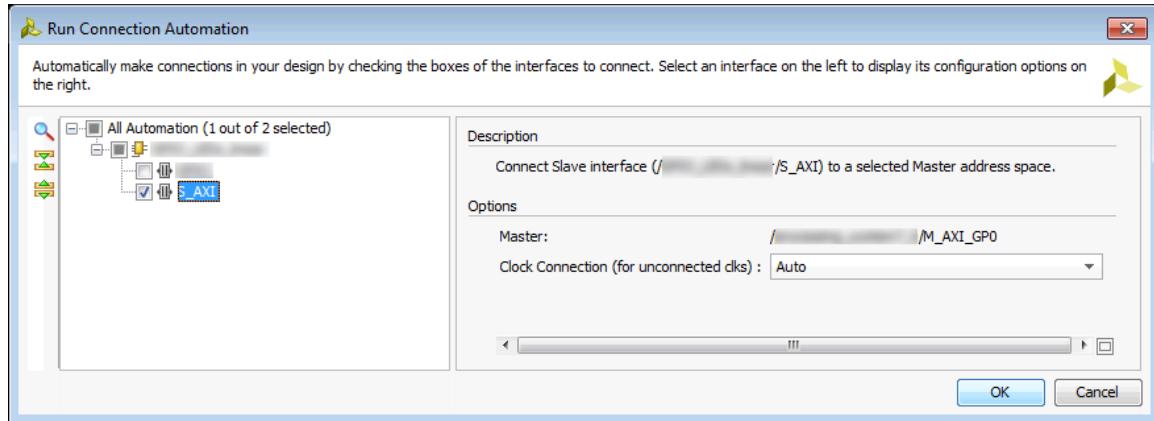


Figure 84: Run Connection Automation Dialog Box

- 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

Many IP blocks are supported by the Designer Assistance feature for automating the configuration of an IP block as well as block-specific connections. This is referred to as Block Automation and allows designers to quickly configure new IP blocks for common use cases.

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 85: Designer Assistance Offering Block Automation

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.
- 1-1-3. 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

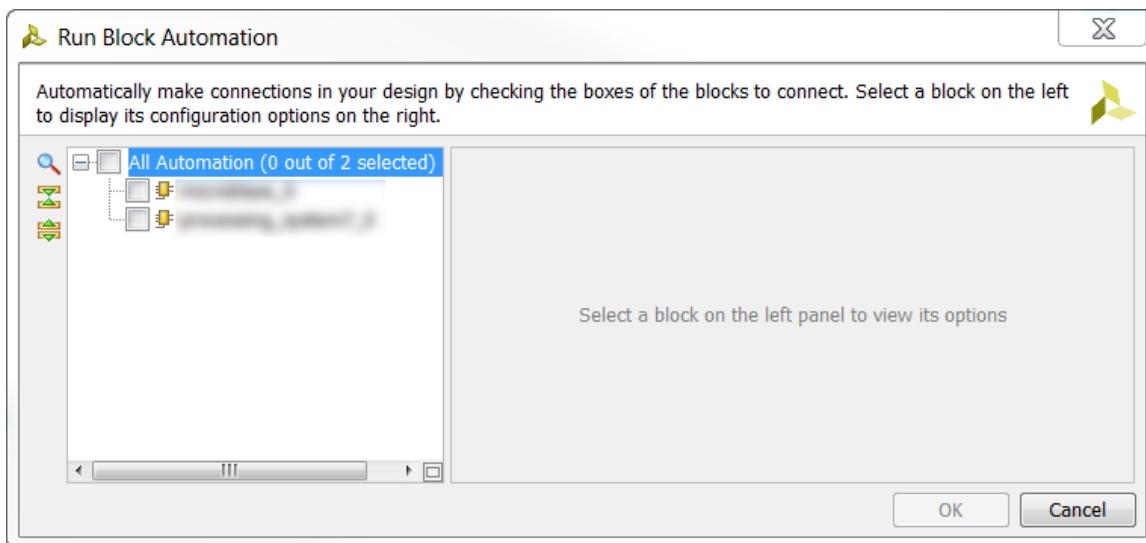


Figure 86: Run Block Automation Dialog Box

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

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.

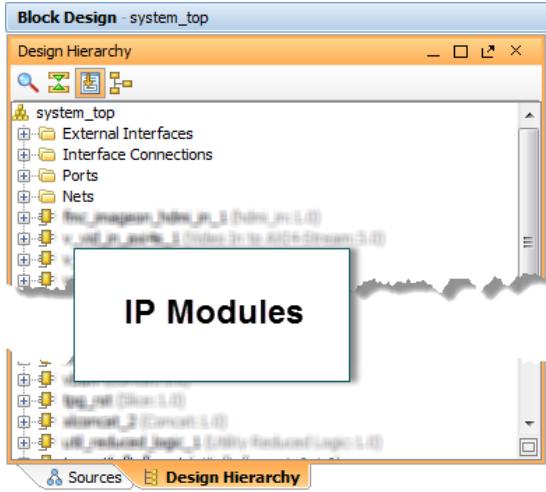


Figure 87: Locating the Design Hierarchy View

- 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.

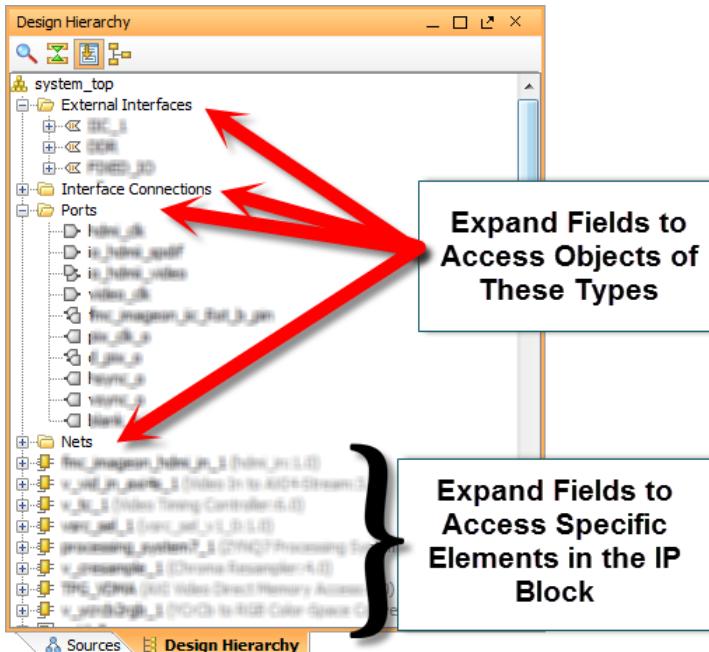


Figure 88: Expanding Branches in the Design Hierarchy Tree

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

The Schematic view will highlight (select) that item and zoom to the area around it.

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.

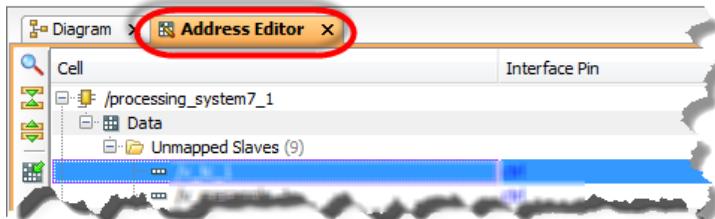


Figure 89: Locating the Address Editor Tab

1-1-2. Click the **Expand** icon (green triangle) 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.

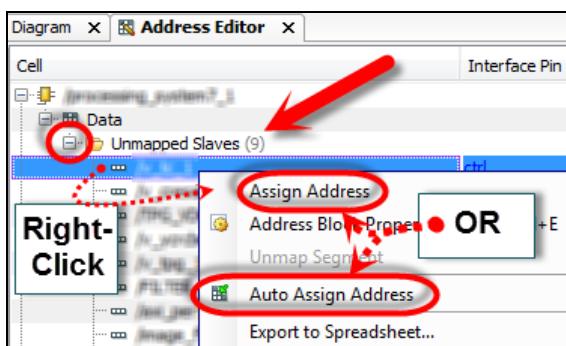


Figure 90: Opening the Dialog Box for Assigning an Address

Generating Output Products

Once a block design has been created, its output products need to 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. Right-click to highlight the object(s) you want to generate the output products for from the Sources window.

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

- 1-1-2. Select **Generate Output Products**.

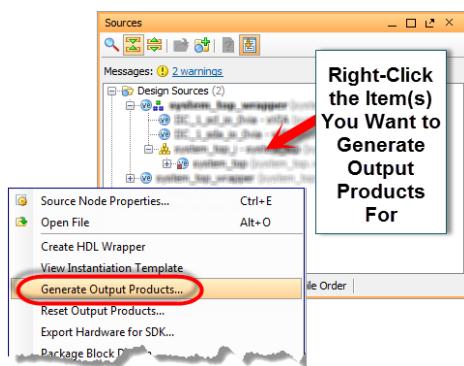


Figure 91: Selecting the Items to Generate Output Products For

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

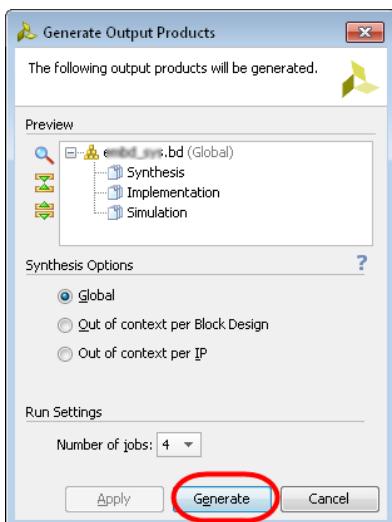


Figure 92: Managing the Output Products

Typically, the default settings for Synthesis Options and Run Settings are used.

- 1-1-3.** Click **Generate** to start the generation process.

When generation completes, a success dialog box opens.

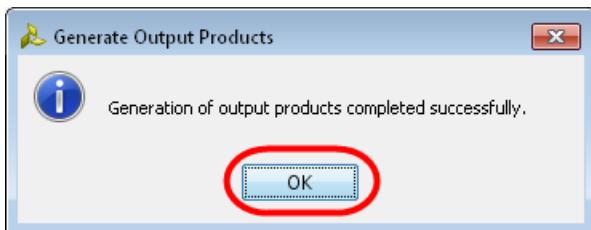


Figure 93: Successful Output Products Generation

- 1-1-4.** Click **OK** to close.

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).
- 1-1-2.** Right-click the block design name under the Block Design > Sources > Design Sources window (2).
- 1-1-3.** Select **Create HDL Wrapper** to begin the wrapper creation process (3).

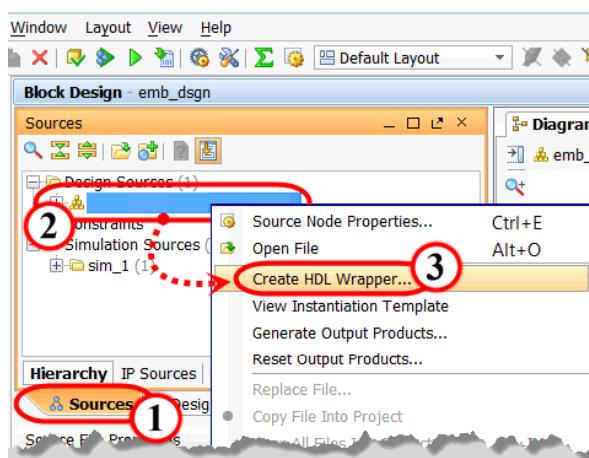


Figure 94: Selecting Create HDL Wrapper

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).

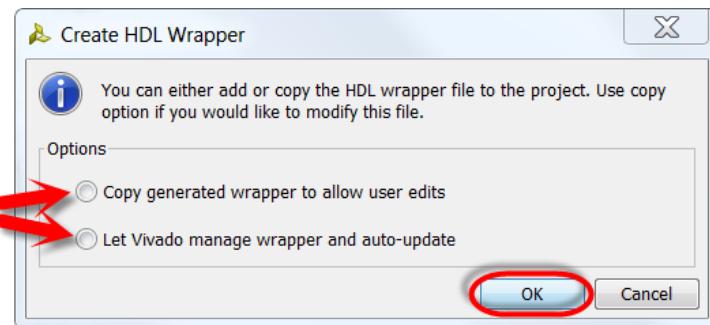


Figure 95: Generating the Wrapper and Copying the Generated File into the Project

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.

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.

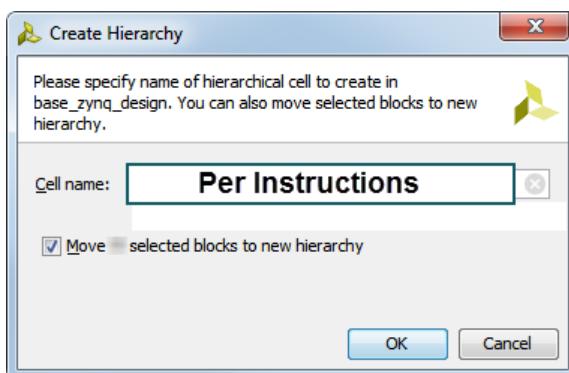


Figure 96: Create Hierarchy Dialog Box

- 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.

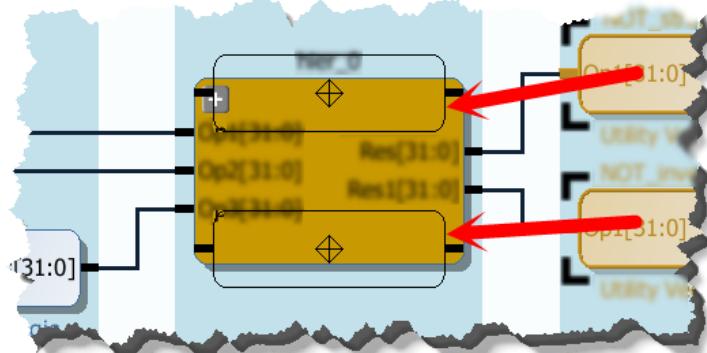


Figure 97: Dragging IP into a Hierarchical Block

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.

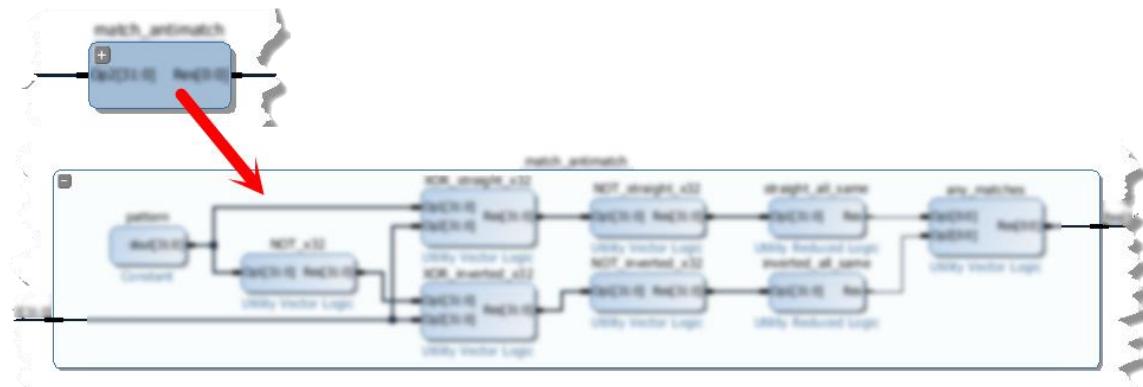


Figure 98: Expanding a Hierarchy Block

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:

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

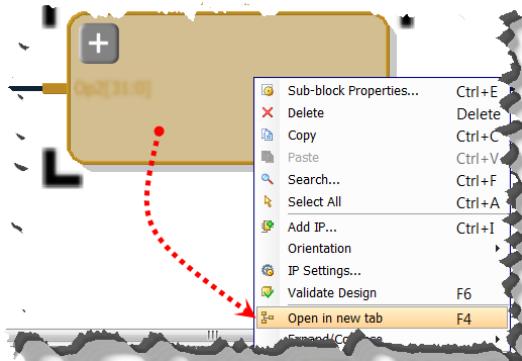


Figure 99: Opening the Hierarchy Block in a New Tab

- o 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).
- 1-1-3. Select **the name of your hierarchy** from the listing of hierarchies to open the hierarchy in a new tab (2).



Figure 100: Access the Hierarchical Blocks in a Block Diagram

The 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.

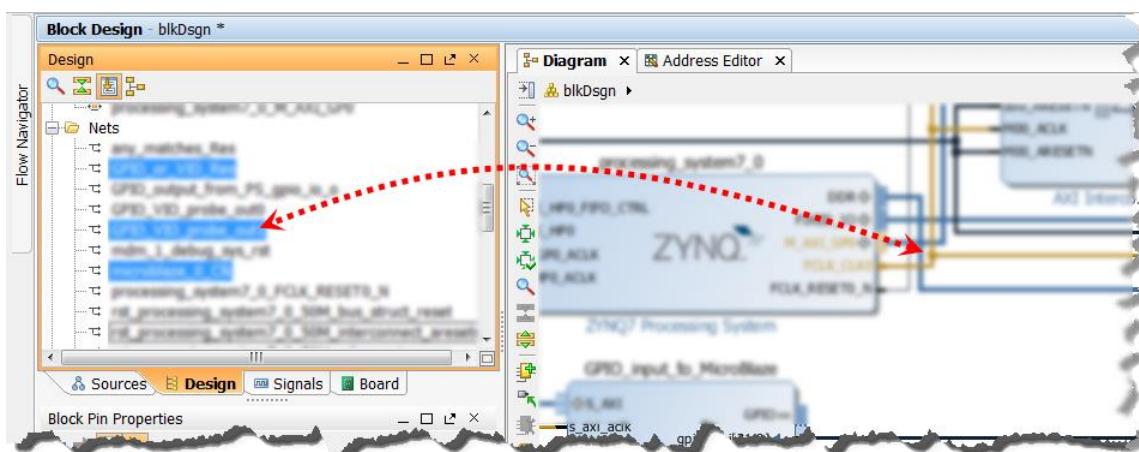


Figure 101: Selecting Nets

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

1-1-3. Select **Mark for Debug**.

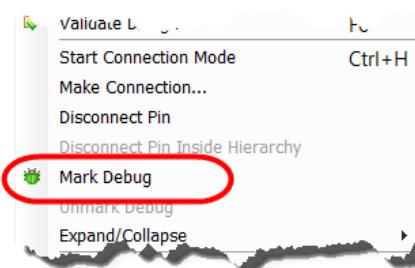


Figure 102: Marking the Nets for Debugging

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.

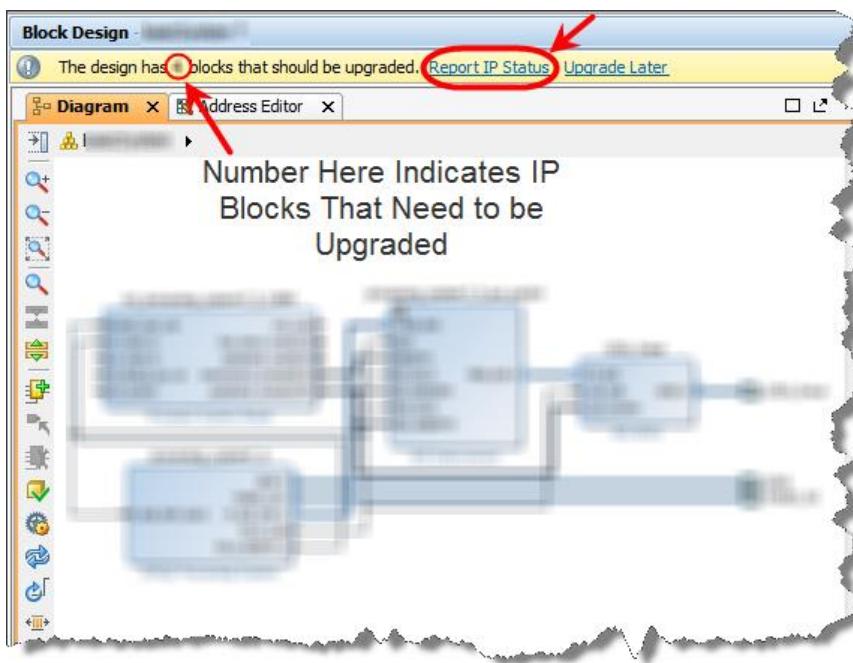


Figure 103: Report IP Status Command in the Block Design

- 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.

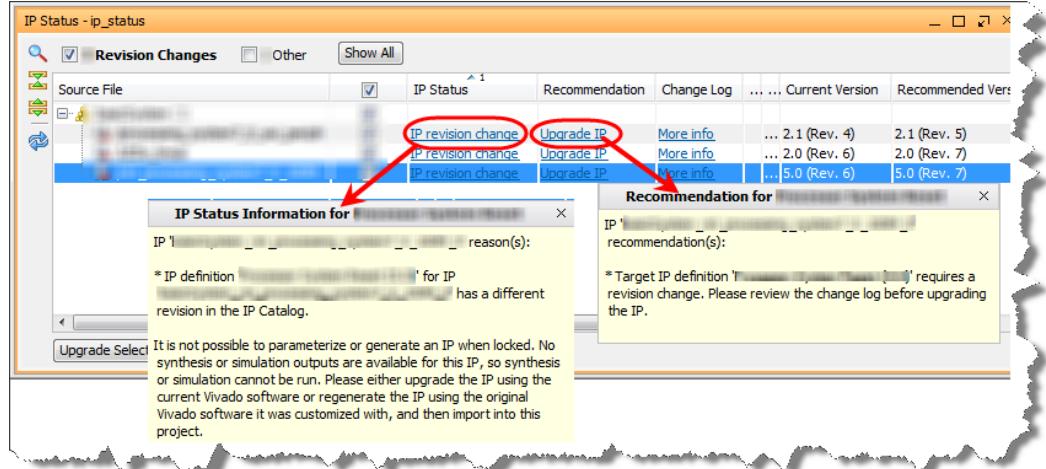


Figure 104: IP Status and Recommendation Columns IP Status Report

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).

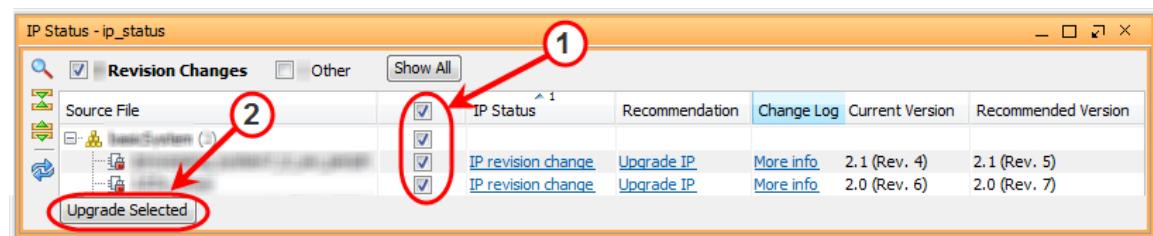


Figure 105: IP Status Report

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

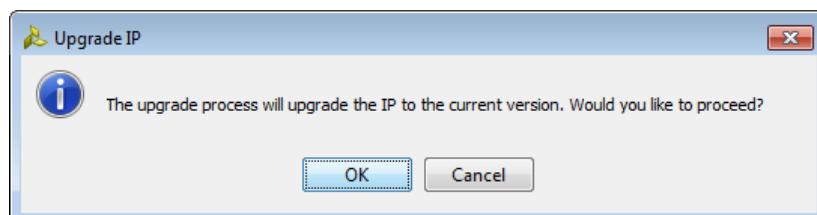


Figure 106: Upgrade IP Dialog Box

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

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**.

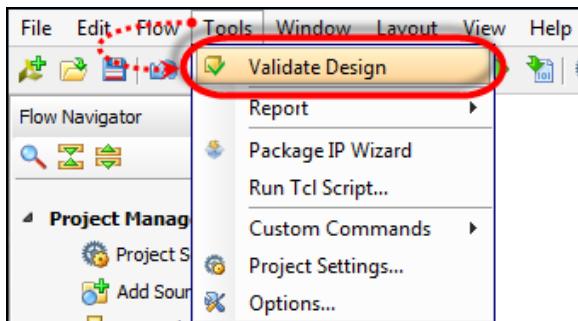


Figure 107: Validating the Design

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.

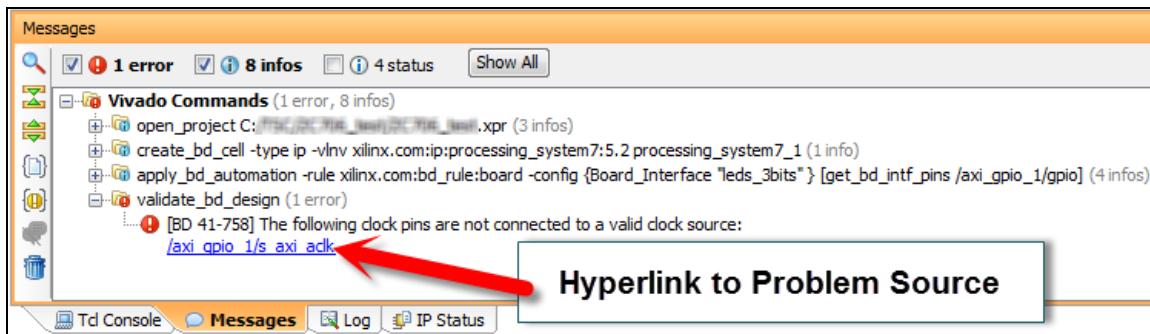


Figure 108: Viewing the Results of the Design Validation

- 1-1-5. Correct the issue(s) and rerun validation.

Validating the Block Design

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

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

Any issues are displayed in the Console window.

- 1-1-2. Look for any errors—there should be none.

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

Vivado Elaborated Design Operations

In This Section

Opening the Elaborated Design.....	80
Creating a Schematic on an Elaborated Design.....	80
Running UltraFast DRC Checks on the Elaborated Design.....	80
Running DRC Checks on the Elaborated Design.....	81
Closing the Elaborated Design.....	82

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 > 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 > Elaborated Design in the Flow Navigator.

The Report DRC dialog box opens.

- 1-1-2. Observe the categories in the **Rule Decks** section.

- 1-1-3. Select **default** in the Rule Decks section.

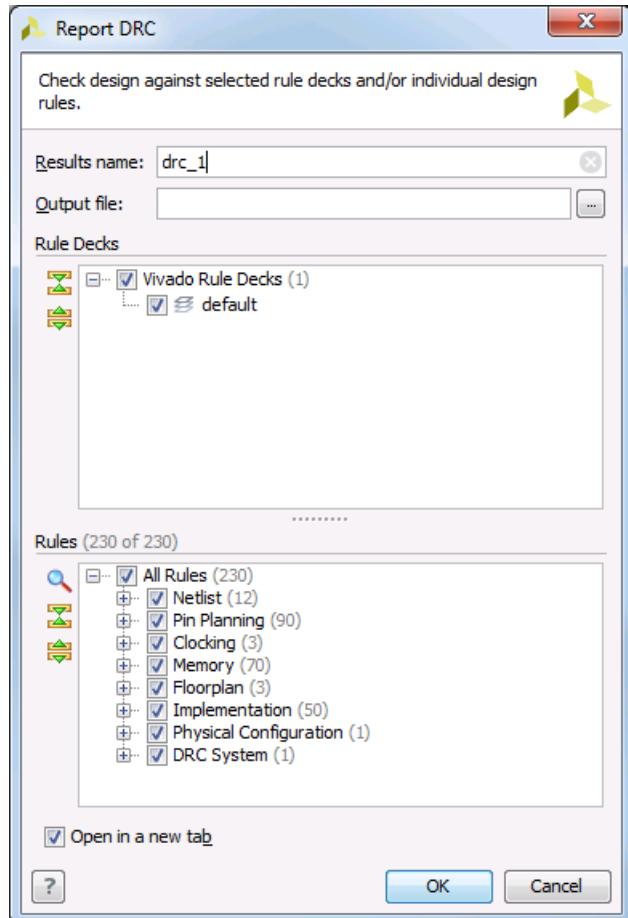


Figure 109: Running DRC Checks on Elaborated Design

- 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.

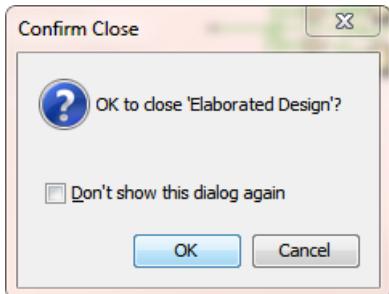


Figure 110: Confirm Close Dialog Box

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

Vivado Simulation Operations

In This Section

Running Behavioral Simulation.....	82
Running Timing Simulation.....	83
Accessing Simulation Settings	83
Closing Vivado Simulation.....	84

Running Behavioral Simulation

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

- 1-1-1. From the Flow Navigator, under Simulation, select **Run Simulation > Run Behavioral Simulation**.

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

Running Timing Simulation

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

- 1-1-1.** From the Flow Navigator, under Simulation, select **Run Simulation > Run Post-Implementation Timing Simulation**.

The simulation window opens and 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.** In the Flow Navigator, under Simulation, select **Simulation Settings**.

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

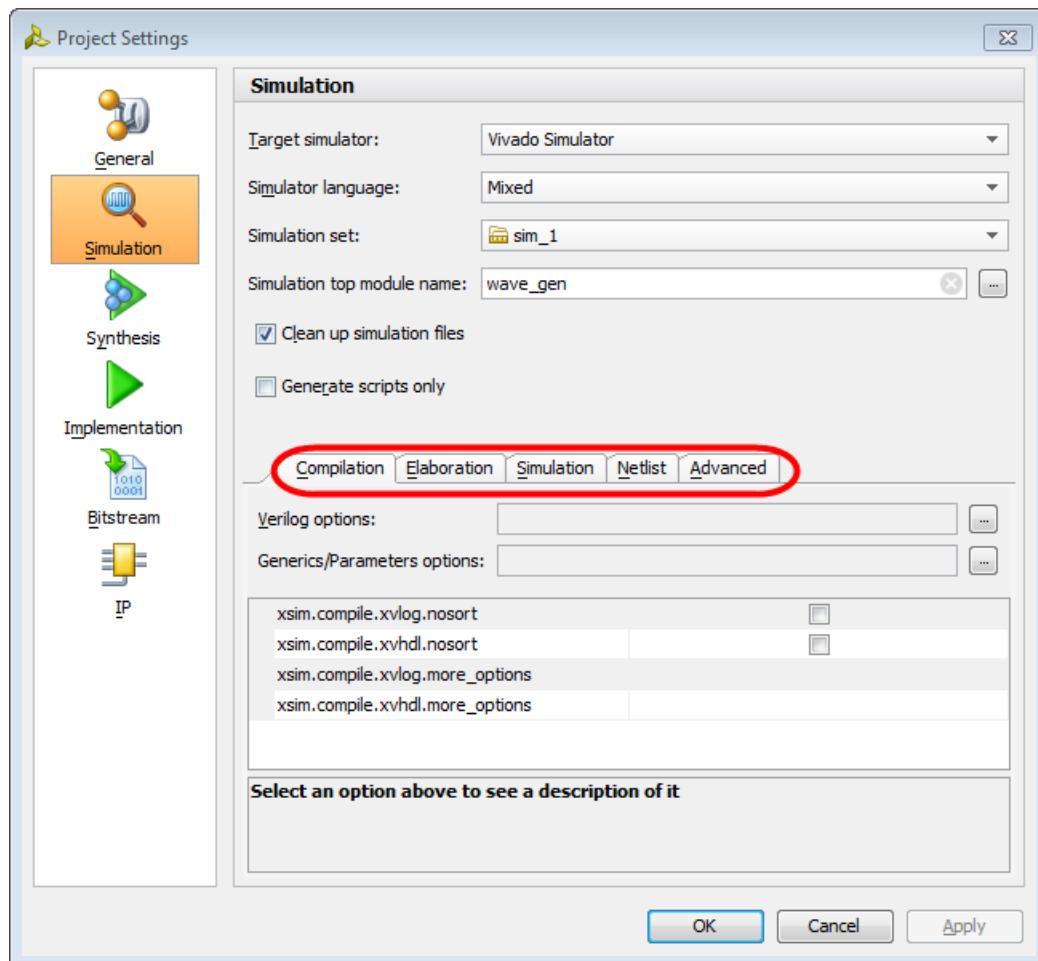


Figure 111: Vivado Simulator Settings

Closing Vivado Simulation

1-1. Close the simulation.

- 1-1-1. In the Flow Navigator, select **Simulation**, 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.....	84
Running Synthesis.....	86
Opening the Synthesized Design.....	87
Reloading the Synthesized Design.....	88
Generating a Utilization Report.....	88
Generating a Clocks Networks Report.....	89
Running a DRC Report on the Synthesized Design	89
Opening the Timing Constraints Window after Synthesis	89
Generating a Timing Summary Report on the Synthesized Design	90
Running Simultaneous Switching Noise (SSN) Analysis.....	91
Generating a Clock Interaction Report on the Synthesized Design	91
Closing the Synthesized Design	92

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**.

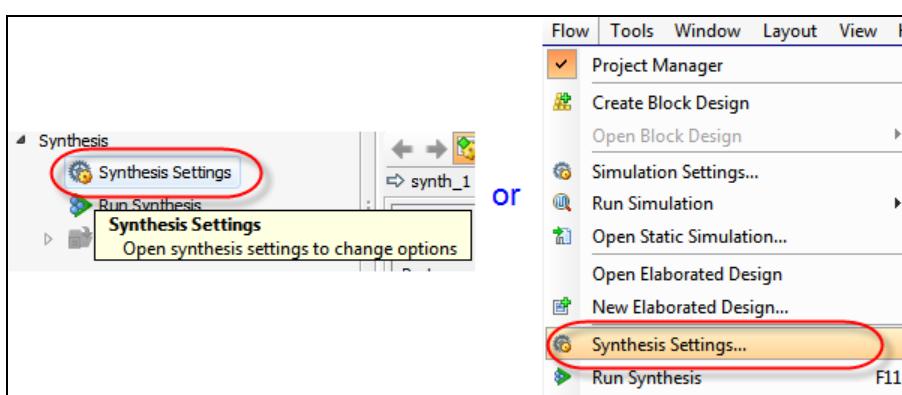


Figure 112: Selecting Synthesis Settings

The Synthesis Project Settings dialog box opens.

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

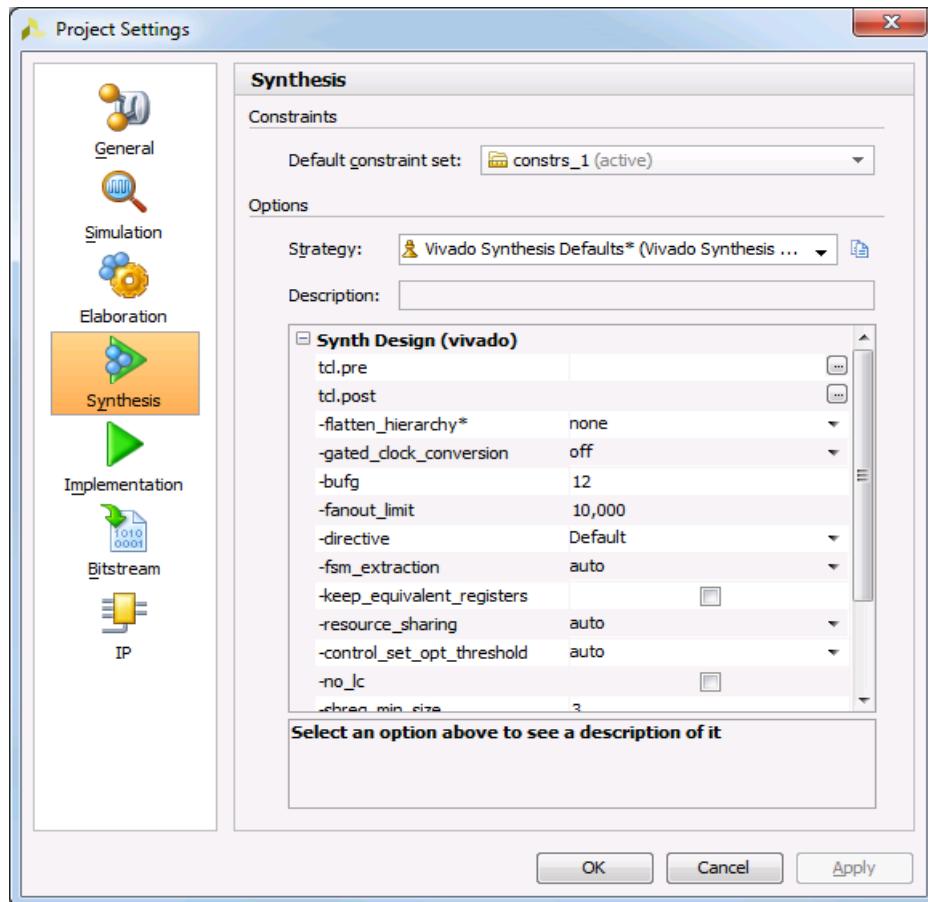


Figure 113: Synthesis Project Setting Dialog Box

- 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.

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

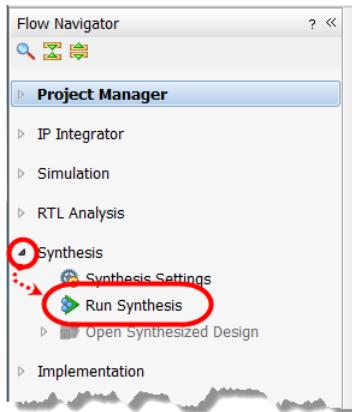


Figure 114: Selecting Run Synthesis

- 1-1-2.** 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.

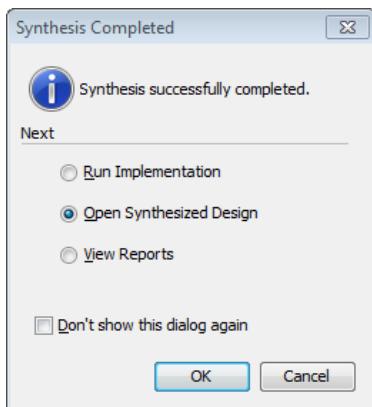


Figure 115: Synthesis Completed Dialog Box

- 1-1-3.** Select whichever option best suits your needs.

Remember that any of these choices can be accessed from other places.

Note: If you do not want to do any of these operations options, you can click **Cancel**. This will not undo the synthesis results.

- 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**.

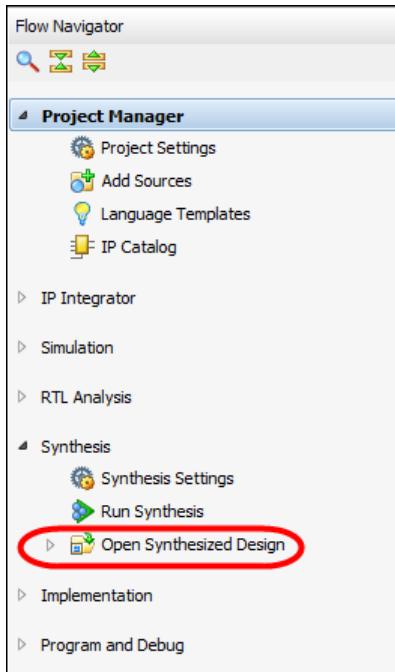


Figure 116: Opening the Synthesized Design

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.

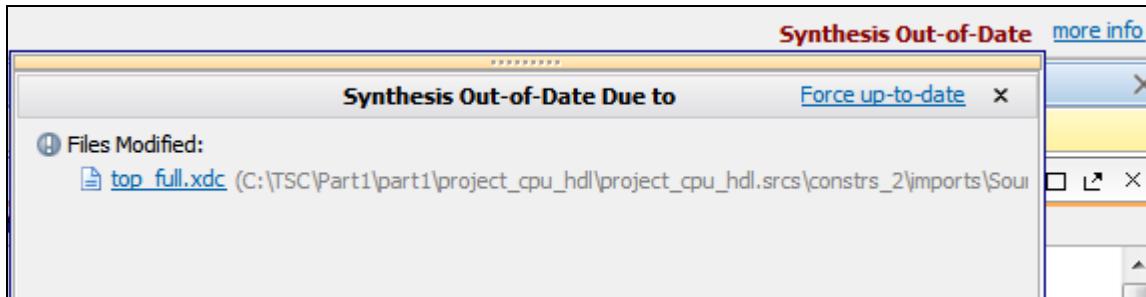


Figure 117: Force-up-to-date Link

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

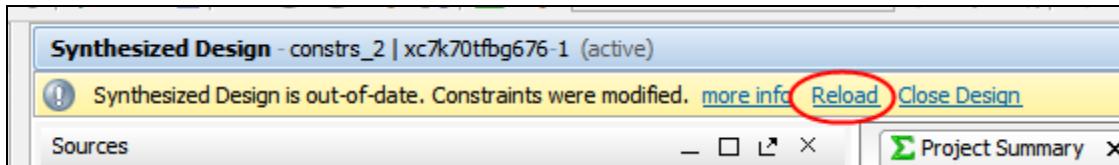


Figure 118: Reload Link

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

Generating a Utilization Report

1-1. Generate a Utilization report.

- 1-1-1. In the Flow Navigator, under Synthesized Design, click **Report Utilization**.

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

Generating a Clocks Networks Report

1-1. Generate a Clocks Networks report.

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

Alternatively, you can select **Tools > Report > 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 > Synthesized Design** in the Flow Navigator.

The Report Timing Summary dialog box opens.

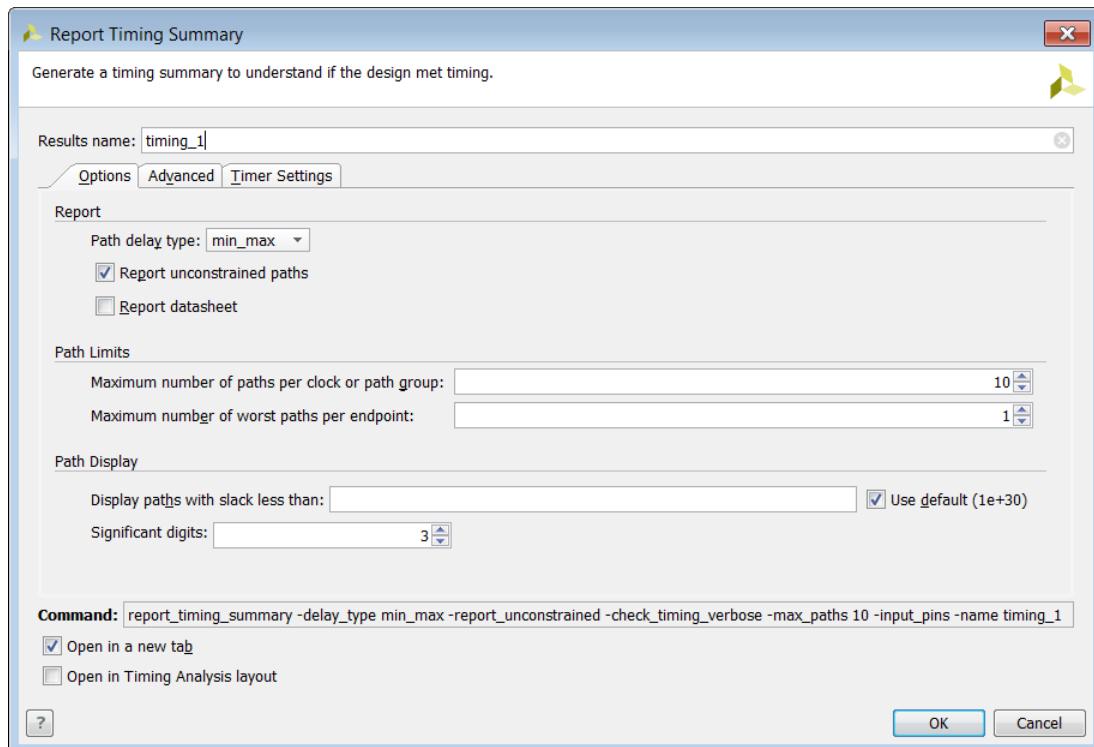


Figure 119: Report Timing Summary Dialog Box

- 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 120: 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	93
Running Implementation	94
Running All Tools – RTL to Bitstream	95
Generating a Utilization Report on the Implemented Design	97
Opening the Implemented Design.....	98
Generating a Timing Summary Report on the Implemented Design	99
Generating a Clock Interaction Report	100
Closing the Implemented Design	100
Generating Clock Networks	101
Generating a Power Report on an Implemented Design.....	101

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**.

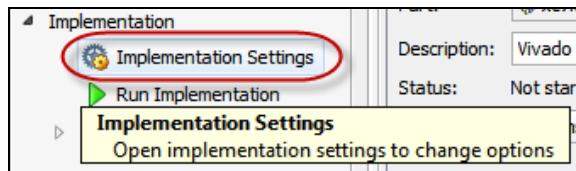


Figure 121: Clicking Implementation Settings

The Implementation Settings dialog box opens.

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

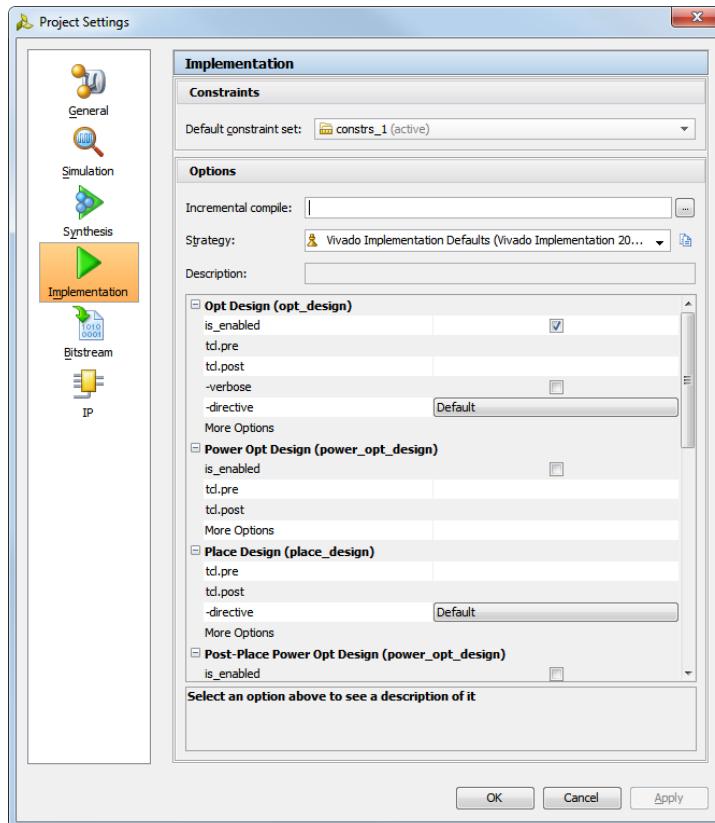


Figure 122: Implementation Settings Dialog Box

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**.

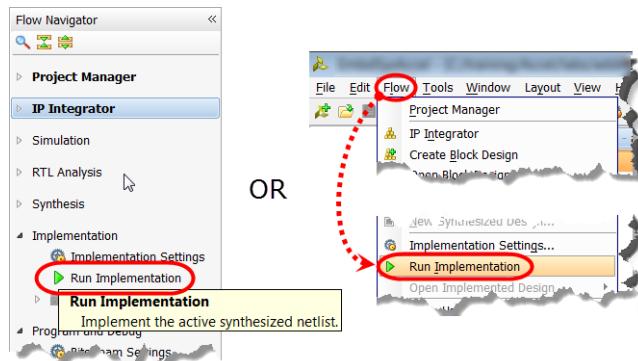


Figure 123: Selecting Run Implementation

Note: If needed, click **Yes** 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.

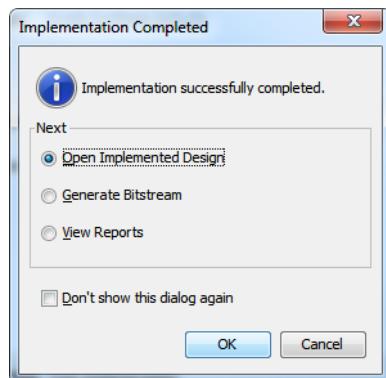


Figure 124: Implementation Completed Dialog Box

- 1-1-2.** Select **Open Implemented Design**.

- 1-1-3.** 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 will 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**.

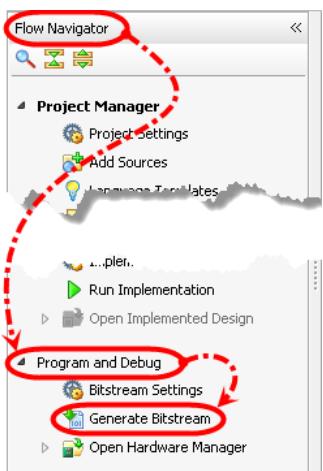


Figure 125: Generating the Bitstream

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

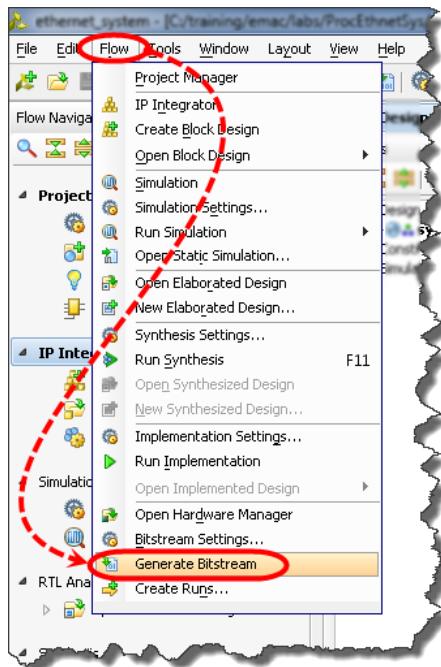


Figure 126: Generating the Bitstream

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.

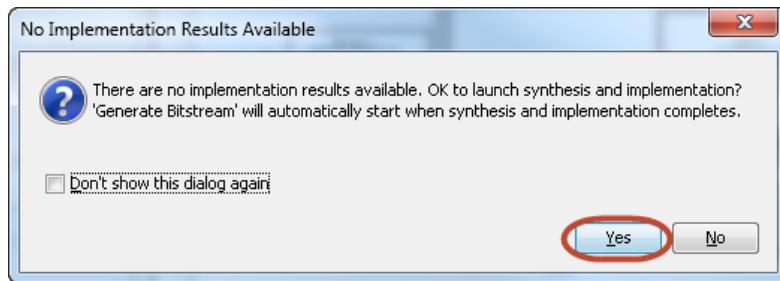


Figure 127: No Implementation Results Available Dialog Box

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 128: 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 **Tools > Report > Report Utilization**.

The Report Utilization dialog box opens.

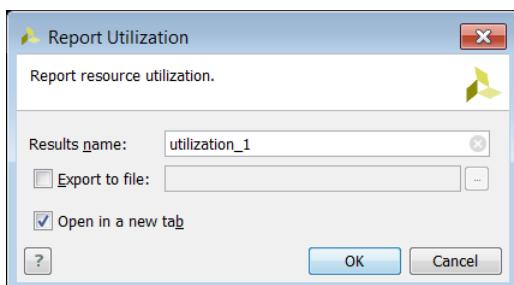


Figure 129: Report Utilization Dialog Box

- 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**.

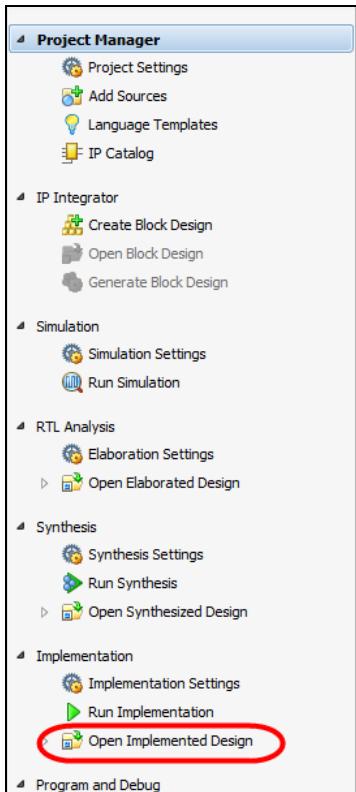


Figure 130: Opening the Implemented Design

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.

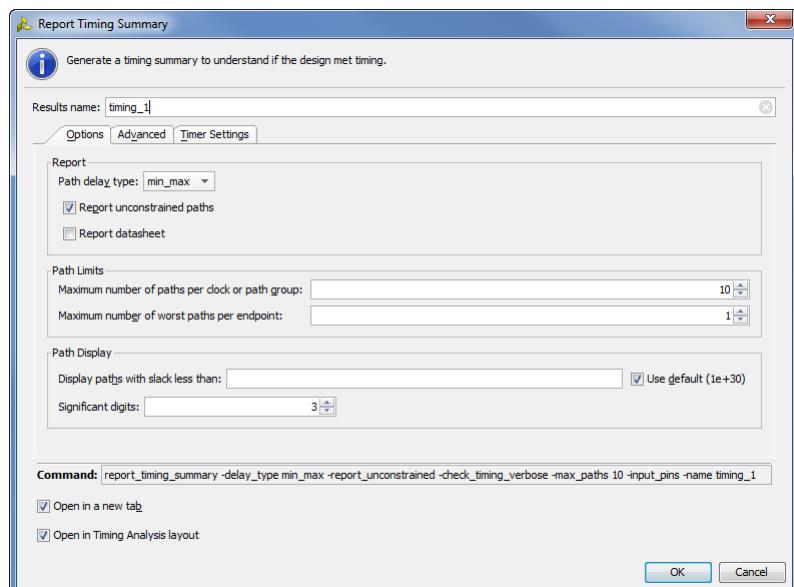


Figure 131: Report Timing Summary Dialog Box

- 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.

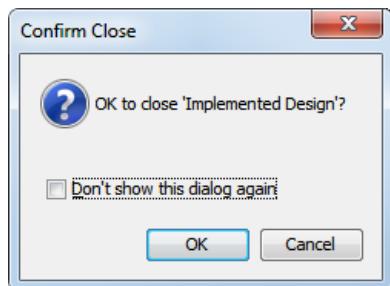


Figure 132: Confirm Close Dialog Box

- 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.

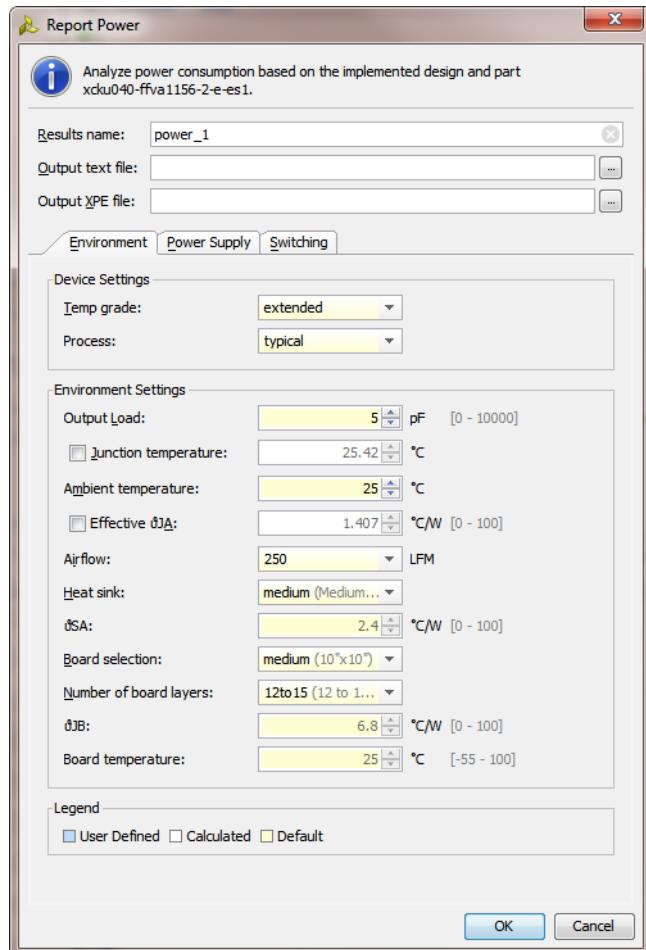


Figure 133: Report Power Dialog Box

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

Vivado Tcl Operations

In This Section

Running a Tcl Script from within the Vivado Design Suite	102
Clearing the Tcl Console.....	104
Generating a Timing Summary Report with the Tcl Interface.....	104
Using the llength Command to Obtain the Number of Paths Between Two Domains ...	105
Using the join Command to Join All Timing Paths Between Two Domains	106
Building a Custom Timing Report with the Tcl Interface	107
Generating a List of High Fanout Nets in the Design	108
Building a Custom Timing Report.....	109

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:

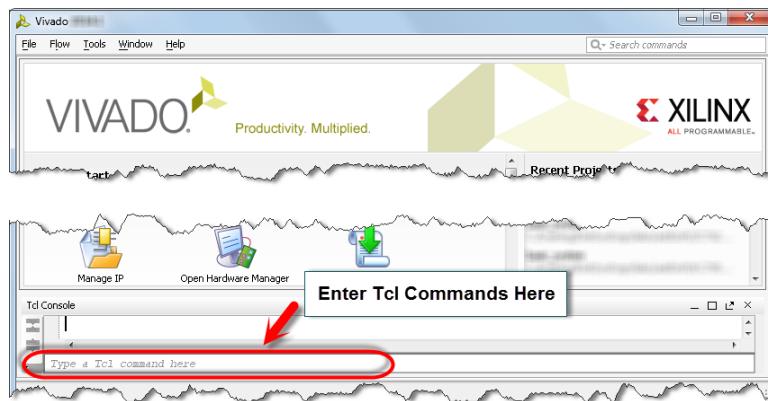


Figure 134: Accessing the Tcl Console from the Getting Started Page

From an opened project:

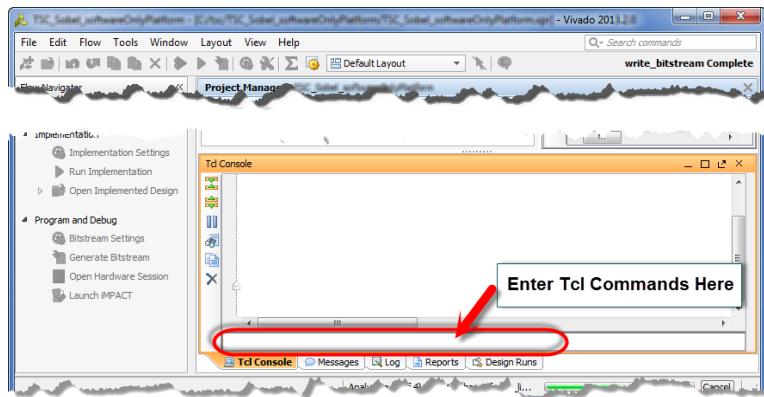


Figure 135: Entering Commands into the Tcl Console from an Open Project

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 where the Tcl script is located by entering:

```
cd the location of the Tcl file to open
```

Remember that the Tcl environment is based on Linux and requires the '/' character to delimit hierarchical paths.

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

```
pwd
```

The current working directory is displayed. If you are not where you want to be, use the `cd` command to change to the location of the Tcl file to open.

- 1-1-4.** Enter the following Tcl command:

```
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**.

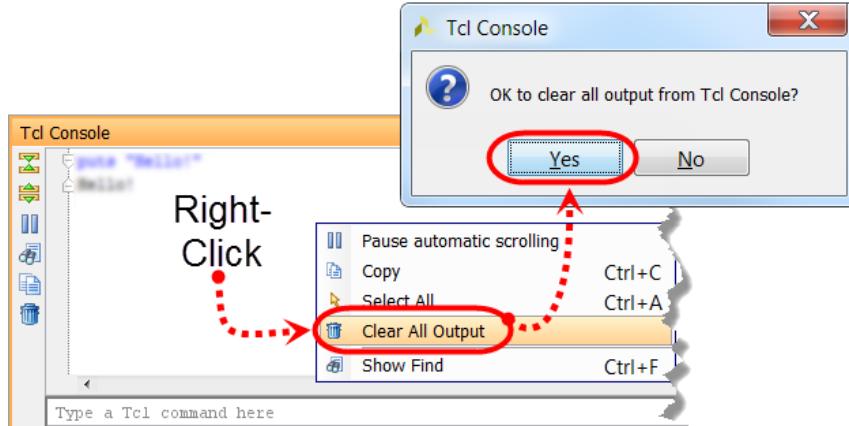


Figure 136: Clearing the Tcl Console

- 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:

- o -max_nets = 10 → Number of nets to report in the report.
- o -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.
- o -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.

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/O4[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

Figure 137: 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.

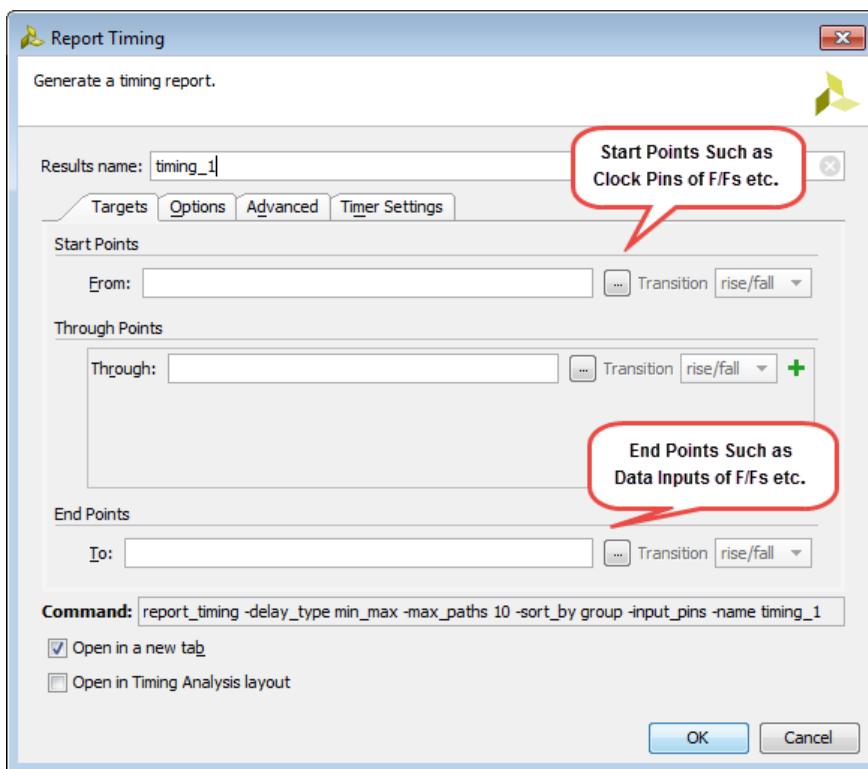


Figure 138: Report Timing Dialog Box: Targets Tab

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	110
Identifying the Failing Paths from a Timing Report.....	112
Understanding the Path Detail Info from a Timing Path Report	112
Generating a Custom Schematic to Display Selected Number of Failing Timing Paths..	115

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.

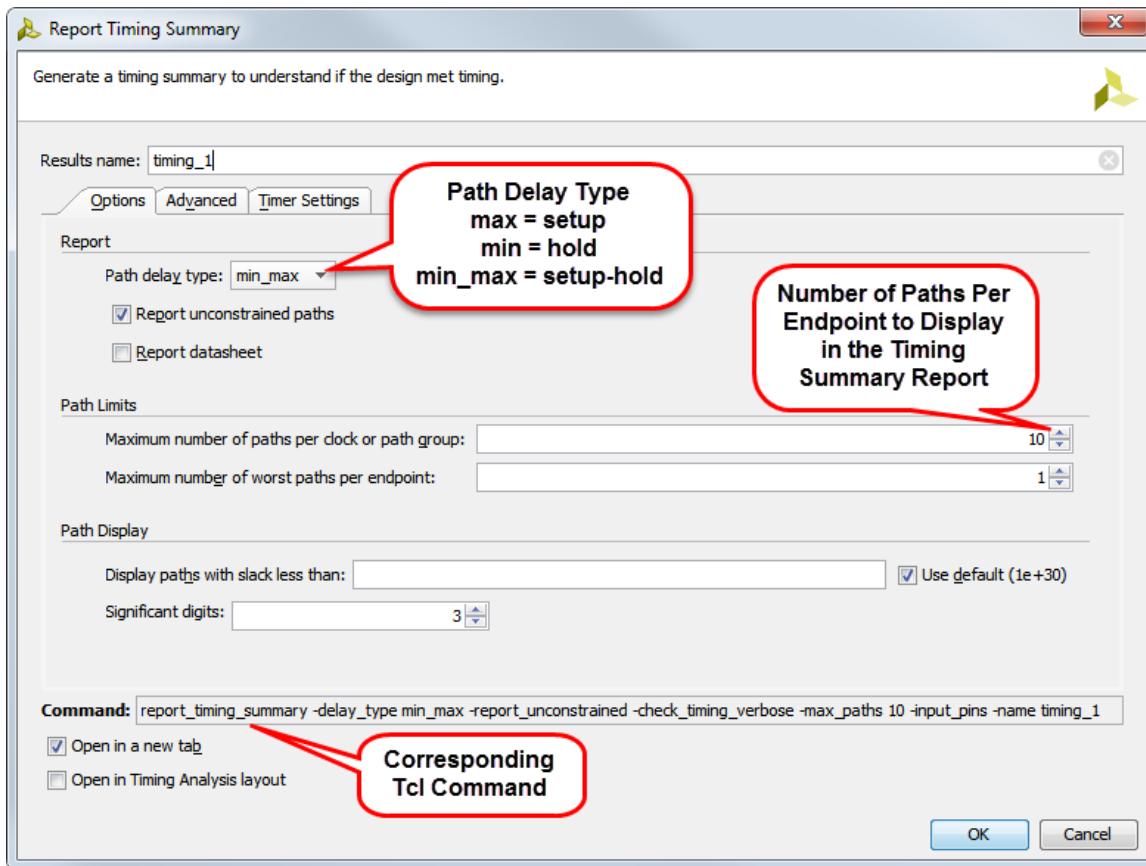


Figure 139: Timing Summary Report Dialog Box: Options Tab

- 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.

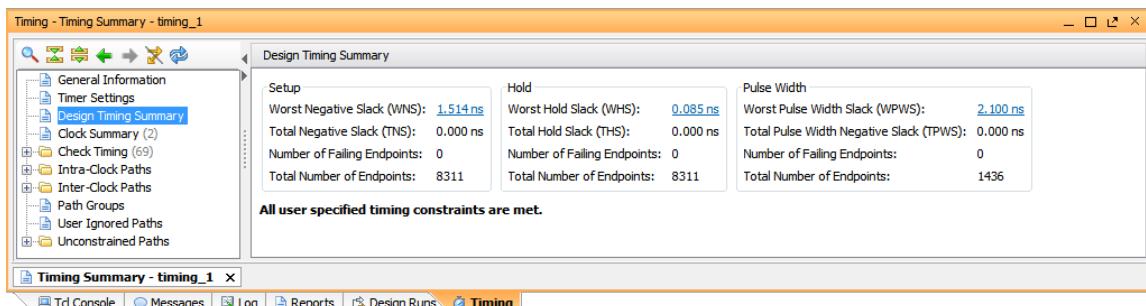


Figure 140: Design Timing Summary Window

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.

- 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.

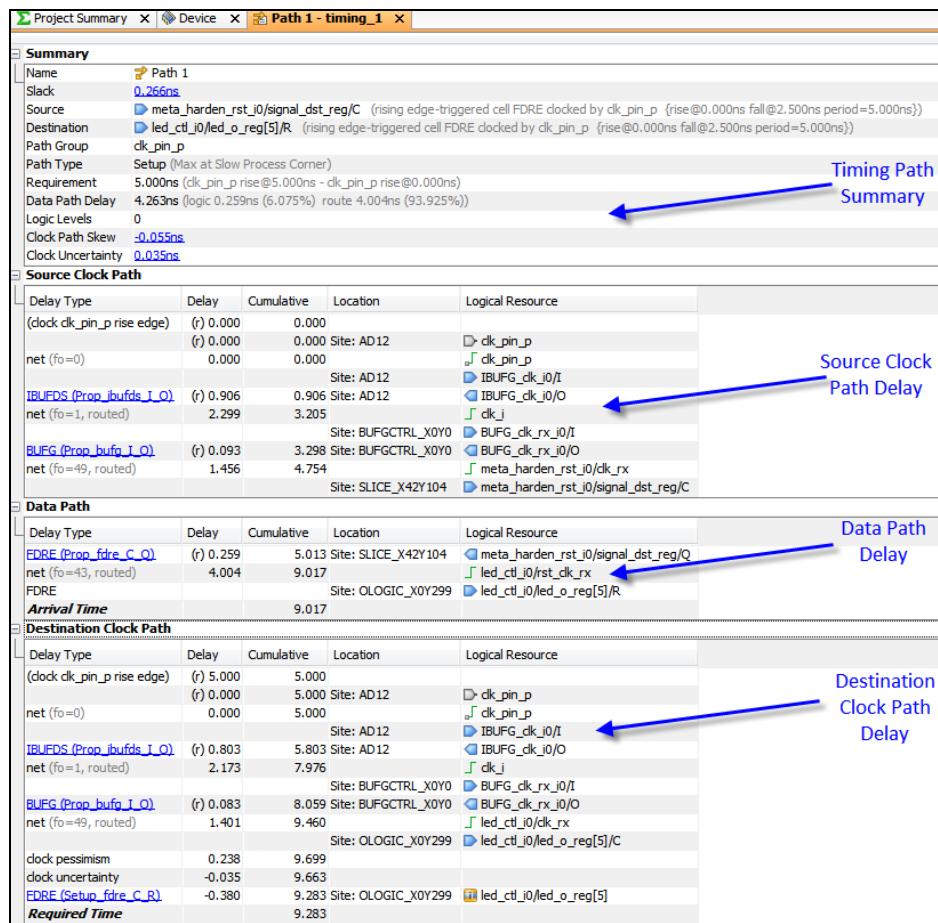


Figure 141: Path Detail Information of a Selected Path

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.

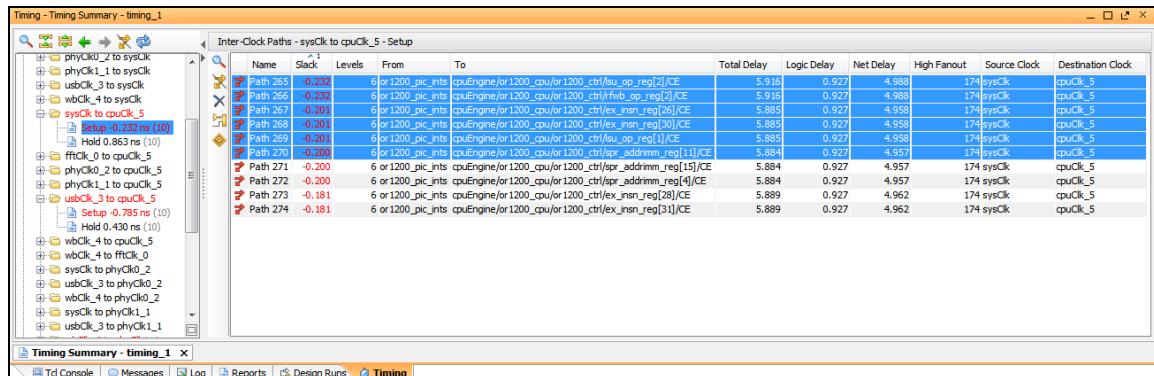


Figure 142: Selecting Multiple Failing Timing Paths

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

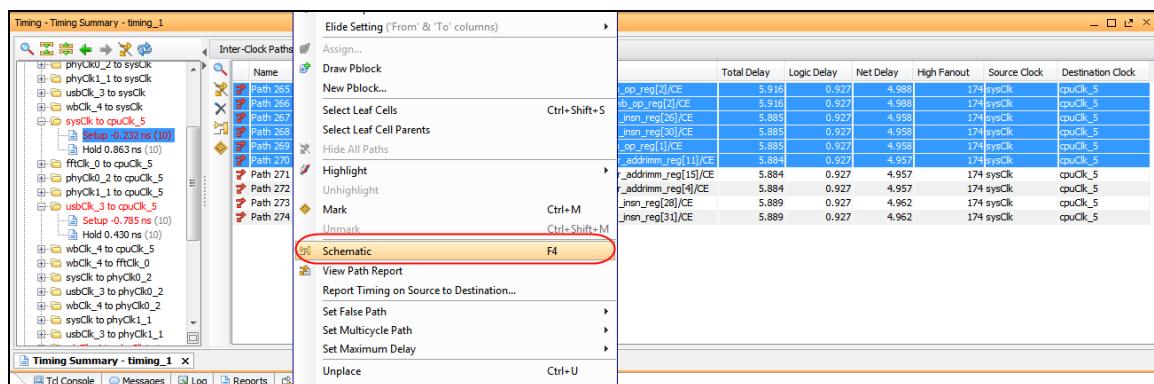


Figure 143: Selecting the Schematic Option for the Selected Paths

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

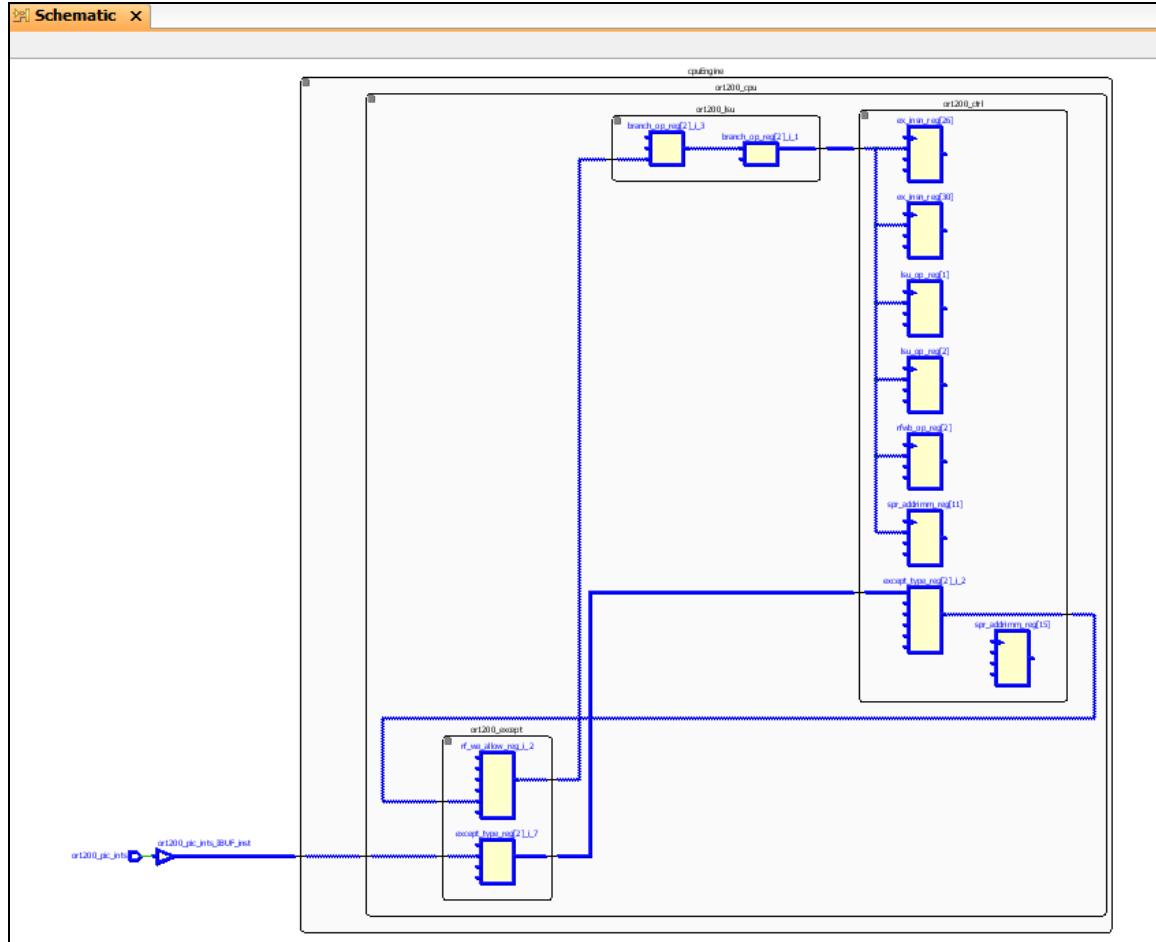


Figure 144: Schematic Tab in the Main Workspace Area

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	117
Opening the Hardware Manager	118
Downloading a Bitstream Using the Hardware Manager	120
Generating the Bitstream and Downloading the Design	121
Generating the Bitstream	124
Launching and Configuring the Tera Term Terminal Program in Serial Port Mode	124

Implementing a Design and Generating a Bitstream

1-1. Generate the implemented design and bitstream.

- 1-1-1.** From the Flow Navigator, under Implementation, click **Generate Bitstream** to run all operations necessary to generate the bitstream.

This includes synthesis if the netlist has not already been generated, and the implementation processes if not already performed.

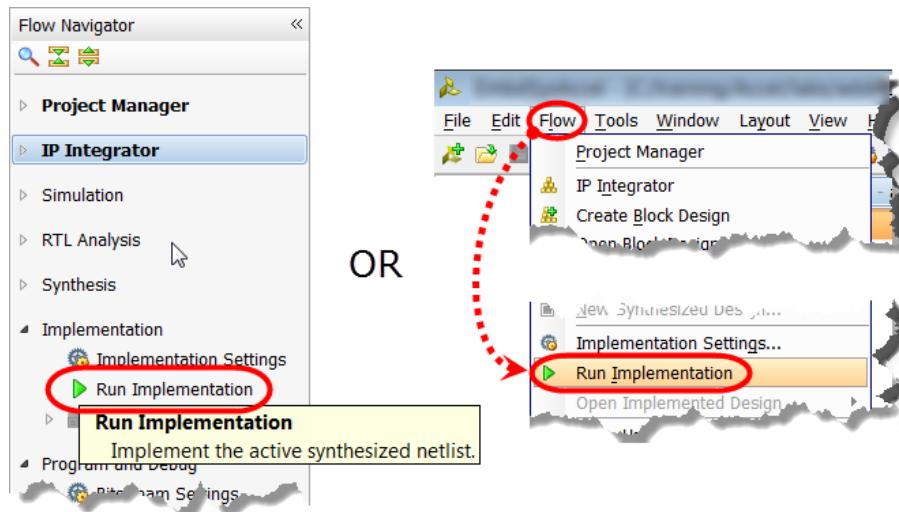


Figure 145: Two Ways to Run Implementation

- 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.

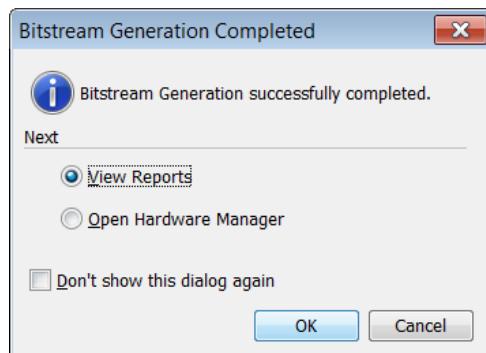


Figure 146: Bitstream Generation Completed Dialog Box

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 neither of these tasks and complete the bitstream generation process.

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.

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.

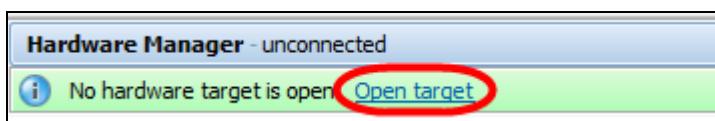


Figure 147: Opening a Hardware Target

1-2. Connect the target through the New Target Wizard that guides you through the process.

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



Figure 148: Open Hardware Target Dialog Box

1-2-2. Click **Next**.

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

Typically this is left at its default value.

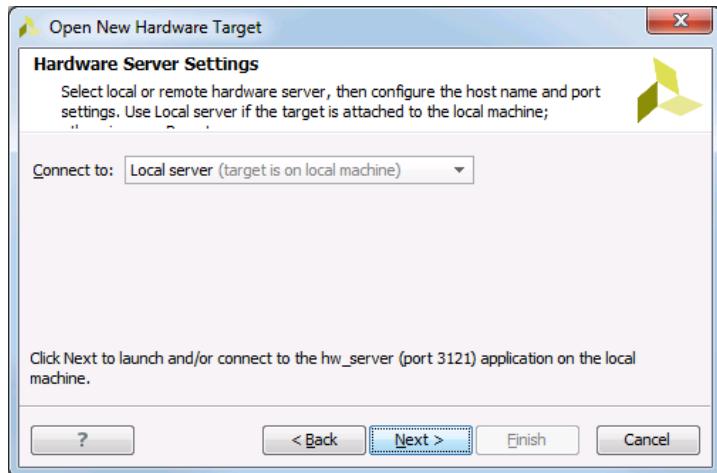


Figure 149: Setting the Server Name for the New Hardware Target

1-2-4. Click **Next**.

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.

1-2-6. Leave the frequency at its default value.

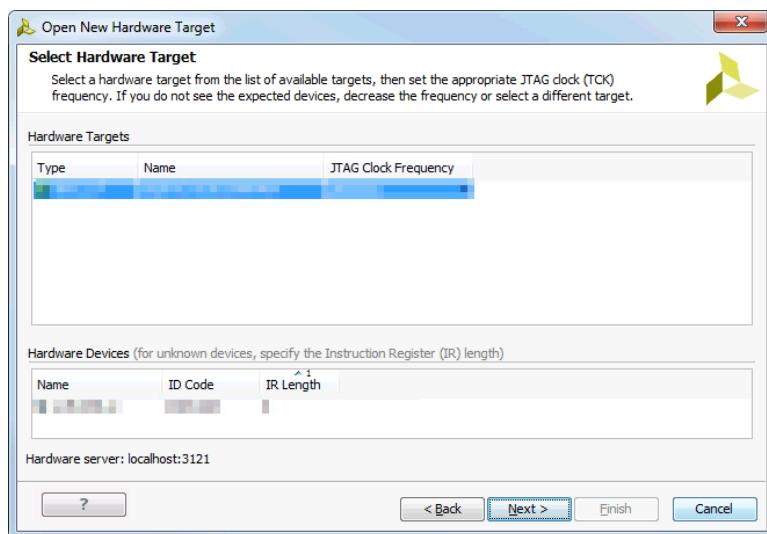


Figure 150: Selecting the Hardware Target

1-2-7. Click **Next**.

A summary of the connection is displayed.



Figure 151: Summary of the Open Hardware Target Settings

1-2-8. Click **Finish**.

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**.

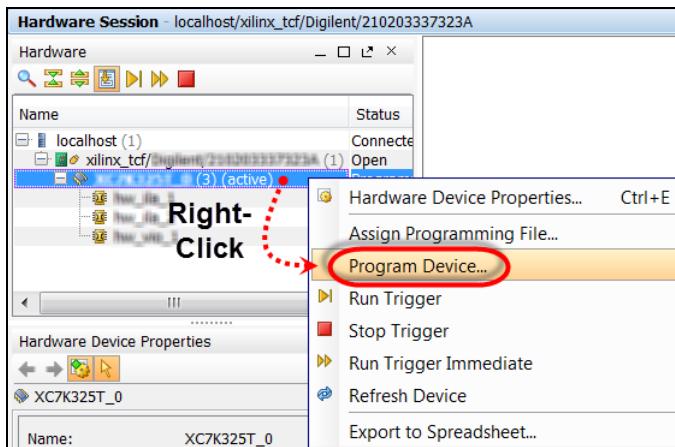


Figure 152: Selecting Program Device

- 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.

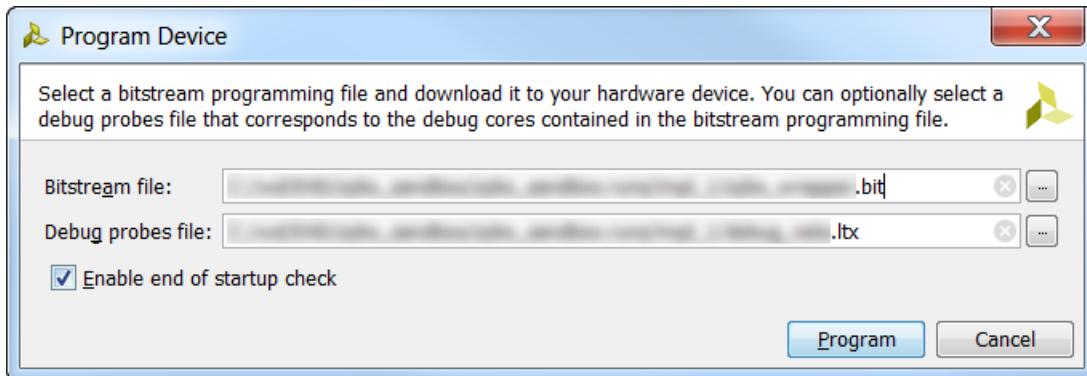


Figure 153: Selecting the Bitstream and Programming the Device

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

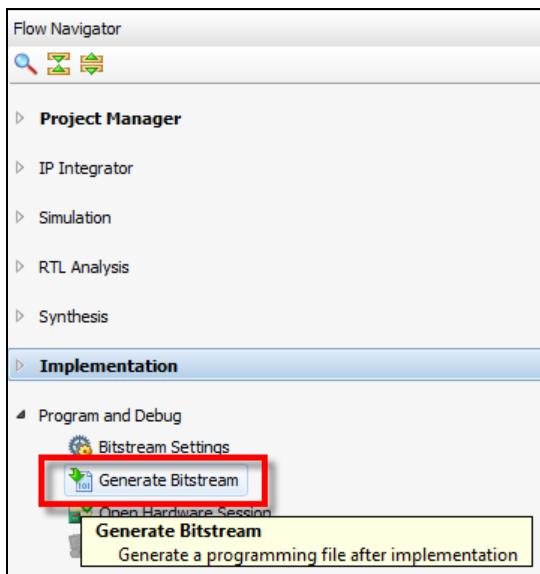


Figure 154: Generate Bitstream

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.
- 1-1-3. 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. Similarly, 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.

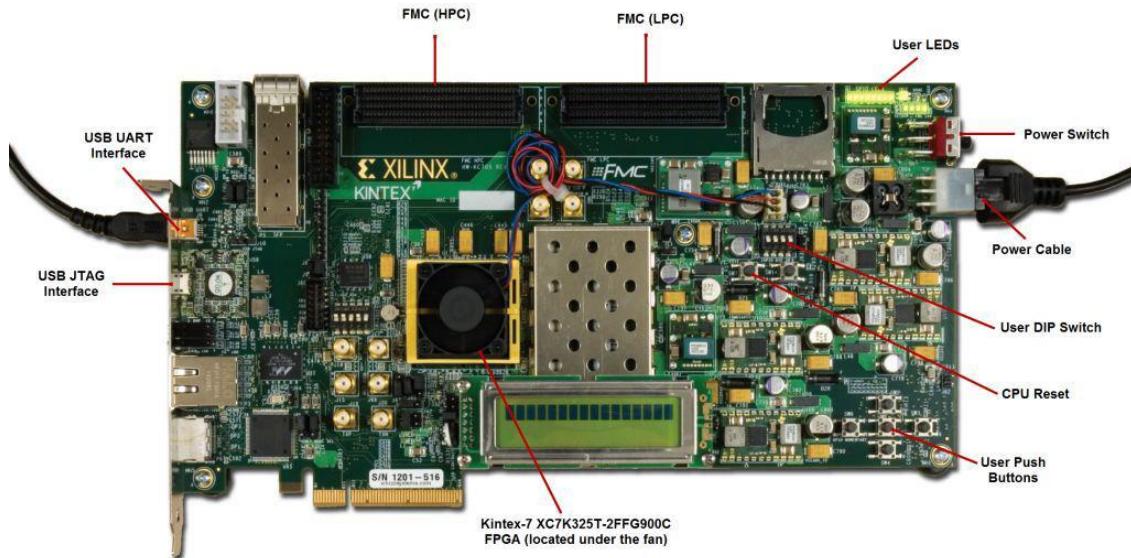


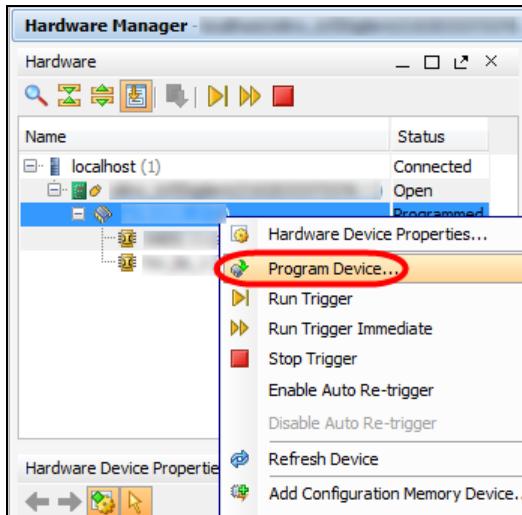
Figure 155: KC705 Evaluation Board

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-7 FPGA on the your board board using the hardware manager.

- 1-3-1. In the Flow Navigator, under Program and Debug, click **Open Hardware Manager**.
- 1-3-2. Click **Open Target > Open New Target**.
- 1-3-3. In the Open New Hardware Target dialog box, click **Next**.
- 1-3-4. In the Hardware Server Settings dialog box, click **Next**.
- 1-3-5. In the Select Hardware Target dialog box, click **Next**.
- 1-3-6. Click **Finish**.

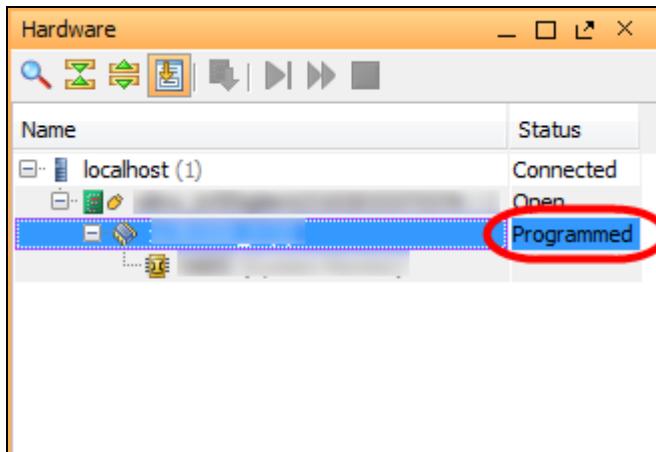
1-3-7. Right-click **xc7k325t_0** and select **Program Device**.**Figure 156: Programming the Device**

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 157: 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**.

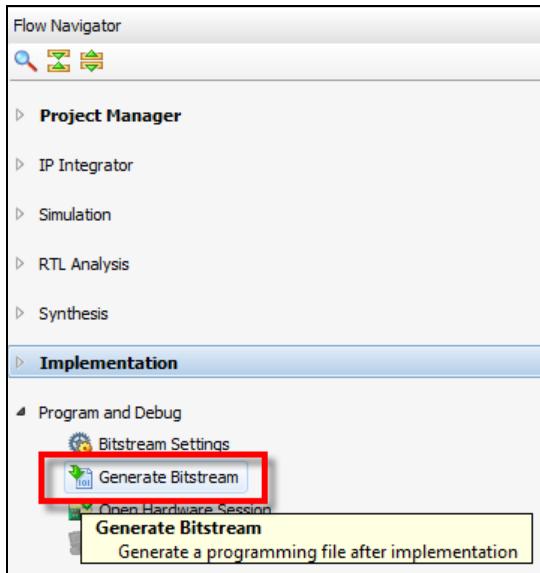


Figure 158: Generate Bitstream

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

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. It is an alternative to using the Terminal view that is built into SDK.

1-1. Launch the Tera Term terminal program.

- 1-1-1. From the Windows desktop, double-click the **Tera Term** icon to launch Tera Term.

Alternatively, you can select **Start > All Programs > Tera Term > Tera Term**.

- 1-1-2. Select **Serial** as the connection (1).

- 1-1-3. 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.

1-1-4. Select the COM # discovered in the last step (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.

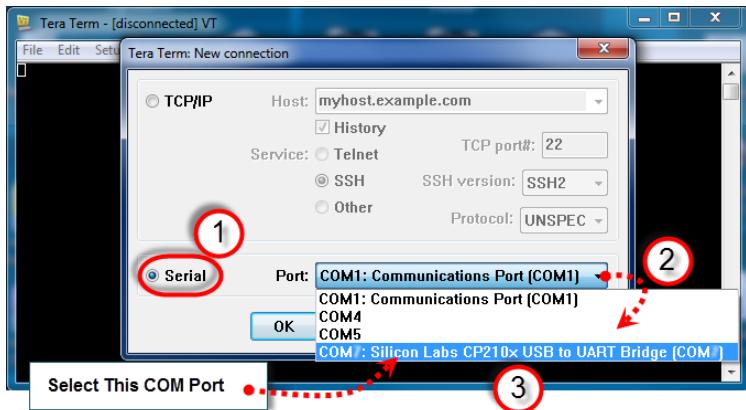


Figure 159: Selecting the COM Port

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.

1-1-5. Click **OK**.

The terminal console window opens.

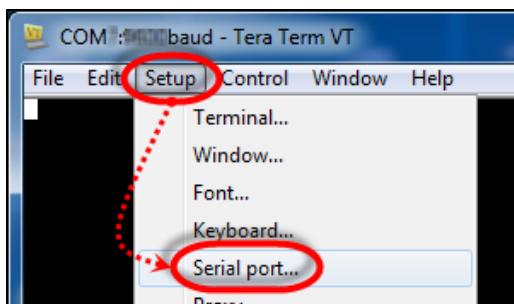
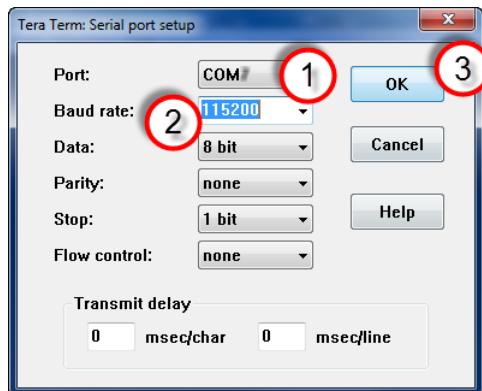
1-1-6. Select **Setup > Serial Port**.

Figure 160: Opening the Tera Term Serial Port Setup Window

The Tera Term Serial Port Setup dialog box opens.

1-1-7. Confirm that the proper serial port has been selected (1).

1-1-8. Set the baud rate to **115200** (2).**Figure 161: 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.

1-1-9. Click **OK** (3).

Tera Term is now configured to receive and transmit serial information to/from the evaluation board.

Vivado Add Sources Operations

In This Section

Adding an HDL Source File.....	127
Adding a Simulation Source File	128
Adding a Constraint File.....	130
Creating a SystemVerilog Simulation Source File.....	132
Adding Existing Constraints.....	134

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.

- Under the Flow Navigator tab, under Project Manager, select **Add Sources**.

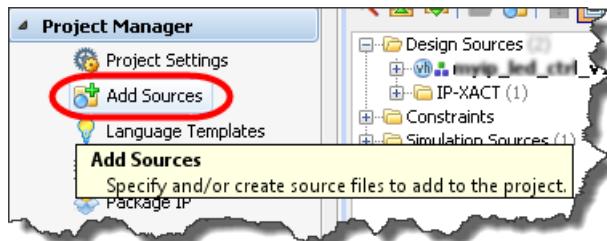


Figure 162: Selecting Add Sources

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

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



Figure 163: Selecting Add or Create Design Sources

- Click **Next** to begin selecting source files.

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.

- Click the **Plus (+)** icon and select **Add Files**.

- Browse to your source directory if it is not open already.

1-1-6. Select your HDL sources.

1-1-7. Click **OK** in the Add Source Files dialog box to select the file(s).

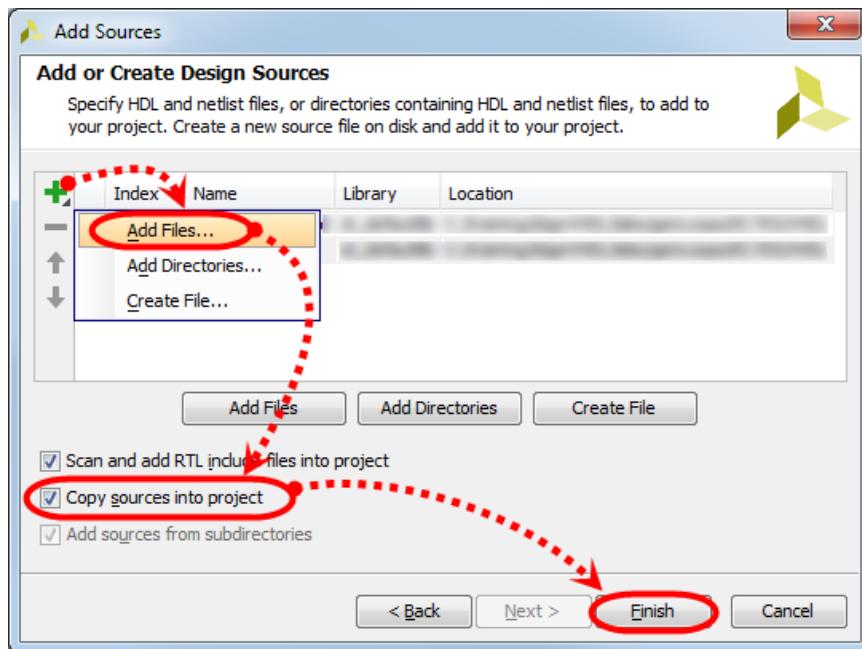


Figure 164: Selecting Add Files

1-1-8. Make sure that the **Copy sources into project** option is selected.

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.

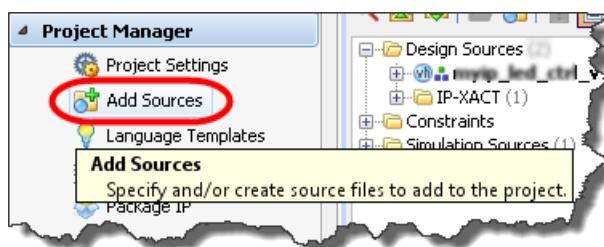


Figure 165: Selecting Add Sources

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

1-1-3. Click **Next**.**Figure 166:** Add Sources Dialog Box

- 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 **C:\training****\labs\lab name\your board\<language> or your project name** 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.** 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**.

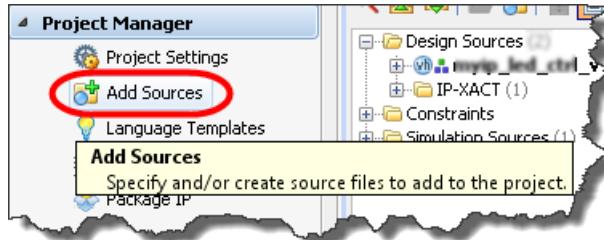
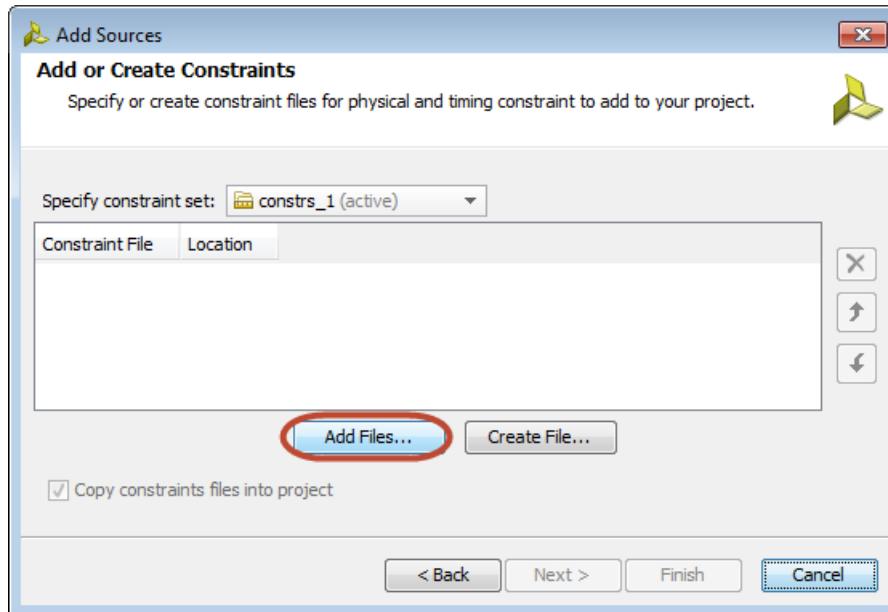


Figure 167: Selecting Add Sources

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



Figure 168: Selecting Add or Create Constraints

1-1-3. Click **Add Files**.**Figure 169:** Selecting Add Files

- 1-1-4.** Browse to C:*training*****\labs\lab name\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. In the Flow Navigator, under Project Manager, select **Add Sources**.

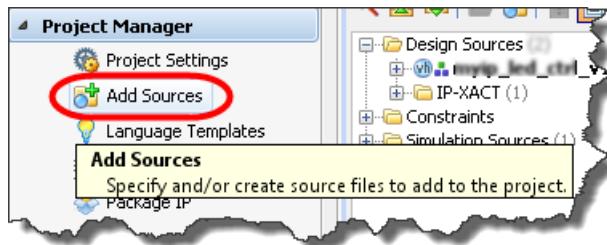


Figure 170: Selecting Add Sources

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



Figure 171: 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**.

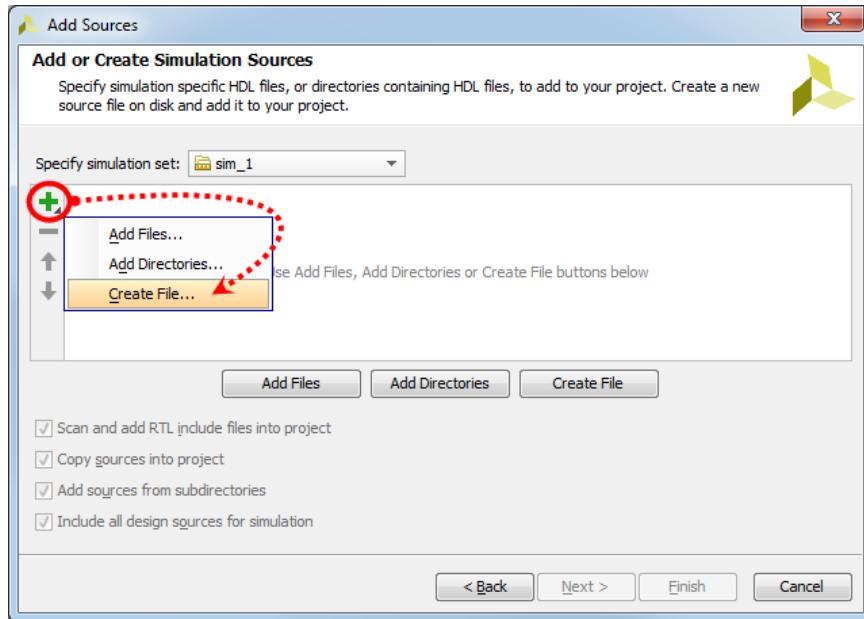


Figure 172: Selecting Create File

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.

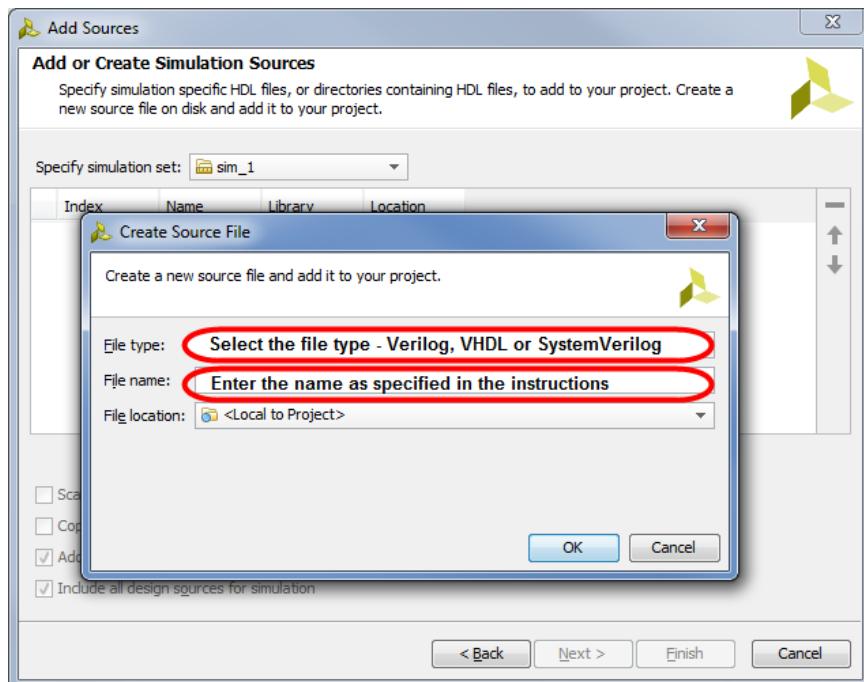


Figure 173: Entering Testbench Filename and Type

- 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. Given that this module doesn't require any ports, click **OK** to create the module without ports.

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 174: 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.

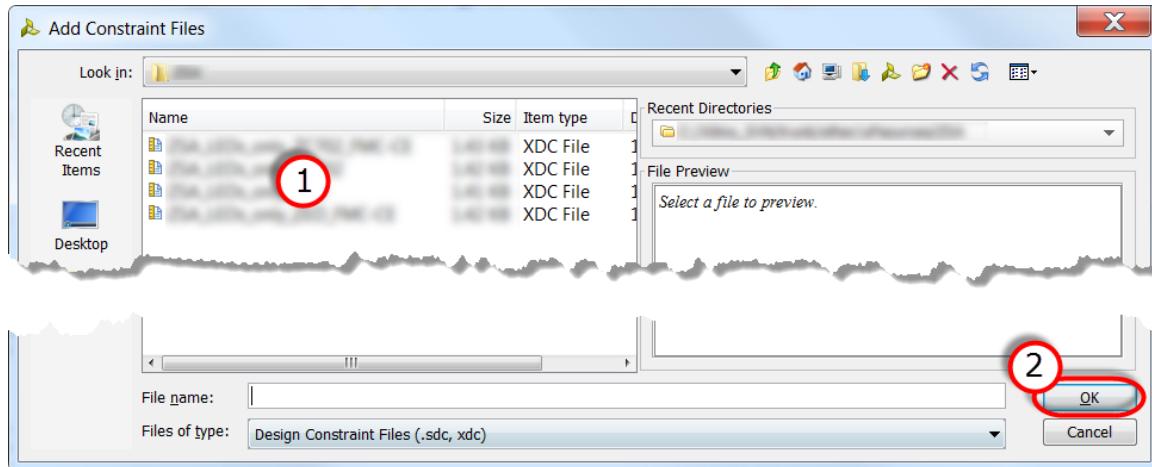


Figure 175: Selecting One or More Constraint Files

- 1-1-7.** Click **OK** to select the file.

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

Vivado Analyzer Operations

This sub-section covers many of the common activities associated with configuring IP cores and collecting data.

In This Section

Informative Information	135
Instructions	138

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.

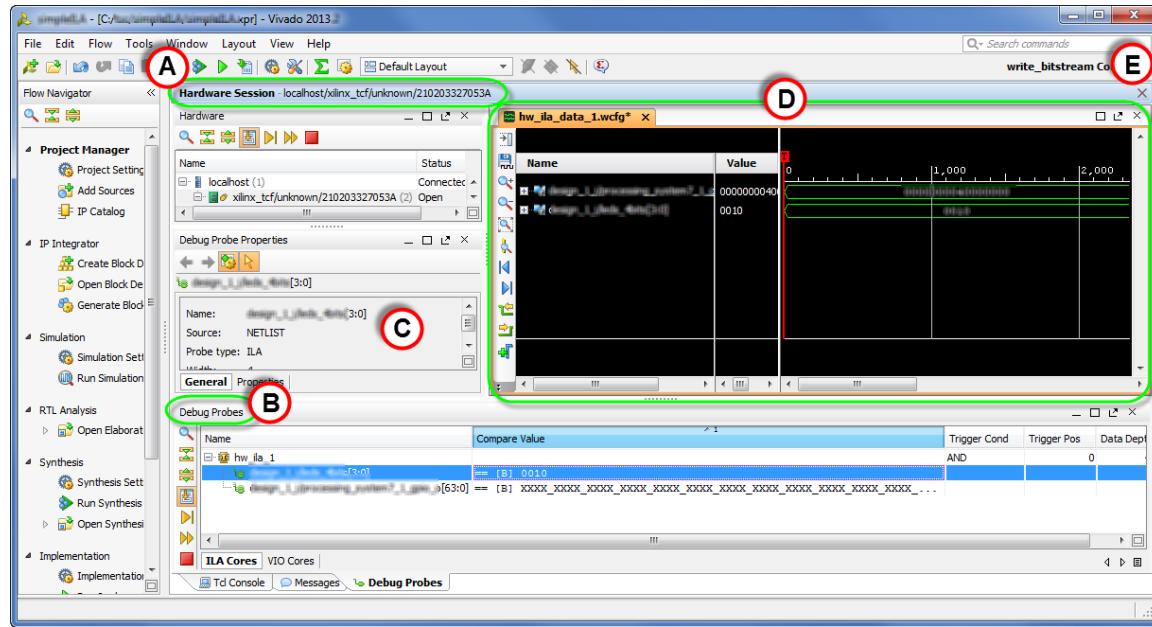


Figure 176: Vivado Analyzer - Hardware Session Screen Usage

- 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:

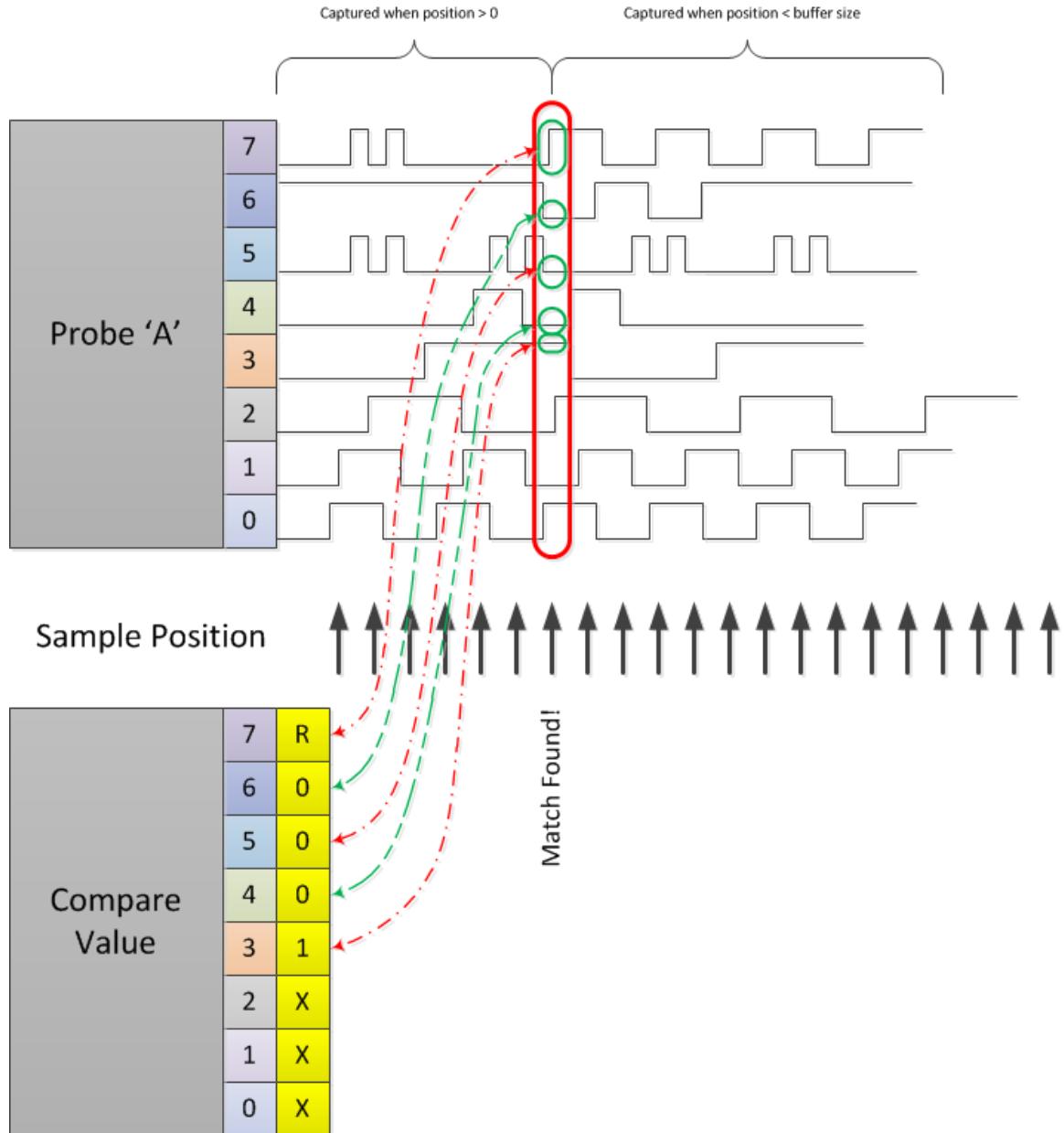


Figure 177: An Individual Probe's Compare Value

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

Opening a Vivado Analyzer Hardware Manager.....	138
Adding Probes (Signals) to the Waveform Viewer	141
Removing Probes (Signals) from the Waveform Viewer.....	142
Setting the Trigger Condition.....	142
Loading a Waveform Configuration	144
Arming the Vivado Analyzer Core	145
Triggering the ILA Immediately	146

Opening a 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. From the Flow Navigator, expand **Program and Debug**.

1-1-2. Double-click **Open Hardware Manager**.

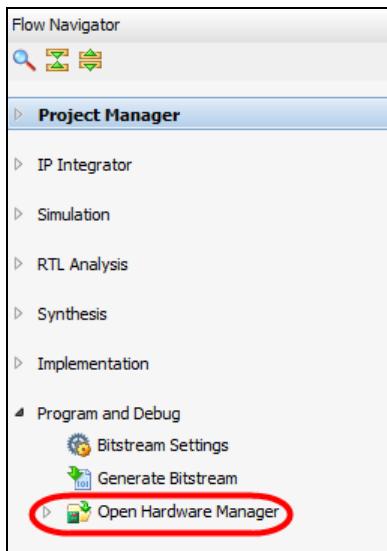


Figure 178: Opening the Hardware Manager from an Open Project

1-1-3. From the Hardware Session window, click **Open target** > **Open New Target** to open a wizard that will facilitate making a connection to the USB platform JTAG download cable.

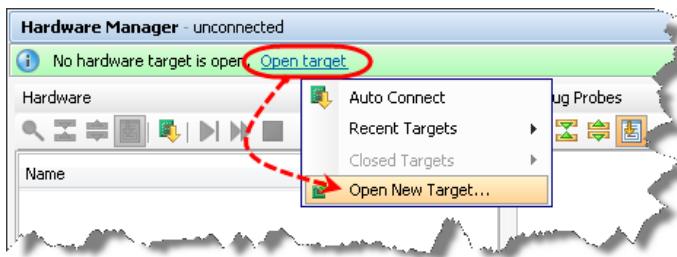


Figure 179: Selecting Open New Hardware Target

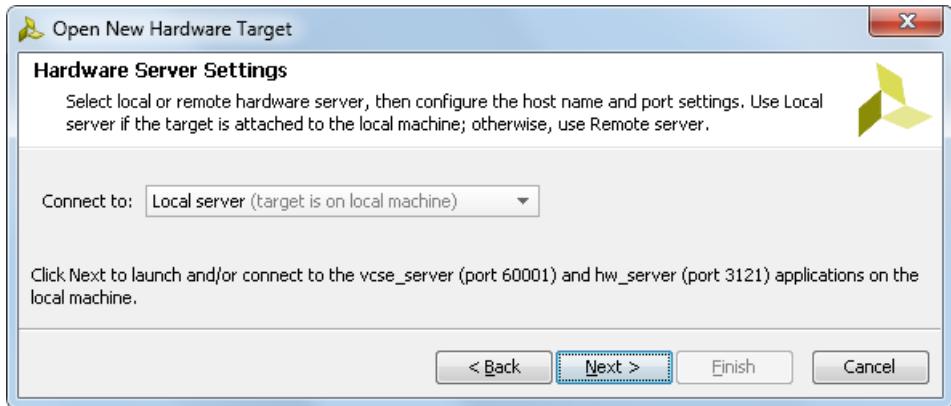
1-1-4. Click **Next** to bypass the welcome dialog box.



Figure 180: 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.



- 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.

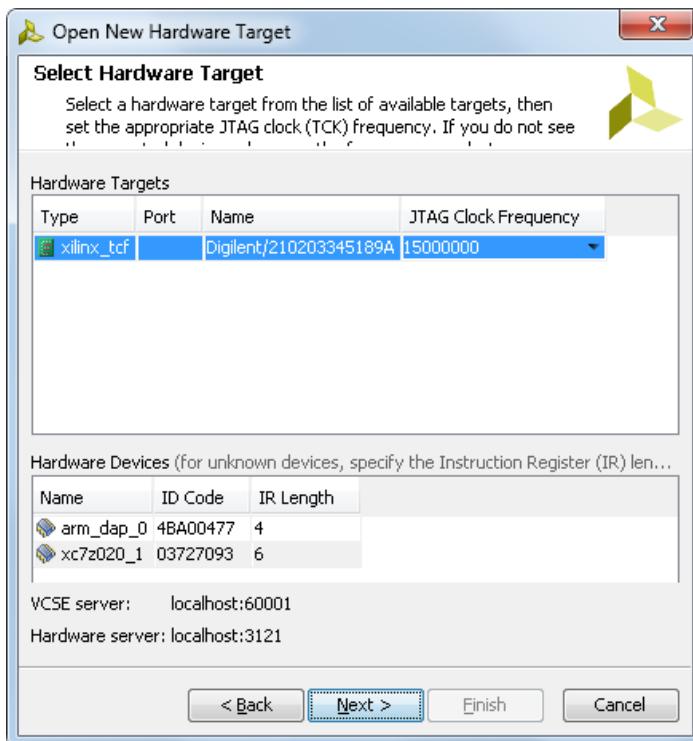


Figure 181: Viewing the Hardware Target and Devices

1-1-7. Click **Next** to advance to the summary.



Figure 182: 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

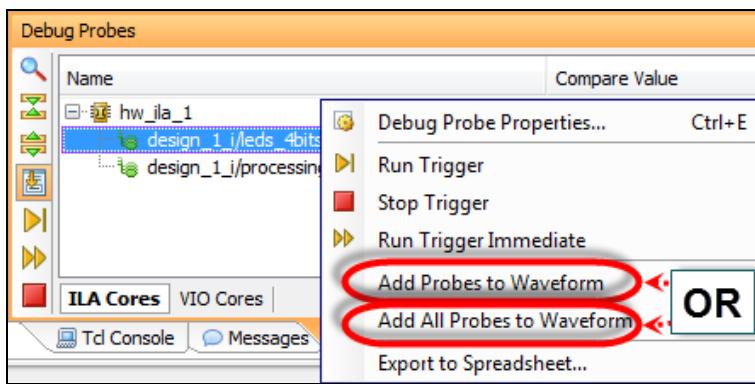


Figure 183: Adding Selected (or All) Probes to the Waveform

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**.

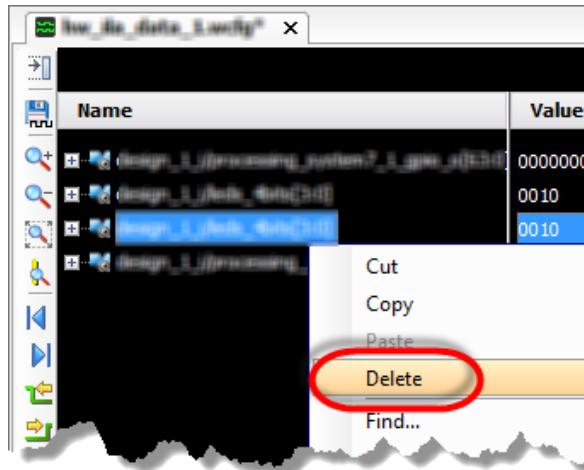


Figure 184: Deleting a Probe from the Waveform Viewer

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.

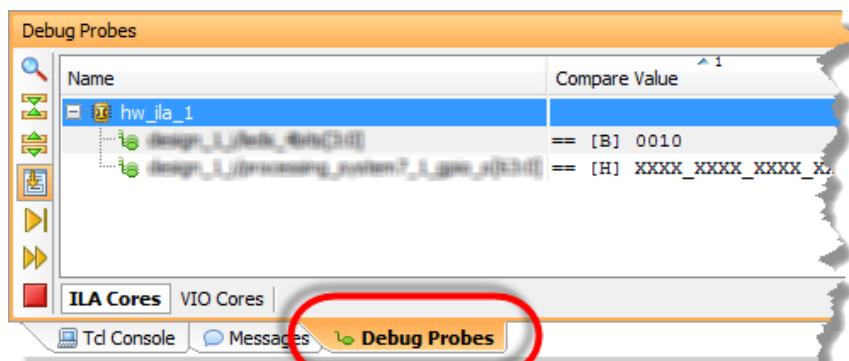


Figure 185: Locating the Debug Probes Tab

- 1-1-2. Click in the Compare Value column next to the probe that you want to configure.

A small dialog box opens.

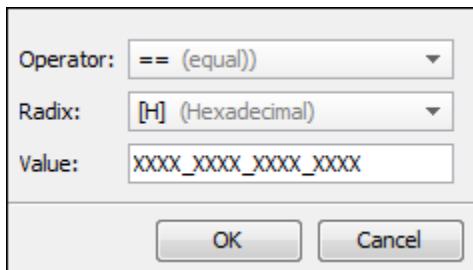


Figure 186: Compare Value Dialog Box

- 1-1-3.** Set the operator to test against.

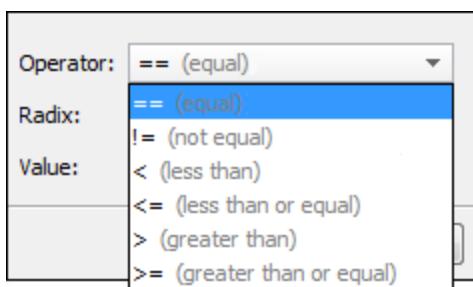


Figure 187: Setting the Comparison Operator

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.

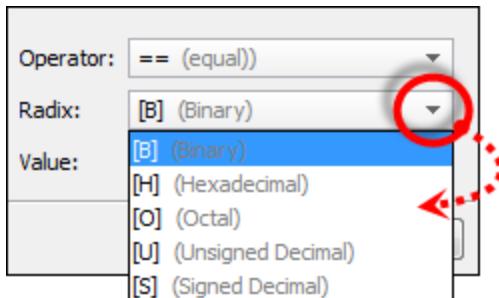


Figure 188: Selecting the Radix for the Compare Value

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**.

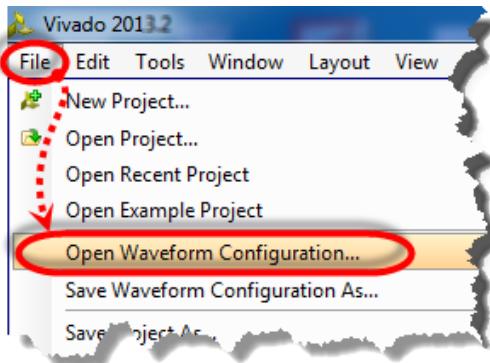


Figure 189: Opening a Waveform Configuration File (.wcfg)

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.

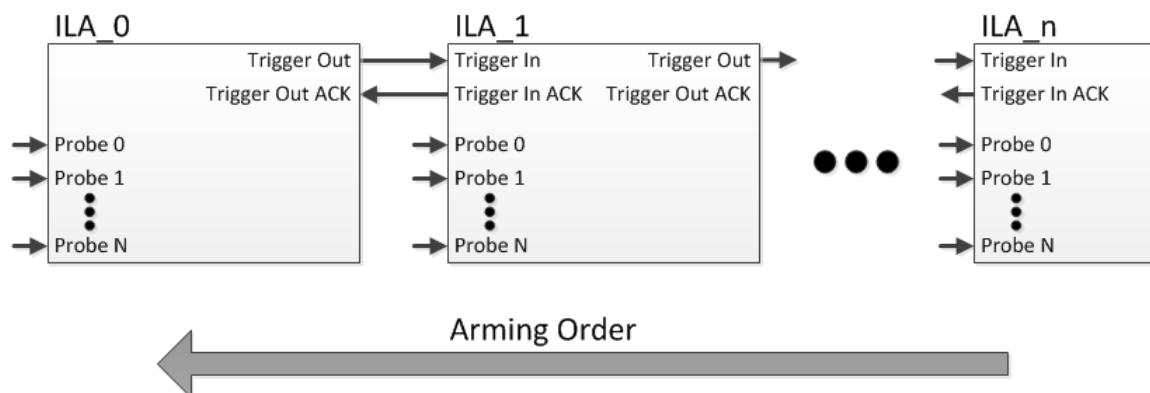


Figure 190: Connections and Arming Order for Multiple ILAs

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

This section contains instructions for commonly performed tasks using the Vivado High-Level Synthesis (HLS) tool.

In This Section

Launching the Vivado HLS Tool.....	147
Creating a Vivado HLS Project	148
Opening a Vivado HLS Project.....	151
Adding Synthesizable Source(s) to a Vivado HLS Project.....	152
Adding Simulation Source(s) to a Vivado HLS Project.....	153
Creating a New Source for an HLS Project.....	154
Simulating a Vivado HLS Design.....	155
Setting Synthesis Options.....	157
Synthesizing the Vivado HLS Design.....	158
Simulating a Vivado HLS Design.....	158
Exporting a Vivado HLS Design as IP.....	161
Analyzing an HLS Design	162

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. Select **Start > All Programs > Xilinx Design Tools > Vivado 2016.1 > Vivado HLS > Vivado HLS 2016.1.**

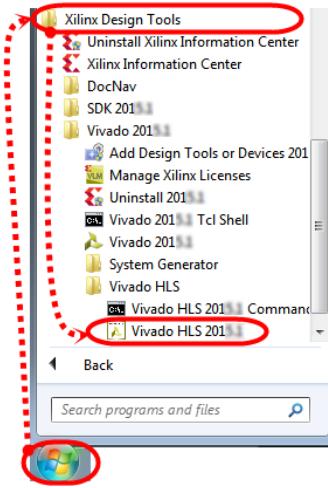


Figure 191: Launching the Vivado HLS Tool

-- OR --

Double-click the **Vivado HLS** shortcut icon () on the desktop.

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 192: Vivado HLS Welcome Screen

Creating a Vivado HLS Project

Here you will learn to create a new Vivado HLS project from scratch.

1-1. Create a Vivado HLS project named *your HLS project name*.

- 1-1-1. From the Welcome Page, click **Create New Project**.



Figure 193: Creating a New Vivado HLS 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 **the location at which the project should be placed** in the Location field (2).

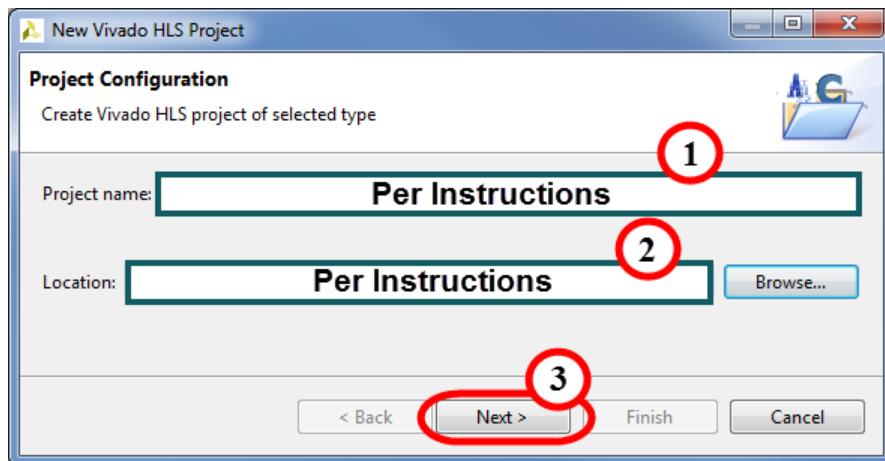


Figure 194: Configuring a New HLS Project

- 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.

The Open File dialog box opens.

Note: If do not have existing files at this moment and you want to create new ones, click **New File**.

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 **Open to add these files.**

Note 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.**

1-3-7. Enter **the name of the top function in the Top Function field.**

Note: You can also manually enter the name of the top function in the Top Function field.

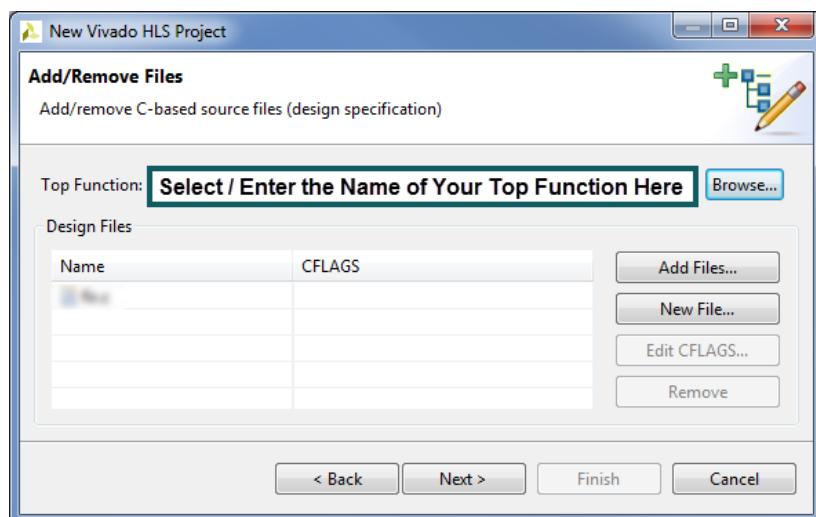


Figure 195: Adding Files to a New Vivado HLS Project

1-3-8. Click **Next.**

1-4. Add any existing testbench files.

If you have (or want) any testbench files they can be entered here.
Sometimes the testbench 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.

1-5-1. By default, **your solution name** is populated in the Solution Name field.

No changes are required.

1-5-2. Set the clock period to **your clock period**.

You can leave the Uncertainty field blank.

1-5-3. Click the **Browse** button to select a part or board.

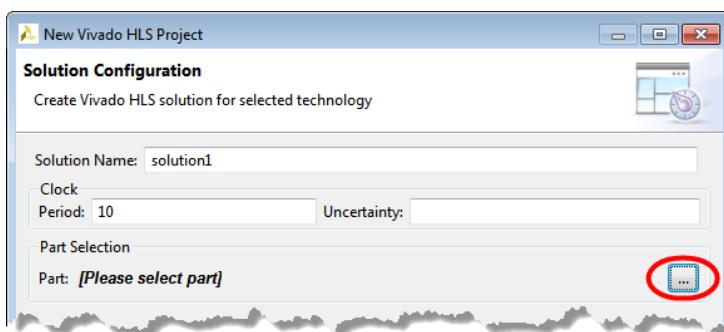


Figure 196: Locating the Board Browse Button

1-5-4. Click **Boards** as shown below.

1-5-5. Enter **the family name** in the Search field.

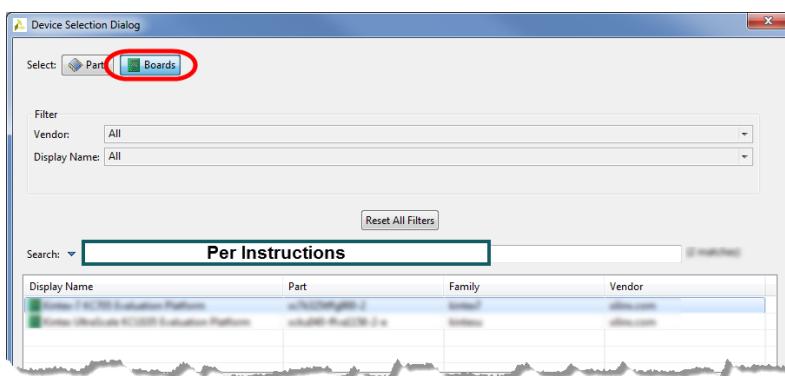


Figure 197: Filtering by Boards and Zynq to Quickly Locate Target Platforms

1-5-6. Select **your board** from the search list.

1-5-7. Click **OK** to select the board.

1-5-8. 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 198: 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...**.

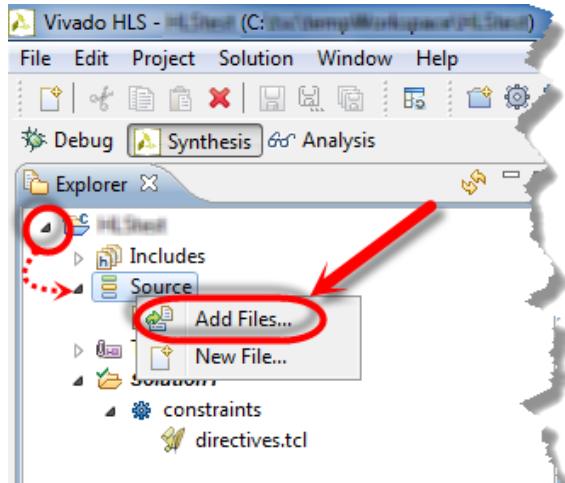


Figure 199: Add Existing Files or Create New Files for the HLS Project

-- OR --

Select **Project > Add Source...**.

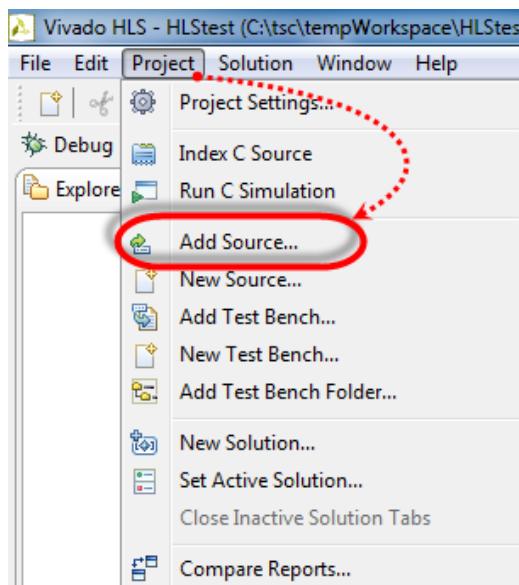


Figure 200: Adding a New Source from the Project Menu

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**.

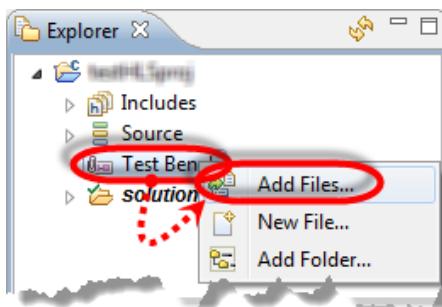


Figure 201: Adding Files to a Vivado HLS Testbench

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...**.

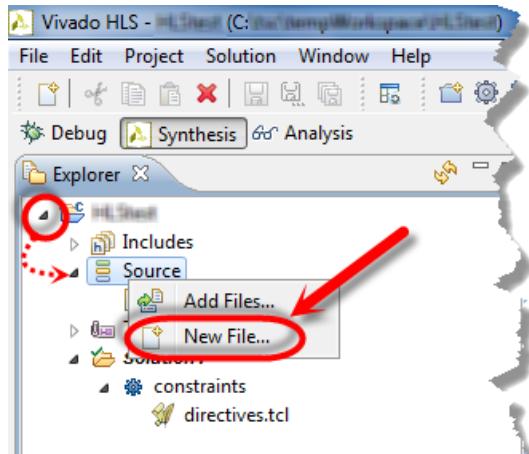


Figure 202: Creating a New HLS Source File

-- OR --

Select **Project > New Source**.

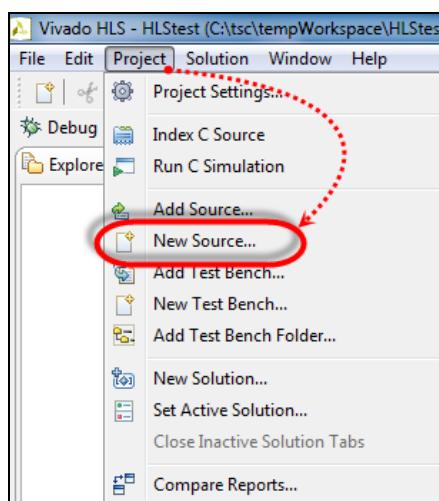


Figure 203: Creating a New Source from the Project Menu

- 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 ().

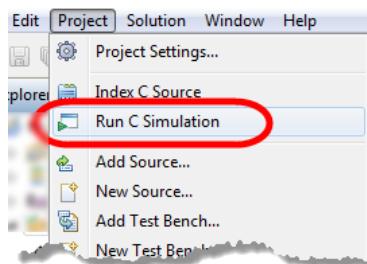


Figure 204: Launching the C Simulation

The Run C Simulation dialog box opens.

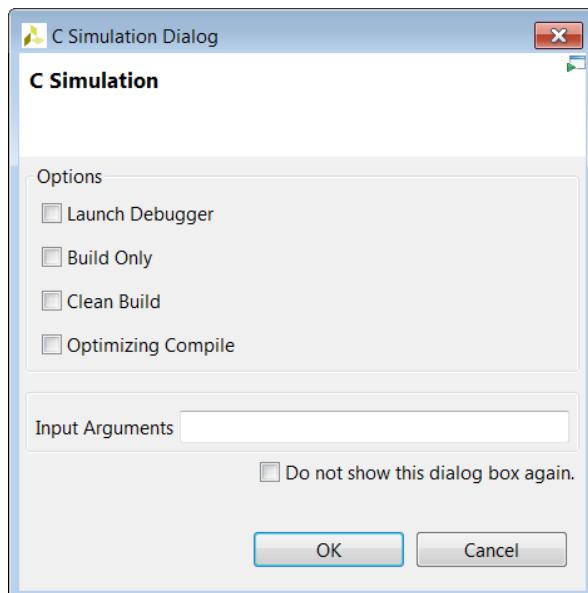


Figure 205: C Simulation Dialog Box

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:** Compile the object/netlist files but do not execute.
- **Clean Build:** Remove previously compiled files before compiling.
- **Optimizing Compile:** Use 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.

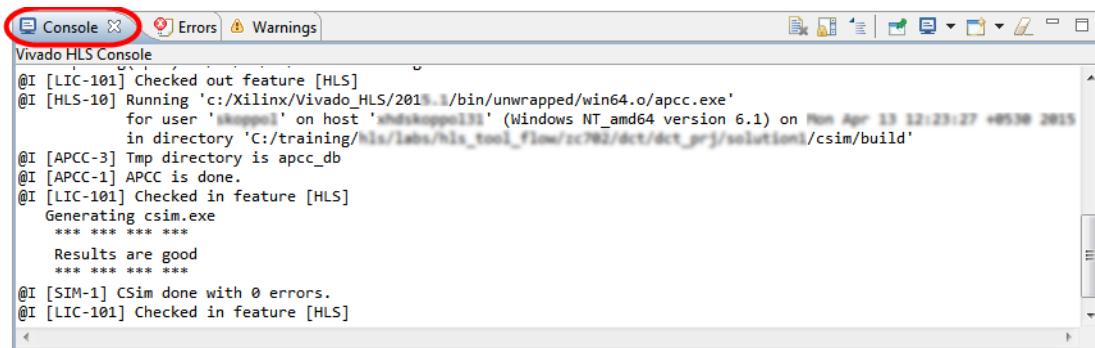
1-2. **View the simulation report.**

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, the text area displays the following simulation log:

```

Vivado HLS Console
@I [LIC-101] Checked out feature [HLS]
@I [HLS-10] Running 'c:/Xilinx/Vivado_HLS/2015.1/bin/unwrapped/win64.o/apcc.exe'
    for user 'shdkoppell' on host 'shdkoppell31' (Windows NT_amd64 version 6.1) on Mon Apr 13 12:23:27 +0530 2015
    in directory 'C:/training/hls/labs/hls_tool_flow/zc702/dct/dct_prj/solution/csim/build'
@I [APCC-3] Tmp directory is apcc_db
@I [APCC-1] APCC is done.
@I [LIC-101] Checked in feature [HLS]
Generating csim.exe
*** *** ***
Results are good
*** *** ***
@I [SIM-1] CSim done with 0 errors.
@I [LIC-101] Checked in feature [HLS]

```

Figure 206: Example Output After a 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.

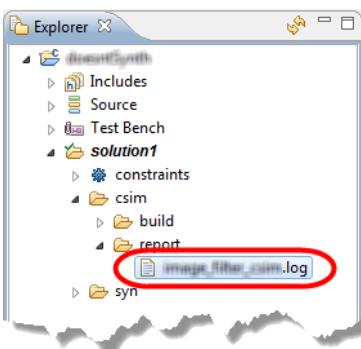
1-2-2. Expand **your HLS project name > solution1 > csim > report** in the Explorer pane.**1-2-3.** Double-click the log file name to open it in the editor pane.

Figure 207: 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 (⚙).**

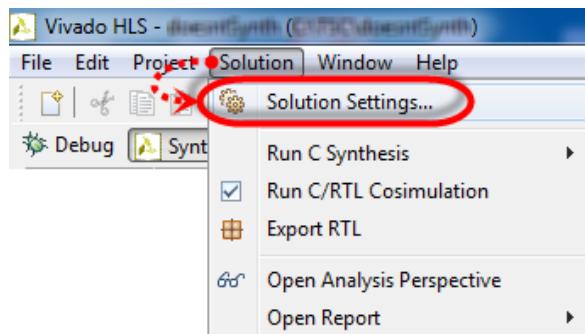


Figure 208: Accessing Synthesis Settings from the Menu Bar

The Solutions Settings dialog box opens.

- 1-1-2. Click the **Synthesis** option in the navigation panel.**

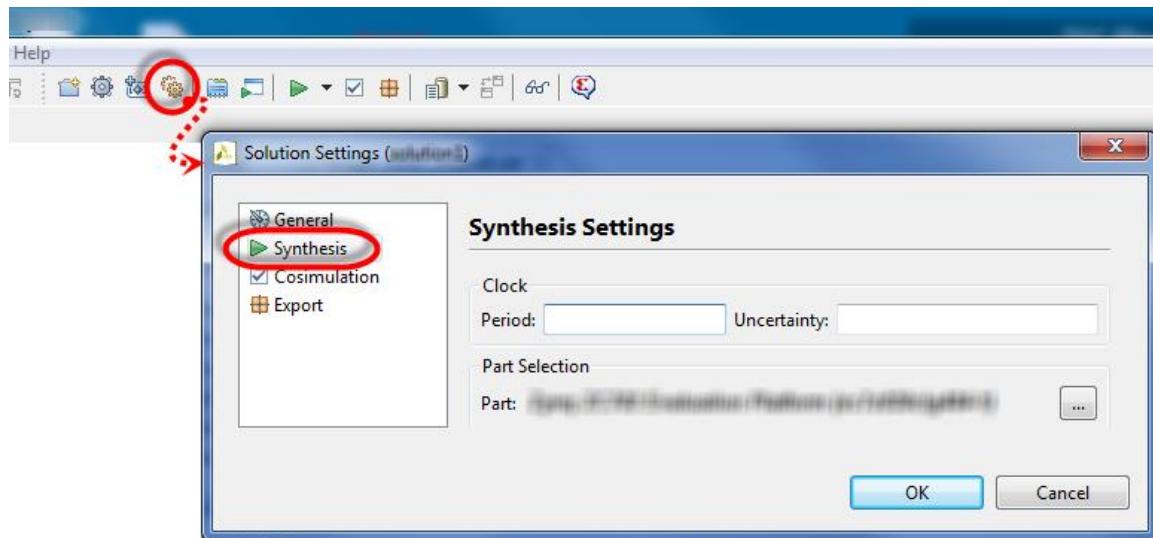


Figure 209: Accessing the Synthesis Options in the Solutions Setting Dialog Box

- 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 the Vivado HLS Design

1-1. Synthesize the design.

- 1-1-1.** Select **Solution > Run C Synthesis > Active Solution** or click the **Run Synthesis** icon in the menu bar.



Figure 210: 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.

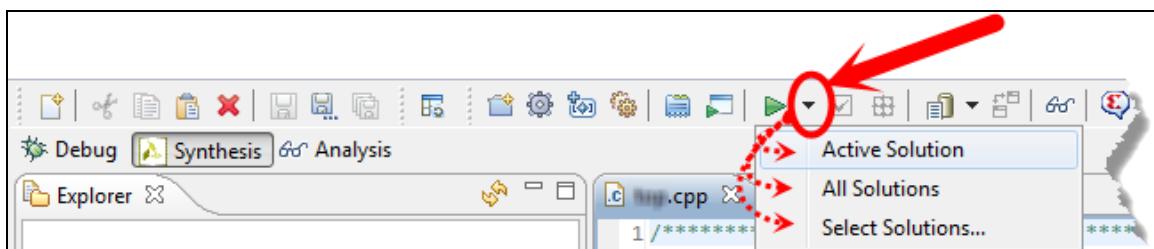


Figure 211: Options for What to Synthesize

Simulating 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 ().

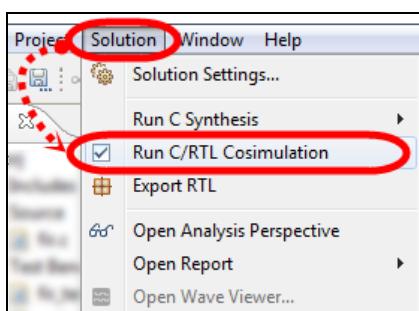


Figure 212: Launching from the Menu

The Run C/RTL Co-simulation dialog box opens.

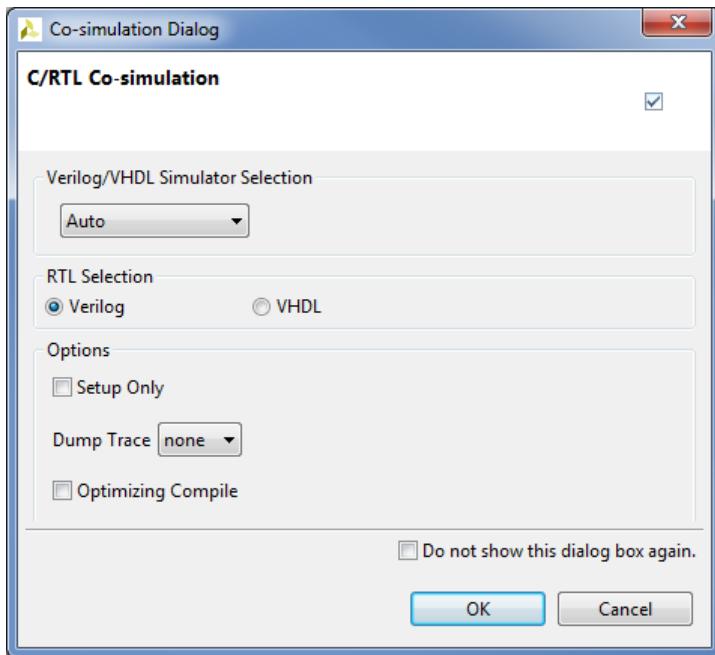


Figure 213: Co-simulation Dialog Box

Each of the options controls how C/RTL Cosimulation 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 testbench. Using this option increases the compile time but the simulation executes faster.

1-1-2. Select the options that you want.

1-1-3. Click OK.

The simulation log will be displayed in the editor pane.

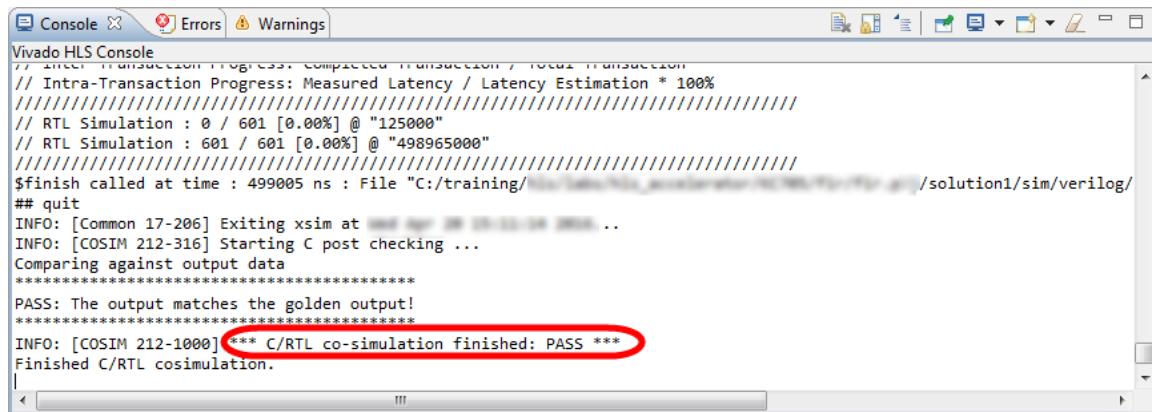
1-2. View the Cosimulation report.

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.



```

Console X Errors Warnings
Vivado HLS Console
// Intra-Transaction Progress: Measured Latency / Latency Estimation * 100%
// RTL Simulation : 0 / 601 [0.00%] @ "125000"
// RTL Simulation : 601 / 601 [0.00%] @ "498965000"
$finish called at time : 499005 ns : File "C:/training/ /solution1/sim/verilog/
## quit
INFO: [Common 17-206] Exiting xsim at ...
INFO: [COSIM 212-316] Starting C post checking ...
Comparing against output data
*****
PASS: The output matches the golden output!
*****
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
Finished C/RTL cosimulation.
|
```

Figure 214: Example Output After a C/RTL Cosimulation

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 > sim > report** in the Explorer pane.

1-2-3. Double-click the log file name to open it in the editor pane.

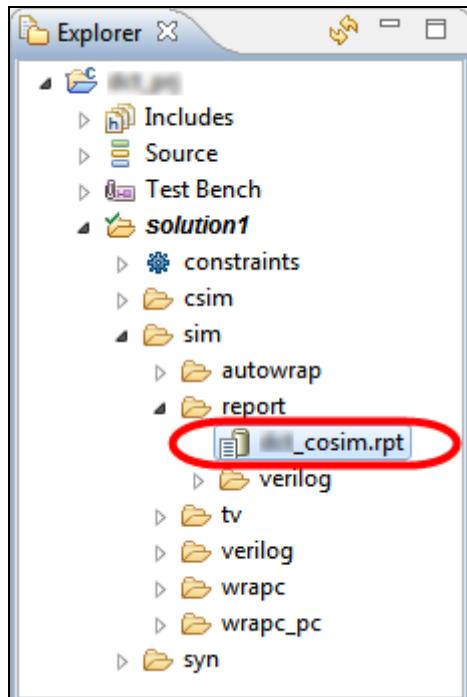


Figure 215: Locating the Co-Simulation Log File

The Cosimulation Report in HTML format will be displayed in the main viewing area.

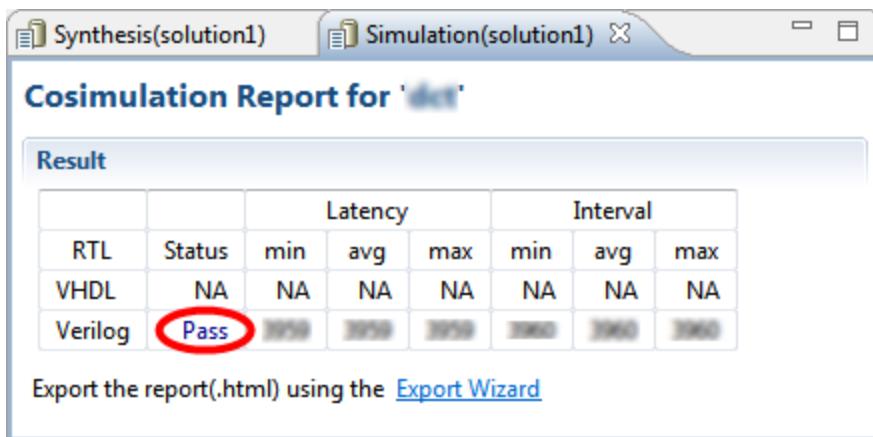


Figure 216: Cosimulation Report - HTML

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**.

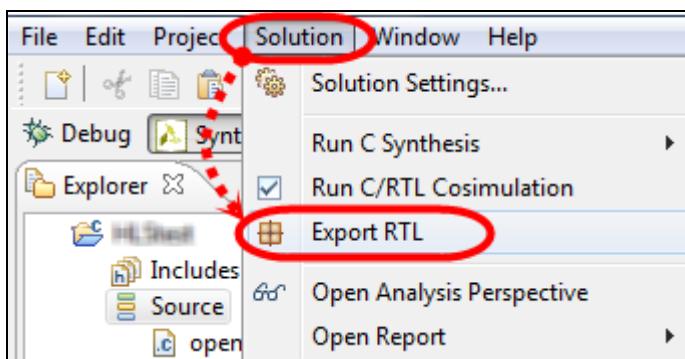


Figure 217: Exporting the RTL from Vivado HLS using the Menu Bar

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.

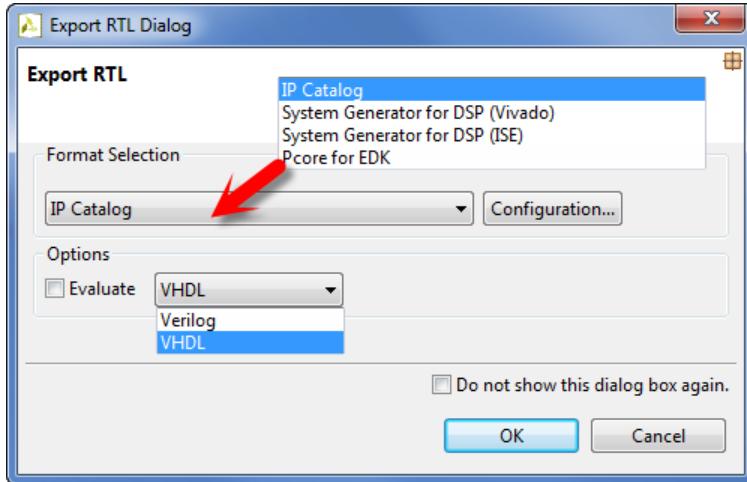


Figure 218: Selecting the Export Format for the IP

- 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.

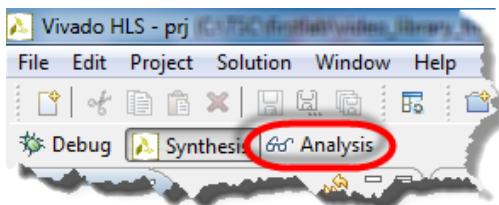


Figure 219: Locating the Analysis Button

The data-flow diagram opens.

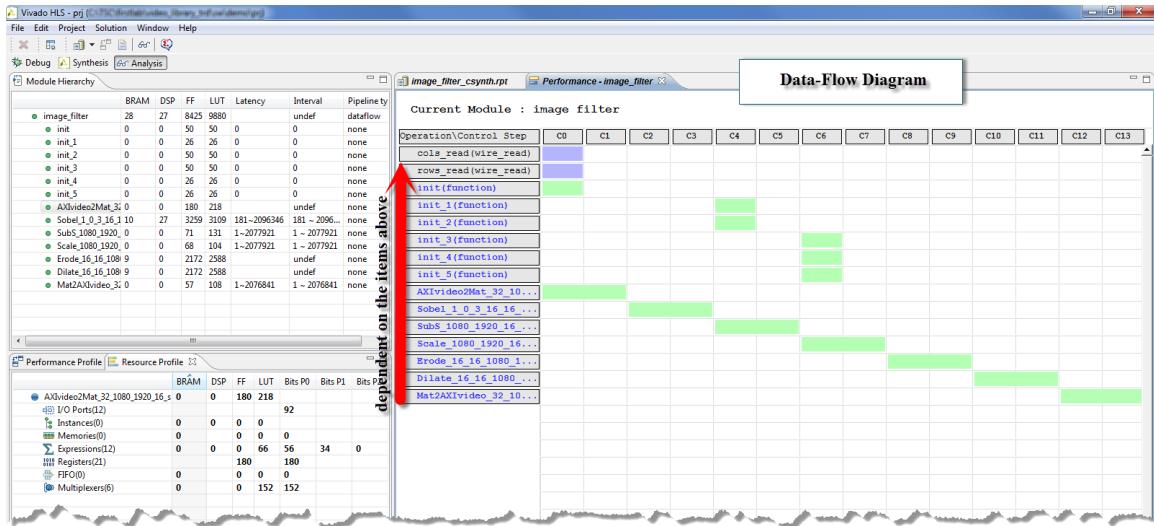


Figure 220: Example of Analysis View

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.

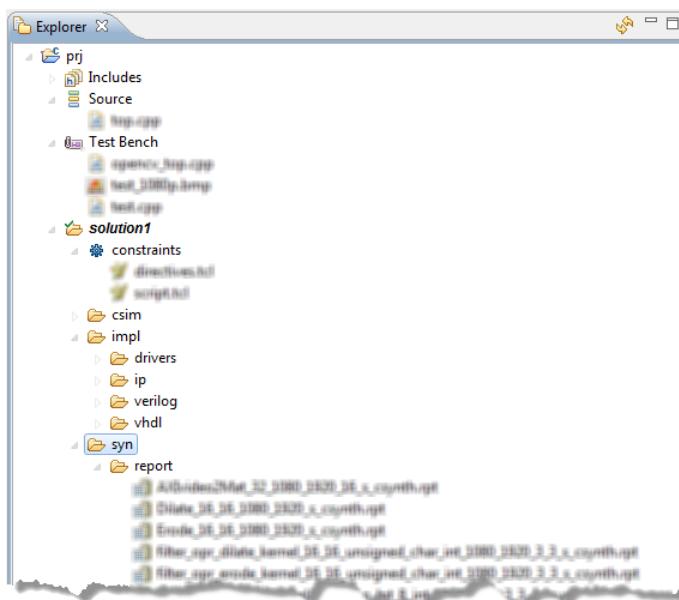


Figure 221: Accessing the Synthesis Reports

- 1-2-3.** Locate the performance approximations in the report.

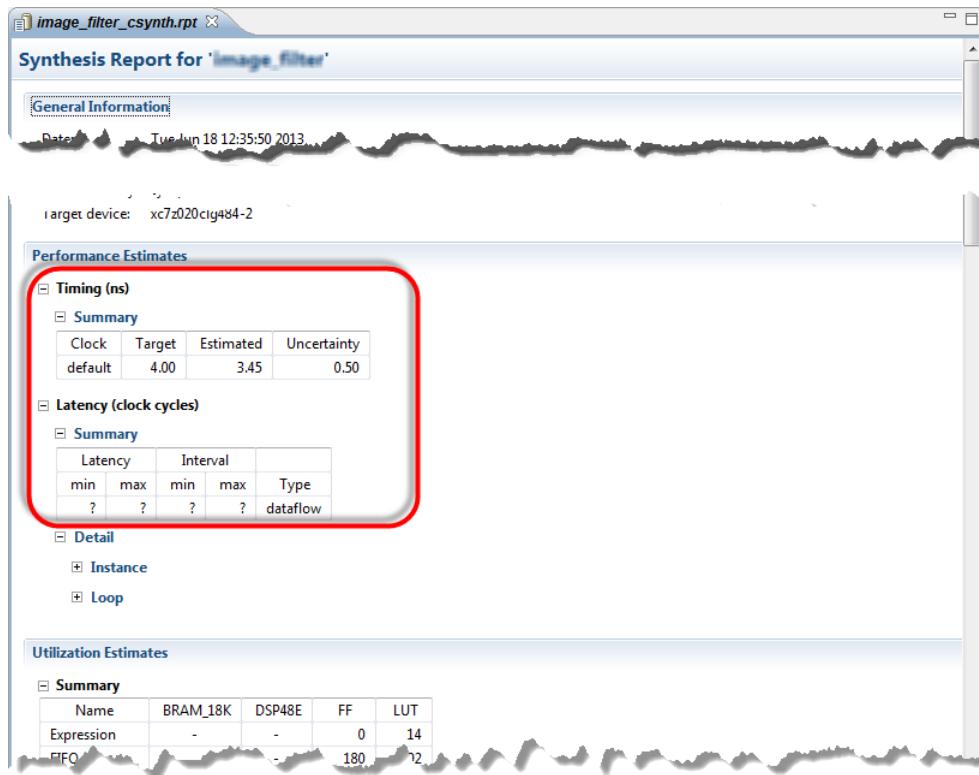


Figure 222: Locating the Performance Approximations in the Synthesis Report

SDK Tool Operations

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.

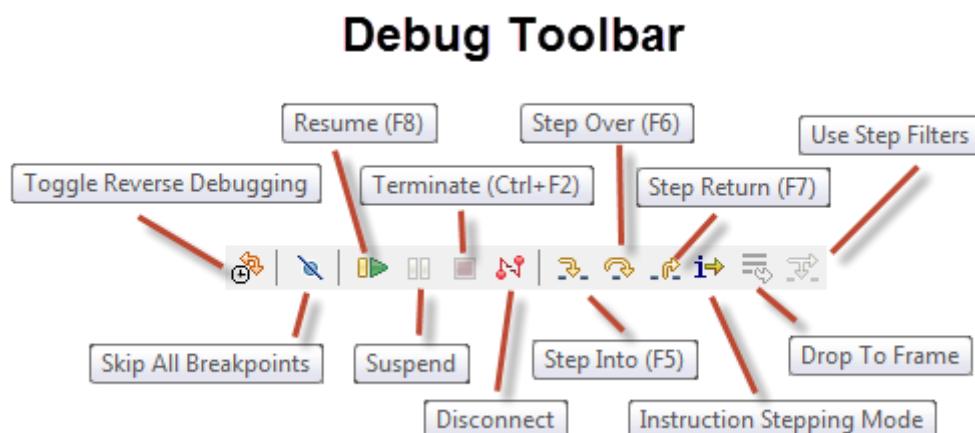


Figure 223: Debug Toolbar

Instructions

The following sections contain only the detailed instructions for the specified task.

In This Section

Launching the SDK or SDSoc Tool and Setting the Workspace.....	167
Launching the SDK or SDSoc Tool and Setting the Workspace.....	169
Adding a Library to a BSP	171
Import Example Application from Library.....	172
Creating a Hardware Platform Specification.....	172
Creating a Hardware Platform Specification and BSP	175
Creating a Basic Board Support Package.....	176
Enabling C Libraries.....	178
Creating a C/C++ Application Project	179
Creating a C/C++ Application Project from the SDK Welcome Window.....	181
Importing Sources to an Application.....	183
Importing an Existing Project into the SDK or SDSoc Tool	185
Setting Compiler Options	187
Setting Compiler Options	190
Opening a Source File in the Editor	192
Finding Text in a Source File.....	193
Enabling Line Numbering in the SDK Text Editor.....	194
Saving and Compiling an Application.....	194
Customizing the Linker Script	195
Programming the Device from SDK.....	196
Creating a Boot Image File	197
Configuring a Local Repository	199
Setting Up a Run Configuration	201
Setting Up a Run Configuration (System Debugger)	203
Running with a Default Configuration	206
Debugging.....	206
Running the Application on Hardware	236
Linux and Remote System Explorer.....	236
Launching a Software Application on Hardware.....	258
Switching Perspectives.....	259
Configuring the SDK Terminal	260
Configuring the SDK Terminal	262
Terminating a Running Application	263
Closing the Xilinx SDK Tool.....	263
Adding Symbols to a BSP.....	264
Adding Symbols to Application Project Settings.....	265

Launching the SDK or SDSoc Tool and Setting the Workspace

1-1. Launch the SDK or SDSoc tool and set the workspace.

- 1-1-1.** Select **Start > All Programs > Xilinx Design Tools > SDK or SDSoc 2016.1 > Xilinx SDK or SDSoc 2016.1** to launch the tool.

Alternatively, you can launch the tool from its desktop shortcut, if available.

The Workspace Launcher opens after a moment.

The SDK or SDSoc tool creates a workspace environment that initially only contains a thin structure that tracks tool settings and maintains the SDK or SDSoc 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 SDK or SDSoc tool (select **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.

The default location for the SDK or SDSoc software workspace (when launching from within the Vivado Design Suite) is the root directory of your hardware project; however, a long path name can lead to problems on Windows-based machines. There is no default location for SDK or SDSoc tool projects. Placing your project at the root level or one hierarchical level below helps keep the path names as short as possible and is recommended.

Many of the Xilinx labs do not follow this guidance as it is important to keep a predictable structure through the various courses and labs. These labs have been tested to ensure that path name lengths do not cause problems.

- 1-1-2.** When the Workspace Launcher opens, you can either enter **You can either leave this field at its default setting, which will place the workspace under the Vivado Design Suite project hierarchy, or you can select your own location.** into the Workspace field, or you can use the Browse button.

Note that when you use the Browse button, you will need to select the **You can either leave this field at its default setting, which will place the workspace under the Vivado Design Suite project hierarchy, or you can select your own location.** directory and click **OK**.

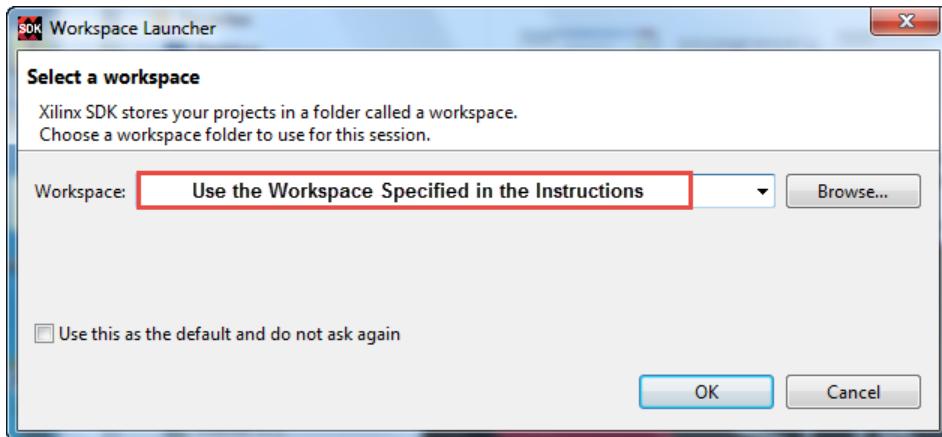


Figure 224: Setting Up the Workspace Environment Path

- 1-1-3.** Click **OK** to close the Workspace Launcher dialog box and open the new workspace.

A workspace location and hardware platform are created when the **Export Hardware Design for SDK or SDSoc** command is performed from the Vivado Design Suite (or they can be created manually). While not a requirement, it is a good idea to keep the related files together.

Note that SDK must associate with a hardware system that has been previously exported so that an appropriate software platform or board support package can be built. However, the SDSoc™ development environment can take advantage of inbuilt hardware profiles.

When the SDK or SDSoc 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.

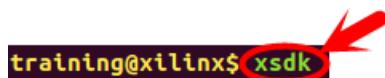
- 1-1-4.** Close the Welcome tab if it appears.

This will give you more room to view your project. You may also want to maximize the SDK or SDSoc window—there will be a lot to see.

Launching the SDK or SDSoc Tool and Setting the Workspace

1-1. Launch the SDK or SDSoc tool and set the workspace.

- 1-1-1.** Launch the SDK tool by entering xsdk at the command prompt:



```
training@xilinx$ xsdk
```

Figure 225: Launching xsdk from the Linux Prompt

The .bashrc file sources the scripts for the PetaLinux tools (source /opt/pkg/petalinux-v2016.1-final/settings.sh) and Xilinx SDK (source /opt/pkg/Xilinx/SDK/2016.1/settings64.sh).

Note: This assumes that the PetaLinux tools are installed in the /opt/pkg directory.

The Workspace Launcher opens after a moment.

The SDK or SDSoc tool creates a workspace environment that initially only contains a thin structure that tracks tool settings and maintains the SDK or SDSoc 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 SDK or SDSoc tool (select **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.

The default location for the SDK or SDSoc software workspace (when launching from within the Vivado Design Suite) is the root directory of your hardware project; however, this can lead to long path names, which may cause problems on Windows-based machines. There is no default location for SDK or SDSoc tool projects. Placing your project at the root level or one hierarchical level below helps keep the path names as short as possible and is recommended.

- 1-1-2.** When the Workspace Launcher opens, you can either enter **You can either leave this field at its default setting, which will place the workspace under the Vivado Design Suite project hierarchy, or you can select your own location.** into the Workspace field, or you can use the Browse button.

Note that when you use the Browse button you will need to select the **You can either leave this field at its default setting, which will place the workspace under the Vivado Design Suite project hierarchy, or you can select your own location.** directory and click **OK**.

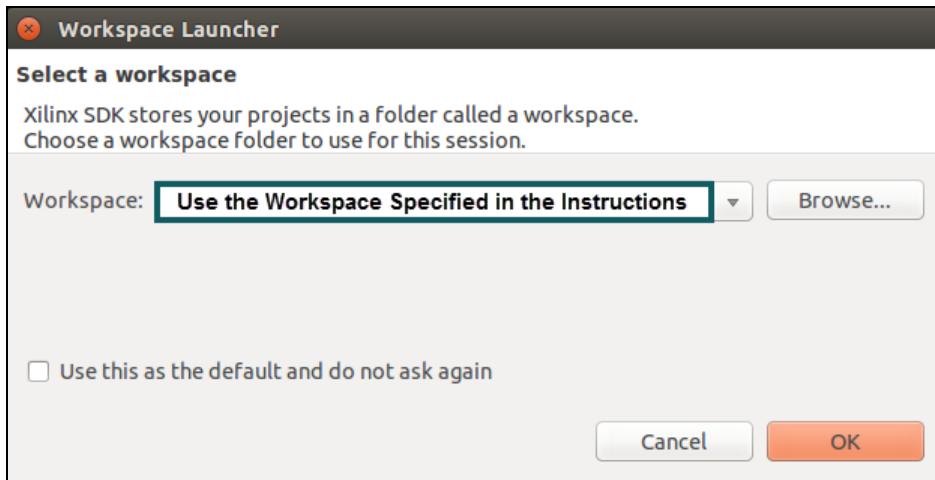


Figure 226: Setting Up the Workspace Environment Path

- 1-1-3.** Click **OK** to close the Workspace Launcher dialog box and open the new workspace.

A workspace directory location and hardware platform are created when the **Export Hardware Design for SDK or SDSoc** command is performed from the Vivado Design Suite (or they can be created manually). While not a requirement, it is a good idea to keep the related files together.

Note that SDK must associate with a hardware system that has been previously exported so that an appropriate software platform or board support package can be built. However, the SDK or SDSoc tool can take advantage of built-in hardware profiles.

When the SDK or SDSoc 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 by clicking the 'X' in the Welcome tab.

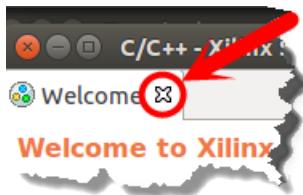


Figure 227: Closing XSDK's Welcome Tab

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.** If the Overview view is not selected, click **Overview** in the left window pane (4).

- 1-1-5.** Select **your library** under the Supported Libraries section (5).

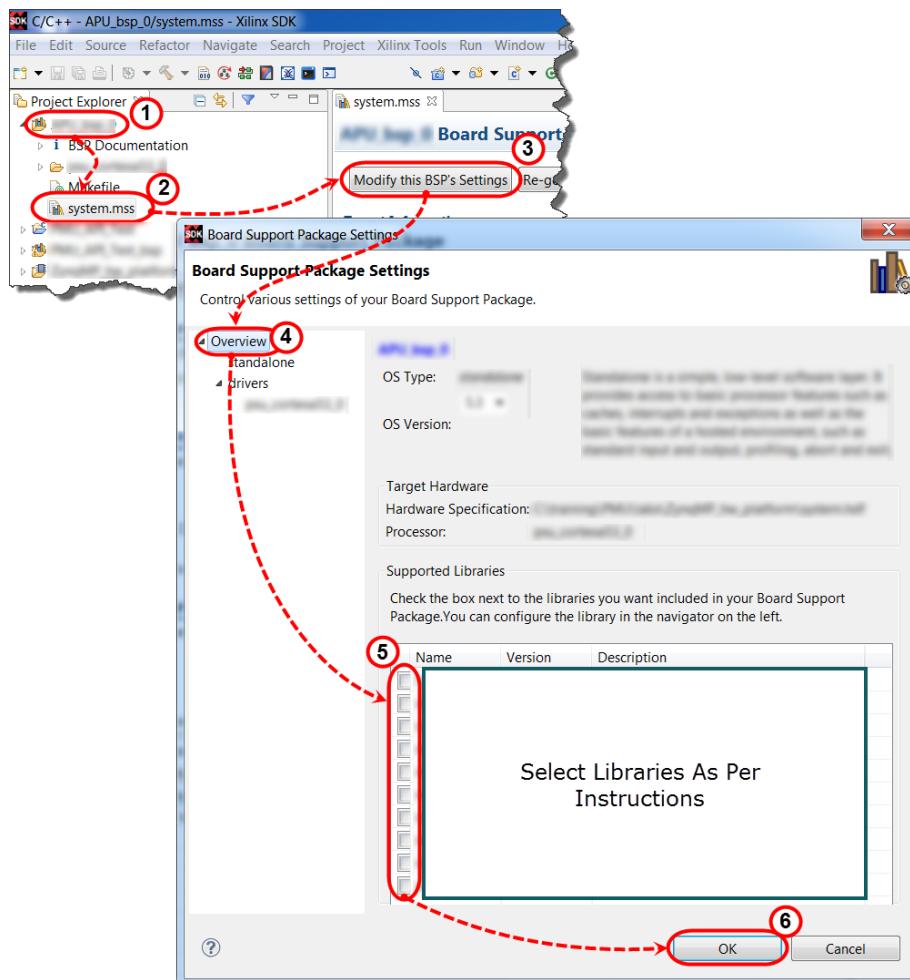


Figure 228: Accessing the BSP Customization Dialog Box

- 1-1-6.** Click **OK** to modify the BSP and rebuild the BSP's sources and binaries (6).

Import Example Application from 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).

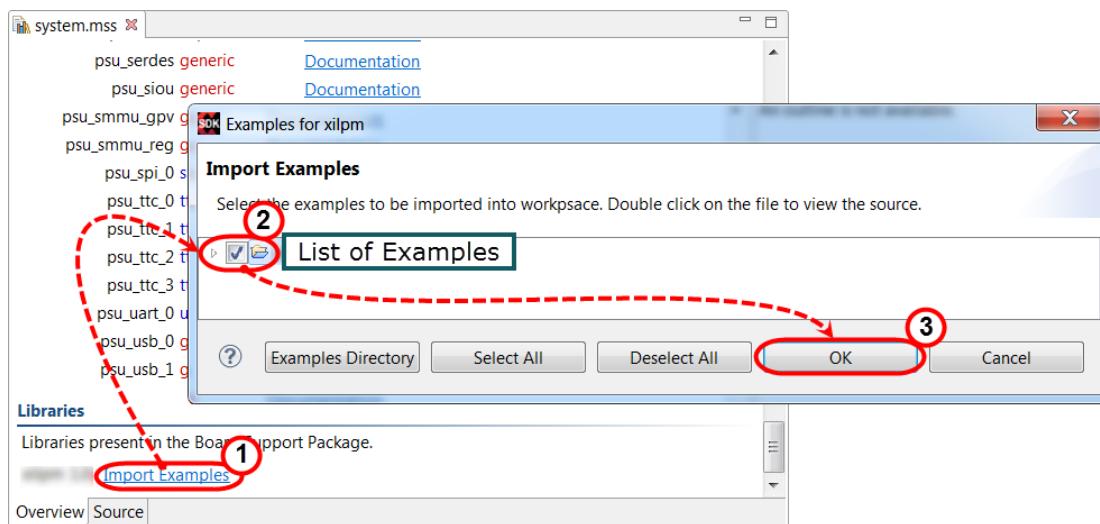


Figure 229: Importing Library Example

- 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 All Programmable SoC-based systems or a list of all peripherals for a non-Zynq All Programmable 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.

1-1. Create the hardware platform specification.

- 1-1-1.** Select **File** (1) > **New** (2) > **Other** (3) to see the Xilinx-specific options.

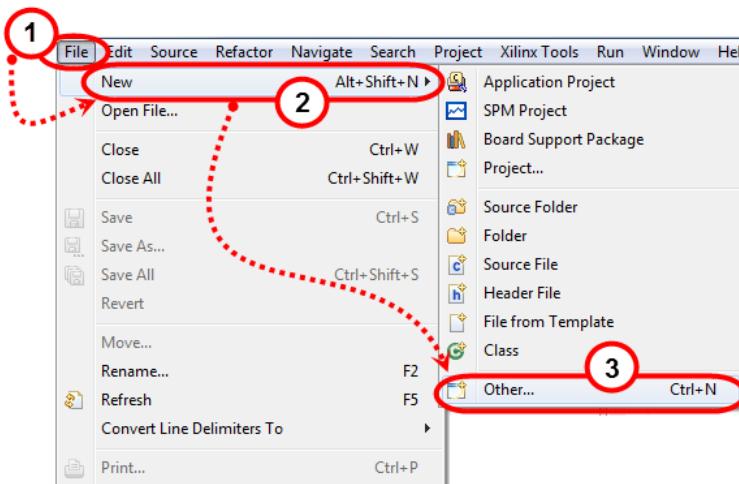


Figure 230: Accessing the New Wizards

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).

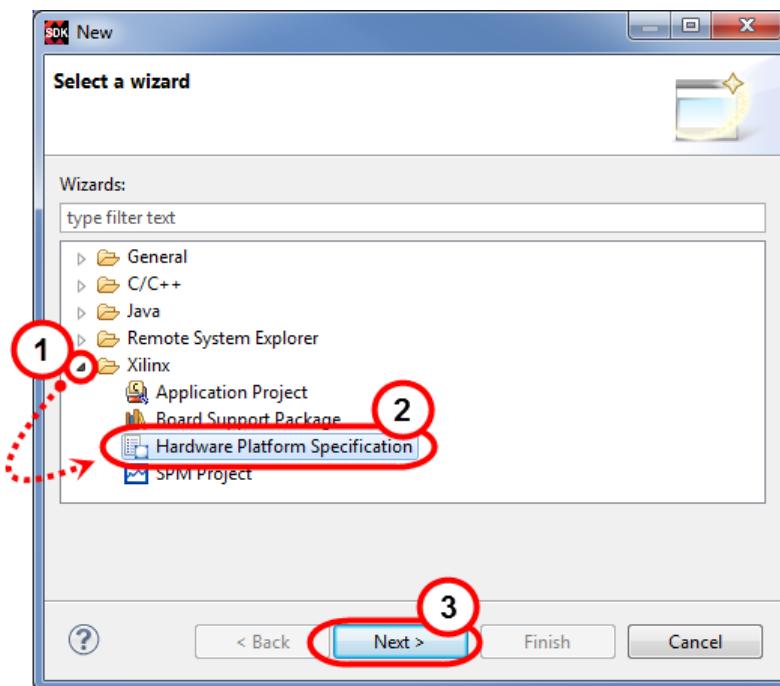


Figure 231: Selecting the Hardware Platform Specification Wizard

- 1-1-4.** Click **Next** (3) to open the New Hardware Project dialog box.

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. Under the Target Hardware Specification region, browse to the *the location of the files to be exported from the Vivado Design Suite* directory and select the **name of your block design** file.

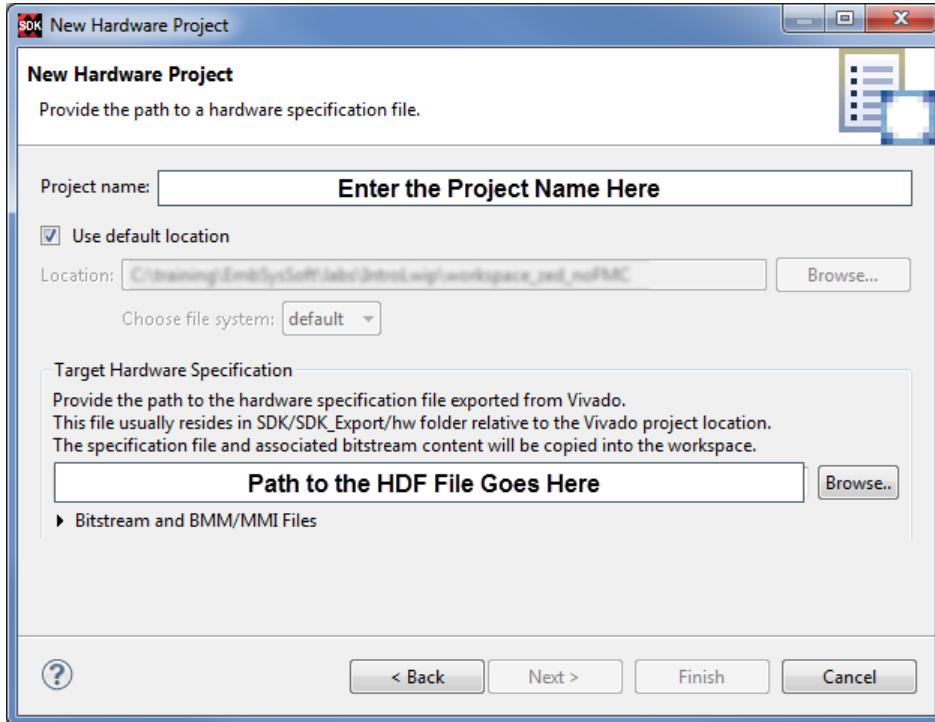


Figure 232: New Hardware Project Dialog Box

- 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**.

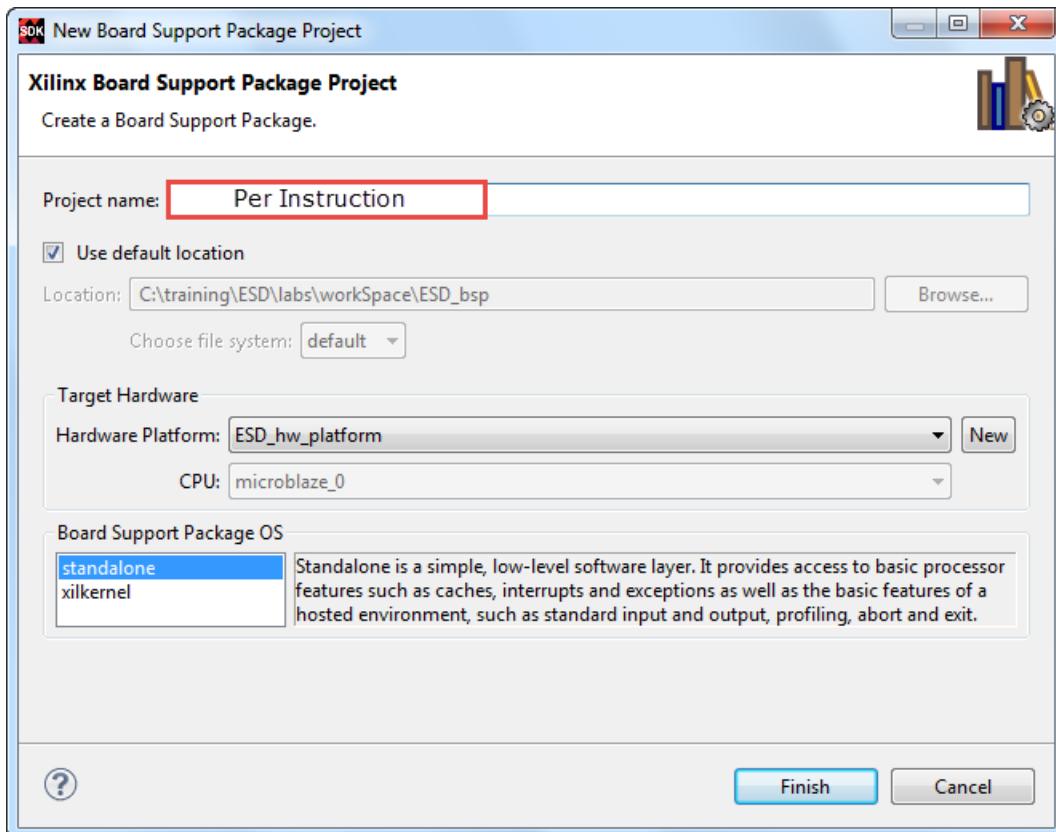


Figure 233: Newly Created Hardware Project

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.

Creating a Basic Board Support Package

1-1. Create a basic board support package.

1-1-1. Select **File > New > Board Support Package**.

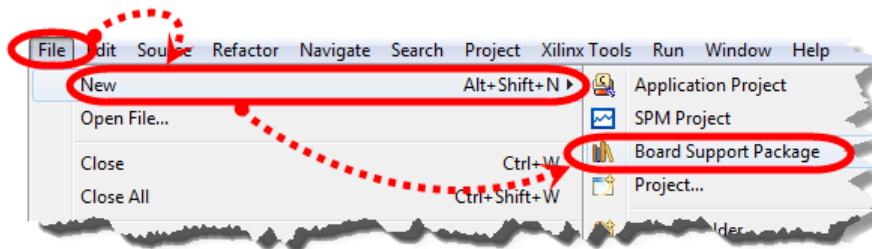


Figure 234: Launching the Board Support Package Wizard

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.

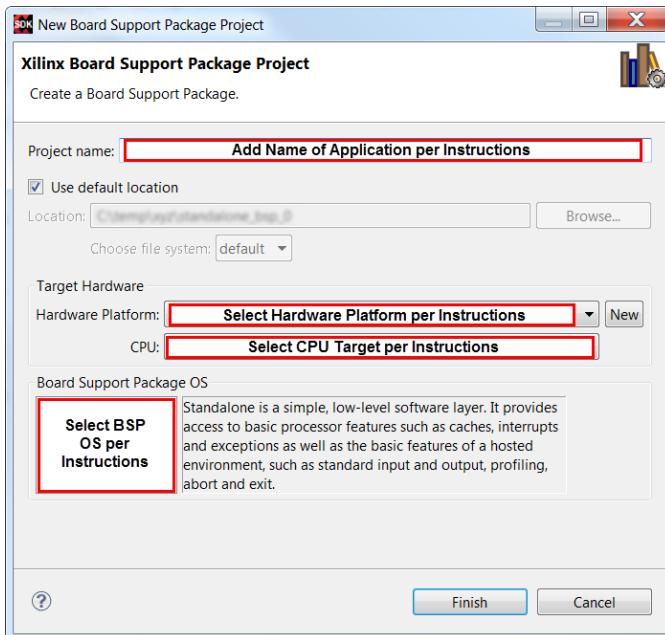


Figure 235: Board Support Package Project Dialog Box

1-1-4. Click **Finish.**

Now that the BSP has been created, it can be customized if so desired.

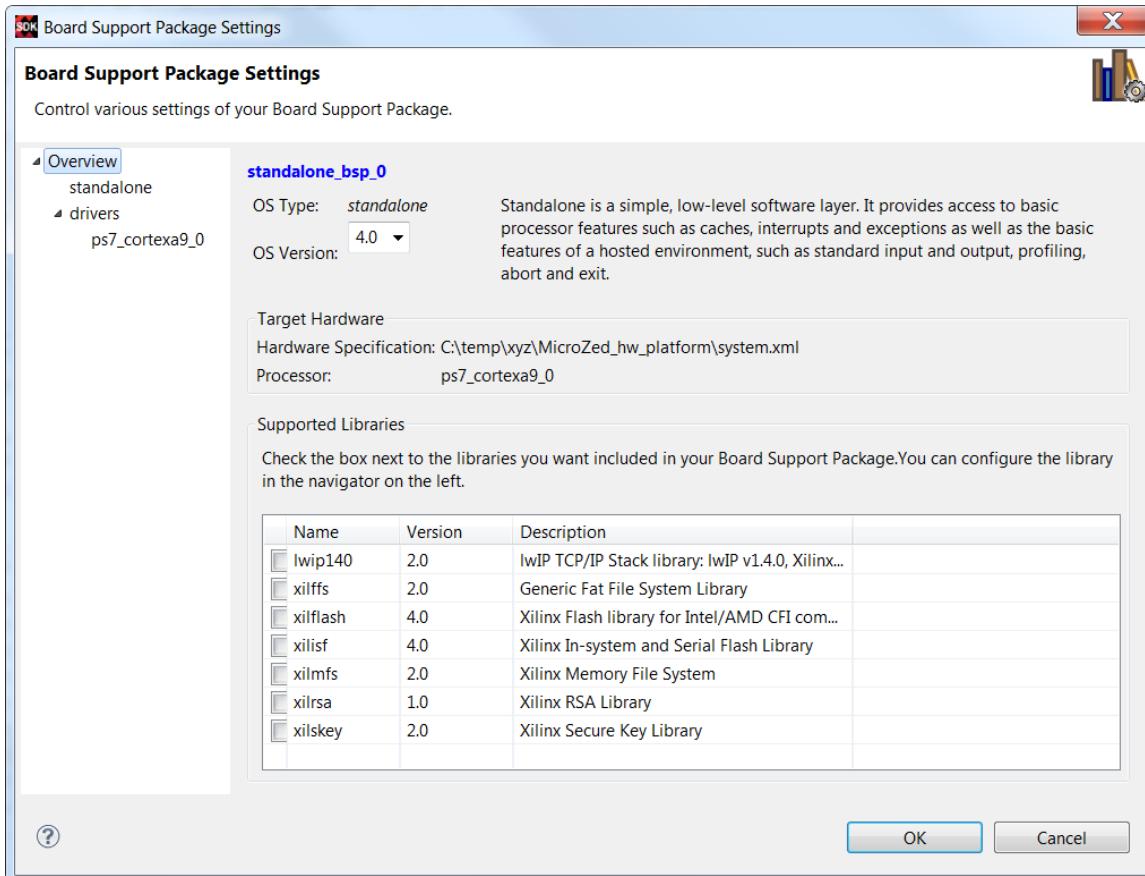


Figure 236: Board Support Package Settings Dialog Box

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**.

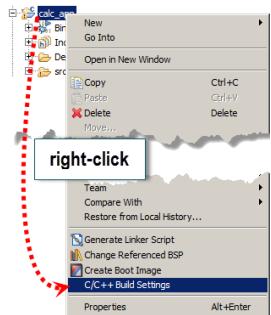


Figure 237: Accessing the C/C++ Build Options

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.

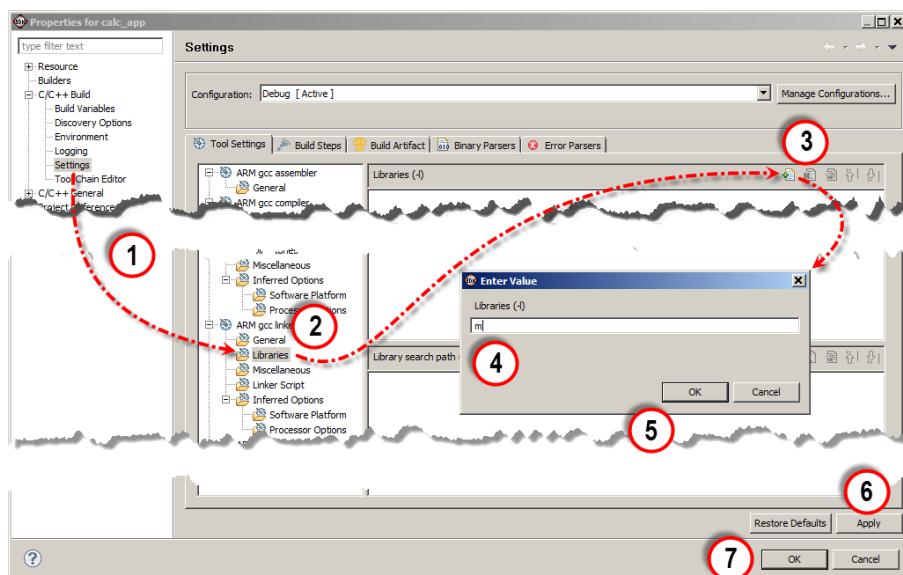


Figure 238: Adding the Math Library to the Linker

- 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

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.

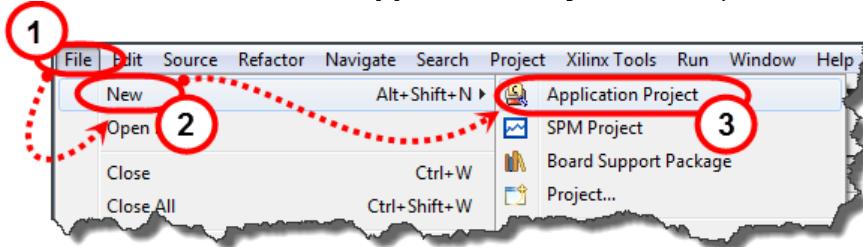


Figure 239: Creating an Application Project

- 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.

This will populate the OS Platform drop-down list accordingly.

- 1-1-5.** Ensure that **your desired operating system** is selected from the OS Platform drop-down list.

- 1-1-6.** 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 the name for the BSP as **your BSP name** (or you can use the default name provided).

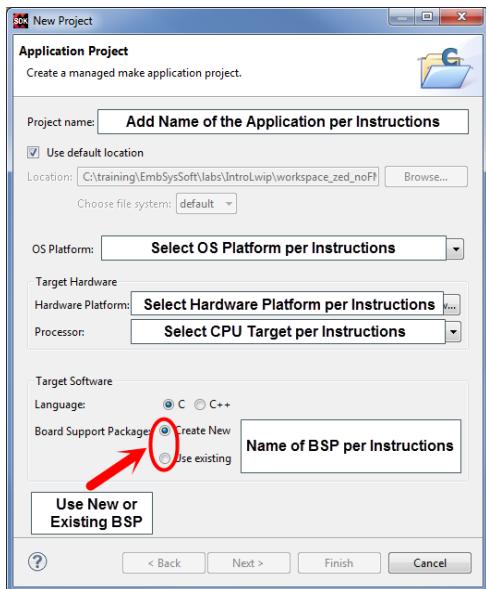


Figure 240: Entering Application Project Information

- 1-1-7.** Click **Next** to select the template for this application.
1-1-8. Select **an application template**.

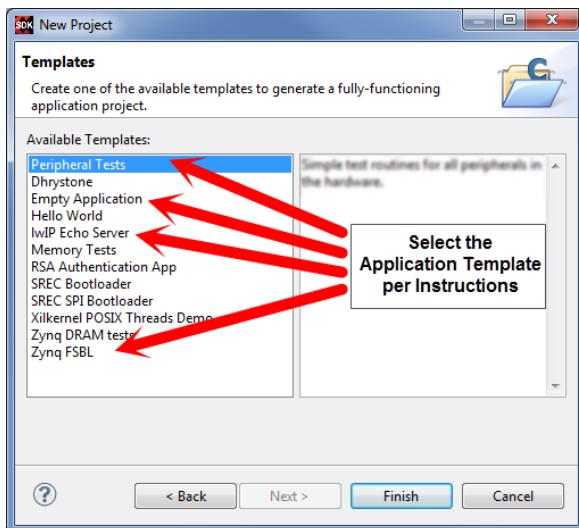


Figure 241: Selecting an Application Template (Selection in Figure May Not Match Instructions)

- 1-1-9.** Click **Finish** to create the new project.

As the project is created, the new BSP is compiled, and any sources from the templates are also compiled. 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.

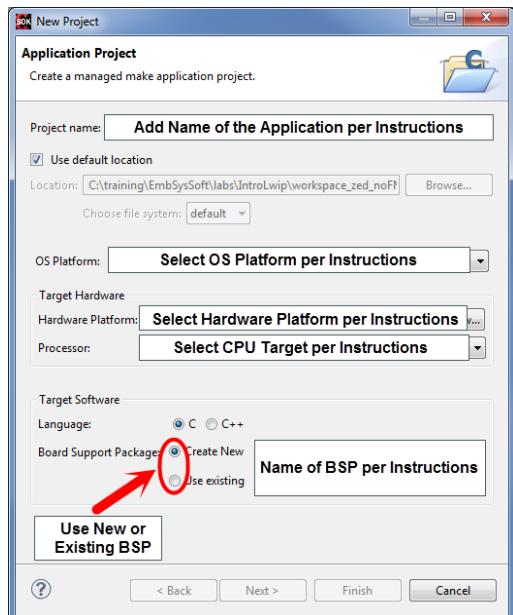


Figure 242: Entering Application Project Information

- 1-1-6.** Click **Next**.

- 1-1-7.** Select an application template.

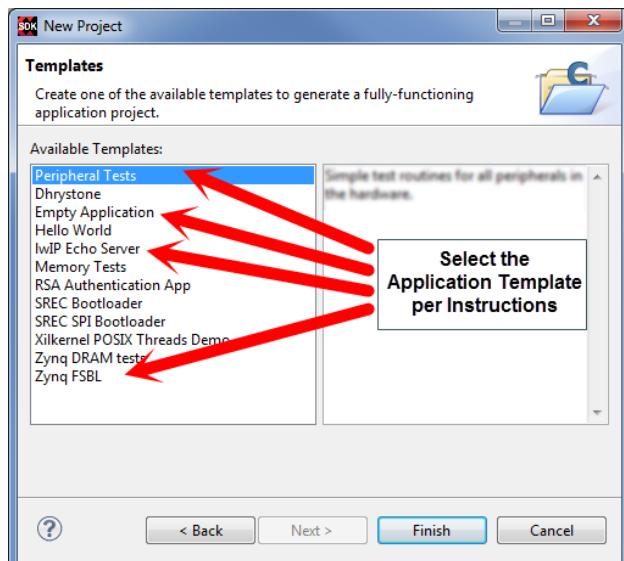


Figure 243: Selecting an Application Template (Selection in Figure May Not Match Instructions)

- 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 Eclipse framework 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. Using the Project Explorer, expand the project named **your application project name > src**.
- 1-1-2. Right-click the desired destination directory in the project that you want to place the resource files (typically the src directory) (1).
- 1-1-3. Select **Import** to open the Import Wizard (2).

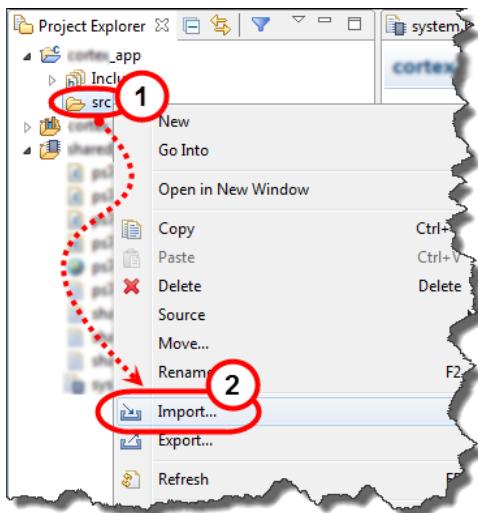


Figure 244: Importing a Resource File

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).

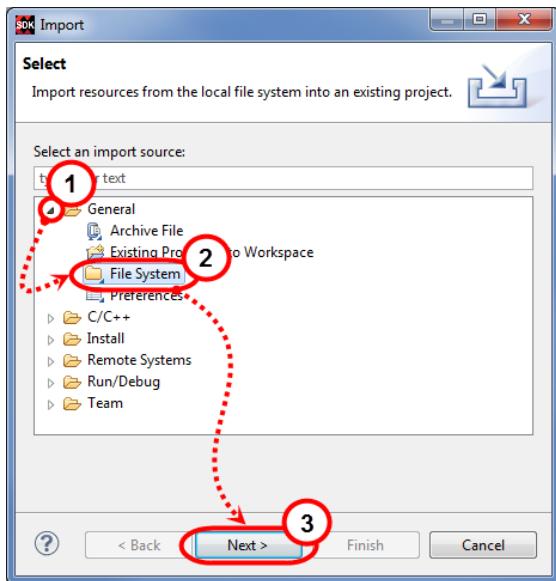


Figure 245: Selecting File System

1-1-6. Click **Next** to advance to specifying the files to import (3).

1-1-7. Using the *From directory* field, browse to *the location of your files*.

1-1-8. Select **your files**.

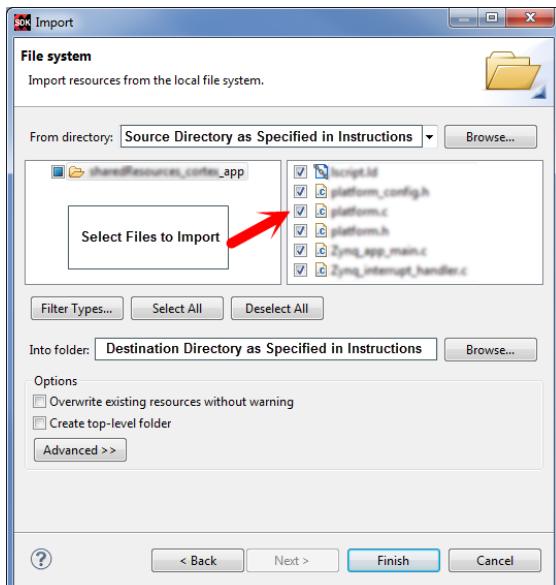


Figure 246: Selecting Resource Files

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.

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 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 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.

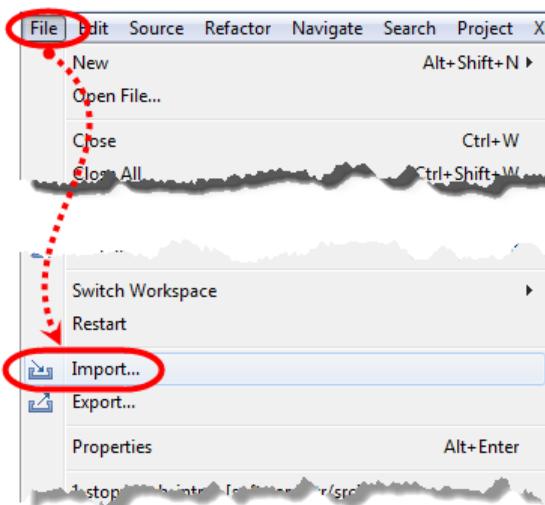


Figure 247: Accessing the Import Wizard

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).

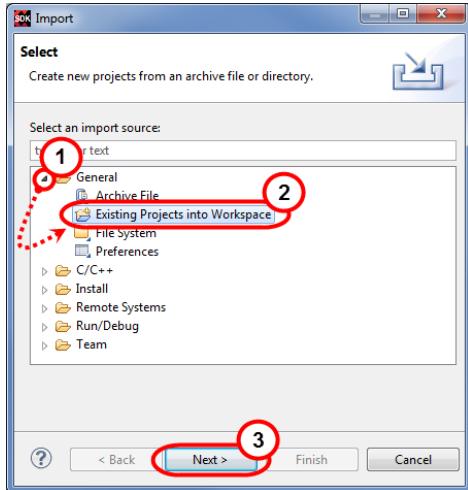


Figure 248: Import Wizard - Choosing to Import an Existing Project into the Workspace

- 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).

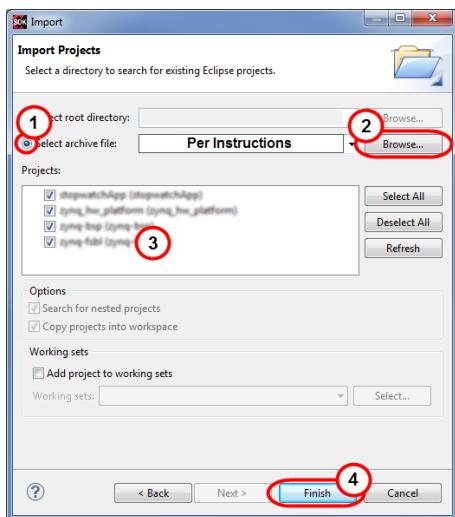


Figure 249: Import Settings for Archived Projects

- 1-1-10.** Click **Finish** to perform the importing of the project (4).

Setting Compiler Options

Compiler options describe the designer's intentions and desires 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.

- 1-1-1. Right-click the application to change/verify compiler options for and select **C/C++ Build Settings**.

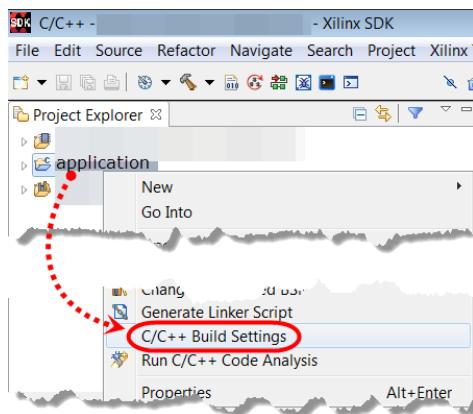


Figure 250: Accessing the Compiler Settings for a Specific Application

The C/C++ Build Settings dialog box opens.

- 1-1-2. Click **Settings** to access the specific settings for the compiler.

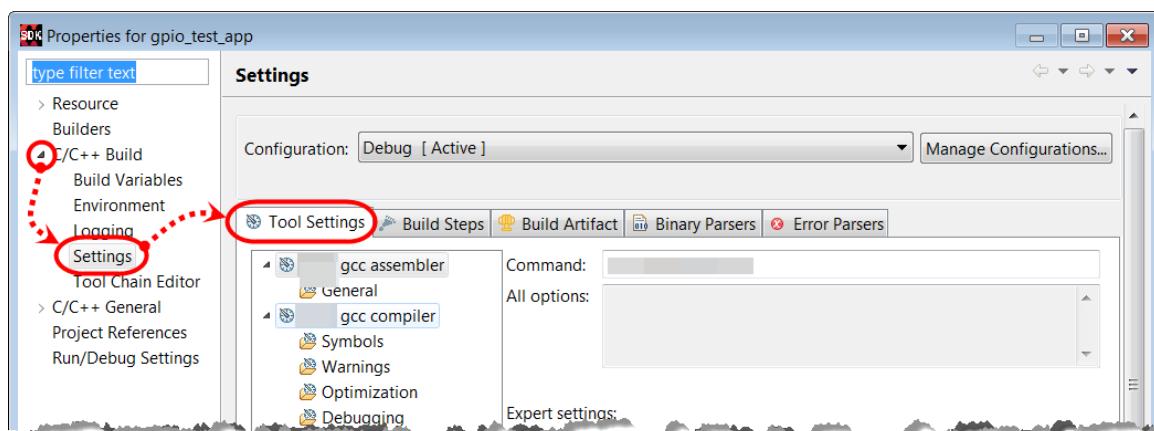


Figure 251: Accessing the Compiler Settings

- 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 Tool Settings.
- 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**.

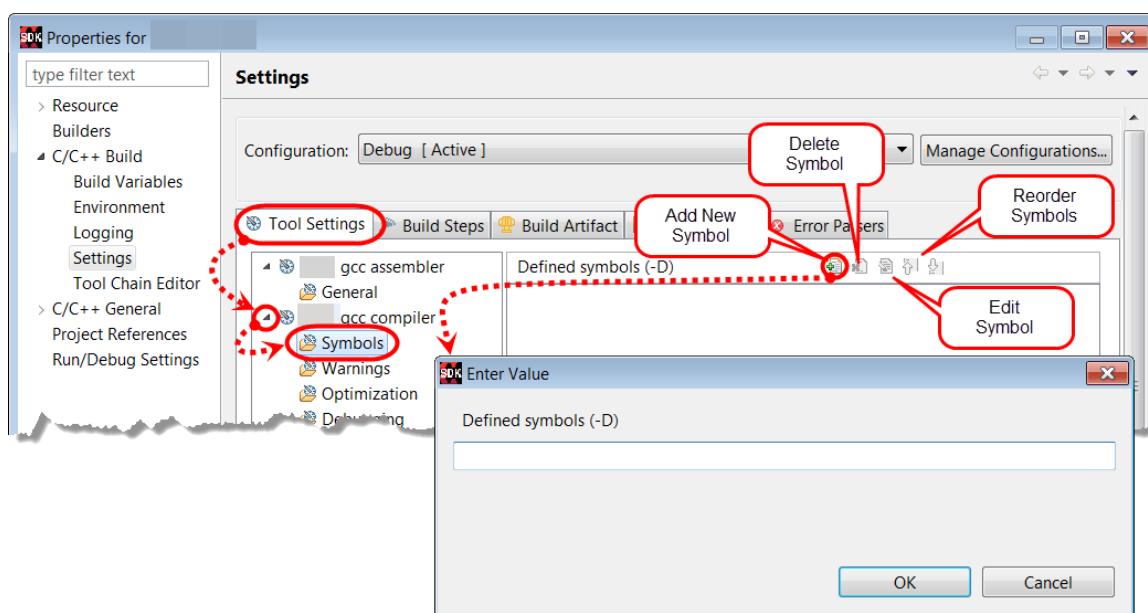


Figure 252: Creating a New Symbol

- 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**.

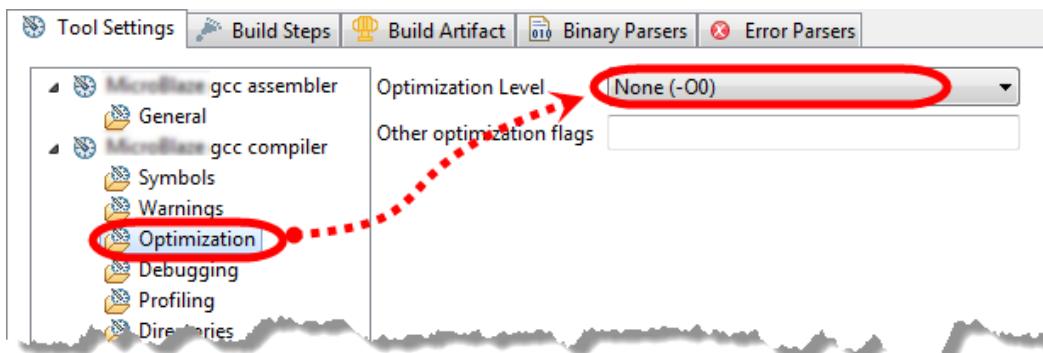


Figure 253: Setting Optimization Levels

- 1-3-5. Click **OK** to save your settings and exit.
- 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.

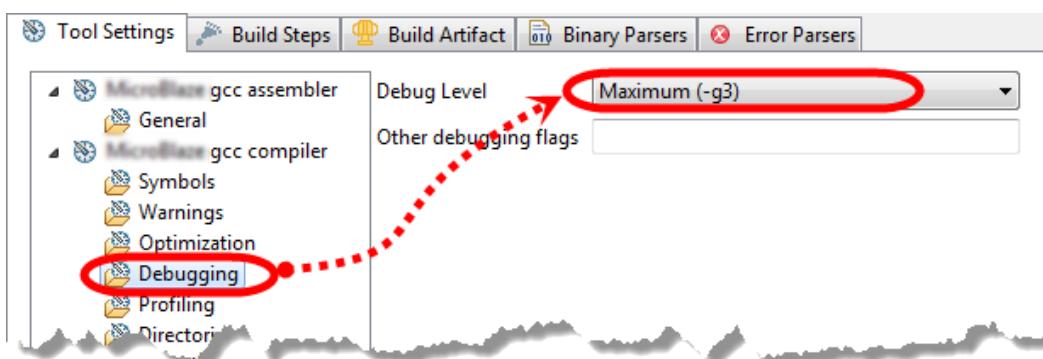


Figure 254: Setting Debug Level

- 1-4-3. Click **OK**.
- The application is rebuilt. A successful build is indicated when the program size is returned.

Setting Compiler Options

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.

- 1-1-1. Right-click the your application project name that you want to change/verify compiler options for and select **C/C++ Build Settings**.

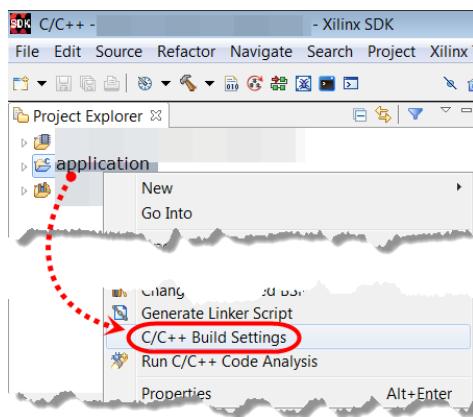


Figure 255: Accessing the Compiler Settings for a Specific Application

The C/C++ Build Settings dialog box opens.

- 1-1-2. Click **Settings** to access the specific settings for the compiler.

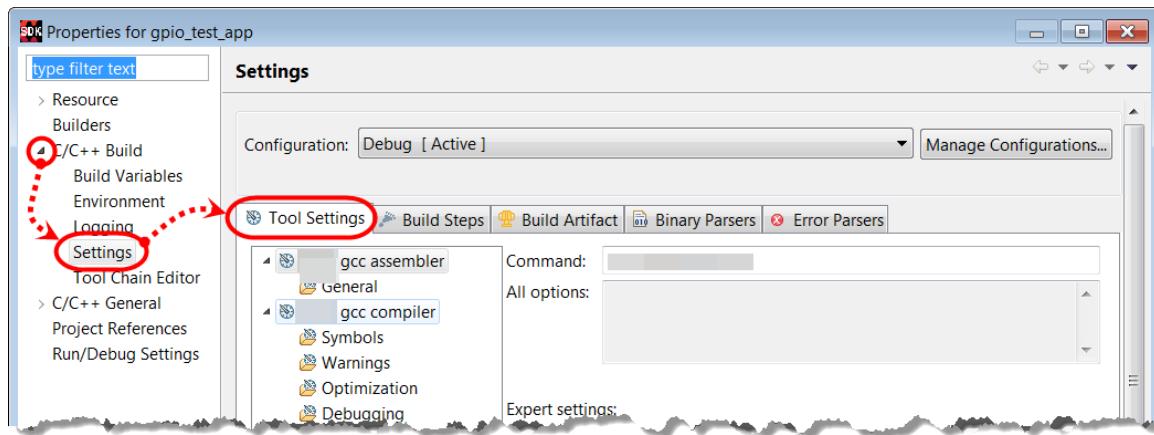


Figure 256: Accessing the Compiler Settings

- 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 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**.

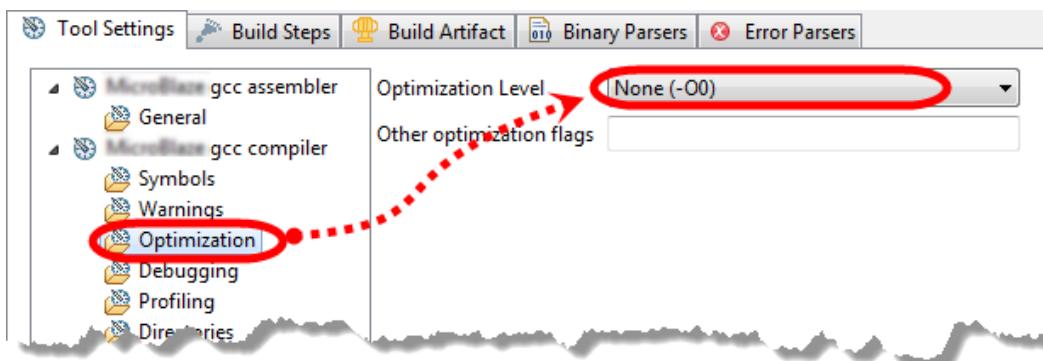


Figure 257: Setting Optimization Levels

- 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.

Opening a Source File in the Editor

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

- 1-1-1. Using the Project Explorer pane, locate **the desired source file**.

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.

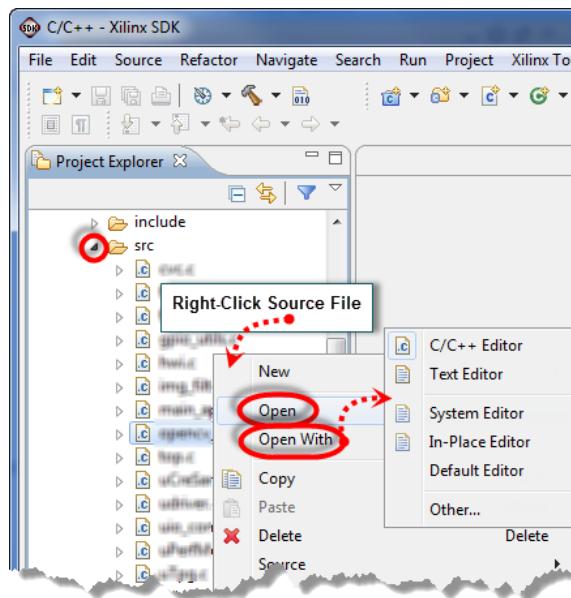


Figure 258: Opening a Source File via Right-Click

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.

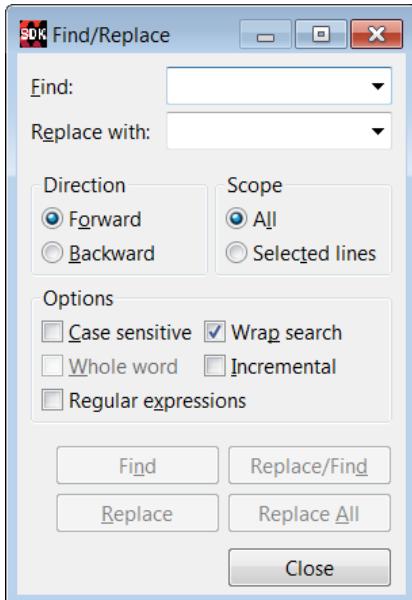


Figure 259: Default Find/Replace Dialog Box

- 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. If the source file is not open in the editor, open it now.
- 1-1-2. Right-click in the left margin and select **Show Line Numbers**.

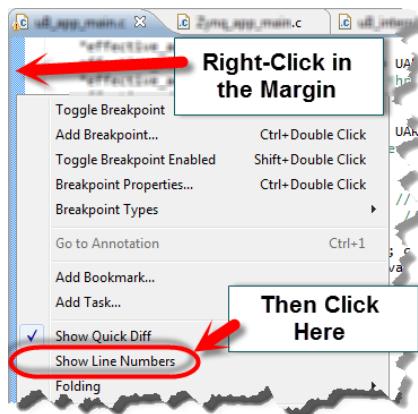


Figure 260: Enabling Line Numbering

Saving and Compiling an Application

1-1. Save and compile your open source file.

- 1-1-1. Press <Ctrl + S>, click the **Save** icon (disk) or the **Save All** icon (disk with multiple files), 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**.

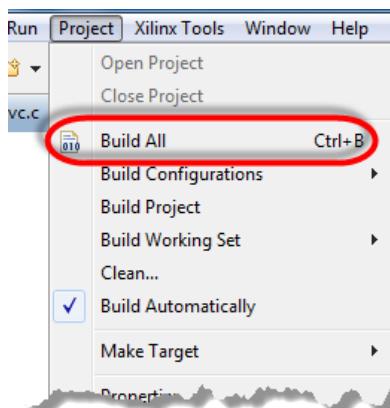


Figure 261: Selecting Build All

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**.

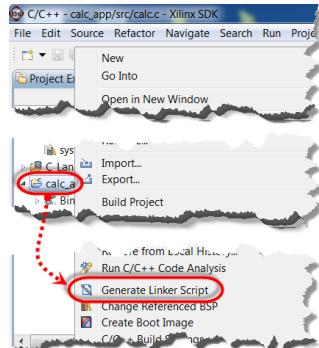


Figure 262: Accessing the Generate Linker Script Capability

The Generate Linker Script dialog box appears.

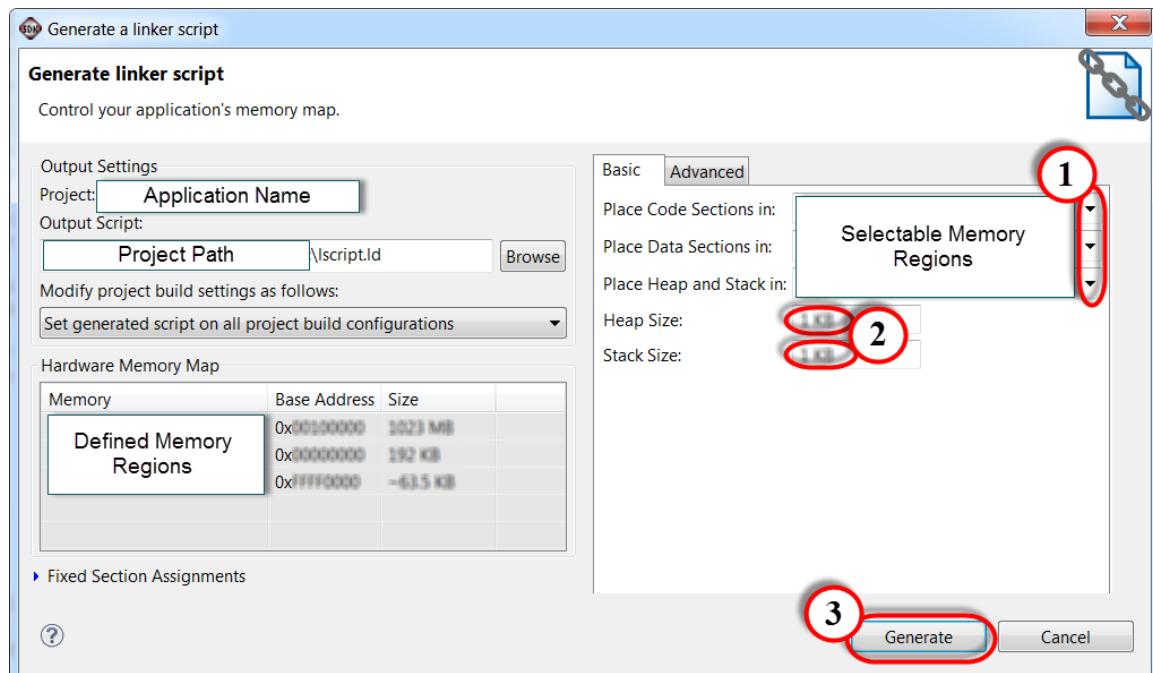


Figure 263: Common Sections of the Generate Linker Script Dialog Box

From here, you can select where your code, data, and help 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.

- 1-1-1. Select **Xilinx Tools > 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 All Programmable 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.

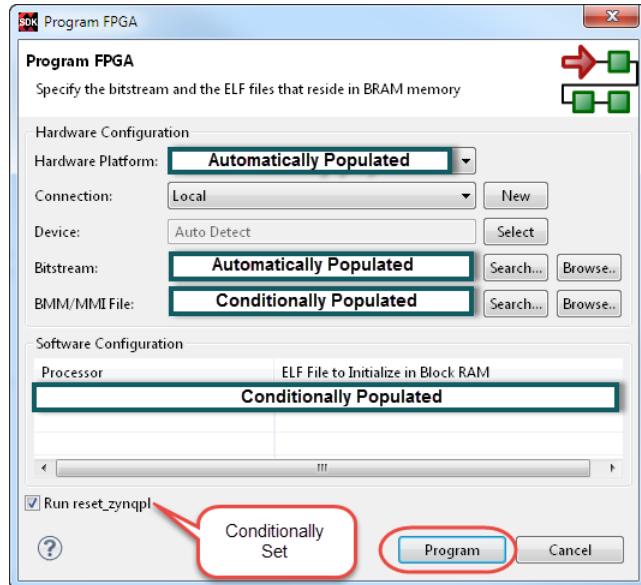


Figure 264: Programming the FPGA

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 (*****) for the **your application** application.

- 1-1-1. Right-click the **your application** software application in the Project Explorer tab (1).
- 1-1-2. Select **Create Boot Image** (2).

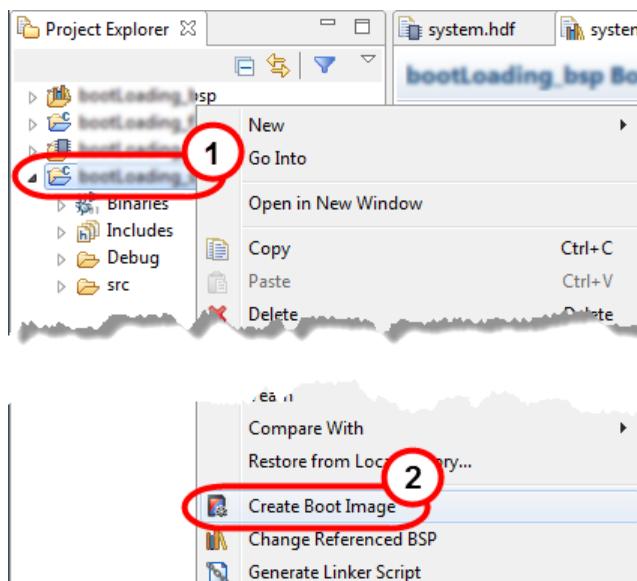


Figure 265: Selecting Create Boot Image

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 *You can either leave this field at its default setting, which will place the workspace under the Vivado Design Suite project hierarchy, or you can select your own 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 *****.

Note: To boot from Flash, the MCS file is required. If you want to boot from the SD card, the BIN file is required.

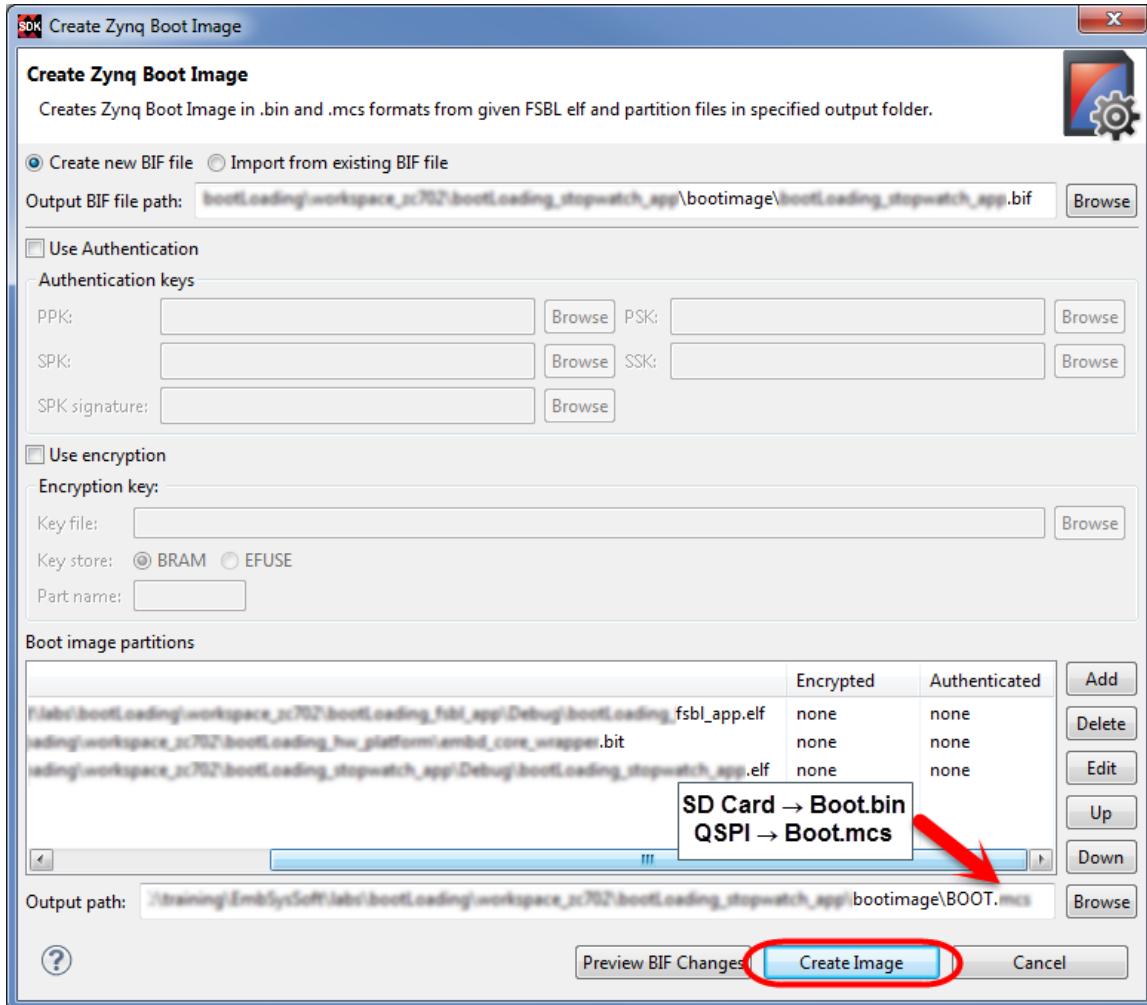


Figure 266: Create Zynq Boot Image Dialog Box

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.

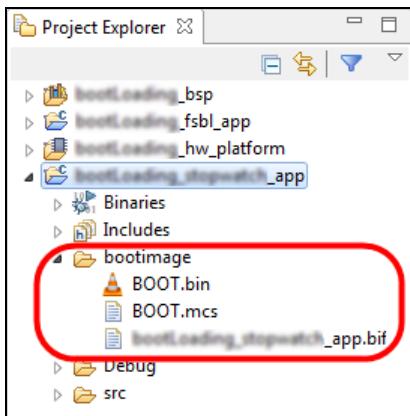


Figure 267: bootimage Folder Created

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**.

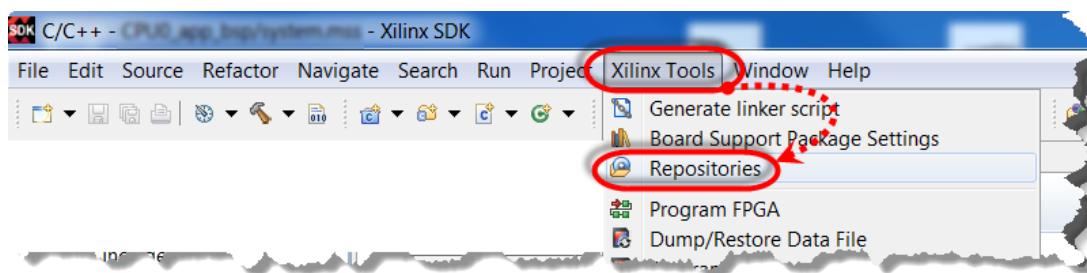


Figure 268: Selecting Repositories

The Repository Preferences dialog box opens.

- 1-1-2.** Next to the Local repositories text box, click **New**.

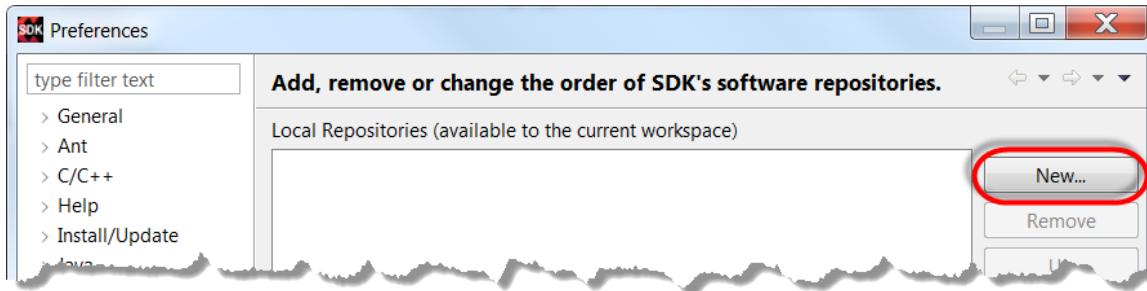


Figure 269: Creating a New Repository

- 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**.

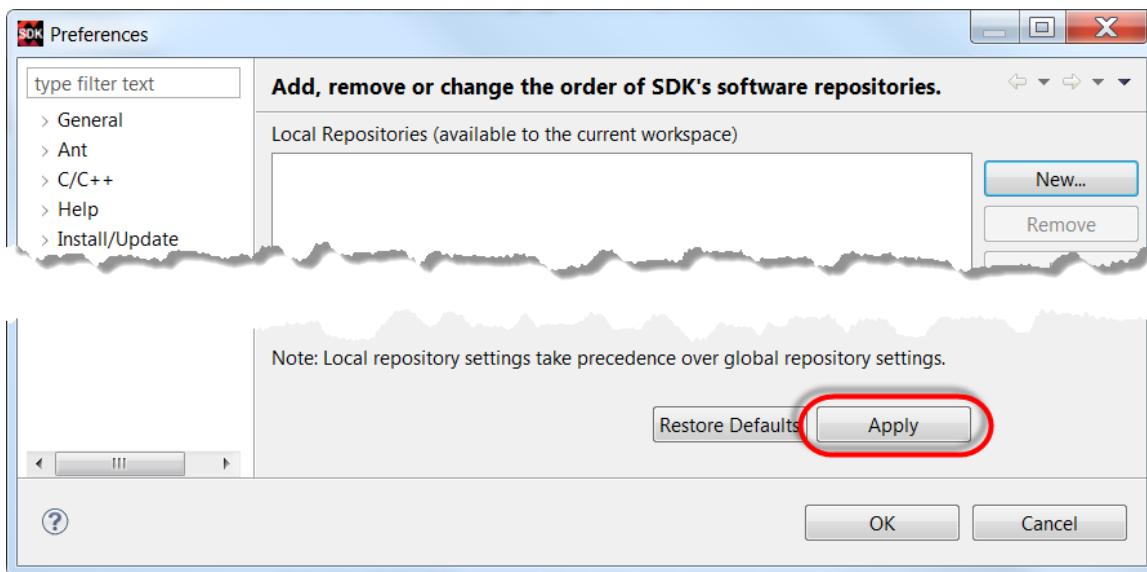


Figure 270: Applying Selections

- 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. From the Project Explorer pane, right-click **your application project name** (1).
- 1-1-2. Select **Run As** from the context menu (2).
- 1-1-3. Select **Run Configurations** (3).

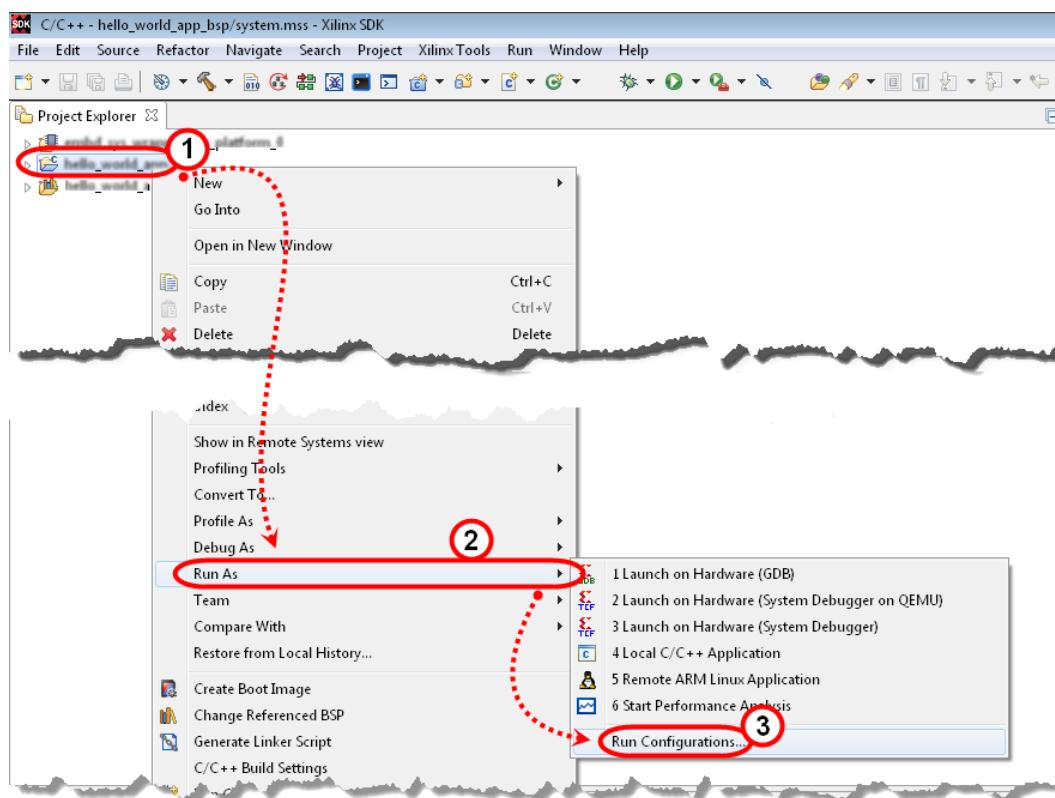


Figure 271: Creating a Run Configuration for an Application

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.

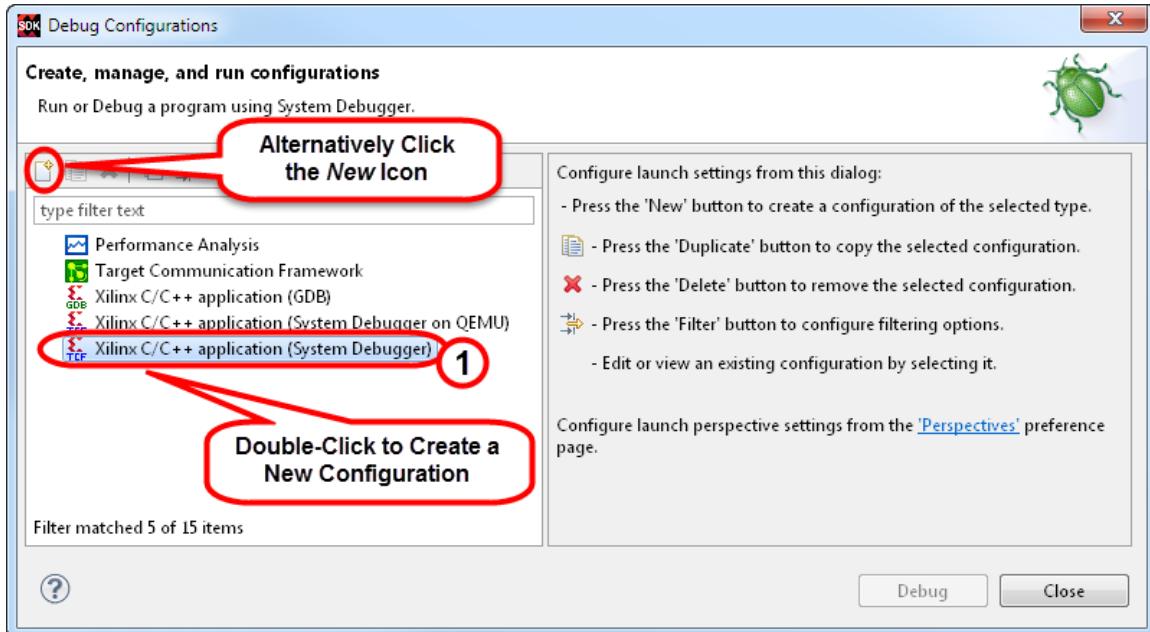


Figure 272: Creating a New System Run/Debug Configuration

A new Run Configuration is created named ***your application project name_Debug*** (2).

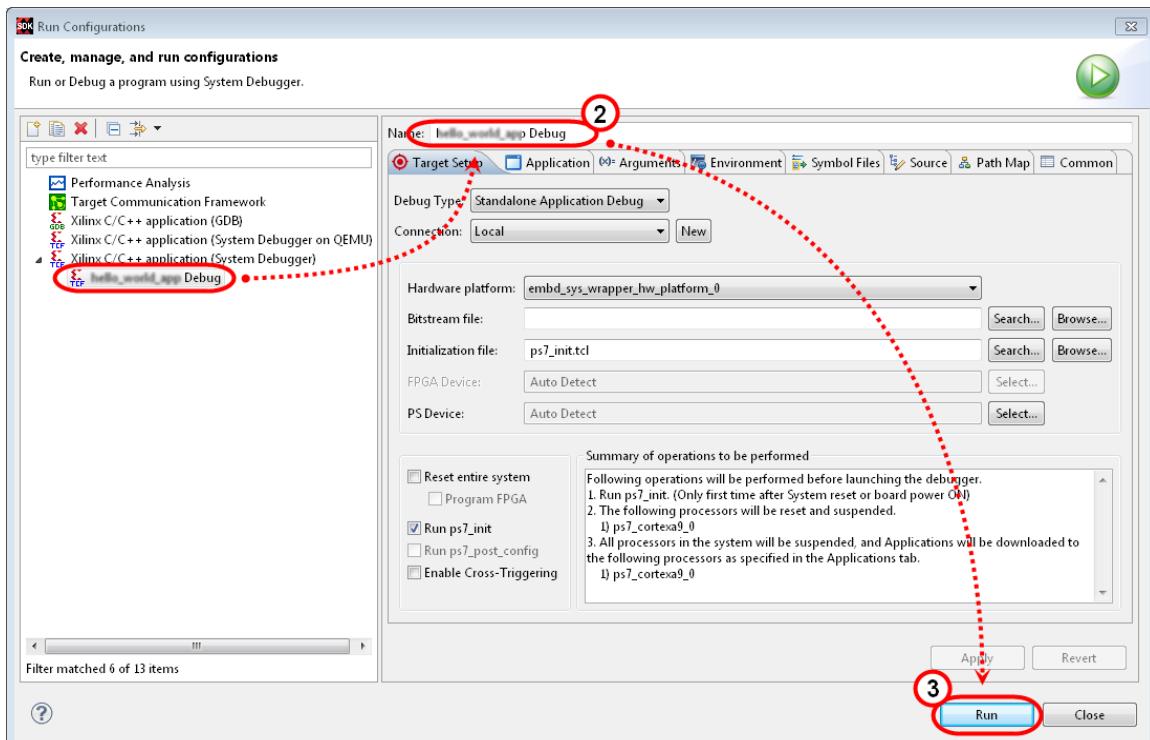


Figure 273: Run Configurations Dialog Box (Zynq AP SoC)

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.

Note that figure above is for the Zynq All Programmable SoC. The primary difference when using the MicroBlaze Processor is that the "Initialization file" field will not be active. If you are using a Zynq All Programmable SoC, this field is active and pre-populated.

- 1-1-5.** Click **Run** to close the dialog box and launch the software application on the hardware evaluation board (3).

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 the **your application** project in the Project Explorer window.

1-1-3. Select **Run As > Run Configurations**.

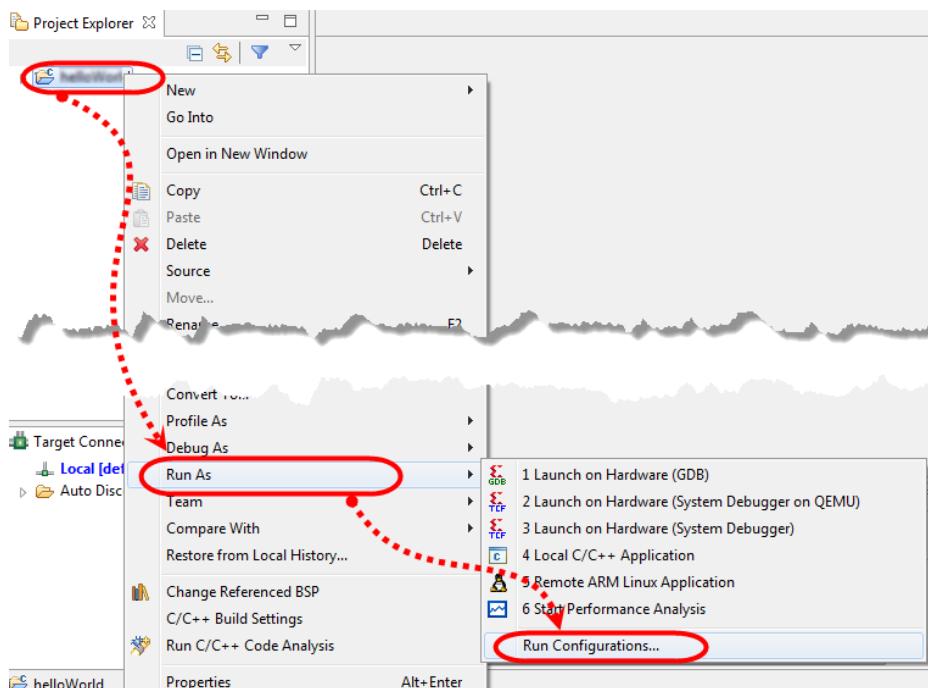


Figure 274: Selecting Run Configurations

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 **192.168.1.10** in the Host field.
- 1-2-6. Enter **1534** in the Port field.

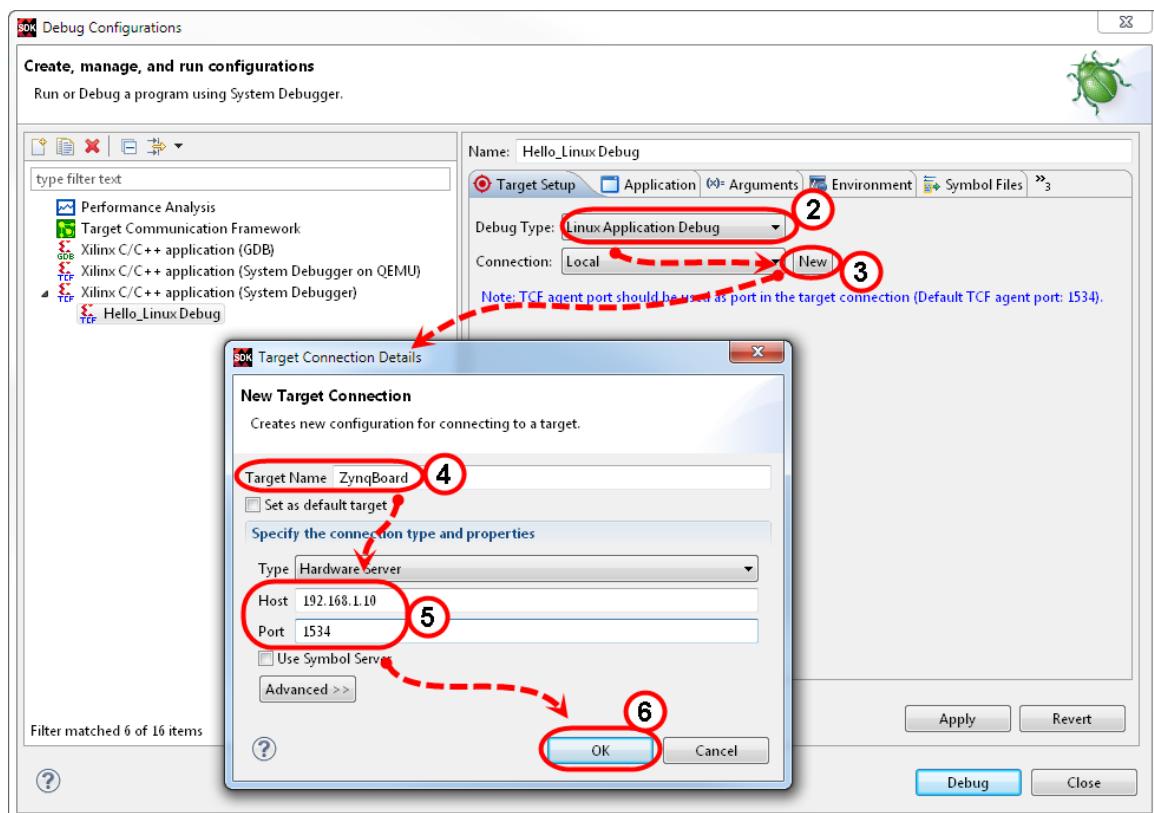


Figure 275: Selecting the Debug Type and Connection

- 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. Select the local file path to be

*C:\training****\labs\workspace_linux_<target_board>\your application\Debug\your application.elf.*

Where *<target_board>* is *zc702* for the ZC702 board and *zed* for the ZedBoard.

1-3-4. Enter **/tmp/your application.elf** in the Remote File Path field.

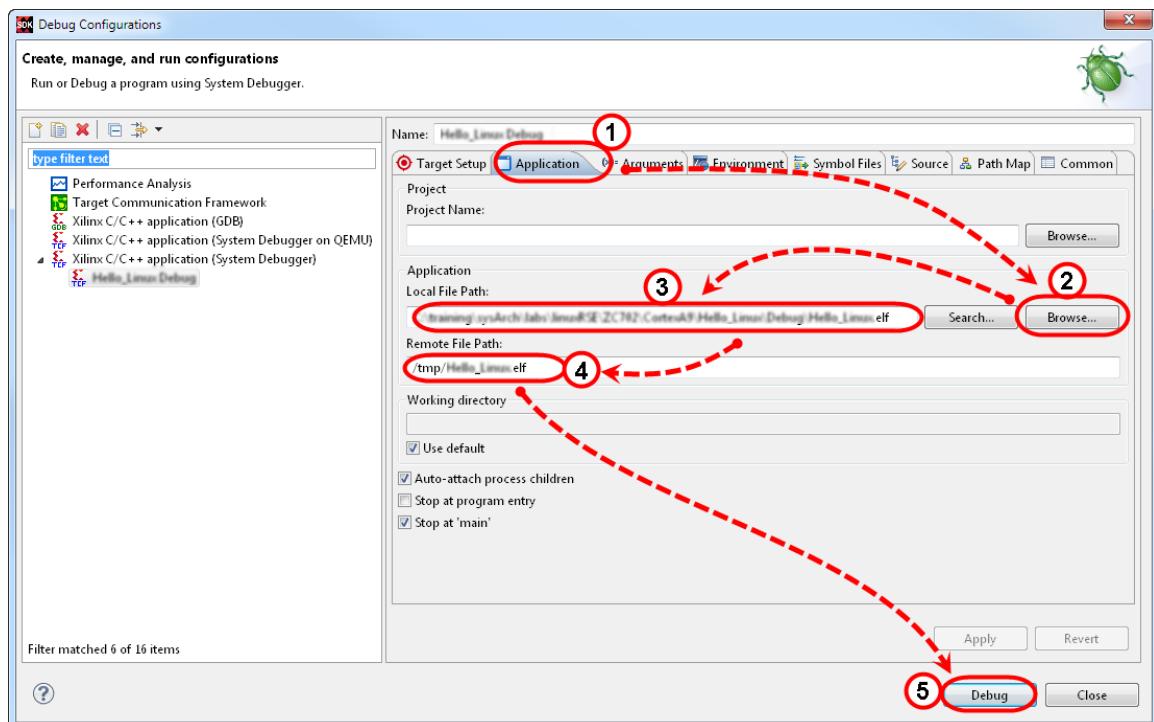


Figure 276: Selecting the Local File Path and Remote File Path

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 (

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

Setting Up a GDB Debug Configuration	207
Enabling Instruction Stepping	211
Launching an Existing Debug Configuration	212
Launching an Existing Run Configuration	213
Setting Up a System Debugger Debug Configuration	213
Setting Up a Debug Configuration (System Debugger – Linux)	215
Launching System Debug of a Software Application on Hardware	219
Enabling Cross Trinngering in a GDB Debug Configuration	220
Controlling Execution	222
Managing Breakpoints	223
Monitoring Variables	229
Managing Expressions	230
Changing the Value of a Variable	232
Using Expressions	234
Setting Up the Linux Console for Debugging	235

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).

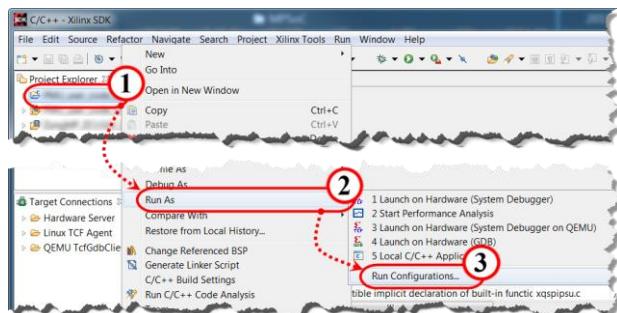


Figure 277: Accessing the Debug Configuration Dialog Box

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).

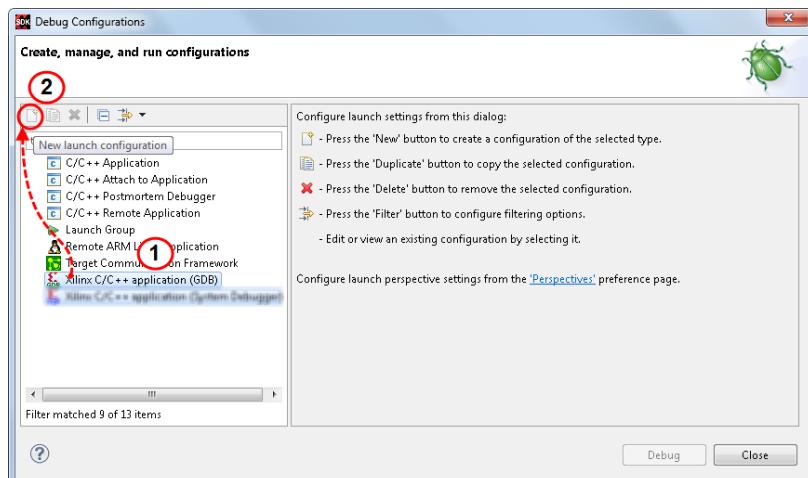


Figure 278: Creating a New Debug Configuration

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.

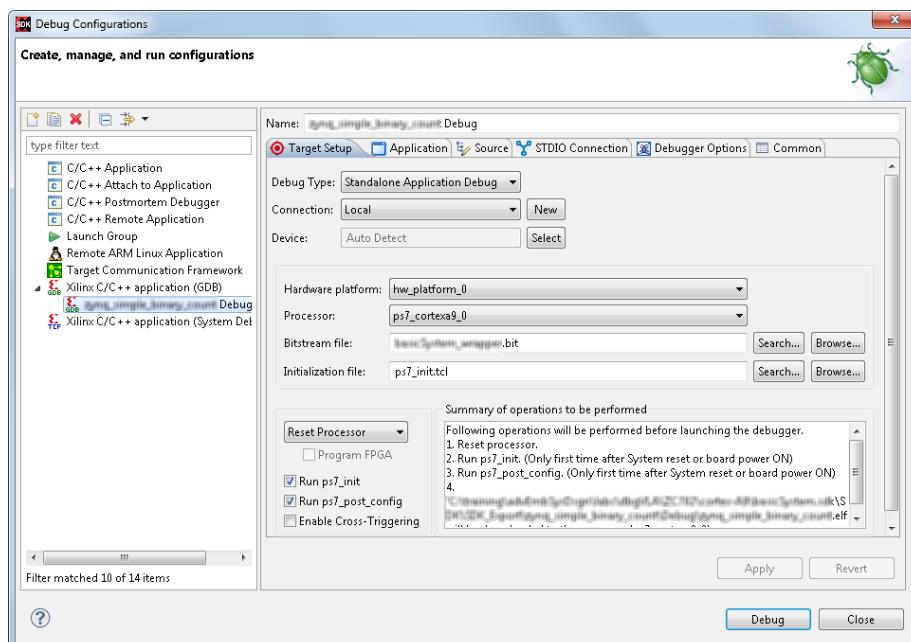


Figure 279: Creating the New Debug Configuration

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:

- Single Cortex A-9 processor systems
- The Cortex A-9 processor designated as the master processor in a system

Typical systems that use "Reset Processor Only" are:

- Any slave processor in a multi-processor system

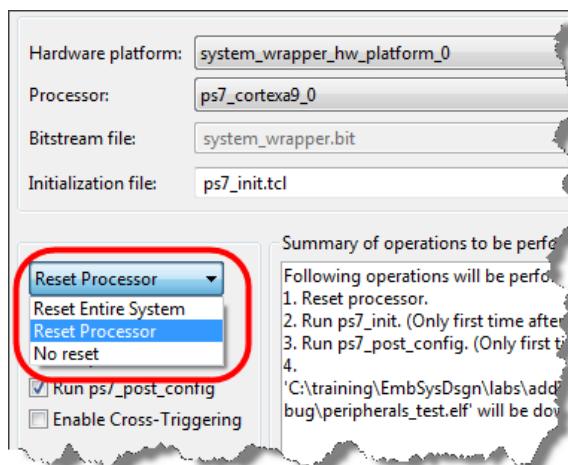


Figure 280: Selecting Reset Behavior

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.

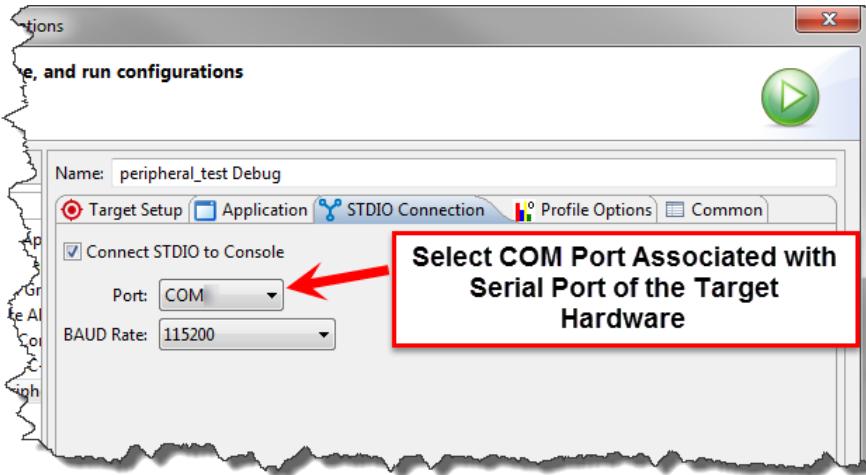


Figure 281: Directing the STDIO to the Console

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**.

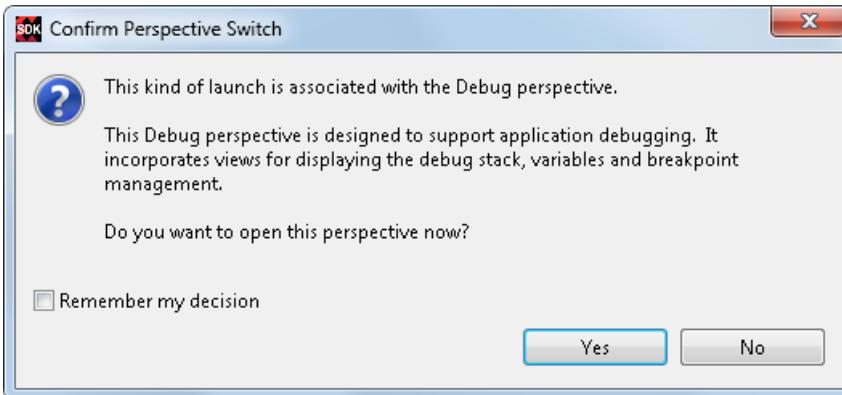


Figure 282: Confirming Switch of Perspective

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 283: Instruction Stepping

- 1-1-1. Click the **Instruction Stepping Mode** () 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 (1) to select from the drop-down list (1).
- 1-1-2. Select the appropriate existing debug configuration to launch a new debug session (2).

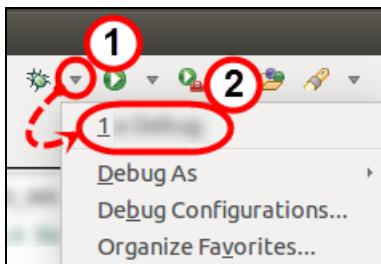


Figure 284: Launching an Existing Debug Configuration

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. If the Confirm Perspective Switch dialog opens, informing you that the perspective view is being changed (from C/C++ to Debug) click **Yes**.

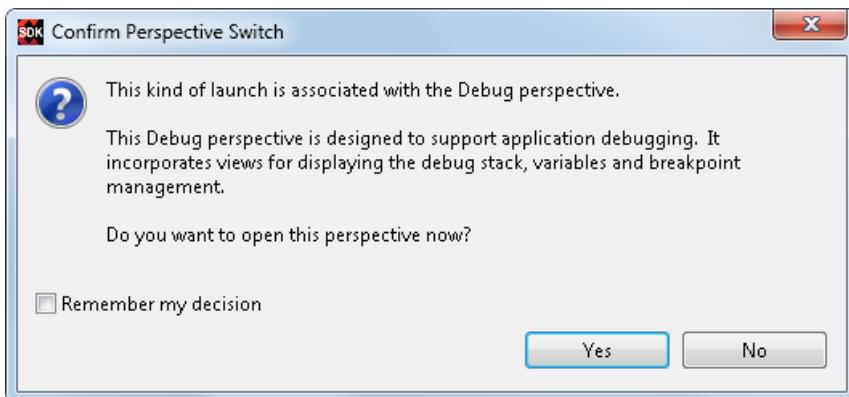


Figure 285: Confirming Switch of Perspective

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.

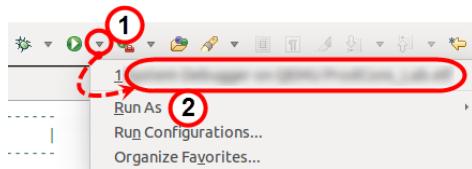


Figure 286: Launching an Existing Run Configuration

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. 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).

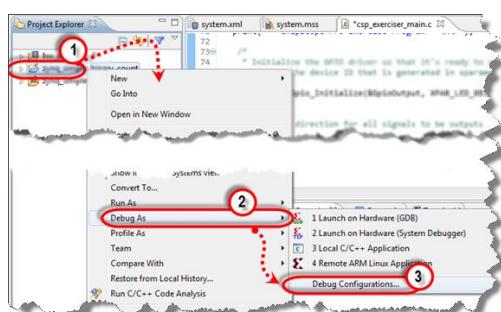


Figure 287: Creating a Debug Configuration

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).

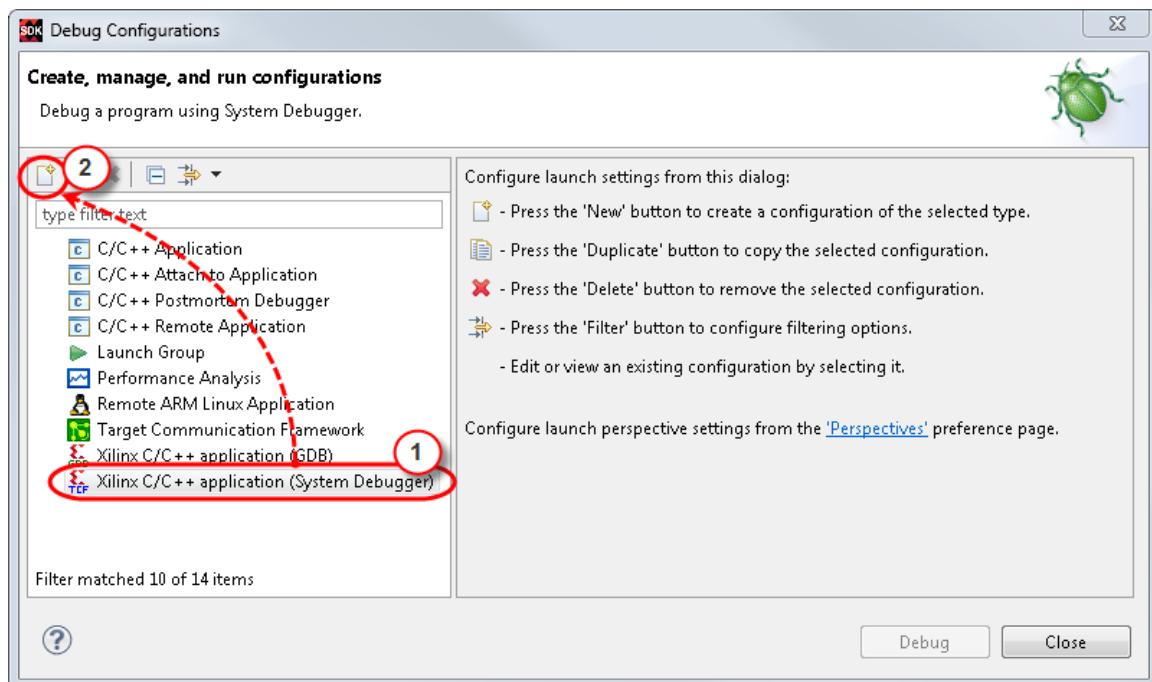


Figure 288: Creating a New Debug Configuration

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.

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.

If the Confirm Perspective Switch dialog box appears, click **Yes**.

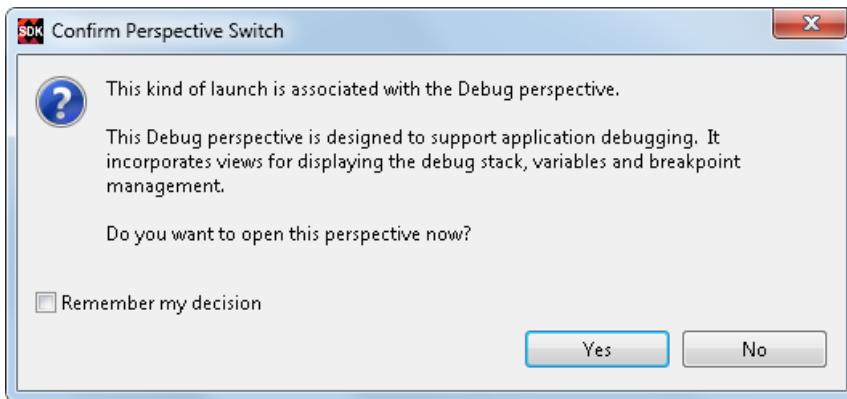


Figure 289: Confirming Switch of Perspective

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 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. A project can have multiple configurations to save various setups of this information.

- 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.

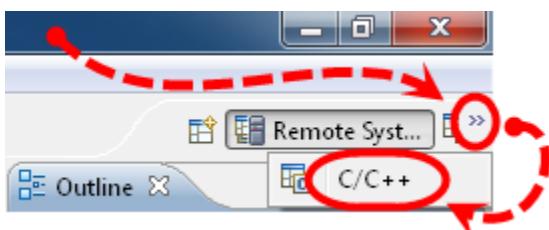


Figure 290: Changing Perspective

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).

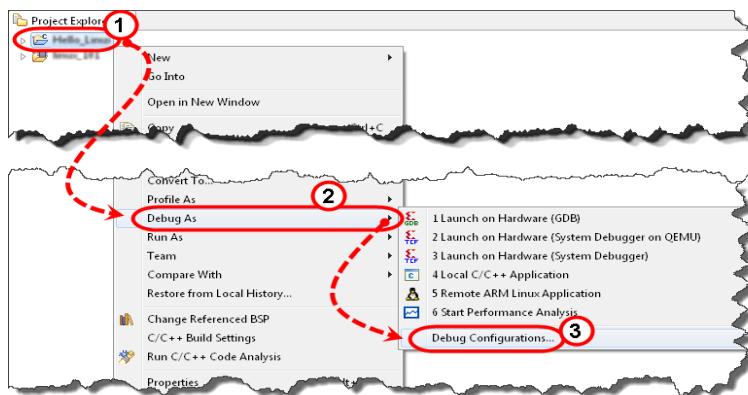


Figure 291: Selecting Debug Configurations

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.

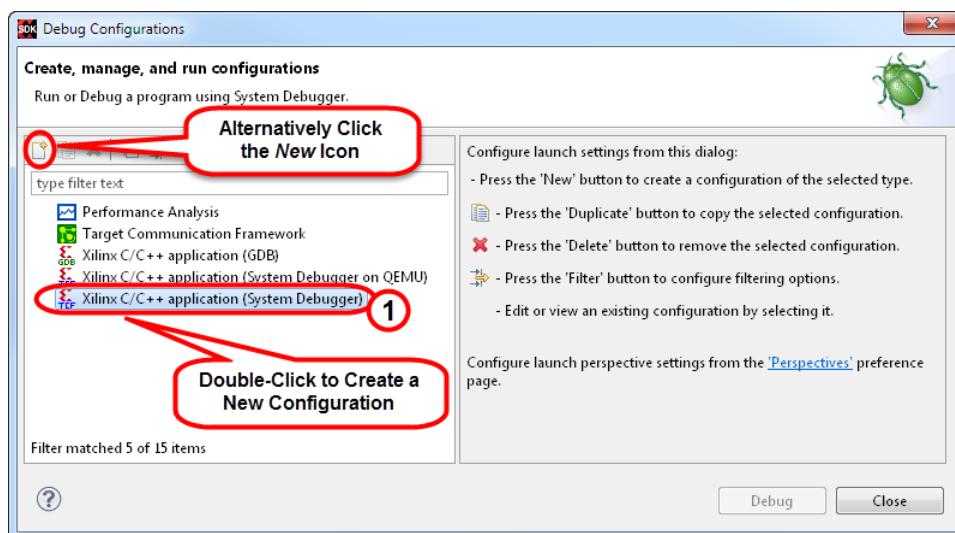


Figure 292: Creating a New System Run/Debug Configuration

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).

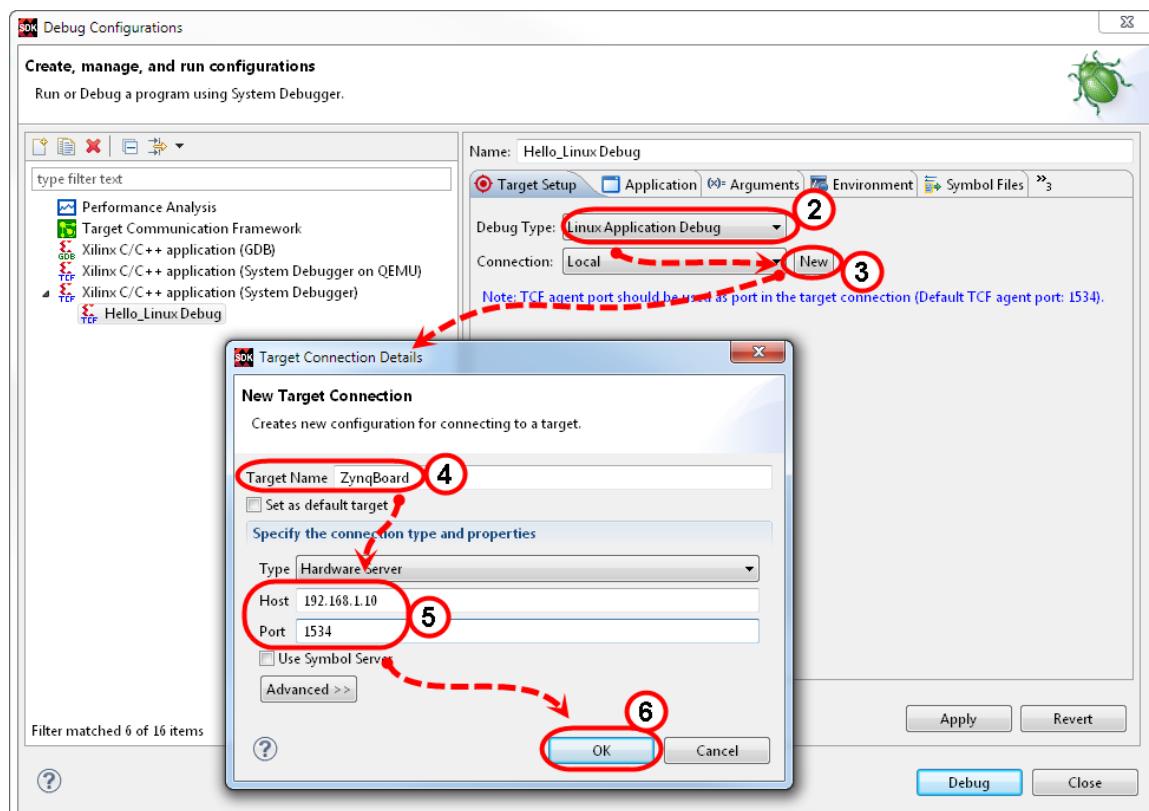


Figure 293: Selecting the Debug Type and Connection

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**\labs\lab name\your board\your processor\your application\Debug\your application.elf (3).**

Where <target_board> is zc702 for the ZC702 board and zed for the ZedBoard.

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).

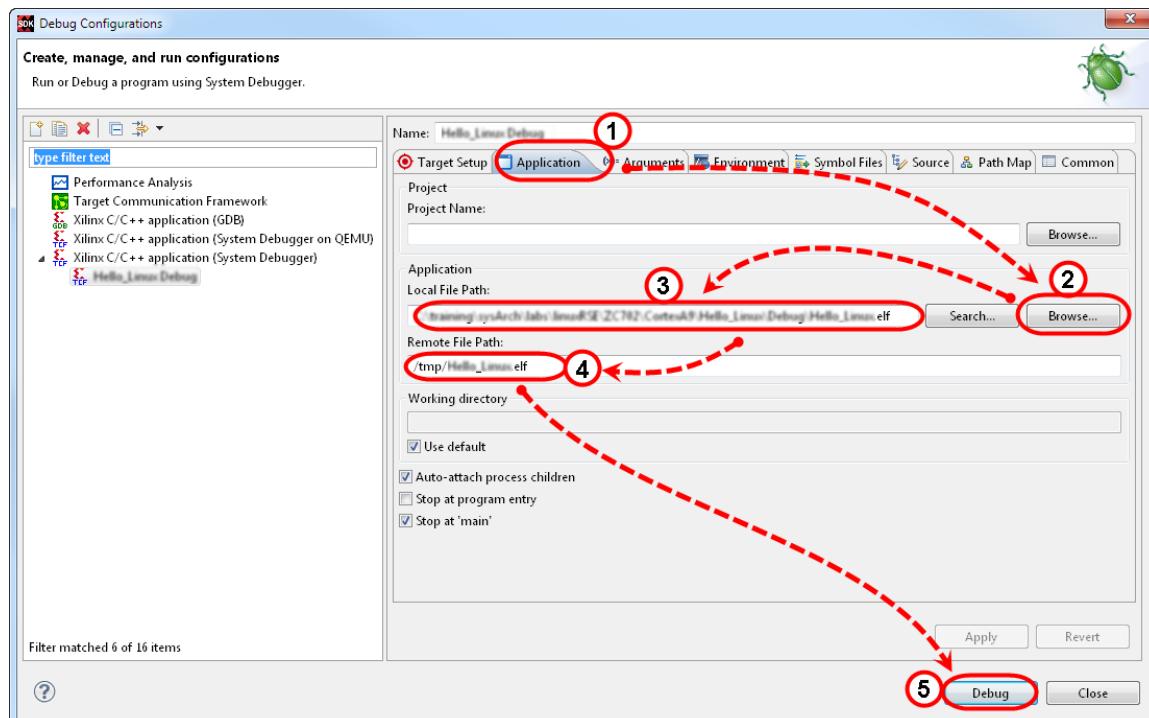


Figure 294: Selecting the Local File Path and Remote File Path

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.

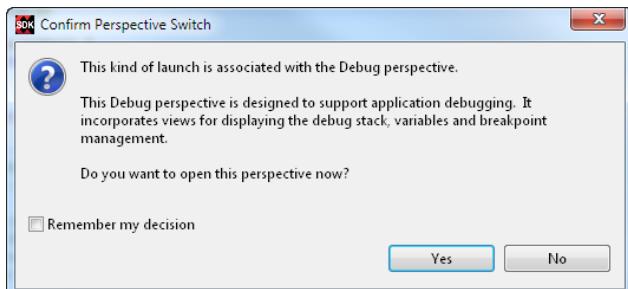


Figure 295: Confirming Perspective Switch

The Debug perspective opens.

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. From the Project Explorer pane, right-click the application project that you want to launch (1).

1-1-2. Select **Debug As** (2).

1-1-3. Select **Launch on Hardware (System Debugger)** (3).

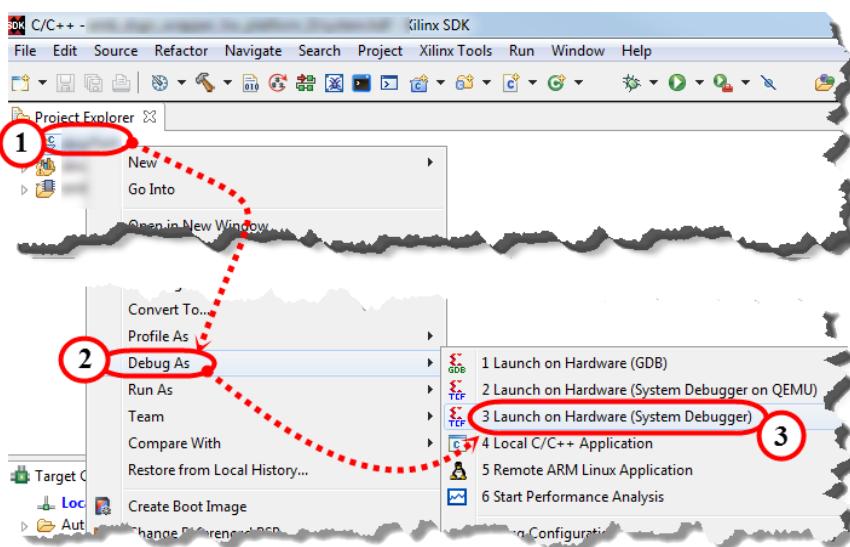


Figure 296: Selecting Launch on Hardware

The Confirm Perspective Switch dialog box opens.

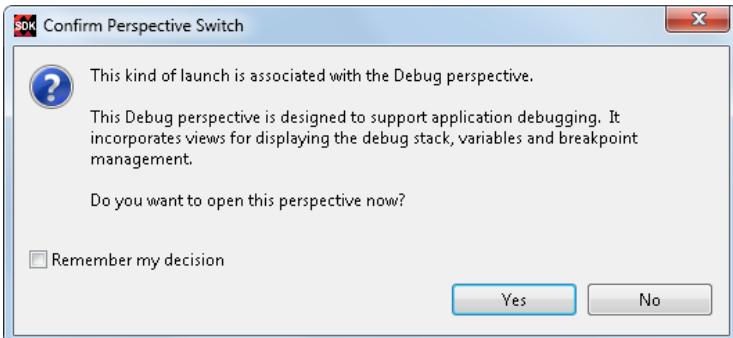


Figure 297: Confirming Switch of Perspective

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 (System Debugger) debug configuration. This is typically performed from the C/C++ perspective.

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 edit the Debug configuration for (1).
- 1-1-2.** Select **Debug As** (2).
- 1-1-3.** Select **Debug Configurations** (3) to open the configurations manager.

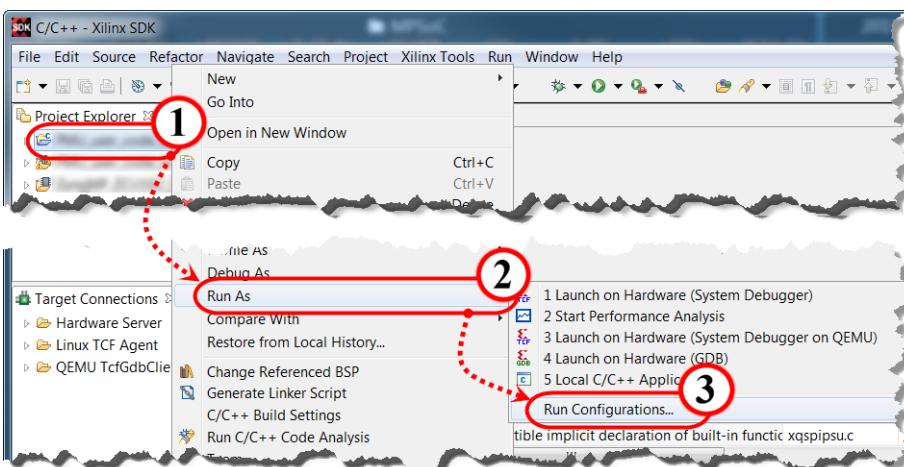


Figure 298: Accessing the Debug Configuration Dialog Box

The Debug Configurations dialog box opens.

- 1-1-4. Expand (1) **Xilinx C/C++ application (System Debugger)** (1) to access the existing debug configurations.
- 1-1-5. Select **the Debug configuration**.
- 1-1-6. Select the **Target Setup** tab (2), which contains the enable box for cross-triggering.
- 1-1-7. Select the **Enable Cross-Triggering** option (3).
- 1-1-8. Click **Apply** (4) and then **Debug** (5) to save the cross-triggering option and launch the debugger.

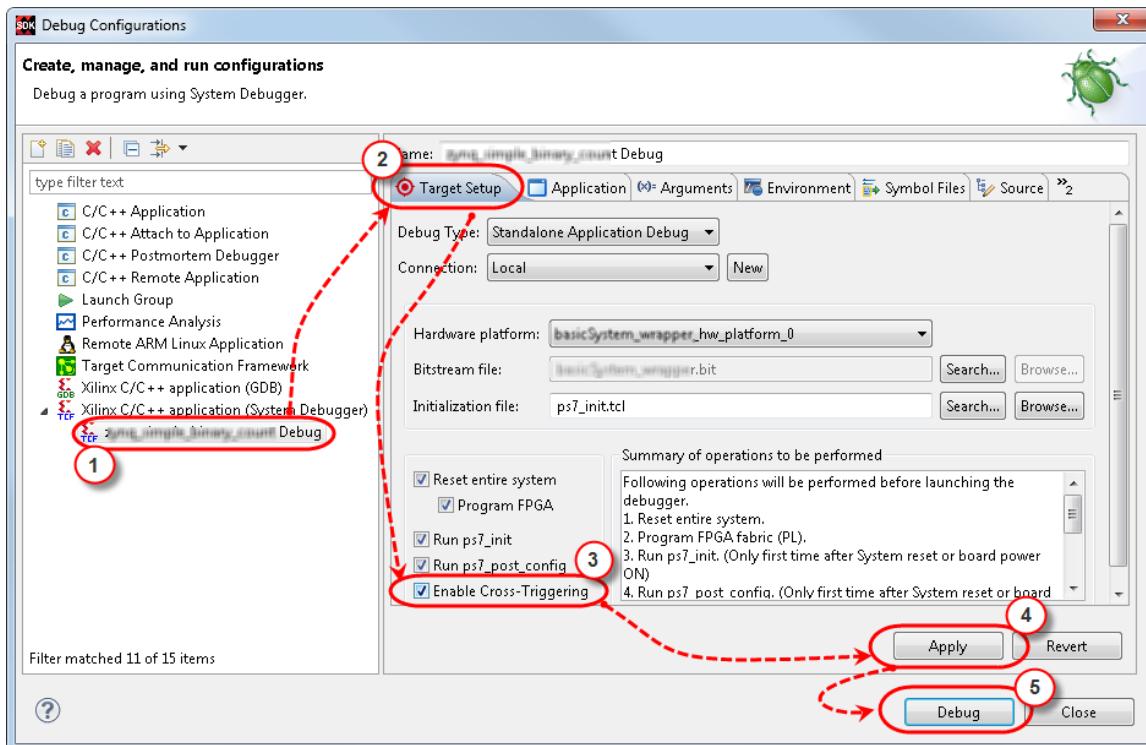


Figure 299: Enabling Cross-Triggering

- 1-1-9. Click **OK** to re-launch the Debug perspective view.
- 1-1-10. If the Processor in use dialog box appears, click **Yes** to terminate the previous launch.

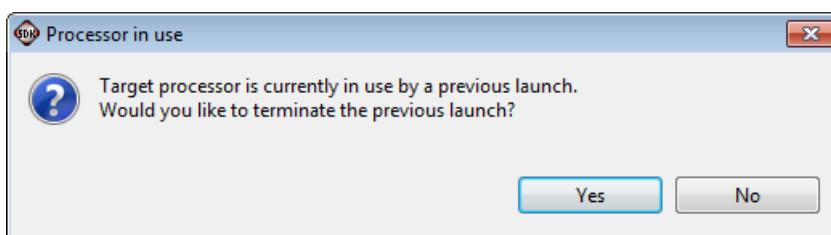


Figure 300: Terminating the Previous Launch

If the Confirm Perspective Switch dialog box appears, click **Yes**.

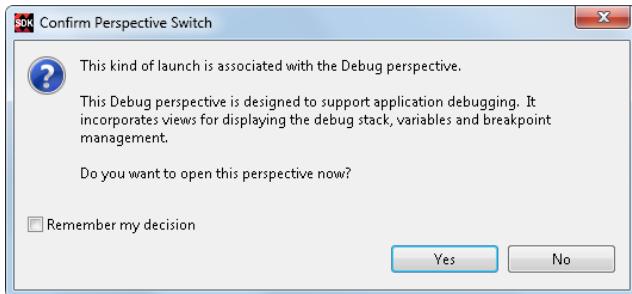


Figure 301: Confirming Switch of Perspective

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	Shortcut	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.

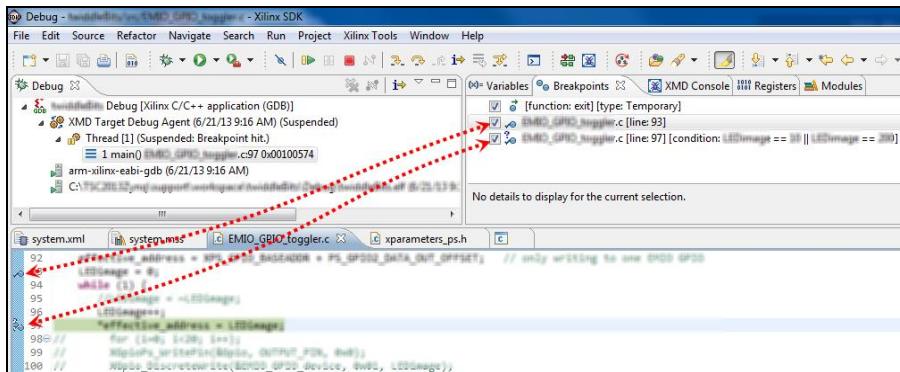


Figure 302: Relationship Between Breakpoints and the Breakpoint Window

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.

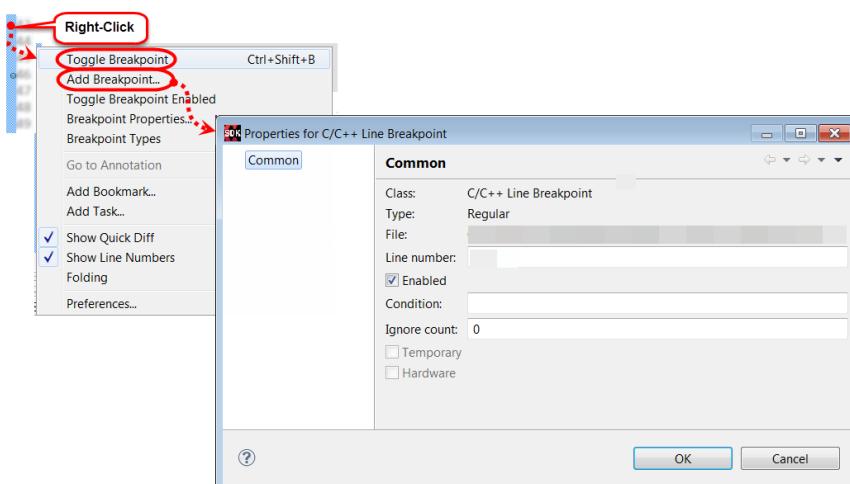


Figure 303: Adding a Simple or Conditional Breakpoint

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**.

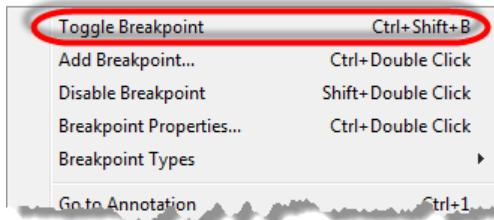


Figure 304: Toggling a Breakpoint into and out of Existence

-- 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.

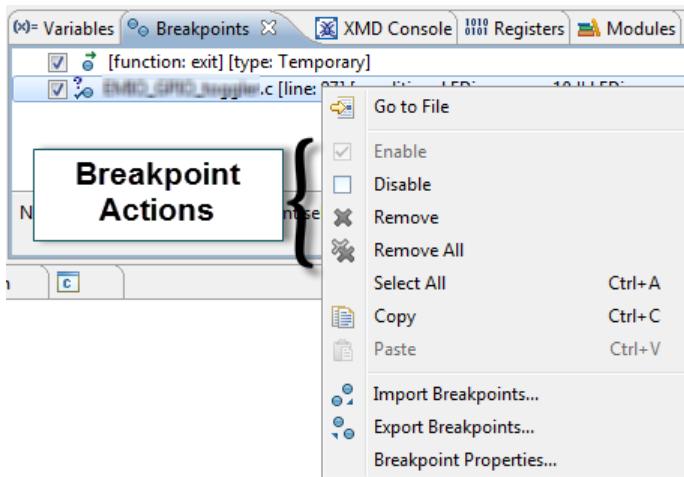


Figure 305: Actions from the Breakpoint Window

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.

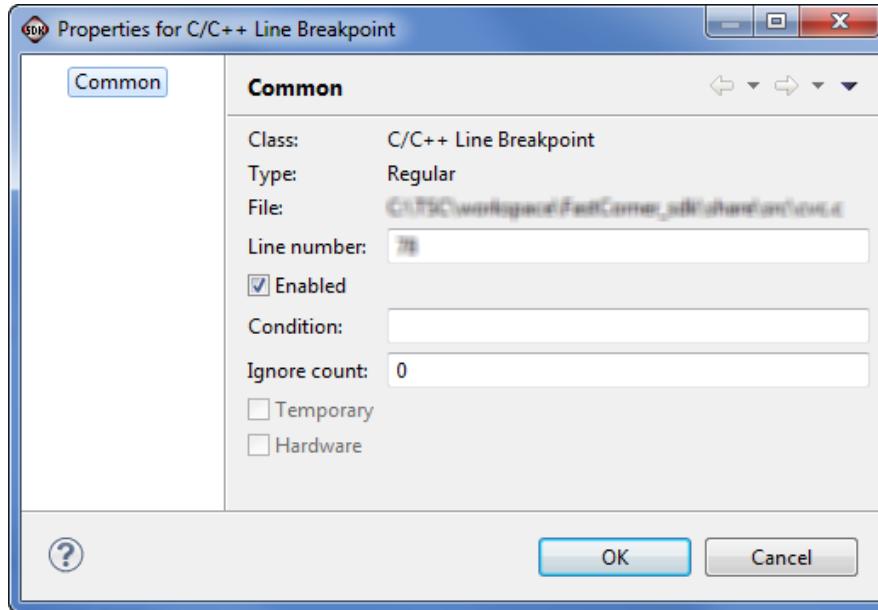


Figure 306: Breakpoint Properties Dialog Box

- 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**.

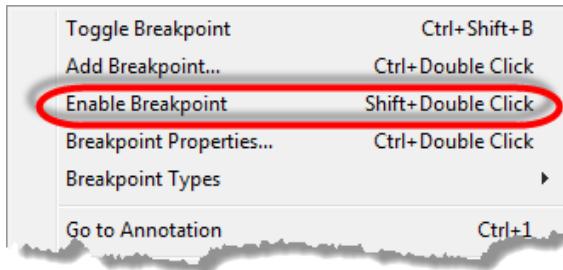


Figure 307: Enabling Breakpoint from the Pop-Up Menu

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**.

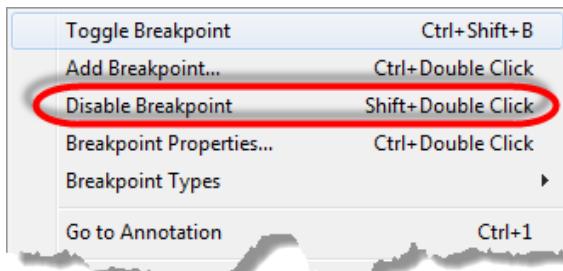


Figure 308: Disabling a Breakpoint from the Pop-Up Window

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.

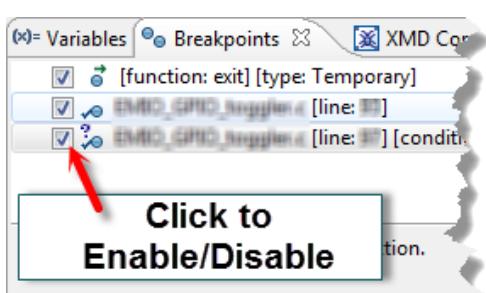


Figure 309: Enabling and Disabling Breakpoints

Running to a Breakpoint

Breakpoint(s) must be set before launching a run. Once the application is started, the Resume, Pause, and Stop icons become active.

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
- 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.

- 1-1-1. If the Variables tab is not visible, select **Window > Show View > Variables**.

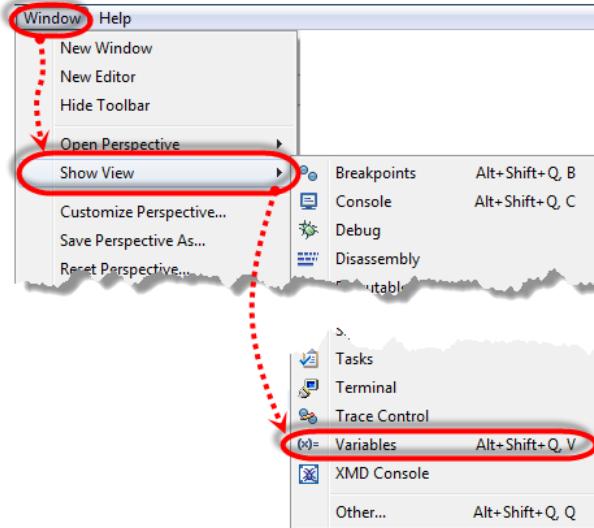


Figure 310: Making the Variables Tab Visible

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		'\000'
Hex: 00, Dec: 0, Oct: 0		
Bin: 0000,0000		
Size: 1 byte, Type: char		
Address: 0xffffdce6d1		

Figure 311: 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**.

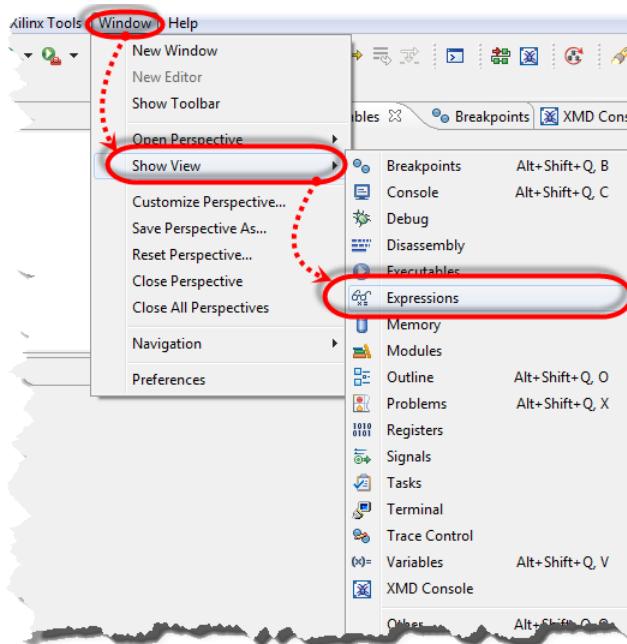


Figure 312: Accessing the Expressions Window from the Debug Perspective

1-2. Create (add) a new expression.

- 1-2-1. Click the green Plus icon (+) in the Expressions tab.

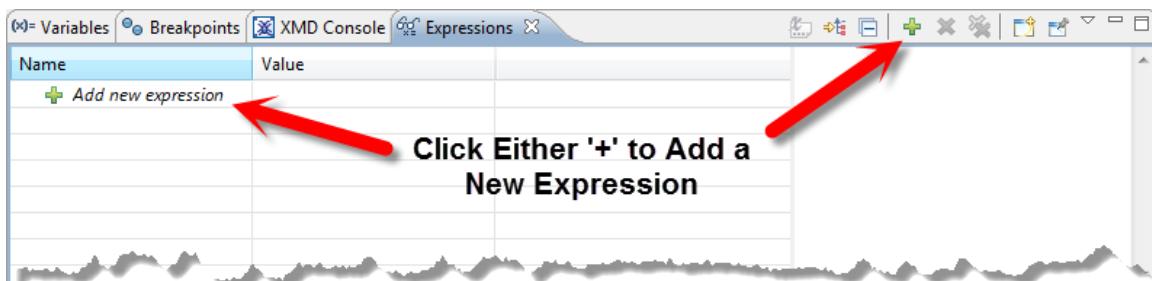


Figure 313: Adding a New Expression

- 1-2-2. Type in the new expression.

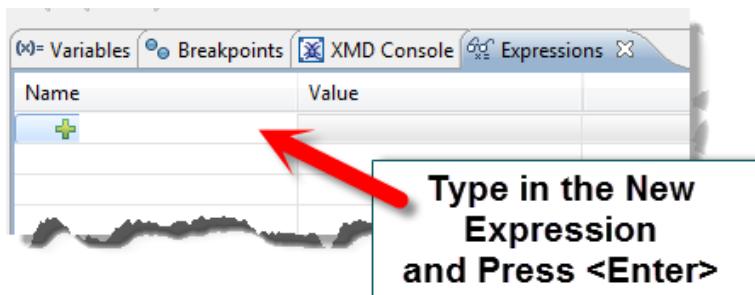


Figure 314: Entering the New Expression

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**.

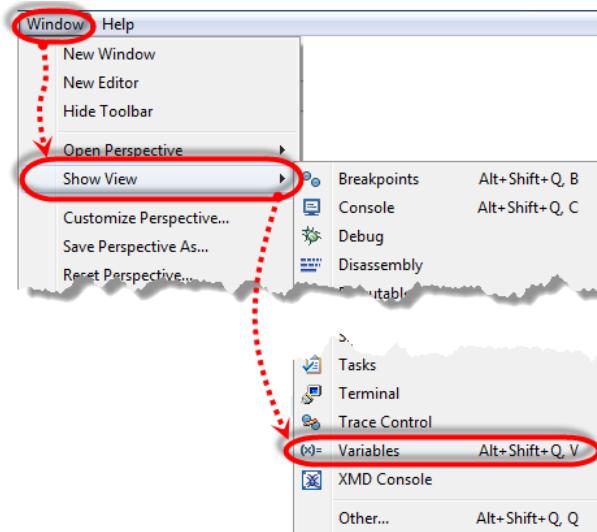


Figure 315: Making the Variables Tab Visible

This exposes the Variables tab.

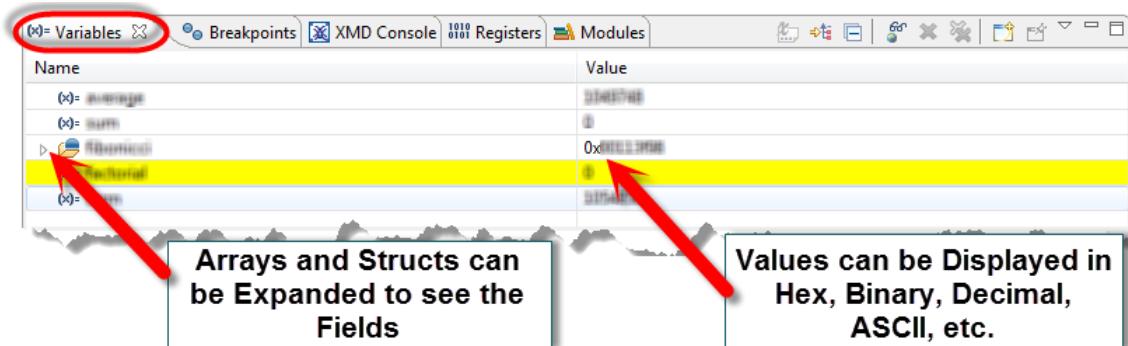


Figure 316: Example View of the Variables Tab

Variables are listed alphabetically.

- 1-1-2. Change the radix of a variable by right-clicking the variable name and selecting **Change Value**.

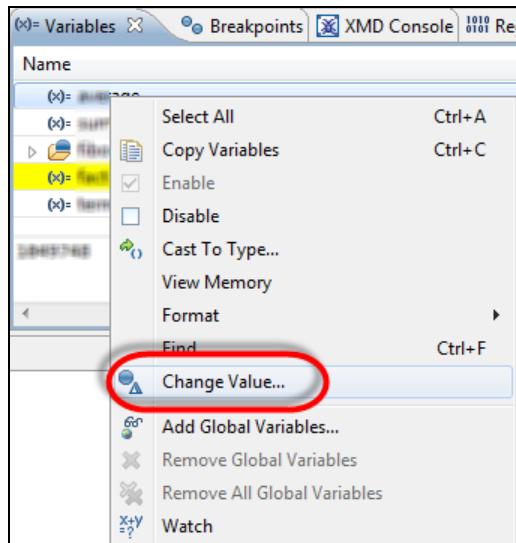


Figure 317: Changinge the Value of a Variable

- 1-1-3. Enter the new value of the variable.

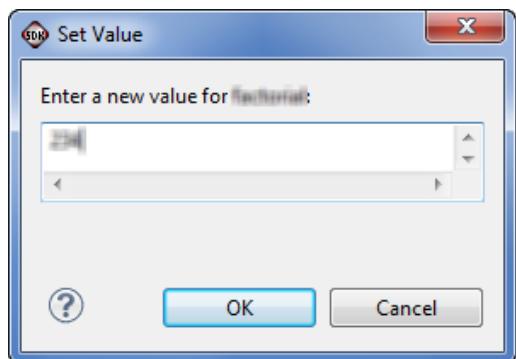


Figure 318: Setting the Value of a Variable

- 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**.

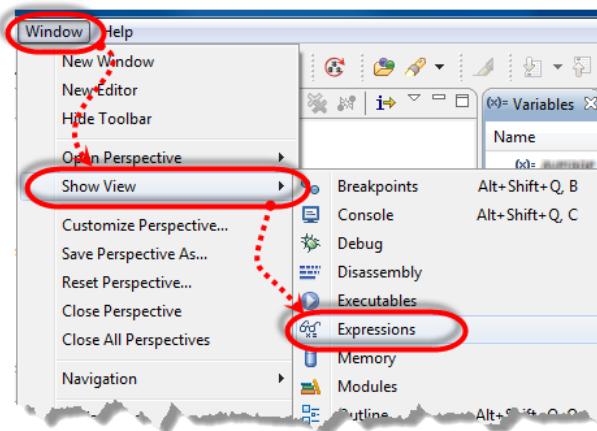


Figure 319: Making the Expressions Tab Visible

- 1-1-2. Select the **Expressions** tab.

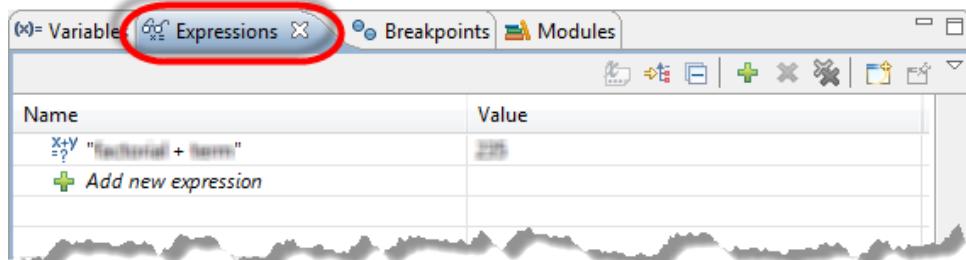


Figure 320: Expressions Tab

- 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 321: 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.

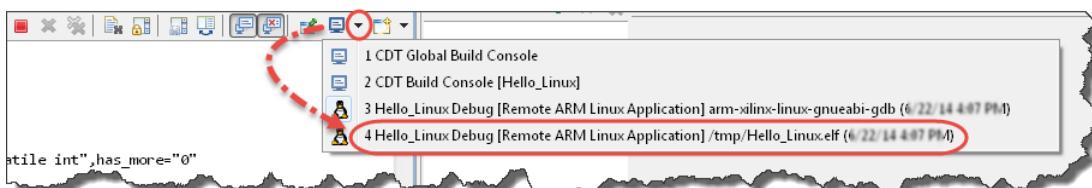


Figure 322: Display Selected Console

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 (GDB)**.

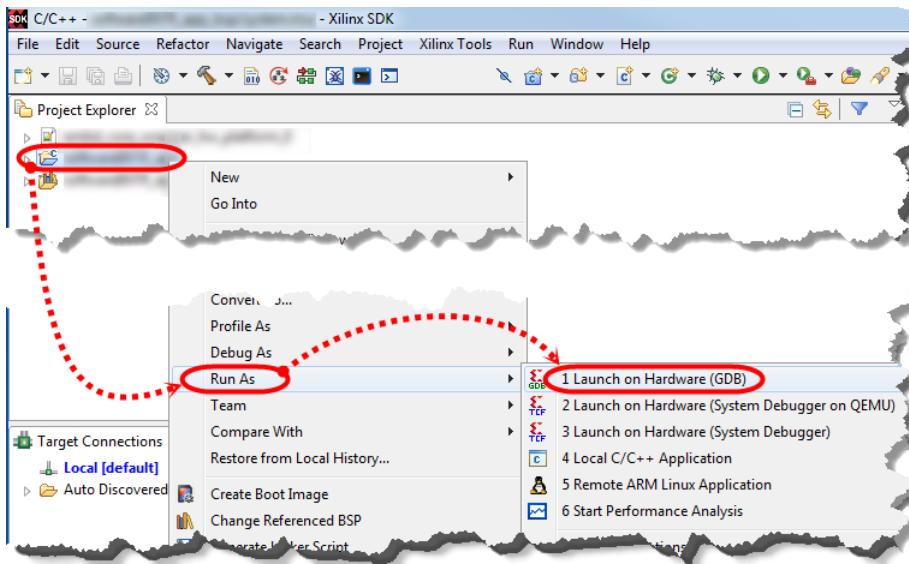


Figure 323: Launching a Run on Hardware

1-1-3. If you are asked if you want to terminate a previous run, click **Yes**.

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.....	237
Creating a Linux Debug Configuration.....	238
Configuring the PC's Ethernet Port for Remote System Explorer.....	243
Opening the Remote System Explorer.....	244
Setting the IP Address of the Development Board	248
Verifying the Host's Static IP Ethernet Port from the Development Board	249
Downloading and Running the Linux Application Using RSE	250
Configuring the Ethernet Port for Use with the Remote System Explorer.....	257

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.

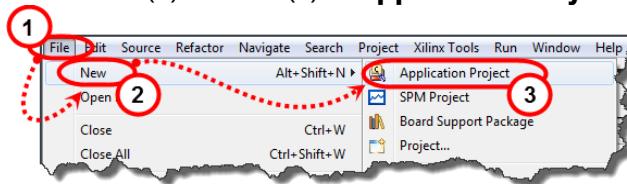


Figure 324: Creating an Application Project

- 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 & 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.

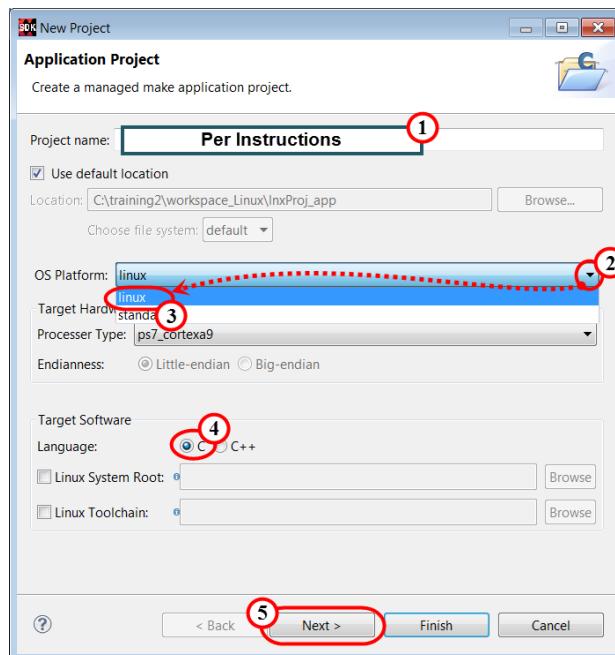


Figure 325: Selecting Linux for the New Application Project

1-1-6. Click **Next** to view the templates available for Linux applications.

1-1-7. Select **an application template** (1).

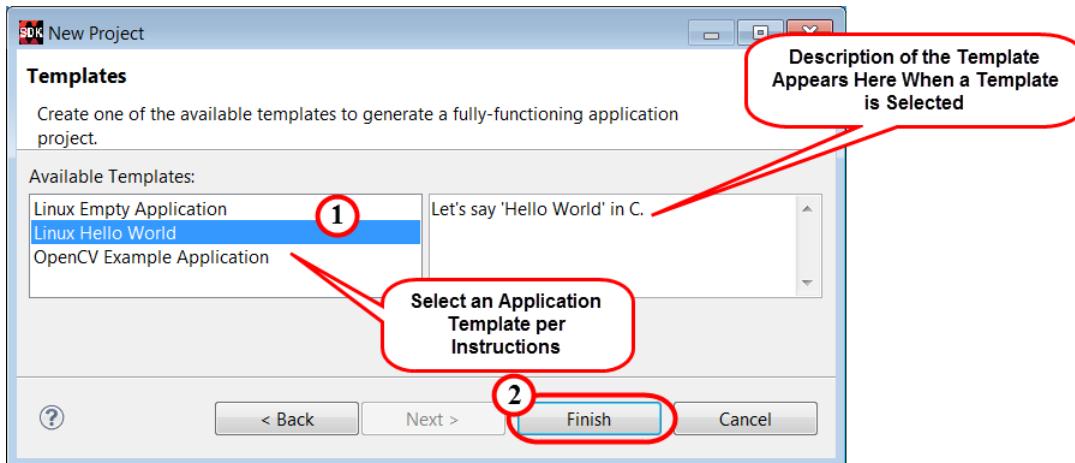


Figure 326: Selecting a Linux Template

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.

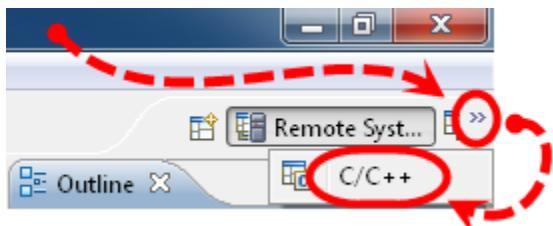


Figure 327: Changing Perspective

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 the **your application project name** project and select **Debug as** (2) > **Debug Configurations** (3).

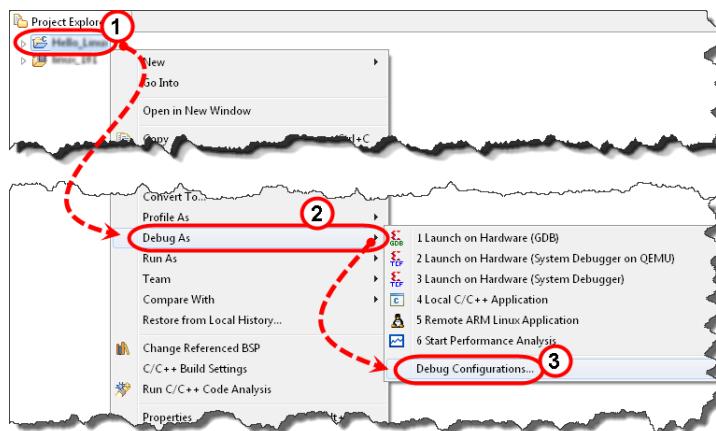


Figure 328: Selecting Debug Configurations

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.

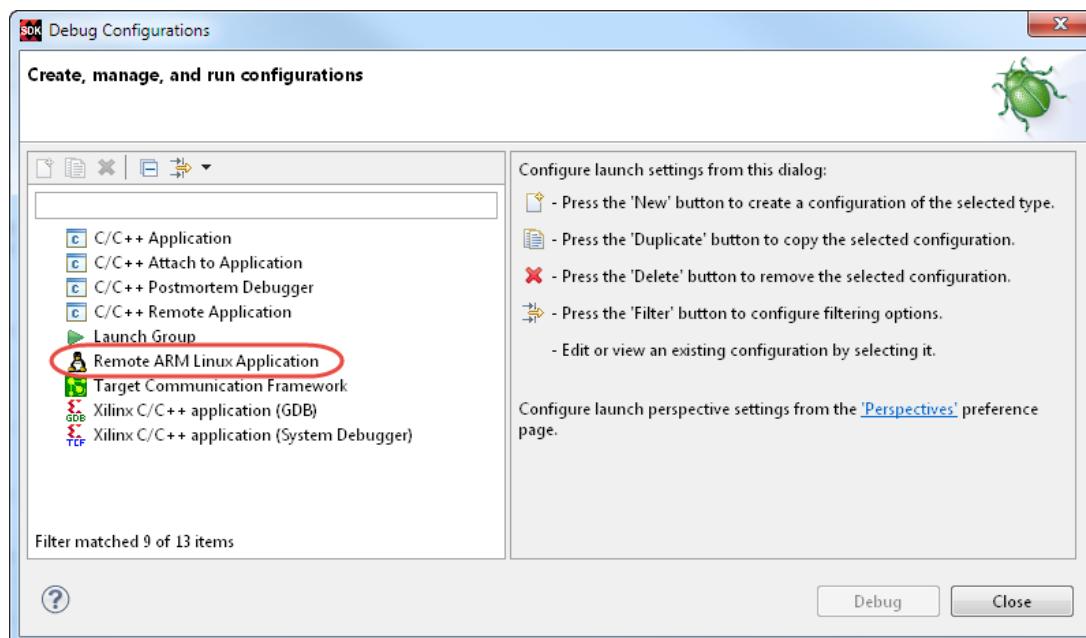


Figure 329: Selecting Remote ARM Linux Application

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.

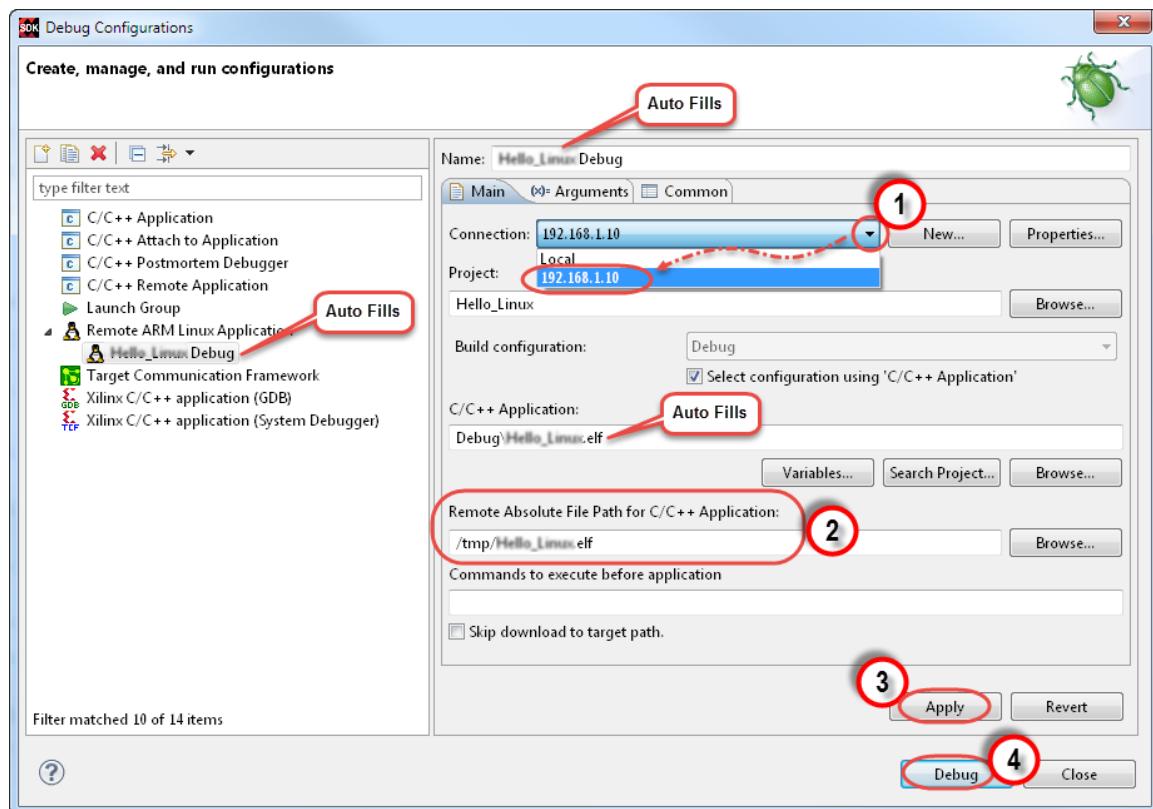


Figure 330: Linux RSE Configuration Dialog Box

- 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.

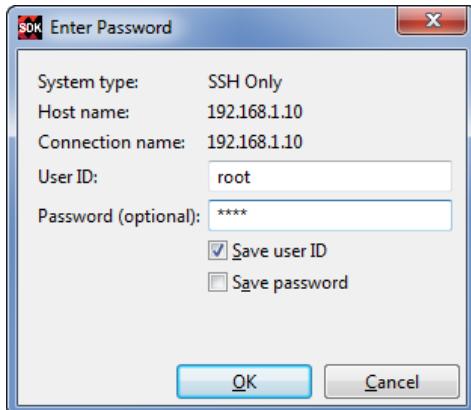


Figure 331: Entering the User ID and Password

1-2-2. Click **OK** to log in.

1-2-3. Click **Yes** to confirm opening the Debug perspective.

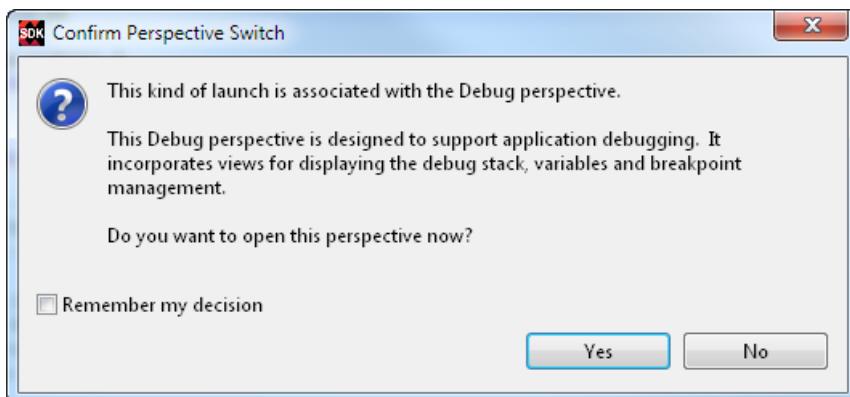


Figure 332: Confirming Perspective Switch

The Debug perspective opens.

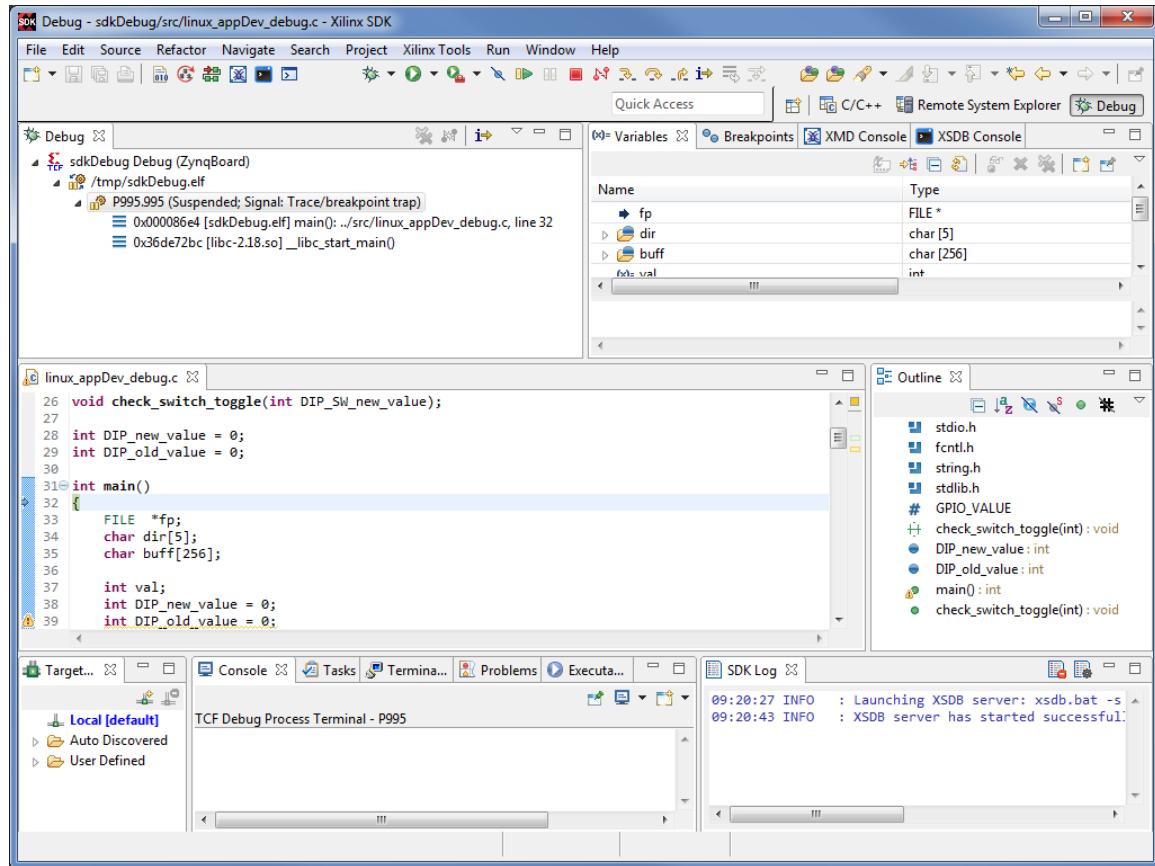


Figure 333: Debug Perspective

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 in addition and similar to the terminal connection that has already been established, but is much higher performance in nature. This connection is used when performing application program debug.

1-1. Verify that the host PC Ethernet port is set to a static IP address of 192.168.1.11.

You may have already performed this step earlier, in which case you can proceed.

If you are unfamiliar with how to complete this task, refer to "Configuring the PC's Ethernet Port for Remote System Explorer" section under SDK Operations > Linux and Remote System Explorer in the *Lab Reference Guide*.

1-2. Use the Remote System Explorer (RSE) tool to download the software application to the Linux platform.

Using this Ethernet connection is much faster than downloading over the serial connection.

1-2-1. Select Window > Open Perspective > Other to open a new SDK perspective.

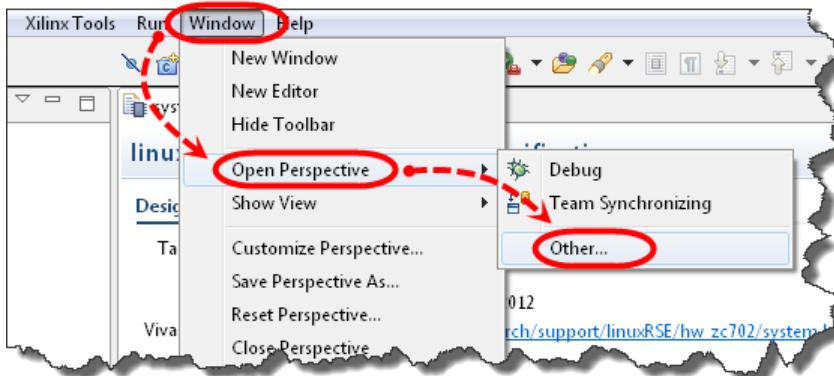


Figure 334: Selecting Open Perspective - Other

1-2-2. 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.**

- 1-2-3.** Click **OK** to open a new Remote System Explorer perspective.

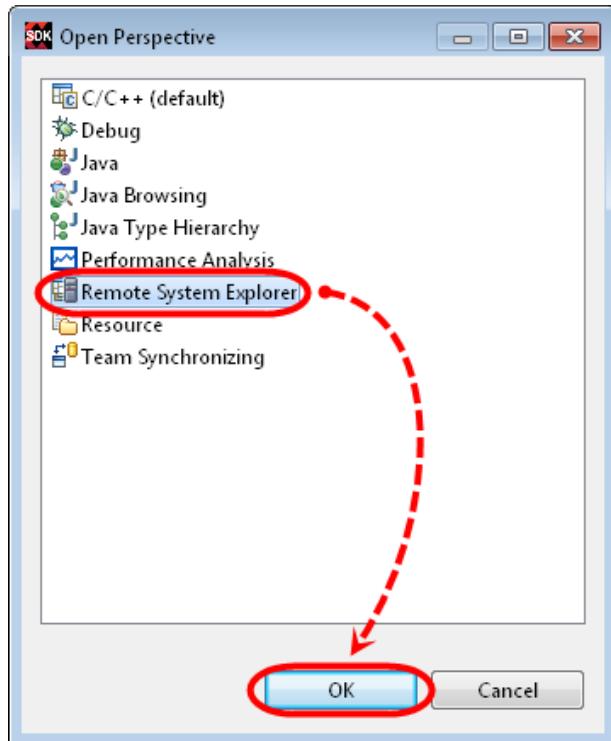


Figure 335: Selecting Remote System Explorer

- 1-3.** Establish a new connection by using the industry-standard SSH protocol over the Ethernet port.

- 1-3-1.** Using the Remote Systems tab (top left), right-click **Local**.

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.

- 1-3-2.** Select **New > Connection**.

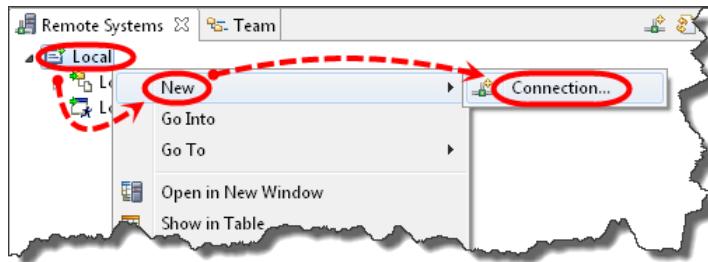


Figure 336: Selecting New Connection

- 1-3-3.** From the Select Remote System Type dialog box, select **SSH Only**.

- 1-3-4.** Click **Next** to advance to the next dialog box.

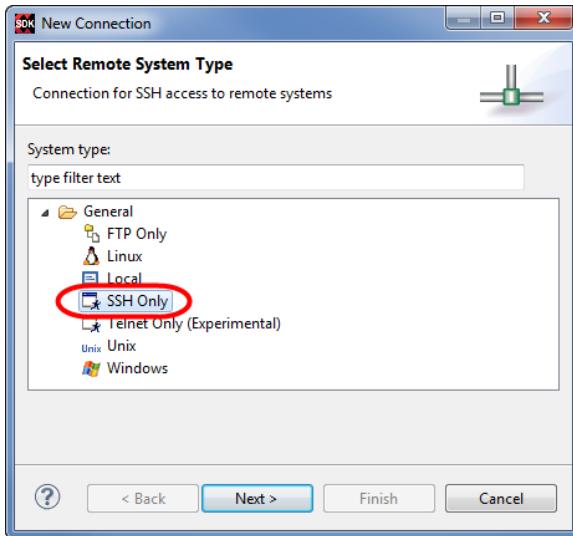


Figure 337: Creating a New SSH RSE Connection

- 1-4.** Select the IP address for 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.
- 1-4-1.** Enter **192.168.1.10** in the Host name and Connection name fields.

The Parent profile field can stay at its default, which will be specific to your machine.

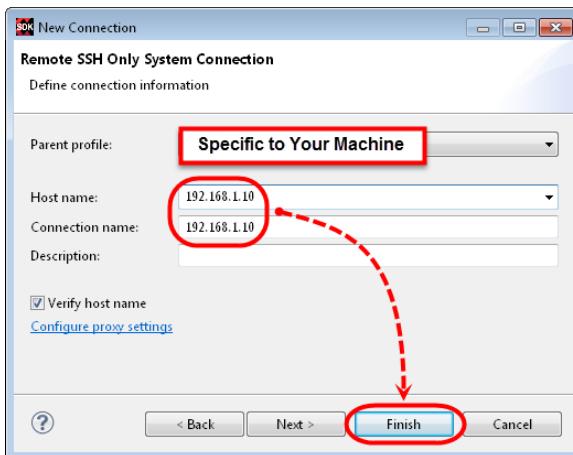


Figure 338: Entering the Host and Connection Names

- 1-4-2.** Click **Finish** to create the new connection.

1-5. The new connection is now present in the Remote Systems tab. You can now access the Linux file system via this connection.

1-5-1. Browse the directory structure of the **192.168.1.10** connection.

This is the Linux file system in the DDR3 RAM of the board. The Local connection is the directory structure of the host machine. You can even drag-and-drop files between the two connections or other Windows directory panes.

1-5-2. Expand the **Sftp Files** folder and then expand the **My Home** (this will be empty) and **Root** directories.

The first time you attempt to logically connect to the Linux host, you will be asked for SSH credentials, User ID and Password. They will be the same as used to log into Linux after it booted up: `root`.

If a dialog box does not automatically appear asking for credentials, right-click **192.168.1.10** and select **Connect**. Make sure you that disable your firewall before proceeding.

You will be prompted for a user ID and password (see the next step).

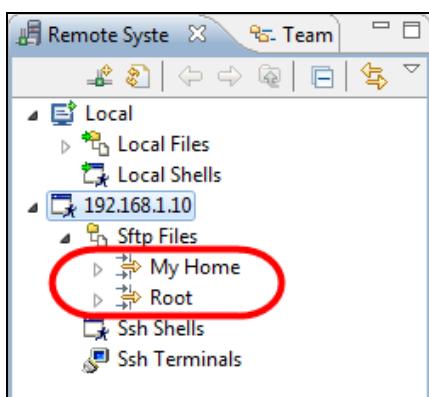


Figure 339: Remote Systems Tab

1-5-3. Enter **root** in the User ID and Password fields.

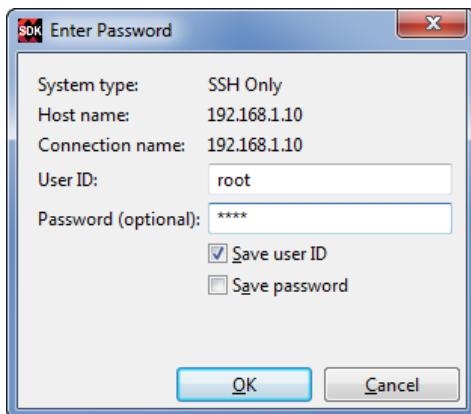


Figure 340: Entering the User ID and Password

- 1-5-4.** Click **OK** to establish the secure SSH connection.

A changed connection security warning message will appear.

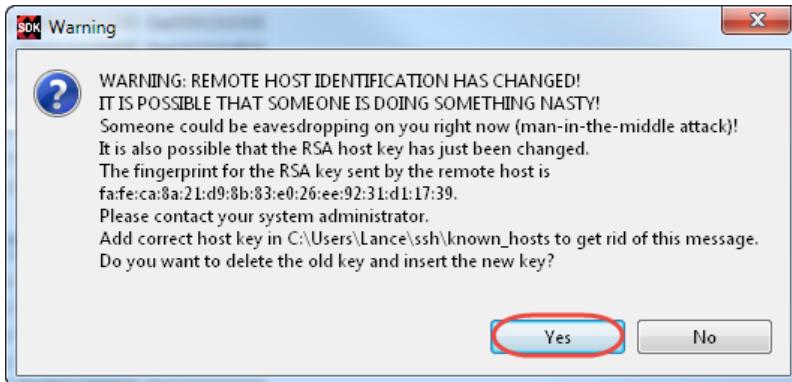


Figure 341: Changed Connection Security Warning

- 1-5-5.** Click **Yes** to continue.

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 IP address of the board:

ifconfig eth0 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. Using the Linux terminal console, ping the host 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 in addition and similar to the terminal connection that has already been established, but is much higher performance in nature. This connection is used when performing application program debug.

1-1. Verify that the host PC Ethernet port is set to a static IP address of 192.168.1.11.

You may have already performed this step earlier, in which case you can proceed.

If you are unfamiliar with how to complete this task, refer to "Configuring the PC's Ethernet Port for Remote System Explorer" section under SDK Operations > Linux and Remote System Explorer in the *Lab Reference Guide*.

1-2. Use the Remote System Explorer (RSE) tool to download the software application to the Linux platform.

Using this Ethernet connection is much faster than downloading over the serial connection.

1-2-1. Select Window > Open Perspective > Other to open a new SDK perspective.

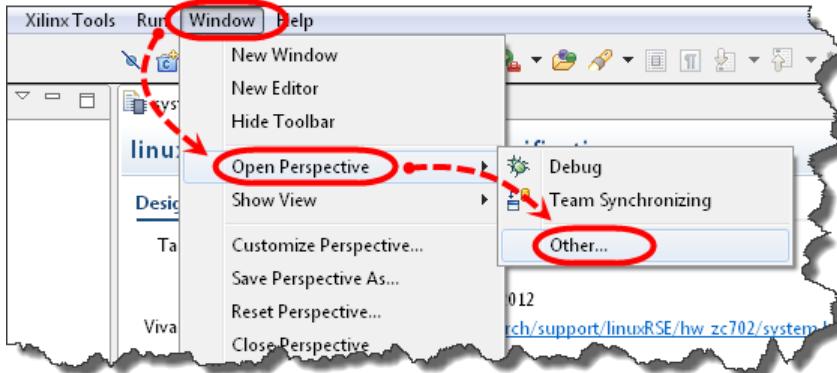


Figure 342: Selecting Open Perspective - Other

1-2-2. 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.**

- 1-2-3.** Click **OK** to open a new Remote System Explorer perspective.

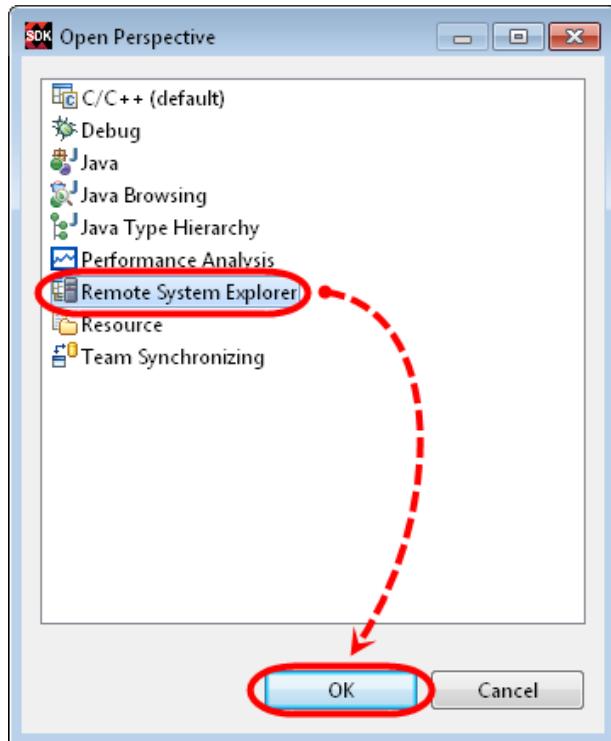


Figure 343: Selecting Remote System Explorer

- 1-3.** Establish a new connection by using the industry-standard SSH protocol over the Ethernet port.

- 1-3-1.** Using the Remote Systems tab (top left), right-click **Local**.

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.

- 1-3-2.** Select **New > Connection**.

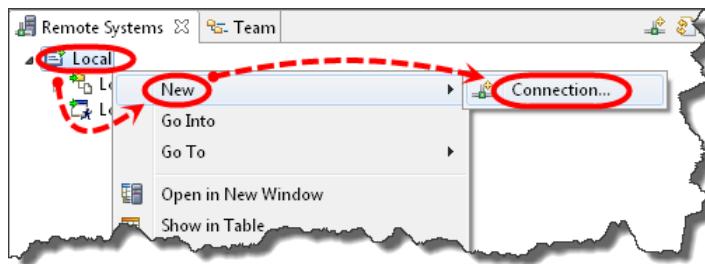


Figure 344: Selecting New Connection

- 1-3-3.** From the Select Remote System Type dialog box, select **SSH Only**.

- 1-3-4.** Click **Next** to advance to the next dialog box.

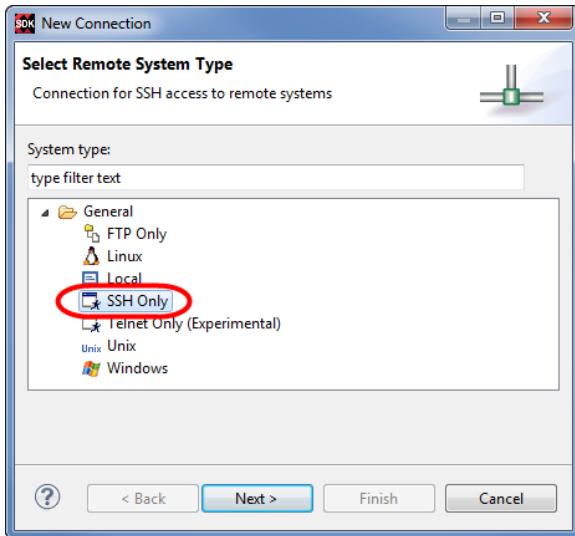


Figure 345: Creating a New SSH RSE Connection

- 1-4.** Select the IP address for 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.
- 1-4-1.** Enter **192.168.1.10** in the Host name and Connection name fields.

The Parent profile field can stay at its default, which will be specific to your machine.

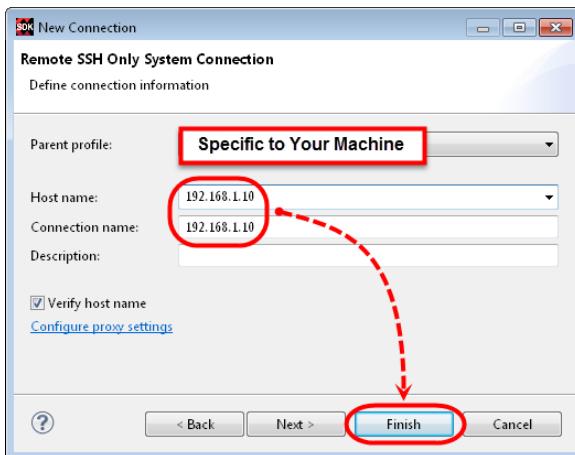


Figure 346: Entering the Host and Connection Names

- 1-4-2.** Click **Finish** to create the new connection.

1-5. The new connection is now present in the Remote Systems tab. You can now access the Linux file system via this connection.

1-5-1. Browse the directory structure of the **192.168.1.10** connection.

This is the Linux file system in the DDR3 RAM of the board. The Local connection is the directory structure of the host machine. You can even drag-and-drop files between the two connections or other Windows directory panes.

1-5-2. Expand the **Sftp Files** folder and then expand the **My Home** (this will be empty) and **Root** directories.

The first time you attempt to logically connect to the Linux host, you will be asked for SSH credentials, User ID and Password. They will be the same as used to log into Linux after it booted up: `root`.

If a dialog box does not automatically appear asking for credentials, right-click **192.168.1.10** and select **Connect**. Make sure you that disable your firewall before proceeding.

You will be prompted for a user ID and password (see the next step).

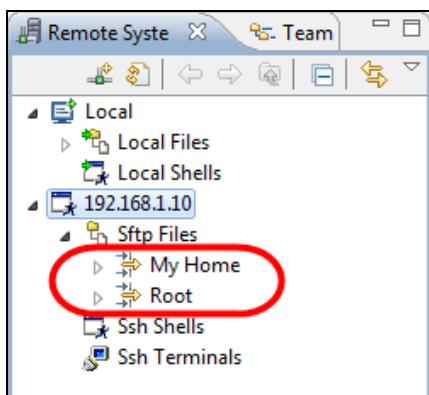


Figure 347: Remote Systems Tab

1-5-3. Enter **root** in the User ID and Password fields.

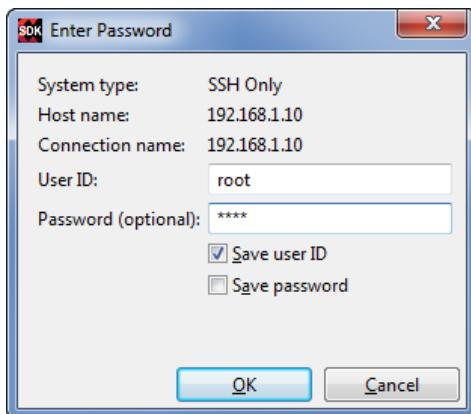


Figure 348: Entering the User ID and Password

- 1-5-4.** Click **OK** to establish the secure SSH connection.

A changed connection security warning message will appear.

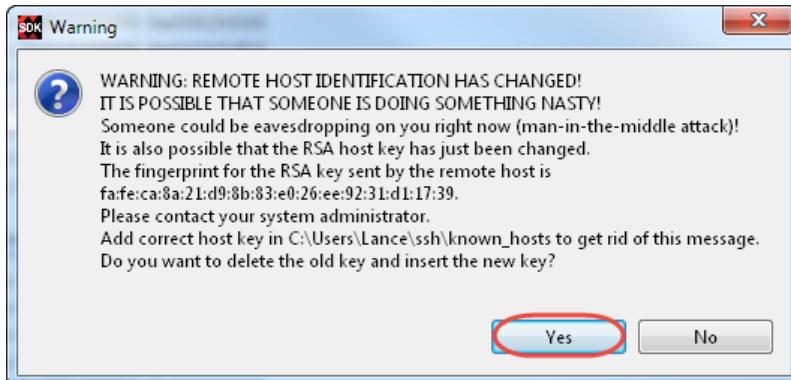


Figure 349: Changed Connection Security Warning

- 1-5-5.** Click **Yes** to continue.

1-6. 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.

- 1-6-1.** Click the **C/C++** perspective (top right) to return to the original perspective.

- 1-6-2.** In the Project Explorer tab, right-click **your project name**.

- 1-6-3.** Select **Run as > Run Configurations**.

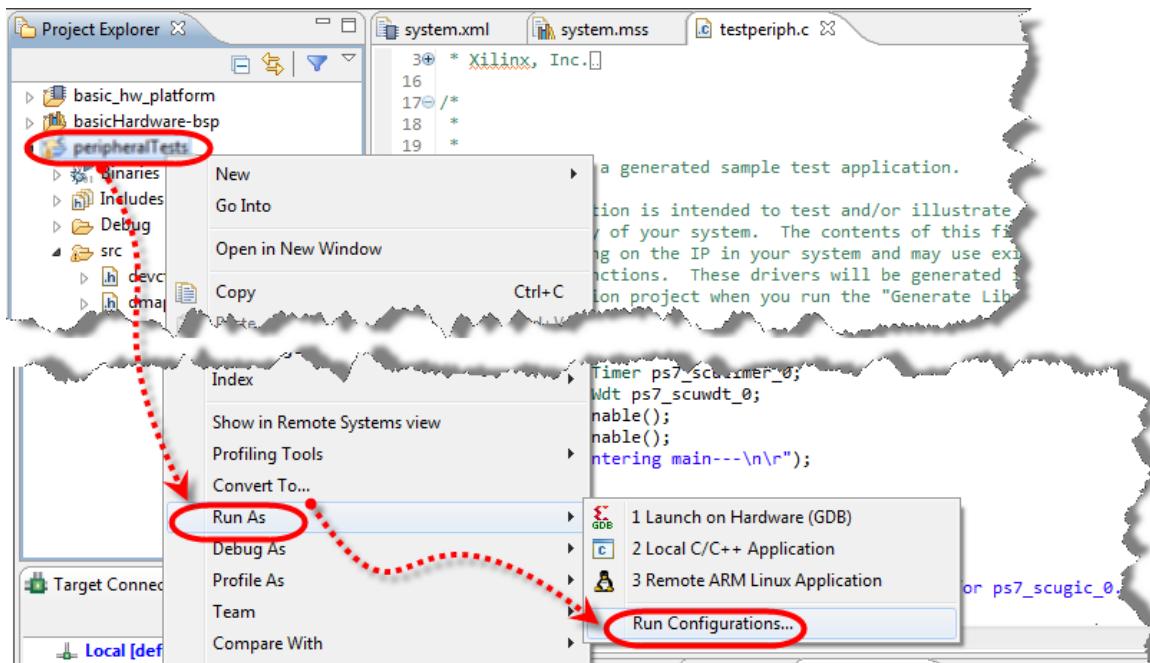


Figure 350: Selecting Run Configurations

1-7. 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.

- 1-7-1.** Double-click **Remote ARM Linux Application**.
- 1-7-2.** Select **192.168.1.10** from the Connection drop-down list.

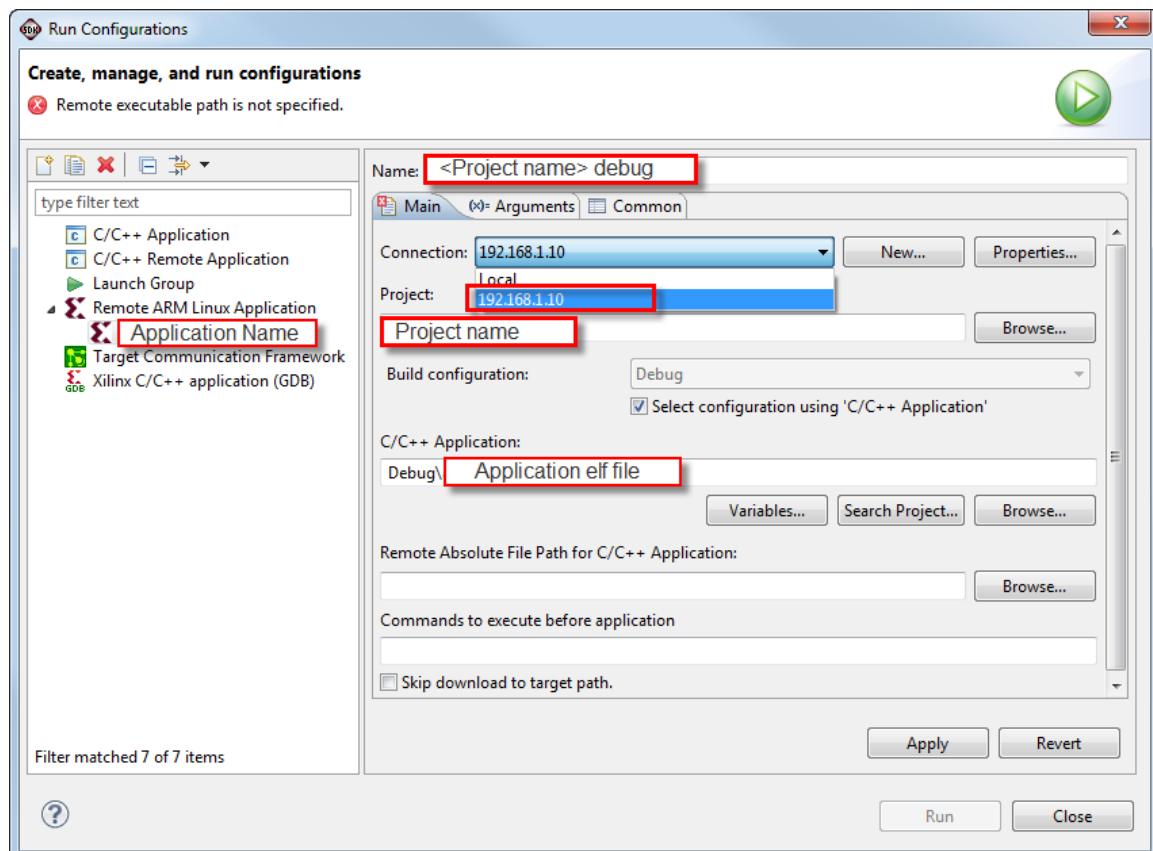


Figure 351: Selecting the IP Address

- 1-7-3.** In the Remote Absolute File Path for C/C++ Application field, enter **/tmp/your application.elf**.

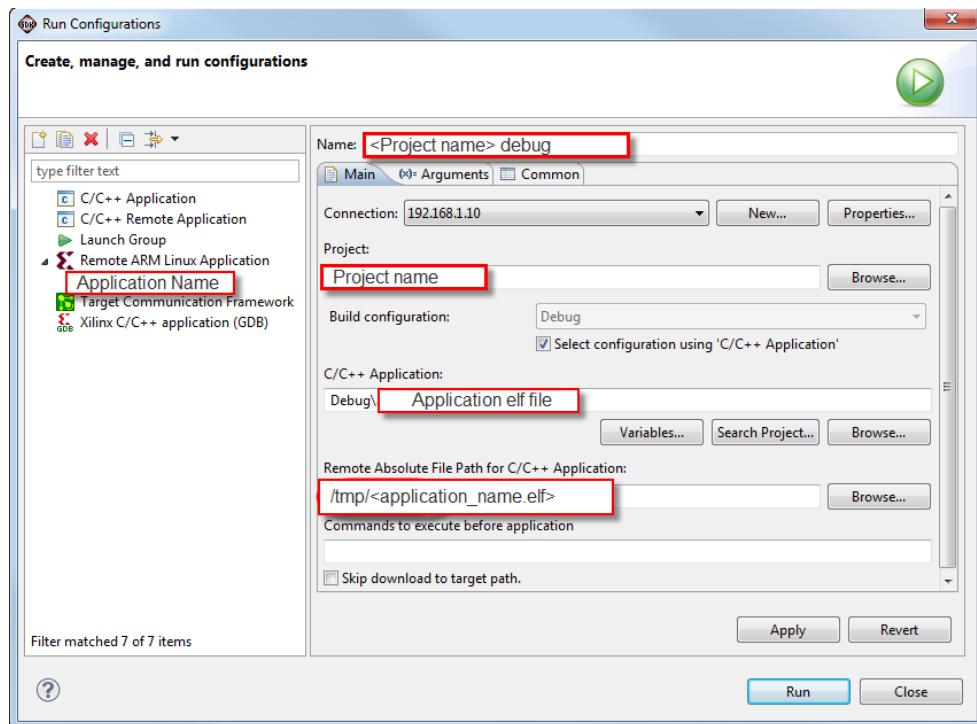


Figure 352: Setting the Remote File Path for the Application

1-8. Run the program.

- 1-8-1.** Click **Apply**.
1-8-2. Click **Run**.
1-8-3. Enter **root** in the User ID and Password fields.

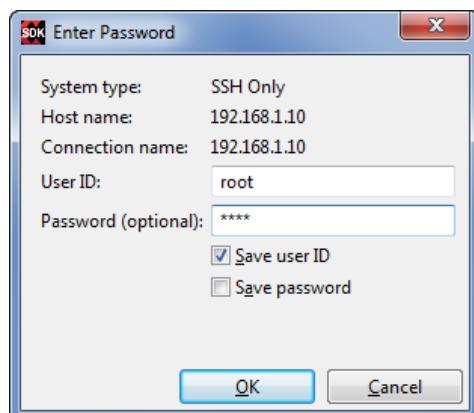


Figure 353: Entering the User ID and Password

- 1-8-4.** Click **OK**.
1-8-5. 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 your board 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.

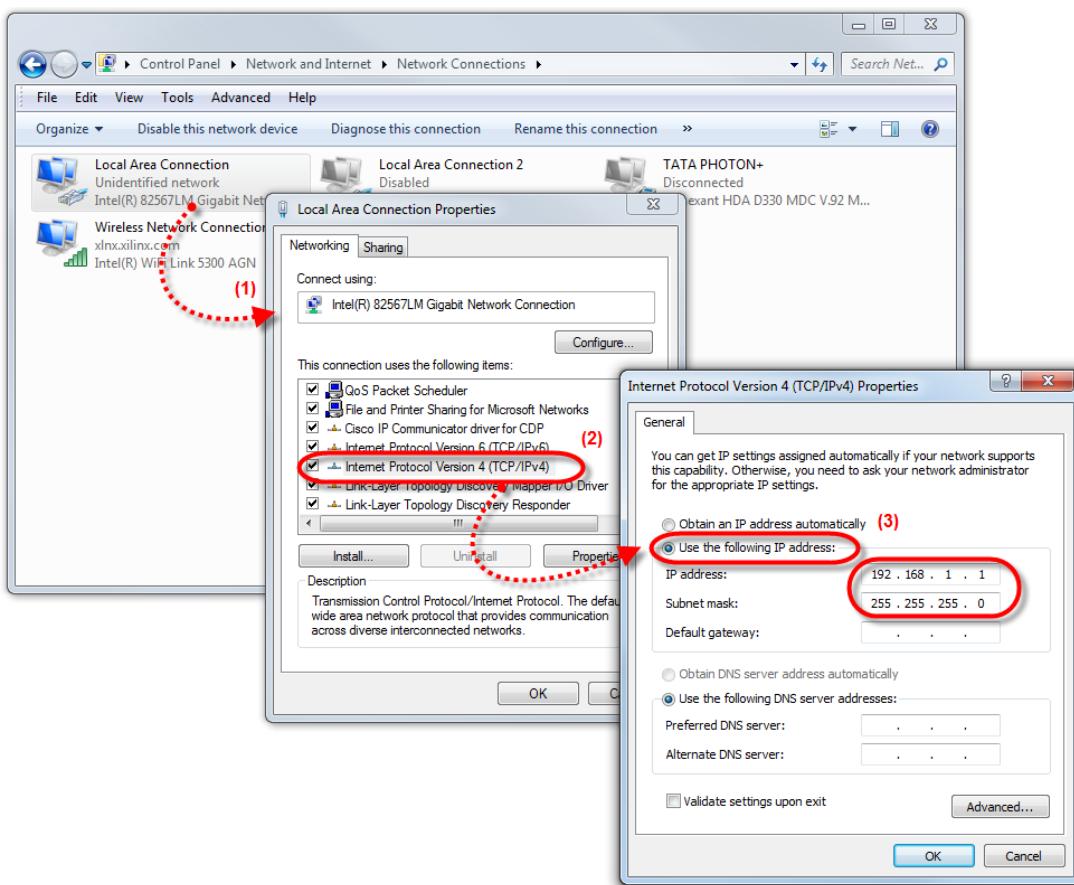


Figure 354: Configuring the Ethernet Port (RSE)

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).

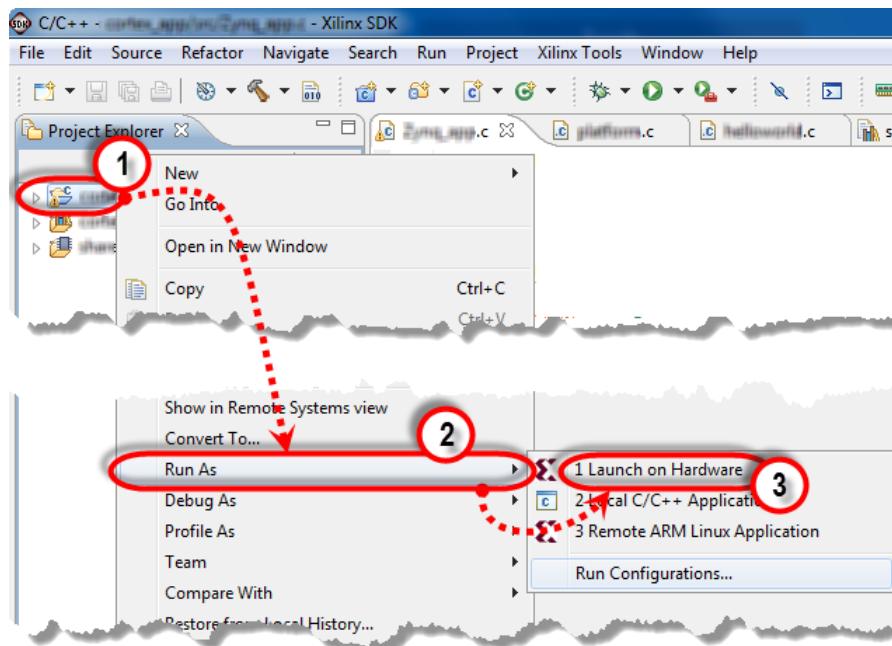


Figure 355: Selecting Launch on Hardware

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.

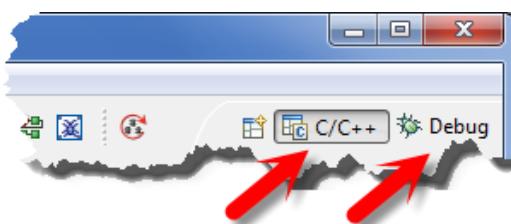


Figure 356: Clicking the Desired Icon

- 1-1-3.** If the perspective is not readily available, click the **Open Perspective** icon.
- 1-1-4.** Select the desired perspective from the list.

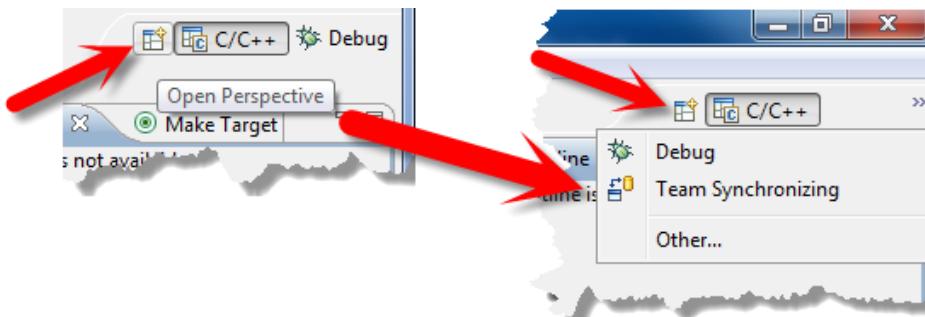


Figure 357: Selecting the Desired Perspective

Configuring the SDK 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).

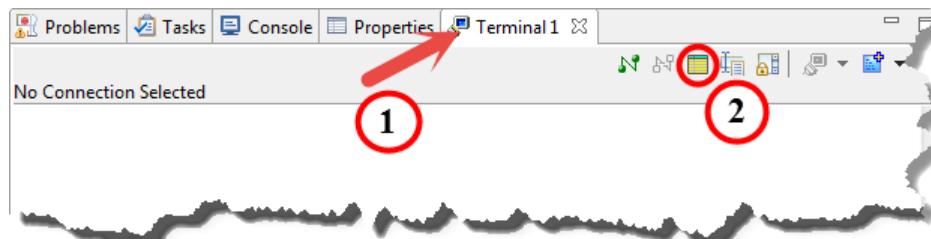


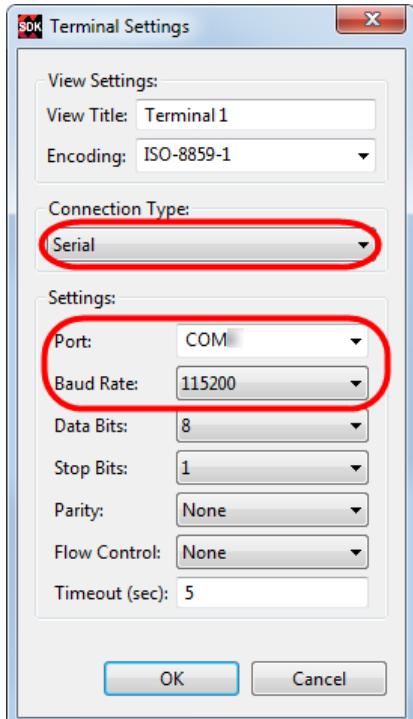
Figure 358: Accessing the Terminal Settings

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 359: 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 always available in the same panel that you would find the console.**
- 1-1-1.** If it is not available, select **Window > Show View > Other > Xilinx > SDK Terminal** to open the SDK Terminal tab.

- 1-2. Configure the SDK Terminal.**

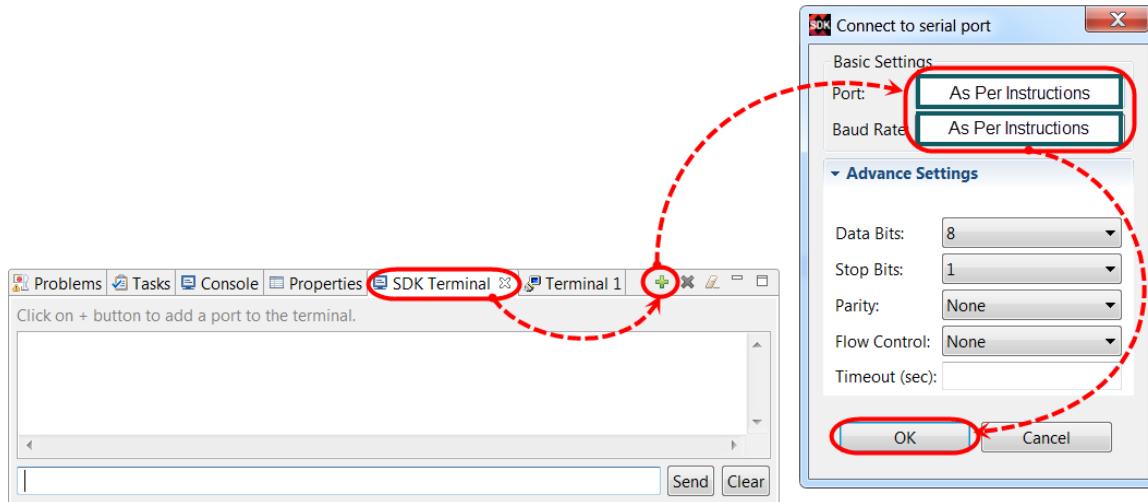


Figure 360: Configuring the SDK Terminal

- 1-2-1.** Click the green '+' sign to open the Connect to Serial Port dialog box.
- 1-2-2.** Select the serial port that is connected to the device you want to communicate with.
- 1-2-3.** Set the baud rate to **115200**.
- 1-2-4.** Leave the other settings at their defaults.
- 1-2-5.** Click **OK** to save these settings and begin the terminal session.

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.

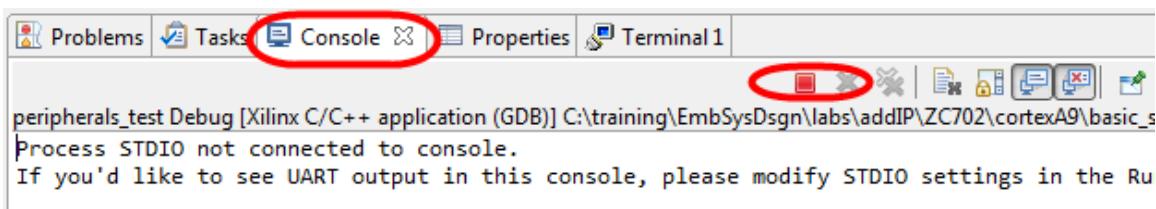


Figure 361: Terminating the Application

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. If this dialog box appears, click **Yes** to conclude the saving of the SDK configuration and exiting of the tool.

Adding Symbols to a BSP

Adding a symbol to an application project is equivalent to adding a "#DEFINE SYM_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 BSP settings

- 1-1-1. Open the project build settings:
 - o Right-click the application project folder (**your application project name**) in the Project Explorer pane.
 - o Select **C/C++ Build Settings** from the context menu.
- 1-1-2. From the Settings view of the Properties dialog box, select **SDSCC Compiler > Symbols**.
- 1-1-3. In the Defined symbols area, click the **Add** icon to open a dialog box where you can enter a new symbol name.
- 1-1-4. Enter a symbol name as supplied below and click OK to add the new symbol. Repeat as necessary if multiple symbols are supplied (each symbol is a single word, and multiple symbols will be separated by commas).
 - o **<application_project_symbols>**
- 1-1-5. Once all symbols are added click OK to apply changes and close the Properties dialog box.
SDK will automatically re-build 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:
 - o Right-click the BSP project folder (e.g., **your BSP name**) in the Project Explorer pane.
 - o 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.
 - o **-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.

QEMU Operations

In This Section

Configuring and Starting QEMU from SDK.....	266
Opening the VirtualBox and Running an OS.....	268
Running an Application on QEMU from SDK.....	269
Selecting the QEMU Console Log.....	271
Setting Up a (QEMU) System Debugger Debug Configuration.....	272
Stopping QEMU.....	274
Starting the QEMU.....	274
Shutting Down a Running QEMU Virtual Machine Using the Command Prompt	274

Configuring and Starting QEMU from SDK

1-1. Configure and start QEMU.

1-1-1. Select Xilinx Tools > Configure QEMU Settings.

The Configure QEMU Settings window opens.

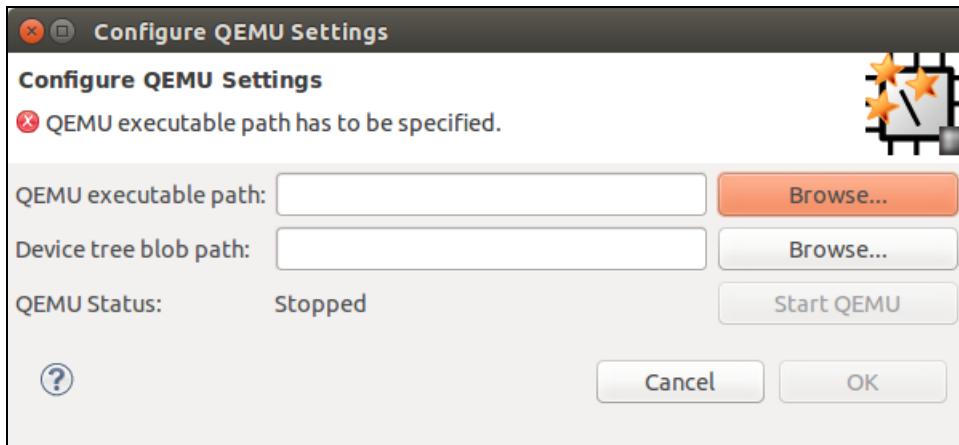


Figure 362: Configuring QEMU Settings

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. From the left panel titled *Places*, click the **File System** icon to navigate to the root of the directory structure (1).

1-1-4. Double-click the **opt** folder icon (2).

1-1-5. Navigate to the

`pkg/petalinux-v2016.1-final/tools/linux-i386/petalinux/bin` directory.

1-1-6. Select the QEMU executable file **qemu-system-aarch64**.

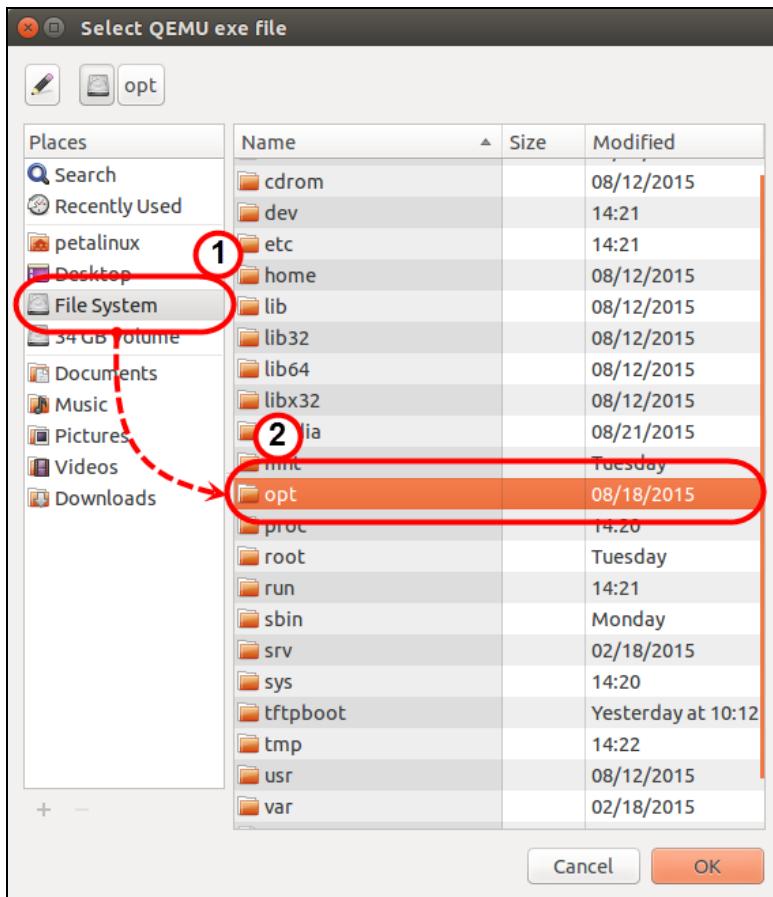


Figure 363: Selecting the QEMU EXE File

- 1-1-7.** Click **OK** to close the file selection dialog box.
- 1-1-8.** Click **Browse** next to the *Device tree blob path* field to open a file selection dialog box.
- 1-1-9.** Double-click the **File System** entry under the *Places* quick navigation area.
- 1-1-10.** Navigate to the `/home/xilinx/training/*****/support/lab_name` directory.
- 1-1-11.** Select the device tree blob **zynqmp-qemu-arm.dtb**.
- 1-1-12.** Click **OK** to close the file selection dialog and return to the Configure QEMU Settings dialog box.

- 1-1-13.** Click **Start QEMU** to launch the QEMU virtual machine.

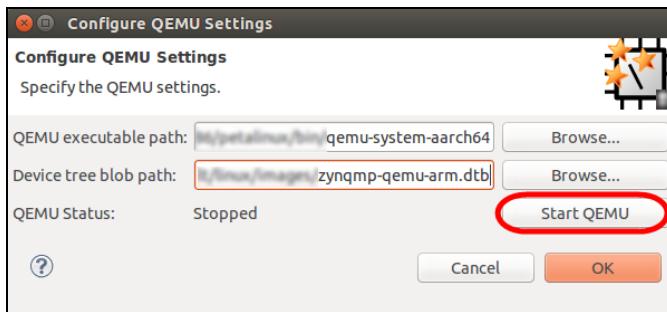


Figure 364: Start Running QEMU

A progress information window will open and show you the progress of launching the QEMU.

Note: The QEMU VM is now running as a background process and the QEMU status should now be set to Running.

- 1-1-14.** 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.

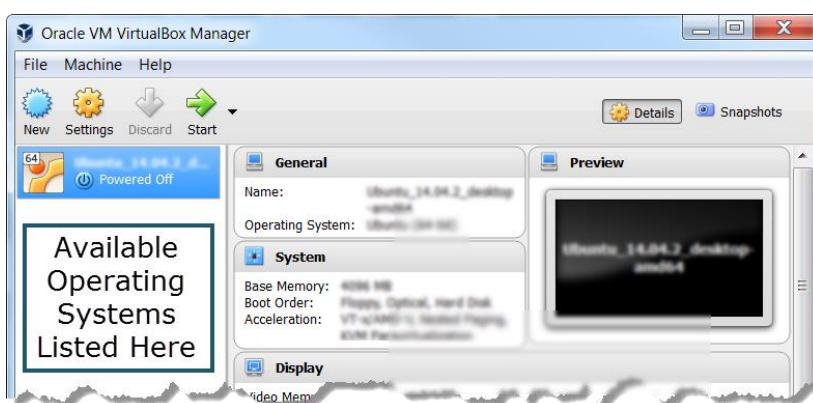


Figure 365: Opening VirtualBox

- 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.

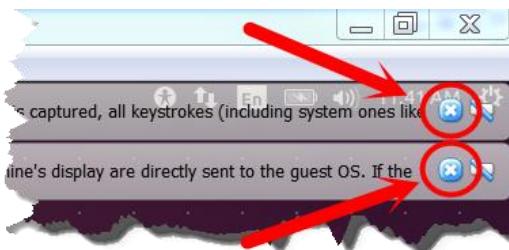


Figure 366: Closing Information Messages

Running an Application on QEMU from SDK

Configurations are a very powerful ally in setting up a run for a very specific set of criteria. Here you will 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. Double-click **Xilinx C/C++ application (System Debugger on QEMU)** to create a launch configuration.
1-1-4. Verify the Target Setup tab (no changes are required).

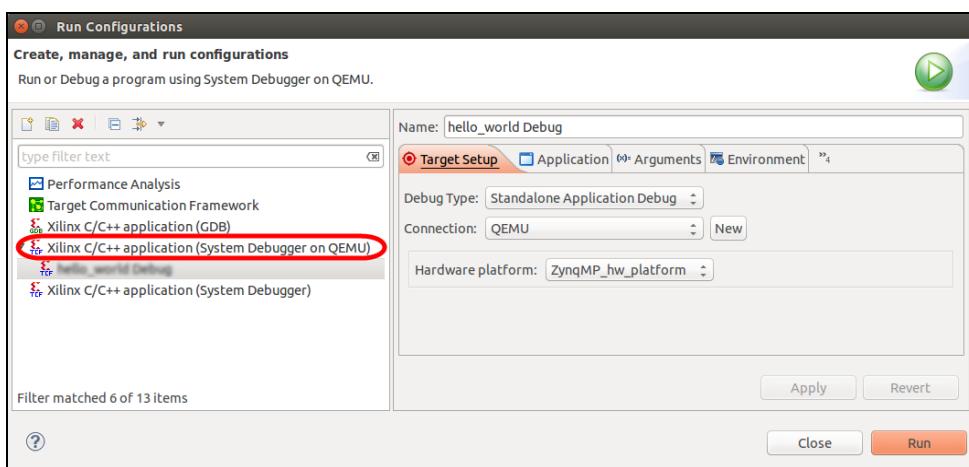


Figure 367: Verifying the Target Setup Tab

- 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).
- 1-1-7. Verify that the project name and application ELF file are correct (no changes should be required).

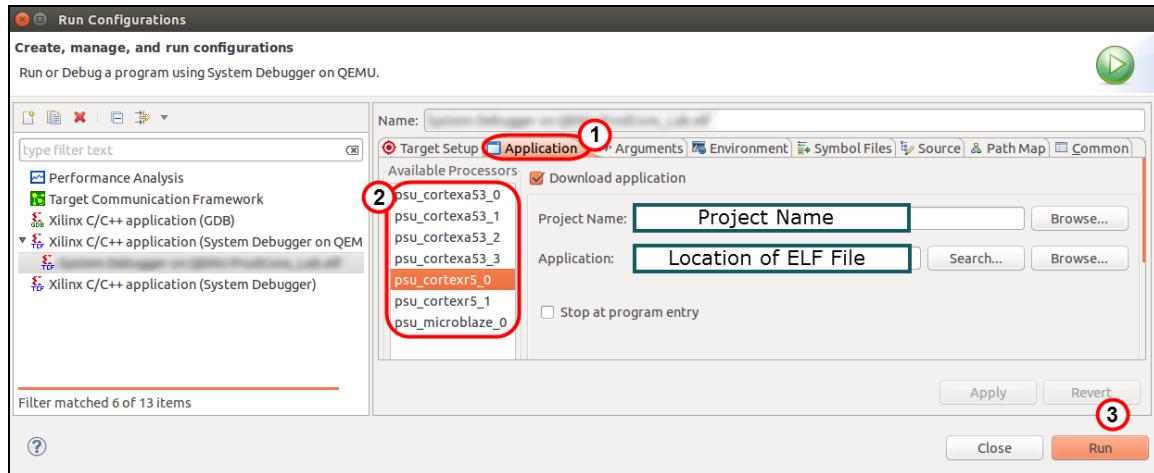


Figure 368: Verifying the Application Tab

Note: It is possible to retarget the processor the application will execute on by selecting a different processor and configuring it accordingly.

- 1-1-8. Click **Run** (3).

Selecting the QEMU Console Log

1-1. Select the QEMU console log.

- 1-1-1.** Select the **Console** tab to open the Console view (1).

If the tab is not available, it can be opened by selecting **Window > Show View > Console**.

- 1-1-2.** Click the down arrow next to the Console icon to open the drop-down list of available consoles (2).
- 1-1-3.** Select the **QEMU Console log** listing to direct the console input and output to QEMU (3).

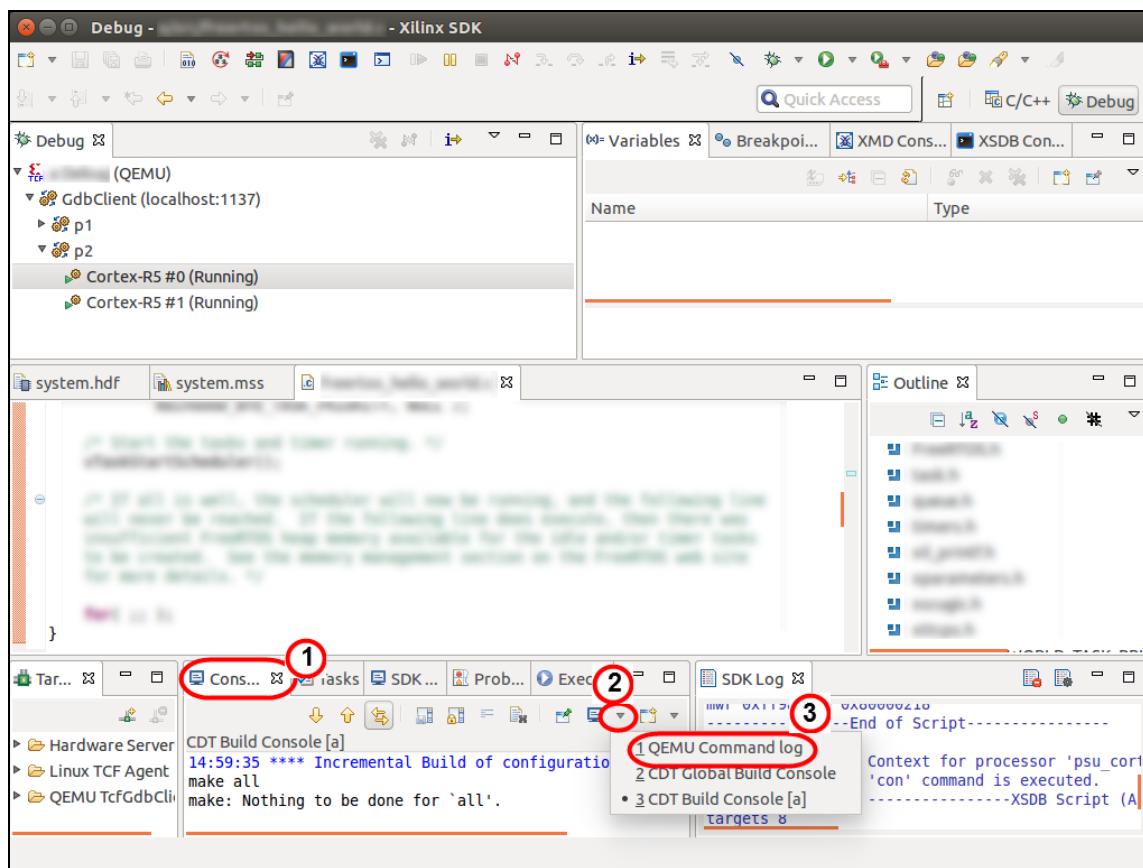


Figure 369: Selecting the QEMU Command Log

The QEMU command log is now active in the Console view.

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. From the Project Explorer pane, right-click the **your project name** project 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).

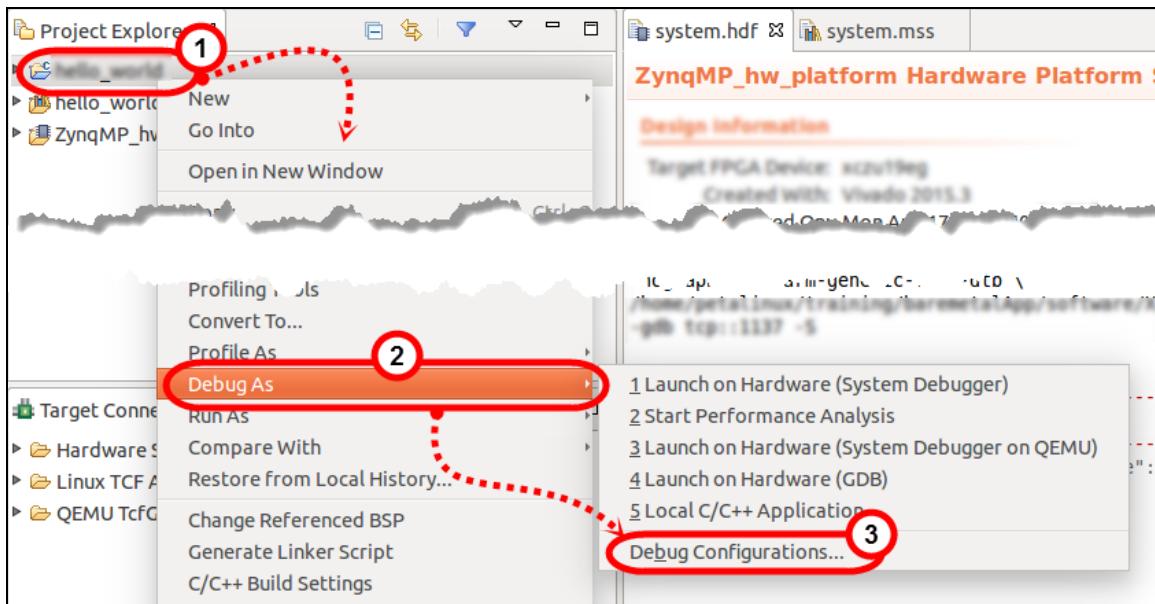


Figure 370: Creating a Debug Configuration (QEMU)

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.

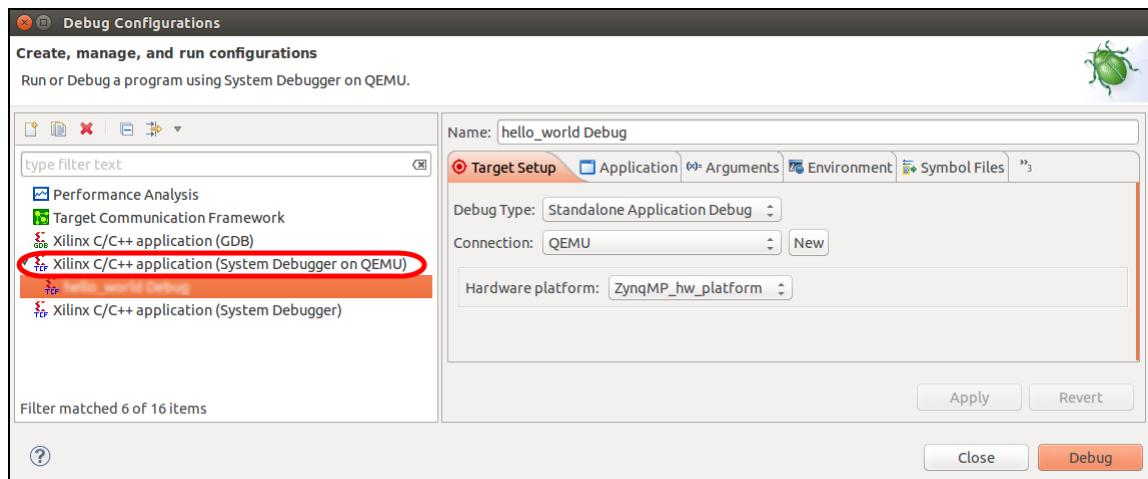


Figure 371: Creating a New Debug Configuration (QEMU)

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.

If the Confirm Perspective Switch dialog box appears, click **Yes**.

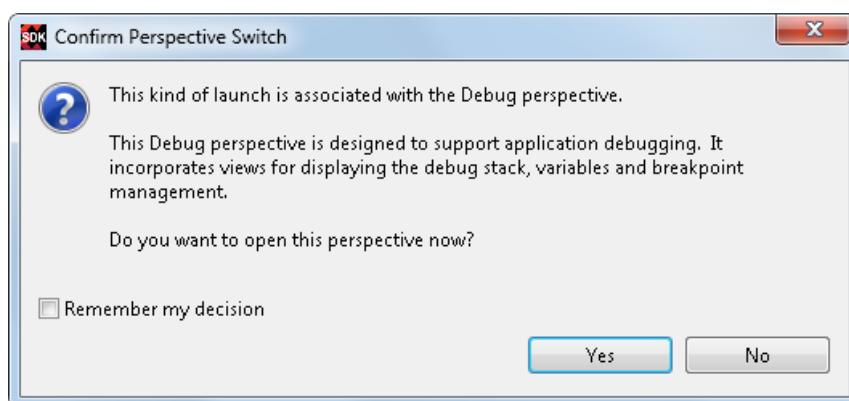


Figure 372: Confirming Switch of Perspective

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):

poweroff

Several system messages will be displayed on the console as the Linux instance is shutting down.

1-1-2. Enter the key combination **<Ctrl + A>** followed by **X**.

This will terminate the QEMU session.

SDSoC Tool Operations

In This Section

Launching the SDSoC Tool.....	275
Highlighting a Process for Debug	277
Configuring the SDSoC Tool for Linux.....	278
Configuring the SDSoC Tool for Standalone.....	282
Launching a Debug Session.....	283
Setting Up a System Debugger Debug Configuration (Standalone)	285
Setting Up a System Debugger Debug Configuration (Linux).....	289
Creating a C/C++ SDSoC Tool Project.....	297
Configuring the Terminal.....	298
Generating the Software Estimate from the SDSoC Tool	300
Opening a Previously Run Performance Estimate	302
Adding a Watchpoint to a Global Variable	303
Selecting a Build Configuration	305
Marking a Function for Hardware (Dashboard Method)	306
Viewing the Generated System Hardware.....	307
Viewing the Generated Accelerator	307
Cleaning and Building a Project	308

Launching the SDSoC Tool

1-1. Launch the SDSoC design environment and select a workspace.

- 1-1-1.** Select **Start > All Programs > Xilinx Design Tools > SDSoC 2016.1 > SDSoC 2016.1** to launch the SDSoC design environment.

Alternatively, you can launch the SDSoC tool from the desktop shortcut (), if available.

The Workspace Launcher opens after a moment.

The SDSoC 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.** When the Workspace Launcher opens, you can either enter **You can either leave this field at its default setting, which will place the workspace under the Vivado Design Suite project hierarchy, or you can select your own location.** into the workspace field, or you can use the Browse button.

Note that when you use the Browse button, you will need to select the **You can either leave this field at its default setting, which will place the workspace under the Vivado Design Suite project hierarchy, or you can select your own location.** directory and click **OK**.

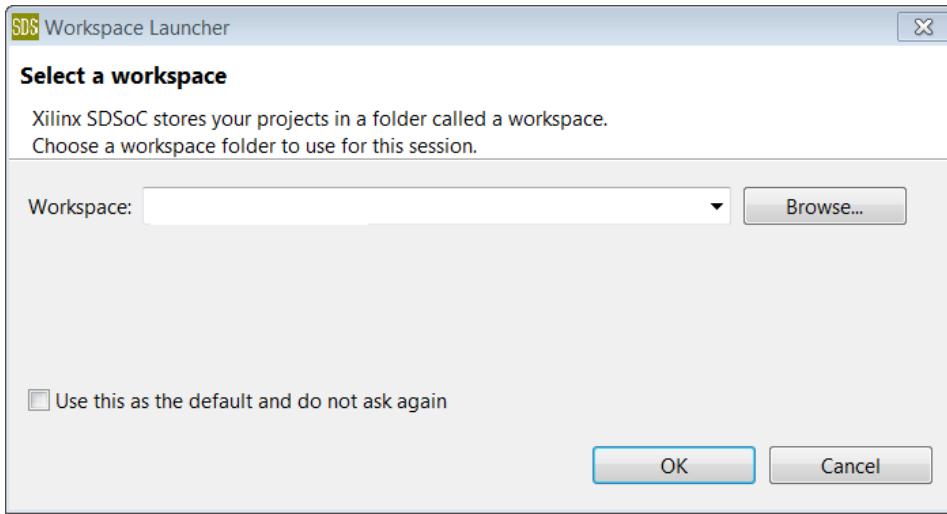


Figure 373: Selecting a Workspace

- 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 SDSoc window—there will be a lot to see.

Highlighting a Process for Debug

1-1. Optional for Linux: 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. Using the Debug view, if it is not already expanded, click the down arrow to expand the top-level debug session (1).
- 1-1-2. Click the down arrow (if it is not already expanded) to expand the next level showing the binary that is being executed (2).
- 1-1-3. Click the process that is currently suspended (3).

Note: The location of the process on the remote device (**/mnt/SDSoC_lab_design.elf**) is also visible.

Note: This will tell the Debug tools which remote application/process to debug and enable the flow control tools.

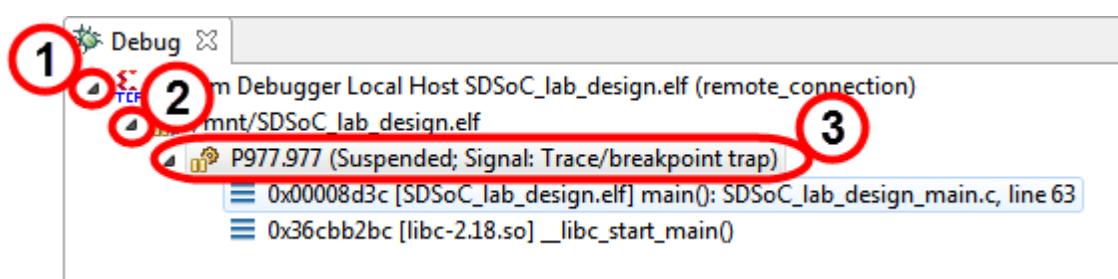


Figure 374: Highlighting the Debug Session Application

Tip: In the figure above and below, P977 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.

```
sh-4.3# ps
  PID USER      VSZ STAT COMMAND
    1 root      1860 S      init [5]
    2 root          0 SW    [kthreadd]
    3          0 R      [kthreadd]

  942   c    77L..  /usr/sbin/tci-agent -d -L- -10
  946 root    3052 S    /bin/sh
  977 root    3296 t    /mnt/SDSoC_lab_design.elf
  981 root          0 SWK  [kworker/1:1H]
  996 root    2956 R      ps
sh-4.3# ~
```

The terminal window shows the output of the `ps` command. The process with PID 977 is highlighted with a red oval, matching the suspended process shown in Figure 374.

Figure 375: Linux Terminal – ps Command

Configuring the SDSoc Tool for Linux

Several tasks must be performed in order to successfully run Linux. These tasks may have been performed for you by your training provider in a classroom environment. If you are outside of this environment, you can refer to the following topics in the *Lab Setup Guide*:

- Configure the jumper settings on the board to boot from the SD card.
- Set the host's Ethernet address to a static IP (suggested: the host IP address).
- Identify the proper COM port (must be done with the board powered on).

Note: If you have a preconfigured SD card available as part of the lab, the following *Preparing an SD card* instruction is optional. If you do not have a preconfigured SD card, it may be necessary to build the project for SDEstimate or copy from the default directory as shown below.

Linux must boot from a properly configured SD card. Many tools generate the necessary file images for Linux, FreeRTOS, and standalone that can be simply copied from a specified location on the host machine to the SD card.

1-1. Prepare an SD card.

1-1-1. Insert an SD card into the PC's SD card slot.

1-1-2. Using Windows Explorer, browse to the SD card drive.

Optional: You may want to erase and/or reformat the SD card at this time, which may prevent any unwanted interaction with other files that may be on the card.

1-1-3. Open a second Windows Explorer window to browse to the files that you will copy to the SD card.

1-1-4. Navigate to the image located at *the location of the files you want to copy to the SD card*.

1-1-5. Drag-and-drop all the files from the source directory to the SD card.

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*.

Insert the SD card into its slot on the hardware platform.

1-2. Apply power to (turn on) the your board hardware platform.

1-2-1. Make sure that AC power is connected to the power brick.

- 1-2-2.** Slide the power switch to the on position.

Some LEDs on the board will illuminate when the board is powered.

1-3. Launch and configure your serial port terminal emulator.

The SDSoc tool Terminal tab can be used for the ZC702 board; however, there is an issue with drivers when information is sent to a ZedBoard. Tera Term is a freeware serial port terminal emulator that has been tested successfully with the target boards.

- 1-3-1.** Identify the COM port associated with your board.

If you are unfamiliar with this process, refer to the *Lab Setup Guide*.

- 1-3-2.** Within the serial port terminal emulator, set the serial port connection to **115200** baud, **8** data bits, **No parity**, **1** stop bit.

If you are unfamiliar with how to use Tera Term, refer to the *Lab Setup Guide*.

If you are unfamiliar with how to use the Terminal tab, refer to the *Lab Reference Guide* under either **SDSoC** or **SDK Tool Operations > Configuring the Terminal**.

Now that the board is powered on and Linux has finished its boot process, the Linux environment must be configured by using the serial port emulator terminal.

Note: You may have to wait for a few minutes for Linux to boot.

1-4. Set the IP address of the board.

- 1-4-1.** Enter the following at the Linux command prompt to change the IP address of the board to IP address of the board:

```
ifconfig eth0 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:192.168.1.11   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
```

1-5. 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.

- 1-5-1.** Using the Linux terminal console, ping the host 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*.

1-6. Configure the target connection in the SDSoc development environment.

1-6-1. Return to the SDSoc tool.

1-6-2. 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**.

1-6-3. Select and right-click **Linux Agent [default]**.

1-6-4. Select **Edit** from the context menu.

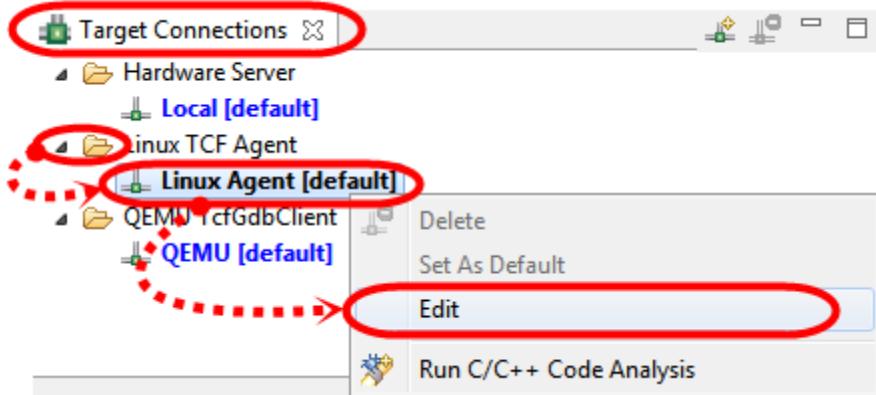


Figure 376: Accessing Linux Agent Target Connection

The Target Connection Details dialog box opens.

1-6-5. Change the host address to **IP address of the board**.

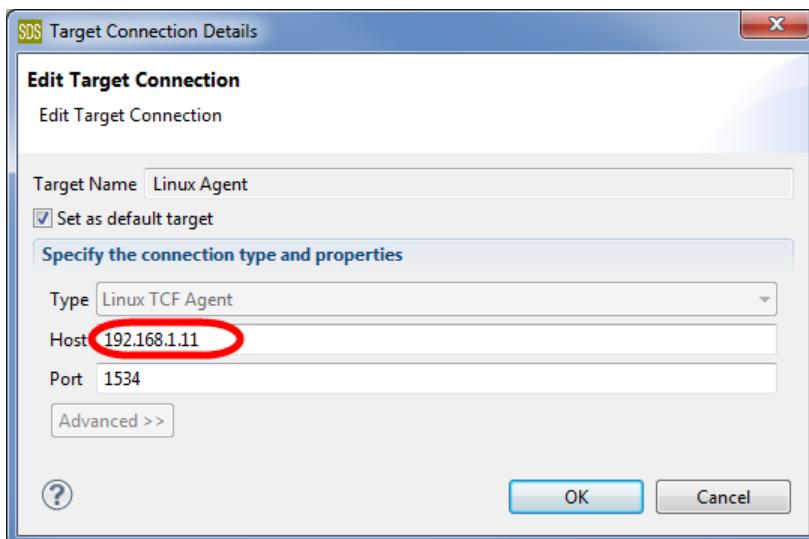


Figure 377: Changing Host Address for Target Connection

1-6-6. Click **OK** to set the connection.

Configuring the SDSoC Tool for Standalone

A couple of tasks must be performed in order to successfully run Standalone. These tasks may have been performed for you by your training provider in a classroom environment. If you are outside of this environment, refer to the following topics in the *Lab Setup Guide*:

- Configure the jumper settings on the board to boot from JTAG.
- Connect and power up the board.
- Identify the proper COM port (must be done with the board powered on).

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. Slide the power switch to the on position.

Some LEDs on the board will illuminate when the board is powered.

1-1-3. Connect the JTAG and UART cable from the host PC to the laptop.

1-2. Launch and configure Tera Term (or equivalent serial port terminal emulator). The SDSoC tool Terminal tab can be used for the ZC702 board; however, there is an issue with drivers when information is sent to a ZedBoard.

1-2-1. Identify the COM port.

If you are unfamiliar with this process, refer to the *Lab Setup Guide*.

1-2-2. Set the serial port connection to **115200 8N1**.

If you are unfamiliar with how to use Tera Term, refer to the *Lab Setup Guide*.

Now that the board is powered on, it is ready to be programmed.

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 up a debug configuration for an SDSoc tool project.

- 1-1-1. Using the Project Explorer pane, ensure that the project (**your project name**) you want to build the Debug configuration for is expanded.
- 1-1-2. Right-click **project.sdsoc** to open the context menu (1).
- 1-1-3. Select **Open** to bring up the project overview page in the main workspace (2).

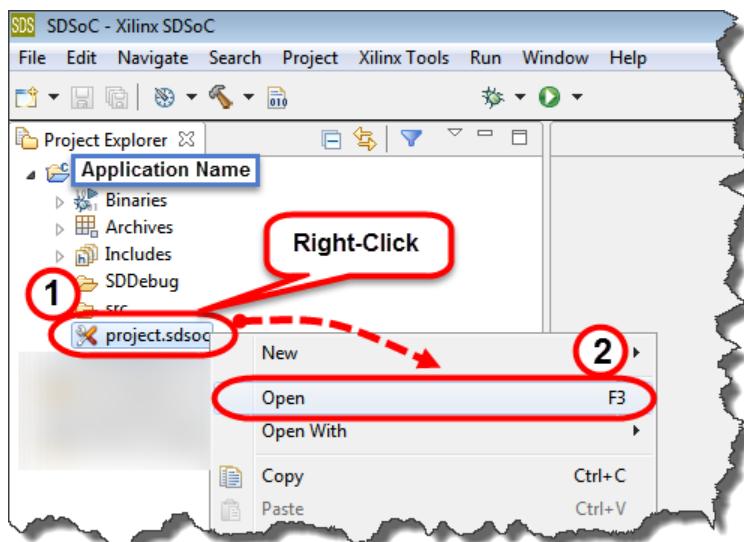


Figure 378: Opening the Project Overview

- 1-1-4. Under the Actions area in the SDSoc Project Overview, ensure that the connection is set to:
 - o Standalone: Use the default settings (local).
 - o Linux: Make sure the connection is set to **Linux Agent** or any custom connection that may have been created.

1-1-5. Click the **Debug application** link (1).

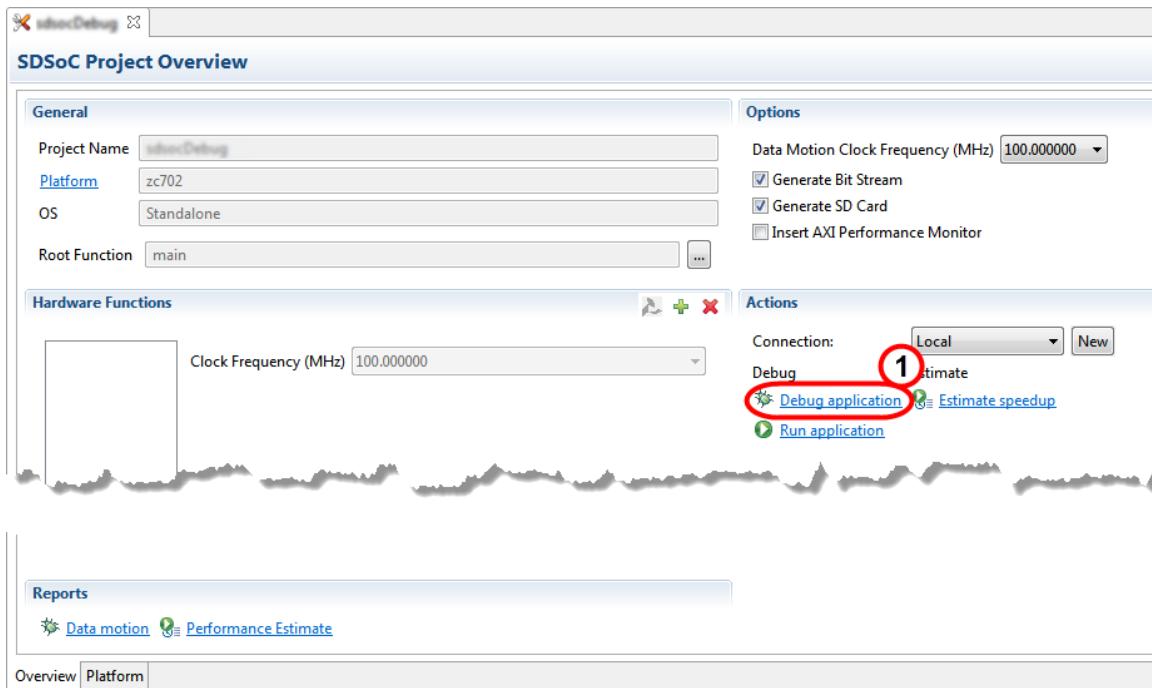


Figure 379: Clicking Debug Application

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.

1-1-6. If a dialog box asking to perform a perspective switch appears, click **Yes**.

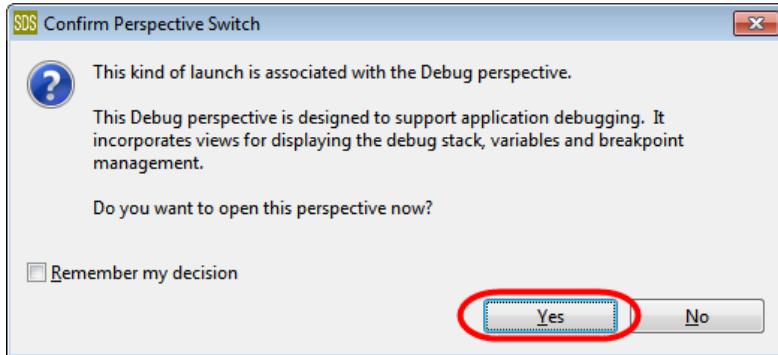


Figure 380: Confirm Perspective Switch to Debug

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).

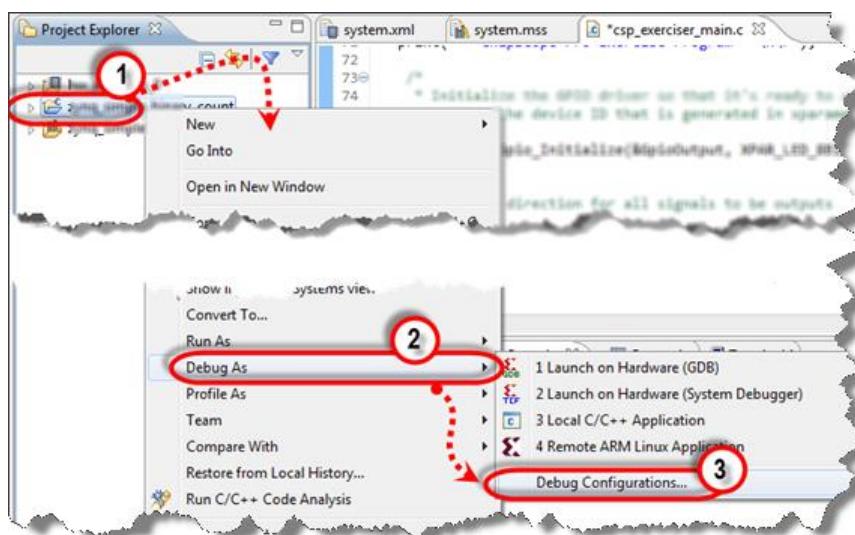


Figure 381: Creating a Debug Configuration

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).

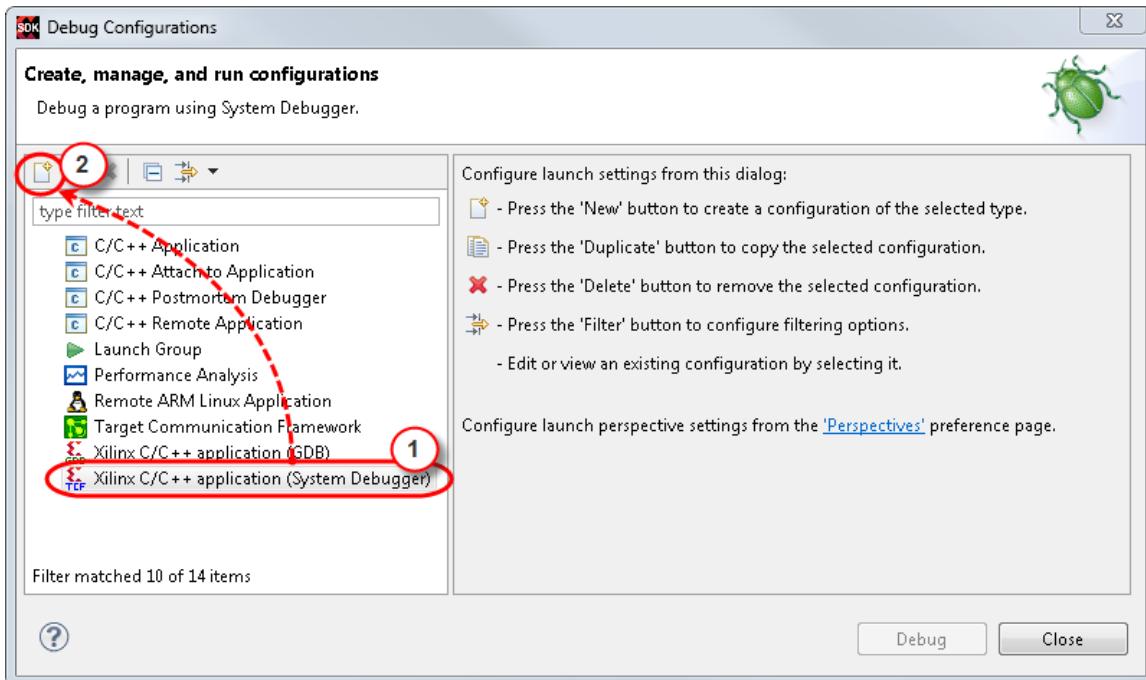


Figure 382: Creating a New Debug Configuration

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**.

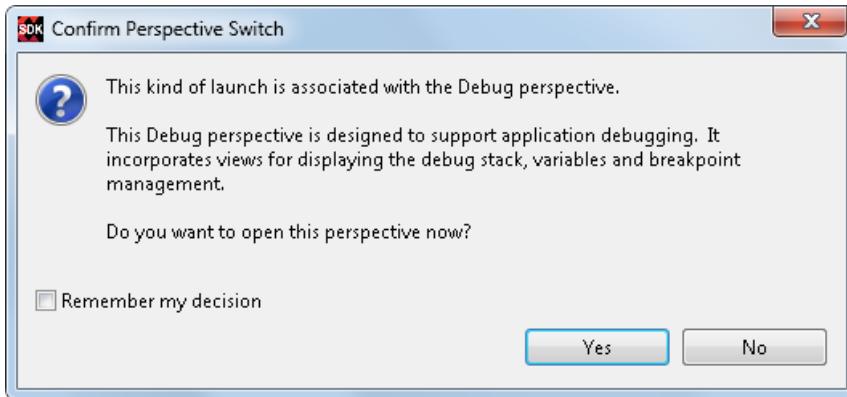


Figure 383: Confirming Switch of Perspective

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).

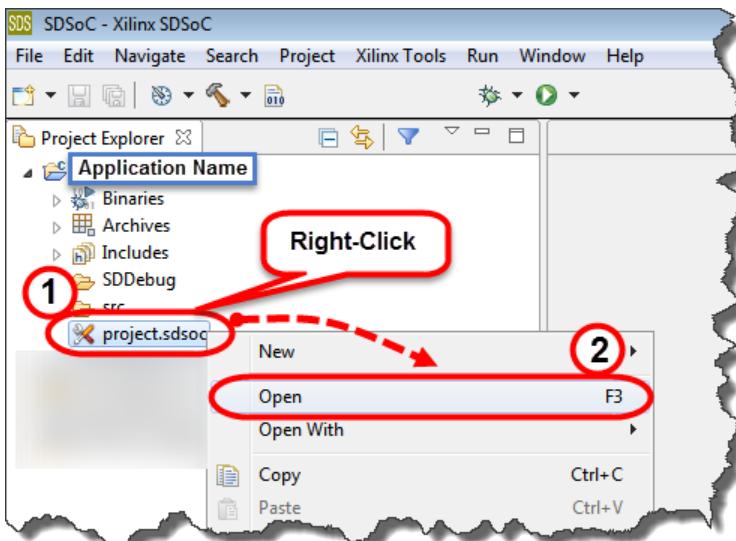


Figure 384: Opening the Project Overview

1-2-4. Click the **Debug application** link (1).

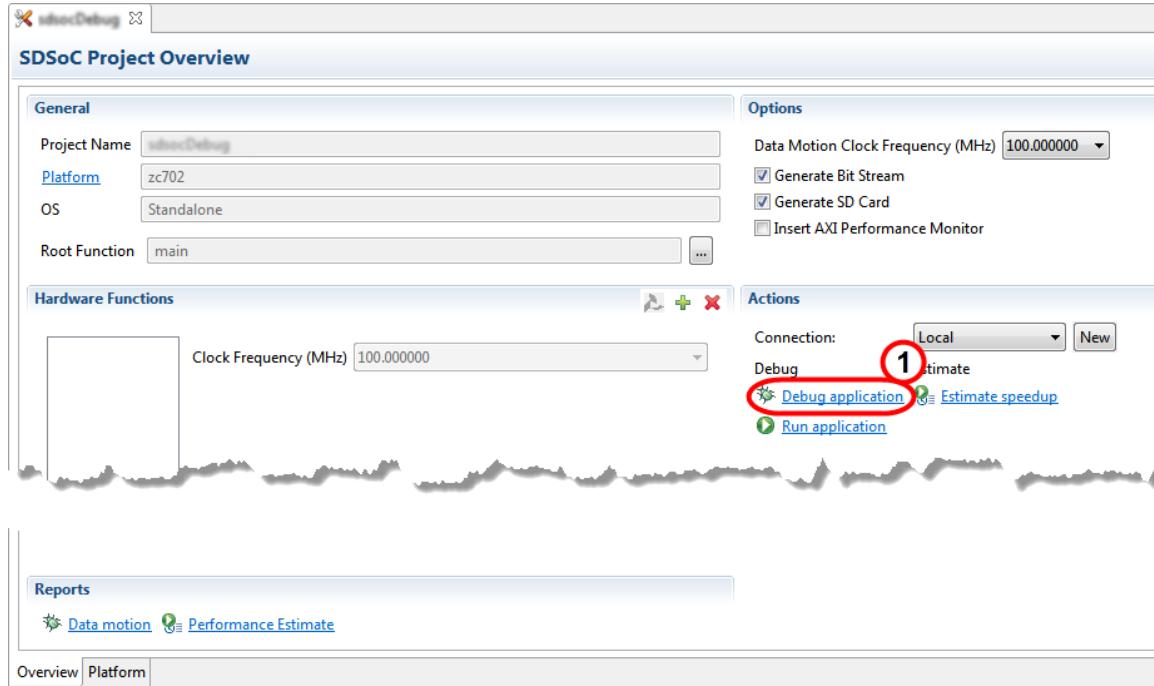


Figure 385: Clicking Debug Application

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**.

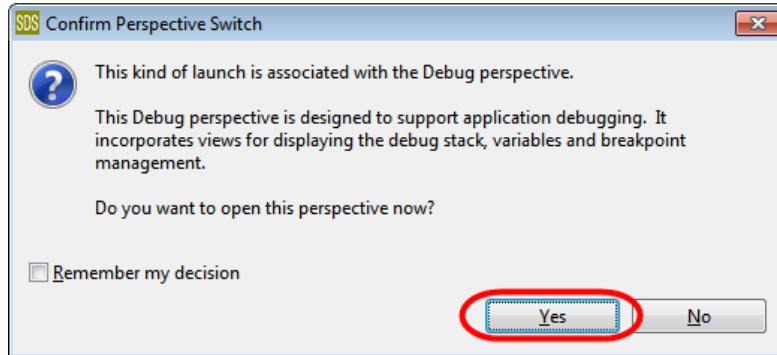


Figure 386: Confirm Perspective Switch to Debug

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.

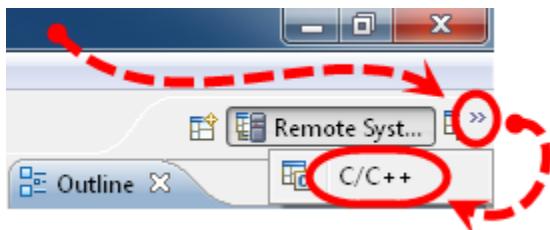


Figure 387: Changing Perspective

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).

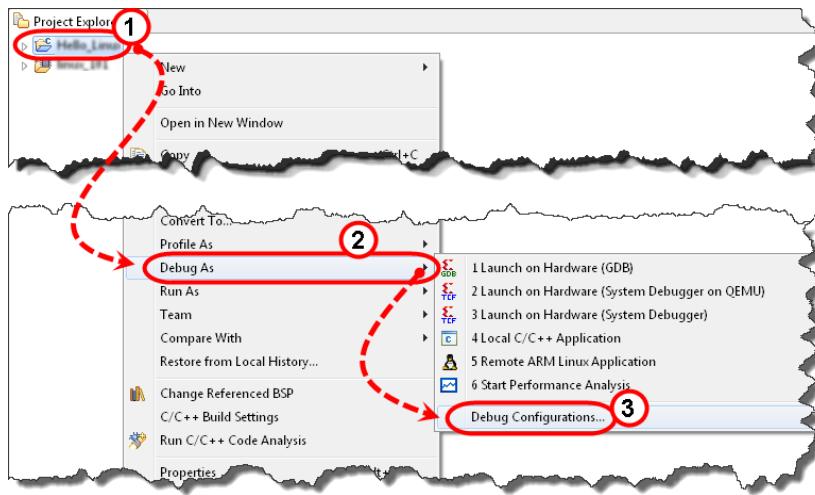


Figure 388: Selecting Debug Configurations

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.

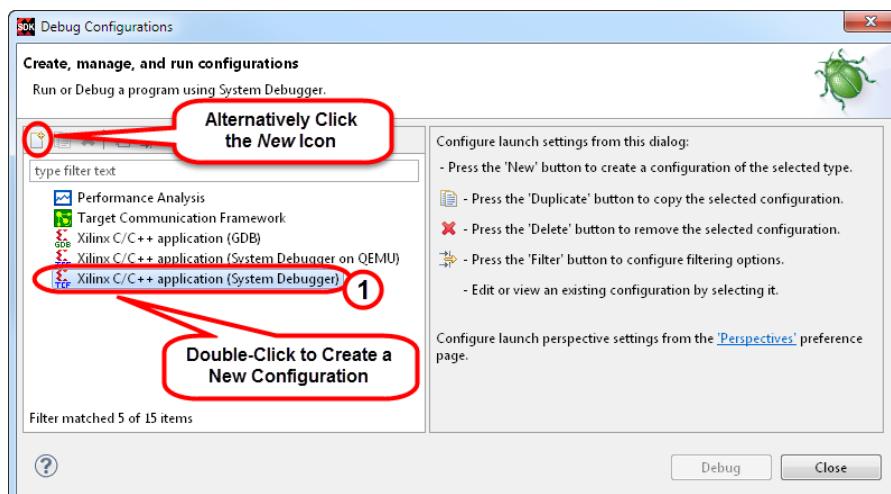


Figure 389: Creating a New System Run/Debug Configuration

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).

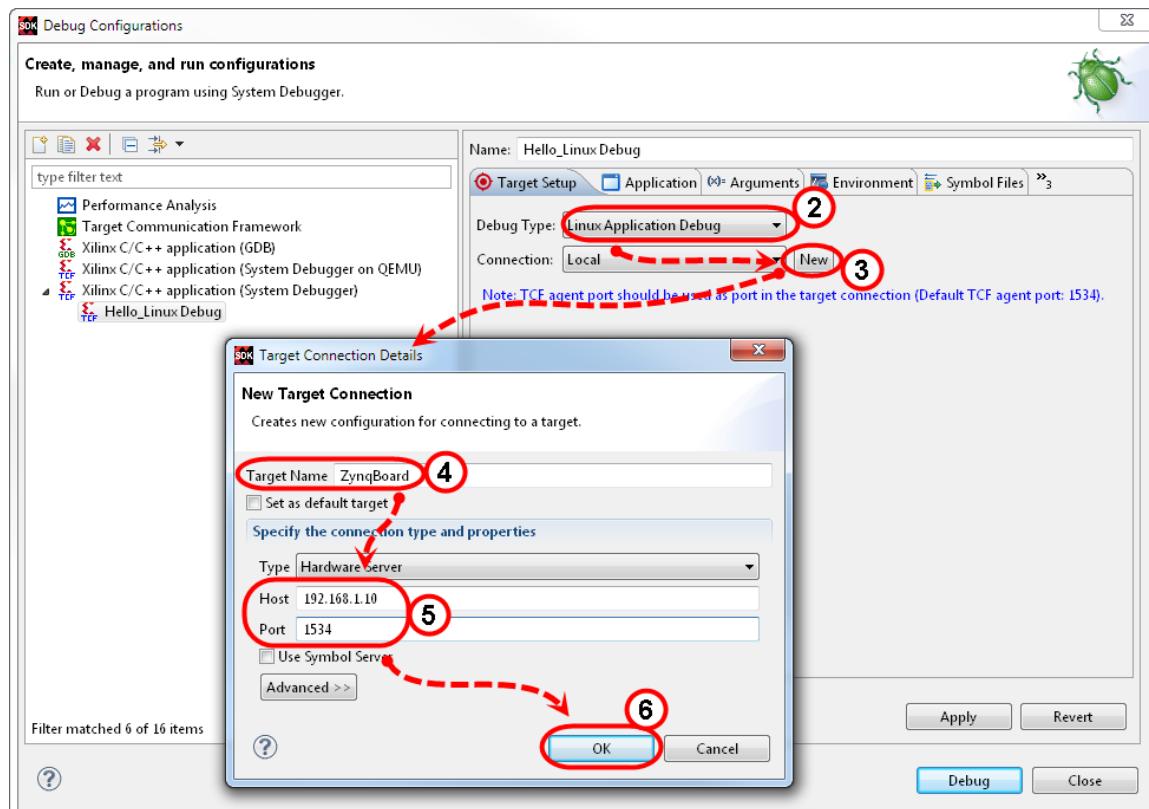


Figure 390: Selecting the Debug Type and Connection

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**\labs\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).

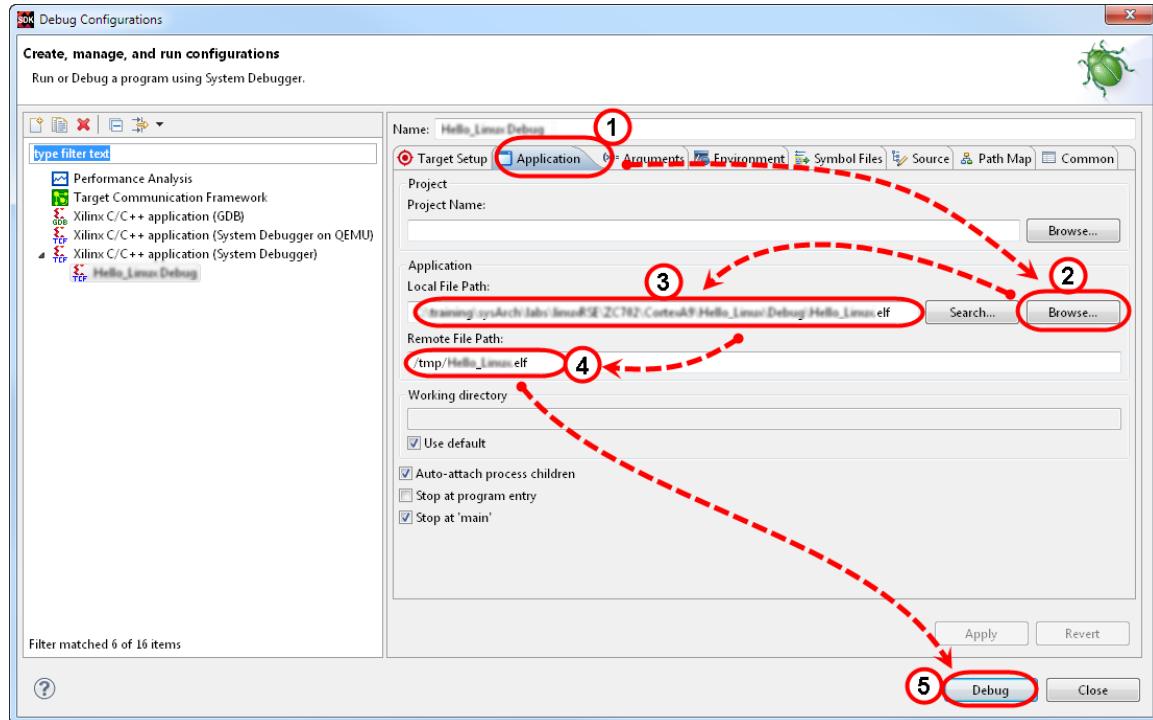


Figure 391: Selecting the Local File Path and Remote File Path

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.

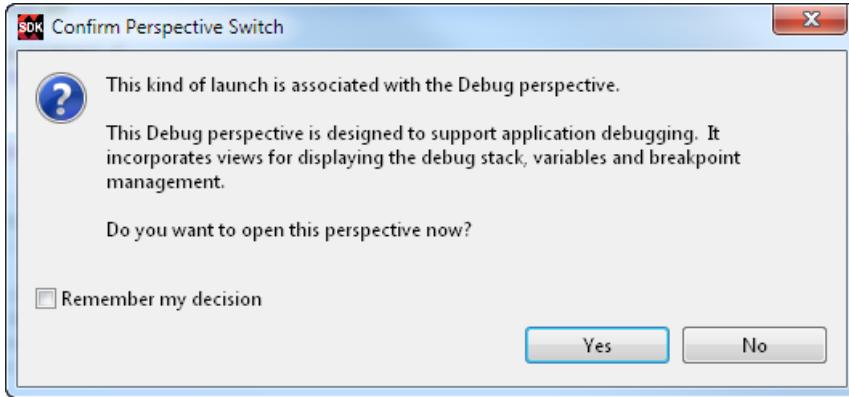


Figure 392: Confirming Perspective Switch

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).

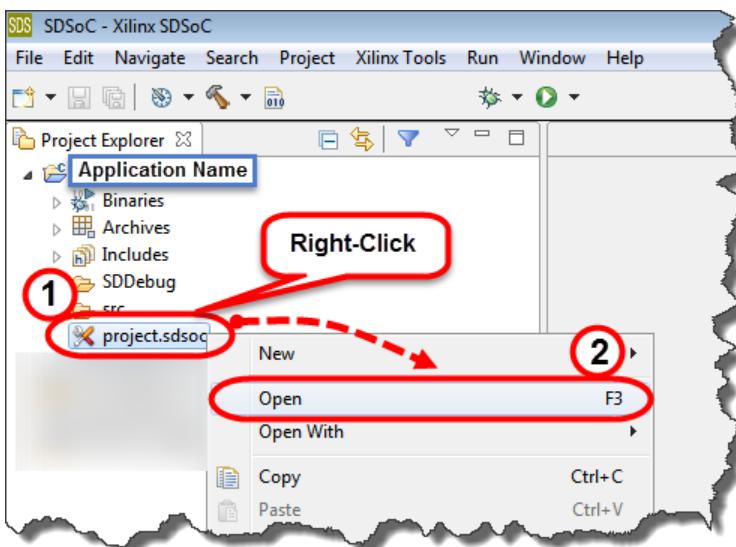


Figure 393: Opening the Project Overview

- 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.

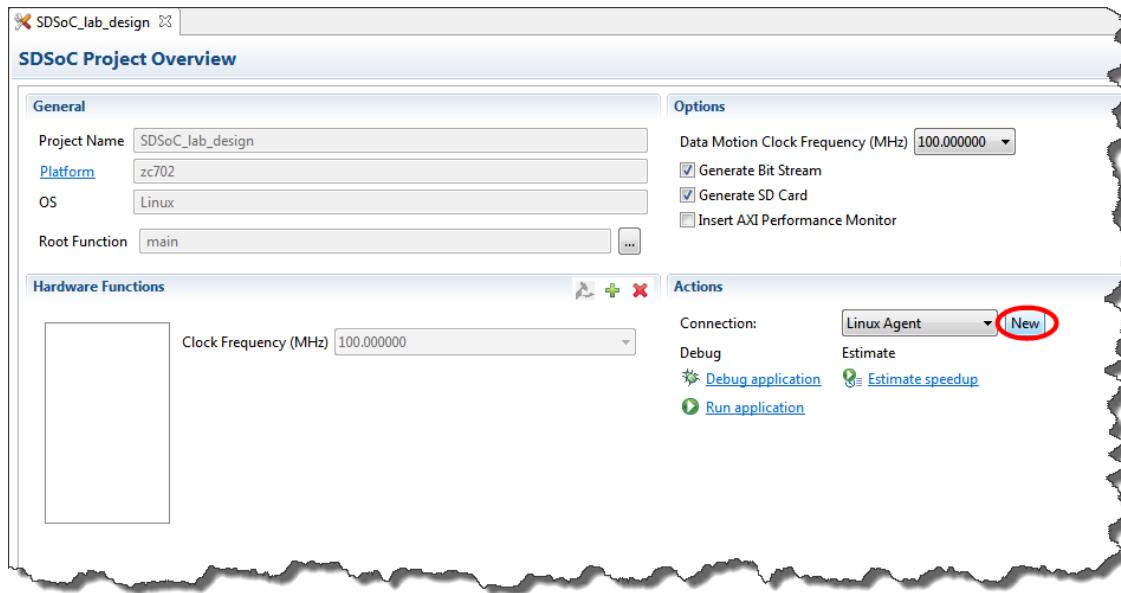


Figure 394: Creating a New Hardware Connection

- 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.

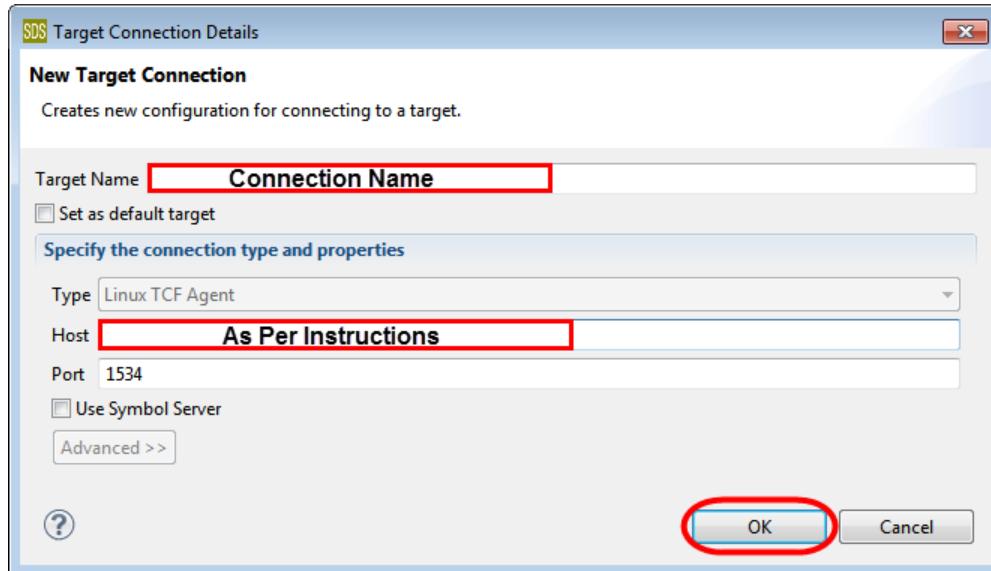


Figure 395: Target Connection Details

- 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).

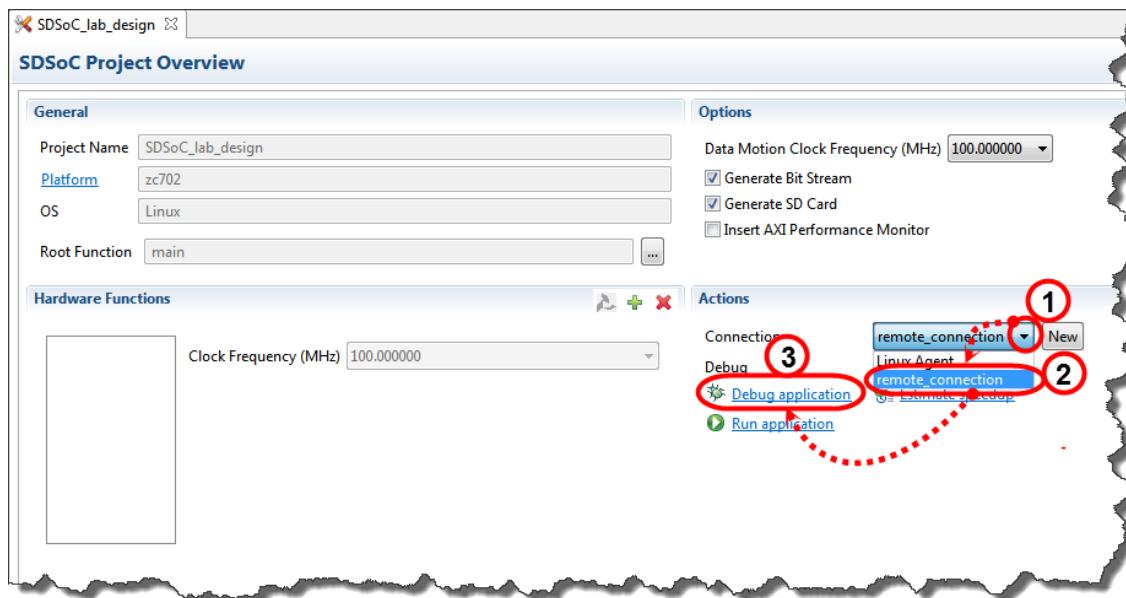


Figure 396: Selecting Connection and Debug

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**.

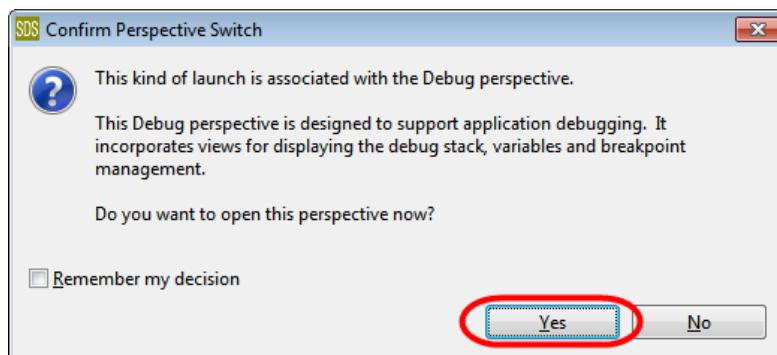


Figure 397: Confirm Perspective Switch to Debug

The SDSoc toll will now program the PS/PL and switch the perspective to the Debug view.

1-6. Optional for Linux: 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. Using the Debug view, if it is not already expanded, click the down arrow to expand the top-level debug session (1).
- 1-6-2. Click the down arrow (if it is not already expanded) to expand the next level showing the binary that is being executed (2).
- 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.

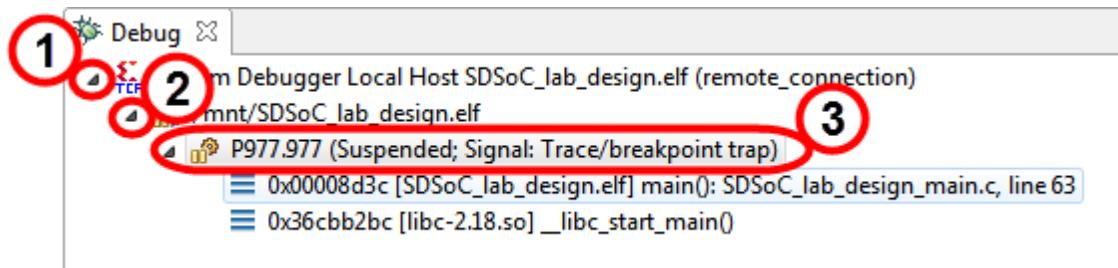


Figure 398: Highlighting the Debug Session Application

Tip: In the figure above and below, P977 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.

The screenshot shows a Linux terminal window with several tabs at the top: Console, Tasks, Terminal 1, Problems, Executables, and Memory. The terminal is connected via serial port COM5. The command `sh-4.3# ps` is entered and its output is displayed. The output shows a list of processes, with one entry circled in red: '977 root 3296 t /mnt/SDSoC_lab_design.elf'. This corresponds to the suspended process shown in Figure 398.

PID	USER	VSZ	STAT	COMMAND
1	root	1860	S	init [5]
2	root	0	SW	[kthreadd]
3		0		
942	c	77L..	L	/usr/sbin/tc1-agent -d -L - -10
946	root	3052	S	/bin/sh --
977	root	3296	t	/mnt/SDSoC_lab_design.elf
981	root	0	SWK	[kworker/1:1h]
996	root	2956	R	ps

Figure 399: Linux Terminal – ps Command

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.

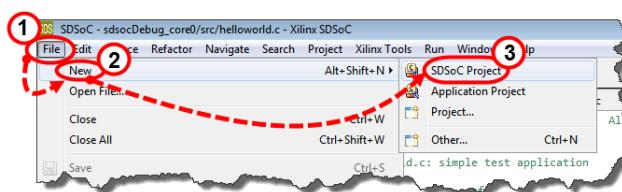


Figure 400: Creating an SDSoc Tool Project Using the Wizard

- 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.

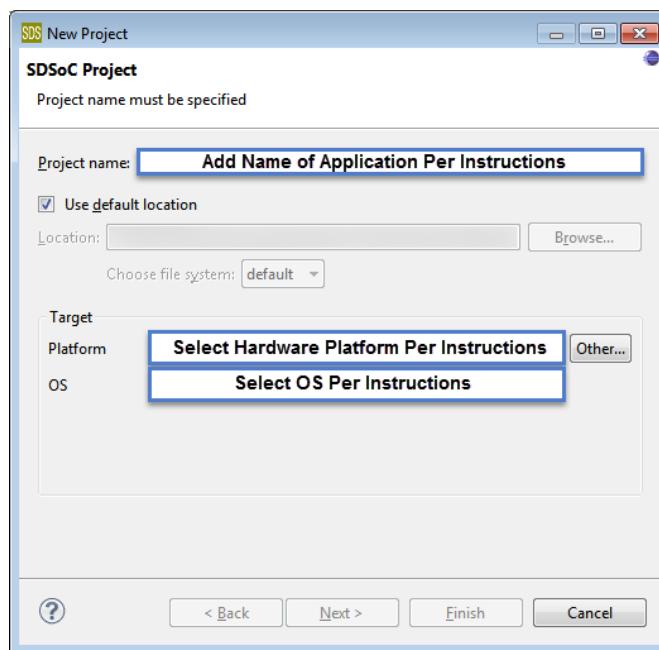


Figure 401: New SDSoc Project Dialog Box

1-1-5. Click **Next** to select the template for this application.

1-1-6. Select **an application template**.

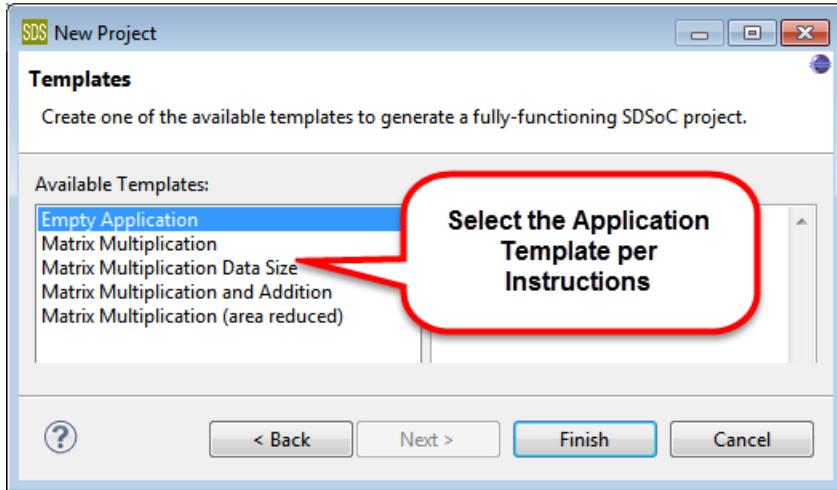


Figure 402: Selecting Application Template

1-1-7. Click **Finish** to create the new project.

Configuring the Terminal

1-1. Open the Terminal tab.

1-1-1. Select the **Terminal** tab to access the terminal icons (1).

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 (2).

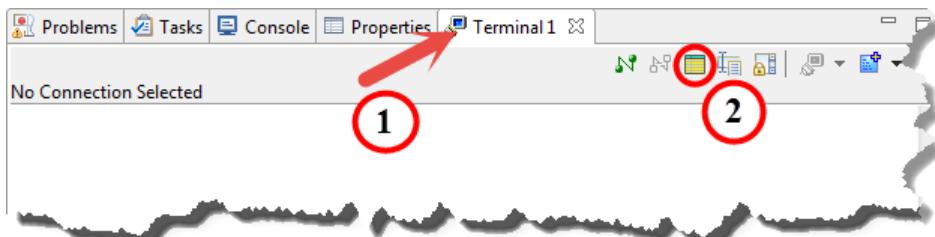


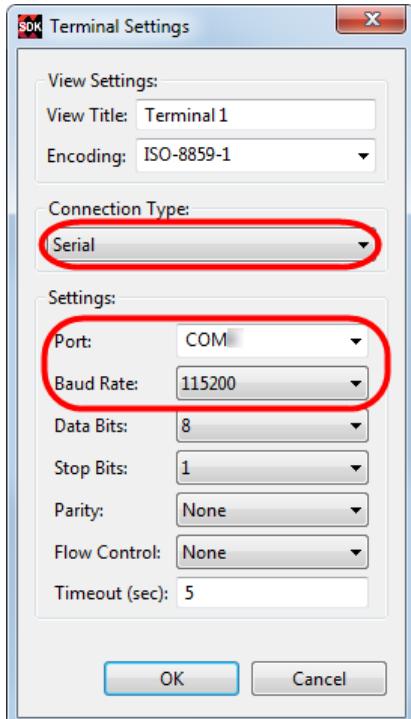
Figure 403: Accessing the Terminal Settings

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 404: Configuring the Terminal Settings****1-2-4.** Click **OK**.

The terminal session will be connected to the associated COM port on the PC.

Generating the Software Estimate from the SDSoc Tool

1-1. Generate the software estimation.

- 1-1-1.** If the SDSoc Report Viewer is not open and showing the results for an estimation, right-click **SDEstimate > _sds > est > perf.est** and select **Open** to open the project performance report.

Note that when a previously run report is opened, the software-only application performance and speed up is not available—it must be rerun. This information is available when SDEstimate is run and the board is properly connected and powered on.

- 1-1-2.** Click the **Click Here** link above the Details section to run only the "software-only application performance and speedup" test.

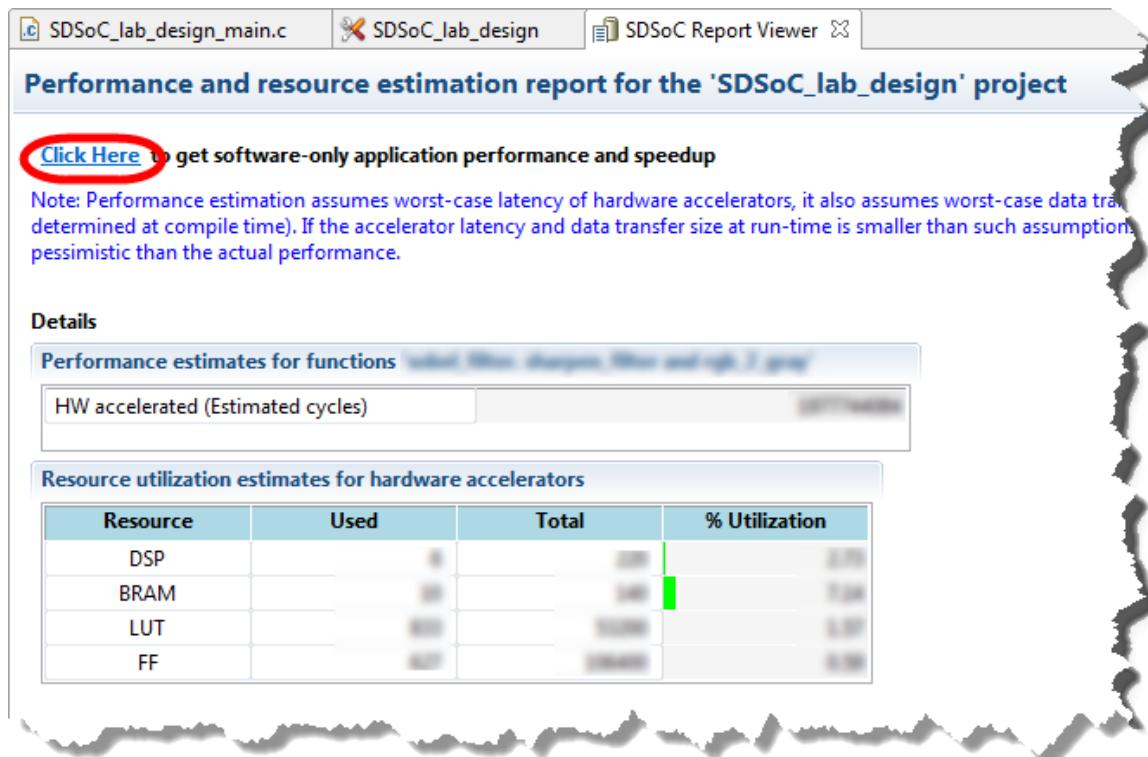


Figure 405: Gathering Software-only Performance Information

The Run Application To Get Its Performance dialog box opens.

- 1-1-3.** Establish a software connection between the host machine and the target board.

Standalone users: Use the default settings when your board is connected to your PC (local).

Linux users: Make sure that the Connection is set to Linux Agent or the New Connection set by you.

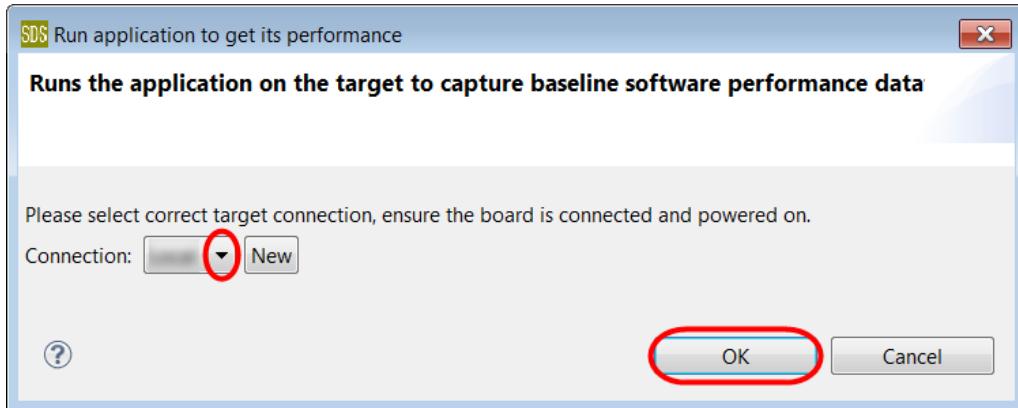


Figure 406: Run Application To Get Its Performance Dialog Box

- 1-1-4. 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.

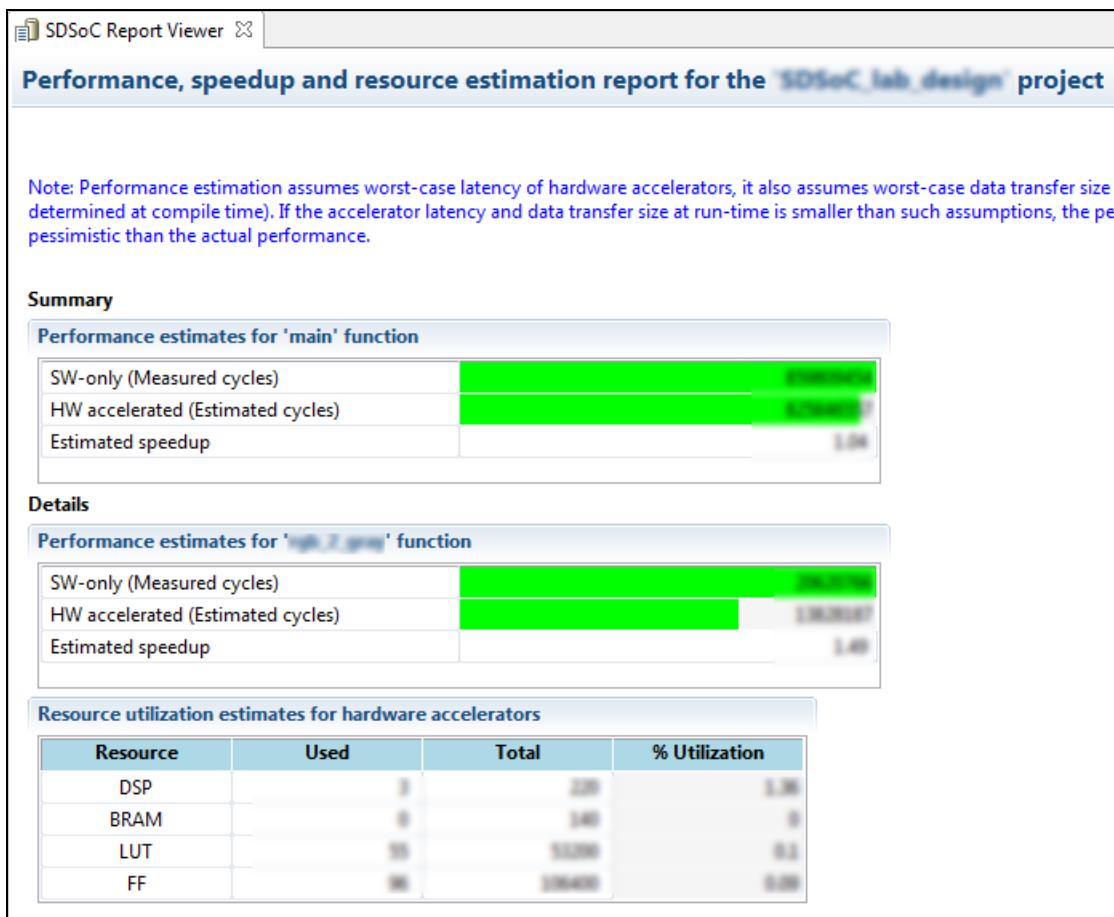


Figure 407: Full Estimate Report for Optimized Design

If the SDSoc Report View tab closes, it can be reopened by double-clicking **SDEstimate** > **_sds > est > perf.est**.

The SDSoc Report Viewer 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.** Under the project listed in the Project Explorer, expand the desired project > **SDEstimate** > **_sds > est**.

Note: If the *SDEstimate* folder is not seen, then you will need to rebuild the SDEstimate configuration. The fastest way to do this is to click the pull-down menu next to the Build icon (🔨) and select **SDEstimate**. This will set the build configuration to SDEstimate and build the SDEstimate.

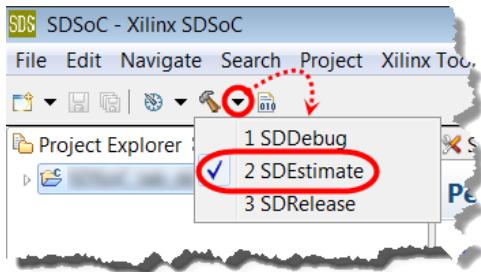


Figure 408: Setting the SDEstimate Configuration and Build

- 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).

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 > Show View > Outline**.

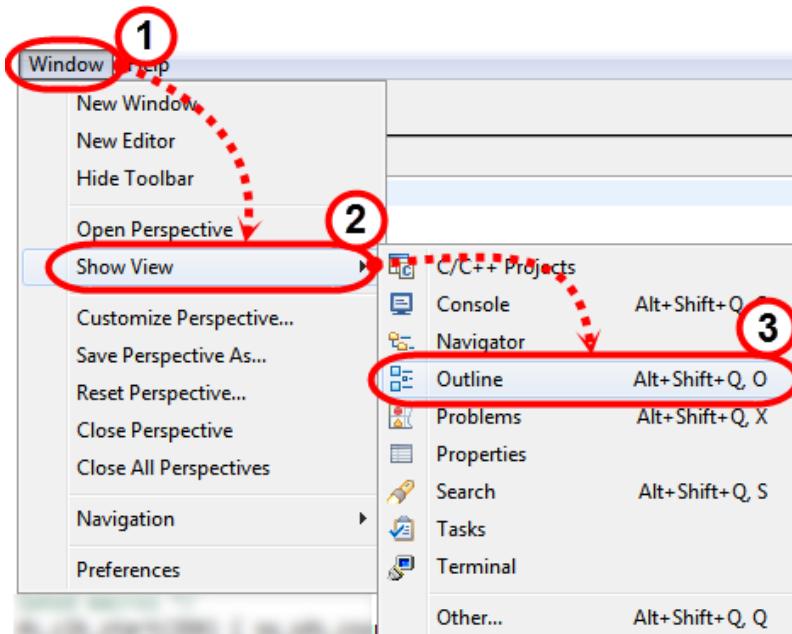


Figure 409: Opening the Outline View

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. If the Write option is unchecked, select it so that, like the Read option, if the variable is written to, it too will trigger a pause in execution (4).

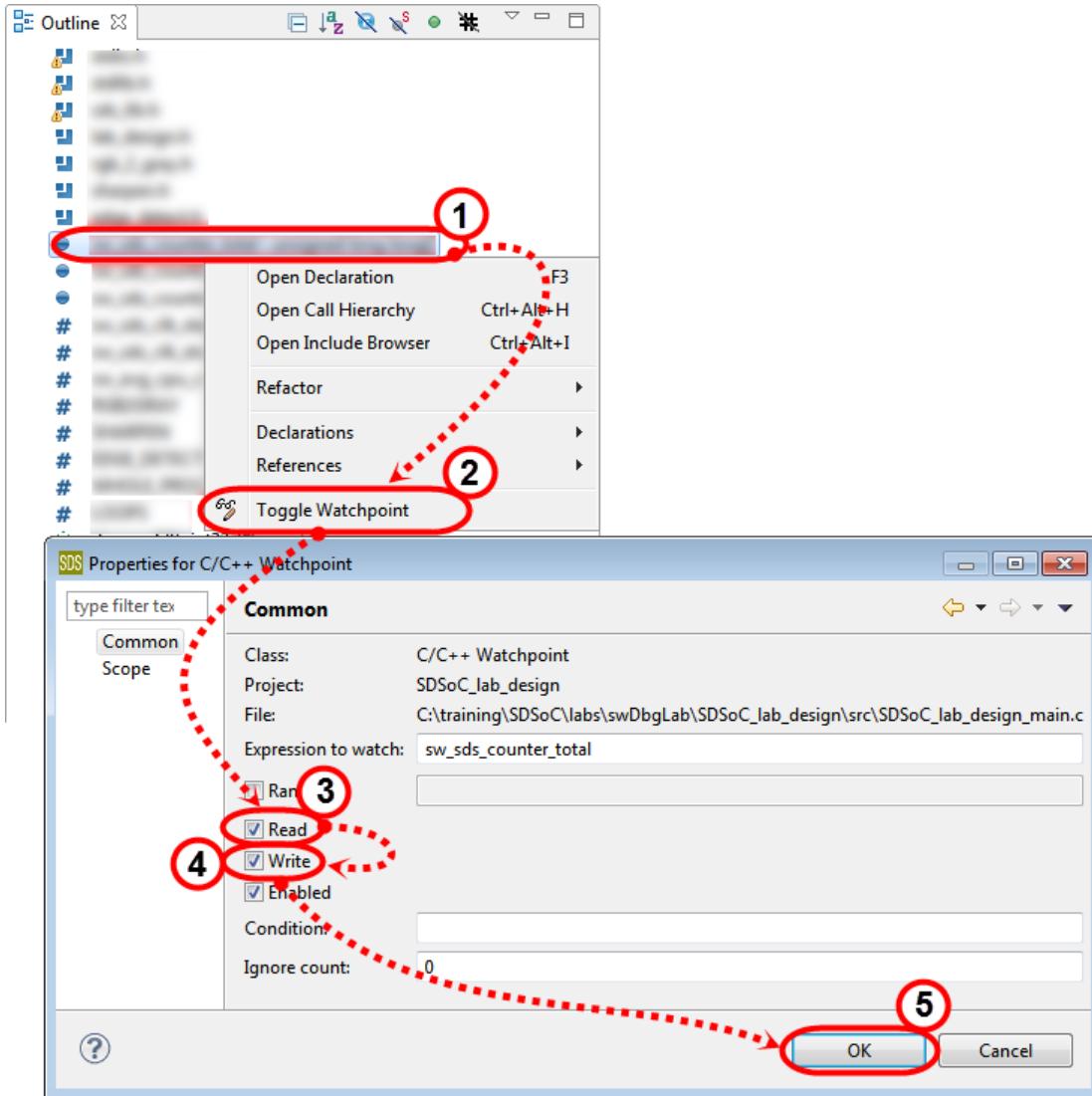


Figure 410: Toggling the Watchpoint

- 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).
- 1-1-2.** Select **Build Configurations** to view the options for building the different types of configurations (2).
- 1-1-3.** Select **Set Active** to see the three available configurations (3).
- 1-1-4.** Select **the needed build configuration (SDEstimate, SDDebug, or SDRelease)** to build the desired configuration (4).

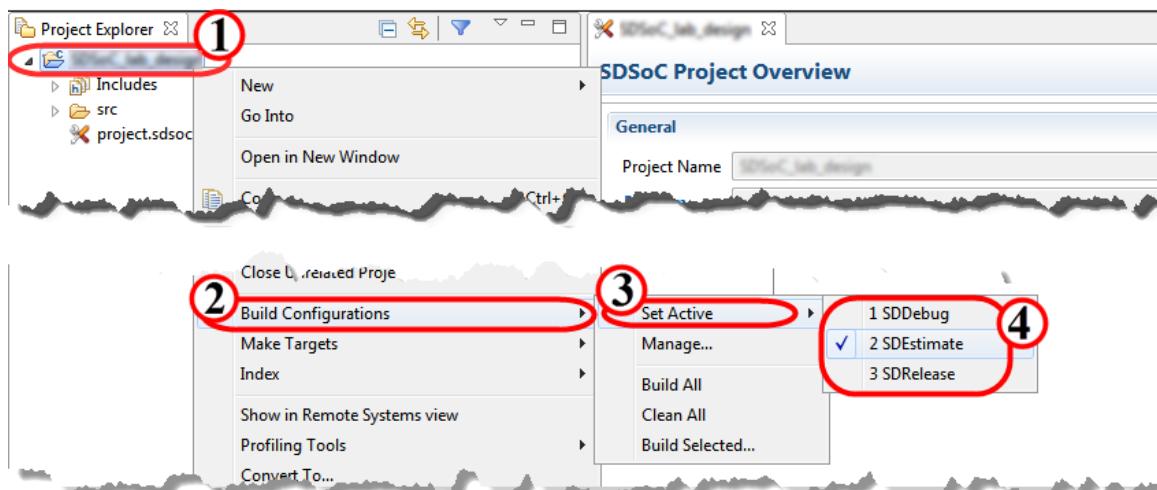


Figure 411: Selecting the SDEstimate Build Configuration (Example)

Marking a Function for Hardware (Dashboard Method)

1-1. Mark *the functions you want accelerated* for hardware acceleration.

- 1-1-1. Using the Project Explorer tab, expand the project for which you want to mark functions for hardware.
- 1-1-2. Double-click the **project.sdsoc** entry to open the main dashboard.
- 1-1-3. If not already active, select the tab named with the project (1) to access the Hardware Functions list (2).
- 1-1-4. Click the **Add Hardware Function** icon (+) to open the Function Selection Wizard (3).

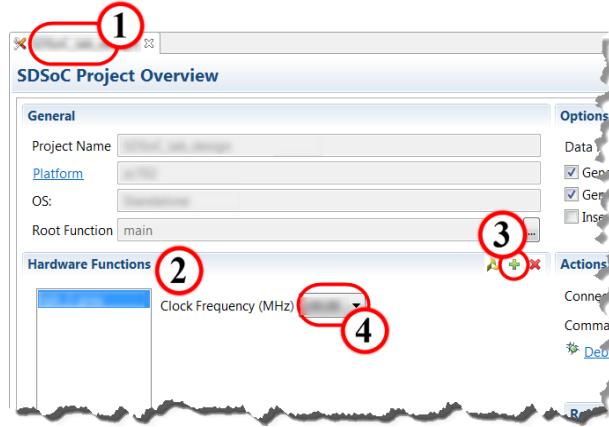


Figure 412: Targeting a Function to Hardware from the SDSoc Project Overview

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).

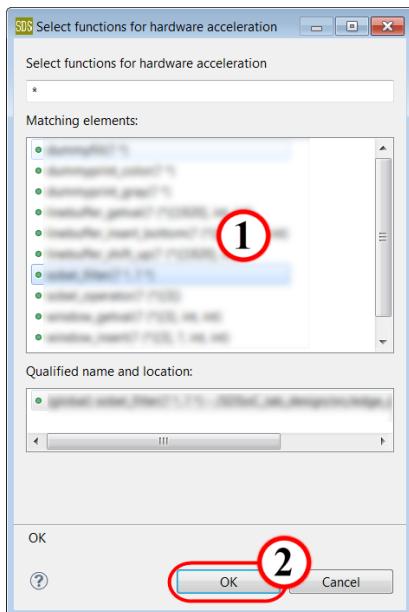


Figure 413: Choosing the Function to Move to Hardware

- 1-1-6.** Click **OK** to accept.

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 SDSoc 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 > *SDRelease > _sds > p0 > ipi*.

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.

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 > SDSoc 2016.1 > Vivado Design Suite > Vivado 2016.1**. Once the tool opens, click the **Open Project** link and navigate to the **XPR** file.

- 1-1-3.** Under the Flow Navigator, click **Open Block Design** to open the IP integrator view, which will show you the block design for this embedded system.

Viewing the Generated Accelerator

- 1-1. Open the Vivado HLS tool projects for an accelerator/hardware function.**

- 1-1-1.** Expand **SDSoC_lab_design > SDEstimate > _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).

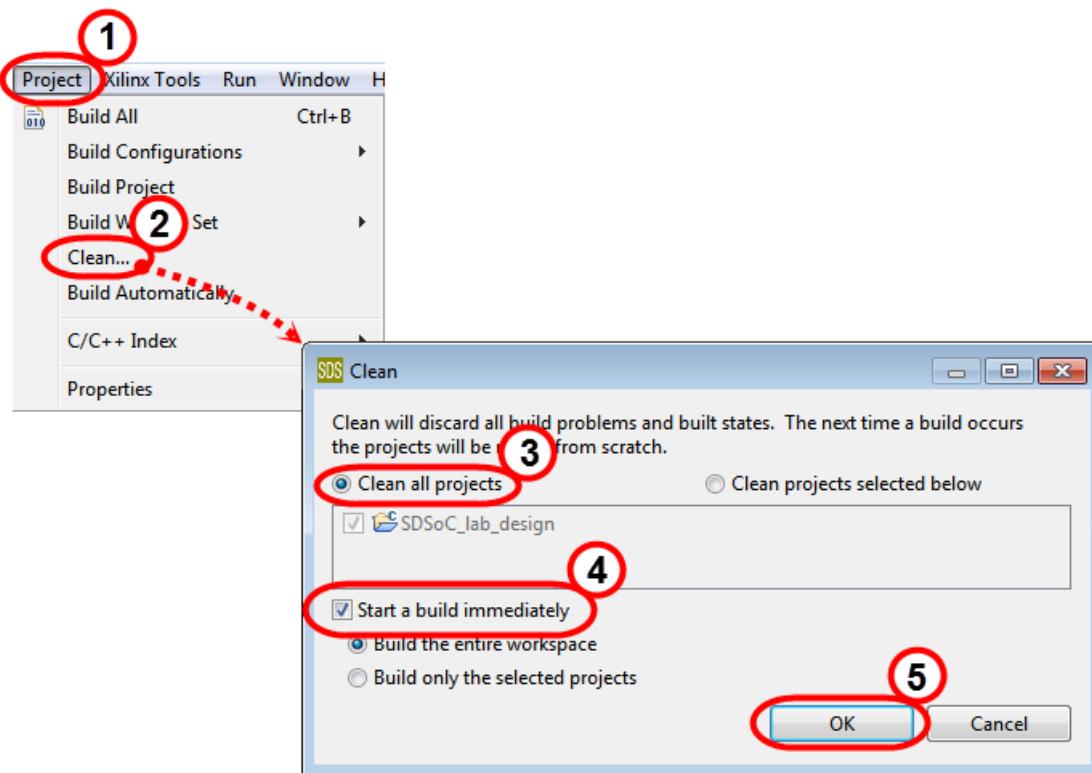


Figure 414: Project Clean and Build

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

In This Section

Creating a New PetaLinux Project	309
Creating a New Application	310
Creating a New Application	310
Creating a New Application	311
Selecting the New Application to be Included in the Build Process.....	312
Building the Image	313
Booting the Linux Image on the Board (Running Netboot)	314
Running the Application in QEMU	315

Creating a New PetaLinux Project

1-1. Use the `petalinux-create` command to create a new embedded Linux platform and choose the platform.

- 1-1-1.** Run the following command from the `lab_name` directory to create a new PetaLinux project:

```
[host] $ petalinux-create -t project -s /opt/pkg/your BSP  
name-v2016.1-final.bsp
```

Note: After the command is executed the following will be displayed.

INFO: Create project:

INFO: Projects:

INFO: * your BSP name-2016.1

INFO: has been successfully installed to
`/home/xilinx/training/*****/labs/lab name/`

INFO: New project successfully created in
`/home/xilinx/training/*****/labs/lab name/`

The above command assumes that the board support package (BSP) is installed in the `/opt/pkg` directory. Modify the path if the BSP is in a different location.

Note: The project directory is `~/training/*****/labs/lab name/` your BSP name-2016.1.

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  
INFO: Create apps: your application  
INFO: New apps successfully created in  
/home/petalinux/training/emblinux/labs/lab  
name/Avnet-Digilent-ZedBoard-2016.1/components/apps/your  
application
```

The new application that you have created can be found in the <project-root>/components/apps/your application directory, where <project-root> is ~/*****/labs/lab name/Avnet-Digilent-ZedBoard-2016.1.

To create a C++ application template, pass the --template c++ option.

Creating a New Application

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  
INFO: Create apps: your application  
INFO: New apps successfully created in  
/home/petalinux/training/emblinux/labs/lab  
name/Avnet-Digilent-ZedBoard-2016.1/components/apps/your  
application  
INFO: Enabling created component...  
INFO: It has been enabled to linux/rootfs
```

The new application you have created can be found in the <project-root>/components/apps/your application directory, where <project-root> is ~/*****/labs/lab name/Avnet-Digilent-ZedBoard-2016.1.

The --enable option automatically enable it in the project's root file system. To create a C++ application template, pass the --template c++ option.

Creating a New Application

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/*****/labs/lab name/your BSP  
name-2016.1/components/apps/your application
```

```
INFO: Enabling created component...
```

```
INFO: It has been enabled to linux/rootfs
```

Note: The new application that you have created can be found in the <project-root>/components/apps/your application directory, where <project-root> is ~/training/*****/labs/lab name/your BSP name-2016.1.

Note: If anything indicating that **webtalk failed** is displayed, it can be safely ignored.

Note: 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 `~/*****/labs/lab name/your BSP name-2016.1`.
- 1-1-2. Launch the rootfs configuration menu by entering the following command:

```
[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 "Error: Failed to config linux/rootfs! Your display is too small to run menuconfig".

The linux/rootfs configuration menu opens.

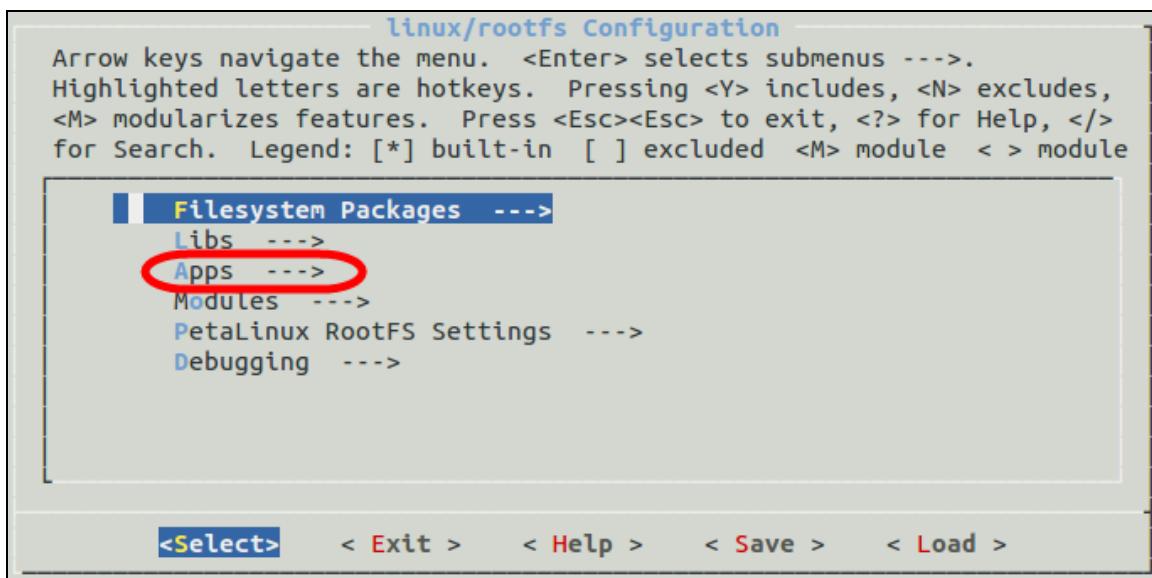


Figure 415: linux/rootfs Configuration Menu

- 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. Move to **your application** and press <Y> to select the application.

Selecting the application will be included in the build process.

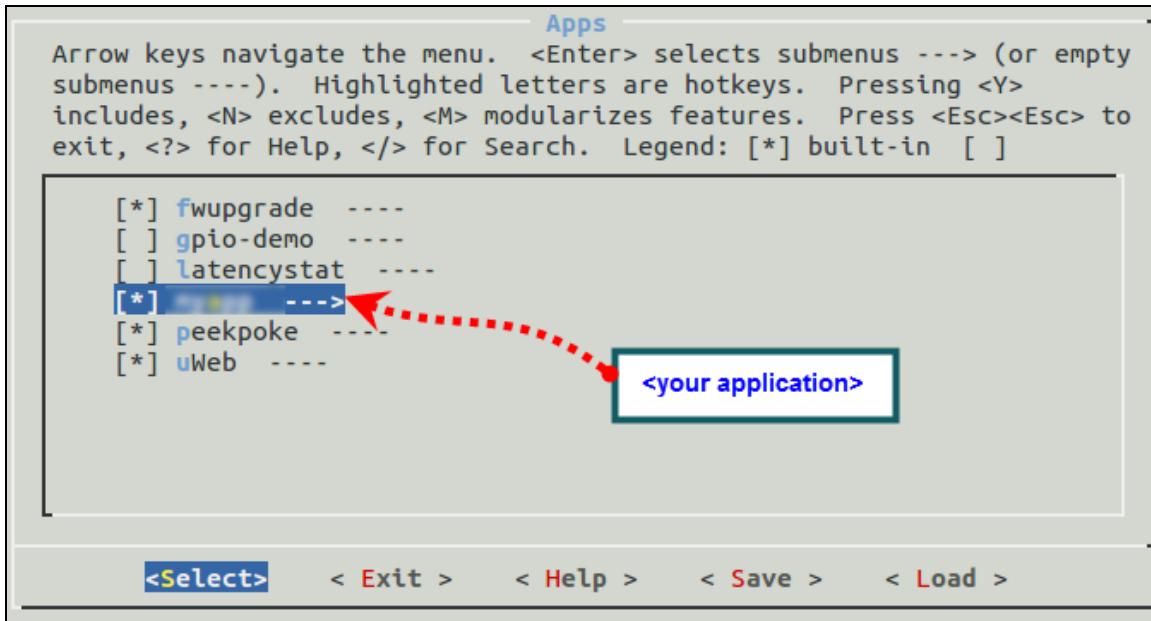


Figure 416: Selecting the New Application

- 1-1-6.** Press <Enter> to open the **your application** options.

By default, there are no additional options for *your application*. Advanced users can modify the Kconfig file in the myapp directory to add custom options.

- 1-1-7.** Select and press **Exit** to return to the main menu.

Building the Image

1-1. Build the image.

- 1-1-1.** Make sure that you are in the root of the PetaLinux project directory.

- 1-1-2.** Enter the following command to build the image:

```
[host] $ petalinux-build
```

Note: This process will take approximately 20 minutes.

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 `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 [0005.04] (Aug 11 2015 - 09:33:00)

DRAM:  ECC disabled 512 MiB
MMC:   zynq_sdhci: 0
SF: Detected S25FL256S_64K with page size 256 Bytes, erase size 64 K
iB, total 32 MiB
In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
Hit any key to stop autoboot:  0
U-Boot-PetaLinux> █
```

Figure 417: Stopping the Autoboot

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:

```
U-Boot-PetaLinux> setenv ipaddr 192.168.1.10
```

- 1-1-3.** Set the TFTP server IP to the host IP by typing the following command in the u-boot console:

```
U-Boot-PetaLinux> setenv serverip 192.168.1.1
```

- 1-1-4.** Save the environment to the SPI flash:

```
U-Boot-PetaLinux> saveenv
```

- 1-1-5.** Download and boot the new image using TFTP by executing this command in the u-boot console:

```
U-Boot-PetaLinux> 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.

Note: This may take approximately 2 to 3 minutes to complete the download. There is a significant delay in the network because of the VirtualBox environment. But generally, if you are running this command on the Linux machine, it should take less time to download 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.

Board, OS, COM, and IP Address Tasks

In This Section

Configuring the SDK or SDSoC Terminal for the ZC702 or Tera Term for the ZedBoard	316
Determining the COM Port Assigned by the USB Driver.....	319
Configuring the SDK Terminal	322
Configuring the SDK Terminal	323
Linux Operations	325
Launch VirtualBox.....	327
Setting the Static Host IP Address on a PC (Windows 7).....	328
Setting Up the Hardware – ZC702 and ZedBoard	334
Verifying Board Jumper/Switch Settings Configured to Boot from the SD Card	338
Powering Up the Hardware Platform	338
Preparing an SD Card	338
Booting Linux.....	339
Launching and Configuring the Tera Term Terminal Program in Serial Port Mode.....	340
Connecting the ZC702 Board	342
Connecting the ZedBoard.....	344
Connecting the ZedBoard without the FMC-CE Card	346
Setting the Jumpers on the ZC702 Board.....	347
Setting the Jumpers on the ZC706 Board.....	350
Setting the Jumpers on the ZCU102 Board.....	354
Setting the Jumpers on the ZedBoard	355
Setting Up the Hardware - KC705	356
Setting up the Hardware - KCU105.....	359

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 (gear).

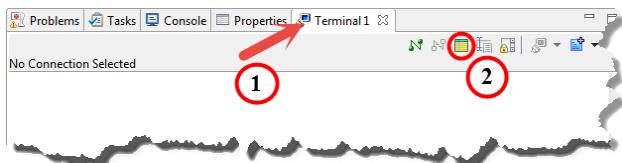


Figure 418: Accessing the Terminal Settings

Alternatively, you can also click the **Connect** icon (green arrow) 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**

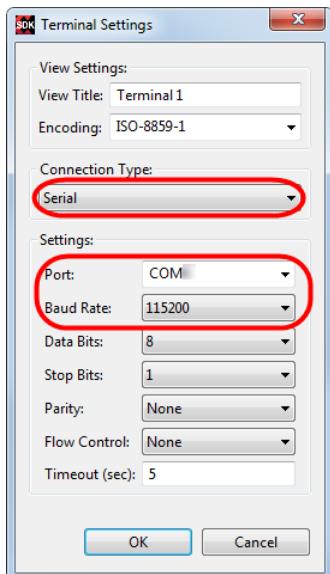


Figure 419: Configuring the Terminal Settings

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.

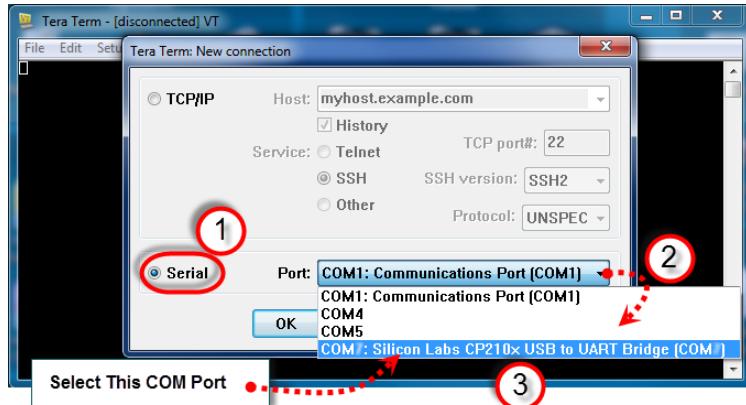
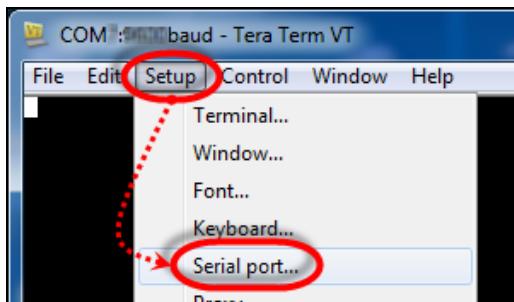


Figure 420: Selecting the COM Port

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.**

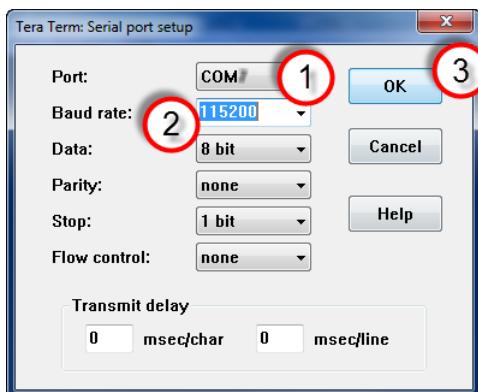
The terminal console window opens.

1-2-7. Select **Setup > Serial Port**.**Figure 421:** 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 422:** 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.

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.

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.

- 1-1-2. Open your computer's **Control Panel**.

Note that the Start button is typically located in the lower left corner of the screen. Occasionally, as shown below, it is in the upper left corner.

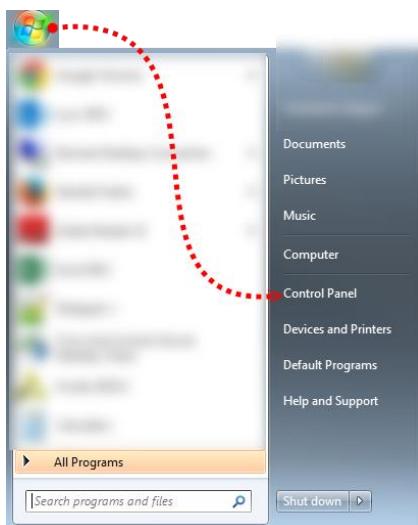


Figure 423: Opening the Control Panel from Windows 7

- 1-1-3.** Select **Small icons** from the View by drop-down list in the upper right to change the view of the Control Panel (1).
- 1-1-4.** Click **Device Manager** (2) to open the Device Manager window.

Note: You may be asked to confirm opening the Device Manager. If so, click **Yes**.

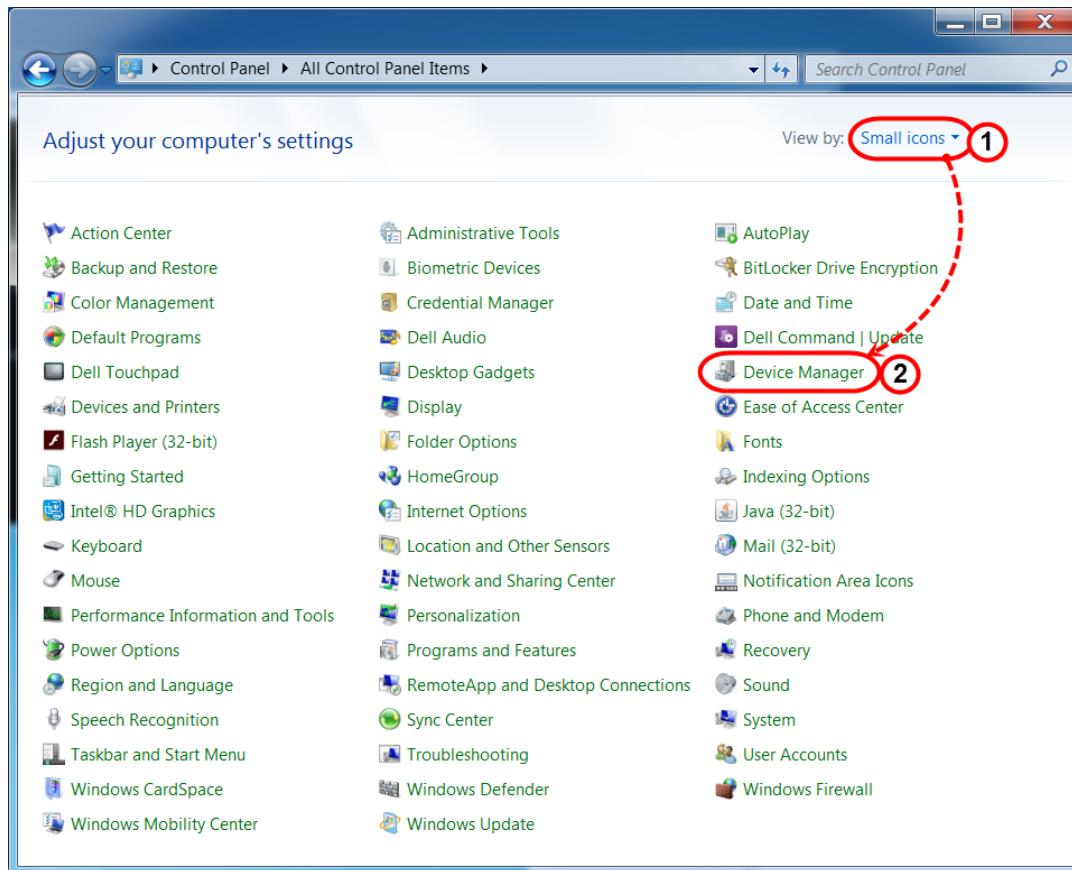


Figure 424: Control Panel - Device Manager

- 1-1-5.** Make certain that your board is powered on so that the UART-to-USB device is enumerated and appears in the following list.
- 1-1-6.** Expand **Ports (COM & LPT)**.
- Various boards use different USB bridge chips. The name of the driver varies correspondingly.
- **ZC702/KC705 board users:** Locate **Silicon Labs CP210x USB to UART Bridge (COM#)**.
 - **ZedBoard users:** Locate **USB/Serial Cypress (COM#)**.
 - **MicroZed users:** Locate **Silicon Labs CP210x USB to UART Bridge (COM#)**.
 - **ZCU102 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 stdio 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:

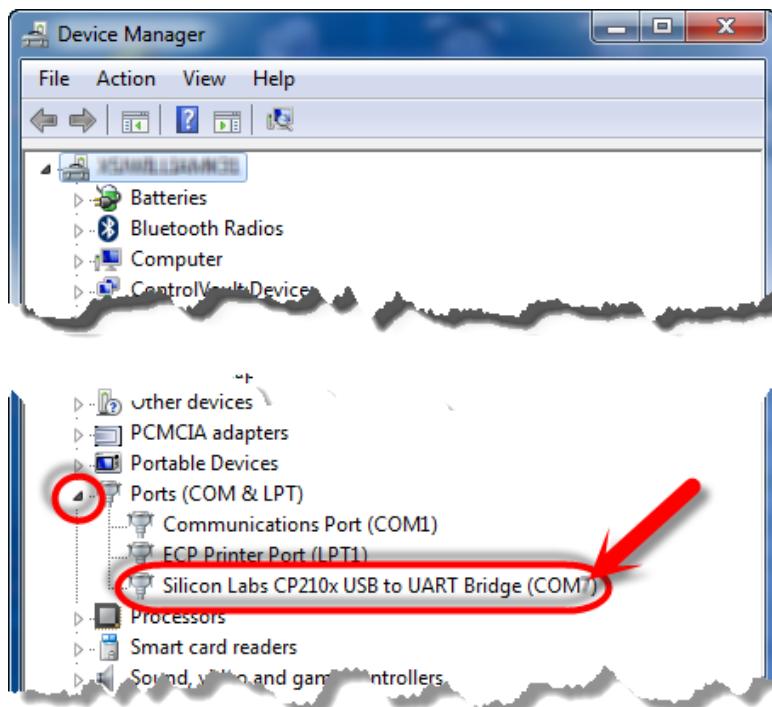


Figure 425: Locating the Silicon Labs Com Port Driver

- 1-1-7. Close the Device Manager by clicking the red 'X' in the upper-right corner of the panel.
- 1-1-8. Close the Control panel by clicking the red 'X' in the upper-right corner of the panel.

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 always available in the same panel that you would find the console.**
- 1-1-1.** If it is not available, select **Window > Show View > Other > Xilinx > SDK Terminal** to open the SDK Terminal tab.

- 1-2. Configure the SDK Terminal.**

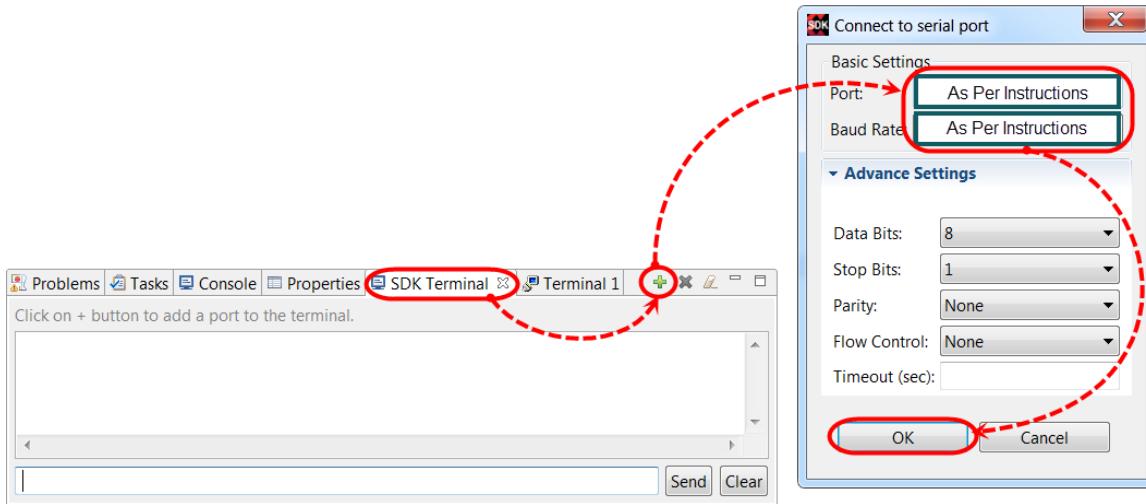


Figure 426: Configuring the SDK Terminal

- 1-2-1.** Click the green '+' sign to open the Connect to Serial Port dialog box.
- 1-2-2.** Select the serial port that is connected to the device you want to communicate with.
- 1-2-3.** Set the baud rate to **115200**.
- 1-2-4.** Leave the other settings at their defaults.
- 1-2-5.** Click **OK** to save these settings and begin the terminal session.

Configuring the SDK 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).

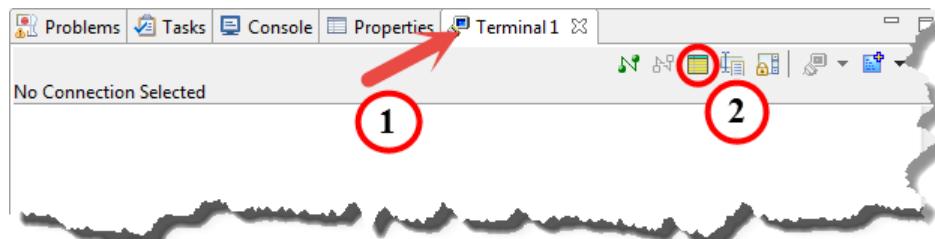


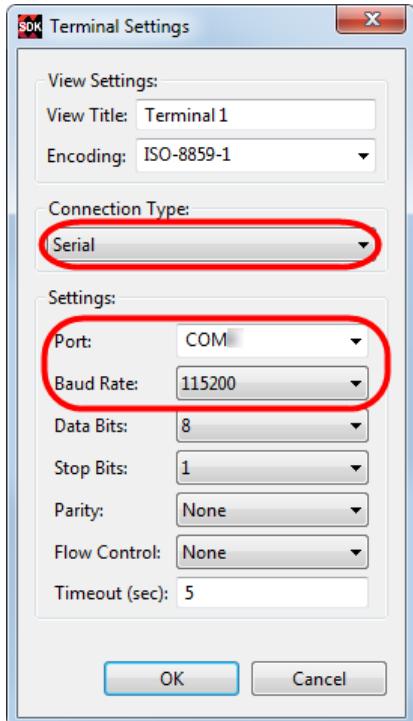
Figure 427: Accessing the Terminal Settings

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 428: Configuring the Terminal Settings****1-2-4.** Click **OK**.

The terminal session will be connected to the associated COM port on the PC.

Linux Operations

Opening a Terminal – Linux Ubuntu

A Linux terminal allows instructions to be entered directly via the command line interface (keyboard), 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. Press <Ctrl + Alt + T>.

This is the fastest and easiest way to launch a new terminal window with the current directory set to the home directory of the user.

Alternatively, you can:

- o Click the Ubuntu symbol on the side bar from the Ubuntu desktop (1).
- o Enter **terminal** in the search box (2).
- o Select the **Terminal** application (3).

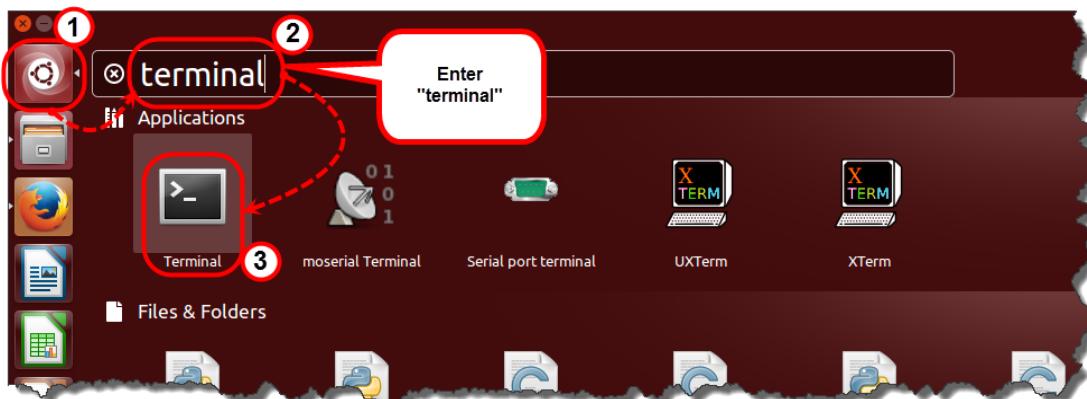


Figure 429: Ubuntu – Locating the Terminal

A new terminal window will open.

Copying Lab Files to the /home Directory

- 1-1. Copy the files and the directory structure from the shared Windows folder to your home directory in Linux.**

This will prevent some file permission problems and lets the tools run a trifle faster. At the end of the lab, you have the option of copying your work back to the shared directory.

For your convenience, a script has been prepared for you to make the copying effort less error prone and tedious.

- 1-1-1.** Enter the following shell script name and argument into the Linux terminal window:

```
/media/sf_training/setup_TopicCluster.sh *****
```

This script will create the necessary directories and copy all the files required for ***** to your working directory.

Shutting Down Ubuntu Linux

- 1-1. 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 **Shut Down** (2).

The Shut Down dialog box opens.

- 1-1-3.** Click **Shut Down** (3).

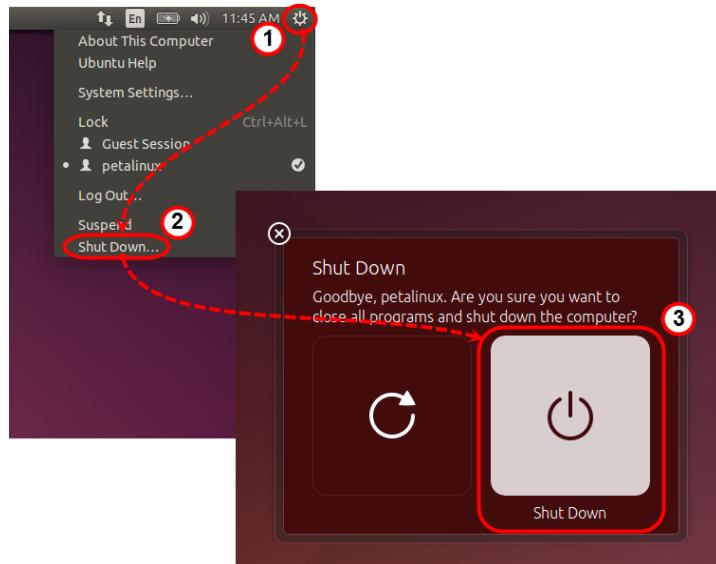


Figure 430: Shutting Down

Launch VirtualBox

1-1. Launch VirtualBox.

- 1-1-1.** Select **Start Menu > All Programs > Oracle VM VirtualBox > Oracle VM VirtualBox** to start the VirtualBox Manager program.

The VirtualBox Manager allows the creation of new virtual machines and facilitates launching and controlling existing virtual machines. The left pane is populated with at least one preconfigured virtual machine. The right pane shows an overview of the details of the selected virtual machine.

- 1-1-2.** Select the **the name of the virtual machine** virtual machine titled from the left pane (1).
- 1-1-3.** Click the green right arrow icon to launch the selected virtual machine (2) -or- double-click the virtual machine (1).

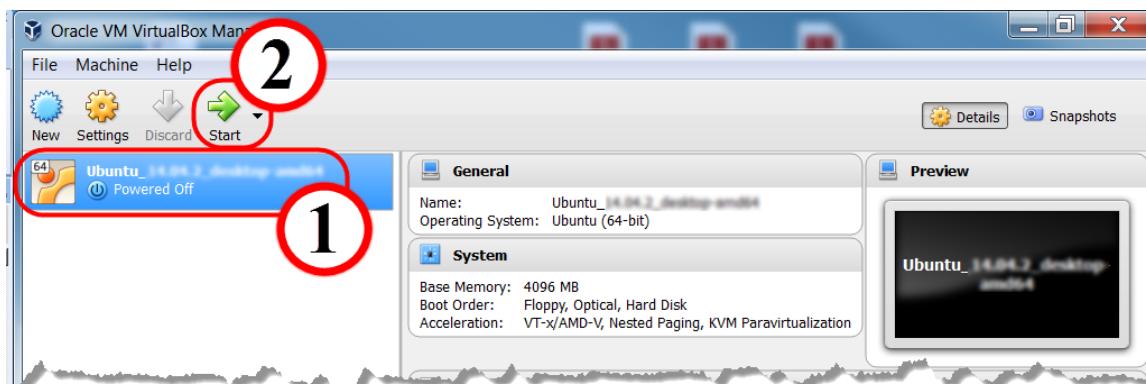


Figure 431: VirtualBox Manager Window

The the name of the virtual machine virtual machine will now launch in a new window. This window can be changed in size, moved, minimized, or maximized just like any other program as required.

- 1-1-4.** If the virtual machine window is not maximized, click the **Maximize** (□) icon in the window title bar to make it full screen.

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 repeat this step and set TCP/IP properties to obtain an IP address automatically.

Most Xilinx software applications use a lab hardware (evaluation demo board) IP address of 192.168.1.10. 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 10 (which is reserved for the Xilinx hardware).

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.** On a Windows 7 machine, select **Start > Control Panel**.

If using a different version of Windows, these instructions are still valid, but selection menus may be different.

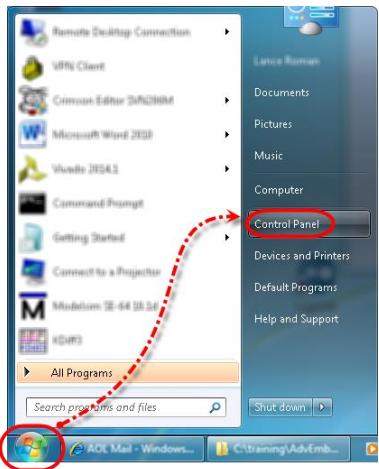


Figure 432: Launching Control Panel

- 1-1-3.** Click **View network status and tasks** under Network and Internet (Windows Default) or Network and Sharing Center (Windows Classic with Details).

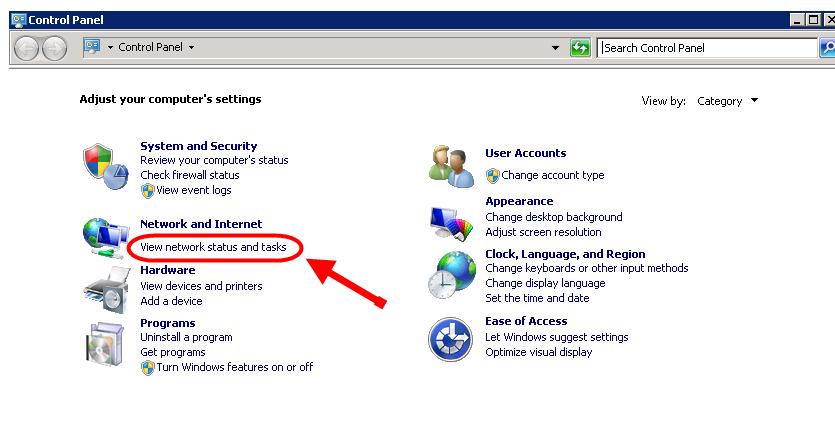


Figure 433: Control Panel (Windows Default)

OR

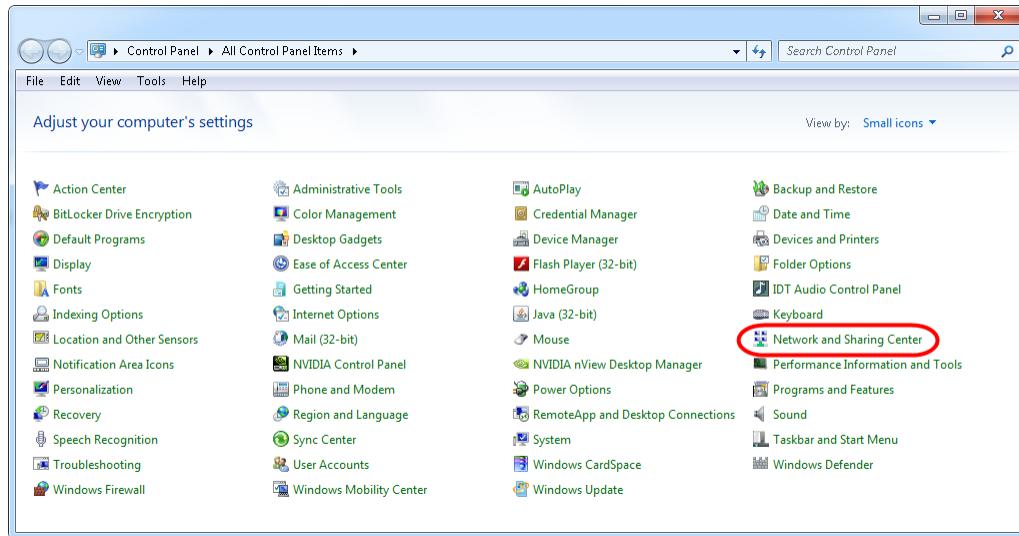


Figure 434: Control Panel (Windows Classic with Details)

- 1-1-4.** In the Network and Sharing Center left pane, select **Change Adapter Settings**.

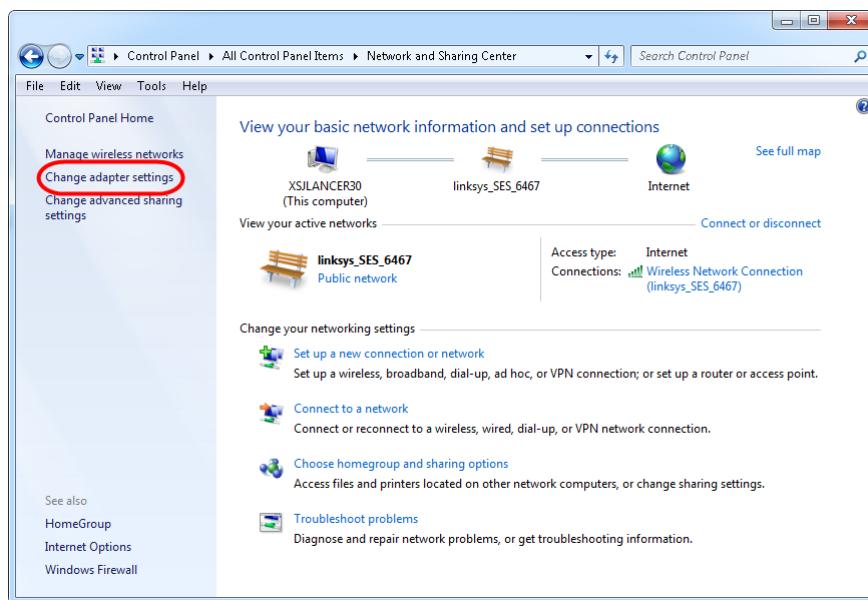


Figure 435: Control Panel Network and Sharing Center

- 1-1-5.** Select the Ethernet adapter to be used.

1-1-6. Right-click and select **Properties**.

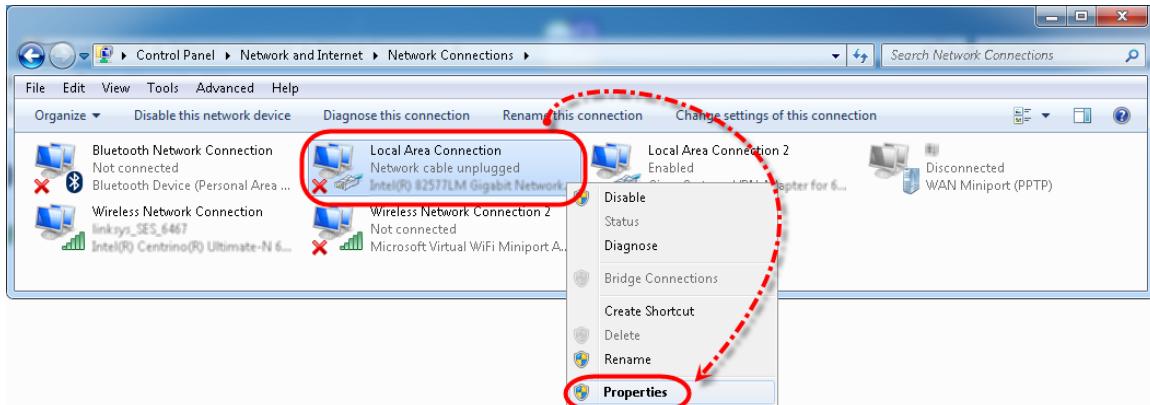


Figure 436: Network Connections

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).

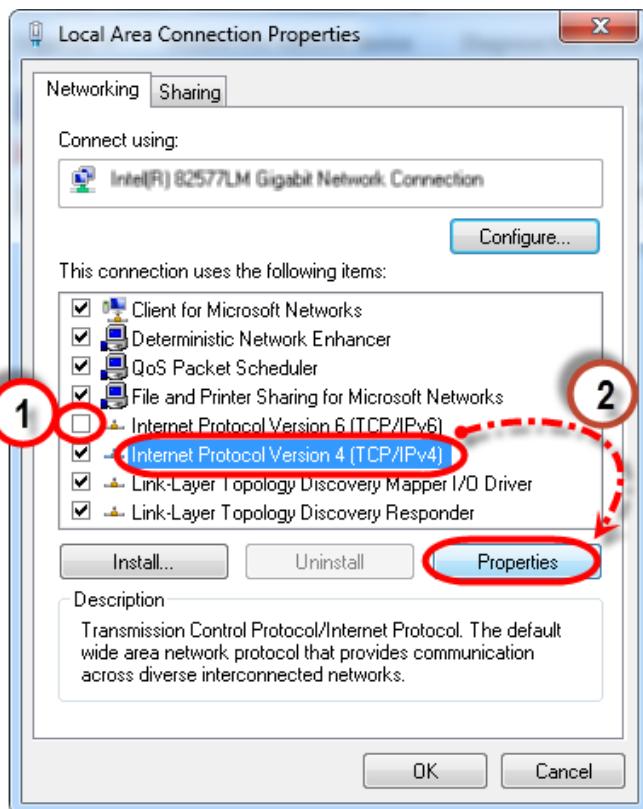


Figure 437: Local Area Connection Properties

1-1-10. Select **Use the following IP address** (1).

1-1-11. Enter **192.168.1.11** 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).

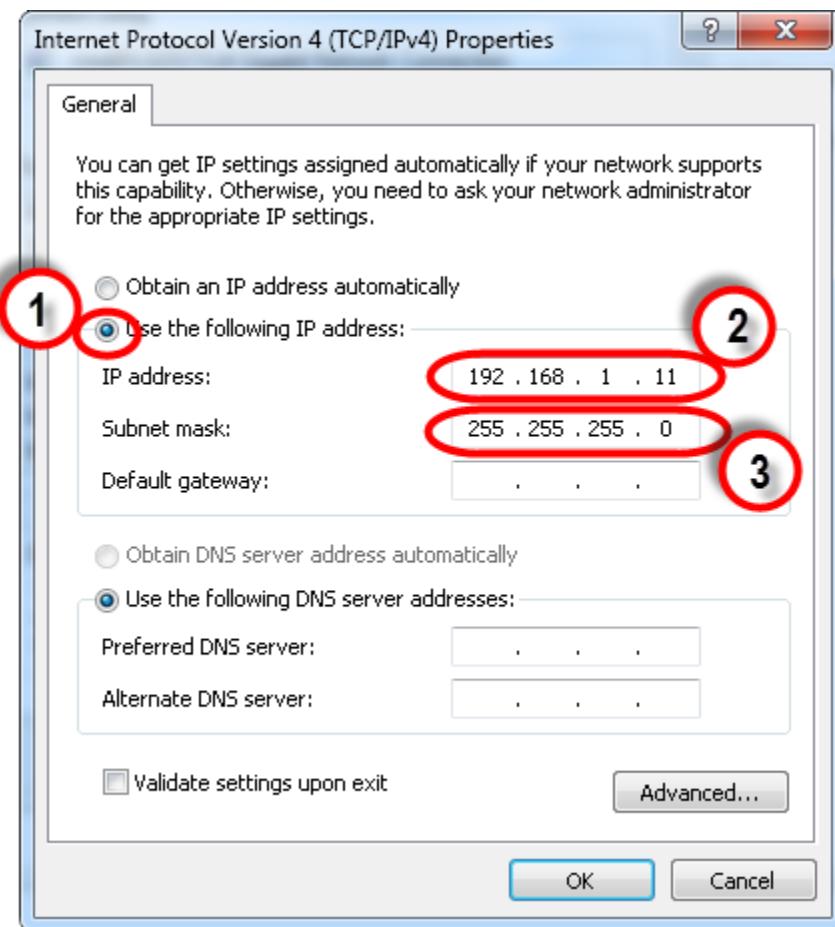


Figure 438: Set TCP-IPv4 Properties for Static Address

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 return 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**.

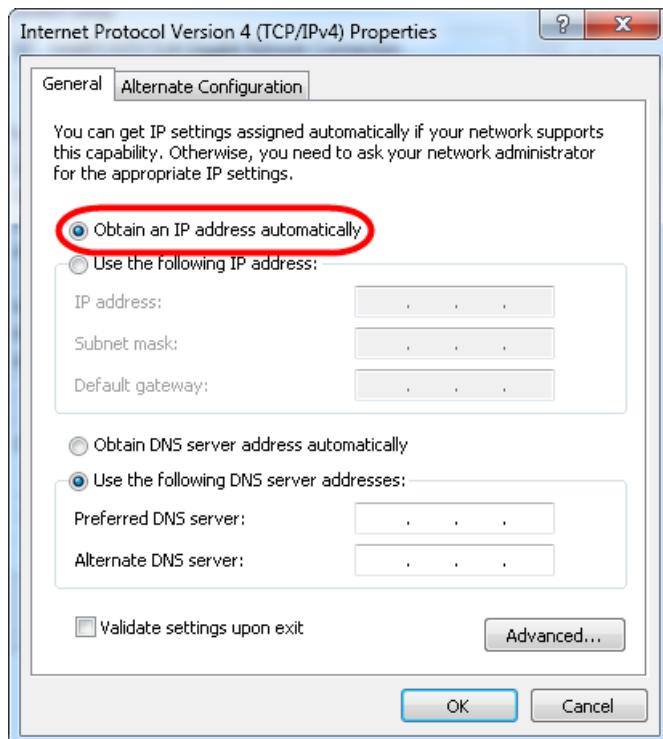


Figure 439: Set TCP-IPv4 Properties for DHCP Obtained Address

- 1-2-4.** Click **OK**.
- 1-2-5.** Click **Close**.
- 1-2-6.** Close the Control Panel.

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 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)
- Note:** Typically the Platform Cable module is not used (shown in the figure for reference).
 - 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

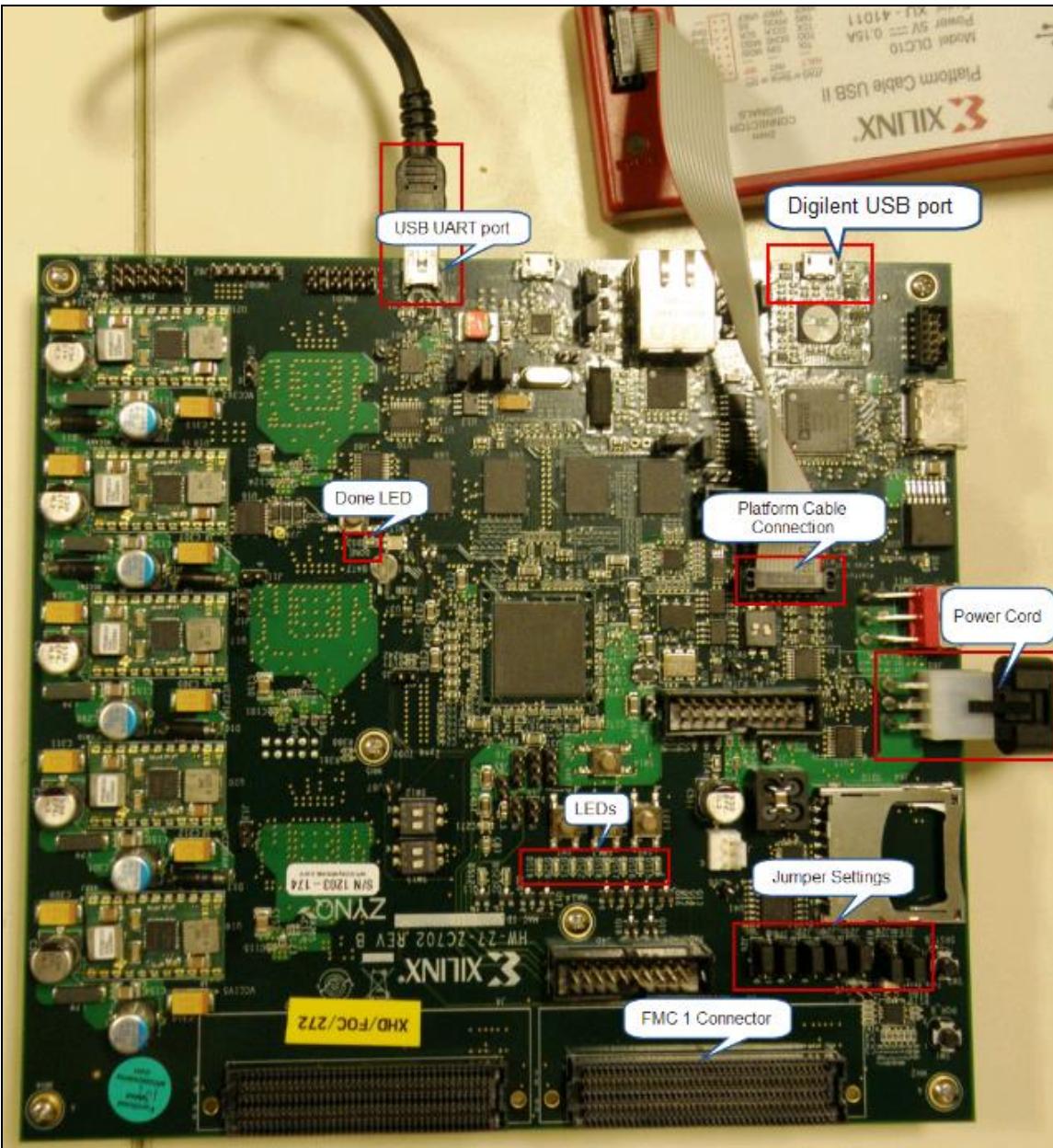


Figure 440: ZC702 Development Board

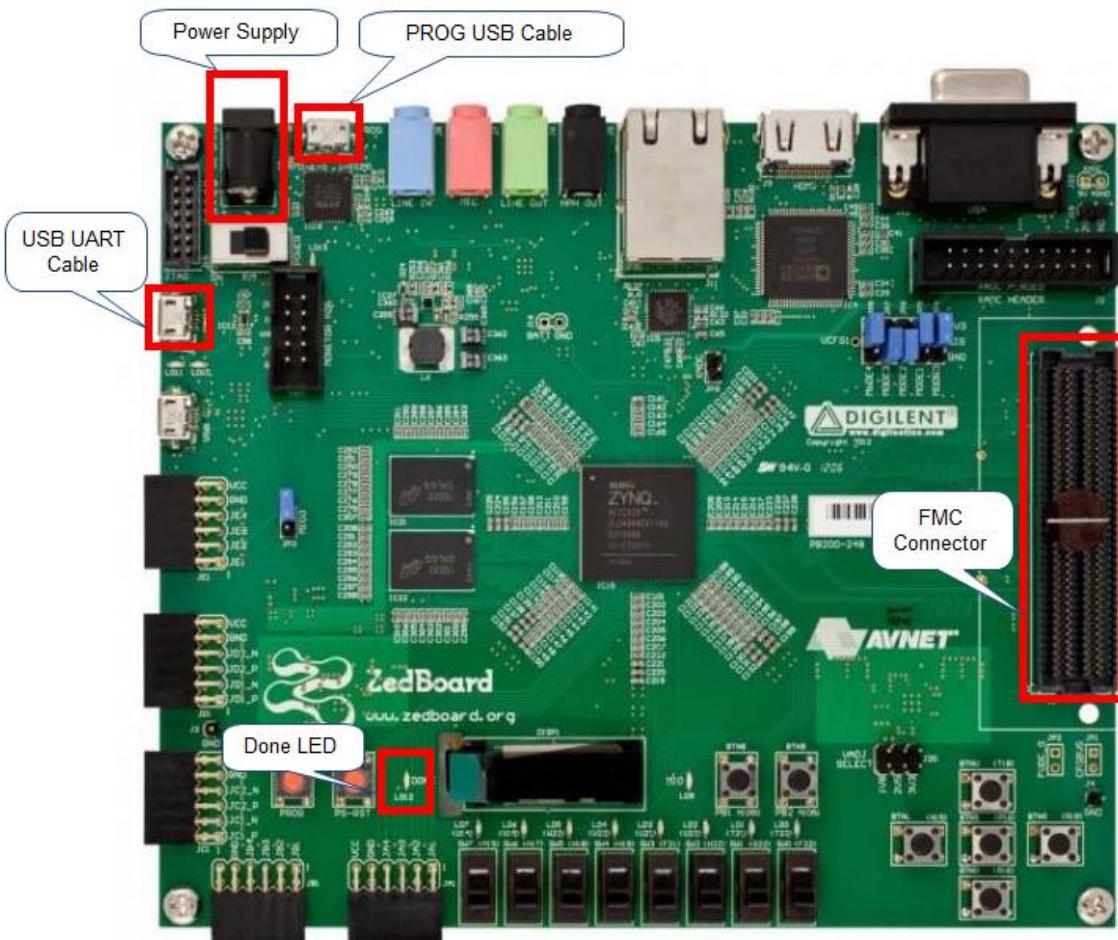


Figure 441: ZedBoard

ZC702 board users: Make sure that the FMC-CE card is plugged into the FMC1 connector.

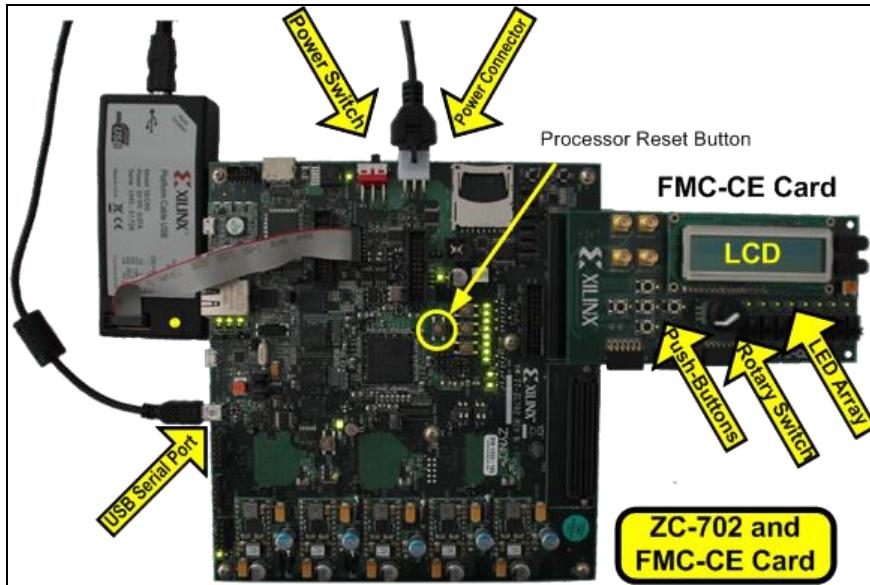


Figure 442: ZC702 and FMC-CE Card

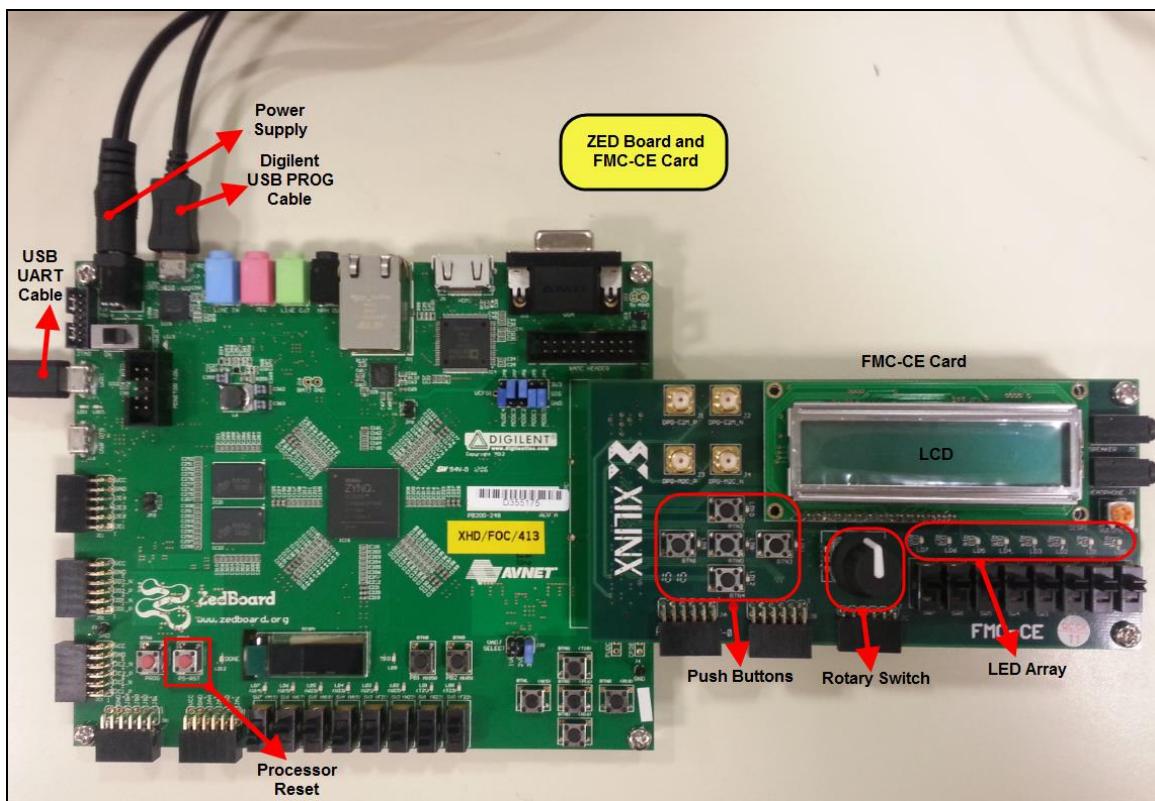


Figure 443: ZedBoard and FMC-CE Card

- 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 your board evaluation hardware platform has jumpers (or switches) to select the boot mode for the processor.

1-1. Verify that the 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.** Slide the power switch to the on position.

Some LEDs on the board will illuminate when the board is powered.

Preparing an SD Card

1-1. Prepare an SD card.

- 1-1-1.** Insert an SD card into the PC's SD card slot.
- 1-1-2.** Using Windows Explorer, browse to the SD card drive.

Optional: You may want to erase and/or reformat the SD card at this time, which may prevent any unwanted interaction with other files that may be on the card.

- 1-1-3.** Open a second Windows Explorer window to browse to the files that you will copy to the SD card.
- 1-1-4.** Navigate to the image located at *the location of the files you want to copy to the SD card*.

1-1-5. Drag-and-drop all the files from the source directory to the SD card.

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 applying 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

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. It is an alternative to using the Terminal view that is built into SDK.

1-1. Launch the Tera Term terminal program.

- 1-1-1. From the Windows desktop, double-click the **Tera Term** icon to launch Tera Term.

Alternatively, you can select **Start > All Programs > Tera Term > Tera Term**.

- 1-1-2. Select **Serial** as the connection (1).

- 1-1-3. 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.

- 1-1-4. Select the COM # discovered in the last step (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.

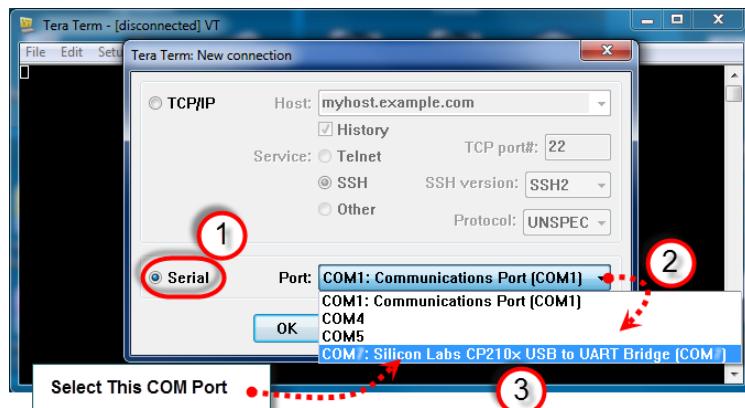


Figure 444: Selecting the COM Port

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.

- 1-1-5. Click **OK**.

The terminal console window opens.

1-1-6. Select **Setup > Serial Port**.

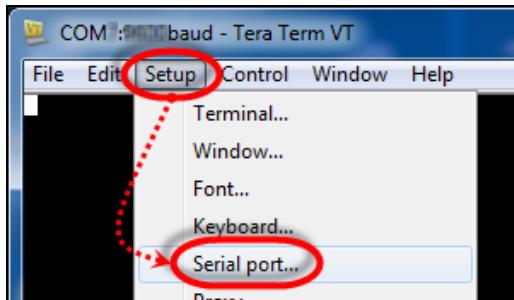


Figure 445: Opening the Tera Term Serial Port Setup Window

The Tera Term Serial Port Setup dialog box opens.

1-1-7. Confirm that the proper serial port has been selected (1).

1-1-8. Set the baud rate to **115200** (2).

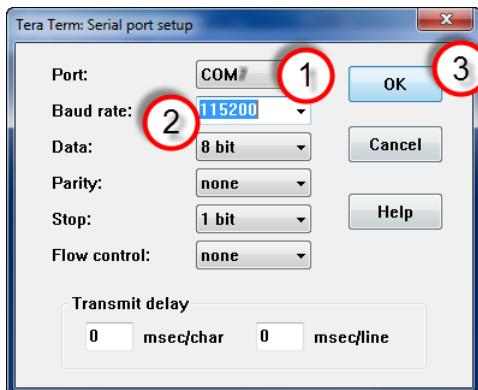


Figure 446: Setting the Parameters for the Serial Port

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.

1-1-9. 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, a number of the following illustrations show the presence of the Customer Education daughter card (FMC-CE).

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
 - Type A male to 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
- Power supply

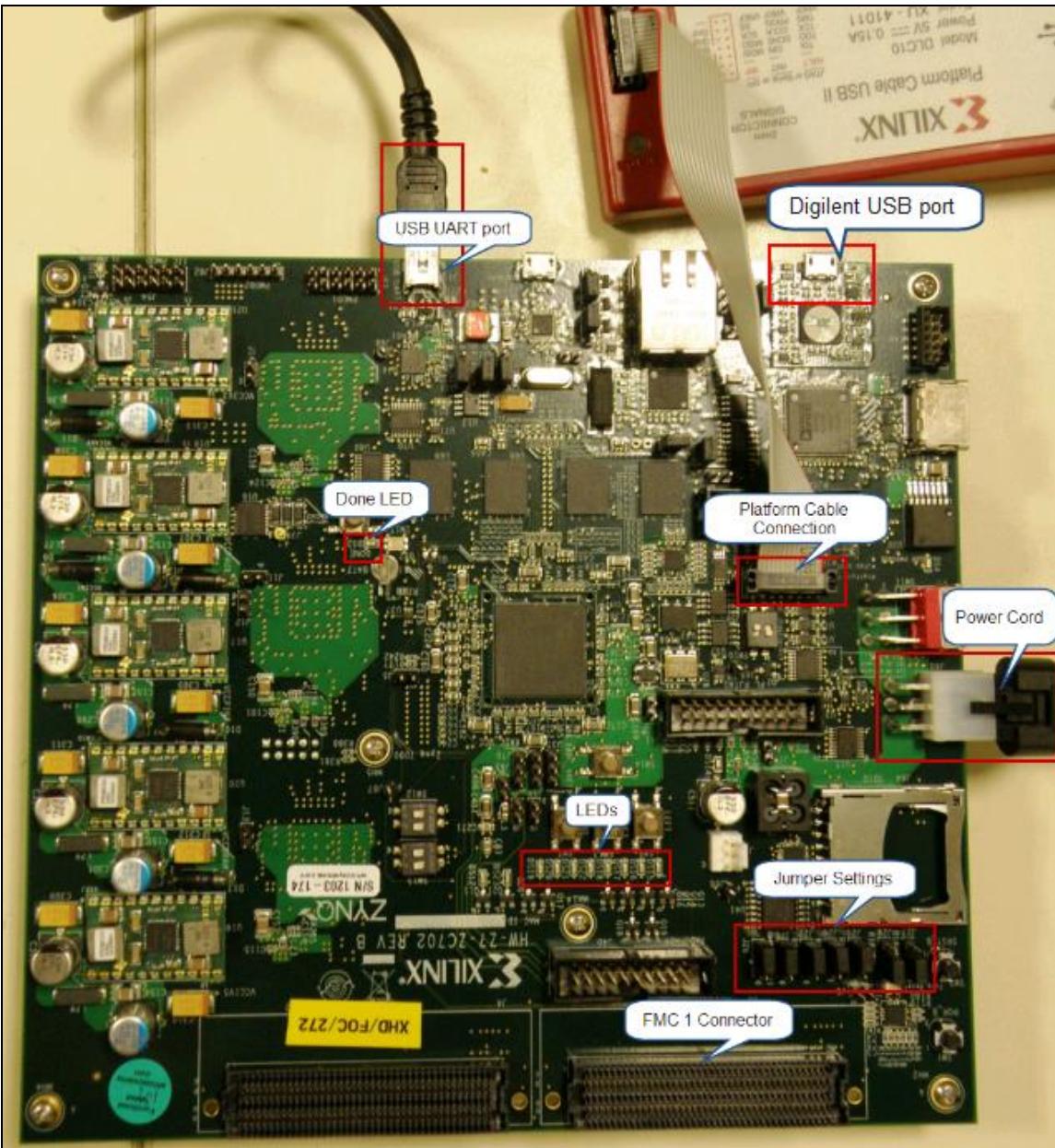


Figure 447: ZC702 Development Board

- 1-1-2.** Make sure that the FMC-CE card is plugged into the FMC1 connector.

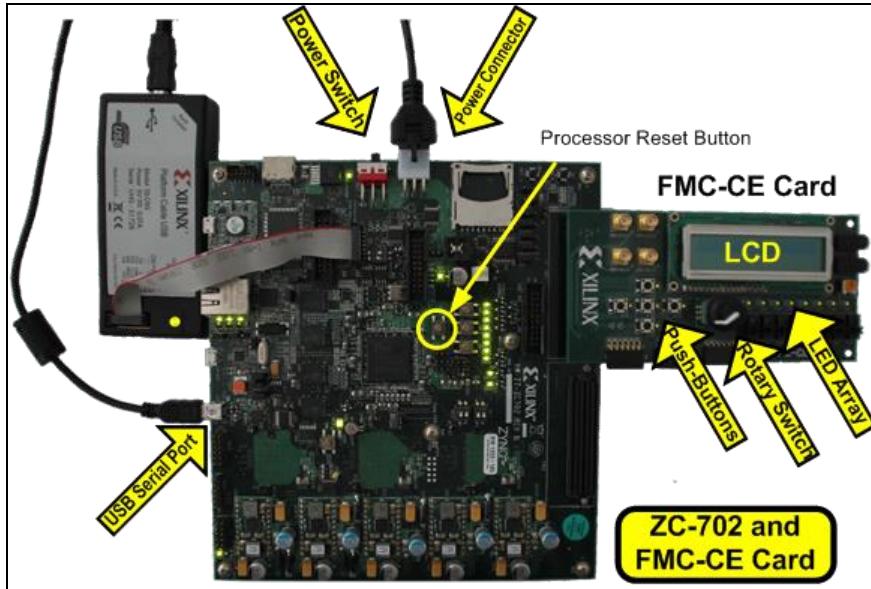


Figure 448: ZC702 and FMC-CE Card

- 1-1-3.** Power up the ZC702 board using the mounted 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. Several of the following figure show the presence of the Customer Education daughter card (FMC-CE).

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

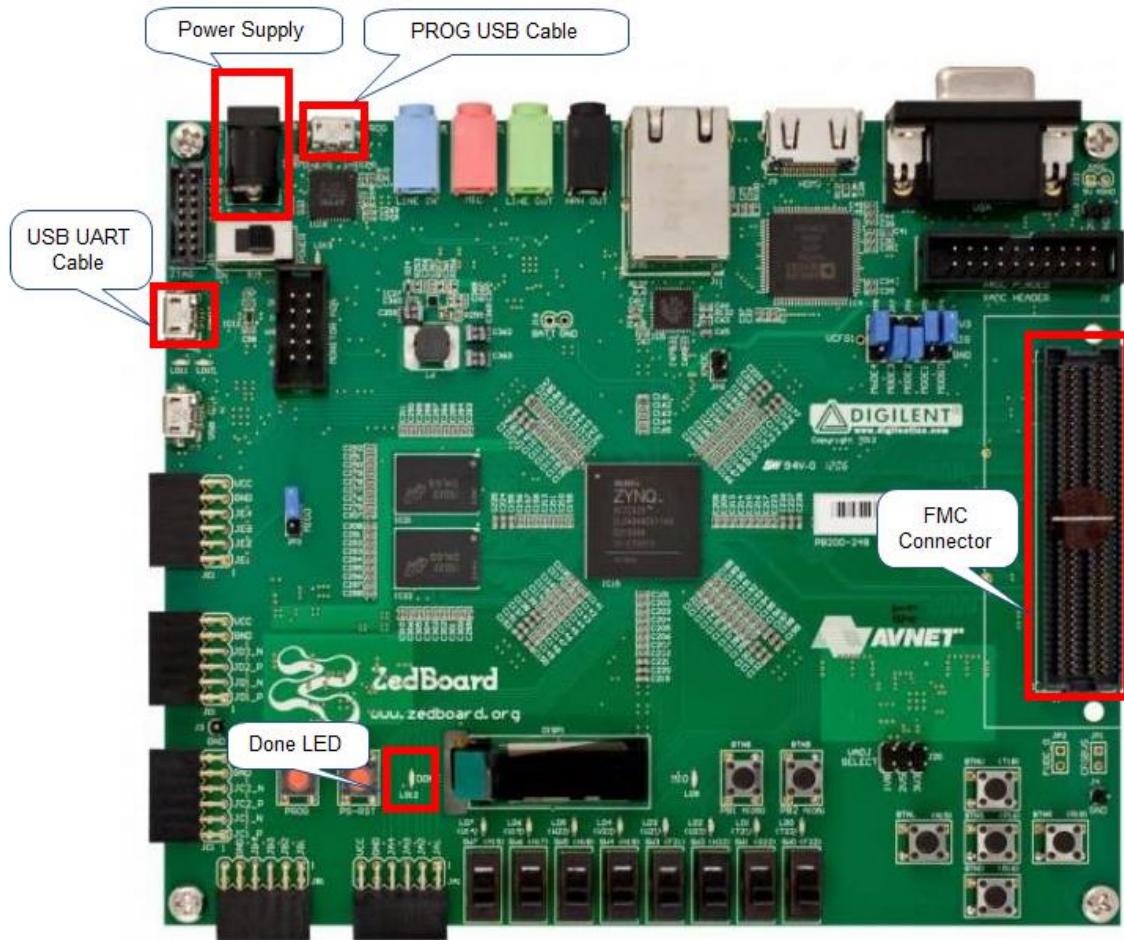


Figure 449: ZedBoard

- 1-1-2.** If an FMC-CE card is needed for the lab, make sure that the FMC-CE card is plugged into the FMC connector (J1).

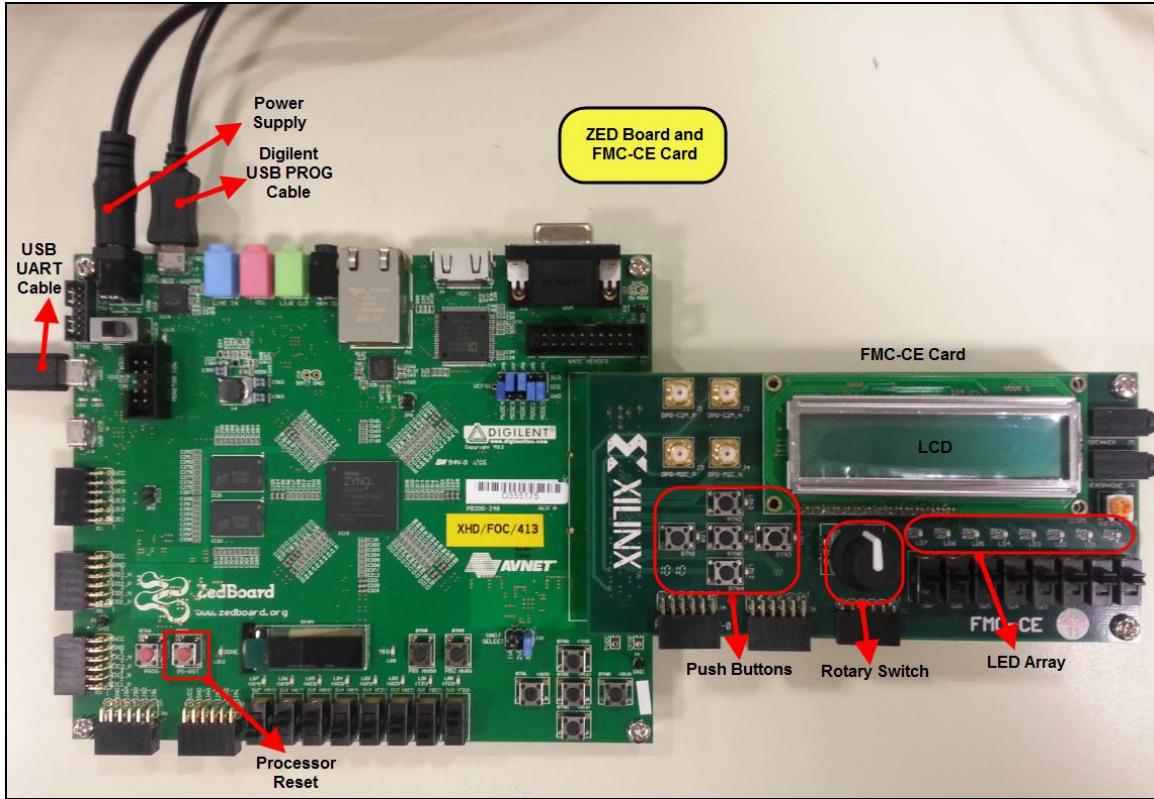


Figure 450: ZedBoard and FMC-CE Card

- 1-1-3.** 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

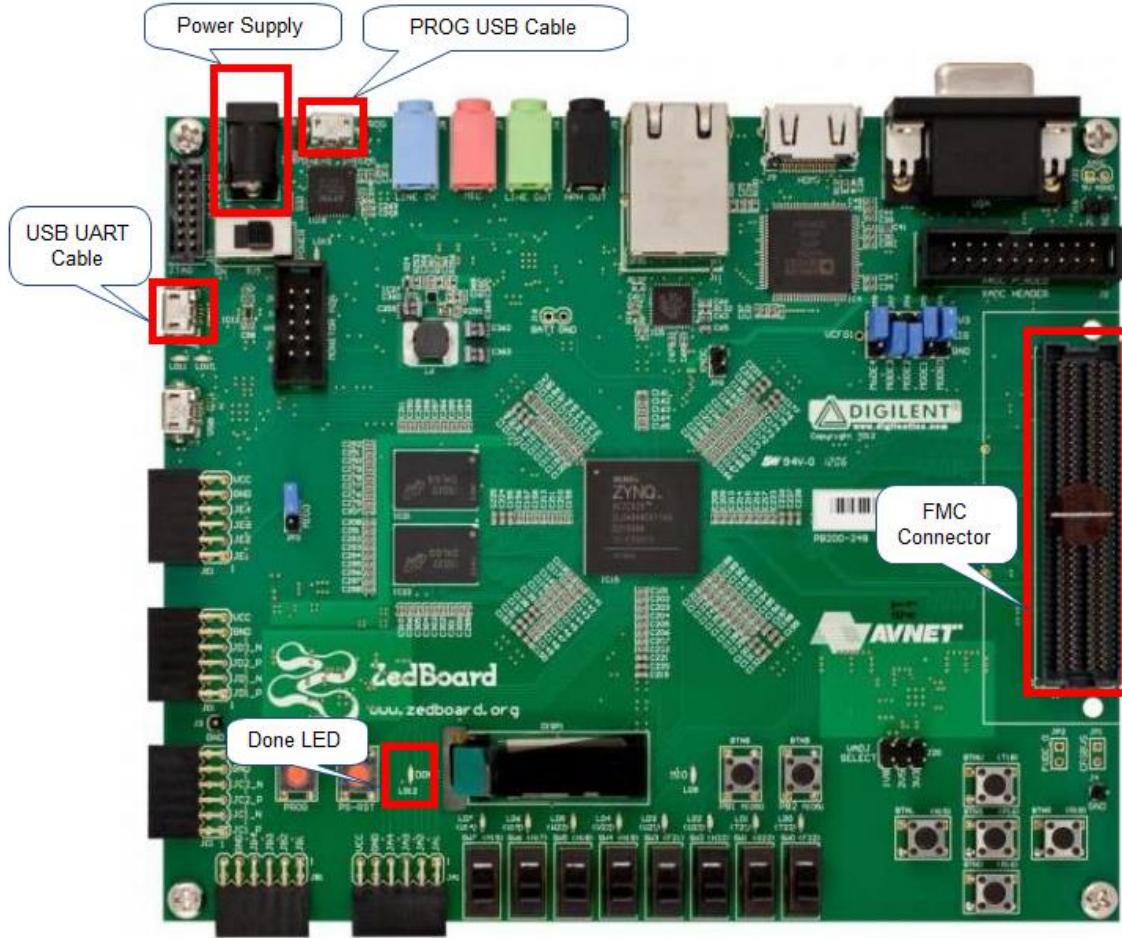


Figure 451: ZedBoard

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 452: SD Card Settings (ZC702 Board)

Jumper/Switch Settings ZC702 - Boot from JTAG

JTAG Mode:



J27 J28 J21 J20 J22 J25 J26

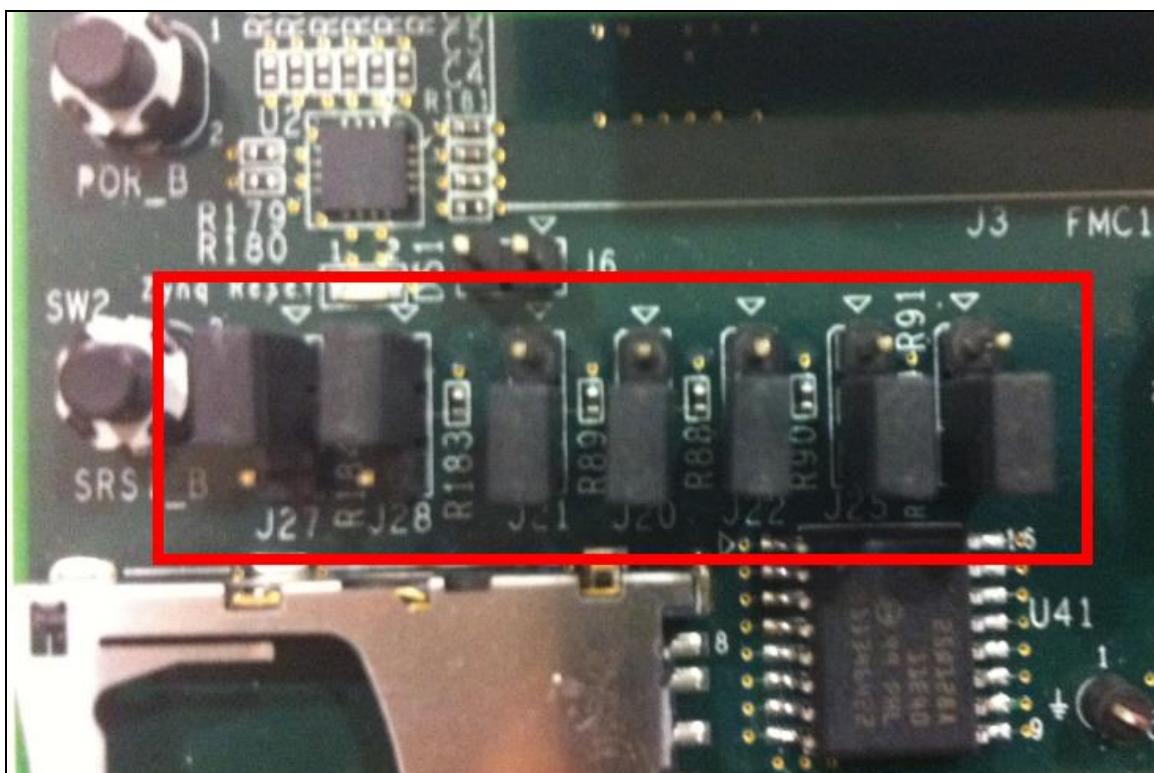


Figure 453: Jumper Settings – JTAG Mode

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.

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:

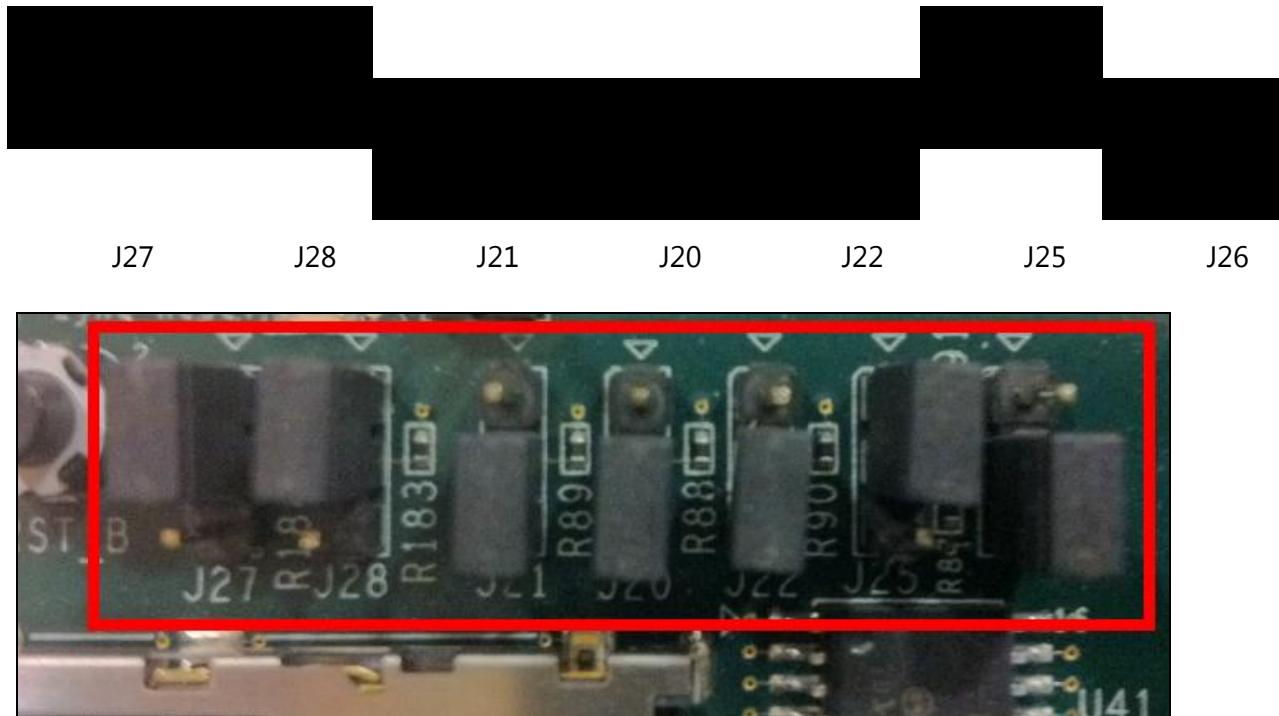


Figure 454: Jumper Settings – Flash Boot Mode

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:

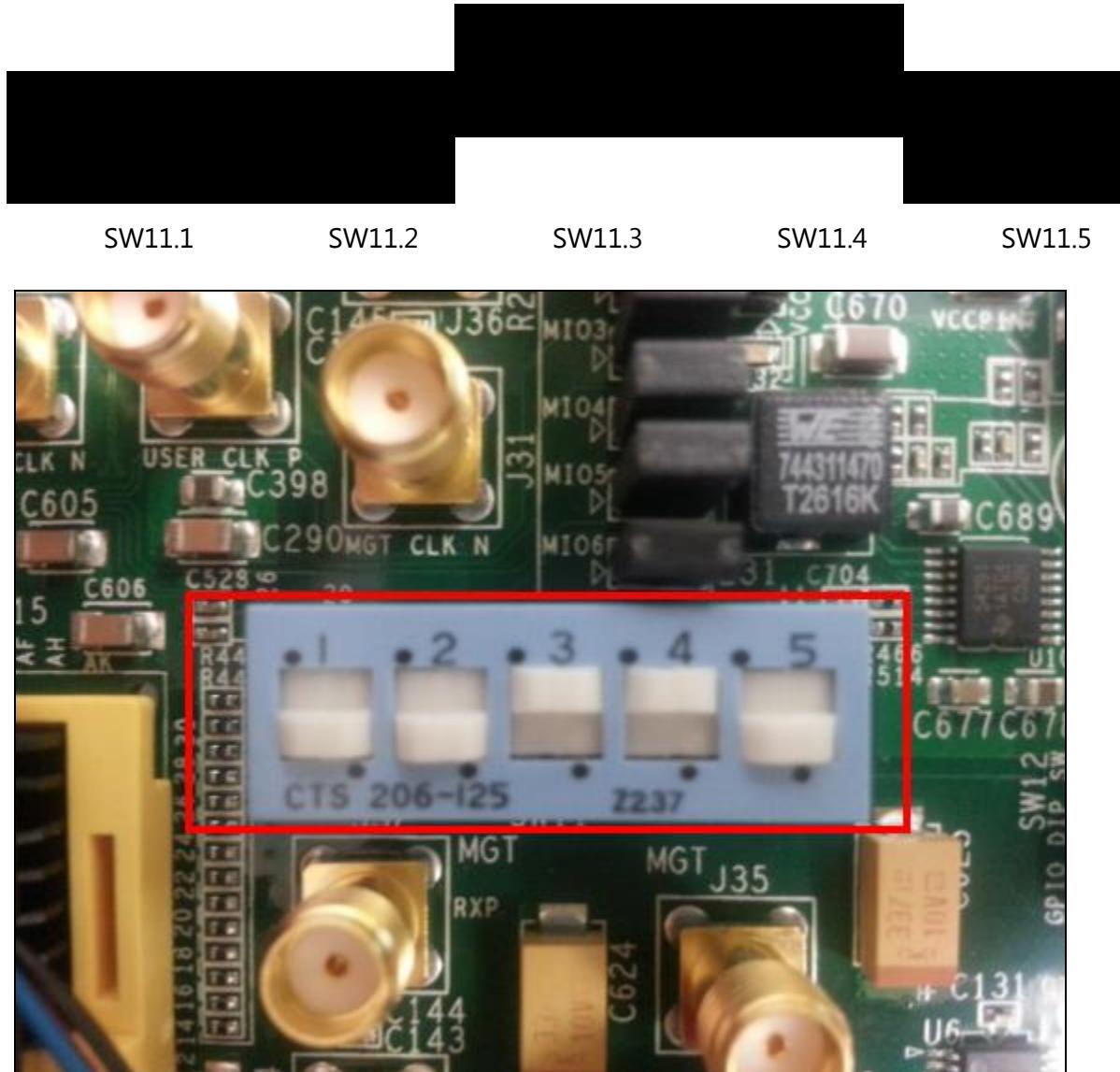
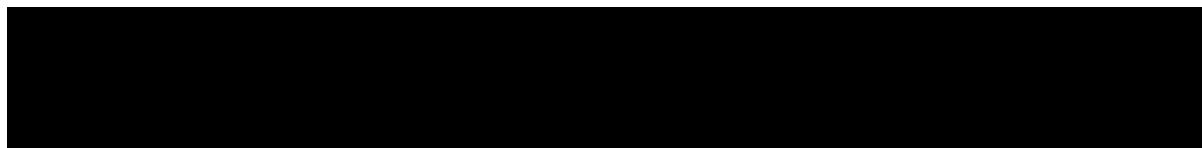


Figure 455: SD Card Settings (ZC706 Board)

Jumper/Switch Settings ZC706 - Boot from JTAG

JTAG Mode:



SW11.1

SW11.2

SW11.3

SW11.4

SW11.5

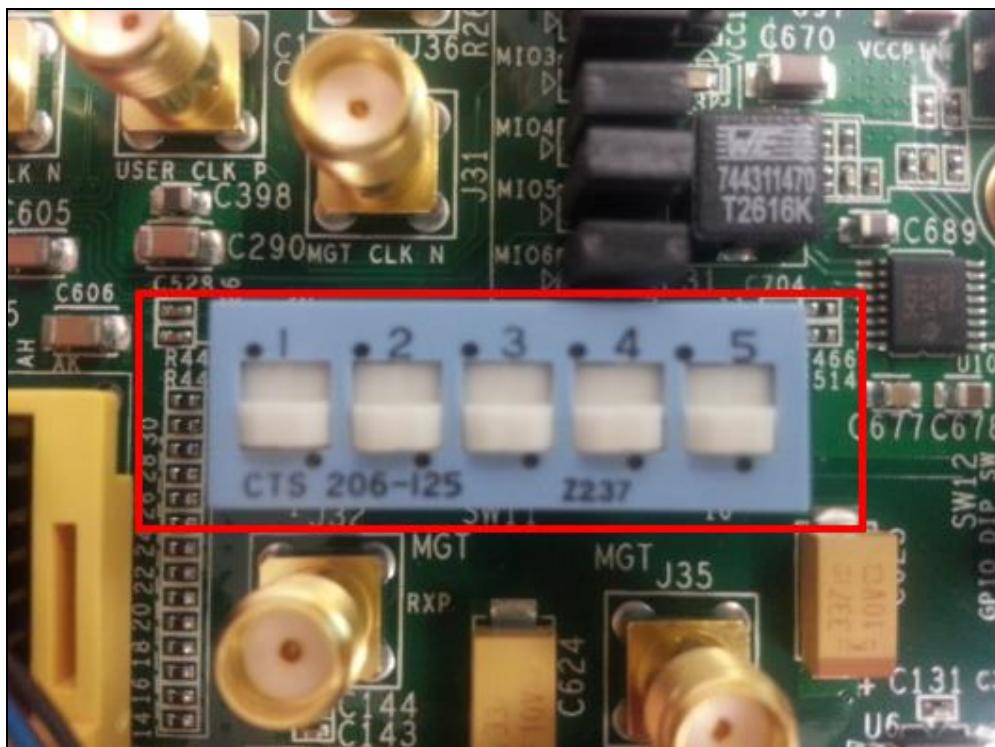


Figure 456: Jumper Settings ZC706 – JTAG Mode

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:

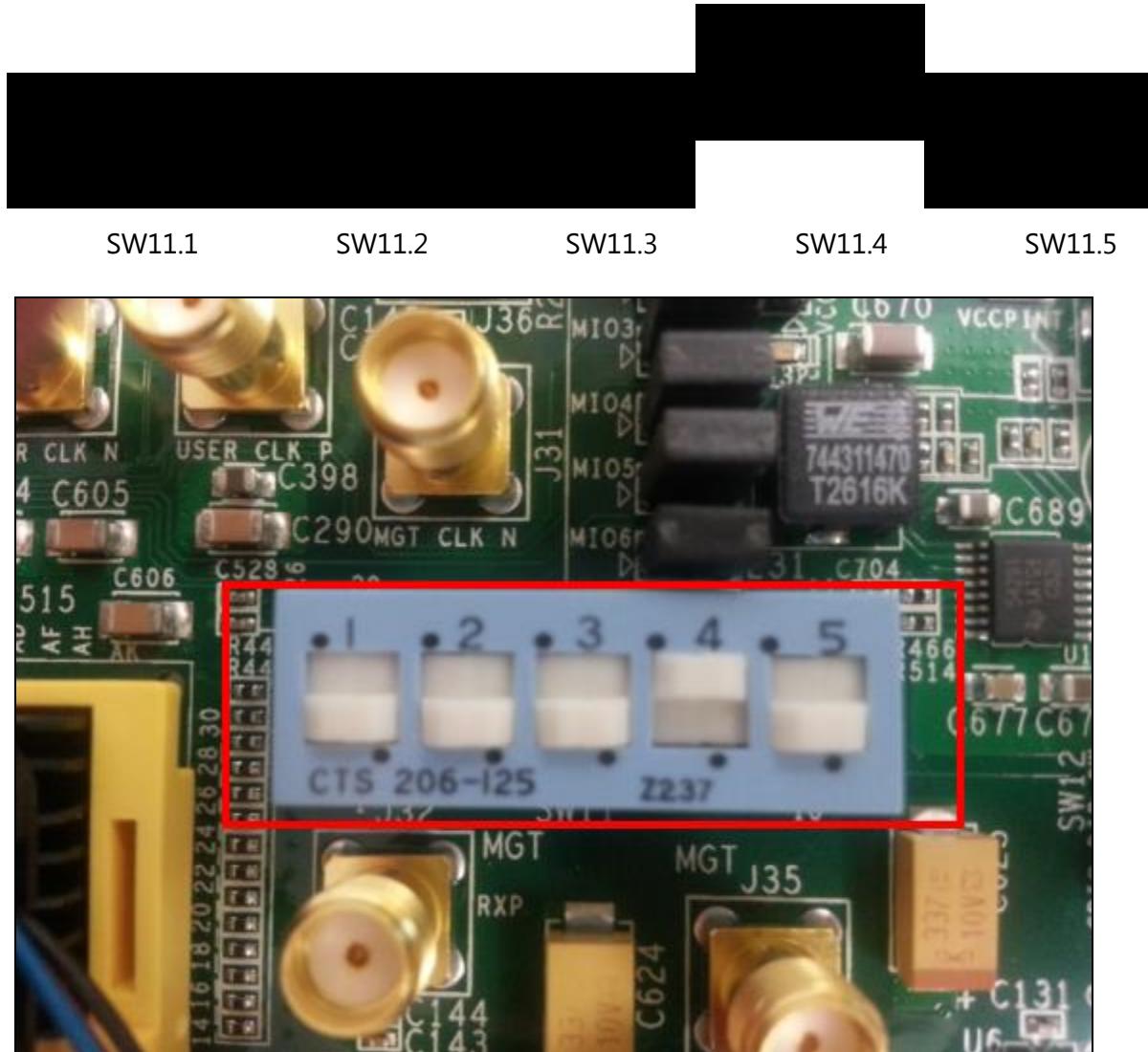


Figure 457: Jumper Settings ZC706 – Flash Boot Mode

Setting the Jumpers on the ZCU102 Board

Jumper/Switch Settings ZCU102 - Boot from SD Card

SD Card Mode:

Up				
Down				
	SW6.1	SW6.2	SW6.3	SW6.4



Figure 458: ZCU102 Switches - Boot SD Card

Jumper-Switch Settings ZCU102 - Boot from JTAG

JTAG Mode:

Up				
Down				
	SW6.1	SW6.2	SW6.3	SW6.4



Figure 459: ZCU102 Switches

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	JP7/JP10	JP6/JP9 -	JP5/JP8 -	JP4/JP7 -
- Mode 4	- Mode 3	Mode 2	Mode 1	Mode 0



Jumper - JP6 (shortened)

VADJ SELECT - 3V3

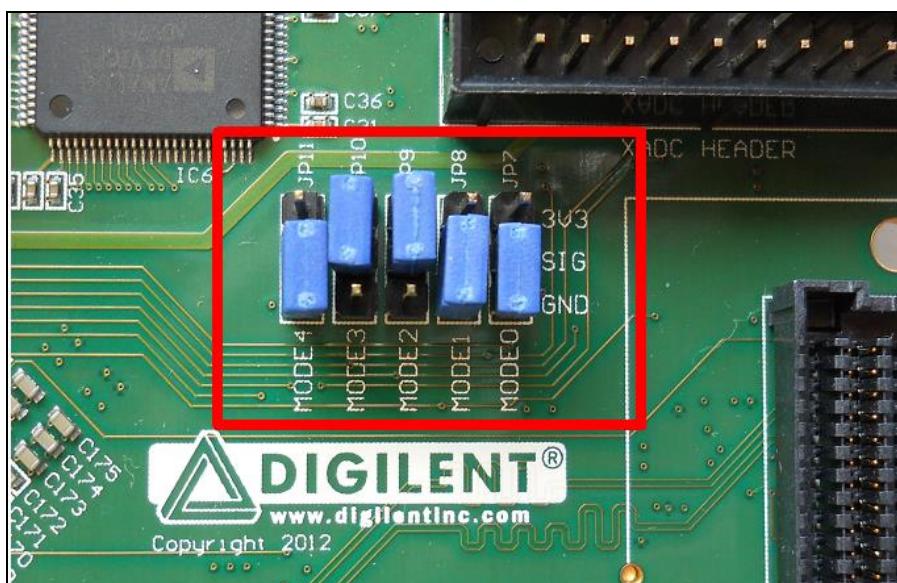


Figure 460: SD Card Settings (ZedBoard)

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.

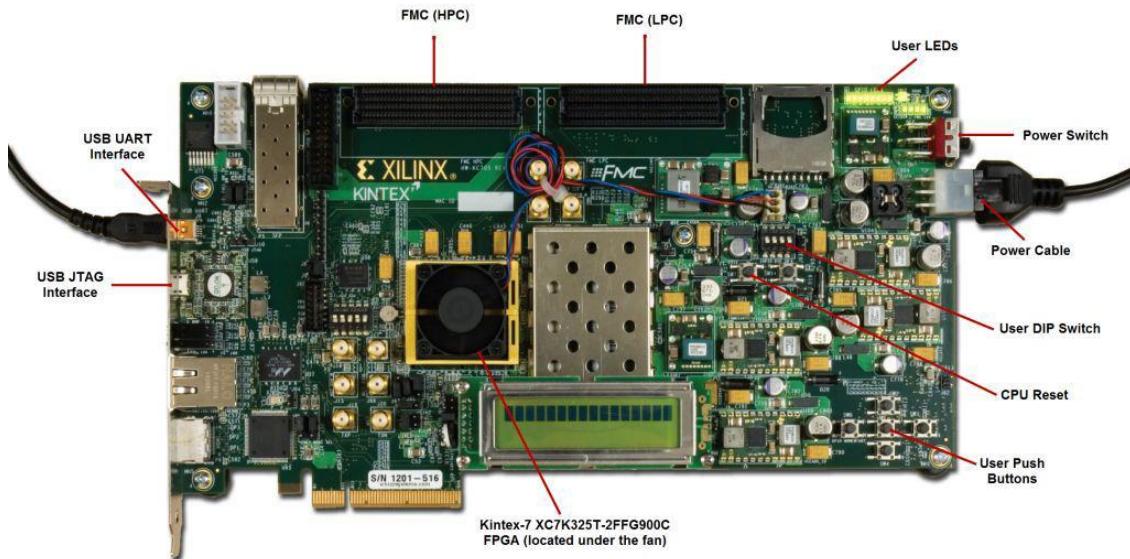


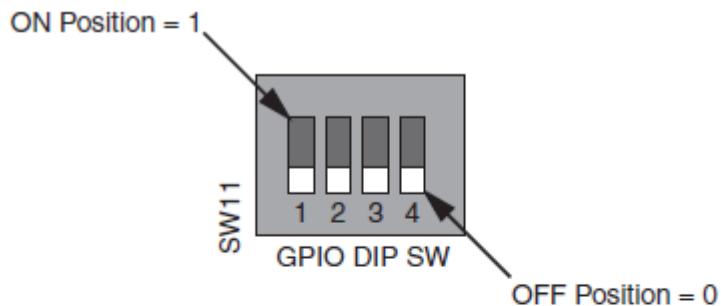
Figure 461: KC705 Evaluation Board

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.



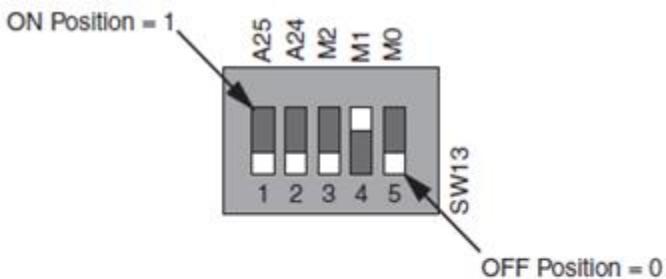
SW11 Default Switch Settings

Position	Function	Default
1	GPIO_DIP_SW3	Off
2	GPIO_DIP_SW2	Off
3	GPIO_DIP_SW1	Off
4	GPIO_DIP_SW0	Off

Figure 462: 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 463: 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.

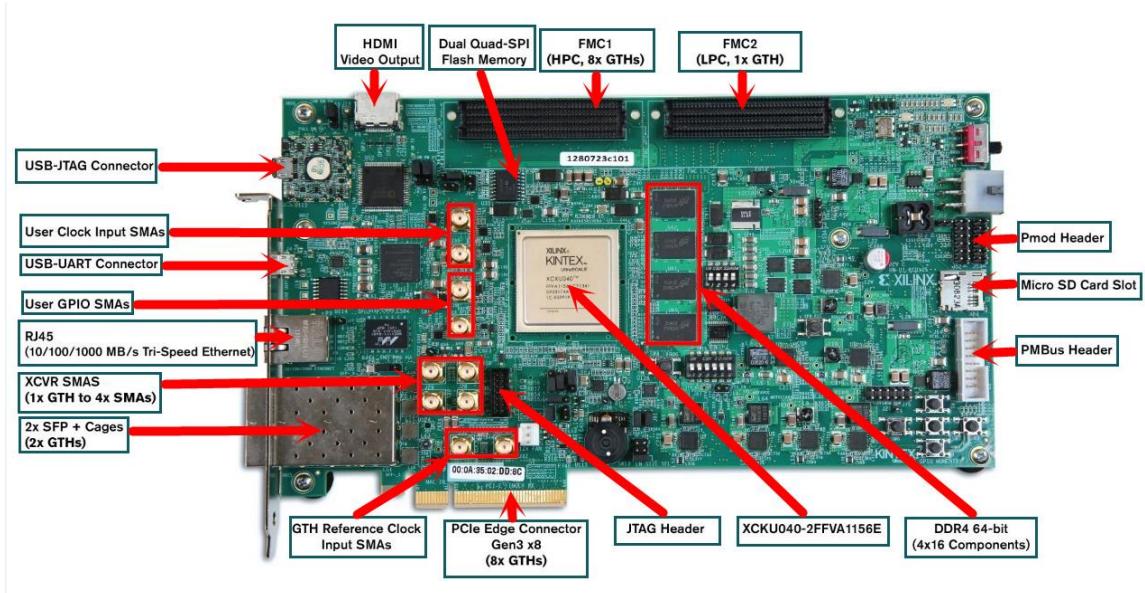


Figure 464: KCU105 Evaluation Board