Vivado Design Suite User Guide

System-Level Design Entry

UG895 (v2018.3) December 5, 2018

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
12/05/2018 Version 2018.3	
Using the Project Summary	Added Overview and Dashboard tabs.
06/06/2018 Version 2018.2	
General updates	Editorial updates only. No technical content updates.
04/04/2018 Version 2018.1	
General updates	Updated menu commands.
Working with Vivado HLS Sources	Added new section.
Working with Model Composer Sources	Added new section.
Reporting DRCs	Added information on the interactive report file and DRC waivers.

Table of Contents

Chapter 5: Debugging the Design

Appendix A: Board File

Appendix B: Vivado Naming Conventions

Appendix C: Additional Resources and Legal Notices

Introduction

Overview

This user guide provides an overview of the Vivado® Design Suite with an emphasis on the different project types, using the tool through the GUI and Tcl, with a project and without. The Vivado® Design Suite enables you to take your design from full register-transfer level (RTL) creation to bitstream generation. System-level design entry consists of setting up your design, including creating a project (if applicable), creating and adding source files, adding block design and IP cores, elaborating the RTL design, and inserting and configuring debug information. You can enter your design using the graphical user interface (GUI), known as the Vivado Integrated Design Environment (IDE), or using Tcl commands and scripts.

Launching the Vivado Design Suite in Project and Non-Project Mode

You can launch the Vivado Design Suite and run the tools using different methods depending on your preference. For example, you can choose a Tcl script-based compilation style method in which you manage sources and the design process yourself, also known as *Non-Project Mode*. Alternatively, you can use a project-based method to automatically manage your design process and design data using projects and project states, also known as *Project Mode*. Either of these methods can be run using a Tcl scripted batch mode or run interactively in the Vivado IDE. For more information on the different design flow modes, see this link in the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) [Ref 1].

Working with Tcl

If you prefer to work directly with Tcl, you can interact with your design using Tcl commands using either of the following methods:

- Enter individual Tcl commands in the Vivado Design Suite Tcl shell outside of the Vivado IDE.
- Enter individual Tcl commands in the Tcl Console at the bottom of the Vivado IDE.

- Run Tcl scripts from the Vivado Design Suite Tcl shell.
- Run Tcl scripts from the Vivado IDE.

For more information about using Tcl and Tcl scripting, see the *Vivado Design Suite User Guide: Using Tcl Scripting* (UG894) [Ref 3], *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4], and *Vivado Design Suite User Guide: Design Flows Overview* (UG892) [Ref 1]. For a step-by-step tutorial that shows how to use Tcl in the Vivado tools, see the *Vivado Design Suite Tutorial: Design Flows Overview* (UG888) [Ref 5].

Launching the Vivado Design Suite Tcl Shell

Use the following command to invoke the Vivado Design Suite Tcl shell either at the Linux command prompt or within a Windows Command Prompt window:

```
vivado -mode tcl
```

Note: On Windows, you can also select **Start > All Programs > Xilinx Design Tools > Vivado <version> > Vivado <version> Tcl Shell**.

Launching the Vivado Tools Using a Batch Tcl Script

You can use the Vivado tools in batch mode by supplying a Tcl script when invoking the tool. Use the following command either at the Linux command prompt or within a Windows Command Prompt window:

```
vivado -mode batch -source <your_Tcl_script>
```

Note: When working in batch mode, the Vivado tools exit after running the specified script.

Working with the Vivado IDE

If you prefer to work in a GUI, you can launch the Vivado IDE from Windows or Linux. For more information on the Vivado IDE, see the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 6] and *Vivado Design Suite User Guide: Design Flows Overview* (UG892) [Ref 1].

RECOMMENDED: *Launch the Vivado IDE from the directory containing your project, or your working directory. This makes it easier to locate the project file, log files, and journal files, which are written to the launch directory.*

TIP: *For quick access to information on different parts of the Vivado IDE, click the Quick Help button ? in the window or dialog box.*

Launching the Vivado IDE on Windows

Select **Start > All Programs > Xilinx Design Tools > Vivado <version> > Vivado <version>**.

Note: You can also double-click the Vivado IDE shortcut icon on your desktop.

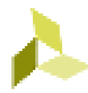

Figure 1-1: **Vivado Desktop Icon**

TIP: *You can right-click the Vivado IDE shortcut icon on the Microsoft Windows desktop, and select* ***Properties*** *to update the Start In field. This makes it easier to locate the project file, log files, and journal files, which are written to the launch directory. See Output Files in the Vivado Design Suite User Guide: Using the Vivado IDE (UG893) [Ref 6] for information on the default location of the log and journal files.*

Launching the Vivado IDE from the Command Line on Windows or Linux

To launch the Vivado Design Suite from the Linux or Windows command line, you must install and configure the tool to run on the local machine. Installation adds the Vivado tools to the PATH.

When the tool is installed, enter the following at the command prompt:

```
vivado
```

When you enter this command, it automatically runs `vivado -mode gui` to launch the Vivado IDE. If you need help, type `vivado -help`.

TIP: *To add the Vivado tools path to your current shell/command prompt, run settings64.bat or settings64.sh from the <install_path>/Vivado/<version> directory.*

Launching the Vivado IDE from the Vivado Design Suite Tcl Shell

When the Vivado Design Suite is running in an interactive Tcl command shell in Tcl mode, you can use the following command at the Tcl prompt to launch the Vivado IDE and open the Vivado tool GUI:

```
start_gui
```

Working with Projects

Overview

When working in Project Mode, you can enter your design using various project types. This chapter describes each project type and explains how to create and manage projects. It also covers the Project Summary, Project Settings, and how to create a project using a Tcl script.

Project Types

Using the Vivado® Design Suite, you can create the following types of projects. Each project type includes different input source types.

- RTL Projects
- Post-Synthesis Projects
- I/O Planning Projects
- Imported Projects

Note: A project *cannot* be changed to a different project type after it is created. The only exception is the I/O planning project, which can be used as the basis for an RTL project.

RTL Projects

You can use the Vivado Design Suite to manage the entire design flow from RTL creation through bitstream generation. You can add RTL source files, IP from the Xilinx IP catalog, block designs created in the Vivado IP integrator, digital signal processing (DSP) sources, and EDIF netlists for hierarchical modules. IP can include XCI or XCIX files generated by the Vivado tools, legacy XCO files generated by the CORE Generator™ tool, and precompiled EDIF or NGC-format netlists. For more detailed RTL information see the Chapter 4, Elaborating the RTL Design.

Note: ISE® IP is only supported for 7 series devices. ISE format IP (.ngc) are no longer supported with UltraScale™ devices. Users should migrate their IP to the native Vivado Design Suite format prior to beginning UltraScale device designs.

From an RTL project, you can elaborate and analyze the RTL to ensure proper syntax and design constructs, launch and manage various synthesis and implementation runs, and analyze the design and run results. You can also experiment with different constraints or implementation strategies to achieve timing closure.

Post-Synthesis Projects

You can create projects using synthesized netlists created using Vivado synthesis, XST, or any supported third-party synthesis tool. For example, the Vivado Design Suite can import EDIF, NGC, or structural Verilog format netlists, XCI files (all output products including the DCP must be already generated), as well as Vivado design checkpoint (DCP) files. The netlist can be made up of a single file that is all-inclusive or a set of files that are hierarchical and consist of multiple, module-level netlists.

IMPORTANT: *NGC format files are not supported in the Vivado Design Suite for UltraScale devices. It is recommended that you regenerate the IP using the Vivado Design Suite IP customization tools with native output products. Alternatively, you can use the NGC2EDIF command to migrate the NGC file to EDIF format for importing. However, Xilinx recommends using native Vivado IP rather than XST-generated NGC format files going forward.*

You can analyze and simulate the netlist logic, launch and manage various implementation runs, and analyze the placed and routed design. You can also experiment with different constraints or implementation strategies.

RECOMMENDED: *Always reference the Vivado IP using the XCI or XCIX file. Xilinx does not recommend reading just the IP DCP file. While the DCP does contain constraints, it does not provide other output products that an IP could deliver and that could be needed, such as ELF, COE, and Tcl scripts.*

Note: ISE IP is only supported for 7 series devices. ISE format IP NGC (.ngc) are no longer supported with UltraScale devices. Users should migrate their IP to native Vivado format prior to beginning UltraScale designs.

Note: When you import an NGC or EDIF file with embedded timing constraints, the constraints are not used by the Vivado Design Suite. Design constraints must be formatted as XDC commands. For information on creating Xilinx® design constraints (XDC) files, see the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 7]. For information on converting user constraints files (UCF) to XDC constraints, see the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 8].

I/O Planning Projects

You can perform clock resource and I/O planning early in the design cycle by creating an empty I/O planning project. You can define I/O ports within the Vivado IDE or import them with either comma separated value (CSV) or XDC input files. You can also create empty I/O planning projects to explore the logic resources available on the different device architectures.

After I/O assignment, the Vivado IDE can create CSV, XDC, and RTL output files for use later in the design flow when RTL sources or netlists are available. The output files can also be used to create schematic symbols for use in the printed circuit board (PCB) design process.

Certain types of IP, such as Memory, GT, PCIe®, and Ethernet interfaces have I/O ports associated with them. These IP need to be configured in a Manage IP project, or a RTL project. See the Migrating to an RTL Design section in the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 9] and Clock Planning for IP with I/O Ports for more info.

Note: You can use an I/O planning project as the basis for an RTL-based design project. For more information, see Migrating to an RTL Design section in the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 9].

Imported Projects

You can import RTL project data from Synopsys Synplify, XST, or ISE® Design Suite Project Navigator to migrate a project into the Vivado tools. The project source files and compilation order are imported, but implementation results and settings are not.

Creating a Project

The New Project wizard takes you through the steps to define a project name and location, add source files and constraint files to the project, and select a target device. Refer to Appendix B, Vivado Naming Conventions for information on naming files and projects.

CAUTION! *The Windows operating system has a 260 character limit for path lengths which can affect the Vivado tools. To avoid this issue, use the shortest possible names and directory locations when creating projects, defining IP or managed IP projects, or creating block designs.*

1. In the Vivado IDE, select **File > Project > New**.

   Note: Alternatively, you can also select **Create Project** on the Getting Started Page.

2. In the New Project wizard, review the overview, and click **Next**.
3. In the Project Name page, set the following options, and click **Next**.
   - **Project name**: Specifies the name of the project (for example, `project_1`).
   - **Project location**: Specifies the location for the new project directory.
   - **Create Project Subdirectory**: Adds a subdirectory with the same name as the project to the specified project location.

     Note: By default, this check box is enabled and the project file (.xpr extension) is created at `<project_location>/<project_name>`. All folders and data files created for the project are stored in the `<project_name>` subdirectory. If you disable this check box, the project file (.xpr extension) is created at <`project_location`>, and all folders and data files created for the project are stored in that project location.

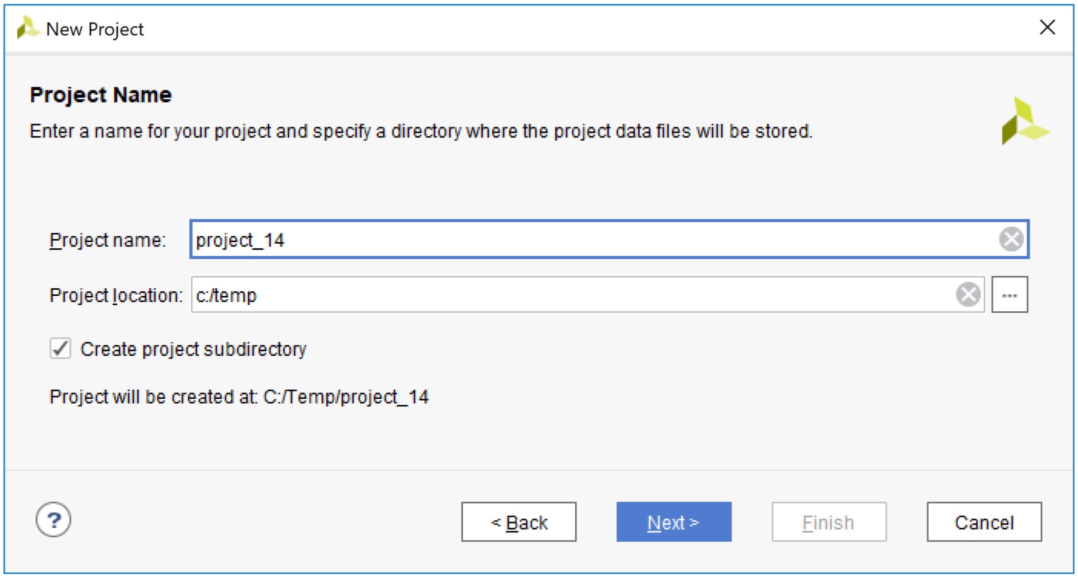

Figure 2-1: **New Project Wizard—Project Name Page**

4. In the Project Type page, specify the type of project, which determines the types of source files that are associated with the project.

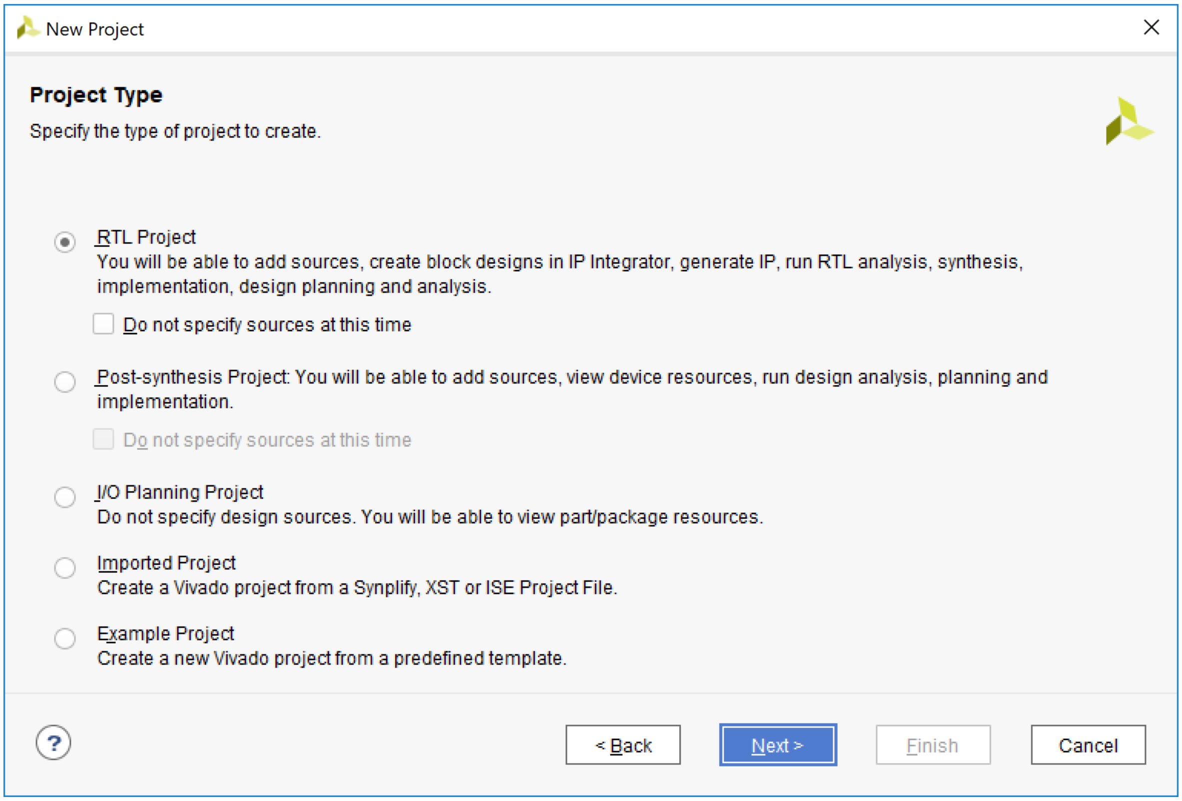

Figure 2-2: **New Project Wizard—Project Type Page**

5. Depending on the type of project you are creating, continue with the instructions in one of the following sections. The remaining pages of the wizard guide you through adding appropriate sources to the project.

- Creating an RTL Project
- Creating a Post-Synthesis Project
- Creating an I/O Planning Project
- Importing an External Project
- Tcl Commands for Working with Projects

Creating an RTL Project

An RTL project may have RTL, Block Design, IP and/or RTL sources. This dialog lets you specify which sources to add during project creation. Addition files can be added later during RTL code development, analysis as well as synthesis and implementation. For more information on RTL development and analysis, see Chapter 4, Elaborating the RTL Design.

1. Follow the steps in Creating a Project.
2. In the Project Type page, select **RTL Project**, and click **Next**.

Note: If necessary, you can select **Do not specify sources at this time**. This skips the steps of adding design sources and enables you to select the target part and create the project.

3. In the Add Sources page, set the following options, and click **Next**:

   - **Add Files**: Opens a file browser so you can select files to add to the project. You can add the following file types to an RTL project: Verilog, VHDL, SystemVerilog, BD, XCI, EDIF, NGC, BMM, ELF, and other file types.

     Note: In the Add Source Files dialog box, each file or directory is represented by an icon indicating it as a file or folder. A small red square indicates it is read only.

   - **Add Directories**: Opens a directory browser to add source files from the selected directories. Files in the specified directory with valid source file extensions are added to the project.

   - **Add Sources from Subdirectories**: Specifies that the tool should scan the listed directory's directory tree for additional sources.

   - **Create File**: Opens the Create Source File dialog box in which you can create new VHDL, Verilog, Verilog header, or SystemVerilog files. Create Source File dialog box, set the following options:

     - **File type:** Specifies one of the following file formats: Verilog file (.v extension), Verilog Header file (.vh extension)., SystemVerilog file (.sv extension), VHDL file (.vhdl extension), or Memory Files (.mem extension).

     - **File name**: Specifies a name for the new HDL source file.

     - **File location**: Specifies a location in which to create the file.

     Note: A placeholder for the file is added to the list of sources. The file is created when you click **Finish**.

   - **Library**: Specifies the RTL library for a file or directory. You can select a library name, or specify a new library name by typing in the Library text field.

     Note: This option applies to VHDL files only. By default, HDL sources are added to the `xil_defaultlib` library. You can create or reference additional user VHDL libraries as needed. For Verilog and SystemVerilog files, leave the library set to `xil_defaultlib`.

   - **HDL Source for**: Specifies whether the source being loaded is an RTL source file for synthesis and simulation or an RTL test bench for simulation only.

   - **Remove**: Removes the selected source files from the list of files to be added.

   - **Move Up / Move Down**: Moves the file or directory up/down in the list order. The order of the files affects the order of elaboration and compilation during downstream processes such as synthesis and simulation.

   - **Scan and Add RTL Include Files into Project**: Scans all RTL source files and adds any referenced Verilog 'include files into the project structure.

   - **Copy Sources into Project:** Copies the added source files and 'include files into the local project directory instead of referencing the original files. If you added directories of source files using Add Directories, the directory structure is maintained when the files are copied locally into the project. For more information, see Using Remote Sources or Copying Sources into Project in Chapter 3.

- **Add Sources from Subdirectories**: Adds source files from the subdirectories of directories specified with Add Directories.
- **Target Language**: Specifies the target language for the design as either Verilog or VHDL. New RTL files default to the specified target language. Output files are generated from the design in the specified target language.
- **Simulator Language**: Specifies the language in which output products are generated for simulation as well as the file types used for third party simulation scripts. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 10].

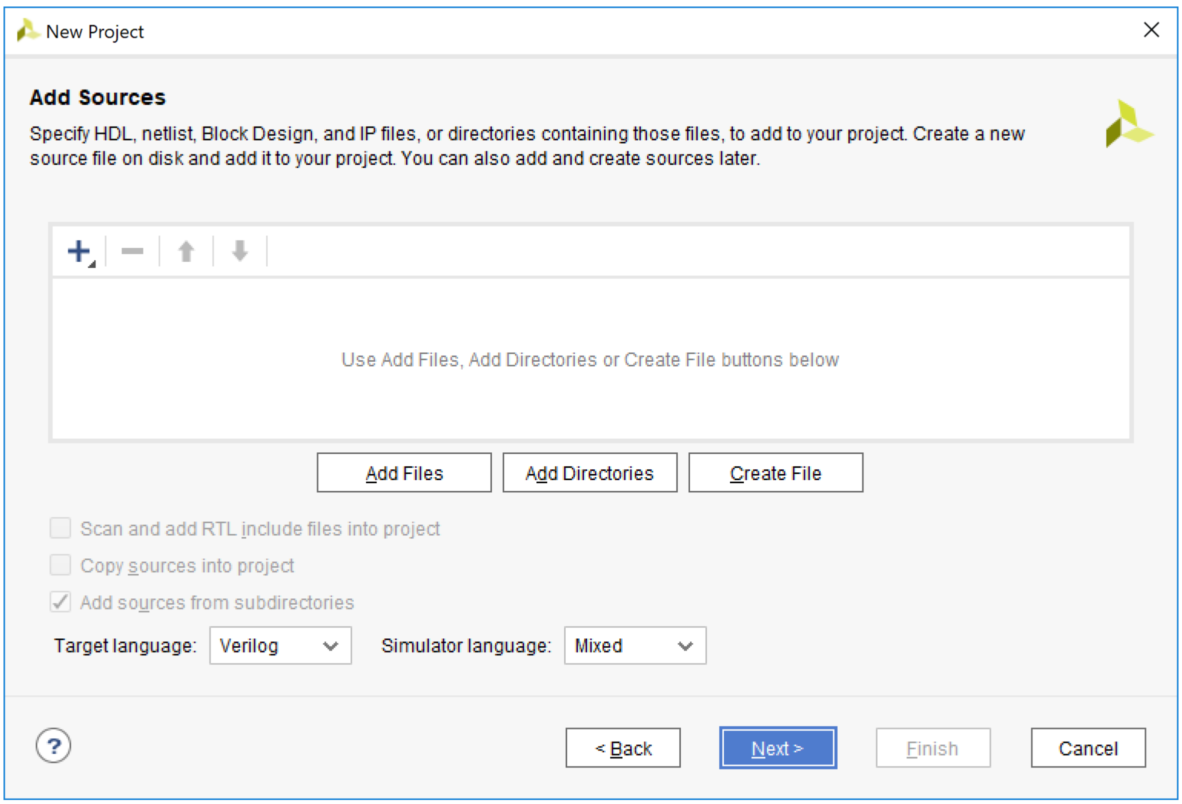

Figure 2-3: **New Project Wizard—Add Sources Page**

- **Add Sources**: Invokes a file browser so you can select Xilinx Core Instance (XCI) files, which are native to the Vivado Design Suite, a Core Container (XCIX) file, which is a single file representation for an IP, or CORE Generator core (XCO) files. You can also add Block Design files (BD) from the Vivado IP Integrator feature, or Mathworks Simulink project files (SLX or MDL) for DSP sub-designs.

  The XCI file is an IP-XACT component instance XML file that records the values of project options, customization parameters, and port parameters used to create the IP. The XCIX is a compressed binary file containing the entire IP directory and all output products, including the XCI, synthesis, simulation and support files. See the Core Container section in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 11] for more details.

Note: When you add XCI or XCIX IP created with the Vivado IP catalog, the Vivado IDE automatically imports all available generated targets, such as HDL sources, into the project. When you run synthesis, the IP and the top-level design are synthesized together.

You can also load parameterized cores into the project from within the Vivado IDE using the IP Catalog, as described in Working with IP Sources in Chapter 3.

4. *Optional*: In the Add Constraints page, set the following options, and click **Next**:
   - **Add Files**: Invokes a file browser so you can select Synopsys Design Constraint (SDC) or XDC files to add to the project.
   - **Create File**: Creates a new top-level XDC file for the project.
   - **Remove**: Removes the selected file from the constraint list.
   - **Move Up / Move Down**: Moves a constraint file up or down in the listed order. Commands are order-dependent; the last-read command of a constraint overwrites the effects of an earlier command.
   - **Copy Constraint Files into Project**: Copies constraint files into the local project directory instead of referencing the original files.

Note: Any SDC or XDC file found in the same directories as the RTL or netlist source files associated with the project are automatically listed as constraint files to be added to the project. You can remove these files as needed.

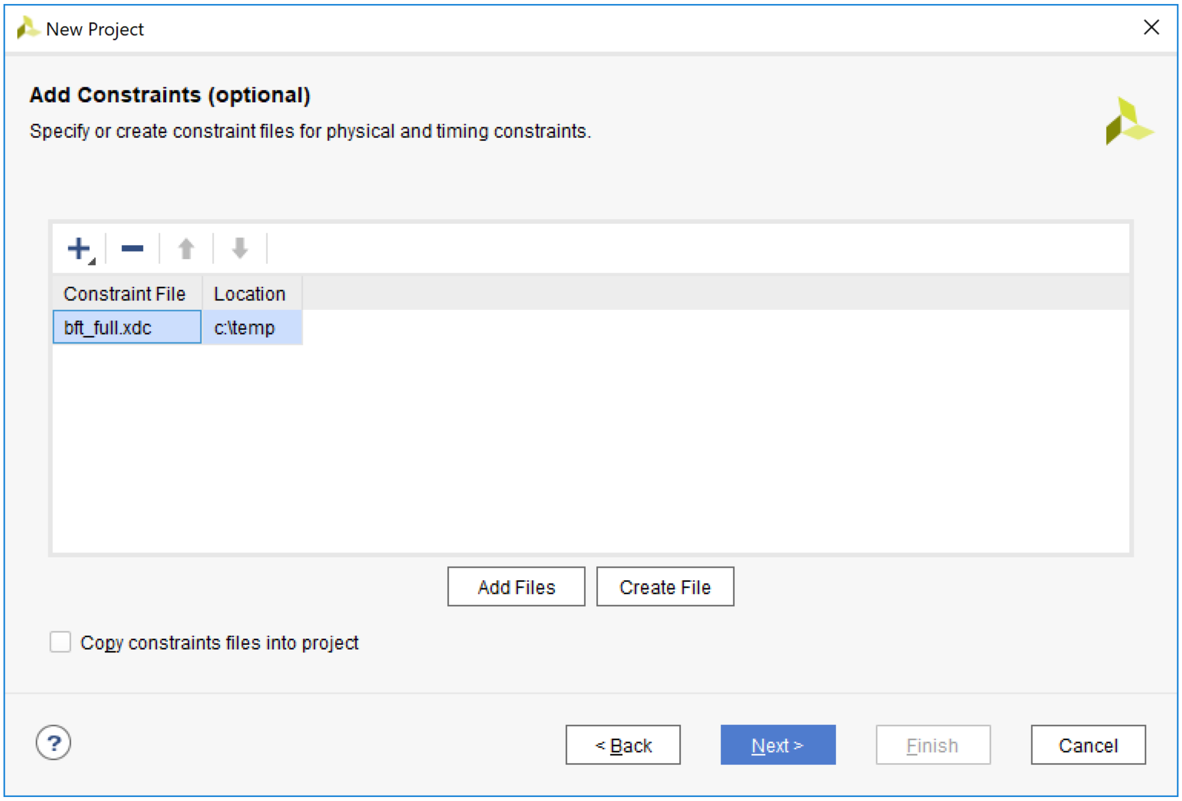

Figure 2-4: **New Project Wizard—Add Constraints Page**

5. In the Default Part page, select a Xilinx part or targeted design platform (TDP) board, and click **Next**:

   - **Parts**: Lists available devices. Information about the device resources displays in a table view, such as I/O pin count, the number of look-up tables (LUTs) and flip-flops (FFs), and available block RAM. You can filter the list using the Product Category, Family, Sub-Family, Package, Speed Grade, and Temp Grade filters. You can also use the Search field to find specific devices.

     The Vivado Design Suite installation process lets you select which Xilinx devices to install in order to reduce the disk space required by the Vivado tool. If you need to target a part that is not currently installed on your system, you must exit the tool and install the additional parts of interest. Refer to this link in *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973) [Ref 2] for more information.

   - **Boards**: Lists available development boards, or TDP boards, and the Xilinx part used on the board. Information about device resources displays in a table view similar to the one shown for Parts. You can filter the list using the Vendor, Display Name, and Board Rev filters.

   - You can also use the Search field to find specific board parts.

RECOMMENDED: *When you select a board that supports the Vivado Design Suite platform board flow, you can take advantage of automated features in the Vivado IP catalog and Vivado IP integrator. For example, you can automatically create I/O constraints for IP that supports the interfaces available on the selected board. For more information, see* [*Using the Vivado Design Suite Platform Board Flow*]*.*

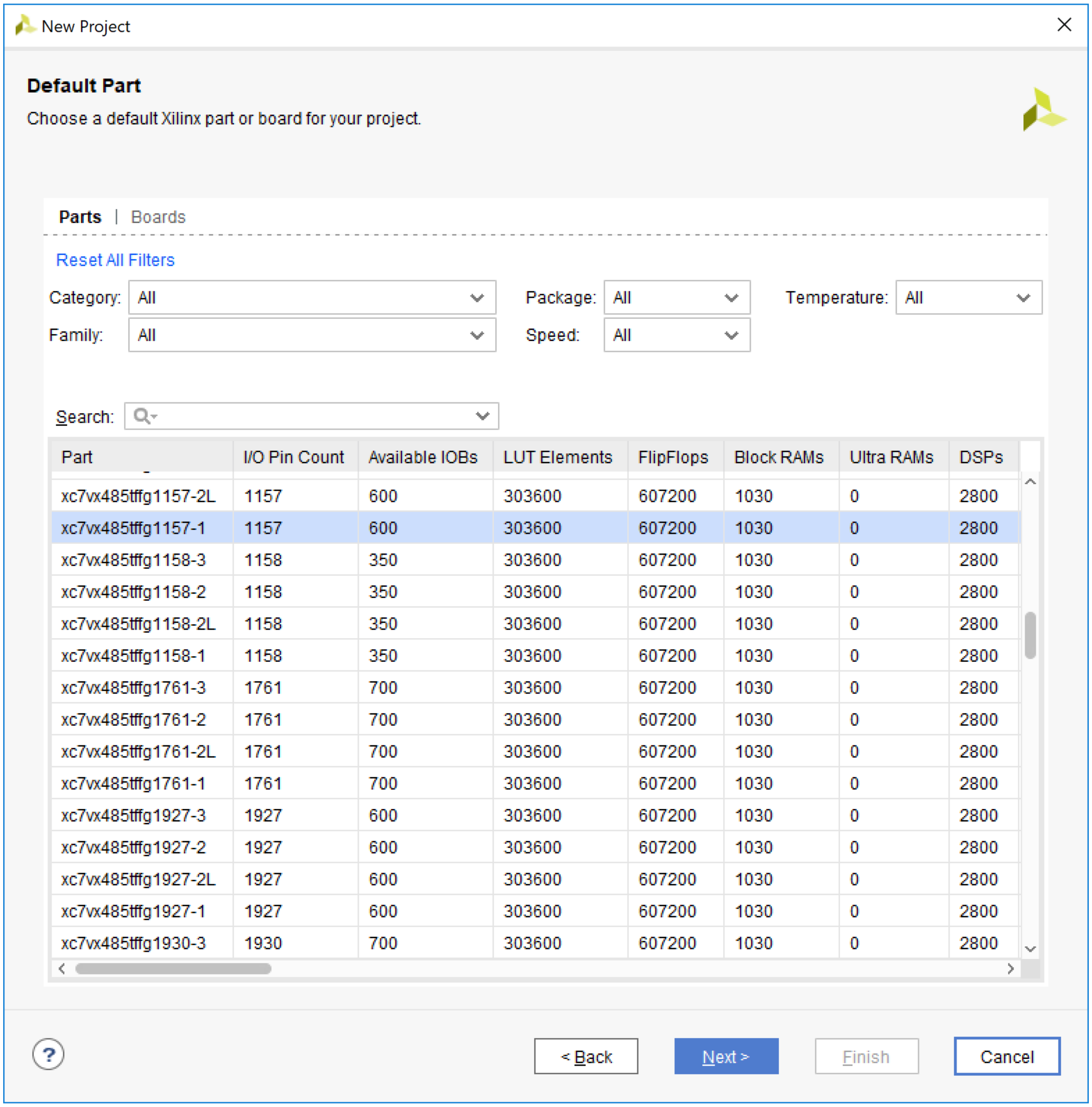

Figure 2-5: **New Project Wizard—Default Part Page**

6. In the New Project Summary page, view the selected options that define the project, and click **Finish**. When you click Finish the project directory structure is created, any files that should be made local to the project are copied, and the project file is written. Any design sources that need to be created must be defined as shown in the following step, and then are written to disk.

7. *Optional*: If you used the **Create File** option in step 3, to create a new HDL module and add it to the project, a Define Module dialog box appears.

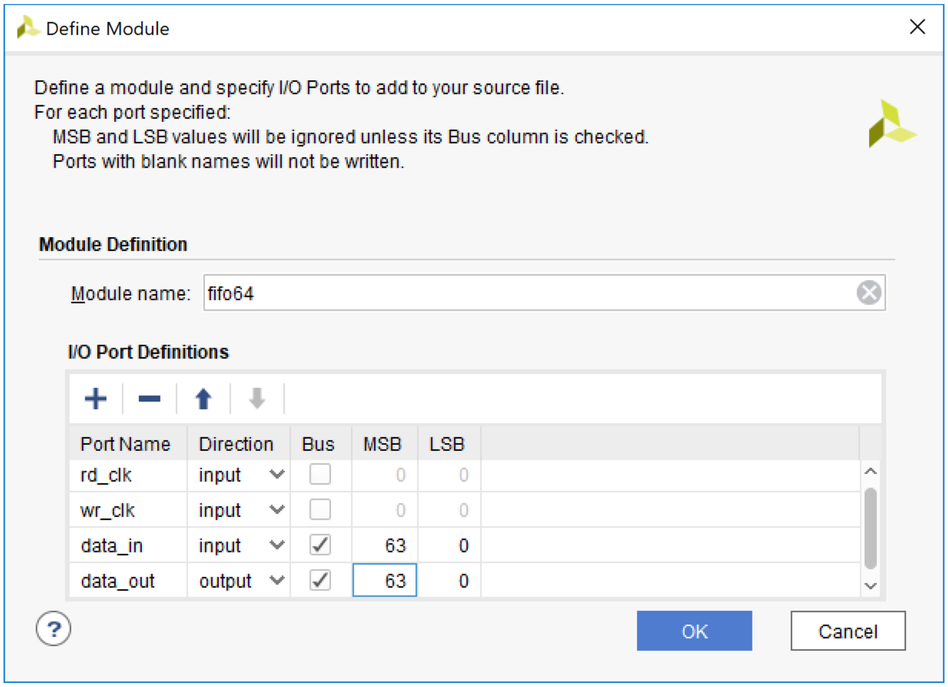

Figure 2-6: **Define Module Dialog Box**

The RTL source files are created and added to your project. The Sources window lists the newly defined modules. These new source files define the Verilog module or VHDL entity, but you must edit the files to define the logic or architecture for these blocks. To edit the new source files in the Vivado IDE Text Editor, double-click the file or select **Open File** from the right-click menu. For information on editing the newly created file, see Editing Source Files.

Creating a Post-Synthesis Project

A post-synthesis project begins with a synthesized netlist, fully generated block designs, fully generated IP, and corresponding constraints. You can then analyze, floorplan, and implement the design.

Note: You can use either XST or third-party synthesis tools to create the synthesized netlist.

IMPORTANT: *When working with EDIF and NGC files, the top cell name must match the name of the file.*

1. Follow the steps in Creating a Project.
2. In the Project Type page, select **Post-Synthesis Project**, and click **Next**.

   Note: If necessary, you can select **Do not specify sources at this time**. This skips the steps of adding design sources and enables you to select the target part and create the project.

3. In the Add Netlist Sources page, use the following options to specify netlist files to read, identify the file containing the top module, and define directories to search for lower-level module netlist, and click **Next**.

    - **Add Files**: Invokes a file browser so you can select netlist files (structural Verilog, SystemVerilog, EDIF or NGC), BD Files, and XCI files (all the output products for the IP must be generated, including the DCP), or design checkpoint files (DCP) to add to the project.

RECOMMENDED: *Always reference the IP using the XCI file. Always reference a Block Design using the BD file Xilinx does not recommend reading just the IP or BD DCP file. While the DCP does contain constraints, it does not provide other output products that an IP or BD could deliver and that could be needed, such as ELF, COE, and Tcl scripts.*

    Note: Enable the **Top** radio button for the file that contains the top-level netlist.

    - **Add Directories**: Invokes a directory browser so you can select directories to search for modules. Files in the specified directory with valid source file extensions are added to the project.
    - **Remove**: The button removes the selected source files and directories from the list.
    - **Move Up / Move Down**: Moves the file or directory up/down in the list order. The order of the files affects the processing order.
    - **Copy Sources into Project**: Copies files into the local project directory instead of referencing the original files. If you added directories of source files using Add Directories, the directory structure is maintained when the files are copied locally into the project. For more information, see Using Remote Sources or Copying Sources into Project in Chapter 3.
    - **Add Sources from Subdirectories**: Looks for netlist files in the subdirectories of directories specified with Add Directories.

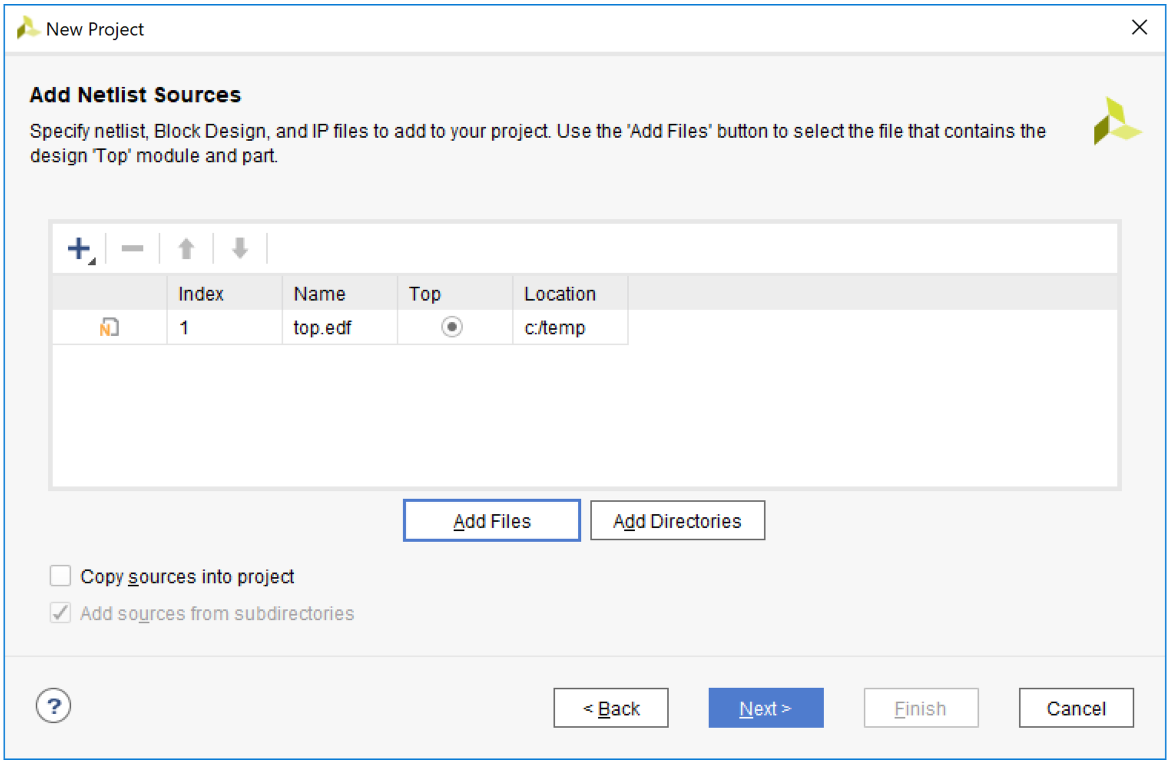

Figure 2-7: **New Project Wizard—Add Netlist Sources Page**

4. *Optional*: In the Add Constraints page, set the following options, and click **Next**:
   - **Add Files**: Invokes a file browser so you can select SDC or XDC files to add to the project.
   - **Create File**: Creates a new top-level XDC file for the project.
   - **Remove**: Removes the selected file from the constraint list.
   - **Move Up / Move Down**: Moves a constraint file up or down in the listed order. Commands are order-dependent; the last-read command of a constraint overwrites the effects of an earlier command.
   - **Copy Constraints into Project**: Copies constraint files into the local project directory instead of referencing the original files.

   Note: Any SDC or XDC file found in the same directories as the RTL or netlist source files associated with the project are automatically listed as constraint files to be added to the project.

5. In the Default Part page, select a Xilinx part or TDP board, and click **Next**:
   - **Parts**: Lists available devices. Information about the device resources displays in a table view. You can filter the list using the Product Category, Family, Sub-Family, Package, Speed Grade, and Temp Grade filters. You can also use the Search field to find specific devices.

- **Boards**: Lists available TDP boards, and the Xilinx part used on the board. Information about device resources displays in a table view, such as I/O pin count, the number of LUTs and flip-flops, and available block RAM. You can filter the list using the Vendor, Display Name, and Board Rev filters. You can also use the Search field to find specific board parts.

6. In the New Project Summary page, view the selected options that define the project, and click **Finish**.

Creating an I/O Planning Project

You can use an I/O planning project for device exploration and for planning the device pinout for an in-progress system-level design. You can create this type of project prior to completing the HDL or the synthesized netlist. For example, this allows you to exchange design information with the system-level or PCB designer. For more information about I/O planning, see the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 9].

1. Follow the steps in Creating a Project.
2. In the Project Type page, select **I/O Planning Project**, and click **Next**.
3. *Optional*: In the Import Ports dialog box, use the following options to select a file for importing I/O Port definitions and constraints, and click **Next**.
    - **Import CSV**: Selects a CSV file with I/O Ports definitions. For more information on CSV files, see the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 9].
    - **Import XDC**: Selects an XDC with I/O Port-related constraints only.
    - **Do not import I/O ports at this time**: Creates an empty project. You can create or import I/Os later.

    Note: Use an RTL project to perform I/O pin planning on a design using RTL header or source files.

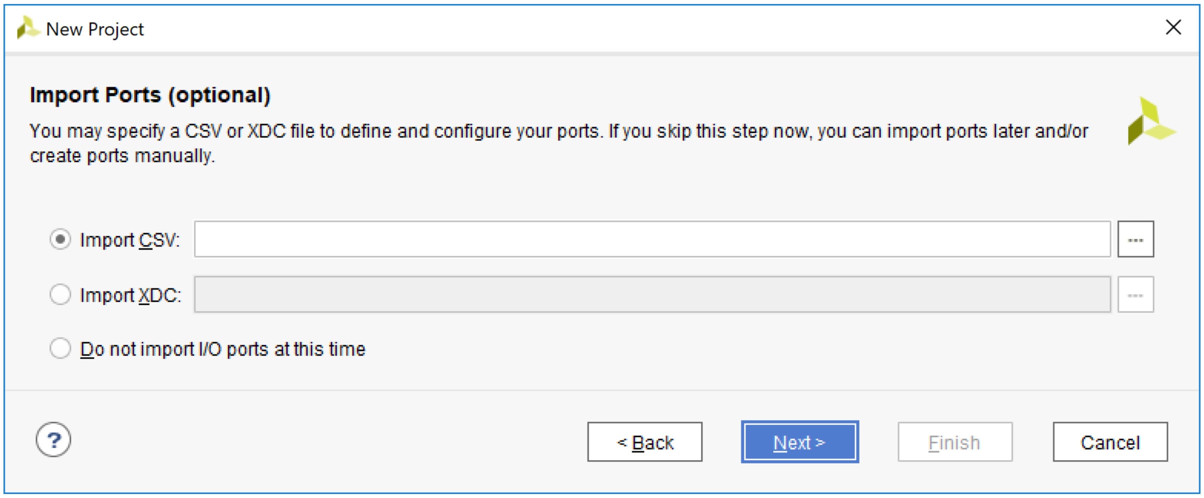

Figure 2-8: **New Project Wizard—Import Ports Page**

4. In the Default Part page (Figure 2-5), select a Xilinx part or TDP board, and click **Next**:
   - **Parts**: Lists available devices. Information about the device resources displays in a table view. You can filter the list using the Product Category, Family, Sub-Family, Package, Speed Grade, and Temp Grade filters. You can also use the Search field to find specific devices.
   - **Boards**: Lists available TDP boards, and the Xilinx part used on the board. Information about device resources displays in a table view, such as I/O pin count, the number of LUTs and flip-flops, and available block RAM. You can filter the list using the Vendor, Display Name, and Board Rev filters. You can also use the Search field to find specific board parts.
5. In the New Project Summary page, review the options you selected to define the project, and click **Finish** to create and open the project.

Note: For more information on Memory IP I/O planning, see the *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899) [Ref 9].

Importing an External Project

You can import an existing RTL-level project file created outside of the Vivado IDE, for example, using Synopsys Synplify, XST, or ISE Design Suite Project Navigator. The Vivado IDE detects the source files in the specified project and automatically adds the files to the new project. Settings such as top module, target device, and VHDL library assignment are imported from the existing project.

Note: For more information on importing an XST or ISE Design Suite project, see the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 8].

1. Follow the steps in Creating a Project.
2. In the Project Type page, select **Imported Project**, and click **Next**.

3. In the Import Project page, use the following options to specify the project file to import, and click **Next**.
   - **ISE**: Imports the specified Xilinx ISE Design Suite (.xise extension) project file.
   - **Synplify**: Imports the specified Synplify (.prj extension) project file.
   - **XST**: Imports the specified XST (.xst extension) project file.
   - **Copy Sources into Project:** Copies files into the local project directory instead of referencing the original files.

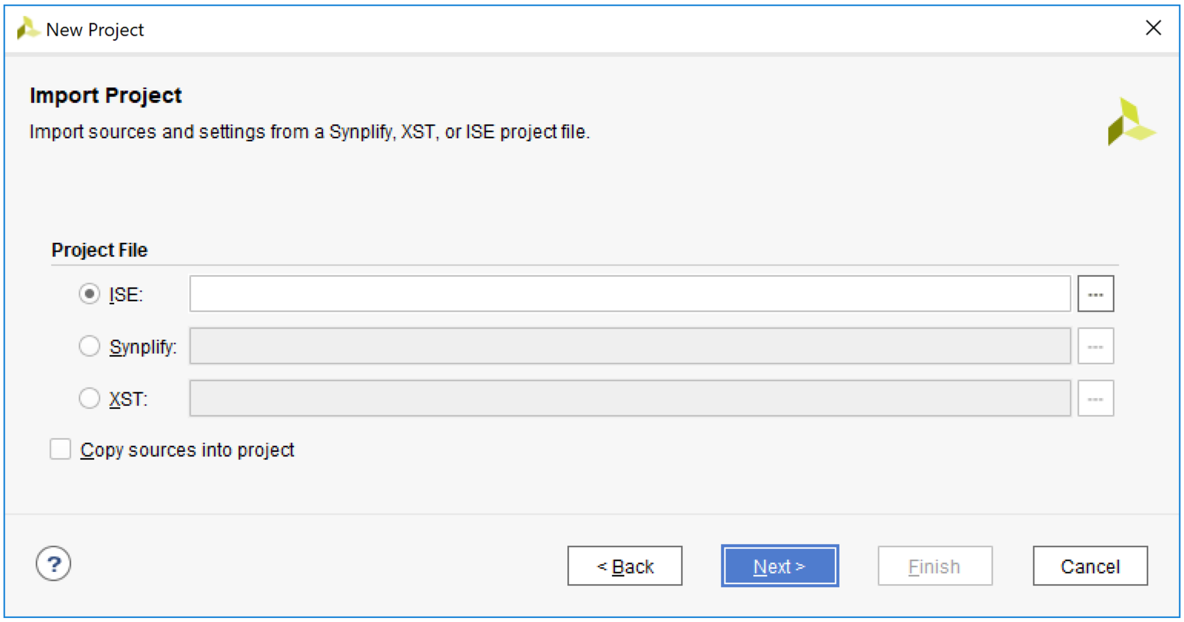

Figure 2-9: **New Project Wizard—Import Project Page**

4. In the New Project Summary page, review the options that define the project, and click **Finish**.

   Note: The target part for the project is defined with the settings of the imported project.

   The Vivado IDE imports the RTL source files and constraint files from the specified project, and creates a project file in the specified directory. The Vivado IDE writes a summary of the import process to the Import Summary Report log file in the new project directory. In this summary file, you can review the steps used in creating the project as well as any errors or warnings.

Tcl Commands for Working with Projects

Following are Tcl commands associated with creating a project. For an example script, see Creating a Project Using a Tcl Script.

Note: For more information on Tcl commands, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4], or type `<command> -help`.

Tcl Commands for Creating a Project

Following are the associated Tcl commands:

- **Tcl Commands**: `create_project` and `set_property`
- **Tcl Command Example (RTL Project)**: `create_project my_project C:/team/designs/my_project -part xc7k325tffg676-2 set_property DESIGN_MODE RTL [current_fileset]`
- **Tcl Command Examples (Post-Synthesis Project)**:

```
create_project my_IO_project C:/team/designs/my_IO_project -part
xc7k325tffg676-2
```

```
set_property design_mode GateLvl [current_fileset]
```

- **Tcl Command Examples (I/O Planning Project)**:

```
create_file project_io C:/projects/project_io -part
xc7vx485tffg1157-1
```

```
set_property design_mode PinPlanning [current_fileset]
```

Tcl Commands for Importing a Project

Following are the associated Tcl commands:

- **Tcl Command**: `create_project` and `import_xise set_property DESIGN_MODE RTL [current_fileset]`
- **Tcl Command Examples**:

```
create_project project_import_ise C:/projects/project_import_ise
```

```
import_xise C:/projects/old/wave_gen_vhd_s6/wave_gen_vhd_s6.xise
-copy_sources
```

Tcl Commands for Adding Design Sources, Constraints Files, and Simulation Sources

Following are the associated Tcl commands:

- **Tcl Command**: `add_files` or `import_files`
- **Tcl Command Examples**:

```
add_files top.v
```

```
import_files -fileset constrs_1 C:/projects/sources/timing.xdc
```

```
add_files -norecurse source_dir

import_files source_dir
```

Note: The `add_files` command references the file from its current location. The `import_files` command copies the file into the project.

CAUTION! *The `read_xdc`, `read_vhdl`, `read_verilog`, `read_ip`, and `read_edif` Tcl commands are designed for use with Non-Project Mode only. For more information, see* *Working with Sources in Non-Project Mode in Chapter 3*.

TIP: *You can use the `PATH_MODE` property with the `add_files` Tcl command to specify whether to use absolute or relative paths. By default, relative paths are used. For more information, see the Vivado Design Suite Properties Reference Guide (UG912) [Ref 14].*

Tcl Commands for Adding Existing IP Sources

Following are the associated Tcl commands:

- **Tcl Command**: `add_files` or `import_ip`
- **Tcl Command Example**: `import_ip C:/projects/sources/char_fifo/char_fifo.xci`

Note: The `add_files` command references the XCI file and associated output products from their current location. The `import_ip` command copies the XCI file and associated output products into the project.

Tcl Commands for Setting the Project Part

Following are the associated Tcl commands:

- **Tcl Command**: `create_project` or `set_property`
- **Tcl Command Examples:**

```
create_project my_project C:/projects/my_project -part
xc7k325tffg676-2

set_property PART xc7k70tfbg676-2 [current_project]
```

Note: You can set the part either when you create the project or after you create the project.

Note: It is easy to get a template script by running the write_project_tcl command on an existing or example project.

Using the Vivado Design Suite Platform Board Flow

The Vivado Design Suite lets you create projects using Xilinx target design platform boards (TDP), or user-specified boards that have been added to a board repository. When you select a specific board, the Vivado design tools show information about the board, and enable additional designer assistance as part of IP customization, and for IP integrator designs.

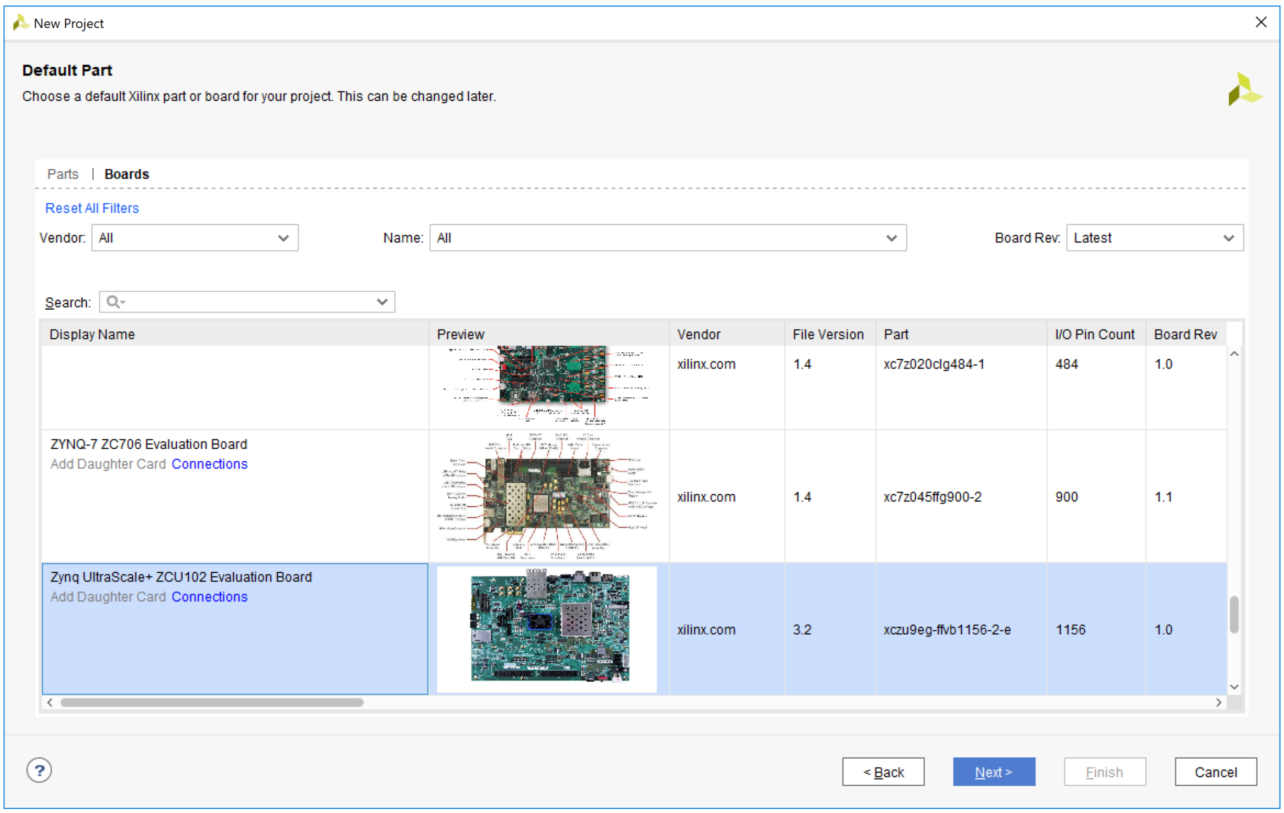

Figure 2-10: **New Project Wizard—Default Part/Board**

Adding User-Boards to a Repository

The Vivado Design Suite installation includes board definition files for the TDP boards that are delivered as part of the tool. You can also create your own board files, using the schema described in Appendix A, Board File, to add to a board repository to be used with Vivado Design Suite.

In order to add your own board files, or third-party files, to the board repository you must define the following parameter either in your `Vivado_init.tcl` file, or soon after opening the Vivado Design Suite:

```
set_param board.repoPaths [list "<path1>" "<path2>" "..."]
```

For more information about the `Vivado_init.tcl` file refer to this link in the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4].

TIP: *The Vivado Design Suite board repository at <Vivado_install_location>/data/boards is always read regardless of the value of this parameter.*

Using the IP Catalog with the Platform Board Flow

A Board tab is available in the Customize IP dialog box when you are working with IP from the Xilinx IP catalog that supports the platform board flow. You can select the board interfaces to use in the IP customization. Based on the IP interfaces supported by the selected board part, IP configuration options change to enable physical constraint generation specific to the board, such as I/O locations and I/O standards. For more information on configuring board-related IP see *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 13].

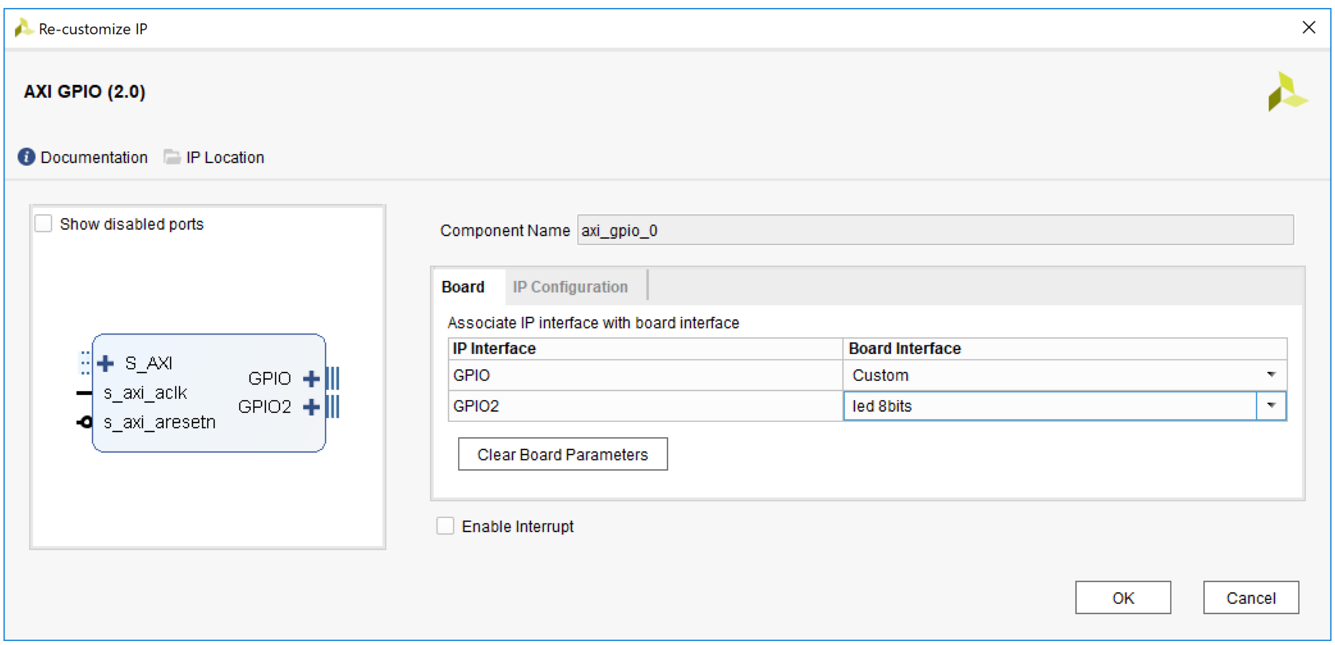

Figure 2-11: **Board Tab in the Customize IP Dialog Box**

Using the Vivado IP Integrator with the Platform Board Flow

Optionally, you can use the Vivado IP integrator to add IP to your block design. If you selected a board for the project, the Board window is available in the IP integrator, showing the IP interfaces that are available on the selected board, and which of those interfaces have been used.

Vivado IP integrator instantiates the pre-configured IP and assigns the physical board constraints, such as I/O location and I/O standards for the IP, as well as any related parameters used for implementation and device configuration.

After configuration, all of the board physical constraints are automatically passed to the downstream synthesis and implementation tools. For more information on using the platform board flow, see this link in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 13].

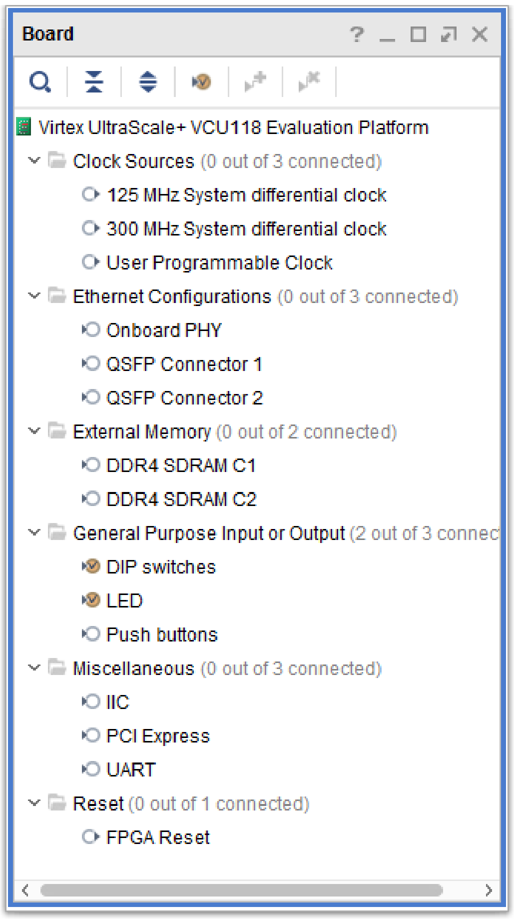

Figure 2-12: **Board Window in Vivado IP Integrator**

Managing Projects

Opening a Project

When a project is opened, the Vivado IDE restores the state of the project from the time the project was closed. The project state includes the current source file order, disabled and enabled source files, active and target constraint files, and the state of synthesis, simulation, and implementation runs.

To open a project, use one of the following methods:

- In the Getting Started page, click **Open Project**.
- Select **File > Project > Open**.
- Click the **Open Project** toolbar button.
- In the Tcl Console, enter the `open_project` command.

From the Open Project dialog box, you can select a project file (.xpr extension). The File Preview window in the Open Project dialog box displays information about the currently selected file.

Note: Alternatively, you can double-click the Vivado IDE project file (.xpr extension) in Windows Explorer to open the project.

Tcl Command for Opening a Project

Following is the associated Tcl command:

- **Tcl Command**: `open_project`
- **Tcl Command Example**: `open_project c:/projects/project_1.xpr`

Opening Multiple Projects

To open multiple projects in a single session, use any of the methods described in Opening a Project to open an additional project while a project is already open. The Vivado IDE prompts you to close the current project. If you do not close the first project, both projects are opened. Each open project has a separate IDE window.

When opening multiple projects from the same Vivado IDE application process, be aware that the commands used in *all open projects* are written to the Tcl Console. When reviewing the transcript of commands, it might not be clear which project the commands are associated with. In addition, there is only a single `vivado.jou` and a single `vivado.log` file for the application for all projects.

Note: System memory requirements can hinder performance when opening multiple projects.

Saving a Project

Projects are automatically saved for you. For example, any time you make a change to a project, such as changes to source configuration, properties on files, or run options, the project is automatically saved on disk.

TIP: *However, changes to the design constraints are not automatically saved as part of the project. You must use the Save Constraints command, or Save Constraints As..., to write constraint changes to disk.*

To save a project to a new location, select **File > Project > Save As**. This copies the entire project directory structure to a specified location and maintains the status of the existing runs when run results are included.

Tcl Command for Saving a Project

Following is the associated Tcl command:

- **Tcl Command**: `save_project_as`
- **Tcl Command Example**: `save_project_as new_project c:/projects/project_1.xpr`

Closing a Project

To close a project, select **File > Close Project**. When you close a project, you are prompted to save any unsaved changes to the design or source files.

Tcl Command for Closing a Project

Following is the associated Tcl command: `close_project`

Archiving Projects

You can create a project archive to store as backup or to send to a remote site. When archiving a project, the Vivado IDE does the following:

- Parses the hierarchy of the design.
- Copies the required source files, include files, and remote files from the library directories.
- Copies the constraints.
- Optionally, copies the results of the various synthesis, simulation, and implementation runs.
- Creates a ZIP file of the project.

To archive a project:

1. Select **File > Project > Archive**.
2. In the Archive Project dialog box, set the following options, and click **OK**.
   - **Archive name**: Specifies the name of the project archive.
   - **Archive location**: Specifies a location to store the project archive file.
   - **Temporary location**: (Windows only) Specifies a temporary directory to copy files to when creating the project archive. The temporary directory is created if it does not exist, and is emptied when the archive process is complete. By default, the Vivado tool creates a temporary directory inside of the current working directory.
   - **Include configuration settings**: Includes the `Vivado_init.tcl` file, which contains Tcl initialization commands that are helpful in debugging your design. For more information, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4].
   - **Include run results**: Includes the settings and results of the runs performed on the project. Including the results of synthesis and implementation runs can significantly increase the size of the project archive.
   - **Include local IP cache results**: Includes the information included in the local IP cache (if any) to speed up IP generation times using version of the tools.

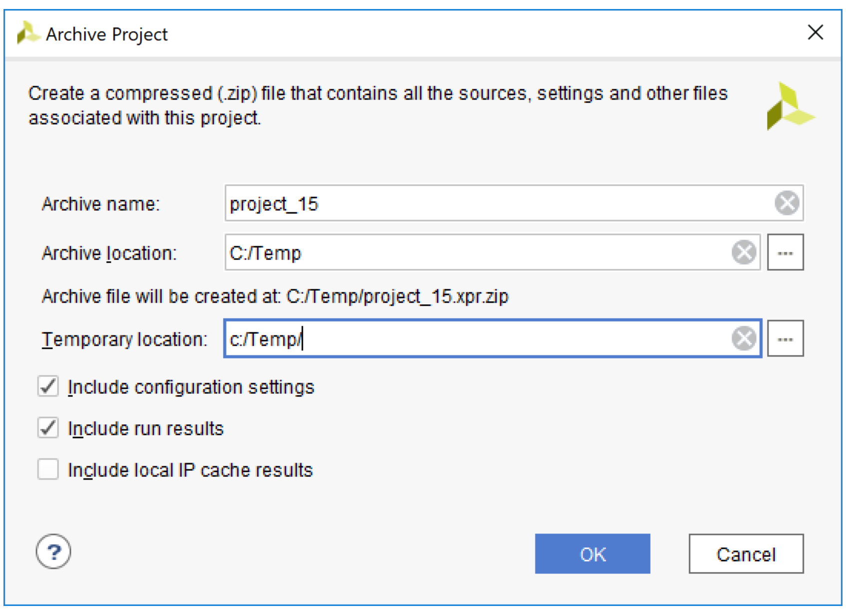

Figure 2-13: **Archive Project Dialog Box**

The Vivado IDE creates a project archive in ZIP file format that contains the required source files, include files, and run files (if specified) as well as an archive.log file of the archival process. You can review the creation of the archive in the archive.log file.

You can also use the write_project_tcl command to generate a tcl script that will re-create the current project. The script will keep the project settings and sources, but may not retain output products or design state.

Tcl Command for Archiving a Project

Following is the associated Tcl command:

- **Tcl Command**: archive_project
- **Tcl Command Example**: archive_project -exclude_run_results proj3.zip

Note: To avoid the 256 character limit on Windows, use the -temp_dir option to specify a temporary directory to copy files to when creating the project archive.

Working with Source Control Systems

VIDEO: *See the* Vivado Design Suite QuickTake Video: Using Vivado Design Suite with Revision Control *for an introduction to working with source control systems.*

Using the Project Summary

The Vivado IDE includes an interactive Project Summary that updates dynamically as design commands are run and as the design progresses through the design flow. The Project Summary includes the Overview tab and a user-configurable Dashboard, as shown in the following figure. For information, see this link in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 6].

To open the Project Summary, do either of the following:

- Select **Windows > Project Summary**.
- Select the **Project Summary** toolbar button .

Note: The Overview tab in the Project Summary appears by default.

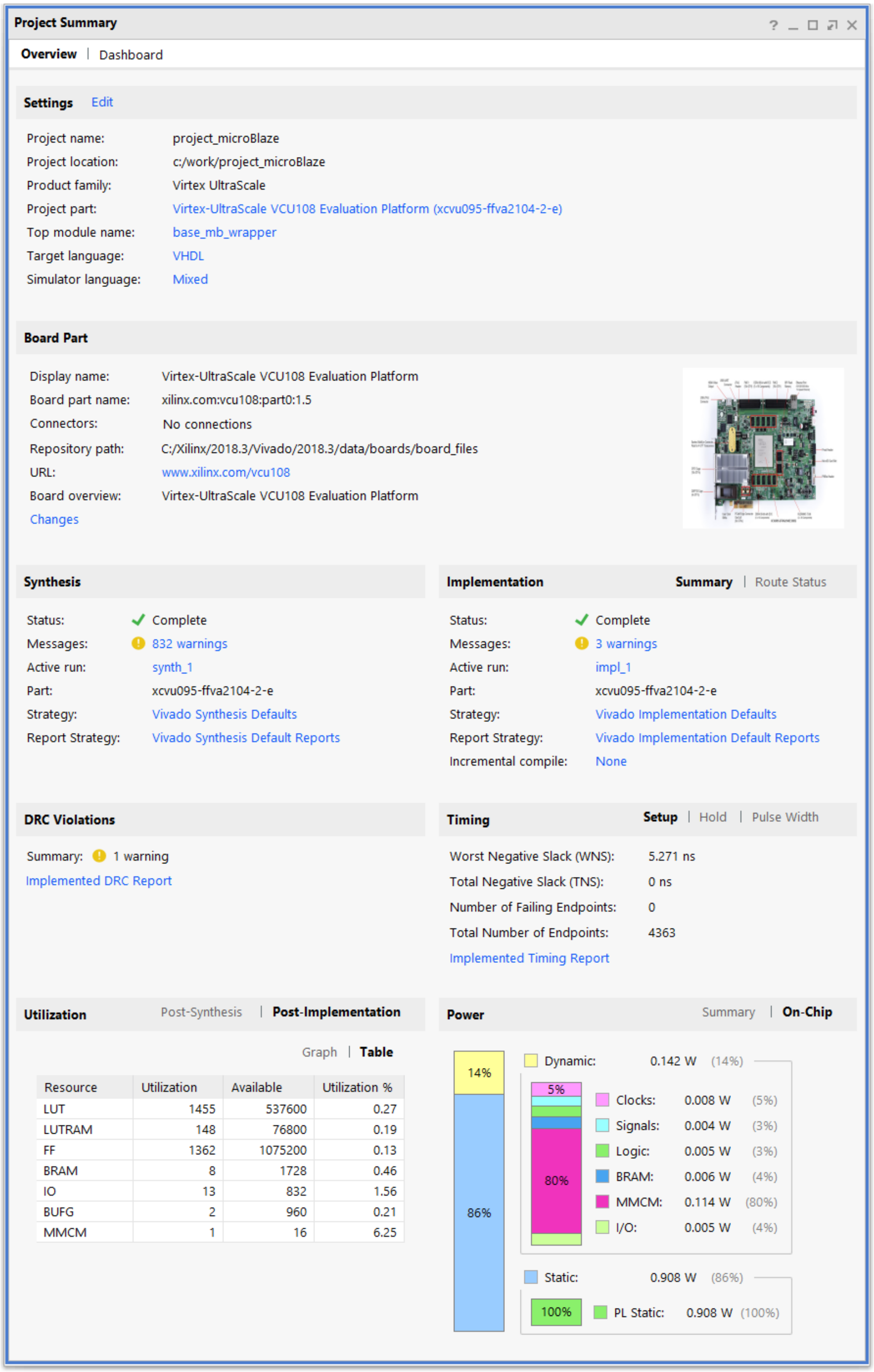

Figure 2-14: **Project Summary**

Configuring Project Settings

You can configure settings to meet specific needs for each project. Settings include general settings related to the top module definition as well as settings for the following: simulation, synthesis, implementation, bitstream, and IP.

To open the Settings dialog box, use any of the following methods:

- Select **Tools > Settings**.
- Click the **Settings** toolbar button .
- In the Flow Navigator, click **Settings** in the Project Manager section, or right click on:
  - **SIMULATION** to get **Simulation Settings**
  - **RTL ANALYSIS** to get **Elaboration Settings**
  - **SYNTHESIS** to get **Synthesis Settings**
  - **IMPLEMENTATION** to get **Implementation Settings**
  - **PROGRAM AND DEBUG** to get **Bitstream Settings**.
- In the Project Summary, click the **Edit** link next to the Settings header, or click the strategy or flow in either the Synthesis or Implementation section.

Depending on how you invoke the Settings dialog box, the appropriate category appears by default. For example, if you click **Simulation Settings** in the Flow Navigator, the Simulation category appears in the Settings dialog box. The following sections provide detailed information for each category.

General Settings

The General settings enable you to specify the project name, part, target language, target simulator, top module name, and language options.

- **Name**: Specifies the project name.
- **Project Device**: Specifies the target device to be used as a default for both synthesis and implementation. Click the browse button to open the Select Device dialog box to choose a device.

  Note: If you have multiple synthesis or implementation runs, you can also change the device used for a specific run by changing the run settings from the Run Properties window. For more information, see the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 6].
- **Target Language**: Specifies the target output language for the design as either Verilog or VHDL. The tool generates RTL output from the design in the specified target language. Specific examples of output controlled by the target language are synthesis, simulation, top-level wrappers, test benches, and IP instantiation templates.

- **Default Library**: Specifies the default library for the project. All files without an explicit library specification are compiled in this library. You can select a library name, or specify a new library name by typing in the Library text field.
- **Top Module Name**: Specifies the top RTL module name of the design. You can also enter a lower-level module name to experiment with synthesis on a specific module. Click the browse button to automatically search for the top module and display a list of possible top modules.
- **Language Options**

IMPORTANT: *The settings here apply to synthesis. You can also define Verilog options and Generics/Parameters options from the Settings - Simulation dialog box. The simulation settings apply to the simulation fileset and affect simulation but not synthesis.*

  - **Verilog Options**: Click the browse button to set the following options in the Verilog Options dialog box:
    - **Verilog Include Files Search Paths**: Specifies the paths to search for files referenced by ’include statements in the source Verilog files.
    - **Defines**: Specifies Verilog macro definitions for the project.
    - **Uppercase all identifiers**: Sets all Verilog identifiers to uppercase.
  - **Generics/Parameters**: VHDL supports generics while Verilog supports defining parameters for constant values. Both of these techniques allow parameterized designs that can be reused in different situations. Click the browse button to define generic and parameter values to override defaults defined in the source files.
  - **Loop Count**: Specifies the maximum loop iteration value. The default is 1000.

    Note: The Loop Count option is used during RTL elaboration but does not apply to synthesis. For synthesis, you must specify the `-loop_iteration_limit` switch in the More Options field of the Synthesis Settings dialog box.

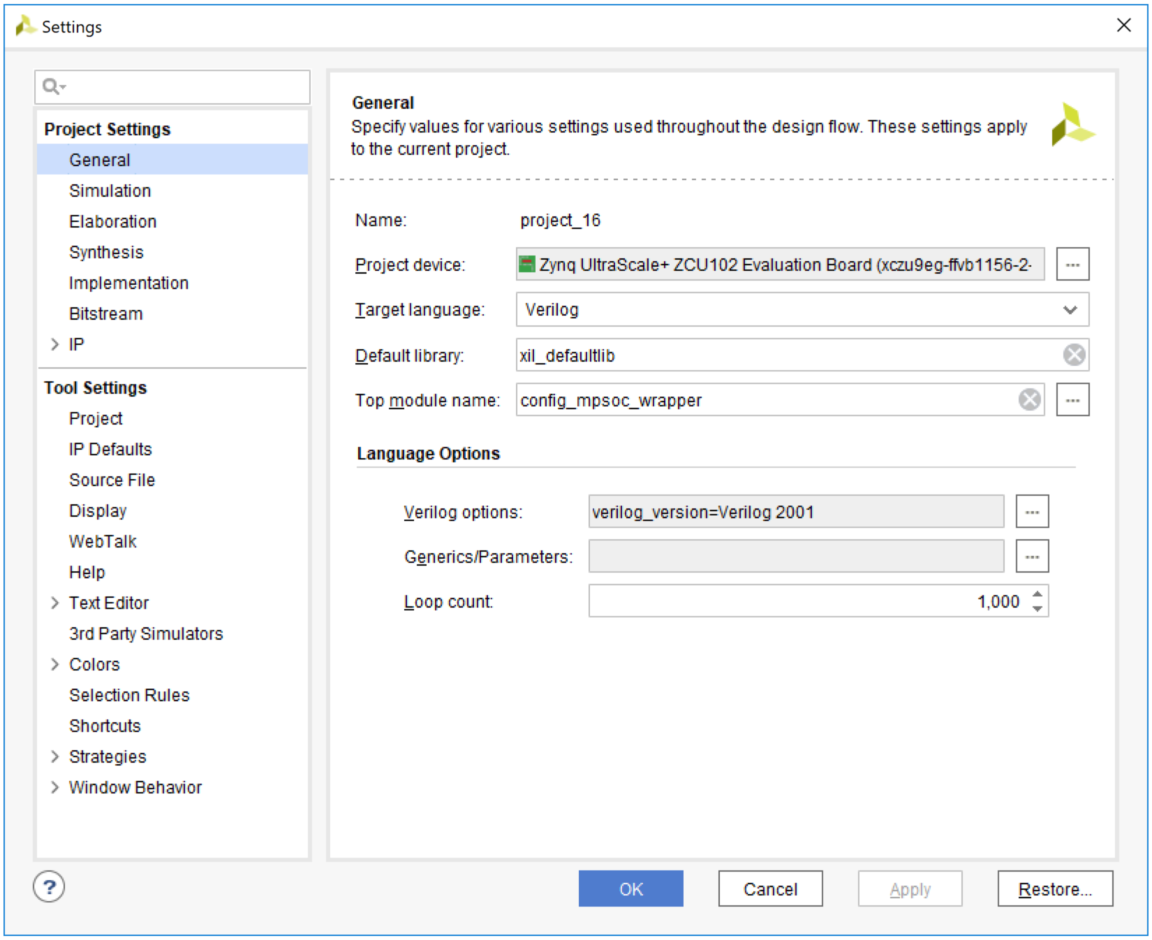

Figure 2-15: **General Settings**

Simulation Settings

The Simulation settings enable you to specify the simulation set, the simulation top module name, and a tabbed listing of compilation and simulation options. You can select an option to see a description at the bottom of the dialog box. For more information on the Simulation Settings, see the Using Simulation Settings section in *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 10].

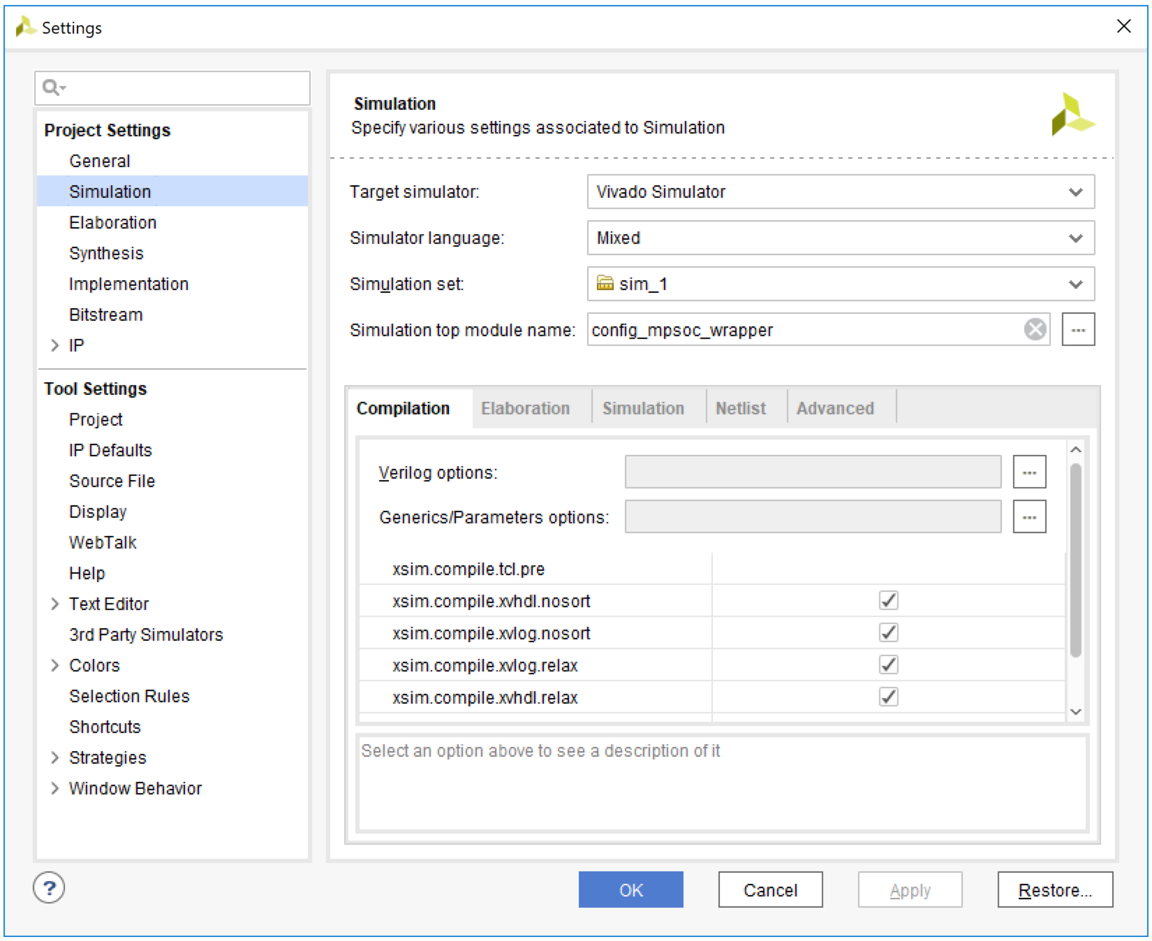

Figure 2-16: **Simulation Settings**

Elaboration Settings

When opening an elaborated design, as discussed in Chapter 4, Elaborating the RTL Design, there are two settings that can be enabled or disabled to change the elaboration of the RTL design, as shown in the following figure.

The Elaboration page allows you to set options for the elaborated netlist view. This view is available from the Flow Navigator in **RTL Analysis > Open Elaborated Design**.

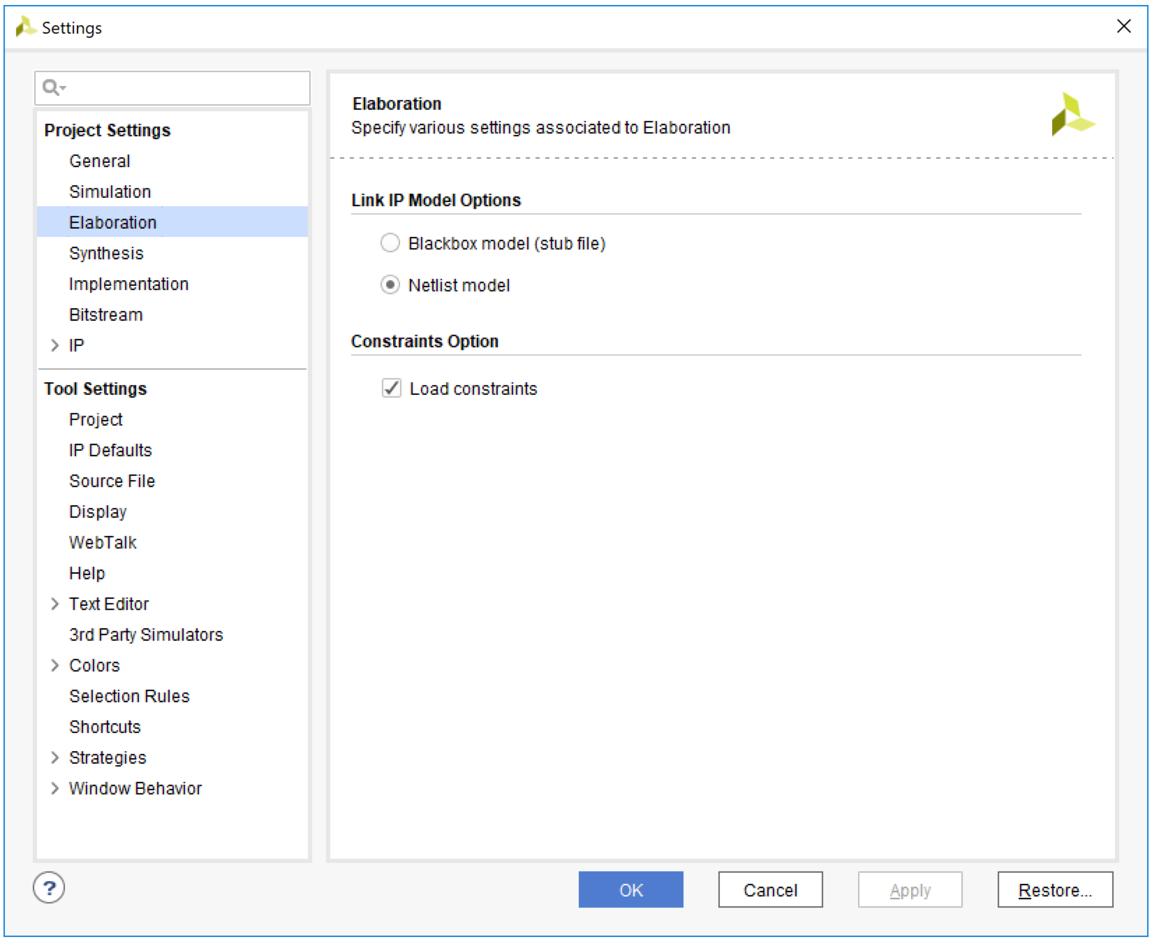

Figure 2-17: **Elaboration Settings**

- **Link IP Module Options:** The Blackbox model (stub file): Treats all IP which were synthesized out-of-context as a black box. The Netlist model: Uses the synthesized netlist for IP that were synthesized out-of-context.
- **Constraint Options:** Load constraints: Applies all active constraints to the elaborated design (timing and physical).

The following Tcl commands can be defined on the source fileset to enable the RTL elaboration settings:

```
set_property elab_link_dcps true [current_fileset]
set_property elab_load_timing_constraints true [current_fileset]
```

Note: Use `false` to disable these settings.

Synthesis Settings

The Synthesis Settings enable you to specify the constraints set, the synthesis strategy, the synthesis options, and what reports to generate. The options are defined by the selected synthesis strategy or synthesis reporting strategy, but you can override these with your own settings. You can select an option to see a description at the bottom of the dialog box. For more information on the Synthesis Settings, see the Using Simulation Settings section in the *Vivado Design Suite User Guide: Synthesis* (UG901) [Ref 15].

Note: You can pre-synthesize IP in your project, which decreases the synthesis runtime. For information on using this bottom-up synthesis flow, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 11].

TIP: *You can add Tcl scripts to be sourced before and after synthesis using the* `tcl.pre` *and* `tcl.post` *files. For more information, see the Vivado Design Suite User Guide: Using Tcl Scripting (UG894) [Ref 3].*

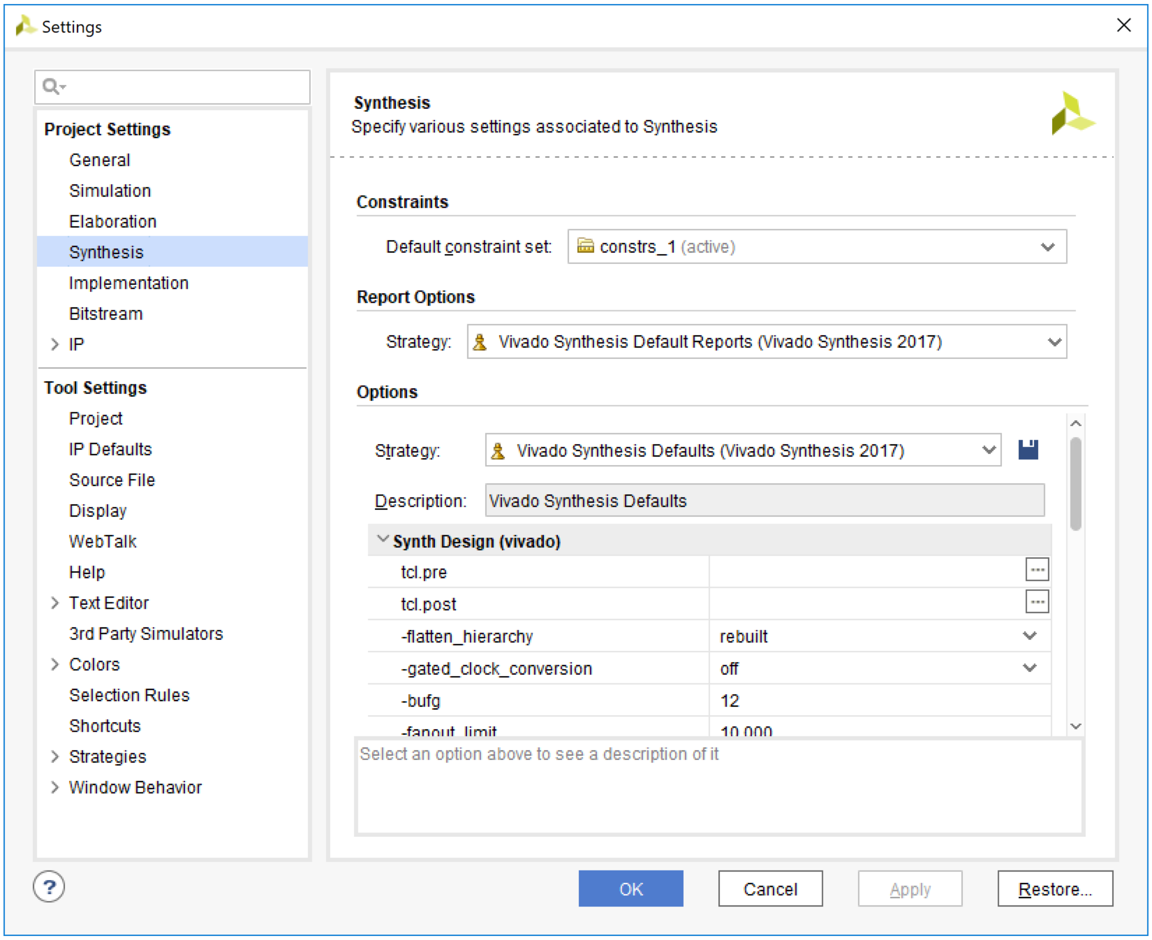

Figure 2-18: **Synthesis Settings**

Implementation Settings

The Implementation Settings enable you to specify the constraints set, the implementation strategy, the implementation options, and what reports to generate. The options are defined by the selected implementation strategy or implementation reporting strategy, but you can override these with your own settings. For example, you can use the options to run optional steps such as power optimization and physical synthesis. You can select an option to see a description at the bottom of the dialog box. For more information on the Implementation Settings, see the Customizing Implementation Strategies in the *Vivado Design Suite User Guide: Implementation* (UG904) [Ref 19].

TIP: *You can add Tcl scripts to be sourced before and after any stage of implementation using the* `tcl.pre` *and* `tcl.post` *files available at each stage. For more information, see the Vivado Design Suite User Guide: Using Tcl Scripting (UG894) [Ref 3].*

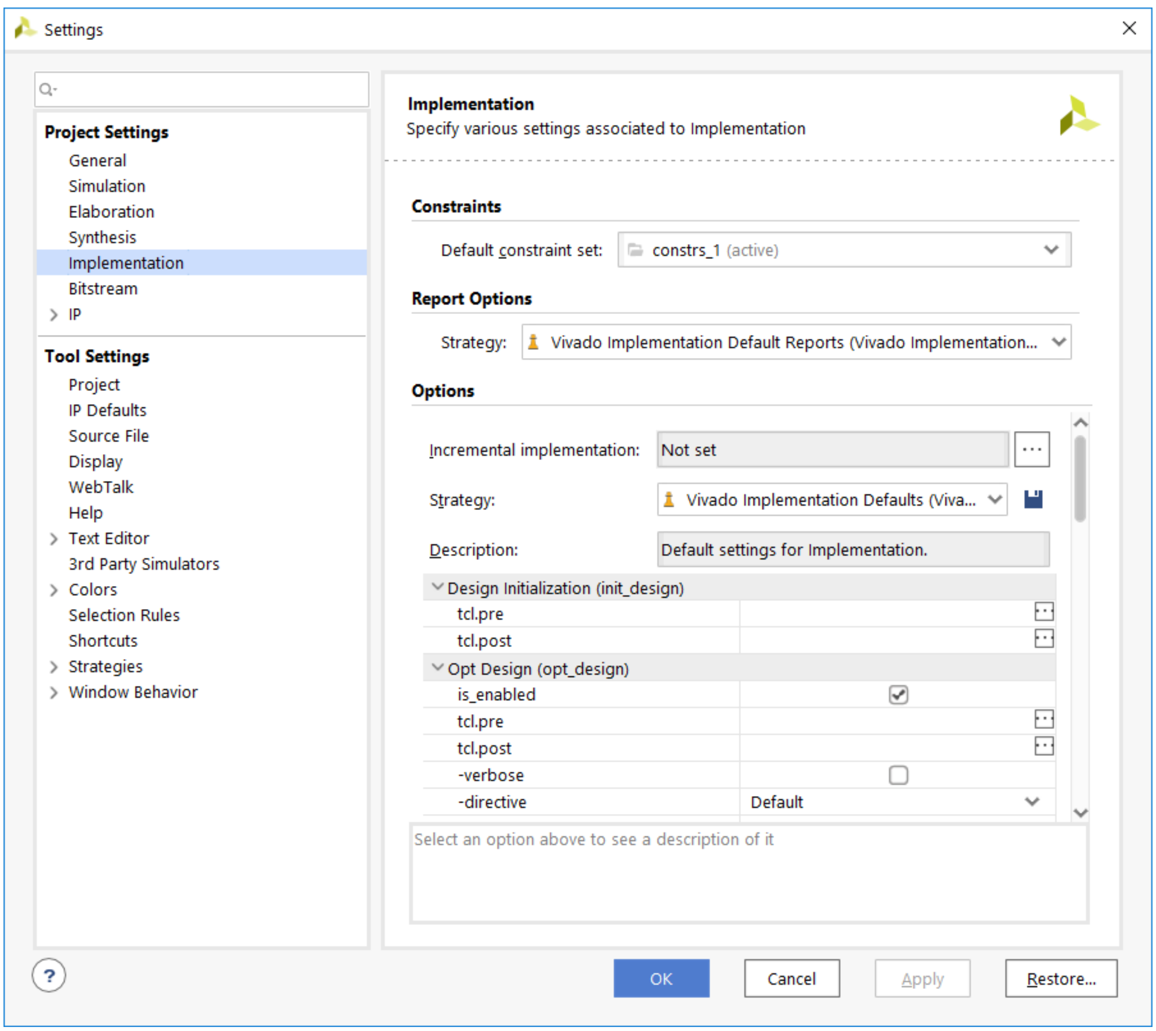

Figure 2-19: **Implementation Settings**

Bitstream Settings

The Bitstream Settings enable you to define options prior to generating the bitstream. You can select an option to see a description at the bottom of the dialog box. For more information on the Bitstream Settings, see the Changing the Bitstream File Format Settings section in the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 20].

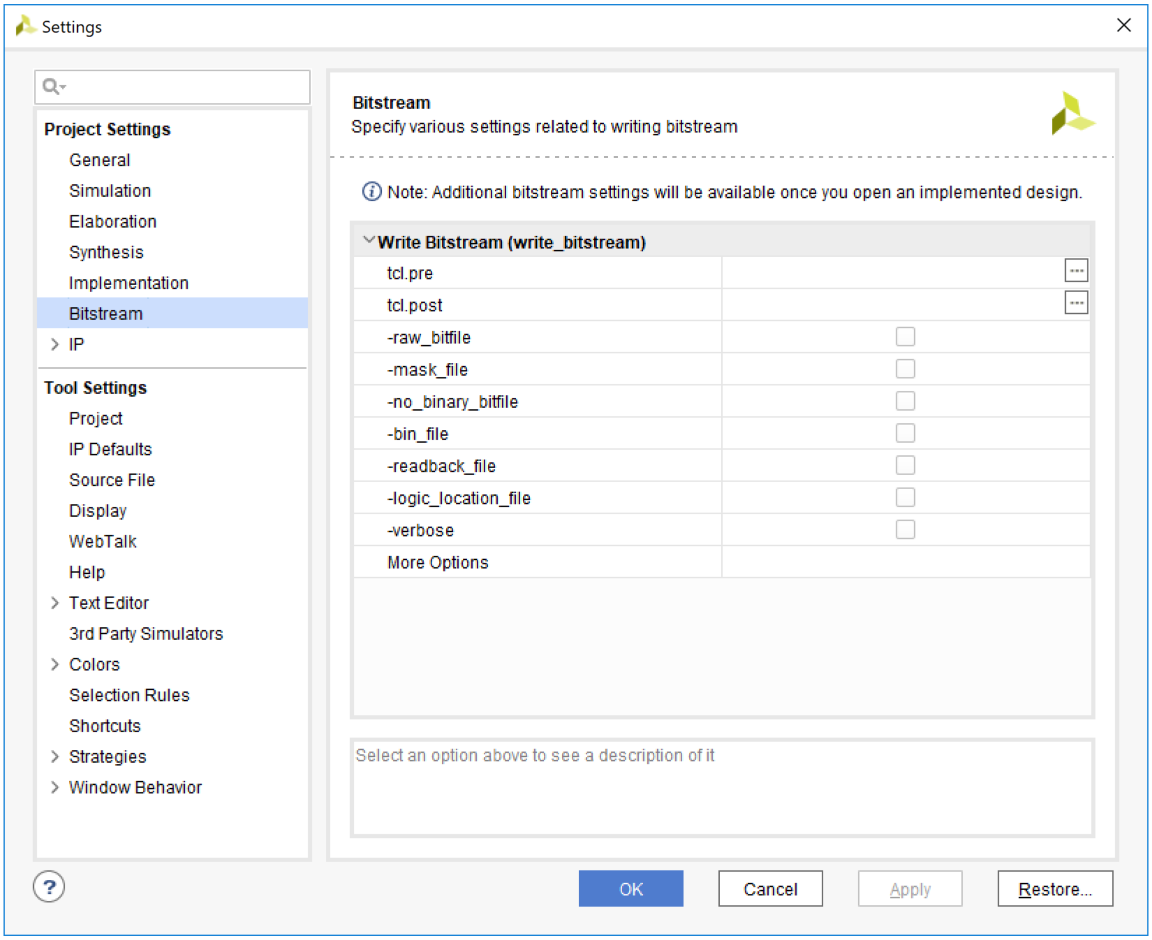

Figure 2-20: **Bitstream Settings**

IP Settings

The IP Settings include the sub-sections:

- **Repository**: Specifies directories to add to the IP repositories list. IP can either be packaged by you or acquired from a third-party supplier. After you click **Add**, to add a repository, you can see the IP within each repository.
- **Packager**: Sets default values for packaging new IP, including vendor, library, and taxonomy. This tab also allows you to set the default behavior when opening the IP Packager and allows you to specify file extensions to be filtered automatically.

  Note: If necessary, you can change the default values for packaging IP during the IP packaging process.

For more information on the IP Settings, see the Using IP Settings section in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 11].

The IP Settings and the Vivado IP catalog are only available when working with an RTL project or when using **Manage IP** from the Getting Started page. When using Manage IP, a subset of the IP settings is available unless a project is created.

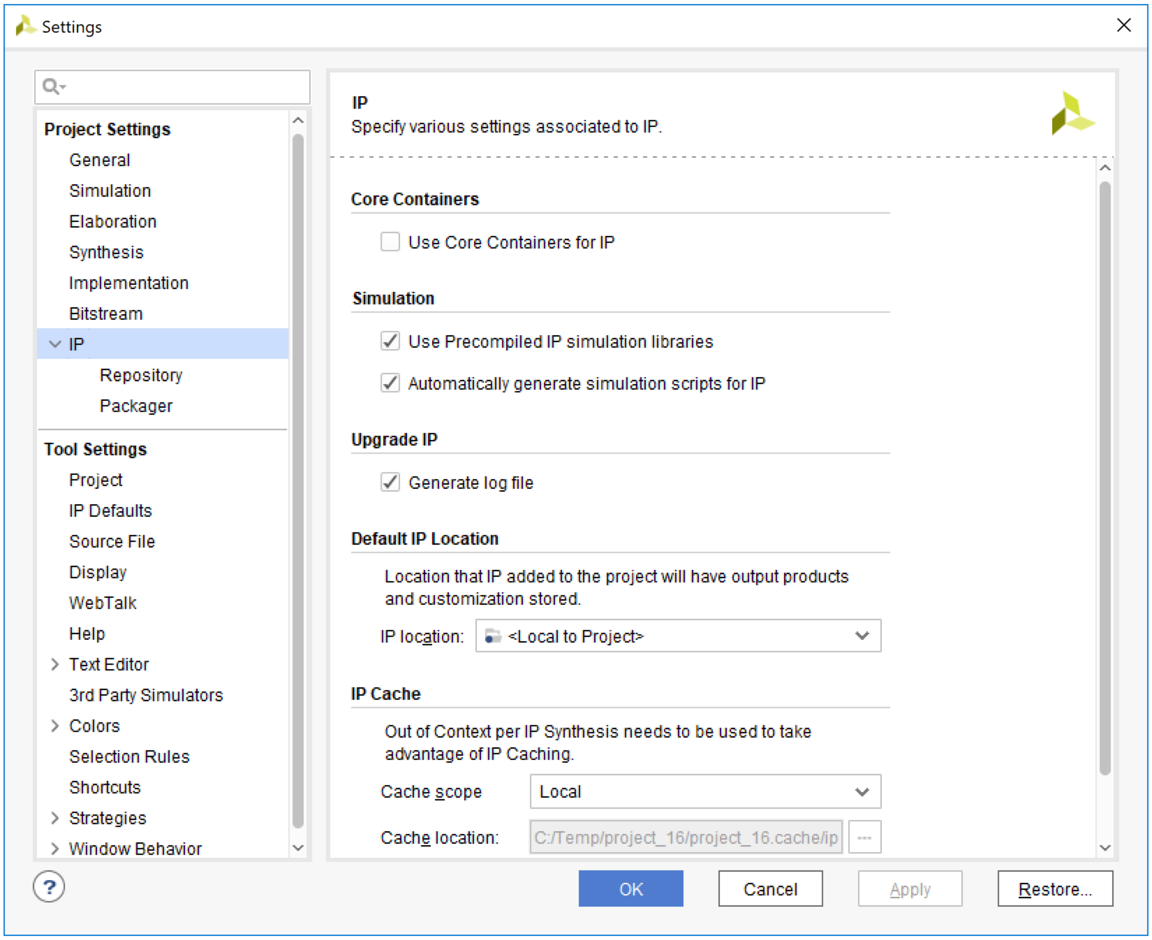

Figure 2-21: **IP Settings**

Tcl Command for Configuring Project Settings

Following is the associated Tcl command for configuring different properties for the project. The example shows how to configure the target language property for the project:

- **Tcl Command**: `set_property`
- **Tcl Command Example**: `set_property target_language Verilog [current_project]`

RECOMMENDED: *You can set multiple properties, including properties for the project or for synthesis or implementation runs. The best way to learn the property name and target is by performing the operation in the Vivado IDE and looking at the corresponding Tcl commands in the Tcl Console.*

Creating a Project Using a Tcl Script

You can use the `write_project_tcl` command to generate a tcl script that will re-create the current project. The script will keep the project settings and sources, but may not retain output products or design state.

As an alternative to creating a project in the Vivado IDE, you can create a project using a Tcl script. Most actions run in the Vivado IDE result in a Tcl command being executed. The Tcl commands appear in the Vivado IDE Tcl Console and are also captured in the `vivado.jou` and `vivado.log` files. The `vivado.jou` file contains just the commands, and the `vivado.log` file contains both commands and any returned messages. You can use these files to develop scripts for use with Project Mode. Refer to Output Files in Appendix A of the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 6] for information on where the `vivado.jou` and log files are written.

For more information on Tcl commands, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4].

Following is a sample script that creates a project, adds various sources, configures settings, launches synthesis and implementation runs, and creates a bitstream.

```
# Typical usage: vivado -mode tcl -source run_bft_project.tcl
# Create the project and directory structure
create_project -force project_bft_batch ./project_bft_batch -part xc7k70tfbg484-2
#
# Add various sources to the project
add_files {./Sources/hdl/FifoBuffer.v ./Sources/hdl/async_fifo.v \
./Sources/hdl/bft.vhdl}
add_files -fileset sim_1 ./Sources/hdl/bft_tb.v
add_files ./Sources/hdl/bftLib/
add_files -fileset constrs_1 ./Sources/bft_full.xdc
#
# Now import/copy the files into the project
import_files -force
#
# Set VHDL library property on some files
set_property library bftLib [get_files {*round_*.vhdl core_transform.vhdl \
bft_package.vhdl}]
#
# Update to set top and file compile order
update_compile_order -fileset sources_1
update_compile_order -fileset sim_1
#
# Launch Synthesis
launch_runs synth_1
wait_on_run synth_1
open_run synth_1 -name netlist_1
#
# Generate a timing and power reports and write to disk
# Can create custom reports as required
report_timing_summary -delay_type max -report_unconstrained -check_timing_verbose \
-max_paths 10 -input_pins -file syn_timing.rpt
report_power -file syn_power.rpt
#
# Launch Implementation
launch_runs impl_1 -to_step write_bitstream
wait_on_run impl_1
#
# Generate a timing and power reports and write to disk
# comment out the open_run for batch mode
open_run impl_1
report_timing_summary -delay_type min_max -report_unconstrained \
-check_timing_verbose -max_paths 10 -input_pins -file imp_timing.rpt
report_power -file imp_power.rpt
#
# Can open the graphical environment if visualization desired
# comment out the for batch mode
#start_gui
```

TIP: *You can break up a line in your Tcl script using the backslash (\) character at the end of a line to indicate the line continuation. The line that follows the backslash is processed as part of the preceding line.*

Working with Source Files

Overview

Source files include design sources, intellectual property (IP) sources added from the Xilinx® IP catalog, RTL design sources, digital signal processing (DSP) sources added from the System Generator tool, and IP subsystems, also known as block designs, created by the IP integrator feature of the Vivado® Design Suite. Source files also include simulation source files and constraints files that specify timing requirements for the design and physical constraints defining the Xilinx device resources used by the design. When working in Project Mode, you can create and add source files using the Vivado IDE, or using Tcl commands or scripts, and the Vivado IDE automatically manages your source files within the project. You can create and manage source files that are local to the current project, or remotely referenced from a library or separate directory. You can add Verilog, VHDL, and SystemVerilog source files to a project at any point in the design flow.

Note: For information on source file management when working with Zynq®-7000 devices, Zynq UltraScale+™ MPSoC devices, and MicroBlaze™ processors, see *Vivado Design Suite User Guide: Embedded Hardware Design* (UG898) [Ref 21].

When working in Non-Project Mode, you can create these source files using Tcl commands or scripts, but you must manually manage your source files. The majority of this chapter covers creating and managing sources in Project Mode. Working with Sources in Non-Project Mode covers creating and managing sources in Non-Project Mode. For more information on Project and Non-Project design flow modes, see this link in the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) [Ref 1].

When source files are added to the Vivado Design Suite, whether in Project Mode or Non-Project Mode, the tool maintains both a relative path and an absolute path to the file. When a design is opened, by default the Vivado Design Suite applies the relative path first to locate files and directories, then applies the absolute path if the file is not found. This feature is controlled through the PATH_MODE property, which defaults to **RelativeFirst**. You can change this behavior for specific design sources by setting the PATH_MODE property for those files. Refer to the PATH_MODE property in *Vivado Design Suite Properties Reference Guide* (UG912) [Ref 14] for more information.

Note: For information on Tcl commands associated with adding sources, see Tcl Commands for Adding Design Sources, Constraints Files, and Simulation Sources in Chapter 2.

Creating and Adding Design Sources

In the Vivado IDE, you can create and manage design source files, including HDL or netlist files. With a project open in the Vivado IDE, the Sources window displays the Design Sources, Constraints, and Simulation Sources that are the collection of files, or filesets, making up the current project.

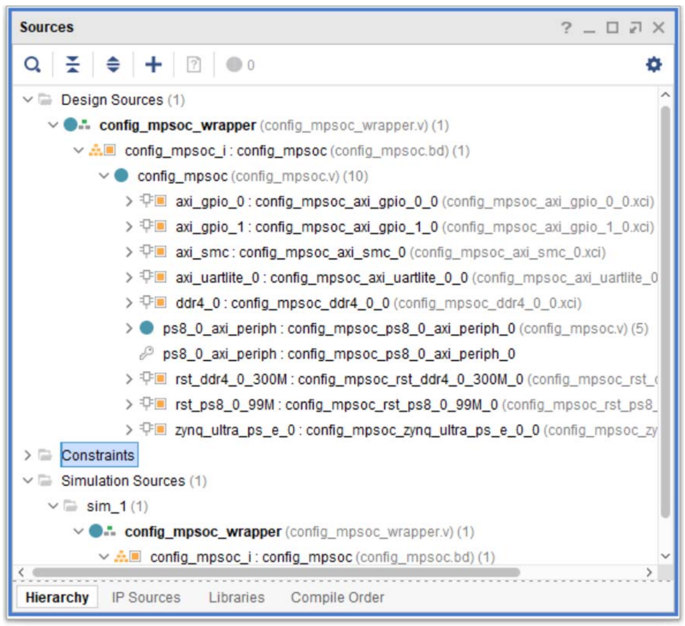

Figure 3-1: **Sources Window**

The Sources window provides different ways of viewing the source files associated with a project, including the following views:

- **Hierarchy** - displays the hierarchy of the design modules and instances, along with the source files that contain them. The Vivado IDE automatically detects the top of the design hierarchy, though you can manually change it as needed.
- **IP Sources** - displays all of the files defined by an IP core, a block design added from the Vivado IP integrator, or a DSP module added from System Generator.
- **Libraries** - displays design sources sorted into their various libraries.
- **Compile Order** - displays source files in the order in that they will be compiled, first to last, and shows the processing order for constraints. The Compile Order view can display the processing order used for synthesis, implementation, or simulation.

TIP: *For information on the icons used in the Sources window, see Using the Sources Window in the Vivado Design Suite User Guide: Using the Vivado IDE (UG893) [Ref 6].*

Creating New Source Files

1. To create new design sources to add to your project, select **File > Add Sources**.

   Note: Alternatively, you can select **Add Sources** from the right-click menu in the Sources window, or click **Add Sources** in the Flow Navigator.

2. In the Add Sources wizard select **Add or Create Design Sources**, and click **Next**.

3. In the Add or Create Design Sources page, press the **Create** button and select the **Create File** command from the sub-menu to create new source files.

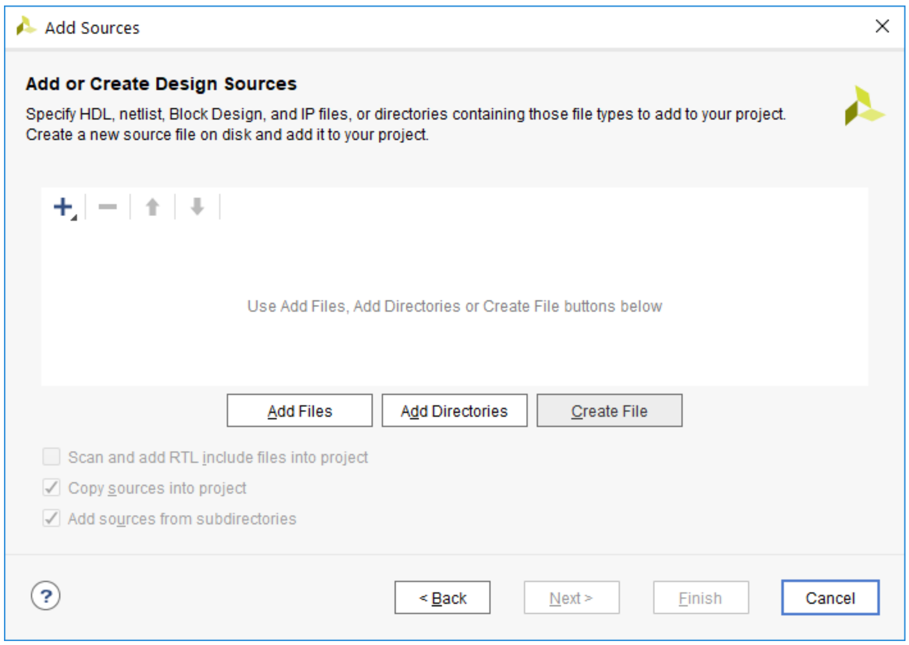

Figure 3-2: **Add Sources Wizard—Add or Create Design Sources Page**

4. In the Create Source File dialog box, set the following options, and click **OK**:

   - **File type**: Specifies one of the following file formats: Verilog file (.v extension), Verilog Header file (.vh extension), SystemVerilog file (.sv extension), VHDL file (.vhdl or .vhd extension), or Memory file (.mem).
   - **File name**: Specifies a name for the new HDL source file.
   - **File location**: Specifies a location in which to create the file.

   A placeholder for the file is added to the list of sources displayed in the Sources window. The file is not created until you click **Finish** in the Add Sources wizard.

TIP: *You can click **Create File** multiple times to define several new modules to add to the project.*

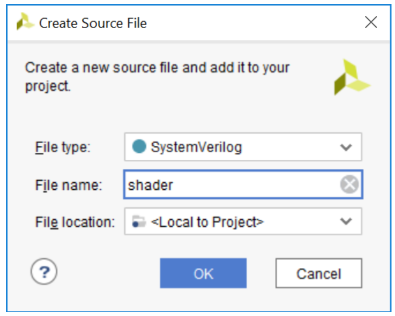

Figure 3-3: **Create Source File Dialog Box**

5. In the Add or Create Design Sources page, specify the appropriate library for the source file.

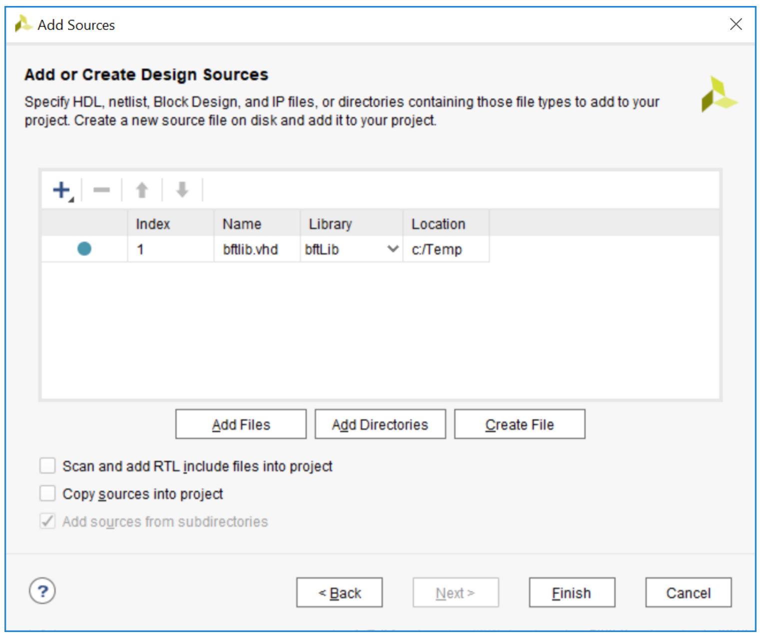

Figure 3-4: **Add Sources Wizard—Setting Library**

By default, all HDL sources are added to the `xil_defaultlib` library. In the Library column, you can reference an existing library name, or manually type a new library name to specify additional user VHDL libraries as needed.

6. Click **Finish** to create the new source files, and add them to the project.

   With a new source file created, the Vivado IDE opens the Define Modules dialog box to help you define the ports for the module or entity declaration.

7. In the Define Modules dialog box, you can define the module or entity for the Verilog, Verilog Header, SystemVerilog, or VHDL code using the following options:

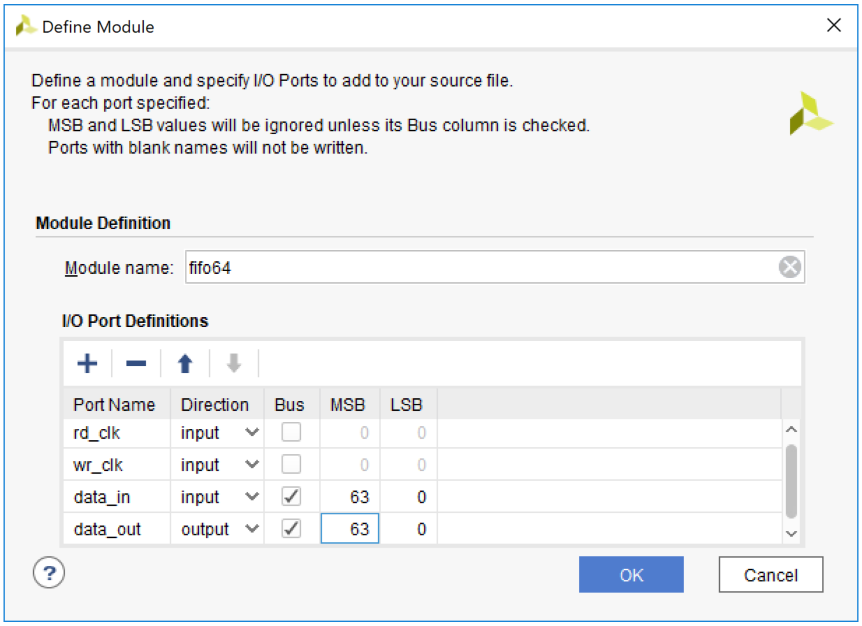

Figure 3-5: **Define Modules Dialog Box**

- **New Source Files**: This field appears if you created multiple files, letting you select the name of the module you want to define.
- **Entity name/Module name**: Specifies the name for the entity construct in the VHDL code or the module name in the Verilog or SystemVerilog code.

  Note: The name defaults to the file name but can be changed.
- **Architecture name**: Specifies the Architecture for VHDL source files. By default, the name is Behavioral.

  Note: This option does *not* appear when defining Verilog or SystemVerilog modules.
- **I/O Port Definitions**: Define the ports to be added to the module definition:
  - **Port Name**: Defines the name of the port to appear in the RTL code.
  - **Direction**: Specifies whether the port is an Input, Output, or Bidirectional port.
  - **Bus**: Specifies whether the port is a bus port. Define the width of the bus using the MSB and LSB options.
  - **MSB**: Defines the number of the most significant bit (MSB). This combines with the LSB field to determine the width of the bus being defined.
  - **LSB**: Defines the number of the least significant bit (LSB).

  Note: MSB and LSB are ignored if the port is not a bus port.

The Sources window lists the newly defined modules. To edit the new source files in the Vivado IDE Text Editor, double-click the file or select **Open File** from the right-click menu. See Using the Text Editor in *the Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) *[Ref 6]* for information on editing the file.

Adding Design Sources

1. Select **File > Add Sources**.

   Note: Alternatively, you can click **Add Sources** in the Flow Navigator, or select **Add Sources** from the right-click menu in the Sources window.

2. In the Add Sources wizard, select **Add or Create Design Sources**, and click **Next**.

3. In the Add or Create Design Sources page, set the following options, and click **Finish**.

   - **Add Files**: Opens a file browser so you can select files to add to the project. You can add the following file types to an RTL project: HDL, EDIF, NGC, BMM, ELF, DCP, and other file types.

     Note: In the Add Source Files dialog box, each file or directory is represented by an icon indicating it as a file or folder. A small red square indicates it is read only.

   - **Add Directories**: Opens a directory browser to add source files from the selected directories. Files in the specified directory with valid source file extensions are added to the project.

   - **Create File**: Opens the Create Source File dialog box in which you can create new VHDL, Verilog, Verilog header, or SystemVerilog files.

   - **Library**: Specify the RTL library for a single file, or the files in a directory, by selecting a library from the currently defined library names, or specify a new library name by typing in the Library text field.

     Note: This option applies to VHDL files only. By default, HDL sources are added to the `xil_defaultlib` library. You can create or reference additional user VHDL libraries as needed. For Verilog and SystemVerilog files, leave the library set to `xil_defaultlib`.

   - **Remove**: Removes the selected source files from the list of files to be added.

   - **Move Up / Move Down**: Moves the file or directory up/down in the list order. The order of the files affects the order of elaboration and compilation during downstream processes such as synthesis and simulation. See Specifying the Top Module and Reordering Source Files.

   - **Scan and Add RTL Include Files into Project**: Scans the added RTL files and adds any referenced Verilog ‘include files into the local project directory structure.

   - **Copy Sources into Project**: Copies files into the local project directory instead of referencing the original files.

     Note: If you added directories of source files using Add Directories, the directory structure is maintained when the files are copied locally into the project. For more information, see Using Remote Sources or Copying Sources into Project.

   - **Add Sources from Subdirectories**: Adds source files from the subdirectories of directories specified using the Add Directories option.

Specifying the Top Module and Reordering Source Files

By default, the Vivado Design Suite automatically determines the top-level of the design hierarchy and the order of elaboration, synthesis, and simulation for source files added to the project. this can be controlled through the use of the Hierarchy Update settings in the right-click menu of the Sources Window. Refer to **Hierarchy Update** in Sources Window Commands in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) *[Ref 6]* for more information.

The hierarchy of the design is displayed in the Hierarchy view of the Sources window. The compilation file order is displayed in the Compile Order view of the Sources window.

You can override the automatic determination of the top module by manually specifying the top of the design hierarchy. To specify the top module, select a module in the Sources window and select **Set as Top** from the right-click menu in the Hierarchy view of the Sources window.

Note: If the specified top module cannot be found in the design source files and the hierarchy update mode is set to automatic, the selected top is automatically reset to the best candidate.

When you change the top module, the Vivado IDE automatically reorders files in the Hierarchy and the Compile Order tabs of the Sources window according to the requirements of the new top module. Select **Refresh Hierarchy** from the right-click menu in the Sources window to reorder files based on changes to the source files.

You can override the automatic determination of the compile order using **Hierarchy Update** from the right-click menu command in the Sources window. You can specify the manual compile order mode by selecting **Hierarchy Update > Automatic Update, Manual Compile Order** or **Hierarchy Update > No Update, Manual Compile Order** in the right-click menu of the Sources window. In manual mode, you can manually order files according to your own requirements. To manually order source files, select a file and drag it up or down in the file list order in the Compile Order view of Sources window. Alternatively, after selecting the file, use **Move Up**, **Move Down**, **Move to Top**, or **Move to Bottom** from the Sources window right-click menu.

To see a full list of the compile or evaluation order for all sources, use the `report_compile_order` command in the Tcl Console. This command lists the order that files are compiled or evaluated for synthesis, implementation, and simulation. RTL compile order is listed for synthesis and simulation. Constraints evaluation order is listed for synthesis and implementation.

Enabling or Disabling Source Files

When you add or create source files, the source files are enabled in the Sources window by default. You can disable source files to prevent them from being elaborated, synthesized, or used in simulation. Enabling and disabling source files at different stages in the design lets you manage different design configurations in a single project.

- To disable source files, select the files in the Sources window, and select the **Disable File** right-click menu command.
- To enable disabled files, select the files in the Sources window, and select the **Enable File** right-click menu command.

Using Remote Sources or Copying Sources into Project

To provide project management flexibility, you can reference source files from a remote location or copy the source files into the local project directory. When you reference remote files, the Vivado IDE automatically detects changes to the referenced file, then prompts you to **Refresh your open designs** or to **Synthesize with the latest updates** made to the file.

If you move or archive the project, you can copy remote files into the project so that the files are contained within the project. To copy sources into the project, do one of the following:

- When you add sources to the project using the Add Sources command, you can copy the sources to the local project directory by selecting the **Copy Sources into Project** option.
- If you initially add the sources as remote sources, but later want to copy them into the project directory, use **Copy File into Project** or **Copy All Files into Project** in the right-click menu in the Sources window to copy some or all individual remote source files into the project directory.

Updating Local Source Files

When referencing remote sources, the Vivado IDE automatically detects source file changes. However, with source files that are copied to the local project, any changes to the original source file are not recognized. You must manually update local source files, if necessary.

You can update source files that are copied into the local project directory using either of the following methods:

- In the Sources window, select the file, and select **Replace File** from the right-click menu.

  A file browser opens with the original source file referenced. If the original location changed, you are required to browse to the location and select the file. Click **OK** to reload the original source file, and update the project file with any changes to the source file.

  Note: You can also specify a different file, and the Vivado IDE replaces the selected file with the new file. For instance, if the original file is `File_1.v`, and you select `File_2.v`, the original `File_1.v` is removed from the project and `File_2.v` is copied into the project.

- In the Sources window, select **Add Sources** from the right-click menu to add the newly updated source files to the project.

  The Vivado IDE imports the added file into the project. However, because there is already a local source with the same name, the Import Source Conflicts dialog box prompts you to resolve the conflict by overwriting the existing file or by not loading the newly added file. This happens only if the **Copy Sources into Project** box is checked in the Add Sources wizard; otherwise, the externally referenced file of the same name is added to the project.

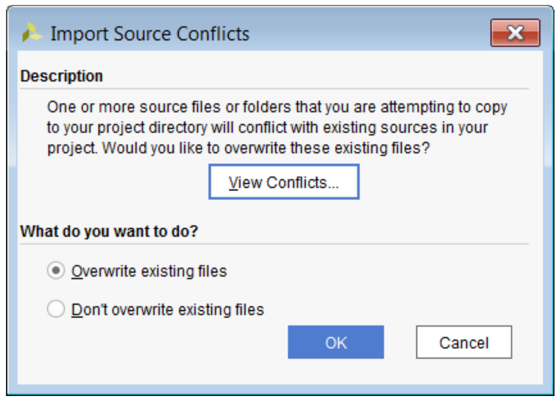

Figure 3-6: **Import Source Conflicts Dialog Box**

Working with IP Sources

Note: For more information on IP, including adding, packaging, simulating, and upgrading IP, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 11].

In the Vivado IDE, you can add and manage the following types of IP cores in an RTL project:

- Vivado Design Suite Xilinx Core Instance files (XCI)

  XCI format IP cores are native to the Vivado Design Suite and can be added to the design or project by customizing the IP core from the Xilinx IP catalog, or by using the **File > Add Sources** command to directly add the files. The XCI file stores the configuration and constraint options for an IP core that you specify, or customize, when you add the IP to a design.

IMPORTANT: *When using IP in either Project Mode or Non-Project Mode, always add the XCI file to the design; not a synthesized DCP file. The use of the XCI file ensures that the output products of the IP core that are needed by the tool are generated and used consistently throughout the design flow.*

- Vivado Design Suite Core Container files (XCIX)

  The Core Container feature simplifies working with revision control systems by providing a single file representation of an IP. The IP configuration and all generated output files are contained in one compressed binary file with an extension of XCIX. This extension is similar to the XCI file used for the IP customization file and works in a similar way. When adding or reading an IP, you can specify the XCIX file. For more information, see this link in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 11].

- User IP packaged with Vivado IP packager (XCI)

  The *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118) [Ref 12], describes how to package user-defined IP for use in the Xilinx IP catalog. User IP repositories can be added to the catalog using the IP_REPO_PATHS property, which defines the path for one or more directories containing third-party or user-defined IP. Refer to the IP_REPO_PATHS property in *Vivado Design Suite Properties Reference Guide* (UG912) [Ref 14] for more information.

- CORE Generator™ IP cores (XCO)

  Legacy IP from the CORE Generator tool are supported by the Vivado Design Suite. These legacy IP are locked when imported into a design, and require a corresponding NGC (netlist) file to support implementation of the IP into a design. Otherwise, if an XCI upgrade for the IP is available, you can right-click the IP core, and select **Upgrade IP** from the right-click menu.

- Third-party IP Netlists

  In some cases, third-party providers offer IP as synthesized NGC or EDIF netlists. You can load these files into a project or design as hierarchical design sources using the Add Sources command. For information, see Creating and Adding Design Sources.

Adding IP from the IP Catalog

TIP: *By default, the IP Catalog only displays IP cores that are compatible with, or supported by the target part (or board) for the current project. You can change the default setting to show all IP in the catalog by deselecting the* ***Hide*** *toolbar button in the Vivado IP catalog.*

You can add IP cores from the Xilinx IP Catalog, into your design hierarchy, by selecting IP from the catalog, and customizing the IP for your design. Customization involves modifying parameters or features of the IP core, and adding the IP source files into your design project. The IP Catalog also lists the interfaces that are available for use in IP Integrator.

1. To begin, select the **IP Catalog** in the Flow Navigator of the Vivado IDE. This opens the catalog as shown below.

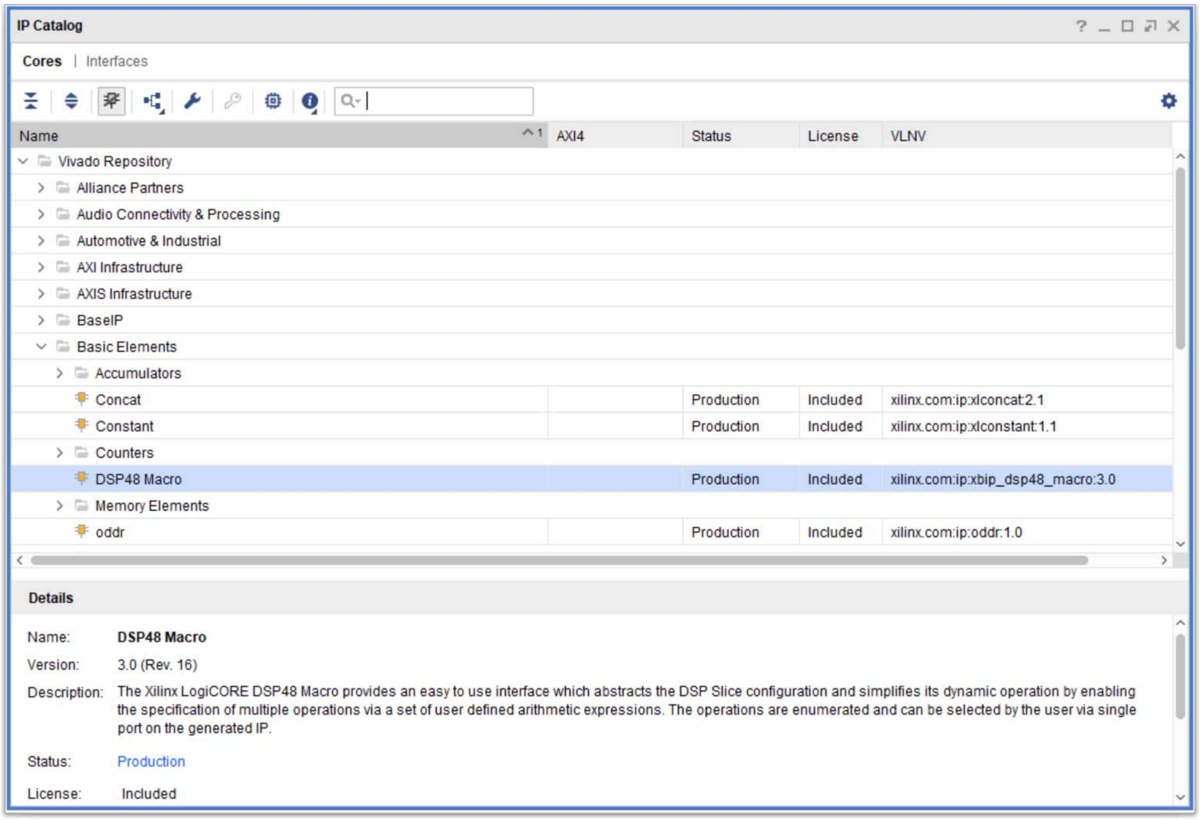

Figure 3-7: **Xilinx IP Catalog**

For information on filtering the IP cores displayed in the IP Catalog, and other details of working with the catalog, see this link in *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 11].

Select an IP from the IP Catalog and customize the IP for use in your design using one of the following methods:

- From the IP Catalog, select the IP and select the **Customize IP** command from the right-click menu.
- Double-click the selected IP to open the **Customize IP** dialog box for the selected IP core.

The Customize IP dialog box shows the various parameters and options available to customize the IP. The contents of the Customize IP dialog box varies, depending on the specific IP you select, and can include one or more tabs in which to enter values.

When you select **OK** to close the Customize IP dialog box, and confirm the settings you have specified, the IP source files, including the HDL definition of the IP module, are added to your design project and displayed in the IP Sources tab of the Sources window.

With the IP added to your design, you must generate any files required to support the IP in your design, such as the instantiation template, XDC constraints, and simulation sources. These files are referred to collectively as output products. See Generating Output Products for IP Cores for more information.

Adding Existing IP Files

As an alternative to adding and customizing IP from the Xilinx IP Catalog, you can directly add XCI or XCIX files into your project or design. This process is different from customizing IP from the catalog in the following ways:

- The XCI or XCIX file may be an earlier version, or fully customized version of the same or similar IP found in the Xilinx IP Catalog.
- The XCI or XCIX file may include the necessary files, or output products, to support the IP in the design flow. This can include the instantiation template, simulation files, and netlists or design checkpoints (DCPs) needed to support the IP through implementation. The Vivado Design Suite adds these files when the XCI or XCIX file is added to the design.
- If the IP is an earlier version of an IP found in the catalog, you can upgrade it to the latest version from the IP catalog.
- If the IP is an earlier version and includes the needed output products to support the IP in the design, it can be used in its current form, and the IP will be locked to prevent further customization.

To add existing XCI or XCIX files directly into your design or project, select **File > Add Sources**. See Adding Design Sources for detailed information.

Note: Alternatively, select **Add Sources** from the right-click menu, or from the Flow Navigator.

The added IP cores display separately in the IP Sources tab of the Sources window, as well as with other source files in the Hierarchy, Libraries, and Compile Order views. You can select these core files in the Sources window to see the files that make up the core, and to view the properties in the Source File Properties window.

If the XCI or XCIX file included any needed support files, referred to collectively as output products, those files are added when the design source is added to the design. If the XCI or XCIX file does not include these associated files, you must generate the output products required to support the IP in your design, such as the instantiation template, XDC constraints, and simulation sources. See Generating Output Products for IP Cores for more information.

You can run **Reports > Report IP Status** to and review the state of the newly added IP. The IP may be in a locked state if they were generated with an older version of the Vivado design suite, of if they were configured to a different part.

Tcl Command for Reporting IP Status

Following is the associated Tcl command:

- **Tcl Command**: `report_ip_status`

Generating Output Products for IP Cores

The IP core includes, or requires, specific files to support the IP in the overall design flow. These include files such as a Verilog or VHDL instantiation template to facilitate integrating the IP module into your design, design constraints files (XDC) that are included to provide timing or physical constraints for the IP core, and synthesized netlists or design checkpoints to support the IP in the design hierarchy. Collectively these files are referred to as output products. Some of these files are included with the packaged IP in the Xilinx IP Catalog, and some are generated for the customized IP in the current design.

When an IP is customized from the IP Catalog, the Generate Output Products dialog box is opened. However, you can also open this dialog box at any time by right-clicking the IP in the Sources window and selecting the **Generate Output Products** command.

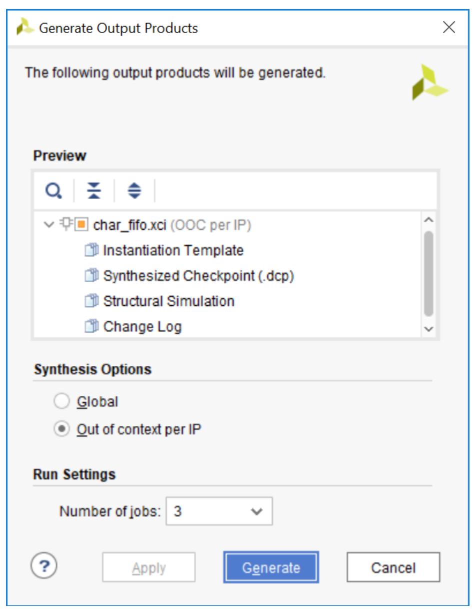

Figure 3-8: **Generate IP Output Products**

By default, synthesized design checkpoint (DCP) files are generated automatically for IP that supports the out-of-context flow. However, you can disable DCP file generation when creating output products by changing to Synthesis Options to Global synthesis. For more information, on the using the Out-of-Context flow see this link in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 11].

With the output products required by the IP core added to your design project, you must now instantiate the IP into your design hierarchy. This involves integrating the IP module or entity into the design as described in Instantiating IP Into the Design.

Instantiating IP Into the Design

An instantiation template is created when you customize and IP and add it to your design or project, regardless of whether you generated output products. The instantiation template provides a Verilog or VHDL instance declaration (`.veo` or `.vho`) that you can copy and paste into your RTL design hierarchy.

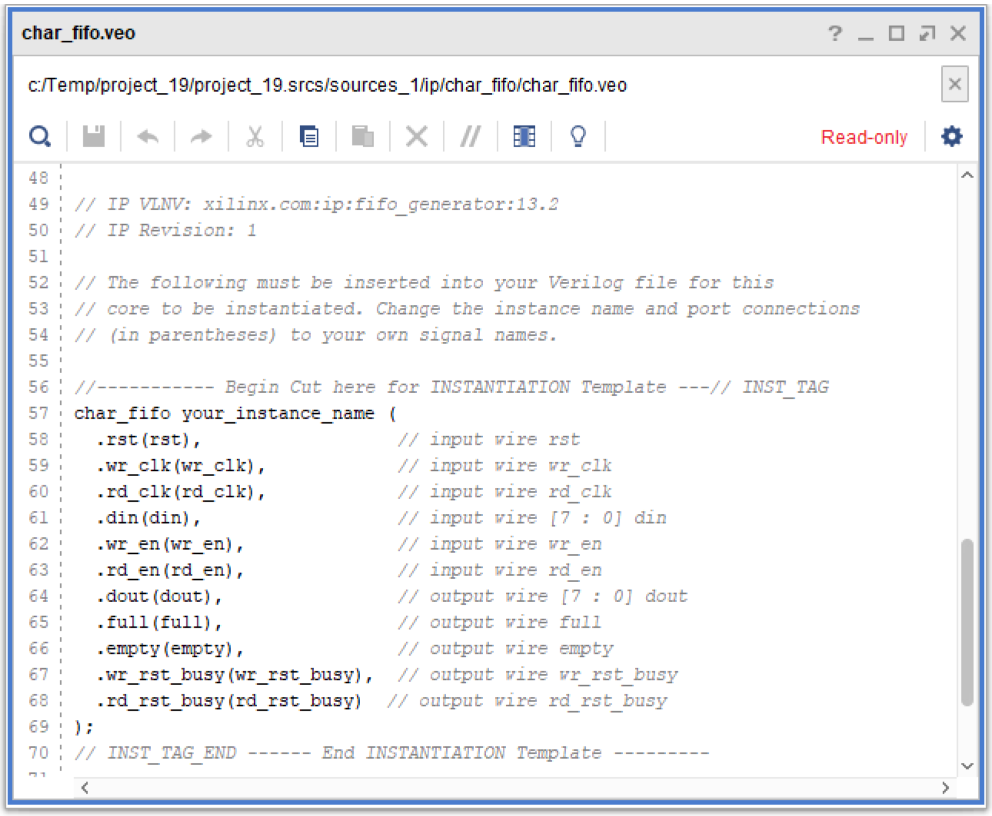

Figure 3-9: **Editing the Instantiation Template**

1. Open the instantiation template in the Vivado IDE Text Editor.
2. Select the instance declaration in the template file, and copy and paste it into the appropriate source file.
3. Edit the signal names on the port definitions to connect to the appropriate signal names in your design.
4. You can repeat this process to create multiple instances of the IP core in your design.

For more information see Instantiating an IP in the *Vivado Design Suite User Guide: Designing with IP (UG896)* [Ref 11].

After you instantiate the IP in your design, the IP core shows in the Hierarchy tab of the Sources window as integrated into the design. The IP can now be synthesized or simulated as part of the overall design, or separately in the out-of-context flow.

Working with IP Integrator Sources

In the Vivado Design Suite, you can add and manage IP subsystem block designs (`.bd`) in an RTL project or design. Using the Vivado IP integrator, you can create an IP subsystem block design. The IP integrator enables you to create complex system designs by instantiating and interconnecting multiple IP cores from the Vivado IP catalog. You can create designs interactively through the IP integrator canvas in the Vivado IDE or programmatically with Tcl commands. For information on using the IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 13].

IMPORTANT: *The Vivado IP integrator is the replacement for Xilinx Platform Studio (XPS) for new embedded processor designs, including designs targeting Zynq®-7000 devices and MicroBlaze™ processors. To move existing XPS designs into the Vivado IP integrator see* Migrating from XPS to IP Integrator *in the ISE to Vivado Design Suite Migration Guide (UG911) [Ref 8].*

Creating a New Block Design

You can create a block design in the context of an open RTL project. To create a new block design source, and automatically add it to the current project, use the following steps:

1. In the Flow Navigator, expand **IP Integrator**.
2. Select **Create Block Design**.

   This opens the Vivado IP integrator design canvas, letting you add and connect IP in the block design. Refer to this link in *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 13] for details of creating a new block design.

3. When you save the new block design, it is automatically added to the current project.

   You can also create the block design outside of the current project, to create a repository of block designs that can be reused and added to many different projects. For more information on creating a block design outside the current project, refer to this link in *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 13].

Note: For information on creating embedded processor block designs, using either MicroBlaze processors or targeting Zynq-7000 devices, see the *Vivado Design Suite User Guide: Embedded Processor Hardware Design* (UG898) [Ref 21].

Adding Existing Block Design Sources

To add a block design source that was created outside of the project, and may reside in a repository of block designs, you can use the Add Sources command as you would for any other source. See Adding Design Sources for detailed information.

When an existing block design is added to the current project, the project *may* target a different Xilinx part than the part originally targeted by the block design. This will result in all of the IP used in the block design being locked, and needing to be updated. Run Tools `-report_ip_status` to determine the state of the IP imported with the Creating a New Block Design.

Tcl Command for Adding Existing Block Design Sources

Following is the associated Tcl command:

- **Tcl Command**: `report_ip_status`
- **Tcl Command Example**: `report_ip_status -name ip_status`

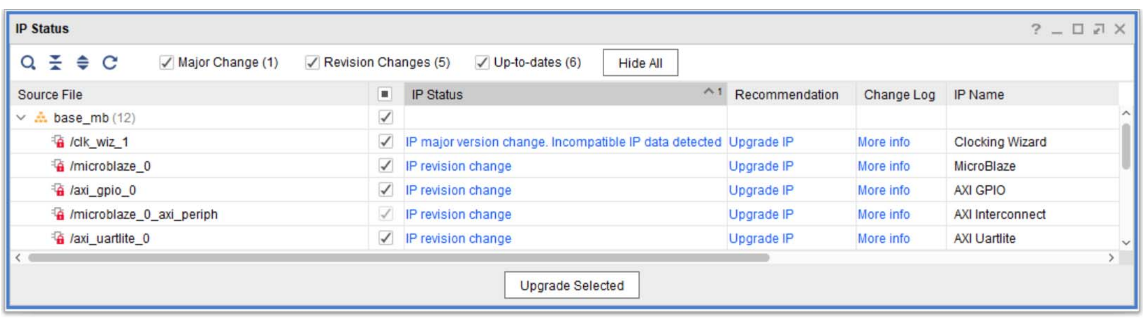

Figure 3-10: **Report IP Status**

IMPORTANT: *Locked IP are reported with the following critical warning message when you try to generate the output products for the block design:*

`[BD 41-1336] One or more IPs are locked in this block design. Please run report_ip_status for more details and recommendations on how to fix this issue.`

---

1. To unlock the IP and the block design, right-click the block design in the IP Sources tab of the Sources window and select the **Report IP Status** command.

   The IP Status report will show the IP part changes needed to unlock the block design.

2. In the IP Status report window, select the **Upgrade Selected** command to upgrade the IP used in the block design to target the new part used in the current project.

   With the block design added to the current project, you must generate the output products required by the Vivado Design Suite to support the block design throughout the design flow.

Generating Output Products for Block Designs

Once the block design is complete and the design is validated, output products must be generated to support the block design throughout the design flow. These output products include files such as a Verilog or VHDL instantiation template, or HDL wrapper files, to facilitate integrating the block design into the current project, design constraints files (XDC) that are included to provide timing or physical constraints for the block design, and synthesized netlists or design checkpoints to support the block design.

The output products for a block design are generated in the target language of the current project. If the source files for a particular IP used in the block design cannot be generated in the target language, a message is returned to the Tcl Console, and the output products will be generated in the available or supported language.

1. To generate output products, right-click on the block design and select the **Generate Output Products** command, or select **Generate Block Design** from the Flow Navigator.

   The Generate Output Products dialog box is displayed as shown below.

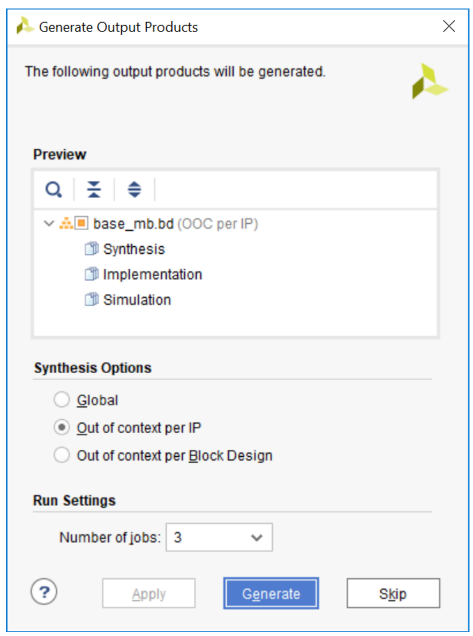

Figure 3-11: **Generate Output Products—Block Design**

Generating the output products also generates the top level netlist of the block design. The netlist is generated in either VHDL or Verilog depending on the target language settings for the current project.

By default, synthesized design checkpoint (DCP) files are created for each IP inside the Block Design to speed up synthesis times. You can change the synthesis mode by selecting the Out of context per Block Design radio button on the Generate Output Products dialog box. For more information, on the using the Out-of-Context flow see this link in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 13].

Once the Block Design is created and generated you need to then instantiate it into your design by selecting either the Block Design, **RMB > Create Wrapper** or by instantiating the Block Design in your own RTL. During creation the dialog box will appear.

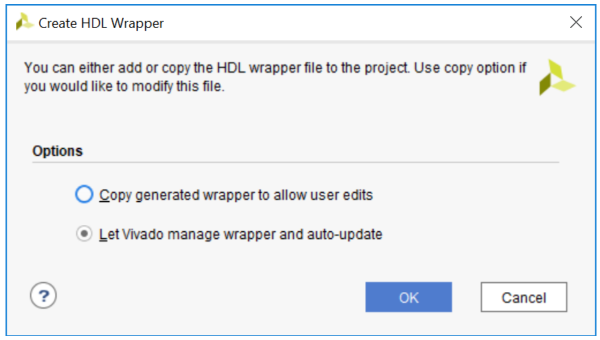

Figure 3-12: **Create HDL Wrapper**

If you want to modify the wrapper, select the **Copy Generated Wrapper** to allow user edits, otherwise select **Let Vivado Manage Wrapper** to auto-update.

Instantiating Block Designs into the Current Project

An IP integrator block design can be instantiated into the hierarchy of an HDL design, or it can be defined as the top-level of the design hierarchy.

To integrate the block design into an existing design hierarchy, open the HDL wrapper for the block design. The HDL wrapper, or instantiation template for the block design is created when you generate the output products. The HDL wrapper provides a Verilog module declaration, or VHDL entity declaration for the block design, and creates an instance of the block design module in the wrapper. You can edit the instance definition in the HDL wrapper and cut and paste it into the design hierarchy as needed.

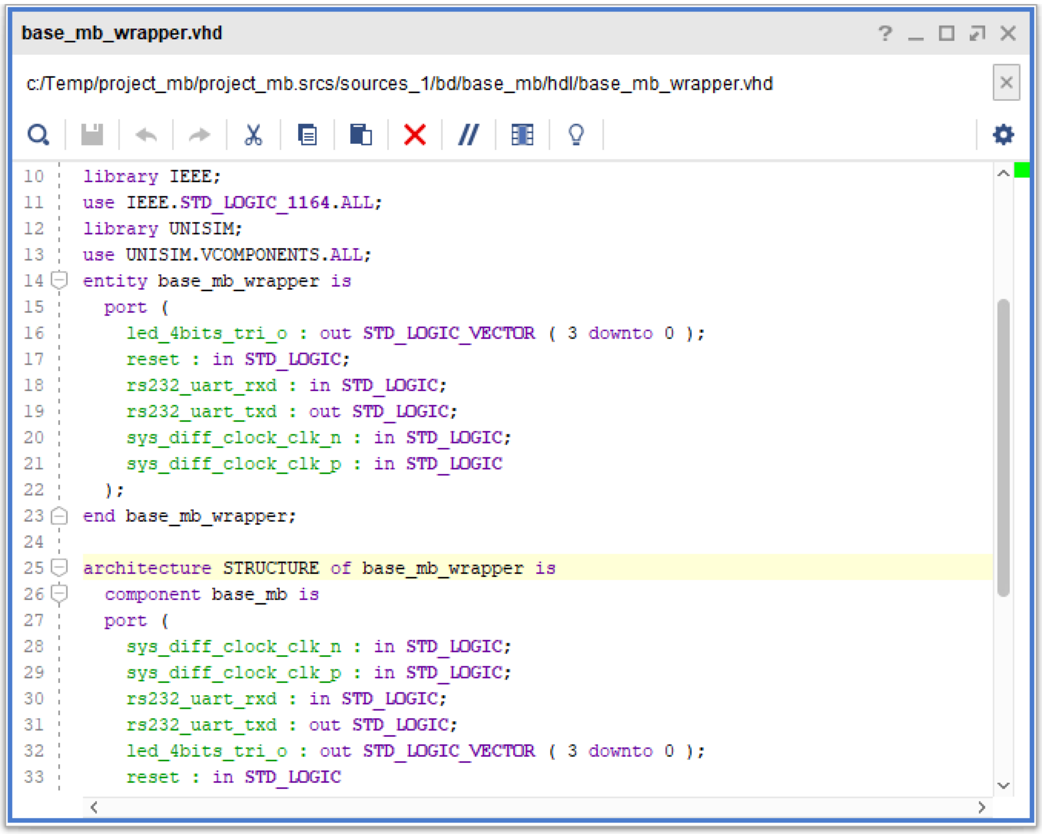

Figure 3-13: **Editing the Block Design Wrapper**

The HDL wrapper can also be used to define the block design as the top-level of the design. For more information see this link in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 13].

Working with Vivado HLS Sources

The Vivado High-Level Synthesis (HLS) tool transforms a C specification into a register transfer level (RTL) implementation that you can synthesize into a Xilinx device. You can write C specifications in C, C++, or SystemC, and the Xilinx device provides a massively parallel architecture with benefits in performance, cost, and power over traditional processors.

The outputs from Vivado HLS include RTL implementation files in hardware description language (HDL) formats that can be synthesized in Vivado synthesis or packaged as an IP block for use from the IP catalog. For more information, see *Vivado Design Suite User Guide: High-Level Synthesis* (UG902) [Ref 16].

Working with Model Composer Sources

Model Composer is a model-based design tool that enables rapid design exploration and accelerates the path to production for Xilinx devices through automatic code generation. Model Composer is built as a Xilinx toolbox that fits into the MathWorks Simulink® software, which is an add-on product to the MATLAB® software that provides an interactive, graphical environment for modeling, simulating, analyzing and verifying system-level designs.

You can express your algorithms in Model Composer using blocks from the Model Composer library as well as user-imported custom blocks. Model Composer transforms your algorithmic specifications to packaged IP blocks using automatic optimizations and leveraging the high-level synthesis technology of Vivado HLS. Add these packaged IP into designs using the Vivado Design Suite or using IP Integrator to integrate the IP into a platform (for example, a platform with a Zynq device, DDR3 DRAM, and a software stack running on the Arm processor). For more information, see *Model Composer User Guide* (UG1262) [Ref 18].

Working with System Generator Sources

Xilinx System Generator for DSP is a design tool from Xilinx that combines RTL source files, Simulink and MATLAB software models, and C/C++ components of a DSP system into a single simulation and implementation environment. For more information on working with System Generator refer to the *System Generator for DSP User Guide* (UG640) [Ref 22].

A System Generator design is often a sub-design that is incorporated into a larger HDL design. The recommended flow is to package the DSP module as an IP core in the Vivado Design Suite, to be added to the Xilinx IP catalog and integrated into any level of the design hierarchy as a sub-module as described in Working with IP Sources,or imported into the top-level of the design. This lets the Vivado IDE manage the project for the FPGA design, while handling the DSP module as an IP source that is developed and managed within System Generator. For more information see IP Catalog Compilation in the *Vivado Design Suite User Guide: Model-Based DSP Design Using System Generator (UG897)* [Ref 22].

Editing Source Files

The Vivado IDE provides a Text Editor in which to create or modify RTL, XDC, Tcl and other text sources. The Text Editor is context-sensitive when editing Verilog, VHDL, XDC, and Tcl files, and uses color-coding to distinguish keywords and constructs. It is a configurable, integrated text editor that supports syntax highlighting and on-the-fly checking, assistance with errors and warnings, code folding, code completion, and file comparison. See Using the Text Editor in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 6] for information on specific commands in the text editor.

You can open multiple files simultaneously, and click the tab for each open file to switch between files. In the tab for the open file, the Vivado IDE appends an asterisk (*) to the file name for modified files that need to be saved. To save the file, use one of the following methods:

- Select **File > Text Editor > Save File**.
- In the Vivado IDE Text Editor, select **Save File** from the right-click menu.
- In the Vivado IDE Text Editor, use the **Save File** toolbar button.

Note: If you attempt to close a file with unsaved changes, the Vivado IDE prompts you to save the changes.

TIP: *Use the **Save As** command to save the source file to a new location.*

Using the Find/Replace in Files Commands

When editing design source files, you may need to find specific objects or instances of objects. You can use **Find** or **Find in Files** to search for any given text string in an open source file or a selected set of source files, or **Replace in Files** to find and replace text strings. You can perform the following actions:

- Enter any text string, including wildcards (*, ?, #, +), or regular expression as search criteria.
- Use the filtering options to search source files, constraint files, and report files.

Using HDL Language Templates

The Vivado IDE provides templates for many Verilog, VHDL, and XDC structures, including Xilinx Parameterized Macros (XPMs) and library primitives. To view the templates:

1. In the Vivado IDE Text Editor, select the **Language Templates** toolbar button.
2. Select **Tools > Language Templates**.

   The Language Templates window appears with folders for Verilog, VHDL, SystemVerilog, XDC, and Debug.

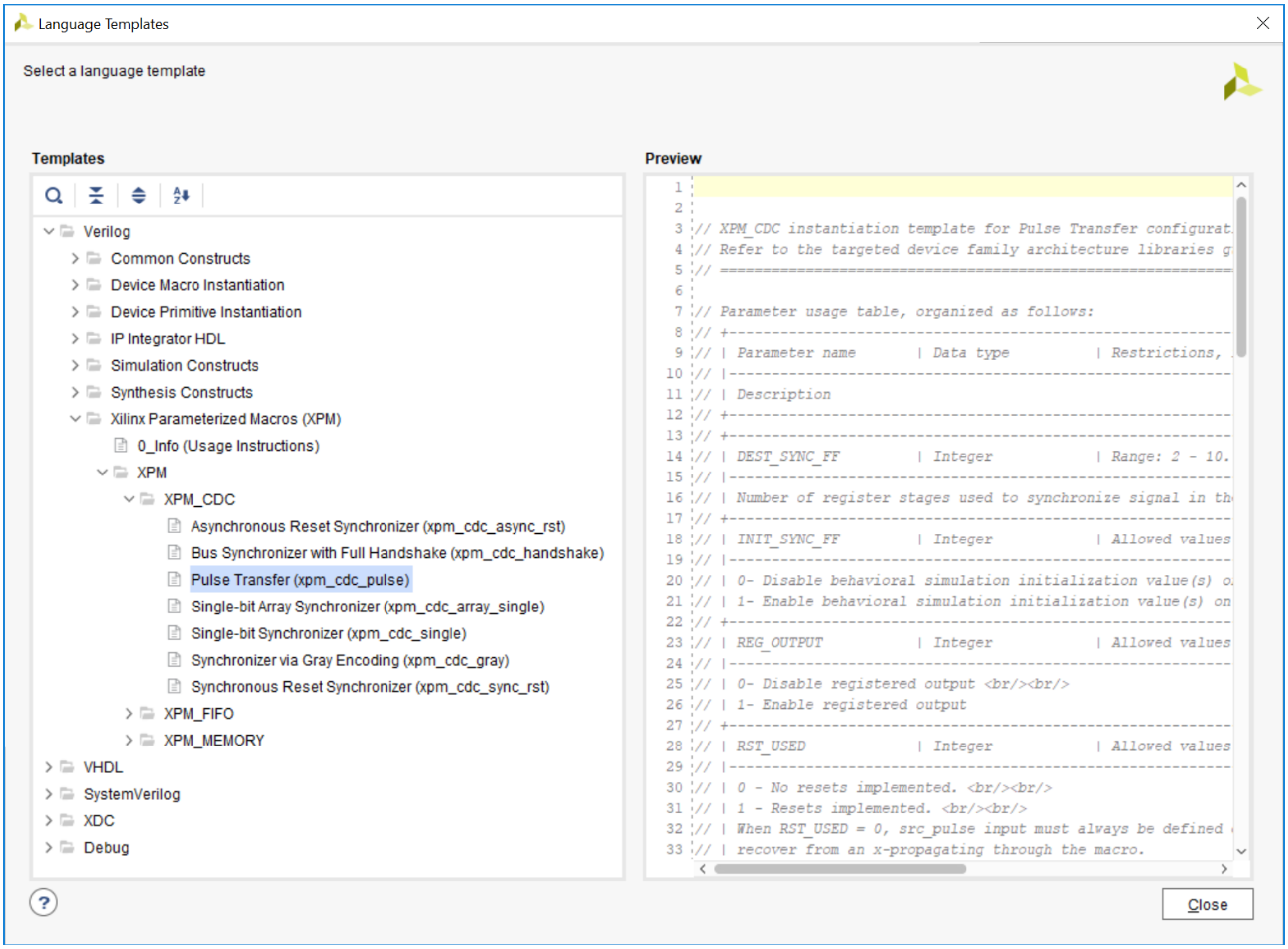

Figure 3-14: **Language Templates Window**

When a template is selected, you can use the **Insert Template** command from the popup menu in the Text Editor. Selecting this command copies the currently selected template text into the file being edited, at the current location of the cursor. Alternatively, you can highlight, and then copy and paste the desired text from the Language Templates window. For supported commands, see this link in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 6].

Using Xilinx Parameterized Macros

XPMs are simple customizable solutions for common use cases in an HDL flow, such as RAM or ROM, clock domain crossings, and FIFOs. XPMs are SystemVerilog HDL code delivered with the Vivado Design Suite, and can be found in the `./data/ip/xpm` folder of the software installation. They can be thought of as parameterized IP, with default values for parameters that can be changed to meet design requirements.

The types of XPMs include:

- XPM_MEMORY with various RAM and ROM memory structures
- XPM_CDC with various safe Clock Domain Crossing (CDC) logic implementations
- XPM_FIFO for synchronous and asynchronous FIFO structures

Enabling Xilinx Parameterized Macros

When using the Project Mode, the Vivado tool will parse the files added to the project and will automatically recognize the XPMs. However, when using XPMs in the Non-Project mode you must issue the `auto_detect_xpm` command prior to reading or importing source files.

Using XPMs

You can use any XPM language templates in your design. The parameters available for a specific XPM are explained in the instantiation template displayed in the Language Templates window.

Select and copy the contents of the instantiation template and paste it into your own source file, or use the **Insert Template** command from the popup menu in the Text Editor. You do not need to copy the comments for the instantiation template into your design source file.

You can change the instance name and wire ports as needed to fit the XPM instance into your design, and modify parameters/generics according to the documentation provided as comments in the language template.

IMPORTANT: *Be sure to read and comply with all code comments in the XPM language template to properly use the XPM.*

Figure 3-15 shows an example of an XPM_CDC instance.

```
// xpm_cdc_single: Clock Domain Crossing Single-bit Synchronizer
// Xilinx Parameterized Macro, Version 2017.3
xpm_cdc_single #(

  //Common module parameters
  .DEST_SYNC_FF   (4), // integer; range: 2-10
  .INIT_SYNC_FF   (0), // integer; 0=disable simulation init values, 1=enable simulation init values
  .SIM_ASSERT_CHK (0), // integer; 0=disable simulation messages, 1=enable simulation messages
  .SRC_INPUT_REG  (1)  // integer; 0=do not register input, 1=register input

) xpm_cdc_single_inst (

  .src_clk  (src_clk),  // optional; required when SRC_INPUT_REG = 1
  .src_in   (src_in),
  .dest_clk (dest_clk),
  .dest_out (dest_out)

);

// End of xpm_cdc_single_inst instance declaration
```

Figure 3-15: **XPM_CDC Example**

Some XPMs deliver constraints that are defined in Tcl files located in the `./data/ip/xpm/<xpm>/tcl` folder of a specific XPM. The constraints are applied during synthesis and appear in the synthesis log file along with other constraints that are processed. The constraints can have dependencies on a clock object present on the net that connects to the XPM. This is because some XPMs query the period property of the clock for setting a constraint. If the clock object is not present, a critical warning is generated.

IMPORTANT: *When using the report_compile_order command, the Tcl constraint files for the XPMs in the design are not shown unless you have opened the elaborated, synthesized, or implemented design.*

For details on the various XPMs and their parameterization options, see this link in the *UltraScale Architecture Libraries Guide* (UG974) [Ref 23], or the *Vivado Design Suite 7 Series FPGA and Zynq-7000 SoC Libraries Guide* (UG953) [Ref 24].

Cross Probing to Source Files

The Vivado IDE provides cross probing to RTL source files from the following windows:

- Schematic window (RTL elaborated, synthesis, or implementation)
- Netlist window (after synthesis or implementation)
- Device window (for an implemented design)

To cross probe, select a cell from any of these windows, and select the **Go To Definition** or **Go To Source** right-click command. The RTL source opens, and the line with the instance is highlighted.

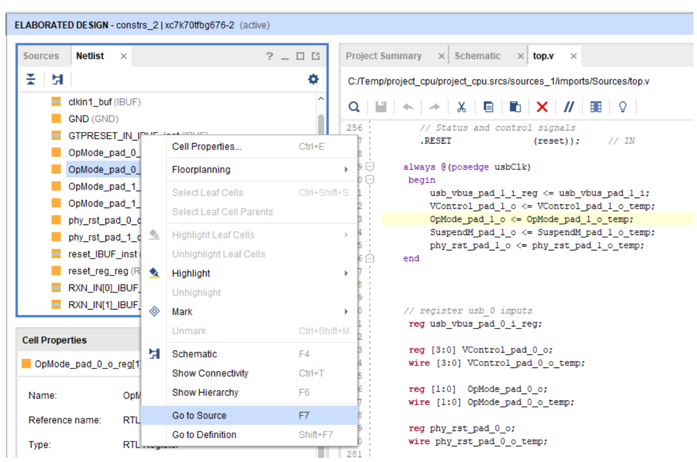

Figure 3-16: **Cross Probing the Elaborated Netlist to an RTL Source**

Tcl Commands for Cross Probing to Source Files

You can use the `FILE_NAME` and `LINE_NUMBER` properties on a cell to get information about where the cell is located in the RTL source. You can then open the RTL source in a text editor and navigate to the appropriate line number. Following is the associated Tcl command:

- **Tcl Command**: `get_cells`

Note: By default, `get_cells` truncates the returned results in the Tcl Console and log file after the first 500 results. For more information, including how to change the default setting, see this link in the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4].

- **Tcl Command Example**: The following example shows how to use the `get_cells` Tcl command in an open design to get a specific instance of a cell, query the properties of that cell object, and report the file and line of interest:

```
set cellName dac_spi_i0
set fileName [get_property FILE_NAME [get_cells $cellName]]
set lineNum [get_property LINE_NUMBER [get_cells $cellName]]
puts "Cell: $cellName is instanced in file: $fileName \
at line number $lineNum"
```

Using Alternate Text Editors

In the Vivado IDE, you can select an alternative text editor as follows:

1. Select **Tools > Settings**.
2. In the Settings dialog box Text Editor page, select an alternate editor from the Current Editor drop-down list.

   When you select an editor from the list, an executable name appears in the settings. The path to the executable needs to be in your path. See the appropriate Windows or Linux documentation for help on how to add a path to your environment.

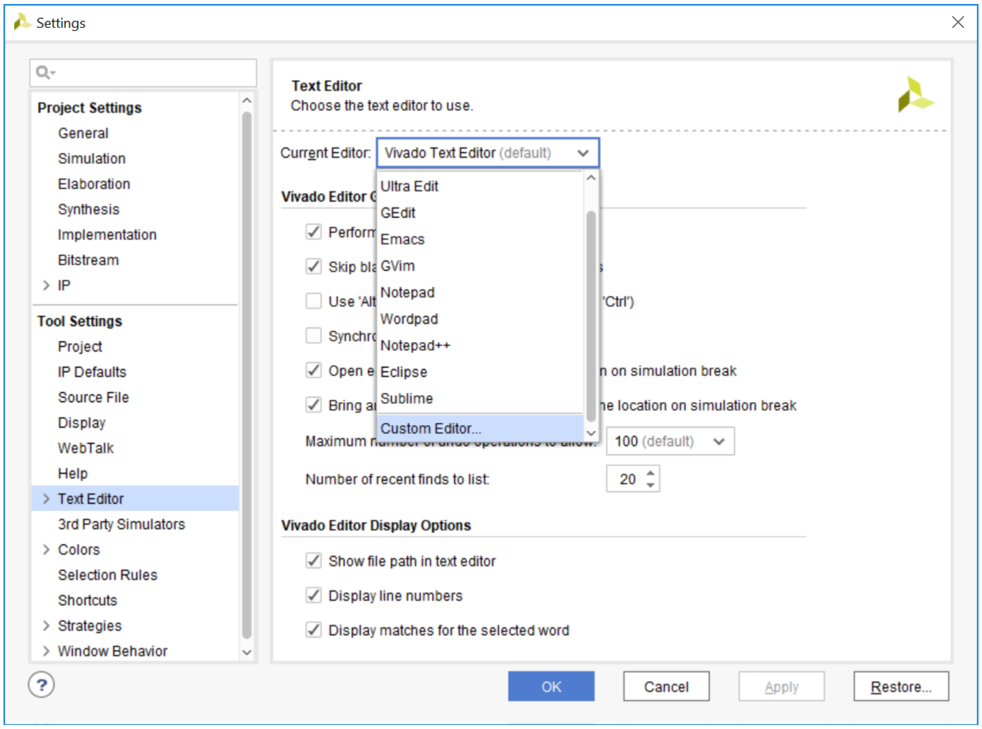

Figure 3-17: **Settings Dialog Box—Text Editor Page**

If your editor is not listed, select **Custom Editor**. In the Custom Editor Definition dialog box, enter the name or location of the executable and the command line syntax used to run the editor.

TIP: *When using an alternative text editor, cross probing works differently. The file opens in the external editor but does not go to the line number automatically.*

Working with Simulation Sources

In the Vivado IDE, you can add simulation sources to the project for behavioral simulation of an RTL Project. Simulation source files include hardware description language (HDL)-based test bench files to use as a stimulus for simulation. Simulation sources are used for behavioral simulation in the Vivado simulator.

The Vivado IDE stores simulation source files in simulation sets that display in folders in the Sources window, and are remotely referenced or stored in the local project directory. Simulation sets enables you to define different sources for different simulation configurations. For example, one simulation source can provide stimulus for behavioral simulation using one test bench while another can contain a different test bench. When adding simulation sources to the project, you can specify which simulation set into which to add files.

Note: For more information, see Adding or Creating Simulation Source Files in the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 10].

Adding and Creating Simulation Source Files

1. Select **File > Add Sources**.

   Note: Alternatively, select **Add Sources** from the right-click menu or from the Flow Navigator.

2. In the Add Sources wizard select **Add or Create Simulation Sources**, and click **Next**.

   ◦ In the Add or Create Simulation Sources dialog box, set the following options, and click **Finish**.

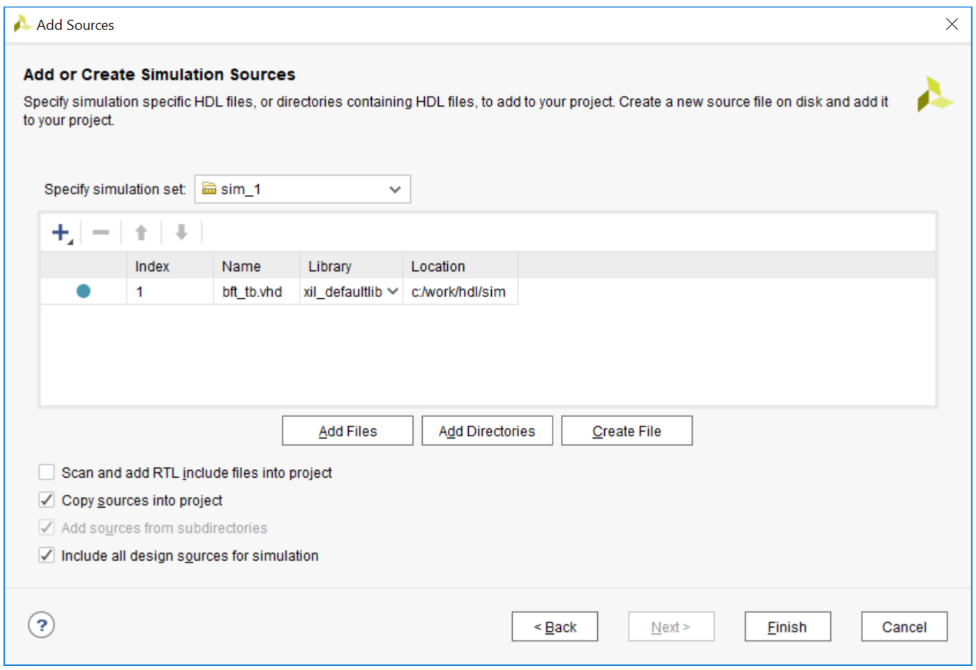

Figure 3-18: **Add Sources Wizard—Add or Create Simulation Sources Page**

   ◦ **Specify Simulation Set**: Enters the name of the simulation set to put test bench files and directories. Select the **Create Simulation Set** option from the drop-down menu to define a new simulation set.

   ◦ **Add Files**: Click the '+' sign and select **Add Files** from the drop-down menu to open a file browser so you can select simulation source files to add to the project.

   ◦ **Add Directories**: Opens a directory browser to add all simulation source files from the selected directories. Files in the specified directory with valid source file extensions are added to the project.

- **Create File**: Opens the Create Source File dialog box in which you can create new simulation source files.
- **Remove**: Removes the selected source files from the list of files to be added.
- **Move Up / Move Down**: Moves the file up/down in the list order.
- **Library**: This column specifies the library for an added file or directory by selecting one from the currently defined library names, or specify a new library name by typing in the Library text field.

  Note: This option applies to VHDL files only. By default, HDL sources are added to the `xil_defaultlib` library. You can create or reference additional user VHDL libraries as needed. For Verilog and SystemVerilog files, leave the library set to `xil_defaultlib`.
- **Scan and Add RTL Include Files into Project**: Scans the added RTL files and adds any referenced include files.
- **Copy Sources into Project**: Copies the original source files into the project and uses the local copied version of the file in the project.

  Note: If you selected to add directories of source files using the Add Directories command, the directory structure is maintained when the files are copied locally into the project. For more information, see Using Remote Sources or Copying Sources into Project.
- **Add Sources from Subdirectories**: Adds source files from the subdirectories of directories specified in the Add Directories option.
- **Include all design sources for simulation:** Copy all design source files from the `sources_1` fileset into the simulation fileset.

Working with Constraints

The Vivado IDE supports the Xilinx® design constraint (XDC) and Synopsys design constraint (SDC) file formats. The SDC format is for timing constraints while the XDC format is for both timing and physical constraints. Constraints can include placement, timing, and I/O restrictions. You can create constraints during various steps in the design flow, including RTL analysis, synthesis, and implementation. For more information on constraint files, constraint sets, and the various types of constraints, refer to *Vivado Design Suite User Guide: Using Constraints (UG903)* [Ref 7].

The Vivado Design Suite provides flexibility in defining and using constraints in a project. You can use a single XDC file to add and maintain the design constraints, or you can use multiple XDC files to organize the constraints into separate files. You can create multiple constraint sets to experiment with various types of constraints, or store multiple versions of constraints. Each constraint set can contain one or more constraint files.

The Vivado Design Suite also lets you define constraints in Tcl scripts which can either be sourced in the Tcl shell or Tcl console, or added to a constraint set in your design. Defining constraints in Tcl scripts allows you to use standard Tcl commands as part of the constraint scope and definition. However, defining constraints in Tcl scripts also has certain limitations, such as not being able to save changes to design constraints back to the source Tcl script.

Note: For more information on working with Tcl scripts, see this link in the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 7].

You can open multiple designs referencing a single constraint set. However, you must be careful to manage changes made to multiple designs that reference the same constraint set. If the Vivado IDE detects unsaved changes in multiple open designs, it prompts you to select which design to save to the referenced constraint file.

CAUTION! *When saving constraints files, be careful not to overwrite any unsaved constraint definitions in an unsaved design.*

An implemented design saves a snapshot of the constraints used during the implementation run along with a reference to the original constraint file lines. When opening an implemented design, the constraints loaded from the implementation run might be older than the implementation constraints from the constraints set in the project. This can cause the loss of newer constraints in the project constraint files when you save the design from an implemented run after adding or editing the constraints in memory. Generally, the Vivado IDE manages these revision issues and prompts you to take the appropriate action as needed. However, you should be aware of the potential conflict between the constraints in memory and the constraints files in the constraints set associated with the implementation run.

In the Vivado IDE, the following windows enable you to create and work with constraints:

- **Timing Constraints Window**: Shows all XDC file timing constraints for the project in a table format. You can an interactively edit existing constraints, which are saved back to the source file, or create new constraints.
- **Device Constraints Window**: Enables you to set various SelectIO™ interface constraints on displayed banks.
- **Physical Constraints Window**: Enables you to create and manage Pblocks.

TIP: *Select **Tools > Timing > Constraints Wizard** on a synthesized design to create a top-level XDC file based on design methodologies recommended by Xilinx. The wizard guides you through specifying clocks, setting up input and output constraints, and properly constraining cross-clock domain clock groups.*

VIDEO: *See the* Vivado Design Suite QuickTake Video: Using the Vivado Timing Constraint Wizard *for an introduction to using the Timing Constraints Wizard.*

Adding and Creating Constraint Files

1. Select **File > Add Sources**.

   Note: Alternatively, you can click **Add Sources** in the Flow Navigator, or select **Add Sources** from the right-click menu in the Sources window.

2. In the Add Sources wizard select **Add or Create Constraints**, and click **Next**.

3. In the Add or Create Constraints page, set the following options, and click **Finish**.

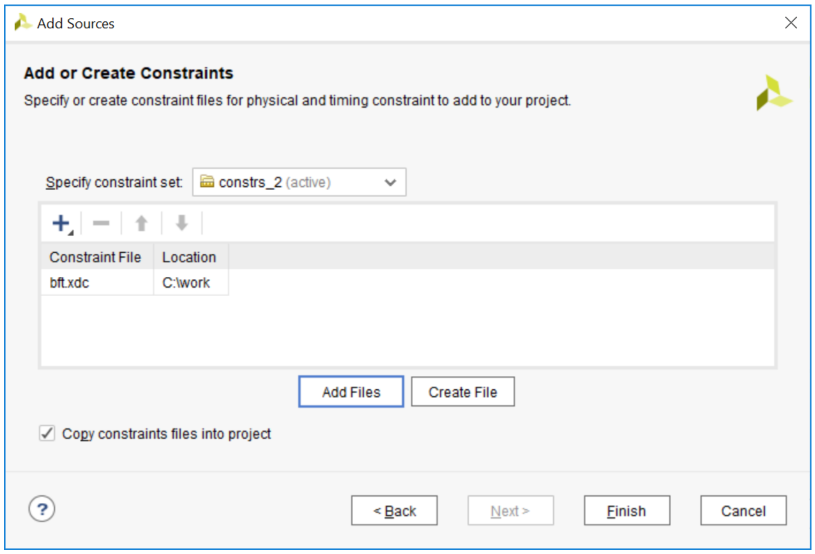

Figure 3-19: **Add Sources Wizard—Add or Create Constraints Page**

- **Specify Constraint Set:** Defines the constraint set into which the constraint files are placed. By default, the currently active constraint set is selected, but you can specify a different constraint set or define a new constraint set using the drop-down menu.
- **Add Files**: Specifies the XDC, SDC, or Tcl script files to add to the project.
- **Create File**: Creates a new top-level XDC for the project.
- **Remove**: Removes the selected file from the Constraint File list.
- **Move Up / Move Down**: Moves a constraint file up or down in the listed order of files. XDC, SDC, or Tcl script files consist of commands that set timing and physical constraints and are order-dependent. Multiple files in a constraint set are read in the order they appear; the first file in the list is the first file processed.

IMPORTANT: *Constraints are read in the order they appear in a constraint set. If the same constraint is defined more than once in a constraint file, or in more than one constraint file, the last definition of the constraint overwrites earlier constraints.*

- **Copy Constraints into Project**: Copies constraint files into the local project directory instead of referencing the original files.

Setting the Target XDC File

When editing a design, modified constraints are written back to the XDC file that are defined in. Newly created constraints are written to the XDC file identified as the target XDC file when you save the constraints. By default, in a new constraint set, there is no target XDC file. When you create new constraints, you must set a target XDC file when you save the constraints.

TIP: *Existing constraints that are modified are written back to the XDC file from which they originated, not the target XDC.*

To indicate that constraints need to be saved, the **Save Constraints** toolbar button is enabled. When you click the **Save Constraints** toolbar button, the Save Constraints File dialog box lets you choose an existing XDC file in the active constraint set, or create a new file to add to the active constraint set.

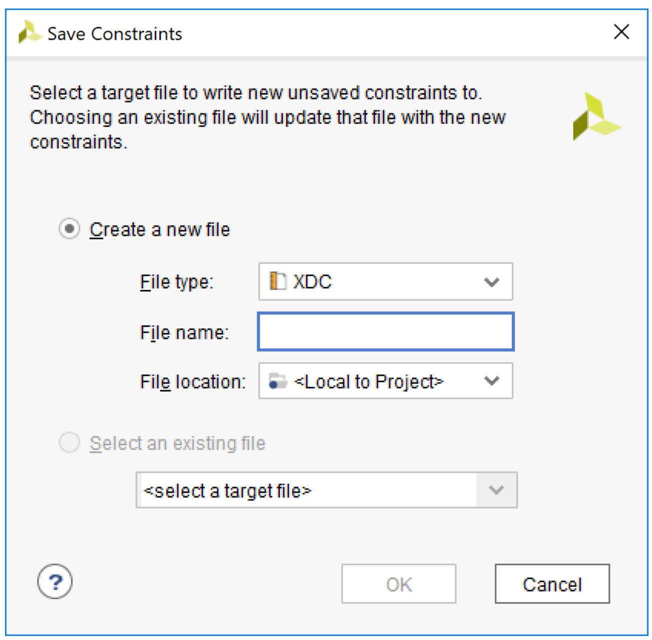

Figure 3-20: **Save Constraints File Dialog Box**

If an XDC file is set as a target, the word "(target)" appears next to the file name in the Sources window. You can change the target XDC file at any time using the **Set as Target Constraint File** right-click menu command in the Sources window.

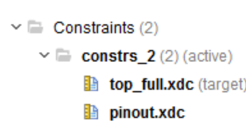

Figure 3-21: **Target XDC File in the Sources Window**

Referencing Original XDC Files or Copying Files

As with other source files, you can reference XDC files from a remote location or copy the files locally into the project directory. If your project references remote files, the Vivado IDE automatically detects changes to the referenced source file and prompts you to **Reload the design** with the latest files.

To copy constraints into the project, do one of the following:

- When you add constraints to the project using the Add Sources command, you can copy the constraints to the local project directory by selecting the **Copy Constraints into Project** option.
- If you initially add the constraints as remote sources but later wish to copy them into the project directory, use **Copy File into Project** or **Copy All Files into Project** in the right-click menu in the Sources window to copy some or all individual remote source files into the project directory.

Note: For more information, see Using Remote Sources or Copying Sources into Project.

Using Constraint Sets

A constraint set is one or more constraint files that are maintained independently and concatenated into the in-memory design for analysis and implementation. A constraint set defines the constraint files to be used at specific moments, or under specific conditions, in the design process. By defining multiple constraints sets, you can, for example, specify different active constraints to resolve floorplanning and timing problems.

The XDC files can be used during synthesis, implementation, or both. By default all XDC files are set to be used in both synthesis and implementation, as defined by the USED_IN property on the constraint file. To change the USED_IN property**,** select the XDC file in the Sources window, and check or uncheck the appropriate box in the General view of the Source File Properties window. Refer to the USED_IN property in *Vivado Design Suite Properties Reference Guide* (UG912) [Ref 14] for more information.

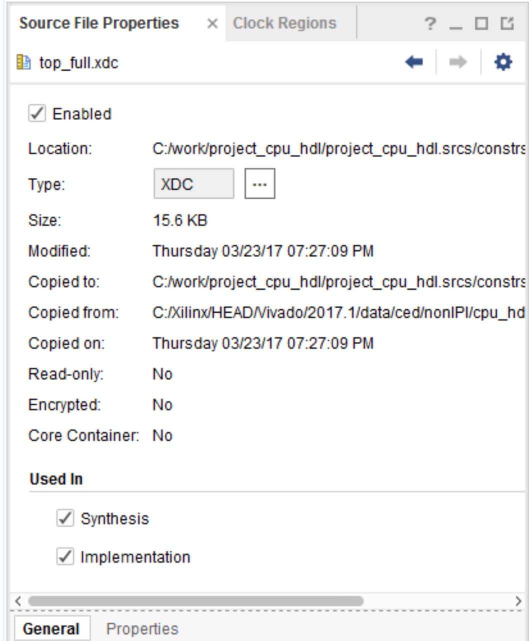

Figure 3-22: **USED_IN Property of an XDC File**

Creating and Editing Constraint Sets

1. In the Sources window, select **Edit Constraint Sets** from the right-click menu.
2. In the Edit Constraint Set dialog box, do one of the following:
   - To edit a constraint set, click the drop-down menu next to the Specify Constraint Set field, and select a constraint set.
   - To create a constraint set, click the drop-down menu next to the Specify Constraint Set field, and select **Create Constraint Set**. In the Create Constraint Set Name dialog box, enter a name for the constraint set, and click **OK**.

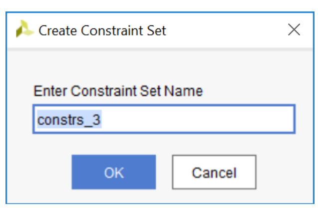

Figure 3-23: **Create Constraint Set Name Dialog Box**

3. In the Edit Constraint Set dialog box, set the following options, and click **OK**:
    - **Add Files**: Specifies XDC or SDC files to add to the constraint set.
    - **Create File**: Specifies a name and location for a new XDC file to add to the constraint set.
    - **Remove**: Removes the selected file from the Constraint File list.

      Note: You can only remove files that have not yet been added to the constraint set using the **OK** button. To remove a file that was already added to the constraint set, select the file in the Sources window, and select **Remove File from Project** from the right-click menu.
    - **Up / Down**: Moves a constraint file up or down in the listed order of XDC and SDC files. XDC and SDC files consist of commands that set timing and physical constraints and are order-dependent. If there are multiple files in a constraint set, the order in which they appear in the Sources window corresponds to the order that the Vivado IDE processes the files. The first file in the list is the first file processed. If the same constraint appears in more than one constraint file, the last file read has precedence in defining the constraint.
    - **Copy Constraints into Project**: Copies constraint files into the local project directory instead of referencing the original files.

Defining the Active Constraint Set

If more than one constraint set exists, you must designate the active constraint set. The Vivado IDE uses the active constraint set by default when you launch the synthesis or implementation runs or when you open an elaborated, synthesized or implemented design.

To set the active constraint set, select the constraint set in the Sources window, and click **Make active** from the right-click menu. In the Sources window, the active constraint set appears in bold with the word "(active)" next to it.

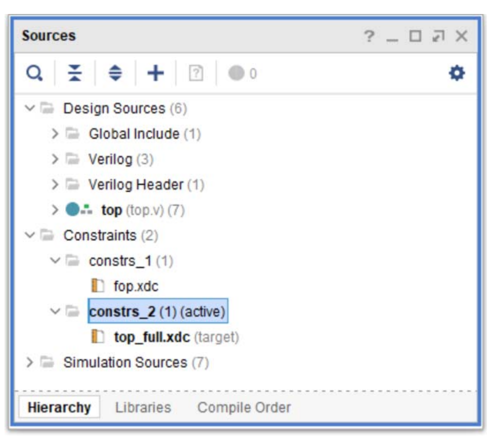

Figure 3-24: **Active Constraint Set**

Creating Constraints Sets Using the Save Constraints As Command

At any time in the design flow you can also create a copy of the active constraint set using the **Save Constraints As** command. With multiple places to make changes to constraint files, or to model the affect of different constraints, it is useful to save changes to a new constraint set to manage changes or support "what-if" analysis.

Select **File > Constraints > Save As** to open the Save Constraints As dialog box, and enter a new constraint set name in which to save all constraints.

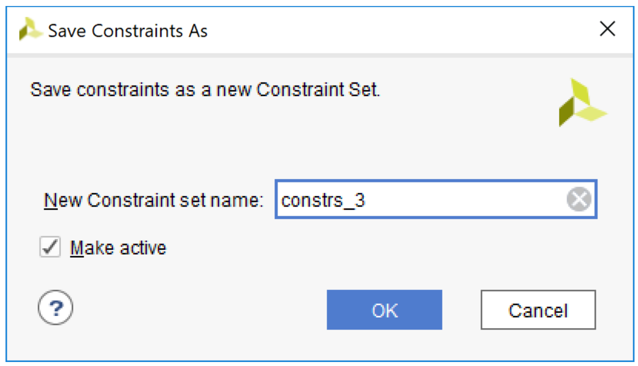

Figure 3-25: **Save Constraints As Dialog Box**

The Save Constraints As dialog box does the following:

- Creates a new constraint set.
- Copies the active constraint files into the new constraint set in the local project directory.
- Writes any modifications to the constraints to the copied constraint files, leaving the original XDC files unchanged.
- Provides an option to make the new constraint set active in the project.

Exporting Constraints

In some cases, you might want to use the Vivado IDE to write a constraints file for use in scripted design flows. To export all the constraints applied to the in-memory design to a single constraints file, select **File > Export > Export Constraints**.

In addition, you can export the I/O standard constraints for I/O ports and banks (both user-specified values and default values assigned by the Vivado IDE) to an XDC file. To export I/O constraints, select **File > Export > Export I/O Ports**, and generate an XDC file.

Enabling or Disabling Constraint Files

When you add or create constraint files, the files are enabled in the Sources window by default. You can disable constraint files to prevent them from being used during elaboration, synthesis, or in implementation.

- To disable constraint files, select the files in the Sources window, and select the **Disable File** right-click command.
- To enable disabled files, select the files in the Sources window, and select the **Enable File** right-click command.

Changing the Constraint Evaluation Order

You can reorder user constraints within the associated constraint set. In the Sources window, drag and drop the XDC files to rearrange the order.

To get an ordered list of all XDC files that the Vivado IDE processes, use the following command in the Tcl Console: `report_compile_order -constraints`. This lists all the constraints in the design, including user constraints and IP.

Note: For more information on how to change the order of constraints, see the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 7].

Working with Sources in Non-Project Mode

Unlike Project Mode in which source files are managed for you, source files are under your control in Non-Project Mode. Using Tcl commands, you specify the files to process and output files to generate, including netlist, bitstream, and report files. Project Mode and Non-Project Mode Commands shows commonly used Project Mode commands and corresponding Non-Project Mode commands. For more information on Project Mode and Non-Project Mode, see this link in the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) [Ref 1]. For more information on Tcl commands, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4].

Note: In Non-Project Mode, files are compiled in the order the `read_*` commands are listed in the Tcl script.

Note: To Select a new part in the current installation, close the current project and upgrade the current installation to add additional part.

Table 3-1: **Project Mode and Non-Project Mode Commands**

Action	Project Mode Command	Non-Project Mode Command
Reading design sources	`add_files` `import_files`	`read_<file_type>` (for example: `read_verilog`, `read_vhdl`, `read_xdc`, `read_edif`, and `read_ip`) ***Note:*** You can import an NGC file in Non-Project Mode using the `read_edif` command.
Running synthesis	`launch_runs synth_1`	`synth_design`

Running implementation	`launch_runs impl_1`	`opt_design`
		`place_design`
		`phys_opt_design`
		`route_design`

	`launch_runs impl_1 -to_step write_bitstream`	`write_bitstream <file_name>`

Generating reports	`report_timing`
	`report_timing_summary`
	`report_drc`
	`report_methodology`
	`report_clock_interaction`
	`report_utlization`
	`report_route_status`
	Note: In Project Mode, many reports are automatically generated. For a list of all reports, enter: `help report_*`.
		Same as Project Mode

Running simulation	`launch_xsim`	`xsim` ***Note:*** This command launches Vivado simulation outside of the Vivado IDE.
Writing design checkpoints	`write_checkpoint <file_name>.dcp` ***Note:*** In Project Mode, DCP files are automatically created for each stage of implementation.	Same as Project Mode

Following is an example of a Non-Project Mode script, which reads in various source files:

```
# create_bft_batch.tcl
# bft sample design
# A Vivado script that demonstrates a very simple RTL-to-bitstream batch flow
#
# NOTE:  typical usage would be "vivado -mode tcl -source create_bft_batch.tcl"
#
# STEP#0: define output directory area.
#
set outputDir ./Tutorial_Created_Data/bft_output
file mkdir $outputDir
#
# STEP#1: setup design sources and constraints
#
read_vhdl -library bftLib [ glob ./Sources/hdl/bftLib/*.vhdl ]
read_vhdl ./Sources/hdl/bft.vhdl
read_verilog  [ glob ./Sources/hdl/*.v ]
read_xdc ./Sources/bft_full.xdc
#
# STEP#2: run synthesis, report utilization and timing estimates, write checkpoint
design
#
synth_design -top bft -part xc7k70tfbg484-2 -flatten rebuilt
write_checkpoint -force $outputDir/post_synth
report_timing_summary -file $outputDir/post_synth_timing_summary.rpt
report_power -file $outputDir/post_synth_power.rpt
#
# STEP#3: run placement and logic optimzation, check against the UltraFast
methodology checks, report utilization and timing estimates, write checkpoint design
#
opt_design
report_methodology -file $outputDir/post_opt_methodology.rpt
place_design
phys_opt_design
write_checkpoint -force $outputDir/post_place
report_timing_summary -file $outputDir/post_place_timing_summary.rpt
#
# STEP#4: run router, report actual utilization and timing, write checkpoint design,
run drc, write verilog and xdc out
#
route_design
write_checkpoint -force $outputDir/post_route
report_timing_summary -file $outputDir/post_route_timing_summary.rpt
report_timing -sort_by group -max_paths 100 -path_type summary -file
$outputDir/post_route_timing.rpt
report_clock_utilization -file $outputDir/clock_util.rpt
report_utilization -file $outputDir/post_route_util.rpt
report_power -file $outputDir/post_route_power.rpt
report_drc -file $outputDir/post_imp_drc.rpt
write_verilog -force $outputDir/bft_impl_netlist.v
write_xdc -no_fixed_only -force $outputDir/bft_impl.xdc
#
# STEP#5: generate a bitstream
#
write_bitstream -force $outputDir/bft.bit
```

Chapter 4

Elaborating the RTL Design

Overview

The Vivado® Design Suite has three views of the design represented by the source files and design constraints added to the project, or read into memory in Non-Project Mode: the elaborated RTL design, the synthesized design, and the placed and routed design.

RTL elaboration of the top-level design runs RTL linting checks, performs high-level optimizations, infers logic from the RTL and builds design data structures, and optionally applies design constraints. In the default Out-of-Context design flow, you can also have the Vivado Design Suite include synthesized design checkpoints (DCPs) of IP cores, block designs, DSP modules, or hierarchical blocks into the elaborated design.

TIP: *Including the out-of-context modules into the elaborated design can add time to the elaboration process as the Vivado tool will synthesize any modules that are not up-to-date in the design. This feature can be enabled or disabled under the Elaboration tab of the Settings dialog box. See* *Elaboration Settings* *for more information.*

The Vivado Design Suite offers many analysis capabilities for an RTL design. For example, you can:

- Visualize design details with Schematic and Hierarchy windows
- Cross probe between windows
- Run design rule checks (DRCs)
- Check messaging
- Search the RTL netlist produced with the Find command
- Create and apply constraints at the RTL level

  Note: You *cannot* run timing analysis at this stage.

Elaborating the Design in Project Mode

Enabled RTL source files in the project are elaborated automatically during synthesis. You can also elaborate source files manually for constraint development and RTL netlist exploration. The Messages window shows the messages from elaboration and compilation. You can select the HDL language options used during elaboration in the Vivado IDE Project Settings. For information, see General Settings in Chapter 2.

Elaboration results are *not* saved with the design. Every time you open the elaborated design, it is re-elaborated. However, you can save any constraints that were created in the elaborated design.

After design source files are imported into the project, use either of the following methods to elaborate and open the design:

- Select **Flow > Open Elaborated Design**.
- In the RTL Analysis section of the Flow Navigator, select **Open Elaborated Design** to load the elaborated netlist, the active constraint set, and the target device into memory.

To specify the design name to elaborate, use either of the following methods:

- Select **Flow > New Elaborated Design**.
- In the Flow Navigator, select **New Elaborated Design** from the RTL Analysis right-click menu.

If there are out-of-context design modules, IP cores, block designs, DSP modules, in your design sources, the message appears when you open an elaborated design. The message indicates that the **Link IP from OOC runs** and **Load constraints** options from the Elaboration Settings dialog box may impact the performance of opening the elaborated design. You can disable these settings to speed elaboration. Refer to Elaboration Settings for more information.

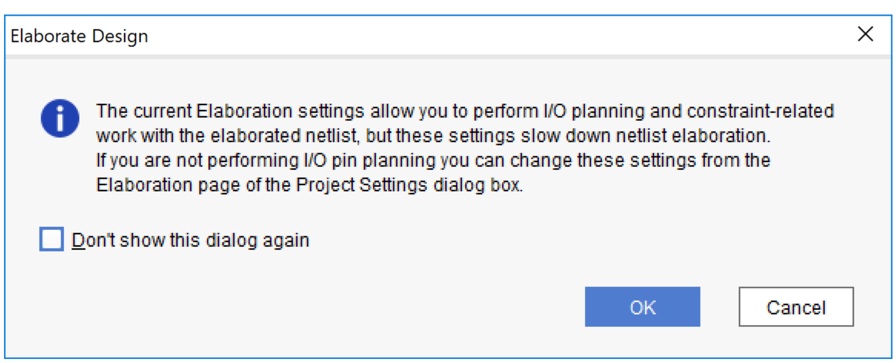

Figure 4-1: **Elaborate Design Message**

When you open an elaborated design, the Vivado Design Suite automatically checks and compiles the RTL source files, generates the top schematic view, and displays the design in the default view layout.

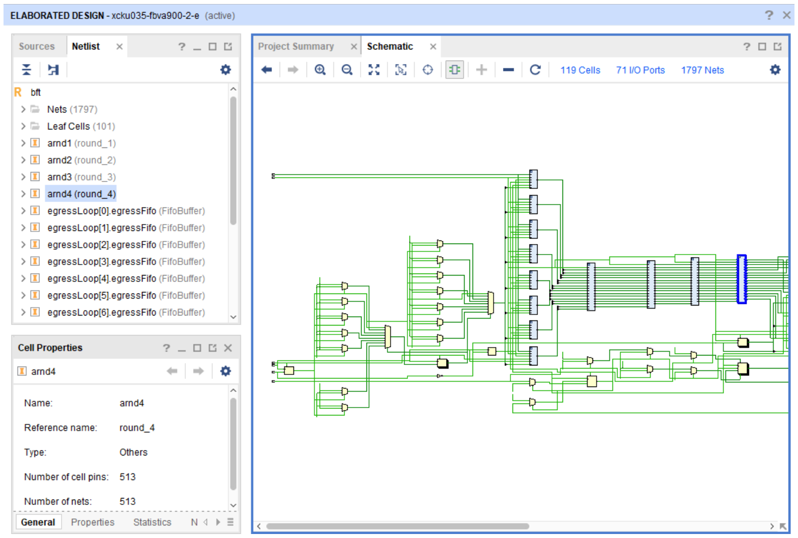

Figure 4-2: **Elaborated Design in the RTL Schematic Window**

The Vivado IDE automatically identifies the top module for the design in most cases. In some cases, where there might be multiple candidates, the tool prompts you to choose the top module for the design. You can also manually define the top module by selecting **Set as Top** from the right-click menu in the Sources window.

Note: In the Hierarchy view of the Sources window, the top module icon identifies the current top module.

Tcl Command for Elaborating the Design in Project Mode

Following is the associated Tcl command:

- **Tcl Command**: `synth_design -rtl -name <project_name>`
- **Tcl Command Example**: `synth_design -rtl -name rtl_1`

Viewing Elaboration Messages

The Messages window displays the results of the compilation and flags irregularities in the RTL source files under the Elaborated Design section.

You can filter the Messages window to display errors or warnings or informational messages from the results of RTL elaboration. To enable or disable the display of Errors, Critical Warnings, Warnings, or Informational messages, select a check box in the banner of the Messages window.

You can select any of the warning or error messages in the Messages window to load the corresponding RTL source file with the selected source code highlighted in the Vivado IDE Text Editor.

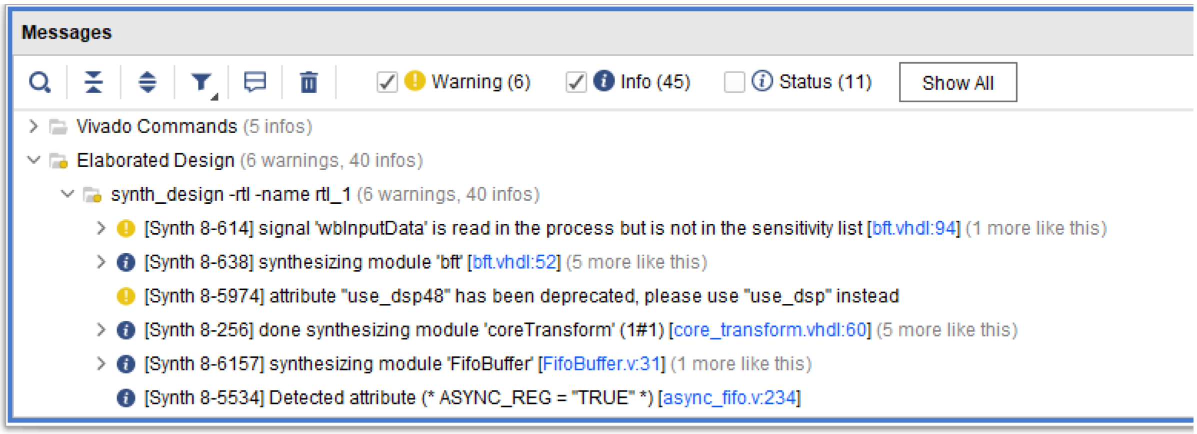

Figure 4-3: **Elaborated Design Messages**

Analyzing the RTL Logic Hierarchy

The Vivado IDE provides the following views into the logical design hierarchy:

- **Netlist View**: Shows an expandable logic tree of the RTL hierarchy and primitives.
- **Hierarchy View:** Shows a graphical representation of the logic hierarchy.
- **Schematic View**: Shows the logic and hierarchy in an explorable schematic representation.

By default, when you elaborate a design by running **Open Elaborated Design** in the Flow Navigator, the RTL Schematic displays for the entire design. All views cross-select offering a unique set of capabilities to explore and analyze the logical design. For more information, see the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 6].

Exploring the Elaborated Design Schematic

You can select any level of logic hierarchy in the RTL Netlist window and display it in the RTL Schematic window. To invoke the RTL Schematic window for any selected logic, do one of the following:

- Select **Tools > Schematic**.
- In the Netlist window, select **Schematic** from the right-click menu.

For more information on traversing, expanding, and exploring the RTL Schematic, see Using the Schematic Window in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 6].

You can also use the **Find** command to search for logic objects in the elaborated design using a range of filtering techniques. See Finding Design or Device Objects in the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 6] for more information.

Using the Hierarchy Window

The Vivado IDE includes an Hierarchy window, which is useful for viewing the hierarchy of a design. To invoke the Hierarchy view for any selected logic, do one of the following:

- Select **Tools > Show Hierarchy**.
- In the Netlist or Schematic window, select **Show Hierarchy** from the right-click menu.

These windows have full cross probing support. Logic selected in the Netlist or Schematic window is highlighted in the Hierarchy window.

Exploring the RTL Source Files

You can select any logic element in the Netlist view or Schematic and open the instantiation of that object in the RTL source file it is instantiated in. You can also open the definition of the logic in the RTL file it is defined in.

To open the instantiation or definition of any selected logic in the RTL source file, select the object and select **Go To Source** or **Go to Definition** from the right-click menu. The Vivado IDE opens the appropriate source file with the specific instance highlighted.

Running Methodology Checks

The Vivado Design Suite provides automated methodology checks based on the *UltraFast™ Design Methodology Guide for the Vivado Design Suite* using the Report Methodology command.

You can generate a methodology report on an opened, elaborated, synthesized, or implemented design. For an elaborated design, the methodology report checks the XDC and RTL files. For information on running the methodology report using Tcl commands, see this link command in the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4].

RECOMMENDED: *Running the methodology report allows you to find design issues early during the elaboration stage prior to synthesis, which saves time in the design process. It is highly recommended that you run these checks on your design and address any issues identified.*

Running Report Methodology

1. From the Flow Navigator under RTL Analysis, select **Open Elaborated Design**.
2. Once the design has been elaborated, select **Report Methodology** from the Flow Navigator under RTL Analysis. Alternatively, select **Reports > Report Methodology**.

Note: Alternatively, you can enter this command in the Tcl Console: `report_methodology -name <results_name>`.

3. In the Report Methodology dialog box, set these options, and click **OK**:
   - **Results name**: Specify the name for the results, which appear in a tab in the Methodology window. Entering a unique name makes it easier to identify the results for a particular run during debugging.
   - **Output file**: Optionally, to write the report to a file, check the **Export to file** box and enter a file name. To select a path other than the default, use the browse button.
   - **Interactive report file**: Save the report to a file.
   - **Rules**: Allows you to explore and specify which rules to run.
   - **New tab**: By default a new tab is created for the report. To disable this option, uncheck the **Open in a new tab** box.

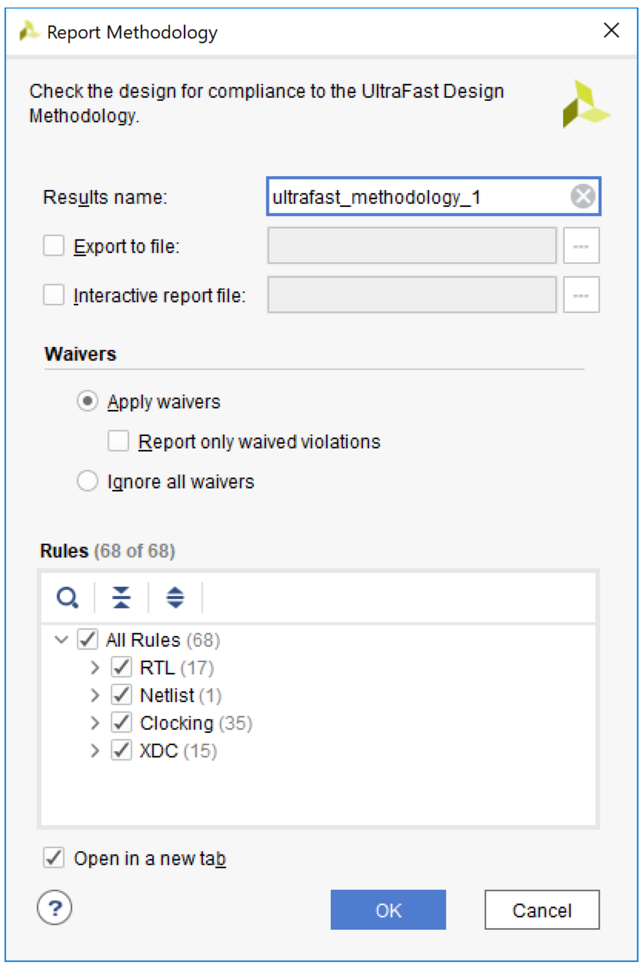

Figure 4-4: **Report Methodology Dialog Box**

Analyzing the Methodology Report

If violations are found, the Methodology window opens. The window displays the violations, grouped under the various rule categories.

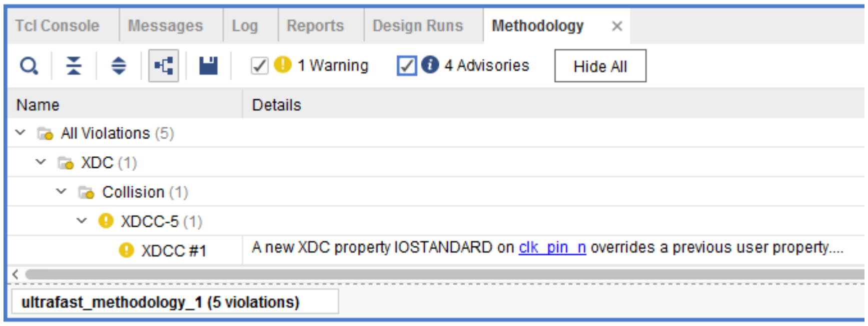

Figure 4-5: **Report Methodology Window with Violations in the Elaborated Design**

Reporting DRCs

The following sections describe selecting DRCs rules and analyzing DRC violations in the Vivado IDE. For information on running DRCs using Tcl commands, see this link in the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 4]. For information on creating custom DRCs, see the *Vivado Design Suite User Guide: Using Tcl Scripting* (UG894) [Ref 3].

RECOMMENDED: *Running RTL DRCs enables you to find design issues early, during the elaboration stage prior to synthesis, which saves time over the course of your design.*

Selecting DRC Rules

1. From the Flow Navigator under RTL Analysis, select **Open Elaborated Design**.
2. Once the design has been elaborated, select **Report DRC** from the Flow Navigator under RTL Analysis. Alternatively, select **Reports > Report DRC**.

   Note: Alternatively, you can enter this command in the Tcl Console: `report_drc -name <results_name>`.
3. In the Report DRC dialog box, set the following options, and click **OK**:
   - **Results name**: Specify the name for the DRC results, which appear in a tab in the DRC window. Entering a unique name makes it easier to identify the results for a particular run during debug in the DRC window.
   - **Output file**: Optionally, specify a file name for the DRC results. To select a path other than the default, use the browse button.
   - **Interactive Report File**: Write the result in the Xilinx RPX format to the specified filename. The RPX file is an interactive report that contains all the report

information and can be reloaded into memory in the Vivado Design Suite using the `open_report` command.

- **Waivers**
  - **Apply waivers**: Use the waivers you created to suppress DRCs that you no longer want to view. For more information, see this link in the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [Ref 17].

    Note: Use the **Display only waived violations** to show just the waived violations in the Results window.
  - **Ignore all waivers**: Ignores the waivers you created.
- **Rule Decks**: Specify a rule deck to run on the design. A rule deck is a collection of design rule checks grouped for convenience. During elaboration only the default rule deck is available. Other rule decks are available at different stages of the FPGA design flow, such as after synthesis or implementation.
  - **default**: Runs a default set of checks recommended by Xilinx.
  - **opt_checks**: Runs checks associated with logic optimization.
  - **placer_checks:** Runs checks associated with placement.
  - **router_checks**: Runs checks associated with routing.
  - **bitstream_checks**: Runs checks associated with bitstream generation.
  - **timing_checks**: Runs checks associated with timing constraints.

    Note: The timing_checks rule deck is not supported for elaborated designs.
  - **incr_eco_checks:** Checks validity of incremental ECO design modifications.
  - **eco_checks:** Checks validity of engineering change order (ECO) design modifications.

    Note: For elaborated designs, only the default rule deck is available.
- **Rules**: After specifying a rule deck, modify the rules to run as needed.

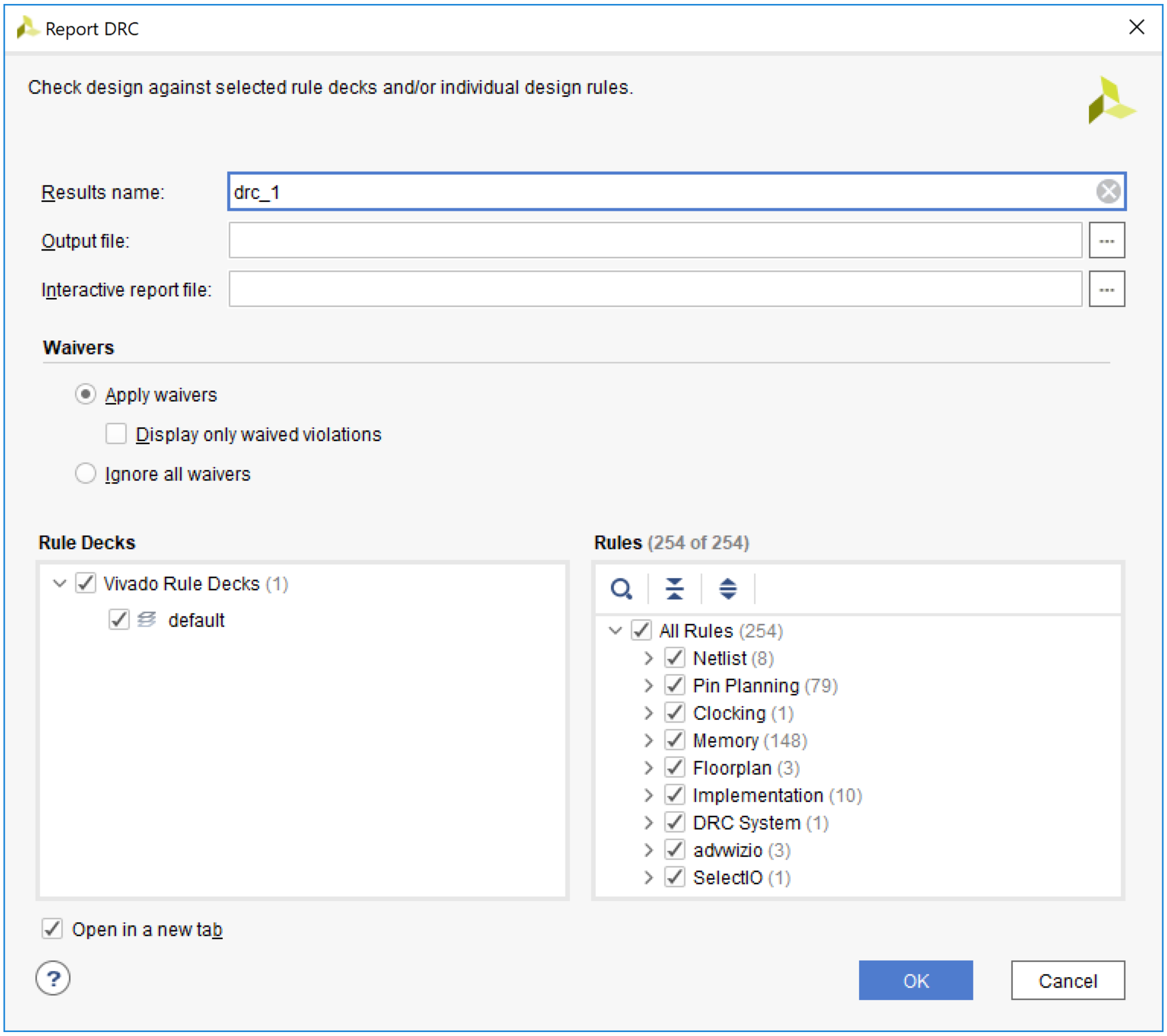

Figure 4-6: **Report DRC Dialog Box**

Analyzing DRC Violations

If violations are found, the DRC window opens. The DRC window displays the rule violations found, grouped under the various rule categories defined in the Run DRC dialog box.

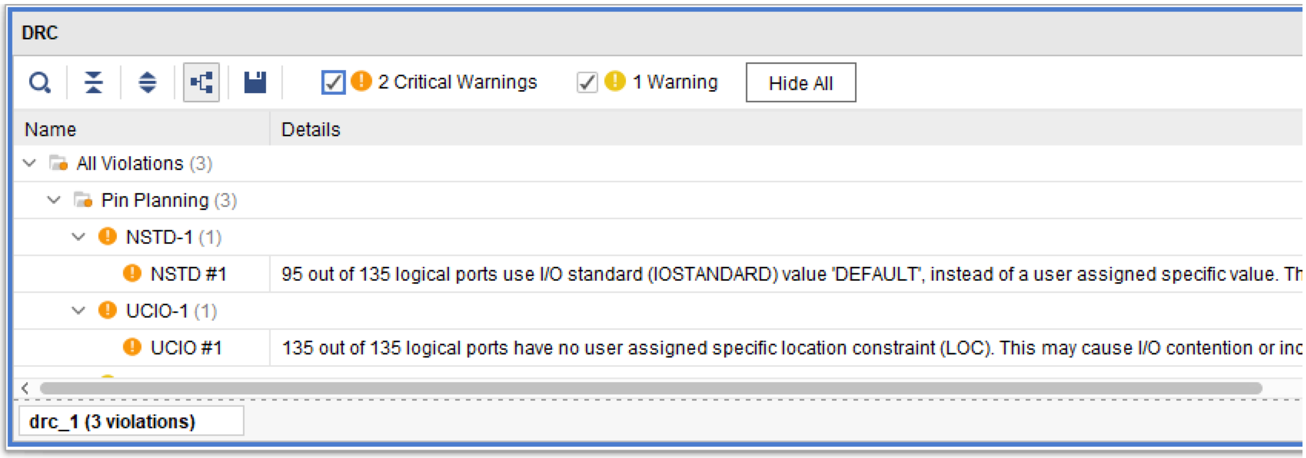

Figure 4-7: **DRC Window with DRC Violations in the Elaborated Design**

The rule violations are categorized by severity and are color coded as follows:

- **Advisory**: Provides general status and feedback on design processing.
- **Warning**: Indicates that design results might be sub-optimal, because constraints or specifications might not be applied as intended.
- **Critical warning**: Indicates that certain user input or constraints will not be applied or do not adhere to best practices. It is highly recommended that you examine these issues and make changes.

  Note: Critical warnings are promoted to errors during bitstream generation.
- **Error**: Indicates an issue that renders design results unusable and cannot be resolved without your intervention. The design flow stops.

TIP: *To see only one message type, double-click the message type in the banner of the Messages window. For example, double-click **errors** to display only error messages.*

You can list DRC violations individually or group violations by rule. To change the display, click the **Group By Rule** toolbar button. When violations are listed individually, you can click the header of the **Severity** column to sort violations by severity. To sort the column:

- Click the column header to sort data in the table in an increasing order.
- Click the column header again to sort the data in the table in a decreasing order.

Note: For more information, see the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 6].

When you select a violation message in the DRC window, the objects associated with the violation are cross-selected in other open windows, such as the Netlist or Schematic windows. This lets you quickly locate and examine the elements of the design that are associated with a specific violation.

In addition, the violation properties are shown in the Violation Properties window by default. In the DRC window, you can also select **Violation Properties** from the right-click menu to open the Violation Properties window. The Violation Properties window shows both a **General** view of the DRC rule violation and specific **Details** of the design elements that violate the rule. The Details view includes links to specific design objects that violate the DRC. Click the links to view the design object in the Netlist window, the Device window, the Schematic window, or the source RTL file.

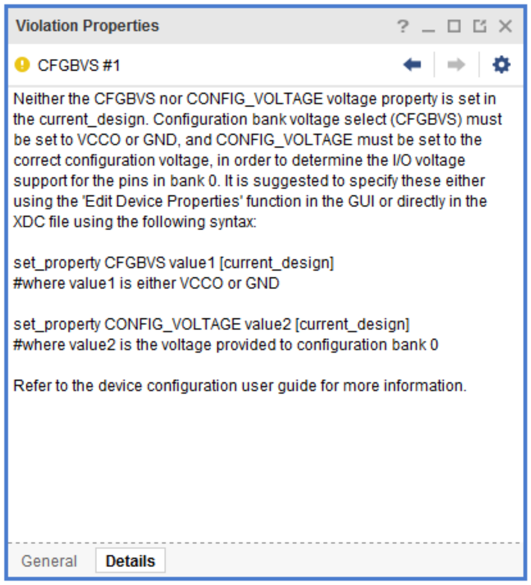

Figure 4-8: **Violation Properties Window**

Tcl Command for Running RTL DRCs

Following is the associated Tcl command:

- **Tcl Command**: `report_drc`
- **Tcl Command Example**: `report_drc -name drc_1`

Note: By default, a text-based report is produced. You can use the `-name` option to create an interactive tab for the report.

Elaborating the Design in Non-Project Mode

In Non-Project Mode, you can perform elaboration of the RTL. You can also cross probe back to the RTL and run DRCs. Cross probing requires that you load the Vivado IDE using the `start_gui` Tcl command. You can perform DRCs with or without the Vivado IDE.

Following is a script that sources various files and elaborates the RTL using the `synth_design` Tcl command with the `-rtl` option. The script also loads the Vivado IDE so you can cross probe back to the RTL source from the schematic or netlist.

Note: When you load Vivado IDE in Non-Project Mode, there is no Flow Navigator. Instead, you must use the Tools menu and Tcl Console to accomplish tasks.

```
# create_bft_batch.tcl
# bft sample design
# A Vivado script that demonstrates a very simple RTL-to-bitstream batch flow
#
# NOTE:  typical usage would be "vivado -mode tcl -source create_bft_batch.tcl"
#
# STEP#0: define output directory area.
#
set outputDir ./Tutorial_Created_Data/bft_output
file mkdir $outputDir
#
# STEP#1: setup design sources and constraints
#
read_vhdl -library bftLib [ glob ./Sources/hdl/bftLib/*.vhdl ]
read_vhdl ./Sources/hdl/bft.vhdl
read_verilog  [ glob ./Sources/hdl/*.v ]
read_xdc ./Sources/bft_full.xdc
#
# STEP #2 Elaborate the RTL and start the GUI for interaction
#
synth_design -top bft -part xc7k70tfbg484-2 -rtl
start_gui
# Use stop_gui to quit the GUI and return back to the Vivado IDE Tcl command line
```

Chapter 5

Debugging the Design

Overview

Debugging an FPGA design is a multi-step, iterative process. Like most complex problems, it is best to break the FPGA design debugging process down into smaller parts, for example, by focusing on making a smaller section of the design work rather than trying to make the whole design work at one time. An example of a proven design and debug methodology is to iterate through the design flow, adding one module at a time and making it function properly in the context of the whole design. You can use this design and debug methodology in any combination of the following design flow stages:

- RTL-level design simulation
- In-system debugging

In addition to using the Set up Debug wizard, you can also use Tcl commands to create, connect, and insert debug cores into your synthesized design netlist. For more information on debugging, see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 20].

RTL-Level Design Simulation

You can functionally debug the design during the simulation verification process. Xilinx® provides a full design simulation feature in the Vivado® simulator. You can use the Vivado simulator to perform RTL simulation of your design. The benefits of debugging your design in an RTL-level simulation environment include full visibility of the entire design and the ability to quickly iterate through the design and debug cycle. For more information on how to configure and launch simulation, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 10].

In-System Debugging

The Vivado IDE also includes a logic analysis feature that enables you to perform in-system debugging of the post-implemented design an FPGA device. The benefit of in-system debugging is that you are able to debug a timing-accurate, post-implemented design in the actual system environment at system speeds. The limitations of in-system debugging includes somewhat lower visibility of debug signals compared to using simulation models and potentially longer design, implementation, and debug iterations, depending on the size and complexity of the design.

The Vivado IDE provides several different ways to debug your design. You can use one or more of these methods to debug your design, depending on your needs. For more information, see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 20].

Board File

Introduction

The board file uses an XML format to define information about system-level boards that use or include Xilinx® devices. The information contained in the board file can be used by the Vivado® Design Suite, and Vivado IP integrator, to facilitate and validate connection of the Xilinx device to the board. This chapter discusses the different sections of the board file and their usage.

The examples shown in this appendix use the Xilinx Kintex®-7 KC705 evaluation board to show details of the board file. This evaluation board can be found at the following location in the Vivado Design Suite software installation:

```
<install_dir>/Vivado/<version>/data/boards/board_files/kc705/<board_version>
```

Where <*install_dir*> is the directory the Vivado Design Suite was installed into, <*version*> is the software version, and <*board_version*> is the latest version of the board.

TIP: *You can edit an existing Vivado Design Suite board file from the installation directory, using an XML editor or text editor, as an easy way to create a new Board file.*

The board file uses standard XML elements to define the board. As such, XML tags are used to define elements of the board, and must have opening <tags> and closing </tags>.

```
<board>
</board>
```

Elements of the Board file can have child elements, or nested elements, to define a hierarchy of elements as shown:

```
<board>
  <component>
    <pins>
    </pins>
  </component>
</board>
```

Elements without content, or without nested elements, can be self-closing using an alternate syntax, and do not require the closing </tag>. These elements use the following syntax:

```
<net index="17" typical_delay="6" min_delay="4" max_delay="8"/>
```

The closing tag is implied by the “/>” that finishes the line. You will see this syntax occasionally used in the examples from the KC705 Board file.

Elements of the board, defined by <tags>, may also have various attributes defined as:

```
<board name=”XYZ” version=”1.2”>
```

Attribute values must be in quotation marks (name="KC705").

IMPORTANT: *XML is case-sensitive, so <tags> and 'attributes=' must be entered as specified.*

For more information regarding XML standards and conventions, refer to http://www.w3.org/XML/, or another appropriate source.

Understanding the Platform Board Flow

The Vivado Design Suite lets you create projects using Xilinx Target Design Platform (TDP) boards, or boards defined in the Board file format, that have been added to a board repository. When you select a specific board, the Vivado design tools enable additional features to provide designer assistance as part of IP customization, and for creating IP integrator designs. See Chapter 10, *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 13].

Note: FMC cards are also supported by the board flow and use the same XML syntax as other board files. For more information, see the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 13].

Elements of the Platform Board Flow

The list of files used in the Vivado Design Suite platform board flow include the following:

- Board file: The Board file is the file described in this appendix, and must be named `board.xml`. It lists the components used on a system-level board, including the Xilinx device, defines the different operating modes supported by those components, lists the signal interfaces implemented by those components, lists the preferred IP to implement those interfaces in a design project, and maps the logical ports of the interface definition to the physical ports and component pins of the Xilinx device.

  Xilinx standard board definitions can be found at the following location in the Vivado Design Suite software installation:

  ```
  <install_dir>/Vivado/<version>/data/boards/board_files
  ```

  Where <*install_dir*> is the directory the Vivado Design Suite was installed into, and <*version*> is the software version.

  You can create user-defined Board files by using the Xilinx standard board definition files as a starting point for customization. User-defined or third-party Board files, and associated files, can be added to a board repository for use by the Vivado Design Suite by setting the following parameter when launching the Vivado tool:

  ```
  set_param board.repoPaths [list "<path1>" "<path2>" "..."]
  ```

  Where <*path*> is the path to a directory containing a single Board file and files referenced by the `board.xml` file, such as `part0_pins.xml` and `preset.xml`. The <path> can also specify a directory with multiple subdirectories, each containing a separate Board file. For example:

  ```
  set_param board.repoPaths [list "C:/Data/usrBrds" "C:/Data/othrBrds"]
  ```

TIP: *You should define the* ***board.RepoPaths*** *parameter in your* `Vivado_init.tcl` *file, or soon after opening the Vivado Design Suite. For more information about the* `Vivado_init.tcl` *file refer to this* link *in the Vivado Design Suite Tcl Command Reference Guide (UG835) [Ref 4].*

- Pins file: Maps the component pin name on the Xilinx device, as found in the <port_map> of the Board file, to a physical pin location on the device package. This facilitates I/O assignment of signals coming into the Xilinx device to pins on the packaged part. This file is located in the board repository, in the same folder or directory as the Board file.
- Preset file: Provides a list of predefined IP configuration options for the different IP used to implement bus interfaces in a design project. The preset file is used by the Vivado Design Suite when the IP is customized from the IP catalog and added into the design. This file is located in the board repository, in the same folder or directory as the Board file.

- Interface file: Defines the logical ports and attributes of the signals that make up the interface file. A bus interface is a grouping of signals that share a common function. Interface definitions provide the capability to group functional signals to quickly define connections between IP in a Vivado IP integrator block diagram. For more information refer to this link in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 13].

  Xilinx standard interface definitions can be found at the following location in the Vivado Design Suite software installation:

  ```
  <install_dir>/Vivado/<version>/data/ip/interfaces
  ```

  You can also define custom interfaces using the Vivado IP packager, as described here in the *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118) [Ref 12].

- IP file: The IP definition is stored in an XML file based on the IP-XACT standard, `component.xml`, which includes a list of logical ports and bus interfaces found on the IP core, that can be connected to the interfaces implemented by the system-level board.

  Xilinx IP definitions can be found in the Vivado Design Suite software installation:

  ```
  <install_dir>/Vivado/<version>/data/ip/interfaces
  ```

Defining Board Files

Board

The <board> tag is the root of the board file. It includes attributes to identify basic information about the board.

```
<board schema_version="2.1" vendor="xilinx.com" name="kc705" display_name="Kintex-7
KC705 Evaluation Platform" url="www.xilinx.com/kc705" preset_file="preset.xml"
image="kc705_board.jpeg">
  <file_version>1.2</file_version>
  <description>Kintex-7 KC705 Evaluation Platform</description>
```

The attributes of the <board> tag, and their usage are:

Table A-1: **<board> Attributes and Tags**

Tag	Usage	Example (KC705)
schema_version=	Identifies the schema version of the board file. The latest version of the schema is 2.1.	`schema_version="2.1"`
vendor=	Web address of the board provider.	`vendor="xilinx.com"`
name=	Short name, which forms part of the key for selection of the board. This is different from the board description described later in the chapter.	`name="kc705"`
display_name=	Name as given in the **Display Name** column in the list of boards displayed in the Select Device dialog box in the Vivado IDE.	`display_name="Kintex-7 KC705 Evaluation Platform"`
url=	Web address for board specific information.	`url="www.xilinx.com/kc705"`
preset_file=	Preset file name, which is used to list all presets for each interface as applicable. For more information on how the preset_file is organized, please refer to Understanding Preset Files.	`preset_file="preset.xml"`
image=	Specifies a JPG file to be used when displaying the board in the **Project Summary** window in the Vivado IDE.	kc705_board.jpeg
<file_version>	Specifies the version of the Board file as a *version.revison* number. The <file_version> should be increased for any changes made to the current board file: • Update the *revision* for minor changes. • Update the *version* for significant changes.	1.2
<description>	A short description of the board defined by the file.	Kintex-7 KC705 Evaluation Platform

The following attributes and tags are mandatory when defining the <board>: schema_version, vendor, name, display_name, <file_version>, and <compatible_board_revisions>.

TIP: *The "preset_file=" attribute is optional, but is required to support the generic preset mechanism. See Understanding Preset Files for more information.*

First-level Tags in the Board File

The following table lists the first-level tags that can be nested under the <board> tag of schema version 2.1 of Board file:

Table A-2: **First-level Tags**

Tag	Usage/Description
<compatible_board_revisions>	Lists all revisions of the board to which this board file applies. See Compatible Board Revisions for details.
<parameters>	Parameters define features or properties of the board. See Parameters for details.
<jumpers>	Defines jumpers found on the board. See Jumpers for details.
<components>	Defines the various components present on the board. Components include FPGA devices, DDR, QSPI flash, FMC, etc. For more information, please refer Components.
<jtag_chains>	Boards can have multiple JTAG chains. Each chain can include several components as detected by Vivado Hardware Tools. The <jtag_chains> tag identifies the position of each component in a JTAG chain. For more information, see JTAG Chains.
<connections>	Describes connections between components ex: part0(FPGA) and LED. See Connections for details.
<ip_associated_rules>	Limits the choices of available board interfaces for specific IP. See IP Associated Rules.

Compatible Board Revisions

This tag lists the compatible revisions of the board that the current Board file applies to. Changes to the physical board may also trigger changes in board file, and therefore a new board <file_version>. However, revisions to the Board file may not require revisions to the physical board; and revisions to the physical board can include changes that do not necessitate an updated board file. Therefore it is possible for a board file to support multiple revisions of a physical board.

TIP: *Revisions to the board are possible without triggering revisions to the board file. Therefore a specific board file can be used to define multiple board revisions.*

The <compatible_board_revisions> tag includes one or more <revision> tags that list the supported board revisions:

```
<revision id="0">1.1</revision>
```

The <revision> tag includes an index "id" for each revision listed in the <compatible_board_revisions > tag. In the following example, the id is "0", and “1.1” is a supported revision of the board.

```
<compatible_board_revisions>
  <revision id="0">1.1</revision>
</compatible_board_revisions>
```

Parameters

The <parameters> tag is used to list miscellaneous parameters of the board. It includes one or more nested <parameter> tags that define different features or properties of the board.

Each <parameter> includes multiple attributes as defined in parameter Attributes.

```
<parameters>
  <parameter name="heat_sink_type" value="medium" value_type="string"/>
  <parameter name="heat_sink_temperature" value_type="range" value_min="20.0"
  value_max="30.0"/>
</parameters>
```

Table A-3: **<parameter> Attributes**

Tag	Usage/Description	Example (KC705)
name=	Name of an interface parameter used to configure connected IP cores.	name=“heat_sink_temperature”
value_type=	Type of parameter: string or range	value_type=“range”
value=	Defines the value of string-type parameter.	value=“medium”
value_min=/value_max=	Min and Max values of range-type parameter.	value_min="20.0" value_max="30.0"

Jumpers

The <jumpers> section lists all the jumpers present on the board, that can affect the <components> or <interfaces> on the board. The <jumpers> tag includes multiple nested <jumper> tags.

TIP: *Please note that switches on the board are also defined using the <jumpers> tag.*

```
<jumpers>
  <jumper name="SW13_M0" default_value="false">
    <description>Impacts connection between flash_qspi and flash_bpi.If
    value=true, flash_qspi will be enabled</description>
  </jumper>
  <jumper name="SW13_M1" default_value="true">
    <description>Impacts connection between flash_qspi and flash_bpi.If
    value=true, flash_bpi will be enabled</description>
  </jumper>
</jumpers>
```

A short description of the attributes and tags of the <jumper> tag are provided below.

Table A-4: **<jumper> Attributes and Tags**

Tag	Usage/Description	Example (KC705)
name=	Name of the jumper or switch on the board.	name="SW13_M0"
default_value=	Default value of the jumper or switch.	default_value="false"
<description>	A short note on how this jumper impacts different connections on board.	<description>Impacts connection between flash_qspi and flash_bpi. If value=true, flash_qspi will be enabled. </description>

Components

IMPORTANT: *The <component> section forms a very important part of the board file because it defines the components found on the board, as well as different operating modes of the components, and the settings needed to enable these modes.*

This section gives a list of all the components present on the board, as well as details such as part name, type of component, and vendor. Some examples of components include Xilinx FPGA, DDR3, Quad SPI flash, Ethernet Phy, LED, and DIP Switches. The <components> section includes one or more nested <component> tags.

```
<components>
  <component name="part0" display_name="Kintex-7 KC705 Evaluation Platform"
  type="fpga" part_name="xc7k325tffg900-2" pin_map_file="part0_pins.xml"
  vendor="xilinx" spec_url="www.xilinx.com/kc705">
    <description>FPGA part on the board</description>
  </component>
  <component name="ddr3_sdram" display_name="DDR3 SDRAM" type="chip" sub_type="ddr3"
  major_group="External Memory" part_name="MT8JTF12864HZ-1G6G1" vendor="Micron"
  spec_url="www.micron.com/memory">
    <description>1 GB DDR3 memory SODIMM </description>
  </component>
</components>
```

In the KC705 board file the first declared component is "part0", which is the Xilinx FPGA device. Xilinx devices on the board, listed as "fpga" type components, should be named sequentially starting with part0. Additional <component> elements define the other components on the board, and any interfaces needed to connect from the Xilinx device to the board component.

The different attributes and <tags> of the <component> tag are explained in the following table:

Table A-5: **<component> Attribute and Tags**

Tag	Usage/Description	Example (KC705)
name=	Name of the component.	name="part0"
display_name=	The name displayed in the Board tab of the Vivado IP integrator.	`display_name="Kintex-7 KC705 Evaluation Platform"`
type=	Type of part: "fpga" - Specifies the Xilinx FPGA parts on the board. "chip" - Defines all components which have a chip on the board, except FPGAs or FMC connectors. Examples of chips include ddr3_sdram and linear_flash. "connector" - Defines FPGA Mezzanine Card (FMC) connectors.	type="fpga"
sub_type=	Subtype of the component	sub_type="ethernet"
major_group=	Major group that the component is a part of. The Board tab in the Vivado IP integrator organizes components of the board according to the major_group. See Figure A-1.	major_group="ethernet"
part_name=	Part identifier. ***Note:*** For type=fpga, the part_name must be a valid Xilinx supported part, or the board files will not be loaded into the Vivado Design Suite.	part_name="xc7k325tffg900-2"
pin_map_file=	A file where the constraints for each pin are provided. Refer to Pin Map for more information.	pin_map_file="part0_pins.xml"
vendor=	Vendor of the part.	vendor="xilinx"
spec_url=	URL for the device specification or other information related to the part.	spec_url="www.xilinx.com/kc705"
<description>	Short description of the component.	<description> FPGA device on the board. </description>
<parameters>	Lists parameter name and value pairs.	Refer to Parameters for details and examples.
<pins>	List of pins identified for the component.	See Pins for details and examples.
<component_modes>	Modes specific to the defined <component>.	See Component Modes for details and examples.
<interfaces>	Defines the interfaces implemented by the specified component.	See Interfaces for more information and examples.

The <components> defined in the Board file are listed in the Board tab of the Vivado IP Integrator, as shown in Figure A-1. The components are grouped according to the 'major_group=' attribute of the <component> element, and the 'display_name=' is displayed.

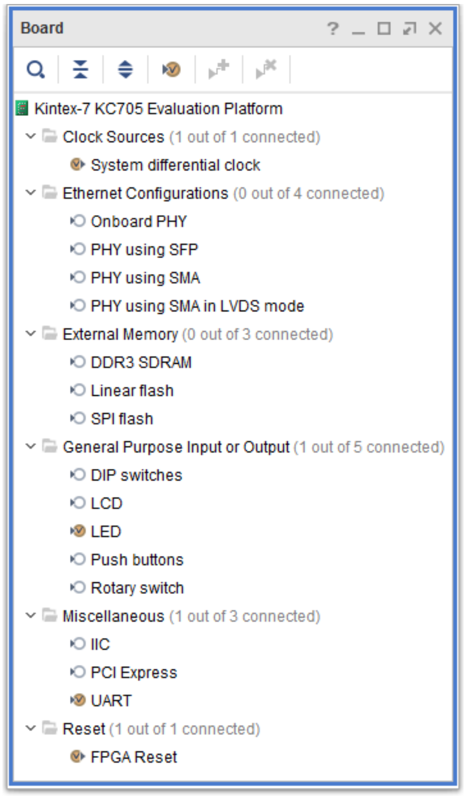

Figure A-1: **Board Window in Vivado IP Integrator**

Double-clicking a component in the Board tab opens the **Connect Board Component** dialog box in the Vivado IP integrator. This lets you select the preferred IP to add into the design canvas of the block diagram, to implement the necessary signal interfaces to connect to the component on the board.

Parameters

Parameters of a <component> are used to specify added details of the component, like clock frequency for clock components. The <parameters> section can include one or more <parameter> tags nested within. Each <parameter> has a "name" and "value" attribute pair.

```
<parameters>
  <parameter name="frequency" value="200000000" /parameter>
</parameters>
```

IMPORTANT: *The <parameters> for the <board> objects have different attributes from the <parameters> for <component> objects.*

Pins

The <pins> section lists all the pins on the defined <component>, as well as properties associated with those pins. The <pins> section can include one or more <pin> tags nested within.

The same properties can be defined with the <component pin_map_file=""> attribute. However, the property values defined in the <pins> section take precedence over the property values defined in the "pin_map_file=".

TIP: *You can use the <pins> section to define all the pins of a <component>, and eliminate the need for a "pin_map_file".*

The <pins> section lets you override the general pin properties defined in the "pin_map_file" with specific property values for specific circumstances. For instance, when pins with different properties, like IOSTANDARD, share the same pins of an FPGA (or any other component) through a switch or a jumper, the pin properties can be defined in the <pins> section to override the pin properties defined in the "pin_map_file" for the same pin.

```
<pins>
  <pin index="0" name="rs232_uart_USB_TX" iostandard="LVCMOS25"/>
  <pin index="1" name="rs232_uart_USB_RX" iostandard="LVCMOS25"/>
</pins>
```

Component Modes

The <component_modes> section describes the different modes in which a component can be used. The <component_modes> section can include one or more <component_mode> tags nested within. Each component mode includes details like associated interfaces, preferred IP, and enabling dependencies.

```
<component name="phy_onboard" display_name="Onboard PHY" type="chip"
sub_type="ethernet" major_group="Ethernet Configurations"
part_name="M88E1111_BAB1C000" vendor="Marvell" spec_url="www.marvell.com">
  <description>PHY on the board</description>
  <parameters>
    <parameter name="devicetree_vendor" value="marvell"/>
  </parameters>
  <component_modes>
    <component_mode name="mii" display_name="MII mode">
      <description>To enable this mode jumpers need to be {J29_P1_P2 true}
      {J30_P1_P2 true} {J64 false}</description>
      <interfaces>
        <interface name="mii" order="0"/>
        <interface name="mdio_io" order="1" optional="true"/>
        <interface name="phy_reset_out" order="2" optional="true"/>
      </interfaces>
      <preferred_ips>
        <preferred_ip vendor="xilinx.com" library="ip" name="axi_ethernet"
        order="0"/>
      </preferred_ips>
    </component_mode>
    <component_mode name="gmii" display_name="GMII mode">
      <description>To enable this mode jumpers need to be {J29_P1_P2 true}
      {J30_P1_P2 true} {J64 false}</description>
      <interfaces>
        <interface name="gmii"/>
        <interface name="mdio_io"/>
        <interface name="phy_reset_out" optional="true"/>
      </interfaces>
      <preferred_ips>
        <preferred_ip vendor="xilinx.com" library="ip" name="axi_ethernet"
        order="0"/>
      </preferred_ips>
    </component_mode>
</component>
```

When one mode of the component is selected in the Board tab, all the interfaces listed in this mode are automatically added in IPI. Order for interface in a mode defines the order in which the interfaces should be connected. If no order is mentioned, interfaces will be added in IPI in the order mentioned in list.

For <interfaces> listed under a <component_mode>, the ‘optional=’ attribute helps in the filtering of IP when you connect the interface in the Board tab of Vivado IP integrator. The default value is “optional=false", meaning that the IP must have this interface listed for the mode being used. If “optional=true” the interface is not required for the specified mode.

When “optional=true", any IP which has the required interfaces, but not the optional interfaces, will also be listed for use with the component mode being used.

```
<interfaces>
  <interface name="mii" order="0"/>
  <interface name="mdio_io" order="1" optional="true"/>
  <interface name="phy_reset_out" order="2" optional="true"/>
</interfaces>
```

TIP: *Preferred IPs mentioned in <component_modes> have a higher priority than the ones mentioned in individual <interfaces>.*

The <enablement_dependencies> list the jumper settings required to enable a specific <component_mode>. The information regarding the jumper settings to use on the board, based on selected component modes, is available in the Vivado Design Suite.

```
<enablement_dependencies>
  <jumpers>
    <jumper name="J29_P1_P2">true</jumper>
    <jumper name="J30_P1_P2">true</jumper>
    <jumper name="J64">false</jumper>
    </jumpers>
</enablement_dependencies>
```

Interfaces

IMPORTANT: *Interfaces can only be defined inside a <component> of type=”fpga”.*

The interfaces section provides a listing of all the physical interfaces available on a <component>. The <interfaces> section contains one or more <interface> tags nested within. An interface is defined by multiple ports through use of the <port_map> tag. Interfaces can be defined only inside a <component> of "type=fpga". For more information refer to Port Map.

The following is a partial example of the dip_switches_4bits interface definition from the KC705 board definition file:

```
<interfaces>
  <interface mode="master" name="dip_switches_4bits"
  type="xilinx.com:interface:gpio_rtl:1.0" of_component="dip_switches">
    <description>4-position user DIP Switch</description>
    <preferred_ips>
      <preferred_ip vendor="xilinx.com" library="ip" name="axi_gpio" order="0"/>
    </preferred_ips>
    <port_maps>
      <port_map logical_port="TRI_I" physical_port="dip_switches_tri_i" dir="in"
      left="3" right="0">
        <pin_maps>
          <pin_map port_index="0" component_pin="GPIO_DIP_SW0"/>
          <pin_map port_index="1" component_pin="GPIO_DIP_SW1"/>
          <pin_map port_index="2" component_pin="GPIO_DIP_SW2"/>
          <pin_map port_index="3" component_pin="GPIO_DIP_SW3"/>
        </pin_maps>
      </port_map>
    </port_maps>
  </interface>
  <interface …>
    ...
    ...
  </interface>
</interfaces>
```

Interface

IMPORTANT: *Interface names must be defined using all lower case letters.*

The following are attributes and <tags> of the <interface>.

Table A-6: **<interface> Attributes and Tags**

Tag	Usage/Description	Example (KC705)
mode=	Indicates the logical direction of the interface. Typically, the mode will be set to "master", but in cases like clocks and reset where the logical direction is for the signals to be input to the FPGA, the mode is marked as "slave".	master
name=	A unique name to identify the interface definition in the board file. This name will also be used to drive connection automation, and be seen on the connected port in the block diagram.	dip_switches_4bits
type=	Specifies the type of the interface from a standard set of interface types supported by the Vivado Design Suite. These standard bus interfaces are defined on Xilinx IP cores, to enable easy it connection of the IP or block design to the board. The list of available bus interface types can be found in the Vivado Design Suite installation: `<install_location>/Vivado/<version>/data/ip/interfaces`	xilinx.com:interface:gpio_rtl:1.0
of_component=	Names the associated component from the <components> section.	dip_switches
<description>	A brief description of the interface.	4-position user DIP Switch
<preferred_ip>	Lists the preferred IP to connect to, in VLNV (or VLN) format. ***Note:*** The version of the IP is not required as the Vivado tool will pick the latest version.	vendor="xilinx.com" library="ip" name="axi_gpio" order="0"
order=	Specifies the priority of the `preferred_ip` for the interface. The priority counts down, with 0 being the highest priority.	0
preset_proc=	Specifies predefined configuration options for IP implementing the specified interface. Refer to Understanding Preset Files for more information.	preset_proc="emc_preset"
<port_map>	Maps the logical pins of an interface to the physical ports of the Xilinx device.	See Port Map for details and examples.

Port Map

Each interface is further broken down into individual port maps. These port maps serve as a map of a logical port, that is defined in the interface, with a physical port, that relates to a physical package pin on the Xilinx device.

```
<port_map logical_port="TRI_I" physical_port="dip_switches_tri_i" dir="in" left="3"
right="0">
  <pin_maps>
    <pin_map port_index="0" component_pin="GPIO_DIP_SW0"/>
    <pin_map port_index="1" component_pin="GPIO_DIP_SW1"/>
    <pin_map port_index="2" component_pin="GPIO_DIP_SW2"/>
    <pin_map port_index="3" component_pin="GPIO_DIP_SW3"/>
  </pin_maps>
  </port_map>
```

Table A-7: **<port_map> Attributes and Tags**

Tag	Usage/Description	Example (KC705)
logical_port=	Logical port names are found on the bus interface definition. Predefined interfaces can be found in the Vivado Design Suite installation. For example, the GPIO interface definition is found at: <install_location>\Vivado\<version>\data\ip\interfaces\gpio_v1_0	TRI_I
physical_port=	Provides the mapping to port names defined on the board interface in the subsequent section. physical_port can be a std_logic or std_logic_vector.	dip_switches_tri_i
dir=	Each port has a direction. Allowed values are **in**, **out**, and **inout**.	in
left=	The high index on the port. For example, a 4-bit bus port [3:0] will be marked as 3.	3
right=	The low index for a port. For example, a 4-bit bus port [3:0] will be marked as 0.	0
<pin_maps>	Maps the physical port of a Xilinx device to a specific pin of the packaged part.	See Pin Map for details and examples.

Pin Map

In the <pin_map> section, each physical port is broken down into one or more individual pins. The number of pins in the pin map is determined by the width of the port being mapped. Pins can be shared across different physical ports of the interfaces they are defined in.

Each <pin_map> has a port_index attribute, that maps to an index of the bus port, and a component_pin attribute, that maps to a package pin on the Xilinx device. These are defined as follows:

Table A-8: **<pin_map> Attributes**

Tag	Usage/Description	Example (KC705)
port_index=	Indicates the index of a bus port that is defined in the <port_map>. This is a numeric value within the range defined by the width of port.	3
component_pin=	Name of the component pin on the Xilinx device. The component_pin name maps to the name= attribute in the `part0_pins.xml` file of the FPGA-type <component>. The `part0_pins.xml` file is located in the same folder as the Board file.	GPIO_DIP_SW0

The Pin Map file, commonly named `part0_pins.xml`, lists the pin names of the Xilinx device, or "fpga" type <component>, and defines the IOSTANDARDs and package pin locations for these component pins. The format of the pins defined in the Pin Map file is as follows:

```
<part_info part_name="xc7k325tffg900-2">
  <pins>
    <pin index="0" name="GPIO_DIP_SW0" iostandard="LVCMOS25" loc="Y29"/>
    <pin index="1" name="GPIO_DIP_SW1" iostandard="LVCMOS25" loc="W29"/>
    <pin index="2" name="GPIO_DIP_SW2" iostandard="LVCMOS25" loc="AA28"/>
    <pin index="3" name="GPIO_DIP_SW3" iostandard="LVCMOS25" loc="Y28"/>
  </pins>
```

In the Pin Map file, the following attributes are used to define I/O related constraints for each of the <pins> found on the Xilinx device:

Table A-9: **Attributes of the Pin Map File**

Tag	Usage/Description	Example (KC705)
index=	An index assigned to the <pin> object in the Pin Map file.	0
name=	The component pin name on the Xilinx device, used in the Board file.	GPIO_DIP_SW2
iostandard=	A valid IOSTANDARD for the Xilinx device pin, as defined by the board designer. Valid values include IOSTANDARDs supported by the Vivado Design Suite for the specific component pin.	LVCMOS25
loc=	The pin location on the Xilinx device package.	Y29

JTAG Chains

This lists the different JTAG chains available on the defined board. Each chain is listed under <jtag_chain> along with a name for the chain, and the <position> tag that defines the name and position of components in the chain:

```
<jtag_chains>
  <jtag_chain name="chain1">
    <position name="0" component="part0"/>
  </jtag_chain>
</jtag_chains>
```

The <jtag_chain> tag specifies the name of the chain with the name= attribute.

The <position> tag lists each component in the <jtag_chain>. Details are given in below table:

Table A-10: **<position> Attributes**

Tag	Usage/Description	Example (KC705)
name=	Indicates the sequence of components in the jtag chain.	0
component=	Indicates the component on the board.	Part0

Connections

The <connections> section defines the connections between different components. The <connection> tag identifies two components associated with a connection. The <connection_map> tag describes the bus connection between the two components. The details of a <connection> are used by the Vivado Design Suite to look up corresponding constraints in the part0_pins.xml file when one of the components is the FPGA type <component>.

```
<connections>
  <connection name="part0_dip" component1="part0" component2="dip_switches">
    <connection_map name="part0_dip_1" typical_delay="5" c1_st_index="0"
    c1_end_index="3" c2_st_index="0" c2_end_index="3"/>
  </connection>
</connections>
```

A <connection> can have the following attributes:

Table A-11: **<connection> Attributes**

Tag	Usage/Description	Example (KC705)
name=	Name given to the connection.	part0_dip
component1=	The first component in the connection.	part0
component2=	The second component in the connection.	dip_switches

The <connection_map> has the following attributes:

Table A-12: **<connection_map> Attributes**

Tag	Usage/Description	Example (KC705)
typical_delay=	The delay on the connection between components.	5
c1_st_index=	This is the <pin> index of the starting pin in the connection for component1. ***Note:*** If component1 is an FPGA, this index is taken from the `part0_pins.xml` file.	0
c1_end_index=	This is the <pin> index of the final pin in the connection for component1.	3
c2_st_index=	This is the <pin> index of the starting pin for component2.	0
c2_end_index=	This is the <pin> index of the final pin in the connection for component2.	3

IP Associated Rules

The <ip_associated_rules> tag is used to define a preferred board interface, or a prioritized list of board interfaces that can be assigned to the IP interfaces on a specific IP. This tag is new in the 2.1 schema version of the Board file.

The Designer Assistance feature of the Vivado IP integrator tool lists the available board interfaces for a given IP interface. See the following link in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) for more information on Designer Assistance. The <ip_associated_rules> lets you define which board interfaces can be applied to a specific IP interface.

TIP: *The <ip_associated_rules> tag can be used to define any IP interface, but is most useful for clocks and resets which typically have more defined board interfaces.*

```
<ip_associated_rules>
  <ip_associated_rule name="default">
    <ip vendor="xilinx.com" library="ip" name="axi_ethernet" version="*"
ip_interface="mgt_clk">
      <associated_board_interfaces>
  <associated_board_interface name="sgmii_mgt_clk" order="1"/>
  <associated_board_interface name="sma_mgt_clk" order="0"/>
      </associated_board_interfaces>
    </ip>
    <ip vendor="xilinx.com" library="ip" name="axi_ethernet" version="*"
ip_interface="phy_rst_n">
      <associated_board_interfaces>
  <associated_board_interface name="phy_reset_out" order="0"/>
      </associated_board_interfaces>
    </ip>
    <ip vendor="xilinx.com" library="ip" name="gig_ethernet_pcs_pma" version="*"
ip_interface="gtrefclk_in">
      <associated_board_interfaces>
  <associated_board_interface name="sgmii_mgt_clk" order="1"/>
  <associated_board_interface name="sma_mgt_clk" order="0"/>
      </associated_board_interfaces>
    </ip>
  </ip_associated_rule>
</ip_associated_rules>
```

IP Associated Rule

The <ip_associated_rule> tag defines the specific rules associated with the specified IP. It supports the `name=` tag to define the name of the rule.

IMPORTANT: *Currently only one rule named "default" is supported, so there can be only one pair of <ip_associated_rule name="default"></ip_associated_rule> opening and closing tags defined in the Board file.*

IP

The <ip> tag defines an IP, and interface, that the associated rules apply to. The <ip> tag identifies the Vendor, Library, Name, and Version (VLNV) of the IP, and a specific interface on the IP.

```
<ip vendor="xilinx.com" library="ip" name="axi_ethernet" version="*"
ip_interface="mgt_clk">
  <associated_board_interfaces>
    <associated_board_interface name="sgmii_mgt_clk" order="1"/>
    <associated_board_interface name="sma_mgt_clk" order="0"/>
  </associated_board_interfaces>
</ip>
```

TIP: *The <ip_associated_rules> can define multiple <ip> tags identifying multiple IP in the Vivado IP catalog, as well as identifying different interfaces on a single IP.*

An <ip> can have the following attributes or tags:

Table A-13: **<ip> Attributes and Tags**

Tag	Usage/Description	Example (KC705)
vendor=	The vendor name associated with the IP. Cores provided by Xilinx have the ”xilinx.com” vendor identity.	vendor="xilinx.com"
library=	Specifies the IP library that the core can be found in.	library="ip"
name=	Specifies the name of the IP core in the catalog.	name="axi_ethernet"
version=	Specifies the version of the IP that the rules apply to. ”*”can be used to indicate all versions of the IP.	”*”
ip_interface=	Specifies the name of the interface on the IP core to associate with interfaces defined in the Board file.	ip_interface="mgt_clk"
<associated_board_interfaces>	Specifies the name and priority of the board interface.	

The <associated_board_interface> tag defines a prioritized list of board interfaces that can be assigned to the associated IP interface.

```
<associated_board_interfaces>
  <associated_board_interface name="sgmii_mgt_clk" order="1"/>
  <associated_board_interface name="sma_mgt_clk" order="0"/>
</associated_board_interfaces>
```

The <associated_board_interface> has the following attributes:

Table A-14: **<associated_board_interface> Attributes**

Tag	Usage/Description	Example (KC705)
name=	Name of board interface associated with the IP interface.	name="sgmii_mgt_clk"
order=	The order of preference for assigning the board interface. The Designer Assistance feature of the Vivado IP integrator will automatically assign a board interface with order=0. Other board interfaces will be listed in the specified order from lowest to highest.	0

Understanding Preset Files

Preset file helps customize an IP core in a particular configuration. The PS7, axi_emc, and Memory IP for DDR3_SDRAM use the preset_file feature when a linear_flash or ddr3_sdram interface is selected in the Board tab in the Vivado IP integrator.

Preset Files use the XML format. The preset_file is defined for a specific Board file, and can be used to apply presets to multiple IP.

```
<ip_presets schema="1.0">
  <ip_preset preset_proc_name="emc_preset">
    <ip vendor="xilinx.com" library="ip" name="axi_emc" version="3.0">
      <user_parameters>
        <user_parameter name="CONFIG.C_INCLUDE_DATAWIDTH_MATCHING_0" value="1"/>
        <user_parameter name="CONFIG.C_MAX_MEM_WIDTH" value="16"/>
        <user_parameter name="CONFIG.C_MEM0_TYPE" value="2"/>
        <user_parameter name="CONFIG.C_MEM0_WIDTH" value="16"/>
        <user_parameter name="CONFIG.C_TAVDV_PS_MEM_0" value="130000"/>
        <user_parameter name="CONFIG.C_TCEDV_PS_MEM_0" value="130000"/>
        <user_parameter name="CONFIG.C_TWPH_PS_MEM_0" value="12000"/>
        <user_parameter name="CONFIG.C_TWP_PS_MEM_0" value="70000"/>
        <user_parameter name="CONFIG.C_WR_REC_TIME_MEM_0" value="100000"/>
      </user_parameters>
    </ip>
  </ip_preset>
  <ip_preset preset_proc_name="ddr3_sdram_preset">
    <ip vendor="xilinx.com" library="ip" name="mig_7series">
      <user_parameters>
        <user_parameter name="CONFIG.XML_INPUT_FILE" value="mig.prj"
value_type="file"/>
      </user_parameters>
    </ip>
  </ip_preset>
</ip_presets>
```

IP_Presets

The <ip_presets> is the root of the preset_file, and defines the presets for one or more IP cores. The <ip_presets> can have one or more <ip_preset> tags nested within it.

```
<ip_presets schema="1.0">
```

Table A-15: **<ip_presets> Attribute**

Tag	Usage/Description	Example (KC705)
schema=	Identifies the schema version of the preset file. The current version of the schema is 1.0.	1.0

IP_Preset

The <ip_preset> section defines the preset configuration to be applied to specific IP cores.

```
<ip_preset preset_proc_name="emc_preset">
```

Table A-16: **<ip_preset> Attribute**

Tag	Usage/Description	Example (KC705)
preset_proc_name=	Identifies the preset process being defined. The preset_proc_name used here will also be specified in the Board file for interfaces that implement this preset process.	emc_preset

IP

Within the <ip_preset> the <ip> section defines the specific IP that the preset values will apply to.

```
<ip vendor="xilinx.com" library="ip" name="axi_emc" version="3.0">
```

Table A-17: **<ip> Attributes**

Tag	Usage/Description	Example (KC705)
vendor=	Address of the board provider's company website.	xilinx.com
library=	Library that the core is part of.	ip
name=	Name of the IP core.	axi_emc
version=	Version of the IP.	3.0

User Parameters

Within the <ip> section, the <user_parameters> and <user_parameter> tags define the various configuration presets to apply to the specified IP core.

```
<user_parameters>
  <user_parameter name="CONFIG.C_INCLUDE_DATAWIDTH_MATCHING_0" value="1"/>
  <user_parameter name="CONFIG.C_MAX_MEM_WIDTH" value="16"/>
  <user_parameter name="CONFIG.C_MEM0_TYPE" value="2"/>
</user_parameter>
```

Table A-18: **<user_parameter> Attributes**

Tag	Usage/Description	Example (KC705)
name=	Name of the pre-configured property of the IP core.	CONFIG.C_WR_REC_TIME_MEM_0
value=	Preset value of the property.	100000

Additional Files and Special Considerations

Memory IP 7 Series Support

Memory IP in Xilinx 7 series devices requires special handling. Board designers should test the Memory IP configuration on the board before adding the PRJ file into the board support area.

PS7 Presets

For PS7 IP cores, specifying the preset configuration is the same as any other supported IP in the catalog. The <user_parameters> must name the various preconfigured properties and their values in the preset_file.

```
<ip_preset preset_proc_name="ps7_preset">
  <ip vendor="xilinx.com" library="ip" name="processing_system7" version="*">
    <user_parameters>
      <user_parameter name="CONFIG.preset" value="ZC702"/>
      <user_parameter name="CONFIG.PCW_CAN0_PERIPHERAL_ENABLE" value="0"/>
    </user_parameters>
  </ip>
</ip_preset>
```

IP Bus Interfaces with Tri-state Ports

IP bus interface exposes three signals (I, O, and T) for tri-state ports. Based on the IP configuration, one or all three signals are exposed as single external ports via the tri-state buffer.

In the interface logical to physical port mapping, `<port_map>`, section only the exposed signal needs to be defined for GPIO, whereas for IO all three signals need to be mapped to a physical port.

Example

```
<interface mode="master" name="dip_switches_4bits"
type="xilinx.com:interface:gpio_rtl:1.0">
      <port_maps>
        <port_map logical_port="TRI_I" physical_port="dip_switches_tri_i"/>
      </port_maps>
</interface>

<interface mode="master" name="iic_main" type="xilinx.com:interface:iic_rtl:1.0">
      <port_maps>
        <port_map logical_port="SDA_I" physical_port="iic_main_sda_i"/>
        <port_map logical_port="SDA_O" physical_port="iic_main_sda_o"/>
        <port_map logical_port="SDA_T" physical_port="iic_main_sda_t"/>
        <port_map logical_port="SCL_I" physical_port="iic_main_scl_i"/>
        <port_map logical_port="SCL_O" physical_port="iic_main_scl_o"/>
        <port_map logical_port="SCL_T" physical_port="iic_main_scl_t"/>
      </port_maps>
</interface>
```

Ethernet Clock Handling

To differentiate ethernet clocks from regular clocks, the interface parameter 'TYPE' is defined in the schema. This parameter provides additional filtering while searching an appropriate IP for a board interface. The value of the TYPE parameter should be the same in the board interface and the IP interface.

For example, the `sgmii_mgt_clk` interface in the KC705 board has the TYPE parameter with the value defined as `ETH_MGT_CLK`. Similarly the IP `component.xml` file has the interface parameter `TYPE =ETH_MGT_CLK`.

See <vivado_install_dir>/data/ip/xilinx/gig_ethernet_pcs_pma_v15_0/component.xml as an example.

GT Location Constraint

For generating a GT location constraint, Ethernet related interfaces have a GT_LOC parameter in the board interface. For example, the `sgmii` interface of the KC705 board has GT_LOC= GTXE2_CHANNEL_X0Y9. Here the Ethernet IP assumes that if this parameter is not present in the board interface, then it is running in LVDS mode, so the IP customization will generate a pin location constraint (LOC) instead of the GT location constraint (GT_LOC).

Appendix B

Vivado Naming Conventions

Introduction

The following are the required naming conventions when working with the Vivado® Design Suite. Failing to follow these naming conventions might introduce potential risk to the design or the tool, and cause unpredictable behavior in the design flow.

- Source files names must start with a letter (A-Z, a-z) and must contain only alphanumeric characters (A-Z, a-z, 0-9) and underscores (_).
- Output files names must start with a letter (A-Z, a-z) and must contain only alphanumeric characters (A-Z, a-z, 0-9) and underscores (_).
- Project names must start with a letter (A-Z, a-z) and must contain only alphanumeric characters (A-Z, a-z, 0-9) and underscores (_).
- Project directory names must start with a letter (A-Z, a-z) and should contain only alphanumeric characters (A-Z, a-z, 0-9), tilde (~) and underscores (_).

CAUTION! *The Windows operating system has a 260 character limit for path lengths which can affect the Vivado tools. To avoid this issue, use the shortest possible names and directory locations when creating projects, defining IP or managed IP projects, or creating block designs.*

The following characters are not supported for project, file, or directory names:

- ! # $ % ^ & * ( ) ` ; < > ? , [ ] { } ' " |
- tab (\t)
- return (\r)
- new line (\n)
- / or \ (As part of the directory or file name rather than as a path delimiter)

The following character is not supported for directory names:

- . (dot as terminal character)

The following character is not supported for file or project names:

- @

Note: In the Vivado IDE, the @ character is not supported for new file or project names. The Vivado IDE does allow an existing file on disk that uses the @ character to be added to a project. The Vivado IDE can open a project that includes the @ character in the project name. Using the Tcl Console, you can create a project with a name that contains the @ character.

IMPORTANT: *Spaces in directory and file names are supported by the Windows operating system. However, you should avoid using spaces in order to preserve portability of the project or files between the Windows and Linux operating systems.*

The Vivado Design Suite supports the use of forward slashes (/) as path delimiters for both Windows and Linux platforms. Backslashes (\) are allowed as path delimiters on the Windows platform only.

Any characters not explicitly mentioned above are not supported for project, file, or directory names.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Documentation Navigator and Design Hubs

Xilinx Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter: `docnav`

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

References

1. *Vivado® Design Suite User Guide: Design Flows Overview* (UG892)
2. *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973)
3. *Vivado Design Suite User Guide: Using Tcl Scripting* (UG894)
4. *Vivado Design Suite Tcl Command Reference Guide* (UG835)
5. *Vivado Design Suite Tutorial: Design Flows Overview* (UG888)
6. *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893)
7. *Vivado Design Suite User Guide: Using Constraints* (UG903)
8. *ISE to Vivado Design Suite Migration Guide* (UG911)
9. *Vivado Design Suite User Guide: I/O and Clock Planning* (UG899)
10. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
11. *Vivado Design Suite User Guide: Designing with IP* (UG896)
12. *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118)
13. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)
14. *Vivado Design Suite Properties Reference Guide* (UG912)
15. *Vivado Design Suite User Guide: Synthesis* (UG901)
16. *Vivado Design Suite User Guide: High-Level Synthesis* (UG902)
17. *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906)
18. *Model Composer User Guide* (UG1262)
19. *Vivado Design Suite User Guide: Implementation* (UG904)
20. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
21. *Vivado Design Suite User Guide: Embedded Hardware Design* (UG898)
22. *Vivado Design Suite User Guide: Model-Based DSP Design Using System Generator* (UG897)
23. *UltraScale Architecture Libraries Guide* (UG974)
24. *Vivado Design Suite 7 Series FPGA and Zynq-7000 SoC Libraries Guide* (UG953)
25. Vivado Design Suite Documentation

Training Resources

Xilinx provides a variety of training courses and QuickTake videos to help you learn more about the concepts presented in this document. Use these links to explore related training resources:

1. Designing FPGAs Using the Vivado Design Suite 1 Training Course
2. Designing FPGAs Using the Vivado Design Suite 2 Training Course
3. Designing FPGAs Using the Vivado Design Suite 3 Training Course
4. Designing FPGAs Using the Vivado Design Suite 4 Training Course
5. Vivado Design Suite QuickTake Video: Using Vivado Design Suite with Revision Control
6. Vivado Design Suite QuickTake Video: Using the Vivado Timing Constraint Wizard
7. Vivado Design Suite QuickTake Video Tutorials

Please Read: Important Legal Notices