

**AMD**  
together we advance\_



## **AMD ADAPTIVE COMPUTING CUSTOMER TRAINING**

Lab Reference Guide

[lab-reference-guide-2023.2-wkb-rev1-prelim](#)



# Lab Reference Guide 2023.2



© 2023 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, MicroBlaze, Kintex, Versal, Virtex, Vitis, Vivado, UltraFast, UltraScale, UltraScale+, Zynq, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. Linux is the registered trademark of Linus Torvalds in the U.S. and other countries. PCI Express and PCIe are registered trademarks of PCI-SIG Corporation. Python is a trademark of the Python Software Foundation. All other trademarks are the property of their respective owners.

## DISCLAIMER

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18



## Table of Contents

---

|  |            |
|--|------------|
| <b>Lab Reference Guide .....</b>   | <b>3</b>   |
| <b>Preliminary Version.....</b>  | <b>3</b>   |
| <b>Introduction.....</b>   | <b>3</b>   |
| <b>Understanding the Lab Environment.....</b>                                | <b>3</b>   |
| <b>Vivado Design Suite Operations [Last Updated Version 2023.2] .....</b>    | <b>4</b>   |
| <b>Vivado Analyzer Operations [Last Updated Version 2017.1].....</b>         | <b>144</b> |
| <b>Vitis HLS Tool Operations [Last Updated Version 2022.1] .....</b>         | <b>157</b> |
| <b>Vitis Model Composer Operations [Last Updated Version 2022.1] .....</b>   | <b>172</b> |
| <b>Vitis IDE Operations [Last Updated 2023.2] .....</b>                      | <b>173</b> |
| <b>QEMU Operations [Last Updated Version 2019.1].....</b>                    | <b>243</b> |
| <b>PetaLinux Operations [Last Updated Version 2021.1] .....</b>              | <b>251</b> |
| <b>Board, OS, COM, and IP Address Tasks [Last Updated Version 2023.2] ..</b> | <b>262</b> |



# 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 2023.2 version, go to <https://www.amd.com/en/training/customer/adaptive-computing/downloads.html>.

## Introduction

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

The guide is divided into the following sections:

- Vivado Design Suite Operations
- Vivado Analyzer Operations
- Vitis HLS Tool Operations
- Vitis Model Composer Operations
- Vitis IDE Operations
- QEMU Operations
- PetaLinux Operations
- Board, OS, COM, and IP Address Tasks

Each section may contain sub-sections.

## Understanding the Lab Environment

### Understanding the Lab Environment

The labs and demos provided in this course are designed to run on a Linux® platform.

One environment variable is required: `TRAINING_PATH`, which points to the location of the lab files. This variable comes configured in the CloudShare/CustEd\_VM environments.

Some tools can use this environment variable directly (that is, `$TRAINING_PATH` is recognized and automatically expanded), and some tools require manual expansion (`/home/amd/training` for the CloudShare/CustEd\_VM environments). The lab instructions describe what to do for each tool. Other environments require the definition of this variable for the scripts to work properly.

The Vivado™ Design Suite and the Vitis™ Unified IDE offer a Tcl environment used in many labs. When either tool is launched, it starts with a clean Tcl environment with none of the procs or variables remaining from any previous launch of the tools.

If you sourced a Tcl script or manually set any Tcl variables and you closed the tool, when you reopen the tool, you will need to re-source the Tcl script and set any variables that the lab requires. This is also true of terminal windows—any variable settings will be cleared when a new terminal opens.

## Nomenclature

Formal nomenclature is used to explain how different arguments are used. The following are some of the more commonly used symbols:

| Symbol | Description                    | Example               | Explanation  |
|--------|--------------------------------|-----------------------|--|
| <text> | Indicates a field              | cd <dir>              | <dir> represents the name of the directory. The < and > symbols are NOT entered. If the directory to change to is XYZ, then you would enter <code>cd xyz</code> into the environment.  |
| [text] | Indicates an optional argument | ls [   more]          | This could be interpreted as <code>ls &lt;Enter&gt;</code> or <code>ls   more &lt;Enter&gt;</code> . The first instance lists the files in the current Linux directory, and the second lists the files in the current Linux directory, but additionally runs the output through the <code>more</code> tool, which paginates the output. Here, the pipe symbol ( ) is a Linux operator. |
|        | Indicates choices              | cmd <ZCU104   VCK190> | The <code>cmd</code> command takes a single argument, which could be <code>ZCU104</code> OR <code>VCK190</code> . You would enter either <code>cmd ZCU104</code> or <code>cmd VCK190</code> .  |

## Vivado Design Suite Operations [Last Updated Version 2023.2]

### In This Section

|  |     |
|--|-----|
| Creating and Opening Operations .....        | 5   |
| Vivado Add Sources Operations .....          | 25  |
| Embedded Operations .....                    | 33  |
| Vivado IP Integrator Operations.....         | 37  |
| Vivado Elaborated Design Operations .....    | 92  |
| Vivado Simulation Operations.....            | 94  |
| Vivado Synthesis Operations .....            | 97  |
| Vivado Implementation Operations .....       | 105 |
| Vivado Tcl Operations.....                   | 115 |
| Vivado Timing Miscellaneous Operations ..... | 123 |
| Vivado Hardware Session Operations .....     | 129 |

## Creating and Opening Operations

### In This Section

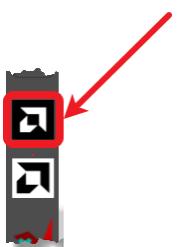
|  |    |
|--|----|
| Launching the Vivado Design Suite .....                        | 5  |
| Launching the Vivado Design Suite Using a Linux Terminal ..... | 6  |
| Launching the Vivado Design Suite Tcl Shell .....              | 7  |
| Creating a New Vivado Design Suite Project with Sources .....  | 7  |
| Creating a Blank Vivado Design Suite Project.....              | 14 |
| Opening a Vivado Design Suite Project.....                     | 17 |
| Opening Source Code in the Editor .....                        | 18 |
| Creating an HDL Source File .....                              | 18 |
| Opening a File in the Editor .....                             | 21 |
| Importing IP.....  | 22 |
| Closing the Vivado Design Suite Project .....                  | 24 |
| Closing the Vivado Design Suite.....                           | 24 |

## Launching the Vivado Design Suite

Here are two ways to open the Vivado Design Suite.

### 1-1. Open the Vivado Design Suite.

- 1-1-1. Click the **Vivado** icon () from the taskbar.



**Figure 1: Launching the Vivado Design Suite from the Taskbar**

**Note:** It takes a few moments to launch. The order of the icons in your environment may be different.

Alternatively, open the Linux terminal window (<Ctrl + Alt + T>) and enter the following:

```
source /opt/amd/Vivado/2023.2/settings64.sh; vivado
```

**Note:** This installation path is valid for the CustEd VM and CloudShare environments. Use the proper path for your environment.

The tool opens with a Welcome window. From here you can create a new project, open an existing project, enter Tcl commands, and access documentation and examples.

- 1-1-2. [Optional] Maximize the window as there is a lot of information to see.

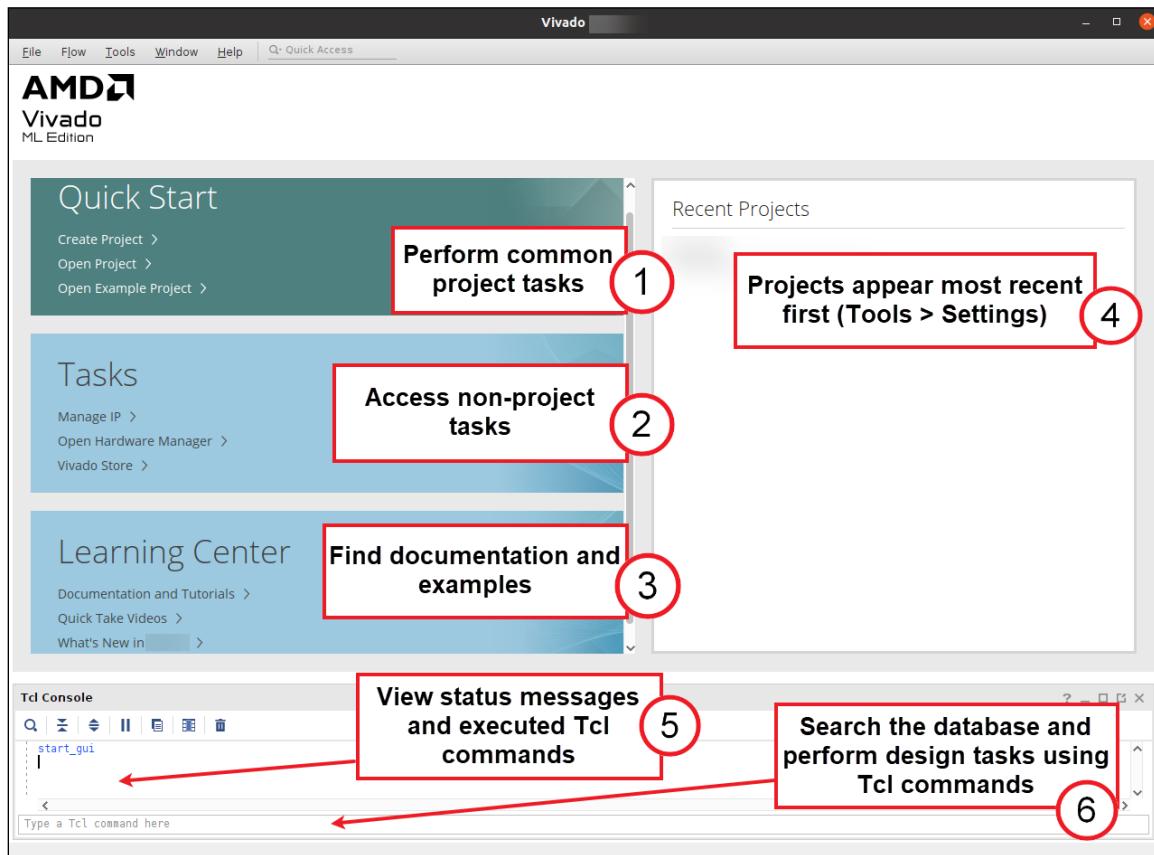


Figure 2: Vivado Design Suite Welcome Screen

**Hint:** If the Tcl Console is not visible, double-click the **Tcl** tab to make it visible.

## Launching the Vivado Design Suite Using a Linux Terminal

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

- 1-1-1. Press **<Ctrl + Alt + T>** to open a new terminal window.  
1-1-2. Enter the following command to set up the Vivado Design Suite environment appropriately:

```
source /opt/amd/Vivado/2023.2/settings64.sh; vivado
```

**Note:** This installation path is valid for the CustEd VM and CloudShare environments. Use the proper path for your environment.

## Launching the Vivado Design Suite Tcl Shell

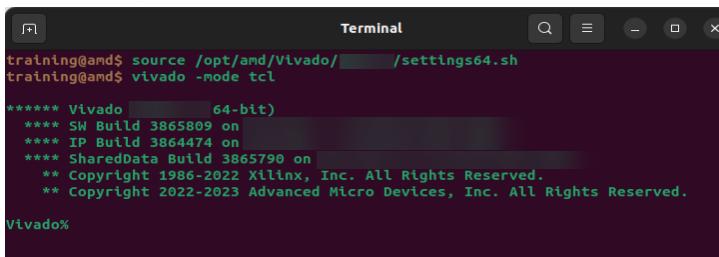
### 1-1. Launch the Vivado Design Suite Tcl shell.

- 1-1-1. Open a Linux terminal window (press **<Ctrl + Alt + T>**) and enter the following commands:

```
[host]$ source /opt/amd/Vivado/2022.2/settings64.sh
[host]$ vivado -mode tcl
```

**Note:** The installation path (`/opt/amd`) is valid for the CustEd VM and CloudShare environments. Modify the path as necessary for your environment.

This opens the Tcl interactive shell in the terminal itself.



```
Terminal
[host]$ source /opt/amd/Vivado/2022.2/settings64.sh
[host]$ vivado -mode tcl

***** Vivado 2022.2 64-bit
***** SW Build 3865809 on
***** IP Build 3864474 on
***** SharedData Build 3865790 on
** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
** Copyright 2022-2023 Advanced Micro Devices, Inc. All Rights Reserved.

Vivado%
```

Figure 3: Vivado Tcl Interactive 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).

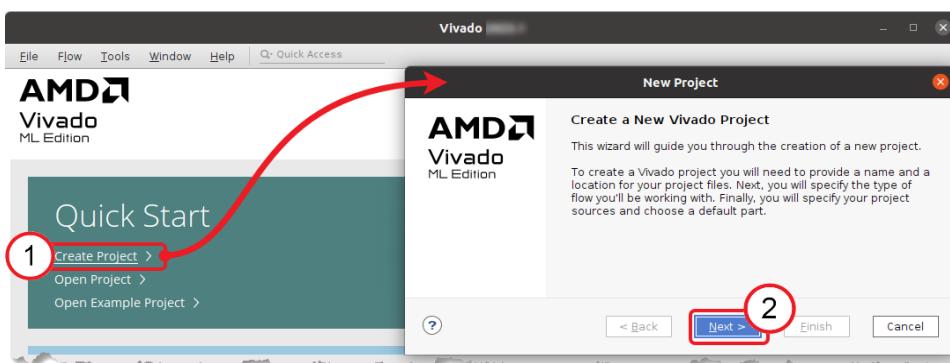


Figure 4: Creating a New Vivado Design Suite Project

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

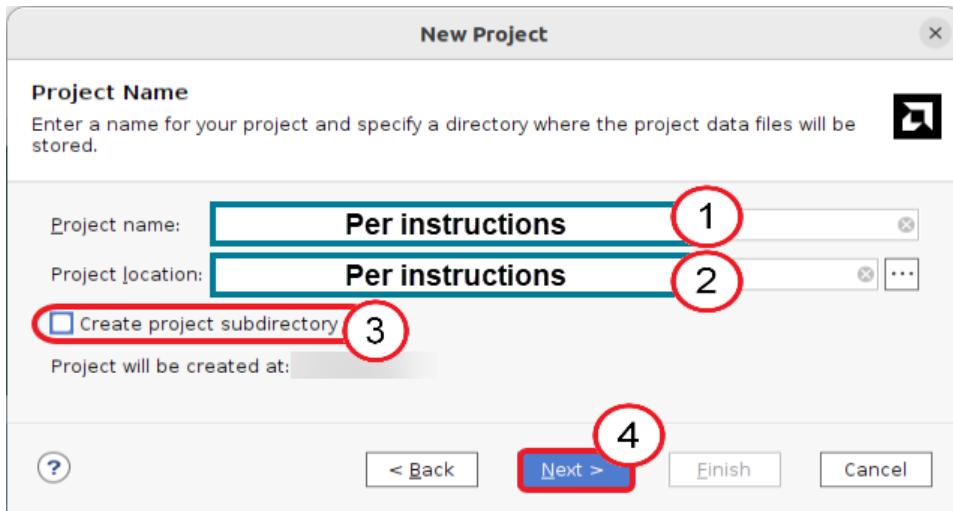
project

**Important:** The Vivado Design Suite is capable of expanding variables when running under both the Linux and Windows environments. The available environment variables (regardless of the OS) must be predicated with the '\$' symbol.

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

- 1-2-3.** Uncheck the **Create Project Subdirectory** option if it is selected (3).

Leaving this checked will create an unnecessary level of hierarchy for the lab.



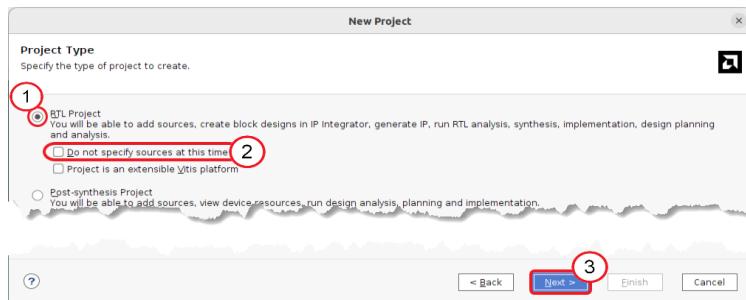
**Figure 5: Entering the Project Name and Location**

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

Here you will specify your project type as either an RTL project or a post-synthesis project. 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** since this project will be based on source code (rather than a netlist) (1).

**1-2-6.** Since you will be adding RTL files in the next instruction, deselect the **Do not specify sources at this time** option (2).



**Figure 6: Setting the Project Type to RTL**

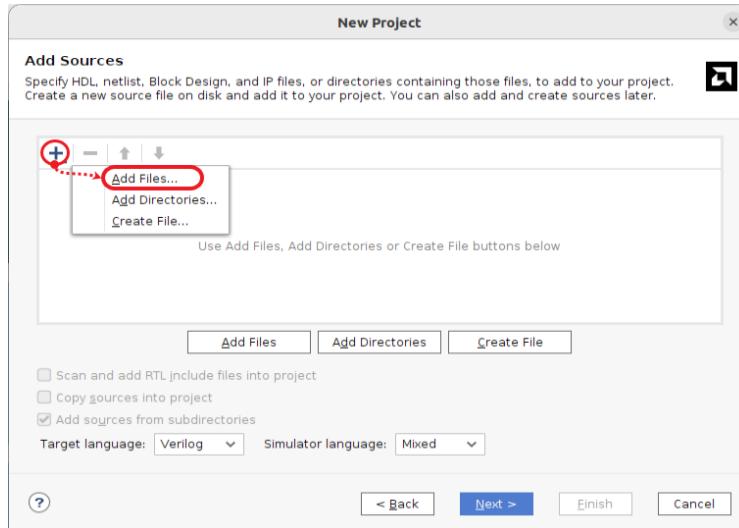
**1-2-7.** Click **Next** to accept the selection and advance to the adding sources stage (3).

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

### Begin by adding the sources to your project.

**1-3-1.** Click the **Plus** icon (+) to begin adding objects to the project.

**1-3-2.** Select **Add Files** from the pop-up menu to begin adding your source files to the project.

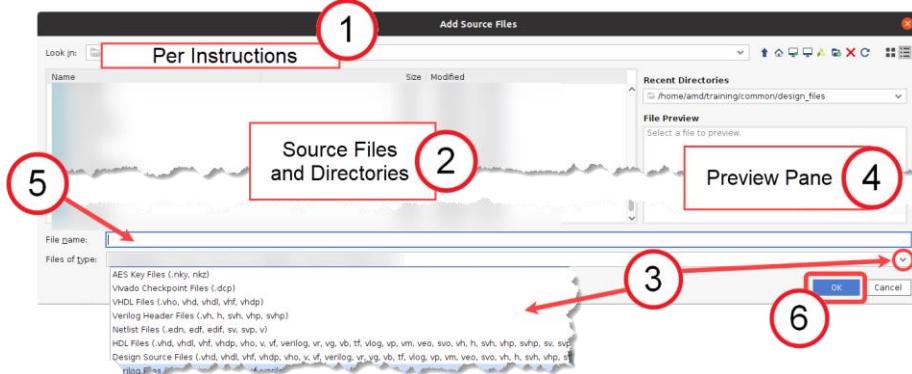


**Figure 7: Adding Sources to the Project**

After you add all the necessary files, the remainder of this dialog box will be addressed.

The Add Source Files dialog box opens.

**1-3-3.** Browse to the **project** directory (1).



**Figure 8:** Adding Source Files

**1-3-4.** Multi-select your **HDL sources** from the list of source files and directories (2).

**Note:** There are equivalent Verilog and VHDL sources. Here you can choose either. Verilog files have the extension `.v`, whereas VHDL files end with `.vhd`.

**Hint:** To use multi-select, press `<Ctrl>` and select each file.

**Hint:** Since this directory contains both VHDL and Verilog sources, you can use the File Type drop-down list and select VHDL files or Verilog files to filter what files are presented (3).

If you select a file, its contents are shown in the preview pane (4).

As you select files, they appear in the File Name field (5).

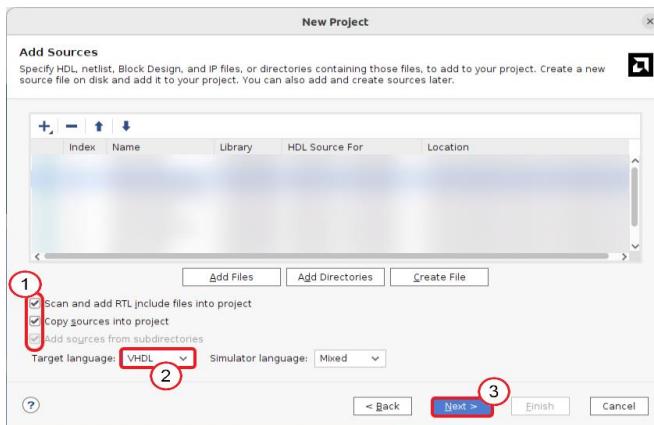
**1-3-5.** Click **OK** to accept the selected files and add them as sources to the project (6).

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

**1-3-6.** Ensure that the **Scan and Add RTL Include Files into Project** and **Copy Sources into Project** options are selected (1).

**1-3-7.** Select the target language using the drop-down list (2).

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



**Figure 9:** Setting Options in the Add Sources Window

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

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

- 1-4-1.** Click the **Plus** icon (+) again to select the IP files.

- 1-4-2.** Select **Add Files** to open the Add Source Files dialog box.

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

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

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

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

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

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

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

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

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

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

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

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

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

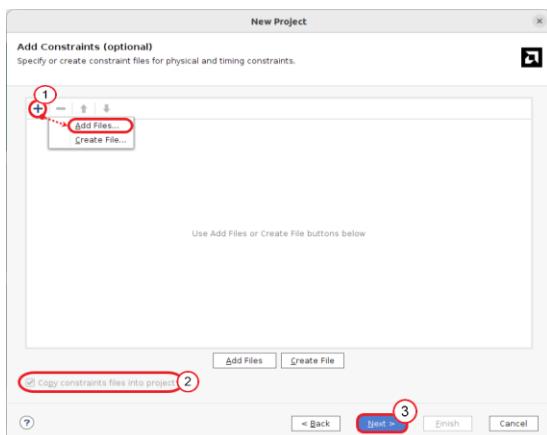
- 1-5-1. Click the **Plus** icon (+) to select the type of object you want to import (1).

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

The Add Source Files dialog box opens.

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

- 1-5-3. Browse to the **project** directory.



**Figure 10: Adding Constraints**

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

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

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

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

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

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

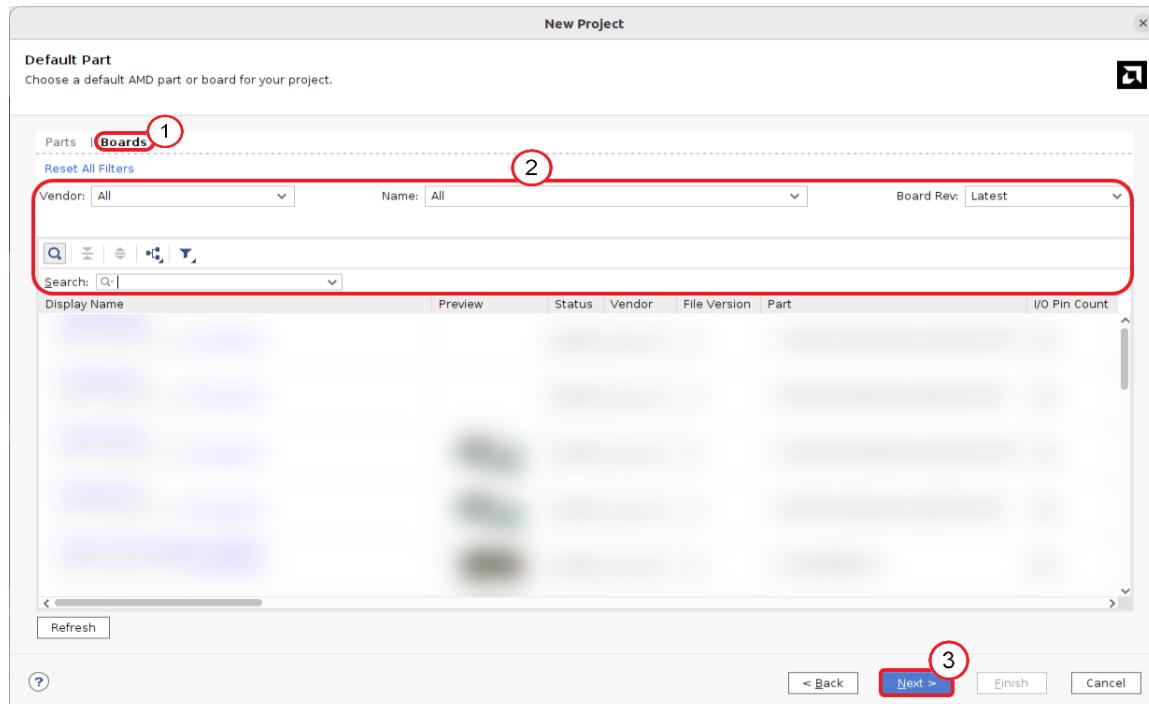
- 1-6. Select the target part by first filtering by board and then by family. If you are not using a supported board, you will need to filter by part.**

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

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

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

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



**Figure 11: Selecting the Board for the Project**

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

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

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

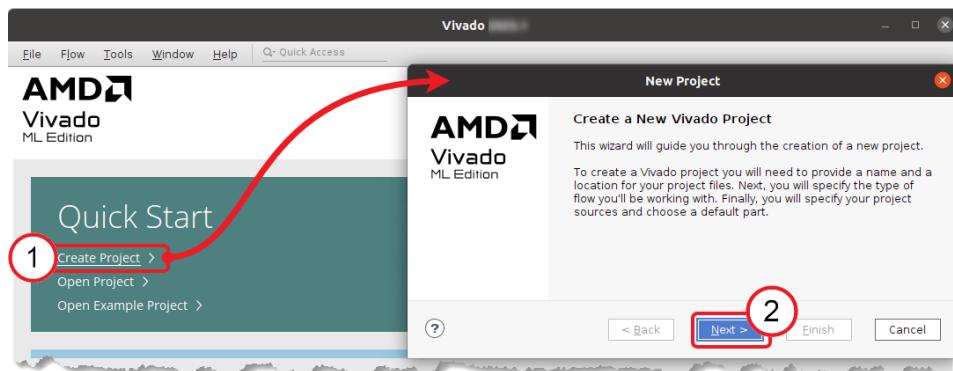
Your project is constructed, and you are presented with the Vivado Design Suite main workspace environment.

## Creating a Blank Vivado Design Suite Project

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

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

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



**Figure 12: Creating a New Vivado Design Suite Project**

This will launch the New Project Wizard.

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

### 1-2. Describe the various aspects of the project.

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

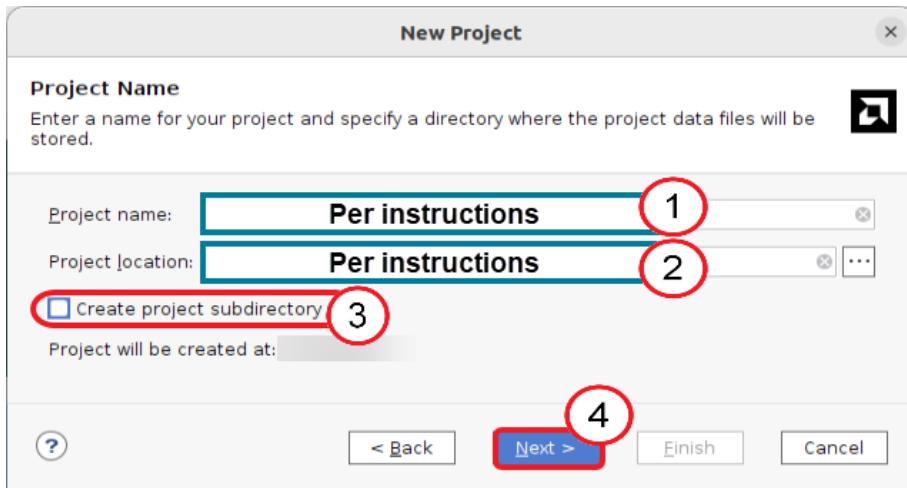
**Enter project location here**

**Important:** The Vivado Design Suite is capable of expanding variables when running under both the Linux and Windows environments. The available environment variables (regardless of the OS) must be predicated with the '\$' symbol.

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

- 1-2-3. Uncheck the **Create Project Subdirectory** option if it is selected (3).

Leaving this checked will create an unnecessary level of hierarchy for the lab.



**Figure 13: Entering the Project Name and Location**

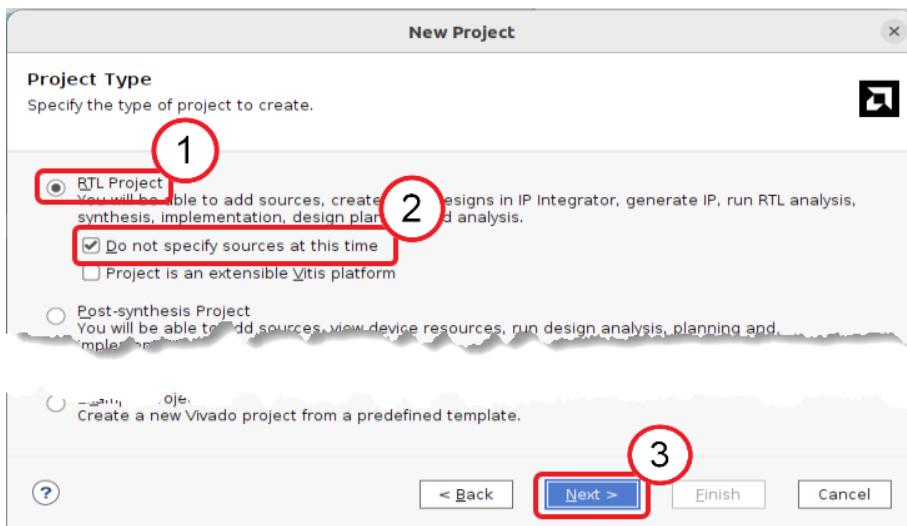
- 1-2-4.** Click **Next** to advance to the next dialog box (4).

Here you will specify your project type as either an RTL project or a post-synthesis project. 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** to instruct the Vivado tool to create a blank project (2).

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



**Figure 14: Specifying Project Options**

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

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

- 1-3-1.** Click **Boards** from the *Default part* area to filter by the board type 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).

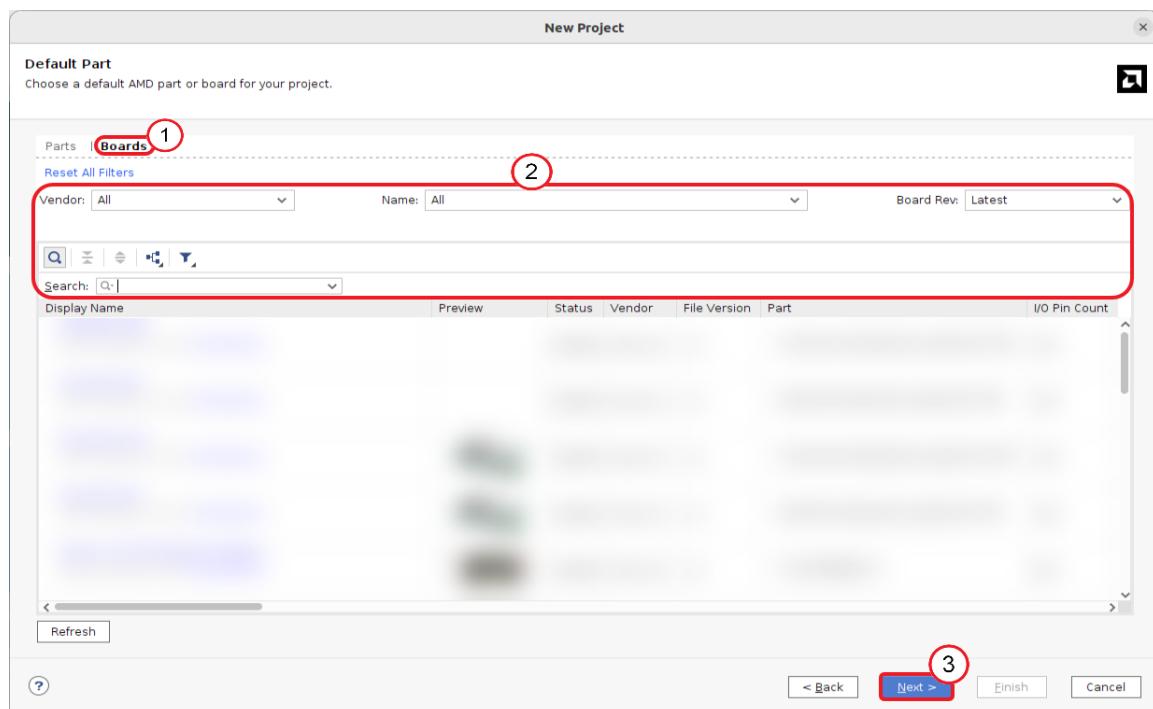
This limits the number of boards seen to those manufactured by the specified vendor.

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

If you accidentally click the hyperlink, a web page will open for that board. You can close the browser page.

If the board you want to use is not immediately visible, click the **Refresh** button to update the board catalog.

**Note:** While the web page contains important information and resources for the board, these details are not needed to complete this lab.



**Figure 15: Selecting the Board for the Project**

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

A summary of your project is displayed. If you want to change any of the information that you entered, you can do so now by clicking **Back** until you reach the correct dialog box. After the project is created, the project properties can still be edited.

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

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

## Opening a Vivado Design Suite Project

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

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

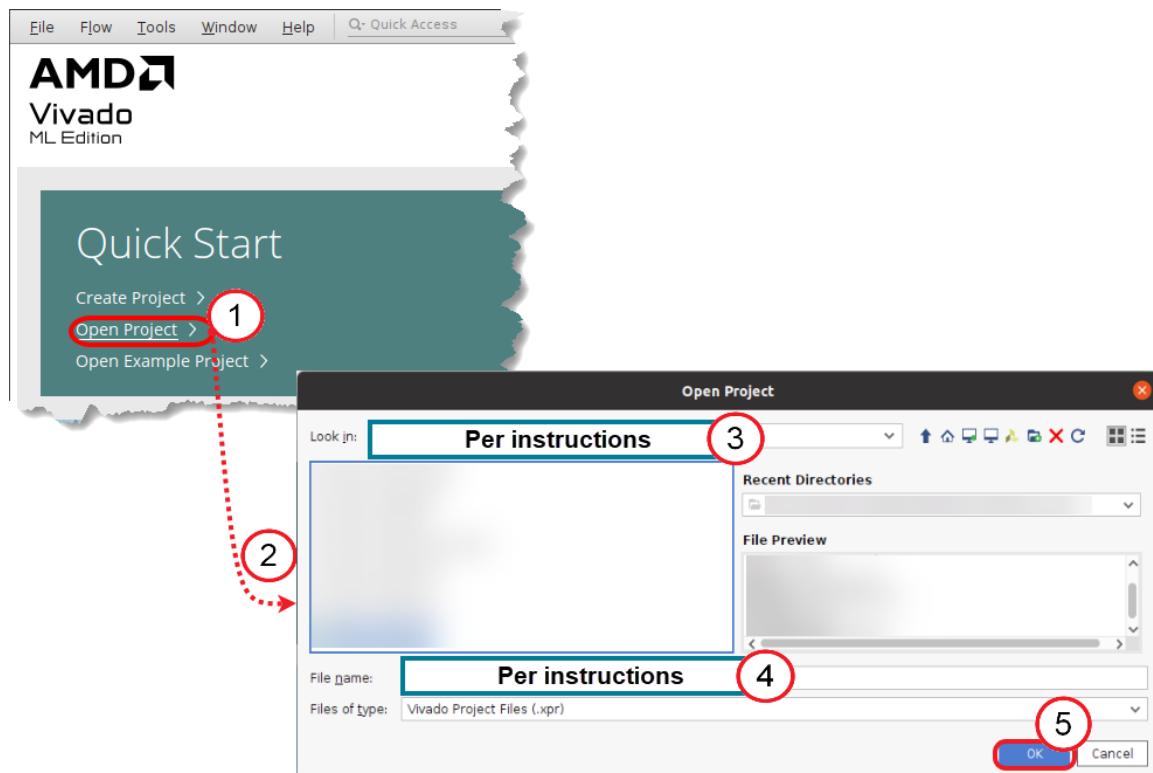
The Open Project dialog box opens (2).

- 1-1-2. Browse to the following directory in the Look in field (3):

project

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

- 1-1-3. Select **your project name** from the files displayed (4).



**Figure 16: Opening an Existing Project**

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

The project now opens in the Vivado Design Suite.

## Opening Source Code in the Editor

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

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

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

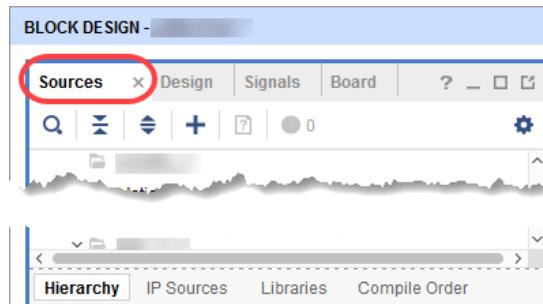


Figure 17: Sources Tab

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

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

## Creating an HDL Source File

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

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

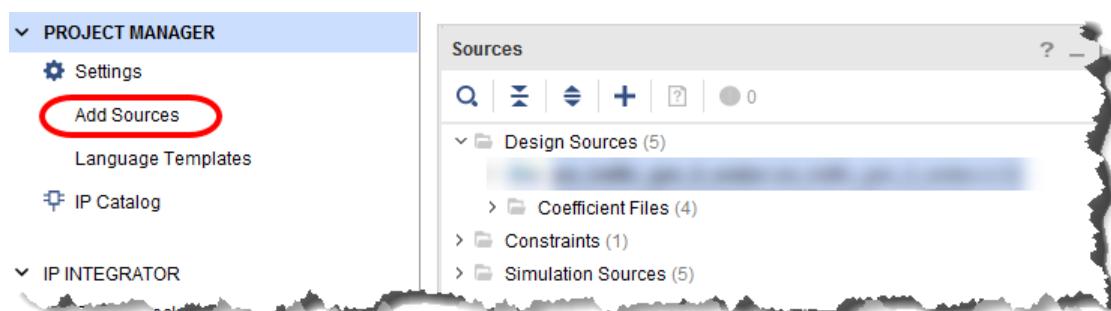
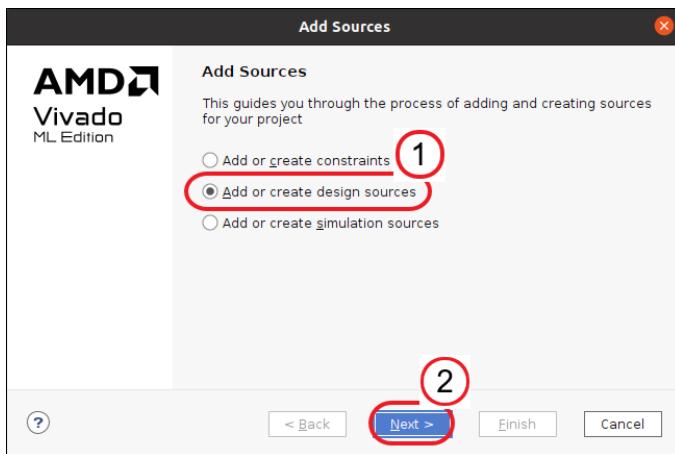


Figure 18: Selecting Add Sources

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

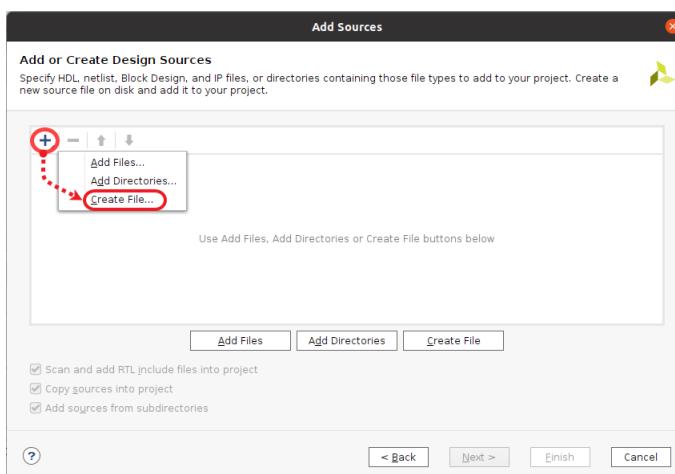


**Figure 19: Selecting Add or Create Design Sources**

**1-1-3. Click Next (2).**

The Add or Create Design Sources dialog box opens.

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

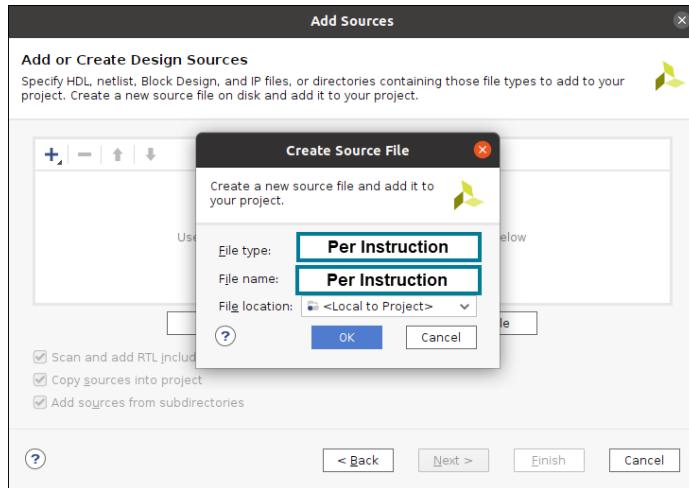


**Figure 20: Selecting Create File**

The Create Source File dialog box opens.

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

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



**Figure 21: Entering File Name and Type**

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

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

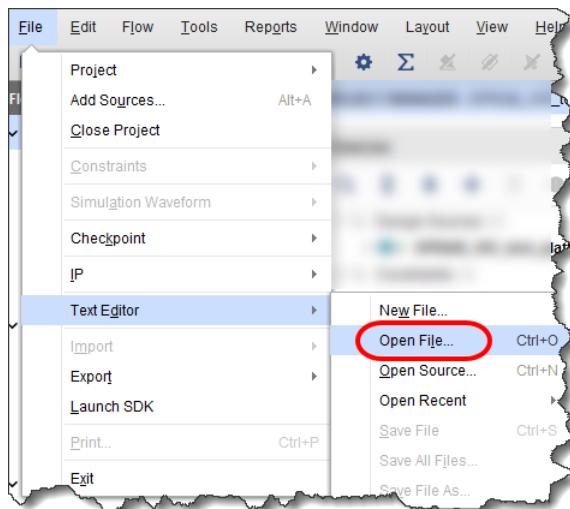
The Define Module dialog box opens.

## Opening a File in the Editor

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

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

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



**Figure 22: Opening a Text File from the File Menu**

A file browser window opens.

- 1-1-2. Navigate to the location of the text file to open.**
- 1-1-3. Select your file.**
- 1-1-4. Click OK.**

The text file opens in the workspace area.

## Importing IP

IP can be imported from a number of tools provided that they adhere to the IP-XACT standard. Once created, the Vivado Design Suite must be made aware of the presence of the IP. This is typically performed from an open project.

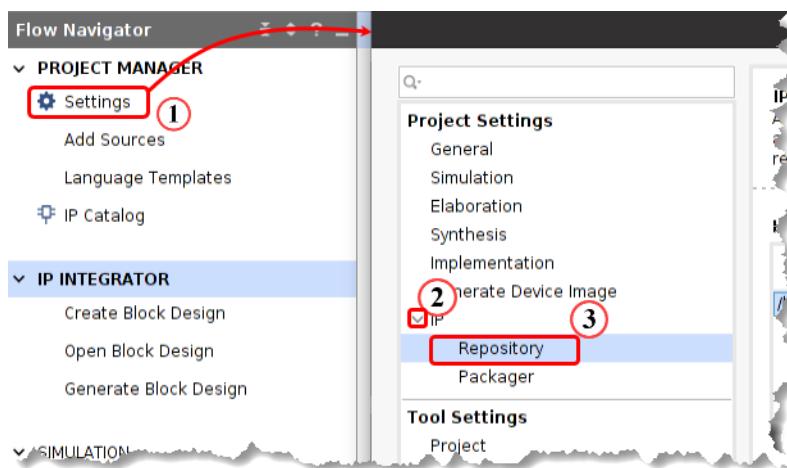
The instructions below describe the process of importing a single IP; however, the steps can be repeated as necessary to collect different IPs from various locations.

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

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

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

- 1-1-2. Expand **IP** (2) and click **Repository** to access the IP settings (3).



**Figure 23: Accessing the Project's IP Settings**

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

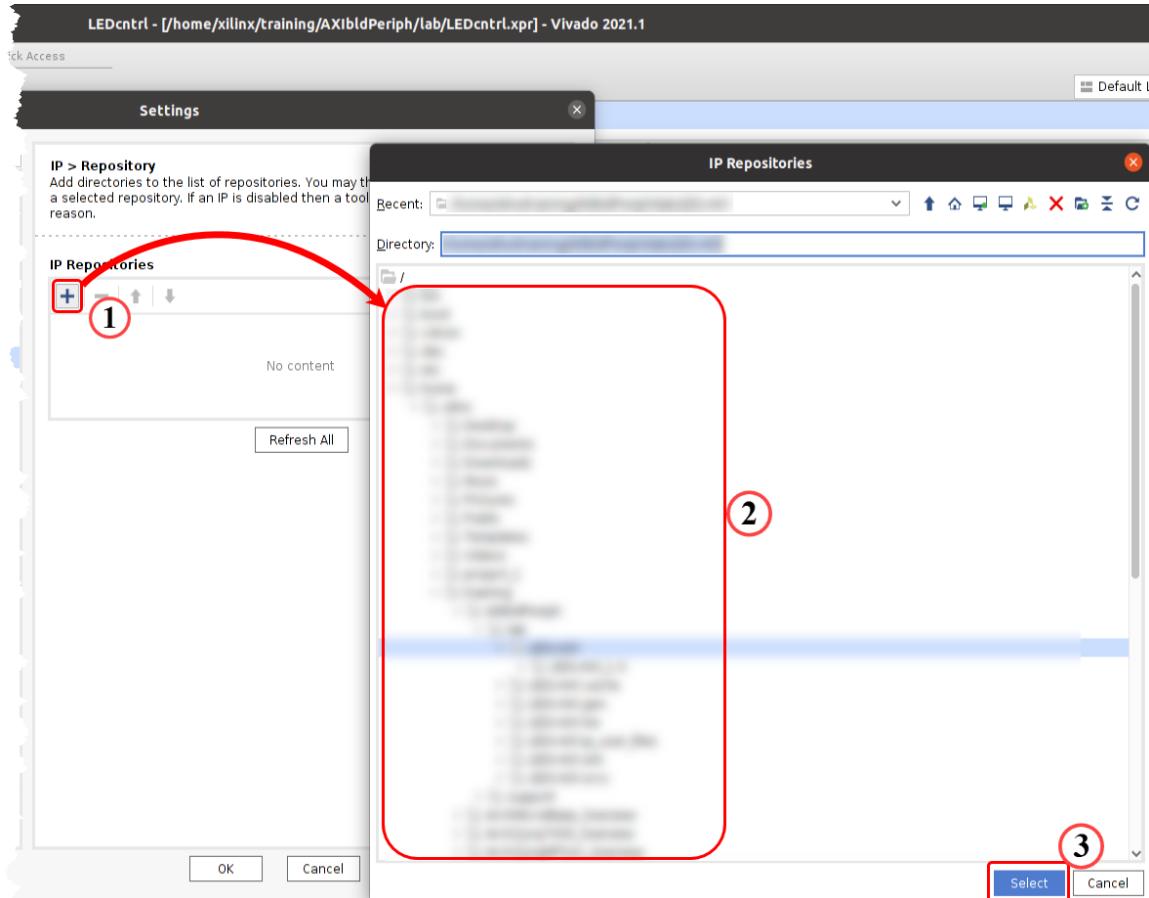
- 1-1-4. Browse to *where your IP is located*.

- 1-1-5. Click **Select**.

**IMPORTANT:** If the IP repository that you have selected has the IP in zip format, then you will need to continue as described below. If, however, your IP has been expanded (for example, if you are revisiting this step or adding another piece of IP), then it will

automatically appear in the IP in Selected Repository field. You can simply click **OK** to complete the repository inclusion process.

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



**Figure 24: Adding the IP Repositories and the Specific IP**

A window opens, showing the number of IPs being added to the project.

- 1-1-6. Click **OK** to add and close this window.
- 1-1-7. Click **OK** again to exit the project settings.

## Closing the Vivado Design Suite Project

---

### 1-1. Close the project.

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

The Close Project dialog box opens.

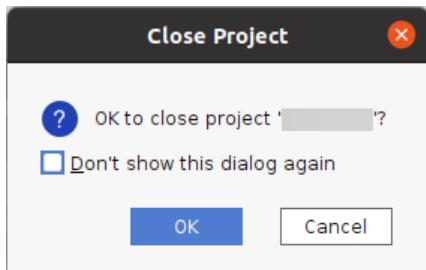


Figure 25: Close Project Dialog Box

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

## Closing the Vivado Design Suite

---

### 1-1. Close the Vivado Design Suite.

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

The Exit Vivado dialog box opens.



Figure 26: Exit Vivado Dialog Box

- 1-1-2. If you are asked to save the project or a portion of the project, select whichever elements of the project you want to save, then click **Save** to save the selected elements; otherwise, click **Don't Save**.
- 1-1-3. Click **OK** when you are asked to exit the Vivado Design Suite.

**Note:** You can choose to select the *Don't show this dialog again* option to avoid being asked for confirmation when exiting the Vivado Design Suite.

## Vivado Add Sources Operations

### In This Section

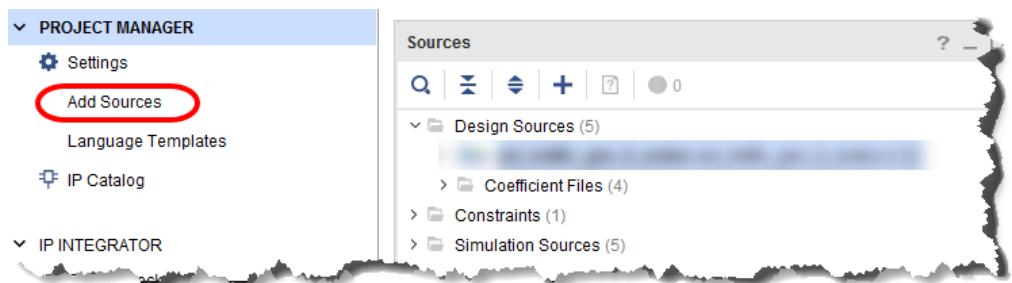
|   |    |
|---|----|
| Adding an HDL Source File .....                       | 25 |
| Adding a Simulation Source File .....                 | 27 |
| Adding a Constraint File.....                         | 28 |
| Creating a SystemVerilog Simulation Source File ..... | 29 |
| Adding Existing Constraints .....                     | 32 |

## Adding an HDL Source File

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

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

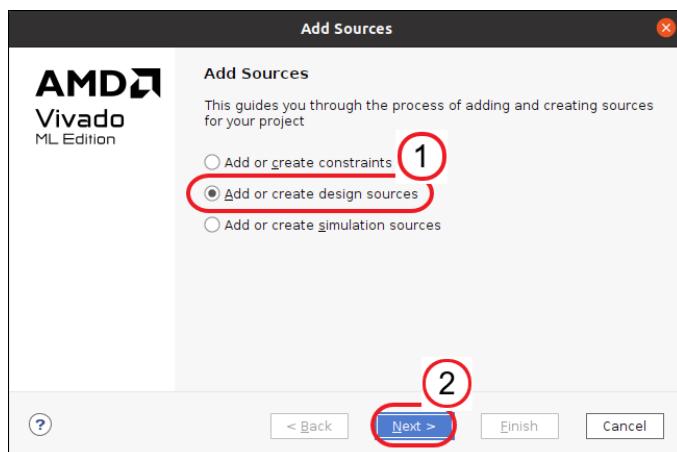
- 1-1-1. Select **Add Sources** under the Flow Navigator tab in the Project Manager.



**Figure 27: Selecting Add Sources**

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

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

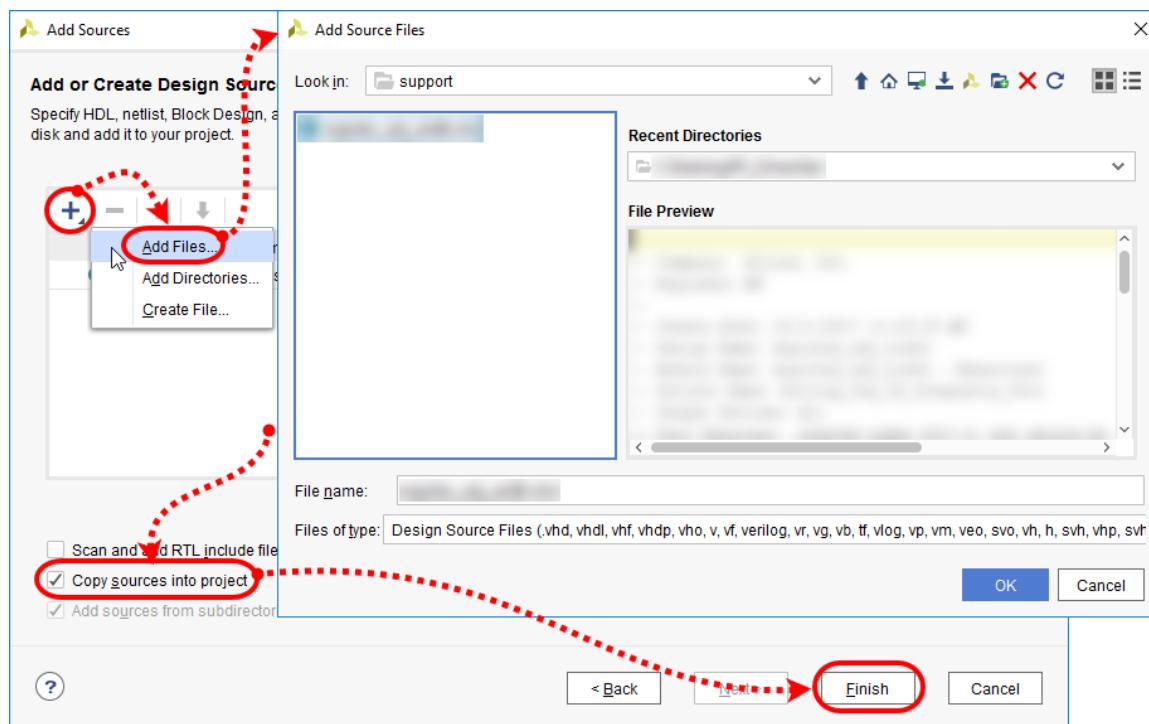


**Figure 28: Selecting Add or Create Design Sources**

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

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

- 1-1-4. Click the **Plus (+)** icon to open the context menu.
- 1-1-5. Select **Add Files** to begin adding the source files to the project.
- 1-1-6. Browse to your source directory if it is not open already.
- 1-1-7. Select **your HDL sources**.
- 1-1-8. Double-click the source file name in the Add Source Files dialog box to select the file(s) or click **OK**.
- 1-1-9. Ensure that the **Copy sources into project** option is selected (when building IP this will be listed as **Copy sources into IP Directory**).



**Figure 29: Selecting Add Files**

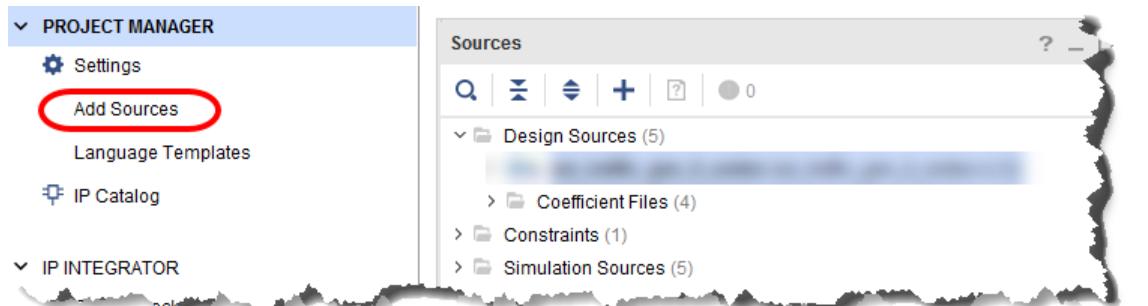
- 1-1-10. Click **Finish** in the Add or Create Design Sources dialog box to add the HDL sources to the project.

## Adding a Simulation Source File

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

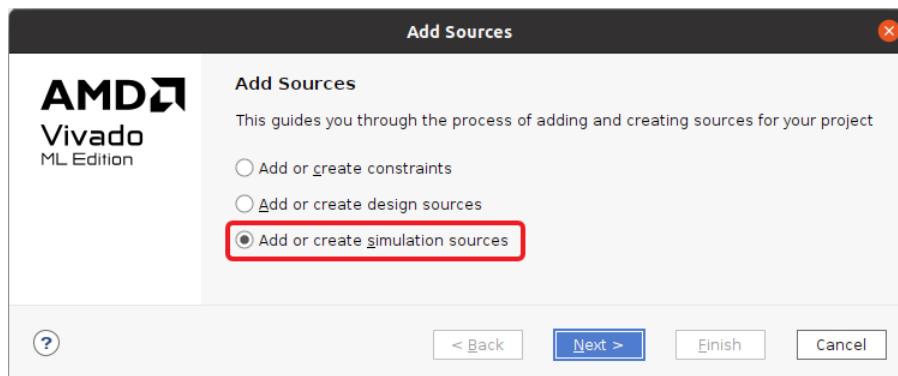
### 1-1. Add simulation files to the design.

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



**Figure 30: Selecting Add Sources**

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



**Figure 31: Add Sources Dialog Box**

- 1-1-3. Click **Next**.
- 1-1-4. Click the **Plus (+)** icon to open the pop-up menu.
- 1-1-5. Select **Add Files** to open the Add Source Files dialog box which allows you to browse to the desired directory.
- 1-1-6. Browse to the \$TRAINING\_PATH/<the topic cluster name>/support directory if it is not open already.
- 1-1-7. Select **your HDL sources**.
- 1-1-8. Click **OK** in the Add Source Files dialog box.
- 1-1-9. Ensure that the **Copy sources into project** option is selected.
- 1-1-10. Click **Finish** to add the file(s) to the project.

## Adding a Constraint File

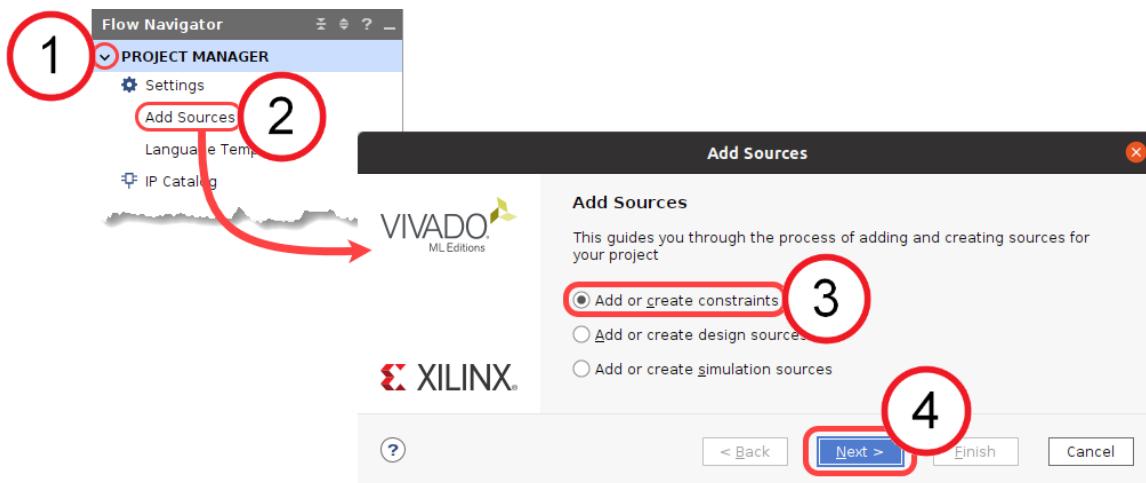
Constraint files can be added to the design at any time. Here's how to add an existing constraint file.

### 1-1. Add the constraint file to the Vivado Design Suite project.

1-1-1. From the Flow Navigator, expand **Project Manager** (1).

1-1-2. Click **Add Sources** to open the wizard (2).

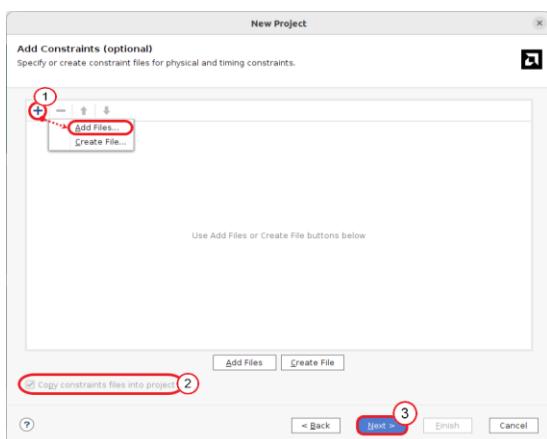
1-1-3. Select **Add or create constraints** (3).



**Figure 32: Accessing the Add Source for Constraints**

1-1-4. Click **Next** to advance to the next screen (4).

1-1-5. Click the **Plus** icon (+) to select whether you want to import or create a constraint file (1).



**Figure 33: Adding Constraints**

1-1-6. Click **Add Files** to open the Add Constraint Files dialog box.

The Add Constraint Files dialog box opens.

- 1-1-7. Browse to the location of your files directory.

[KCU105 users]: Select **your constraints file**.

[VCK190 users]: Select **VCK190\_uart\_led\_Project\_Flow.xdc**.

- 1-1-8. 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.

- 1-1-9. 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.

- 1-1-10. Click **Finish** in the Add or Create Design Sources dialog box to add the sources to the project.

## Creating a SystemVerilog Simulation Source File

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

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

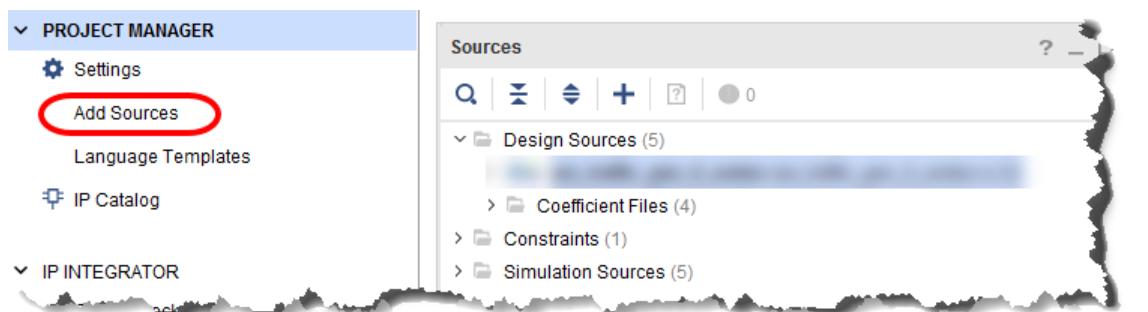


Figure 34: Selecting Add Sources

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

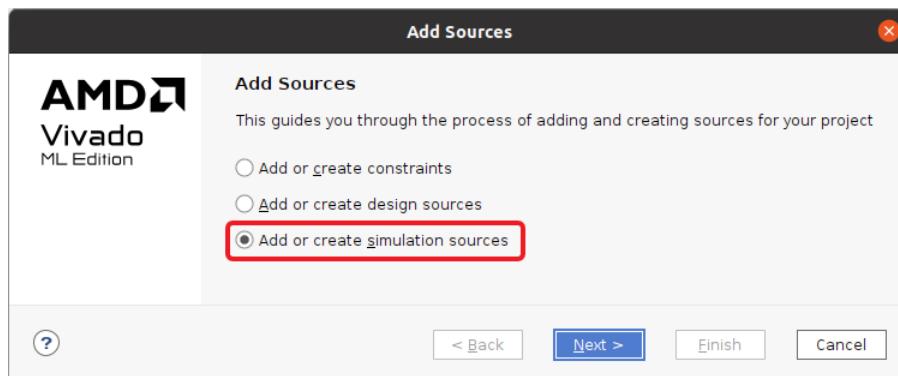


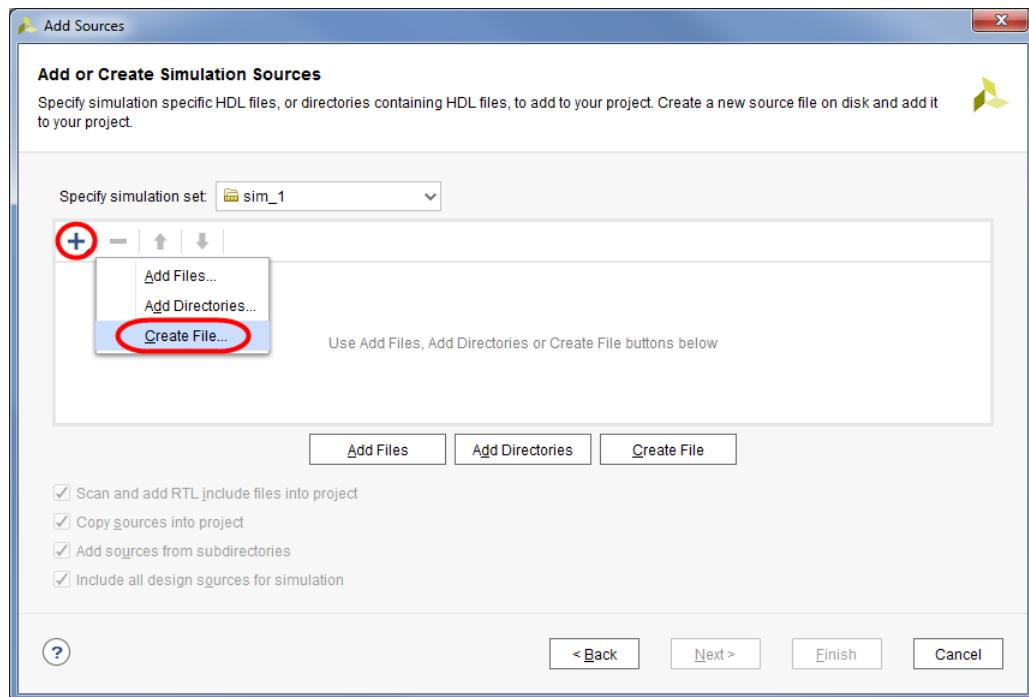
Figure 35: Add Sources Dialog Box

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 36: 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.

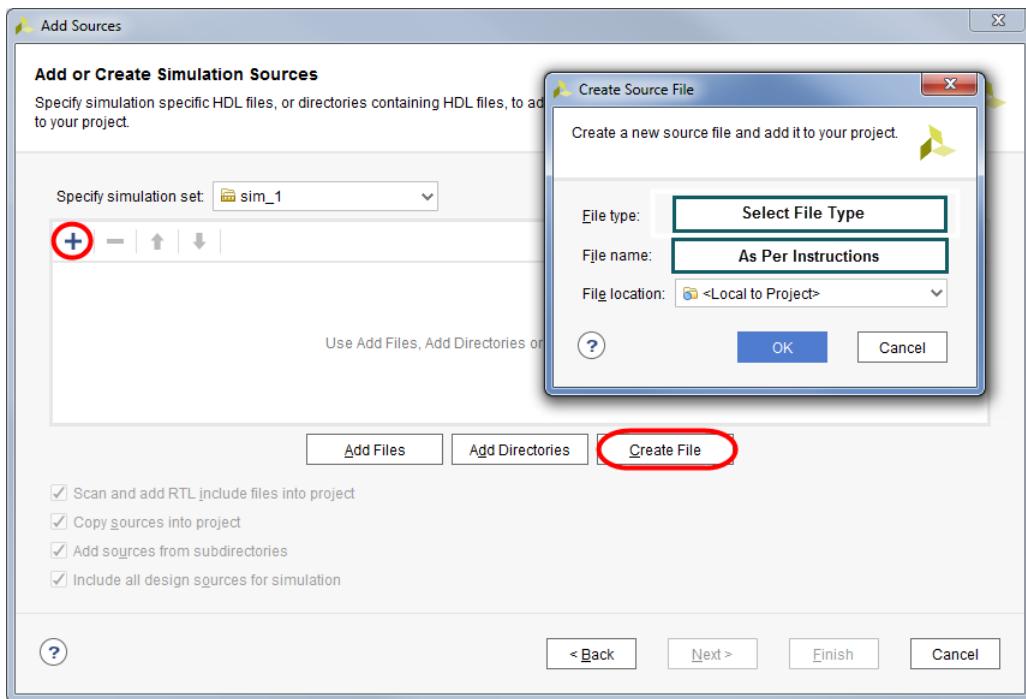


Figure 37: Entering Testbench Filename and Type

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

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

The Define Module dialog box opens.

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

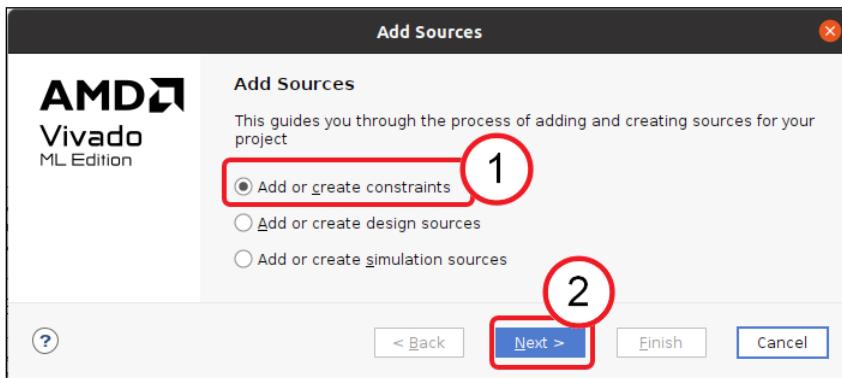
- 1-1-10. Click **Yes** to confirm that there the module definition has not been changed.

## Adding Existing Constraints

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

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

1-1-2. Select **Add or create constraints** (1).



**Figure 38: Adding a Constraint File to a Vivado Design Suite Project**

1-1-3. Click **Next** to advance to the next dialog box (2).

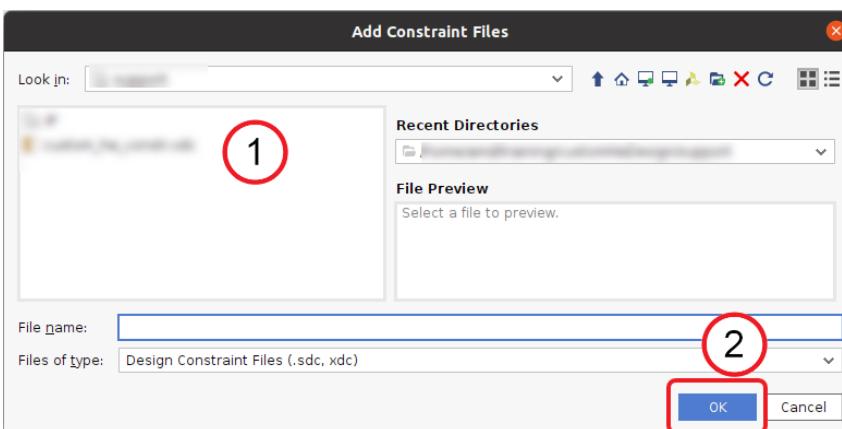
1-1-4. Click the blue **Plus** icon (+).

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

1-1-6. Browse to the \$TRAINING\_PATH/CustEdIP directory.

**Remember:** You can always open a terminal window and enter `echo $TRAINING_PATH` to see the value of this variable.

1-1-7. Select **your constraints file** as the desired constraint file (1).



**Figure 39: Selecting One or More Constraint Files**

1-1-8. Click **OK** to select the file (2).

1-1-9. Click **Finish** to add the constraint file to the project.

## Embedded Operations

### In This Section

|  |    |
|--|----|
| Exporting Only the Hardware Description .....                                  | 33 |
| Exporting the Hardware Description and Bitstream for Software Development..... | 35 |

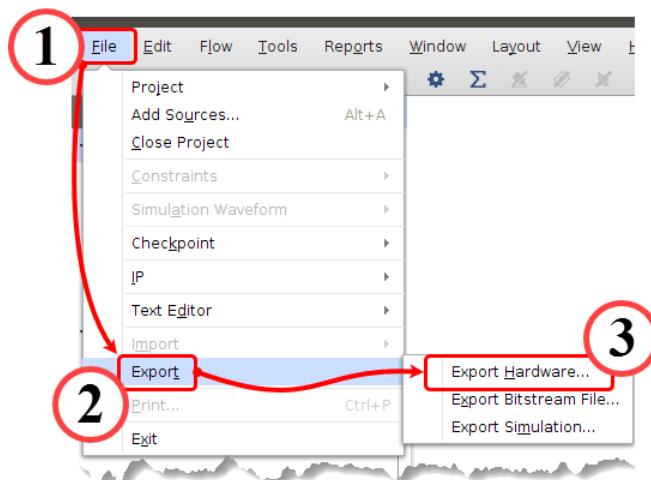
## Exporting Only the Hardware Description

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

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



**Figure 40: Exporting a Design**

The Export Hardware dialog box opens.

- 1-1-2. When the Export Hardware Platform welcome page opens, click **Next**.

### 1-1-3. Select the type of output (1).

If your design is a PS-only design, select **Pre-synthesis**.

Otherwise, ensure that your design is completely implemented with a generated bitstream and select **Include bitstream**.



**Figure 41: Selecting What to Export**

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

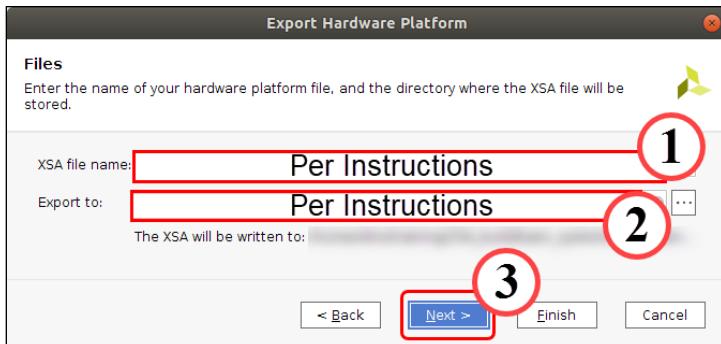
### 1-1-5. Select the XSA file name, which is the name of the output file (1).

This is automatically populated; however, you can select a different name.

### 1-1-6. Populate the Export to field with *the location of the files to be exported from the Vivado Design Suite* as the export location (2).

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

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



**Figure 42: Identifying the Export File Name and Location**

### 1-1-7. Click **Next** to advance to the next dialog box (3).

### 1-1-8. Click **Finish** to proceed with the export.

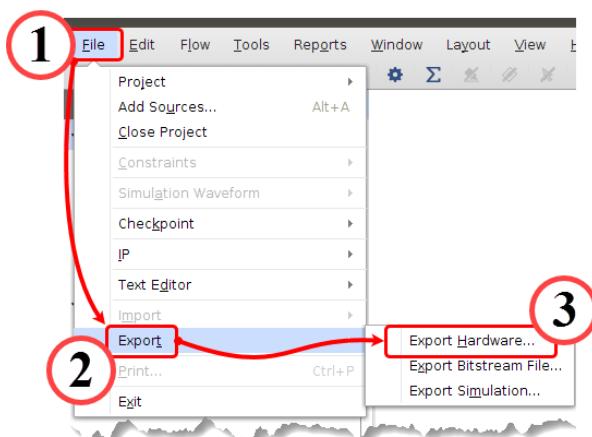
This will create a file containing a description of the hardware and may contain a bitstream as well at the selected export path.

## Exporting the Hardware Description and Bitstream for Software Development

### 1-1. Export the design for the software development tools.

The Vivado Design Suite enables you to export a description of the embedded system design in the form of an XSA file. This XSA can contain the design's bitstream.

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



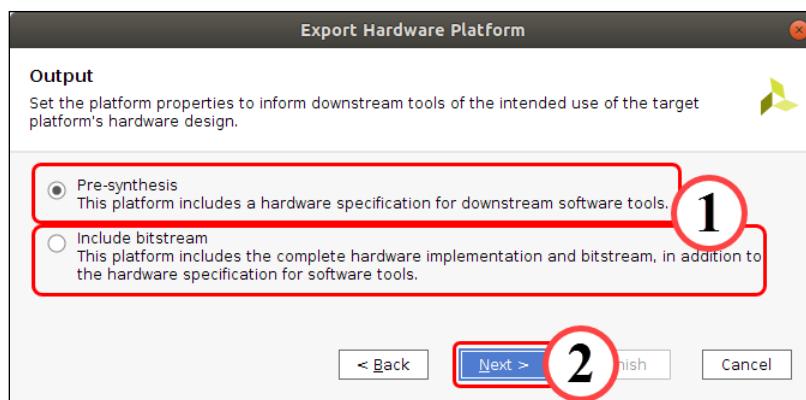
**Figure 43: Exporting a Design**

The Export Hardware Platform dialog box opens.

- 1-1-2. Click **Next** to advance past the explanation panel.
- 1-1-3. Select **the type of output** (1).

If your design is a PS-only design, select **Pre-synthesis**.

Otherwise, ensure that your design is completely implemented and a bitstream is created, and select **Include bitstream**.



**Figure 44: Selecting What to Export**

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

- 1-1-5. Enter **your XSA file name** as the XSA file name, which is the name of the output file (1).

Usually, this is automatically populated; however, you can change this name.

- 1-1-6. Leave the *Export to* field with its auto-populated path, unless you want to select a different path (2).

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.

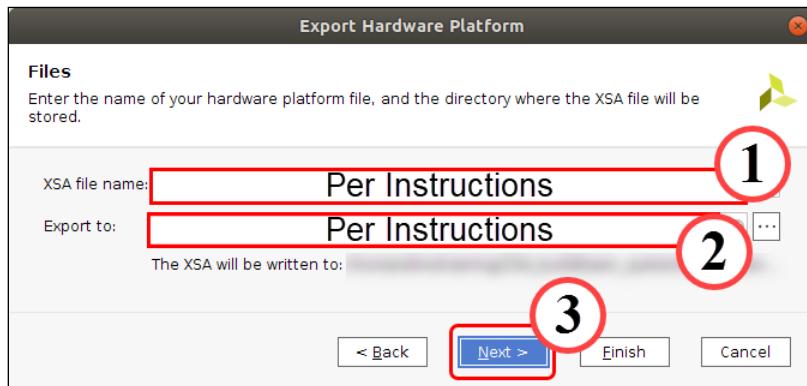


Figure 45: Identifying the Export File Name and Location

- 1-1-7. Click **Next** to advance to the next dialog box (3).

- 1-1-8. Click **Finish** to proceed with the export.

This will create a file containing a description of the hardware and may contain a bitstream as well at the selected export path.

## Vivado IP Integrator Operations

### In This Section

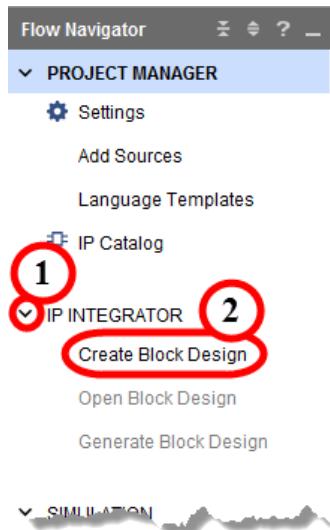
|  |    |
|--|----|
| Creating an IP Integrator Block Design.....                              | 38 |
| Closing a Block Design .....   | 39 |
| Opening a Block Design .....   | 40 |
| Zoom Operations in the IPI Block Diagram Editor .....                    | 41 |
| Adding a Processor to the Block Design.....                              | 42 |
| Adding a Zynq Family Processor Block .....                               | 44 |
| Adding a Processing System Block .....                                   | 47 |
| Running Block Automation for the PS.....                                 | 51 |
| Customizing the Processing System IP .....                               | 52 |
| Customizing the Zynq UltraScale+ MPSoC PS .....                          | 56 |
| Selecting a Zynq7 PS Preset Template .....                               | 58 |
| Customizing the MicroBlaze Processor .....                               | 58 |
| Adding IP to an IP Integrator Block Design.....                          | 61 |
| Customizing IP .....   | 63 |
| Making Manual Connections Between Two Objects in the IP Integrator ..... | 63 |
| Renaming an IP Block .....   | 66 |
| Adding a Port to an IP Integrator Block Diagram .....                    | 66 |
| Adding an Interface Port to an IP Integrator Block Design .....          | 69 |
| Making a Pin or Interface External.....                                  | 70 |
| Running Connection Automation .....                                      | 71 |
| Running Connection Automation for Multiple Modules .....                 | 72 |
| Running Block Automation.....  | 73 |
| Locating Objects in the Board Design.....                                | 75 |
| Mapping Peripheral Addresses .....                                       | 77 |
| Generating Output Products .....   | 78 |
| Creating a Hierarchical Block in a Block Design.....                     | 80 |
| Adding IP to a Hierarchy Block .....                                     | 81 |
| Expanding the Contents of a Hierarchical Block.....                      | 82 |
| Opening a Hierarchical Block in a New Tab .....                          | 83 |
| Marking Signals for Debugging .....                                      | 84 |
| Updating IP .....  | 86 |
| Regenerating the Layout.....   | 88 |
| Running a Design Rule Check/Design Validation .....                      | 88 |
| Validating the Block Design .....  | 89 |
| Saving the Block Design .....  | 90 |
| Generating a Wrapper for a Block Design .....                            | 90 |

## 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.

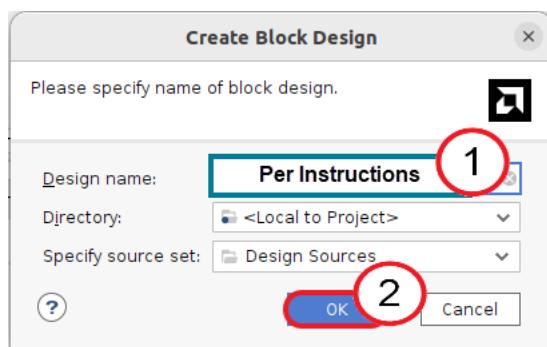
### 1-1. Create a Vivado IP integrator block diagram.

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



**Figure 46: Launching the IP Integrator**

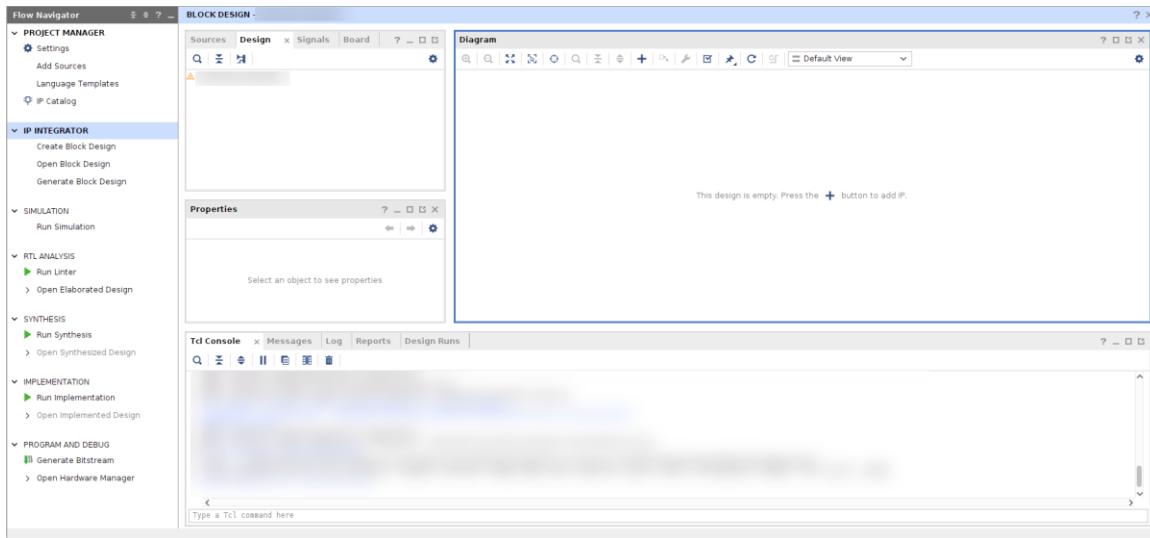
- 1-1-3. Name the design **name of your block design** when the Create Block Design dialog box opens (1).



**Figure 47: 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 48: Initial View of the IP Integrator Tool**

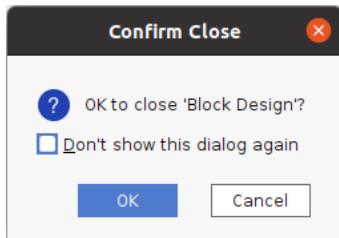
## Closing a Block Design

### 1-1. Close the block design.

1-1-1. [Optional] If you want to preserve the block design, press **<Ctrl + S>**.

1-1-2. Select **File > Close Block Design**.

Alternatively, you can click the "X" in the upper right-hand corner of the Block Design view.



**Figure 49: Confirming Block Design Closure**

**Note:** You can select **Don't show this dialog again** if you do not want to confirm closure in the future.

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

## Opening a Block Design

---

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

1-1-1. Access the **Sources > Hierarchy** view.

1-1-2. Locate the block design in the listing of the hierarchical sources.

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

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

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

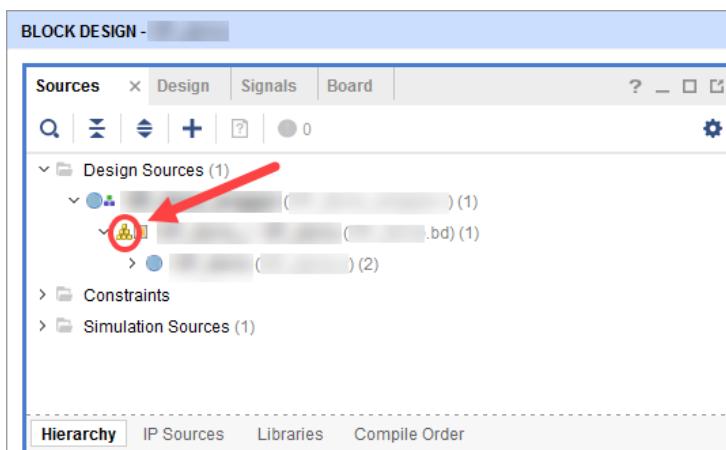
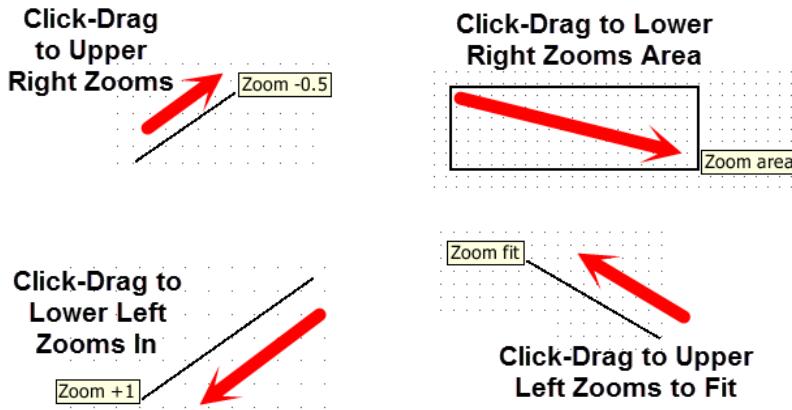


Figure 50: Opening an Existing Block Design

## Zoom Operations in the IPI Block Diagram Editor

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

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



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

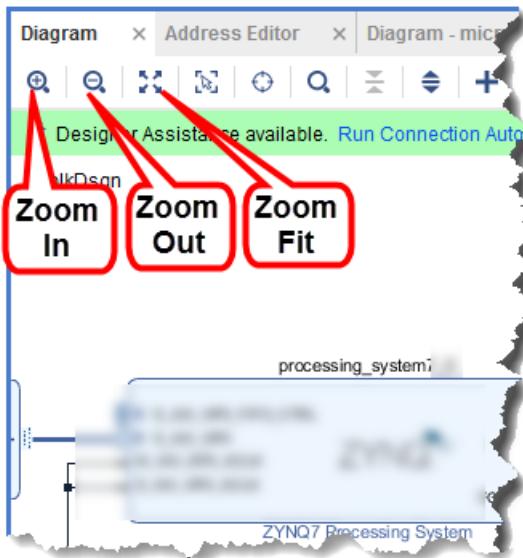


Figure 51: Block Diagram Zoom Toolbar

## 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.

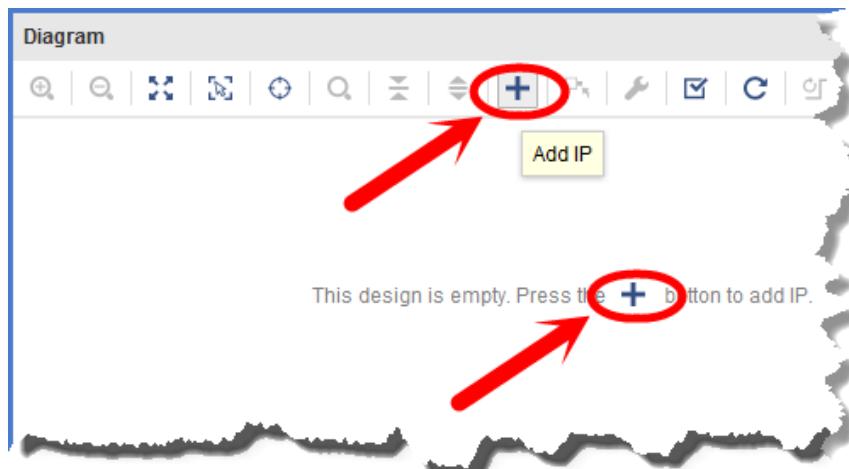


Figure 52: Opening the IP Catalog from the Add IP Icon

-- OR --

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

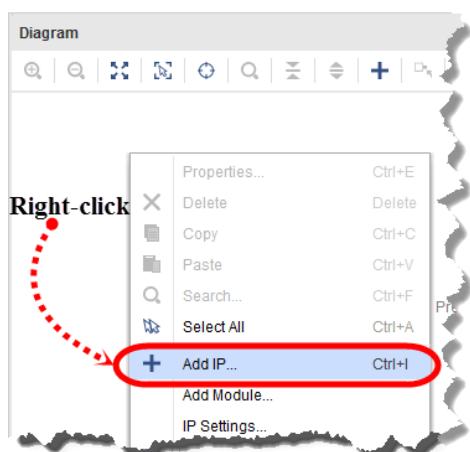
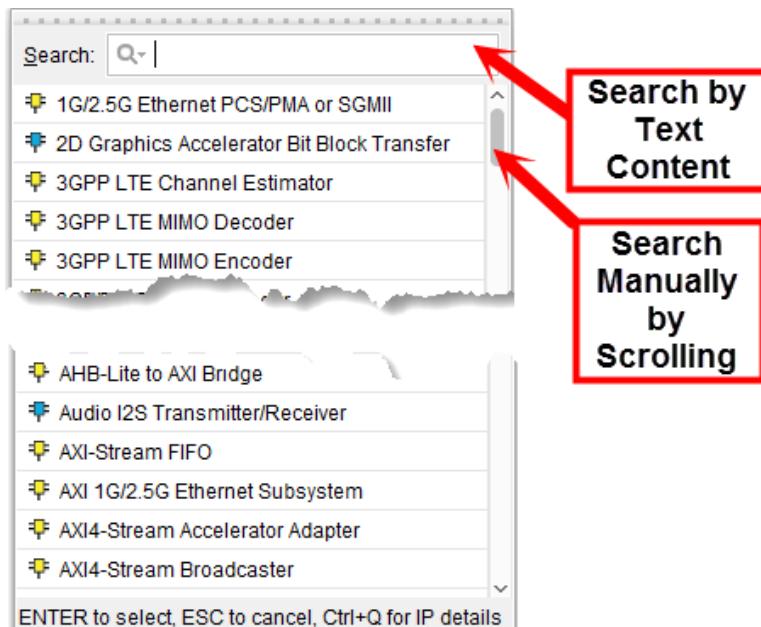


Figure 53: Opening the IP Catalog from Right-Clicking the Workspace

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

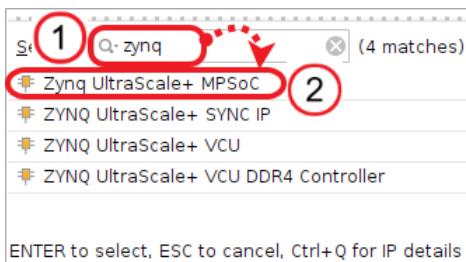
Once the IP catalog opens, you can search for the processor block.



**Figure 54: Two Mechanisms to Search for IP**

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

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



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

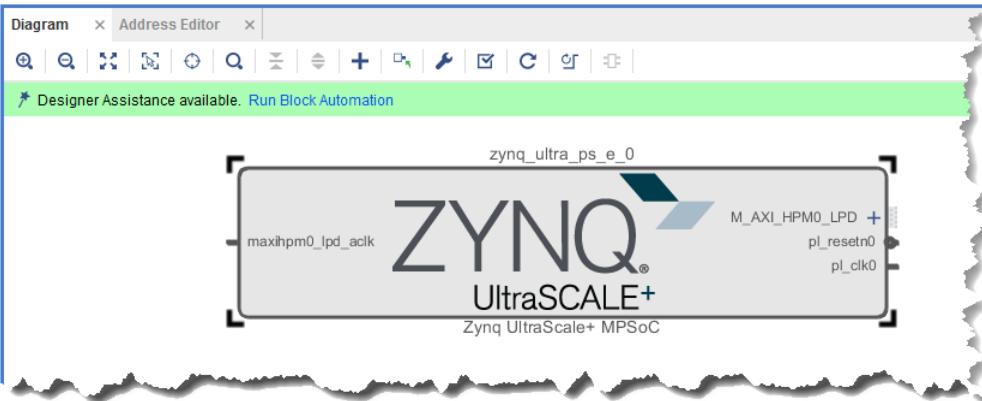


Figure 55: Zynq UltraScale+ MPSoC Block Symbol

## Adding a Zynq Family Processor Block

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

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

- 1-1-1. Use one of the following methods to open the IP catalog:

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

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

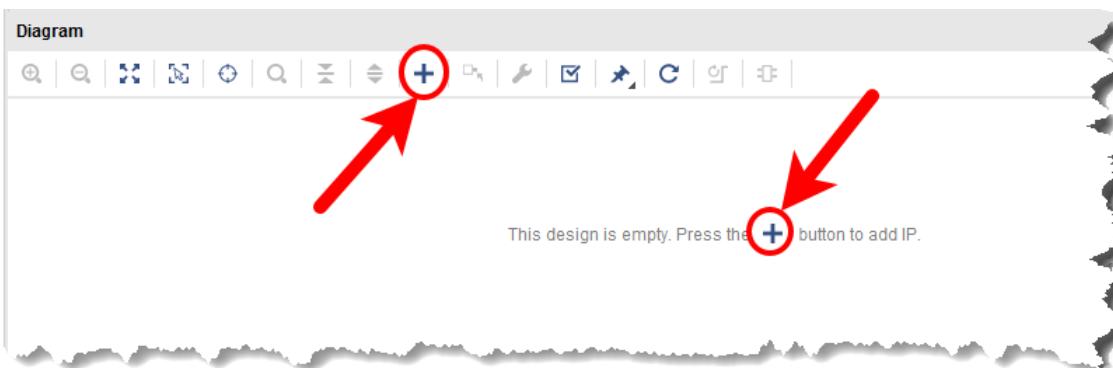
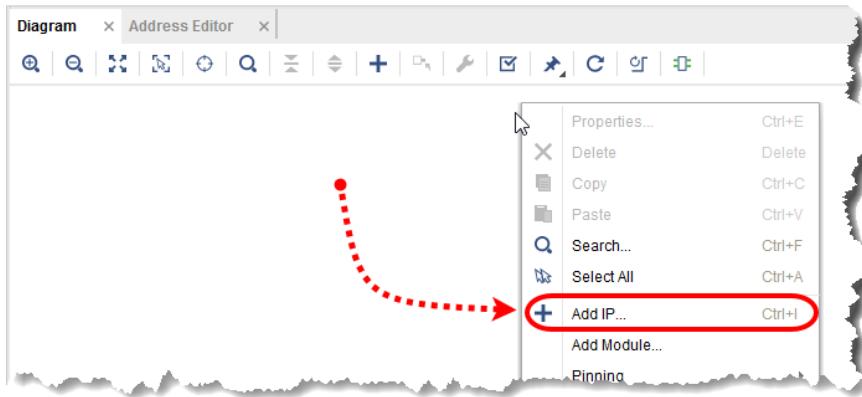


Figure 56: Opening the IP Catalog from the Initial Information Bar Display

-- OR --

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



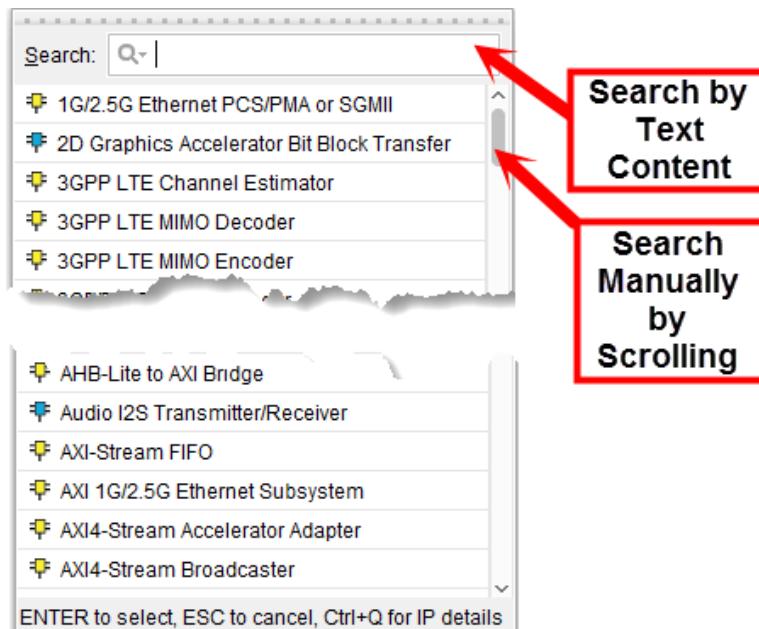
**Figure 57: Opening the IP Catalog by Right-Clicking the Workspace**

-- OR --

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

**Note:** Although the convention is to show the capital letter, the key combination uses a lower case 'i'.

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



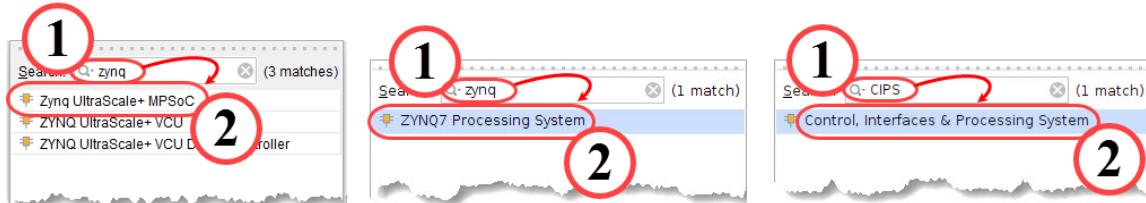
**Figure 58: Two Mechanisms to Search for IP**

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

The proper IP will appear depending on the board or device that you selected. The PS of the Zynq-7000 device will appear as ZYNQ7 while the MPSOC/RFSOCs will show as

Zynq UltraScale+ MPSoC, and so forth. Only the IP supporting the device you selected will appear in the IP list.

Note that the processing system for the Versal devices is referred to as "Control, Interfaces & Processing System" and will NOT appear under a search for "Zynq" as it is not in the Zynq family of devices.



**Figure 59: Selecting Multiple Architectures**

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

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



**Figure 60: Zynq UltraScale+ MPSoC/RFSoC PS Block Symbol**

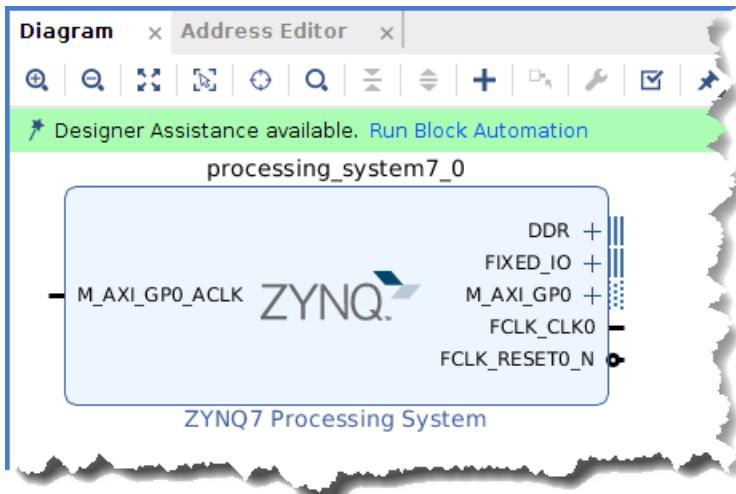


Figure 61: Zynq-7000 SoC PS Block Symbol

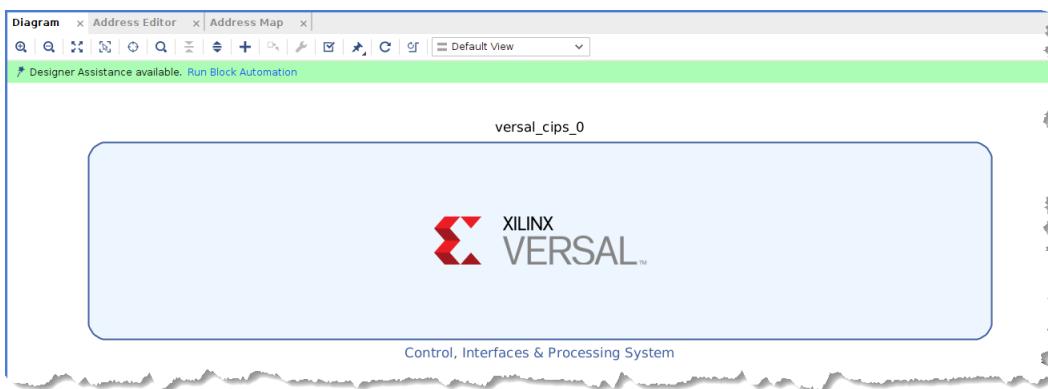


Figure 62: Versal ACAP CIPS Block Symbol

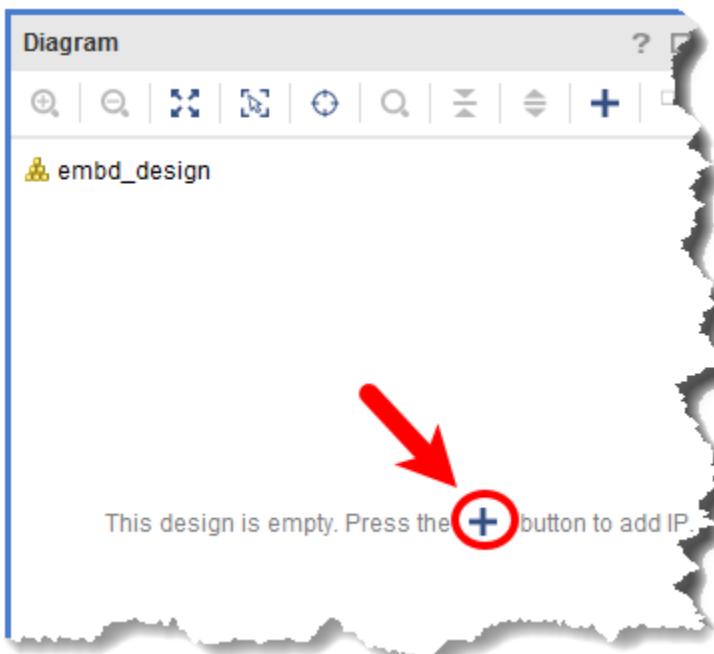
## Adding a Processing System Block

### 1-1. Add a processing system block to the design.

The processing system is available for multiple devices, including the Zynq 7000 SoC, MPSoCs, RFSoCs, and other families. Only the processing system for the specific family can be added to the design. That is, you cannot add a Zynq 7000 SoC processing system to a Versal part.

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.

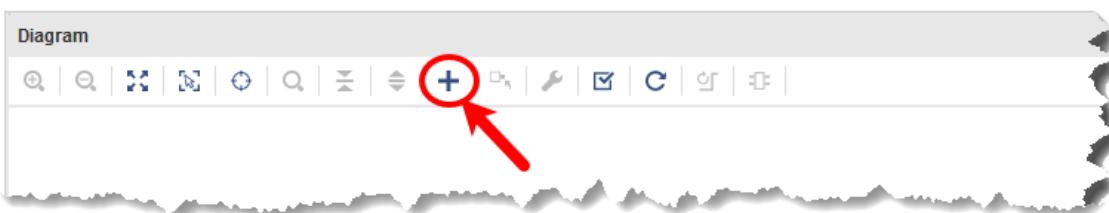
- 1-1-1. Click the **Add IP** icon (+) to open the IP catalog.



**Figure 63: 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:

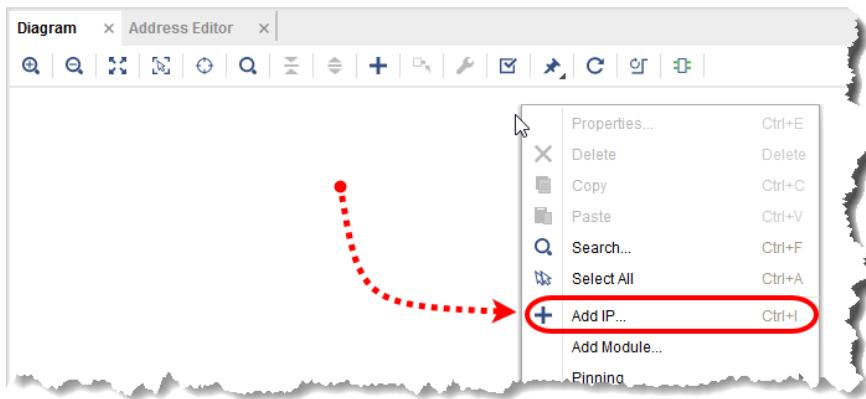
- Click the **Add IP** icon (+) in the toolbar of the workspace.



**Figure 64: Opening the IP Catalog from the Add IP Icon**

-- OR --

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



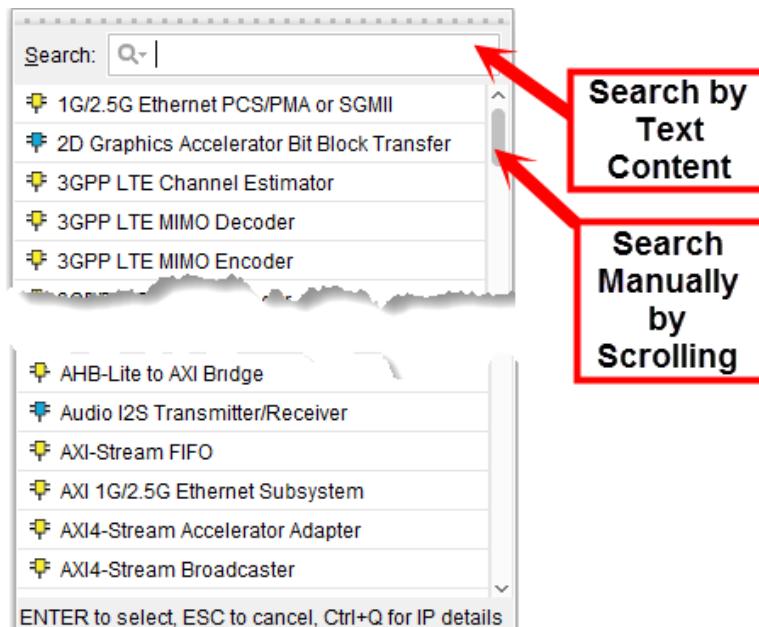
**Figure 65: Opening the IP Catalog by Right-Clicking the Workspace**

-- 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.

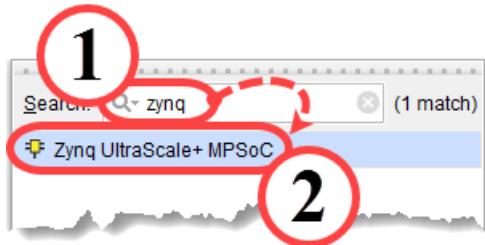
Once the IP catalog opens, you can search for the processing system block.



**Figure 66: Two Mechanisms to Search for IP**

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

The processing system for the selected family appears. The exception to this is that an MPSoC PS is used by both the MPSoC and RFSoC parts, so if an RFSoC device is targeted, a Zynq UltraScale+ MPSoCs PS is used.



**Figure 67: Selecting the Zynq UltraScale+ MPSoC IP**

This is the IP for the entire processing system.

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



**Figure 68: Zynq UltraScale+ MPSoC Block in the Block Diagram**

## Running Block Automation for the PS

Many IP blocks 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 IP block.

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



**Figure 69: Selecting the Diagram Tab**

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

The Run Block Automation dialog box opens.

If multiple IPs have not yet been configured with the Block Automation, they will appear in the left-hand pane. This allows bulk selecting IP.

- 1-1-3. Select only the name of the PS block from the left-hand pane (1).

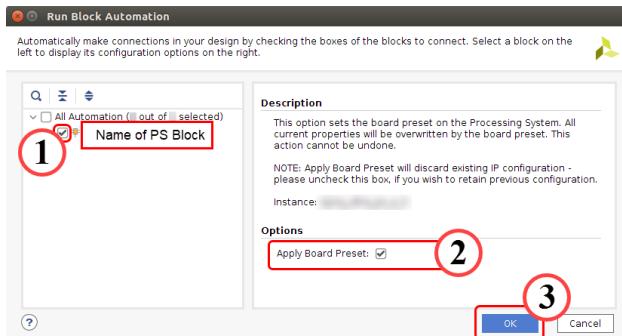
This will bring up the specific options for the PS in the right-hand pane.

You now have an opportunity to apply a board preset. If you apply the board preset, this will undo any modifications that you made to the PS using the Re-customization Wizard.

Select this option ONLY if you have not made any changes to the PS's configuration.

- 1-1-4. Select or deselect the **Apply Board Preset** option (2).

**Warning:** Having this selection enabled will overwrite any work done using the Re-customization Wizard.



**Figure 70: Running the Designer Automation Assistance on the PS**

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

- 1-1-5. Click **OK** to run the block automation on the PS (3).

## Customizing the Processing System IP

The processing system (PS) contains a large number of customizable features. The following instructions will illustrate how to access various sections of the PS. Because the Re-customization Wizard differs among the different processing systems, explicit directions regarding what to turn on or off is omitted. This step is one of exploration rather than achieving a specific outcome.

While the Re-customization Wizard for the Versal devices is significantly different, it fundamentally performs the same types of tasks: identifying which peripherals are used, how they are configured, which AXI connections are available to the PL, etc.

The figures and instructions focus on the Zynq family of PS, and the details of the Versal ACAP PS are not discussed here; however, notes regarding the Versal device are included where appropriate.

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

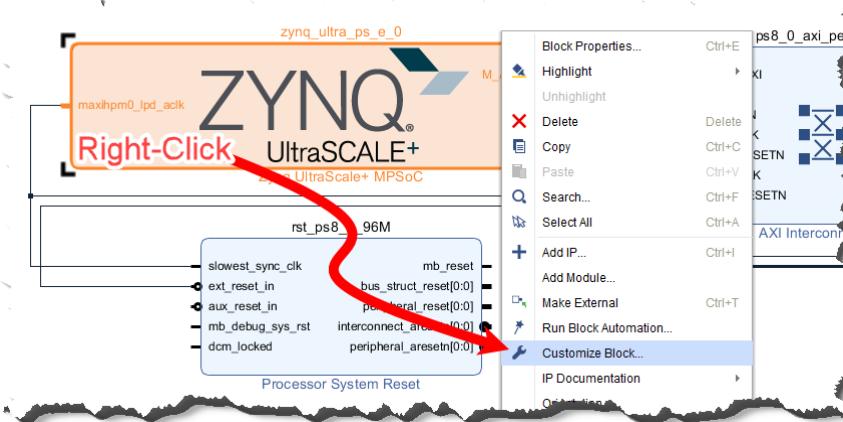
#### 1-1-1. Double-click the PS block.

The specific name of this block varies with the type of device. Some variation of "Zynq" will bring up the PS for the Zynq-7000 devices as well as the MP/RFSoC devices. The Versal device has its processing system IP as "CIPS" for Control, Interfaces & Processing System.

Users with projects targeting a ZCU104 board or any other MPSoC or RFSoC device will see a ZYNQ UltraScale+ block. Zed/ZC702 users will see a Zynq entry specific to the part or board selected. Versal ACAP users (VCK190) will see the CIPS module.

-- OR --

Right-click the PS/CIPS block and select **Customize Block**.



**Figure 71: Opening the Re-customization Wizard for the PS (MP/RFSoC Example Shown)**

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

## 1-2. [Zynq-7000 SoC users]: Import board-specific settings.

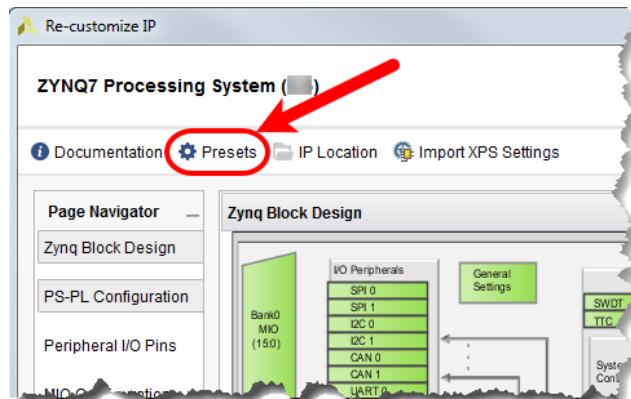
If you are using a supported evaluation board, you will probably want to load the pre-defined settings as it contains many of the parameters specific to that board (such as DDR memory timing parameters, enabled peripherals supported by that board, etc.) A similar capability is available for the MPSOC/RFSOC devices and is explained elsewhere.

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

The Zynq UltraScale+ MPSOC presets behave differently than the Zynq-7000 device presets. The Zynq-7000 device presets contain settings for various evaluation boards while Zynq UltraScale+ MPSOC presets enable you to save and recall your own specific settings. The equivalent functionality can be found for the MP/RFSOC devices using Block Automation.

### 1-2-1. Click **Presets** to access the pre-defined values associated with one of the supported boards that supports the Zynq-7000 SoC (such as the ZC702 or ZedBoard).

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



**Figure 72: Accessing the Presets (Zynq PS)**

If you have a Tcl script that describes the configuration of the PS, you can load it by selecting *Presets > Apply Configuration*.

### 1-2-2. Select **your board** for the preset.

This will configure the PS for the settings specific to that board, including DDR timing parameters, peripherals, etc.

**Note:** The Versal device contains a multitude of presets and individual groups that can be uniquely preset for specific purposes. These settings can be saved as a system-level preset and recalled at will.

### 1-2-3. Skip the next instruction as it duplicates this instruction, but for Versal device users.

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.

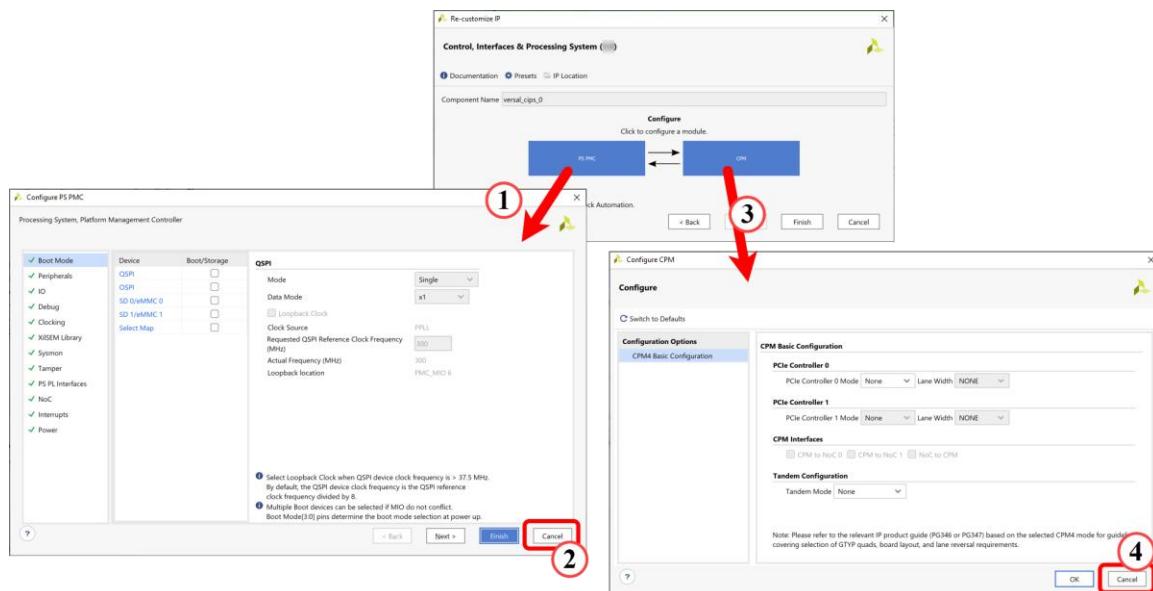
### 1-3. [Versal ACAP users]: Explore the Re-customization Wizard.

- 1-3-1. When the wizard opens, click **Next** to move past the "Presets" page.
- 1-3-2. Click the **PS PMC** block to open a configuration window for the processing system and platform management controller (1).

This block more closely aligns with the Zynq families in that it provides access to the configuration information for the peripherals, AXI interconnects, clocking, etc.

New in the Versal family's architectures is the network on chip (NoC), which facilitates rapid data movement among the major blocks in the system and, unlike the other processing systems which contains their own DDR memory controllers, the CIPS block requires the NoC to gain access to up to four independent or coordinated DDR memory controllers.

- 1-3-3. Click each element in the list to see the options available for that sub-section (or click **Next** which will proceed through the list).
- 1-3-4. Click **Cancel** to return to the previous page without making any changes (2).
- 1-3-5. Click the **CPM** block to access the CPM (PCIe block and CPM interface) configuration settings (3).
- 1-3-6. Explore the various modes available for the PCIe blocks.
- 1-3-7. Click **Cancel** to return to the previous page without making any changes (4).



**Figure 73: CIPS Re-customization Wizard - Page 2**

- 1-3-8. Click **Cancel** to exit the wizard.

- 1-3-9.** Enter the following in the Tcl command line to run the Tcl proc to configure the Versal ACAP CIPS:

```
versalCIPSconfigure
```

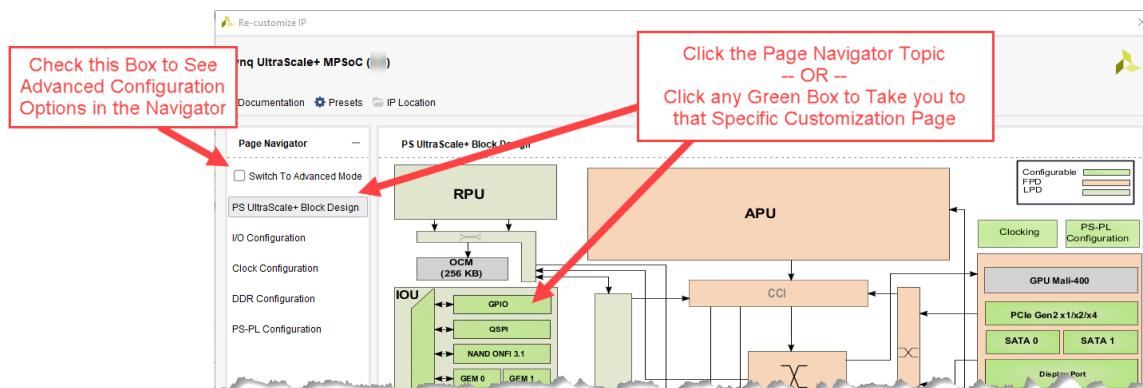
This proc configures the CIPS IP so that the following instructions can be performed.

#### **1-4. [All Zynq device users]: Select the page or specific block to re-customize.**

- 1-4-1.** Click the topic in the Page Navigator or any green (active) box from the diagram.

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

**Note:** Advanced mode is available for the MPSOC/RFSOC families, but not for the Zynq-7000 family. The figure below represents the MPSOC devices' Re-customization main dialog box. The Zynq-7000 family is similar, but less complex.



**Figure 74: Navigating the Zynq PS Recustomization Dialog Box**

- PS-PL Configuration: This page contains all the signals that cross the PS-PL boundary. For the Zynq-7000 devices, these signals are broken down further into:
  - General: Contains default baud rates for the UARTs, FTM trace buffer settings, PS-PL cross triggering enables, enables for the clock triggers, and reset.
  - DMA Controller: Contains enables for the four peripheral request interfaces.
  - GP Master AXI Interface: Enables/disables access to the GP Master AXI Interfaces to the PL.
  - GP Slave AXI Interface: Enables/disables access to the GP Slave AXI Interfaces from the PL.
  - HP Slave AXI Interface: Enables/disables access to the 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 (for Zynq-7000 SoC only)
- I/O Configuration (MIO Configuration for Zynq-7000 SoC)

- Clock Configuration
- DDR Configuration
- SMC Timing Calculation (for Zynq-7000 SoC only)
- Interrupts (Zynq UltraScale+ has this setting under PS-PL Configuration)
- PS-PL Configuration

## Customizing the Zynq UltraScale+ MPSoC PS

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

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

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

-- OR --

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

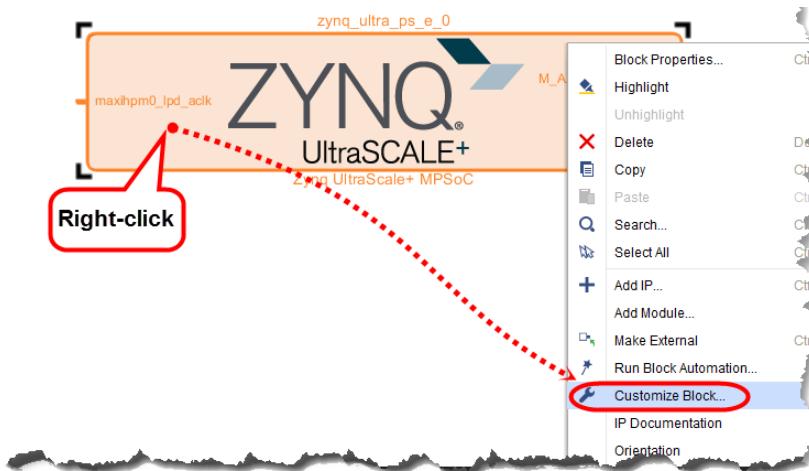


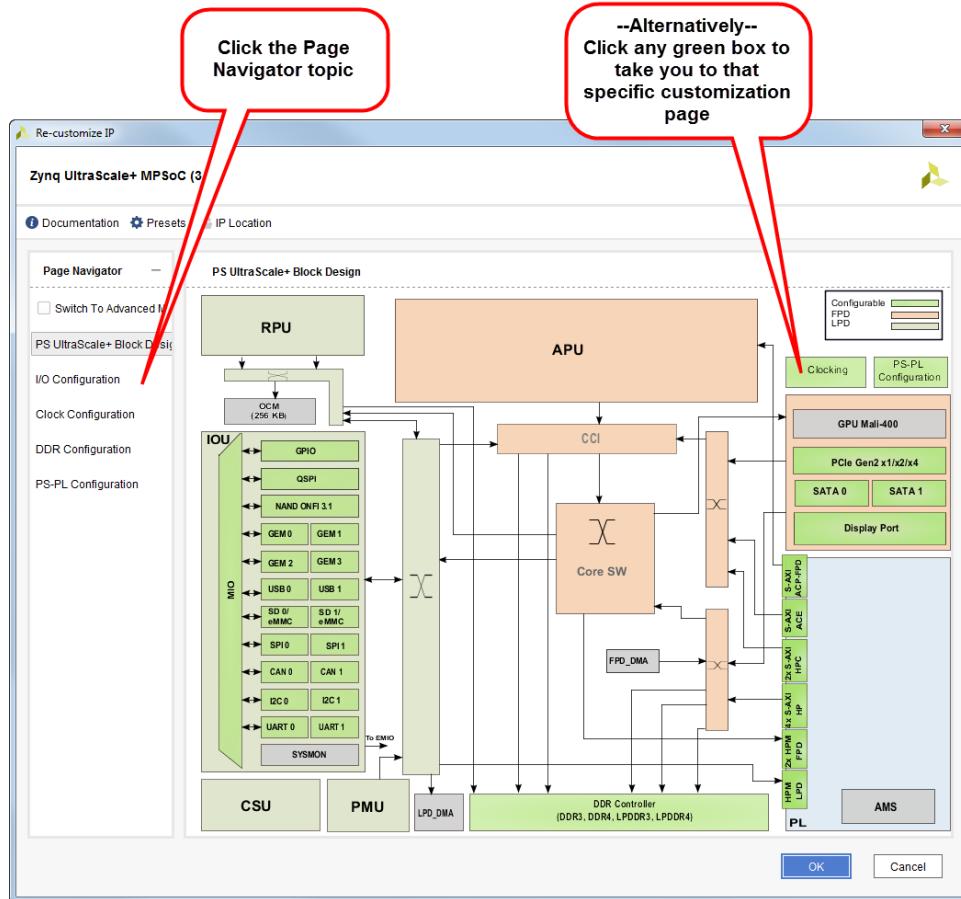
Figure 75: Customizing the Zynq UltraScale+ MPSoC

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

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

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

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



**Figure 76: Re-customize IP Dialog Box**

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

## Selecting a Zynq7 PS Preset Template

- 1-1. Configure the ZYNQ7 processing system using the Preset template.
  - 1-1-1. Double-click the **ZYNQ7 Processing System** IP block to open the Re-customize IP dialog box.
  - 1-1-2. Click **Presets** in the Re-customize IP dialog box.

This preset contains settings appropriate for your board, such as peripheral pin locations, DDR memory parameters, etc.
  - 1-1-3. Select the preset for your board.

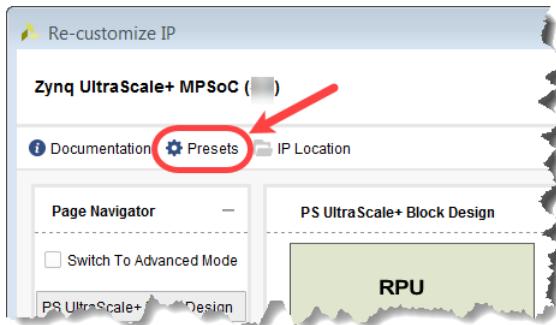


Figure 77: Selecting a Preset Template

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

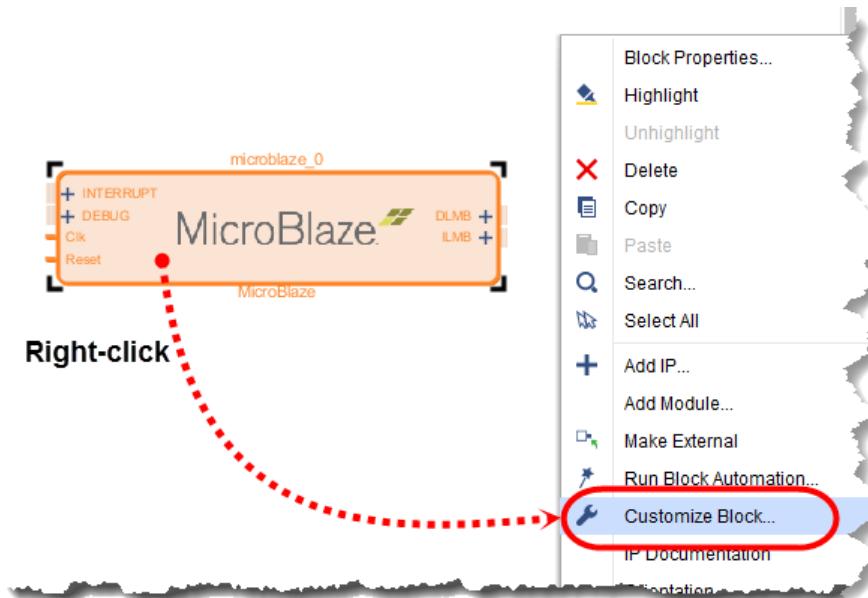
## Customizing the MicroBlaze Processor

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

The Typical configuration is one of several customizable configurations. It includes an MDM interface for debugging, instruction and data caches for use with DDR memory, and exception/interrupt handling. This

configuration provides a balance between frequency, area, and overall performance.

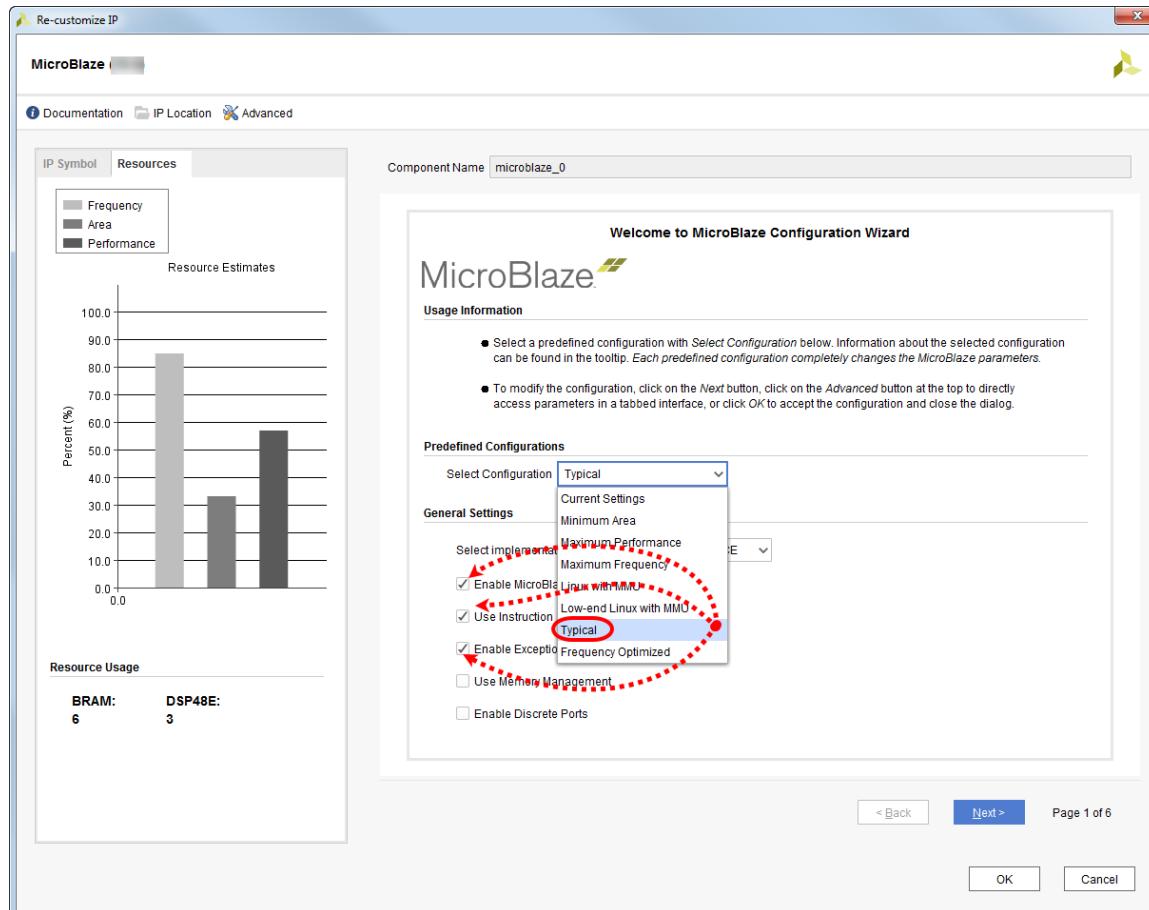
- 1-1-1. Double-click the **MicroBlaze** IP block or right-click the **MicroBlaze** IP block and select **Customize IP**.



**Figure 78: Starting Customization on the MicroBlaze Processor**

The Re-customization IP dialog box opens.

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



**Figure 79: Selecting the Typical Configuration for the MicroBlaze Processor**

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

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

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

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

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

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

**1-1-6.** If selected, deselect the **Use Instruction and Data Caches** option.

- 1-1-7.** Ensure that the **Enable MicroBlaze Debug Module Interface** and **Enable Exceptions** options are selected.

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

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

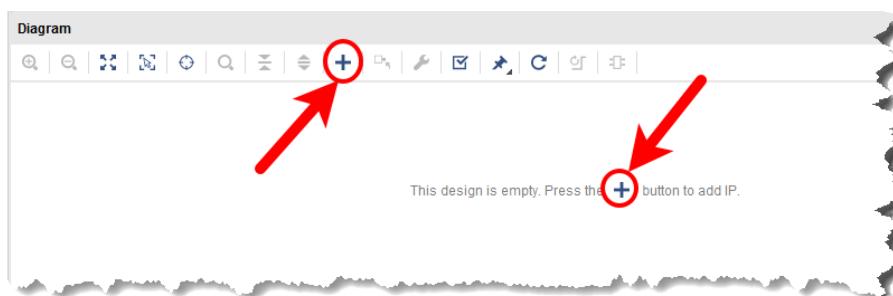
- 1-1-8.** Click **OK** to save this configuration.

## Adding IP to an IP Integrator Block Design

### 1-1. Open the IP catalog.

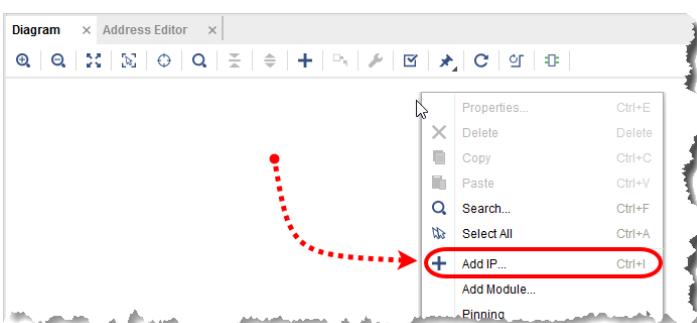
- 1-1-1.** The IP catalog can be opened in several ways:

- If the design is empty, add IP by clicking either '+' icon.



**Figure 80: Opening the IP Catalog from the Initial Information Bar Display**

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



**Figure 81: Opening the IP Catalog by Right-Clicking the Workspace**

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

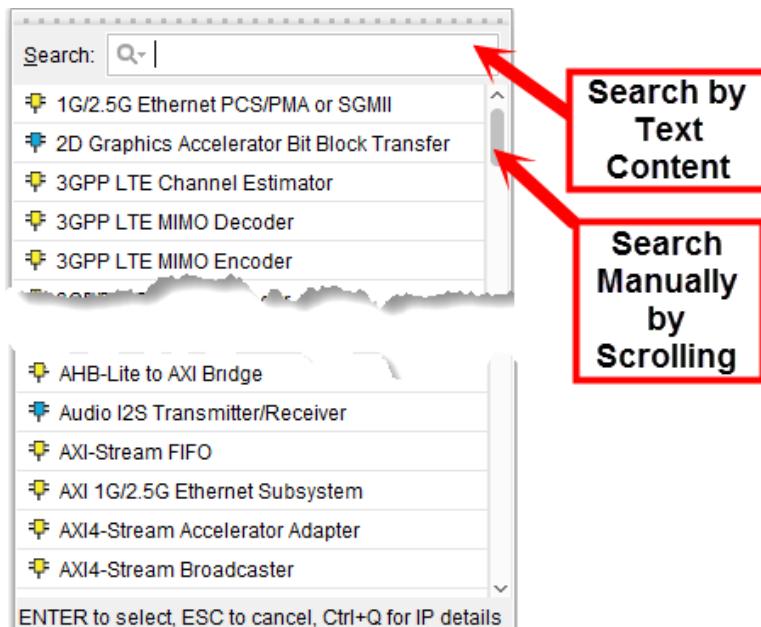
**Important Note:** Using Windows > IP Catalog will add the new IP to the top level of the design's hierarchy—it will NOT add the IP to the diagram! It is, however, possible to float this window and drag-and-drop IP into the Block Design canvas.

**1-2. Once the IP catalog opens, search for *the desired piece of IP* and add it to the canvas.**

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

- 1-2-2.** Locate **the desired piece of IP**.

**Hint:** You can scroll with the mouse or use the scroll bar on the right side of the list. The more of the IP name that you provide, the fewer items that you will need to manually scroll through.



**Figure 82: Two Mechanisms to Search for IP**

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

or

Drag-and-drop the IP into the workspace.

The IP catalog remains open once the IP has been added to the design. This is a convenient way to quickly add multiple pieces of IP. Pressing the <Esc> key or clicking an open space in the canvas will close the IP catalog.

## Customizing IP

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

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

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

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

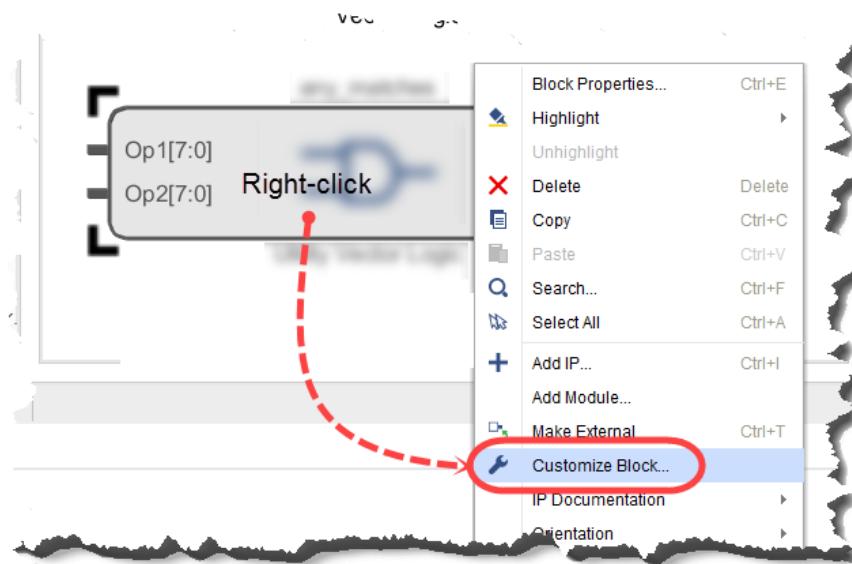


Figure 83: Opening the Recustomization Wizard for a Piece of IP

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

## Making Manual Connections Between Two Objects in the IP Integrator

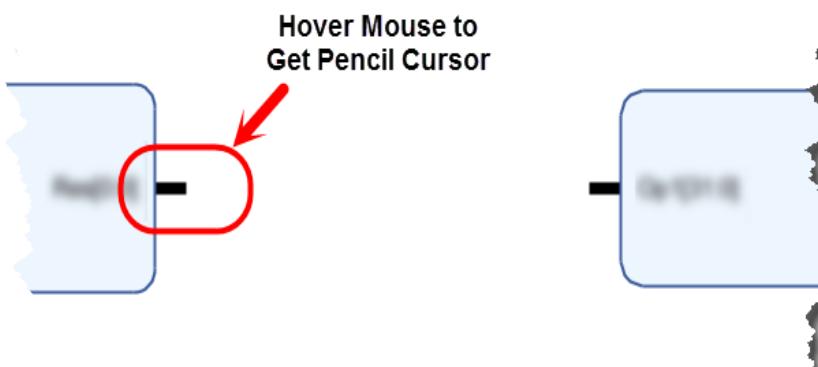
Making connections is a simple process in the IP integrator tool. Hover over a port or interface on a block with your mouse (the cursor changes to a pencil icon from the arrow icon). You can then click-drag the connection to a legal port or interface. As the connecting wire is stretched to its destination,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

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

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

A "net" is another name for the wire, signal, or connection between two or more pins.

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



**Figure 84: Hovering the Mouse Over a Port to Get the Pencil Icon**

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

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

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



**Figure 85: Finding Legal Connection Points**

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



**Figure 86: Finishing the Connection**

Interfaces are connected in exactly the same way, except that interfaces are comprised of a bundle of signals rather than a single wire. The tools are designed to automatically make the proper connections between two compatible interfaces.

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

## Renaming an IP Block

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

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

- 1-1-1. Click the desired IP block in the Block Design window to open its Block Properties dialog box (1).

The current name of the selected IP block appears in the Block Properties window when the General tab (located towards the bottom left of the window) is selected.

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

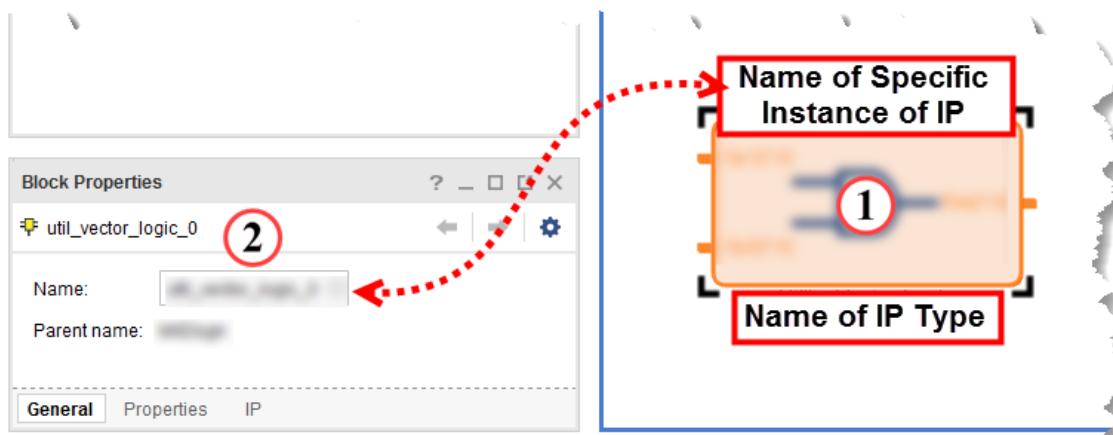


Figure 87: Renaming an IP Block

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

## Adding a Port to an IP Integrator Block Diagram

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

| Port Type    | Meaning  |
|--------------|--|
| Clock        | Clock signals: placed on clock networks; enter the frequency in the Frequency field                              |
| Reset        | Reset network  |
| Interrupt    | For embedded systems: ports marked as interrupt are entered into the interrupt select table for the processor(s) |
| Data         | General-purpose data signals   |
| Clock Enable | Clock enable signals: used with clocks   |
| Other        | Undefined signals: cannot be connected to ports other than those marked as "other"                               |

### 1-1. Create a port.

1-1-1. Right-click an unoccupied section of the Vivado IP integrator canvas (1).

1-1-2. Select **Create Port** (2).

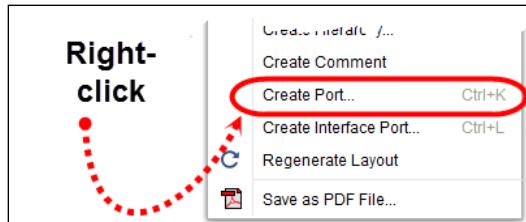


Figure 88: Creating a New Port

The Create Port dialog box opens.

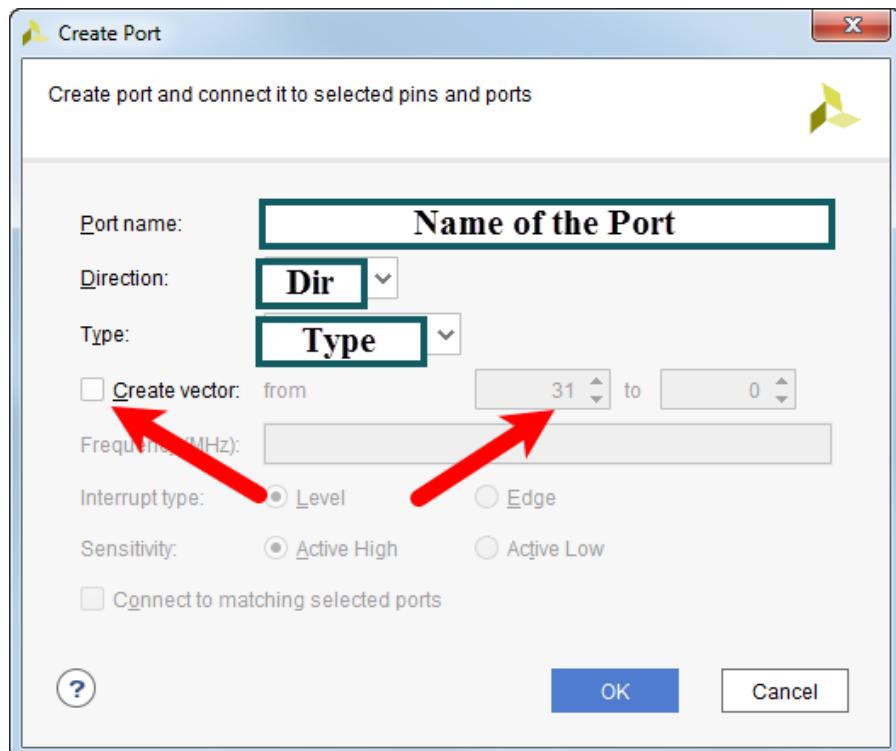


Figure 89: Create Port Dialog Box

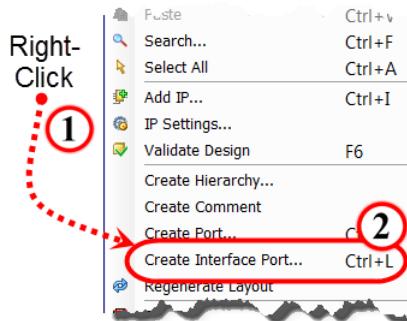
- 1-1-3. Enter the **name of the port** in the Port name field.
- 1-1-4. Select the port direction as **input, output, or inout/bidirectional**.
- 1-1-5. Select the type to be **according to the table below**.
- 1-1-6. Ensure that the **Create vector** box is checked and If this port is more than one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

## Adding an Interface Port to an IP Integrator Block Design

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

### 1-1. Create an interface port.

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



**Figure 90: Creating an Interface Port**

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

The Create Interface Port dialog box opens.

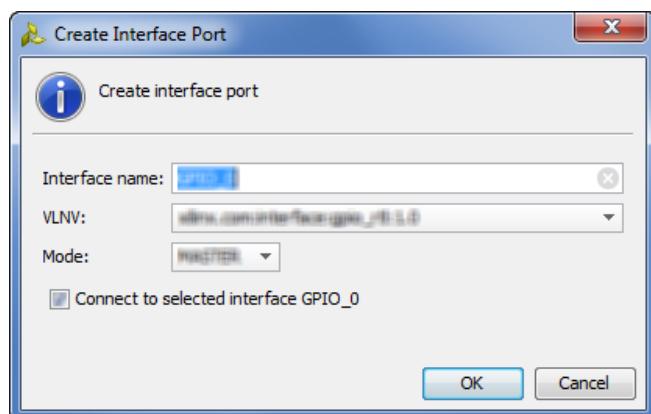
- 1-1-3. Enter the name of the interface port.

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

VLVN stands for vendor, library, name, and version. Typically this option is populated by the tool.

- 1-1-5. Select if the interface port is a master, slave, or monitor.

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



**Figure 91: Create Interface Port Dialog Box**

- 1-1-7. Click OK.

## Making a Pin or Interface External

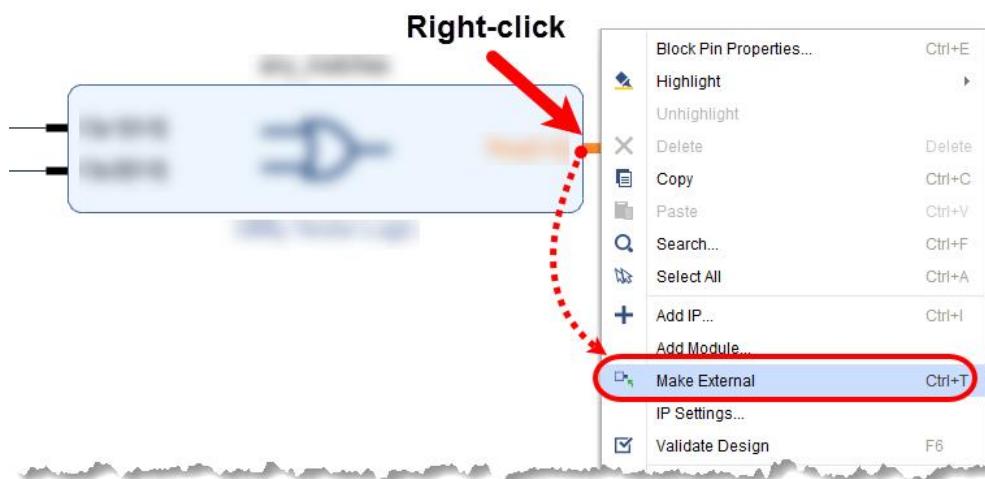
There are several mechanisms for making a pin or interface external. The easiest way is described here.

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

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

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

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



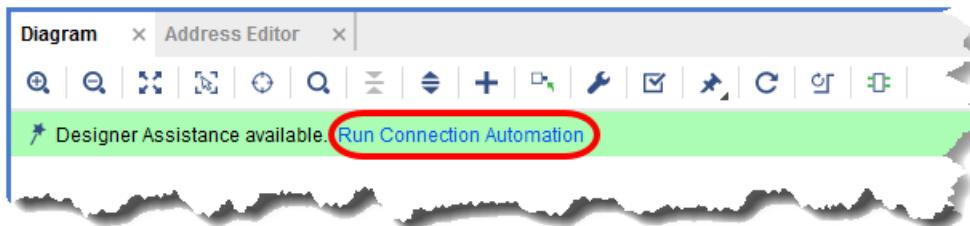
**Figure 92: Making a Port or Interface External**

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

## Running Connection Automation

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

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

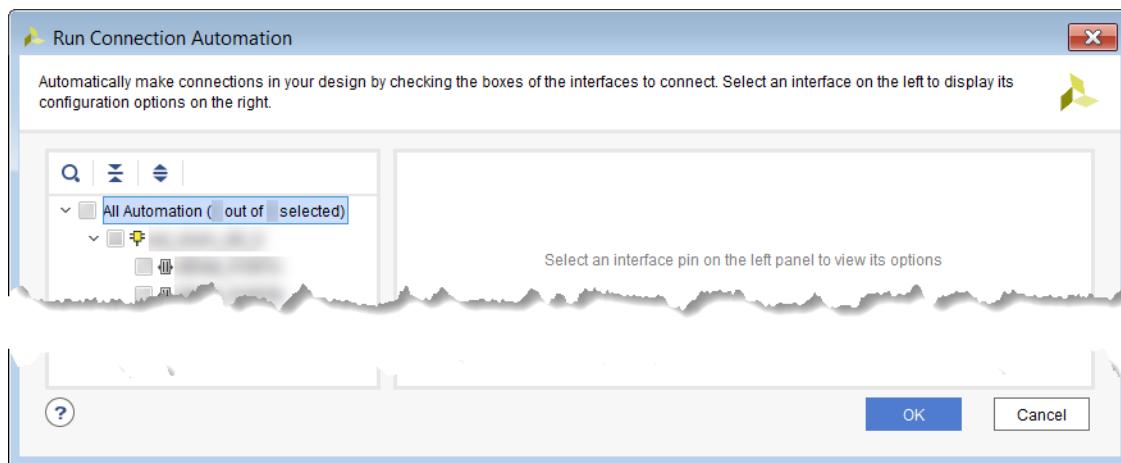


**Figure 93: Running Connection Automation**

This opens a Run Connection Automation dialog box listing all the IP blocks 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 an automation will be shown on the right whenever an interface node is selected on the left.

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

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



**Figure 94: Generic Run Connection Automation Wizard**

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

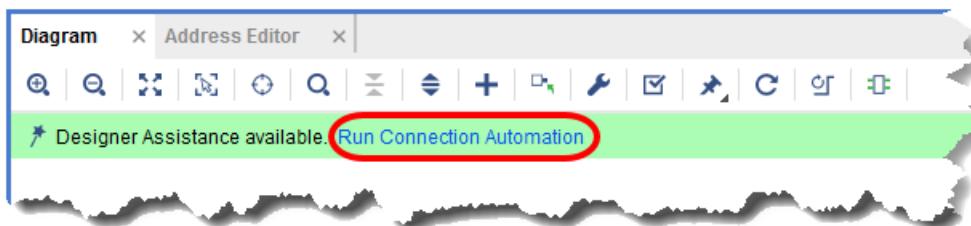
Connection Automation is fully capable of performing multiple connections—here a single connection was illustrated to show the basics of the capability.

## Running Connection Automation for Multiple Modules

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

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

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

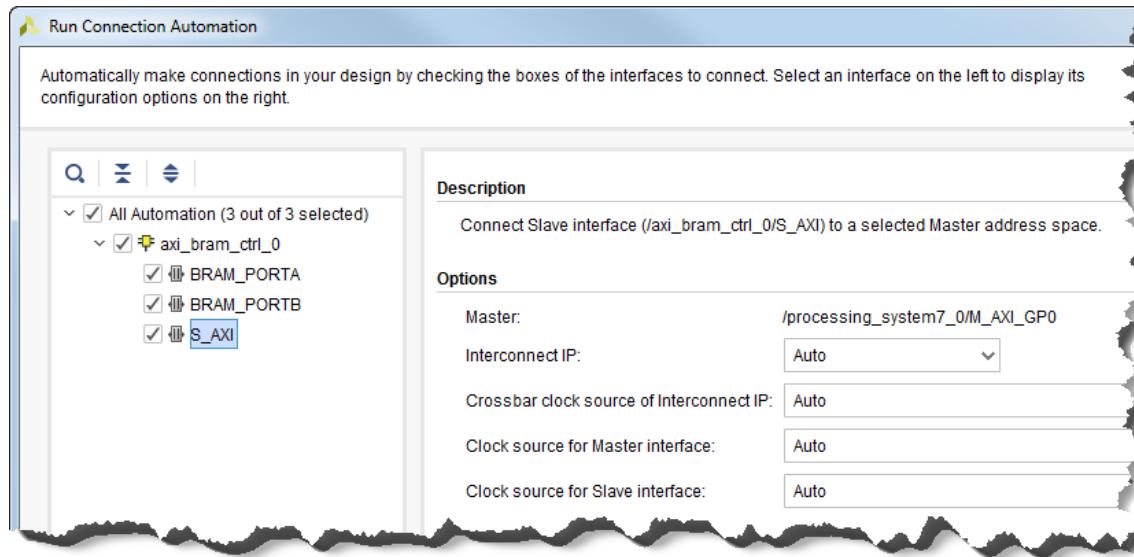


**Figure 95: Running Connection Automation**

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 ( $\equiv$ ) 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 96: Run Connection Automation Connecting the PS to MicroBlaze Processor (Zynq SoC)**

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

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

## Running Block Automation

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

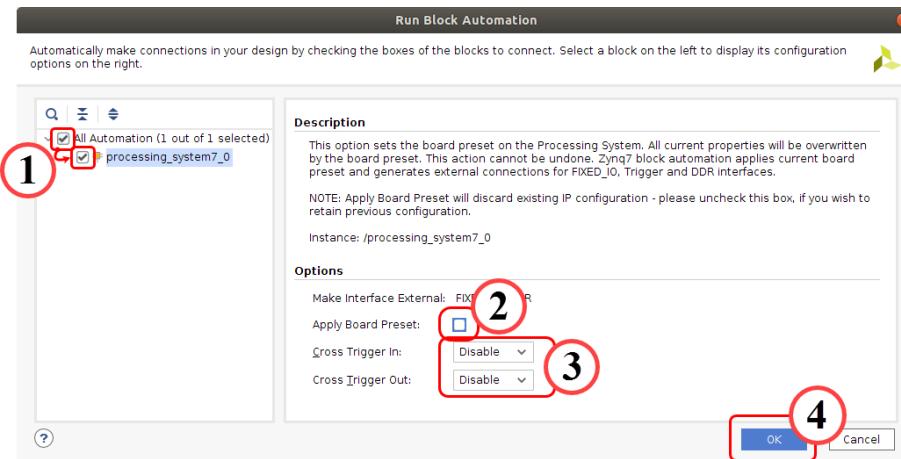
- 1-1-1. Click **Run Block Automation** from the Designer Assistance information bar.



**Figure 97: Designer Assistance - Block Automation (Run Connection Automation May be Absent)**

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

- 1-1-2. Click the **Expand All** icon () to ensure that the list of available automations is fully visible (1).
- 1-1-3. Select either the top-level **All Automation** check box or individual blocks as listed below.
  - Blocks: the desired piece of IP
- 1-1-4. Deselect **Apply Board Preset** when using Zynq devices, as this will undo any work that you have done in the processor's Re-customization Wizard (2).



**Figure 98: Run Block Automation Dialog Box**

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

## Locating Objects in the Board Design

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

### 1-1. Locate the object.

#### 1-1-1. Locate the **Design** tab next to the Sources tab.

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.

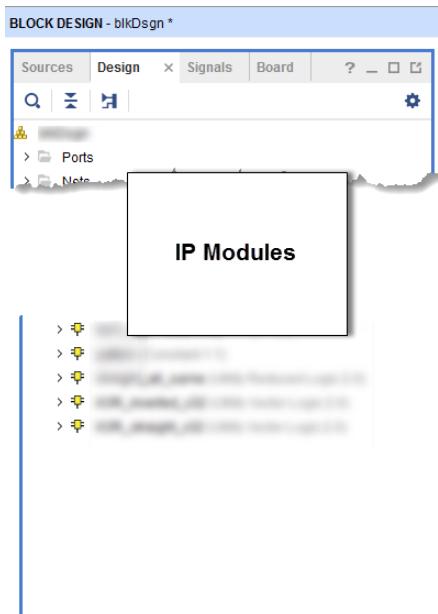
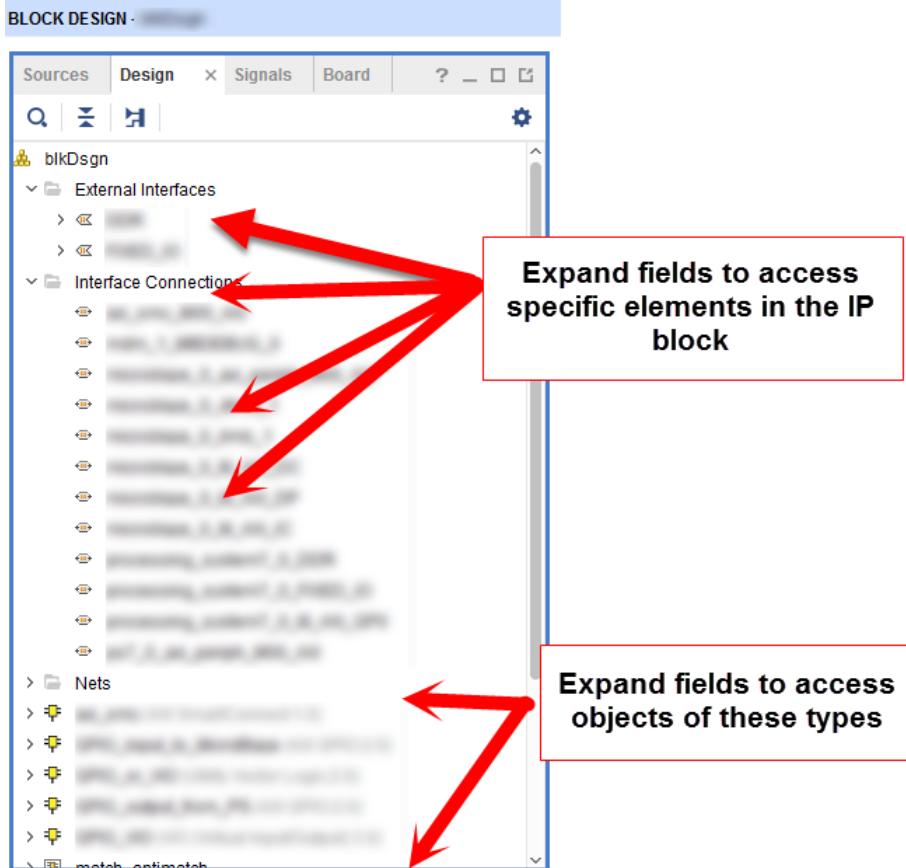


Figure 99: Locating the Design Hierarchy View

#### 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 100: Expanding Branches in the Design Hierarchy Tree**

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

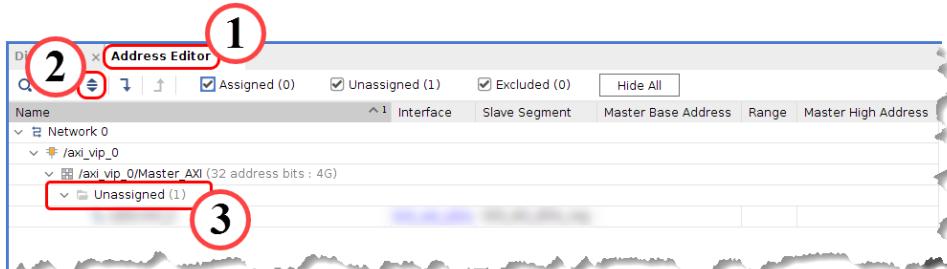
The Schematic view will highlight (select) that item.

## Mapping Peripheral Addresses

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

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

- 1-1-1. Select the **Address Editor** tab (1).
- 1-1-2. Click the **Expand** icon (2) to expand all the entries in the window (2).
- 1-1-3. Search for any field containing the text "Unmapped Slaves" (3).



**Figure 101: Verifying Peripheral Addresses**

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

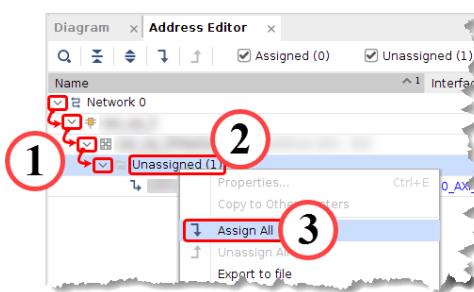
If there are unmapped devices, continue with the next instruction; otherwise skip the next instruction.

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

- 1-2-1. Expand the tree until you see the unmapped devices (1).
- 1-2-2. Right-click the peripheral that you want to assign an address to or select **Unassigned** to assign addresses to all unassigned elements (2).
- 1-2-3. Select **Assign Address** to assign addresses peripheral by peripheral.

-- OR --

Select **Assign All** to assign addresses to all peripherals that do not have an address assigned to them (3).



**Figure 102: Opening the Dialog Box for Assigning an Address**

## Generating Output Products

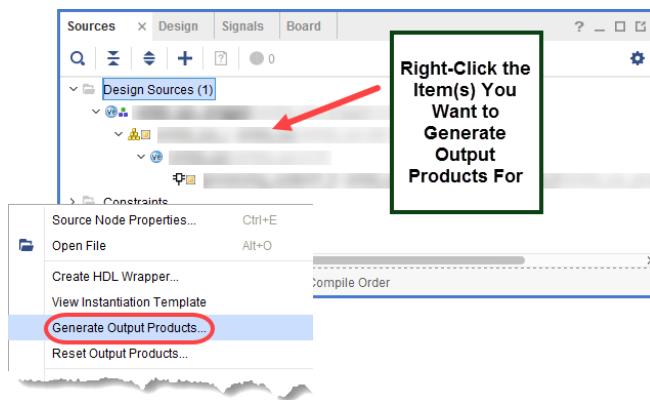
Once a block design is complete, its output products can be generated. The synthesis task automatically runs this phase; however, other operations require the output products, such as exporting a block design.

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

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

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

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



**Figure 103: Selecting the Items to Generate Output Products For**

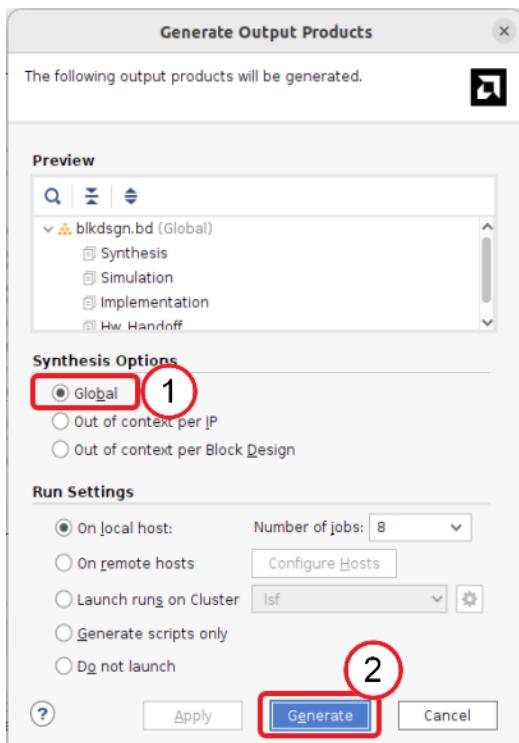
The Generate Output Products dialog box opens, listing which output products will be generated.

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

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

If you are changing the settings from out of context to global, click **Apply**.

**Note:** Out of context is helpful when developing large designs with many blocks. Since each block is synthesized independently, if that block is not changed, then it is not resynthesized, which can offer significant time savings. Contrast this with the Global synthesis option in which all aspects of the design will be resynthesized if any changes are detected.



**Figure 104: Managing the Output Products**

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

- 1-1-4.** Click **Generate** to start the generation process (2).

When generation completes, a success dialog box opens.



**Figure 105: Successful Output Products Generation**

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

## Creating a Hierarchical Block in a Block Design

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

### 1-1. Create and name a hierarchy block.

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

**These two tasks can be performed in either order:**

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

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

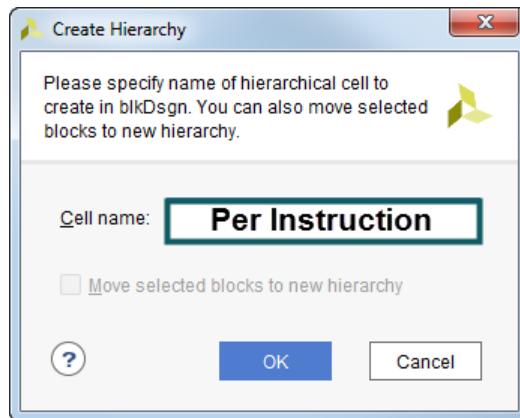
**The following describes how to create an empty hierarchy block.**

#### 1-1-1. Right-click in the block diagram.

#### 1-1-2. Select **Create Hierarchy** from the context menu.

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

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



**Figure 106: 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.

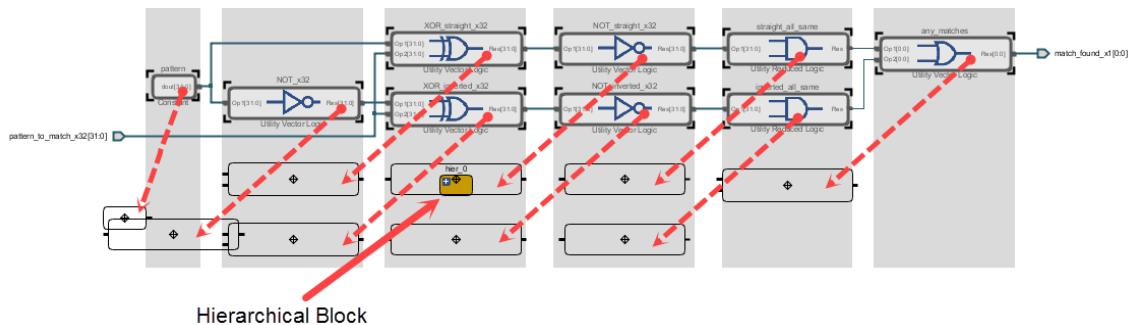
This can be done graphically by using <**Ctrl + click**> to select each piece of IP from the canvas as well as selecting the IP from the Design tab.

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 selected IP into the name of your hierarchy.

This is performed by pressing the <**Ctrl**> key and clicking (but not releasing) one of the IP blocks. Drag the selected item into the the name of your hierarchy by moving the cursor of the hierarchy block and then releasing the mouse button.

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 107: Example of Dragging IP into a Hierarchical Block**

Alternatively, before creating the hierarchical block, you can select the IP from the canvas or from the Design tab and then right-click the canvas to open the context menu and select **Create Hierarchy**. The hierarchical block will be created with the selected IPs added to it.

## 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 108: Expanding a Hierarchy Block**

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

## Opening a Hierarchical Block in a New Tab

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

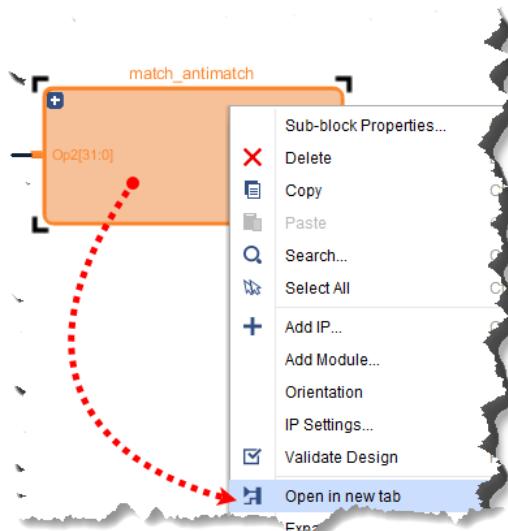
### 1-1. Open the name of your hierarchy in a new tab.

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

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

This can be done in one of two ways:

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



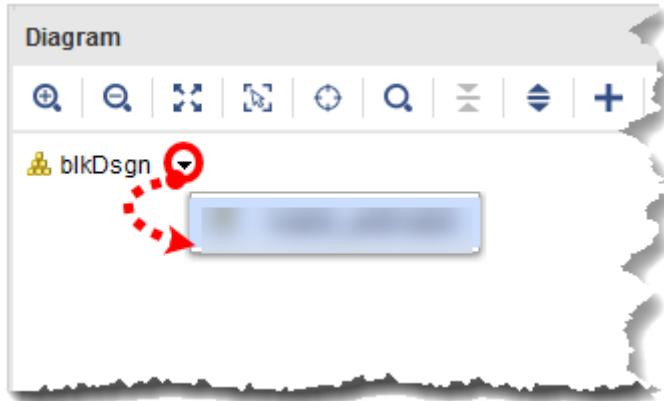
**Figure 109: Opening the Hierarchy Block in a New Tab**

- Alternatively, you can use the hierarchy browser, which is part of the Sources tab. When the block design is expanded, the individual pieces of IP are shown. Similarly, the Design tab divides the canvas into ports, nets, IP, and hierarchies. Each hierarchy is further sub-divided into ports, nets, IP, and other hierarchies.

- 1-1-2. From the Design tab (next to the Sources tab), double-click the **match\_antimatch** entry to both expand the ports, nets, and IP within this hierarchy as well as open a new tab with only the contents of the hierarchy present.
- 1-1-3. Select **the name of your hierarchy** from the list of hierarchies to open the hierarchy in a new tab.

**Note:** If the block design hierarchy list is not as shown in the figure, click the **Settings** symbol in the extreme top-right corner. Go to the General section and select the **Show**

**Hierarchy** option to enable this capability. The hierarchy can be opened from this location as well as the Design tab.



**Figure 110: Access the Hierarchical Blocks in a Block Diagram**

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

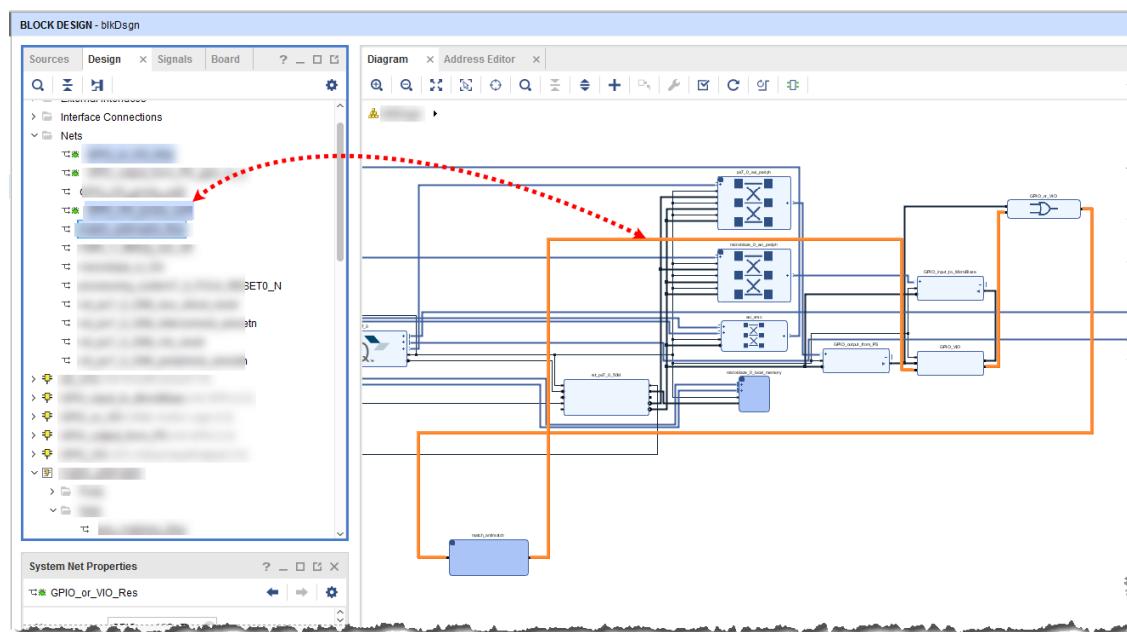
## Marking Signals for Debugging

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.

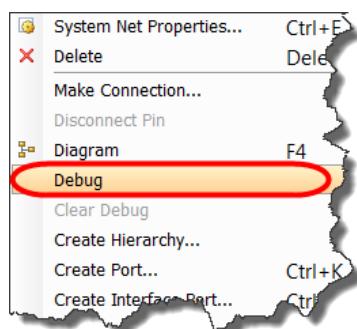
You can select them from the canvas or Design list window.



**Figure 111: Selecting Nets**

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

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



**Figure 112: Marking the Nets for Debugging**

## Updating IP

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

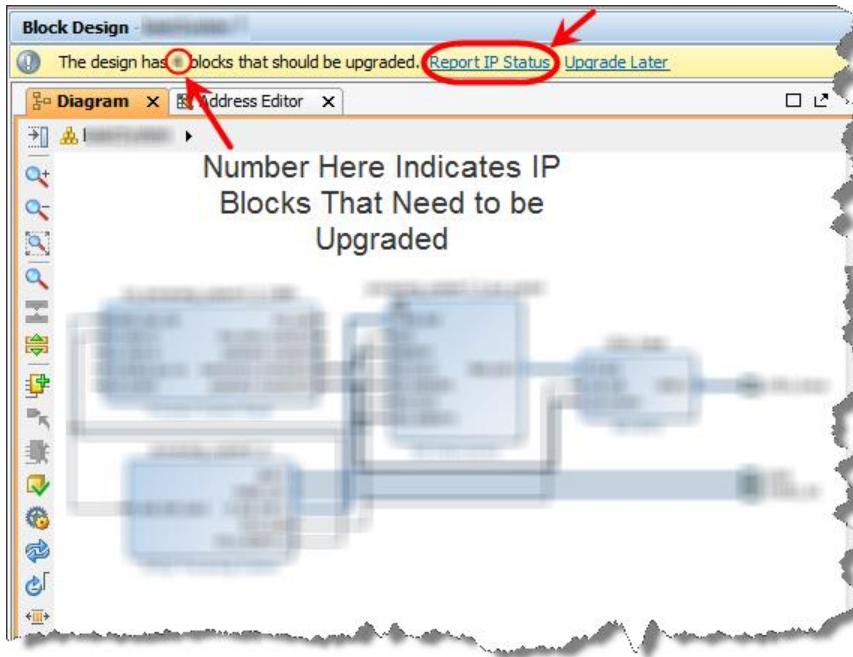
When older designs are brought into a newer version of the Vivado Design Suite, there is often a mismatch between the IP versions in the design and those that are available in the newer version of the tools. These discrepancies are reported in both a pop-up dialog box as well as the Report IP Status 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 113: 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.

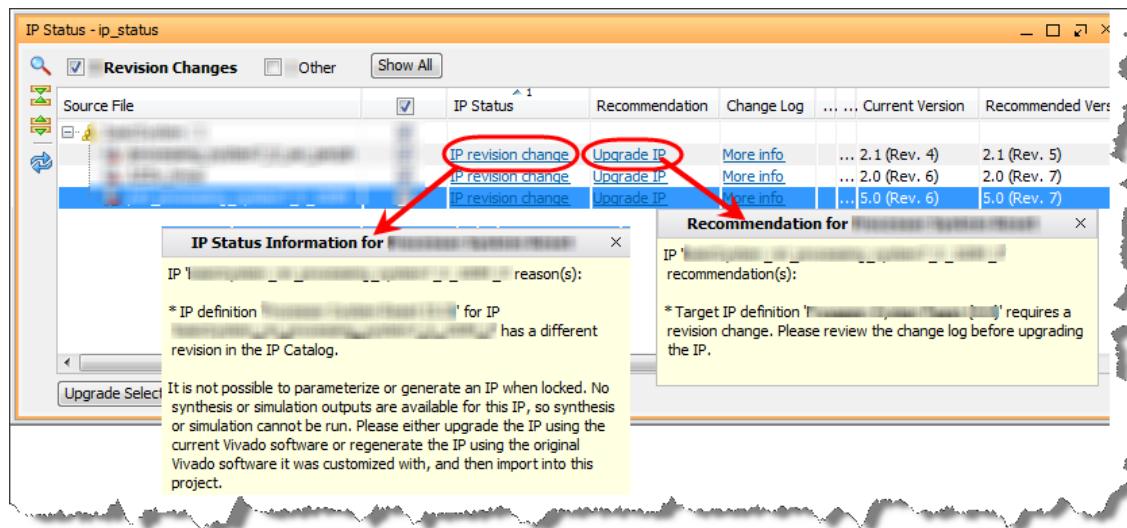


Figure 114: IP Status and Recommendation Columns IP Status Report

## 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).

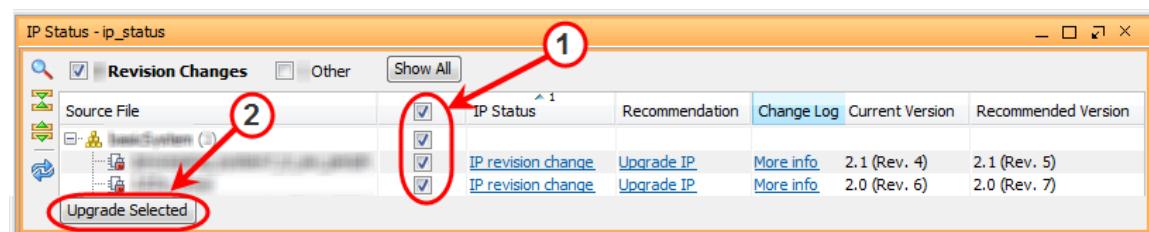


Figure 115: IP Status Report

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

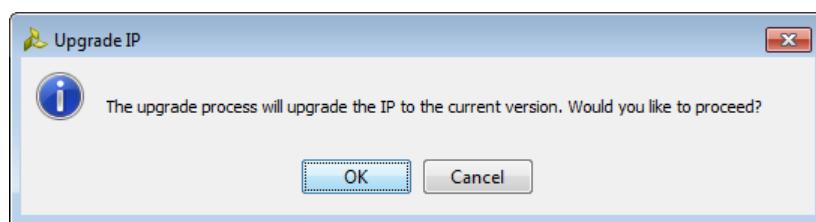


Figure 116: Upgrade IP Dialog Box

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

## Regenerating the Layout

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

### 1-1. Regenerate the layout.

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

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

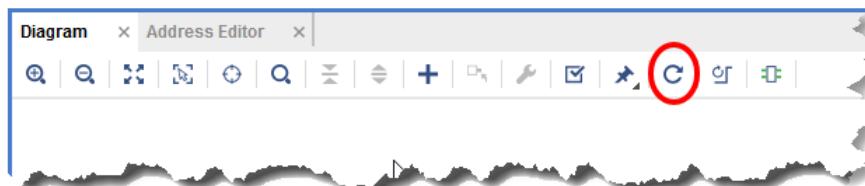


Figure 117: Regenerating the Layout

## Running a Design Rule Check/Design Validation

Each time an IP is connected, a suite of partial design rule checks automatically runs. Design validation is a more comprehensive (but not exhaustive) block design-wide verification and must be run manually.

### 1-1. Run a manual design validation.

- 1-1-1. Select **Tools > Validate Design** or press **<F6>**.

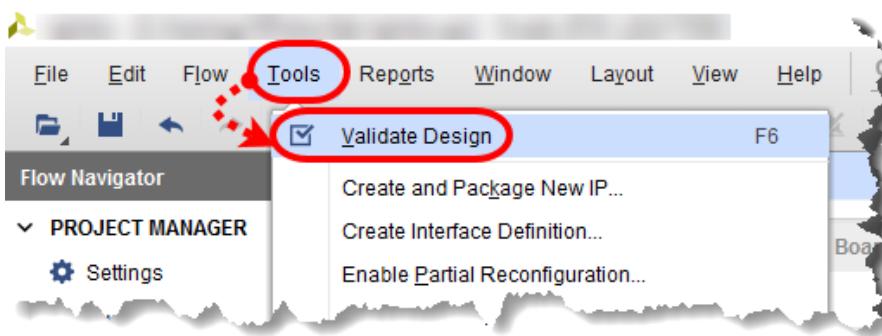


Figure 118: Validating the Design

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

Because many 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 results by selecting or deselecting the types of messages you want to see:  
 Error, Infos, or Status messages.

The display is automatically updated.

Hyperlinks are provided to help you quickly locate the issue.



**Figure 119: Viewing the Results of the Design Validation**

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

## Validating the Block Design

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

- 1-1-1. Select **Tools > Validate Design** or press **<F6>**.

Any issues are displayed in the Console window.

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

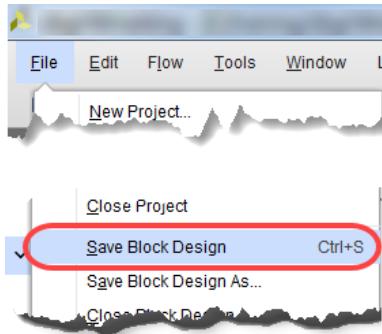
## Saving the Block Design

---

### 1-1. Save the block design.

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

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



**Figure 120: Saving the Block Design**

## Generating a Wrapper for a Block Design

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

Typically, once the design is complete, an HDL wrapper is created that instantiates the design and provides a platform for adding other logic. This wrapper can be automatically generated (and optionally terminated) by the tools, or it can be added manually by the user.

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

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

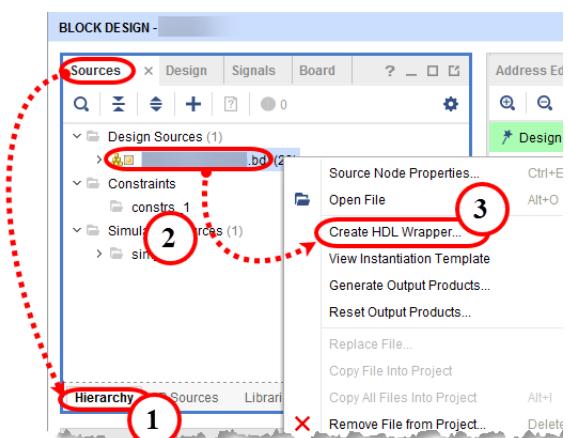
While the block diagram can be found under any of the tabs found at the bottom of the pane (Hierarchy, IP Sources, Libraries, or Compile Order), it is shown here from the Hierarchy tab as this is a common practice.

- 1-1-2. Using the Sources > Hierarchy tab, right-click the block design under the Design Sources node (2).

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

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

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



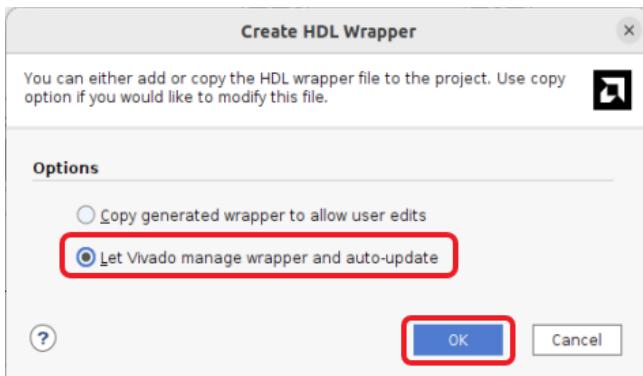
**Figure 121: Selecting Create HDL Wrapper**

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 fully described by one or more block designs.

**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 122: Generating the Wrapper and Copying the Generated File into the Project**

This will produce a wrapper file in the target 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.

**Note:** Many simulations require Verilog. If this is necessary for your simulation tool, select **Project Manager > Settings > Target Language** to cause the tools to generate all sources in Verilog, including the wrapper.

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

## Vivado Elaborated Design Operations

### In This Section

---

|   |    |
|---|----|
| Opening the Elaborated Design .....                         | 93 |
| Creating a Schematic on an Elaborated Design .....          | 93 |
| Running UltraFast DRC Checks on the Elaborated Design ..... | 93 |
| Running DRC Checks on the Elaborated Design .....           | 94 |
| Closing the Elaborated Design .....                         | 94 |

## Opening the Elaborated Design

---

### 1-1. Open the elaborated design.

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

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

The Elaborate Design dialog box opens.

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

## Creating a Schematic on an Elaborated Design

---

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

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

The RTL Schematic opens in the main workspace area.

## Running UltraFast DRC Checks on the Elaborated Design

---

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

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

The Report Methodology dialog box opens.

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

## Running DRC Checks on the Elaborated Design

---

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

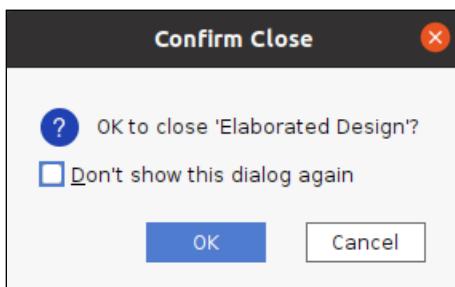
- 1-1-1. Click **Report DRC** under RTL Analysis > Open Elaborated Design in the Flow Navigator.  
The Report DRC dialog box opens.
- 1-1-2. Select **default** in the Vivado Rule Decks section.
- 1-1-3. Observe the categories in the **Rules** section.
- 1-1-4. Click **OK** to generate the DRC report.

## Closing the Elaborated Design

---

### 1-1. Close the elaborated design.

- 1-1-1. Select **File > Close Elaborated Design** to close the elaborated design.  
The Confirm Close dialog box opens.



**Figure 123: Confirm Close Dialog Box**

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

## Vivado Simulation Operations

### In This Section

---

|                                     |    |
|-------------------------------------|----|
| Running Behavioral Simulation ..... | 95 |
| Running Timing Simulation .....     | 95 |
| Accessing Simulation Settings.....  | 96 |
| Closing Vivado Simulation .....     | 96 |

## Running Behavioral Simulation

---

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

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

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

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

## Running Timing Simulation

---

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

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

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

## Accessing Simulation Settings

### 1-1. Open the simulation settings.

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

The simulation settings open. There are five sub-tabs: Compilation, Elaboration, Simulation, Netlist, and Advanced.

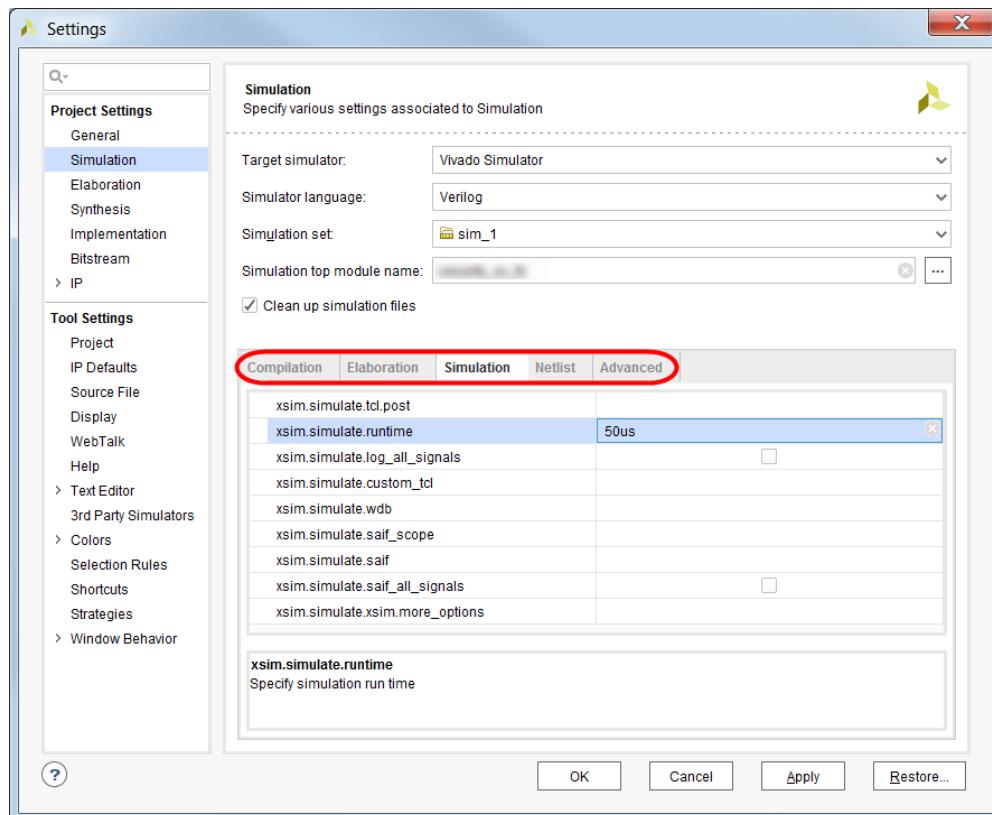


Figure 124: Vivado Simulator Settings

## Closing Vivado Simulation

### 1-1. Close the simulation.

- 1-1-1. Select **Simulation** in the Flow Navigator, then right-click and select **Close Simulation**.  
 1-1-2. Click **OK** to close the simulation.

## Vivado Synthesis Operations

### In This Section

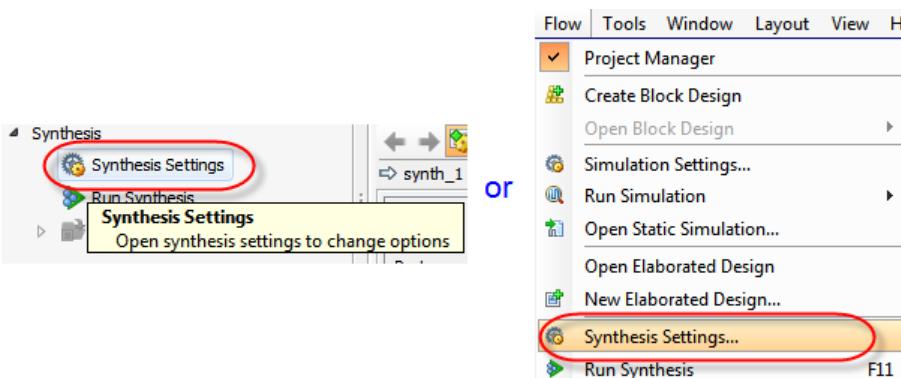
|   |     |
|---|-----|
| Setting Synthesis Options .....                                       | 97  |
| Running Synthesis .....   | 99  |
| Opening the Synthesized Design.....                                   | 100 |
| Reloading the Synthesized Design .....                                | 101 |
| Generating a Utilization Report.....                                  | 101 |
| Generating a Clocks Networks Report.....                              | 102 |
| Running a DRC Report on the Synthesized Design .....                  | 102 |
| Opening the Timing Constraints Window after Synthesis .....           | 102 |
| Generating a Timing Summary Report on the Synthesized Design.....     | 102 |
| Running Simultaneous Switching Noise (SSN) Analysis .....             | 104 |
| Generating a Clock Interaction Report on the Synthesized Design ..... | 104 |
| Closing the Synthesized Design .....                                  | 104 |

## 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 125: Selecting Synthesis Settings**

The Synthesis Project Settings dialog box opens.

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

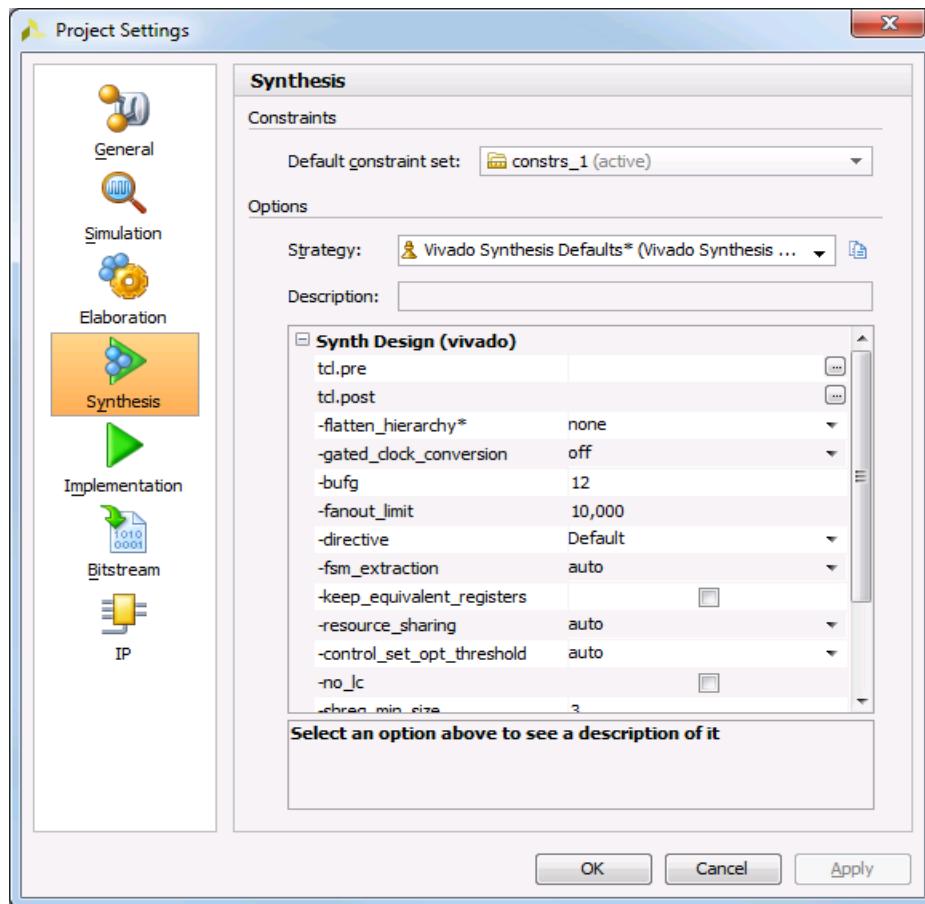


Figure 126: Synthesis Project Setting Dialog Box

- 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**.

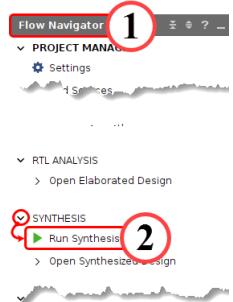
## Running Synthesis

### 1-1. Run synthesis.

1-1-1. Expand **Synthesis** in the Flow Navigator (1).

1-1-2. Click **Run Synthesis** (2).

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



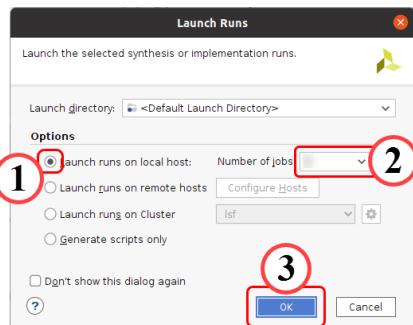
**Figure 127: Running Synthesis**

The Launch Runs dialog box opens.

1-1-3. Ensure that the **Launch runs on local host** option is selected (1).

1-1-4. Select the largest values from the *Number of jobs* drop-down list (2).

This allocates the number of processor cores recruited to run synthesis. Generally, the more cores, the faster synthesis completes.



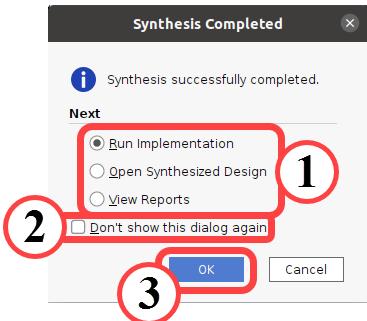
**Figure 128: Setting the Launch Run Configuration**

1-1-5. Click **OK** to launch the runs (3).

1-1-6. Click **Save** if you are asked to save your files.

Once synthesis completes, you are asked what task you want to perform next: run implementation, open the synthesized design, view reports, or none of the above.

- 1-1-7. Select **whatever process you want to perform next or Cancel to exit the dialog box (1).**
- 1-1-8. [Optional] If you are familiar with accessing these various capabilities, you can disable this dialog box from appearing again by selecting **Don't show this dialog again (2).**



**Figure 129: Selecting Post-Synthesis Options**

- 1-1-9. Click **OK** to take the action you just selected (3) or click **Cancel** to simply close the dialog box.

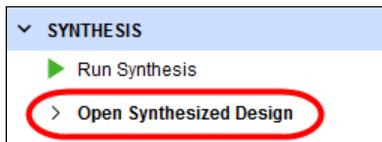
## Opening the Synthesized Design

---

### 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 130: 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.

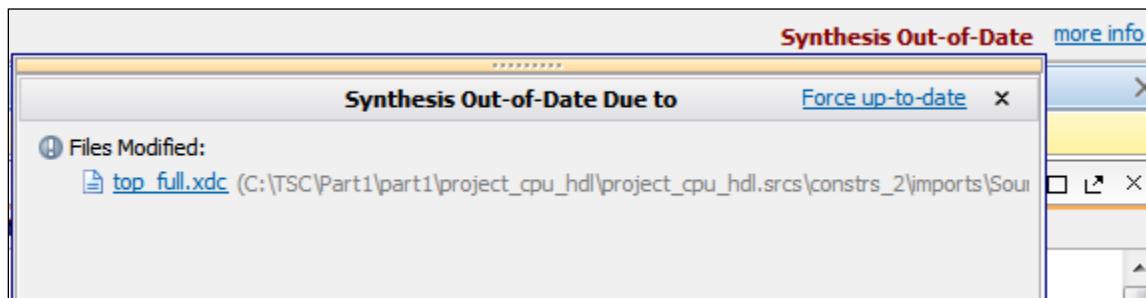


Figure 131: Force-up-to-date Link

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

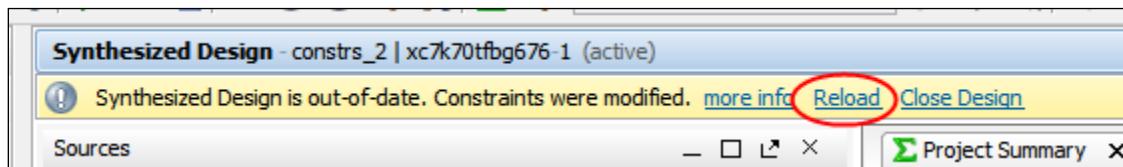


Figure 132: Reload Link

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

## Generating a Utilization Report

### 1-1. Generate a Utilization report.

- 1-1-1. Click **Report Utilization** under Synthesized Design in the Flow Navigator.  
Alternatively, you can select **Reports > Report Utilization**.

## Generating a Clocks Networks Report

---

### 1-1. Generate a Clocks Networks report.

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

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

The Report Clock Networks dialog box opens.

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

## Running a DRC Report on the Synthesized Design

---

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

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

The Report DRC dialog box opens.

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

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

## Opening the Timing Constraints Window after Synthesis

---

### 1-1. Open the Timing Constraints window.

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

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

The Timing Constraints window opens in the main workspace area.

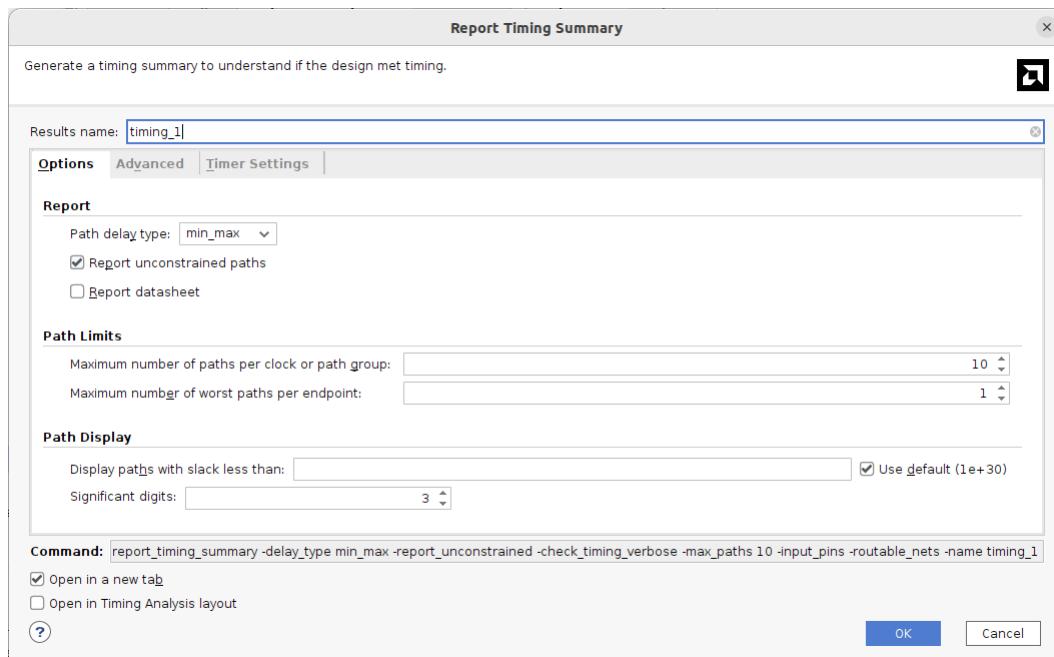
## Generating a Timing Summary Report on the Synthesized Design

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

### 1-1. Generate a Timing Summary report.

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

The Report Timing Summary dialog box opens.



**Figure 133: 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.

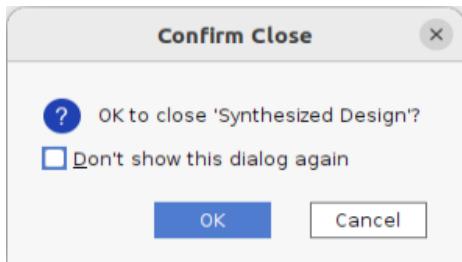
The Clock Interaction Report opens in GUI mode.

## 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 134: 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

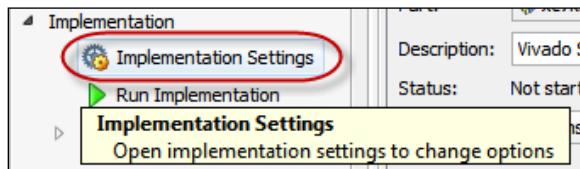
|   |     |
|---|-----|
| Setting Implementation Options .....                                  | 106 |
| Running Implementation .....  | 107 |
| Running All Tools – RTL to Bitstream .....                            | 108 |
| Generating a Utilization Report on the Implemented Design .....       | 110 |
| Opening the Implemented Design .....                                  | 111 |
| Generating a Timing Summary Report on the Implemented Design .....    | 111 |
| Generating a Clock Interaction Report on the Implemented Design ..... | 112 |
| Closing the Implemented Design .....                                  | 112 |
| Generating Clock Networks .....                                       | 113 |
| Generating a Power Report on an Implemented Design .....              | 113 |
| Generating the Bitstream .....  | 114 |

## Setting Implementation Options

### 1-1. Set the implementation options.

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

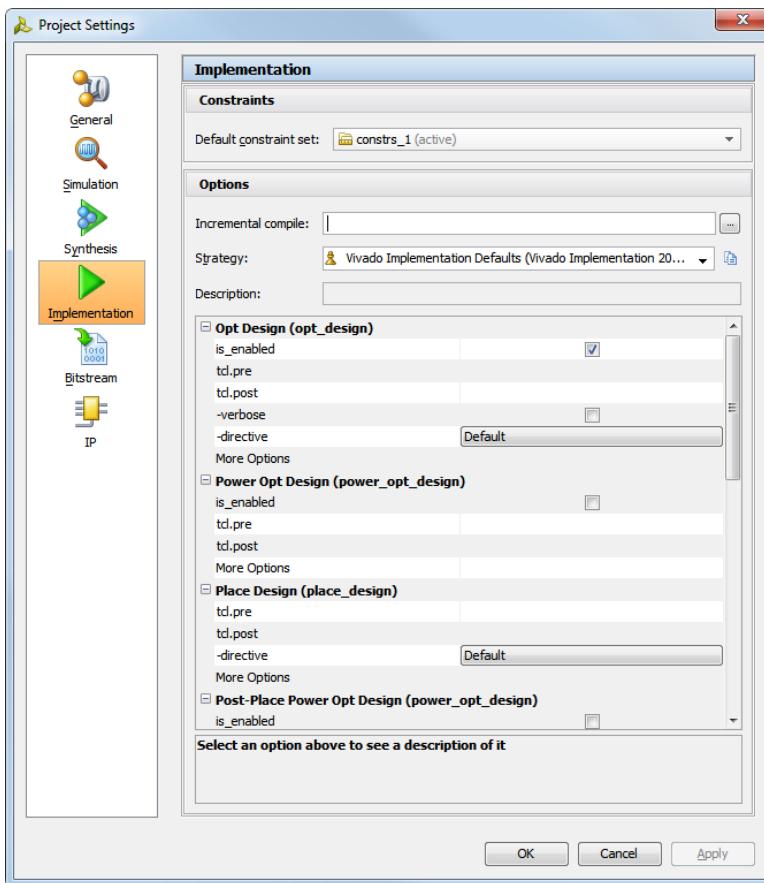
Alternatively, you can select **Flow > Implementation Settings**.



**Figure 135: Clicking Implementation Settings**

The Implementation Settings dialog box opens.

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



**Figure 136: Implementation Settings Dialog Box**

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

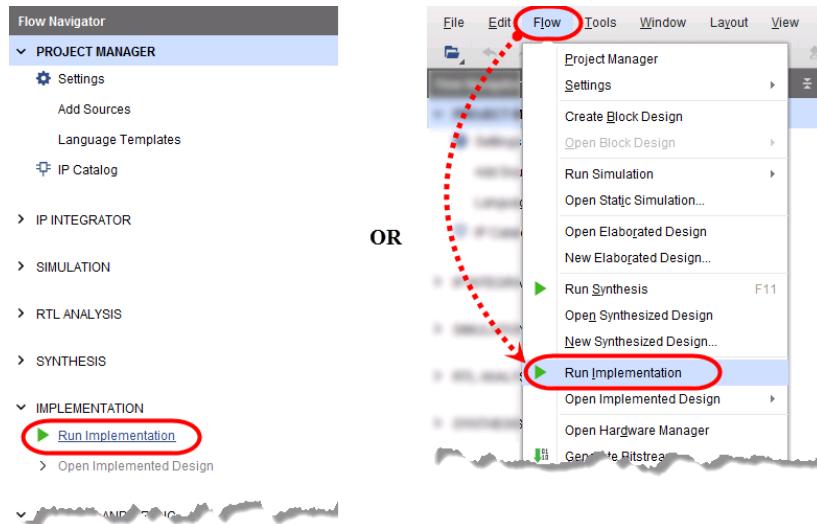
- 1-1-2. Click **OK**.

## 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 137: Selecting Run Implementation**

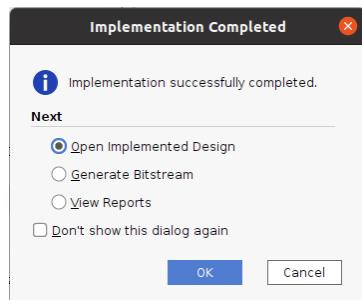
**Note:** If needed, click **OK** to launch synthesis first if prompted.

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

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

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

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



**Figure 138: Implementation Completed Dialog Box**

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

## Running All Tools – RTL to Bitstream

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

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

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

This tool 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**.

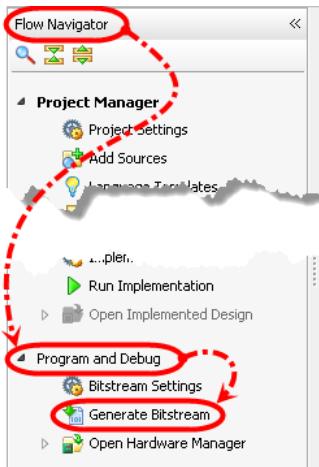
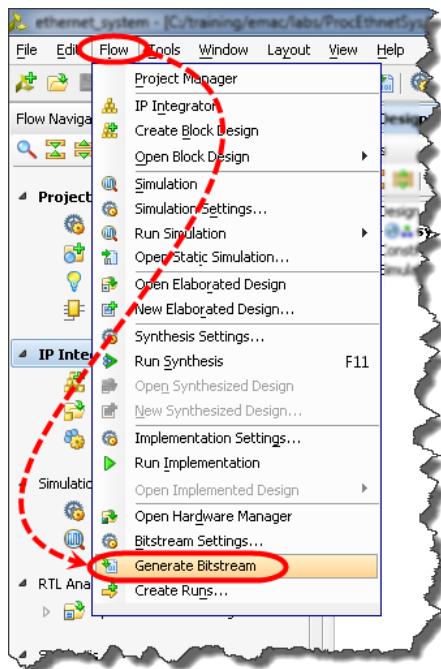


Figure 139: Generating the Bitstream

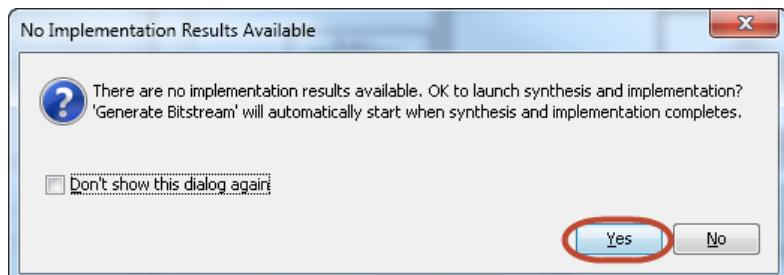
Alternatively, you can select **Flow > Generate Bitstream**.



**Figure 140: 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 141: 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 142: Bitstream Generation Completed Dialog Box**

## Generating a Utilization Report on the Implemented Design

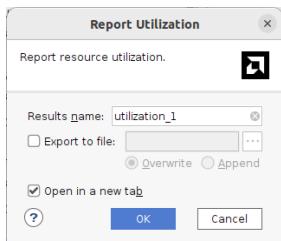
---

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

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

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

The Report Utilization dialog box opens.



**Figure 143: Report Utilization Dialog Box**

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

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

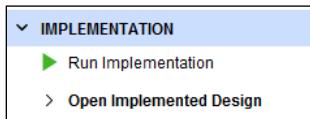
Note that the FPGA resources are listed in a hierarchical format for each RTL design module.

## 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 144: Opening the Implemented Design**

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

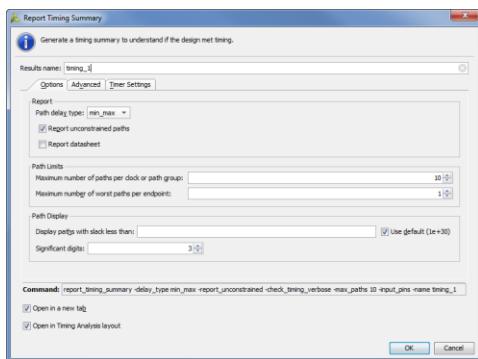
## Generating a Timing Summary Report on the Implemented Design

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 145: 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 on the Implemented Design

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.

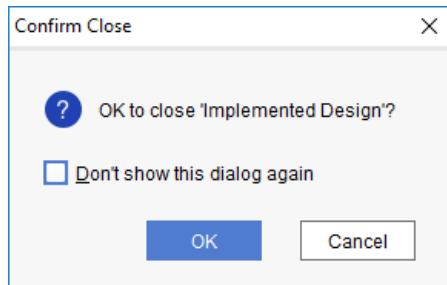


Figure 146: Confirm Close Dialog Box

- 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**.

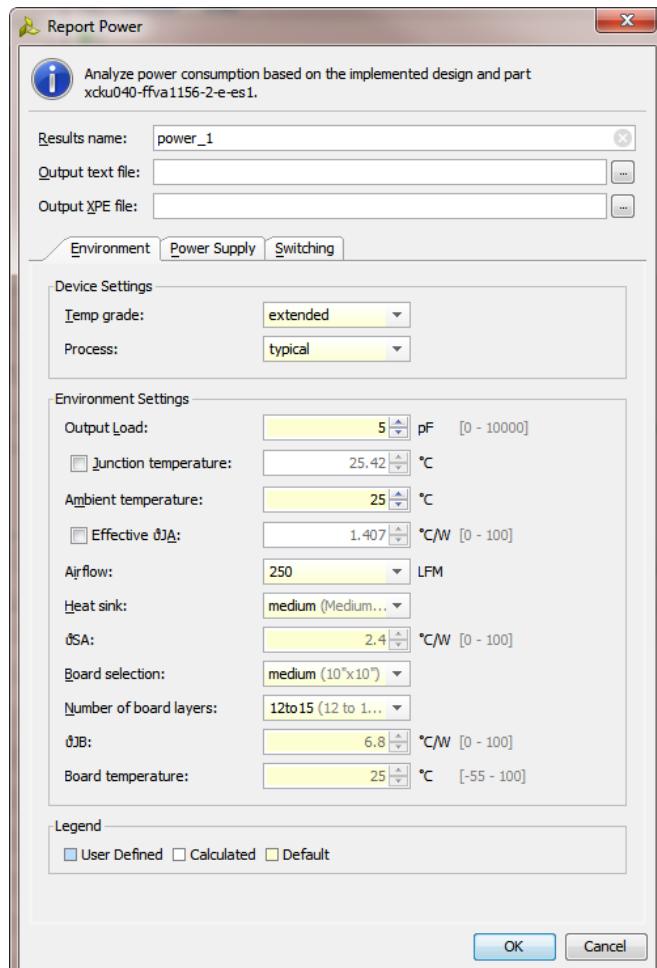
Alternatively, you can select **Tools > Report > Report Clock Networks**.

## 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 147: Report Power Dialog Box**

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

## Generating the Bitstream

For users that have a few hours, it is time to generate the bitstream. If you are running in class or on CloudShare, only review this and the next step.

**Note:** Do not perform this step as generating the bitstream will take approximately 3-4 hours depending on your system configuration. The instructions below are provided for review on what the necessary actions are.

For CloudShare users, if you are going to perform this step, you should do so in your local environment, not in CloudShare. This is because the memory available in CloudShare is 12 GB, and 16 GB is the minimum requirement. Generating the bitstream will still take approximately 3-4 hours.

### 1-1. Generate the bitstream.

- 1-1-1. Locate the **Generate Bitstream** entry under Program and Debug in the Flow Navigator.



Figure 148: Generate Bitstream in the Flow Navigator

- 1-1-2. Click **Generate Bitstream** to start the bitstream generation.

If neither synthesis nor implementation has been run, you will see a "No Implementation Results available" message.

- 1-1-3. Click **Yes** to continue the process as both synthesis and implementation will be run during the generation of the bitstream.

- 1-1-4. When the Launch Runs window opens, ensure that the **Launch runs on local host** option is selected.

- 1-1-5. Set the number of jobs to the highest number available.

This specifies how many CPU cores are allocated to this task.

- 1-1-6. Click **OK** to begin the bitstream generation process.

Track the progress using the status indicator in the upper-right corner of the workspace as well as in the design runs console. Successful completion of the bitstream is indicated with a message in the Console tab.

Errors are reported through pop-up messages.

## Vivado Tcl Operations

### In This Section

|   |     |
|---|-----|
| Running a Tcl Script from within the Vivado Design Suite .....                    | 115 |
| Clearing the Tcl Console.....   | 117 |
| Generating a Timing Summary Report with the Tcl Interface .....                   | 117 |
| Using the llength Command to Obtain the Number of Paths Between Two Domains ..... | 118 |
| Using the join Command to Join All Timing Paths Between Two Domains .....         | 119 |
| Building a Custom Timing Report with the Tcl Interface .....                      | 119 |
| Generating a List of High Fanout Nets in the Design.....                          | 121 |
| Building a Custom Timing Report .....   | 122 |

## Running a Tcl Script from within the Vivado Design Suite

The Vivado Design Suite offers both GUI and scripted control. Tcl provides scripted control. These Tcl commands can be entered directly into the tool one at a time (or block copied), or an entire Tcl script can be loaded and executed (sourced).

### 1-1. Run a Tcl script.

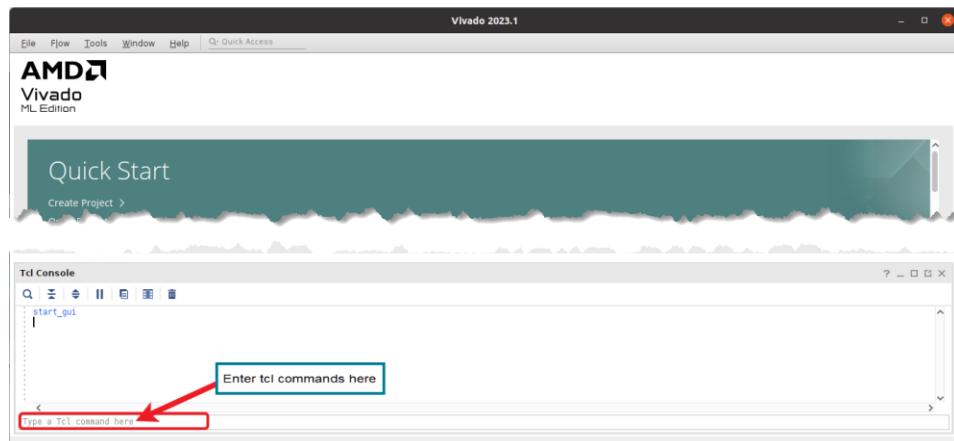
- 1-1-1. Select the **Tcl Console** tab to view the console.

**Hint:** If the Tcl Console is minimized, clicking the tab will partially expand the view.

- 1-1-2. Locate the Tcl command line entry.

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

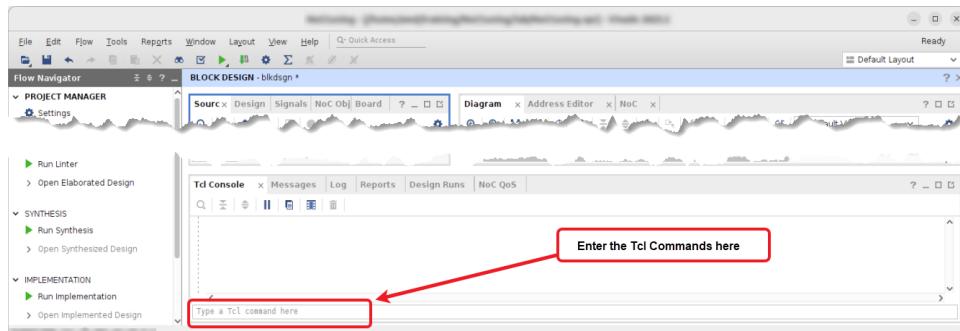
From the Welcome screen:



**Figure 149: Accessing the Tcl Console from the Getting Started Page**

You can always click the Tcl tab to maximize/restore it.

From the Tcl Console tab in an open project:



**Figure 150: Entering Commands into the Tcl Console from an Open Project**

Initially, the directory location begins within the tool installation directory.

- 1-1-3.** Enter the following command to change the current working directory to where the Tcl script is located:

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

**Note:** The Tcl proc `env` reaches out to the operating system and returns the value of the environment variable given by `TRAINING_PATH`. The `$` indicates that the value of that variable should be returned and the two colons (`::`) indicate from which namespace the information should be pulled. Because there is nothing in front of the `::`, the global or base layer of the namespace is to be used.

- 1-1-4.** Verify that you are now where you want to be by entering the following into the Tcl command line:

```
pwd
```

This should show that you are in the `support` directory under the `<the topic cluster name>` directory in the `training` directory.

Note that the `training` directory can be anywhere in the system. What is essential is that you are currently working from the `support` directory under `<the topic cluster name>`.

- 1-1-5.** Run the following Tcl command to run the helper Tcl script for this lab:

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

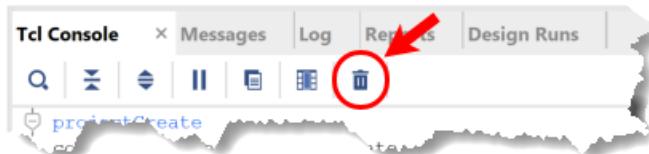
The Tcl script is run as though you typed each command included in the Tcl script into the Tcl command line. You can follow the execution of the script and monitor for any errors or warnings in the Tcl Console.

## Clearing the Tcl Console

### 1-1. Clear the Tcl Console window.

**This helps to make it clear how each command behaves.**

- 1-1-1. From the Tcl Console, click the **Trash Can** icon in the Tcl Console toolbar.



**Figure 151: Clearing the Tcl Console**

A dialog box opens, asking you to confirm the clearing of the Tcl console.

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

## Generating a Timing Summary Report with the Tcl Interface

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

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

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

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

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

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

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

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

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


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

```

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

---

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

For example:

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

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

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

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

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

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

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

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

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

For example:

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

returns --> 1 2 3

```
join $Vivado \n
```

returns --> 1  
2  
3

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

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

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

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

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

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

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

## Building a Custom Timing Report with the Tcl Interface

---

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

The `report_timing` command displays specific timing paths only.

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

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

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

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

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

## Generating a List of High Fanout Nets in the Design

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

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

```
report_high_fanout_nets
```

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

The default settings are:

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

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

| Net Name  | Fanout | Driver Type |
|---|--------|-------------|
| <code>rst_gen_i0/reset_bridge_clk_rx_i0/rst_clk_rx</code>     | 253    | FDPE        |
| <code>clk_gen_i0/clk_core_i0/CLK_OUT2</code>                  | 149    | BUFG        |
| <code>rst_gen_i0/reset_bridge_clk_tx_i0/rst_clk_tx</code>     | 129    | FDPE        |
| <code>resp_gen_i0/Q[0]</code>                                 | 43     | FDRE        |
| <code>resp_gen_i0/Q[1]</code>                                 | 39     | FDRE        |
| <code>cmd_parse_i0/O4[1]</code>                               | 35     | FDRE        |
| <code>clk_gen_i0/clk_div_i0/n_0_cnt[0]_i_2</code>             | 32     | LUT5        |
| <code>clk_gen_i0/clk_div_i0/n_0_cnt_reg[0]</code>             | 32     | FDRE        |
| <code>uart_rx_i0/uart_rx_ctl_i0/O8</code>                     | 32     | FDRE        |
| <code>rst_gen_i0/reset_bridge_clk_samp_i0/rst_clk_samp</code> | 31     | FDPE        |

Figure 152: report\_high\_fanout\_nets Report

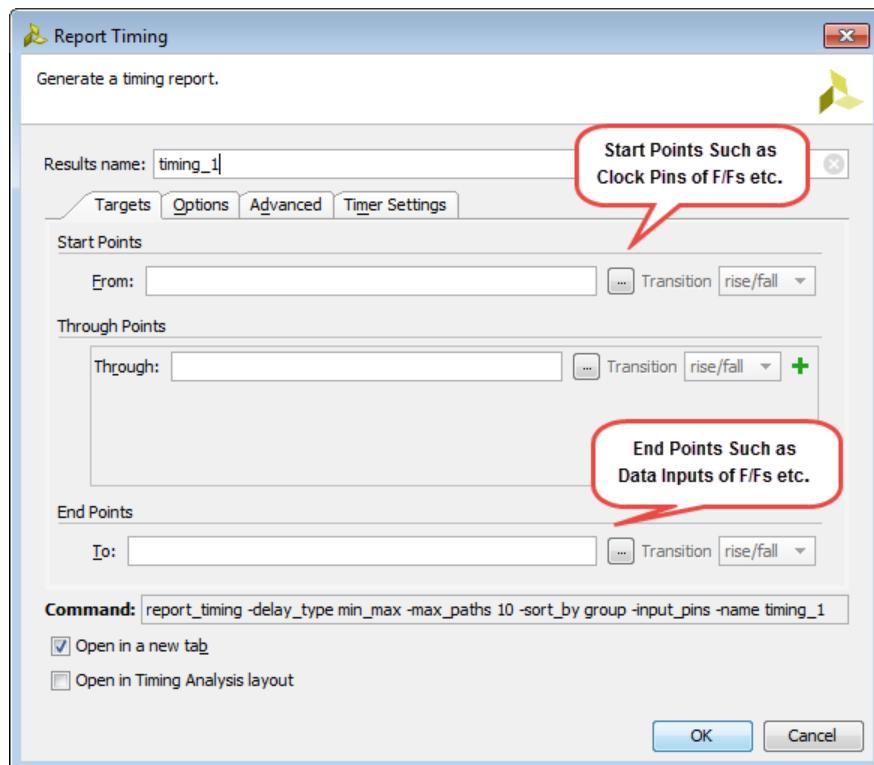
## Building a Custom Timing Report

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 153: 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 .....                                | 123 |
| Understanding the Path Detail Info from a Timing Path Report.....                        | 125 |
| Generating a Custom Schematic to Display a Selected Number of Failing Timing Paths ..... | 128 |

## Generating a Timing Summary Report with Description

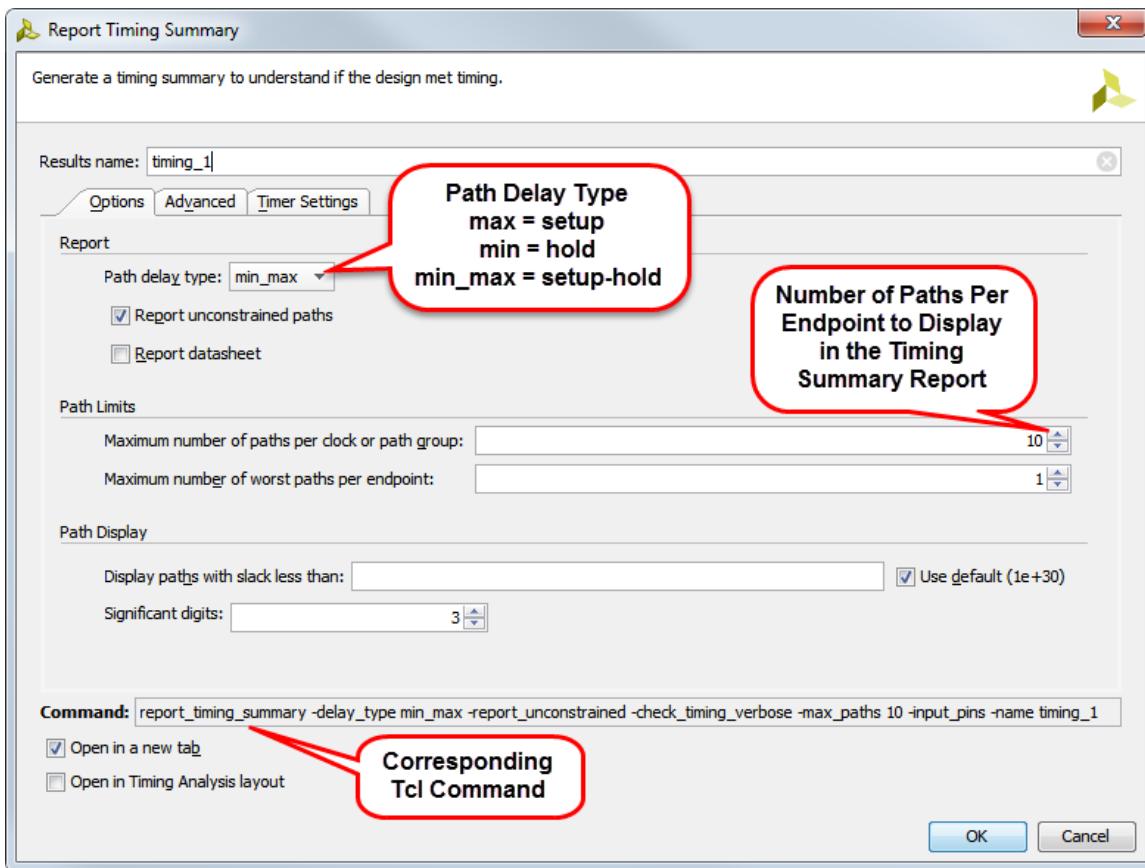
---

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.

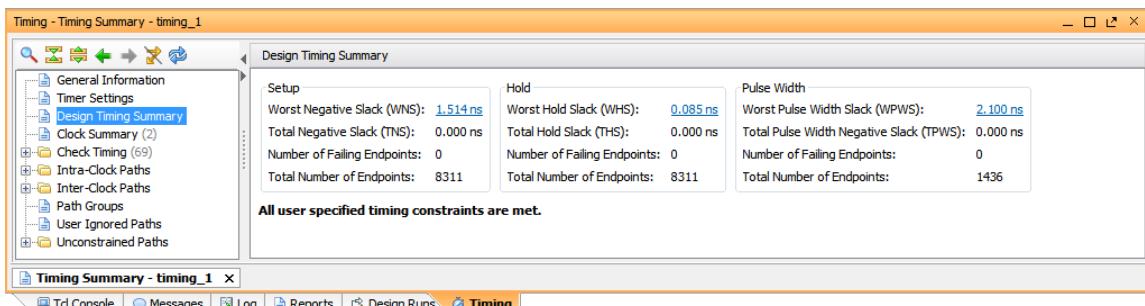


**Figure 154: Timing Summary Report Dialog Box: Options Tab**

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

The Timing - Timing Summary -timing\_1 window opens in the Timing tab at the bottom of the GUI.

Note that timing\_1 is the default name of the timing report. You can change the name in the Report Timing Summary dialog box.



**Figure 155: Design Timing Summary Window**

## Understanding the Path Detail Info from a Timing Path Report

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

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

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

This opens a Path <number> - report\_name tab in the main workspace area.

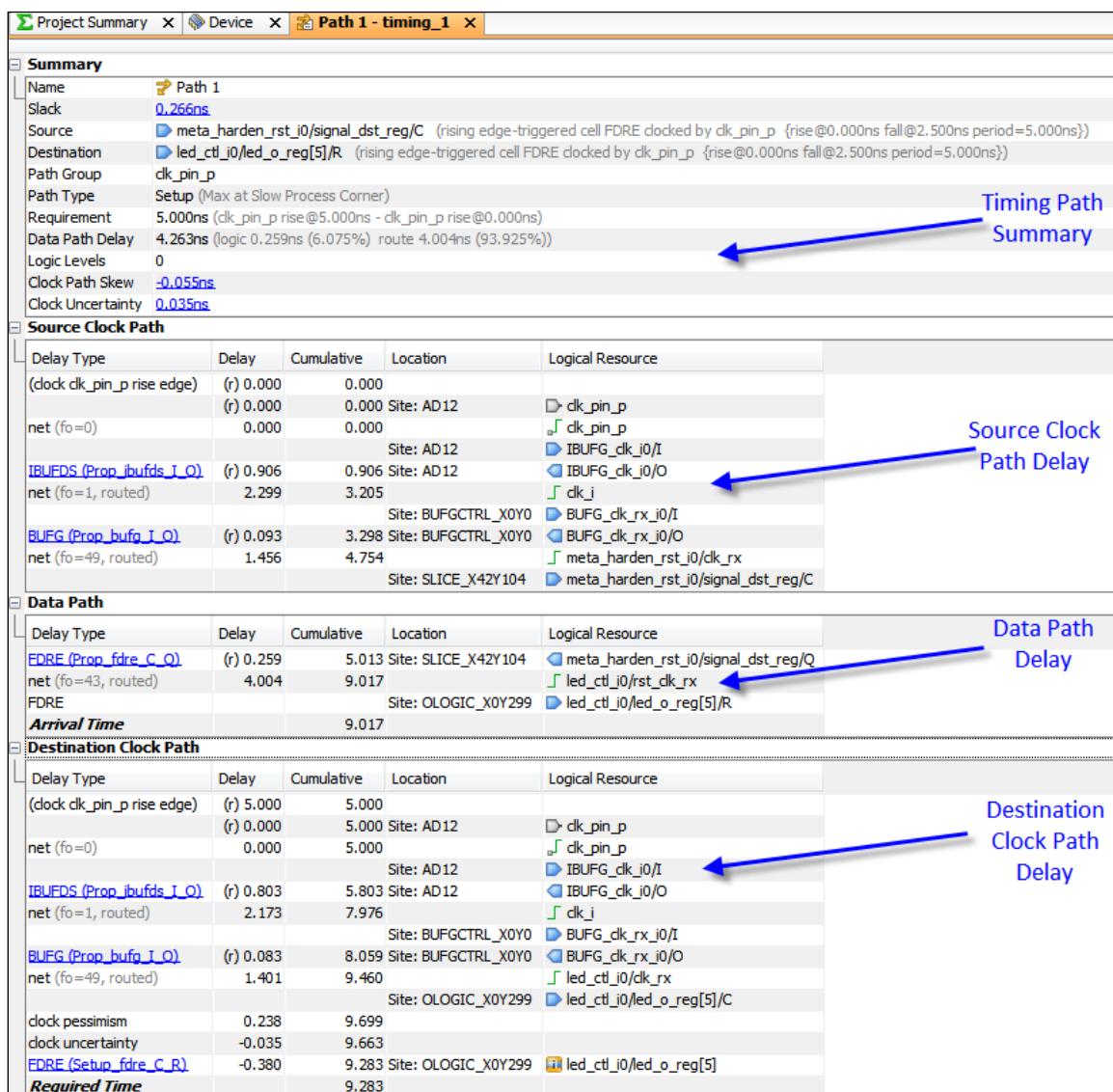


Figure 156: Path Detail Information of a Selected Path

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 a 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 <Shift> or <Ctrl> key.
- N** = required number such as 10, 11 , 20, etc.

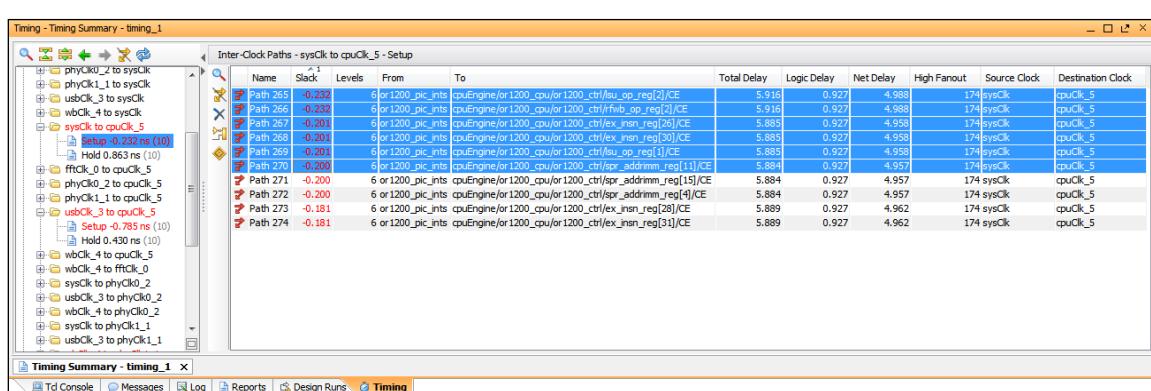


Figure 157: Selecting Multiple Failing Timing Paths

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

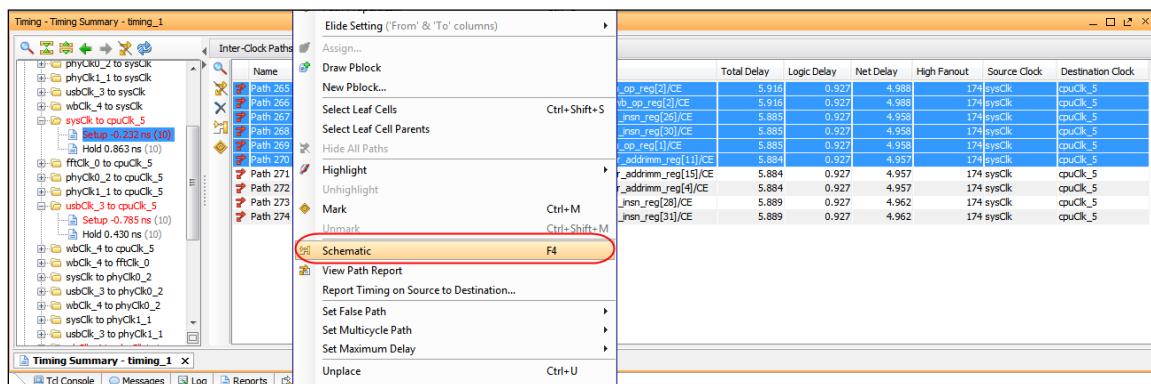
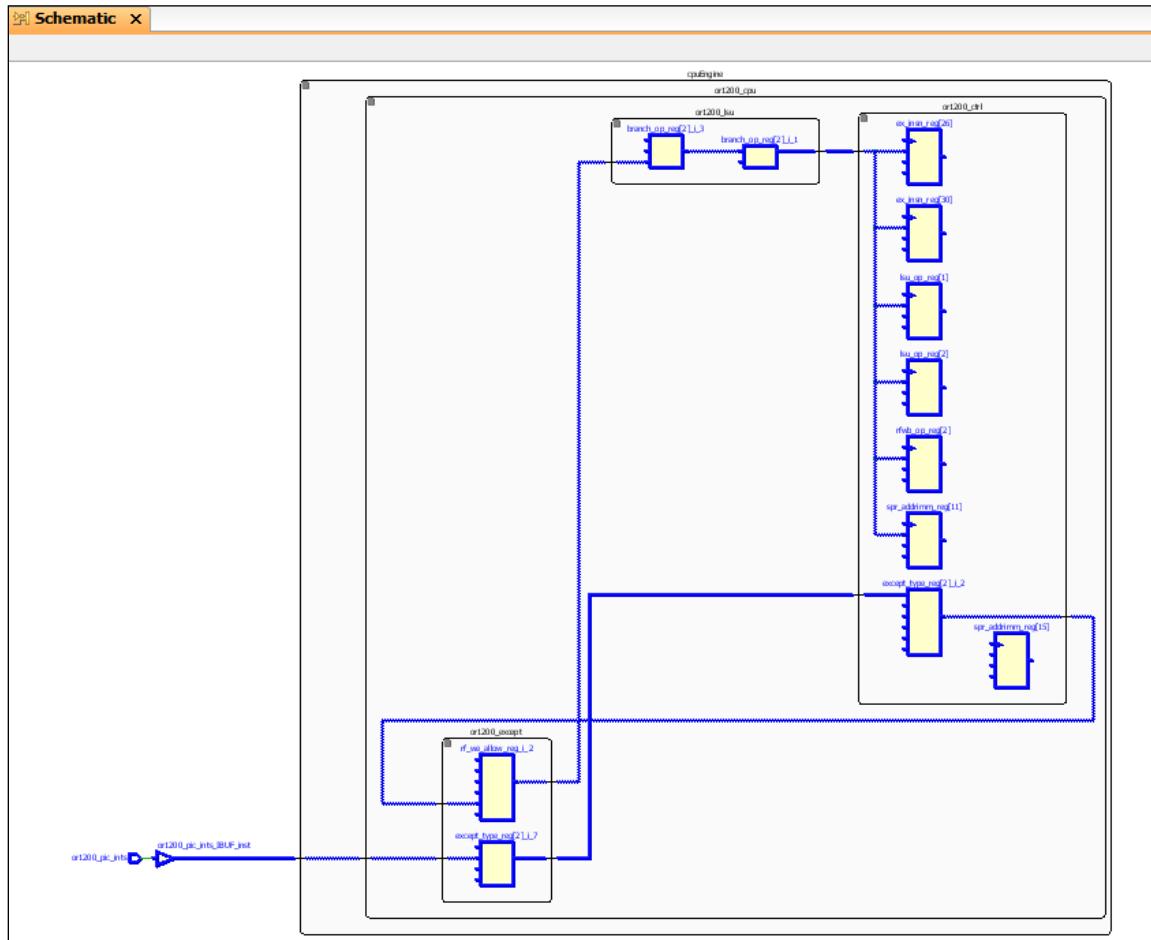


Figure 158: Selecting the Schematic Option for the Selected Paths

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



**Figure 159: 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

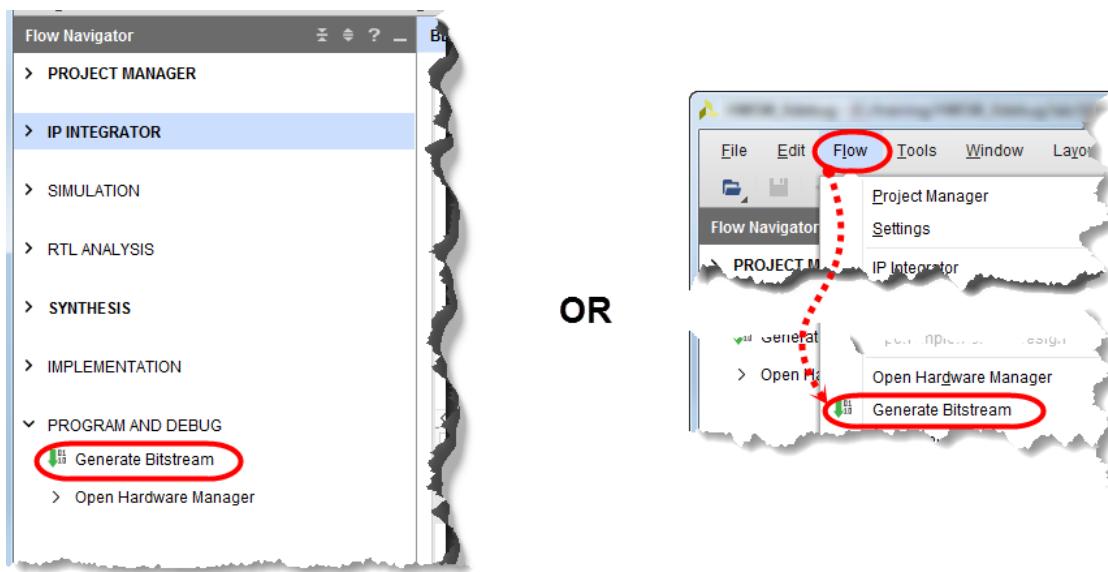
|   |     |
|---|-----|
| Implementing a Design and Generating a Bitstream .....    | 130 |
| Opening the Hardware Manager.....                         | 132 |
| Downloading a Bitstream Using the Hardware Manager .....  | 135 |
| Generating the Bitstream and Downloading the Design ..... | 136 |
| Generating the Bitstream .....                            | 139 |
| Launching and Configuring the Terminal Emulator.....      | 139 |

## Implementing a Design and Generating a Bitstream

### 1-1. Generate the implemented design and bitstream.

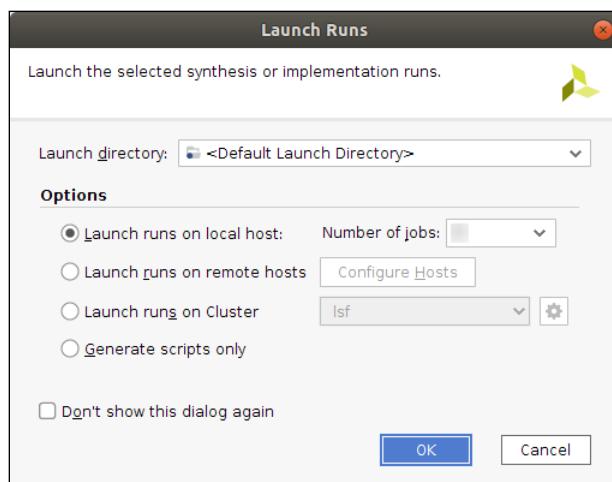
The Vivado Design Suite runs the equivalent of a "make" file that understands dependencies. If synthesis or implementation has not been run, and bitstream generation is requested, then the tools are smart enough to synthesize whatever modules need to be synthesized and implement the design before generating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

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



**Figure 160: Launching the Generate Bitstream Process**

- 1-1-2. If you are presented with a warning that there aren't any implementation results available, click **Yes**.



**Figure 161: Launch Runs Dialog Box**

The default settings are sufficient for this run.

**1-1-3.** Click **OK** to proceed with synthesis, implementation, and bitstream generation.

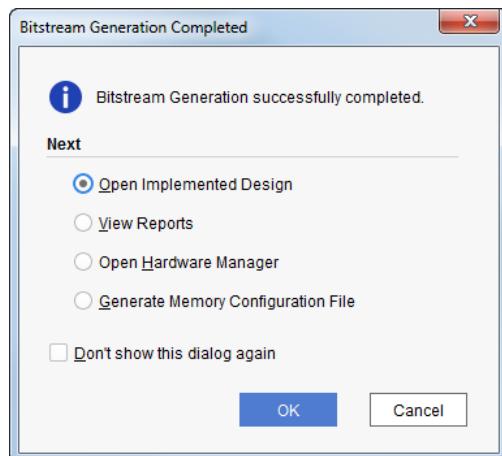
**1-1-4.** Address any critical warning messages dialog boxes that appear and continue the bitstream generation operation.

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 162: 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.

**1-1-5.** Click **Cancel** to perform none of these tasks and end the bitstream generation process.

OR

If you would like to quickly jump to one of these frequently used next steps, select the option of your choice and click **OK**.

## Opening the Hardware Manager

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

### 1-1. Open the hardware manager.

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

The Hardware Manager window opens.

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

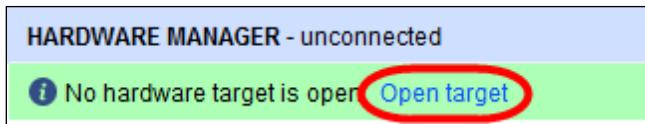


Figure 163: Opening a Hardware Target

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

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

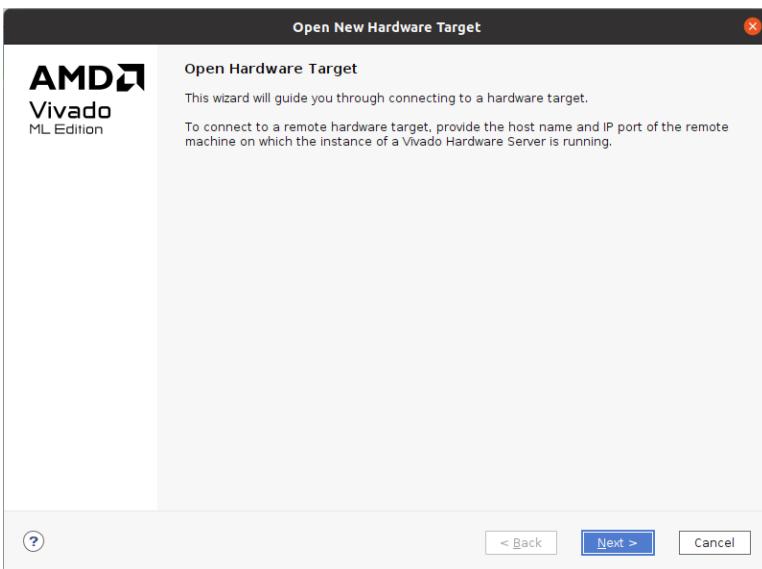
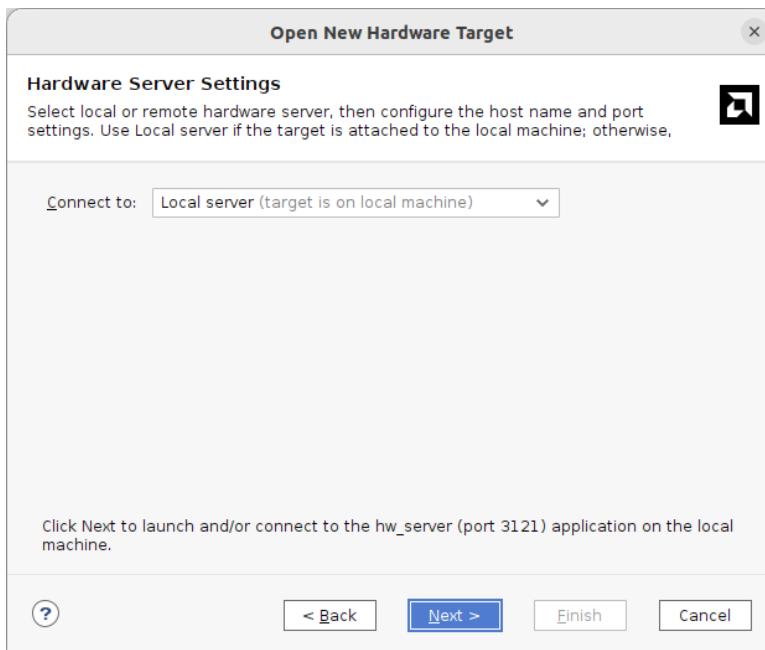


Figure 164: Open Hardware Target Dialog Box

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

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

Typically, this is left at its default value.



**Figure 165: Setting the Server Name for the New Hardware Target**

- 1-2-4. Click **Next** to select the hardware target.

- 1-2-5. Verify the hardware target.

This becomes important when there are multiple targets connected to the PC.

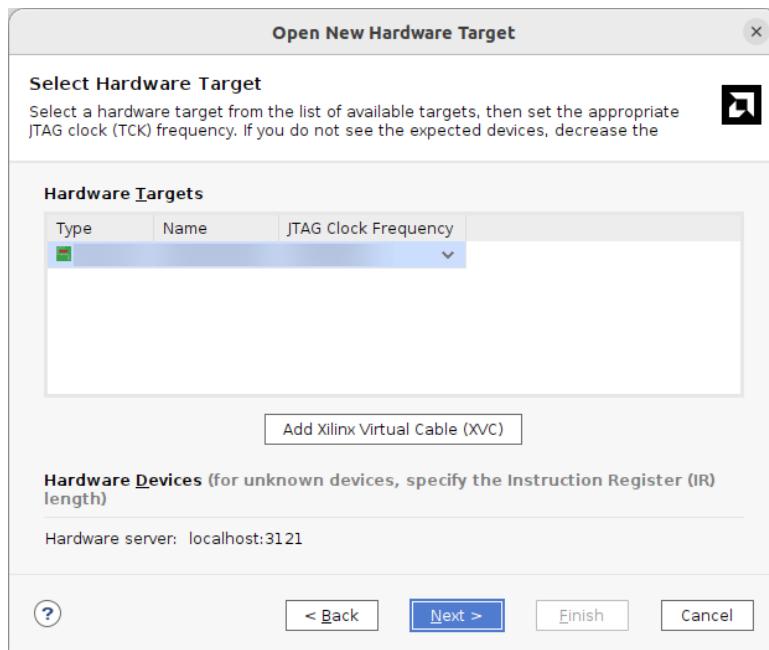
You can change the frequency of the JTAG cable if you are experiencing communications problems.

**[Linux users]:** If you receive an error saying that no active target is found, check the USB connections by selecting **Devices > USB** in the VirtualBox toolbar at the top.



**Figure 166: Selecting the Hardware Target**

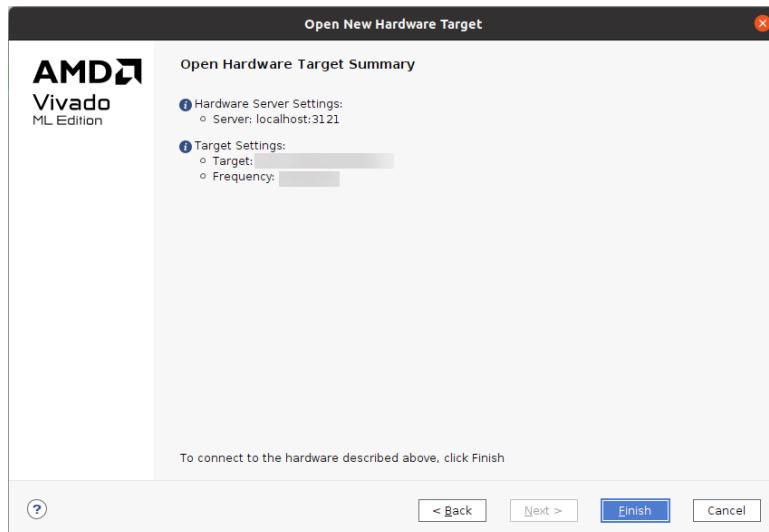
- 1-2-6.** Leave the frequency at its default value.



**Figure 167: Selecting the Hardware Target**

- 1-2-7.** Click **Next** to view the hardware target summary.

A summary of the connection is displayed.



**Figure 168: Summary of the Open Hardware Target Settings**

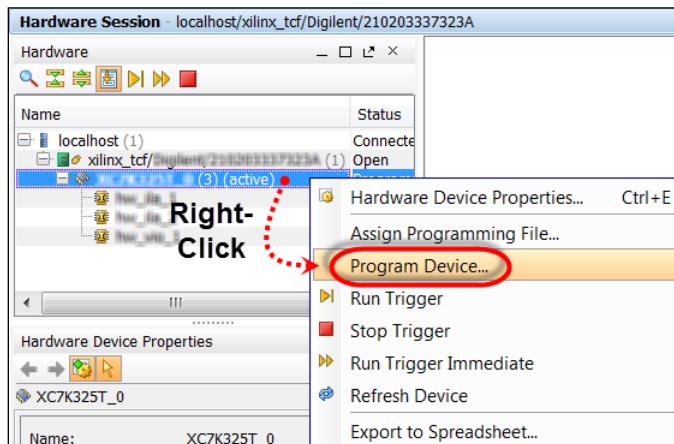
- 1-2-8.** Click **Finish** to connect to the new hardware target.

## Downloading a Bitstream Using the Hardware Manager

Now that a hardware session is established, your task is to download a bitstream to your board.

### 1-1. Download a *bitstream* to the target board.

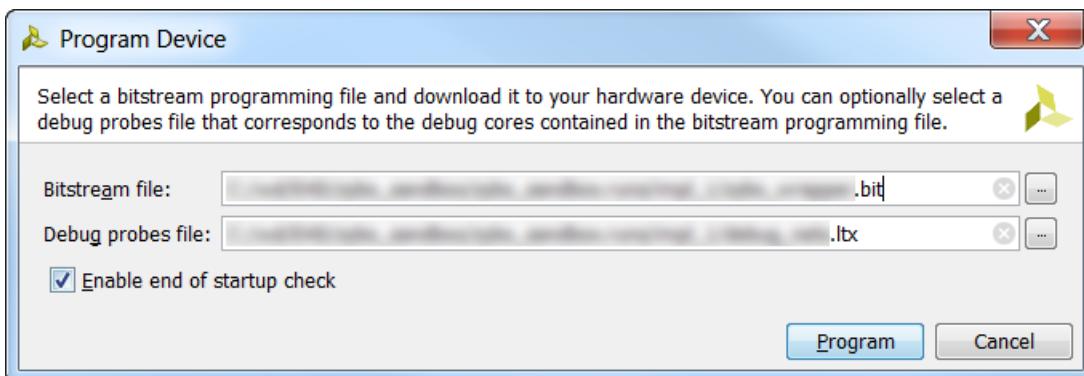
- 1-1-1. From the Hardware pane, identify the FPGA/SoC that you would like to program.
- 1-1-2. Right-click that entry and select **Program Device**.



**Figure 169: 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 170: Selecting the Bitstream and Programming the Device**

- 1-1-4. Click **Program**.

A progress bar appears and, when complete, the dialog box closes.

## Generating the Bitstream and Downloading the Design

### 1-1. Generate the bitstream.

- 1-1-1. Click **Generate Bitstream** from the Flow Navigator, under Program and Debug.

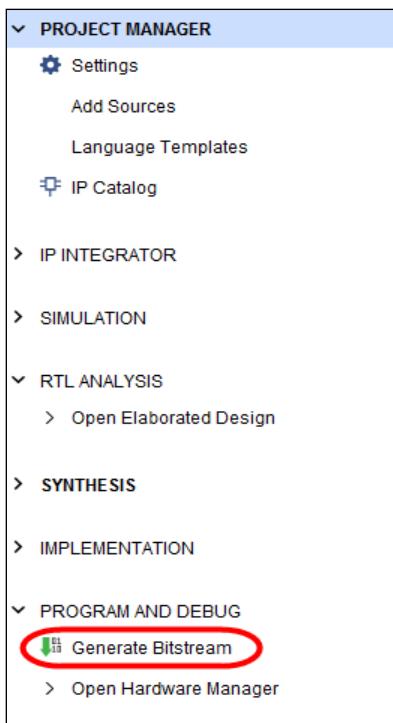


Figure 171: Generate Bitstream

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

Note that the Generate Bitstream process will try to resynthesize and implement the design if any process is out of date.

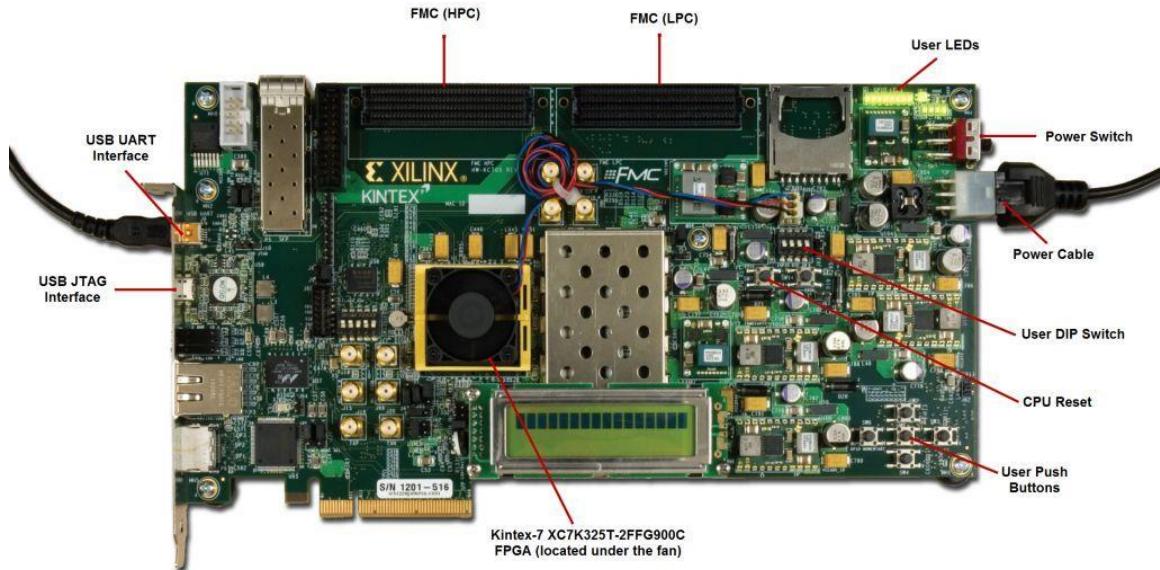
- 1-1-3. Click **Yes** if prompted to rerun any of the processes that are needed for bitstream generation.

- 1-1-4. Click **Cancel** in the Bitstream Generation Completed dialog box.

### 1-2. Power on the board.

- 1-2-1. Connect a USB cable from a USB port on your computer to the USB UART connector on the evaluation board.
- 1-2-2. Connect the USB cable to the Digilent USB JTAG interface.
- 1-2-3. Ensure that the power cord is plugged in and turn on the evaluation board.

**1-2-4.** Make sure that the board settings are proper.



**Figure 172: KC705 Evaluation Board**

You may be prompted to install drivers when the board is first connected. Do not allow the driver installation to search the Web but allow it to search for the drivers on your computer.

**1-2-5.** Ensure that all DIP switches (SW11) are in the off position.

### **1-3. Program the Kintex FPGA on the board using the hardware manager.**

**1-3-1.** Click **Open Hardware Manager** in the Flow Navigator, under Program and Debug.

**1-3-2.** Click **Open Target > Open New Target**.

**1-3-3.** Click **Next** in the Open New Hardware Target dialog box.

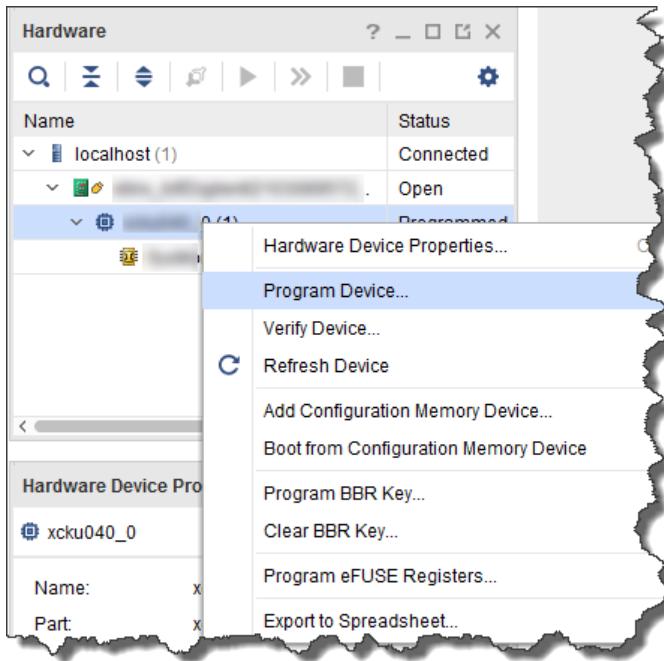
**1-3-4.** Click **Next** in the Hardware Server Settings dialog box.

**1-3-5.** Click **Next** in the Select Hardware Target dialog box.

**1-3-6.** Click **Finish**.

**1-3-7.** For the Kintex-7 board, right-click **xc7k325t\_0** and select **Program Device**.

For the Kintex UltraScale board, right-click **xcku040\_0** and select **Program Device**.



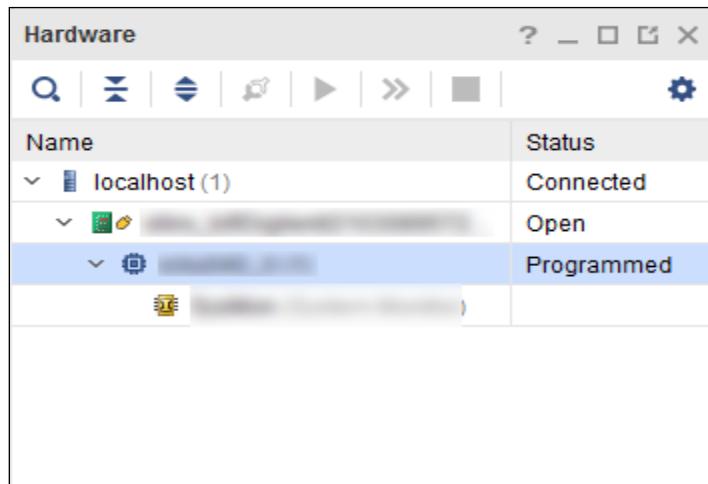
**Figure 173: Programming the Device**

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 174: Programming Status of the Device**

## Generating the Bitstream

### 1-1. Generate the bitstream.

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

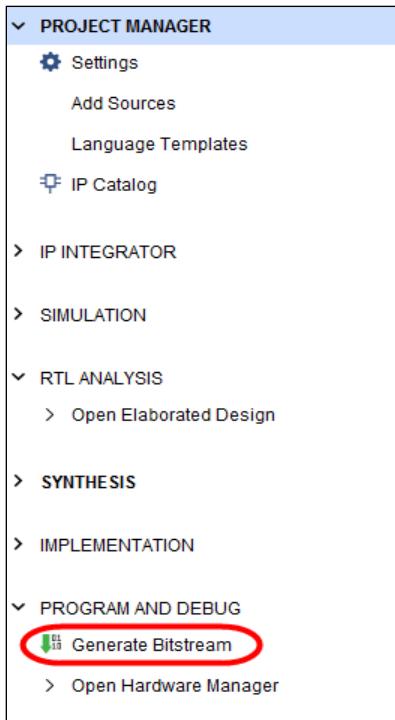


Figure 175: Generate Bitstream

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 Terminal Emulator

---

Maintaining consistency in the instructions is an important aspect of the lab experience. The labs will use general and vague commands such as "Open the terminal emulator". Depending on the system that you are using, you must know which terminal emulator to configure and launch. Here are the instructions for opening a terminal emulator for both the Linux and Windows environments.

Generally, the software labs will use the serial terminal built into the Vitis platform; however, there are some labs that will require you to communicate with the board without using the Vitis environment. These situations require the use of a third-party serial port emulator.

We have tested Tera Term for the Windows environment and GTKTerm for Linux. The GTKTerm tool is pre-installed with the Customer Training VM. Both terminal emulators listed here run independently of the tools. If you have another terminal emulator that you prefer, you can certainly use it; however, you are responsible for figuring out how to configure it.

### Linux

GTKTerm is a simple GTK+ terminal used to communicate with the serial port. Other terminal emulators may be used; however, the installation and configuration instruction provided here are for GTKTerm.

GTKTerm is already installed in the VM provided by the Customer Training team; however native Linux users may need to install GTKTerm.

#### 1-1. [Native Linux users] Acquire and install the GTKTerm software from the command line.

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

1-1-2. Download and install GTKTerm:

```
[host] $ sudo apt install gtkterm
```

1-1-3. When prompted for the password, enter the super user password.

The tools will install in about a minute or less.

#### 1-2. Launch GtkTerm and set the port configuration.

1-2-1. Click the **GtkTerm** icon from the quick launch toolbar.

Alternatively, GtkTerm can be launched from a Linux terminal window (**<Ctrl + Alt + T>**) and entering:

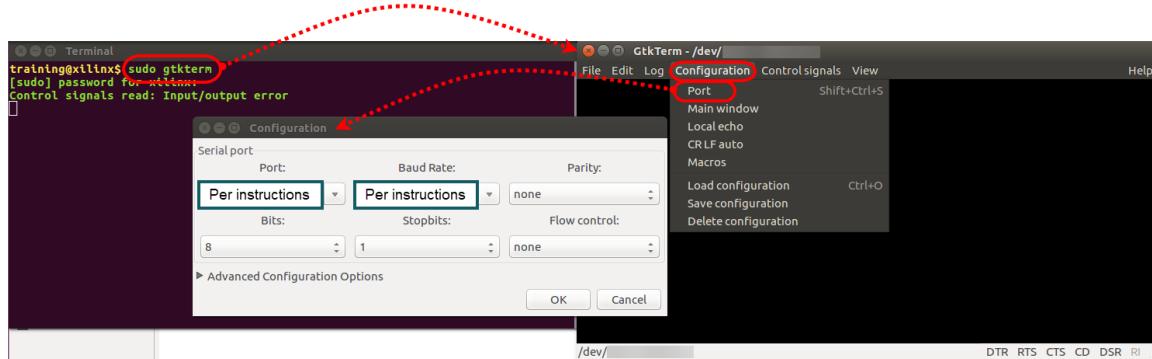
```
[host] $ sudo gtkterm
```

**Note:** While the application will run as a regular user, you must be a super user to access the ports.

When the GtkTerm window opens, perform the following.

- 1-2-2. Select **Configuration > Port** to open the Configuration dialog box.
- 1-2-3. Identify the port associated with your board and set the port as **/dev/ttyUSBx** (where x could be 0, 1, 2, 3, etc.)
- 1-2-4. Set the baud rate to **115200**.
- 1-2-5. Leave the rest of the settings at their default.

**Note:** You can open multiple instances of **/dev/USBx** if you are unable to find out which port your UART is connected to.



**Figure 176: Opening GtkTerm and Selecting the Port Configuration**

[Optional]: You can save these settings so that you do not have to reconfigure GtkTerm each time you open it by selecting **Configuration > Save configuration**.

If you save the configuration as "default", this configuration will open when GtkTerm starts. Otherwise, you can save the configuration with another name; however, you will then need to load the configuration each time you start GtkTerm.

- 1-2-6. Click **OK** to save the settings and leave the terminal open.

## Windows

Tera Term is a popular public domain terminal emulation program. It is capable of operating as a serial port terminal or as a telnet client.

### 1-3. Download and install Tera Term.

- 1-3-1. Acquire and install the Tera Term software from any legitimate site. Recommended sites:
  - [ttsch2.osdn.jp/index.html.en](http://ttsch2.osdn.jp/index.html.en) (Tera Term home page with documentation)
  - [osdn.net/projects/ttsch2](http://osdn.net/projects/ttsch2)
  - [en.sourceforge.jp/projects/ttsch2](http://en.sourceforge.jp/projects/ttsch2)
  - [logmatt.com](http://logmatt.com)
- 1-3-2. Install per instructions.

There are two types of downloads: a traditional zip install, and a self-installing version, which is recommended.

[Optional]

Certain drivers like installing their com port numbers using high numbered serial ports. Tera Term does not accept these port numbers by default, so you will need to override the Tera Term settings:

- 1-3-3. Open **TERATERM.INI** (found in the install path for Tera Term) with an ASCII text editor.
- 1-3-4. Set to MaxComPort=256.
- 1-3-5. Save and close the INI file.
- 1-3-6. Use the **Setup > Save Setup** option to save the setup.

#### 1-4. Launch the Tera Term terminal program.

- 1-4-1. Double-click the **Tera Term** icon on the Windows desktop to launch Tera Term.

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

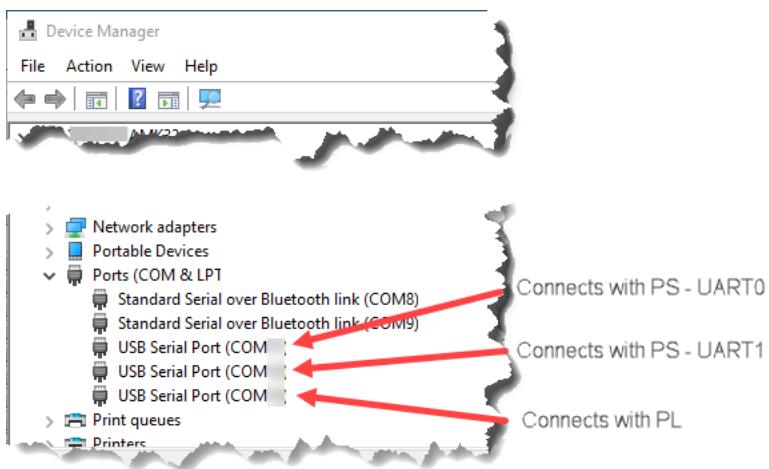
- 1-4-2. Select **Serial** as the connection (1).

- 1-4-3. 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 restart this step.

- 1-4-4. Select the COM # (3).

**Hint:** MPSOC and RFSoC devices display three COM ports. Their specific numbers may vary based on the USB enumeration process; however, the first two COM ports connect to the UARTs in the PS and the third connects to PL pins.



**Figure 177: Locating and Identifying COM Ports from the Windows Device Manager**

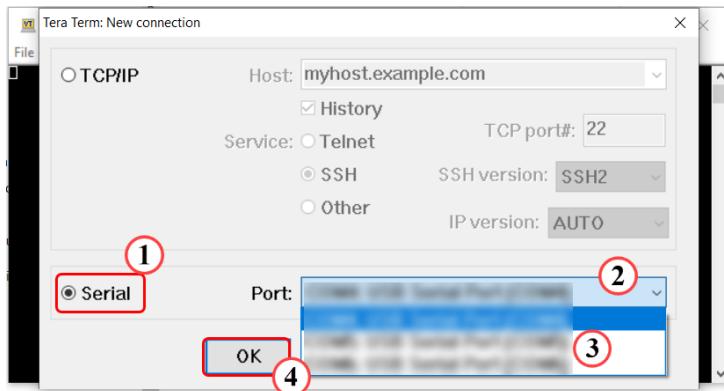


Figure 178: Selecting the COM Port

**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.

- 1-4-5. Click **OK** (4).

The terminal console window opens.

- 1-4-6. Select **Setup > Serial Port**.

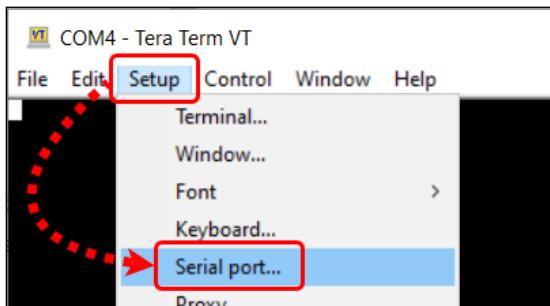
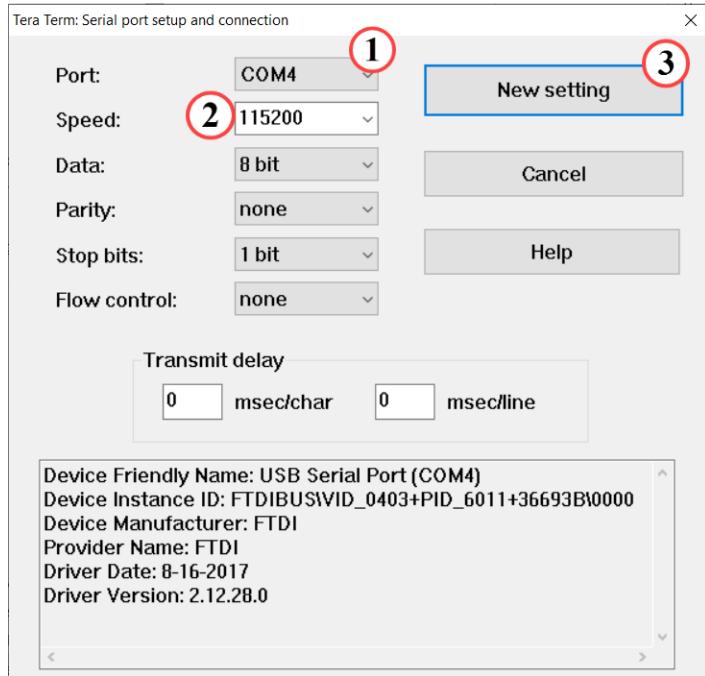


Figure 179: Opening the Tera Term Serial Port Setup Window

The Tera Term Serial Port Setup dialog box opens.

**1-4-7.** Confirm that the proper serial port has been selected (1).

**1-4-8.** Set the baud rate to **115200** (2).



**Figure 180: 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.

**1-4-9.** Click **OK** (3).

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

## Vivado Analyzer Operations [Last Updated Version 2017.1]

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

### In This Section

|                               |     |
|-------------------------------|-----|
| Informative Information ..... | 144 |
| Instructions.....             | 147 |

### Informative Information

The following information is for background understanding of the materials discussed in the Instructions section.

## Overview of the Vivado Analyzer Hardware Session

The following is a quick review of the Hardware Session screen usage.

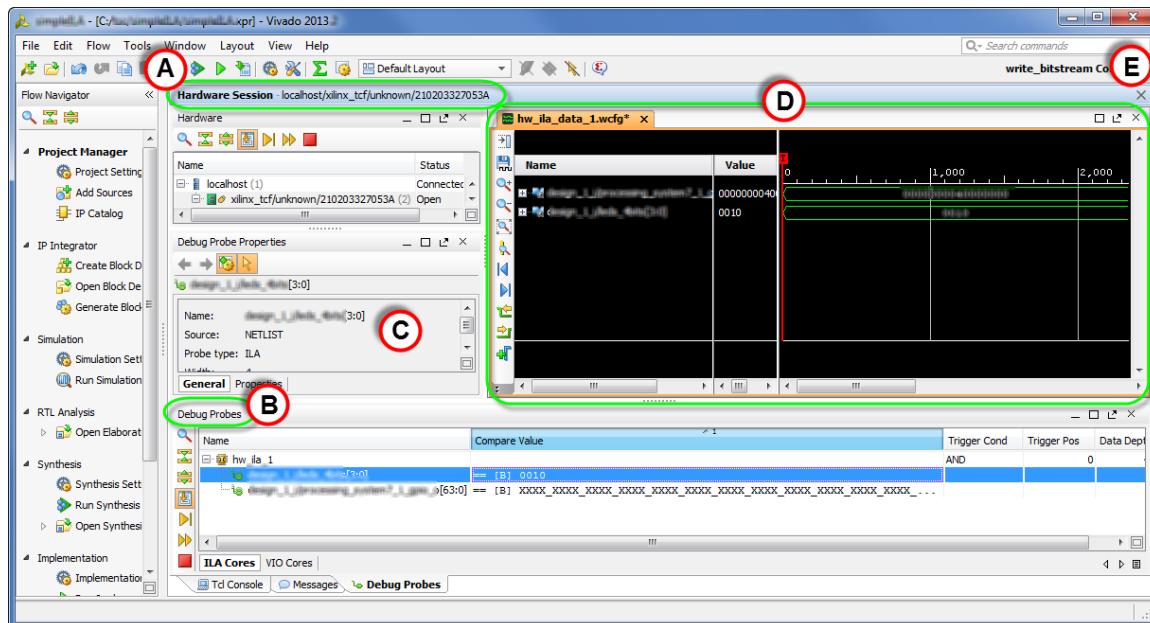


Figure 181: Vivado Analyzer - Hardware Session Screen Usage

- 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:

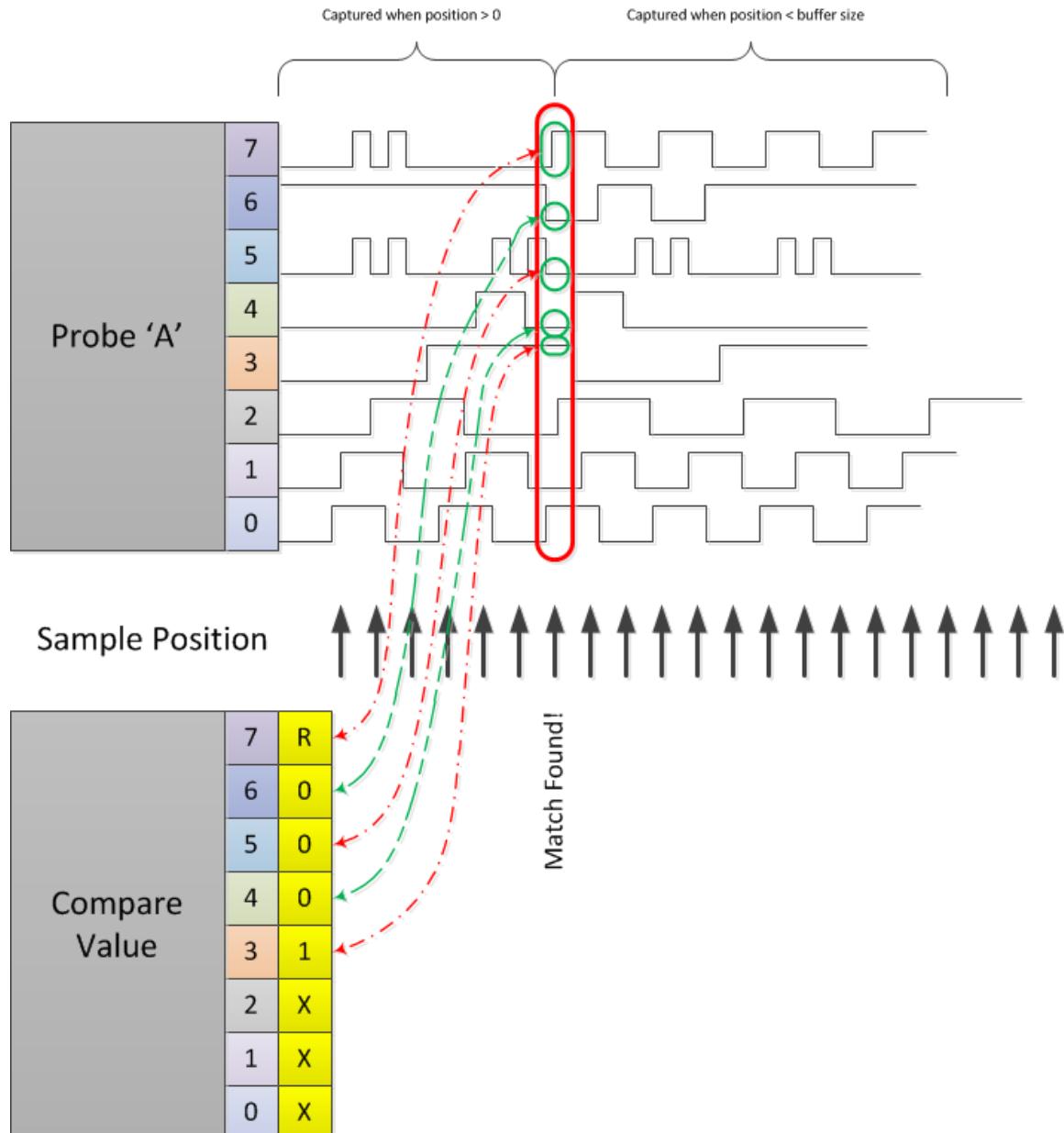


Figure 182: An Individual Probe's Compare Value

Each position or signal within a given probe corresponds to one of five possible match conditions (R,F,0,1,X) as specified in the Compare Value setting. The first time each signal matches its specified "compare value", the combination becomes "true". When all of the signals simultaneously matches all of the "compare values", the probe condition becomes true. If there is only one probe, then the trigger conditions are met and data is captured.

If there are multiple probes within the ILA/VIO, then the results of the probes can be ANDed or ORed together to cause a trigger. For example, if it does not matter which probe triggers in order to capture data, you would set the ILA trigger condition to "OR". If both probes must be simultaneously "true", then set the trigger condition to "AND", which will then cause the trigger.

## Instructions

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

## In This Section

---

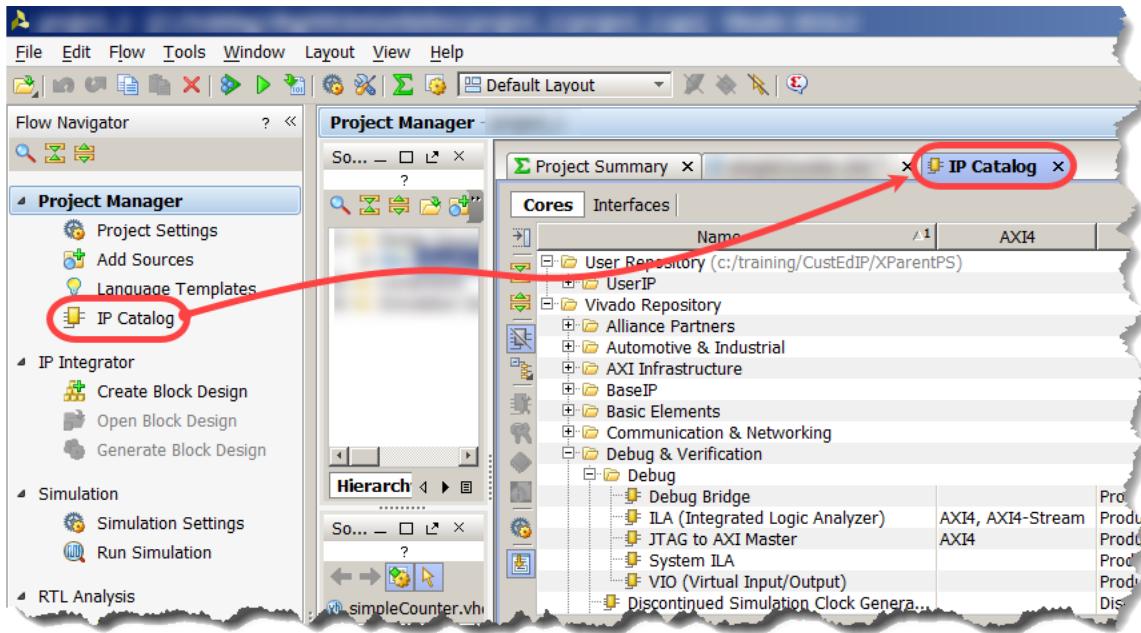
|  |     |
|--|-----|
| Adding IP from the IP Catalog to the Project .....       | 148 |
| Opening the Vivado Analyzer Hardware Manager.....        | 149 |
| Adding Probes (Signals) to the Waveform Viewer .....     | 152 |
| Removing Probes (Signals) from the Waveform Viewer ..... | 153 |
| Setting the Trigger Condition .....                      | 153 |
| Loading a Waveform Configuration .....                   | 155 |
| Arming the Vivado Analyzer Core .....                    | 155 |
| Triggering the ILA Immediately .....                     | 157 |

## Adding IP from the IP Catalog to the Project

### 1-1. Add the desired piece of IP to the project.

- 1-1-1. If necessary, expand **Project Manager** under the Flow Navigator to expose the IP Catalog entry.
- 1-1-2. Double-click the **IP Catalog** entry.

The IP Catalog tab opens in the Project Manager work area.



**Figure 183: Opening the IP Catalog from the Project Manager**

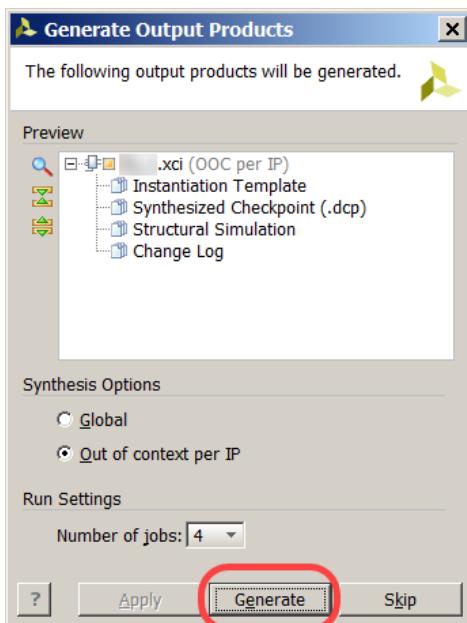
The IP catalog is organized into categories to help you quickly locate the IP you want.

- 1-1-3. Expand **the appropriate IP category** to expose the desired piece of IP.
  - 1-1-4. Double-click **the desired piece of IP** to begin customization.
- You can customize the IP here or do so later.
- 1-1-5. Click **OK** to conclude the customization and advance to the output products generation.

Out-of-context IP generation will generate a netlist for the IP independent of other IP. This is the preferred way to generate outputs as any changes you make will only affect the IP that the changes were made to and accelerates the design process.

You can generate the netlist now, which takes a few moments, or defer generation until synthesis.

- 1-1-6.** Click **Generate** to generate the output products for this core.



**Figure 184: Generating the Out-of-Context IP**

This will typically take a minute or so depending on the amount of code that must be synthesized.

- 1-1-7.** When this process completes, click **OK** to close the summary dialog box.

## Opening the Vivado Analyzer Hardware Manager

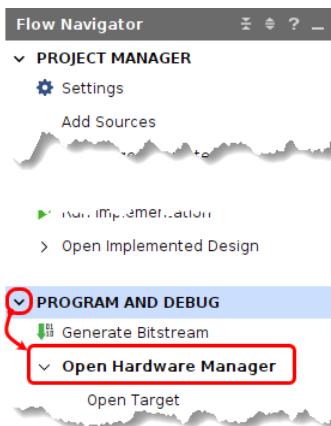
The Vivado logic analyzer requires a connection to the target hardware board. The actual connection is typically made using a USB-to-JTAG intelligent cable. Many of our evaluation boards provide support for the intelligent cable as a module component on the board.

When you open a hardware session, a cable server program is launched that identifies the cable type and JTAG components on the board. A TCP/IP port is opened for a connection that may be to a logic analyzer or other application.

### 1-1. Open the hardware manager.

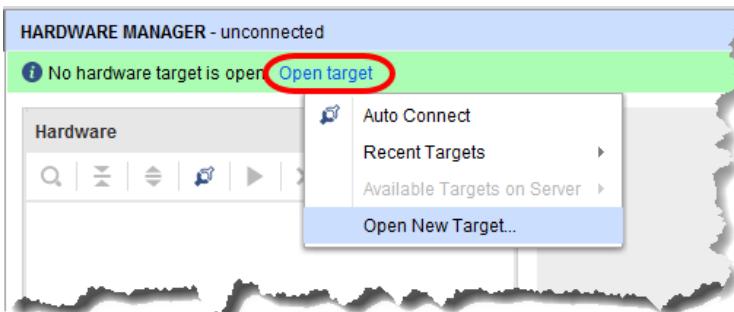
- 1-1-1.** Expand **PROGRAM AND DEBUG** from the Flow Navigator.

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



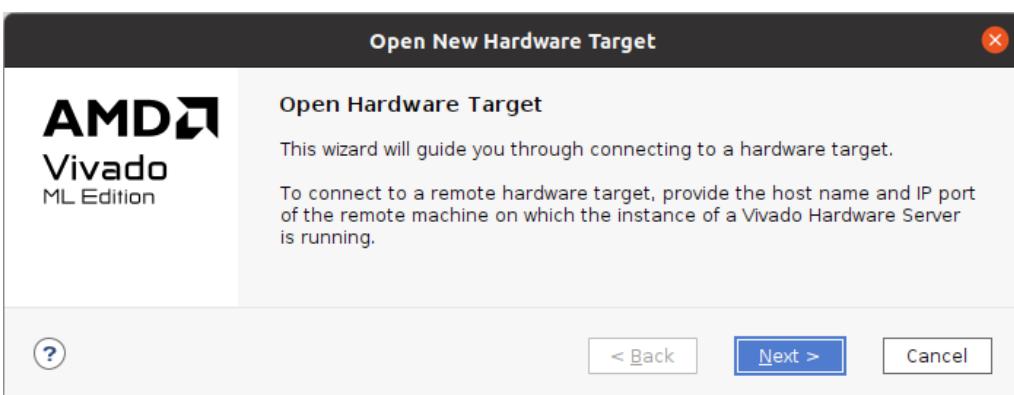
**Figure 185:** Opening the Hardware Manager from an Open Project

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



**Figure 186:** Selecting Open New Hardware Target

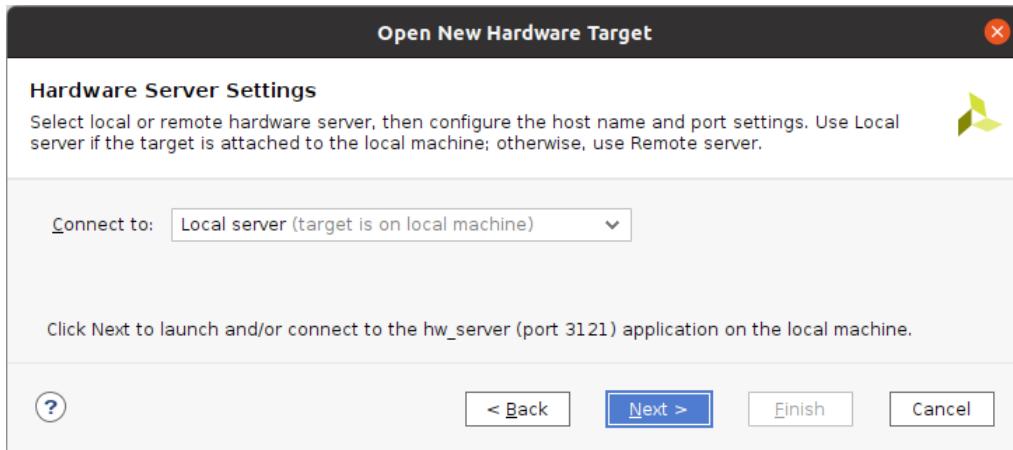
**1-1-4.** Click **Next** to bypass the Welcome dialog box.



**Figure 187:** Opening a New Hardware Target

- 1-1-5.** Use the default local server name since your board is connected to the machine that you are running the Vivado Design Suite on.

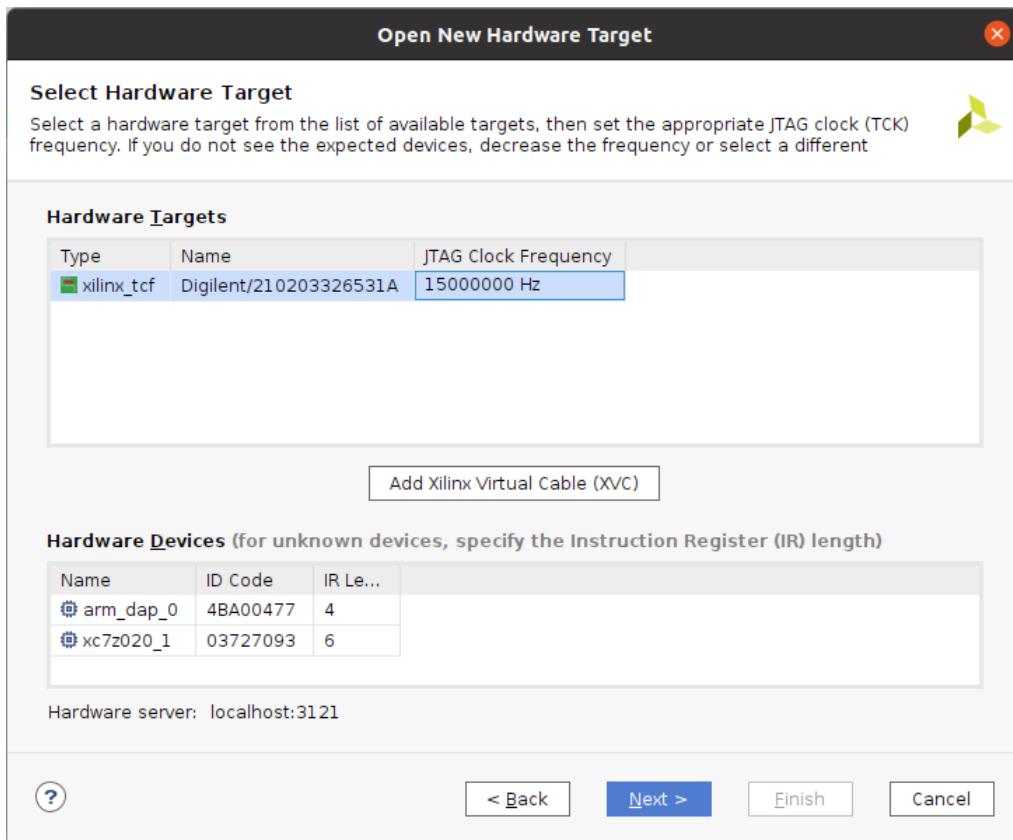
If this is not the case, you will need to select the connection to the machine that is attached to the board.



**Figure 188: Using the Default Server Name**

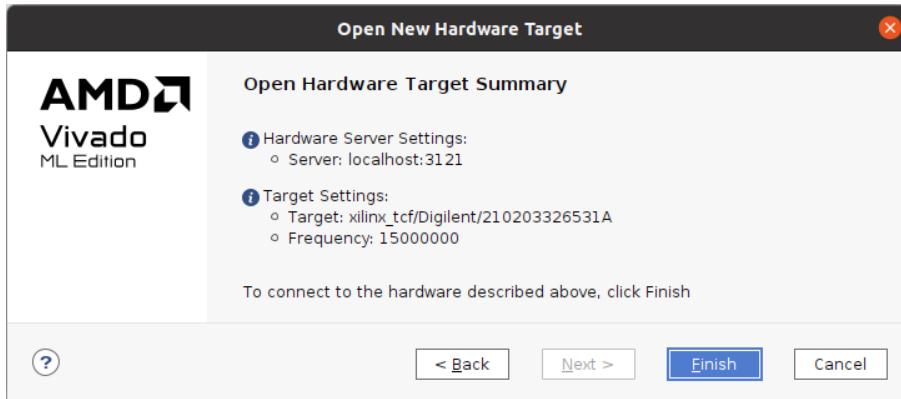
- 1-1-6.** Click **Next** to scan the JTAG chain and view the nodes within the chain.

You should now see the xilinx\_tcf hardware target and the hardware devices present in the JTAG chain. The JTAG clock speed is also shown.



**Figure 189: Viewing the Hardware Target and Devices**

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



**Figure 190: Summary Dialog Box**

- 1-1-8. Click **Finish** to connect to the target.

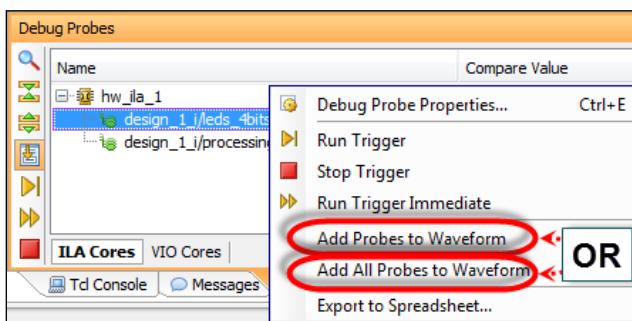
## Adding Probes (Signals) to the Waveform Viewer

---

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.
  - Select **Add Probes to Waveform** to add the selected probes to the waveform or
  - Select **Add All Probes to Waveform** to add all the probes
- 1-1-2. Right-click the probe to add to the waveform from the Design Probes window.
  - Select **Add Probes to Waveform** to add the selected probes to the waveform or
  - Select **Add All Probes to Waveform** to add all the probes



**Figure 191: 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**.

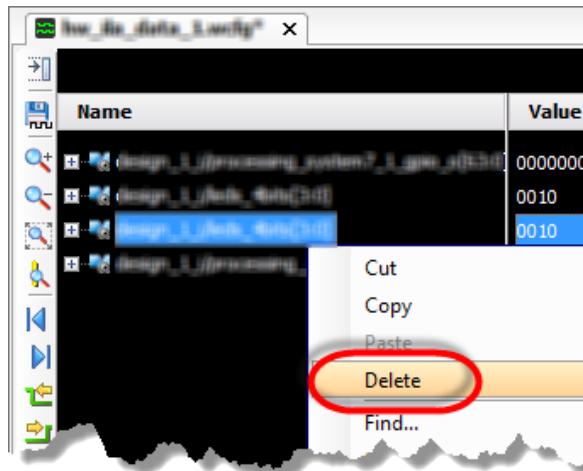


Figure 192: Deleting a Probe from the Waveform Viewer

## 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.

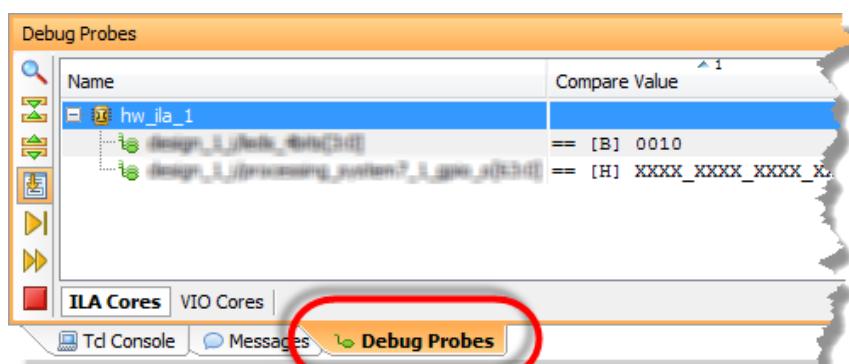
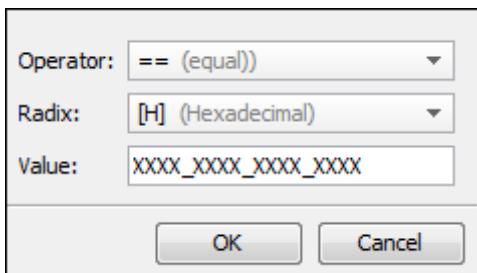


Figure 193: Locating the Debug Probes Tab

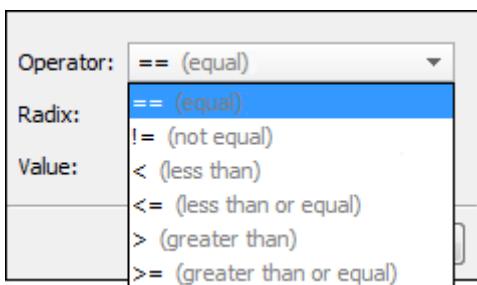
- 1-1-2.** Click in the Compare Value column next to the probe that you want to configure.

A small dialog box opens.



**Figure 194: Compare Value Dialog Box**

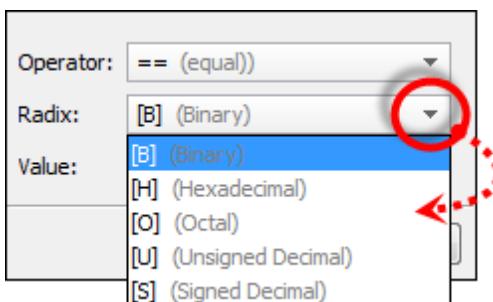
- 1-1-3.** Set the operator to test against.



**Figure 195: 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 196: 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.

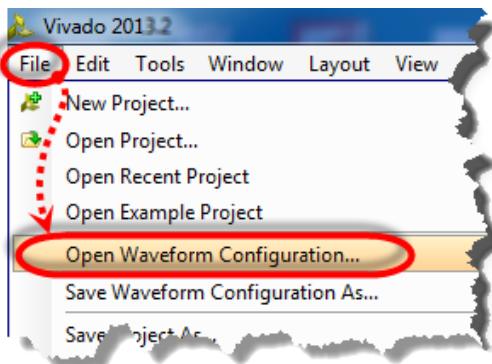
- 1-1-6.** Click **OK**.

## 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 197: Opening a Waveform Configuration File (.wcfg)**

A browser dialog box opens. Files must be entered one at a time.

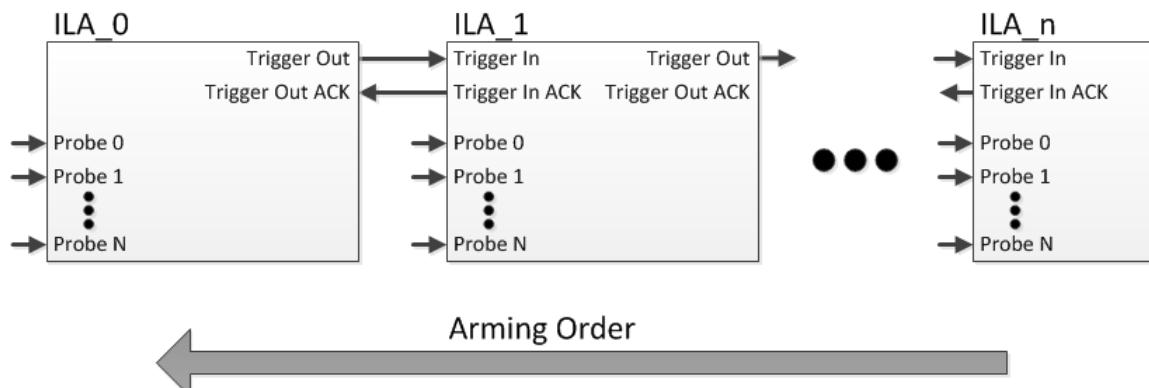
- 1-1-2. Navigate to a waveform configuration file that you want to load.
- 1-1-3. Select the file.
- 1-1-4. Click **Open**.
- 1-1-5. Repeat for additional waveform files.

## 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 198: 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.

## Vitis HLS Tool Operations [Last Updated Version 2022.1]

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

### In This Section

---

|  |     |
|--|-----|
| Launching the Vitis HLS Tool.....                        | 158 |
| Launching the Vitis HLS Tool Using a Linux Terminal..... | 158 |
| Creating a Vitis HLS Tool Project .....                  | 159 |
| Opening a Vitis HLS Tool Project .....                   | 162 |
| Simulating a Vitis HLS Tool Design .....                 | 163 |
| Setting Synthesis Options.....                           | 166 |
| Synthesizing a Vitis HLS Tool Design.....                | 167 |
| Running Cosimulation in the Vitis HLS Tool .....         | 168 |
| Exporting a Vitis HLS Tool Design as IP .....            | 171 |

## Launching the Vitis HLS Tool

There are several ways to launch the Vitis HLS tool. The most popular are shown here.

### 1-1. Launch the Vitis HLS tool.

- 1-1-1. Click the **Vitis HLS** icon (  ) from the desktop or toolbar.

If the desktop or toolbar icon is not available, then you can launch the tool by:

**[Windows 10 users]:** Selecting **Start > Xilinx Design Tools > Vitis HLS 2023.2.**

**[VM Linux users]:** Clicking the **Show Applications** icon in the toolbar (  ), typing **HLS** into the search window, and clicking the **Vitis HLS** icon.

The Vitis HLS tool opens to the Welcome window. From the Welcome window, you can create a new project, open examples, and access documentation and examples.

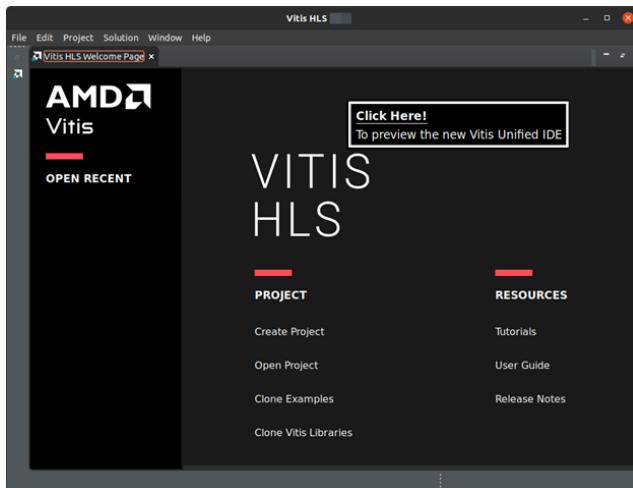


Figure 199: Vitis HLS Welcome Page

## Launching the Vitis HLS Tool Using a Linux Terminal

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

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

- 1-1-2. Enter the following command to set up the Vitis HLS tool environment appropriately:

```
[host] $ source $VITIS_PATH/settings64.sh
```

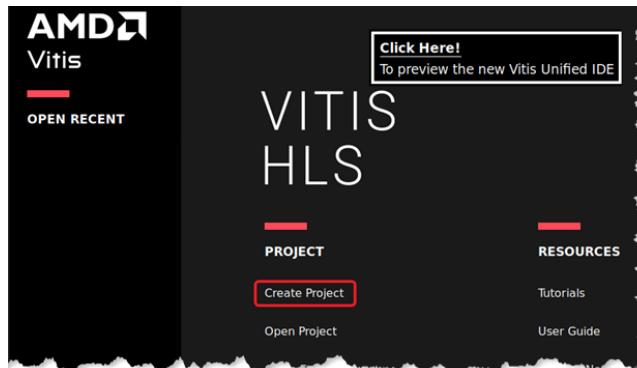
- 1-1-3. Type **vitis\_hls -i** in the terminal and press **<Enter>** to launch the Vitis HLS tool command line interface.

## Creating a Vitis HLS Tool Project

Here you will create a new Vitis HLS tool project from scratch.

### 1-1. Create a Vitis HLS tool project named *your HLS project name*.

- 1-1-1. Click **Create Project** from the Welcome Page.

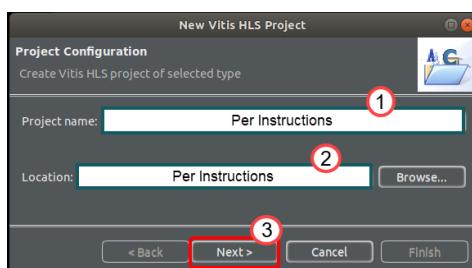


**Figure 200: Creating a New Vitis HLS Tool Project**

The Project Configuration dialog box asks for a project name and location.

- 1-1-2. Enter **your HLS project name** in the Project name field (1).
- 1-1-3. Enter **where your files are located** in the Location field (2).

You can also browse to the desired path and click **Open**.



- 1-1-4. Click **Next** (3).
- 1-2. **The Add/Remove Design Files dialog box opens. Here you will be invited to add existing files or create new sources.**
  - 1-2-1. Click **Add Files**.
  - 1-2-2. Browse to **where your files are located**.
  - 1-2-3. Select **your HLS source files**.

The Vitis 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 **Edit CFLAGS**.

**1-2-4.** Click **Open** to add these files.

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

Note also that you can add compiler directives specific to each entry at this point.

**1-2-5.** 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.

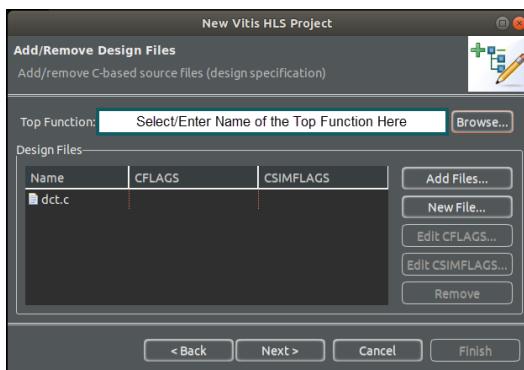
**1-2-6.** Select **the name of the top function (the name of the top file)** from the list and click **OK**.

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

In any C program, the top-level function is called main(). In the Vitis HLS tool design flow, you can specify any sub-function below main() as the top-level function for synthesis. You cannot synthesize the top-level function main().

The following are additional rules:

- Only one function is allowed as the top-level function for synthesis.
- Any sub-functions in the hierarchy under the top-level function for synthesis are also synthesized.
- If you want to synthesize functions that are not in the hierarchy under the top-level function for synthesis, you must merge the functions into a single top-level function for synthesis.



**Figure 201: Adding Files to a New Vitis HLS Project**

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

### 1-3. Add any existing test bench files.

If you have (or want) any test bench files, they can be entered here. Sometimes the test bench is built into the synthesizable file.

1-3-1. Click **Add Files**.

1-3-2. Navigate to where your files are located.

1-3-3. Select your HLS testbench files.

1-3-4. Click **Open** to add these files.

1-3-5. Click **Next**.

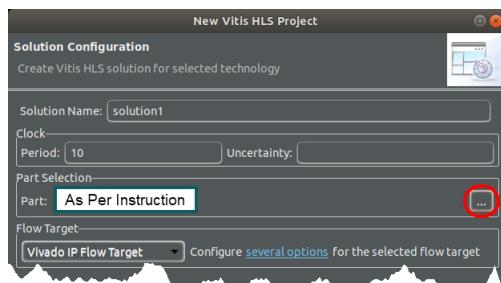
### 1-4. Finally, it is time to specify some of the physical parameters of the design.

By default, your solution name is populated in the Solution Name field. No changes are required.

1-4-1. Set the clock period to **your clock period**.

You can leave the Uncertainty field blank.

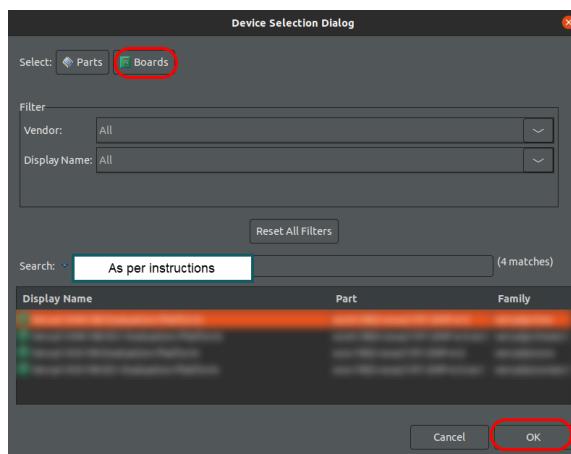
1-4-2. Click the **Browse ( . . . )** icon from the Part Selection option to select a part or board.



**Figure 202: Locating the Board Browse Button**

1-4-3. Click **Boards** as shown below.

1-4-4. Enter the family name in the Search field.



**Figure 203: Filtering to Quickly Locate Target Platforms**

- 1-4-5. Select **your board** from the search list.
- 1-4-6. Click **OK** to select the board.
- 1-4-7. Make sure that **Vivado IP Flow Target** is selected from the drop-down list under the Flow Target field.
- 1-4-8. Click **Finish**.

You will see the created project in the Explorer tab.

## Opening a Vitis HLS Tool Project

---

- 1-1. Open the existing Vitis HLS project *your HLS project name*.

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

The Browse For Folder dialog box opens.

- 1-1-2. Browse to the location at which the project should be placed/your HLS project name directory (2).

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

- 1-1-3. Click **Open** (3).

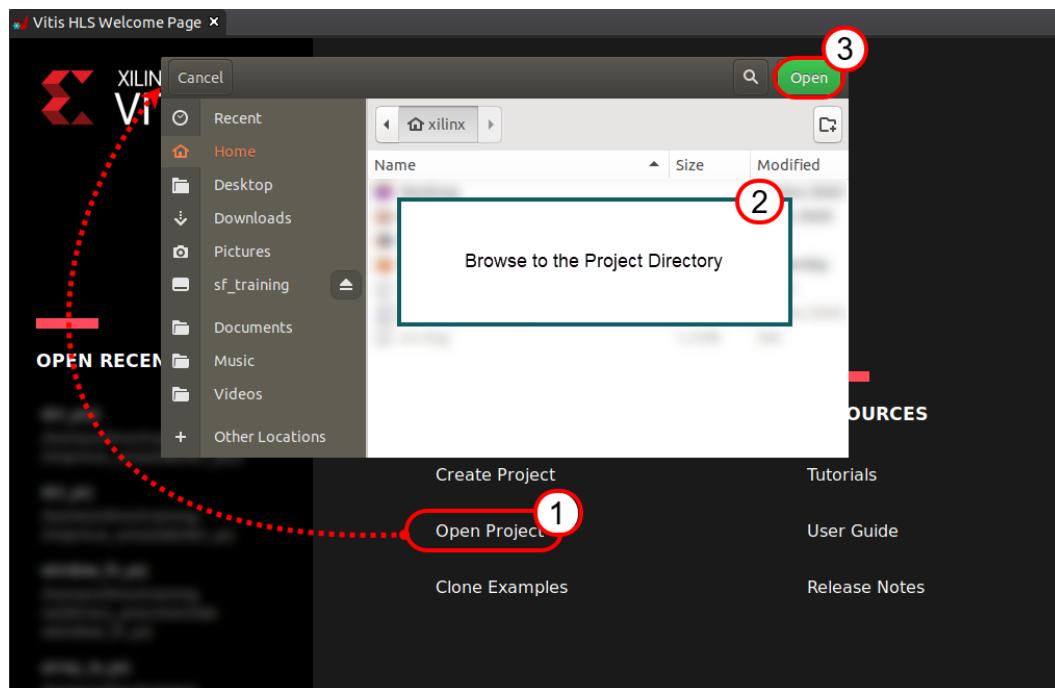


Figure 204: Opening a Vitis HLS Tool Project

The HLS project now opens in the Vitis HLS tool.

## Simulating a Vitis HLS Tool Design

### 1-1. Simulate the Vitis HLS tool design.

#### 1-1-1. Select Project > Run C Simulation.

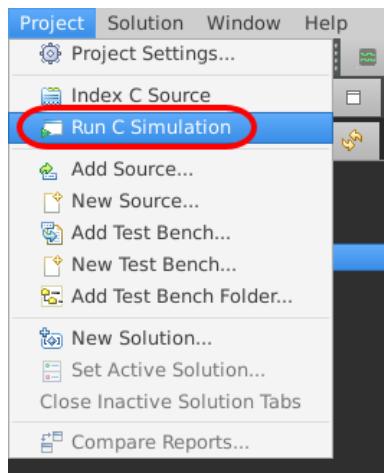


Figure 205: Launching the C Simulation

You can also run C simulation from the Flow Navigator by clicking **Run C Simulation** under C SIMULATION.

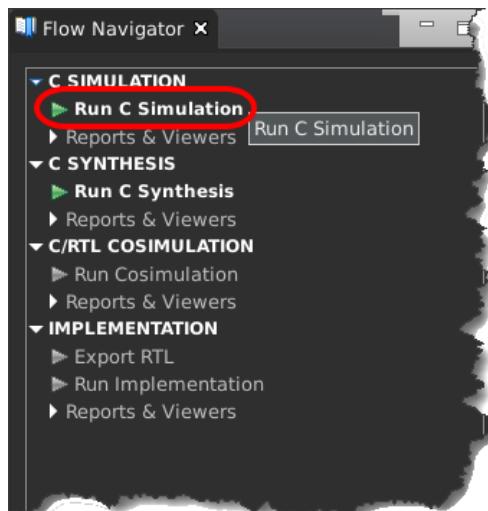
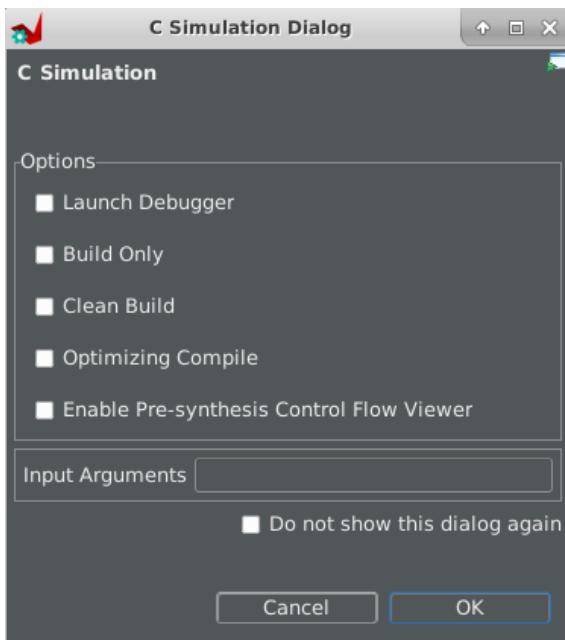


Figure 206: Launching the C Simulation Using the Flow Navigator

The Run C Simulation dialog box opens.



**Figure 207: C Simulation Dialog Box**

Each of the options controls how the simulation is run:

- Launch Debugger: After compilation the debug perspective automatically opens for you to step through the code.
- Build Only: Compiles the source code and the test bench, but the simulation does not run. This option can be used to test the compilation process and resolve any issues with the build before running the simulation.
- Clean Build: Removes any existing executable and object files before compiling the code.
- Optimizing Compile: By default, the design is compiled with debug information enabled, allowing the compilation to be analyzed and debugged. The Optimizing Compile option uses a higher level of optimization effort when compiling the design but does not add the information required by the debugger. This increases the compile time but should reduce the simulation runtime.
- Enable Pre-Synthesis Control Flow Viewer: Generates the Pre-synthesis Control Flow report.
- Input Arguments: Specify any inputs required by your test bench main() function.

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

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

The simulation log will be displayed in the editor pane.

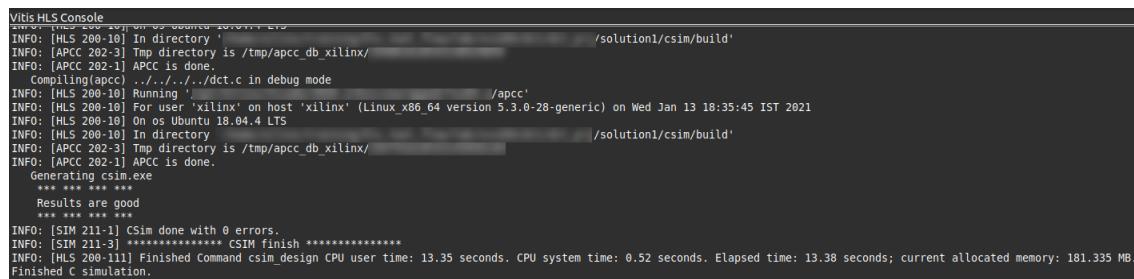
## 1-2. View the simulation report.

The information generated by the Vitis 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, but the full set of simulation engine messages as well. The simulation log provides only key 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.



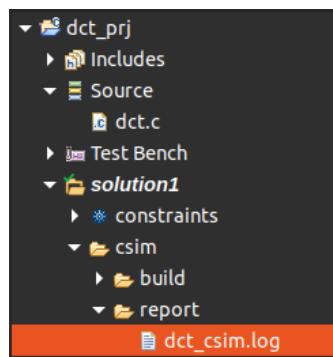
```
Vitis HLS Console
...
INFO: [APCC 200-10] In directory '/tmp/apcc_db_xilinx/'                                     /solution1/csimm/build'
INFO: [APCC 202-3] Tmp directory is /tmp/apcc_db_xilinx/
INFO: [APCC 202-1] APCC is done.
Compiling(apcc) ./dct.c in debug mode
INFO: [HLS 200-10] Running 'apcc'...                                                       /apcc'
INFO: [HLS 200-10] For user 'xilinx' on host 'xilinx' (Linux x86_64 version 5.3.0-28-generic) on Wed Jan 13 18:35:45 IST 2021
INFO: [HLS 200-10] On os Ubuntu 18.04.4 LTS
INFO: [HLS 200-10] In directory '/tmp/apcc_db_xilinx/'                                     /solution1/csimm/build'
INFO: [APCC 202-3] Tmp directory is /tmp/apcc_db_xilinx/
INFO: [APCC 202-1] APCC is done.
Generating csim.exe
*****
Results are good
*****
INFO: [SIM 211-1] Csim done with 0 errors.
INFO: [SIM 211-3] ***** CSIM finish *****
INFO: [HLS 200-111] Finished Command csim_design CPU user time: 13.35 seconds. CPU system time: 0.52 seconds. Elapsed time: 13.38 seconds; current allocated memory: 181.335 MB.
Finished C simulation.
```

**Figure 208: Example Output After Simulation**

The other location, described below, provides only a few simulation engine messages and the simulated code output. Typically, this is opened after the simulation completes; however, if you need to access it after closing the log pane, here's how to access the simulation report.

### 1-2-2. Expand your HLS project name > solution1 > csim > report in the Explorer pane.

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



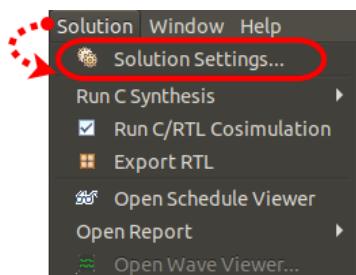
**Figure 209: Locating the Simulation Log File**

## Setting Synthesis Options

Various information relevant to how the tools will synthesize a design can be entered into the Vitis 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.**

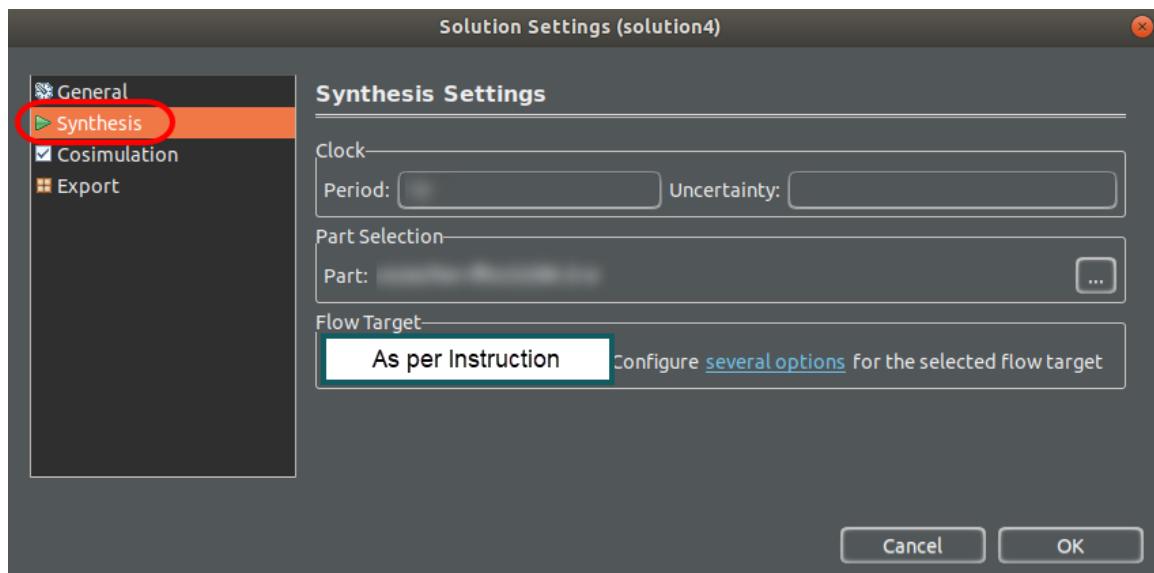
- 1-1-1. Select Solution > Solution Settings.**



**Figure 210: Accessing Synthesis Settings from the Menu Bar**

The Solutions Settings dialog box opens.

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



**Figure 211: 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. Select the Flow Target to **Vivado IP Flow Target**.**
- 1-1-6. Click **OK**.**

## Synthesizing a Vitis HLS Tool Design

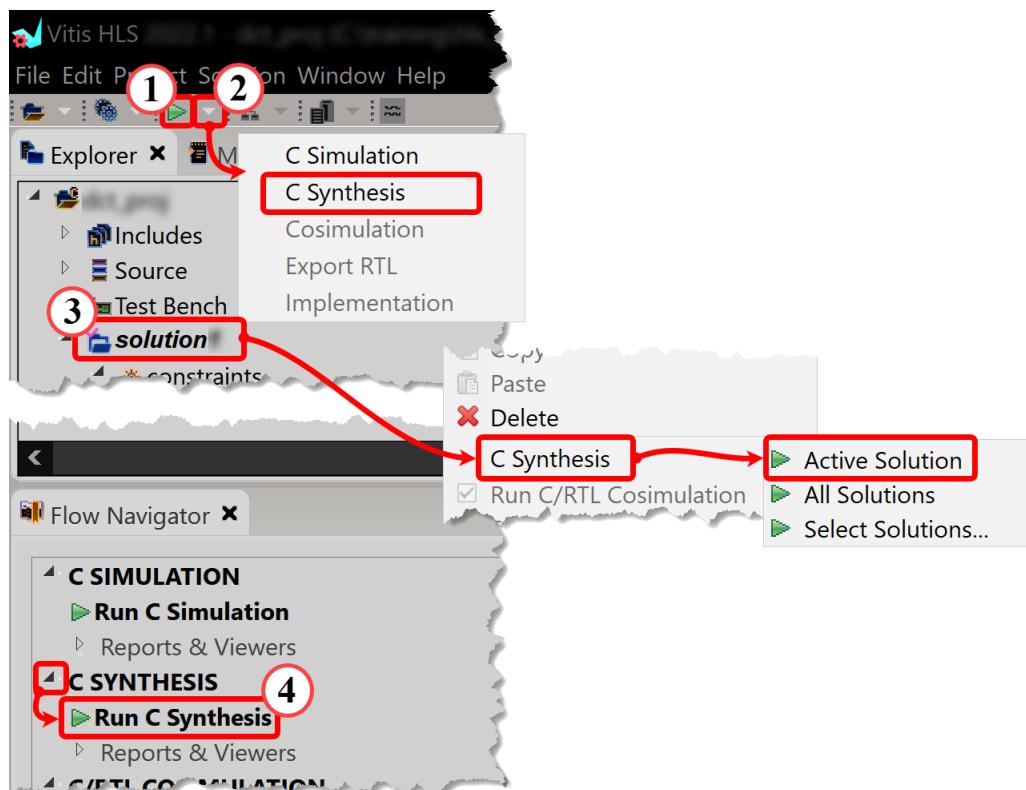
### 1-1. Synthesize the design.

#### 1-1-1. Use one of the following methods to launch the C synthesis tool:

- Click the **Run Flow** icon (▶) in the toolbar (1).
- Click the pull-down next to the Run Flow icon in the toolbar and select **C Synthesis** (2).
- From the Explorer tab, select the desired solution to synthesize and right-click and select **C Synthesis > Active Solution** (3).

**Note:** More complex projects allow multiple solutions to be batched which, is what the other options in this menu support.

- From Flow Navigator, select **C Synthesis > Run C Synthesis** (4).



**Figure 212: Multiple Methods for Launching C Synthesis**

The C Synthesis - Active Solution dialog box opens.

The tool automatically fills in any information it already knows, such as the project's clock period, part selection, and flow type. You can modify these here if needed.

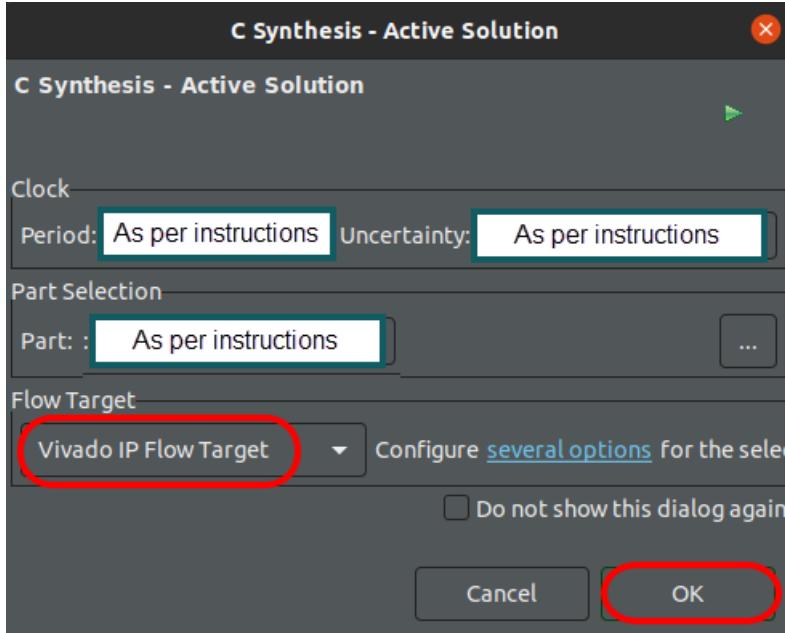


Figure 213: C Synthesis Dialog Box

- 1-1-2. Click **OK** to begin the synthesis process.

## Running Cosimulation in the Vitis HLS Tool

---

### 1-1. Run co-simulation on the solution.

This will compare the simulation results of the C/C++ and RTL designs.

- 1-1-1. Select **Solution > Run C/RTL Cosimulation** to launch the co-simulation.

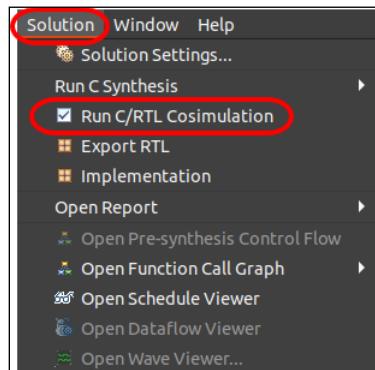


Figure 214: Launching from the Menu

Alternatively, from the Flow Navigator, expand **C/RTL Co-simulation** and click **Run Cosimulation**.

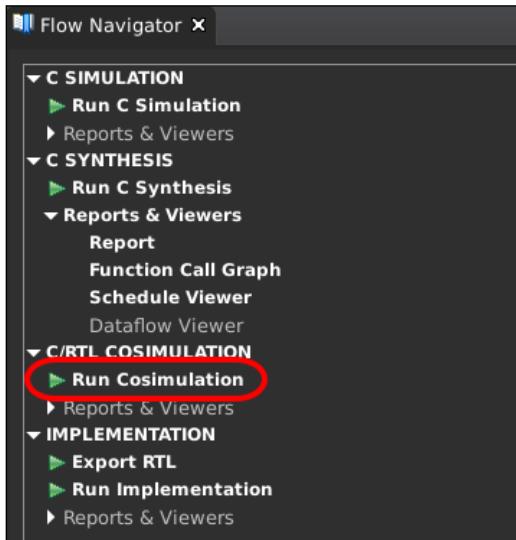


Figure 215: Launching Cosimulation from Flow Navigator

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

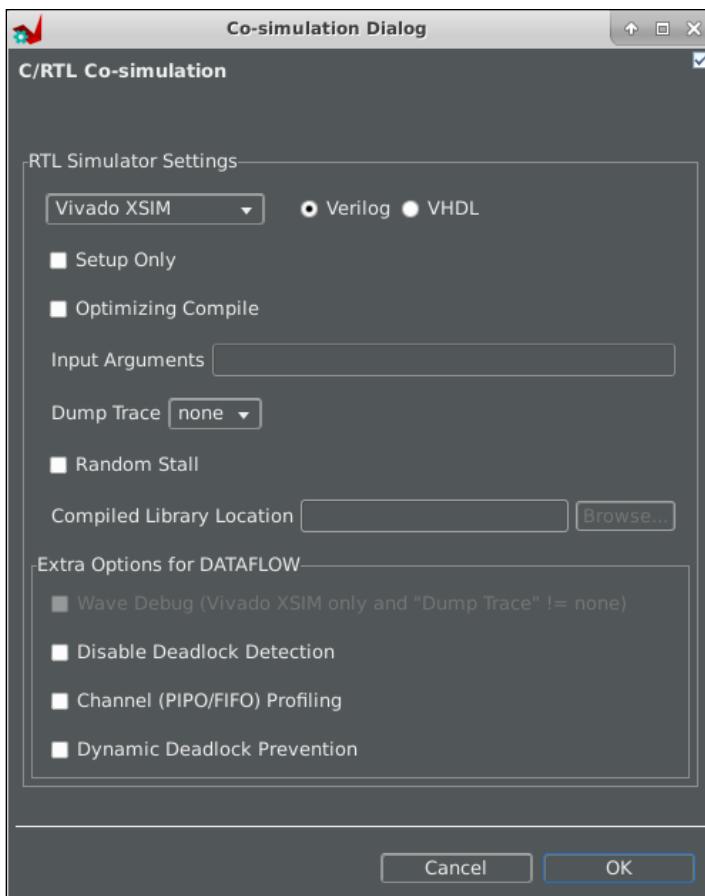


Figure 216: Co-simulation Dialog Box

The dialog box includes the following settings:

- Simulator: Choose from one of the supported HDL simulators in the Vivado Design Suite. The Vivado simulator (Vivado XSIM) is the default simulator. Licenses may be required for other simulators.
- Language: Specify Verilog or VHDL as the output language for simulation.
- Setup Only: Creates the required simulation files but does not run the simulation. The simulation executable can be run from a command shell at a later time.
- Optimizing Compile: Improve the runtime performance through enhanced optimization, if possible, which typically takes longer to compile.
- Input Arguments: Specify additional command-line arguments for the C test bench.
- Dump Trace: Specifies the level of trace file output written to the *sim/Verilog* or *sim/VHDL* directory of the current solution when the simulation executes. Options include:
  - all: Output all port and signal waveform data being saved to the trace file.
  - port: Output waveform trace data for the top-level ports only.
  - none: Do not output trace data.
- Random Stall: Applies a randomized stall for each data transmission.

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

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

The simulation log will be displayed in the Console pane.

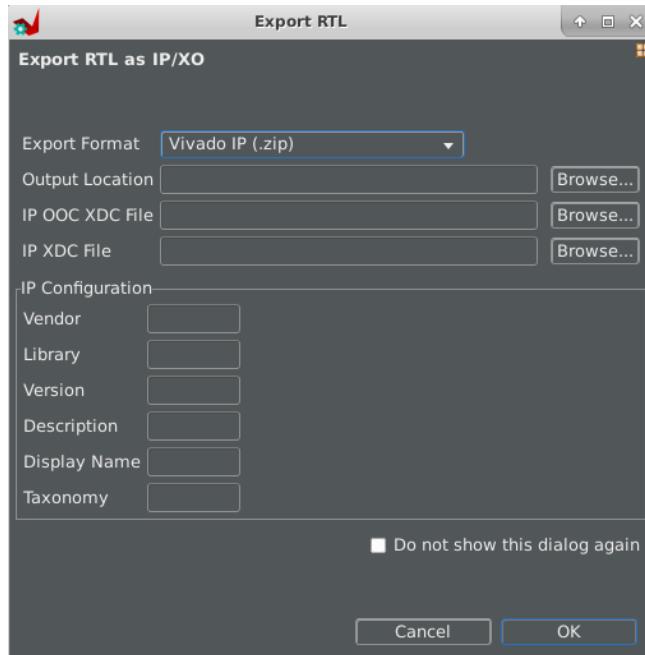
## Exporting a Vitis HLS Tool Design as IP

### 1-1. Export the Vitis HLS tool design as a piece of IP core.

#### 1-1-1. Select **Solution > Export RTL**.

**Note:** If you see the Feedback Request dialog box, click **Cancel**.

#### 1-1-2. Ensure that **Vivado IP (.zip)** is selected from the Format Selection drop-down list.



**Figure 217: Export RTL Dialog Box**

#### 1-1-3. Click **Configuration** next to the Format Selection drop-down list.

Notice that you can provide information about the IP, such as the vendor, library, version, and description in the IP Identification dialog box.



**Figure 218: IP Identification Dialog Box**

#### 1-1-4. Click **Cancel** in the IP Identification dialog box.

1-1-5. Make sure that **Verilog** is selected as the RTL to be evaluated.

1-1-6. Select the **Vivado synthesis, place and route** option.

1-1-7. Keep the rest of the settings at their defaults.

These options will enable Vivado RTL synthesis and implementation to be performed on the generated IP. Implementation is run to evaluate and provide confidence that the RTL will meet its estimated timing and area goals and that these results are not included as part of the exported package.

1-1-8. Click **OK** in the Export RTL dialog box.

You can observe the progress in the Console tab. Once implementation completes, the Implementation report will open in the Information pane.

Observe the "Timing met" message in the report in the Final Timing section. This shows that the final timing of the implemented design has been achieved.

## Vitis Model Composer Operations [Last Updated Version 2022.1]

This section contains instructions for commonly performed tasks using Vitis Model Composer.

### In This Section

---

|  |     |
|--|-----|
| Launching Vitis Model Composer .....             | 172 |
| Simulating a Model in the Simulink Software..... | 173 |

## Launching Vitis Model Composer

---

### 1-1. Launch Vitis Model Composer.

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

1-1-2. Enter the following command to source Vitis Model Composer:

```
[host]$ source /opt/amd/Model_Composer/2023.2/settings64.sh
```

**Note:** The installation path (`/opt/amd`) is valid for the Customer Training VM and CloudShare environments. Modify the path as necessary for your environment.

1-1-3. Enter the following command to set the installed MATLAB software path:

```
[host]$ export PATH=$PATH:<Path_to_your_Matlab_installation>/bin
```

For the Customer Training VM and CloudShare environments, the MATLAB software is installed in the `/home/amd/matlab` path. Enter the following command:

```
[host]$ export PATH=$PATH:/home/amd/matlab/bin
```

1-1-4. Enter the following command to launch Vitis Model Composer:

```
[host]$ model_composer
```

## Simulating a Model in the Simulink Software

### 1-1. Simulate the model.

- 1-1-1. Click the **Run** icon  in the toolbar from the SIMULATION tab in the Simulink software.

## Vitis IDE Operations [Last Updated 2023.2]

### Instructions

#### In This Section

|   |     |
|---|-----|
| Launching the Vitis Unified IDE and Opening Workspace .....             | 174 |
| Opening a Vitis Unified IDE Workspace from the Welcome Window.....      | 176 |
| Creating a C/C++ Application Project from the Vitis Welcome Window..... | 176 |
| Launching the XSCT Tool.....  | 180 |
| Opening the XSCT Console.....   | 181 |
| Running a Tcl Script.....   | 182 |
| Creating a Platform Project.....  | 183 |
| Creating a Platform Project (Versal Device) .....                       | 185 |
| Creating a Bare-Metal Application Component for an Embedded Flow .....  | 189 |
| Creating an Application Project with a Custom Platform .....            | 191 |
| Creating an AI Engine Application Project in the Vitis IDE .....        | 195 |
| Creating a Linux C/C++ Application Project.....                         | 197 |
| Importing Sources to an Application .....                               | 202 |
| Importing an Existing Project into the Vitis IDE .....                  | 202 |
| Viewing/Importing a Peripheral Example Application .....                | 205 |
| Adding Source Files in the Vitis IDE .....                              | 206 |
| Opening a Source File in the Editor .....                               | 207 |
| Finding Text in a Source File .....                                     | 208 |
| Enabling Line Numbering in the Text Editor .....                        | 210 |
| Setting Compiler Options .....  | 210 |
| Setting Compiler Optimization Options .....                             | 214 |
| Adding Symbols to Application Settings .....                            | 216 |
| Building the Projects .....   | 218 |
| Running with a Default Configuration .....                              | 219 |
| Setting Up a Run Configuration for a Linux Application .....            | 220 |
| Setting Up a Run Configuration for a Hardware Target .....              | 227 |
| Setting Up a Run Configuration for an Emulator Target .....             | 229 |
| Creating and Running the Default Debug Configuration.....               | 231 |
| Switching Perspectives .....  | 235 |
| Debugging.....  | 235 |
| Configuring the Vitis Serial Terminal.....                              | 240 |
| Closing the Vitis IDE .....   | 243 |

## Launching the Vitis Unified IDE and Opening Workspace

### 1-1. Launch the Vitis Unified IDE.

- 1-1-1. Click the **Vitis** icon (▣) from the taskbar or desktop to launch the tool.



**Figure 219: Launching the Vitis Unified IDE from the Taskbar**

**Note:** The Vivado (hardware) tool has a similar icon but with inverted colors. The CloudShare and CustEd VM environments display both icons in the toolbar with the Vivado tool icon visible just above the Vitis Unified IDE icon. Make certain that you select the correct icon.

Alternatively, from a Linux terminal window (<Ctrl + Alt + T>), enter the following:

```
source /opt/amd/Vitis/2023.2/settings64.sh; vitis
```

**Note:** This installation path is valid for the CloudShare and CustEd VM environments. Use the proper path for your environment.

The Workspace Launcher creates a workspace that initially contains a thin structure that tracks tool settings in a log file. The workspace includes all types of projects. Switch between workspaces by selecting **File > Open Workspace**.

It is important to note that not all tools included in the Vitis suite of tools are supported under Windows.

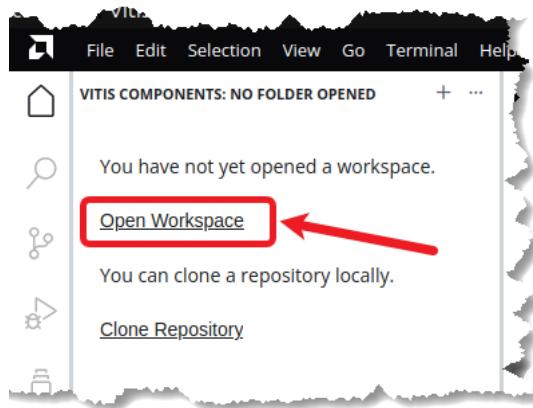
- 1-1-2. If not already maximized, click the Maximize icon the upper-right corner of the tool window so that you can see all the view windows.

## 1-2. Set the workspace.

**Best practices recommends opening a workspace regardless of the flow you are following as it makes it easier to find the-tool generated logs and scripts.**

**Alternatively, you can immediately jump into one of the development flows, where you will eventually be asked to specify a workspace. This causes the log and script files to be placed in two different directories.**

- 1-2-1. From the Vitis Components window, click the **Open Workspace** link.



**Figure 220: Opening the Workspace in the Vitis Unified IDE**

- 1-2-2. Browse to the following location:

```
$TRAINING_PATH/<the topic cluster name>/lab
```

- 1-2-3. Click **OK** to open the new workspace.

The tool relaunches using the new workspace.

- 1-2-4. Close the **Welcome** tab if it appears.

This tab opens various types of projects, enables access to documentation and examples, quickly switches workspaces, etc. If you want to reopen the Welcome tab, select **Help > Welcome**.

## Opening a Vitis Unified IDE Workspace from the Welcome Window

### 1-1. Set the workspace.

Best practices recommends opening a workspace regardless of the flow you are following as it makes it easier to find the tool generated logs and scripts.

Alternatively, you can immediately jump into one of the development flows, where you will eventually be asked to specify a workspace. This causes the log and script files to be placed in two different directories.

- 1-1-1. From the Vitis Components window, click the **Open Workspace** link.

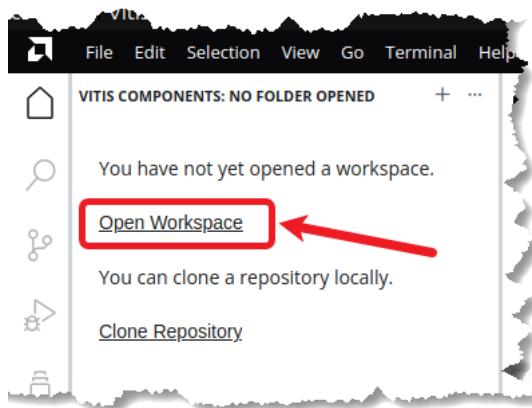


Figure 221: Opening the Workspace in the Vitis Unified IDE

- 1-1-2. Browse to the following location:

```
$TRAINING_PATH/<the topic cluster name>/lab
```

- 1-1-3. Click **OK** to open the new workspace.

The tool relaunches using the new workspace.

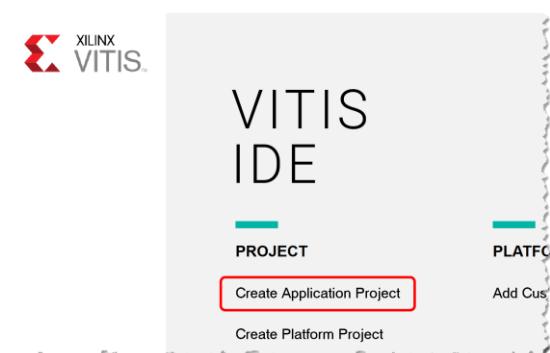
- 1-1-4. Close the **Welcome** tab if it appears.

This tab opens various types of projects, enables access to documentation and examples, quickly switches workspaces, etc. If you want to reopen the Welcome tab, select **Help > Welcome**.

## Creating a C/C++ Application Project from the Vitis Welcome Window

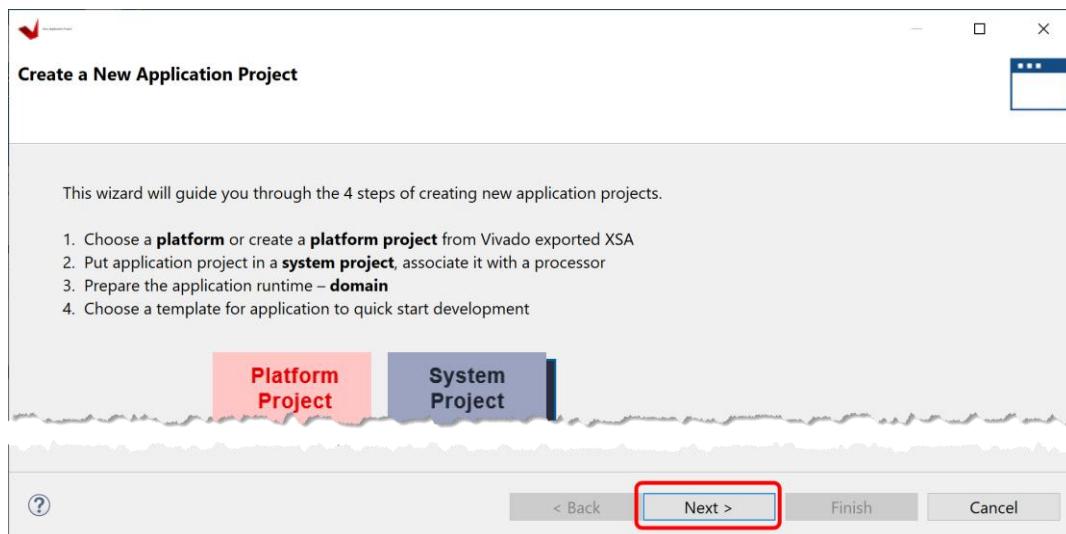
Using the Application Project wizard is a quick way to set up a C or C++ software application project that targets an existing processor and OS platform (standalone or Linux). You can automatically generate the board support package (BSP) or select an existing one. Based on the dialog box choices, the 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**



**Figure 222: Creating an Application Project from the Vitis Welcome Screen**

The Create a New Application Project dialog box appears. This dialog box provides a brief, high-level view of the elements to be created by this wizard.



**Figure 223: Create a New Application Project Overview**

- 1-1-2. Click **Next** to advance to the first page of the wizard.

You must now specify the platform that the application will run on. If the workspace has existing platforms, they will be listed here.

If there is a hardware description available as an XSA file (which is exported from the Vivado Design Suite), you can add it to the list shown here or select an existing platform from the list.

Alternatively, if you do not have an XSA file, you can choose to build a new platform based on the PS found in one of the provided XSA options. These options are available from the *Create a new platform from hardware (XSA)* tab.

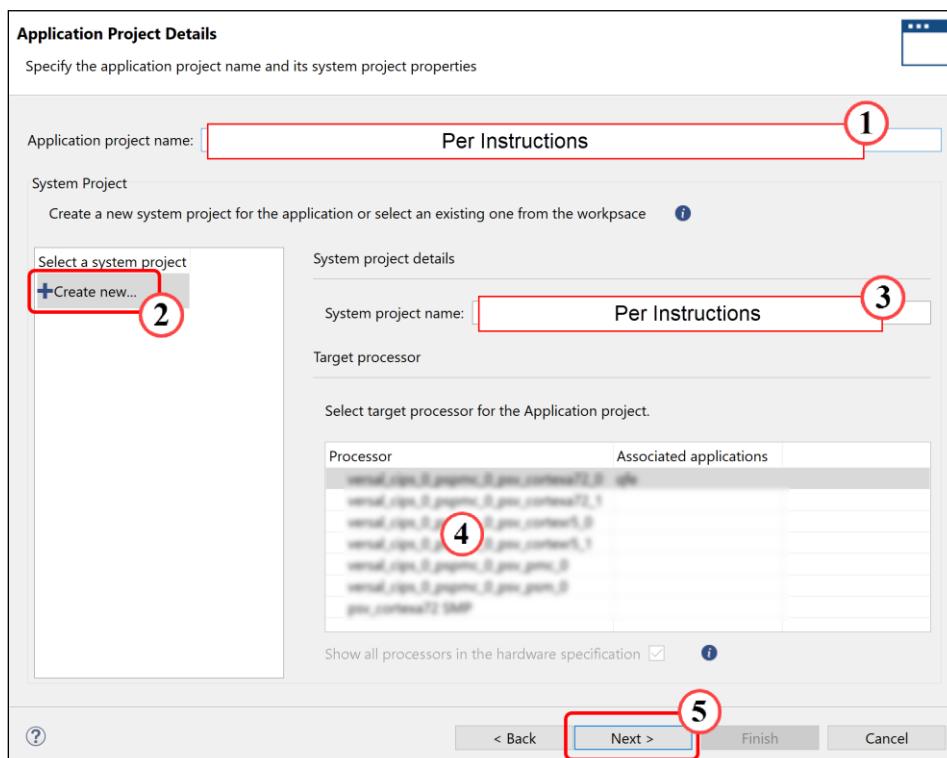
Let's assume that there aren't any user-defined XSA files.

- 1-1-3. Select the **Create a new platform from hardware (XSA)** tab.
- 1-1-4. Select your board from the pre-built board descriptions.
- 1-1-5. Click **Next** to proceed to begin defining the application project.
- 1-1-6. Enter **your application project name** as the application project name (1).

You can select an existing system project if one is available.

- 1-1-7. If not, click **Create new** to create a new system project (2).
- 1-1-8. Name the system project **your system project name** (3).
- 1-1-9. Select a desired processor to associate an application with a processor (4).

This will tell the Vitis IDE which toolchain to use.



**Figure 224: Specifying Application and System Projects**

- 1-1-10. Click **Next** to advance to selecting or creating a domain for the application (5).

Remember that a domain is a container for a BSP or operating system.

You can select an existing domain or create a new domain for this application.

**1-1-11.** Click **Create new** to build a new domain (1).

**1-1-12.** Enter **the name of your domain** in the Name field (2).

The display name will automatically assume the same name as the Name field; however, you can override this by entering a display name into this field (3).

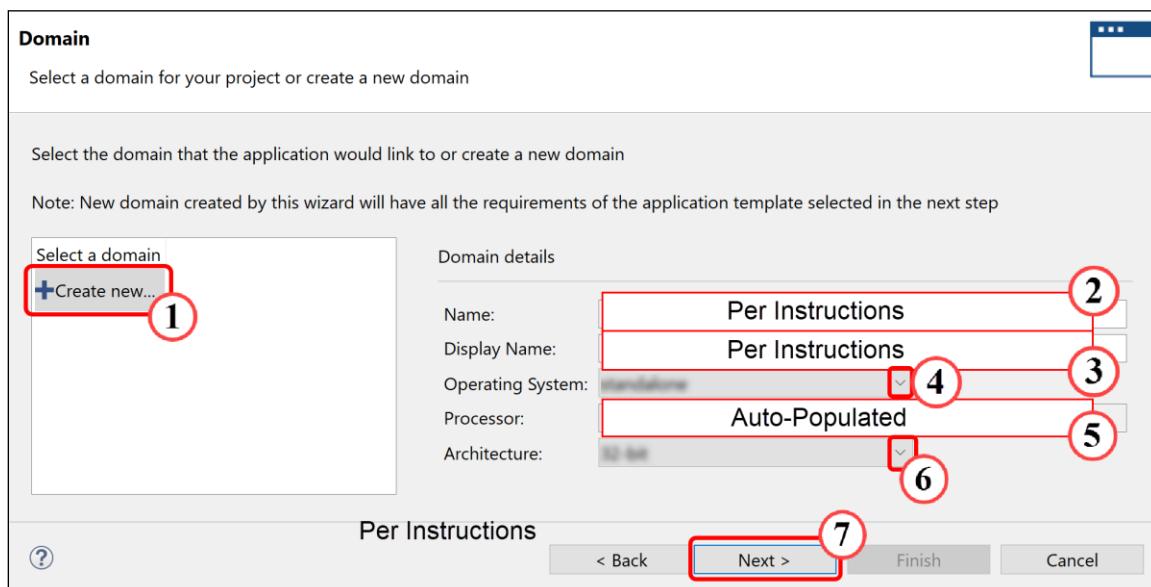
Multiple operating systems are available for most processors.

**1-1-13.** Select **the name of your operating system** from the Operating System drop-down list (4).

The processor name is automatically filled in from the previous page (5).

Some processors can run under multiple architecture configurations, such as 64-bit processors being able to run in 32-bit mode.

**1-1-14.** Select **your desired bit width** from the Architecture drop-down list (6).



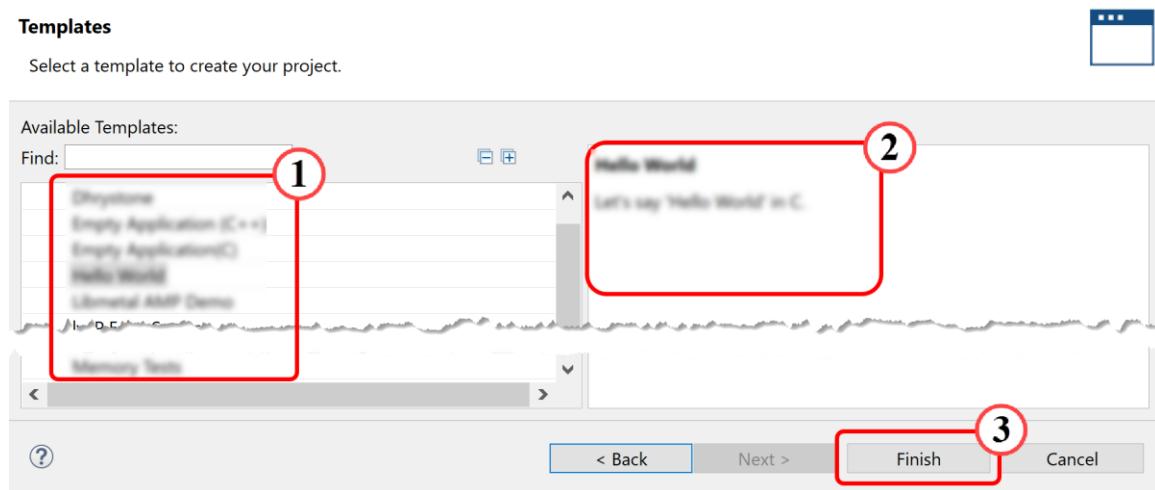
**Figure 225: Configuring the Domain**

**1-1-15.** Click **Next** to advance to the Templates page (7).

This final page allows you to select one of a number of application templates, including blank templates (Empty Application) for both C and C++ code. These Empty Application templates are ideal for importing your own source code from other projects.

**1-1-16.** Select **desired template** from the list (1).

A description of the selected template appears in the right-hand pane (2).



**Figure 226:** Selecting a Template for the Application Project

**1-1-17.** Click **Finish** to build the projects (3).

## Launching the XSCT Tool

### 1-1. Launch the XSCT Tool.

**1-1-1.** Click the **XSCT** icon from the desktop or toolbar to launch XSCT.

Alternatively, you can press **<Ctrl + Alt + T>** to launch a terminal and enter **xsct**.

The XSCT tool opens.



**Figure 227:** XSCT Opening Screen

You can now enter Tcl commands into the tool after the **xsct%** prompt.

## Opening the XSCT Console

Access the XSCT console in the Vitis IDE via a command terminal window. This console enables you to enter Tcl commands directly into the environment, allowing you to configure Vitis tool projects without the use of the GUI.

Having Tcl commands available is especially useful when you are building a design up to a point, performing repetitive processes, or rebuilding a design from a well-documented, step-by-step process.

Here you will explore using the XSCT console in the Vitis IDE tool.

### 1-1. Open the XSCT console so that you can enter Tcl-based commands directly into the Vitis IDE.

The process shown here illustrates how any view can be opened, using the XSCT Console as an example. If you are already familiar with this process, you can click the XSCT Console icon (terminal) to bypass the lengthier process and continue with the next instruction.

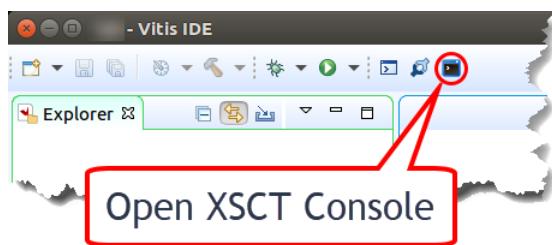


Figure 228: Opening the XSCT Console

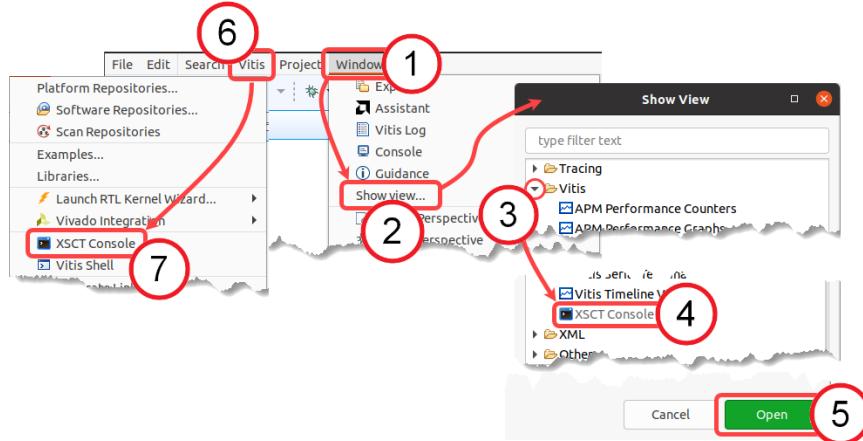
1-1-1. Select **Window** (1) > **Show view** (2) to see a list of commonly opened views.

1-1-2. Expand **Vitis** (3) to access the Vitis tool-specific views.

1-1-3. Select **XSCl Console** (4).

1-1-4. Click **Open** to open the XSCl Console view (5).

Alternatively, some common tasks are listed from the **Vitis** menu (6). Here you can directly select **XSCl Console** (7).



**Figure 229: Opening the XSCl Console - Two Methods**

Typically, this console opens next to the group of tabs in the Build console. You can also click-drag the tab to other locations or resize it for easier use.

## Running a Tcl Script

---

1-1. Run the provided Tcl script.

Enter the following Tcl commands into the XSCl console's Tcl command line.

1-1-1. If the XSCl console is not open in the Vitis IDE, click the XSCl icon (monitor icon) in the toolbar.

The XSCl console can also be opened via **Vitis** > **XSCl Console**.

1-1-2. Enter the following command to change the directory to the location of the Tcl script you want to run:

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

The \$::env() notation is used to extract the value of a variable from the environment into the Tcl shell.

1-1-3. Enter the following command to run the script:

```
source your Tcl script
```

This loads the Tcl environment with the procs contained in this Tcl script and run any Tcl code not included in the procs.

## Creating a Platform Project

A platform component contains a thorough description of a hardware design, including:

- Which processors are available
- The PL-based peripherals and the enabled PS peripherals
- A full system memory map

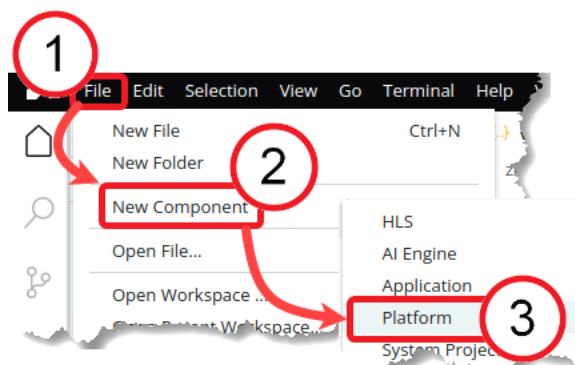
Based on this description, software, such as the board support package (BSP), and applications can be tailored to the hardware.

Normally, you would use your hardware description file, which is typically generated by the Vivado Design Suite; however, there are situations when you just want to quickly test your code using the standard peripheral set for the supported device. Choose from the provided collection of predefined hardware templates for various boards.

The following instructions illustrate how to include custom hardware.

### 1-1. Create a platform component.

- 1-1-1. Navigate to the toolbar and click **File** (1).
- 1-1-2. Hover over or click **New Component** to open its context menu (2).
- 1-1-3. Select **Platform** from the options (3).



**Figure 230: Opening the Create Platform Component Wizard**

The Create Platform Component dialog box appears.

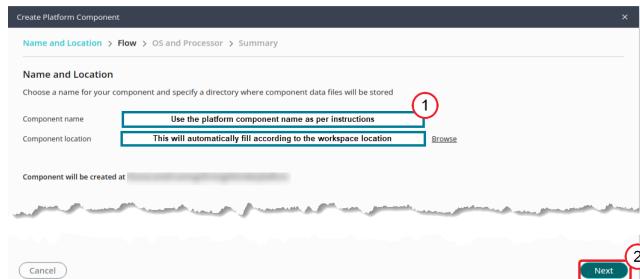
**Note:** There are other ways to access this wizard; however, this is the easiest.

**1-1-4.** Enter **your platform project name** in the Component name field (1).

**Hint:** Make certain that there aren't any spaces in this field.

This sets the name of the platform component. The associated directories and files are populated within the current workspace.

**Note:** The Component location field is automatically set to the workspace location. The location can be changed if you want to place this component somewhere else.



**Figure 231: Entering the Platform Component Name and Location**

**1-1-5.** Click **Next** to advance to the next page of the wizard (2).

**1-1-6.** Select the **Hardware Design** option so that you can import an embedded design from one of the supplied boards or one of your own creation (1).

**1-1-7.** Click **Browse** next to the Hardware Design (XSA) field (2).

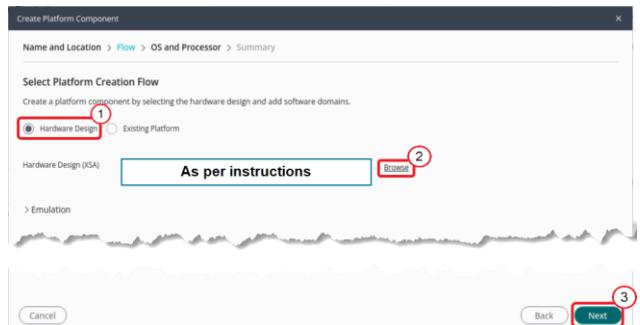
**1-1-8.** Browse to the following directory:

the location of the files to be exported from the Vivado Design Suite

**1-1-9.** Double-click **your hardware platform description name.xsa**.

Remember that the Vitis Unified IDE does not automatically expand the environment variables. You must type out or navigate to the path. For more details about environment variables, refer to the Introduction section of this lab.

**1-1-10.** Expand **Emulation** so that you can identify an XSA file for emulation.



**Figure 232: Specifying the XSA File**

**1-1-11.** Click **Next** to select the operating system and processor (3).

**1-1-12.** If not already selected, select **the operating system** from the Operating system drop-down list (1).

**1-1-13.** If not automatically populated, select **the processor you wish to target** from the Processor drop-down list (2).

Depending on the processor selection, additional options may appear, enabling you to select the processor's instruction set architecture or provide options for boot artifacts. Unless there is a specific reason to do so, leave this at its default setting.

**Note:** Zynq UltraScale+ MPSoC-based boards will allow you to select the architecture, whereas this option is not available for Versal devices.

**1-1-14.** Select the appropriate boot components as follows (3):

**Note:** The following platforms are listed here for completeness. Follow the guidance for the board or device you selected.

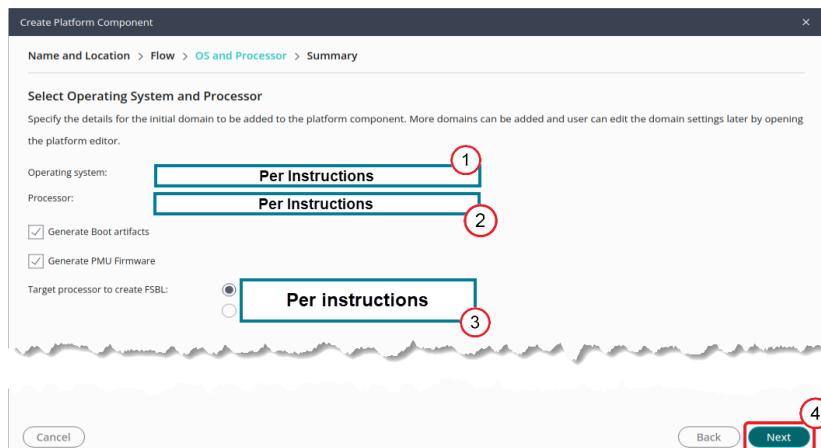
**[Versal users]:** This field is not available for this target—continue with the next task.

**[MPSoC users]:** Select **Target processor to create FSBL**, which generates an FSBL that runs on the specified processor. Under most circumstances, it doesn't matter which processor is chosen.

Select **Generate PMU Firmware**, which generates the firmware for the PMU present in Zynq UltraScale+ MPSoCs.

**[Zynq 7000 users]:** Select **Generate boot artifacts**. Since there is only one type of processor in this device's PS, the choice to select a processor type is unavailable.

**Note:** The boot process is handled differently in Versal devices; hence the absence of this field when devices are selected.



**Figure 233: Specifying the OS and Target Processor for the Platform Project**

**1-1-15.** Click **Next** to proceed to the Summary section (4).

**1-1-16.** Verify the information for the platform component and then click **Finish**.

**Note:** It takes a few minutes to process. The status is usually indicated in the bottom-left corner, and progress messages appear in the Output window.

Along with the platform component, the wizard creates a domain within the platform component based on the selected operating system and processor.

## Creating a Platform Project (Versal Device)

The platform project contains a thorough description of the hardware design, including:

- The types of processors that are present
- The active peripherals in the fabric and dedicated silicon portions of adaptive SoC-based systems
- A full system memory map

Based on this description, software, such as the board support package (BSP), and applications can be tailored to the hardware.

Normally you would use your own hardware description file, which is typically generated by the Vivado Design Suite; however, there are situations when you just want to quickly test your code using the standard peripheral set in a supported device. You can choose from the provided collection of predefined hardware templates for various boards.

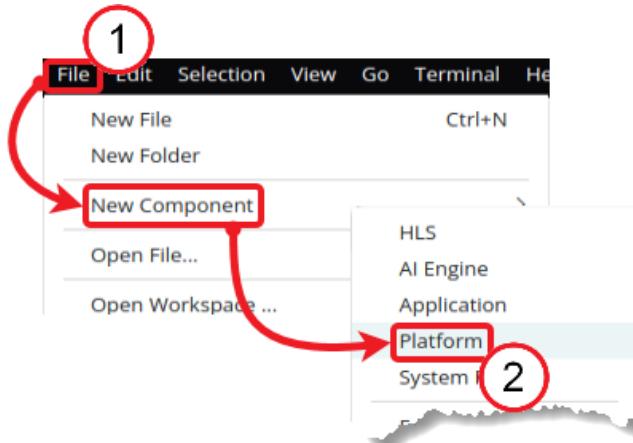
The following instructions illustrate how to include custom hardware.

### 1-1. Create the platform project.

- 1-1-1. Click the arrow (▼) next to the Create New icon (1).

A pop-up window appears showing the available wizards.

- 1-1-2. Select **Platform Project** (2).



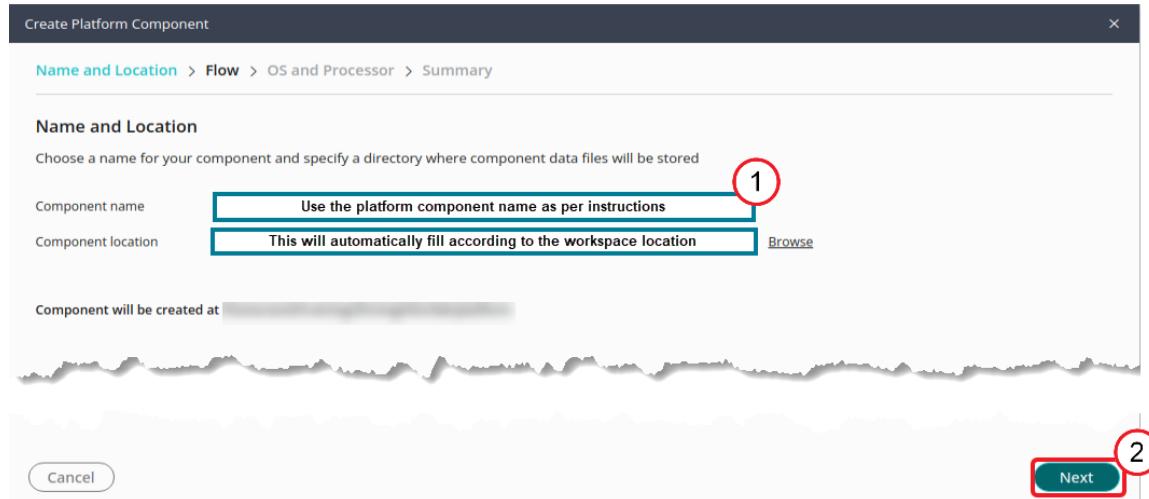
**Figure 234: Opening the Create Platform Component Wizard**

The New Platform Project dialog box appears.

**Note:** There are other ways to access this wizard; however, this is the easiest.

**1-1-3.** Enter **your platform project name** in the Platform project name field (1).

This sets the name of the platform project, and the associated directories and files are then populated within the current workspace.

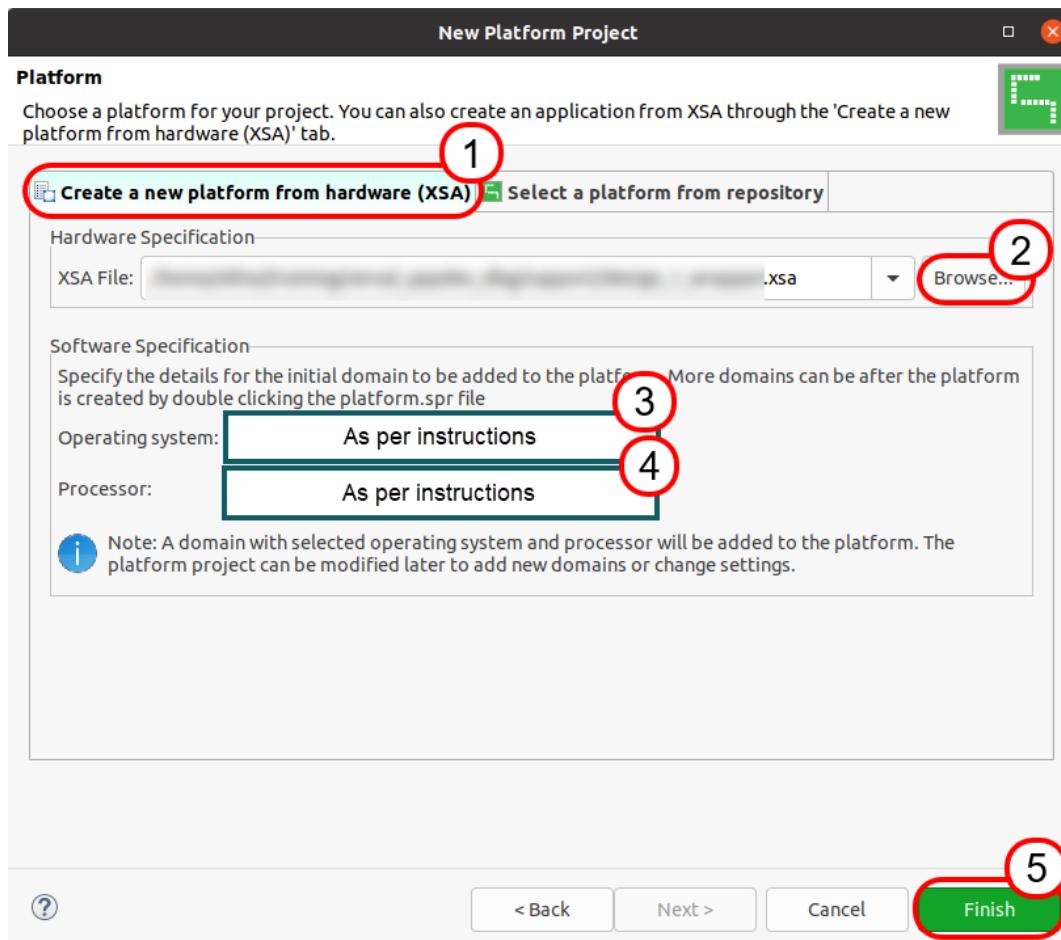


**Figure 235:** Entering the Platform Component Name and Location

**1-1-4.** Click **Next** to advance to specifying the location of the hardware description file (2).

- 1-1-5. Ensure that the **Create a new platform from hardware (XSA)** tab is selected (1).
- 1-1-6. Click the **Browse** button (2).
- 1-1-7. Navigate to the location of the files to be exported from the Vivado Design Suite and select **your hardware platform description name.xsa**. Remember that the Vitis IDE does not automatically expand the environment variables. You must type out or navigate to the path. If you have questions about any environment variable, please refer to the Introduction section in this lab.
- 1-1-8. If not already selected, select **the operating system** from the Operating system drop-down list (3).
- 1-1-9. If not automatically populated, select **the processor you wish to target** from the Processor drop-down list (4).

Depending on the processor selection, an additional option may appear.



**Figure 236: Specifying the XSA File, OS, and Target Processor for the Platform Project**

- 1-1-10. Click **Finish** to create the new platform project (5).

Along with the platform project, the wizard creates a domain within the platform project based on the selected operating system and processor.

## Creating a Bare-Metal Application Component for an Embedded Flow

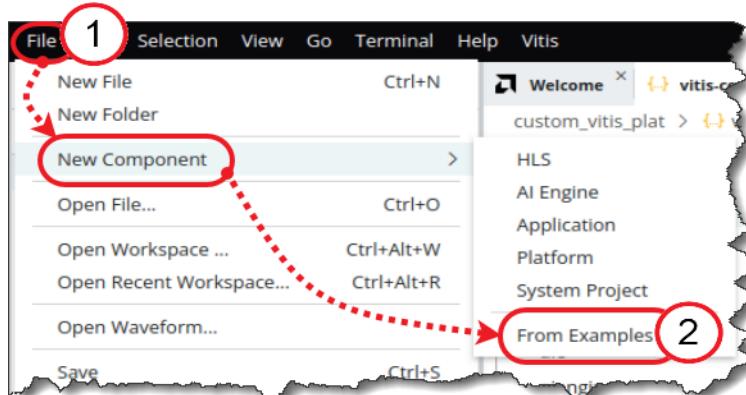
Using the Create Application Component Wizard is a quick way to create an application after a platform component is created.

Based on the dialog box choices, the appropriate toolchain is selected for pre-processing, compiling, assembling, and linking.

### 1-1. Create an application component.

1-1-1. Navigate the toolbar and click **File** (1).

1-1-2. Hover over **New Component** and select **From Examples** from the options (2).

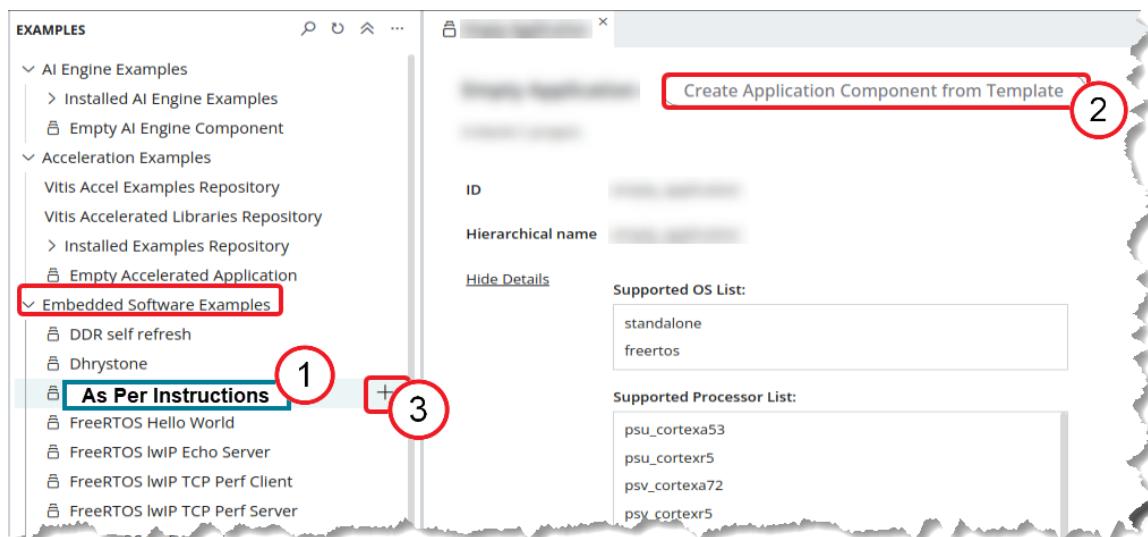


**Figure 237: Creating an Application Component from Examples**

The Examples window becomes visible.

1-1-3. Select **application template** under Embedded Software Examples (1).

1-1-4. Click **Create Application Component from Template** to create the application component (2).



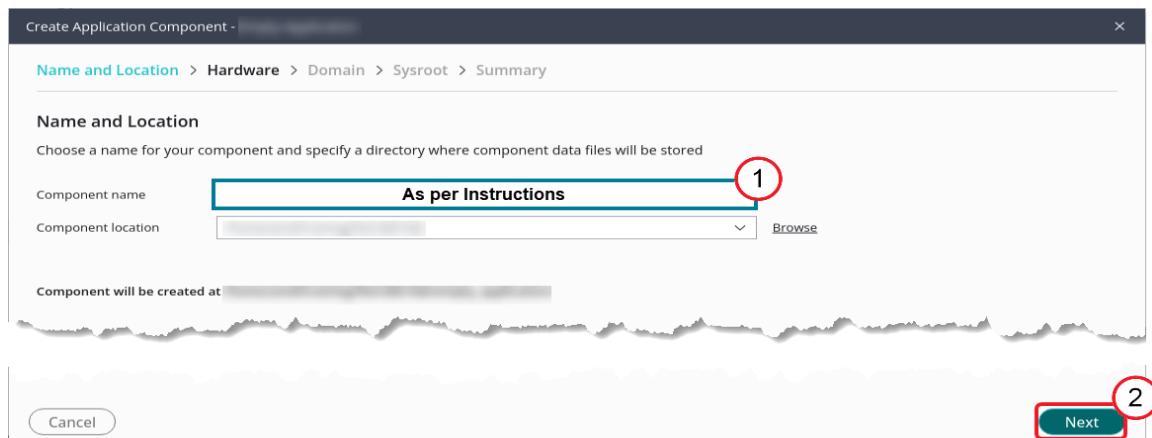
**Figure 238: Selecting an Example Template**

The Create Application Component - application template dialog box appears.

### 1-1-5. Enter **your application project name** in the Component name field (1).

**Hint:** Make certain that there aren't any spaces in this field.

This sets the name of the application component. While the "app" suffix is not required, using it simplifies identifying the component type



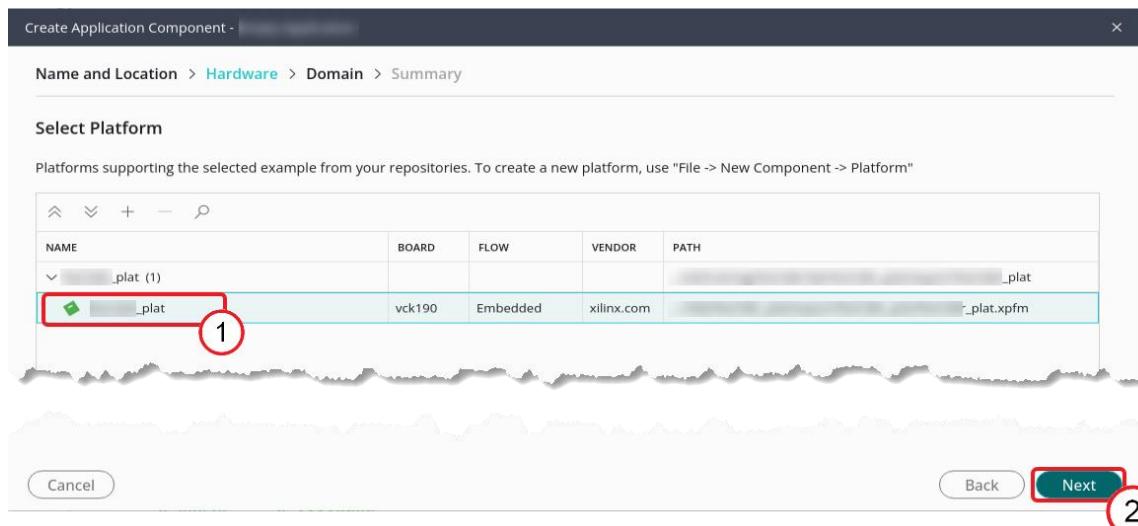
**Figure 239: Naming the Application**

### 1-1-6. Click **Next** to continue with the process (2).

You will now associate the application with a platform and domain.

### 1-1-7. Select **your platform project name** from the list (1).

**Note:** The platform information view will be populated with values from the selected platform.

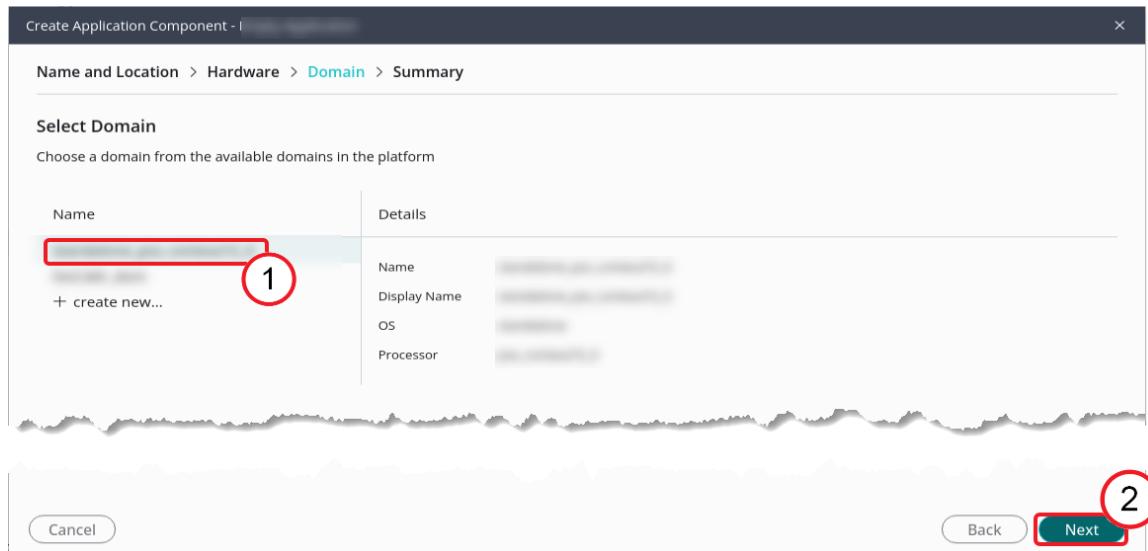


**Figure 240: Selecting the Platform Component**

### 1-1-8. Click **Next** to continue to the domain selection (2).

You can now associate this application project with an existing domain or create a new domain.

- 1-1-9.** If not already selected, select **your domain name** to link to the application project (1).



**Figure 241:** Associating the Application with a Domain

- 1-1-10.** Click **Next** to continue to view the summary (2).

- 1-1-11.** Click **Finish** to build the application project.

**Note:** This assembles all the project pieces but does not compile the application project.

## Creating an Application Project with a Custom Platform

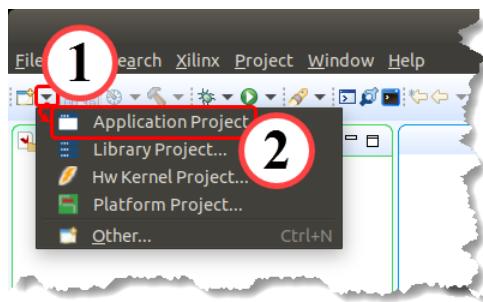
Using the Application Project Wizard is a quick way to create an application for a system project in the Vitis IDE. Based on the dialog box choices, the appropriate toolchain is selected for pre-processing, compiling, assembling, and linking.

### 1-1. Create a new application targeted for the the processor you wish to target.

- 1-1-1.** Click the arrow (▼) next to the Create New icon (1).

A pop-up window appears showing the available wizards.

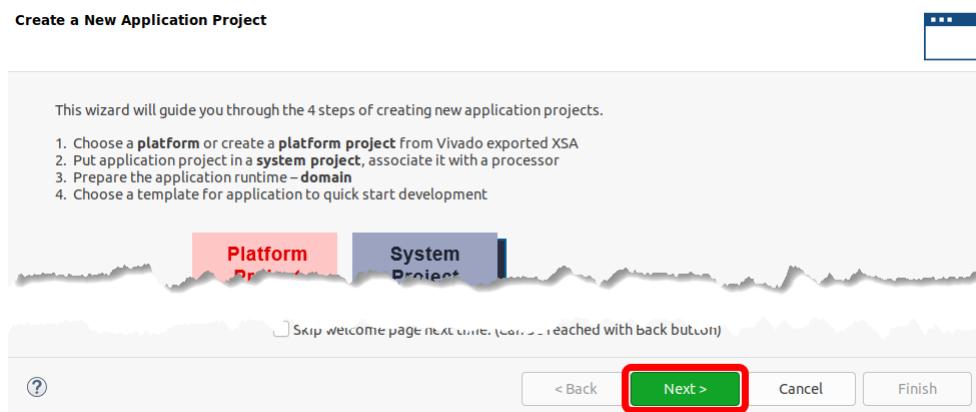
- 1-1-2.** Select **Application Project** to start the Application Project Wizard (2).



**Figure 242:** Opening the Application Project Wizard

**Note:** There are other ways to access this wizard; however, this is the easiest.

The Create a New Application Project window appears and outlines the next steps that you will be guided through during the application creation process.

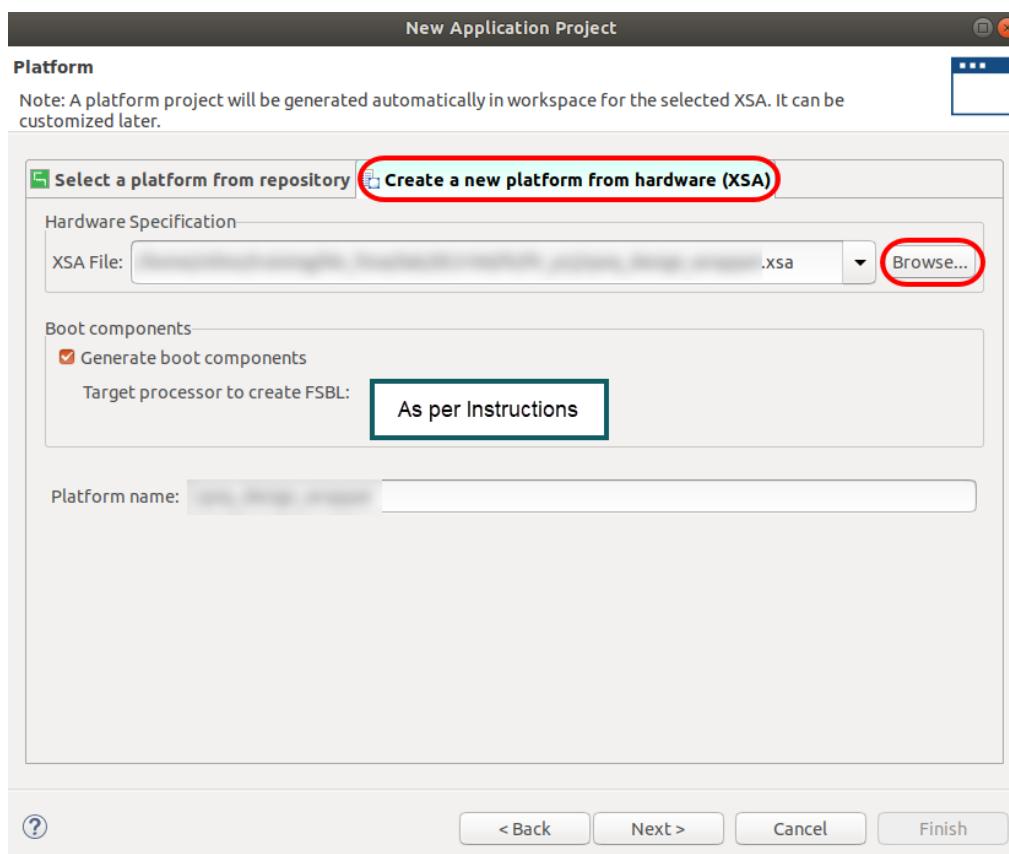


**Figure 243: Creating a New Application Project Wizard - Overview of the Process**

- 1-1-3.** Click **Next** to continue with the process.

You will now associate the application project with a platform and domain.

- 1-1-4.** Select the **Create a new platform from hardware (XSA)** tab to add a new custom platform.



**Figure 244: Creating a New Platform**

**1-1-5.** Browse to the **project directory** from the XSA File field.

**1-1-6.** Select **your hardware platform description name** file.

The Platform information view will be populated with values from the selected platform. You can keep the default platform project name or change it from the Platform name field.

**1-1-7.** Click **OK**.

**1-1-8.** Ensure that the **Generate boot components** option is enabled if that option is available for the selected hardware platform.

**1-1-9.** Select **the processor you wish to target** as the target processor to create the FSBL field.

**1-1-10.** Click **Next** to continue.

**1-1-11.** Enter **your application project name** in the Application project name field (1).

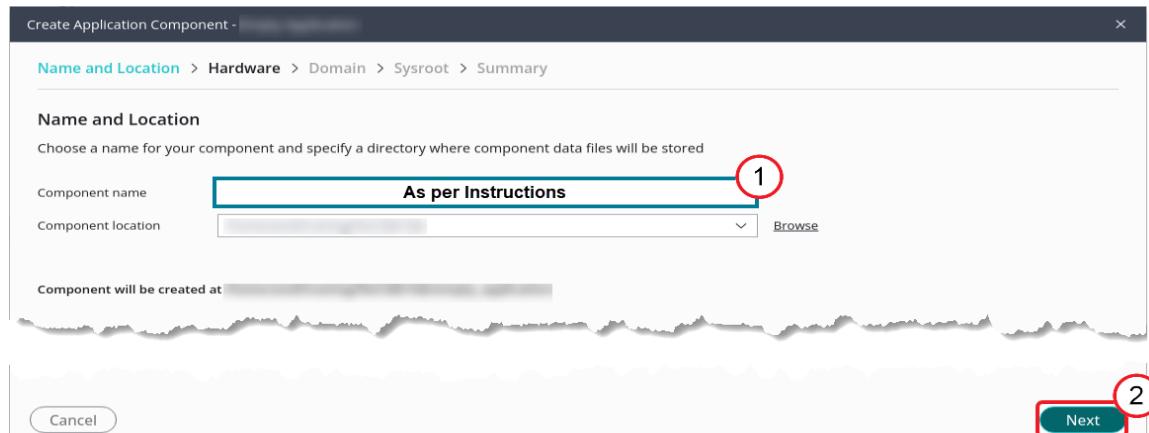
**Hint:** Make certain that there aren't any spaces in this field.

This sets the name of the application project. The wizard will automatically populate the System project name field.

You can also select **Create New** from the System project list (2) and change the system project name (3).

**1-1-12.** If not already selected, choose **your processor** as the target for this application (4).

**Note:** This list is derived from the defined domains in the platform project.

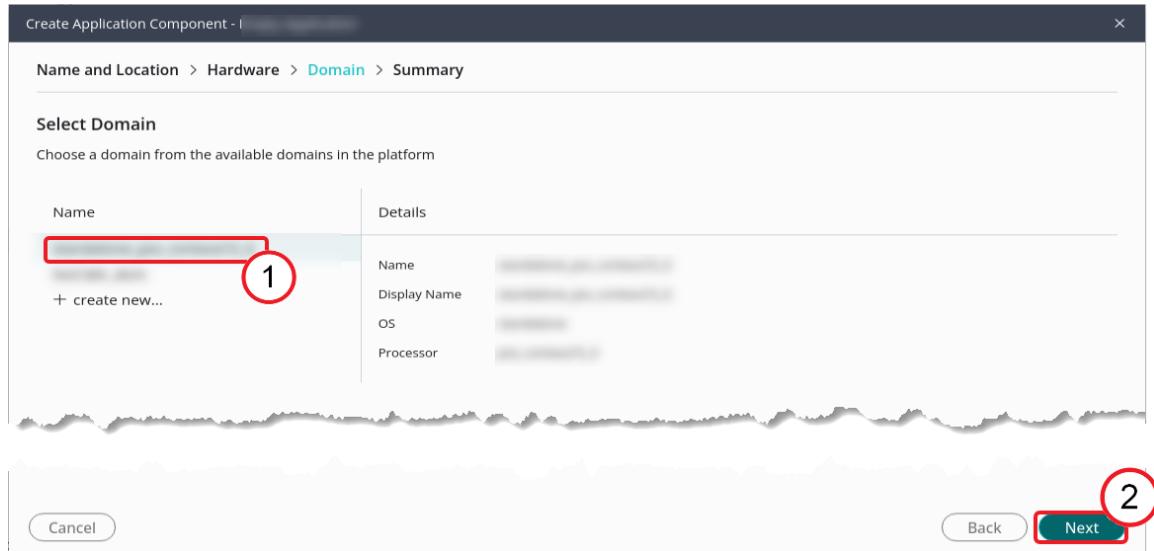


**Figure 245: Naming the Application**

**1-1-13.** Click **Next** to continue to the domain selection (5).

You can now associate this application project with an existing domain or create a new domain.

**1-1-14.** Keep the default settings in the Domain page (1).

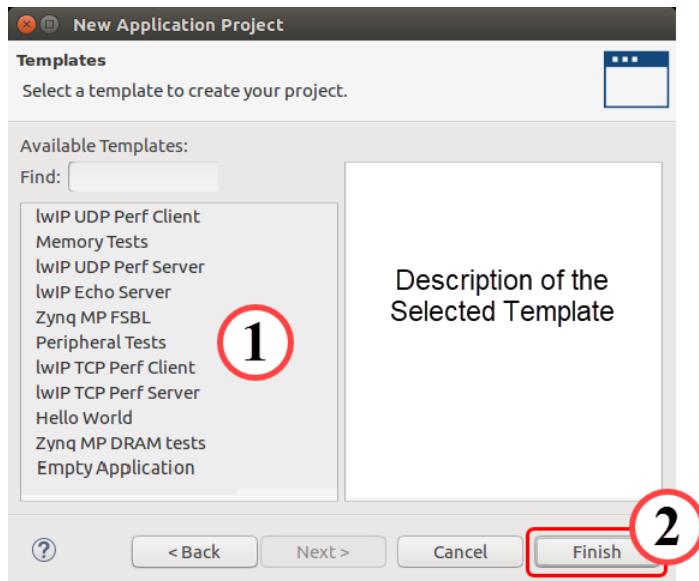


**Figure 246:** Associating the Application with a Domain

**1-1-15.** Click **Next** to continue to the selection of the application template (2).

**1-1-16.** Select the **application template** template from the list (1).

Note that the available templates will change based on the OS choice and processor type.



**Figure 247:** Selecting the Application Template

**1-1-17.** Click **Finish** to build the application project (2).

**Note:** This assembles all the project pieces but does not compile the application project.

## Creating an AI Engine Application Project in the Vitis IDE

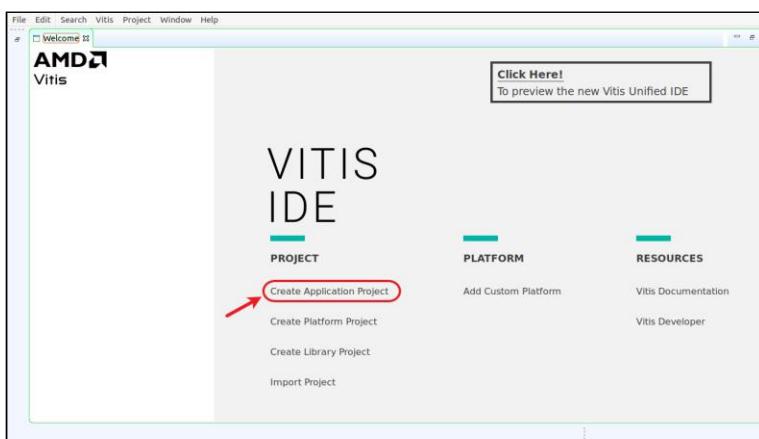
Using the Application Project Wizard is a quick way to create an application for a system project and tie it to a domain project.

Based on the dialog box choices, the appropriate toolchain is selected for pre-processing, compiling, assembling, and linking.

### 1-1. Create an AI Engine application project in the Vitis IDE.

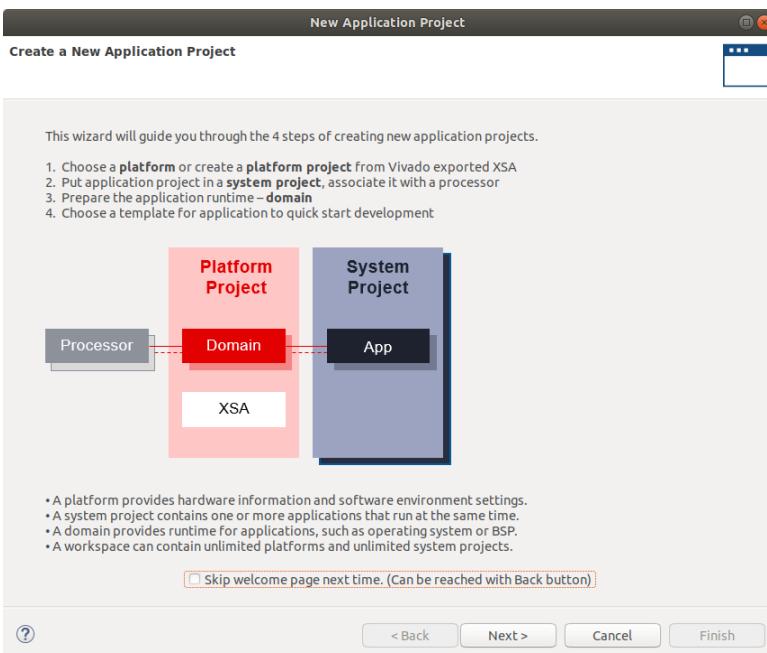
#### 1-1-1. Select **Create Application Project** under PROJECT.

Alternatively, you can select **File > New > Application Project**.



**Figure 248: Selecting Create Application Project**

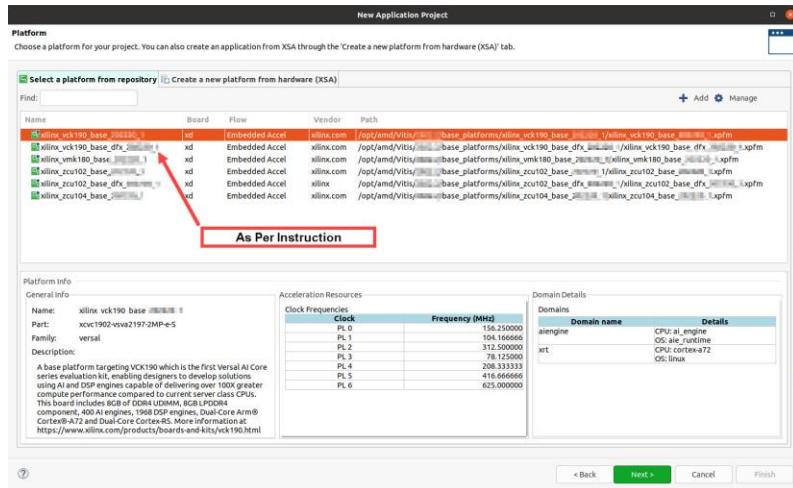
This will display the Welcome page of the New Application Project Wizard.



**Figure 249: New Application Project Wizard**

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

**1-1-3. Select your board platform from the list in the *Select a platform from repository* tab.**



**Figure 250: Selecting the Target Platform**

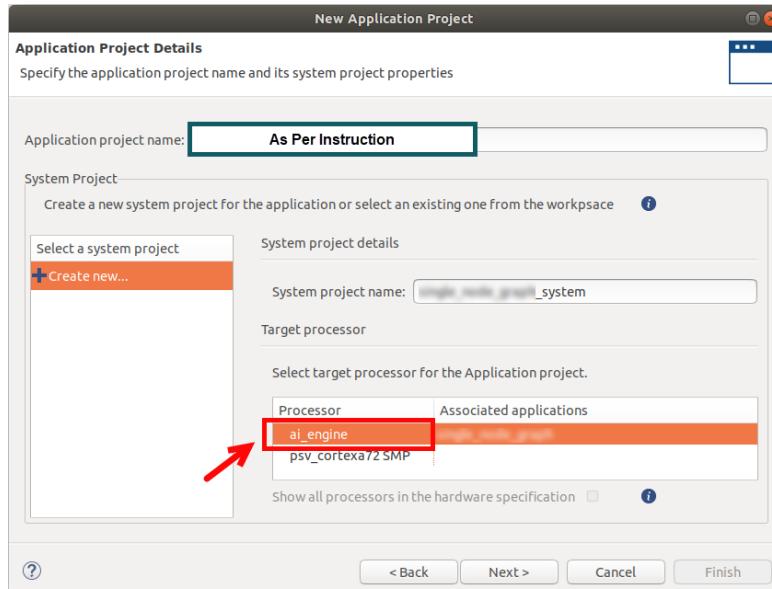
You will see information on the platform, such as device details, clock frequencies, and domain details.

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

**1-1-5. Enter the application project name as **your project name** in the Application project name field.**

Notice that the System project name will be updated automatically after you enter the application project name, but this can be modified.

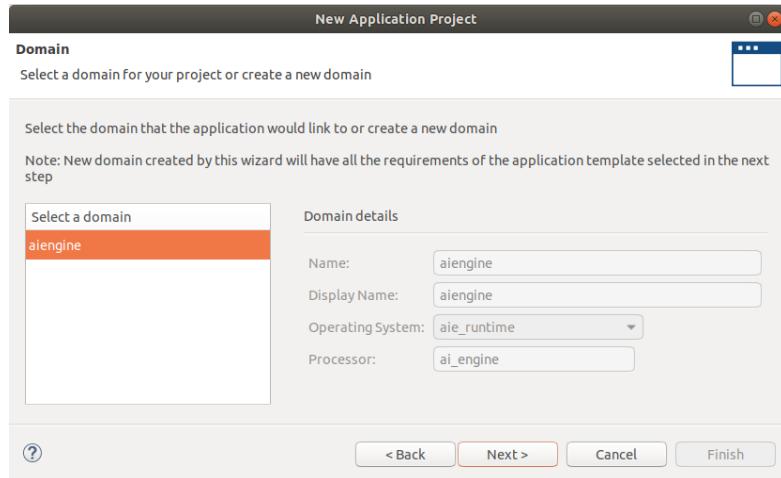
**1-1-6. Select **ai\_engine** as the target processor for the application project.**



**Figure 251: Entering the Application Project Name and Selecting the Target Processor**

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

If there are multiple domains defined in your platform, the Domain page will allow you to choose the specific domain.



**Figure 252: Selecting a Domain**

**Note:** No action is required here.

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

## 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. The wizard allows you to create a new domain or use an existing domain. 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. Click the arrow (▼) next to the Create New icon (1).

A pop-up window appears showing the available wizards.

- 1-1-2. Select **Application Project** to start the Application Project Wizard (2).

Alternatively, you can select **File > Application Project**.

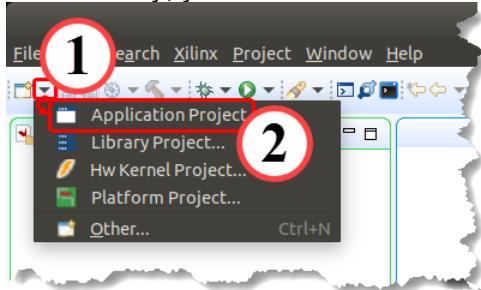


Figure 253: Opening the Application Project Wizard

**1-1-3.** Enter **your application project name** in the Project name field (1).

This defines the name of the application project. While the "app" suffix is not required, it does make identifying what kind of project the entry is much easier.

**Note:** When building an application that will be used with the PetaLinux tools, avoid using any underscores in the name as PetaLinux will not process it correctly.

Typically, the *Use default location* option is left selected so that the location of this project is kept within the workspace (2).

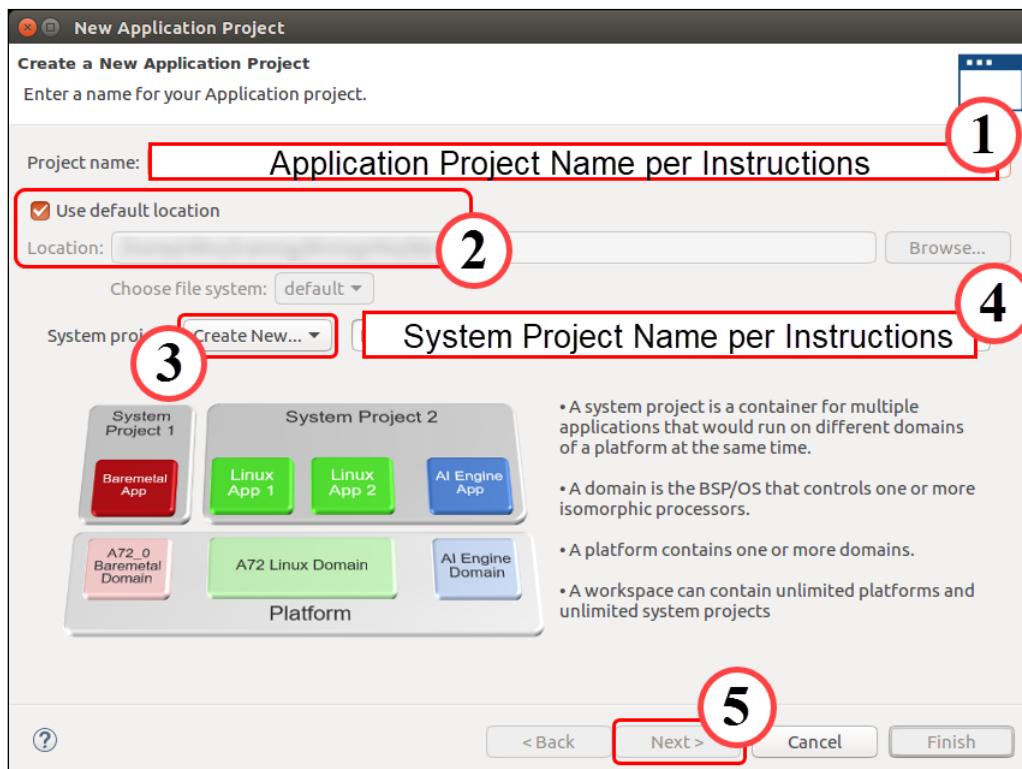
If existing systems exist that this application can be added to, it would be added here. These set of instructions discuss the creation of a new system project so proceed with the following tasks.

**1-1-4.** Ensure that **Create New** is selected from the System project drop-down list (3).

The wizard will automatically populate the System project name field.

**1-1-5.** Change the system project name to **your system project name** (4).

As with the naming of the application project name, the "sys" suffix is not required, but it makes project identification easier.



**Figure 254: Creating a New Application and BSP Project**

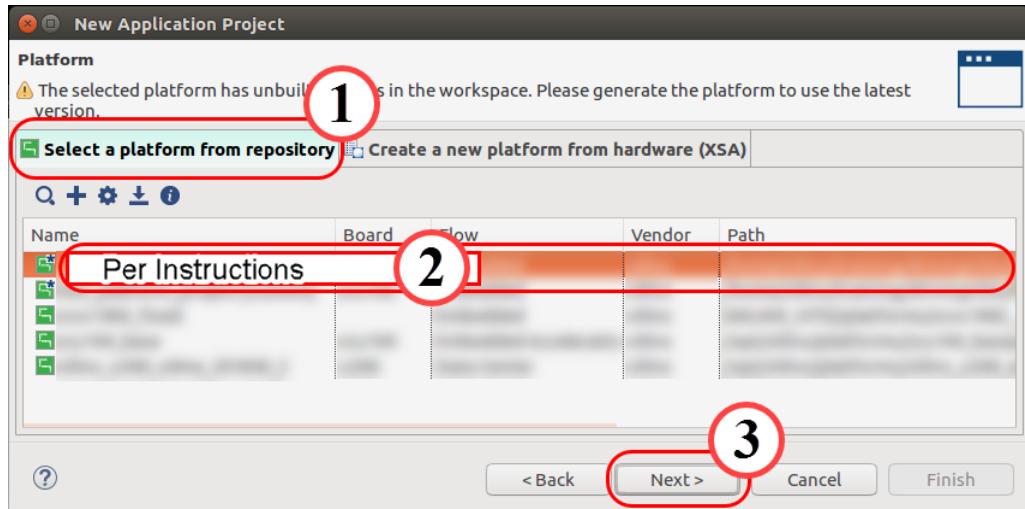
**1-1-6.** Click **Next** to advance and specify the platform project that will associate the platform, system, and application projects (5).

**1-1-7.** Select the **Select a platform from repository** tab (1).

The tool comes with several predefined platforms and this list includes new custom platform projects.

**1-1-8.** Select **your platform project name** from the list (2).

**Note:** If you imported a hardware platform, this will have "[custom]" appended to the name to make it easier to identify.



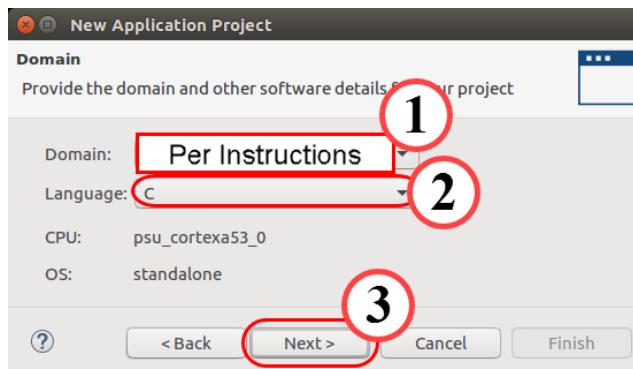
**Figure 255: Selecting the Platform**

**1-1-9.** Click **Next** to proceed to associate a domain project to this application project (3).**1-1-10.** Select **your domain name** as the domain (1).**1-1-11.** Select **C** as the language (2).

This will cause the tools to use the proper toolchain for this language.

Note that the CPU and operating system are set to values configured in other sections of the wizard.

**Note:** The Sysroot path can remain empty as it contains a minimal Linux file system that is not needed because you specified a full Linux image when creating the Linux domain.

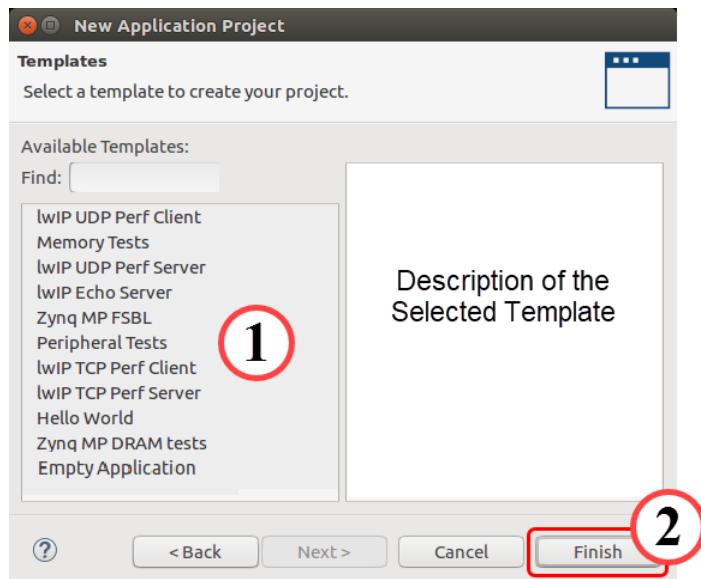


**Figure 256: Selecting the Domain and Language for the Application Project**

**1-1-12.** Click **Next** to select the template (3).

**1-1-13.** Select the **application template** template from the list (1).

Note that the available templates will change based on the OS choice and processor type.



**Figure 257: Selecting the Application Template**

**1-1-14.** Click **Finish** to build the application project (2).

**Note:** This does not compile the application project; rather all the project pieces are assembled.

## Importing Sources to an Application

Add your *files* to the application component.

### 1-1. Import the sources.

- 1-1-1. If not already done, expand **your application project name > Sources** so that the **src** folder is visible in the Vitis Components window (1).
- 1-1-2. Right-click the **src** directory in the project as this is where you want to place the resource files (2).

The tools require certain files to be in specific locations. The tools look in the **src** directory for source files and linker scripts. Other types of files can be imported into other directories.

- 1-1-3. Hover over **Import** and select **Files** to select the necessary files (3).

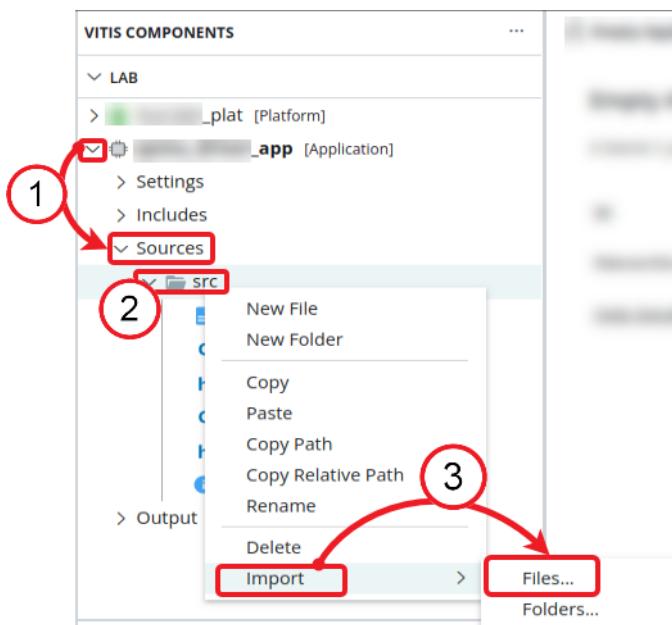


Figure 258: Importing Sources

- 1-1-4. Browse to the following directory:

the directory where your files are located

- 1-1-5. Press **<Ctrl>** and click **your files** to select the file(s).

- 1-1-6. Click **Open** to add all the files to the src directory.

## Importing an Existing Project into the Vitis IDE

One or more projects can be exported as a single zipped file. This is convenient when passing projects or parts of projects to teammates or clients. Once the recipient has received this archive, the zipped file must be processed, and one or more projects can then be imported.

### 1-1. Import an existing project.

- 1-1-1. Select **File > Import** to open the Import Projects Wizard.

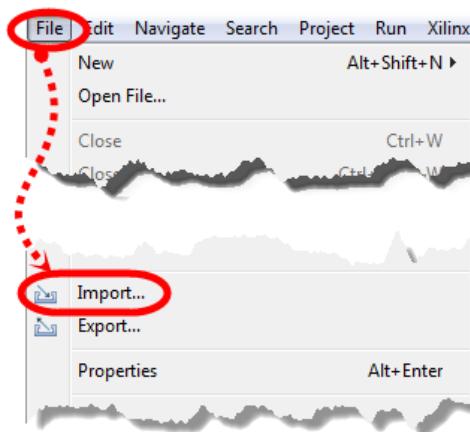


Figure 259: Accessing the Import Projects Wizard

The Import Projects dialog box opens.

- 1-1-2. Select the **Vitis project exported zip file** option and click **Next**.

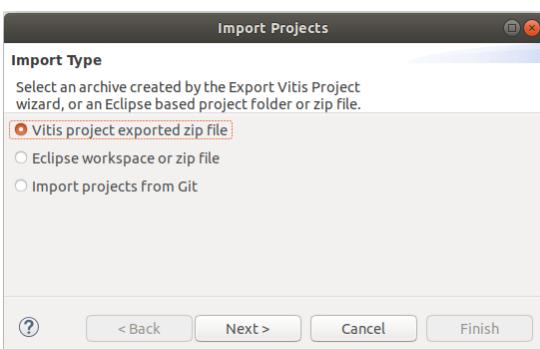
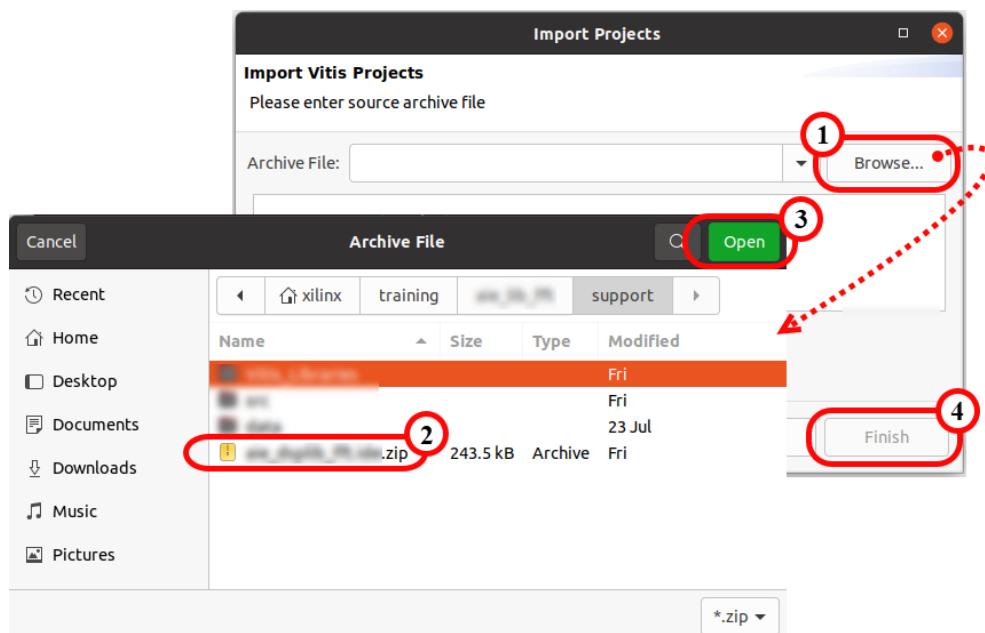


Figure 260: Choosing to Import a Vitis Project Exported Zip File

1-1-3. Click **Browse** in the Archive File field (1).

1-1-4. Browse to the \$TRAINING\_PATH/<the topic cluster name>/support directory and select **your project archive file** (2).

**Hint:** Remember to expand \$TRAINING\_PATH as described in the Introduction section of this lab.



**Figure 261: Browsing and Selecting the File**

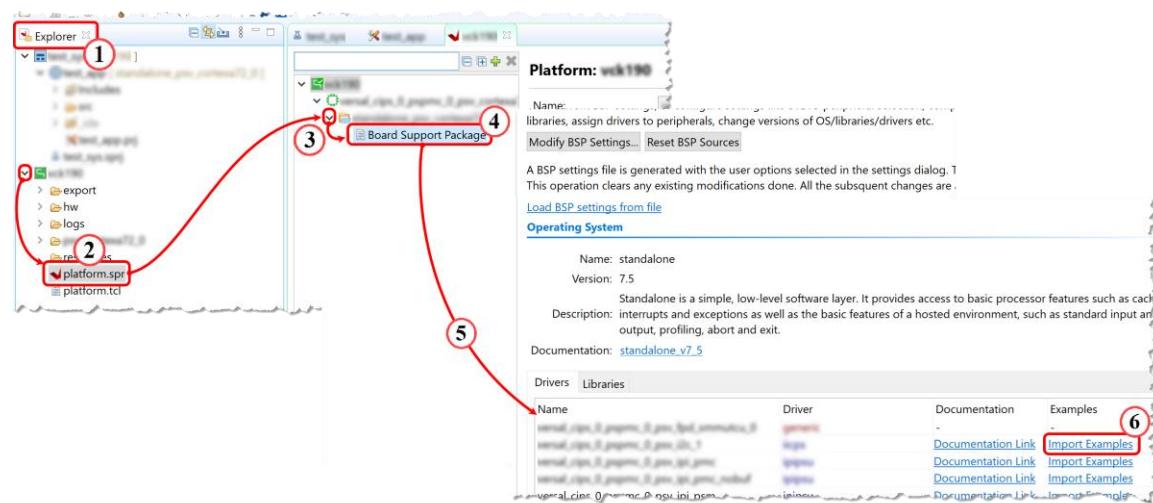
1-1-5. Click **Open** to view the project contents (3).

1-1-6. Click **Finish** to import the project to the workspace (4).

## Viewing/Importing a Peripheral Example Application

### 1-1. Import a peripheral's example application.

- 1-1-1. From the Explorer tab, expand the platform project (1).
- 1-1-2. Select the platform's spr entry so that the domains associated with this platform are visible in the work area (2).
- 1-1-3. Expand the processor running the domain that contains the peripheral that you would like to access (3).
- 1-1-4. Select the specific domain under that processor (4).
- 1-1-5. Locate the peripheral from the table under the operating system's driver list (5).
- 1-1-6. Click **Import Examples** to show various examples associated with this peripheral.



**Figure 262: Locating the Documentation and Examples Available for a Peripheral**

There are two ways to use these examples.

If you just want to see the code example:

- 1-1-7. Expand the desired entry to show the source code example(s) (1).
- 1-1-8. Double-click the source file name to view the code in a pop-up window (2).  
If you want to create an example application project:
  - 1-1-9. Select the example(s) that you want to import (3).
  - 1-1-10. Click **OK** to create the application project (4).

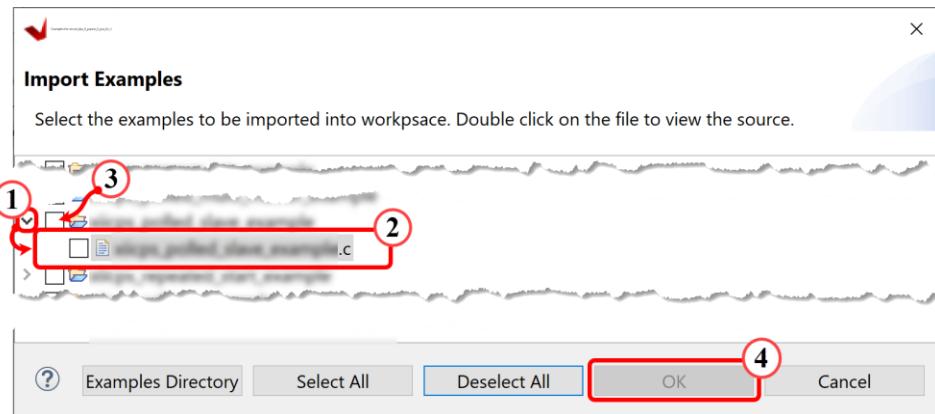


Figure 263: Viewing a Specific Example

## Adding Source Files in the Vitis IDE

---

### 1-1. Import the source (design) files.

1-1-1. Right-click **src** in the Explorer.

1-1-2. Select **Import Sources**.

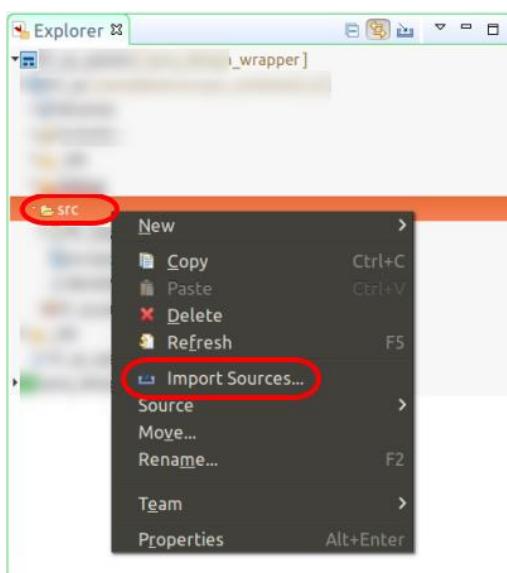
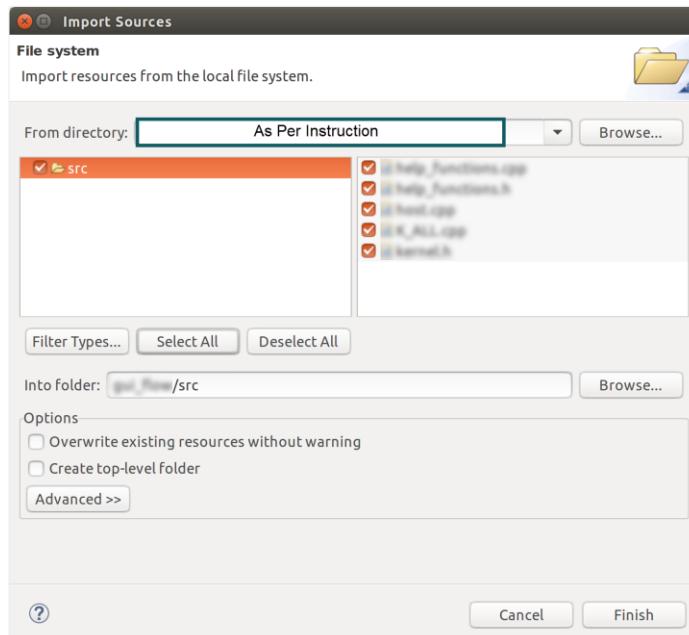


Figure 264: Selecting Import Sources to Add the Source Files

1-1-3. Browse to the support directory \$TRAINING\_PATH/<the topic cluster name>/support in the From directory field and click **OK**.

1-1-4. Select the files **your files**.



**Figure 265: Selecting the Source Files**

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

## Opening a Source File in the Editor

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

#### 1-1-1. Locate **the desired source file** in the Vitis Components window.

**Note:** Source files are typically located in the application's src folder and can be found under your application project name > Sources > src (1).

However, some source files are created as software projects under a platform project. This is typical for boot support projects in Zynq UltraScale+ MPSoC and RFSoC designs.

These sources would then be found under your platform project name > your application project name.

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

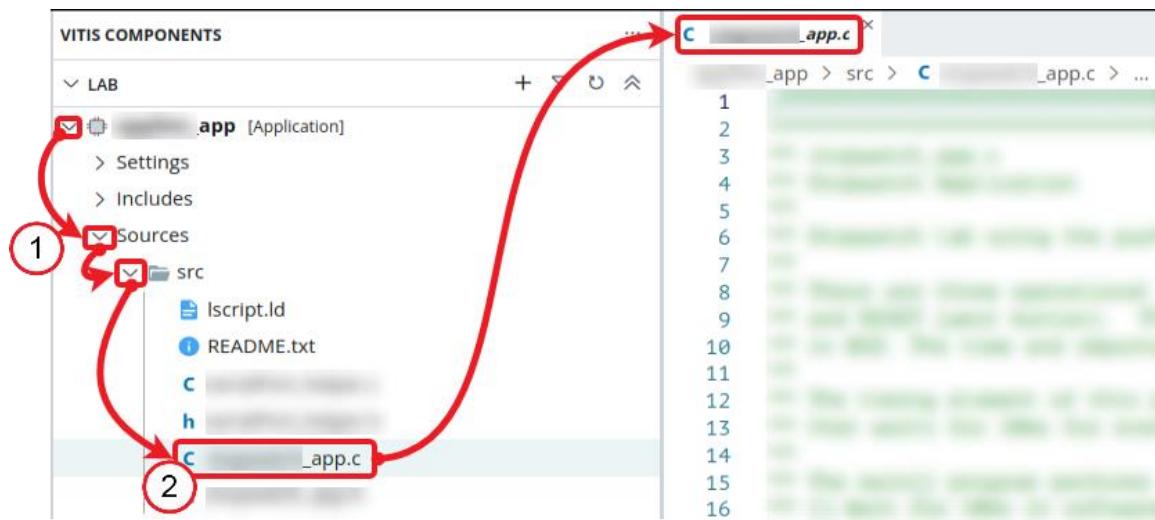


Figure 266: Opening a Source File

## Finding Text in a Source File

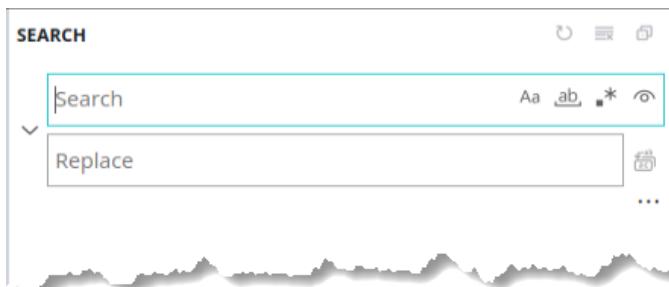
### 1-1. Find *the text that you are looking for*.

#### 1-1-1. Select **View > Search** or press **<Ctrl + F>**.

The Search view or Find/Replace dialog box opens based on how it was invoked.

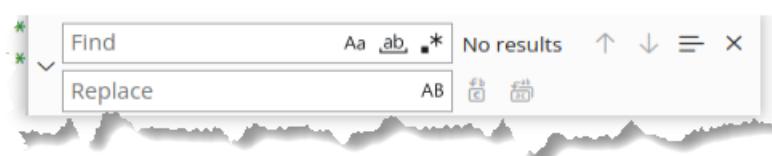
When invoked from a view window or from selecting View from the toolbar, the Search view is opened.

**Note:** The Search view can also be invoked by clicking the magnifying glass icon in a view window.



**Figure 267: Search View**

When the keyboard shortcut is used, the Find/Replace dialog box appears on the top-right side in the editor.



**Figure 268: Search/Replace Dialog Box**

**Note:** To view the Replace field, click the arrow to the left side of the Search or Find field to expand the view or dialog box.

#### 1-1-2. Enter *the text that you are looking for* in the Search or Find field.

#### 1-1-3. Press **<Enter>** to find the next occurrence of *the text that you are looking for*.

#### 1-1-4. Continue pressing **<Enter>** or clicking the down arrow until you locate the specific instance that you are looking for.

If you are looking for text within a specific region of the code, you would first highlight the region to perform the search in and then launch the find/replace function using the keyboard shortcut as described in this set of instructions.

#### 1-1-5. Click the magnifying glass icon to close the Find/Replace dialog box.

## Enabling Line Numbering in the Text Editor

Line numbering is on by default and can be toggled on and off.

### 1-1. Turn on or off line numbering in the text editor.

- 1-1-1. Open the source file in the editor if it is not already open.
- 1-1-2. Right-click in the left margin of the text editor to open the context menu.  
This is where the line numbers appear if line numbering is enabled.
- 1-1-3. Select **Show Line Numbers** to toggle the line numbering.

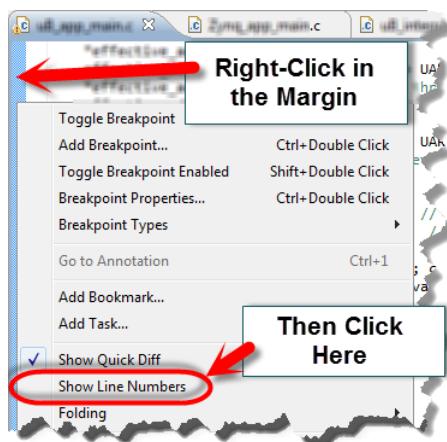


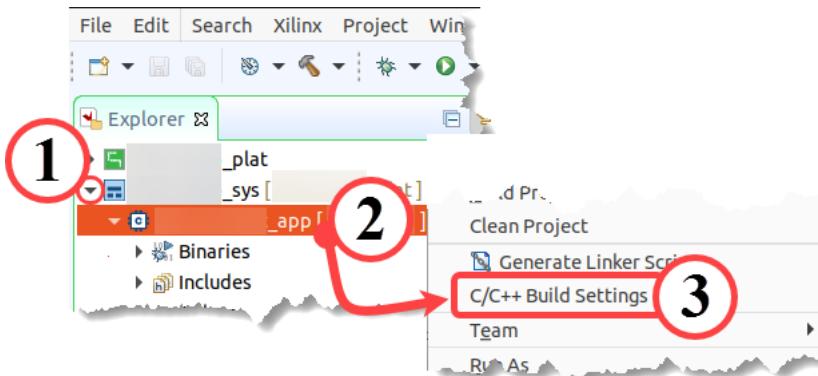
Figure 269: Enabling Line Numbering

## Setting Compiler Options

Compiler options describe the designer's intentions and goals to the compiler. These options directly affect how the object file is constructed. Here you will visit several of the common options.

## 1-1. Access the compiler options for your application project name.

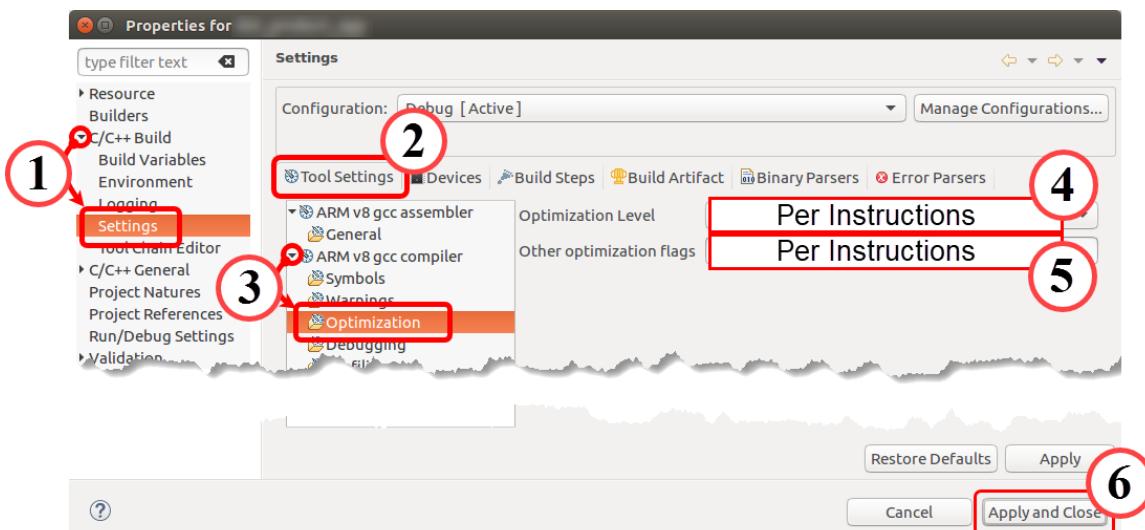
- 1-1-1. Right-click the application to change or verify compiler options for and select **C/C++ Build Settings**.



**Figure 270: 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.



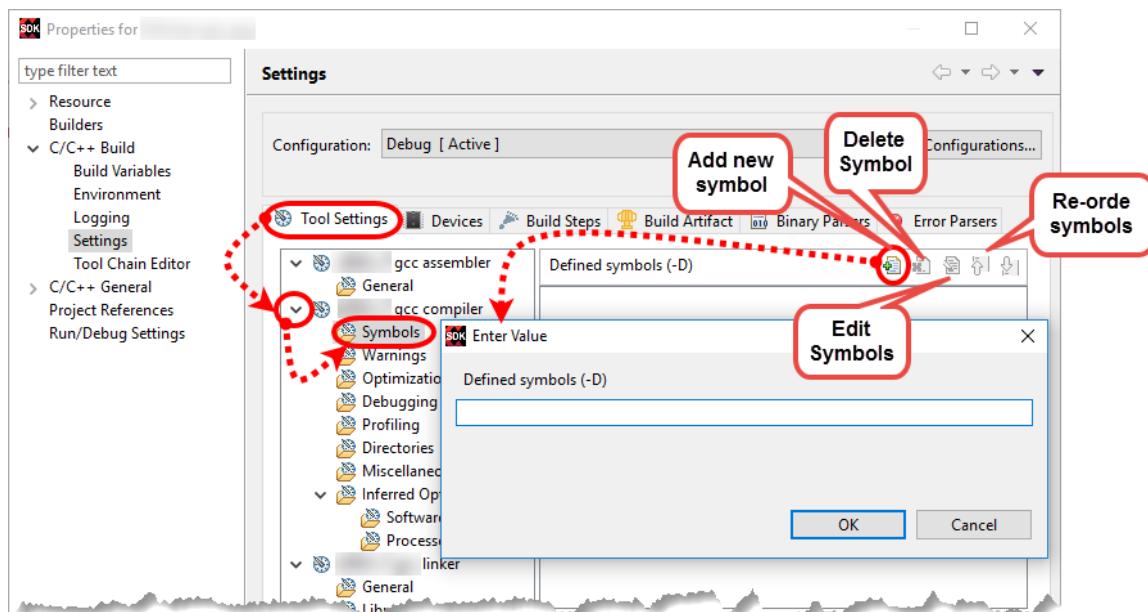
**Figure 271: Accessing the Compiler Settings**

- 1-2. Compiler symbols enable you to define symbols for the pre-processor. By defining symbols here, no changes need to be made to the source code. The Defined Symbols entry enables you to add, remove, edit, and reorder your symbols.**

Remember that `#ifdef` only tests to see if a preprocessor symbol has been defined or not, and `#if` can test against specific values (e.g., assigning a value to a symbol can be done like this: `NEW_SYMBOL=123`).

- 1-2-1. Click **Symbols** under the compiler entry under the Tool Settings tab.
- 1-2-2. Click the green + icon to create a new symbol.
- 1-2-3. Enter the **symbol name (and optionally a symbol value)** into the Enter Value dialog box.
- 1-2-4. Click **OK** to complete the creation of this new symbol.

The above steps can be repeated as necessary to create as many symbols as you need.



**Figure 272: 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.

**Note:** When debugging, you will want to set the optimization to *None (-O0)* so that the assembly code is not reorganized. This reorganization may result in higher performance, but breaks the relationship with the C source code. If this is done, then stepping through the source in the debugger will appear to jump around in unexpected ways.

- 1-3-3. Add any additional flags to the Other optimization flags field.

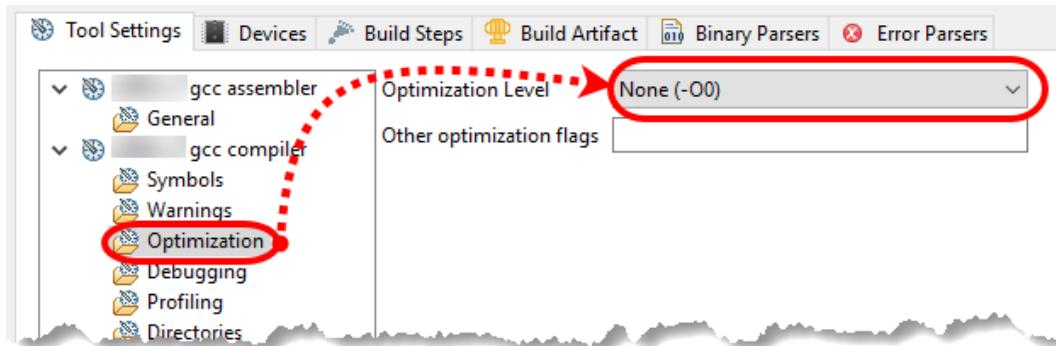
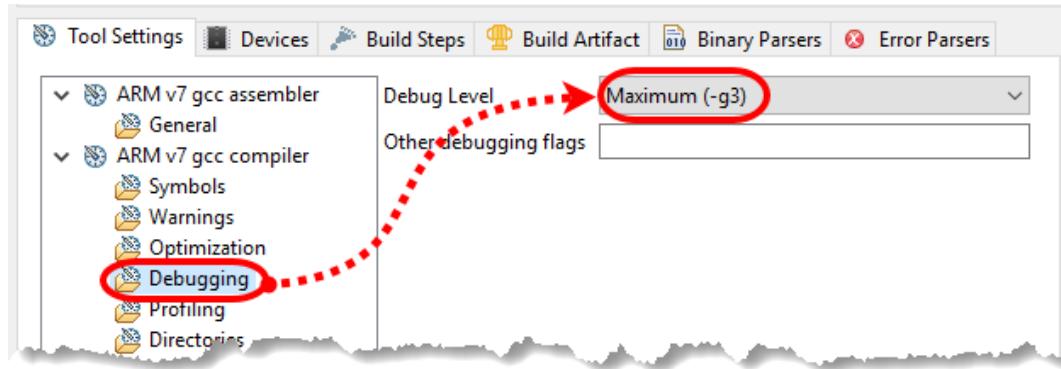


Figure 273: Setting Optimization Levels

**1-4.** The Debugging tab enables you to select the level of debug information to debug the application. The debug levels are -g1 (minimal debug information), -g (default debug information) and -g3 (maximum debug information).

**1-4-1.** Select the Debugging tab.

**1-4-2.** Select **Maximum (-g3)** from the Debug level drop-down list.



**Figure 274:** Setting Debug Level

**1-4-3.** Click on **Apply and Close**.

**1-4-4.** Click **Yes** if prompted to rebuild the application.

The application is rebuilt. A successful build is indicated when the program size is returned.

## Setting Compiler Optimization Options

Compiler options describe the designer's intentions to the compiler. These options directly affect how the object file is constructed.

### 1-1. Set the baseline application's optimization to your desired level of optimization.

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 are collections of specific optimization flags. Refer to the *GNU Compiler Manual* for a listing of which flags are enabled for each optimization level.

If the predefined optimization settings do not suit your needs, you can add your optimization flags to the *Other optimization flags* text field. Remember that optimizations can change the way your code operates.

**1-1-1.** If necessary, expand **your system project name** system project to see the application projects (1).

**1-1-2.** Right-click **your application project name** (2).

**1-1-3.** Select **C/C++ Build Settings** (3).

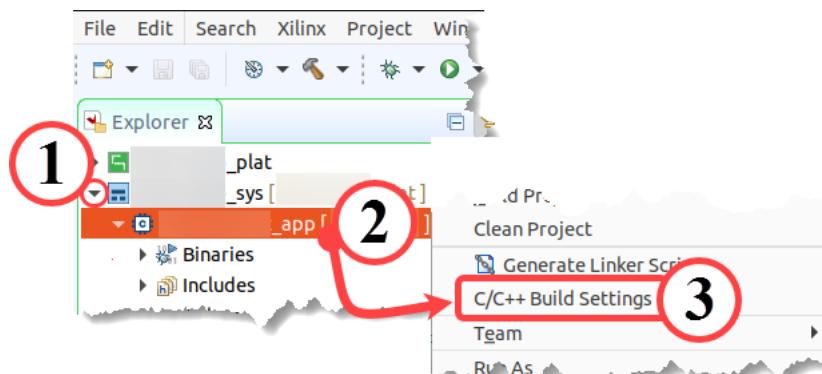
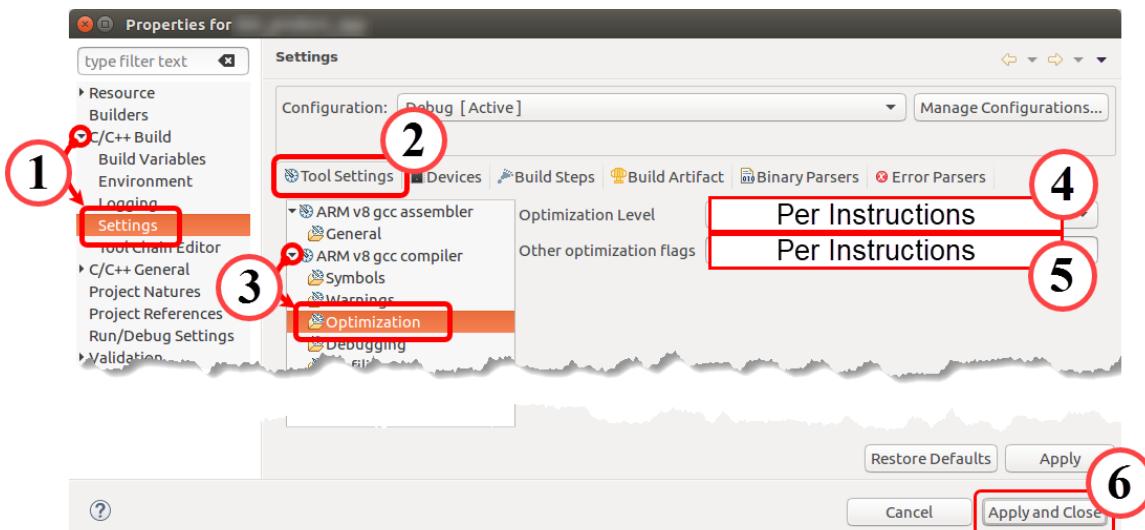


Figure 275: Accessing the Compiler Settings for a Specific Application

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

- 1-1-4. Click **Settings** in the left-hand navigation bar to access the specific settings for the compiler (1).
  - 1-1-5. Ensure that the **Tool Settings** tab is selected (2).
  - 1-1-6. Select the **Optimization** entry in the Tool Settings tab to access the optimization settings (3).
- The Optimization option is found in the compiler group.
- 1-1-7. If the optimization is not already set to your desired level of optimization, then select **your desired level of optimization** from the Optimization Level drop-down list (4).
- If you need to add any additional flags, place them in the *Other optimization flags* field (5).



**Figure 276: Accessing the Compiler Settings**

- 1-1-8. Click **Apply and Close** to save your settings and exit (6).

Remember that you will have to manually rebuild the project for these settings to be applied to the code.

## Adding Symbols to Application Settings

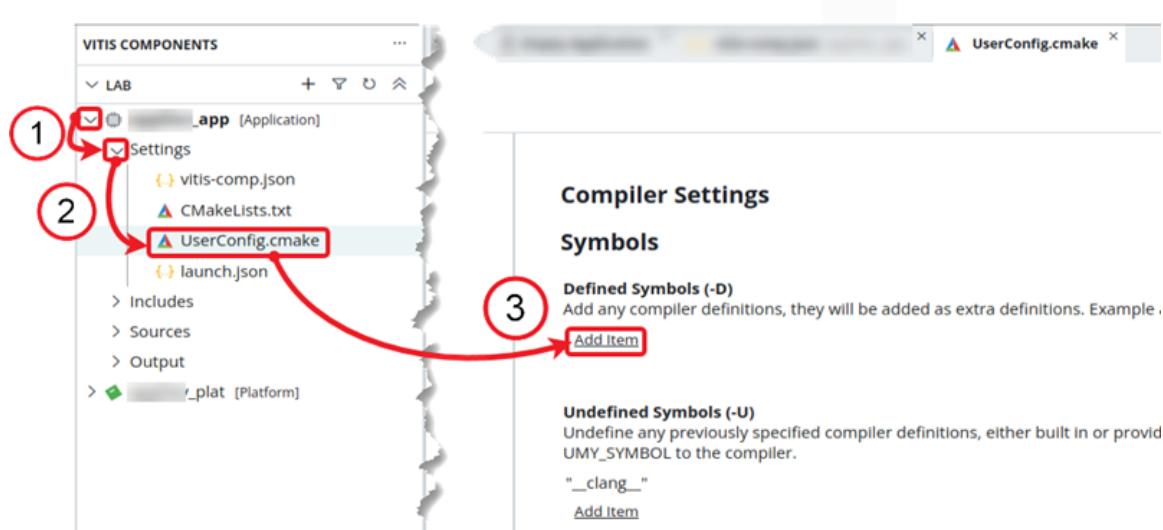
Adding a symbol to an application is equivalent to adding a #DEFINE SYMBOL\_NAME statement that is visible to the entire project. It is common practice to use symbols to conditionally compile code by guarding code with conditional pre-processor statements (e.g., #ifdef SYMBOL\_NAME ... #endif).

Remember that #ifdef only tests to see if a preprocessor symbol has been defined or not, and #if can test against specific values (assigning a value to a symbol can be done like this: NEW\_SYMBOL=123).

### 1-1. Add symbol(s) to the compiler settings.

- 1-1-1. Expand **your application project name > Settings** in the Vitis Components window (1).
- 1-1-2. Double-click **UserConfig.cmake** to open the compiler settings for this project (2).
- 1-1-3. Click **Add Item** under Defined Symbols (-D) to add the first symbol (3).

Symbols are entered one at a time. Follow tasks 3 through 5 for each element in this list: **application project symbol names as required**.



**Figure 277: Accessing the UserConfig Settings for the Application**

1-1-4. Enter the symbol name in the field that opened after the previous instruction (1).

1-1-5. Click **OK** to add each new symbol (2).

**Note:** Symbols can only be added one at a time.

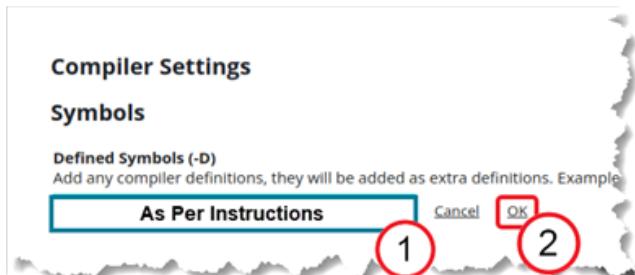


Figure 278: Defining a New Compiler Symbol

1-1-6. Once all the symbols are added, close the **UserConfig.cmake** tab.

## Building the Projects

---

1-1. Confirm the build setting and build the projects.

1-1-1. From the Flow window, click the arrow for the Component drop-down list (1).

1-1-2. Select your application project name (2).

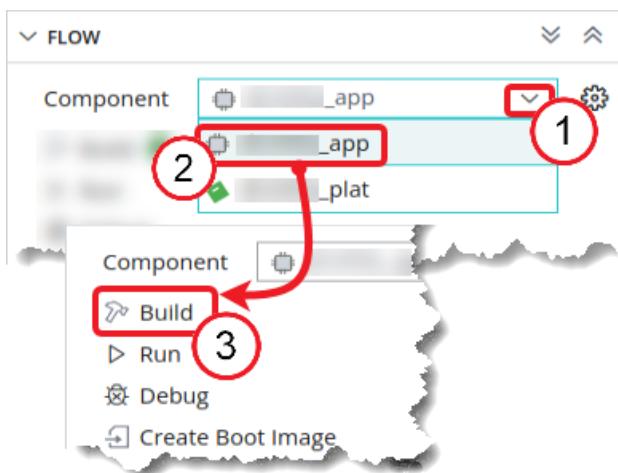


Figure 279: Building the Application Component

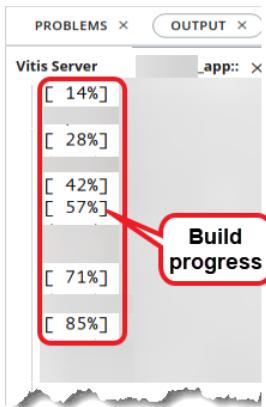
1-1-3. Click **Build** (3).

This will build all projects with the settings specified in the build configuration (if any).

**Note:** Click **OK** when prompted with the Platform Build message.

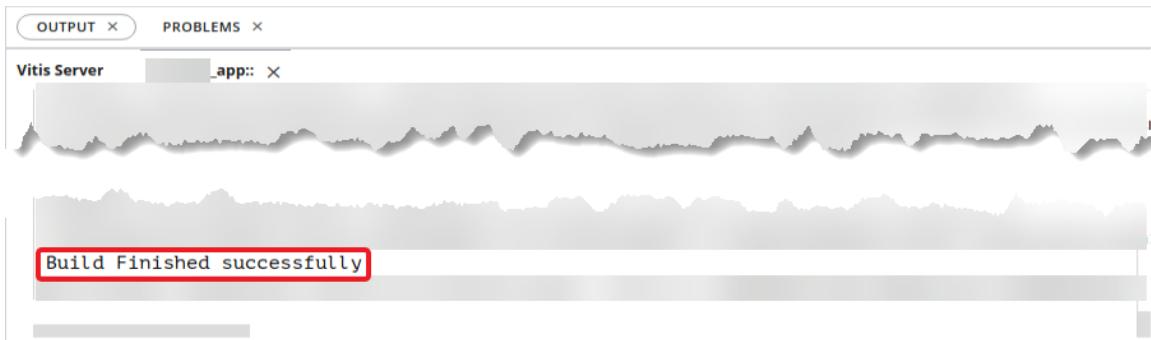
Any issues are shown in the Problems or Output tabs.

**Note:** Monitor the progress of the build via the percentage values displayed in the Output tab.



**Figure 280: Build Progress Indicator**

When the compilation completes successfully, a size summary of the build is displayed.



**Figure 281: Successful Build Console Message**

## Running with a Default Configuration

Configurations are a powerful ally in setting up a run for specific sets of criteria. The default configuration settings are adequate for most basic programs and do not require clicking through multiple dialog boxes just to run the program.

### 1-1. Run *your application project name* with the default configuration settings.

- 1-1-1. Select **your application project name** from the Component drop-down list (1).
- 1-1-2. Click **Run** to begin running the application on the hardware (2).

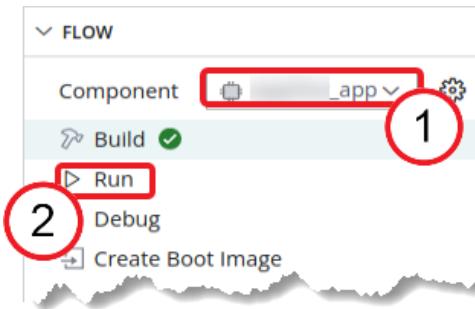


Figure 282: Launching the Application on Hardware Using the Default Configuration Settings

## Setting Up a Run Configuration for a Linux Application

### 1-1. Create a Run configuration.

Run configurations associate an ELF object file to a target for execution. Here, the target is a board accessed over a TCP/IP connection.

- 1-1-1. Click the C/C++ perspective (top right) to return to the original perspective.
- 1-1-2. Right-click **your application project name** from the Explorer pane (1).
- 1-1-3. Select **Run As** (2).
- 1-1-4. Select **Run Configurations** (3).

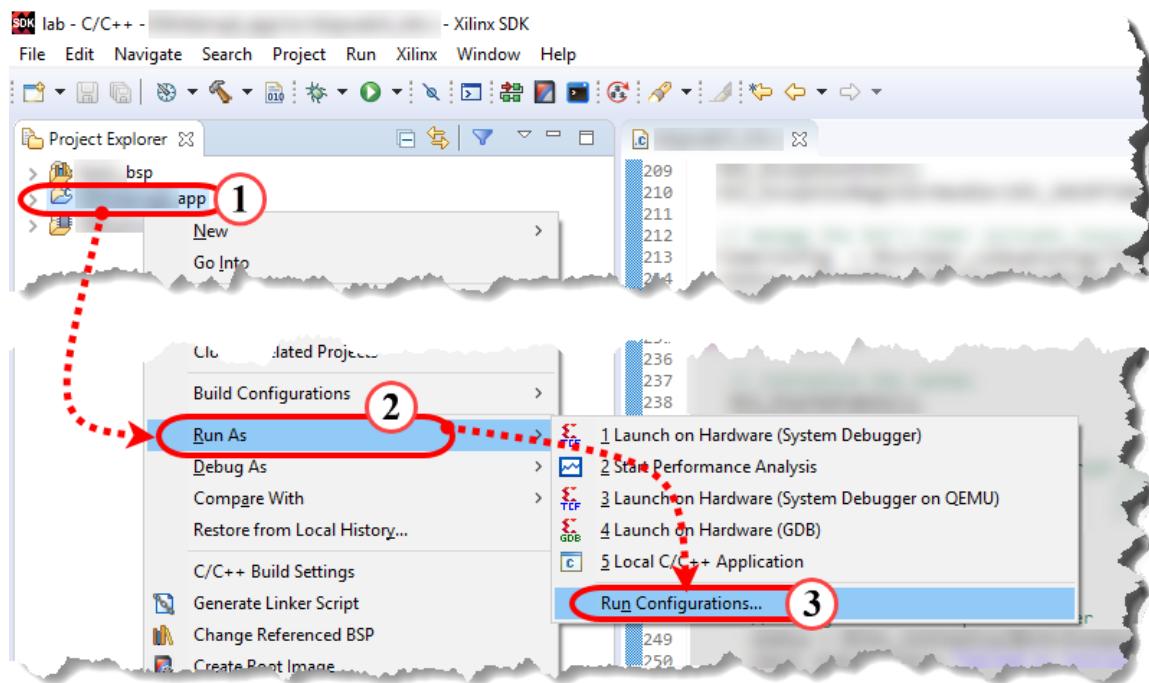


Figure 283: Selecting Run Configurations

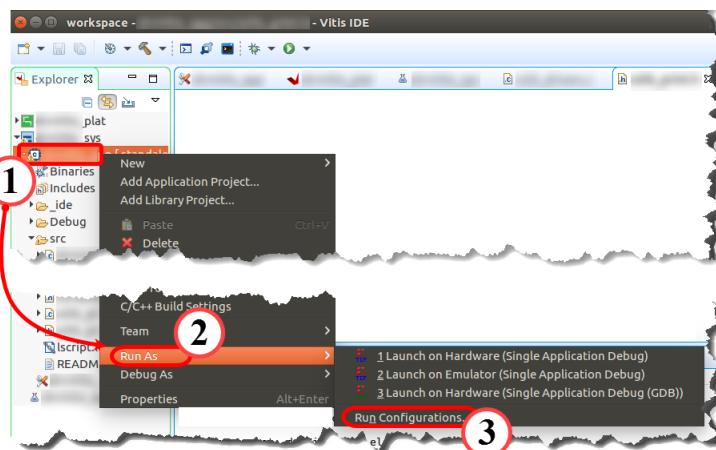
A Run configuration defines how you want the system to work when running an application. While there are a significant number of switches and options, the most common are shown below.

**1-2. Set up a Linux Run configuration for *your application project name*.**

**1-2-1.** Right-click **your application project name** from the Explorer pane (1).

**1-2-2.** Select **Run As** from the context menu (2).

**1-2-3.** Select **Run Configurations** (3).



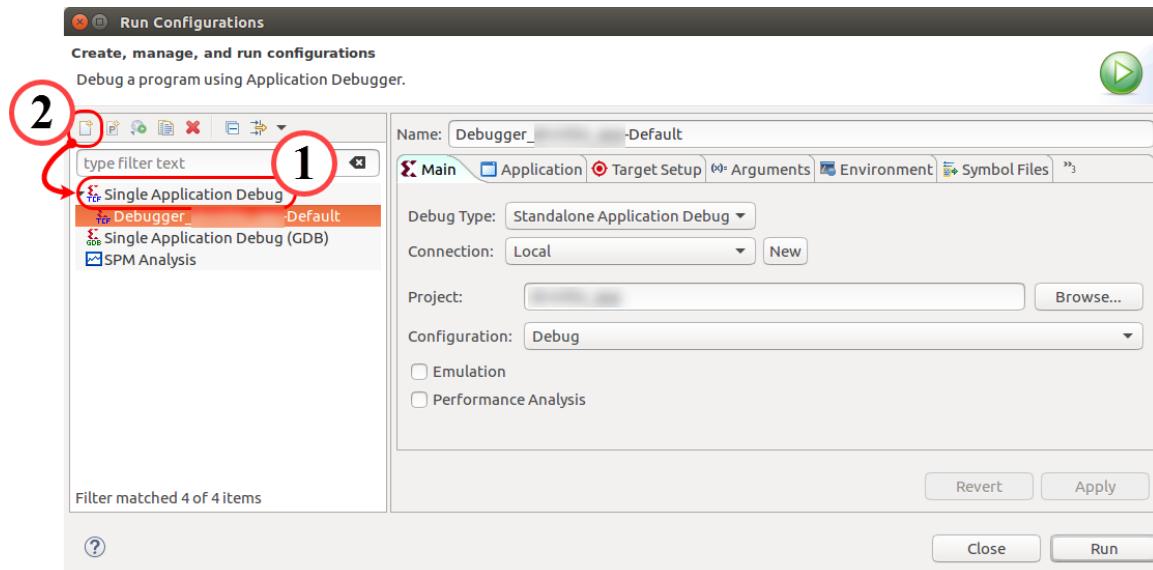
**Figure 284: Creating a Run Configuration for an Application**

The Run Configurations dialog box opens.

**1-2-4.** Double-click **Single Application Debug** to create a launch configuration (1).

Alternatively, you can select **Single Application Debug** and click the **New Launch Configuration** icon (2).

If this is the first debug configuration being created for the project, descriptions of the various options appear in the left pane. If one or more configurations exist, then the last open configuration will be displayed. Regardless, a new configuration can always be added. Existing configurations are shown in the left pane and can be selected for editing.



**Figure 285: Creating a New Run Configuration**

A new Run configuration is created named *System Debugger using Debug\_your application project name.elf on Local* and the configuration information appears in the right pane.

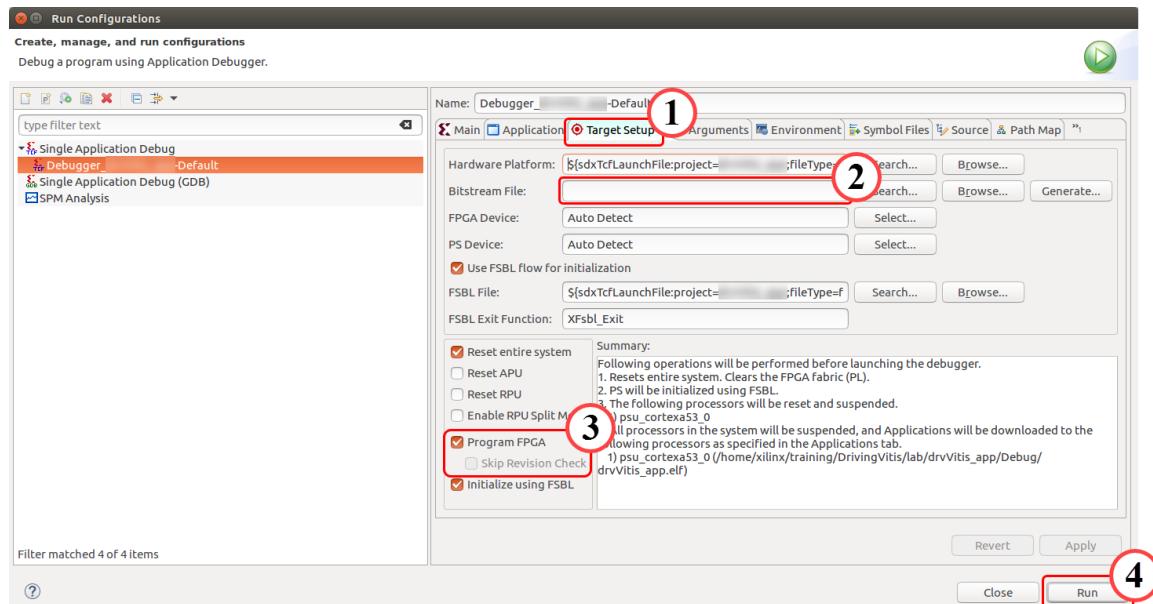
By default, the Main tab is selected and shows general information about the configuration.

- 1-2-5.** Select the **Target Setup** tab so that you can begin specifying how you want the device to boot (1).

- 1-2-6.** Ensure that a bitstream file is present (2).

Typically, bitstreams are missing when there is not a design component in the PL, or the bitstream was not exported as part of the XSA file. Generally, if the XSA includes the bitstream, then this field will be automatically populated.

- 1-2-7.** Ensure that the **Program FPGA** option is selected (3).



**Figure 286: Run Configurations Dialog Box**

The new configuration will appear with other existing configurations and have the name of your application.

You will also note that many fields are automatically filled in for you using the name of your application as the basis. That is, if your application is named *XYZ*, then the Name field will be populated with *XYZ Debug*. Most users will leave the other settings at their default.

- 1-2-8.** Click **Run** to close the dialog box and launch the software application on the hardware evaluation board (4).

### 1-3. Configure the debug type and host connection.

- 1-3-1. Double-click **Xilinx C/C++ application (System Debugger)** to create a launch configuration.
- 1-3-2. Select **Linux Application Debug** from the Debug Type drop-down list (1).
- 1-3-3. Click **New** next to the Connection drop-down list (2).
- 1-3-4. Enter **your board** in the Target Name field (3).
- 1-3-5. Enter **the IP address of the board** in the Host field (4).
- 1-3-6. Enter **1534** in the Port field (4).

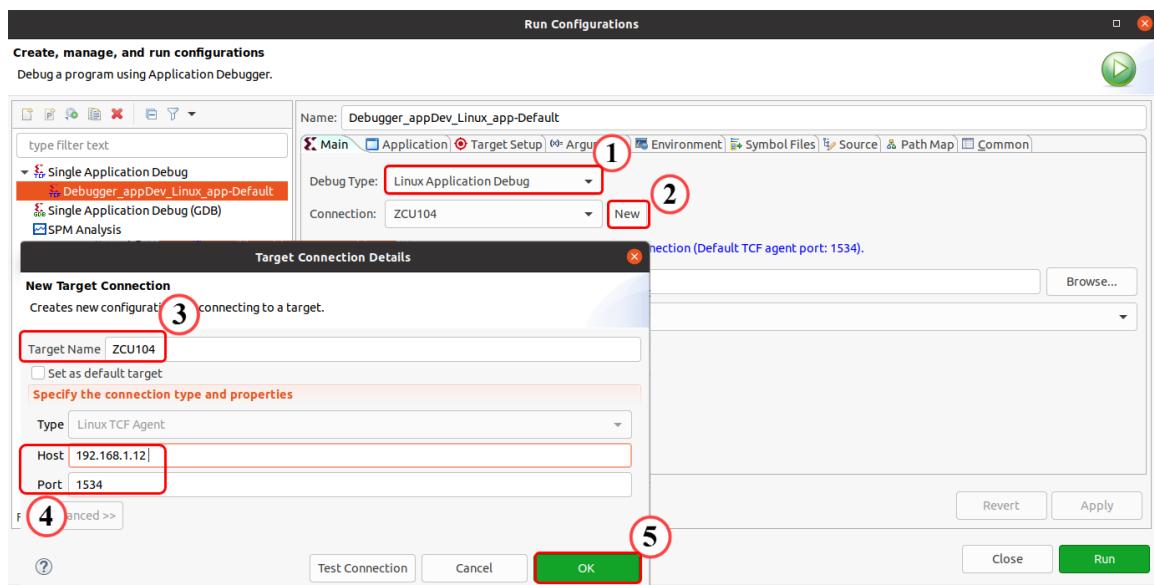
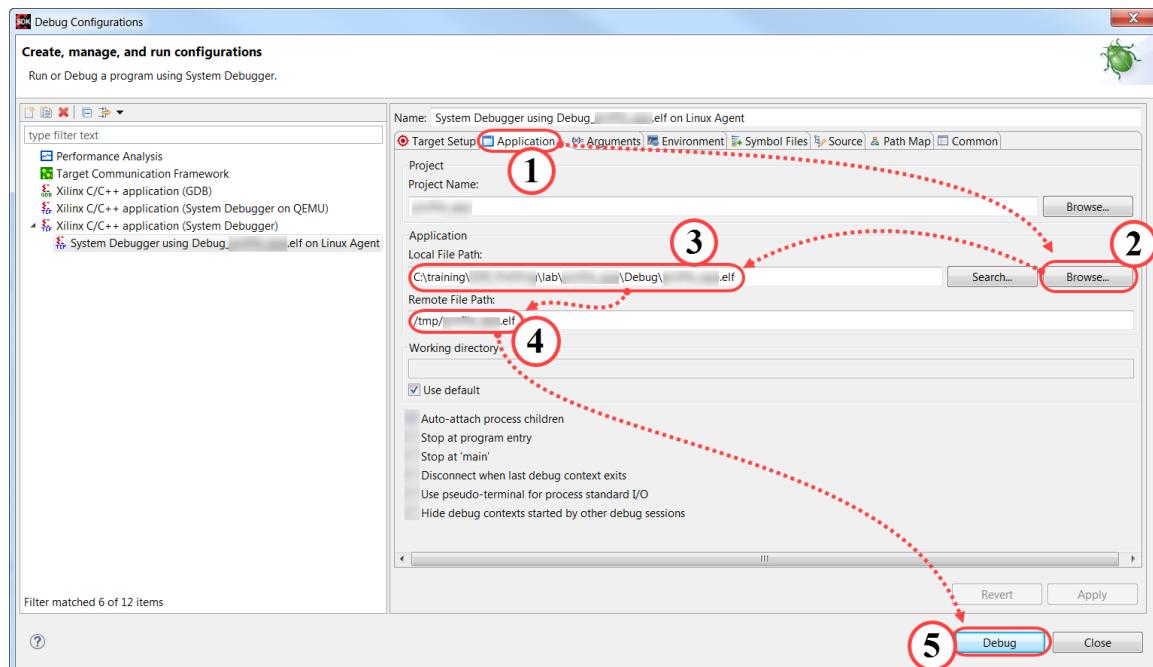


Figure 287: Selecting the Debug Type and Connection

- 1-3-7. Click **OK** (5).

## 1-4. Select the application and remote file path to copy the ELF file to the board.

- 1-4-1. Select the **Application** tab (1).
- 1-4-2. Click **Browse** next to the Local File Path field (2).
- 1-4-3. Select the local file path to be \$<the topic cluster name>/lab/your application project name/Debug/your application project name.elf if not automatically populated (3).
- 1-4-4. Enter /tmp/your application project name.elf in the Remote File Path field (4).



**Figure 288: Selecting the Local File Path and Remote File Path**

- 1-4-5. Click **Run** (5).

## Setting Up a Run Configuration for a Hardware Target

A Run configuration defines how you want the system to work when running an application. While there are a significant number of switches and options, the most common are shown below.

### 1-1. Set up a Run configuration for *your application project name*.

1-1-1. Right-click **your application project name** from the Explorer pane (1).

1-1-2. Select **Run As** from the context menu (2).

1-1-3. Select **Run Configurations** (3).

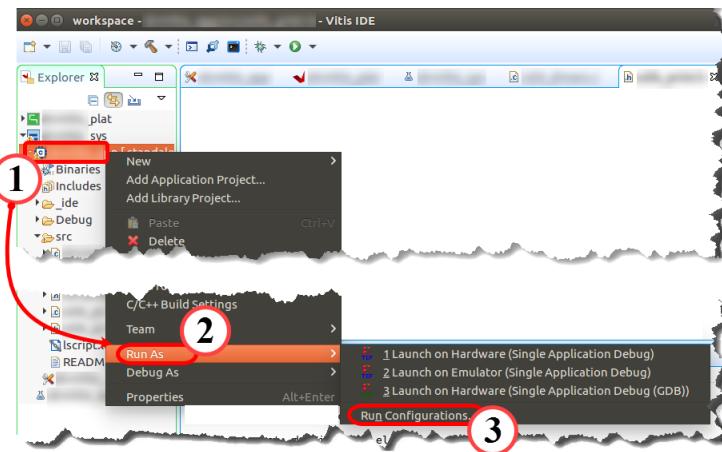


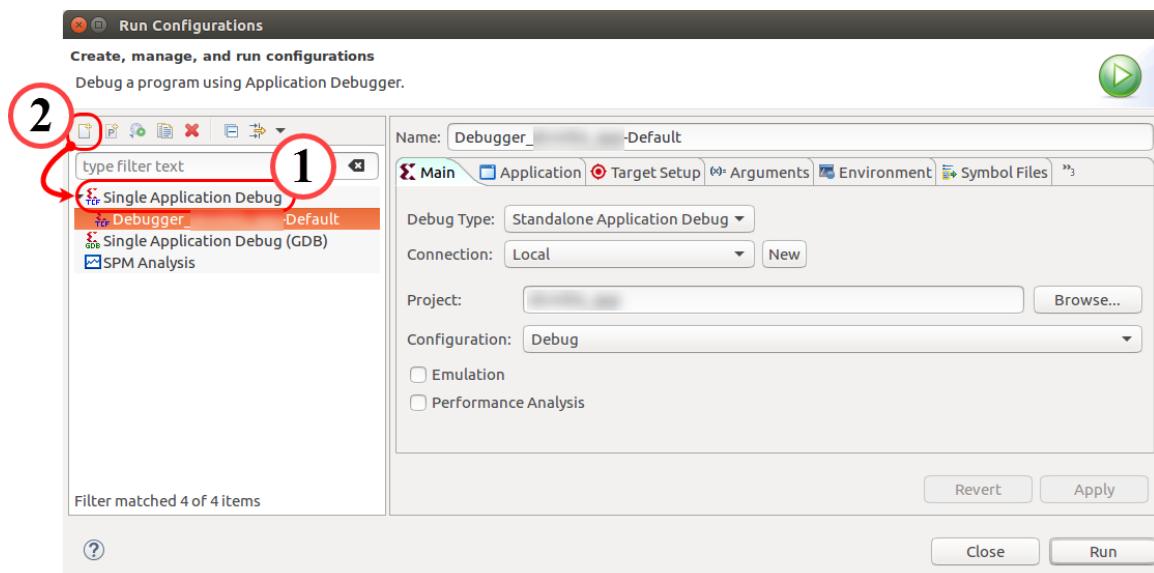
Figure 289: Creating a Run Configuration for an Application

The Run Configurations dialog box opens.

**1-1-4.** Double-click **Single Application Debug** to create a launch configuration (1).

Alternatively, you can select **Single Application Debug** and click the **New Launch Configuration** icon (2).

If this is the first debug configuration being created for the project, descriptions of the various options appear in the left pane. If one or more configurations exist, then the last open configuration will be displayed. Regardless, a new configuration can always be added. Existing configurations are shown in the left pane and can be selected for editing.



**Figure 290: Creating a New Run Configuration**

A new Run configuration is created named *Debugger\_your application project name-Default*, and the configuration information appears in the right pane.

By default, the Main tab is selected and shows general information about the configuration.

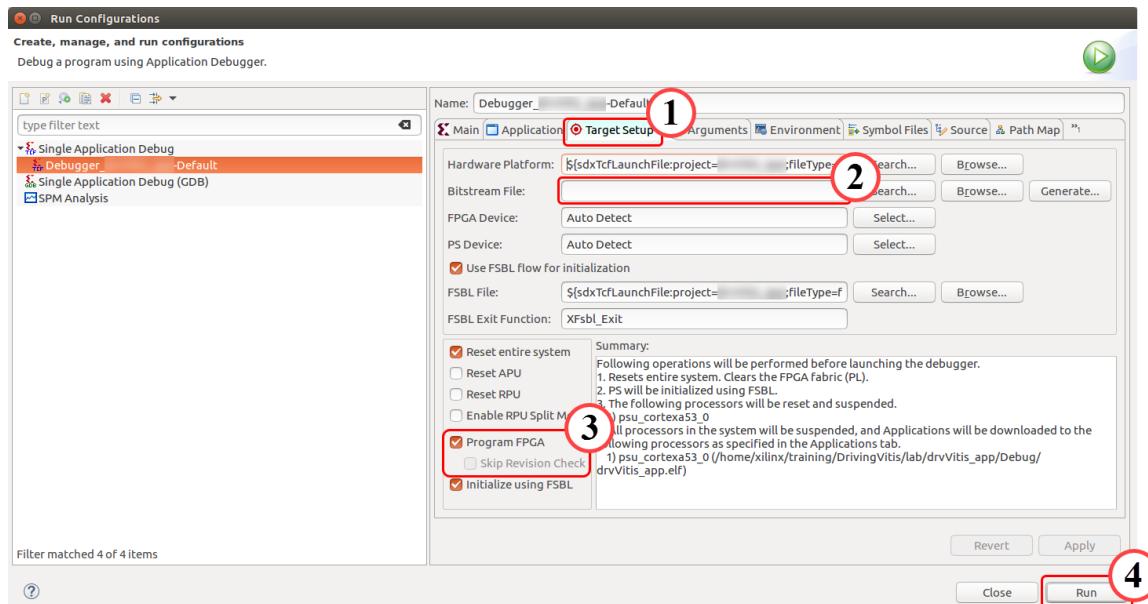
- 1-1-5. Select the **Target Setup** tab so that you can begin specifying how you want the device to boot (1).

- 1-1-6. Ensure that a bitstream or PDI file is present (2).

**Note:** Zynq-7000 SoC and Zynq UltraScale+ MPSoC devices use a bitstream. Versal ACAP devices have their bitstreams wrapped in a PDI file.

Typically, bitstreams are missing when there is not a design component in the PL, or the bitstream was not exported as part of the XSA file. Generally, if the XSA includes the bitstream, then this field will be automatically populated.

- 1-1-7. Ensure that the **Program FPGA** or **Program Device** option is selected (3).



**Figure 291: Run Configurations Dialog Box**

**Note:** This figure illustrates a typical Zynq UltraScale+ MPSoC configuration. Versal ACAP devices are similar.

The new configuration will appear with other existing configurations and have the name of your application.

You may notice that many fields are automatically filled in for you using the name of your application as the basis. That is, if your application is named *XYZ*, then the Name field will be populated with *XYZ Debug*. Most users will leave the other settings at their default.

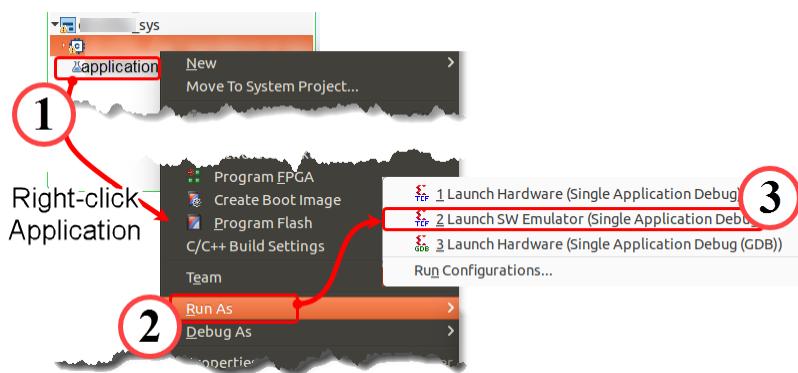
- 1-1-8. Click **Run** to close the dialog box and launch the software application on the hardware evaluation board (4).

## Setting Up a Run Configuration for an Emulator Target

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, commonly used options for emulation are included in the direct launching options shown here. If other settings are required, they can be set by creating a Run Configuration and selecting the desired options.

### 1-1. Execute an emulator run for *your application project name*.

- 1-1-1. Right-click **your application project name** from the Explorer pane (1).
- 1-1-2. Select **Run As** from the context menu (2).
- 1-1-3. Select **Launch SW Emulator (Single Application Debug)** to launch the emulator with the current configuration settings (3).



**Figure 292: Launching the Application on the Emulator**

The Run Configurations dialog box opens, informing you that the launch is proceeding.

- 1-1-4. If you are asked to launch the emulator before running the application, select **Start Emulator and Run**.

The emulator will begin, and you can monitor the progression of the tool in the lower-right corner of the tool. Once the code begins running, any output will appear in the emulator console.

**Note:** The emulation begins with the execution of the First Stage Boot Loader. You will notice the various stages of booting occurring in the Emulation Console.

## Creating and Running the Default Debug Configuration

There are many options as to how to set up a Debug configuration. Often, the default options are enough when you just want to quickly run your application in debug mode.

### 1-1. Launch the XSDB console and determine the TCF port number.

- 1-1-1. Select **Vitis > XSDB Console** to launch the XSDB console.
- 1-1-2. After the console launches, enter the following command to set up a TCF connection:

**Note:** You may have to press **<Enter>** in order to type in the XSDB console.

```
xsdb% gdbremote connect localhost:9000
```

**Note:** If `connection refused` messages appear after running the above command, replace 9000 with 9001 and try again.

```
***** Build date : Oct 13 2023-20:26:23
** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
** Copyright 2022-2023 Advanced Micro Devices, Inc. All Rights Reserved.

XSDB Server URL: TCP:localhost:41541
xsdb% XSDB Server Channel: tcfchan#0
XSDB Started

xsdb%
xsdb% gdbremote connect localhost:9001
xsdb% Info: Cortex-A72 #0 (target 3) Stopped at 0x1f60 (Suspended)
xsdb% Info: Cortex-A72 #1 (target 4) Stopped at 0x0 (Suspended)
xsdb% Info: Cortex-R5 #0 (target 6) Stopped at 0x0 (Suspended)
xsdb% Info: Cortex-R5 #1 (target 7) Stopped at 0x0 (Suspended)
xsdb%
```

**Figure 293: Setting Up a TCF Connection**

- 1-1-3. Enter the following command to display the currently running TCF connections:

```
xsdb% conn -list
```

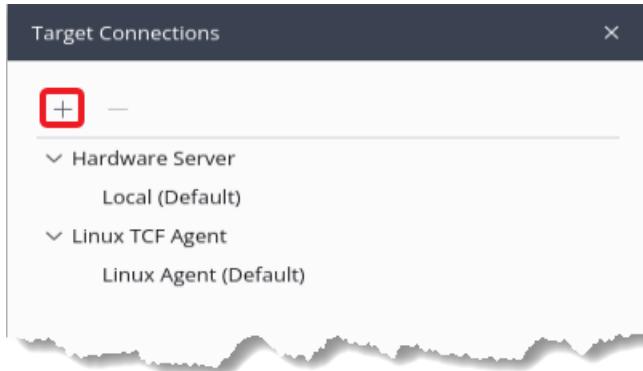
```
xsdb% conn -list
*tcfchan#1 tcp:127.0.0.1 [REDACTED]
xsdb%
```

**Figure 294: List of Running TCF Connections**

Note down the port number—it will be needed for the instructions that follow.

**1-2. Create a new target connection for the Hardware Server.**

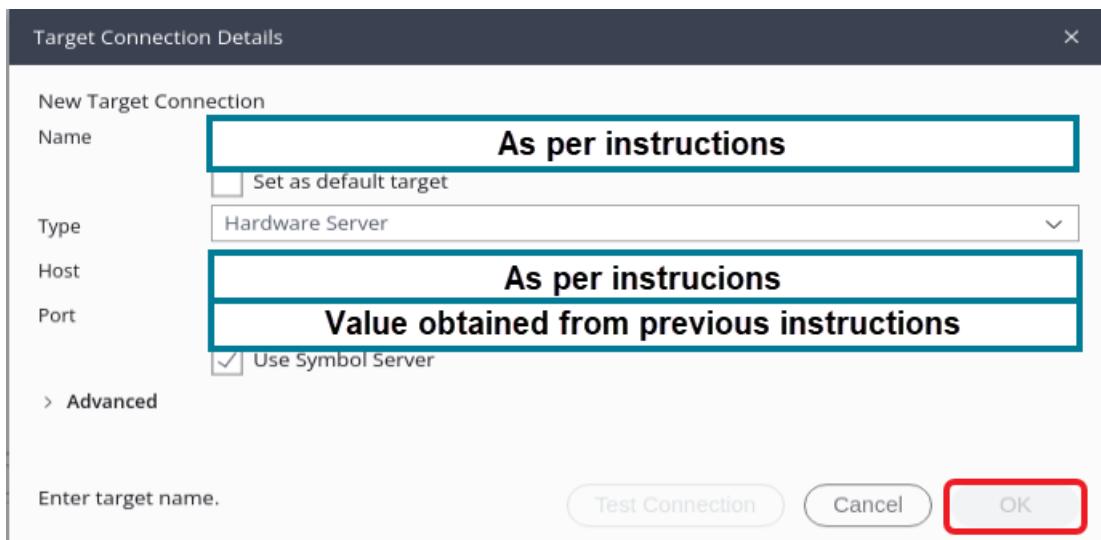
- 1-2-1. Select **Vitis > Target Connections** to create a new target connection.
- 1-2-2. Click the **+** icon for a new target connection.



**Figure 295: Creating a New Target Connection**

The Target Connection Details dialog box opens.

- 1-2-3. Enter **qemu\_debug** in the Name field (1).
- 1-2-4. Ensure that **Hardware Server** is selected as the type (2).
- 1-2-5. Enter **127.0.0.1** as the host IP (3).
- 1-2-6. For the port number, enter the value that you noted down in the previous instructions (4).



**Figure 296: Configuring the Target Connection**

- 1-2-7. Click **OK** to exit the Target Connection Details dialog box (4).
- 1-2-8. Close the Target Connections window.

### 1-3. Create a new launch configuration.

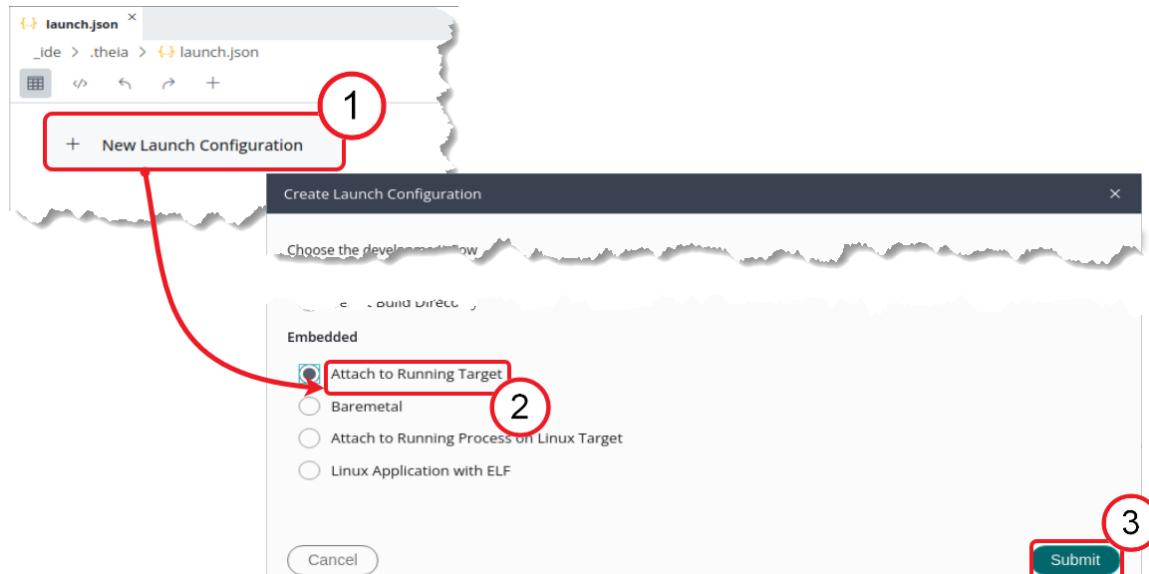
- 1-3-1. Right-click in the empty space of the Explorer pane.

**Note:** The Explorer pane can be opened by selecting **View > Explorer**.

- 1-3-2. Select **Edit Launch Configurations**.

- 1-3-3. Click **New Launch Configuration** to launch the Create Launch Configuration Wizard (1).

- 1-3-4. Select **Attach to Running Target** as the target setup mode (2).



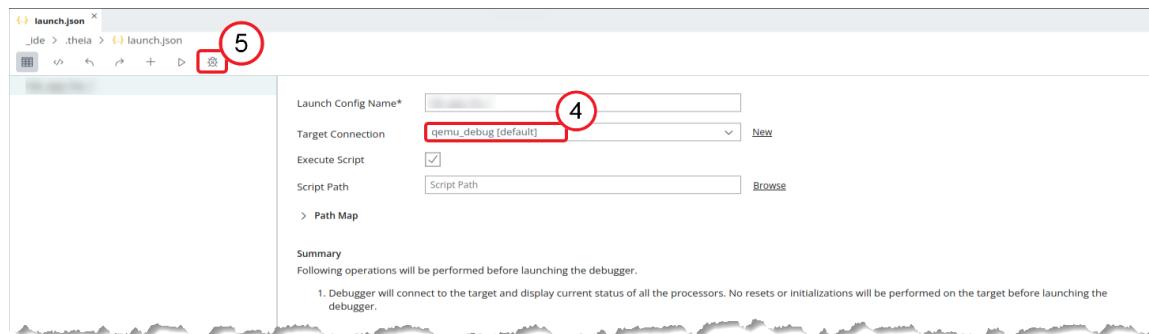
**Figure 297: Creating a New Launch Configuration**

- 1-3-5. Click **Submit** to close the Create Launch Configuration dialog box (3).

A launch configuration with default settings is generated.

- 1-3-6. Ensure that the Target Connection setting is the same as the one created in the previous instructions (4).

- 1-3-7. Click the **Debug** icon to begin a debugging session (5).



**Figure 298: Newly Generated Launch Configuration**

#### 1-4. Add and download a symbol file.

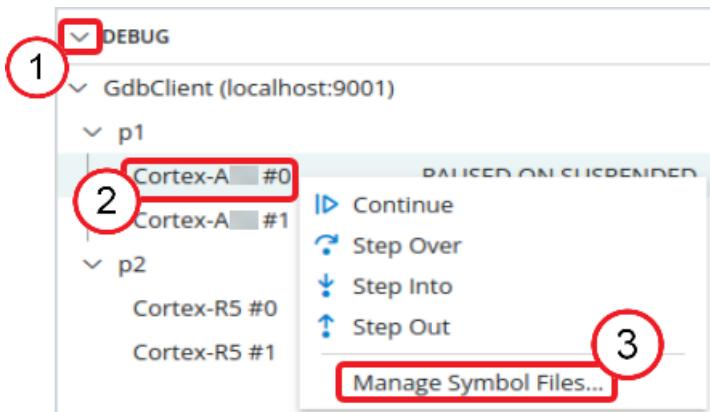
The debugger has launched, but for it to proceed, a symbol file needs to be added. A symbol file is the application ELF file created by the tool.

##### 1-4-1. Expand the **Debug** window (1).

The Debug window lists the processors available in the target board you are emulating.

##### 1-4-2. Right-click [**\*\*\* processor\_name \*\*\***] (2).

##### 1-4-3. Select **Manage Symbol Files** (3).



**Figure 299: Adding a Symbol File to the Debugger**

##### 1-4-4. Click the '+' symbol.

##### 1-4-5. Navigate to the following directory:

location of the ELF file

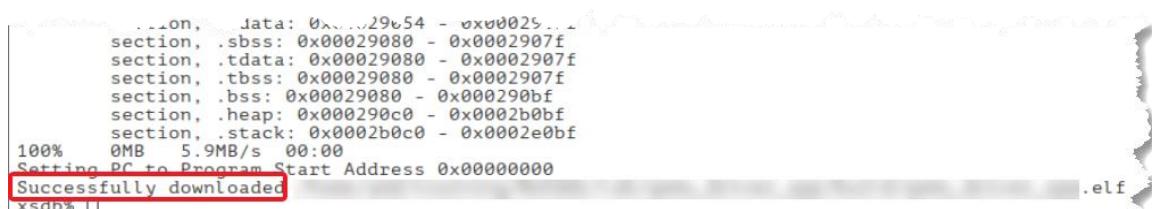
##### 1-4-6. Double-click **name of the ELF file**.

##### 1-4-7. In the XSDB console, enter the following command to download the ELF file into it:

```
xsdb% dow location of the ELF file/name of the ELF file
```

**Note:** \$TRAINING\_PATH may need to be expanded.

The debugger is now configured and launched in the main source file. The following message appears on the XSDB console:



**Figure 300: Successful Download Message**

## Switching Perspectives

- 1-1. A perspective is a collection of views that assist you in a specific task. There are several predefined views, including a Design perspective 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 301: Clicking the Desired Icon

## Debugging

### In This Section

|                   |     |
|-------------------|-----|
| Breakpoints ..... | 235 |
|-------------------|-----|

## Breakpoints

### In This Section

|  |                              |
|--|------------------------------|
| Opening the Breakpoints View.....        | 235                          |
| Adding Breakpoints.....                  | 236                          |
| Running to a Breakpoint .....            | Error! Bookmark not defined. |
| Creating a Conditional Breakpoint .....  | 237                          |
| Enabling and Disabling a Breakpoint..... | 239                          |
| Removing Breakpoints.....                | 240                          |

## Opening the Breakpoints View

---

### 1-1. Open the Breakpoints view.

- 1-1-1. Select the **BREAKPOINTS** view in the Debug window to see all of the automatically generated breakpoints as well as any breakpoints that you have created.

**Tip:** You can reopen the view by selecting Debug.

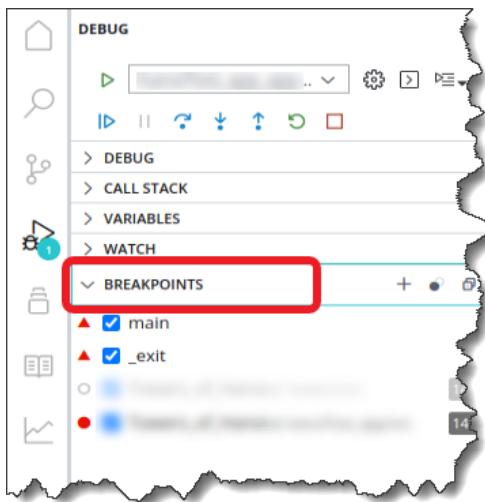


Figure 302: Locating the Breakpoints View

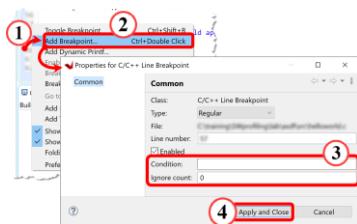
## Adding Breakpoints

### 1-1. Add a breakpoint at the current line.

**There are several methods to add a breakpoint at a specific line.**

**Method 1:** Use the breakpoint Properties dialog box:

- Right-click the line number (1)
- Select **Add Breakpoint** from the context menu (2)
- [Optional] Configure conditional breakpoint settings (3)
- If no specifics are identified, then an unconditional breakpoint is created at the selected line.
- Conditional breakpoints work by pausing at the line that the conditional breakpoint was set. If the conditions are met, the execution is suspended. If the conditions are not met, then execution continues.
- Click **Apply and Close** (4)



**Figure 303: Adding a Breakpoint Using the Full Breakpoint Window**

**Method 2:** Right-click the line number and select **Toggle Breakpoint** from the context menu.

If you select **Toggle Breakpoint**, then an unconditional breakpoint will be created (or removed if one already existed) on the line you specified using the default breakpoint properties.

**Method 3:** Double-click the line number (this is the same behavior as **Toggle Breakpoint**).

### 1-1-1. Using one of the above methods, add a breakpoint at the specified line.

## Running to a Breakpoint

### 1-1. Run to the next breakpoint.

#### 1-1-1. Click the **Continue** icon ( ) to continue operation.

The application will run until an unconditional breakpoint is met or a conditional breakpoint whose condition is satisfied is met.

## Creating a Conditional Breakpoint

Breakpoints can be made conditional. This means that when the breakpoint is set at a 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 a breakpoint. Alternatively, you can access the properties for an existing breakpoint.

### 1-1. Create a conditional breakpoint.

- 1-1-1. Right-click the breakpoint in the Editor window (1).

**Note:** If the breakpoint already exists, you can also right-click the breakpoint in the Breakpoints view.

- 1-1-2. Select **Add Breakpoint** from the context menu (2).

**Note:** If you access the breakpoints from the Breakpoints view, select **Breakpoint Properties** to open the associated dialog box.

- 1-1-3. Enter **your conditional expression** in the Condition field (3).

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.

- 1-1-4. Enter **the number of times the met condition should be skipped** in the Ignore count field (4).

The ignore count capability allows the breakpoint to be reached this many times before execution is paused.

For example, if a conditional breakpoint were placed on a line of code within a loop, and the ignore count value were set to 3, then on the third occurrence of the condition being met, the breakpoint would trigger and pause the execution.

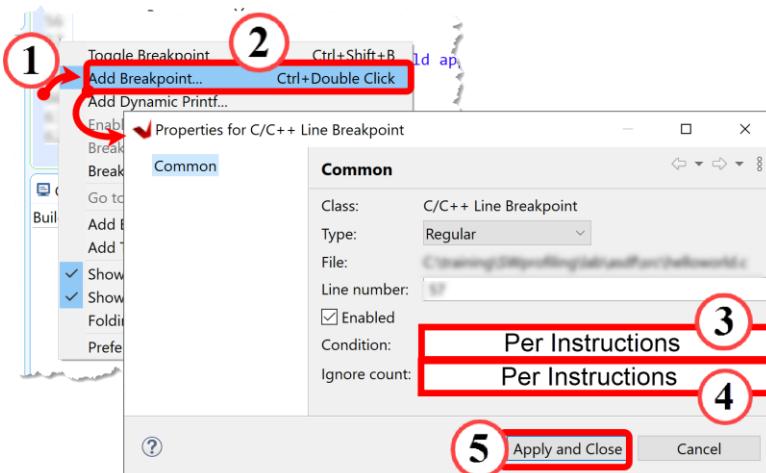


Figure 304: Configuring a Conditional Breakpoint

- 1-1-5. Click **Apply and Close** to save the breakpoint properties and exit (5).

## Enabling and Disabling a Breakpoint

- 1-1. Breakpoints can be enabled or disabled using one of the following methods:

- From the Breakpoints tab
- From the Editor window

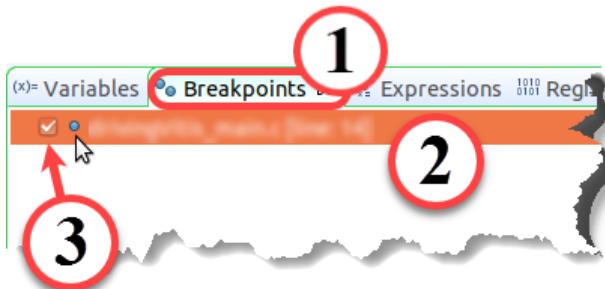
- 1-1-1. Select the **Breakpoints** tab (1).

If the Breakpoints tab is not visible, select using **Window > Breakpoints**.

- 1-1-2. Select the breakpoint to toggle (2).

- 1-1-3. Select the check box to toggle the state of this breakpoint (3).

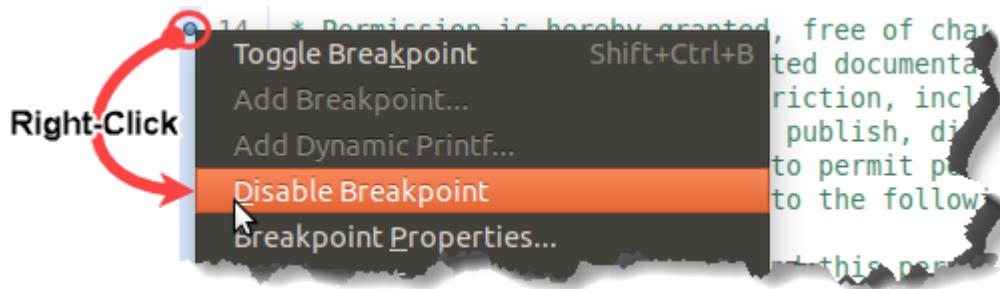
A check mark will be absent when the breakpoint is disabled.



**Figure 305: Setting and Clearing Breakpoints**

Alternatively, breakpoints can be toggled from the Editor window as follows.

- 1-1-4. Locate the breakpoint to toggle in the Editor window.
- 1-1-5. Right-click the **Breakpoint** icon and select either **Enable Breakpoint** or **Disable Breakpoint**.



**Figure 306: Enabling and Disabling a Breakpoint from the Editor**

**Note:** If the breakpoint is currently enabled, then the **Disable Breakpoint** option will appear in the context menu. Conversely, if the breakpoint is currently disabled, then the **Enable Breakpoint** option will appear in the context menu.

## Removing Breakpoints

### 1-1. Remove a breakpoint from a line of code.

1-1-1. Right-click the breakpoint that you want to remove.

1-1-2. Select **Toggle Breakpoint**.

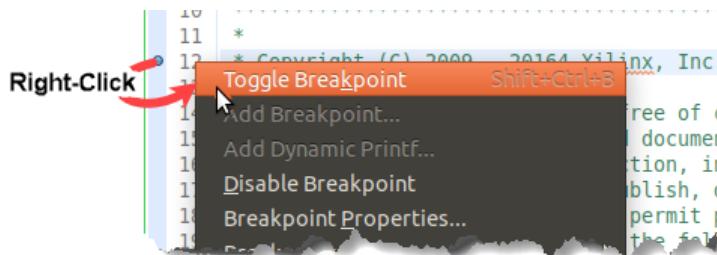


Figure 307: Selecting Toggle Breakpoint

-- 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 remove the selected breakpoint.

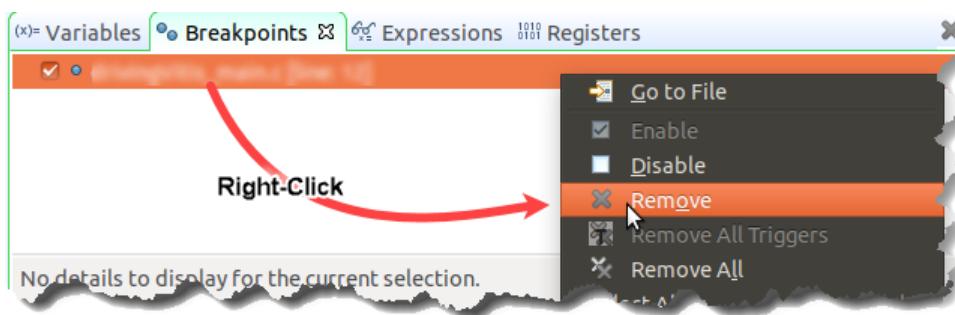


Figure 308: Selecting Remove

## Configuring the Vitis Serial Terminal

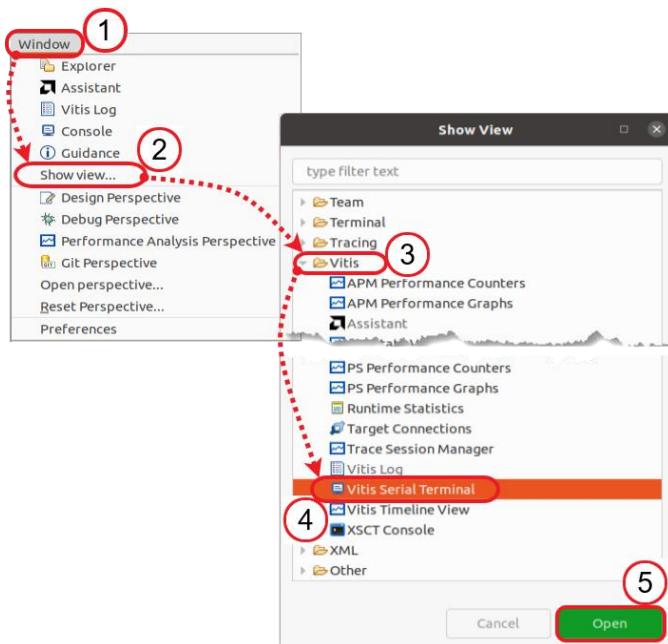
The Vitis serial terminal is an interface that only supports serial port/UART communications. The more general Terminal tab supports other formats, such as SSH and Telnet.

Since serial port communications occur over the USB, which is an enumerated interface, the board must be powered on with the cable connected for the port to be recognized.

If you are using the VM, ensure that the proper USB port is allowed to cross the host/VM threshold by selecting **Devices > USB > Xilinx <board selection> [0800]** in the VirtualBox Manager for Zynq UltraScale+ MPSoC/Versal adaptive SoC boards, or selecting **Devices > USB > Digilent Adept USB Device [0700] \*AND\* Silicon Labs CP2103 USB to UART Bridge Controller [0100]** for Zynq 7000 SoC boards.

### 1-1. Locate the Vitis Serial Terminal tab.

- 1-1-1. If the Vitis Serial Terminal tab is not currently visible, select **Window** (1) > **Show view** (2) to see a list of commonly opened views.
- 1-1-2. Expand the **Vitis** entry (3).
- 1-1-3. Select **Vitis Serial Terminal** (4).



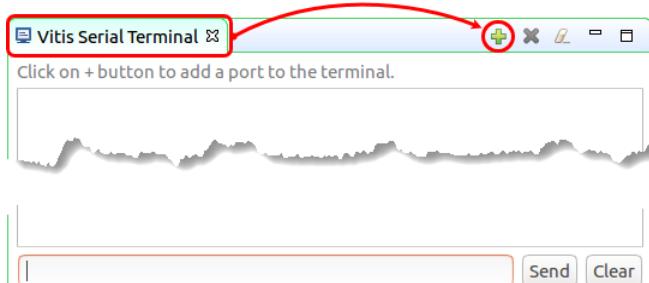
**Figure 309: Opening the Vitis Serial Terminal**

- 1-1-4. Click **Open** to access the Vitis serial terminal (5).

This tab typically opens adjacent to the Console tab. You can leave this terminal in this position or drag the tab to another window region.

## 1-2. Configure the Vitis serial terminal.

- 1-2-1. Click the green '+' sign to open the Connect to Serial Port dialog box.



**Figure 310: Adding or Associating a Port to the Terminal**

- 1-2-2. Select the serial port that is connected to the device you want to communicate with (1).

This is the port number associated with the serial port/USB connection from your board. Your board must be powered on to see this port.

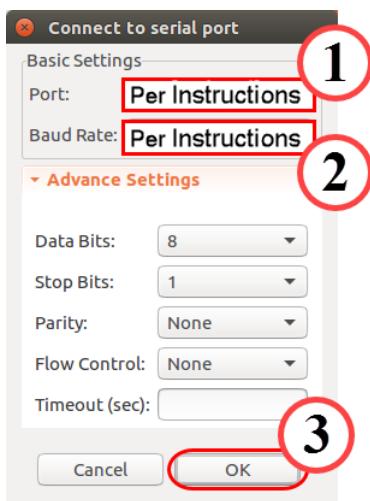
Some boards have two USB ports: one for the serial port and another for JTAG. Other evaluation boards use a multi-ported USB serial port bridge that contains four serial ports, three COM ports, and one serial port dedicated to the JTAG chain(s). If this is the case, the three COM ports will be in numerical order.

You should select the lowest number COM port that is attached to PS uart 0.

**Note:** If you are using the CustEd VM, the port selection for the multi-ported devices will be /dev/ttyUSB1. The Zynq 7000 SoC boards will use /dev/ttyUSB0 for the serial port and /dev/ttyUSB1 for the JTAG connection.

- 1-2-3. Set the baud rate to **115200** (2).

- 1-2-4. Leave the other settings at their defaults.



**Figure 311: Configuring the Terminal**

- 1-2-5. Click **OK** to save these settings and begin the terminal session (3).

## Closing the Vitis IDE

### 1-1. Close the Vitis Unified IDE.

1-1-1. Select **File > Close Window** to close the tool.

## QEMU Operations [Last Updated Version 2019.1]

### In This Section

|   |     |
|---|-----|
| Opening the VirtualBox and Running an OS .....                              | 243 |
| Running an Application on QEMU .....  | 245 |
| Selecting the QEMU Console Log .....  | 247 |
| Setting Up a (QEMU) System Debugger Debug Configuration .....               | 247 |
| Stopping QEMU .....   | 250 |
| Starting QEMU .....   | 250 |
| Shutting Down a Running QEMU Virtual Machine Using the Command Prompt ..... | 250 |

## 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 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.

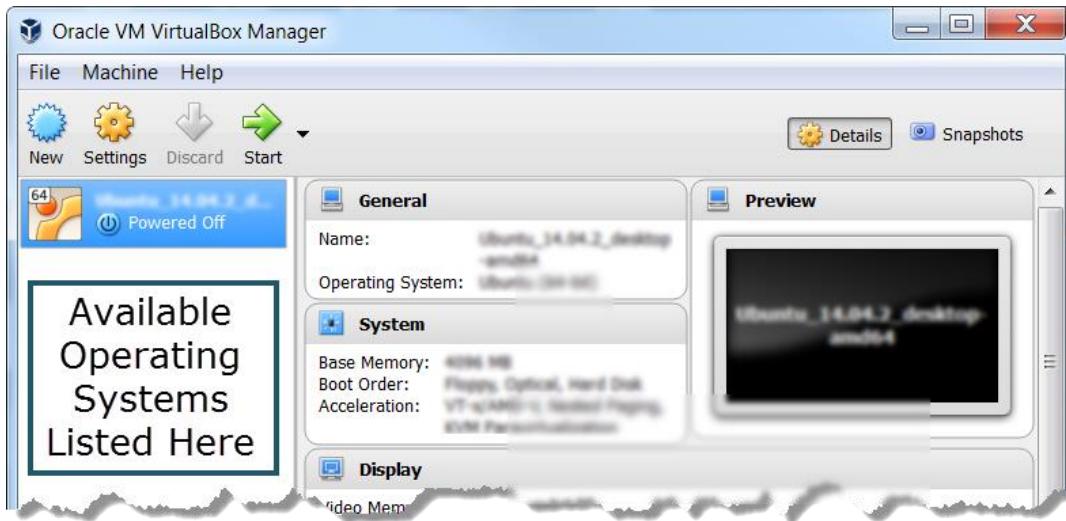


Figure 312: Opening VirtualBox

- 1-1-2. Double-click 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.

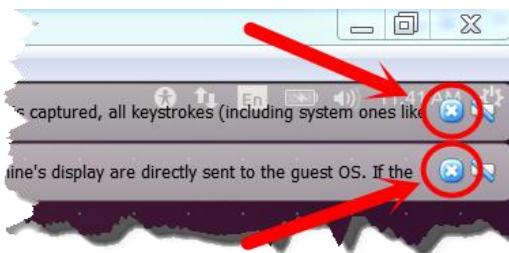


Figure 313: Closing Information Messages

## Running an Application on QEMU

You will now configure the application to run on QEMU.

### 1-1. Run *your application project name* with the default configuration settings.

- 1-1-1. Right-click **your application project name** in the Project Explorer to open the context menu.
- 1-1-2. Select **Run As > Run Configurations**.
- 1-1-3. Select **Xilinx C/C++ application (System Debugger on QEMU)** to indicate which type of configuration to create (1).
- 1-1-4. Click the **New Launch Configuration** icon to create the configuration (2).

This will create the specific configuration (3).

You can review the automatically filled fields under the various tabs.

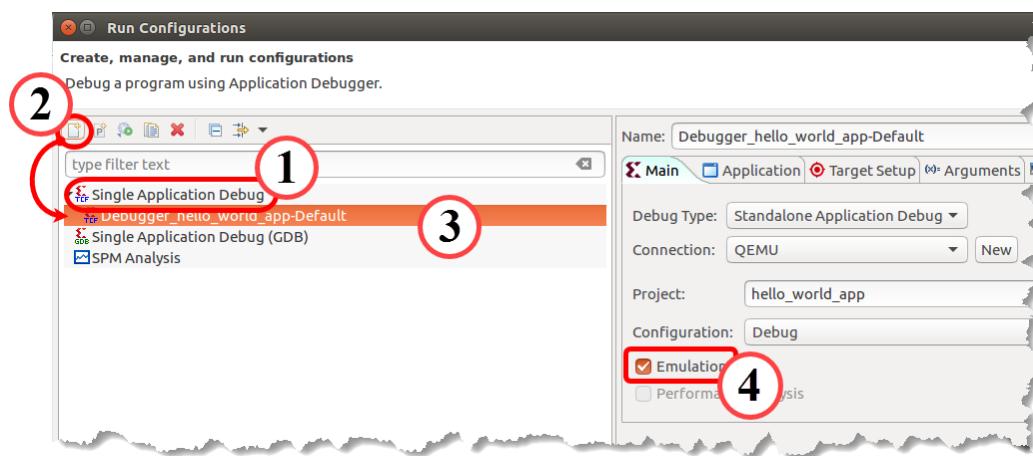
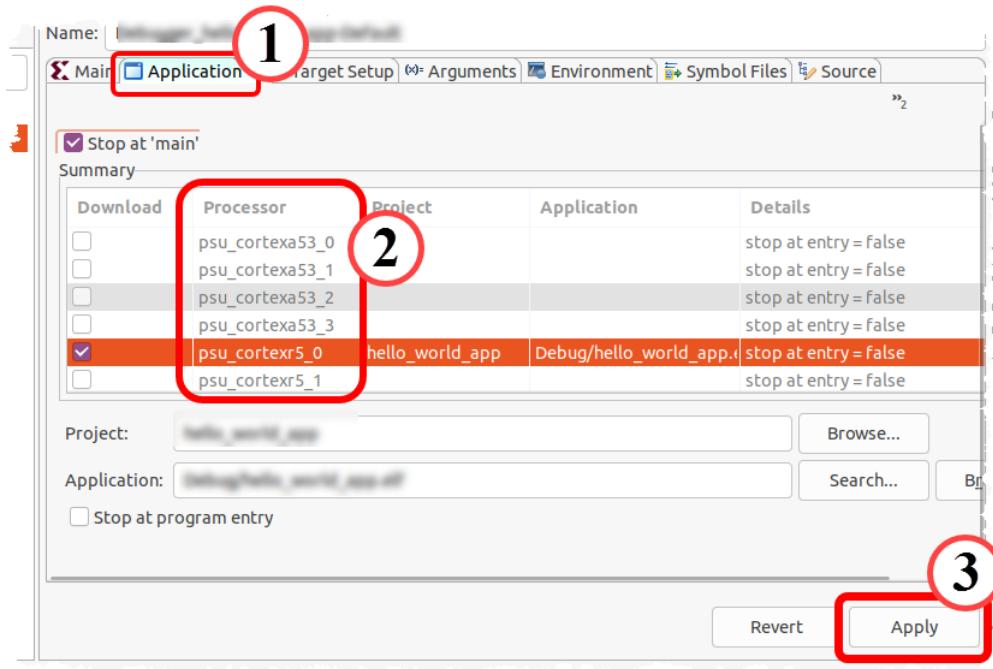


Figure 314: Verifying the Target Setup Tab (Example)

**1-1-5.** Select the **Application** tab to view the details for the application (1).

**1-1-6.** Select the processor you wish to target in the list box (2).

No changes are required for this project; however, it is worth noting how the various applications can be mapped to processors. This is useful when you are launching programs on multiple processors in a complete multi-processor environment rather than just testing an application running on a single processor at a time.



**Figure 315:** Verifying the Application Tab

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

**1-1-7.** Click **Run** to launch QEMU and run the applications listed in the QEMU emulator (3).

You should see a dialog box open indicating that QEMU is being configured and the results posted to the serial port will appear in the Emulation Console.

## Selecting the QEMU Console Log

The QEMU Console, sometimes referred to as the emulation console, displays the information coming from the QEMU emulator. Here you will open this normally hidden view.

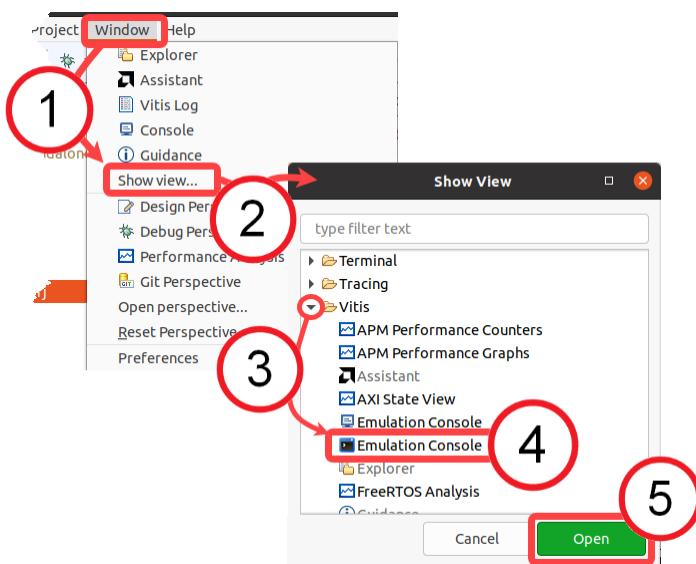
### 1-1. Select the emulation console.

- 1-1-1. Select **Window > Show View > Other** to open the list of views.

A dialog box will open with a list of available views.

- 1-1-2. Expand the **Xilinx** folder to see the options.

- 1-1-3. Select **Emulation Console** to open a view to the emulation console.



**Figure 316: Selecting Emulation Console**

- 1-1-4. Click **OK** to close the dialog box.

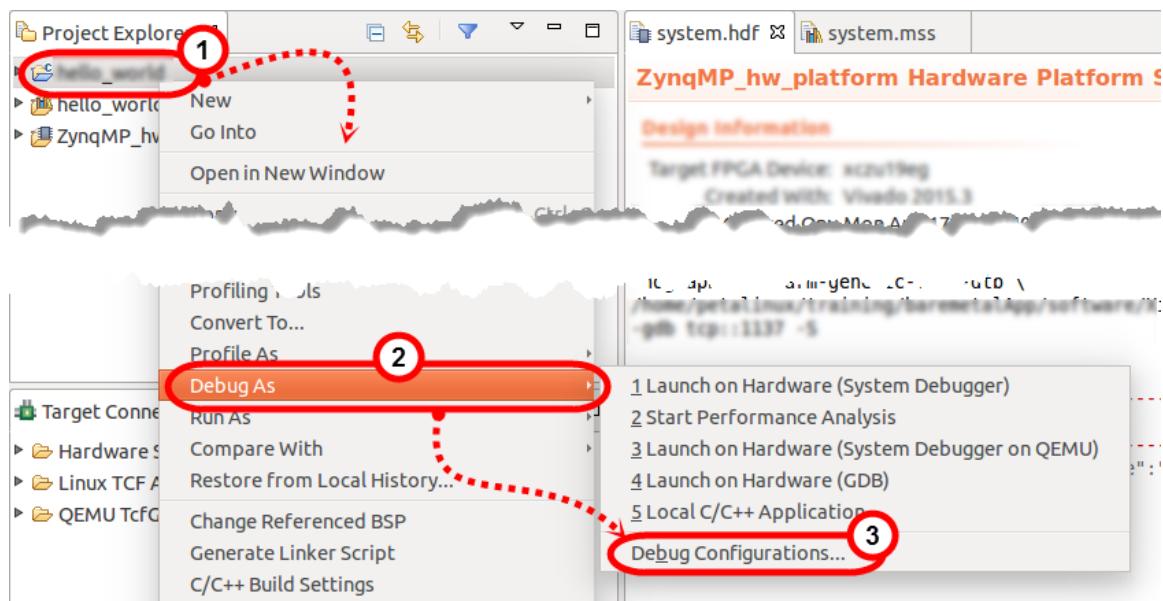
The QEMU Console log is now visible in the C/C++ perspective, usually in the lower-right corner when in the C/C++ (Run) perspective or in the upper-right corner when in the Debug perspective.

## Setting Up a (QEMU) System Debugger Debug Configuration

A Debug configuration defines how you want the system to work when performing a debug operation and maps an ELF object file to a target for execution. Typically, a debug operation switches to the Debug perspective. While there are a significant number of switches and options, the most common are shown below.

### 1-1. Set up a debug configuration for a specific application project.

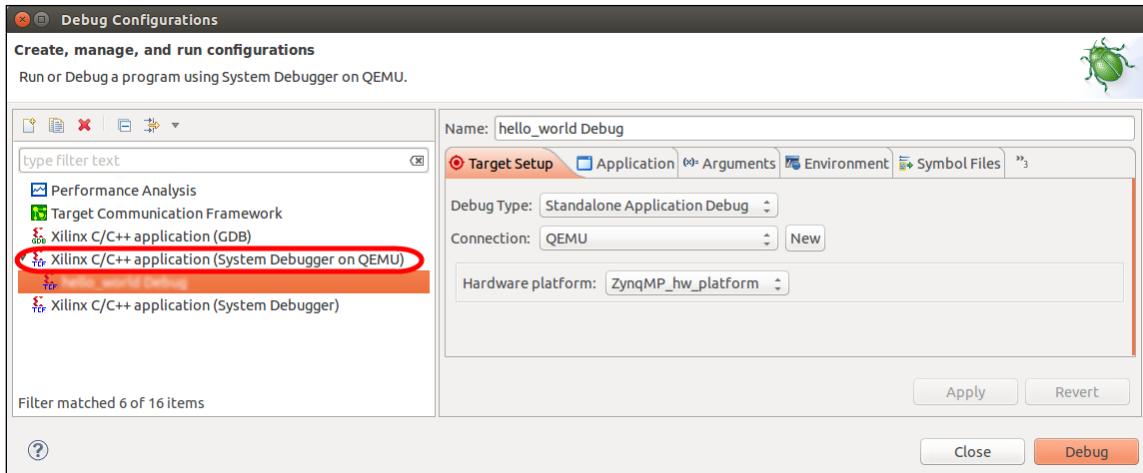
- 1-1-1. Right-click **your project name** in the Project Explorer pane that you want to create the Debug configuration for (1).
- 1-1-2. Select **Debug As** to open the sub-menu of predefined configurations and the configuration manager (2).
- 1-1-3. Select **Debug Configurations** to view all the available debug options (3).



**Figure 317: 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 318: Creating a New Debug Configuration (QEMU)**

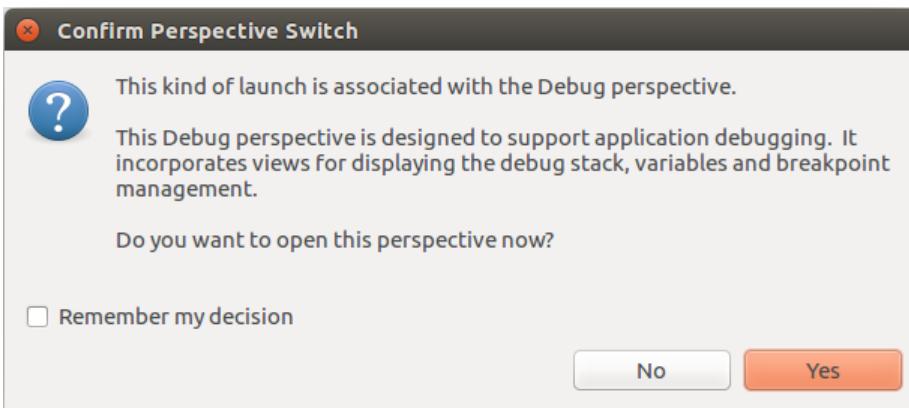
**Note:** A new Debug configuration is created. Note the default configuration name and other parameters that are automatically filled in. In most cases, you just need to click the Debug button to begin the session. This Debug configuration menu is useful when you want to later change debug parameters. 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.

- 1-1-6.** Click **Yes** if the Confirm Perspective Switch dialog box appears.

Optional: Select the **Remember my decision** option to prevent this dialog box from opening when switching to the debug perspective.



**Figure 319: Confirm Switch Perspective**

The Debug Perspective view opens.

## Stopping QEMU

---

### 1-1. Stop 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 QEMU

---

### 1-1. Start QEMU.

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

1-1-2. Click **Start QEMU**.

**Note:** The QEMU Status should change to Running.

1-1-3. Click **OK** to close the Configure QEMU Settings window.

## Shutting Down a Running QEMU Virtual Machine Using the Command Prompt

---

### 1-1. Shut down the running QEMU virtual machine.

1-1-1. Enter the following from the command line of the virtual machine (only for Linux):

```
[host] $ poweroff
```

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

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

# PetaLinux Operations [Last Updated Version 2021.1]

## In This Section

|  |     |
|--|-----|
| Creating a New PetaLinux Project (Using Any BSP with Project Name) ..... | 251 |
| Creating a New Application.....  | 251 |
| Creating a New Application (Enable App).....                             | 252 |
| Creating a New Application (Enable App) Using Any BSP .....              | 253 |
| Selecting the New Application to be Included in the Build Process .....  | 254 |
| Building the Linux Image.....  | 254 |
| Booting the Linux Image on the Board (Running Netboot) .....             | 256 |
| Running the Application in QEMU .....                                    | 257 |
| Configuring Network Settings .....                                       | 257 |
| Adding Ethernet Options.....   | 261 |

## Creating a New PetaLinux Project (Using Any BSP with Project Name)

**1-1. Use the `petalinux-create` command to create a new embedded Linux for the chosen platform.**

**1-1-1.** Change to the lab directory:

```
[host] $ cd ${TRAINING_PATH}/<the topic cluster name>/lab
```

This is done so that when the `petalinux-create` command is run, the project is created at this location.

**1-1-2.** Create a new PetaLinux project:

```
[host] $ petalinux-create -t project -s your BSP name -n your project name
```

**Note:** After the command is executed, information about the created project will be displayed.

**Note:** The PetaLinux project directory is `TRAINING_PATH/<the topic cluster name>/lab/your project name` and this is where all work for this project must be performed.

**1-1-3.** Change the directory to the PetaLinux project:

```
[host] $ cd your project name
```

## Creating a New Application

---

### 1-1. Create a new application.

The PetaLinux tools allow you to create user applications using predefined C, C++, and autoconf templates. The autoconf template is used for GNU autoconfig.

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. Change the directory to the PetaLinux project:

```
[host] $ cd ${TRAINING_PATH}/<the topic cluster name>/lab/your  
project name
```

- 1-1-2. Enter the following command to create a new user application inside the PetaLinux project:

```
[host] $ petalinux-create -t apps --name your application --  
template c
```

## Creating a New Application (Enable App)

---

### 1-1. Create a new application and enable it in the root file system.

The PetaLinux tools allow you to create user applications using predefined C, C++, and autoconf templates. The autoconf template is used for GNU autoconfig.

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 directory:

```
 ${TRAINING_PATH}/<the topic cluster name>/lab/the name of the  
PetaLinux project
```

- 1-1-2. Enter the following command to create a new user application inside a PetaLinux project:

```
[host] $ petalinux-create -t apps --name your application --  
enable
```

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

## Creating a New Application (Enable App) Using Any BSP

### 1-1. Create a new application and enable it in the root file system.

**The PetaLinux tools support multiple predefined templates, enabling you to more easily create a project tailored to your needs. Here you will create a user application from the default C template.**

**Other templates exist to facilitate other types of projects. For more details, refer to UG1144: PetaLinux Tools Documentation: Reference Guide. All templates include application source code and makefiles for easy configuration and compilation.**

- 1-1-1. Make sure that you are in the PetaLinux project directory \$TRAINING\_PATH/<the topic cluster name>/lab/the name of the PetaLinux project.
- 1-1-2. Enter the following command to create a new user application inside a PetaLinux project:

```
[host] $ petalinux-create -t apps --name the desired source file  
--enable
```

**Note:** After execution the following information will be printed.

INFO: Create apps: the desired source file

INFO: New apps successfully created in \$TRAINING\_PATH/<the topic cluster name>/lab/your project name/project-spec/meta-user/recipes-apps/your application project name

INFO: Enabling created component...

INFO: Sourcing build environment

INFO: Silentconfig rootfs

INFO: your application project name has been enabled

**Note:** The new application that you have created can be found in the <project-root>/project-spec/meta-user/recipes-apps/your application project name directory, where <project-root> is \$TRAINING\_PATH/<the topic cluster name>/lab/your project name.

The --enable option automatically enables the just created application in the project's root file system. To create a C++ application template, pass --template c++ as another parameter to the just entered command.

## Selecting the New Application to be Included in the Build Process

**1-1. Select the new application to be included in the build process. The application is not enabled by default.**

**1-1-1.** Ensure that your current working directory (project directory) is \$TRAINING\_PATH/<the topic cluster name>/lab/your project name.

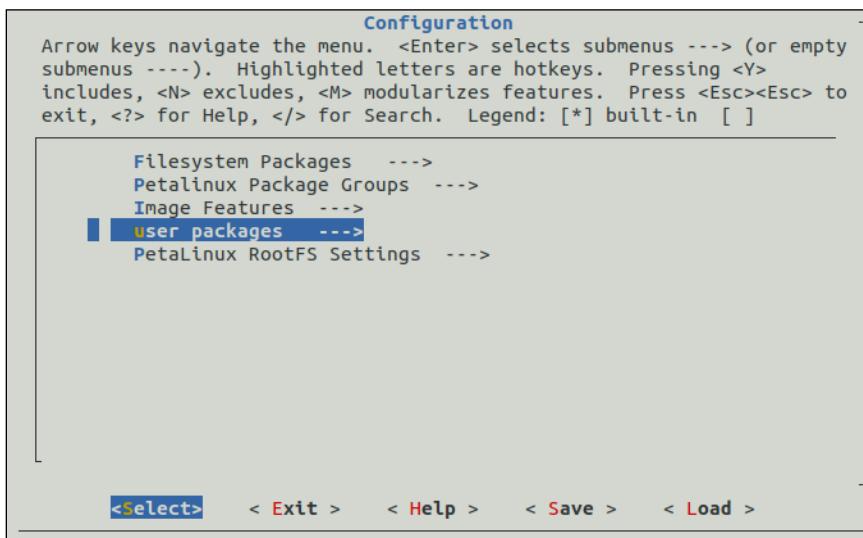
**1-1-2.** Enter the following command to launch the rootfs configuration menu:

```
[host] $ petalinux-config -c rootfs
```

**Note:** Make sure that the terminal window is at least 80 columns wide. If your terminal is not at least 80 columns wide, you will see "Error: Failed to config linux/rootfs! Your display is too small to run menuconfig."

The Linux/rootfs configuration menu opens.

**1-1-3.** Press the **Down Arrow** key (↓) to scroll down the menu to **user packages**.



**Figure 320: Linux/rootfs Configuration Menu**

**1-1-4.** Press <Enter> to go into the user packages submenu.

The new application **helloworld-app** is listed in the menu.

**1-1-5.** Press the <Space Bar> to select the application **helloworld-app**.

Selecting the application will be included in the build process.

**1-1-6.** Select **Exit** and save the configuration.

## Building the Linux Image

---

### 1-1. Build the Linux image.

This process is time consuming, typically taking 60 minutes or more depending on your system configuration. If you have time, you can build the image yourself with the process provided here, or you can skip this instruction and proceed with the next step in the lab.

**Extremely important! Before running the `petalinux-build` command, make sure that you are connected to the internet.**

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

```
[host] $ pwd
```

If you are not at the proper location, you can navigate there by entering:

```
cd ${TRAINING_PATH}/<the topic cluster name>/lab/your project name
```

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

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

Running `petalinux-build` in the project directory will build the system image, including the selected user application `your project name`.

The full compilation log (`build.log`) is stored in the build subdirectory of the PetaLinux project. The Linux software images and the device tree are generated in the `<plnx_project_root>/images/linux` subdirectory of the PetaLinux project.

## Booting the Linux Image on the Board (Running Netboot)

### 1-1. Boot the new Linux image on the board.

- 1-1-1.** Press any key to stop auto-boot when you see messages similar to the following in the GtkTerm window:

```

GtkTerm - /dev/ttyUSB1 115200-8-N-1
File Edit Log Configuration Controlsignals View Help
Xilinx Zynq MP First Stage Boot Loader
Release 201          - 12:19:18
NOTICE: ATF running on XCZU7EV/silicon v4/RTL5.1 at 0xffffea000
NOTICE: BL31: Secure code at 0x0
NOTICE: BL31: Non secure code at 0x8000000
NOTICE: BL31: v2.0(release):xilinx-v2018.3-720-g80d1c790
NOTICE: BL31: Built : 12:18:28, Aug 21 2019
PMUFW: v1.1

U-Boot 2019.01 (Aug 21 2019 - 12:17:16 +0000)

Board: Xilinx ZynqMP
DRAM: 2 GiB
EL Level: EL2
Chip ID: zu7ev
MMC: mmc@ff170000: 0
Loading Environment from SPI Flash... SF: Detected n25q512a with page size
256 Bytes, erase size 64 KiB, total 64 MiB
*** Warning - bad CRC, using default environment

In:    serial@ff000000
Out:   serial@ff000000
Err:   serial@ff000000
Board: Xilinx ZynqMP
Bootmode: LVL_SHFT_SD_MODE1
Reset reason: EXTERNAL
Net:   ZYNQ GEM: ff0e0000, phyaddr c, interface rgmii-id
eth0: ethernet@ff0e0000
U-BOOT for BasicHwDesign

ethernet@ff0e0000 Waiting for PHY auto negotiation to complete.....
..... TIMEOUT !
Hit any key to stop autoboot: 0
ZynqMP>
ZynqMP>
ZynqMP>
ZynqMP> ■

```

/dev/ttyUSB1 115200-8-N-1      DTR RTS CTS CD DSR RI

**Figure 321: Stopping the Autoboot (ZedBoard)**

If the system has booted, reset the board (BTN7) again and stop auto-boot.

**Note:** If you have finished setting the IP address and server IP address and saved once, you can directly run the command `run netboot`.

- 1-1-2.** Set the IP address for the target board so that it can communicate to the host and load the new image:

`Zynq> setenv ipaddr 192.168.1.10`

- 1-1-3.** Set the TFTP server IP to the host IP:

`Zynq> setenv serverip 192.168.1.1`

- 1-1-4. Save the environment to the SPI Flash:

```
Zynq> saveenv
```

- 1-1-5. Download and boot the new image using TFTP:

```
Zynq> run netboot
```

This command will download the `image.ub` file from `/tftpboot` on the host to the main memory of the ARM Cortex MPCore system and boot the system with the image.

## Running the Application in QEMU

### 1-1. Run the application in QEMU.

- 1-1-1. Enter the following command to boot the newly built PetaLinux image through QEMU:

```
# petalinux-boot --qemu --kernel
```

- 1-1-2. After the system boots, log into the system by entering `root` as both the login name and password.

## Configuring Network Settings

### 1-1. Launch VirtualBox.

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

- 1-1-2. Select the **name of the virtual machine** virtual machine titled from the left pane (1).

- 1-1-3. Double-click the virtual machine (1) or click the green right arrow icon to launch the selected virtual machine (2).

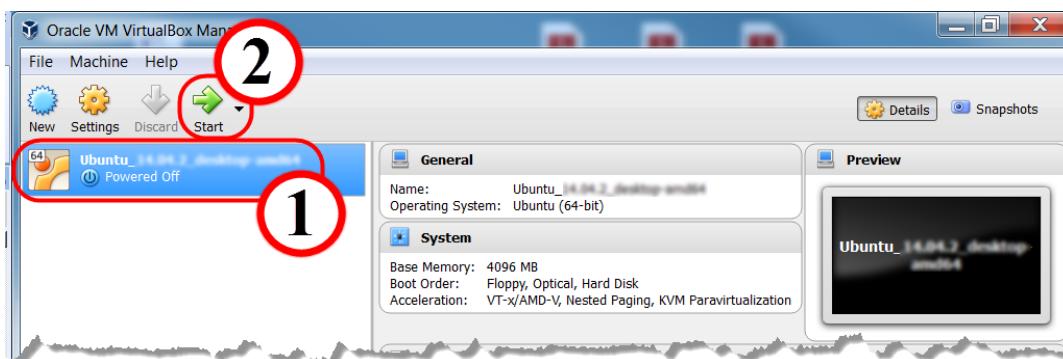


Figure 322: VirtualBox Manager Window

The name of the virtual machine virtual machine will now launch in a new window.

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

### 1-2. Disable the static IP settings.

1-2-1. Press **<Ctrl + Alt + T>** to open a new terminal window.

1-2-2. Enter the following command to open the interface file:

```
gedit /etc/network/interfaces
```

1-2-3. Comment all the lines in the interfaces file except the two lines as shown below:

```
#interfaces(5) file used by ifup(8) and ifdown(8)

auto lo

iface lo inet loopback

#auto eth0

iface eth0 inet static
    address 192.168.1.1
    netmask 255.255.255.0
    broadcast 192.168.1.255
    gateway 192.168.1.
```

1-2-4. Press **<Ctrl + S>** to save the changes.

1-2-5. Click **<X>** in the upper-left corner of the gedit window to exit the editor.

1-2-6. Enter the following command in the terminal to shut down the OS:

```
sudo poweroff
```

1-2-7. Enter **xilinx** as password when asked to enter a password.

### 1-3. Change the VirtualBox network settings.

1-3-1. Select **Network** as shown in the figure below.

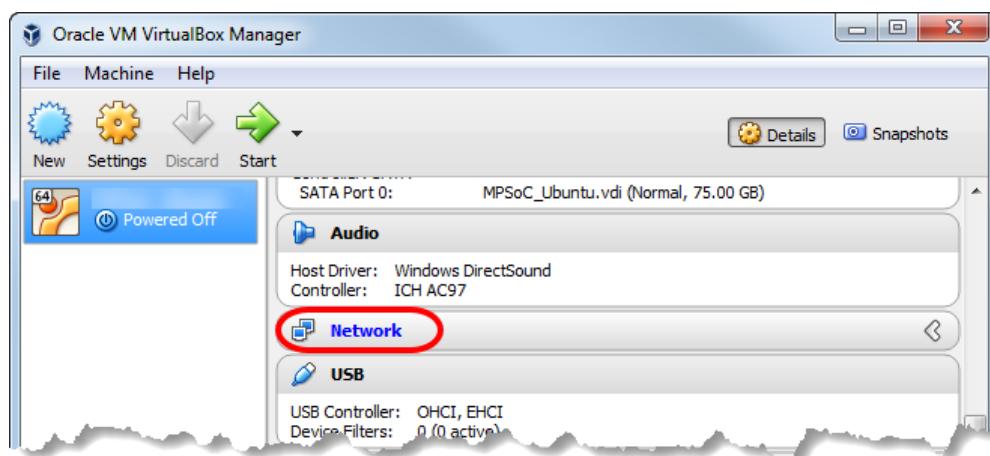


Figure 323: Selecting the Network Tab

- 1-3-2. Select the **Enable Network Adapter** option to enable the network adapter.

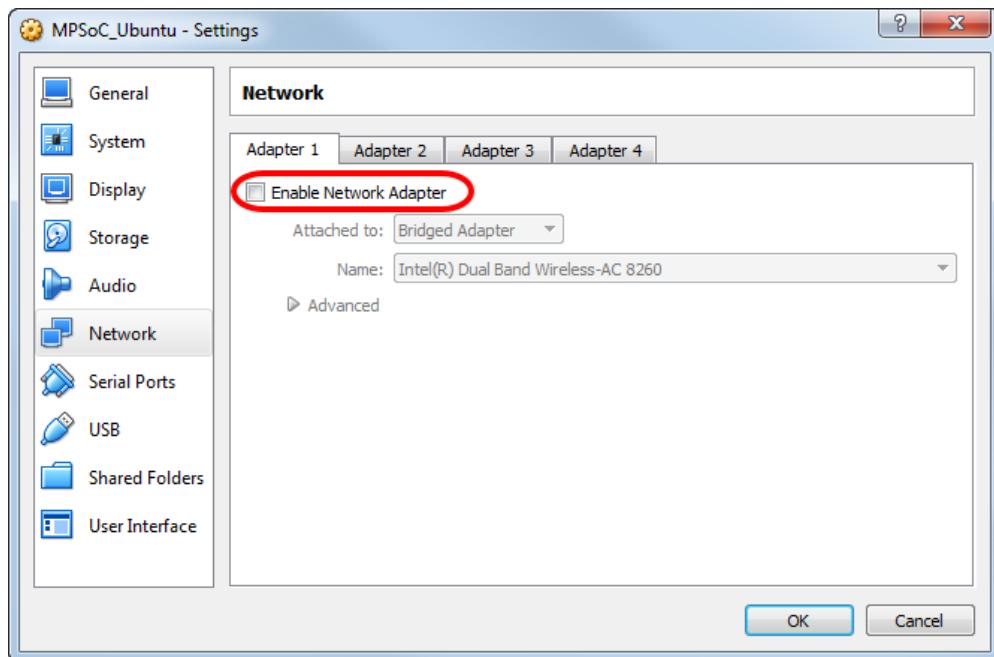


Figure 324: Enabling the Network Adapter

- 1-3-3. Select the appropriate name option from the drop-down list.

This will vary from system to system. The figures shown here are for illustration purposes only.

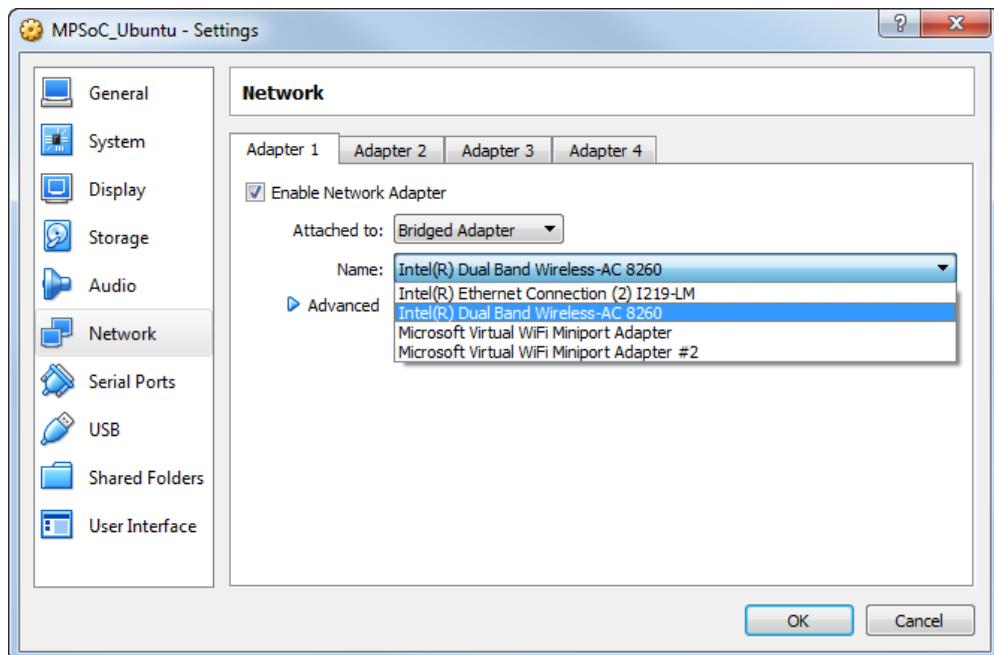
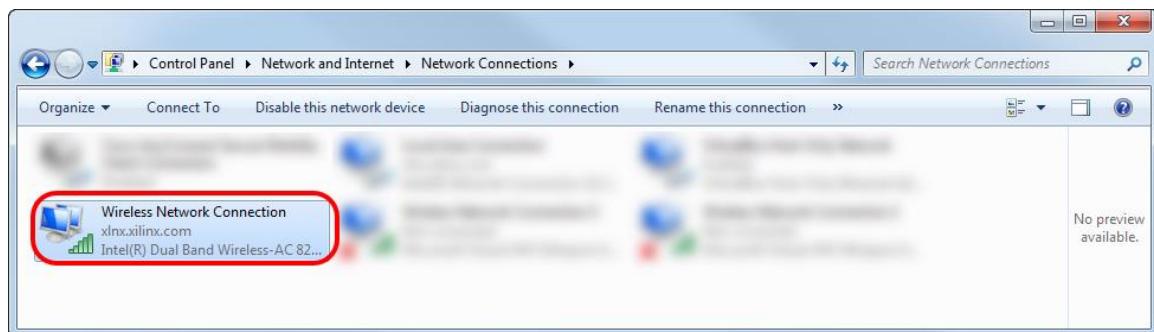


Figure 325: Selecting the Appropriate Network Connection

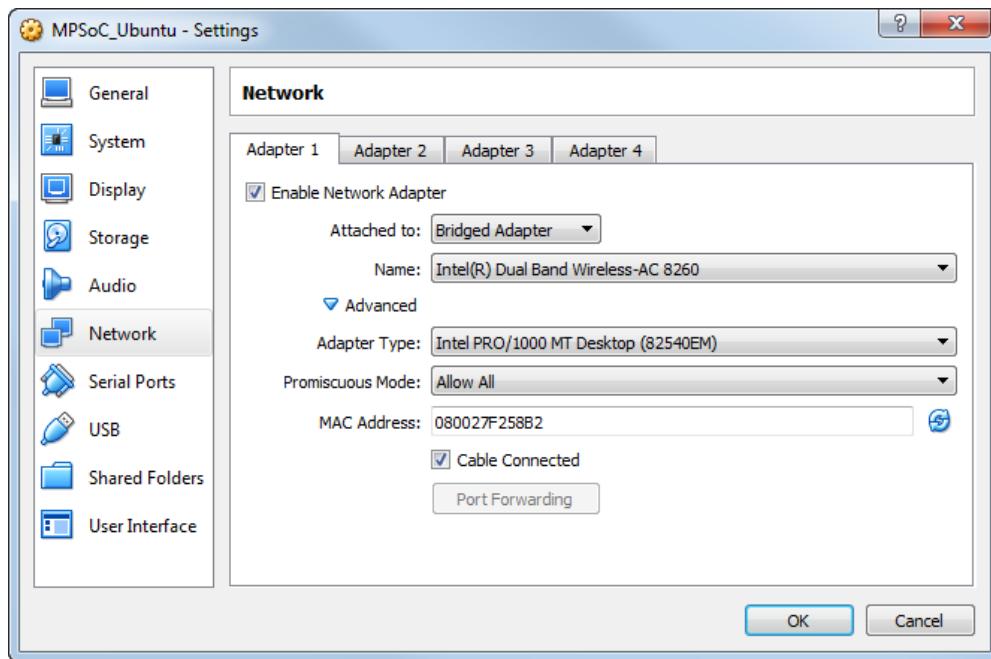
This example shows that **Intel(R) Dual Band Wireless-AC 8260** (or the wired network **Intel(R) Ethernet Connection (2) I2I9-LM**) is selected because it is the name of our network connection on the Windows machine running at the backend as shown below.



**Figure 326: Network Connection on Windows Machine**

You will have to check the network connection of your Windows machine and then select the appropriate name in the VirtualBox.

- 1-3-4. Select the **Advanced** option in the Adapter tab in the VirtualBox to show the extra configuration parameters.
- 1-3-5. Edit the configuration as shown below.



**Figure 327: Configuring Advanced Options**

**Note:** The network connection name may be different in your environment.

- 1-3-6. Click **OK** to bring the changes into effect.

## Adding Ethernet Options

### 1-1. Add the Ethernet information in the device tree.

#### 1-1-1. Change to the device tree folder:

```
[host] $ cd <project-root>/project-spec/meta-user/recipes-bsp/device-tree/files
```

**Note:** <project-root> here is the \$<the topic cluster name>/lab/your project name directory.

#### 1-1-2. Open the system-user.dtsi file:

```
[host] $ gedit system-user.dtsi
```

#### 1-1-3. Add the following lines at the end of the file:

**Note:** You will find the completed system-user.dtsi in the \$<the topic cluster name>/support directory.

```
&gem3 {
    status = "okay";
    local-mac-address = [00 0a 35 00 02 90];
    phy-mode = "rgmii-id";
    phy-handle = <&phy0>;
    phy0: phy@c {
        reg = <0xc>;
        ti,rx-internal-delay = <0x8>;
        ti,tx-internal-delay = <0xa>;
        ti,fifo-depth = <0x1>;
    };
};

&qspi {
    status = "okay";
    flash@0 {
        compatible = "m25p80"; /* 32MB */
        #address-cells = <1>;
        #size-cells = <1>;
        reg = <0x0>;
        spi-tx-bus-width = <1>;
        spi-rx-bus-width = <4>; /* FIXME also DUAL
configuration possible */
        spi-max-frequency = <108000000>; /* Based on DC1 spec
*/
        partition@qspi-fsbl-uboot { /* for testing purpose */
            label = "qspi-fsbl-uboot";
            reg = <0x0 0x100000>;
        };
        partition@qspi-linux { /* for testing purpose */
            label = "qspi-linux";
            reg = <0x100000 0x40000>;
        };
        partition@qspi-device-tree { /* for testing purpose */

```

```
        label = "qspi-device-tree";
        reg = <0x00140000 0x2000000>;
    };
partition@qspi-rootfs /* for testing purpose */
{
    label = "qspi-rootfs";
    reg = <0x02140000 0x80000>;
};
};

};
```

#### **1-1-4. Save and close the file.**

## **Board, OS, COM, and IP Address Tasks [Last Updated Version 2023.2]**

## In This Section

|  |     |
|--|-----|
| Configuring the Terminal.....  | 262 |
| Setting Up the ZCU104 Board .....  | 265 |
| Setting Up the VCK190 Board .....  | 266 |
| Determining the COM Port Assigned by the USB Driver .....                        | 269 |
| Linux Operations .....   | 271 |
| Running a Virtual Machine Using VirtualBox .....                                 | 272 |
| Cleaning Up the VirtualBox File System .....                                     | 273 |
| Setting the Static Host IP Address Using Windows 10 .....                        | 274 |
| Setting the Static Host IP Address Using Linux .....                             | 278 |
| Verifying Board Jumper/Switch Settings Configured to Boot from the SD Card ..... | 280 |
| Powering Up the Hardware Platform .....  | 280 |
| Preparing an SD Card.....  | 281 |
| Booting Linux .....  | 282 |
| Launching and Configuring the Terminal Emulator.....                             | 139 |
| Setting the Jumpers on the ZC702 Board.....                                      | 287 |
| Setting the Jumpers on the ZC706 Board.....                                      | 290 |
| Setting the Jumpers on the ZCU102 Board .....                                    | 293 |
| Setting the Jumpers on the ZCU104 Board .....                                    | 295 |
| Setting the Jumpers on the ZedBoard .....  | 298 |
| Setting Up the Hardware - KC705 .....  | 298 |
| Setting Up the Hardware - KCU105 .....   | 298 |

## Configuring the Terminal

The Terminal tab is used with a serial source (UART), SSH, or Telnet session. The following instructions will show you how to configure this tab to work with a serial UART.

### 1-1. Open the Terminal tab.

- 1-1-1. If the Terminal tab is not visible, select **Windows > Show View > Other > Terminal > Terminal** to open it or create a new Terminal tab.

This will open a Terminal tab in one of the panels. The tab can be moved to any other panel by click-dragging it to the new panel.

**Note:** More than one terminal can be enabled at a time.

- 1-1-2. Select the **Terminal** tab to access the terminal icons (1).

- 1-1-3. Click the **Settings** icon (2).

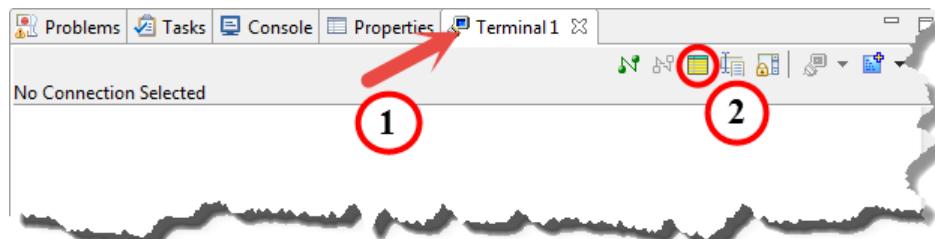


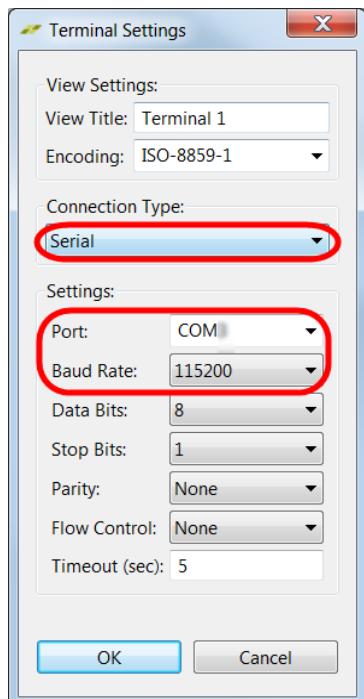
Figure 328: Accessing the Terminal Settings

Alternatively, you can also click the **Connect** icon (green terminal symbol) 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 329: Configuring the Terminal Settings****1-2-4.** Click **OK**.

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

## Setting Up the ZCU104 Board

### 1-1. Bring up the ZCU104 board.

- 1-1-1. Ensure that the board is powered off and that the board is connected to the host with both USB and Ethernet cables.

**Note:** This lab will have you boot using the desired boot mode mode, which requires the mode pins to be set as described below.

- 1-1-2. Locate SW6 on the board.

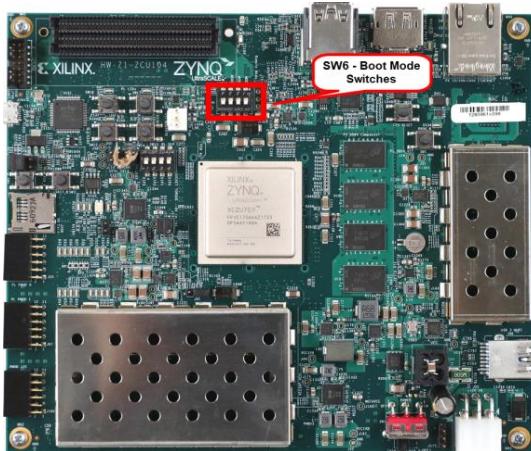


Figure 330: ZCU104 SW6 - Boot Mode Switches

- 1-1-3. Set SW6 as shown below to ensure that the board is configured to boot from **the desired boot mode**.

**Note:** Settings are shown to illustrate different boot mode settings. Make sure you select the desired boot mode.

| Boot Mode | Mode Pins [3:0]           | Mode SW6 [4:1] |
|-----------|---------------------------|----------------|
| JTAG      | 0000 / 0x0                | ON,ON,ON,ON    |
| QSPI32    | 0010 / 0x2 <sup>(1)</sup> | ON,ON,OFF,ON   |
| SD1       | 1110 / 0xE                | OFF,OFF,OFF,ON |

Figure 331: Board Configuration

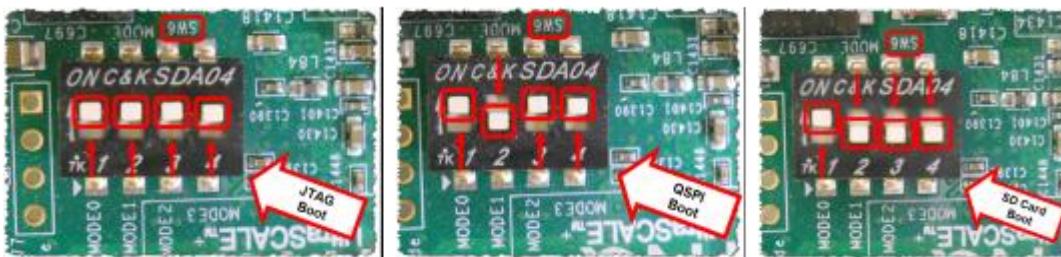


Figure 332: ZCU104 Boot Settings

- 1-1-4. Slide the power switch to the "ON" position to power on the board.

## Setting Up the VCK190 Board

### 1-1. Bring up the VCK190 board.

- 1-1-1. Ensure that the board is powered off by verifying that the power slide switch is in the position closest to the power connector (1).

The figure below shows the switch in the "ON" position.

- 1-1-2. Ensure that the power connector is plugged in (2).



Figure 333: Location of the Power Switch and Plug

- 1-1-3. If not already present, insert the programmed system controller microSD card into the microSD card slot on the bottom side of the board (1).

**Note:** This is NOT the SD card that is used to boot the Versal device—instead, it boots an UltraScale+ device that is used for overall board management.

- 1-1-4. Insert the microSD card containing the Versal device boot image (e.g., BOOT.pdi) into the microSD card slot on the top side of the board (2).

- 1-1-5. Connect the USB-C JTAG/UART cable between the board and the host (3, 4).

**Note:** There are some differences between versions of the board. The specific positioning of the connection is not relevant here.

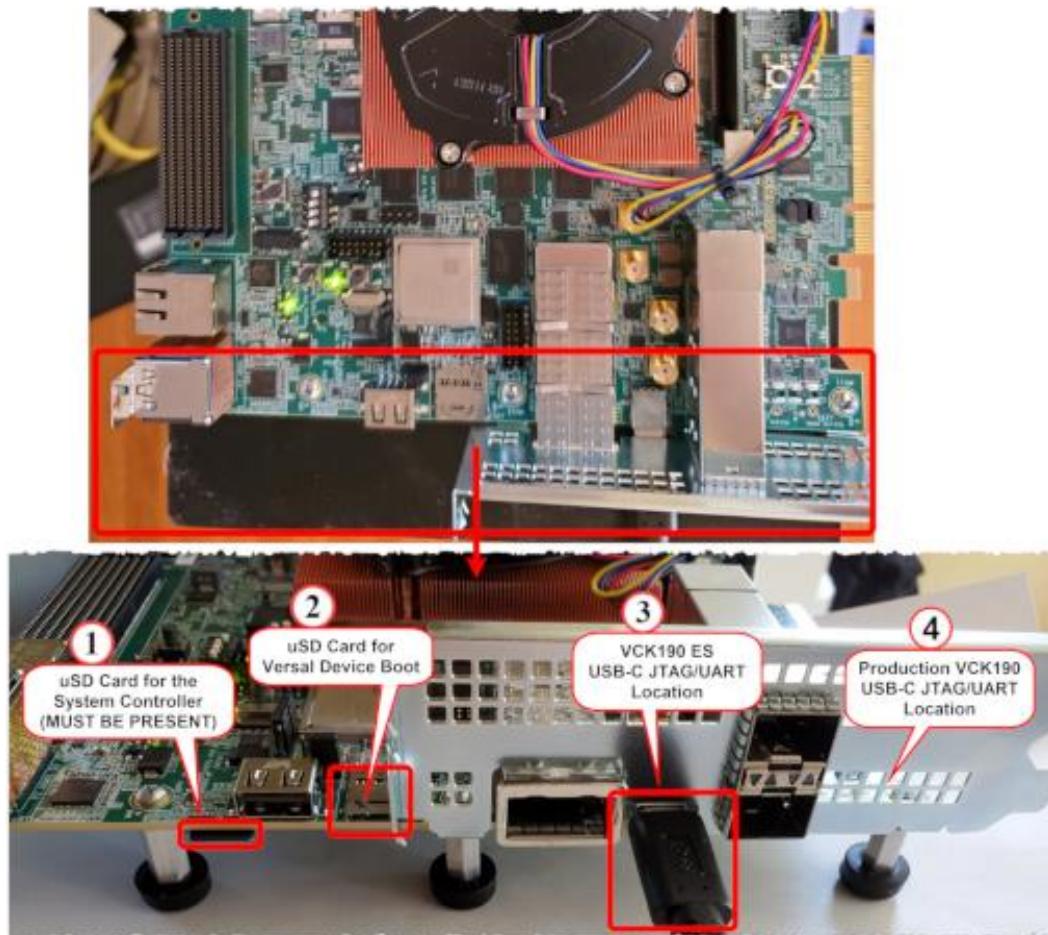


Figure 334: Left Side View of the VCK190

- 1-1-6. Locate SW1 on the board.

This switch configures the Versal device's boot mode, which should be set to **the desired boot mode**.

**Caution:** SW11 (located in the 11 o'clock position relative to the Versal device in the figure below) controls the mode settings for the system controller. The switch settings MUST NOT be changed from the SD card boot setting (1=ON; 2,3,4=OFF); otherwise, the system controller will not boot, blocking the Versal device from booting.

The figure below shows the JTAG configuration for the Versal device.



**Figure 335: JTAG Configuration**

- 1-1-7.** Set SW1 as shown below to ensure that the board is configured to boot from **the desired boot mode**.

**Note:** Settings are also shown below to illustrate different boot mode settings. Make sure you select the desired boot mode.

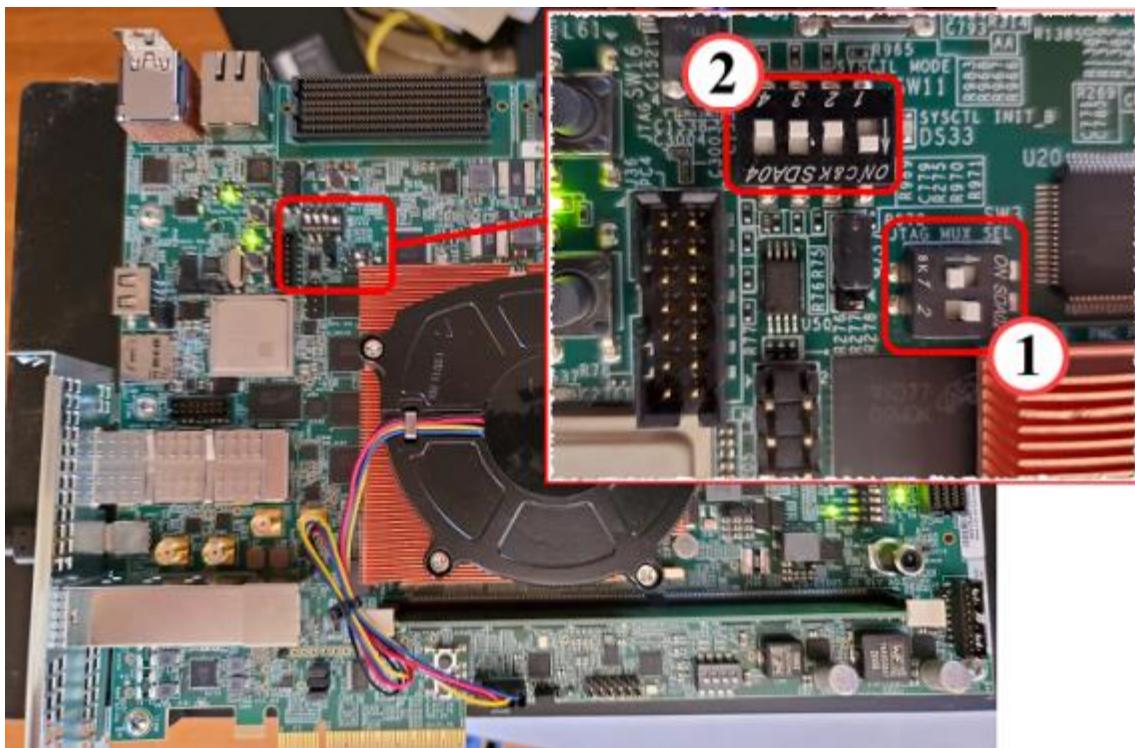
| Boot Mode | Mode Pins [3:0] | Mode SW1 [4:1] |
|-----------|-----------------|----------------|
| JTAG      | 000 / 0x0       | ON,ON,ON,ON    |
| QSPI32    | 0010 / 0x2      | ON,ON,OFF,ON   |
| SD1       | 1110 / 0xE      | OFF,OFF,OFF,ON |



**Figure 336: VCK190ES Boot Settings**

**1-1-8.** Verify that SW3 (which controls how the JTAG chain is routed) is set so that the switch marked "1" is off, and the switch marked "2" is on, as shown (1).

**1-1-9.** Verify that SW11 (which controls the boot mode for the system controller) is set as shown (2).



**Figure 337: Verifying SW3 and SW11**

**1-1-10.** Slide the power switch to the "ON" position to power on the 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 COM<sub>x</sub> 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 Vitis Serial Terminal tab, Tera Term, or other software, it is necessary to identify the COM port number associated with serial connection

to the evaluation board. This is done by identifying the USB COM port driver and which COM port is being assigned.

The MPSoC boards support three USB ports. Two are connected to the UARTs in the PS and one is available to the PL, assuming that there is a UART implemented in the PL.

### **1-1. Determine which COM port is connected to the USB serial port from the board.**

- 1-1-1.** Make sure that the development board is powered on and the serial UART device USB cable is in place.

This ensures that the USB-to-serial bridge will be enumerated by the PC host.

The Device Manager in Windows lists all of the hardware items in the system. While there are a number of routes to opening the device manager, the easiest is as follows.

- 1-1-2.** Click **Start** (1).

- 1-1-3.** Enter the following into the search bar (2):

**device manager**



**Figure 338: Accessing the Windows Application Search**

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

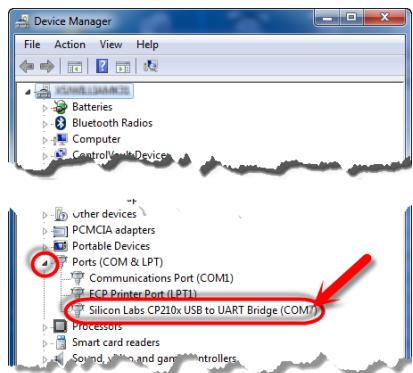
- 1-1-4.** Expand **Ports (COM & LPT)** so that you can see all of the enumerated ports.

Various boards use different USB bridge chips. The name of the driver varies correspondingly.

- **ZCU102/4 users:** Locate **Silicon Labs Quad CP210x USB to UART Bridge: Interface 0 (COM#)**.
- Note that there will be a group of (at least) four COM ports listed. This is because this board has a quad USB to UART bridge. Interface 0 and 1 are attached to 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 339: Locating the Silicon Labs Com Port Driver**

- 1-1-5.** Note the COM# corresponding to the USB bridge chip.

This is the port number you will use when connecting a terminal to the board.

- 1-1-6.** Close the Device Manager by clicking the red 'X' in the upper-right corner of the panel.

## Linux Operations

### Opening a Terminal – Linux Ubuntu

Sometimes it is easier to enter operating system commands via a Linux terminal rather than a GUI. A terminal can completely configure and control a Linux system without the need for a graphical desktop.

Even when there is a graphical desktop available for use, it may sometimes be necessary or more convenient to use the terminal window to launch programs and configure settings.

#### 1-1. Open a Linux terminal window.

- 1-1-1.** Click the Linux terminal window icon in the taskbar.



**Figure 340: Opening the Linux Terminal Window from the Taskbar**

Alternatively, you can press **<Ctrl + Alt + T>**.

The terminal window will open, leaving you in your home directory.

## Shutting Down Ubuntu Linux

---

### 1-1. [Windows users]: Shut down Ubuntu Linux.

1-1-1. Click the **System Settings** icon in the upper-right corner of the Ubuntu desktop (1).

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

The Shut Down dialog box opens.

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

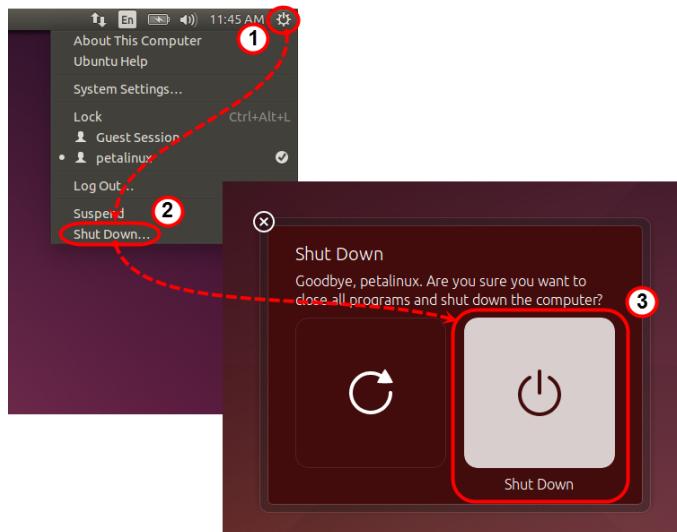


Figure 341: Shutting Down

## Running a Virtual Machine Using VirtualBox

For CloudShare users, you can skip the launching VirtualBox instructions below and proceed with the launching the Vivado Design Suite instructions.

### 1-1. Launch the VirtualBox Manager application.

**Running Linux under Windows is easily achieved using a virtual machine that forms an isolated container for another OS. The Oracle VirtualBox application was selected as the framework for hosting virtual machines as it works well and is available without the hassles of licensing. The Customer Training team provides a VM that is properly configured to run the labs and demos.**

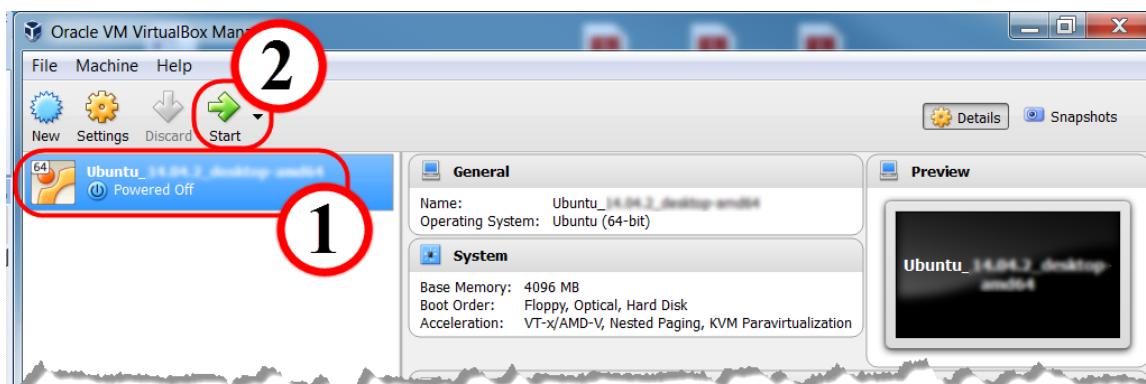
- 1-1-1. Select **Start Menu > Oracle VM VirtualBox > Oracle VM VirtualBox** to launch the VirtualBox Manager program.

Alternatively, if a desktop or taskbar icon is available, you can simply click it.

When the VirtualBox Manager opens, the left-hand pane will show all of the available virtual machines.

- 1-1-2. Double-click the virtual machine (1).

Alternatively, you can single-click the virtual machine of choice, then click the green right arrow icon to launch the selected virtual machine (2).



**Figure 342: VirtualBox Manager Window**

The virtual machine will now launch in a new window. It takes a few moments to boot. If any warning messages appear, such as anything regarding the mouse or keyboard, click the "x" on the line with the error and continue.

- 1-1-3. Click the **Maximize** (□) icon in the window title bar to expand the screen to its full size, if not already maximized.

## Cleaning Up the VirtualBox File System

---

Some systems (particularly VMs) may be memory constrained. Removing the workspace frees a portion of the disk space, allowing other labs to be performed.

You can delete the directory containing the lab you just ran by using the graphical interface or the command-line interface. You can choose either mechanism. Both processes will recursively delete all the files in the \$TRAINING\_PATH/<the topic cluster name> directory.

### 1-1. [Optional] [Only for local VMs—not for CloudShare] Clean up the file system.

Using the GUI:

1-1-1. Navigate to \$TRAINING\_PATH/<the topic cluster name>.

1-1-2. Select <the topic cluster name>.

1-1-3. Press <Delete>.

-- OR --

[Linux users]: Using the command line:

1-1-4. Press <Ctrl + Alt + T> to open a terminal window.

1-1-5. Enter the following command to delete the contents of the workspace:

```
[host]$ rm -rf $TRAINING_PATH/<the topic cluster name>
```

## Setting the Static Host IP Address Using Windows 10

These instructions illustrate how to change to or set up a *static* IP address on the host computer. Follow the reverse of these instructions to change back to a DHCP-received address.

For labs that require an Ethernet connection between the 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 (host computer) with its own *static* IP address. After you are finished with this lab, you can revert this step and set TCP/IP properties to obtain an IP address automatically.

Most of our software applications use an IP address of 192.168.1.11. The host computer is required to have an IP address of 192.168.1.**num**, where **num** is any number between 2 and 255 except 11 (which is reserved for the hardware).

These instructions describe how to perform this task on a Windows 10 machine.

### 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. Right-click the networking icon on the bottom-right corner of the taskbar (1).

- 1-1-3. Select **Open Network and Sharing Center** (2).

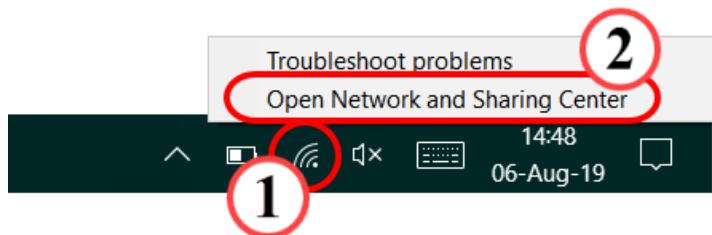


Figure 343: Selecting Open Network and Sharing Center

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

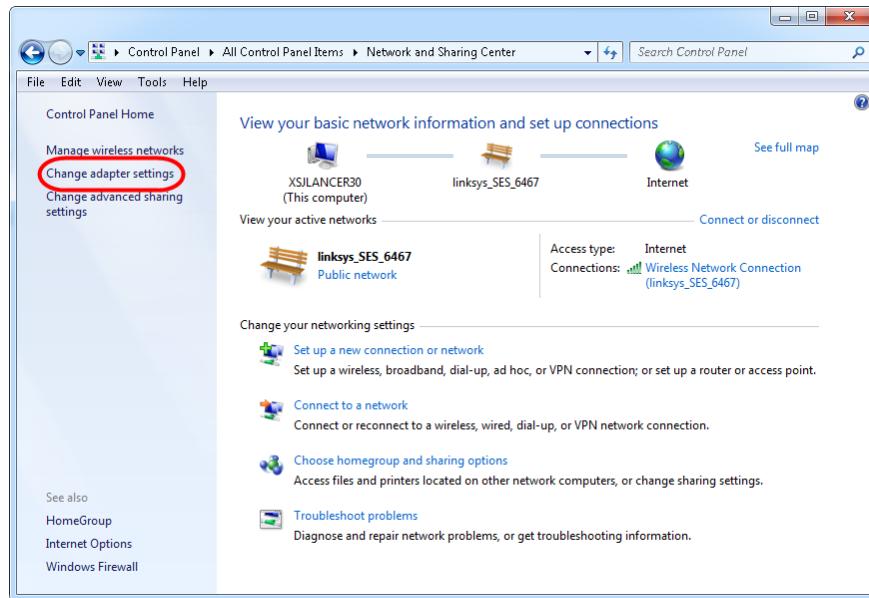


Figure 344: Control Panel Network and Sharing Center

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

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

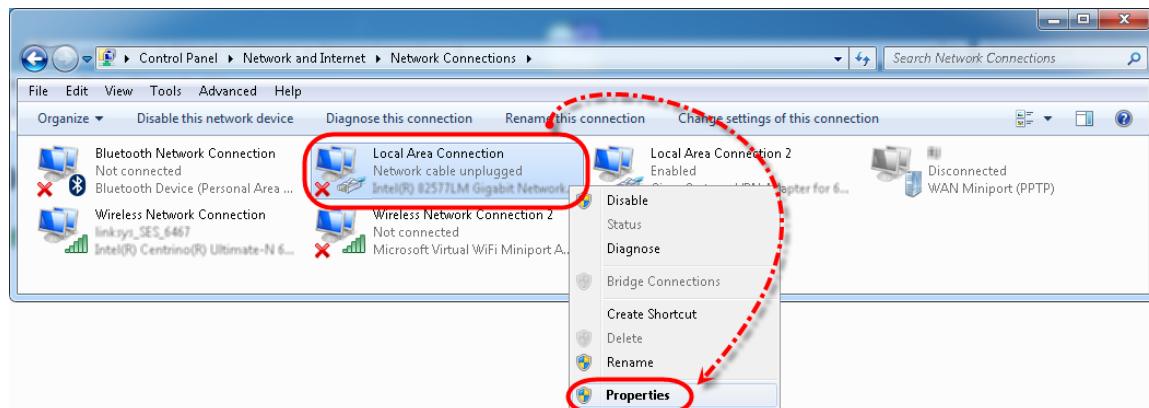


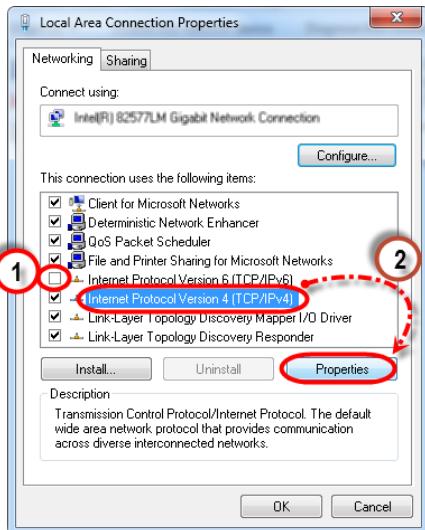
Figure 345: Network Connections

- 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)** (2).

**1-1-10.** Click **Properties** (2).



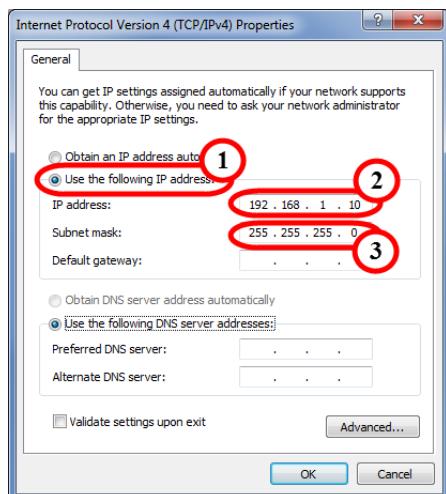
**Figure 346:** Local Area Connection Properties

**1-1-11.** Select **Use the following IP address** (1).

**1-1-12.** Enter **192.168.1.10** in the IP address field (2).

This value is fairly arbitrary, but it *cannot* be the same as the IP address designated for the board.

The Subnet mask field should fill in with **255.255.255.0** automatically after leaving the IP address field (3).



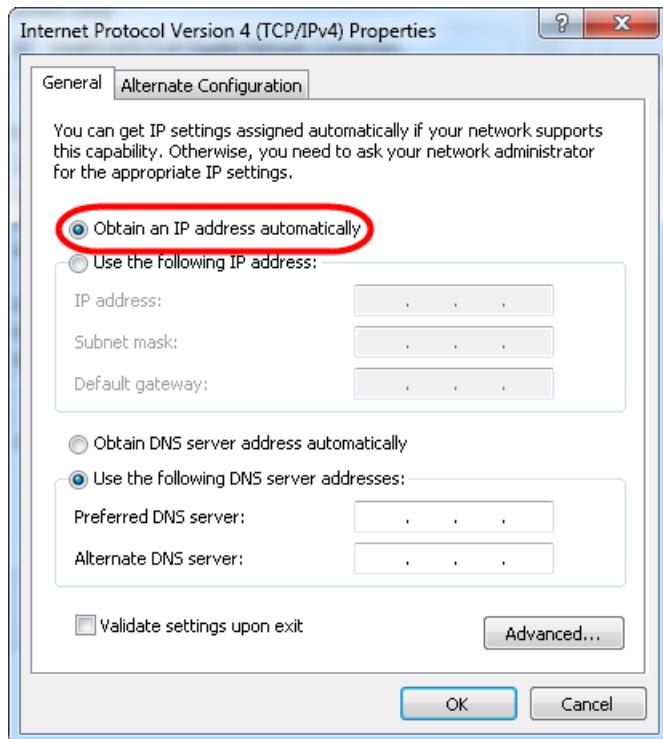
**Figure 347:** Set TCP-IPv4 Properties for Static Address

**1-1-13.** Click **OK**.

**1-1-14.** Click **Close** to close the Control Panel.

**1-2. To return to a DHCP-obtained address, follow the same steps to revert the settings.**

- 1-2-1.** Select the **Internet Protocol Version 6 (TCP/IPv6)** option.
- 1-2-2.** Select **Internet Protocol Version 4 (TCP/IPv4)**
- 1-2-3.** Click **Properties**.
- 1-2-4.** Select **Obtain an IP address automatically**.



**Figure 348: Set TCP-IPv4 Properties for DHCP Obtained Address**

- 1-2-5.** Click **OK**.
- 1-2-6.** Click **Close**.

## Setting the Static Host IP Address Using Linux

These instructions illustrate how to change to or set up a *static* IP address on the host computer. Follow the reverse of these instructions to change back to a DHCP-received address.

For labs that require an Ethernet connection between the laptop host computer, the hardwired Ethernet settings on your host machine must be set up properly to work with the hardware evaluation board. Typically, the PC's Ethernet port is set to automatically request an IP address from the network DHCP client server. Because the hardwire Ethernet connection will attach directly to the hardware evaluation board, there will not be a DHCP server to supply the needed IP address for the host computer. It is therefore necessary to reconfigure the TCP/IP client (laptop) with its own *static* IP address. After you are finished with this lab, you can revert this step and set TCP/IP properties to obtain an IP address automatically.

Most AMD software applications use an IP address of 192.168.1.11. The host computer is required to have an IP address of 192.168.1.**num**, where **num** is any number between 2 and 255 except 11 (which is reserved for the AMD hardware).

These instructions describe how to perform this task on a Linux machine.

### 1-1. Set the static host IP address.

- 1-1-1. Press **<Ctrl + Alt + T>** to open a new terminal.
- 1-1-2. Enter the following command to set the static IP:

```
[host] $ ifconfig eth0 192.168.1.10
```

This value is fairly arbitrary, but it *cannot* be the same as the IP address designated for the board.

- 1-1-3. Exit the terminal:

```
[host] $ exit
```

### 1-2. To return to a DHCP-obtained address, follow the same steps to revert the settings.

- 1-2-1. Reboot the Linux machine or VM to revert the static IP settings that you set in the previous step.

## Verifying Board Jumper/Switch Settings Configured to Boot from the SD Card

---

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

### 1-1. Verify that your board hardware platform is properly connected and the jumpers (or switches) are configured to boot from the SD card.

- 1-1-1. Confirm that the power to the evaluation board is OFF.
- 1-1-2. Ensure that the connections between the host PC and the hardware evaluation board are correct (ask for assistance from the instructor if necessary).  
The connections include power, serial bridge USB, Ethernet, and 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 - <Name of 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) your board hardware platform.

- 1-1-1. Make sure that AC power is connected to the power brick.
- 1-1-2. Ensure that the power brick is connected to the board.
- 1-1-3. Confirm the serial port USB connection between the board and the PC.
- 1-1-4. Confirm the USB-JTAG connection between the board and the PC.
- 1-1-5. Slide the power switch to the on position.

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

## Preparing an SD Card

### 1-1. Prepare an SD card.

- 1-1-1. Insert an SD card into the PC's SD card slot.
- 1-1-2. [Linux users]: Make sure that you copy the SD card files from Linux to the *C:\training* directory by entering the following command:

```
[host] $ cp -rf <path_to_sd_card_files> /media/sf_training
```

- 1-1-3. Browse to the SD card drive using Windows Explorer.

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

- 1-1-4. Open a second Windows Explorer window to browse to *the location of the files you want to copy to the SD card*, which will be copied to the SD card.

Be aware that many SD card images are delivered as compressed files. You will need to ensure that these files are properly decompressed unless otherwise indicated.

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

This will automatically copy the files onto the SD card.

**Note:** The files should go into the root of the SD card unless there are specific instructions to the contrary.

- 1-1-6. Close both Windows Explorer windows.

- 1-1-7. Remove the SD card from the PC card slot.

- 1-1-8. Turn off power to the hardware platform.

**Note:** The boot selection switches or jumpers must be properly set to boot from the SD card. See the appropriate section in the *Lab Setup Guide*.

- 1-1-9. Insert the SD card into its slot on the hardware platform.

## Booting Linux

---

You must have an SD card with a properly constructed Linux boot image. This can be open-source Linux, Petalinux, or a third-party image.

### 1-1. Set the jumpers/switches on your board to boot from the SD card.

If you do not recall how to perform this task, refer to the "Setting the Jumpers on the <Name of Board> Board" section under Board, OS, COM, and IP Address Tasks 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.

## Launching and Configuring the Terminal Emulator

---

Maintaining consistency in the instructions is an important aspect of the lab experience. The labs will use general and vague commands such as "Open the terminal emulator". Depending on the system that you are using, you must know which terminal emulator to configure and launch. Here are the instructions for opening a terminal emulator for both the Linux and Windows environments.

Generally, the software labs will use the serial terminal built into the Vitis platform; however, there are some labs that will require you to communicate with the board without using the Vitis environment. These situations require the use of a third-party serial port emulator.

We have tested Tera Term for the Windows environment and GTKTerm for Linux. The GTKTerm tool is pre-installed with the Customer Training VM. Both terminal emulators listed here run independently of the tools. If you have another terminal emulator that you prefer, you can certainly use it; however, you are responsible for figuring out how to configure it.

## Linux

GTKTerm is a simple GTK+ terminal used to communicate with the serial port. Other terminal emulators may be used; however, the installation and configuration instruction provided here are for GTKTerm.

GTKTerm is already installed in the VM provided by the Customer Training team; however native Linux users may need to install GTKTerm.

### 1-1. [Native Linux users] Acquire and install the GTKTerm software from the command line.

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

1-1-2. Download and install GTKTerm:

```
[host] $ sudo apt install gtkterm
```

1-1-3. When prompted for the password, enter the super user password.

The tools will install in about a minute or less.

### 1-2. Launch GtkTerm and set the port configuration.

1-2-1. Click the **GtkTerm** icon from the quick launch toolbar.

Alternatively, GtkTerm can be launched from a Linux terminal window (**<Ctrl + Alt + T>**) and entering:

```
[host] $ sudo gtkterm
```

**Note:** While the application will run as a regular user, you must be a super user to access the ports.

When the GtkTerm window opens, perform the following.

1-2-2. Select **Configuration > Port** to open the Configuration dialog box.

1-2-3. Identify the port associated with your board and set the port as **/dev/ttyUSBx** (where x could be 0, 1, 2, 3, etc.)

1-2-4. Set the baud rate to **115200**.

1-2-5. Leave the rest of the settings at their default.

**Note:** You can open multiple instances of **/dev/USBx** if you are unable to determine which port your UART is connected to.

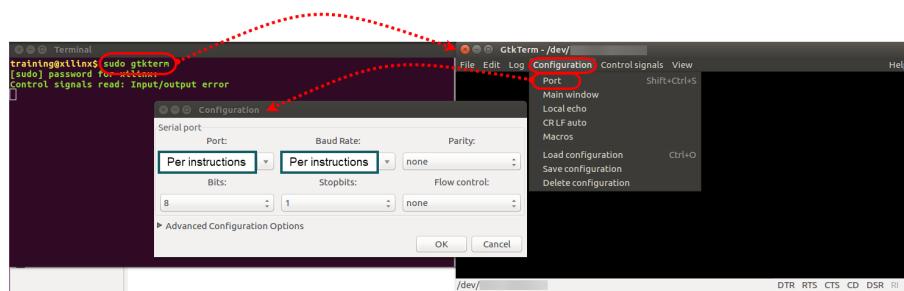


Figure 349: Opening GtkTerm and Selecting the Port Configuration

[Optional]: You can save these settings so that you do not have to reconfigure GtkTerm each time you open it by selecting **Configuration > Save configuration**.

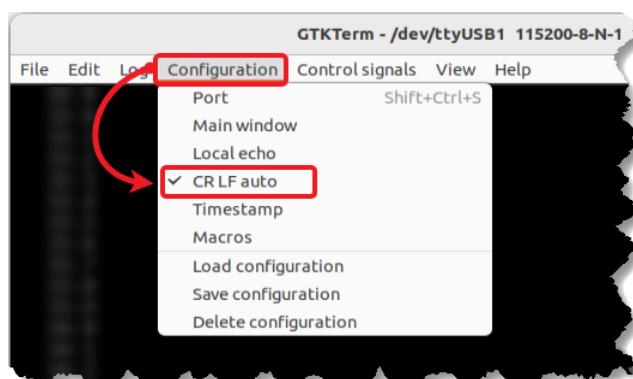
If you save the configuration as "default", this configuration will open when GtkTerm starts. Otherwise, you can save the configuration with another name; however, you will then need to load the configuration each time you start GtkTerm.

- 1-2-6. Click **OK** to save the settings and leave the terminal open.

### 1-3. Enable auto CR/LF mode.

**GtkTerm is a terminal emulator that supports automatic translation of CR (carriage return) characters to LF (line feed) characters and vice versa.**

- 1-3-1. Select **Configuration > CR LF auto** to enable auto CR/LF mode.



- 1-3-2.

Figure 350: Enabling CR/LF auto in GTKTerm

## Windows

Tera Term is a popular public domain terminal emulation program. It is capable of operating as a serial port terminal or as a telnet client.

### 1-4. Download and install Tera Term.

- 1-4-1. Acquire and install the Tera Term software from any legitimate site. Recommended sites:

- [ttssh2.osdn.jp/index.html.en](http://ttssh2.osdn.jp/index.html.en) (Tera Term home page with documentation)
- [osdn.net/projects/ttssh2](http://osdn.net/projects/ttssh2)
- [en.sourceforge.jp/projects/ttssh2](http://en.sourceforge.jp/projects/ttssh2)
- [logmatt.com](http://logmatt.com)

- 1-4-2. Install per instructions.

There are two types of downloads: a traditional zip install, and a self-installing version, which is recommended.

[Optional]

Certain drivers like installing their com port numbers using high numbered serial ports. Tera Term does not accept these port numbers by default, so you will need to override the Tera Term settings:

- 1-4-3. Open **TERATERM.INI** (found in the install path for Tera Term) with an ASCII text editor.
- 1-4-4. Set to MaxComPort=256.
- 1-4-5. Save and close the INI file.
- 1-4-6. Use the **Setup > Save Setup** option to save the setup.

## 1-5. Launch the Tera Term terminal program.

- 1-5-1. Double-click the **Tera Term** icon on the Windows desktop to launch Tera Term.

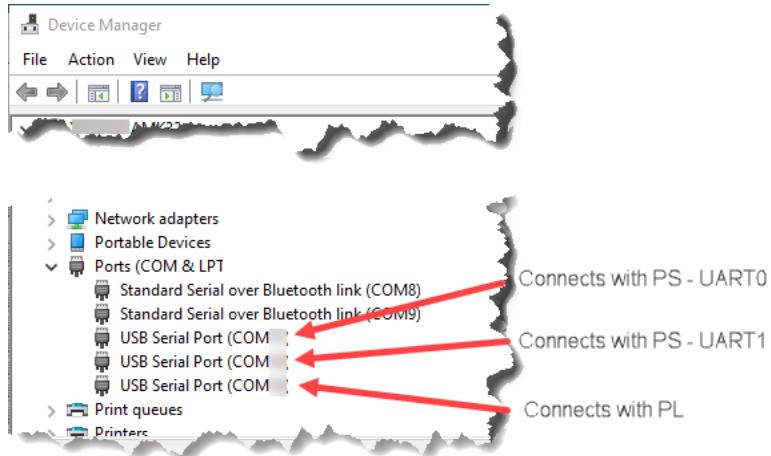
Alternatively, you can select **Start > All Programs > Tera Term > Tera Term**.

- 1-5-2. Select **Serial** as the connection (1).
- 1-5-3. Click the **Port** drop-down list to view the available COM ports (2).

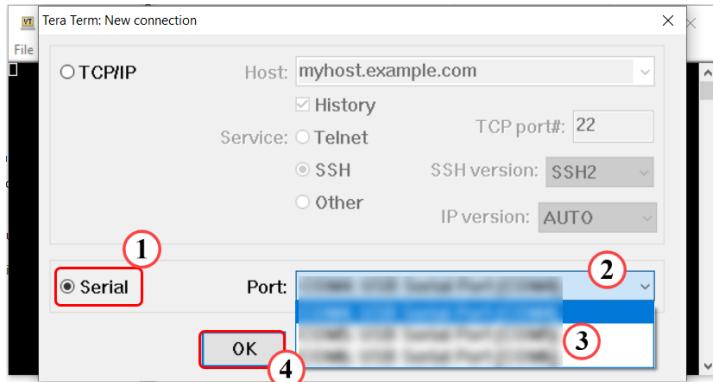
**Note:** If your port is not listed, exit Tera Term, power cycle your board and restart this step.

- 1-5-4. Select the COM # (3).

**Hint:** MPSOC and RFSoC devices display three COM ports. Their specific numbers may vary based on the USB enumeration process; however, the first two COM ports connect to the UARTs in the PS and the third connects to PL pins.



**Figure 351: Locating and Identifying COM Ports from the Windows Device Manager**



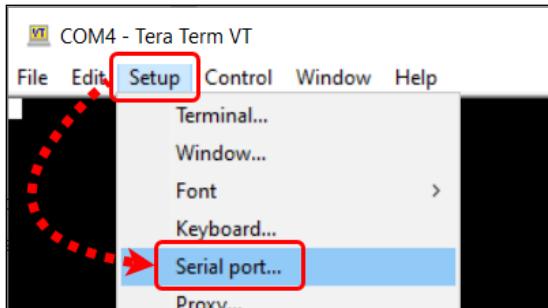
**Figure 352: Selecting the COM Port**

**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.

- 1-5-5. Click **OK** (4).

The terminal console window opens.

- 1-5-6. Select **Setup > Serial Port**.

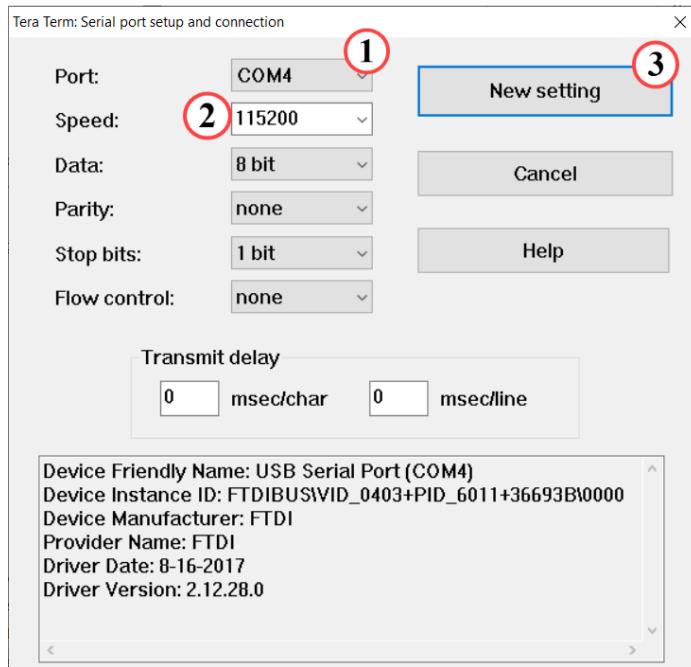


**Figure 353: Opening the Tera Term Serial Port Setup Window**

The Tera Term Serial Port Setup dialog box opens.

1-5-7. Confirm that the proper serial port has been selected (1).

1-5-8. Set the baud rate to **115200** (2).



**Figure 354: 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.

1-5-9. Click **OK** (3).

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

## 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 355: SD Card Boot Settings (ZC702 Board)

## Jumper/Switch Settings ZC702 - Boot from JTAG

JTAG Mode:

|     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |
|     |     |     |     |     |     |     |
| J27 | J28 | J21 | J20 | J22 | J25 | J26 |

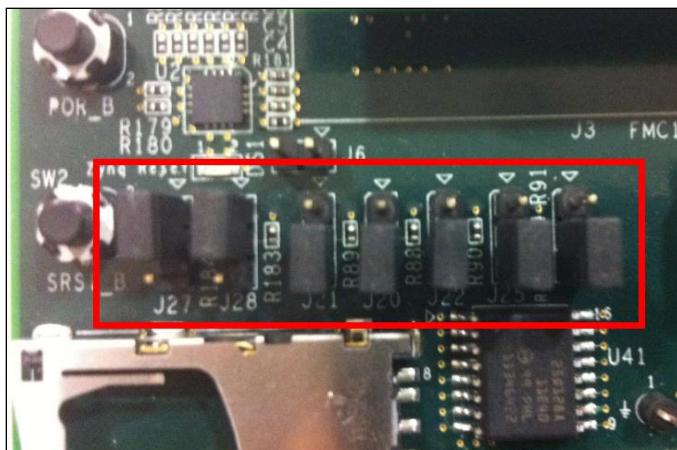


Figure 356: JTAG Boot Settings (ZC702 Board)

It is worth noting that if the SD card settings are used, but no SD card is present in the socket, then the boot mode will fall back to JTAG.

The ZC702 board has two JTAG cable options (most ATPs use option 1):

1. Digilent on-board platform cable USB interface using USB A-mini B cable: Set SW10 ( i.e. JTAG Select dip switches) on Zynq board to "01"
2. Platform USB: Set SW10 ( i.e. JTAG Select dip switches) on Zynq board to "10".

## Jumper/Switch Settings ZC702 - Boot from QSPI

Flash Boot:

|     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |
|     |     |     |     |     |     |     |
| J27 | J28 | J21 | J20 | J22 | J25 | J26 |

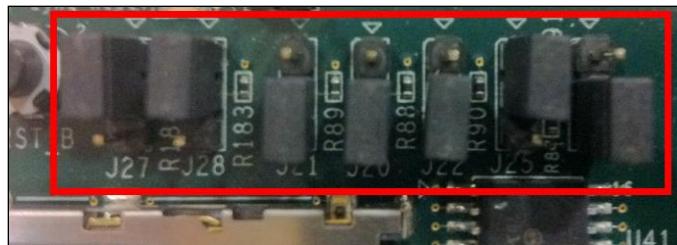


Figure 357: Flash Boot Settings (ZC702 Board)

## Setting the Jumpers on the ZC706 Board

### 1-1. Set the jumpers on the ZC706 board based on your booting requirements.

- Boot from SD card
- JTAG communications with PC (no boot)
- Boot from QSPI

The three following topics cover these cases.

### Jumper/Switch Settings ZC706 - Boot from SD Card

#### SD Card Boot:

|        |        |        |        |        |
|--------|--------|--------|--------|--------|
|        |        |        |        |        |
|        |        |        |        |        |
| SW11.1 | SW11.2 | SW11.3 | SW11.4 | SW11.5 |

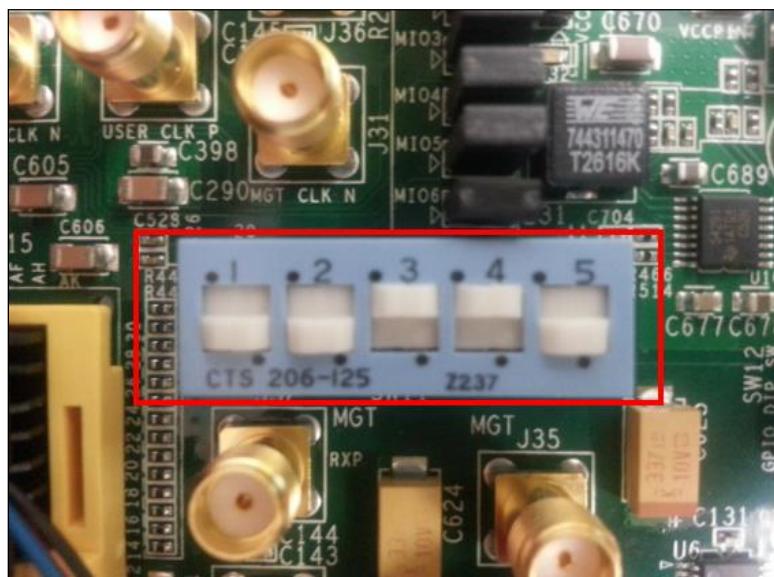


Figure 358: SD Card Boot Settings (ZC706 Board)

## Jumper/Switch Settings ZC706 - Boot from JTAG

### JTAG Mode:

|        |        |        |        |        |
|--------|--------|--------|--------|--------|
|        |        |        |        |        |
|        |        |        |        |        |
| SW11.1 | SW11.2 | SW11.3 | SW11.4 | SW11.5 |

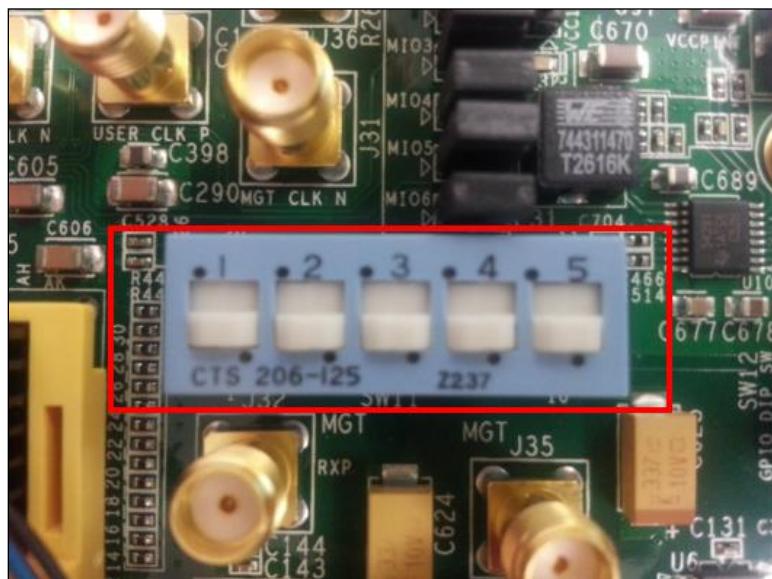


Figure 359: JTAG Boot Settings (ZC706 Board)

It is worth noting that if the SD card settings are used, but no SD card is present in the socket, then the boot mode will fall back to JTAG.

The ZC706 board has a Digilent interface option, and you can use a USB A-mini B cable.

To use this cable instead of Platform USB, make sure that the jumper settings are in JTAG mode as mentioned above.

- Set SW4 (i.e., the JTAG select dip switches) on the board to "01", which is to select the Digilent interface. To select Platform USB, SW4 should be "10".

## Jumper/Switch Settings ZC706 - Boot from QSPI

Flash Boot:

|        |        |        |        |        |
|--------|--------|--------|--------|--------|
|        |        |        |        |        |
|        |        |        |        |        |
| SW11.1 | SW11.2 | SW11.3 | SW11.4 | SW11.5 |

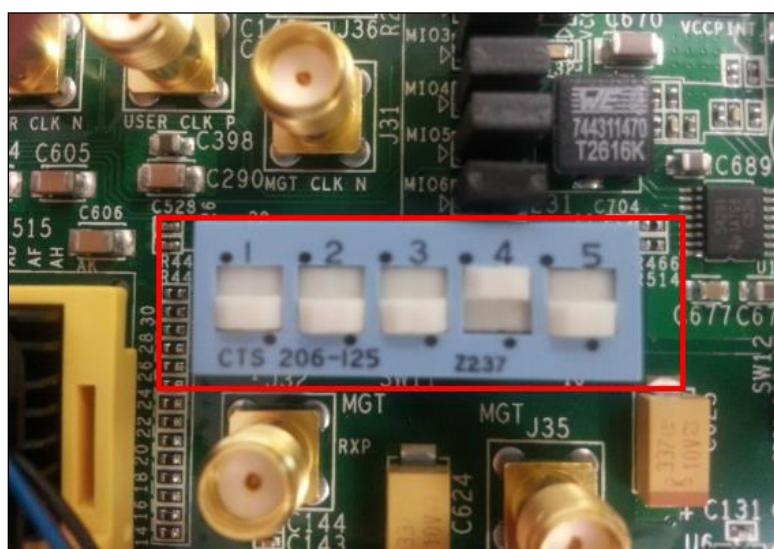


Figure 360: Flash Boot Settings (ZC706 Board)

## Setting the Jumpers on the ZCU102 Board

### 1-1. Set the jumpers on the ZCU102 board based on your booting requirements.

- Boot from SD card
- JTAG communications with PC (no boot)

The following topics cover these cases.

### Jumper/Switch Settings ZCU102 - Boot from SD Card

SD Card Mode:

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



Figure 361: SD Card Boot Settings (ZCU102 Board)

## Jumper/Switch Settings ZCU102 - Boot from JTAG

JTAG Mode:

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

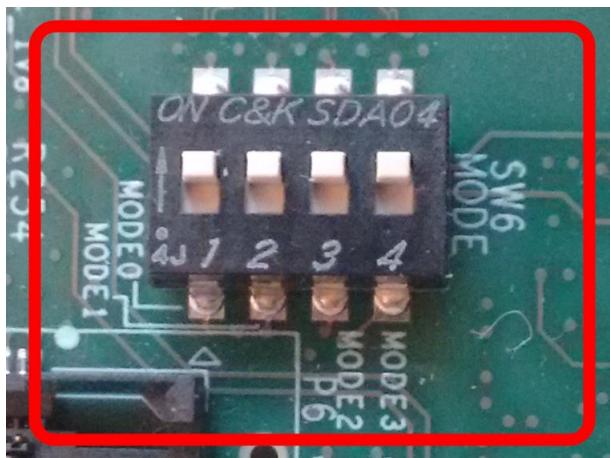


Figure 362: JTAG Boot Settings (ZCU102 Board)

## Setting the Jumpers on the ZCU104 Board

### 1-1. Set the jumpers on the ZCU104 board based on your booting requirements.

- Boot from SD card
- JTAG communications with PC (no boot)

The following topics cover these cases.

### Jumper/Switch Settings ZCU104 - Boot from SD Card

#### SD Card Mode:

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

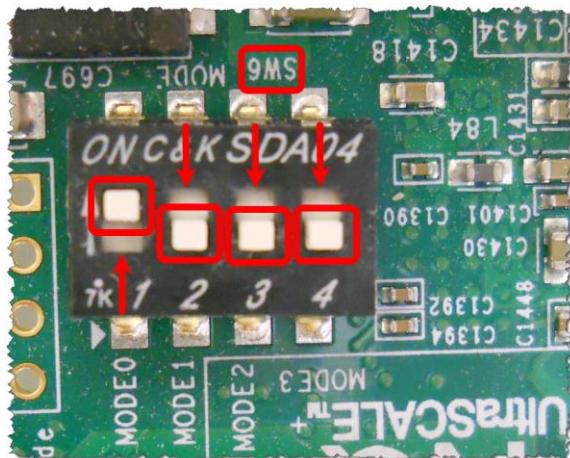


Figure 363: SD Card Boot Settings

## Jumper/Switch Settings ZCU104 - Boot from JTAG

**JTAG Mode:**

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

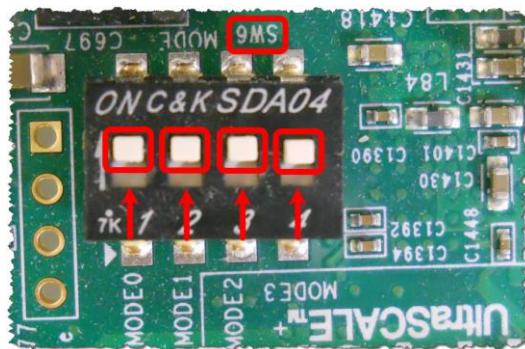


Figure 364: JTAG Boot Settings

## Jumper/Switch Settings ZCU104 - Boot from QSPI

**Flash Boot:**

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

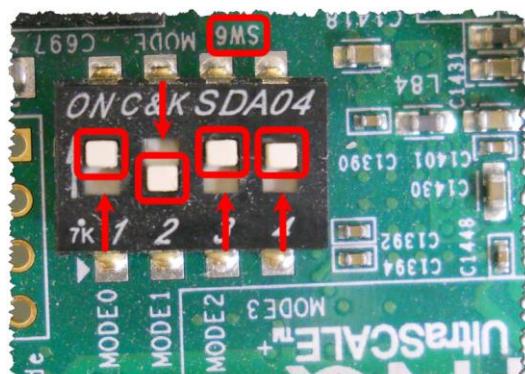


Figure 365: QSPI Boot Settings

## Setting the Jumpers on the ZedBoard

### 1-1. Set the jumpers on the ZedBoard based on your booting requirements.

- Boot from SD card
- JTAG communications with PC (no boot)
- Boot from QSPI

### Jumper Settings ZedBoard - Boot from SD Card

**SD Card Boot:**

| JP8/JP11 - Mode 4 | JP7/JP10 - Mode 3 | JP6/JP9 - Mode 2 | JP5/JP8 - Mode 1 | JP4/JP7 - Mode 0 |         |
|-------------------|-------------------|------------------|------------------|------------------|---------|
|                   |                   |                  |                  |                  | ON/3.3V |
|                   |                   |                  |                  |                  | Off/Gnd |

**Jumper - JP6 (shortened)**

**VADJ SELECT - 3V3**



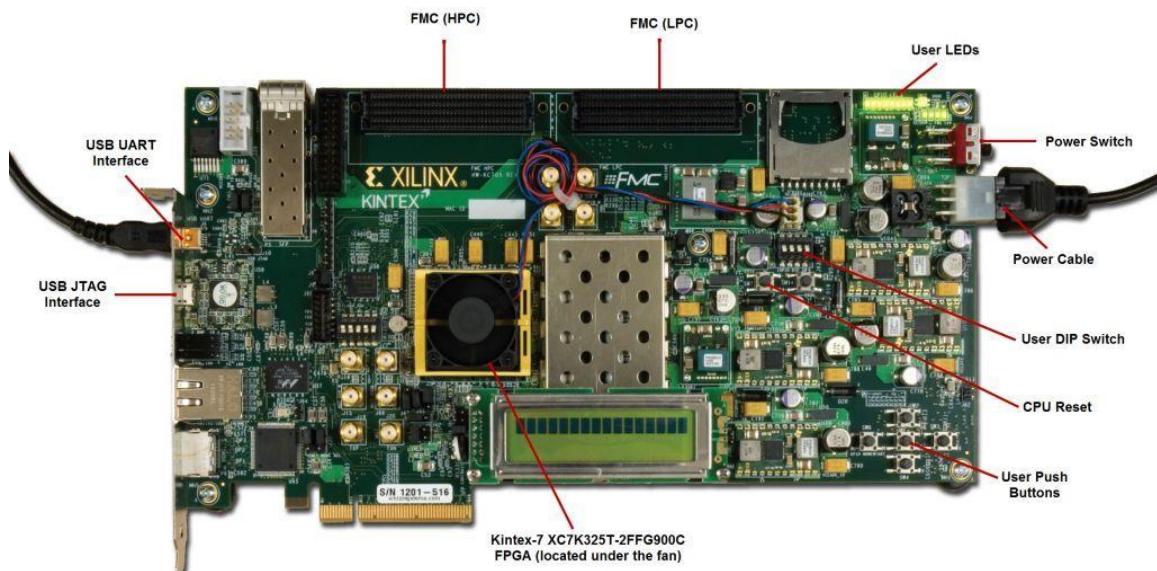
Figure 366: SD Card Boot Settings (ZedBoard)

## Setting Up the Hardware - KC705

Set up and connect the hardware evaluation board or verify that this has properly been done before turning on the power.

### 1-1. Connect the board to your machine as shown below.

- 1-1-1. Connect a USB cable from a USB port on your computer to the **USB UART** connector on the evaluation board.
- 1-1-2. Similarly, connect the USB cable to the **Digilent USB JTAG** interface.
- 1-1-3. Ensure that the power cord is plugged in and turn on the evaluation board.
- 1-1-4. Make sure that the board settings are proper.
- 1-1-5. Ensure that all DIP switches (SW11) are in the off position.



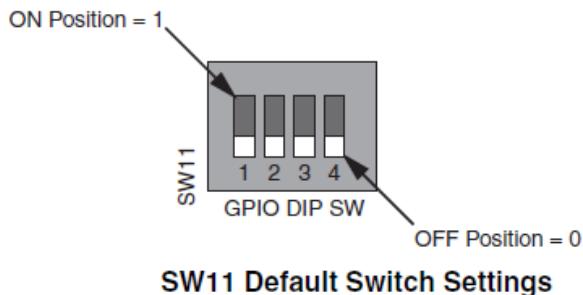
**Figure 367: KC705 Evaluation Board**

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.

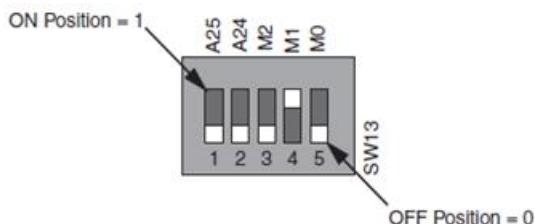


| Position | Function     | Default |
|----------|--------------|---------|
| 1        | GPIO_DIP_SW3 | Off     |
| 2        | GPIO_DIP_SW2 | Off     |
| 3        | GPIO_DIP_SW1 | Off     |
| 4        | GPIO_DIP_SW0 | Off     |

Figure 368: 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.



| Position | Function  | Default |
|----------|-----------|---------|
| 1        | FLASH_A25 | A25     |
| 2        | FLASH_A24 | A24     |
| 3        | FPGA_M2   | M0      |
| 4        | FPGA_M1   | M1      |
| 5        | FPGA_M0   | M3      |

Figure 369: KC705: SW13 Default Switch Settings



**AMD**  
together we advance\_