Vivado Design Suite User Guide

Designing IP Subsystems Using IP Integrator

UG994 (v2014.2) June 4, 2014

XILINX
ALL PROGRAMMABLE™

Revision History

The following table shows the revision history for this document.

Date	Version	Changes
06/04/2014	2014.2	Information updated on the topic **Exporting the Hardware Definition to SDK** (starting on page 42).
05/07/2014	2014.1	Added two new chapters, Using Tcl Scripts to Create IP Integrator Designs within Projects and Using the Board Flow in IP Integrator. Moved the Legal Notices to Appendix B.
12/18/2013	2013.4	No technical updates. Re-published for this release.
10/02/2013	2013.3	The following sections have been added: Chapter 3 - Parameter Propagation in IP Integrator, Chapter 4 - Debugging IP Integrator Designs, Chapter 5 – Migrating IP Integrator Designs from One Release to Another, Chapter 6 - Using Non-Project Mode in IP Integrator, Chapter 7 - Revision Control for IP Integrator Designs, Appendix A - Additional Resources
06/19/2013	2013.2	Initial Xilinx Release.

Table of Contents

Introduction

As FPGAs become larger and more complex, and as design schedules become shorter, use of third party IP and design reuse is becoming mandatory. Xilinx recognizes the challenges designers face, and to aid designers with design and reuse issues, has created a powerful feature within the Vivado® Design Suite called the Vivado IP integrator.

The Vivado IP integrator feature lets you create complex system designs by instantiating and interconnecting IP from the Vivado IP catalog on a design canvas. You can create designs interactively through the IP integrator canvas GUI or programmatically through a Tcl programming interface. Designs are typically constructed at the interface level (for enhanced productivity) but may also be manipulated at the port level (for precision design manipulation).

An interface is a grouping of signals that share a common function. An AXI4-Lite master, for example, contains a large number of individual signals plus multiple buses, which are all required to make a connection. If each signal or bus is visible individually on an IP symbol, the symbol will be visually very complex. By grouping these signals and buses into an interface, the following advantages can be realized. First, a single connection in IP integrator (or Tcl command) will make a master to slave connection. Next, the graphical representation of this connection will be simple – a single connection. Finally, Design Rule Checks (DRCs) that are aware of the specific interface can be run to assure that all the required signals are connected properly.

A key strength of IP integrator is that it provides a Tcl extension mechanism for its automation services so that system design tasks such as parameter propagation, can be optimized per-IP or application domain. Additionally, IP integrator implements dynamic, runtime DRCs to ensure, for example, that connections between the IP in an IP integrator design are compatible and that the IP themselves are properly configured.

Chapter 2 IP Integrator – Basic Operations

Overview

This chapter describes the basic operations and functionality of IP integrator.

Creating a Project

While entire designs can be created using IP integrator, the typical design will consist of HDL, IP, and IP integrator block designs. This section is an introduction to creating a new IP integrator-based design.

As shown in the figure below, you start by clicking on **Create New Project** in the Vivado® IDE graphical user interface (GUI) to create a new project. You can add VHDL or Verilog design files, any custom IP, and other types of design source files to the project using this wizard.

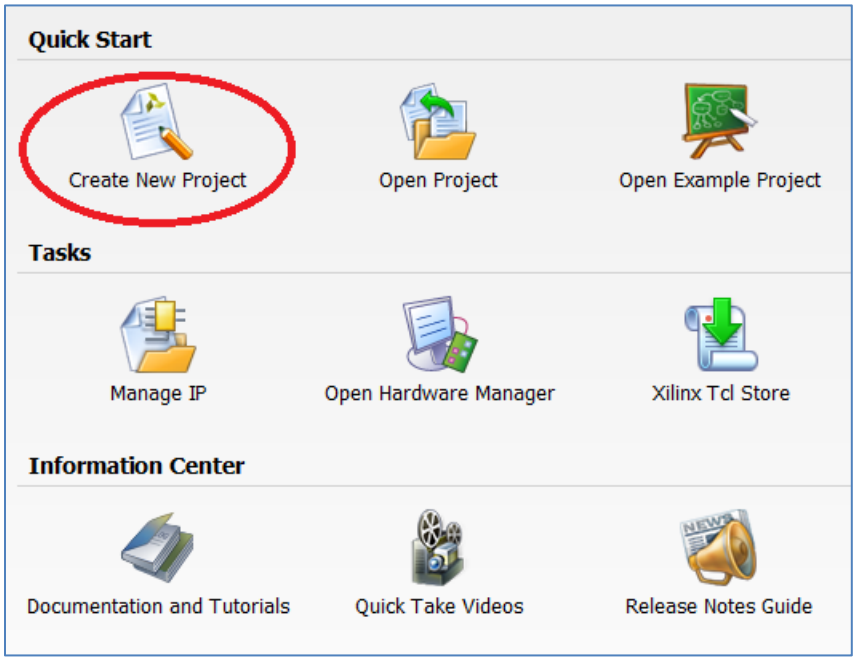


Figure 1: Creating a New Project

As shown in the figure below, you can also select the target device or a Xilinx target board. Vivado tools also support multiple versions of Xilinx target boards, so carefully select your target hardware.

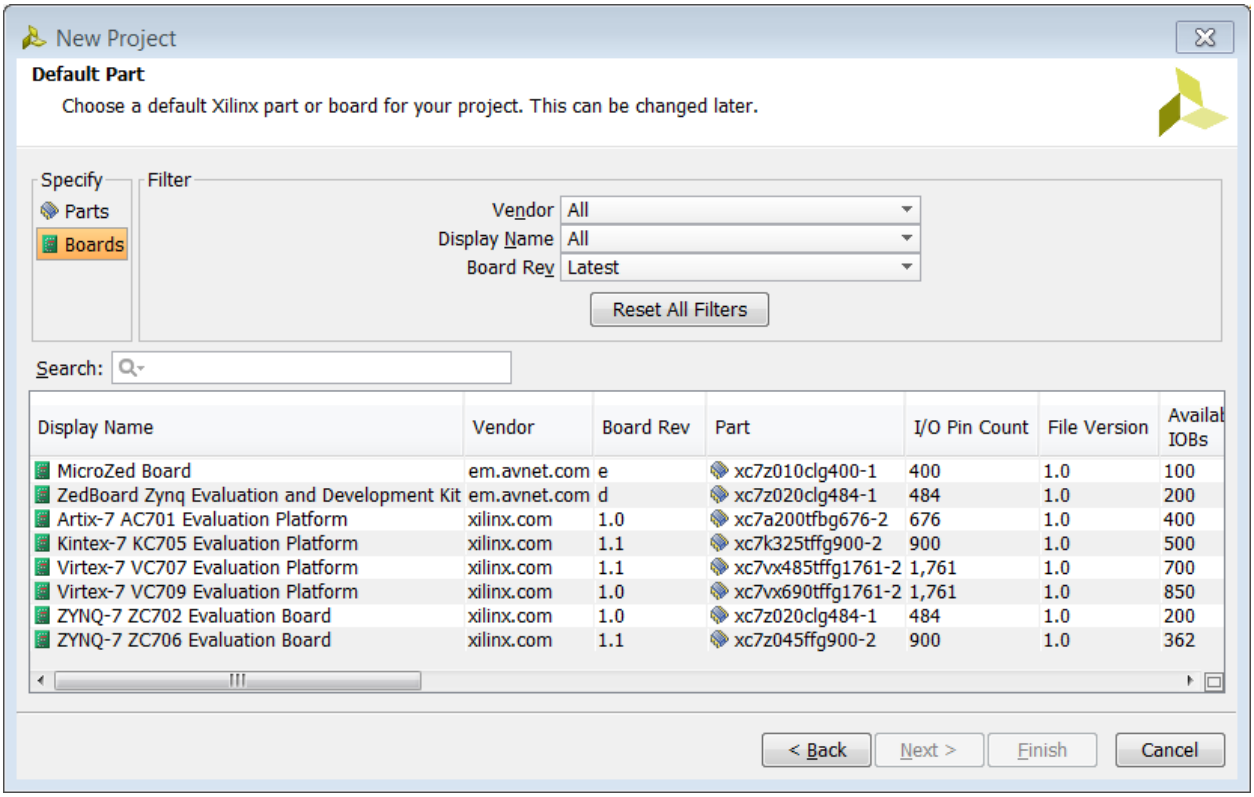


Figure 1: Choosing a New Project Target Device

> ***Note***: *You can perform the same actions by using the following Tcl commands. When displaying the Tcl commands in this document, the symbols< > are used to surround parameters that are specific to your design. The < > symbols should not be included in the command string.*

The Tcl equivalent commands are:

```
create_project xx <your_directory>/xx -part xc7k325tffg900-2
set_property board_part Xilinx.com:kc705:part0:1.0 [current_project]
set_property target_language VHDL [current_project]
```

Designing with IP Integrator

You create a new block design in the Flow Navigator by clicking on **Create Block Design** under the IP Integrator heading.

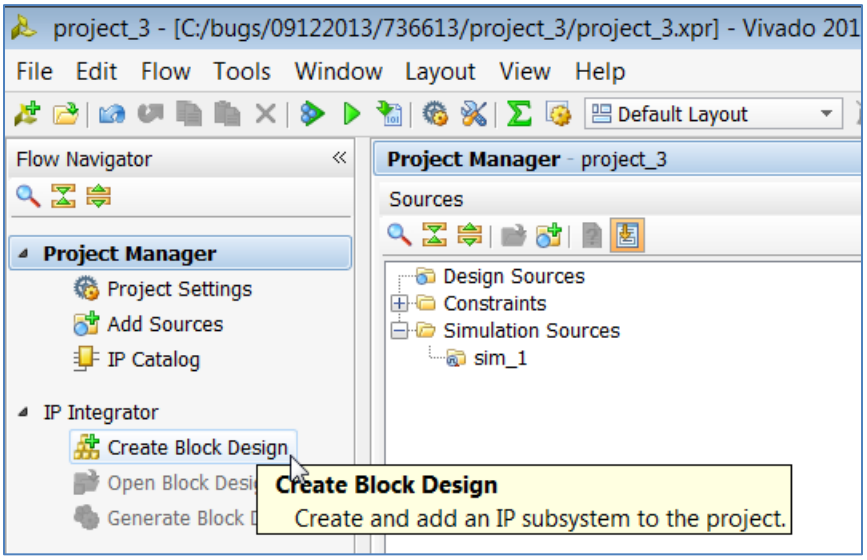


Figure 2: Creating a Block Design

The Tcl equivalent command is:

```
create_bd_design "<your_design_name>"
```

Re-sizing the IP Integrator Diagram

Once the design is created, a canvas is presented that you can use to construct your design. This canvas can be re-sized as much as possible by re-sizing panes in the Vivado IDE GUI. It is possible to move the diagram to a separate monitor by clicking on the **Float Window** button in the upper-right corner of the diagram. You can also double click on the **Diagram** tab at the upper-left corner of the diagram to increase the size of the diagram. When you double click the tab again, the view returns to the default layout.

Changing Layers

To display the layers, click the top-left icon in the Diagram window, as shown by the red circle in the following figure. You can select the Attributes, Nets and Interface connections that you want to view or hide by checking or un-checking the boxes against these.

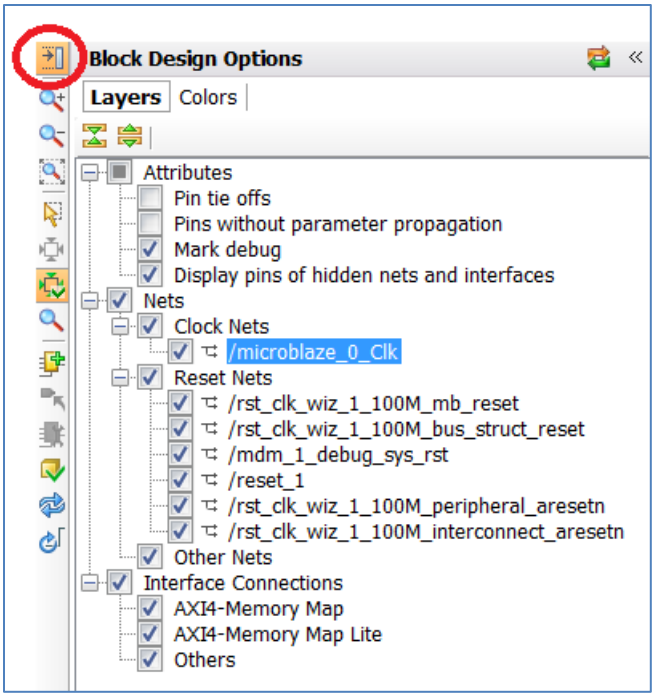


Figure 3: Viewing/Hiding Information on the IP Integrator Canvas

Changing the Window Background Color

You can change the background color of the diagram canvas from the default white color. As shown in the following figure, you can click the **Block Design Options >Colors** button in the upper-left corner of the diagram to change the color.

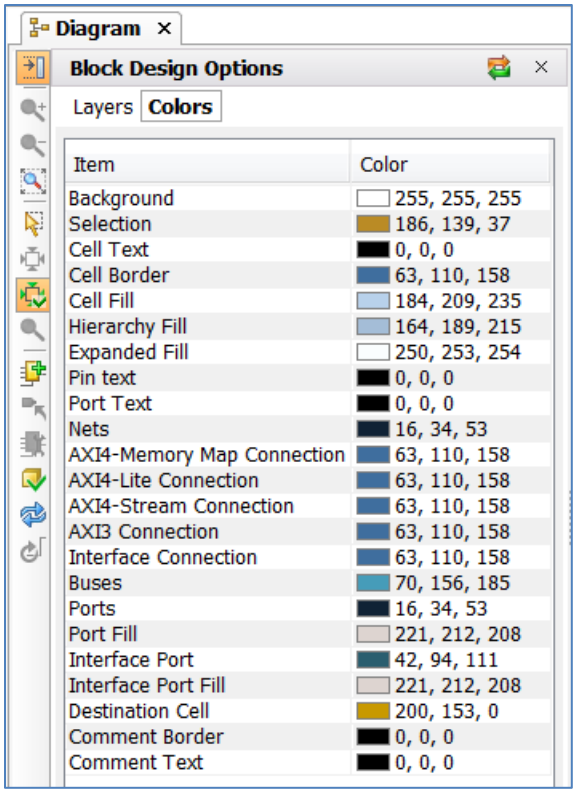


Figure 4: Changing the IP Integrator Background Color

Notice that you can control the colors of almost every object displayed in an IP integrator diagram. For example, changing the background color to 240,240,240 as shown above makes the background light gray. To hide the **Block Design Options**, either click the close button in the upper-right corner, or click the **Block Design Options** button again.

Using Mouse Strokes and the Left-Button Panel

A northwest stroke (lower-right to upper-left) is **Zoom Fit**

A southwest stroke (upper-right to lower-left) is **Zoom In**

A northeast stroke (lower-left to upper-right) is **Zoom Out**

A southeast stroke (lower-right to upper-left) is **Zoom Area**

The buttons on the left side of the canvas allow the following specific actions to be invoked:

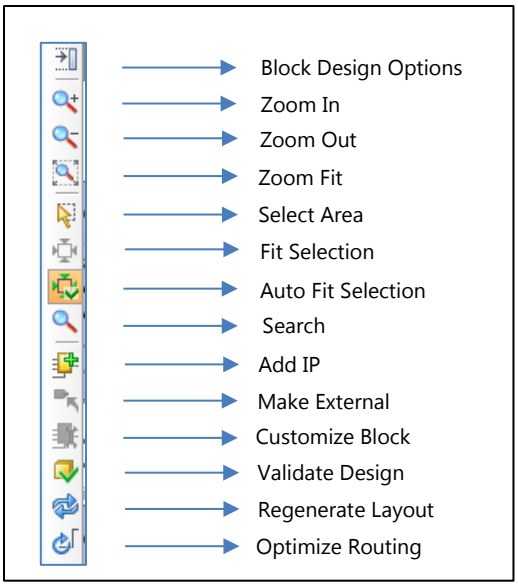


Figure 5: IP Integrator Action Buttons

Adding IP Modules to the Design Canvas

You can add IP modules to a diagram in the following ways:

1. Right click in the diagram and select **Add IP**. A searchable IP Catalog opens.

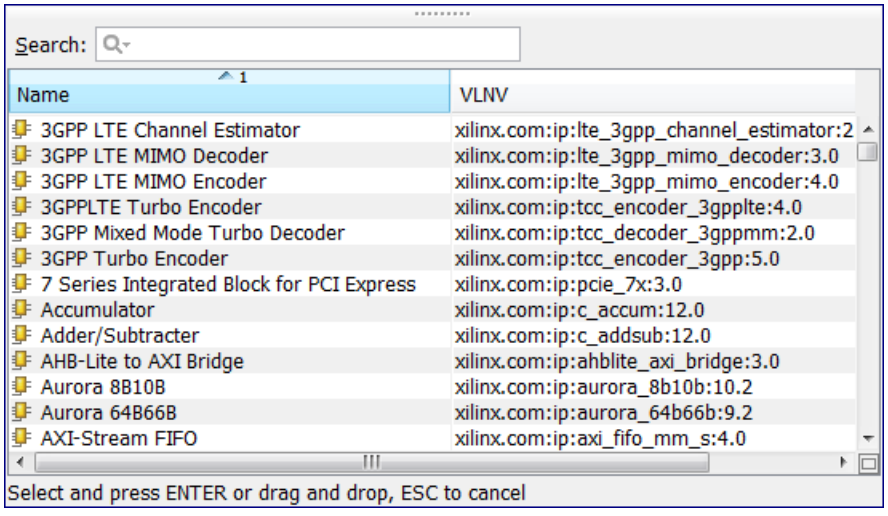


Figure 6: Opening the Vivado IP Catalog

By typing in the first few letters of the IP name in the Search filter, only IP modules matching the string are displayed.

TIP: Different fields associated with an IP such as Name, Version, Status, License, Vendor VLNV etc. can be enabled by right-clicking the Heading column of the IP Catalog and then selecting the appropriate field.

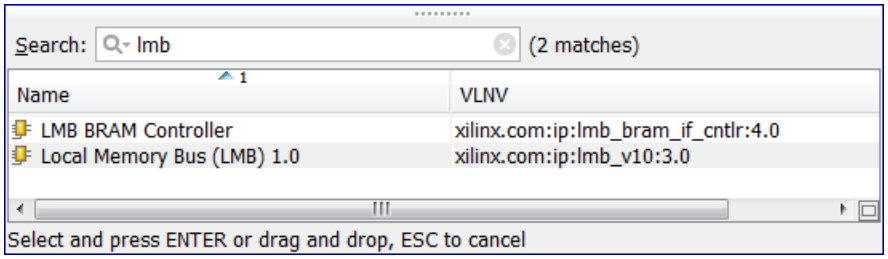


Figure 7: Using the Search Filter in the IP Catalog

2. To add a single IP, you can either click on the IP name and press the **Enter** key on your keyboard, or double click on the IP name.
3. To add multiple IP to the Block Design, you can highlight the additional desired IP (Ctrl-click) and press the **Enter** key.

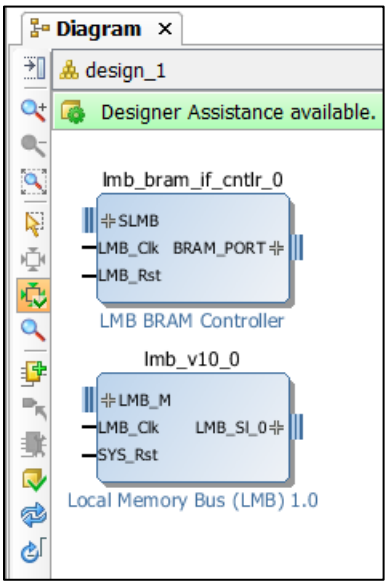


Figure 8: Adding Multiple IP at the Same Time

4. You can also add IP by clicking on the Add IP button on the left side of the canvas.

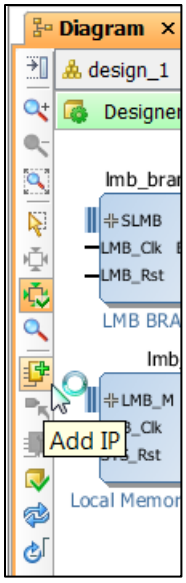


Figure 9: Adding IP by Clicking the Add IP Icon

Alternatively, IP can also be added by clicking on the link Add IP, in the green banner towards the top of the IP integrator canvas.

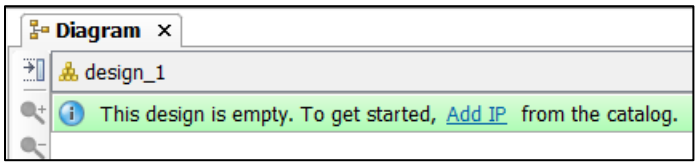


Figure 10: Adding IP by Clicking Add IP Link

You can also add an IP by selecting the IP in the IP Catalog and dragging and dropping it in the canvas.

For the actions described above, the IP is placed near the cursor location when the **Add IP** command is invoked.

The Vivado IP catalog entry in the Flow Navigator can also be displayed and used. If you are using dual monitors, you can open the IP catalog in its own monitor. If you are using a single monitor, you can float the IP catalog to move it away from the diagram. To add IP from the main IP catalog, you can drag and drop a selected piece of IP from the IP catalog onto the diagram.

CAUTION! If you double click on an IP in the IP Catalog, that IP will be added to the Vivado project, but not to the block design.

Making Connections

When you create a design in IP integrator, you add blocks to the diagram, configure the blocks as needed, make interface-level or simple-net connections, and add interface or simple ports.

Making connections in IP integrator is simple. As you move the cursor near an interface or pin connector on an IP block, the cursor changes into a pencil. You can then click on an interface or pin connector on an IP block, hold down the left-mouse button, and then draw the connection to the destination block.

As shown in the figure below, an interface-level connection is indicated by the more prominent connection box on a symbol.

Clicking on the + symbol on a block expands the interface and displays the associated signals and buses.

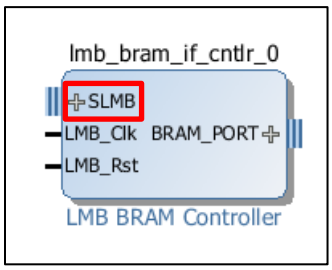


Figure 11: Connection Box on a Symbol

A signal or bus-level connection is shown as a narrow connection line on a symbol. Buses are treated identically to individual signals for connection purposes. As shown in the figure below, when you are making a connection, a green check box appears near the destination connection, indicating a potential connection.

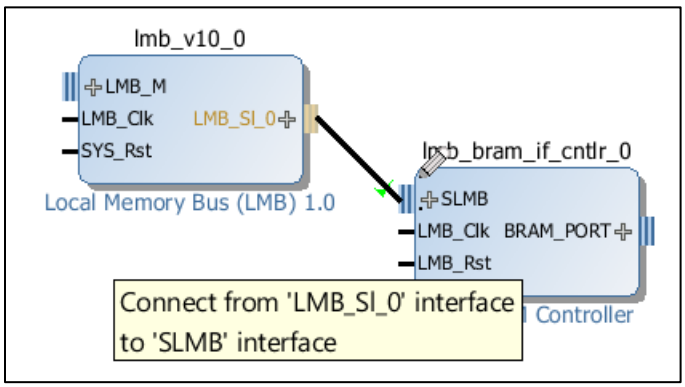


Figure 12: Signal or Bus Connection on a Symbol

As shown in the following figure, when signals are grouped as an interface, you should expand the interface first before making individual signal or bus connections.

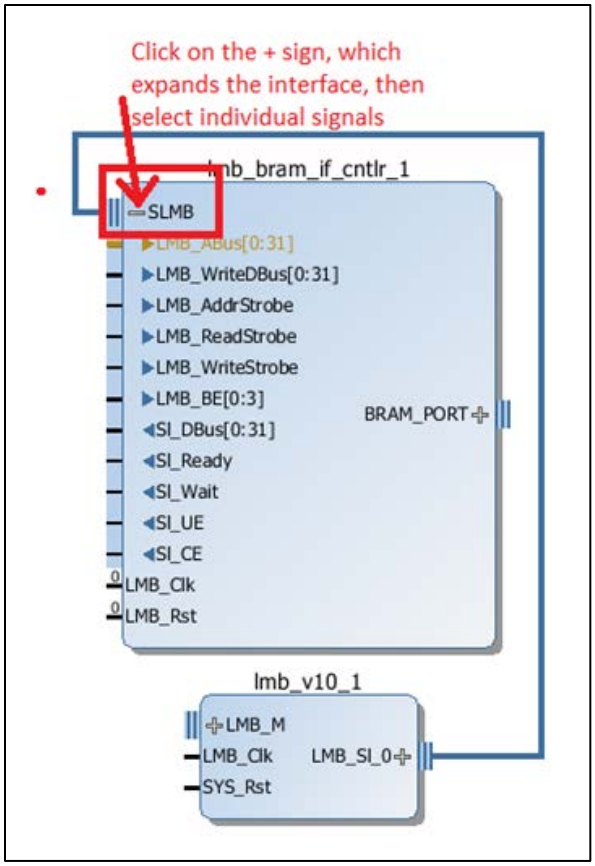


Figure 13: Expanding the Interface before Making a Connection

There are three ways to connect signals and interfaces to external I/O ports.

1. As shown in the following figure, to connect signals or interfaces to external ports on a diagram, you first select a pin, bus, or interface connection. You can then right-click and select **Make External**. It is possible to Ctrl-click multiple pins and invoke the **Make External** one time.

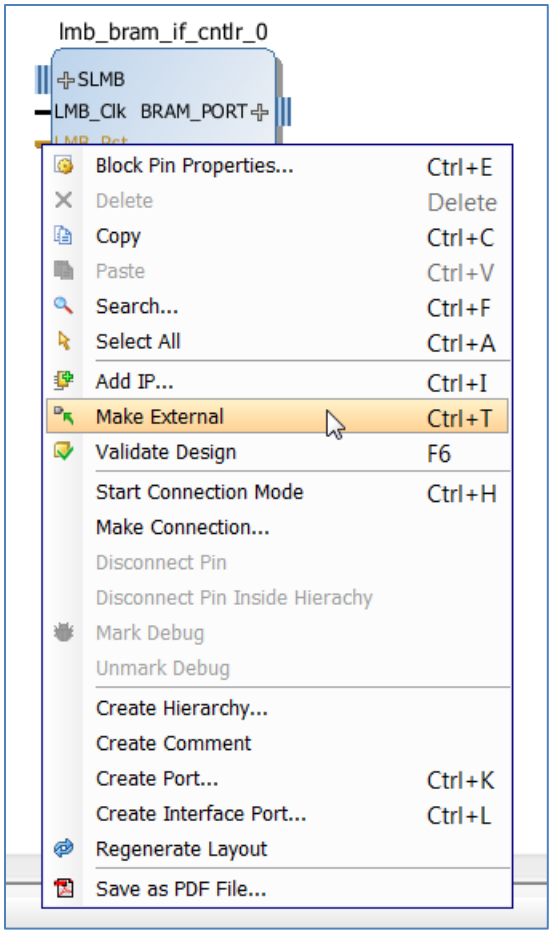


Figure 14: Making External Connections

This command is used to tie a pin on an IP to an I/O port on the block design. IP integrator simply connects the port on the IP to an external I/O.

2. For the second method of making a connection, you can right-click and select **Create Port**, as shown in the following figure. This feature is used for connecting non-interface signals, such as a clock, reset, and uart_txd. **Create Port** gives you more control in terms of specifying the input/output, the bit-width and the type (clk, reset, data) etc. In case of a clock, you can even specify the input frequency.

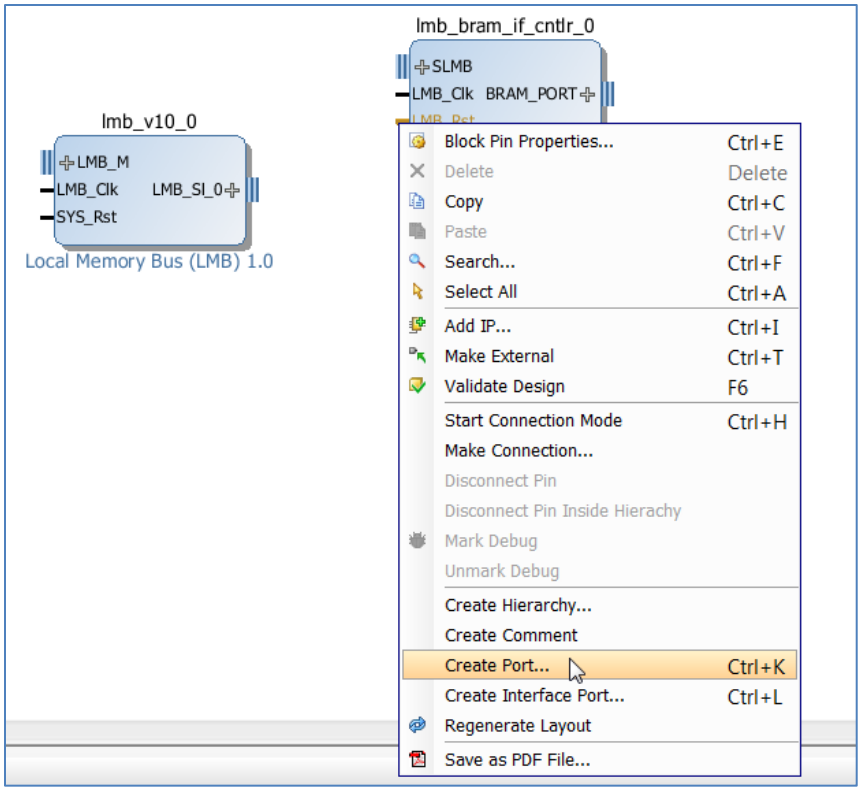


Figure 15: Creating a Port

In the Create Port dialog box, you specify the port name, the direction such as input, output or bi-directional, and type such as clock, reset, interrupt, data, clock enable or custom type. You can also create a bit-vector by checking the Create Vector field and then selecting the appropriate bit-width. When the type is selected as clock, you can also specify the frequency of the clock.

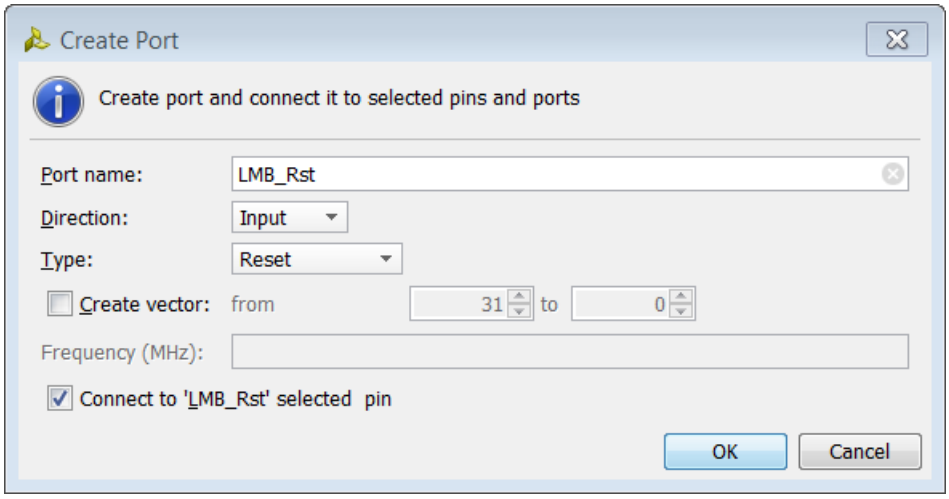


Figure 16: Create Port dialog box

3. For the third connection method, you can right-click and select **Create Interface Port**, as shown in the following figure.

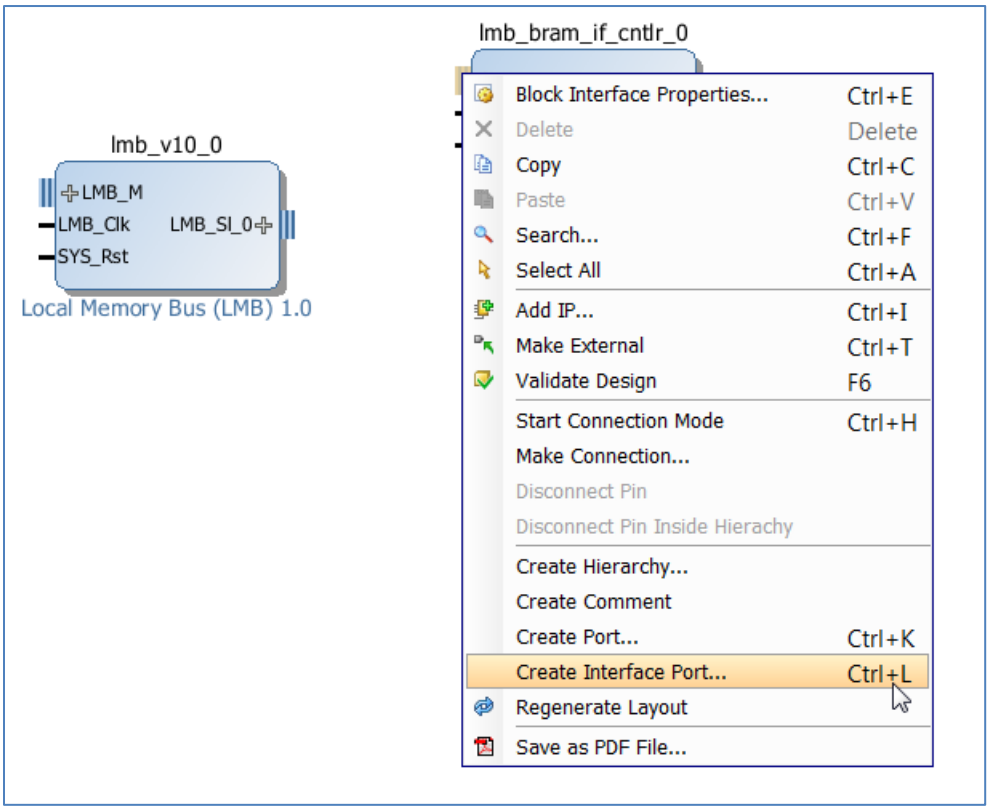


Figure 17: Creating an Interface Port

This command is used to create ports on the interfaces which are groupings of signals that share a common function. For example, the S_AXI is an interface port on several Xilinx IP. The command gives more control in terms of specifying the interface type and the mode (master/slave).

In the Create Interface Port dialog box, you specify the interface name, the vendor, library, name and version (VLNV) field, and the mode field such as MASTER or SLAVE.

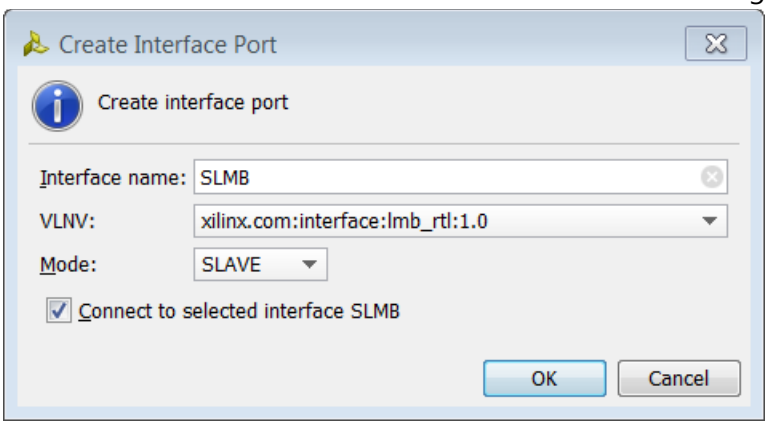


Figure 18: Create Interface Port dialog box

You can double-click on external ports to see their properties and modify them. In this case, the port is a clock input source, so you can specify different properties such as frequency, phase, clock domain, any bus interface, the associated clock enable, associated reset and associated asynchronous reset (frequency) associated with it.

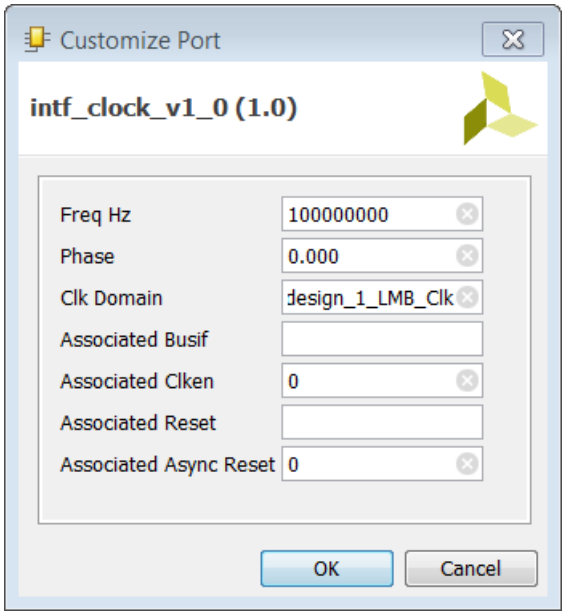


Figure 19: Customizing clock Port Properties

On an AXI interface, double clicking on the port shows the following configuration dialog box.

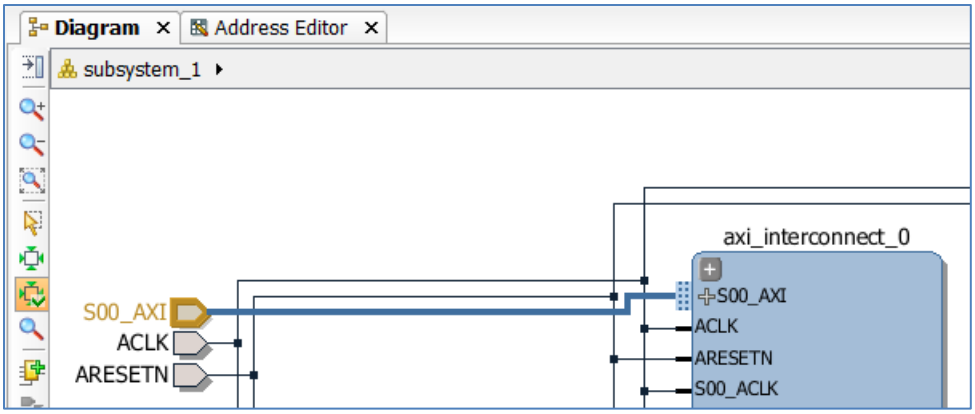


Figure 20: Customizing the Port Properties of an External AXI Interface

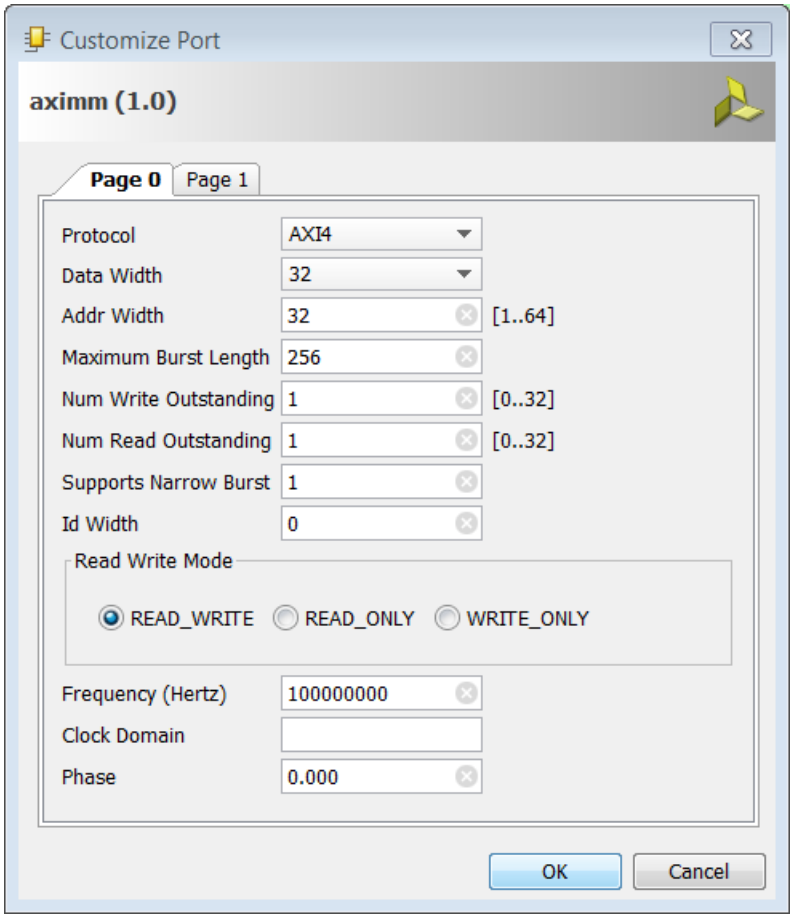


Figure 21: Customizing Port Properties of aximm

The Block Automation and Connection Automation Feature of IP Integrator

The Block Automation and Connection Automation features are provided in IP integrator to assist you in putting together a basic microprocessor system, making internal connections between different blocks and making connections to external interfaces. The Block Automation Feature is provided when a microprocessor such as the Zynq Processing System 7, MicroBlaze processor or some other hierarchical IP such as an Ethernet is instantiated in the IP integrator block design. You click on the **Run Block Automation** link, as shown in the following figure, to get assistance with putting together a simple MicroBlaze System.

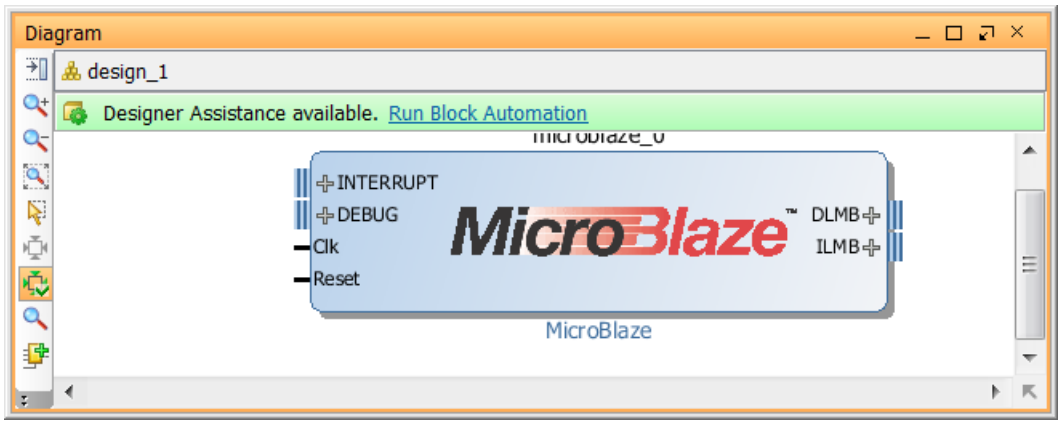


Figure 22: The Run Block Automation Feature

The **Run Block Automation** dialog box allows you to provide input about basic features that the microprocessor system needs.

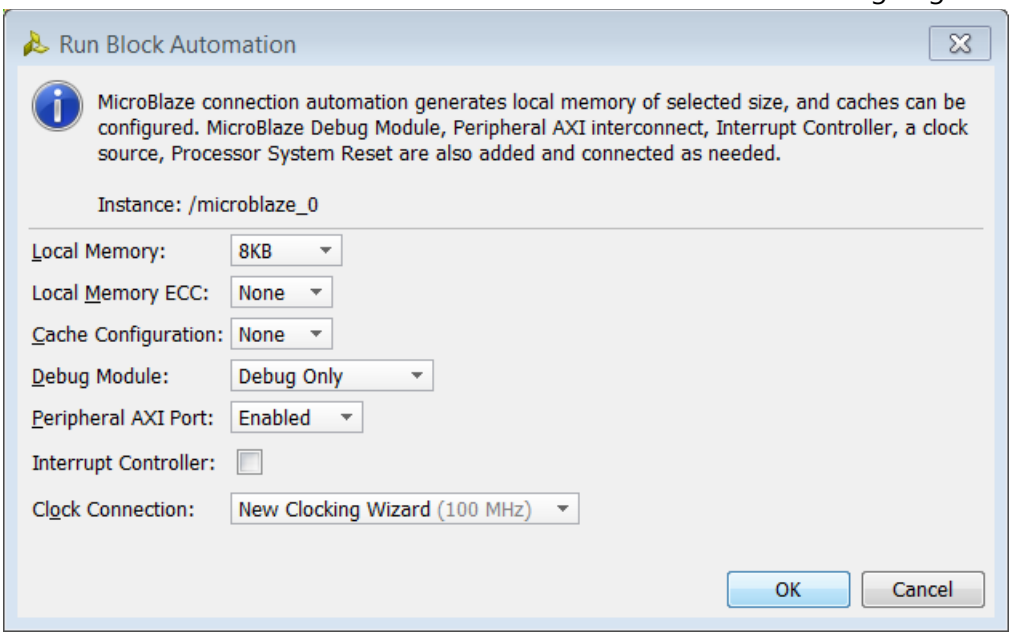


Figure 23: The Run Block Automation Dialog Box

Once you specify the necessary options, the Block Automation feature automatically creates a basic system as shown in the figure below.

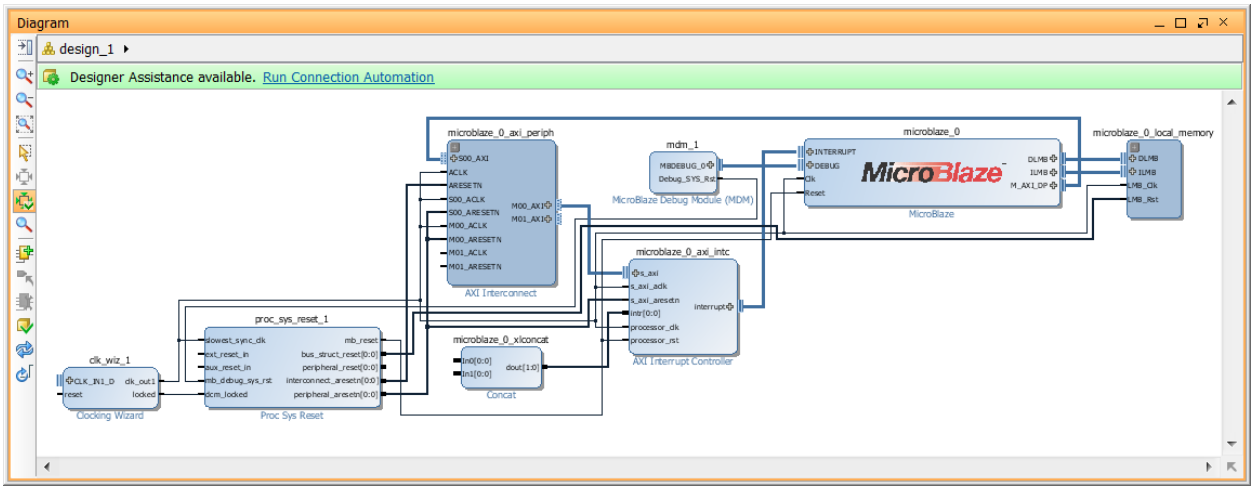


Figure 24: A Basic System Created by the Block Automation Feature

In this case, a basic MicroBlaze System is created that consists of a MicroBlaze Debug Module, a hierarchical block called the microblaze_1_local_memory that has the Local Memory Bus, the Local Memory Bus Controller and the Block Memory Generator, a Clocking Wizard, an AXI Interconnect and an AXI Interrupt Controller. Since the design is not connected to any external I/O at this point, IP integrator provides the **Connection Automation** feature as shown in the light green strip at the top of the canvas in the above figure.

When you click on **Run Connection Automation** you'll get assistance in hooking interfaces and/or ports to external I/O ports.

As shown in the following figure, ports/interfaces that can use the Connection Automation feature are listed.

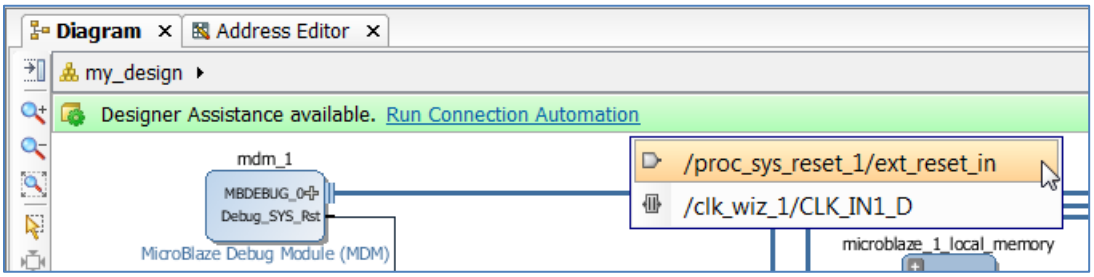


Figure 25: Listing the Ports and Interfaces the can use Connection Automation

For Xilinx's Target Reference Platforms or evaluation boards, the design knows the FPGA pins that are connected /used on the target boards. Based on that information, the IP integrator connection automation feature can assist you in tying the ports in the design to external ports. IP integrator then creates the appropriate physical constraints and other I/O constraints required for the I/O port in question. In the design above, the Proc Sys Reset IP needs to be connected to an external reset port and the Clocking Wizard needs to be connected to an external clock source. When the /proc_sys_reset_1/ext_reset_in option is selected above then the dialog box shown in the following figure opens up.

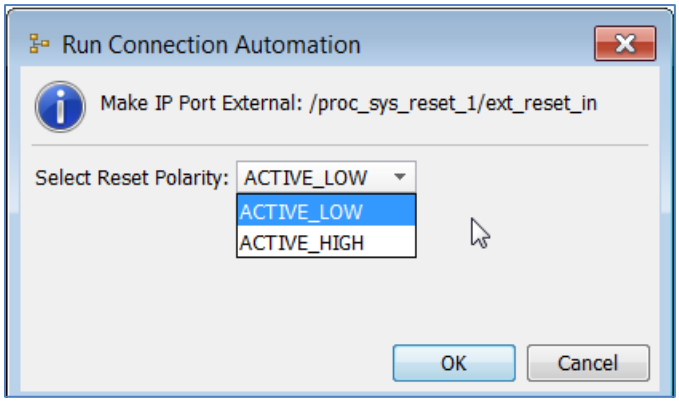


Figure 26: The Run Connection Automation Dialog Box

You can select the reset pin that already exists on the target board, KC705 in this case, or you can specify a custom reset pin to your design. Once specified, the reset pin is tied to the ext_reset_in pin of the Proc Sys Rst IP.

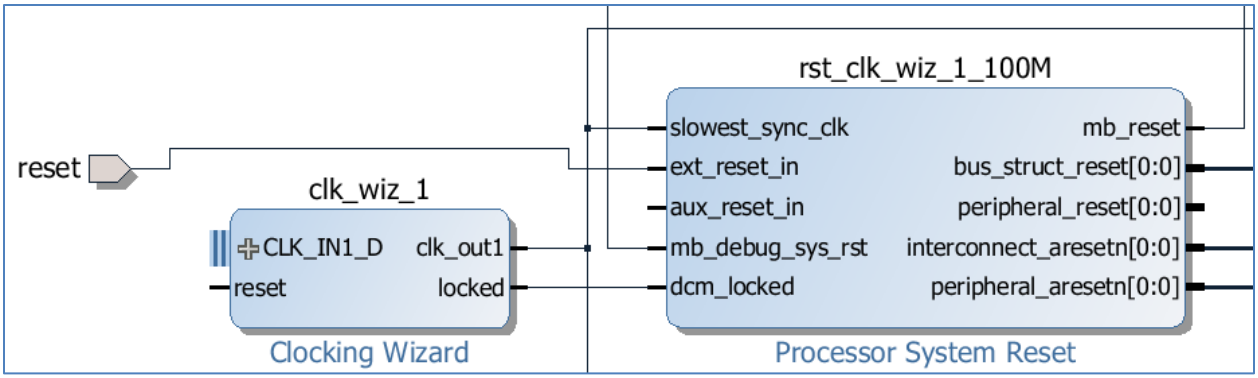


Figure 27: Connecting the reset Pin to the Board Reset Pin

Likewise, you can run connection automation and specify the input clock to the design.

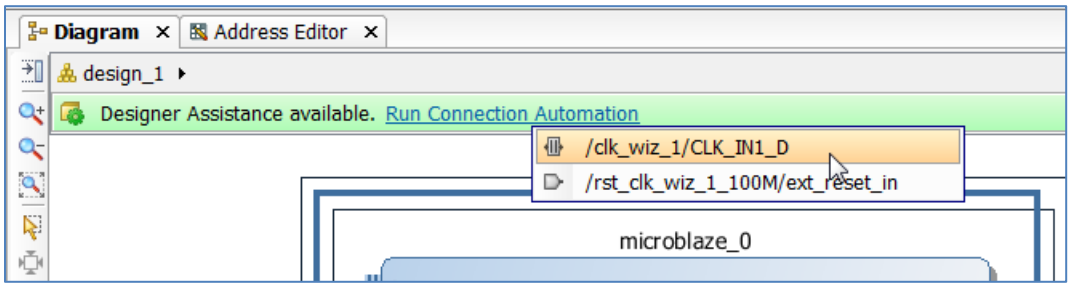


Figure 28: Run connection automation to connect input clock

In the Run Connection Automation dialog box you can specify whether to use the clock already present on the Xilinx target board or use a custom clock.

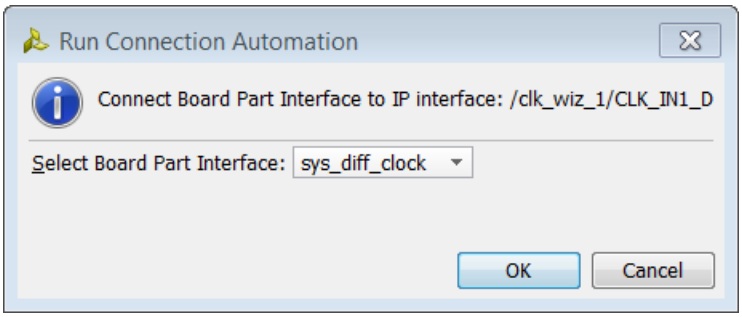


Figure 29: Connecting the CLK_IN1_D pin to the Board Clock

Now, assume you instantiate another IP into the IP integrator design that can use the Connection Automation feature. For example, assume that you instantiate the AXI GPIO IP in to the design. You can then click on **Run Connection Automation** and the feature shows that the S_AXI port of the AXI GPIO can be connected to the MicroBlaze processor via the AXI Interconnect.

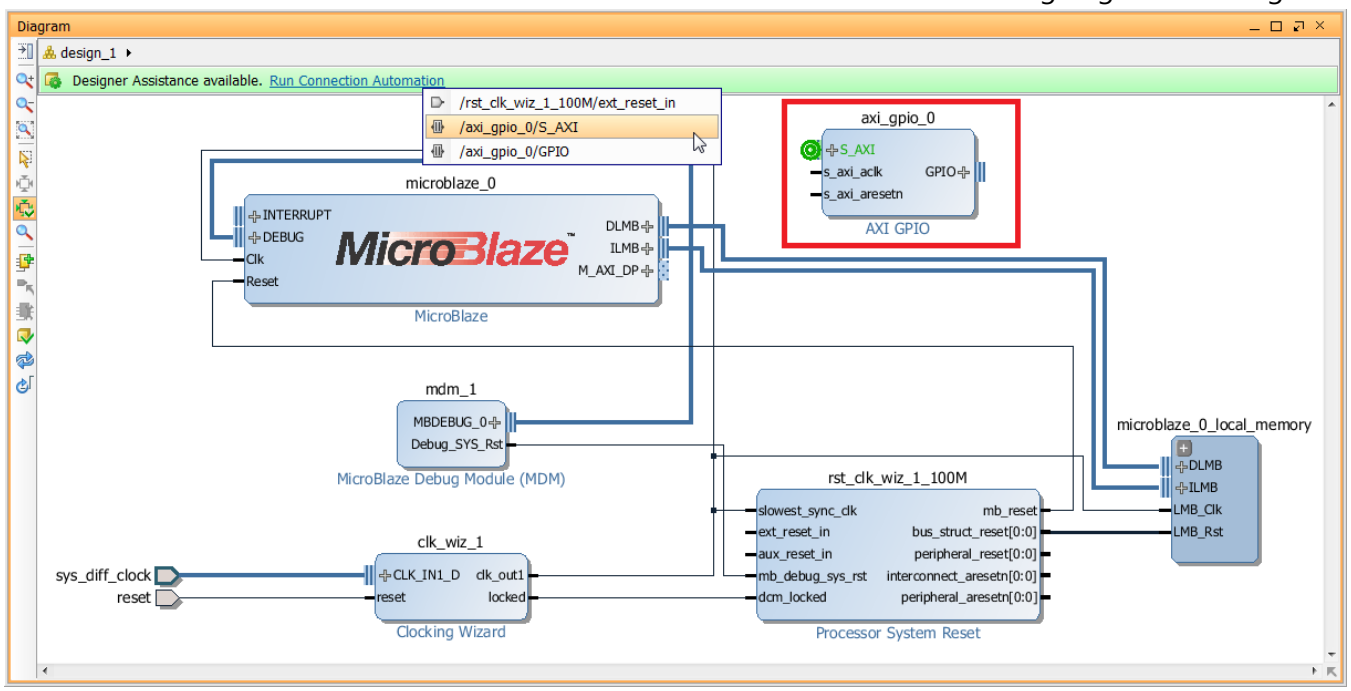


Figure 30: Using Connection Automation to Show Potential Future Connections

When you click on **Run Connection Automation**, the following pop-up box will inform you that the slave AXI port of the GPIO can be connected to the MicroBlaze master. If there are multiple masters in the design, then you will have a choice to select between different masters. You can also specify the clock connection for the slave interface such as S_AXI interface of the GPIO.

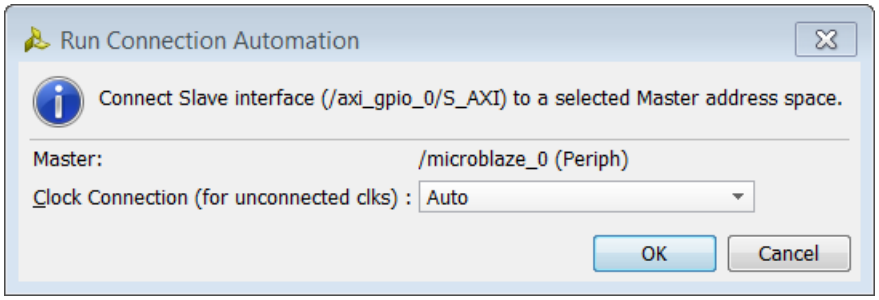


Figure 31: Connecting the Slave Interface s_axi to the MicroBlaze Master

When you click **OK** on the Run Connection Automation dialog box, the connections are made and highlighted as shown in figure below.

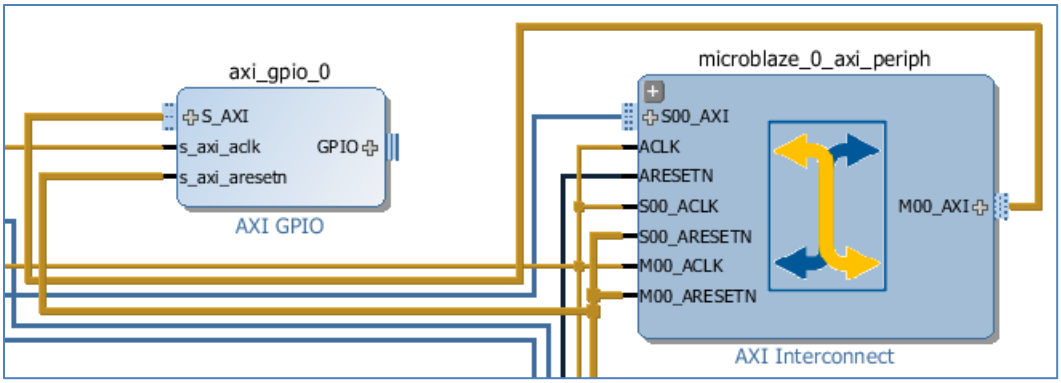


Figure 32: Master/Slave Connections are Made and Highlighted

In case a Xilinx Targeted Reference Platform is used, then further assistance can be available in terms of I/O port connections. Refer to the figure below.

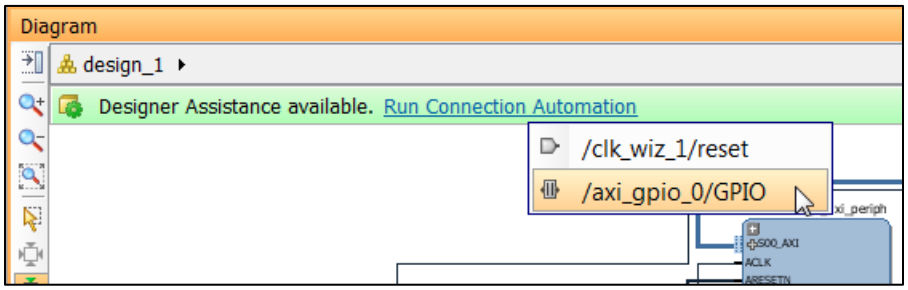


Figure 33: Further Assistance Available for Xilinx Targeted Reference Platforms

When you select the **gpio** port, the following options are presented based on the information available about the board.

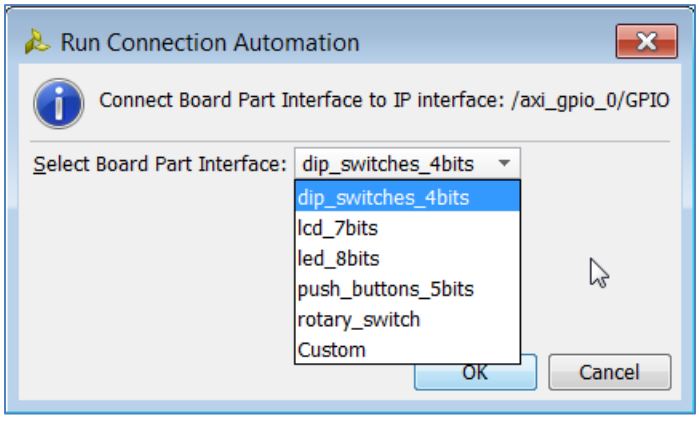


Figure 34: The Options Presented after a Port is Selected

In this case, six different choices are presented. The **gpio** port can be connected to either the **Dip Switches** that are 4-bits, or to the LCD that are 7-bits, **LEDs** that are 8-bits, 5-bits of Push Buttons, the Rotary Switch on the board or can be connected to a custom interface. Selecting any one of the choices above will connect the **gpio** port to the existing connections on the board.

Figure 35: Connecting Board I/O Elements to the Design

Using the Signals Tab to Make Connections

IP integrator also provides for an easy way to make connections to clocks and resets via the Signals window. Once the block design is opened, the Signals window shows up as shown below.

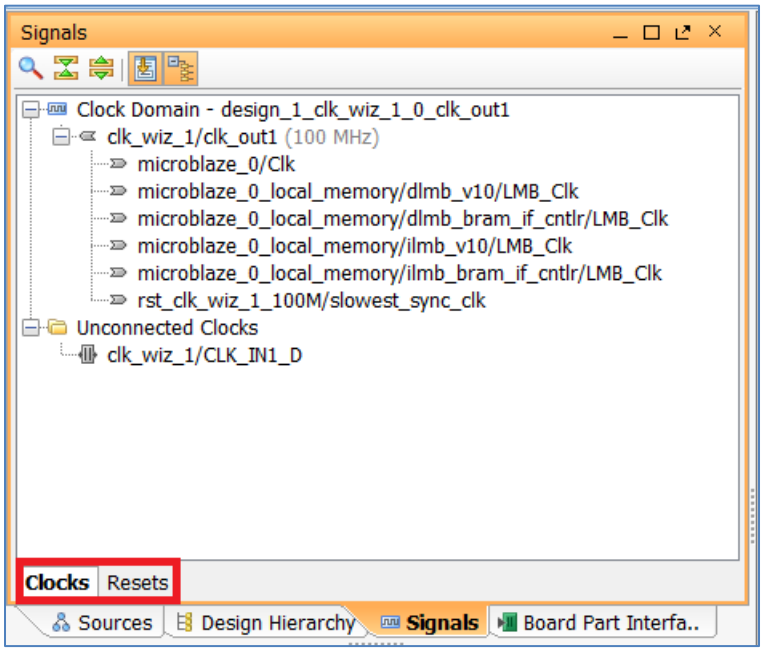


Figure 36: Signals window in IP Integrator

There are two tabs in the Signals window listing Clocks and Resets present in the design. Selecting the appropriate tab shows all the clocks and resets present in the design.

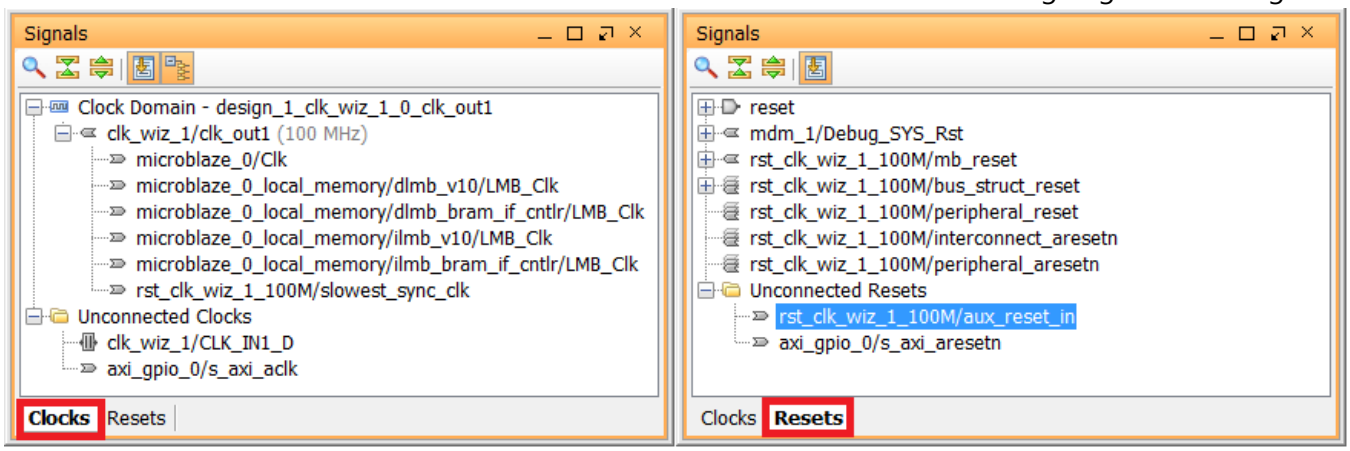


Figure 37: The Clocks and Reset tabs in the Signals window

Clocks are listed in the Clocks tab based on the clock domain name. In the figure above, it can be seen that the clock domain is called **design_1_clk_wiz_1_0_clk_out1** and the output clock clk_out1 with a frequency of 100 MHz is driving several clock inputs of different IP. When you select a clock from the **Unconnected Clocks** folder the appropriate clock port in the block design gets highlighted. Right-clicking on the selection presents you with several options. In this case, there is Designer Assistance available (**Run Connection Automation** option) which can be used to connect the CLK_IN1_D input interface of the Clocking Wizard to the clock pins on the board. You can also select the **Make Connection** option, and connect the input to an existing clock source in the design. Finally, you can tie the pin to an external port by selecting the **Make External** option. Other options for switching the context to the diagram and running design validation are also presented.

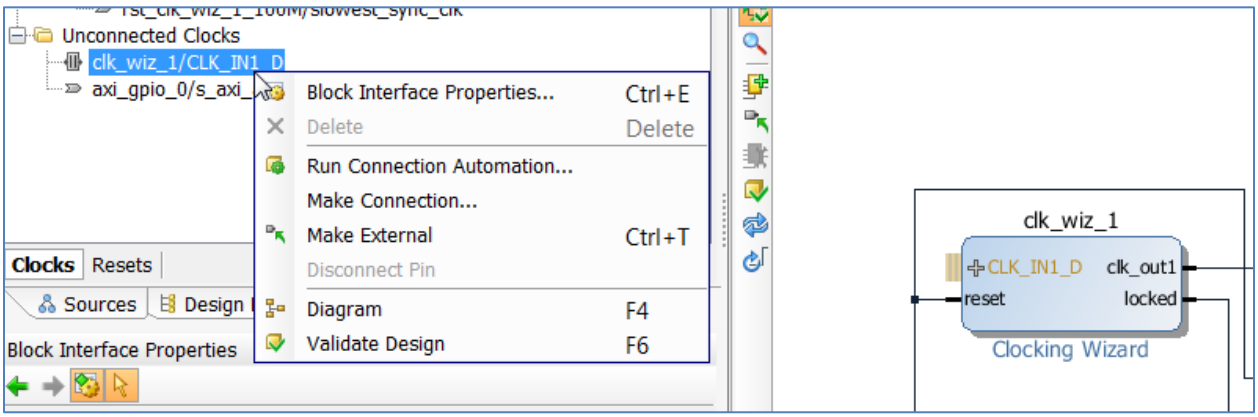


Figure 38: Making Connection using the Signals tab

When you select **Make Connection**, a dialog box pops up if a valid connection can be made.

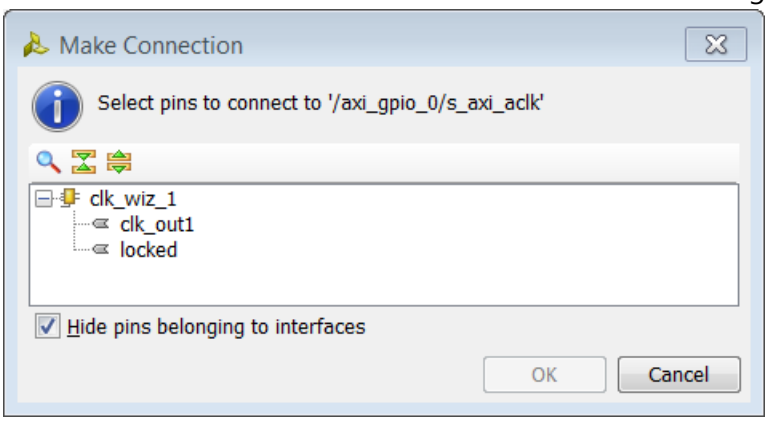


Figure 39: Make Connection dialog box

Selecting the appropriate clock source will make the connection between the clock source and the port/pin in question.

Connections can similarly be made from the Resets tab. Using the Clocks and Resets tab of the Signals window provides you with a visual way to manage and connect clocks in the design.

Using the Make Connections Option to Connect Ports/Pins

Connections to unconnected ports/pins can be made by selecting the port/pin in question and then selecting Make Connection from the pop-up menu.

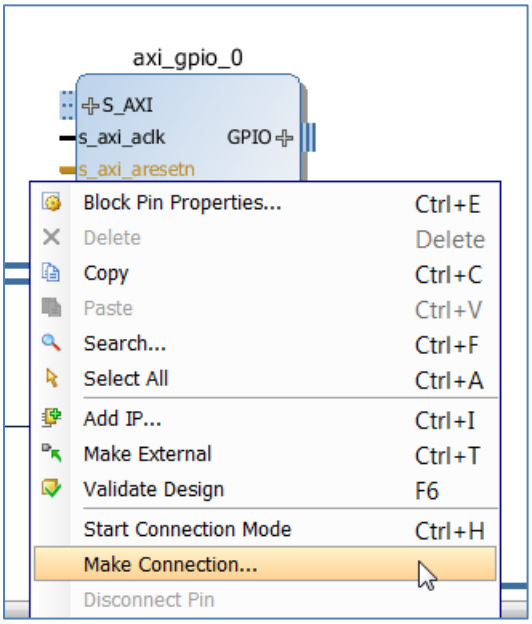


Figure 40: Making Connections to Unconnected Ports/Pins using the Make Connection Option

If a valid connection to the pin in question exists, then the **Make Connection** dialog box showing all the possible sources that the net can be connected to pops-up. From this dialog box you can select the appropriate source to drive the port/pin in question can be selected.

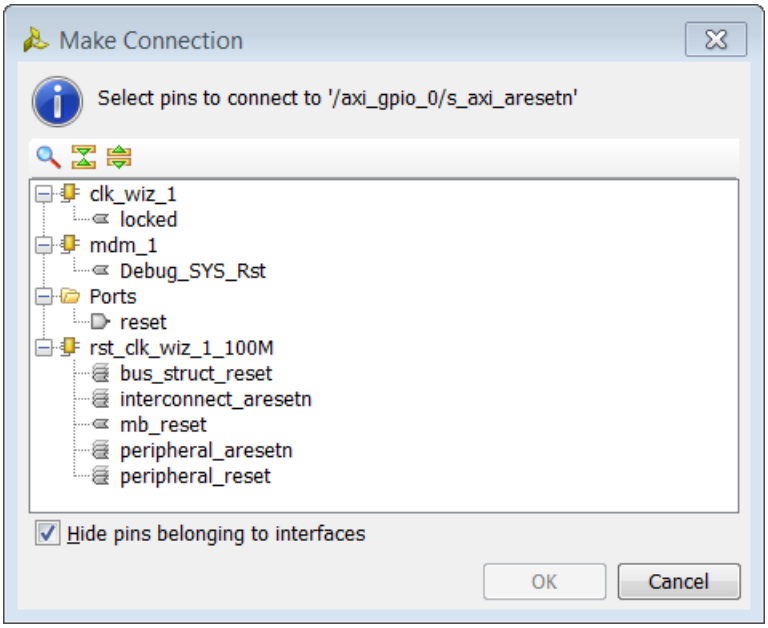


Figure 41: Making Connections using the Make Connection option

Using Start Connection Mode to make connections

You can also make multiple connections at once by clicking on a pin and when the pencil shows up dragging and releasing the mouse.

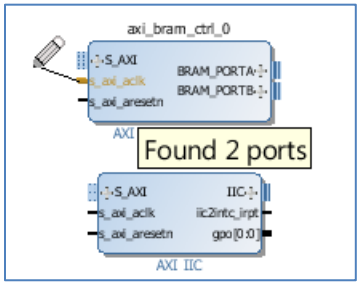


Figure 42: Using Start Connection Mode to make connections

After the connection to the s_axi_aclk pin of the AXI BRAM Controller in the figure above is made, the Start Connection Mode will offer to connect to connect to the s_axi_aclk pin of AXI IIC. In this way connections from the same source pin can be made to multiple different destinations at once.

Re-arranging the IP Blocks

IP blocks placed on the canvas can be re-arranged to get a better view of the block design. To arrange a completed diagram or a diagram in progress, you can click the **Regenerate Layout** button.

It is possible to move blocks manually by clicking on a block, holding the left-mouse button down, and moving the block or you can use the arrow keys. The diagram only allows specific column locations, indicated by the dark gray vertical bars that appear when moving a block. A grid appears on the diagram when moving blocks, which assists you in making better block and pin alignments.

It is also possible to manually place the blocks where desired and then click on **Optimize Routing**. This preserves the placement of the blocks (unlike the Regenerate Layout function) and only modifies the routing to the various blocks.

Cutting and Pasting

You can use Ctrl-C and Ctrl-V to copy and paste blocks in a diagram.

Creating Hierarchies

As shown in the following figure, you can create a hierarchical block in a diagram by using Ctrl-click to select the desired IP blocks, right-click and then select **Create Hierarchy**.

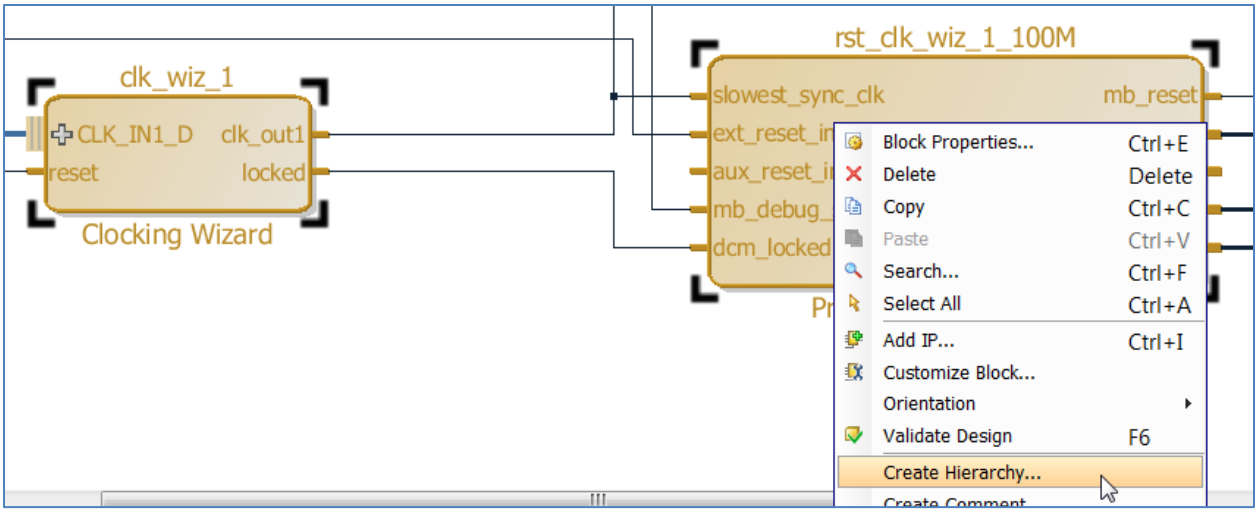


Figure 43: Creating a Hierarchal Block Design

IP integrator creates a new level of hierarchy containing the selected blocks. Creating multiple levels of hierarchy is supported. It is also possible to create an empty level of hierarchy, and later drag existing IP blocks into that hierarchical block. Hierarchies can be expanded when you click on the + sign in the upper-left corner of the block. You can traverse levels of hierarchy in a diagram using the Explorer type path information displayed in the upper-left corner of the IP integrator diagram.

Clicking on **Create Hierarchy** pops-up the Create Hierarchy dialog box, as shown below, where you can specify the name of the hierarchy.

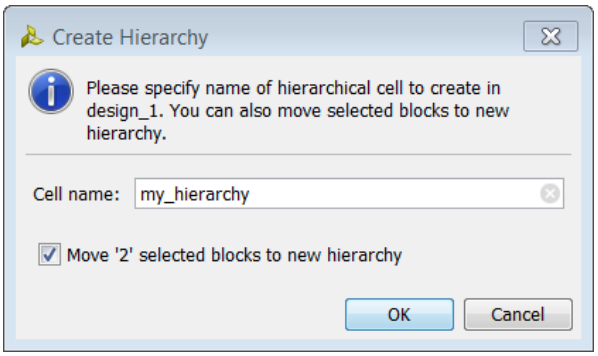


Figure 44: The Create Hierarchy Dialog Box

This action groups the two blocks under one block, as shown below. You can click on the + sign of the hierarchy to view the components underneath. Click on the – sign on the expanded hierarchy to collapse it back to the grouped form.

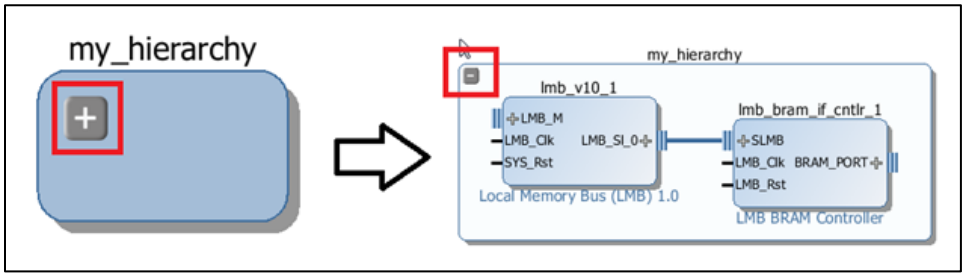


Figure 45: Grouping Two Blocks into One Block

Adding Pins and Interfaces to Hierarchies

As mentioned above, you can create an empty hierarchy and you can define the pin interface on that hierarchy before moving blocks of IP under the hierarchy. You right-click on the IP integrator canvas and select **Create Hierarchy**. In the Create Hierarchy dialog box, you specify the name of the hierarchy. Once the empty hierarchy has been created, the block design should look like the following figure.

Figure 46: Creating an empty hierarchy

You can add pins to this hierarchy by typing the following command on the Tcl Console.

```
create_bd_pin -dir I -type rst /hier_0/rst
```

In the above command, an input pin named reset of rst type was added to the hierarchy. You can add other pins using similar commands. Likewise, you can add a clock pin to the hierarchy using the following Tcl command:

```
create_bd_pin -dir I -type clk /hier_0/clock
```

You can also add interfaces to a hierarchy by using the following Tcl commands. First set the block design instance to the appropriate hierarchy where the interface is to be added, using the following command:

```
current_bd_instance /hier_0
```

Next, create the interface using command as specified below:

```
create_bd_intf_pin -mode Master -vlnv xilinx.com:interface:gpio_rtl:1.0 gpio
```

It is assumed that the right type of interface has been created prior to using the above command. After executing the commands shown above the hierarchy should look as shown in the following figure.

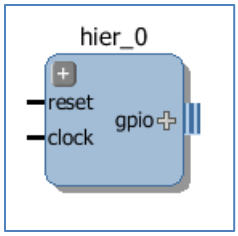


Figure 47: Create Pins in a Hierarchy

Once the appropriate pin interfaces have been created, different blocks can be dropped within this hierarchical block and pin connections from those IP to the external pin interface can be made.

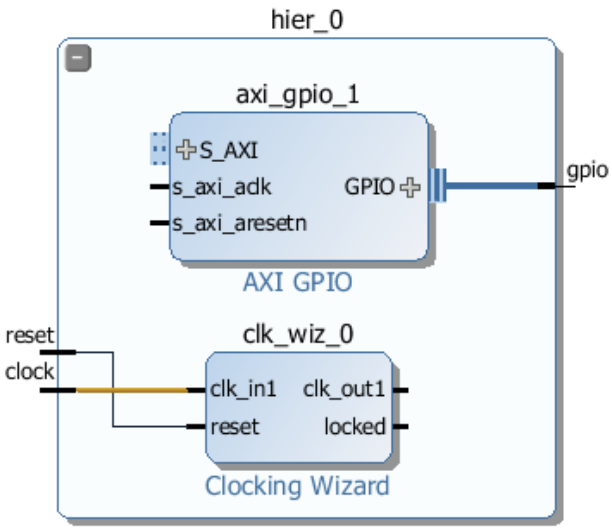


Figure 48: Making Pin-Level Connections of IP to the Hierarchical Pin Interface

Creating a Memory Map

To generate the address map for this design, you can click on the **Address Editor** tab above the diagram. Typically, the addresses are mapped automatically as you instantiate slaves in your block design. However, you can also click the **Auto Assign Address** button (the bottom button on the left side). If you generate the RTL from an IP integrator diagram without first generating addresses, a prompt will appear allowing the selection of automatically assigning addresses. You can also manually set Addresses by entering values in the **Offset Address** and **Range** columns.

***Note**: The Address Editor tab only appears if the diagram contains an IP block that functions as a bus master (such as the MicroBlaze processor in the following diagram) or an external bus master (outside of IP integrator) is present.*

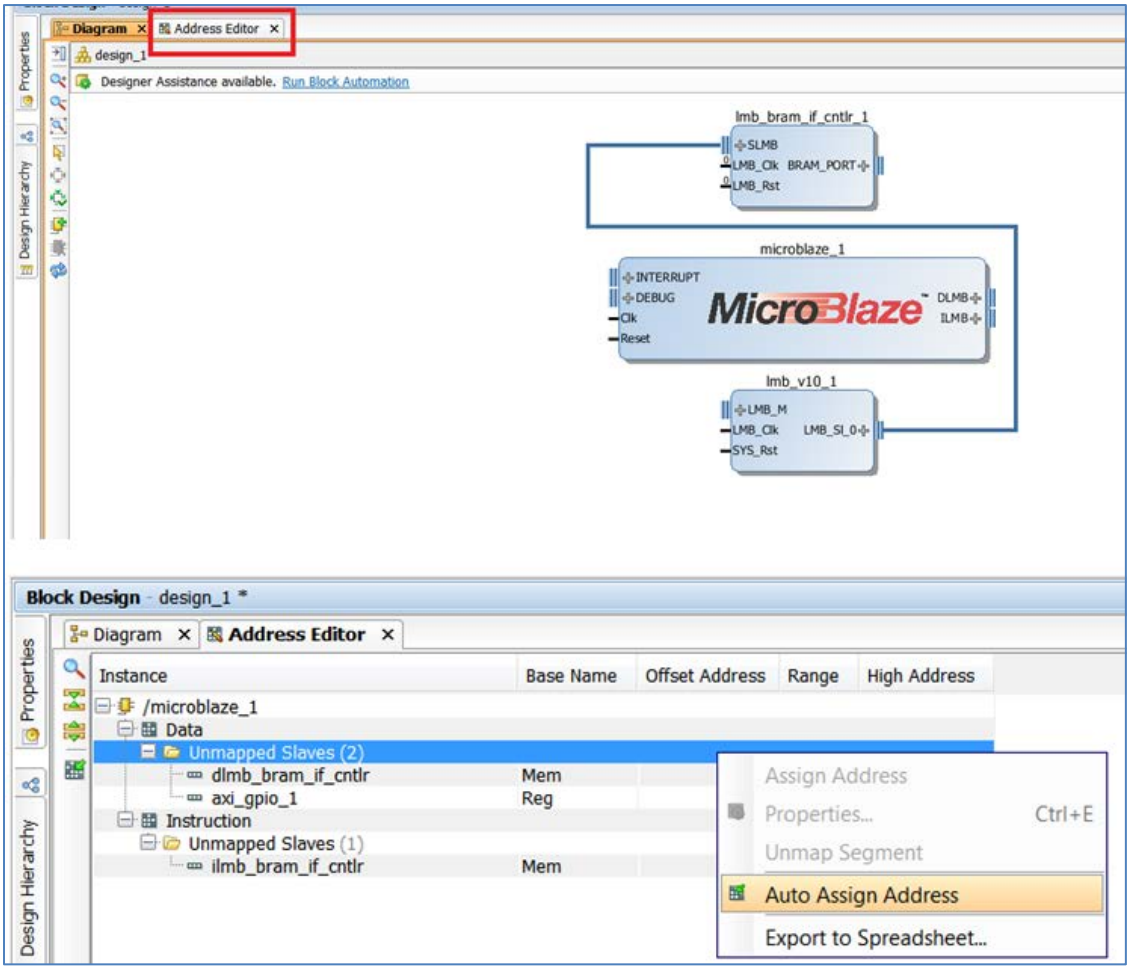


Figure 49: Using the Address Editor Tab

Interfacing with AXI IP Outside of the Block Design

There are situations when an AXI master is outside of the block design. These external masters are typically connected to the block design using an AXI Interconnect. Once the ports on the AXI interconnect are made "external", the address editor is available and memory mapping can be done in these cases.

As an example, consider the block design shown below.

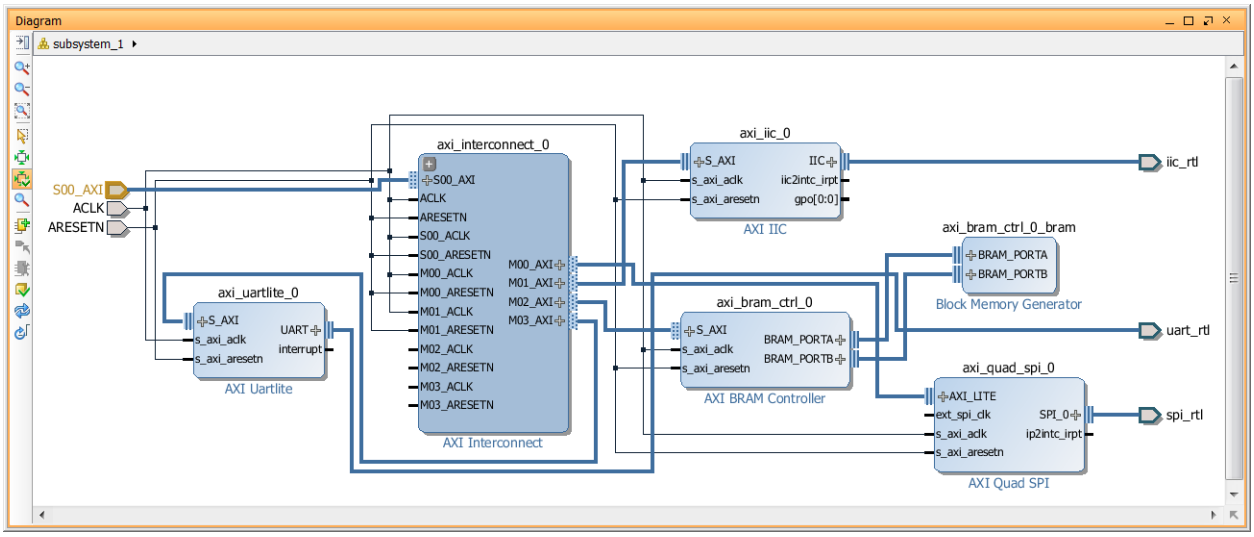


Figure 50: Example of a Design Consisting of an External AXI Master Interfacing the Block Design

When the AXI interface of the Interconnect is made external, the Address Editor tab becomes available and memory mapping all the slaves in the block design can be done in the normal manner.

Running Design Rule Checks

IP integrator runs basic design rule checks in real time as the design is being assembled. However, there is a potential for something to go wrong during design creation. As an example, the frequency on a clock pin may not be set right. As shown in the following figure, you can run a comprehensive design check on the design by clicking on **Validate Design**.

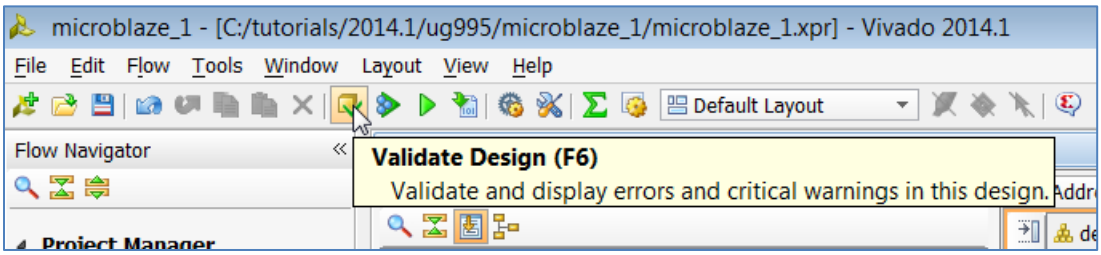


Figure 51: Validating the Design

Design validation can also be run by clicking on the Validate Design icon in the toolbar on the IP integrator canvas. If the design is free of Warnings and/or Errors, a pop-dialog box such as that shown in the figure below is displayed after running **Validate Design**.

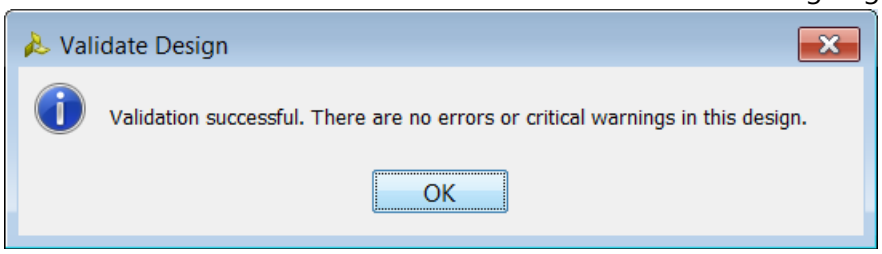


Figure 52: Successful Validation Message

Integrating the Block Design into a Top-Level Design

Once the block design is complete and the design is validated there are two more steps required to complete the design. First, the output products must be generated. This is when the source files and the appropriate constraints for all the IP will be generated and made available in the Vivado Sources pane. Depending upon the target language selected during project creation, appropriate files will be generated. If the source files for a particular IP cannot be generated in the target language specified, then a message stating so will be displayed in the Tcl Console. To generate output products, in the Vivado sources pane, you right click on the block design, as shown in the following figure, and select **Generate Output Products**.

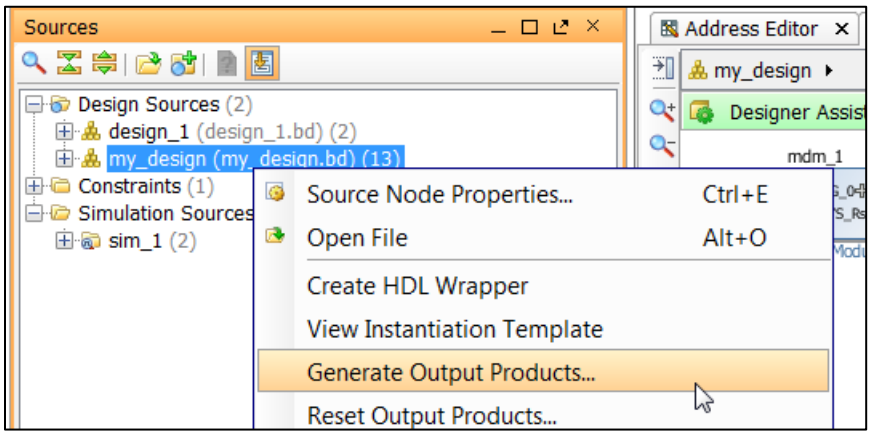


Figure 53: Generating Output Products

An IP integrator block design can be integrated into a higher-level design or it can be the highest level in the design hierarchy. To integrate the IP integrator design into a higher-level design, simply instantiate the design in the top-level HDL file.

A higher-level instantiation of the block design can also be done by selecting the block design in the Vivado IDE **Sources** pane and selecting **Create HDL Wrapper** (shown below). This will generate a top-level HDL file for the IP integrator sub-system.

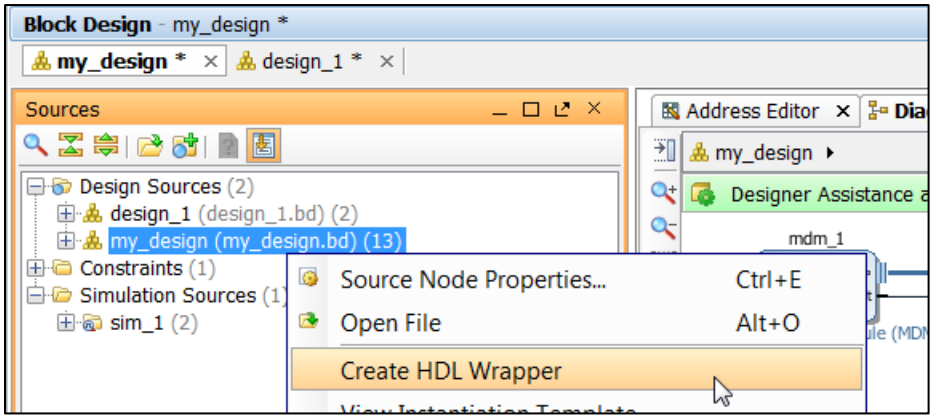


Figure 54: Creating an HDL Wrapper

Selecting Create HDL Wrapper offers two choices. The first choice is to make the wrapper file user editable. You may want to choose this option if you would like to manually edit the wrapper file. Often times a block design is a subset of an overall design hierarchy. This option can be used in this case. You can then instantiate other design components in the wrapper file. You need to make sure that this file is updated any time an I/O interface of the block design changes. The wrapper file created using this method is placed in the <project_name>.srcs/sources_1/imports/hdl directory.

The second choice offered is to allow the Vivado tools to create and manage the top-level wrapper file. If the block design is the only design component in the project or if edits to the wrapper file are not desired, then this option should be chosen. In this case the wrapper file is updated every time output products are generated. The wrapper file created using this method is located in the directory <project_name>.srcs/sources_1/bd/<bd_name>/hdl

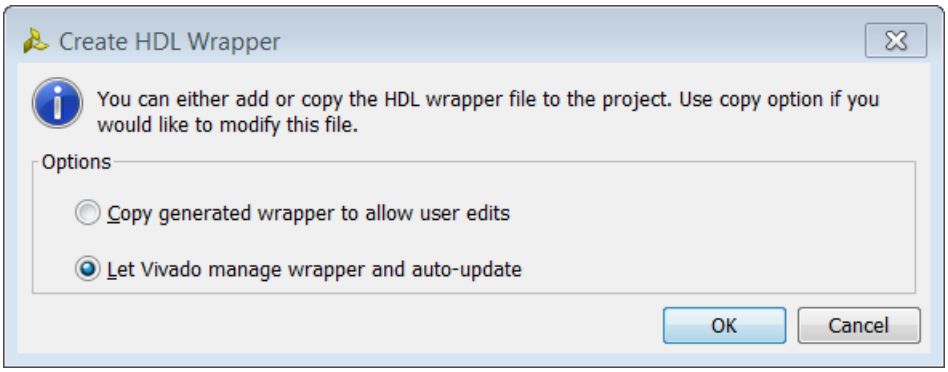


Figure 55: Create HDL Wrapper Dialog Box

At this point, you are ready to take the design through elaboration, synthesis, and implementation.

Creating a Block-Design Outside of the Project

A block design can be created within a project or outside of the project directory structure. A common use case for creating the block design outside of a project is to use the block design in a non-project mode or to use it in a team-based environment.

To create a block design outside the project, click on **Create Block Design** in the IP integrator drop-down list in the Flow Navigator. The Create Block Design dialog box opens. In the Create Block Design dialog box you can specify the name of the block design and also the Directory location. The default value for the Directory field is <Local to Project>. However, you can override this default value by clicking on Directory field and selecting **Choose Location**.

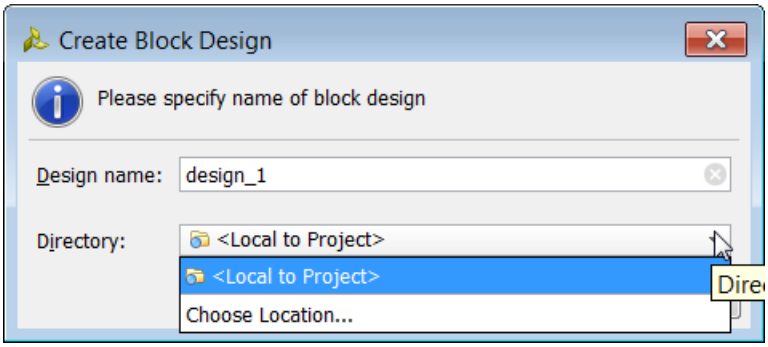


Figure 56: Create Block Design Dialog Box

Click **OK** after choosing the desired location to create the block design. You can continue to work on the block design just as you would in a normal project-based flow. The entire block design directory is created at the chosen location with its own sub-directory structure. You need to keep the entire directory structure of the bd in order to be able to re-open this block design from a different project or by a different user.

Exporting the Hardware Definition to SDK

If you want to start software development before a bitstream is created, you can export the hardware definition to SDK after generating the design. This action exports the necessary XML files needed for SDK to understand the IP used in the design and also the memory mapping from the processor's perspective. After a bitstream is generated and the design is exported, then the device can be downloaded and the software can run on the processor.

When the output products for the block design are generated, an archive that has all the pertinent information for exporting the hardware to SDK is created. This archive is called <top_level_design_name>.hwdef and can be found in the synthesis directory such as <project_name>.runs/synth_1. This archive contains the following files for a Zynq processor-based design.

- ps7_init.c
- ps7_init.h
- ps_init.html
- ps7_init.tcl
- hwdef.xml
- <bd_name>.hwh

For a MicroBlaze-based design this archive contains the following files:

- hwdef.xml
- <bd_name>.hwh

Once the design is implemented and the bitstream is generated, an archive called <top_level_design_name>.sysdef is created. For a Zynq processor-based design, the sysdef archive contains the following files:

- ps7_init.c
- ps7_init.h
- ps_init.html
- ps7_init.tcl
- sysdef.xml
- <bd_name>.hwh
- <top_level_design_name>.bit
- <top_level_design_name>.bmm

For a MicroBlaze-based design, this archive contains the following files:

- <bd_name>.hwh
- <top_level_design_name>.bit
- <top_level_design_name>.bmm
- sysdef.xml

Exporting the hardware simply copies the sysdef file into the appropriate location specified by the user.

To export the hardware definition to SDK, select **File > Export> Export Hardware** from the menu.

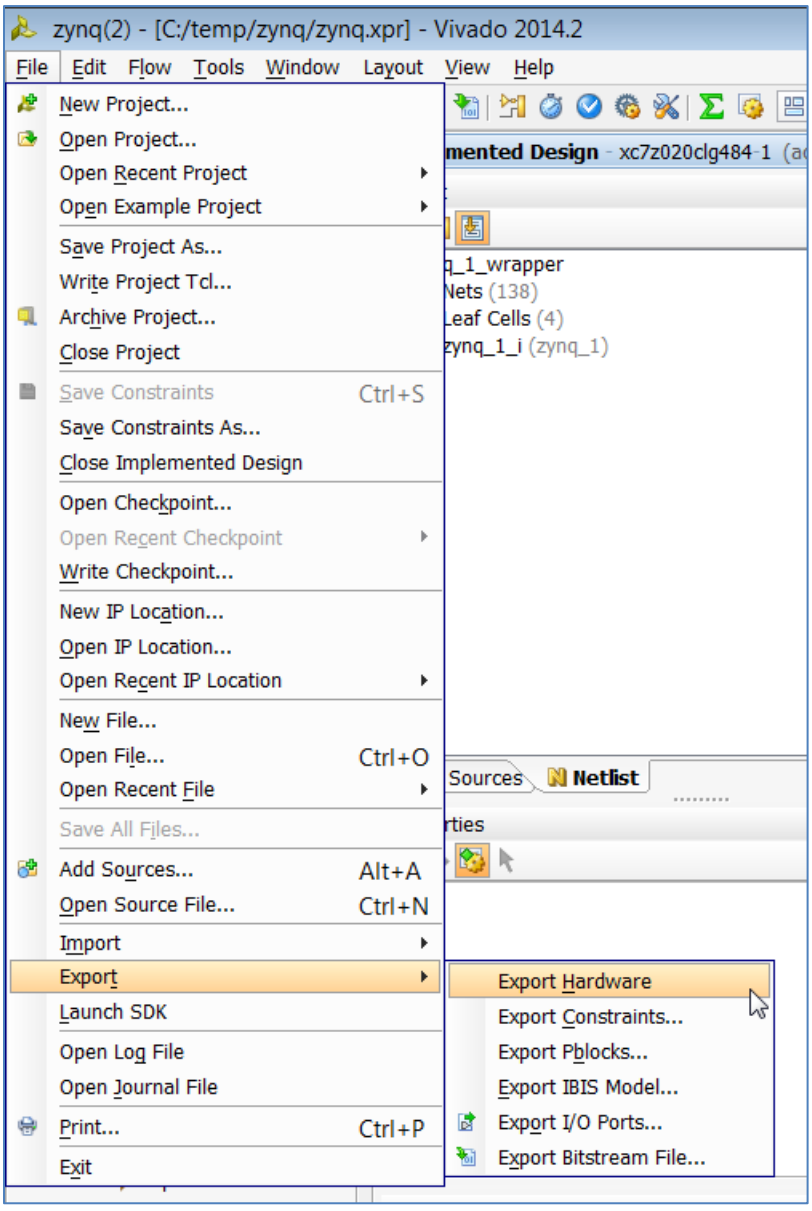


Figure 57: Exporting the Hardware Definition for a Project

The Export Hardware for SDK dialog box opens.

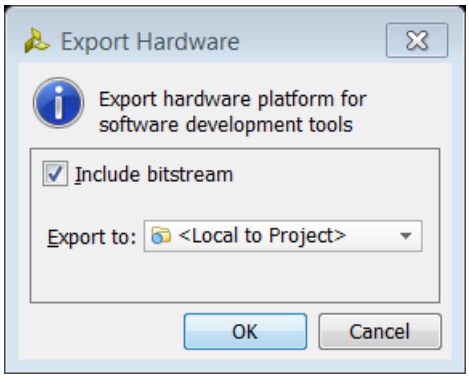


Figure 58: Exporting the Hardware Definition for a Project

There are a couple of options that are presented as can be seen from the above figure. Checking the Include bitstream checkbox will include the bitstream as a part of the exported data to SDK. The Export to field can be set as deemed appropriate by the user. In a typical project based flow, this should be left to its default value of **<Local to Project>**.

In a project-based flow, the hardware is exported at the following location:

```
project_name/project_name.sdk
```

To launch SDK after the hardware has been exported, click **File > Launch SDK**. The Launch SDK dialog box opens.

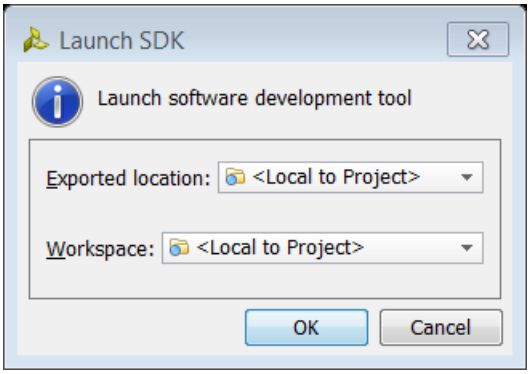


Figure 59: Launch SDK Dialog Box

There are a couple of fields in the Launch SDK dialog box that you can specify. In a typical project based flow, the default options should be left to **<Local to Project>,** for both the Exported location and the Workspace fields. If you choose a different location while exporting the hardware, then the Exported location field should be set to that particular location. The Workspace field can be set as per your need as well.

Once SDK launches, a custom application project can be created in it using the hardware definitions exported. SDK creates the necessary drivers and board support package for the target hardware.

Packaging the Block Design

Often times it happens that you create an IP integrator design, implement it, and test it on target hardware. If you are satisfied with the functionality, you may want to "package" it and convert it into an IP that can be reused in another design. When you package a design, it gets converted into an IP and is available for you in the IP catalog. You can instantiate that IP as part of a different design.

To package a block design, you right-click on the block design in the Sources window in the Vivado IDE and select **Package Block Design**.

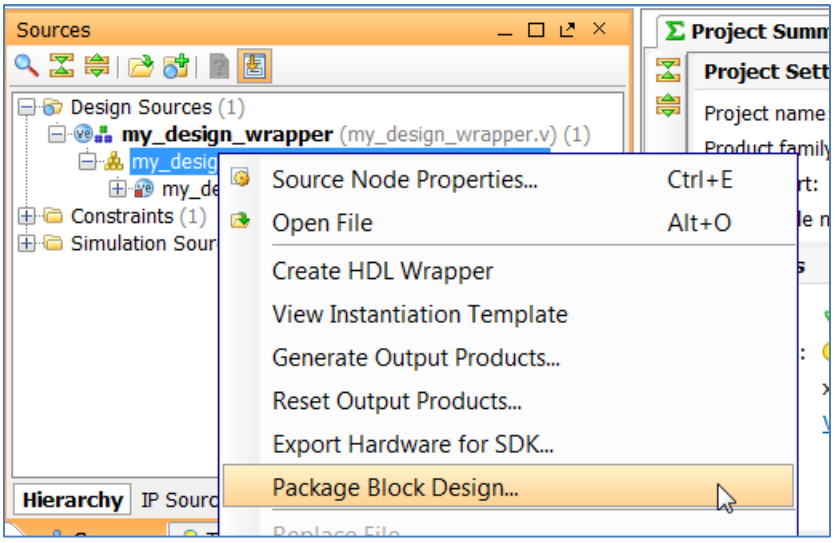


Figure 60**: Selecting Package Block Design**

The Package IP window opens up.

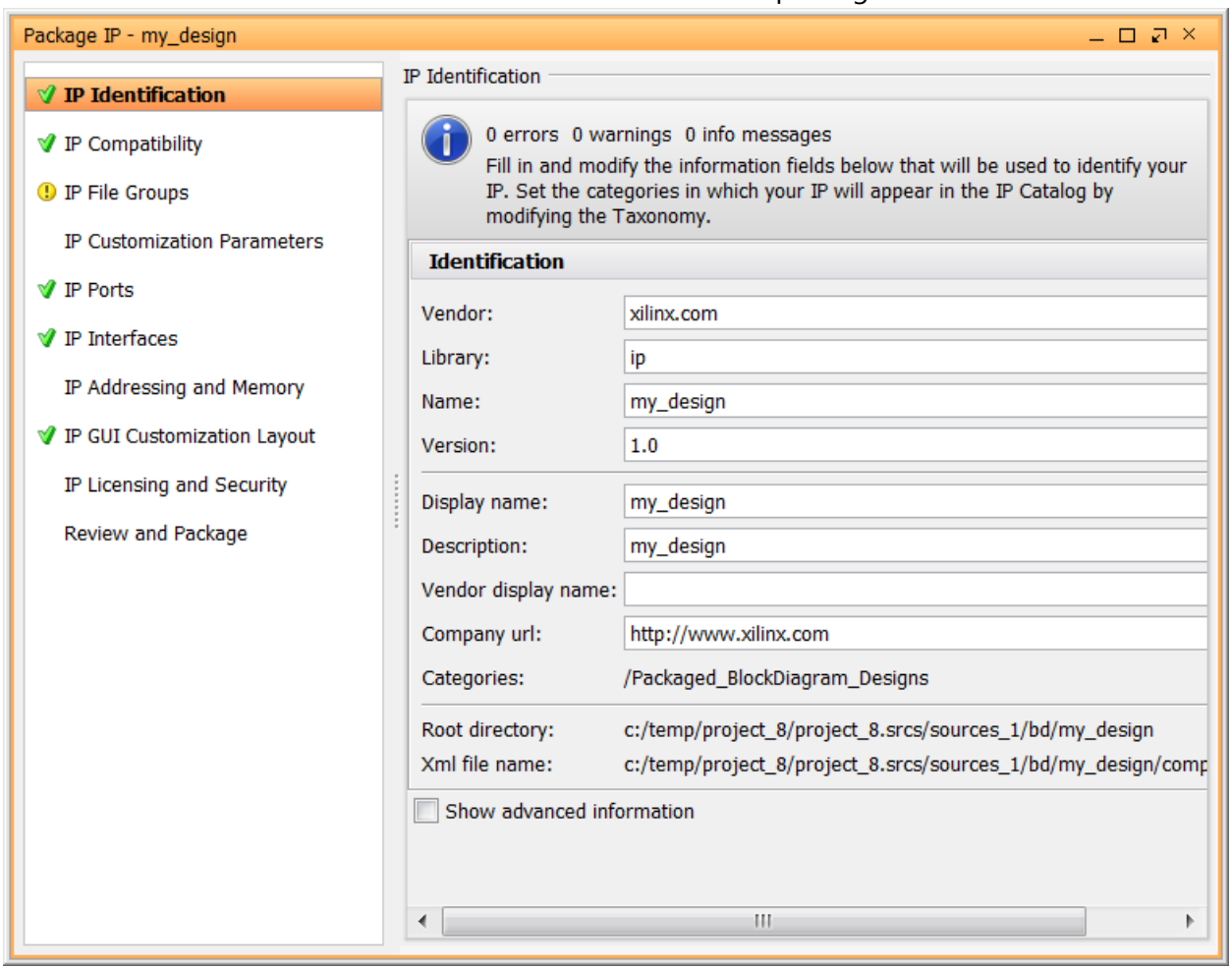


Figure 61: The Package IP Window

You select the **Review and Package** menu item in the dialog box and then click on **Package IP**. Once packaged, the IP is available in the IP integrator catalog as shown below.

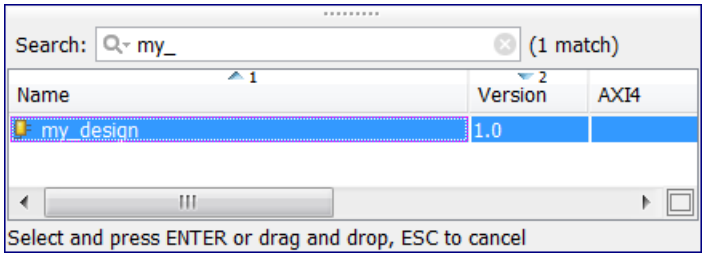


Figure 62: Packaged Design in the IP Integrator Catalog

The newly packaged design is also available in the Vivado IP catalog under the category **Packaged BlockDiagram Designs**. This category can be changed and renamed while packaging the block design.

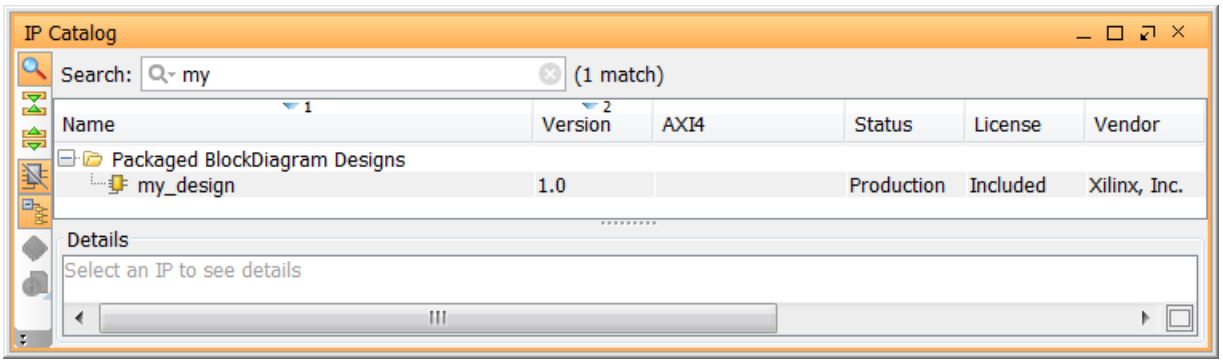


Figure 63: The Packaged Block Diagram Designs Category

Limitations to Packaging the Block Design

There are two limitations to packaging a block design currently as listed below.

- If a packaged block design contains a MicroBlaze or Zynq processor and then the packaged design is instantiated in a different project, the exporting Hardware to SDK poses some issues. Essentially, SDK does not recognize the embedded objects in the packaged block design.
- When a block design is packaged, a package (xml) file is generated that contains references to lower-level IP XCI files. When this packaged diagram is ported to a future release of the software, this could pose problems with upgrades as the IP can get locked if there are newer versions of the IP in the current release.

Adding and Associating an ELF File to an Embedded Design

In a microprocessor-based design such as a MicroBlaze design, an ELF file generated in the SDK (or in other software development tool) can be imported and associated with a block design in the Vivado IDE. A bitstream can then be generated that includes the ELF contents from the Vivado IDE and run on target hardware. There are two ways in which you can add the ELF file to an embedded object.

Add an ELF file as a Design/Simulation Source and then associate ELF to an embedded object

1. The first way to accomplish this task is by selecting and right-clicking **Design Sources** in the Sources window and then selecting **Add Sources** as shown below. Design sources can also be added by clicking on **Add Sources** in the Flow Navigator under the Project Manager drop-down list.

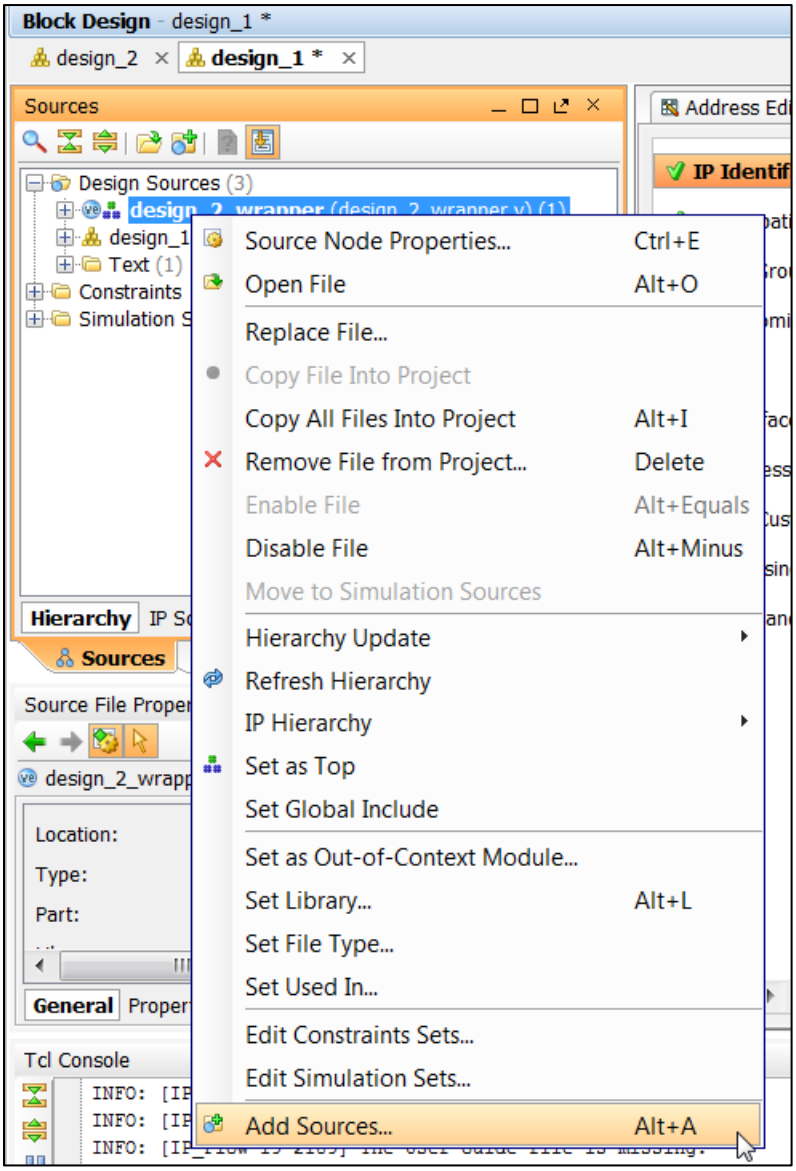


Figure 64: Selecting Add Sources

2. The Add Sources dialog box opens. **Add or Create Design Sources** is selected by default. Selecting this option will add the ELF file as a design as well as a simulation source. If you are adding an ELF file for simulation purposes only, select **Add or Create Simulation Sources**. Click **Next**.
3. In the Add or Create Design Sources dialog box, click on **Add Files**.

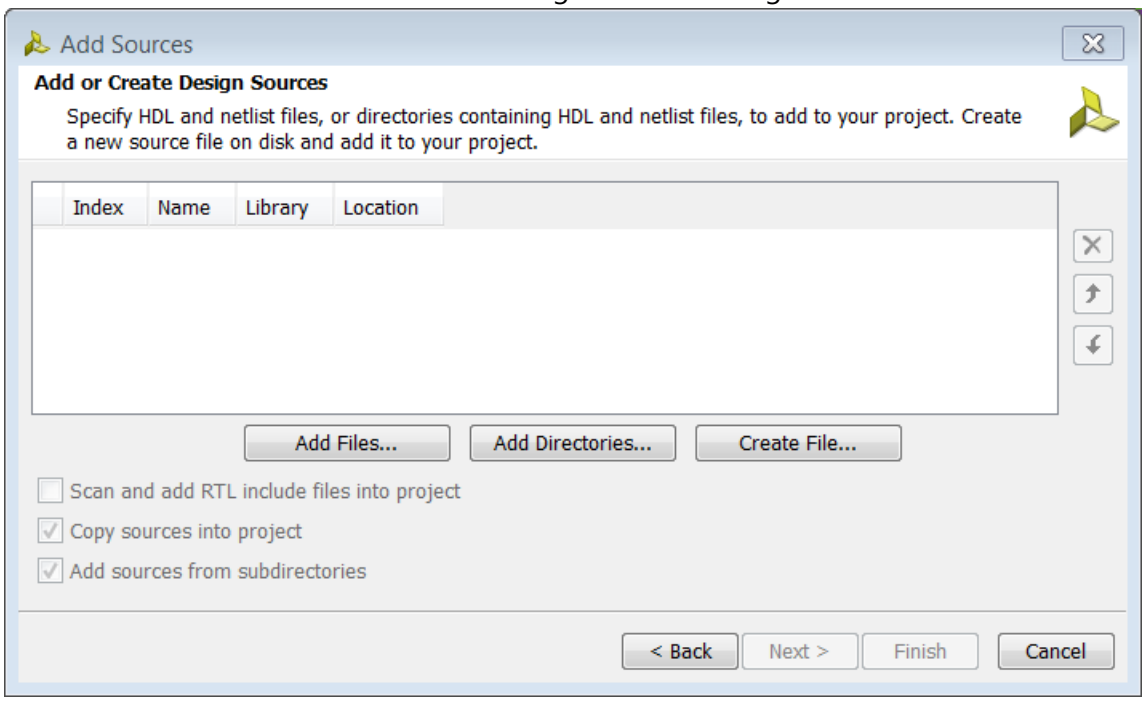


Figure 65: Add Sources Dialog Box

4. The Add Source Files dialog box opens. You then navigate to the ELF file, select it and click **OK**.

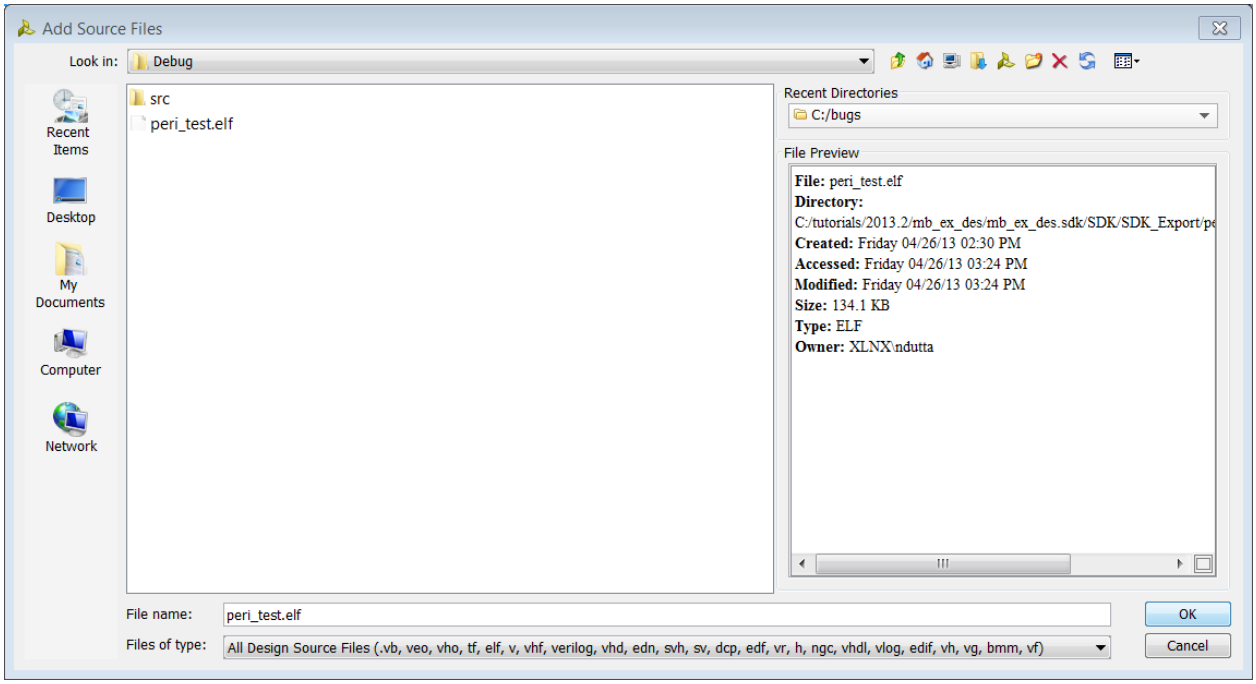


Figure 66: Add Sources Files Dialog Box

5. In the Add or Create Sources dialog box, you can see the ELF file added to the project. Depending on your preference, you can either copy the ELF file into the project by checking **Copy sources into project** or leave that option unchecked if you want to work with the original ELF file. You then click **Finish**.
6. Under the Sources window, you will see the ELF file added under the ELF folder.

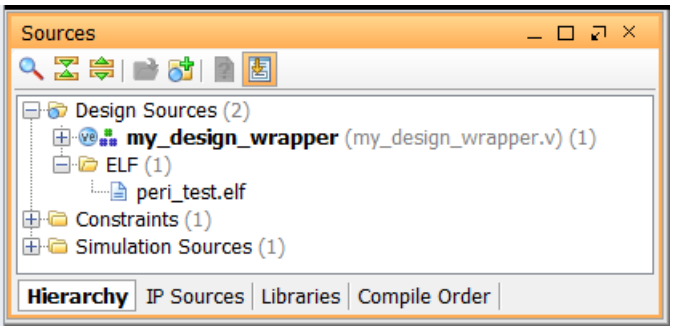


Figure 67: Sources Window with ELF File Displayed

7. Next, you associate that ELF file with the microprocessor design in question. To do this, select and right-click on the block design in the Sources window and select **Associate ELF Files**.

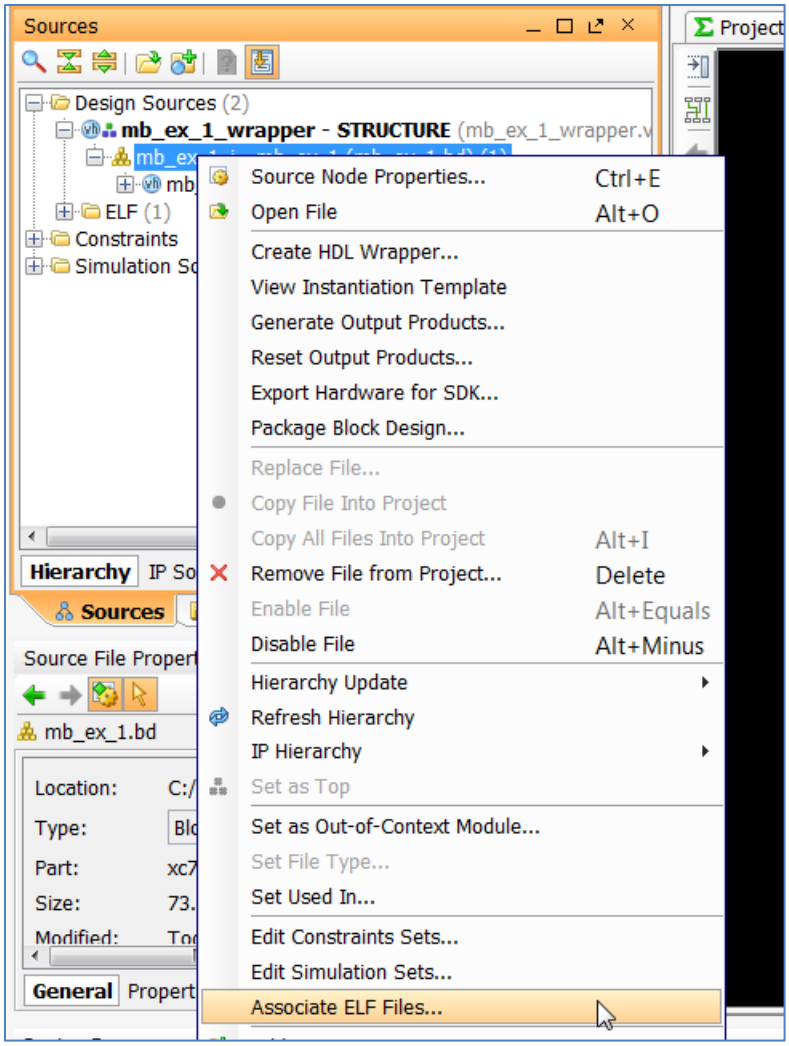


Figure 68: Selecting Associate ELF Files

8. You can associate an ELF for synthesis as well as simulation. Click on the appropriate browse icon (Under Design Sources or Simulation Sources) and browse to the newly added ELF file to the design.

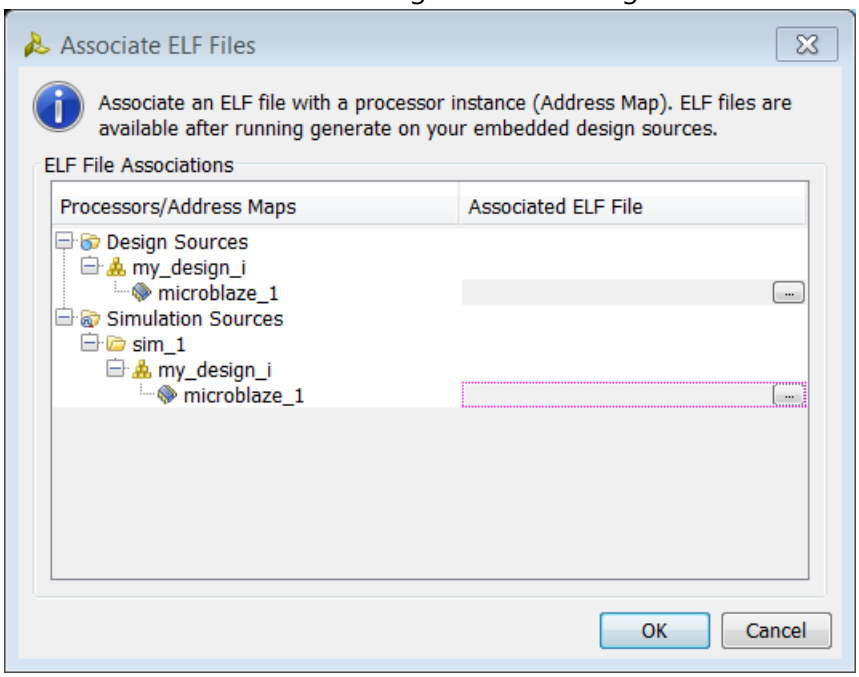


Figure 69: Associating ELF Files with a Microprocessor

The Associate ELF file dialog box opens. Highlight the file, as shown below, and click **OK**.

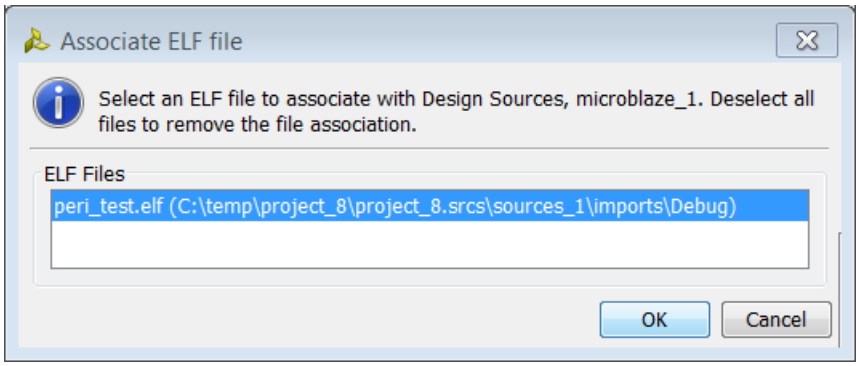


Figure 70: Highlight the ELF File to Associate

Make sure that the ELF file is populated in the Associated ELF File Column and click **OK**.

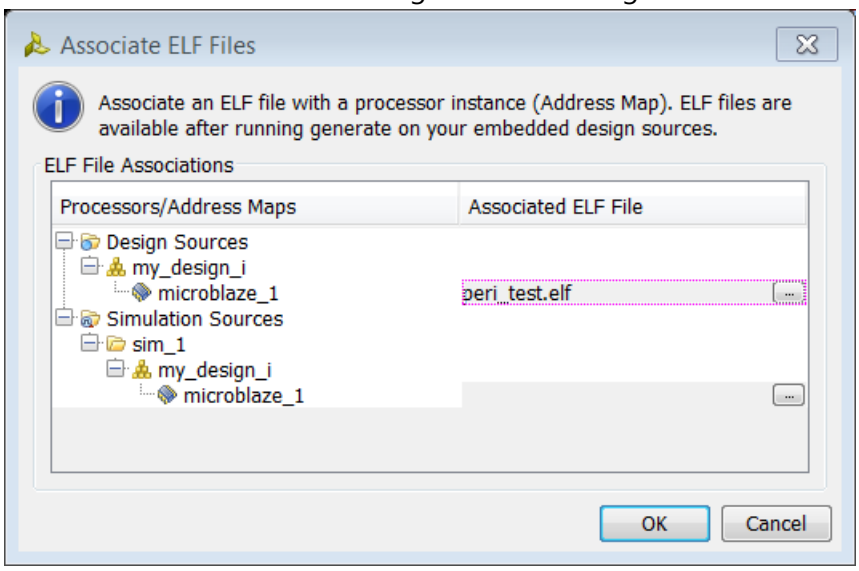


Figure 71: Making Sure the ELF File is Populated

Adding and Associating an ELF File in a Single Step

You can accomplish adding and associating an ELF file at once by following the following steps.

1. Right-click on the block design in Sources window and select Associate ELF files.
2. In the Associate ELF Files dialog box, click on the browse button on either the processor instance under Design Sources or Simulation Sources.

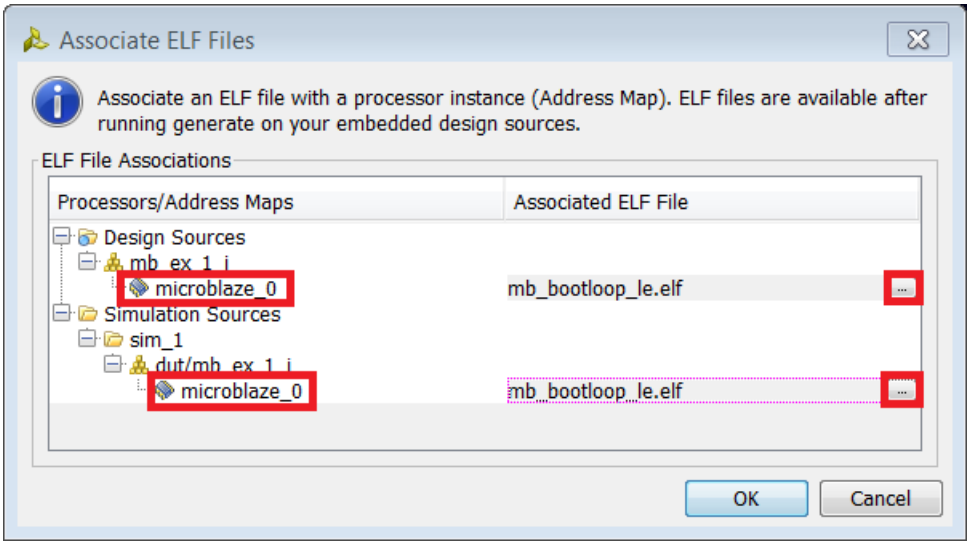


Figure 72: Adding and Associating an ELF file using the Associate ELF Files Dialog Box

3. When the Associate ELF File dialog box pops up, click on **Add Files**.

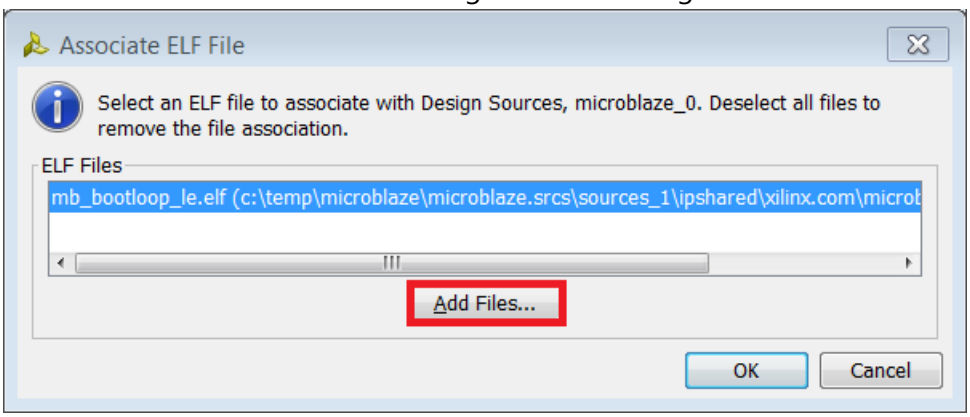


Figure 73: Click on Add Files Button to Add an ELF File

4. In the Add Source Files dialog box, navigate to the folder where the ELF file is located. Select the file and click **OK**.

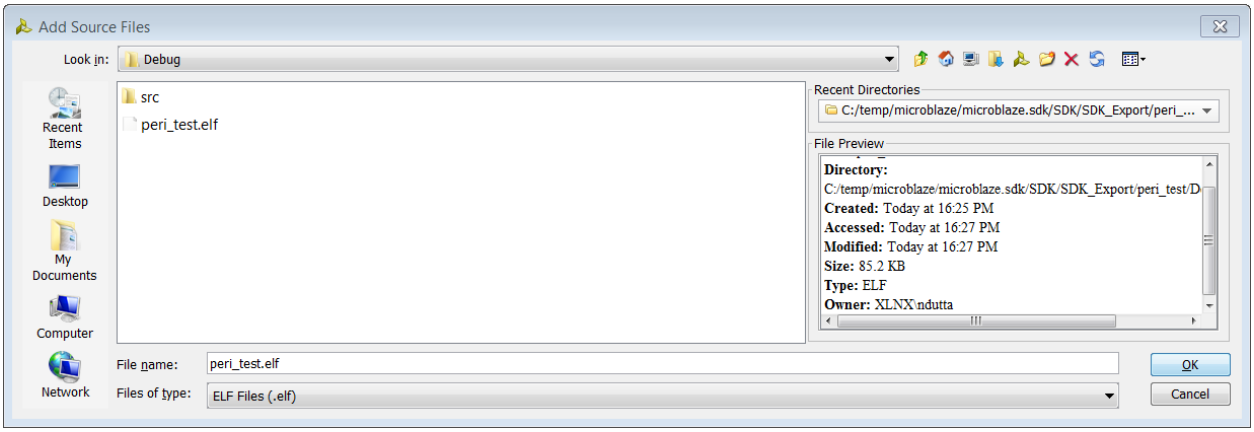


Figure 74: Navigate to the Directory where the ELF File is Located

5. Back in the Associate ELF file dialog box, select the newly added ELF file and click **OK**.

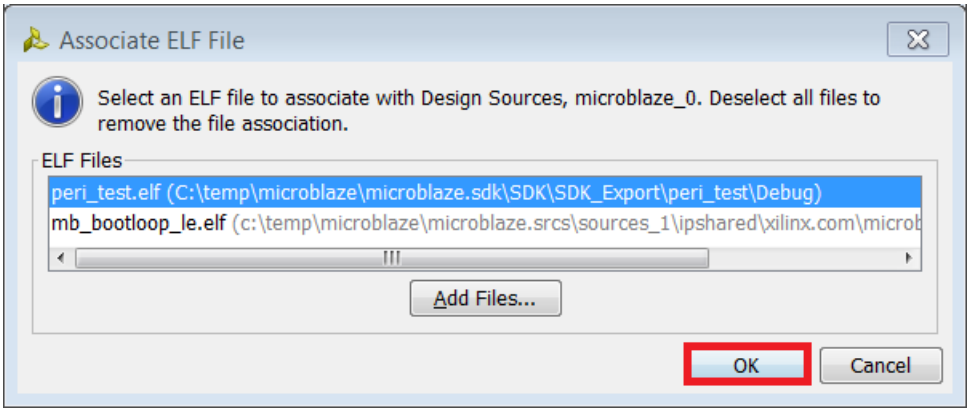


Figure 75: Associate the Newly Added ELF File

6. In the Associate ELF Files dialog box, verify that the new ELF file is shown in the processor instance field and click **OK**.

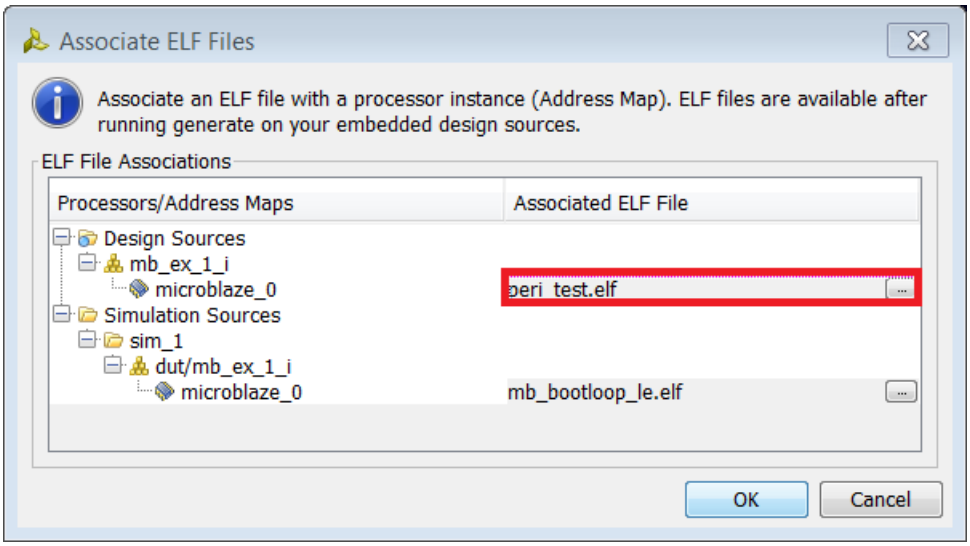


Figure 76: Verify that the Newly Added File is Associated to the Processor Instance

Setting the Block Design as an Out-of-Context (OOC) Module

Hierarchical Design flows enable you to partition the design into smaller, more manageable modules to be processed independently. In the Vivado Design Suite, these flows are based on the ability to implement a partitioned module out-of-context (OOC) from the rest of the design. The most common use situation in the context of IP integrator is that you can set the block design as an out-of-context module which can be synthesized and a design checkpoint (DCP) file created. This block design, if used as a part of the larger Vivado design, does not have to be re-synthesized every time other parts of the design (outside of IP integrator) are modified. This provides in considerable run-time improvements.

CAUTION! The Out-of-Context Mode can only be enabled for the entire block design. You cannot select individual IP in a block design and set them as Out-of-Context modules.

To set a block design as an out-of-context module, you highlight the block design, right-click and select **Set as Out-of-Context Module**.

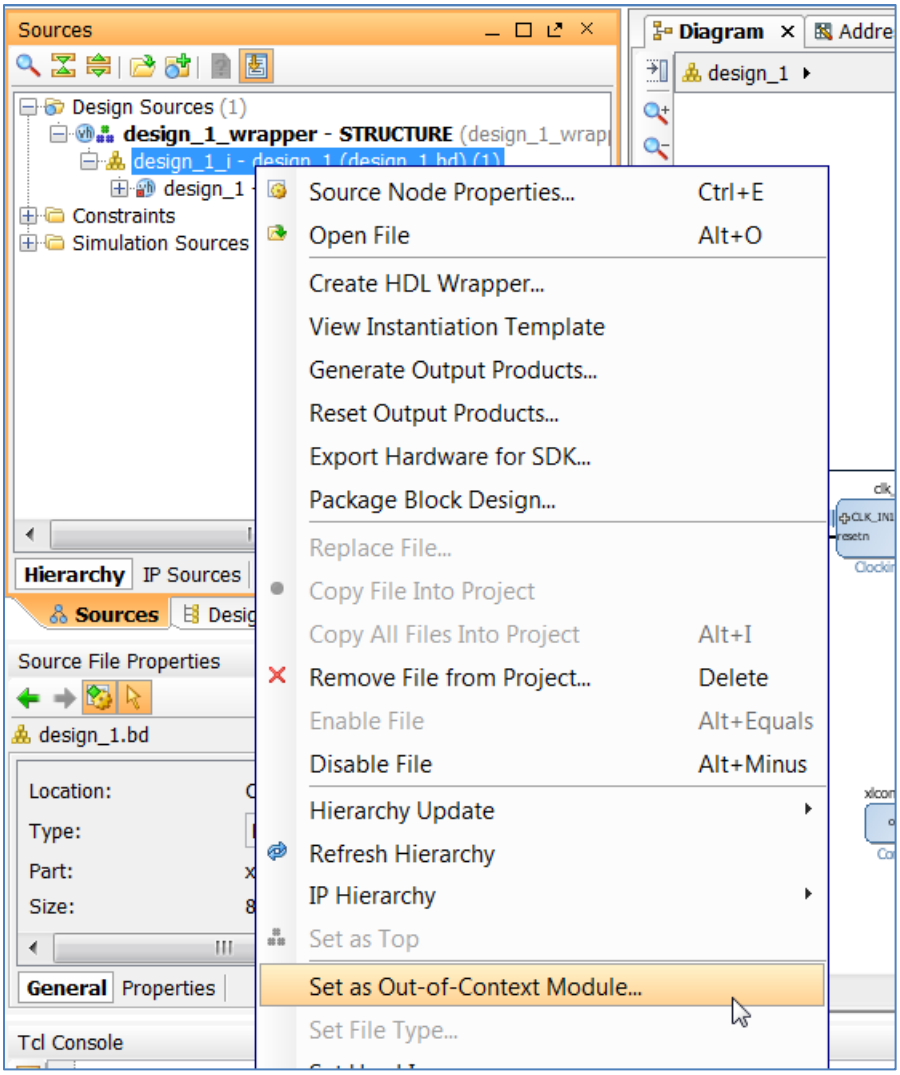


Figure 77: Setting a Block Design as an Out-of-Context Module

The **Set as Out-of-Context Module** dialog box opens informing you that the block design will be synthesized as an Out-of-Context module. You can also check the **Use auto-generated blackbox stub for top-level synthesis** if desired or you can use your own custom stub file (a wrapper essentially that instantiates the block design).

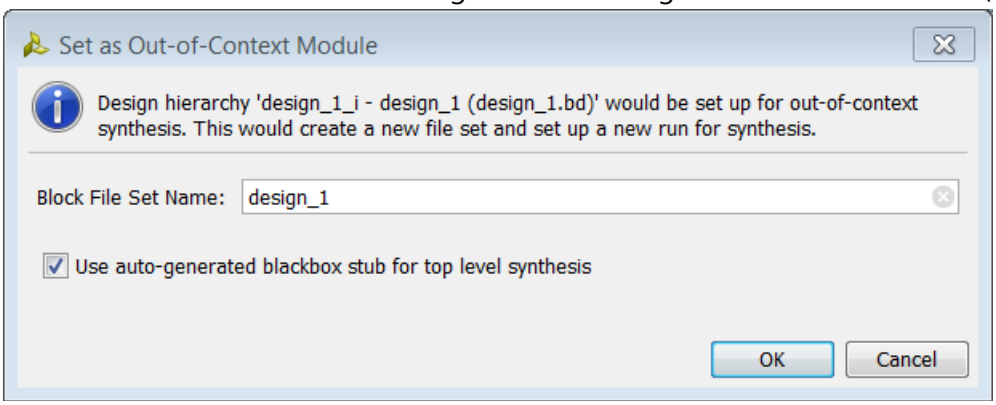


Figure 78: Set as Out-of-Context Dialog Box

You can see if the OOC feature has been enabled on the block design by verifying the square box against the block design in the Sources window as shown by the highlighted box in the following figure.

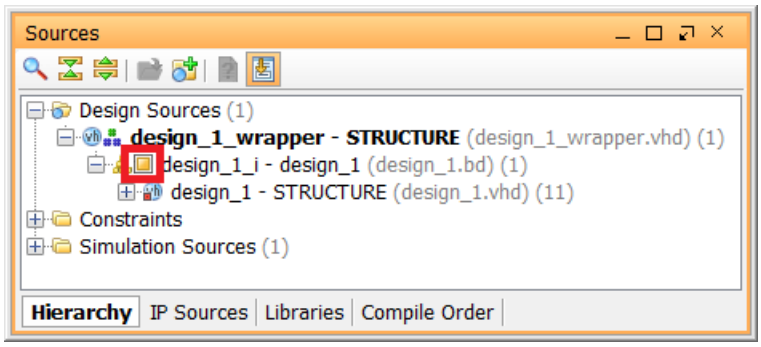


Figure 79: Verify that the Block Design has been Set as an OOC Module

You can then synthesize the block design by selecting **Run Synthesis** from the Flow Navigator.

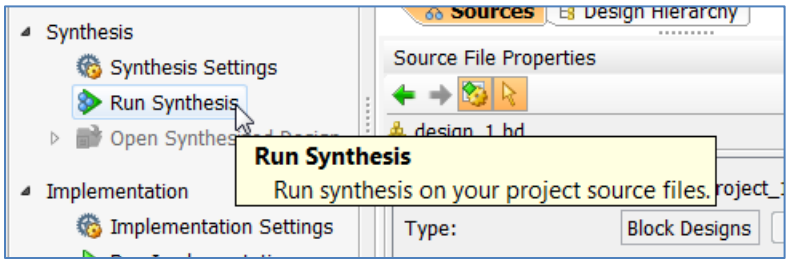


Figure 80: Run Synthesis on the Block Design Set as an OOC Module

As synthesis launches, you can see the Out-of-Context Module runs in the Design Runs window.

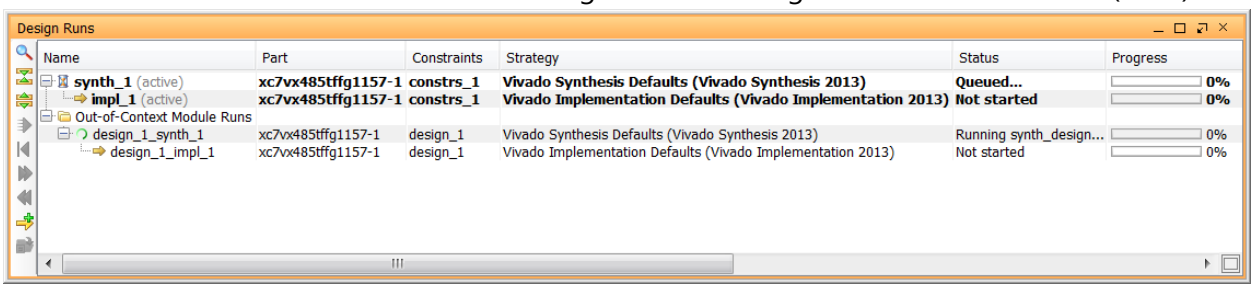


Figure 81: Design Runs Window View for Out-of-Context Flow

When the synthesis run completes, a design checkpoint file (DCP) is created which is an archive consisting of the synthesized netlist and all the constraints necessary for the block design. This DCP file can be found in the synthesis run directory which can be opened by selecting the design_1_synth_1 run, right-clicking and selecting **Open Run Directory**.

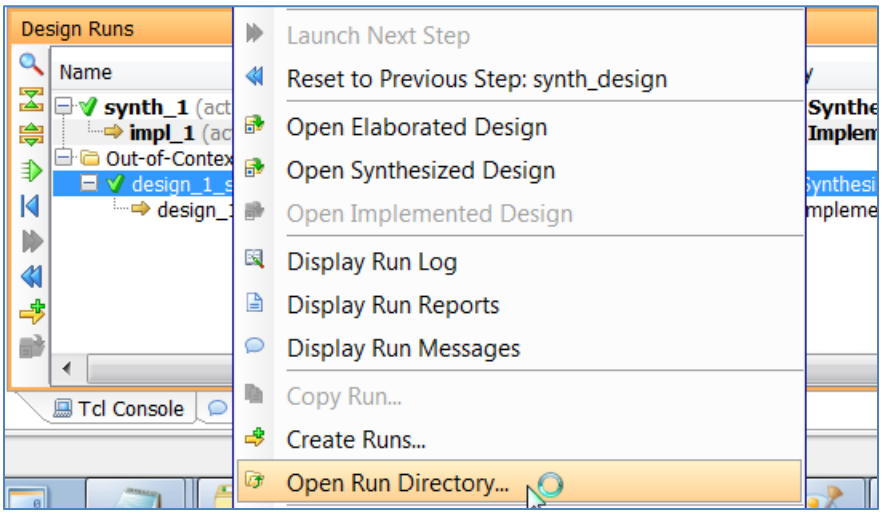


Figure 82: Open the Run Directory Containing the DCP File

The run directory opens in Windows Explorer.

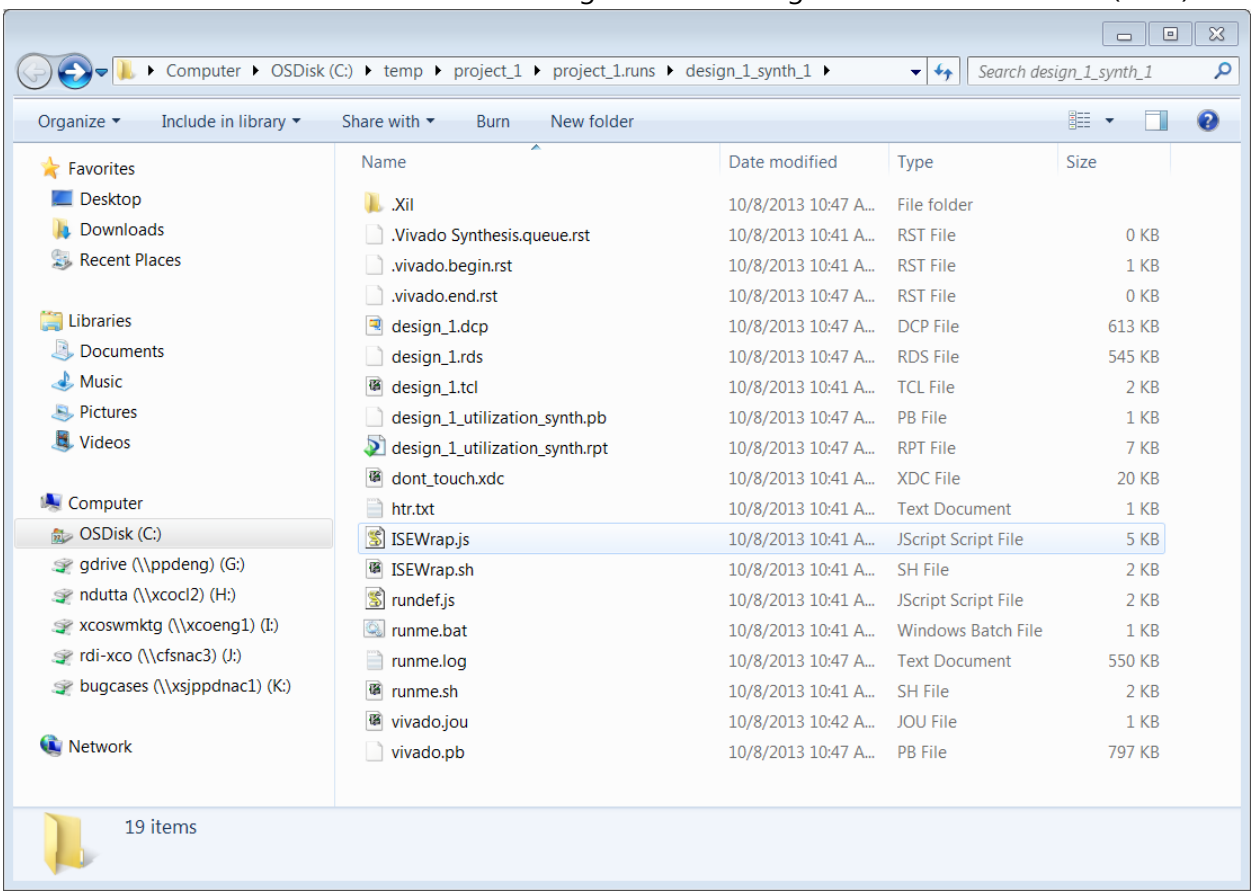


Figure 83: The Run Directory for the OOC Module Containing the DCP File

If the block design is added as a synthesized netlist in other designs, this DCP file can be added to the project where the block design is instantiated.

Chapter 3 Parameter Propagation in IP Integrator

Overview

Parameter propagation is one of the most powerful features available in IP integrator. The feature enables an IP to auto-update its parameterization based on how it is connected in the design. IP can be packaged with specific propagation rules, and IP integrator will run these rules as the diagram is generated. For example, in the following figure, IP0 has a 64-bit wide data bus. IP1 is then added and connected, as is IP2. In this case, IP2 has a default data bus width of 32 bits. When the parameter propagation rules are run, the user is alerted to the fact that IP2 has a different bus width. Assuming that the data bus width of IP2 can be changed through a change of parameter, IP integrator can automatically update IP2. If IP cannot be updated to match properties based on its connection, then an error will be shown, alerting users of potential issues in the design. This is a simple example, but demonstrates the power of parameter propagation. The types of errors that can be corrected or identified by parameter propagation are often errors not found until simulation.

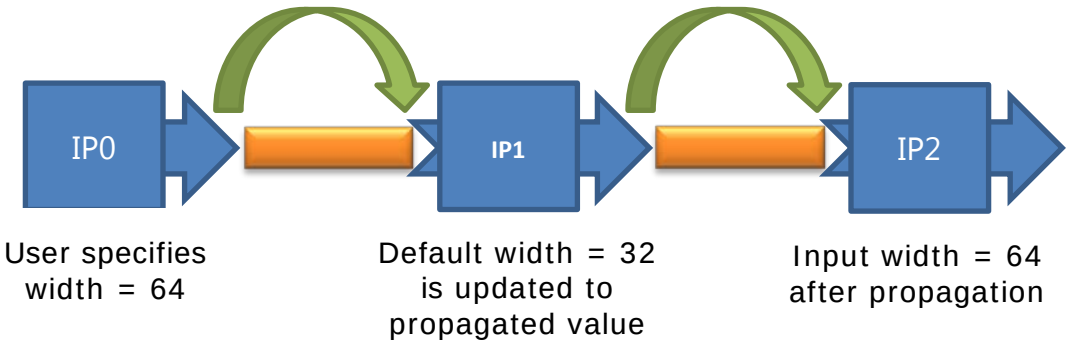


Figure 84: Parameter Propagation Concept

Bus Interfaces

A bus interface is a grouping of signals that share a common function. An AXI4-Lite master, for example, contains a large number of individual signals plus multiple buses, which are all required to make a connection. One of the important features of IP integrator is the ability to connect a logical group of bus interfaces from one IP to another, or from the IP to the boundary of the IP integrator design or even the FPGA IO boundary. Without the signals being packaged as a bus interface, the IP's symbol will show an extremely long and unusable list of low-level ports, which will be difficult to connect one by one.

A list of signals can be grouped in IP - XACT using the concept of a bus Interface with its constituent port map that maps the physical port (available on the IP's RTL or netlist) to a logical port as defined in the IP-XACT abstraction Definition file for that interface type.

Common Internal Bus Interfaces

Some common examples of bus interfaces are buses that conform to the AXI specification such as AXI4, AXI4Lite and AXI-Stream. The AXIMM interface includes all three subsets (AXI4, AXI3, and AXI4Lite). Other interfaces include BRAM.

IO Bus Interfaces

Some Bus Interfaces that group a set of signals going to IO ports are called I/O interfaces. Examples include UART, I2C, SPI, Ethernet, PCIe, DDR etc.

Special Signals

There are five standard signals identified that are used across a wide variety of IP. These interfaces are:

- Clock
- Reset
- Interrupt
- Clock Enable
- Data (for traditional arithmetic IP which do not have any AXI interface e.g. adder/subtractor, multiplier)

These special signals are described in the following sections:

Clock

The clock interface can have the following parameters associated with them. These parameters are used in the design generation process and are necessary when the IP is used with other IP in the design.

- ASSOCIATED_BUSIF: The list contains names of bus interfaces, which run at this clock frequency. This parameter takes a colon ":" separated list of strings as its value. If all interface signals at the boundary do not run at this clock rate, then this field is left blank.
- ASSOCIATED_RESET: The list contains names of reset ports (not names of reset container interfaces) as its value. This parameter takes a colon ":" separated list of strings as its value. If there are no resets in the design, then this field is left blank.
- ASSOCIATED_CLKEN: The list contains names of clock enable ports (not names of container interfaces) as its value. This parameter takes a colon ":" separated list of strings as its value. If there are no clock enable signals in the design, then this field is left blank.
- FREQ_HZ: This parameter captures the frequency in hertz at which the clock is running in positive integer format. This parameter needs to be specified for all output clocks only.
- PHASE: This parameter captures the phase at which the clock is running. The default value is 0. Valid values are 0 to 360. If you cannot specify the PHASE in a fixed manner, then you must update it in bd.tcl, similar to updating FREQ_HZ.
- CLK_DOMAIN: This parameter is a string id. By default, IP integrator assumes that all output clocks are independent and assigns unique ID to all clock outputs across the block design. This is automatically assigned by IP integrator, or managed by IP if there are multiple output clocks of the same domain.

To see the properties on the clock net, select the source clock port/pin and analyze the properties on the port/pin in question.

Figure 85: Analyzing the Clock Properties in IP Integrator

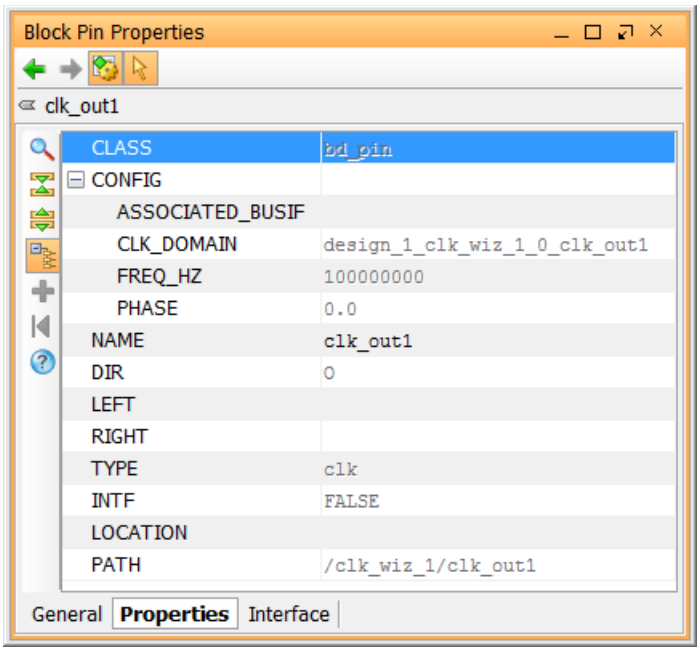


Figure 86: Clock Properties

These properties can also be reported by the following Tcl command:

```
report_property [get_bd_intf_ports sys_diff_clock]
```

```
report_property [get_bd_intf_ports /sys_diff_clock]
Property        Type    Read-only  Visible  Value
CLASS           string  true       true     bd_intf_port
CONFIG.FREQ_HZ  string  false      true     200000000
LOCATION        string  false      true     140 390
MODE            string  true       true     Slave
NAME            string  false      true     sys_diff_clock
PATH            string  true       true     /sys_diff_clock
VLNV            string  true       true     xilinx.com:interface:diff_clock_rtl:1.0
```

Figure 87: Reporting Clock Properties using Tcl Command

You can also double click on the pin to see the configuration dialog box.

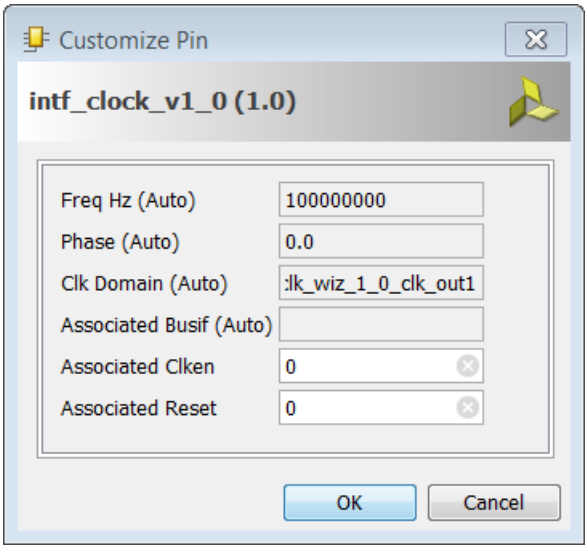


Figure 88: Customize Pin dialog box

Reset

This container bus interface should also include the following parameters with it:

- POLARITY: Valid values for this parameter are ACTIVE_HIGH or ACTIVE_LOW. The default is ACTIVE_LOW.

To see the properties on the clock net, select the reset port/pin and analyze the properties on the port/pin in question.

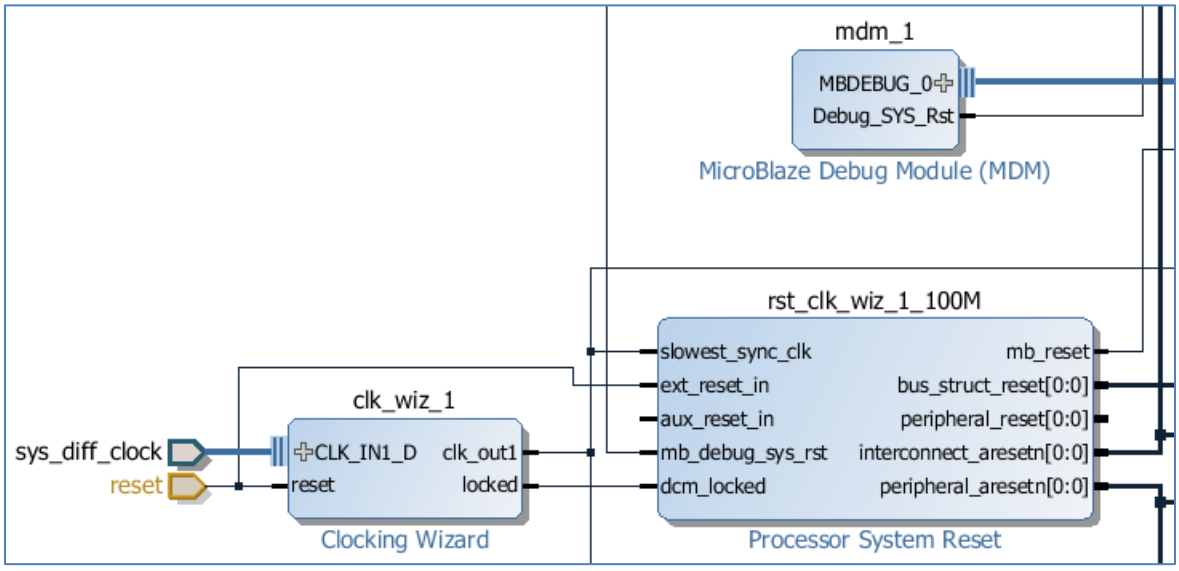


Figure 70: Analyzing the reset Properties in IP Integrator

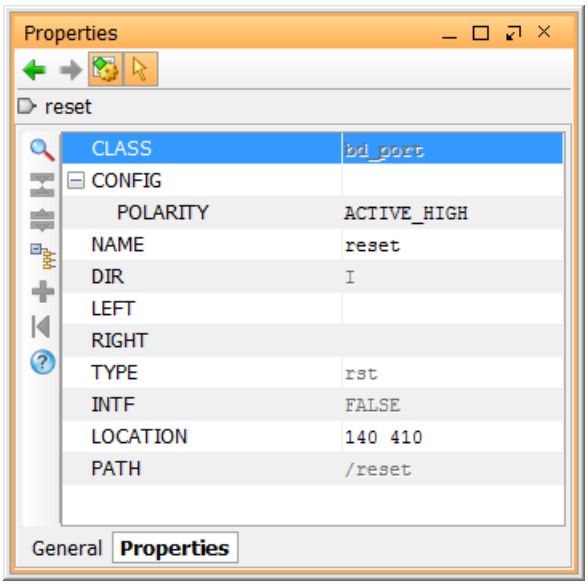


Figure 71: Reset properties

These properties can also be reported by the following Tcl command:

```
report_property [get_bd_ports reset]
```

```
report_property [get_bd_ports /reset]
Property          Type    Read-only  Visible  Value
CLASS             string  true       true     bd_port
CONFIG.POLARITY   string  false      true     ACTIVE_HIGH
DIR               string  true       true     I
INTF              string  true       true     FALSE
LEFT              string  false      true
LOCATION          string  false      true     140 410
NAME              string  false      true     reset
PATH              string  true       true     /reset
RIGHT             string  false      true
TYPE              string  true       true     rst
```

Figure 89: Reporting reset Properties using Tcl Command

Interrupt

This bus interface includes the following parameters:

- SENSITIVITY: Valid values for this parameter are LEVEL_HIGH, LEVEL_LOW, EDGE_RISING, and EDGE_FALLING. The default is LEVEL_HIGH.

To see the properties on the interrupt pin highlight the pin as shown below and look at the properties window.

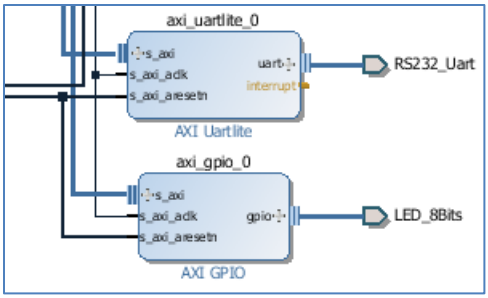


Figure 73: Analyzing the Interrupt Properties in IP Integrator

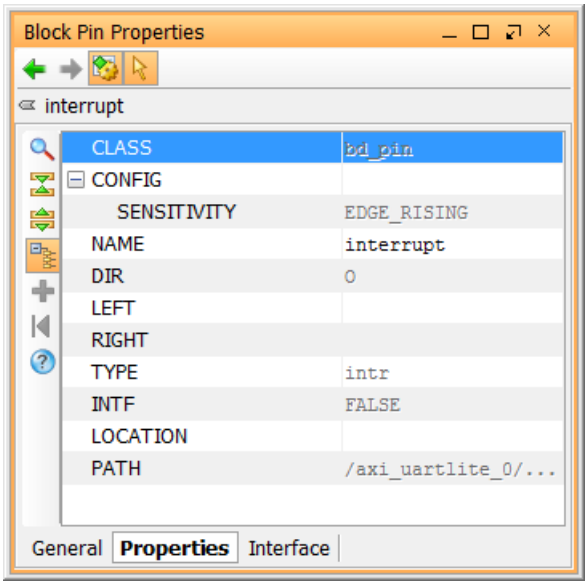


Figure 74: Interrupt Properties

These properties can also be reported by using the following Tcl command:

```
report_property [get_bd_pins /axi_uartlite_0/interrupt]
```

```
report_property [get_bd_pins /axi_uartlite_0/interrupt]
Property            Type    Read-only  Visible  Value
CLASS               string  true       true     bd_pin
CONFIG.SENSITIVITY  string  true       true     EDGE_RISING
DIR                 string  true       true     O
INTF                string  true       true     FALSE
LEFT                string  true       true
LOCATION            string  false      true
NAME                string  false      true     interrupt
PATH                string  true       true     /axi_uartlite_0/interrupt
RIGHT               string  true       true
TYPE                string  true       true     intr
```

Figure 75: Reporting Interrupt Properties using Tcl Command

Clock Enable

There are two parameters associated with Clock Enable: FREQ_HZ and PHASE.

How Parameter Propagation Works

In IP integrator, parameter propagation takes place when you choose to run Validate Design. You can do this in one of the following ways:

- Click on **Validate Design** in the Vivado® IDE toolbar.

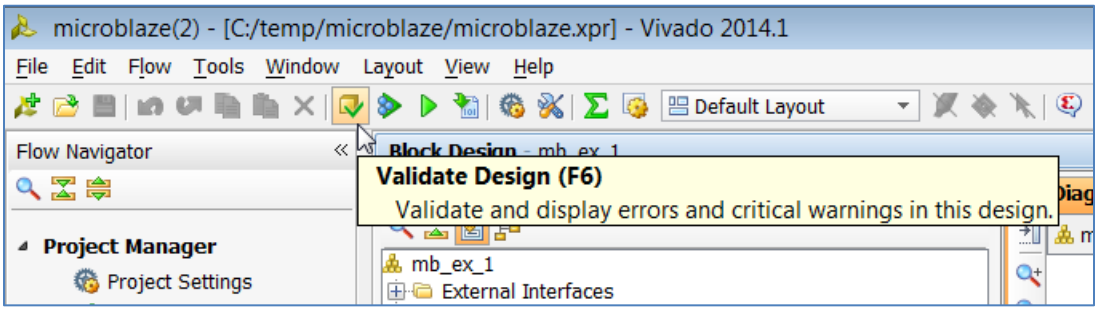


Figure 76: Validating a Design from the Vivado IDE Toolbar

- Click on **Validate Design** in the IP integrator toolbar:

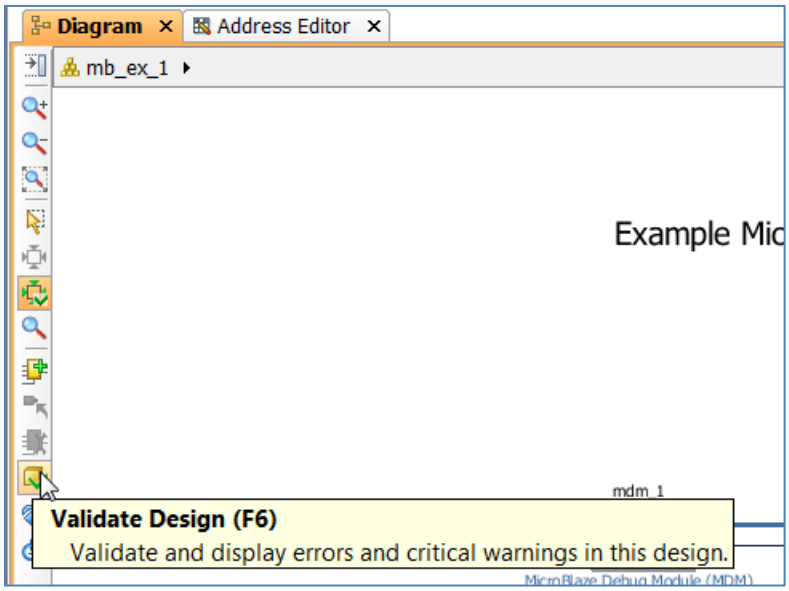


Figure 77: Validating a Design from the IP Integrator Toolbar

- Click on **Tools > Validate Design** from the Vivado Menu.

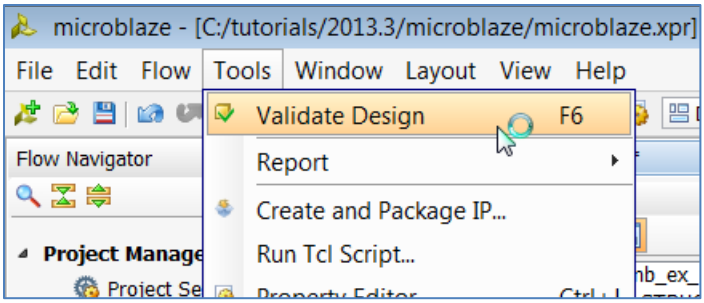


Figure 78: Validating a Design from the Vivado IDE Pulldown Menu

- Design validation can be done by using the Tcl command:
  `validate_bd_design`

The propagation Tcl provides a mechanism to synchronize an IP instance's configuration with that of other instances connected to it. The synchronization of configuration happens at bus interface parameters.

IP integrator's parameter propagation works primarily on the concept of assignment strength for an interface parameter. An interface parameter can have a strength of USER, CONSTANT, PROPAGATED, or DEFAULT. When the tool compares parameters across a connection, it always copies a parameter with higher strength to a parameter with lower strength.

Parameters in the Customization GUI

In the Non-Project Mode, you must configure all user parameters of an IP. However, in the context of IP integrator, any user parameters that are auto updated by parameter propagation are grayed out in the IP customization dialog box. A grayed-out parameter is an indication that you should not set the specific-user parameters directly on the IP; instead, the property values are auto-computed by the tool.

There are situations when the auto-computed values may not be optimal. In those circumstances, you may override these propagated values.

There are four different cases that you will encounter related to parameter propagation.

- Auto-computed parameters – these parameters are auto-computed by IP integrator and you cannot override them. For example, the **Ext Reset Logic Level** parameter in the following figure is greyed out and **Auto** is placed next to the parameter denoting that you cannot change it.

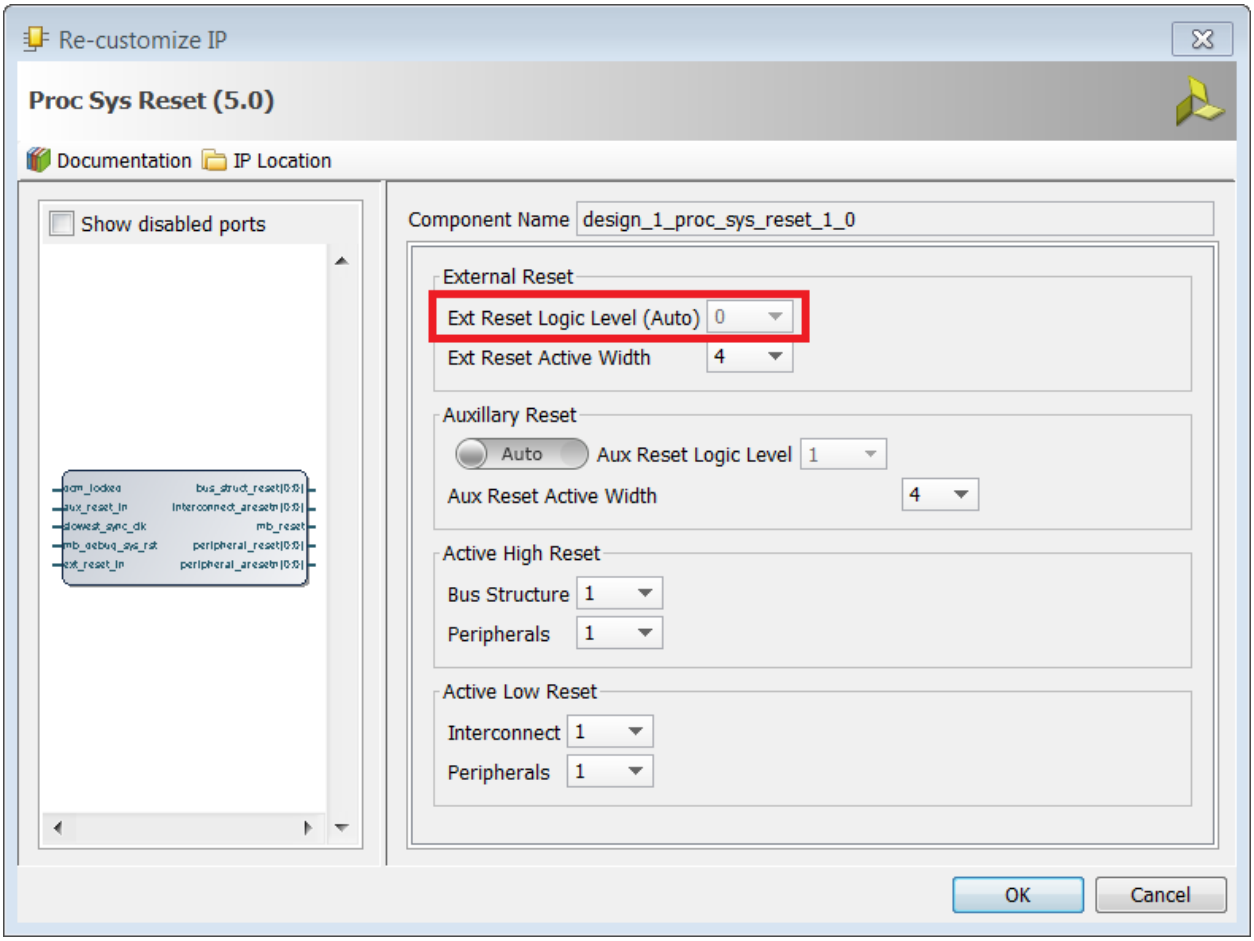


Figure 79: Example of Auto-Computed Parameter

- Override-able parameters – Parameters that you can override. For example, you can change the SLMB Address Decode Mask for the LMB BRAM Controller. When you hover the mouse on top of the slider button, it will tell you that the parameter is controlled by the system; however, you can change it by toggling the button to User from Auto.

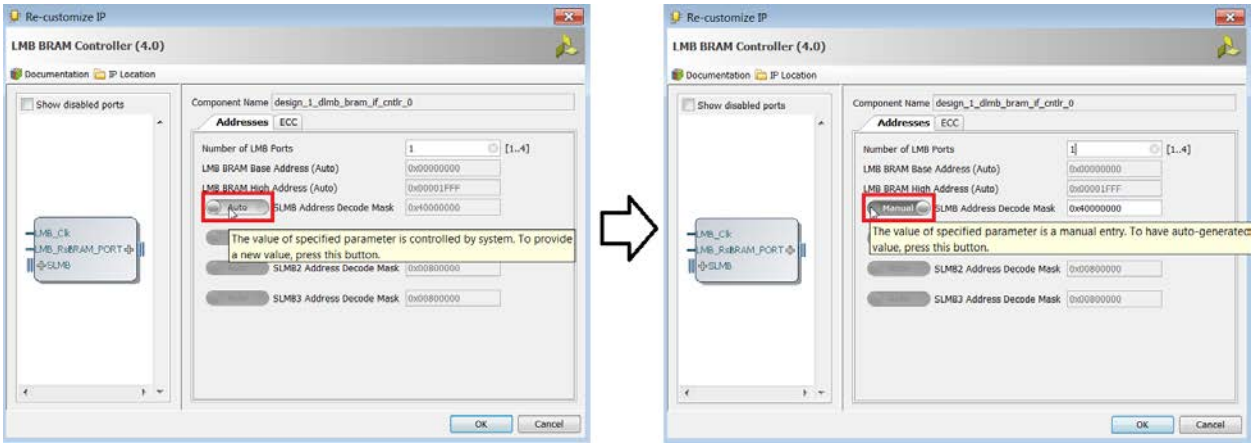


Figure 80: Example of User Override-able Parameter

- User configurable parameters – These parameters are user configurable only and are to be set by the user.

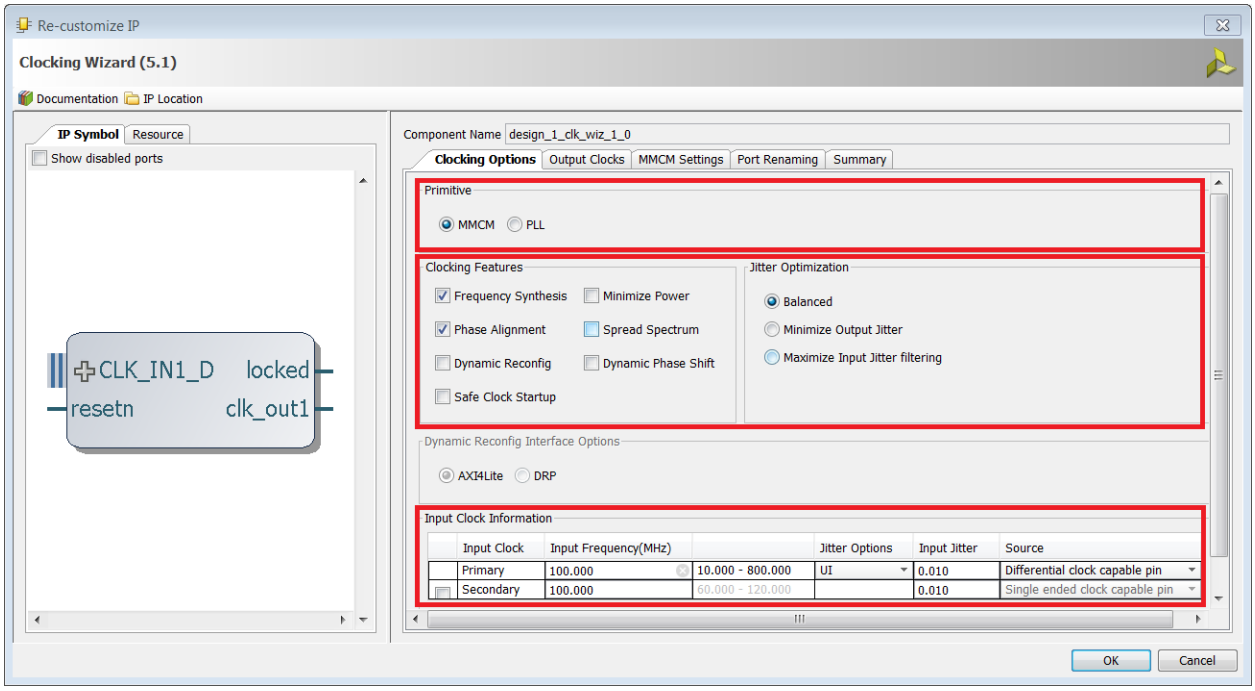


Figure 81: Example of User-Configurable Parameters

- Constants – These are parameters that cannot be set by anyone.

Example of a Parameter Mismatch

The following is an example of a parameter mismatch on the FREQ_HZ property of a clock pin. This error revealed when the design is validated.

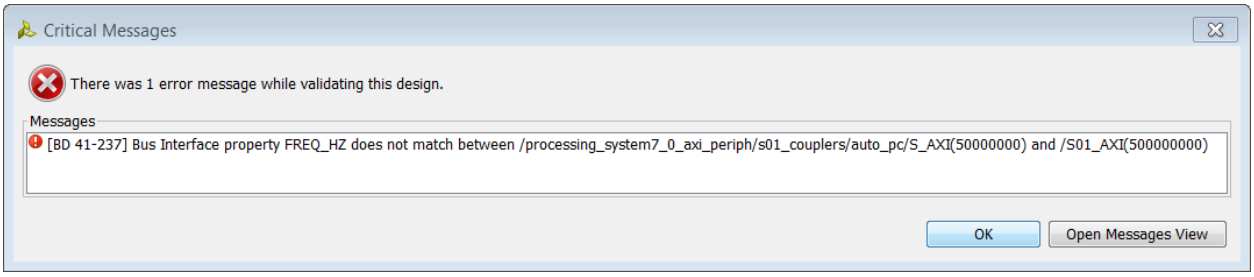


Figure 90: FREQ_HZ property mismatch between the source and destination

In this example, the frequency does not match between the S01_AXI port and the S_AXI interface of the AXI Interconnect.

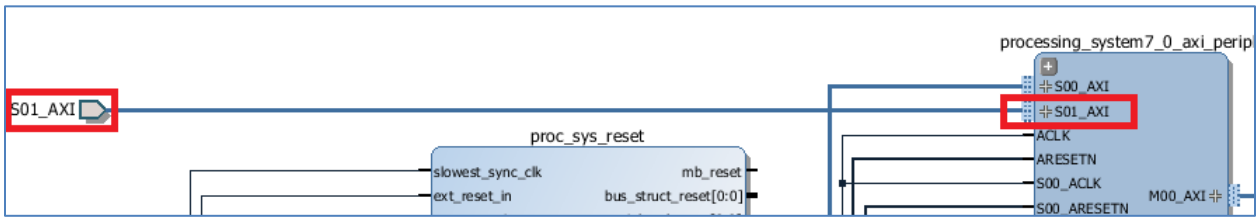


Figure 91: Frequency mismatch between a port and an interface pin of the AXI Interconnect

The port S01_AXI has a frequency of 500 MHz as can be seen in the properties window whereas the S01_AXI interface of the AXI Interconnect is set to a frequency of 50 MHz.

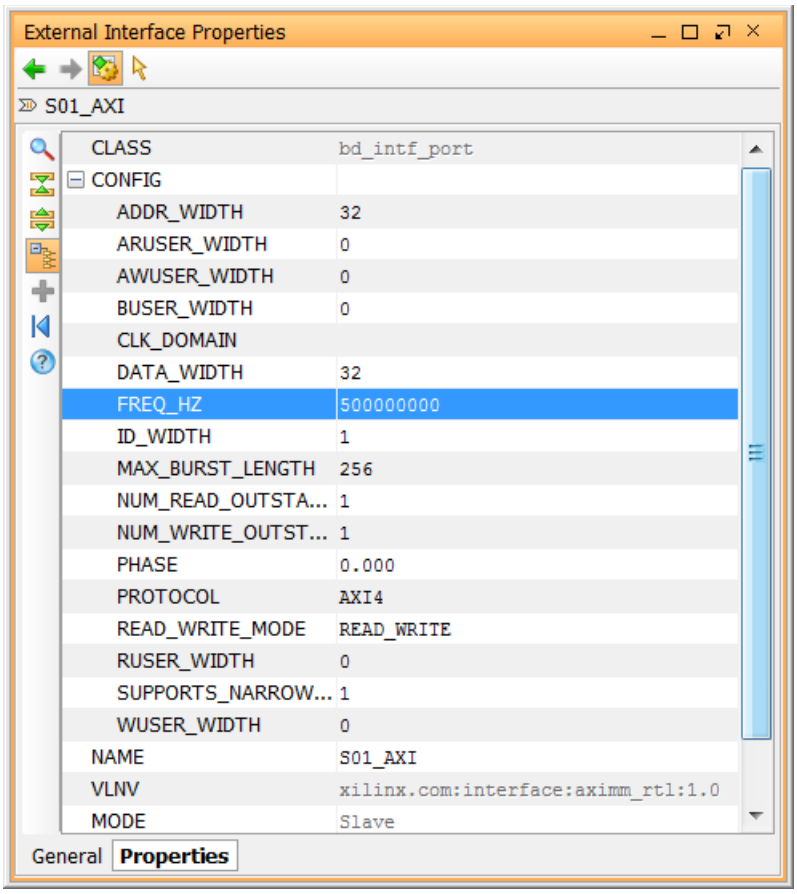


Figure 92: Change the Frequency of the Port in the Properties Window

This type of error can be easily fixed by changing the frequency in the property window by setting the FREQ_HZ property to 50000000. It can also be fixed by double-clicking on the S01_AXI port and setting the right frequency in the Frequency field of the customization dialog box.

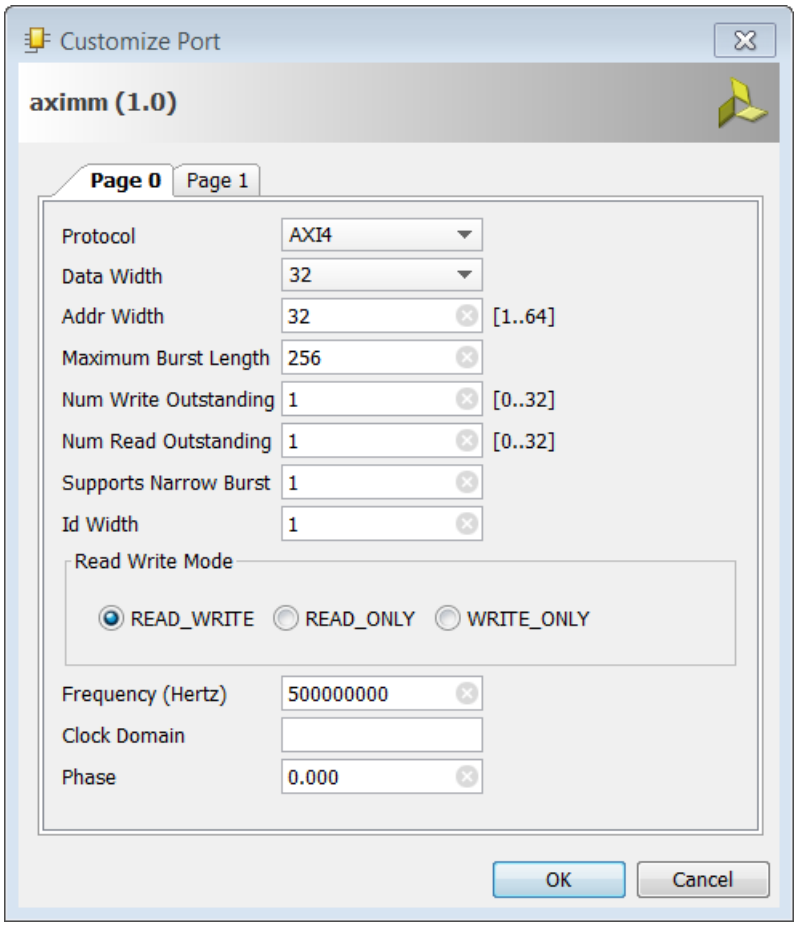


Figure 93: Change Frequency from the Customize Port Dialog Box

Once the frequency has been changed, validate design again to make sure that there are no errors.

Chapter 4 Debugging IP Integrator Designs

Overview

In-system debugging allows you to debug your design in real-time on your target hardware. This is an essential step in design completion. Invariably, you will come across a situation which is extremely hard to replicate in a simulator. Therefore, there is a need to debug the problem right there in the FPGA. In this step, you place an instrument into your design with special debugging hardware to provide you with the ability to observe and control the design. After the debugging process is complete, you can remove the instrumentation or special hardware to increase performance and reduce logic.

IP integrator provides ways to instrument your design for debugging which is explained in the following sections. There are two flows explained: the HDL instantiation flow and the netlist insertion flow. Choosing the flow depends on your preference and types of nets/signals that you are interested in debugging. As an example, if you are interested in performing hardware-software co-verification using the cross-trigger feature of MicroBlaze or Zynq processor, then you can use the HDL instantiation flow. If you are interested in analyzing I/O ports and internal nets, then netlist insertion will be the way to go. In most of the cases, you will be using a combination of both flows to debug your design.

Using the HDL Instantiation Flow in IP Integrator

You can instantiate an Integrated Logic Analyzer (ILA) in the IP integrator design and connect nets to the ILA that you are interested in probing. You can instantiate an ILA by following the steps described below.

1. Right-click on the block design canvas and select Add IP.

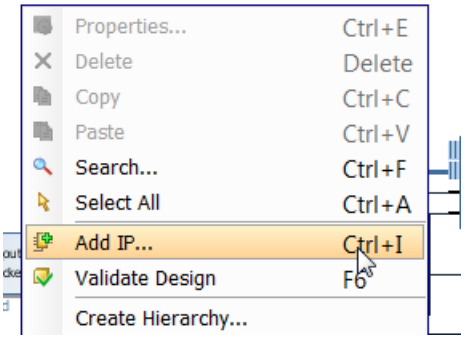


Figure 94: Add IP from the context menu

2. In the IP catalog type ILA in the search field, select and double click on the ILA core to instantiate it on the IP integrator canvas. The ILA core gets instantiated on the IP integrator canvas.

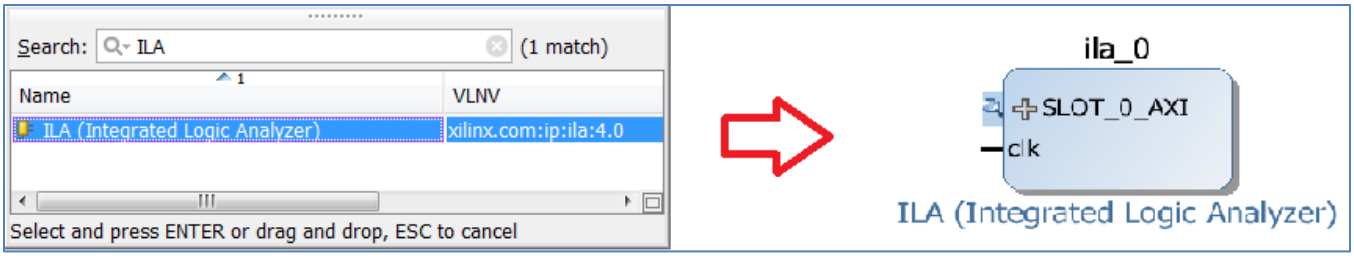


Figure 95: Instantiate ILA Core

3. Double click on the ILA core to reconfigure it. The Re-Customize IP dialog box opens.

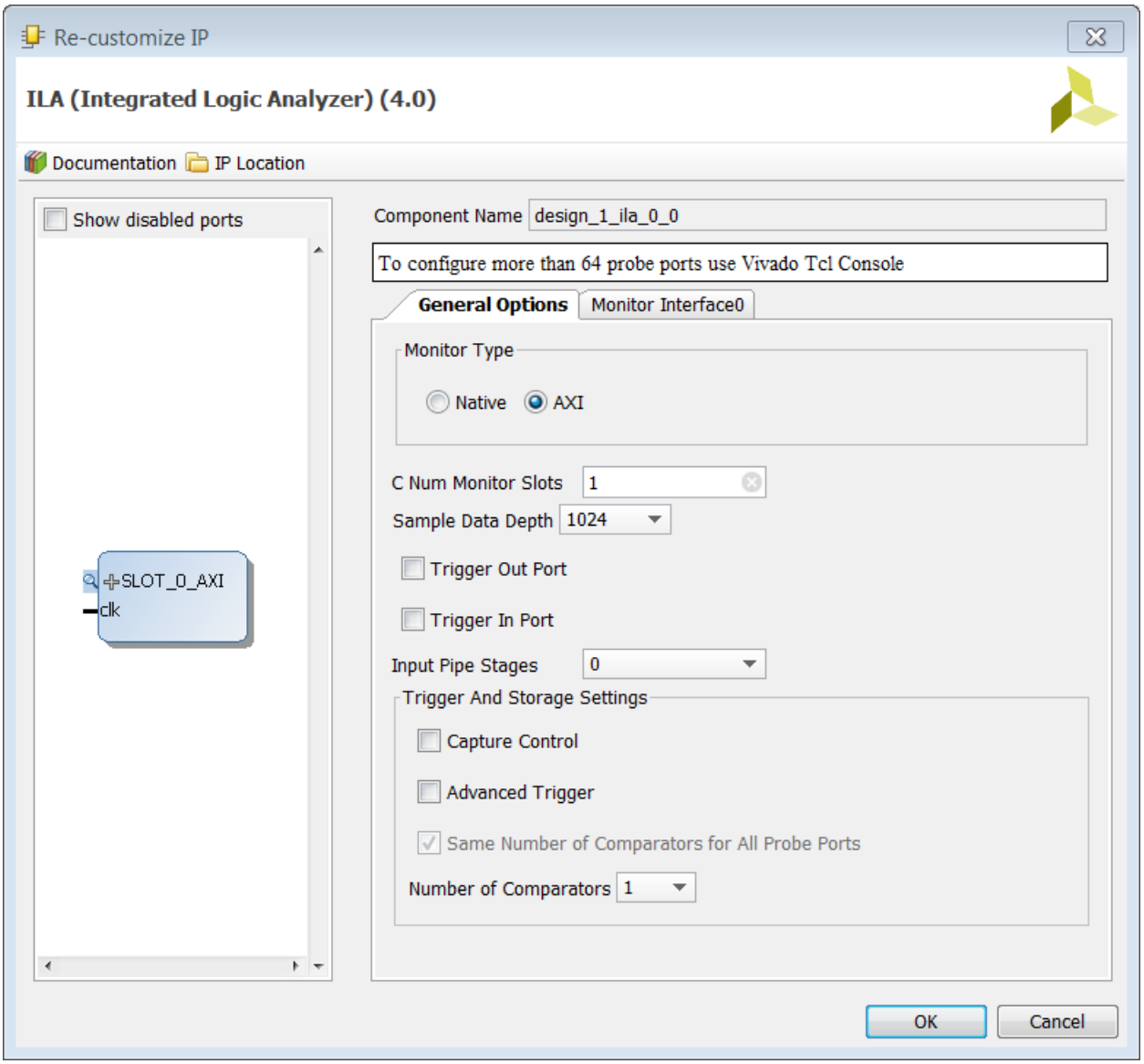


Figure 96: Re-customize IP Dialog Box for the ILA Core

The default option under the General Options tab shows **AXI** as the Monitor Type. Keep this selection in case you are monitoring an entire AXI interface. Change the Monitor Type to **Native** if you are monitoring non-AXI interface signals. You can change the Sample Data Depth and other fields as desired. For more information, please refer to the following document: *Vivado Design Suite User Guide: Programming and Debugging (**UG908**)*

CAUTION! You can only monitor one AXI interface using an ILA. Do not change the value of the C Num Monitor Slots. If more than one AXI interface is desired to be debugged, then instantiate more ILA cores as needed.

In case the Monitor Type is set to Native, you can set the Number of Probes to the desired value. This value should be set to the number of signals that need to be monitored.

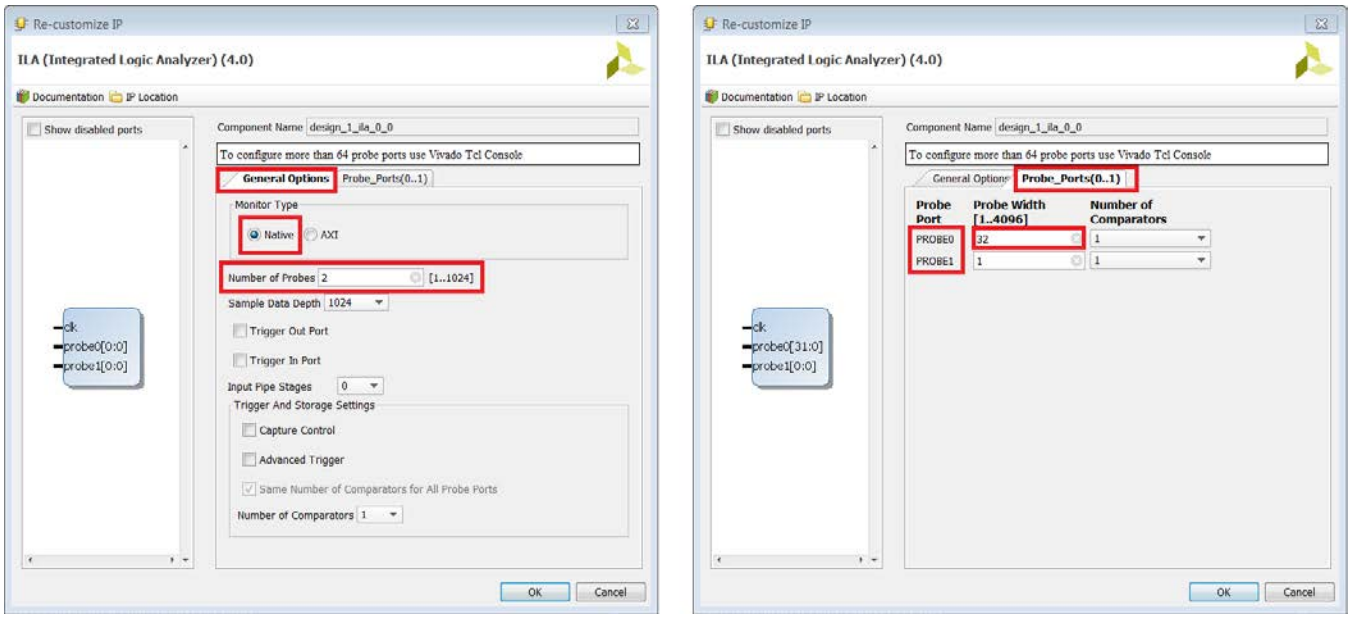


Figure 97: Choosing the Native Monitor Type

In the figure above, the number of Probes is set to 2 in the General Options tab. You can see under the Probe_Ports tab that two ports are displayed. The width of these ports can be set to the desired value. So assuming that you want to monitor a 32-bit bus, set the Probe Width for Probe 0 to 32. Once the ILA is configured, the changes are reflected on the IP integrator canvas.

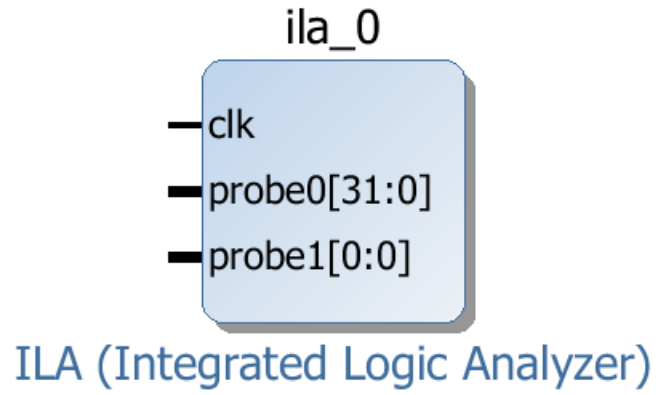


Figure 98: ILA Core after Making Changes in the Re-customize IP Dialog Box

4. Once the ILA is configured as desired, connection can be made to the pins of the ILA on the IP integrator canvas.

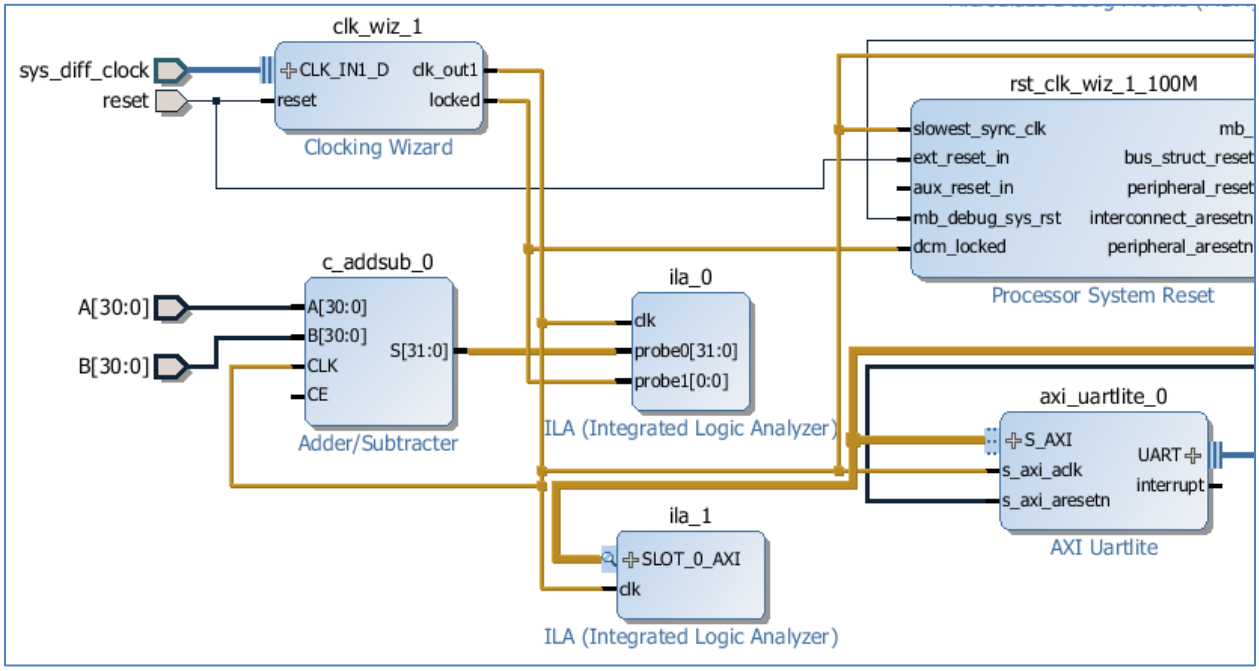


Figure 99: Instantiating ILAs to Monitor AXI and Non-AXI signals.

CAUTION! If a pin connected to an I/O port needs to be debugged, then Mark Debug should be used to mark the nets for debug. This is explained in the following section

---

5. Follow on to synthesize, implement and generate bitstream.

Using the Netlist Insertion Flow in IP Integrator

In this flow, you mark nets that you are interested in analyzing in the block design. Marking nets for debug in the block design offers more control in terms of identifying debug signals during coding and enabling/disabling debugging later in the flow.

Marking Nets for Debug in the Block Design

1. Nets can be marked for debug in the block design by highlighting them, right-clicking and selecting **Mark Debug**.

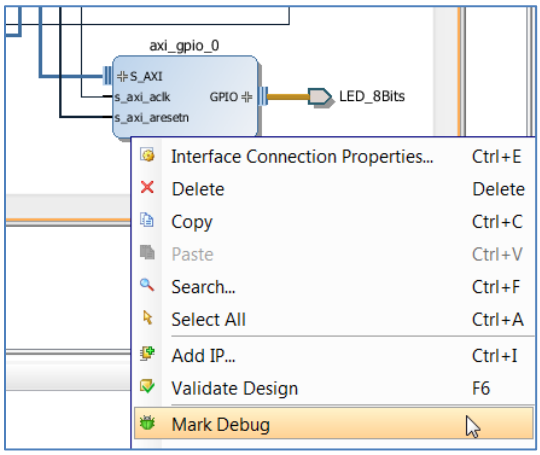


Figure 86 Mark Nets for Debug

The nets that have been marked for debug will show a small bug icon placed on top of the net in the block design. Likewise, a bug icon can be seen placed on the nets to be debugged in the Design Hierarchy window as well, as shown in the following figure.

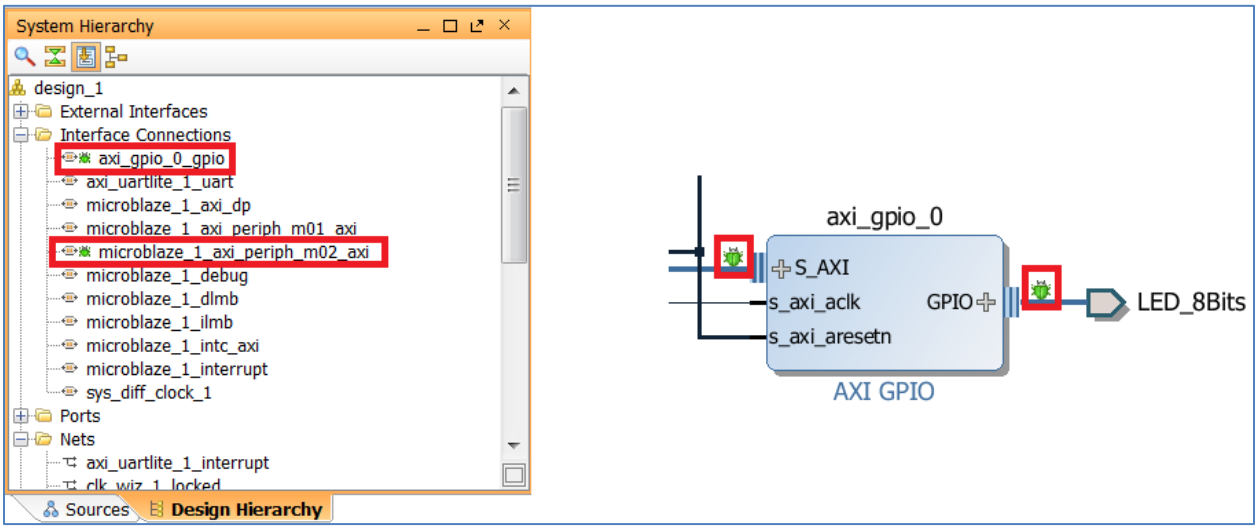


Figure 87: Identifying Nets Marked for Debug

TIP: Multiple nets can be marked for debug at the same time by highlighting them together, right-clicking and selecting **Mark Debug**.

2. You can Generate Output Products by either clicking on **Generate Block Design** in the Flow Navigator or by highlighting the block design in the sources window, right-clicking and selecting **Generate Output Products**.

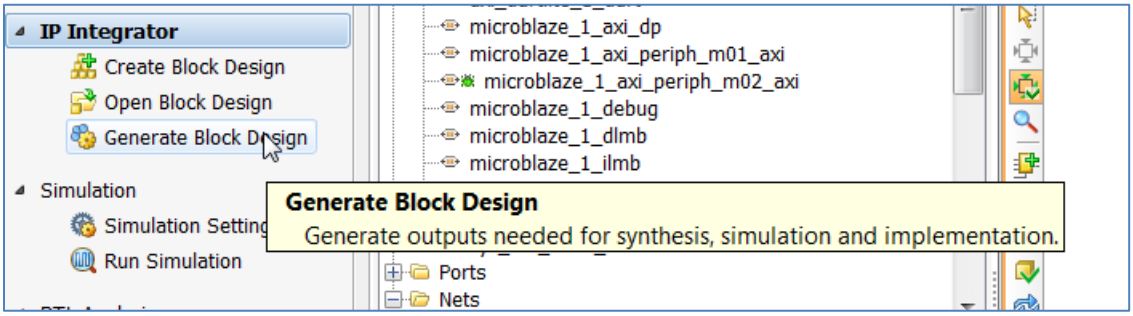


Figure 88: Generate Output Products

3. In the Generate Output Products dialog box, click **Generate**.

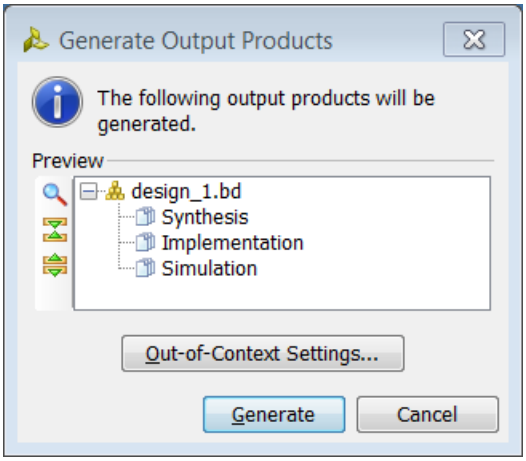


Figure 89: Generate Output Products Dialog Box

4. Marking the nets for debug places the MARK_DEBUG attribute on the net which can be seen in the generated top-level HDL file. This prevents the Vivado® tools from optimizing and renaming the nets.

```
1799  signal VCC_1 : STD_LOGIC;
1800  signal axi_gpio_0_gpio_TRI_O : STD_LOGIC_VECTOR ( 7 downto 0 );
1801  attribute MARK_DEBUG : boolean;
1802  attribute MARK_DEBUG of axi_gpio_0_gpio_TRI_O : signal is true;
1803  signal axi_uartlite_1_interrupt : STD_LOGIC;
1804  signal axi_uartlite_1_uart_RxD : STD_LOGIC;
1805  signal axi_uartlite_1_uart_TxD : STD_LOGIC;
1806  signal clk_wiz_1_locked : STD_LOGIC;
1807  signal mdm_1_debug_sys_rst : STD_LOGIC;
```

Figure 90: MARK_DEBUG Attributes in the Generated HDL File

Synthesize the Design and Insert the ILA Core

1. The next step is to synthesize the design by clicking on **Run Synthesis** from the Flow Navigator under the Synthesis drop-down list.

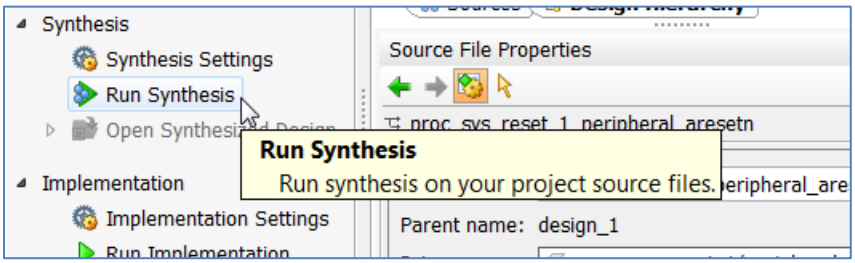


Figure 91: Synthesize the Design

2. After synthesis finishes, the Synthesis Completed pop-up dialog box appears. You select **Open Synthesized Design** to open the netlist, and then click **OK**.

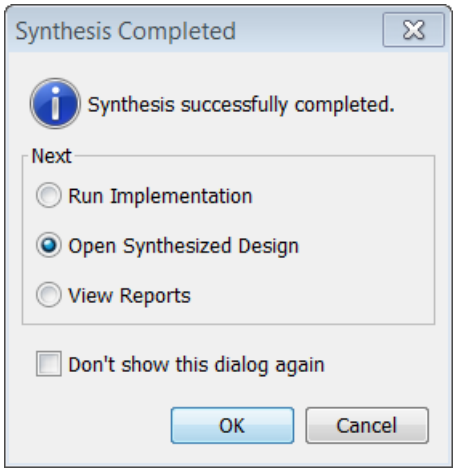


Figure 92: Synthesis Completed Dialog Box

3. The Schematic and the Debug window opens. If the Debug window at the bottom of the GUI is not open, you can always open that window by choosing **Windows > Debug** from the menu.

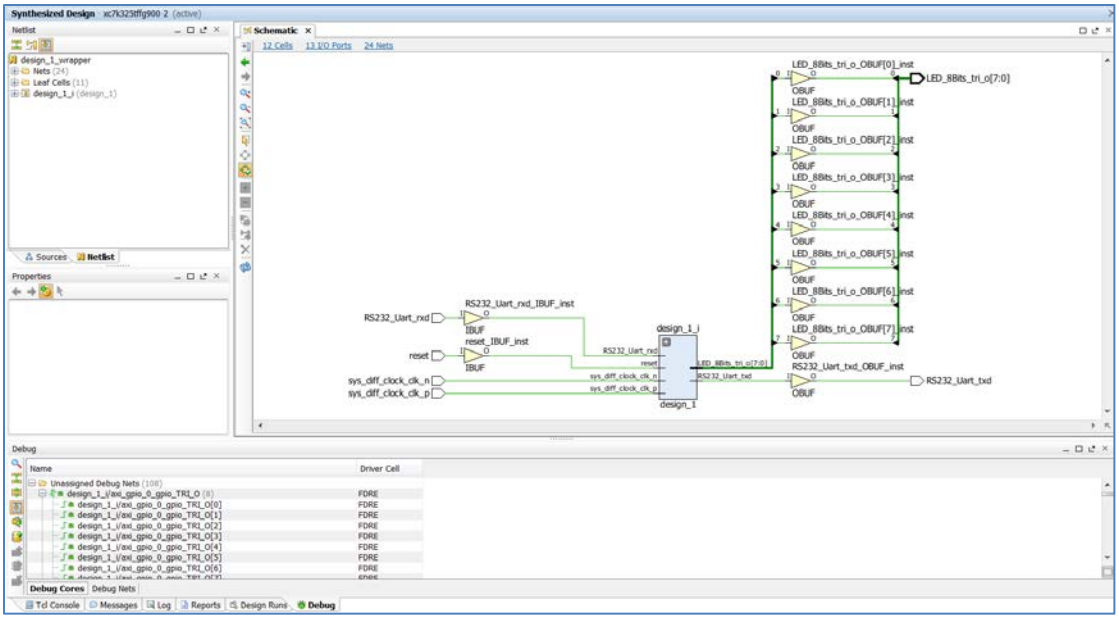


Figure 93: Schematic and Debug Window View in the Vivado IDE

4. You can see all the nets that were marked for debug in the Debug window under the folder **Unassigned Debug Nets**. These nets need to be connected to the probes of an Integrated Logic Analyzer. This is the step where you insert an ILA core and connect these unassigned nets to the probes of the ILA. You click on the **Setup Debug** icon in the Debug window toolbar. Alternatively, you can also select **Tools > Setup Debug** from the menu.

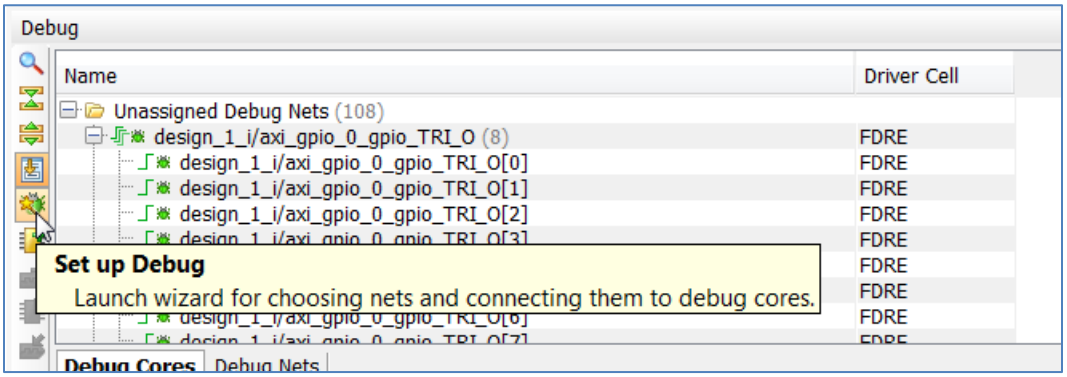


Figure 94: Setup Debug

5. The Setup Debug dialog box opens. You then click **Next**.

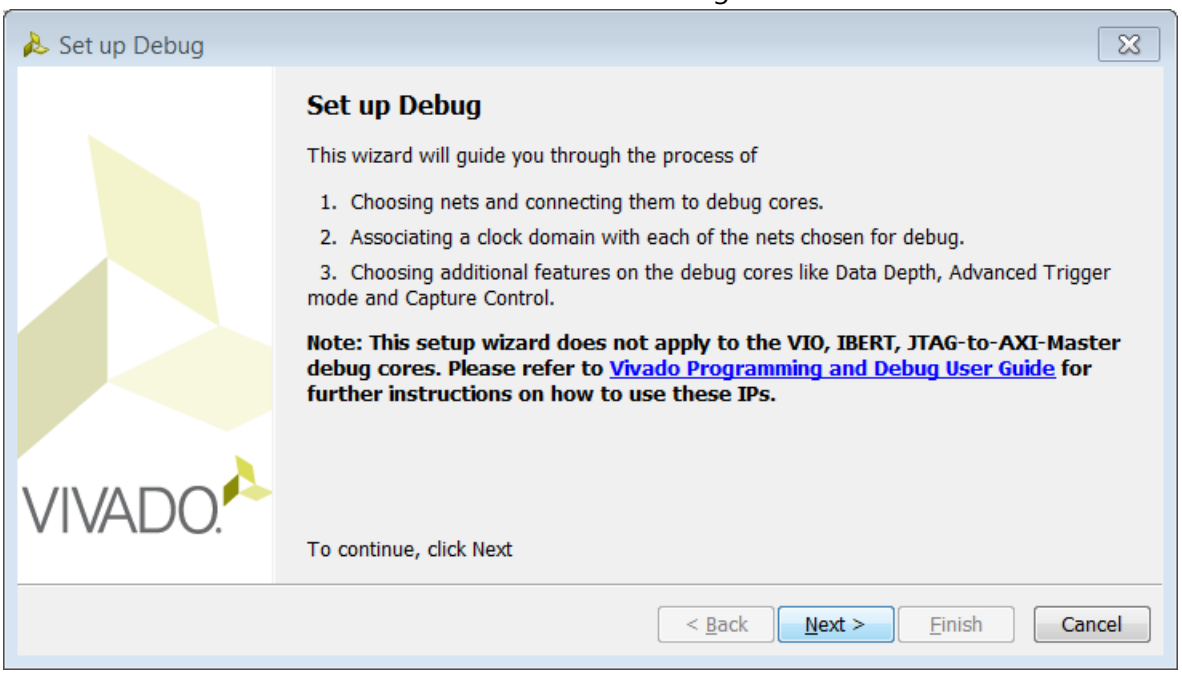


Figure 95: The Set up Debug Dialog Box

6. The Specify Nets to Debug page appears. In this page you can select a subset (or all) of the nets that you want to debug. Every signal must be associated with the same clock in an ILA. If the clock domain association cannot be found by the tool, you will manually associate those nets to a clock domain by selecting all the nets that have the Clock Domain column specified as **undefined**.

CAUTION! You need to mark the entire interfaces that you are interested in debugging. However, if you are concerned with device resource usage, then the nets you do not need for debugging can be deleted while setting up the debug core.

***NOTE**: One ILA per clock domain is inferred by the Set up Debug dialog box.*

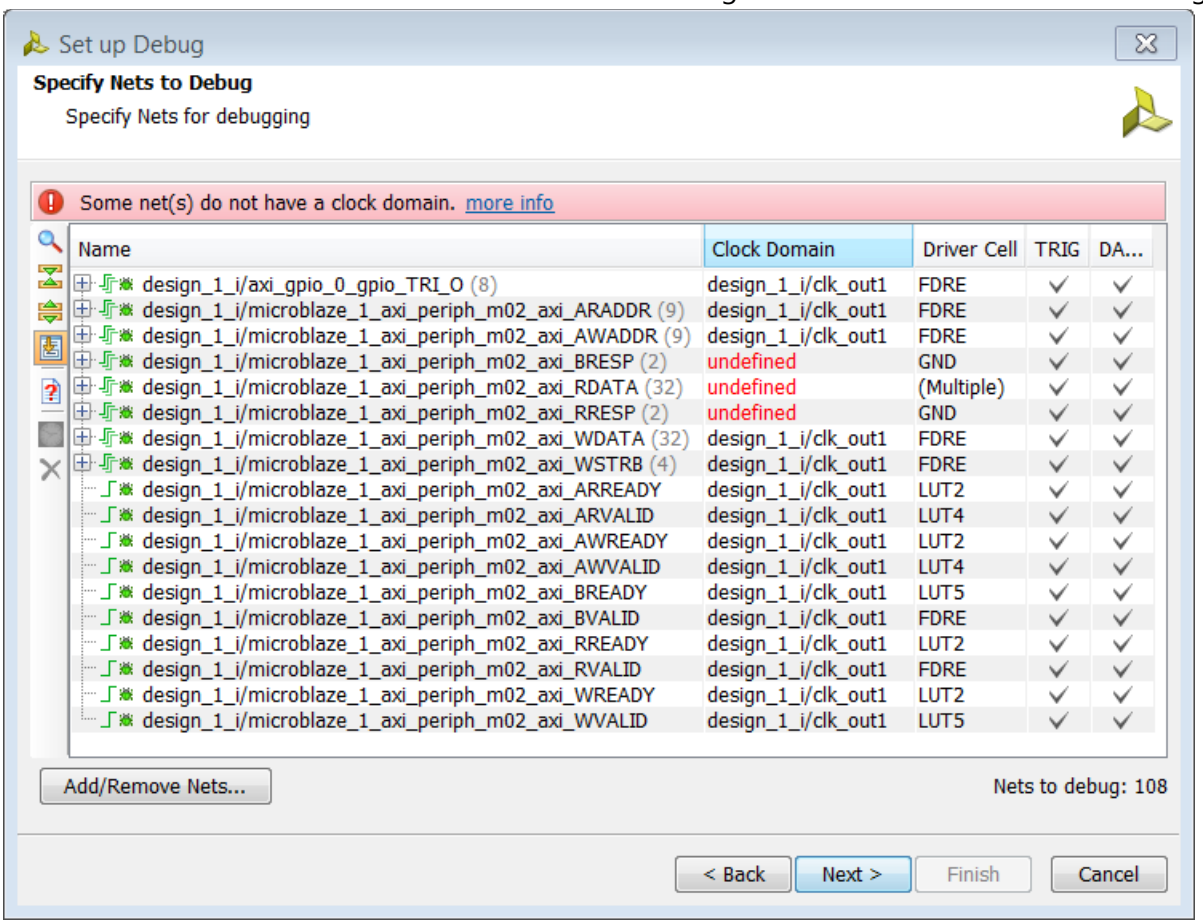


Figure 96: Selecting a Subset (or all) Nets Marked for Debug

To associate a clock domain to the signals that have the Clock Domain column specified as **undefined**, select all the nets in question, right-click and choose **Select Clock Domain**.

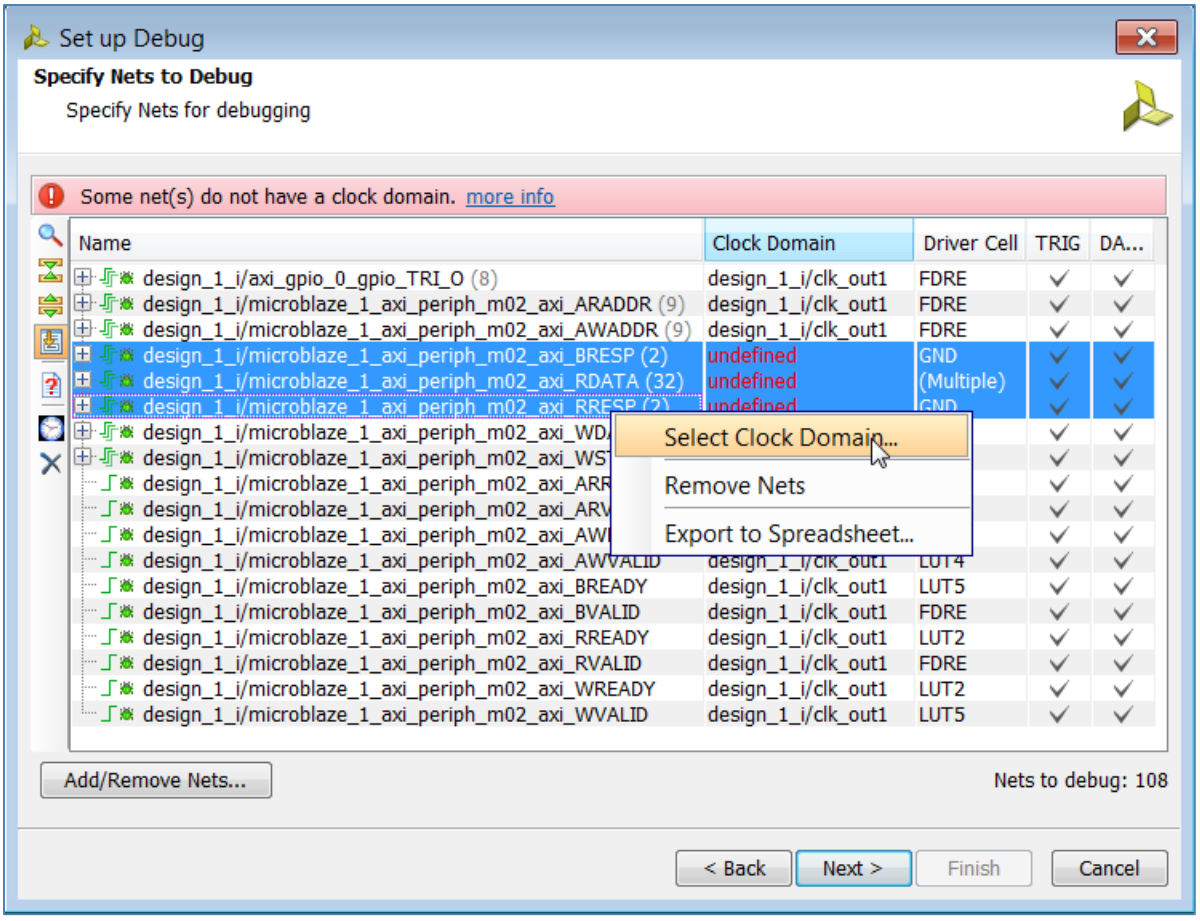


Figure 97: Select Clock Domain

7. In the Select Clock Domain dialog box, select the clock for the nets in question and click **OK**.

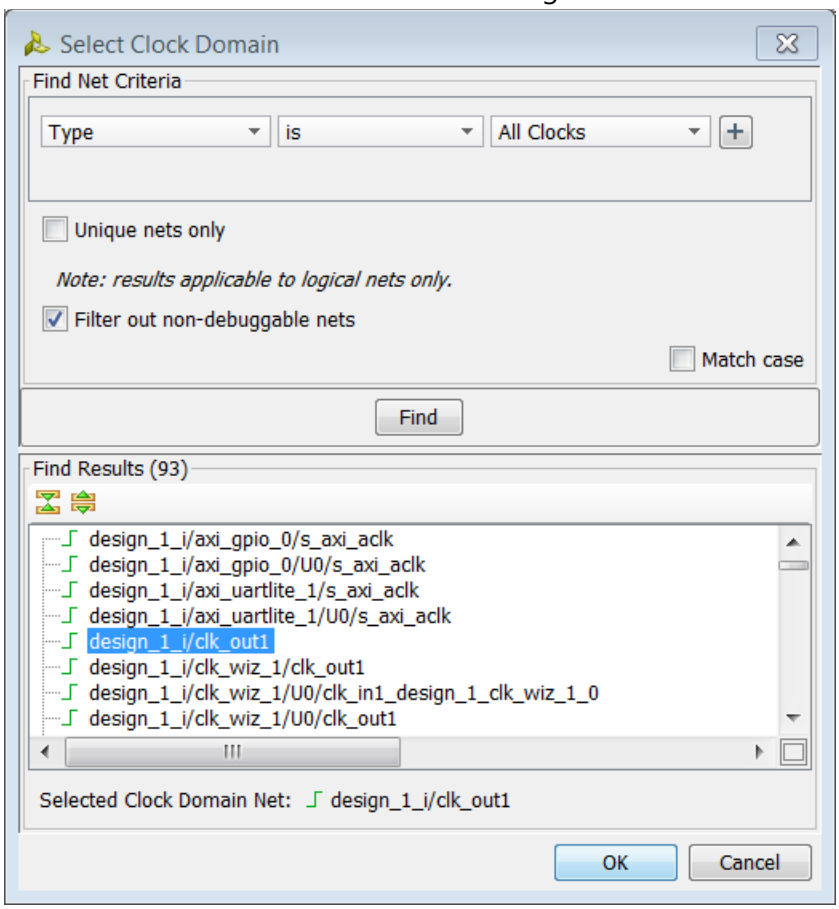


Figure 98: Select Clock Domain Dialog Box

8. In the Specify Nets to Debug dialog box, you click **Next**.

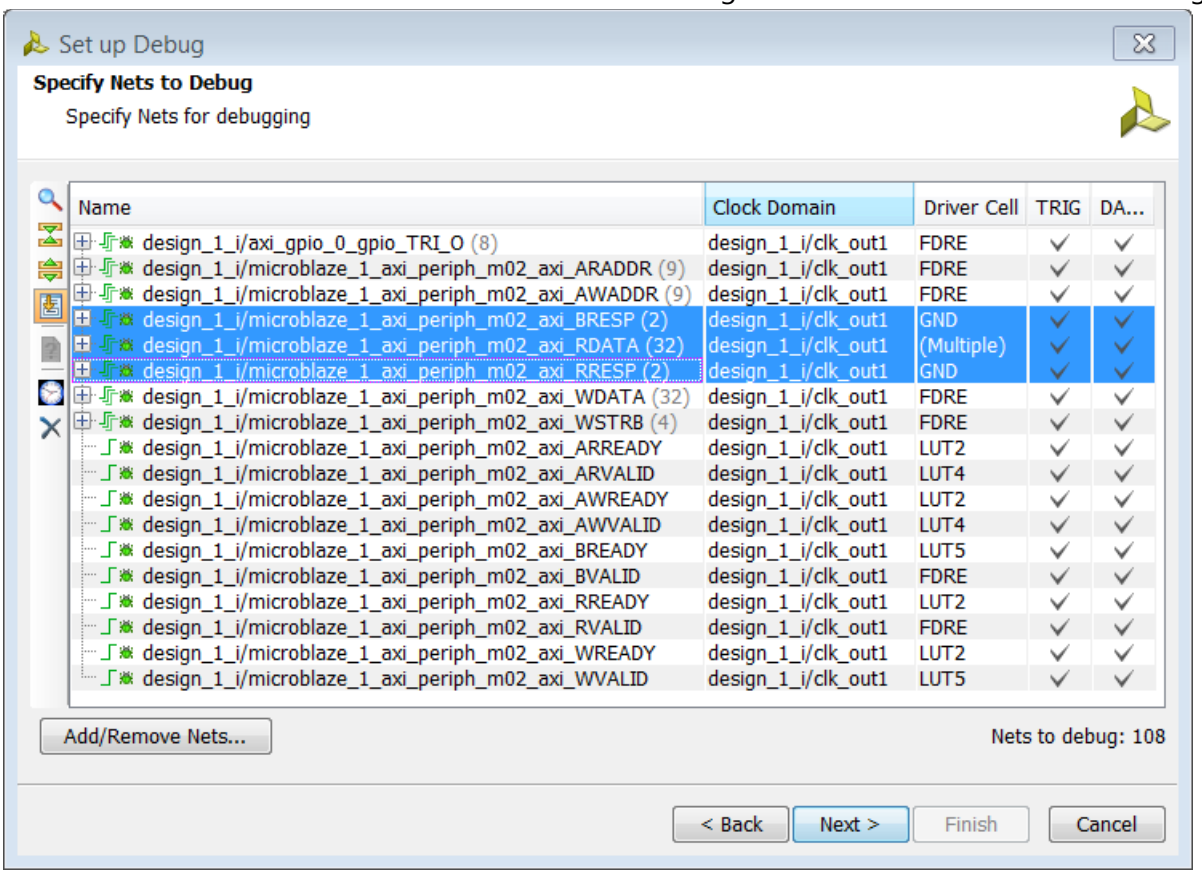


Figure 99: Specify Nets to Debug Dialog Box after Making the Necessary Clock Association

9. In the ILA (Integrated Logic Analyzer) General Options page, select the appropriate options for triggering and capturing data and then click **Next**.

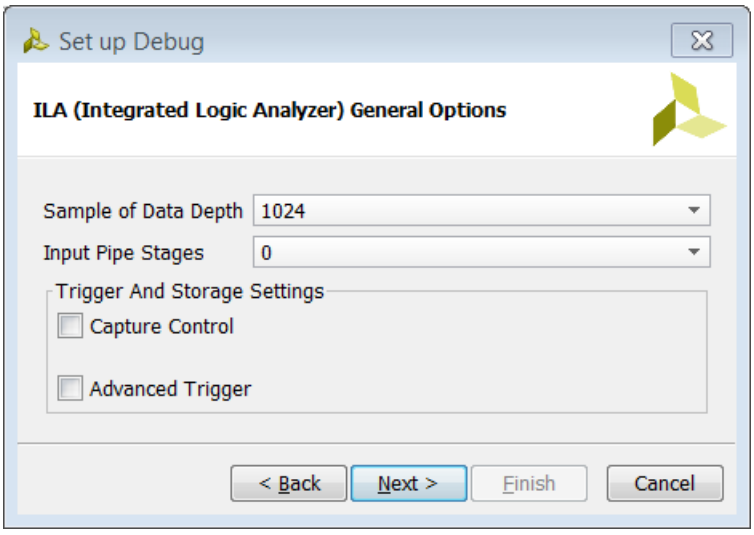


Figure 100: Specify the Trigger and Capture Modes in the ILA

The advanced triggering capabilities provide additional control over the triggering mechanism. Enabling advanced trigger mode enables a complete trigger state machine language that is configurable at runtime. There is 3-way branching per state and there are 16 states available as part of the state machine. Four counters and four programmable counters are available and viewable in the Analyzer as part of the advanced triggering.

In addition to the basic capture of data, capture control capabilities allows you to only capture the data at the conditions where it matters. This will ensure that unnecessary BRAM space is not wasted and provides a highly efficient solution.

In the Summary page, you should verify that all the information looks correct and then click Finish.

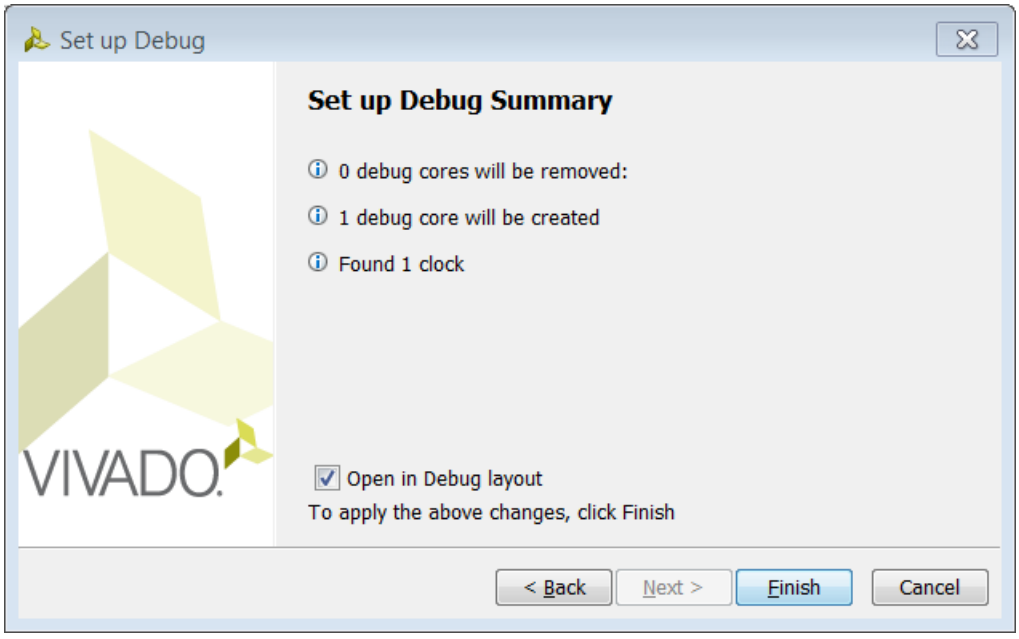


Figure 101: Set up Debug Summary

The Debug window looks as follows after the ILA core has been inserted. Note that all the buses (and single-bit nets) have been assigned to different probes. The probe information also shows how many signals are assigned to that particular probe. For example, in the following figure, probe0 has 32 signals (the 32 bits of the microblaze_1_axi_periph_m02_axi_WDATA) assigned to it.

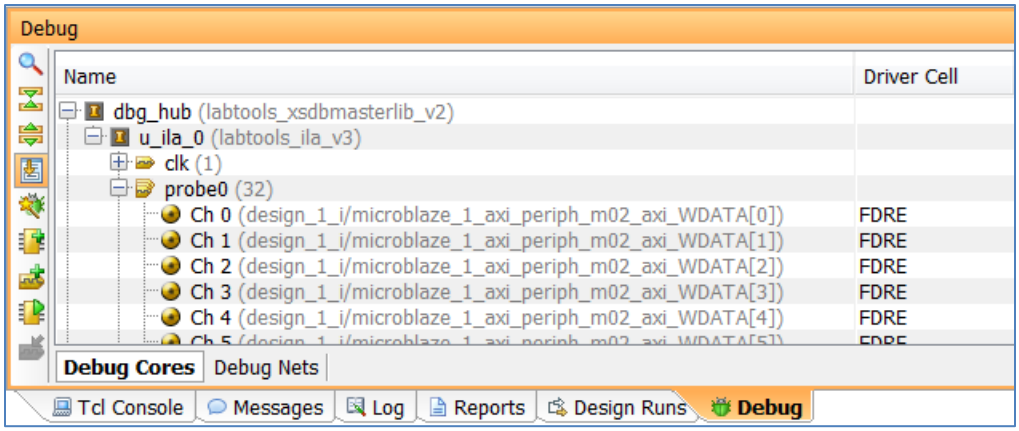


Figure 102: Debug Window after ILA Insertion

10. You are now ready to implement your design and generate a bitstream. You click on **Generate Bitstream** from the Program and Debug drop-down list in the Flow Navigator.

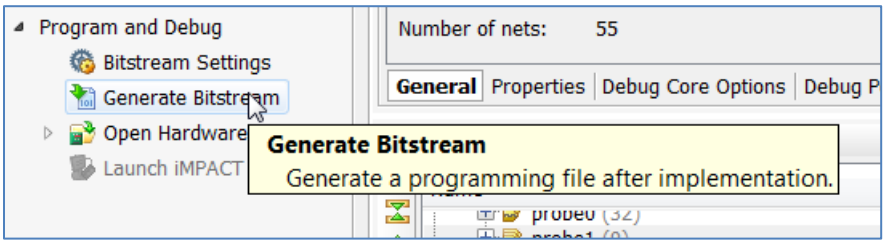


Figure 103: Generate Bitstream after ILA insertion

11. Since you have made changes to the netlist (by inserting an ILA core), a dialog box asking if the design should be saved prior to generating bitstream pops up.

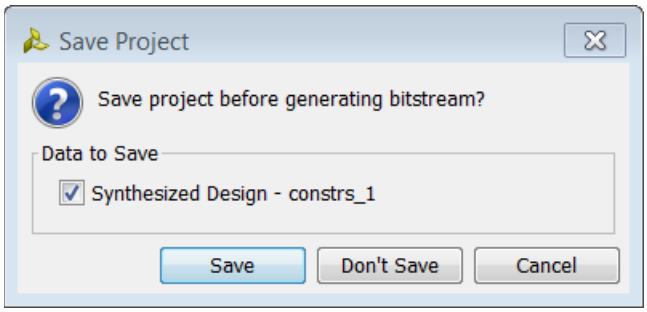


Figure 104: Save Modified Project after ILA Insertion

You can choose to save the design at this point, which will write the appropriate constraints in an active constraints file (if one exists) or will create a new constraints file. The constraints file contains all the commands to insert the ILA core in the synthesized netlist as shown below.

C:/temp/my_migration/my_migration.srcs/constrs_1/new/design_1_wrapper.xdc

```
1 create_debug_core u_ila_0 labtools_ila_v3
2 set_property ALL_PROBE_SAME_MU true [get_debug_cores u_ila_0]
3 set_property ALL_PROBE_SAME_MU_CNT 4 [get_debug_cores u_ila_0]
4 set_property C_ADV_TRIGGER true [get_debug_cores u_ila_0]
5 set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
6 set_property C_EN_STRG_QUAL true [get_debug_cores u_ila_0]
7 set_property C_INPUT_PIPE_STAGES 0 [get_debug_cores u_ila_0]
8 set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
9 set_property C_TRIGOUT_EN false [get_debug_cores u_ila_0]
10 set_property port_width 1 [get_debug_ports u_ila_0/clk]
11 connect_debug_port u_ila_0/clk [get_nets [list design_1_i/clk_out1]]
12 set_property port_width 32 [get_debug_ports u_ila_0/probe0]
```

Figure 105: Constraints File Showing XDC Commands for Core Insertion

The benefit of saving the project is that if the signals marked for debug remain the same in the original block design, then there is no need to insert the ILA core after synthesis manually as these constraints will take care of it. Therefore, subsequent iteration of design changes will not require a manual core insertion.

If you add more nets for debug (or unmark some nets from debug) then you will still need to open the synthesized netlist and make appropriate changes using the Set up Debug wizard.

If you do not chose to save the project after core insertion, none of the constraints show up in the constraints file and you will manually need to insert the ILA core in the synthesized netlist in subsequent iterations of the design.

Connecting to the Target Hardware

1. Once the bitstream has been generated, the Bitstream Generation Completed dialog box pops up. You then select **Open Hardware Manager** and click **OK**.

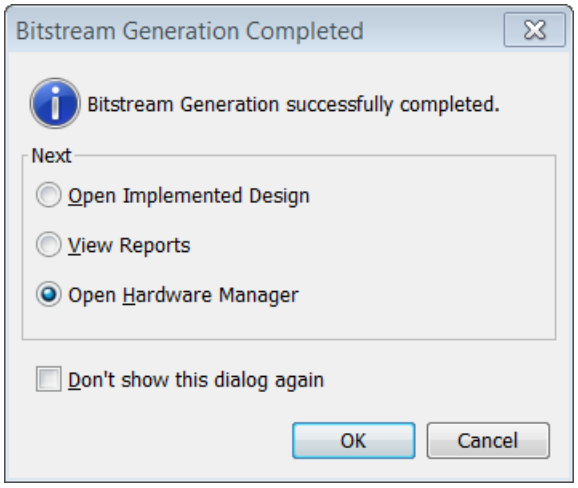


Figure 101: Open Hardware Manager after Bitstream Generation

2. Click on the **Open a new hardware target** link in the **Hardware Manager**.

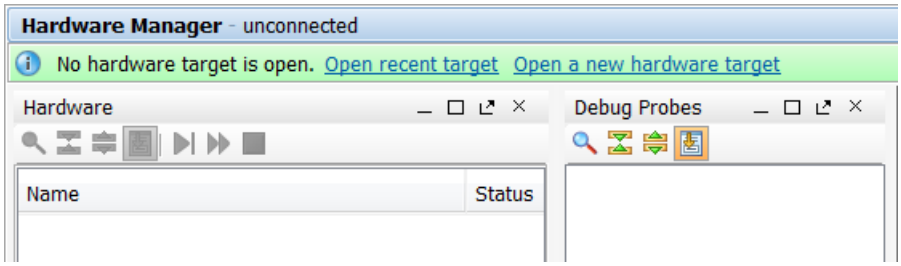


Figure 102: Connect to a Hardware Target

3. The **Hardware Manager** window opens. You now click Open a new hardware target. The **Open New Hardware Target** dialog box opens. Click **Next**.
4. Next, in the Hardware Sever Settings page you specify whether you want to connect to a **Local Server** or a **Remote Server**.

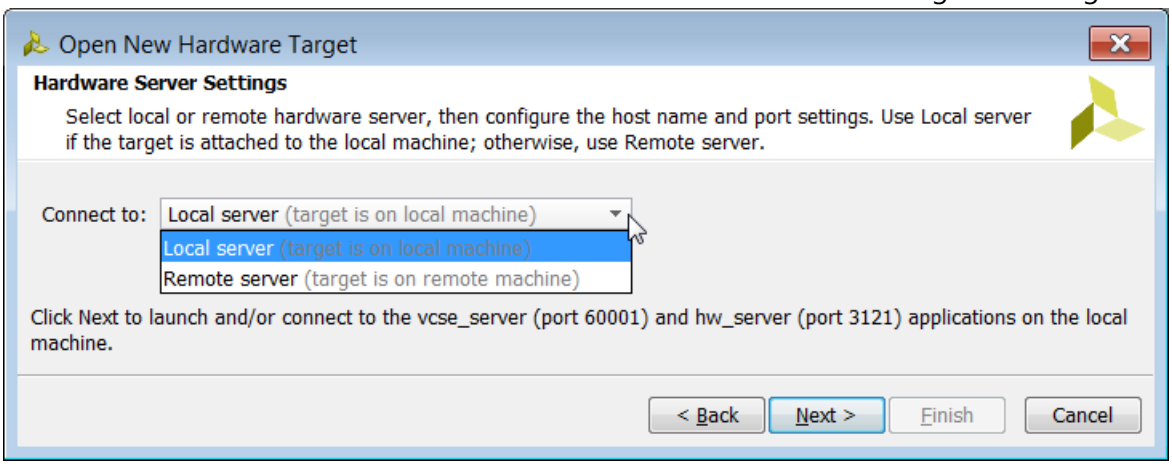


Figure103: CSE Server Name

***Note**: Depending on your connection speed, this may take about 10~15 seconds.*

5. If you are connecting to a Remote Server, you also need to specify the Host name and the Port. Click **Next**.

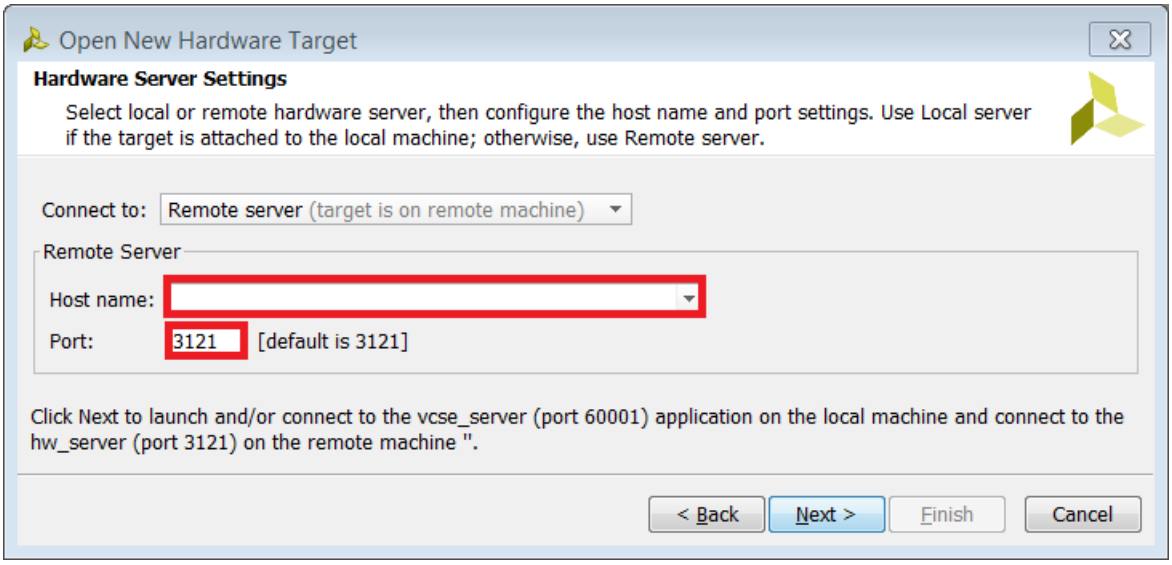


Figure 100: Specify Host Name and Port if Connecting to a Remote Server

6. If there is more than one target connected to the cse_server, you will see multiple entries in the **Select Hardware Target** dialog box. In this case, there is only one target as shown in the following figure. You then click **Next**.

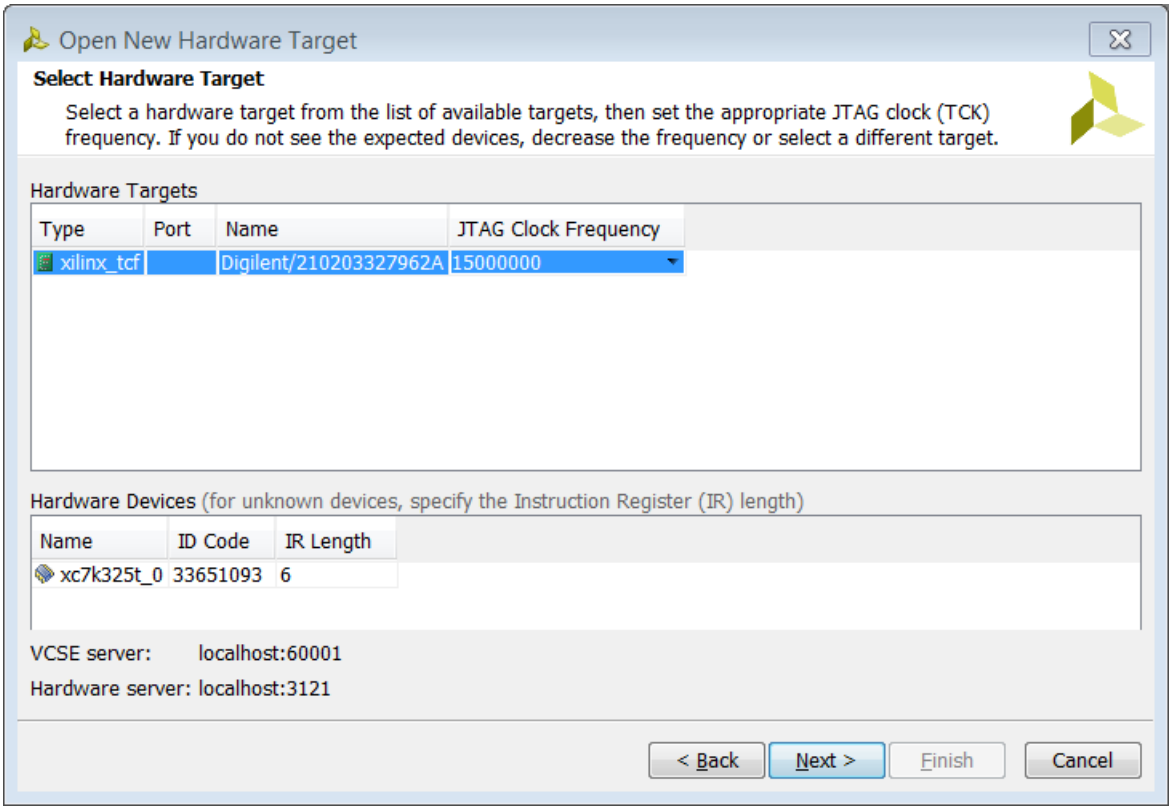


Figure104: Select Hardware Target

7. In the **Open Hardware Target Summary** page, click **Finish** as shown in the following figure.

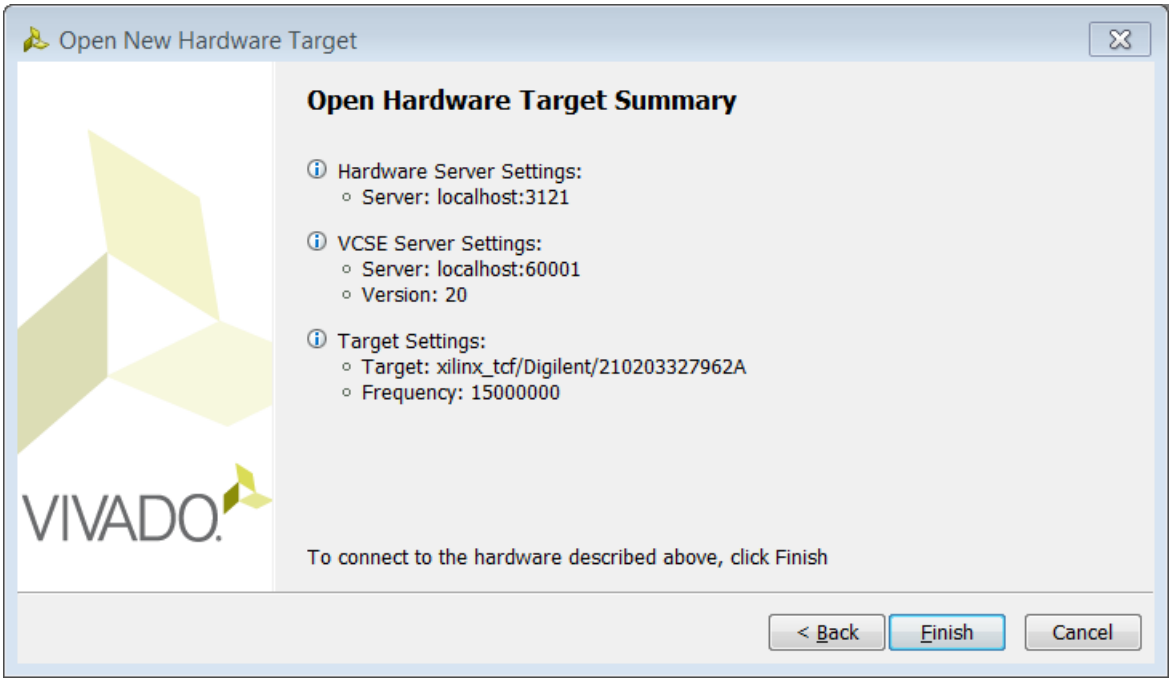


Figure106: Open Hardware Summary

8. Wait for the connection to the hardware to complete. The dialog in the following figure appears while the hardware is connecting.

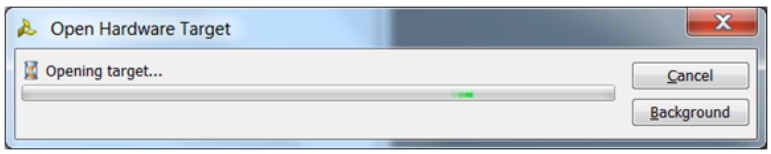


Figure107: Open Hardware Target

Once the connection to the hardware target is made, the dialog shown in the following figure appears.

> ***Note**: The **Hardware** tab in the **Debug** view shows the hardware target and XC7K325T device that was detected in the JTAG chain.*

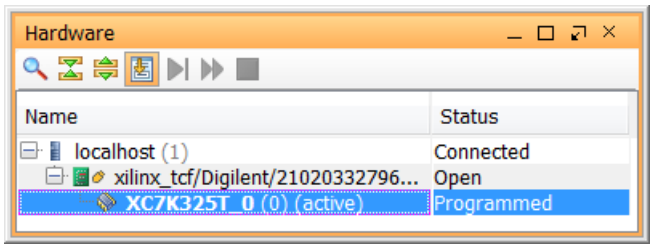


Figure108: Active Target Hardware

9. Next, program the XC7K325T device using the .bit bitstream file that was created previously by right-clicking on the **XC7K325T** device and selecting **Program Device** as shown in the following figure.

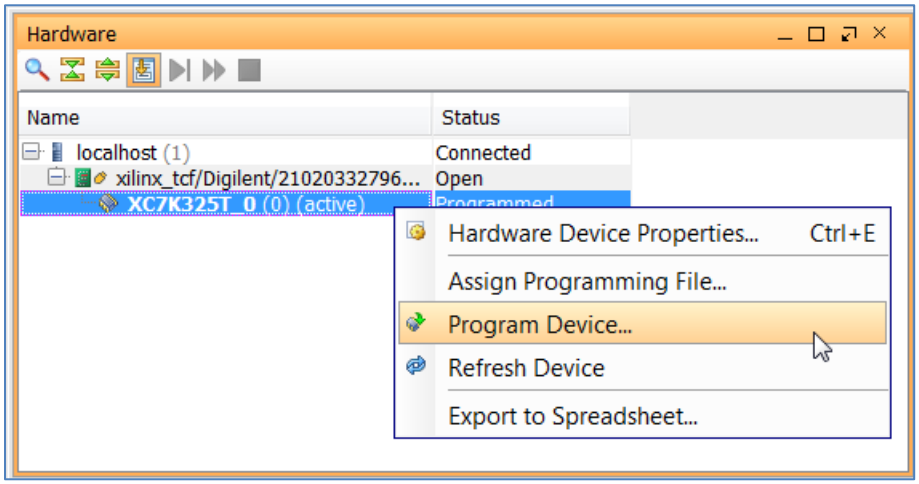


Figure109: Program Active Target Hardware

10. In the Program Device dialog box, you should verify that the BIT file is correct for the design that you are working on and click the **Program** button to program the device as shown in the following figure.

Figure110: Select Bitstream File to Download

***Note**: Wait for the program device operation to complete. This may take few minutes.*

Ensure that an ILA core was detected in the Hardware panel of the Debug view.

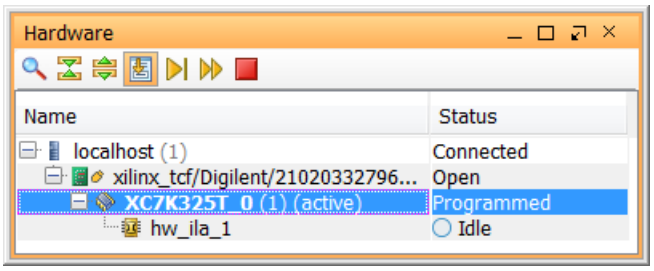


Figure111: ILA Core Detection

The Integrated Logic Analyzer window opens.

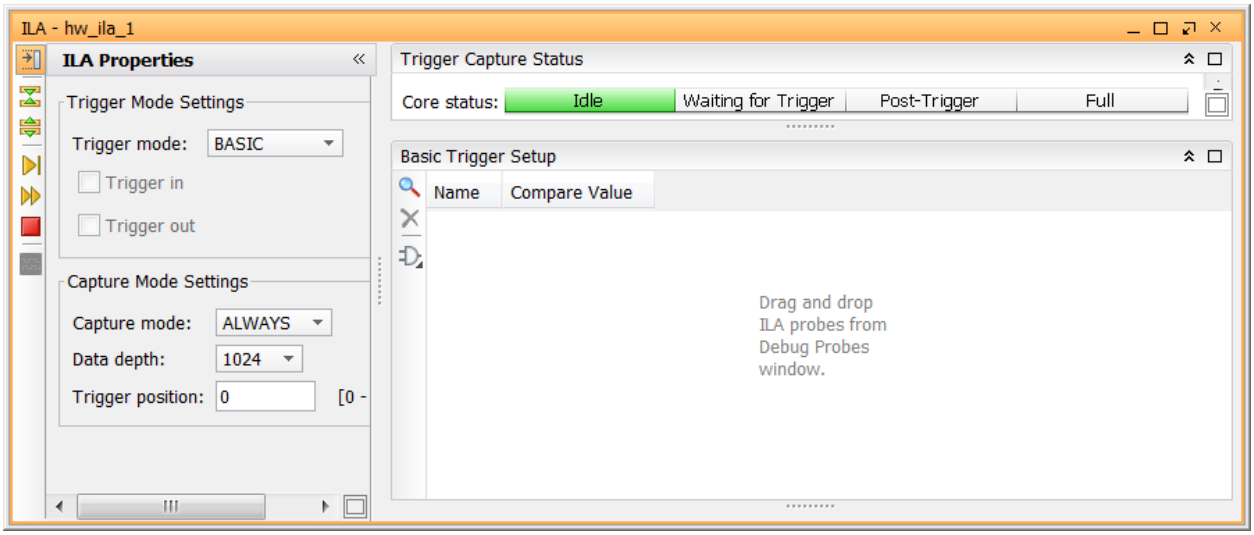


Figure112: The Vivado Integrated Logic Analyzer Window

For more information on Programming and Debug please refer to document *Vivado Design Suite Tutorial: Programming and Debugging* (**UG936**).

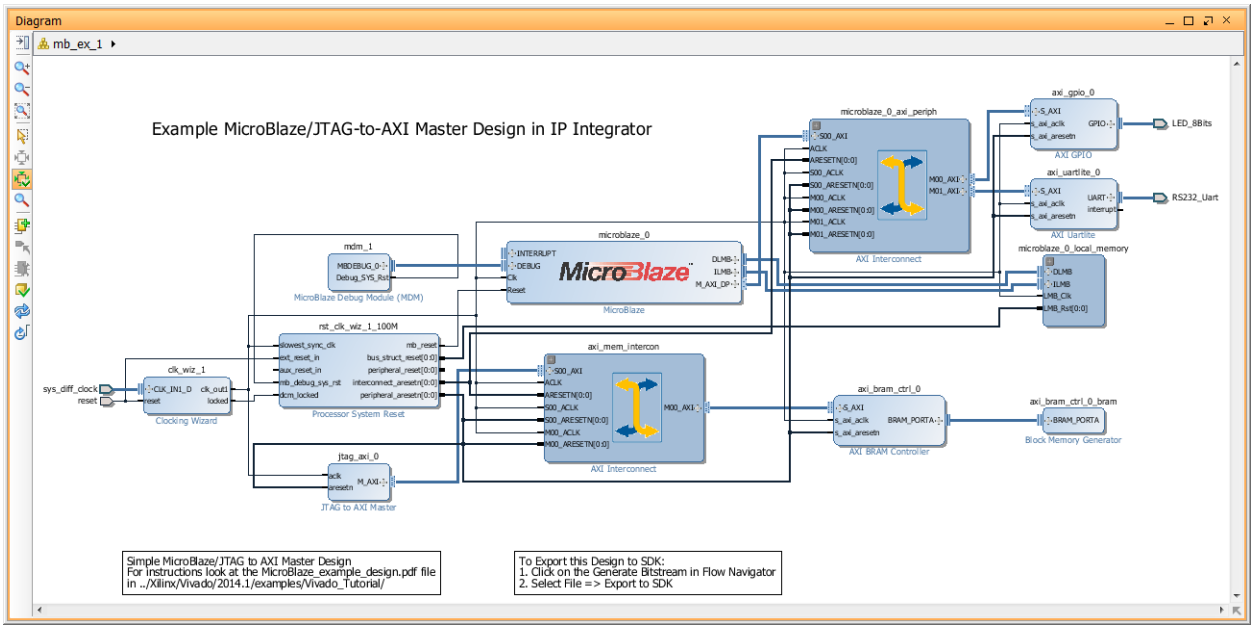
Chapter 5 Using Tcl Scripts to Create IP Integrator Designs within Projects

Overview

Typically you will create a new design in a project-based flow in the Vivado IDE GUI environment. Once the initial design has been put together, you may want to re-create the design using a scripted flow in the GUI or in batch mode. This chapter guides you through creating a scripted flow for block designs.

Create a Design in the Vivado IDE GUI

Create a project and a new block design in the Vivado IDE GUI as mentioned in Chapter 2 of this user guide. Once the block design is complete, your canvas will contain a design like the example in the following figure.



With the block design open, in the Tcl console type the following command:

```
write_bd_tcl <path to file/filename>
```

This creates a Tcl file that can be sourced to re-create the block design. This Tcl file has information embedded in it about the version of the Vivado tools that it was created in and as such this file should not be used across different releases of the Vivado Design Suite. The Tcl file also contains information about all the IP present in the block design, their configuration and the connectivity.

Save the Vivado Project Information in a Tcl File

The overall project settings can be saved by using the write_project_tcl command.

```
write_project_tcl <path to file/filename>
```

For a Vivado Project that consists of a block diagram, the Tcl file generated from write_project_tcl command may look something as follows:

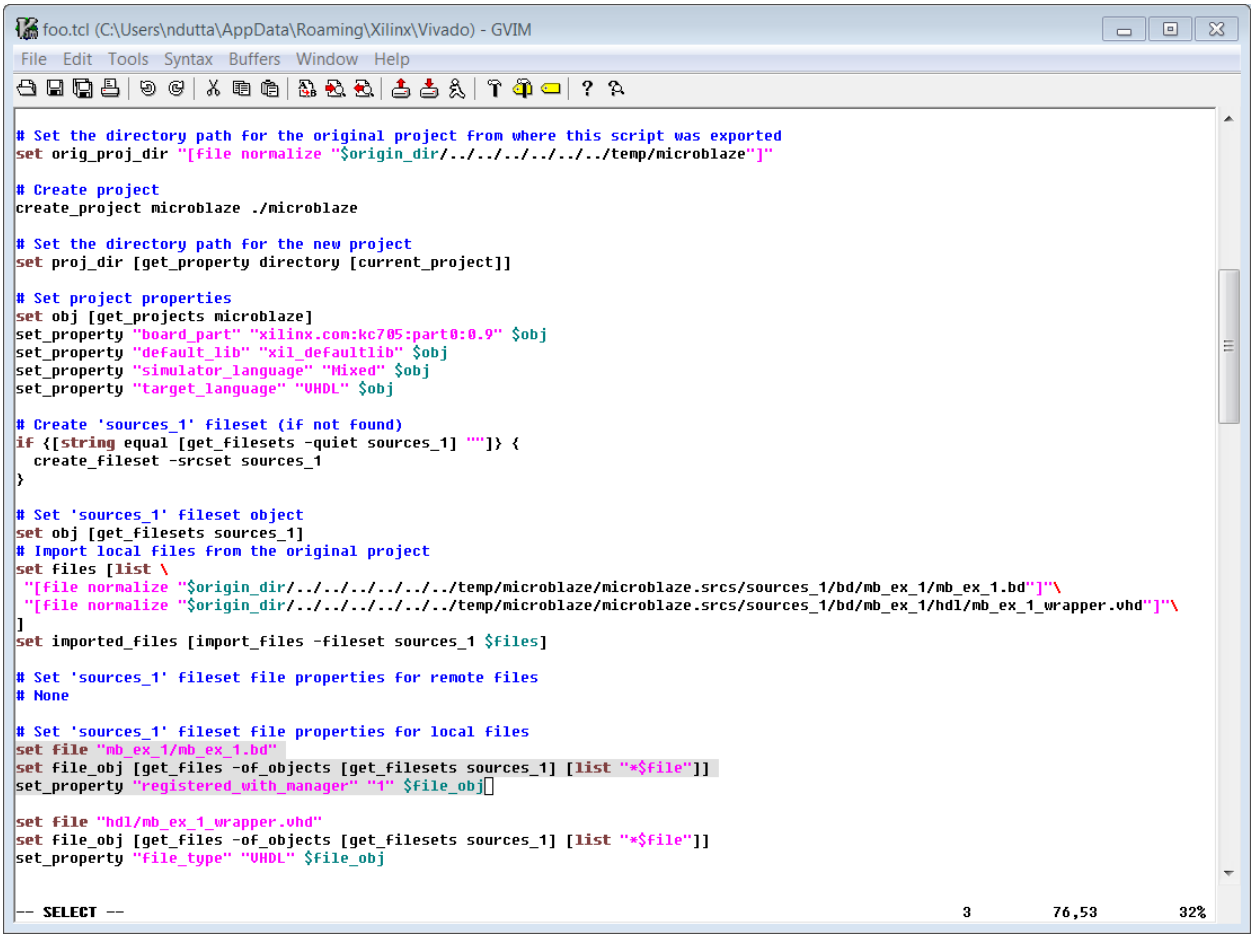


Figure 101: Code Snippet from the Tcl File Generated by using the write_project_tcl Command

In the above Tcl file, the block design file .bd is read explicitly as shown by the highlighted code. If you choose not to re-create the block design and just read the already created block design, then the above lines of code do not need to be modified. However, there are cases in which you may want to re-create the block design. For this purpose, the Tcl file generated by using the `write_project_tcl` command can be modified as follows:

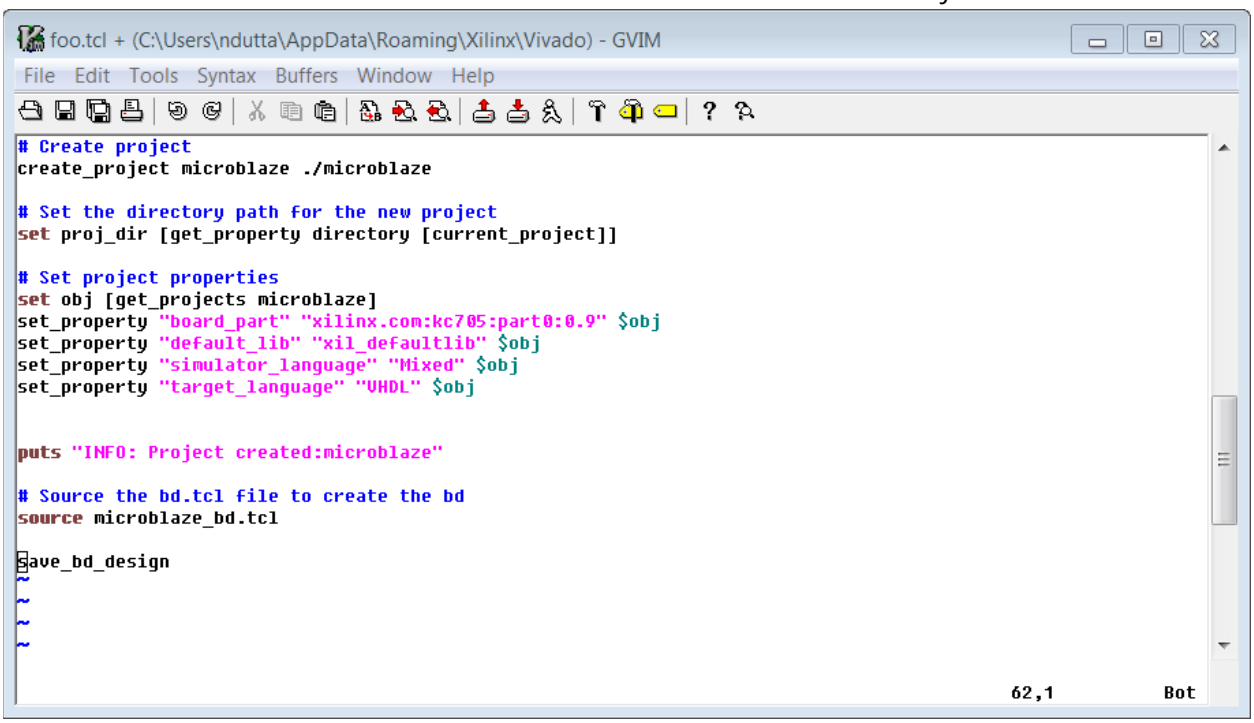


Figure 102: Code Snippet from the Tcl File to Recreate the Block Design using the Output file

As can be seen from the above code snippet, the Tcl file from the write_project_tcl file has been modified to source the output file that was created using write_bd_tcl command. This will re-create the block design every time the Tcl file is run.

Chapter 6 Using Non-Project Mode in IP Integrator

Overview

Non-Project Mode is for users who want to manage their own design data or track the design state. In this flow, Vivado® tools read the various source files and implement the design through the entire flow in-memory. At any stage of the implementation process, you can generate a variety of reports based on your script. When running in Non-Project Mode, it is also important to note that this mode does not enable project-based features such as source file and run management, cross-probing back to source files, and design state reporting. Essentially, each time a source file is updated on the disk, you must know about it and reload the design. There are no default reports or intermediate files created within the Non-Project Mode. You need to have your script control the creation of reports with Tcl commands.

Creating a Flow in Non-Project Mode

Vivado tools can be invoked in Tcl mode instead of the usual Project Mode by issuing following commands in the Tcl Console. The recommended approach in this mode is to create a Tcl script and source it from the Vivado prompt:

```
Vivado% vivado -mode tcl
```

In a non-project mode, you have to create a project in memory and set your project options as shown below:

```
create_project -in_memory -part xc7k325tffg900-2
set_property target_language VHDL [current_project]
set_property board_part xilinx.com:kc705:part0:0.9 [current_project]
set_property default_lib work [current_project]
```

Once the project has been created then the source file for the block design can be added to the project. This can be done in two different ways. First, assuming that there is an existing block design that has been created in a Project Mode with the entire directory structure of the block design intact, you can add the block design using the read_bd tcl command as follows:

```
Vivado% read_bd <absolute path to the bd file>
```

CAUTION! Your need to have the project settings (board, part and user repository) match with the project settings of the original project in which the bd was created. Otherwise, the IP in the block design will be locked.

Once the block design is added successfully, you need to add your top-level RTL files and any top-level XDC constraints.

```
Vivado% read_verilog <top-level>.v
Vivado% read_xdc <top-level>.xdc
```

You can also create top-level HDL wrapper file using the command below since a bd source cannot be synthesized directly.

```
Vivado% make_wrapper -files [<path to bd>/<bd instance name>.bd] –top
add_files -norecurse <path to bd>/< bd instance name >_wrapper.vhd
update_compile_order -fileset sources_1
```

This creates a top-level HDL file and adds it to the source list.

For a MicroBlaze-based design, you should populate the I-LMB with either a Bootloop or your own executable in ELF format. You then needs to add the ELF and associate it with the MicroBlaze instance. The following steps will do this.

```
vivado% add_files <ELF file Targeted to BRAM with .elf extension>
vivado% set_property MEMDATA.ADDR_MAP_CELLS {<bd instance name>/microblaze_0}
[get_files <BRAM Targeted ELF File>]
```

If the design has multiple levels of hierarchy, you need to ensure that the correct hierarchy is provided. After this, you need to go through the usual synthesis, Place and Route steps to get the design implemented. One aspect that needs to be kept in mind is that for the synthesis (synth_design) step, you need to provide the target part as the default target part, which may not be the same as the desired one.

```
synth_design -top <path to top level wrapper file> -part <part>
opt_design
place_design
route_design   write_bitstream top
```

Refer to the document Vivado Design Suite User Guide: Design Flows Overview (**UG892**) for more details about working in a Non-Project Mode flow.

If you want to export the hardware system to SDK, you need to use the following command:

```
vivado% export_hardware [get_files <Path to bd file>] -dir <Absolute Export
Directory Path>
```

Chapter 7 Migrating IP Integrator Designs from One Release to Another

Overview

As you upgrade your Vivado® Design Suite to the latest release, you will need to upgrade the block designs created in IP integrator as well. The IP version numbers change from one release to another. When IP integrator detects that the IP contained within a block design are older versions of the IP, it "locks" those IP in the block design. If the intention is to keep older version of the block design (and the IP contained within it), then you should not do any operations such as modifying the block design on the canvas, validating it and/or resetting output products and re-generating output products. In this case, the expectation is that you have all the design data from the previous release intact. If that is the case, then you can use the block design from the previous release "as is" by synthesizing and implementing the design.

The recommended practice is to upgrade the block design with the latest IP versions, make any necessary design changes, validate design and generate target.

Upgrading can be done in two ways:

1. Using the Vivado IDE GUI in the Project Mode Flow
2. Using a Tcl script in the Non-Project Mode Flow

Both methods are described in this Chapter.

Using the Project Mode Flow

1. Launch the latest version of the Vivado Design Suite.
2. From the Vivado IDE main page, click on **Open Project** and navigate to the design that was created from a previous version of Vivado tools.
3. The **Older Project Version** pop-up opens. **Automatically upgrade for the current version** is selected by default. Although you can upgrade the design from a previous version by selecting the **Automatically upgrade for the current version**, it is highly recommended that you save your project with a different name before upgrading. To do this, select **Open project in read-only mode** and click **OK**.

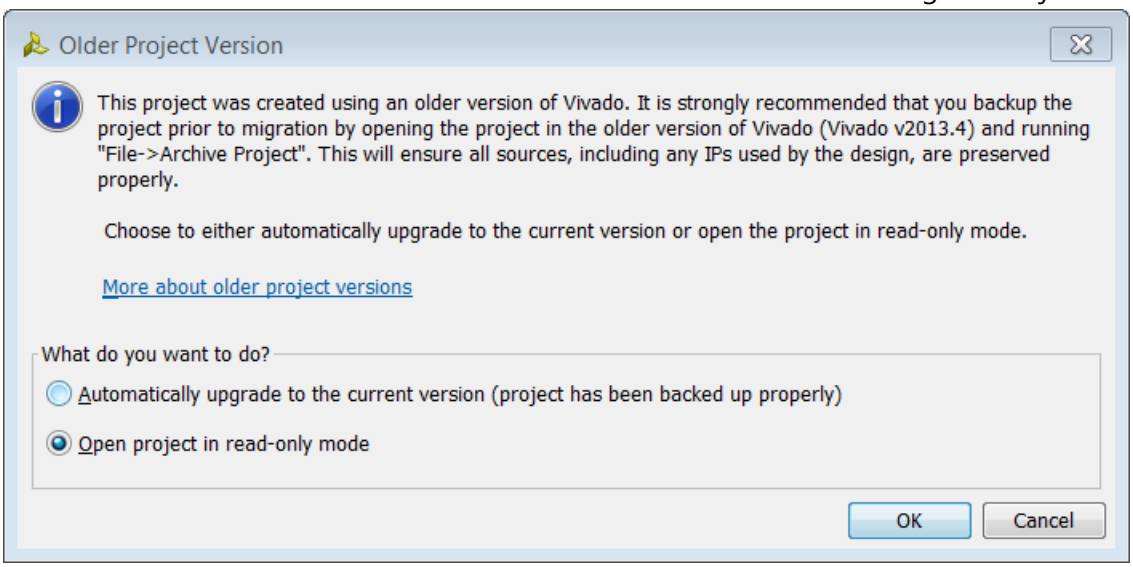


Figure113: Open Project in Read-Only Mode

4. The Project is Read-Only dialog box pops-up. Select **Save Project...**

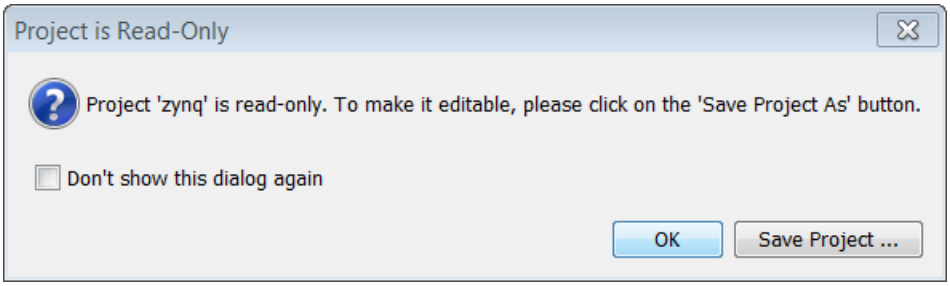


Figure 114: Save Project

5. When the Save Project As dialog box opens, type in the name of the project and click **OK**.

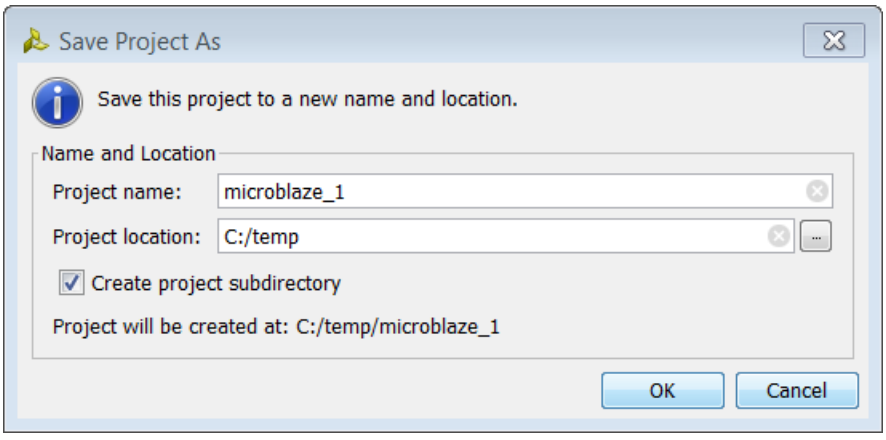


Figure 115: Specify Name of Project

6. From the menu select **Tools > Report > Report IP Status**.

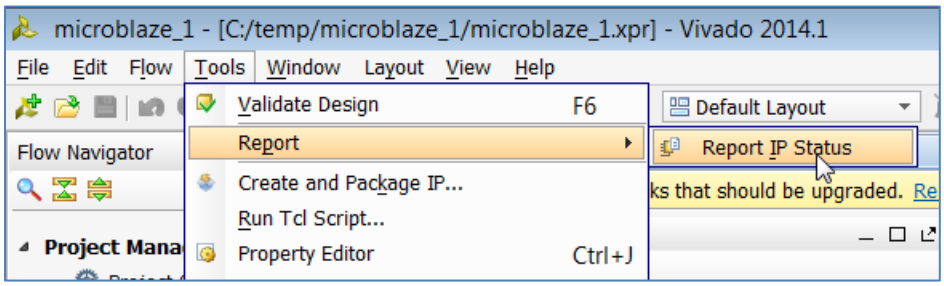


Figure 116: Report IP Status

7. If any of the IP in the design has gone through a major version change, then the following message will pop-up on the screen. Click **OK**.

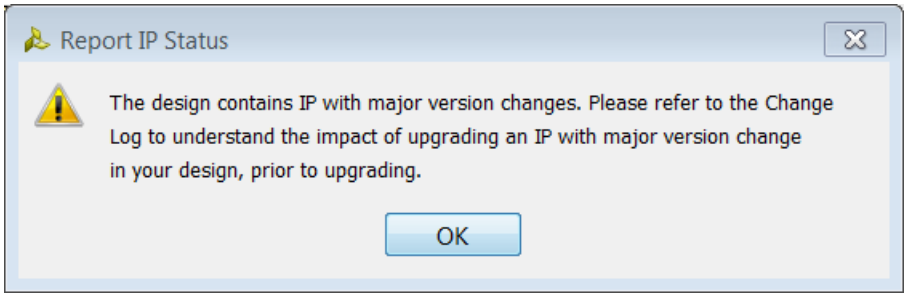


Figure 118: Report IP Status Dialog Box Informing the User of IP Major Version Change

In the IP Status window, look at the different columns and familiarize yourself with the IP Status report. Expand the block design by clicking on the + sign and look at the changes that the IP cores in the block design may have gone through. Also realize that you cannot individually select some IP of a block design for upgrade and let the others remain in their current versions.

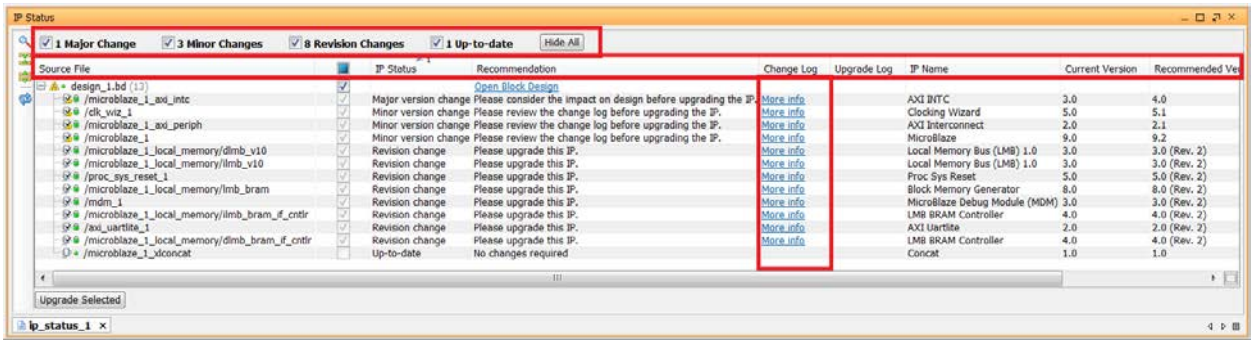


Figure 119: IP Status Window

The very top of the IP Status window shows the summary of the design. It reports how many changes are needed to upgrade the design to the current version. The changes reported are Major Changes, Minor Changes, Revision Changes and Other Changes. These changes are reported in the IP Status column as well.

Major Changes: The IP has gone through a major version change, for example from Version 2.0 to 3.0. This type of change is not automatically selected for upgrade. To select this for upgrade, uncheck the Upgrade column for the block design and then re-check it.

Minor Changes: The IP has undergone a minor version change, for example, from version 3.0 to 3.1.

Revision Changes: A revision change has been made to the IP. For example the IP's current version is 5.0 and the upgraded version is 5.0 (Rev. 1)

You can click on the **More info...** link in the Change Log column to see a description of the change.

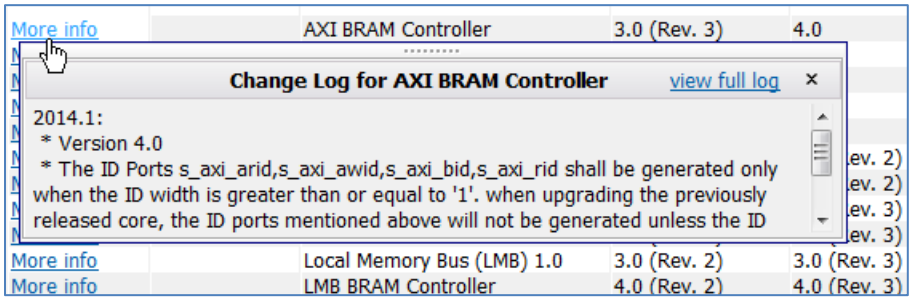


Figure120: Inspect the Change Log by Clicking on More Info link in the Change Log Column

The Recommendation column also suggests that you need to understand what the changes are before selecting the IP for upgrade.

8. Once you understand the changes and the impact of them on your design, you should click on **Upgrade Selected**.

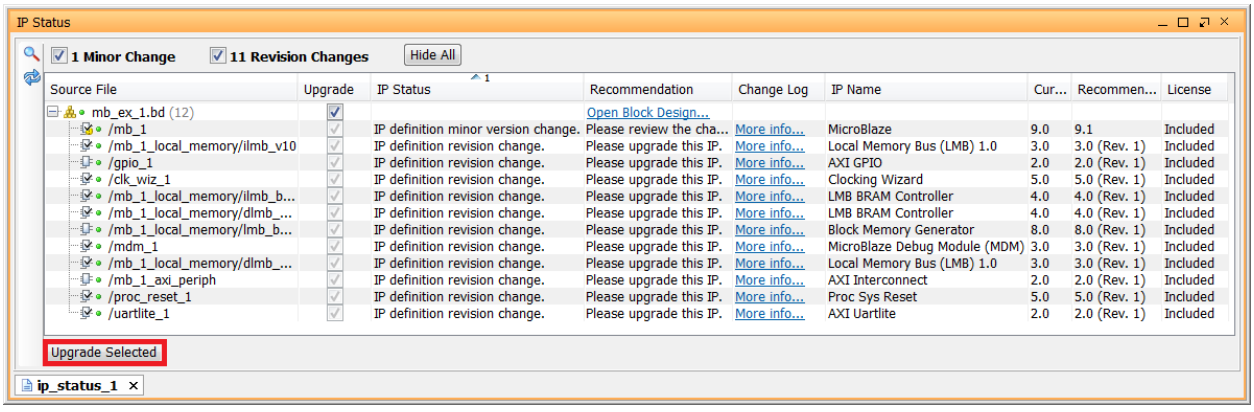


Figure 121: Upgrade IP

9. When the upgrade process is complete, you should review any critical warnings and other messages that may be flagged as a part of the upgrade.

   If there are multiple diagrams in the design, the IP Status window will show the status of IP in all the diagrams as shown below.

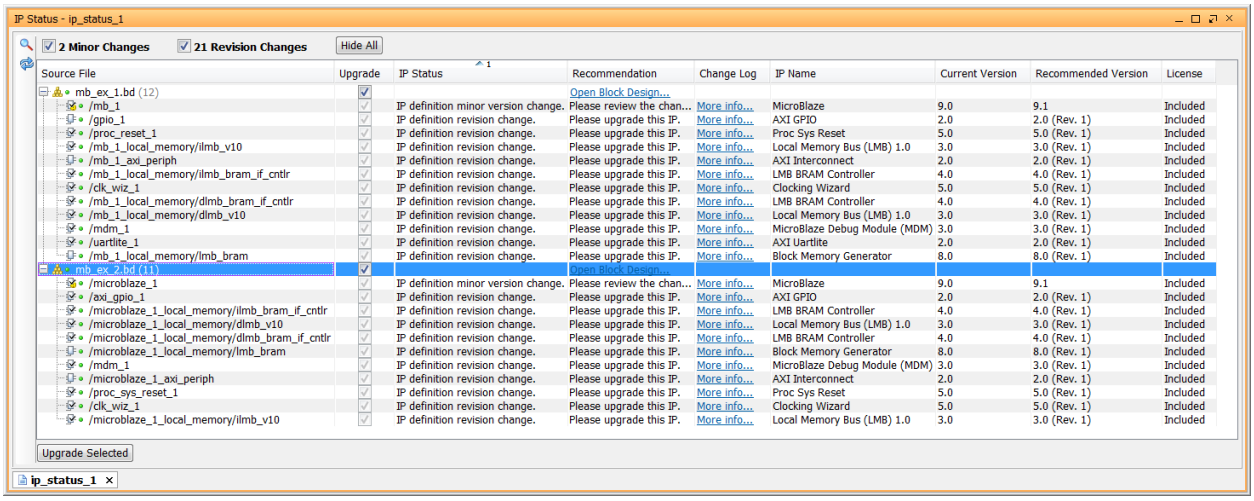


Figure 123: IP Status Window for Multiple Diagrams

When you click on **Upgrade Selected**, all the block diagrams are updated in the design (if the block diagrams are selected for upgrade).

Running Design Rule Checks

From the toolbar, click on **Validate Design**.

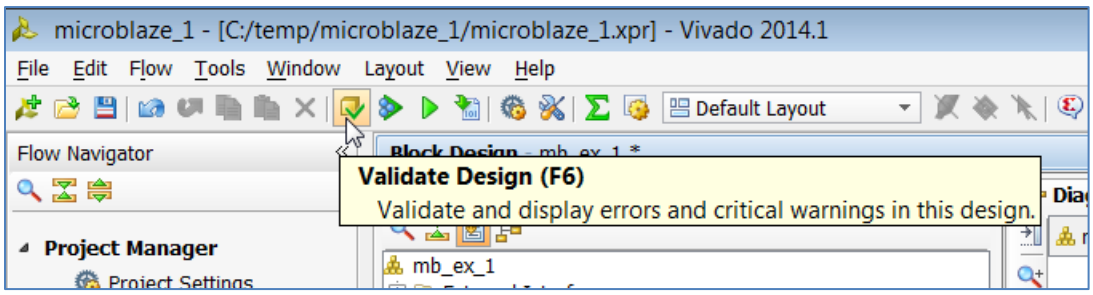


Figure 124: Run Design Rule Checks

You can also do this by clicking the validate design icon in the block design toolbar.

If there are no design rule violations, you will see a pop-up as shown in the following figure.

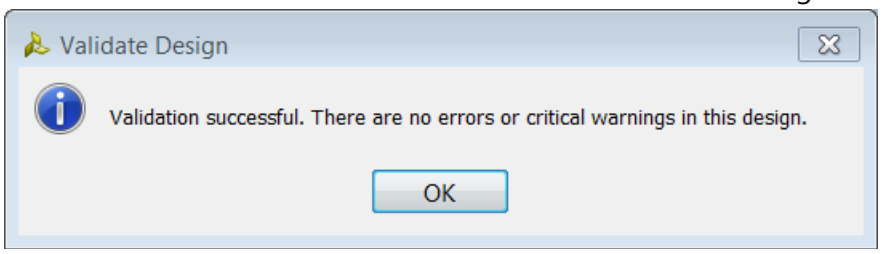


Figure 125: The Validate Design Dialog Box

Regenerating Output Products

1. In the Sources pane in Vivado, right-click on block diagram and select **Generate Output Products**.

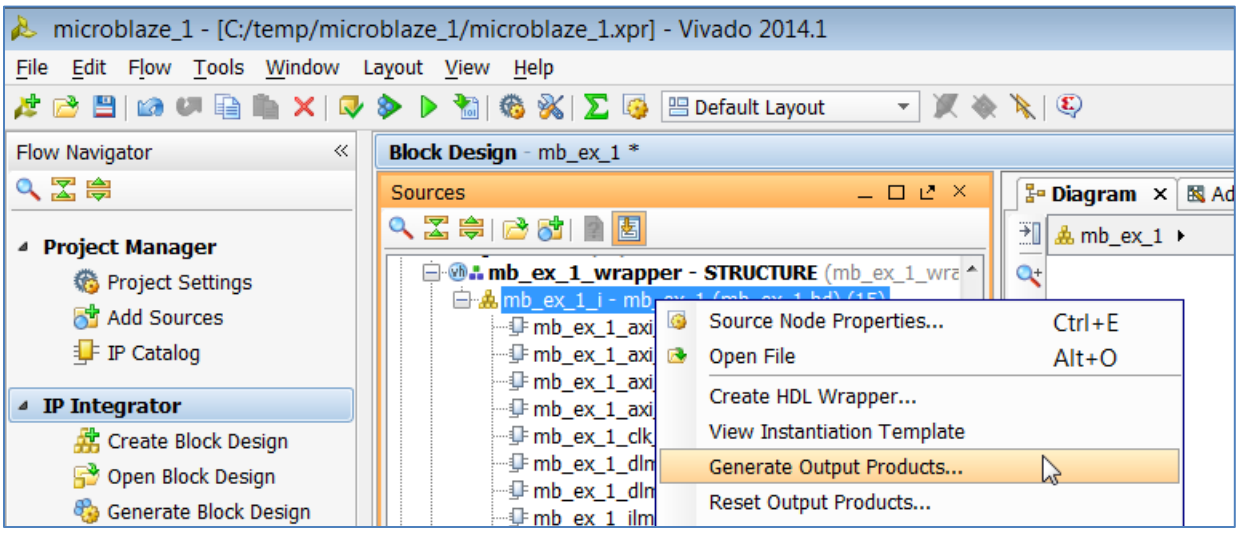


Figure 126: Generate Output Products from the Sources Window

Alternately, you can also click on the **Generate Block Design** in the Flow Navigator under IP integrator drop-down list.

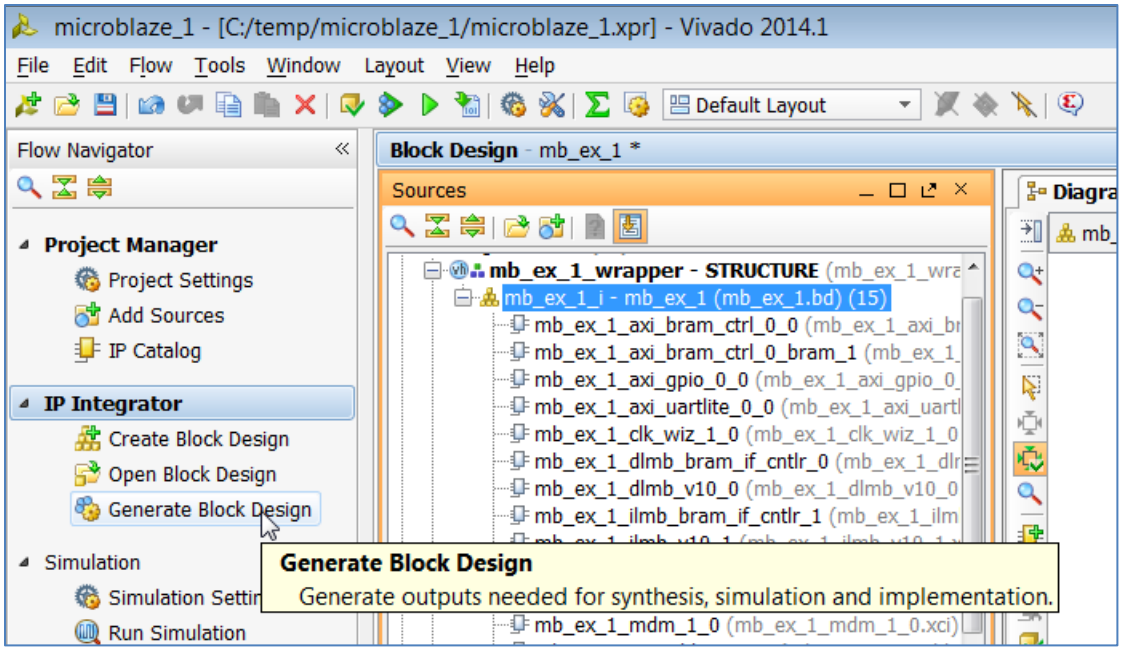


Figure 127: Generate Output Products by Clicking on Generate Block Design

2. In the Generate Output Products dialog box, click **Generate**.

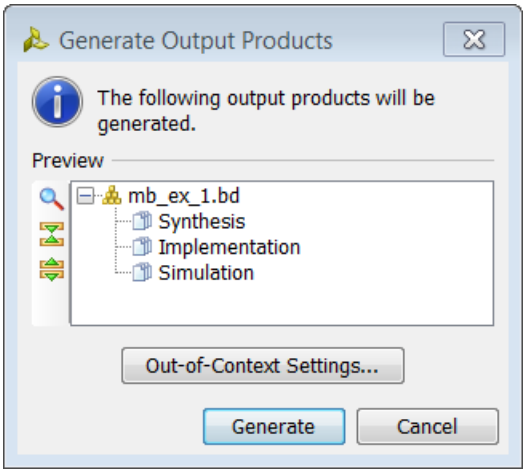


Figure 128: Generate Output Products through the Manage Output Products dialog box

Create/Change the HDL Wrapper

If you previously created an HDL wrapper in the previous version of the design, you may want to re-create it to reconcile any design changes. If you had chosen the option to let the Vivado tools create and manage the top-level wrapper for you, then the wrapper file will be updated as a part of generating the block design or generating output products as defined in the previous section. If you modified the HDL wrapper manually, then you will need to manually make any updates that may be necessary in the HDL wrapper.

1. In the Sources pane in Vivado right-click on the block diagram and select **Create HDL Wrapper**.

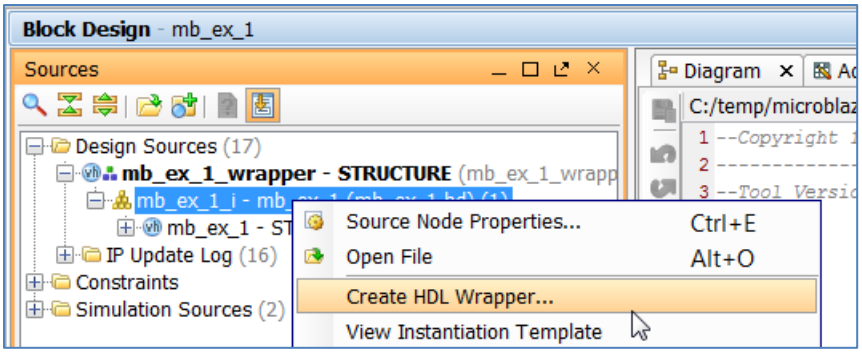


Figure 129: Create HDL Wrapper

2. The Create HDL Wrapper dialog box opens. You have two choices to make at this point. You can create a wrapper file that can be edited or else you can let the Vivado tools manage the wrapper file for you. Click **OK**.

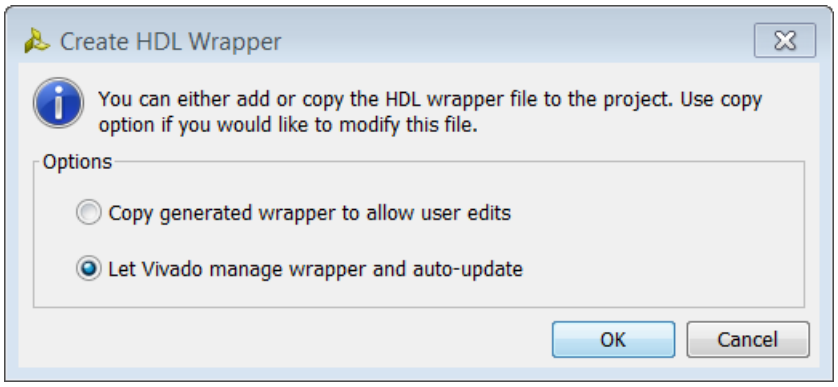


Figure 130: Create HDL Wrapper Dialog Box

3. Continue through implementation.

Upgrading a Block Design in Non-Project Mode

You can open an existing project from a previous release using the Non-Project Mode flow and upgrade the design to the current release of Vivado. You can use the following script as a guideline to upgrade the IP in the block diagram.

```
# Open an existing project from a previous Vivado release
open_project <path_to_project>/project_name.xpr
update_compile_order –fileset sim_1
# Open the block diagram
read_bd {<path_to_bd>/bd_name.bd}
# Make the block diagram current
current_bd_design bd_name.bd
# Upgrade IP
upgrade_bd_cells [get_bd_cells –hierarchical * ]
# Reset output products
reset_target {synthesis simulation implementation} [get_files
<path_to_project>/project_name.srcs/sources_1/bd/bd_name/bd_name.bd]

# Generate output products
generate_target {synthesis simulation implementation} [get_files
<path_to_project>/project_name/project_name.srcs/sources_1/bd/bd_name/bd_name.bd
]
# Create HDL Wrapper (if needed)
make_wrapper -files [get_files
<path_to_project>/project_name/project_name.srcs/sources_1/bd/bd_name/bd_name.bd
] –top
# Overwrite any existing HDL wrapper from before
import_files -force -norecurse
<path_to_project>/project_name/project_name.srcs/sources_1/bd/bd_name/hdl/bd_nam
e_wrapper.v
update_compile_order -fileset sources_1
# Continue through implementation
…
```

Using the Output Tcl File from a Previous Version of the Vivado Design Suite

Using the output file from the write_bd_tcl command between different versions of the Vivado tools is not recommended. These files should be created and used within a particular version of the Vivado Design Suite.

Chapter 8 Revision Control for IP Integrator Designs

Overview

This chapter provides recommendations for using version control systems with IP integrator- based block designs in both Project as well as Non-Project Mode. The IP integrator tool in the Vivado® IDE is a powerful tool that enables the creation of complex IP subsystem designs. As designs get more complex, the challenge is to keep a track of different versions of the design to facilitate project management and collaboration in a team-design environment.

While a project may include multiple design sources and configuration files, only a subset of these files require revision control to recreate a project and reproduce implementation results. Some of these files applicable to block designs are:

- IP-XACT core files (.xci, .mem, .coe)
- Block diagram files (.bd)
- Embedded subsystems and files (.elf, .bmm)
- Xilinx Design constraints files (.xdc)
- Configuration files, including Vivado Simulator and Vivado Integrated Logic Analyzer configuration files (.wcfg)
- RTL file (wrappers for block designs if managed by user: .vhd, .v)

The Vivado Design Suite does not support any particular revision control system. Rather, it is designed to work with any revision control system. In order to make Vivado designs suitable for revision control, the Vivado Design Suite provides the following features:

- Updates the timestamps only when the files are modified. Accordingly, opening a project doesn't change the timestamp on it.
- Supports ASCII-based project files.
- Supports extensive Tcl scripting capabilities.

Design Files Needed to be Checked In for Revision Control

A block design consists of several IP constructed graphically in a GUI environment. A block design directory structure in a Project Mode flow looks as shown in the following figure:

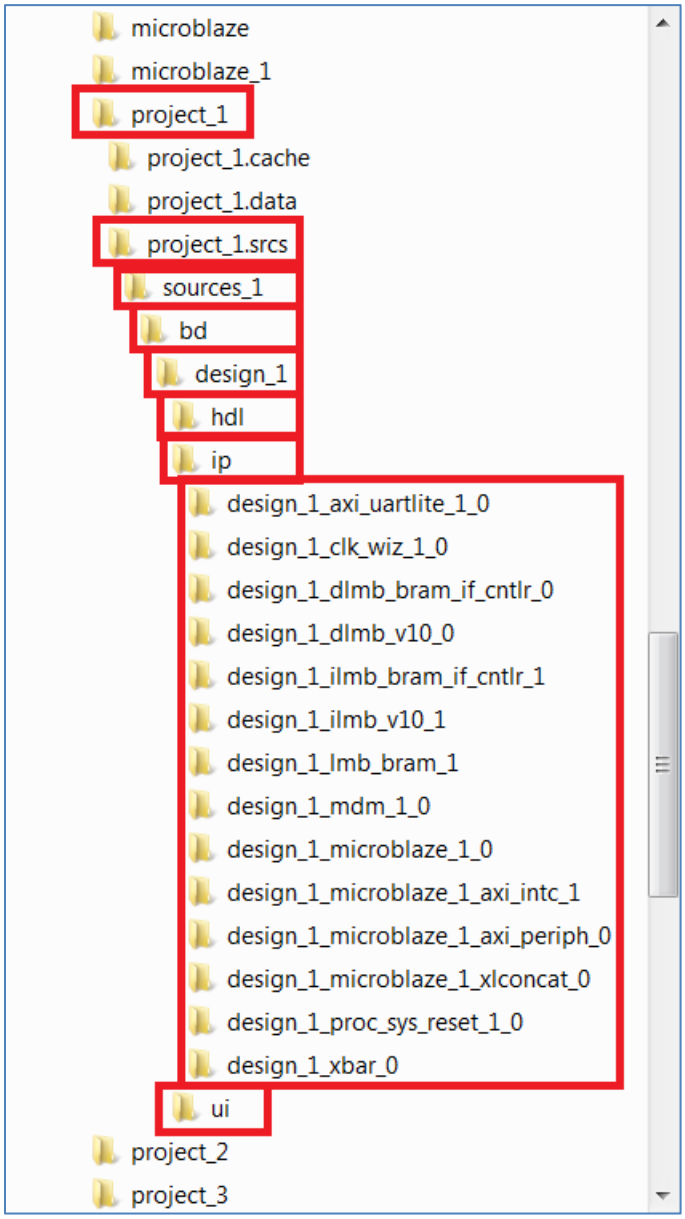


Figure 132: Vivado Project Directory Structure on Disk

In the figure above, the folders mean the following:

- project_1 – Vivado Project folder (project_1 is the name given to the project)

- project_1.srcs – Sources folder containing project specific files
- sources_1 – Folder containing all source file including the bd
- bd – contains block design specific data. May have multiple sub-directories corresponding to each block design
- design_1 – folder containing data for the block design called design_1. If there are multiple block designs in the project, then multiple folders will be present here.
- hdl – folder containing the top-level HDL file and the wrapper file.
- ip – contains subfolders pertaining each of the IP in the block design
- ui – folder containing GUI (IP integrator canvas) data for the block design

The recommendation is to put all the files contained in the bd folder including the entire directory structure under revision control.

Creating a Block Design for Use in a Different Project

IP integrator provides the capability to re-use a block design created in a different project to be imported into other projects for re-use purpose. In order to do this, a block design must be created in a project-based flow. You also need to make sure that the design doesn't flag any DRC violations and synthesizes (and possibly implements) without any issues. Once you are satisfied with the block design, you can delete everything except for the **bd** directory and all the sub-directories included beneath it from the Vivado Project. This way all the block design data including the data for all the IP contained within the block design can be imported into a different Vivado project.

Importing an Existing Block Design into a Different Vivado IDE Project

Assuming that a block design was created using a project-based flow, and all the directory structure including and within the bd folder is available, the block design can be opened in a different Vivado project. The only limitation in such a scenario is that the new project settings in which the existing block design is being imported must be the same as the original project in which the block design was created. If the target device of the projects (this includes devices even within the same family) are different then the IP will get locked and the design will have to be re-generated. In such a scenario there is a likelihood that the behavior of the design may not be the same as the original block design.

1. To import an existing block design, highlight Design Sources in the Sources window, right-click and select **Add Sources**.

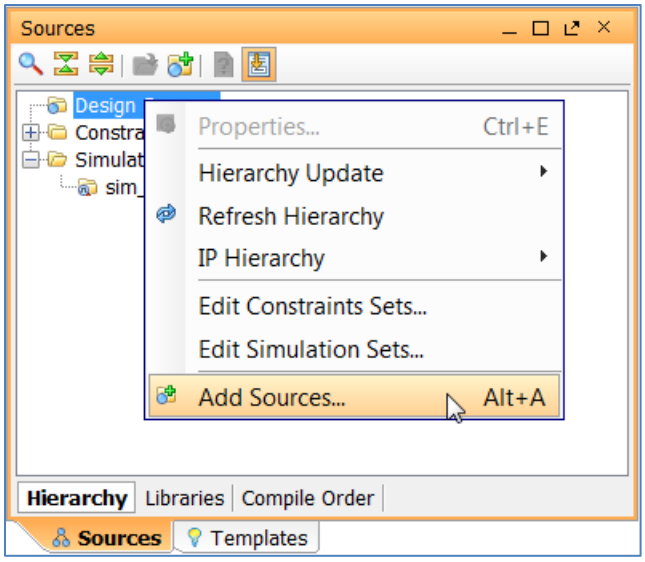


Figure 133: Add Sources to a New Vivado Project

2. The Add Sources dialog box appears. Select **Add Existing Block Design Sources**. Click **Next**.

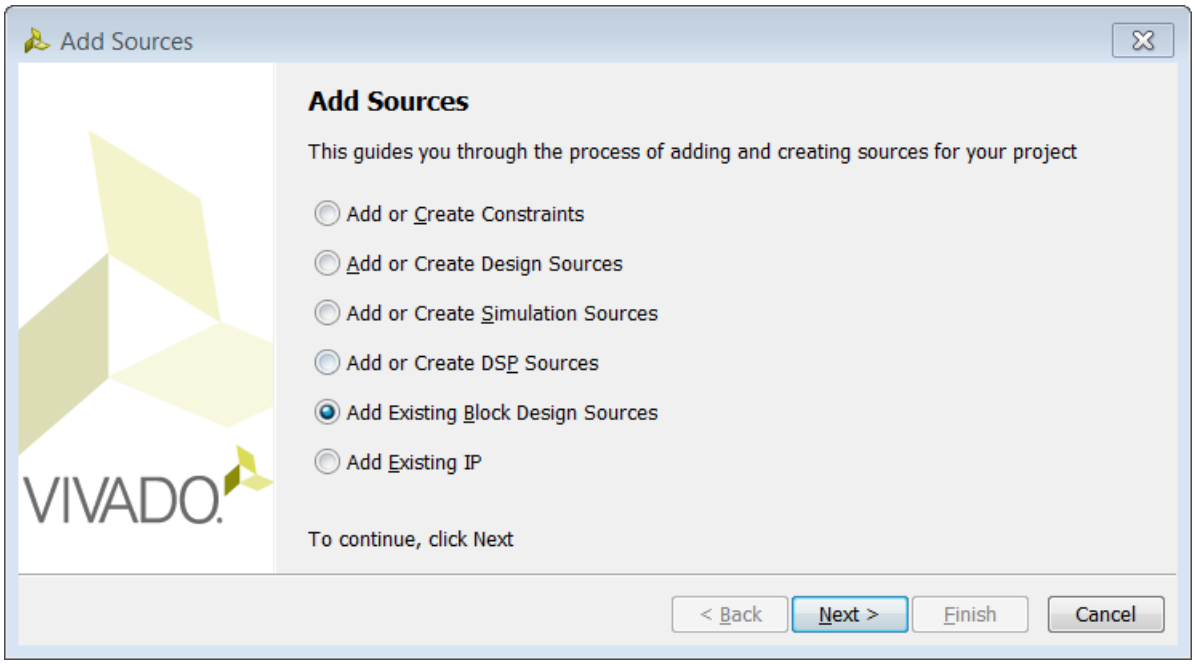


Figure 134: Add Existing Block Design

3. In the Add Sources dialog box, click on **Add Files**.

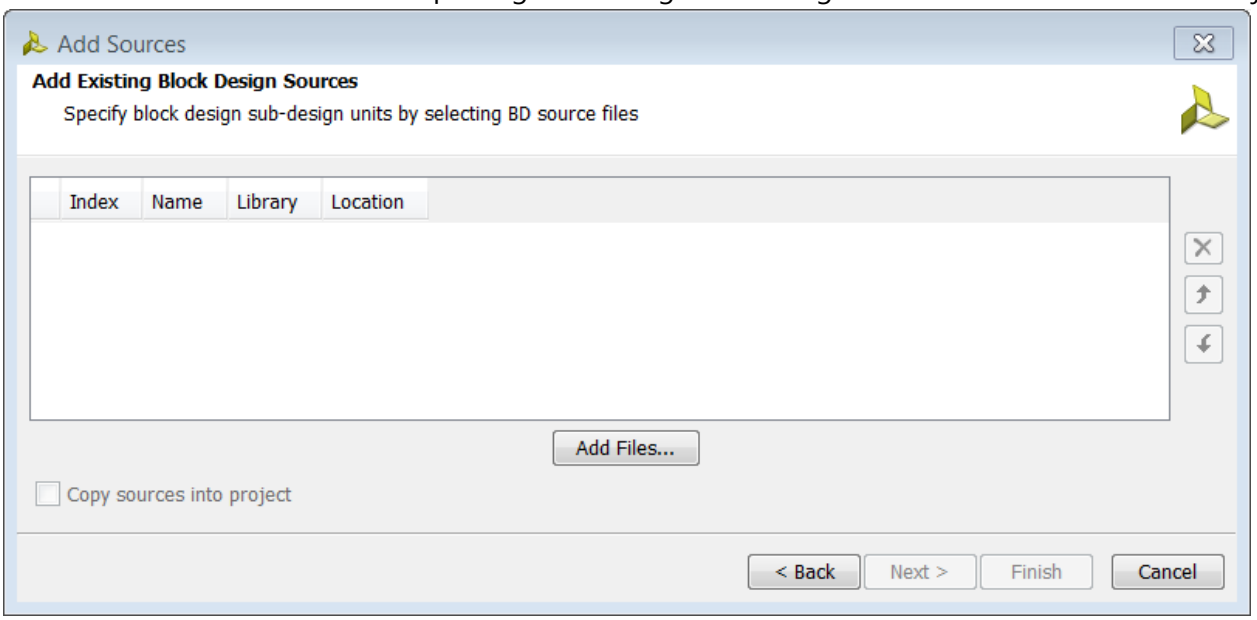


Figure 135: Point to the Existing Block Design File by Clicking on Add Files

4. In the Add Source Files dialog box, browse to the **bd** folder where the block design is located, select the **.bd** file and click **OK**.

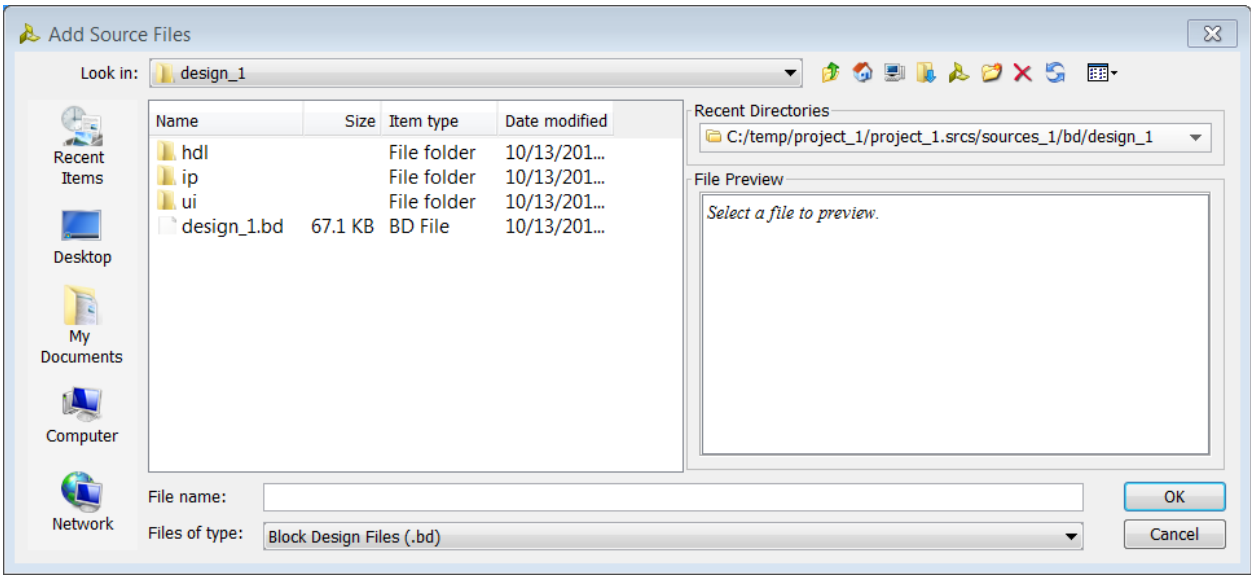


Figure 136: Browse to the Folder Containing the Block Design

5. Click **Finish** once the existing block design has been added.

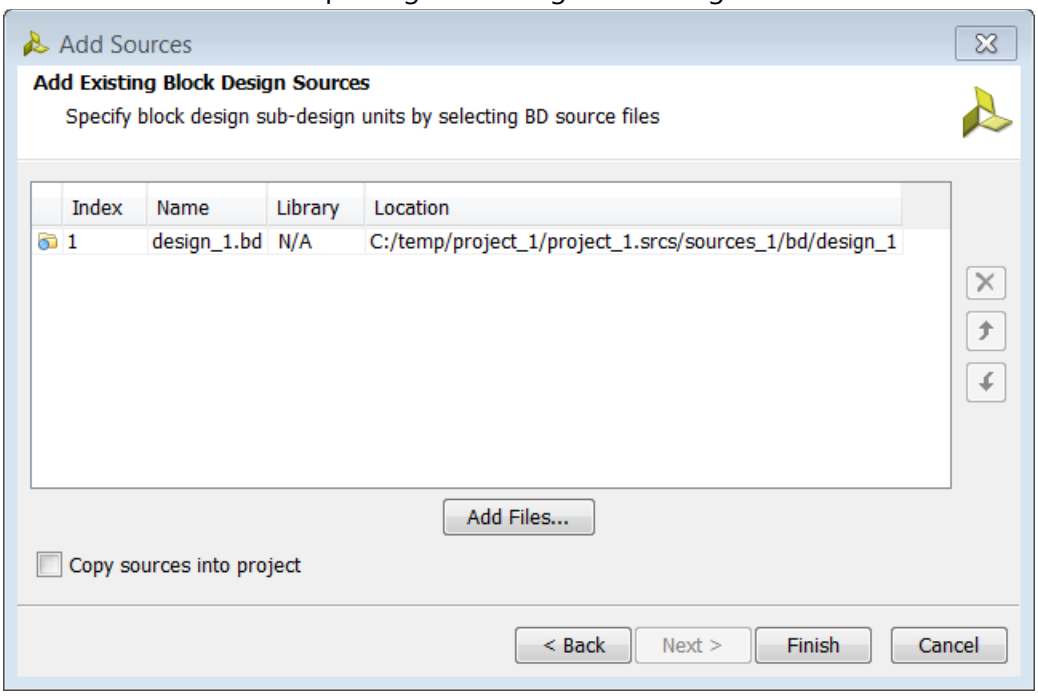


Figure 137: Existing Block Design Sources Added to the Project

In the Sources window, you can see the imported block design under Design Sources.

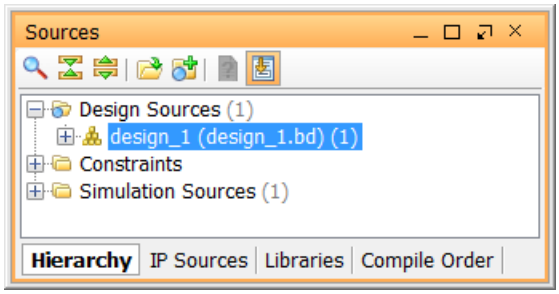


Figure 138: Imported Block Design in the Sources Window

6. Open the block design by double clicking on it.

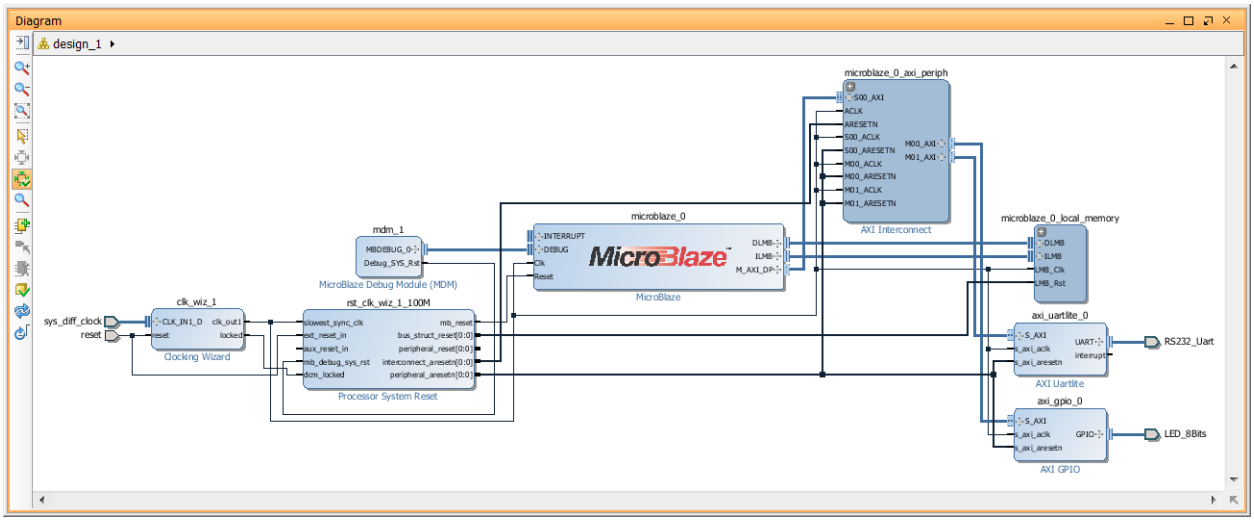


Figure 139: Open the Imported Block Design

7. Validate the design by clicking on **Validate Design** in the IP integrator toolbar.

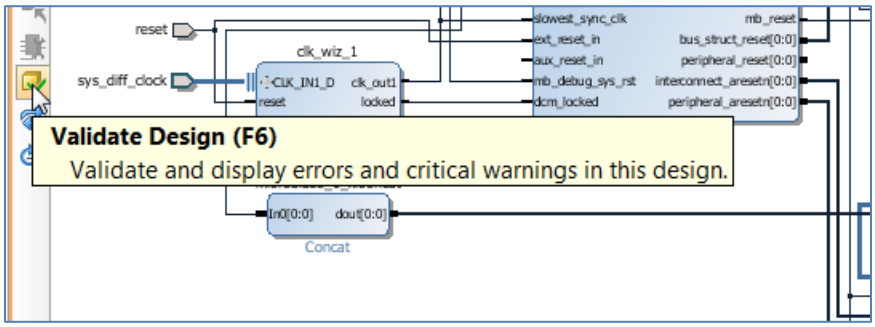


Figure 140: Validate the Imported Block Design

8. Ensure that there are no DRC violations.

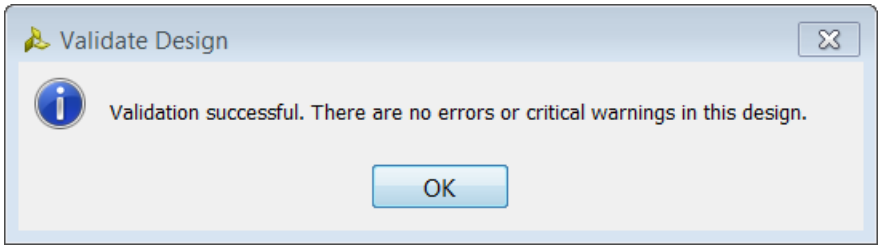


Figure 141: Ensure that no DRC Violations are flagged

9. Follow on through creating a wrapper (if needed) and take the design through implementation.

***Note**: If the existing block design is from a previous version of Vivado Design Suite, then you will see a warning on the Tcl Console as shown below:*

WARNING: [BD 41-1303] One or more IP have been locked in the design <bd_name>. Please run report_ip_status for more details and recommendations on how to fix this issue.

You can open the block design, report IP Status and do an upgrade on the locked (older versions of IP) as previously described in this document.

Chapter 9 Using Third-Party Synthesis Tools with IP Integrator

Overview

Sometimes it may be necessary to use a third-party synthesis tool as a part of the design flow. In this case, you will need to incorporate the block design as a black-box to the top-level user design. You can then synthesize the rest of the design in a third-party synthesis tool, write out a HDL or EDIF netlist and implement the post-synthesis project in the Vivado environment.

This chapter describes the necessary steps that are required to synthesize the black-box of a block design in a third-party synthesis tool. Although the flow is applicable to any third-party synthesis tool, this chapter has been written for Synplify Pro synthesis tool.

Creating a Design Check Point (DCP) File for a Block Design

A design check point can be created for a block design by setting the block design as an Out-of-Context module. To do this you can select the block design in the sources window, right-click and choose **Set as Out-of-Context Module.**

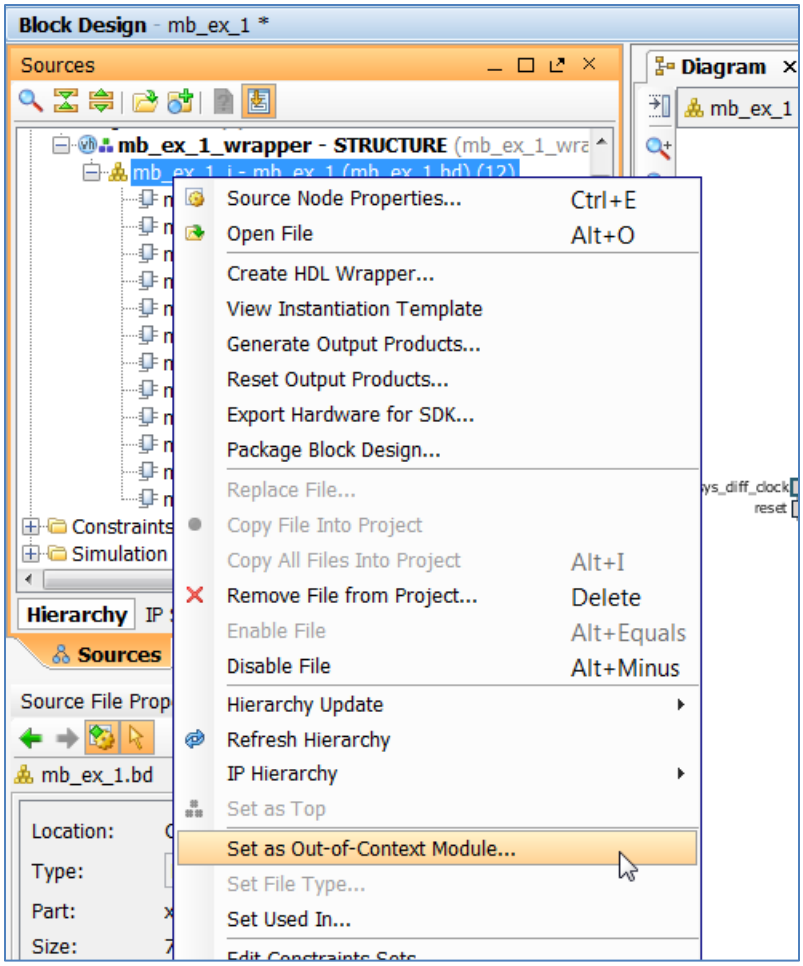


Figure 103: Set Block Design as an Out-of-Context Module

A square is placed against the block design in the Sources view to indicate that the block design has been set as an out-of-context module. The Design Runs window also shows an Out-of-Context module run for the block design.

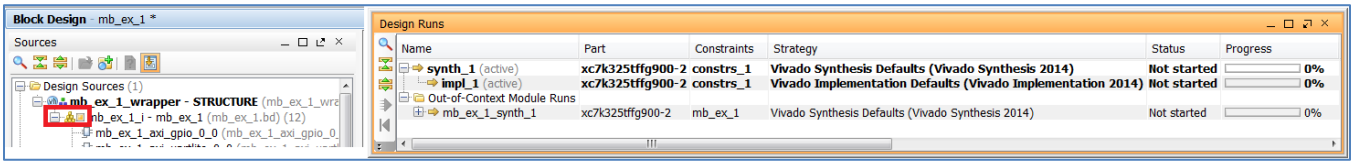


Figure 104: Sources and Design Runs Window after Setting Out-of-Context Module

Next, synthesize the design which will create a design-check-point file for the block design which can be found in the directory shown below.

```
<path_to_design>\<project_name>\<project_name>.runs\<block_design_name>_synth_1
```

Design checkpoints enable you to take a snapshot of your design in its current state. The current netlist, constraints, and implementation results are stored in the design checkpoint. Using design checkpoints, you can:

- Restore your design if needed
- Perform design analysis
- Define constraints
- Proceed with the design flow

CAUTION! A Block Memory Map file is a text file that has syntactic descriptions of how individual block RAMs constitute a contiguous logical data space. Data2MEM uses BMM files to direct the translation of data into the proper initialization form. Design checkpoints do not include the Block Memory Map (BMM) file. The contents of the BMM are incorporated into the netlist after synthesis, so the BMM file is not needed. A BMM file is used in a 7 series device.

For UltraScale devices, an equivalent MMI file is used that contains the memory map information. The equivalent command that uses the MMI file in UltraScale devices is UpdateMem.

Create a Verilog or VHDL Stub File for the Block Design

Once the check-point has been generated, you need to create a stub file which can be instantiated in the top-level HDL file to refer to the block design as a black-box. First open the synthesized design and then to create the HDL stub file using the following commands:

```
write_verilog -mode synth_stub <path_to_file>/<file_name>
write_vhdl -mode synth_stub <path_to_file>/<file_name>
```

CAUTION! The synthesized design must be open in order for the write_verilog/write_vhdl command to work.

An example stub file is shown in the following figure.

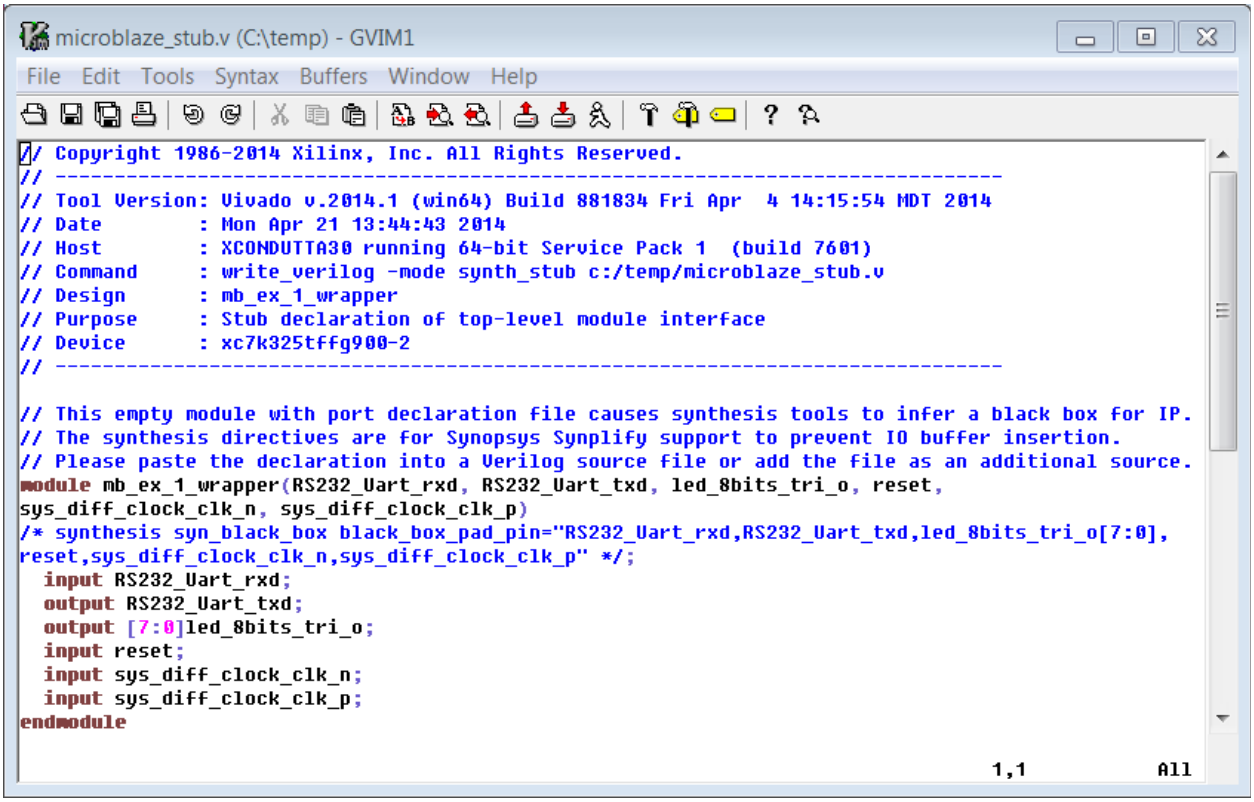


Figure 105: Example Stub File

Create a HDL or EDIF Netlist in the Synplify Project

Create a Synplify project and instantiate the black-box stub file (created in Vivado) along with the top-level HDL wrapper for the block design in the Synplify project. The block design will be treated as a black-box in Synplify. Once the project has been synthesized, an HDL ~~or EDIF~~ netlist for the project can be written out.

Create a Post-Synthesis Project in Vivado and Implement

The next step is to create a post-synthesis project in the Vivado IDE. This can be done by selecting the **Post-synthesis Project** option in the Project type page while creating a New Project in Vivado.

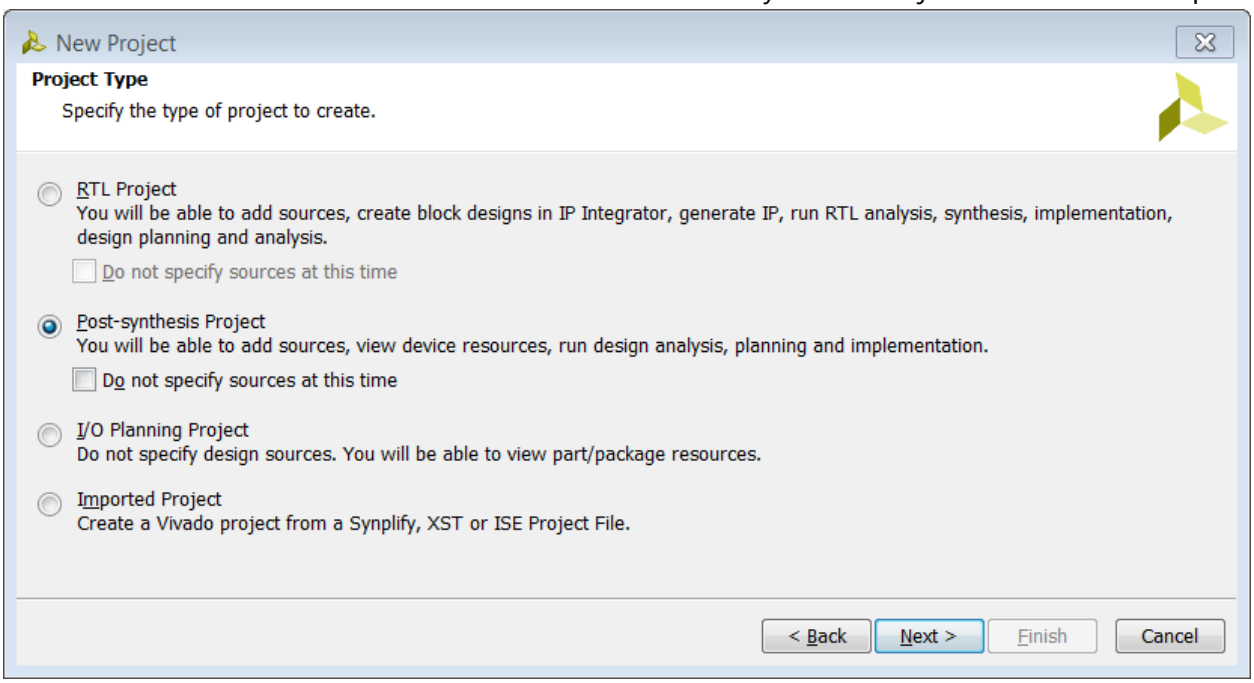


Figure 106: Select the Post-synthesis Project Option

Once the project has been created, add the netlist file and the DCP file for the block design to the project by selecting and right-clicking on Design Sources and then choosing the **Add Sources** option.

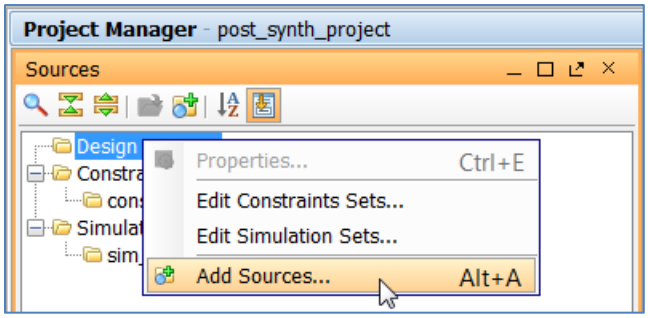


Figure 107: Add HDL Netlist from Synplify and the DCP File to the Project

In the Add Sources dialog box **Add Design Sources** is selected by default. Click **Next**.

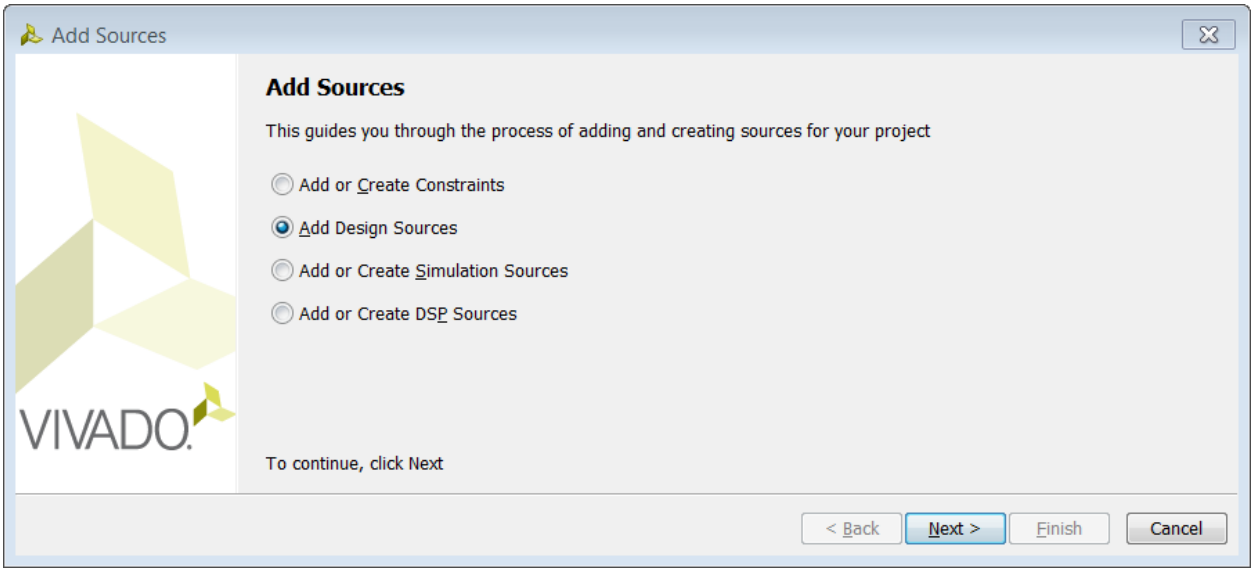


Figure 108: Add Sources dialog box

As shown below in the Add Design Sources page, click on **Add Files**.

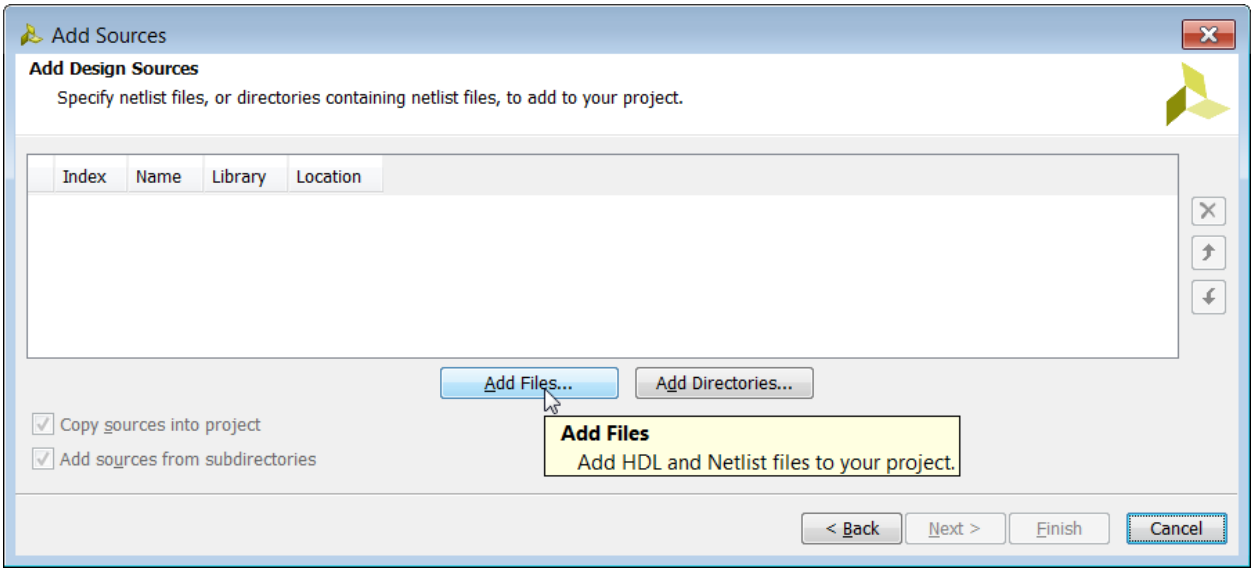


Figure 109: Add Design Sources Page

Select the netlist file by browsing to the right folder. Click **OK**.

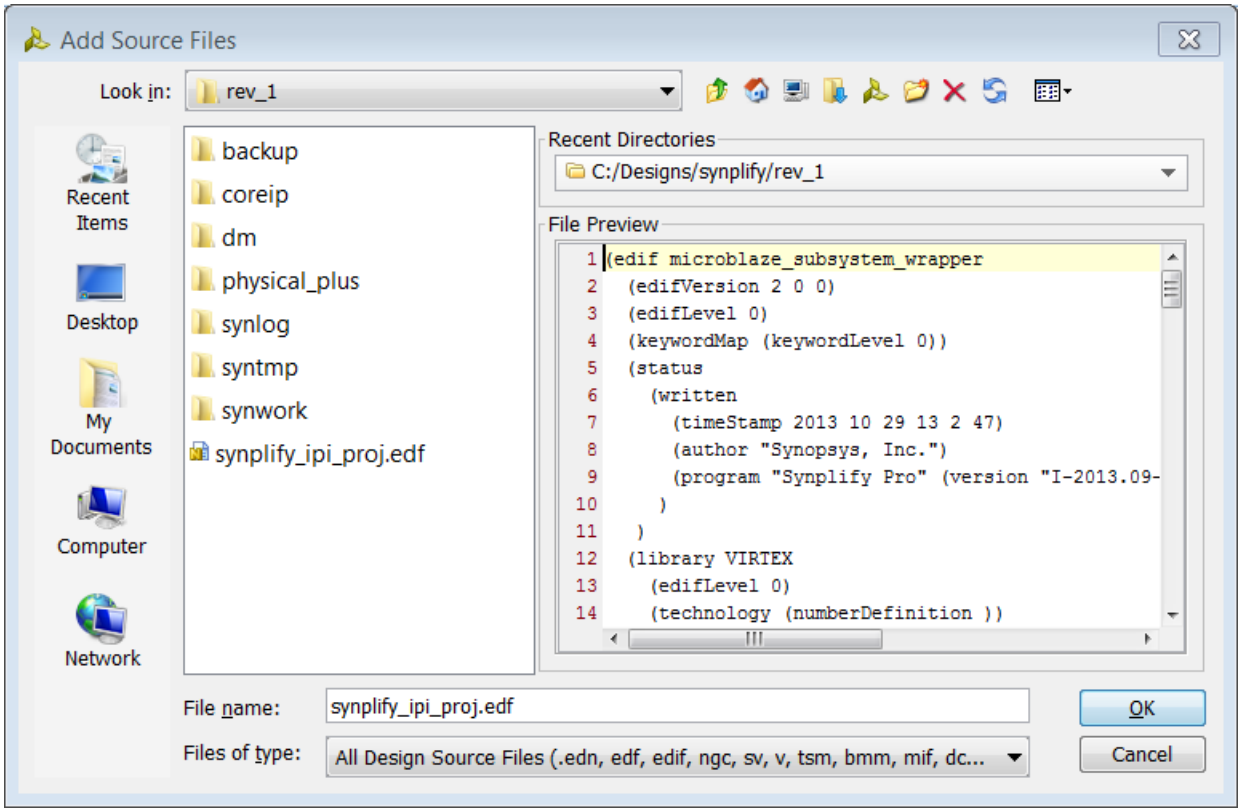


Figure 110: Browse to the Folder Containing the Netlist

Repeat the above steps to add the DCP file as well and then click **Finish**.

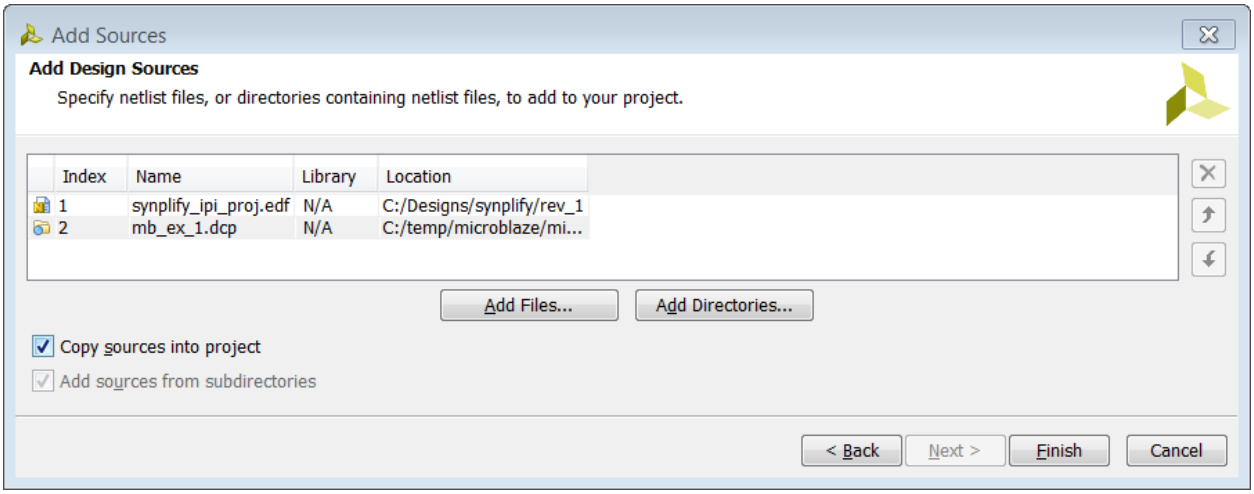


Figure 111: Add Netlist file and DCP File to the Project

Add Top-Level Constraints

Prior to implementing the design, you should add any necessary constraints to the project. The constraints file to the block design is contained with the DCP file. However, if you have changed the hierarchy of the block design, then the constraints file must be modified to make sure that hierarchical paths are properly scoped.

A constraints file can be added to the project just as the netlist and DCP file was added by right-clicking Design Sources in the Sources window, and choosing **Add Sources**. In the Add Sources dialog box select **Add or Create Constraints**.

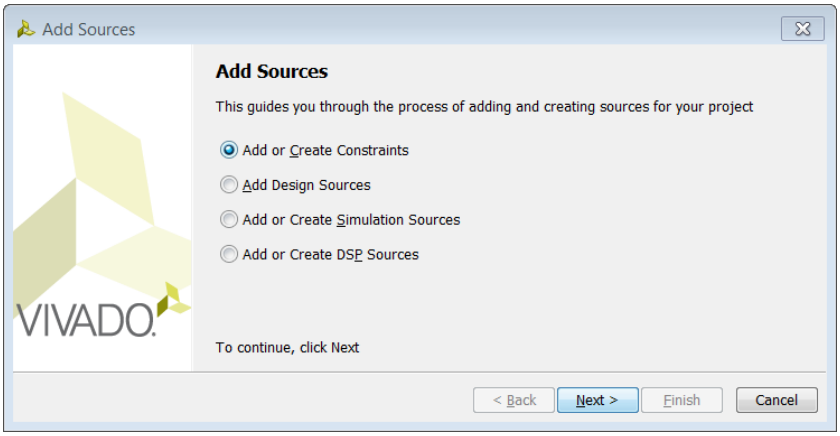


Figure 112: Add Constraints to the top level project

Add ELF File (if present)

If the block design has an ELF file associated to it, then you will need to add the ELF file to the Vivado project. In a post-synthesis project, adding an ELF file via the Vivado IDE GUI is not allowed. However, an ELF file can be added and associated to an embedded object using the following Tcl command:

```
add_files <path_to_elf_file>/<file_name>.elf
```

The added elf files can be seen in the Sources window.

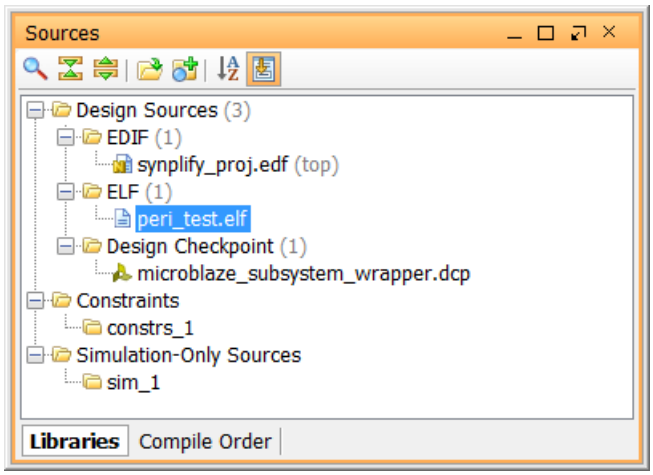


Figure 113: Check to make sure ELF file has been added to the project

Once the ELF file has been added to the project, it can be associated to an embedded object using the following command:

```
set_property SCOPED_TO_CELLS { <processor_instance> } \
[get_files -all -of_objects [get_fileset sources_1] \
```

<path_to_elf_file>/<file_name>.elf]

In the GUI you can do the same thing by selecting the ELF file in the Sources view and then editing the ADDR_MAP_CELLS field in the Source File Properties window.

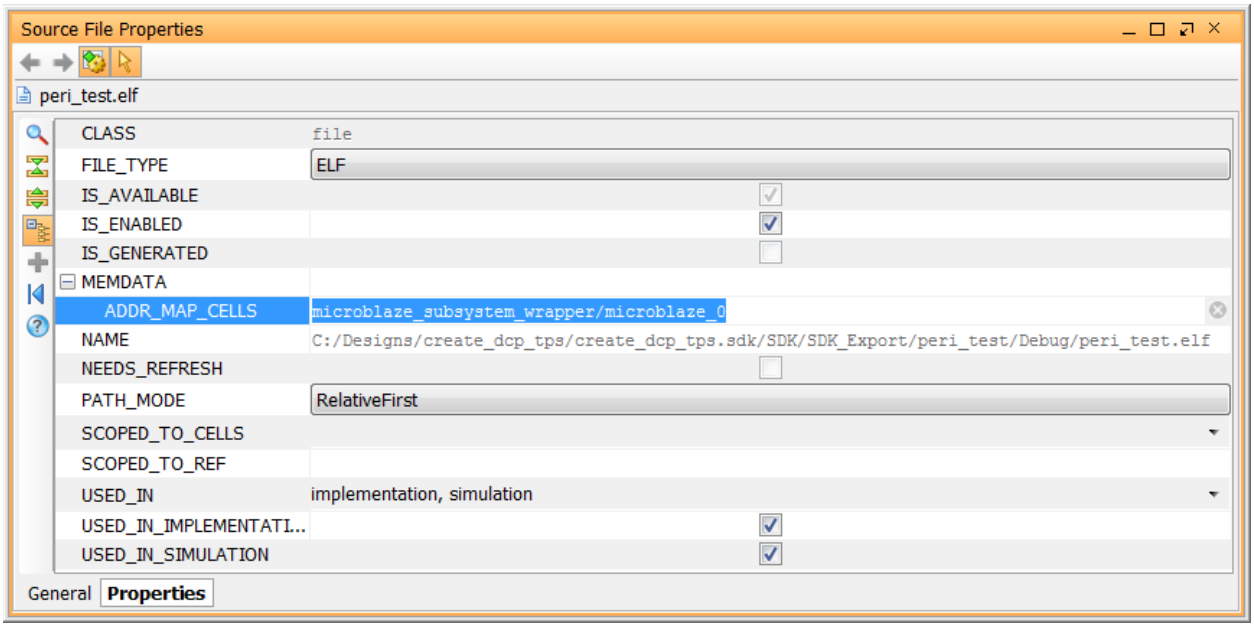


Figure 114: Specify the ADDR_MAP_CELLS field in the Source File Properties window

Implement the Design

Next the design can be implemented and bitstream generated for the design.

Chapter 10 Using the Board Flow in IP Integrator

Overview

The Vivado® Design Suite is board aware. The tools know the various interfaces present on the target boards and therefore, can customize an IP to be instantiated and configured accordingly to connect to a particular board interface. Several 7 series boards are currently supported and support for boards with an Ultrascale part is coming soon. Some other boards from third-party vendors, such as Avnet, are also available now.

The IP integrator shows all the interfaces to the board in a separate tab called the Board Part Interfaces tab. When you use this tab to select the desired interfaces and the Designer Assistance offered by IP integrator, you can easily connect your design to the interfaces of your choice. All the I/O constraints are automatically generated as a part of using this flow.

The following sections describe how the board flow can be used in IP integrator.

Select a Target Board

When a new project is created in the Vivado environment, you have the option to select a target board from the Default Part page of the New Project dialog box.

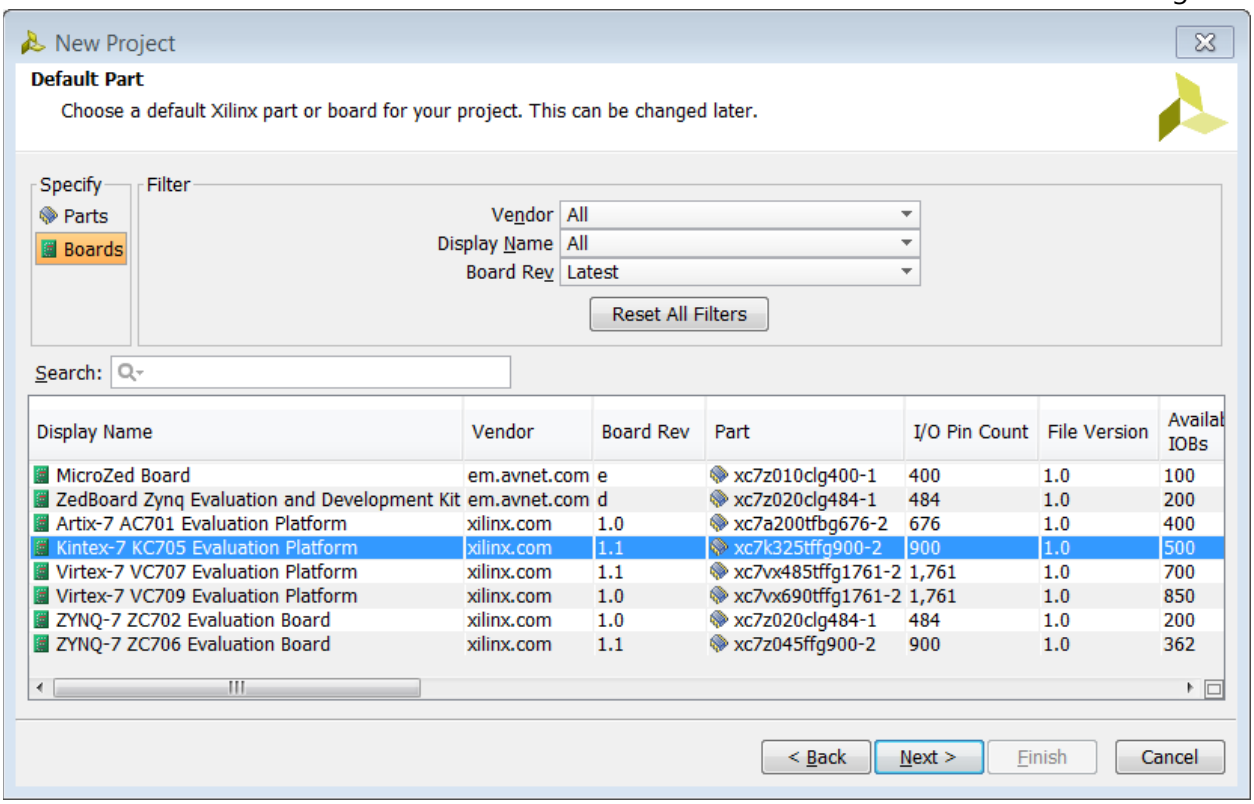


Figure 115: Select a target board

The list of available boards can be filtered based on Vendor, Display Name and Board Revision.

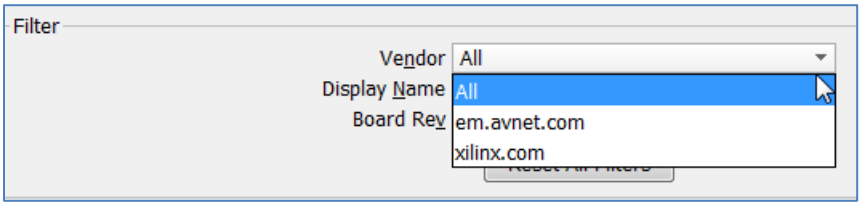


Figure 116: Filter list of available boards

Currently only boards from Xilinx and Avnet are supported. Finally, Board Rev. allows filtering based on the revision of the board. Setting the Board Rev to **All** shows revisions of all the boards that are supported in Vivado. Likewise, setting Board Rev to **Latest** shows only the latest revision of a target board. Various information such as resources available and operating conditions are also listed in the table. When you select a board, the project is configured using the pre-defined interface for that board.

Create a Block Design to use the Board Flow

The real power of the board flow can be seen in the IP integrator tool. Start a new block design by clicking on **Create Block Design** from the drop-down list IP integrator in the Flow Navigator. As the Diagram canvas opens, you will notice a **Board Part Interfaces** window, as shown below.

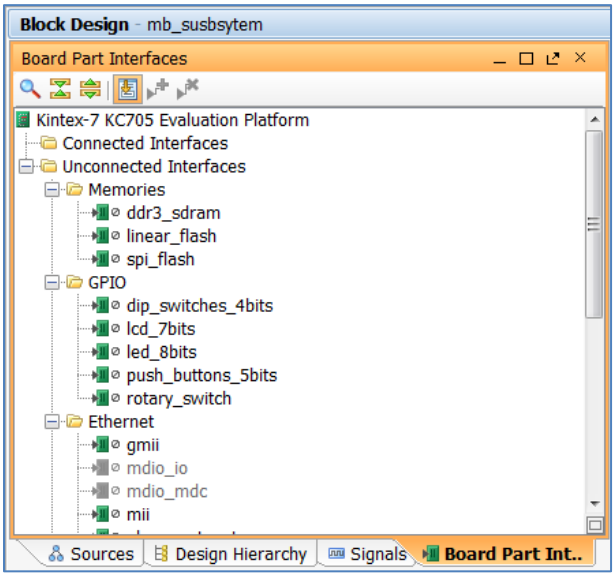


Figure 117: Board Part Interfaces Window

This window lists all the possible interfaces for this evaluation board (in this case the KC705 board). By selecting one of these interfaces, an IP can be quickly instantiated on the block design canvas. As an example, double-click the **ddr3_sdram** interface from the Unconnected Interfaces folder. The Connect Board Part Interface dialog box opens.

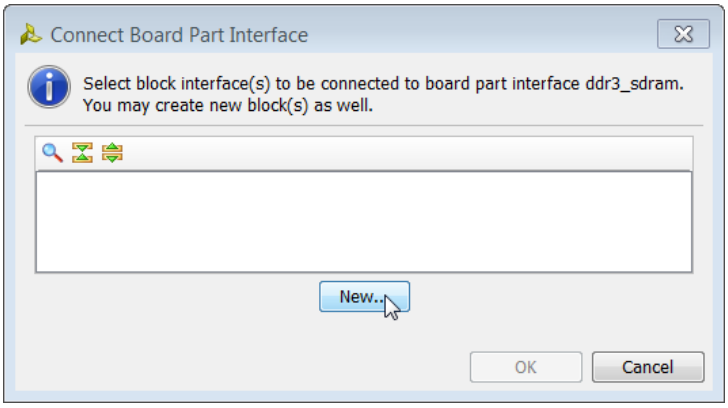


Figure 118: Connect Board Part Interface Dialog Box

Click **New**. The IP catalog opens to show the list of possible IP that can be connected to this interface. In this case there is only one IP that is the Memory Interface Generator. Double-click on the IP of your choice from the IP catalog.

Figure 119: IP Catalog Showing the List of IP that can be Connected to an Interface

The Connect Board Part Interface dialog box now shows the selected IP. Click **OK**.

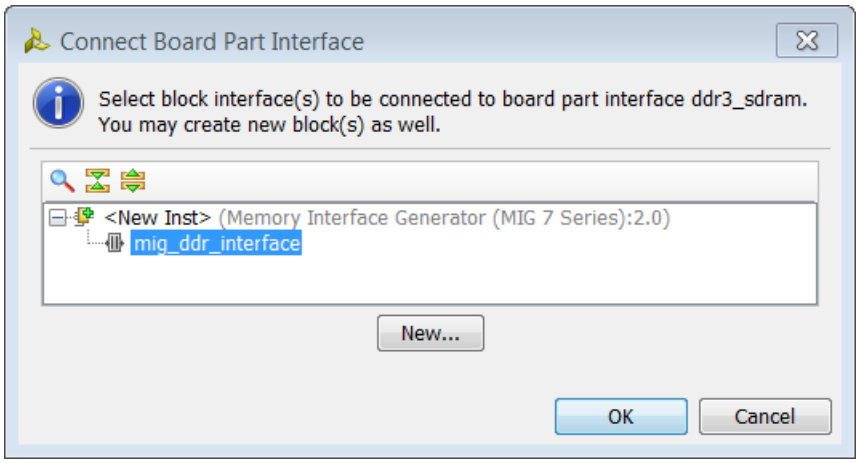


Figure 120: Selected IP Shown in the Connected Board Part Interface Dialog Box

Notice that the IP is placed on the Diagram canvas and connections are made to the interface via I/O ports. The IP is all configured accordingly to connect to that interface.

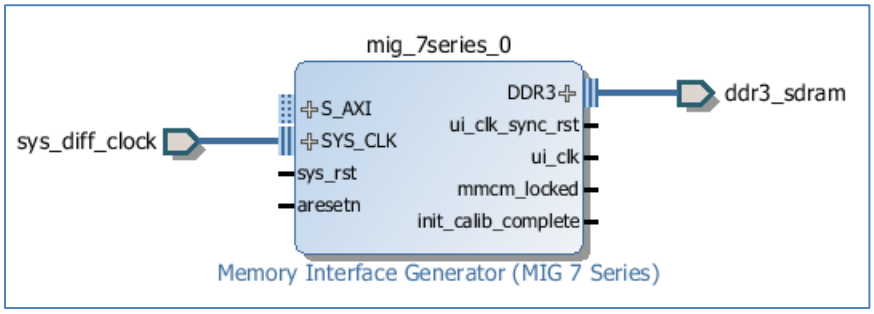


Figure 121: IP instantiated, Configured and Connected to Interfaces on the Diagram Canvas

As an interface is connected, that particular interface now shows up under the Connected Interfaces folder in the Board Part Interfaces window.

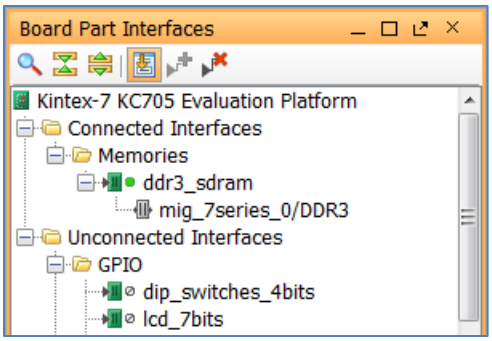


Figure 122: Updated Board Part Interfaces Window after Connecting to an Interface

An interface can also be connected using the Preferred Connection option. To do this, select and right-click on the desired interface and from the menu select **Preferred Connection**.

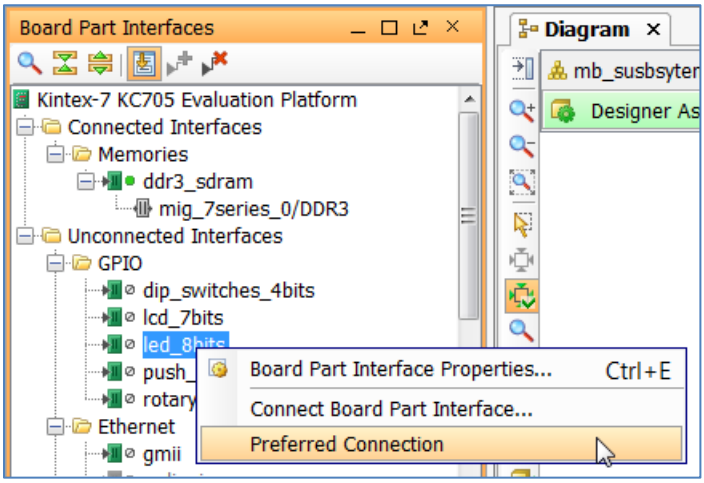


Figure 123: Using the Preferred Connection Option to Connect to a Board Interface

You will notice that the GPIO IP has been instantiated and the GPIO interface is connected to an I/O port.

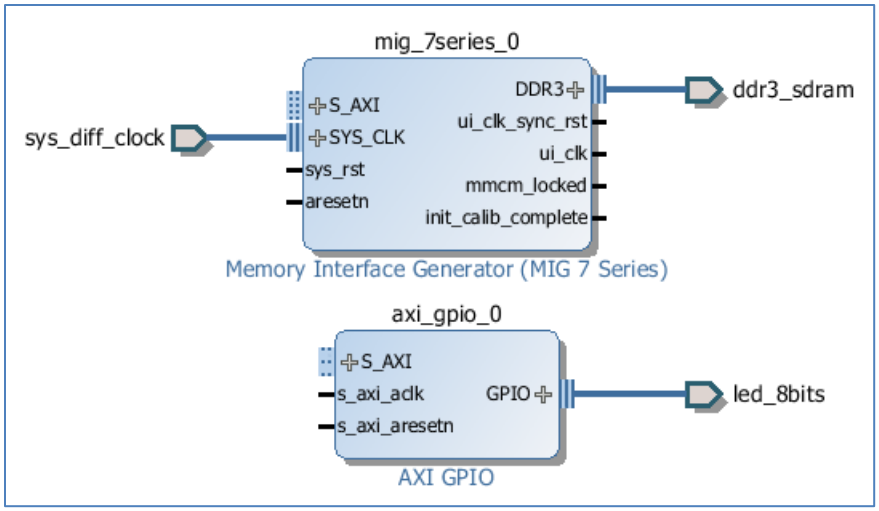


Figure 124: Instantiating an IP using the Preferred Connection Option

If another interface such as dip_switches_4bits is selected, then the board flow is smart enough to know that a GPIO already is instantiated in the design and it re-uses the second channel of the GPIO.

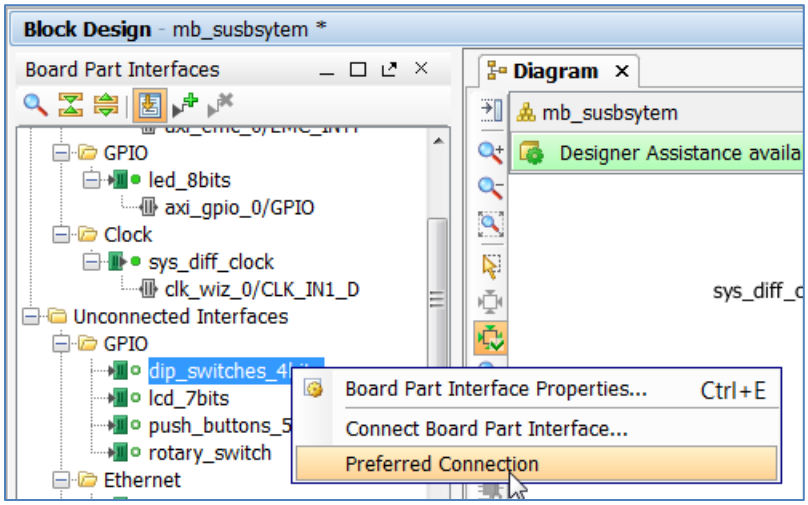


Figure 125: Connecting an interface that can Share an Already Instantiated IP

The already instantiated GPIO is re-configured to use the second channel of the GPIO as shown in the following figure.

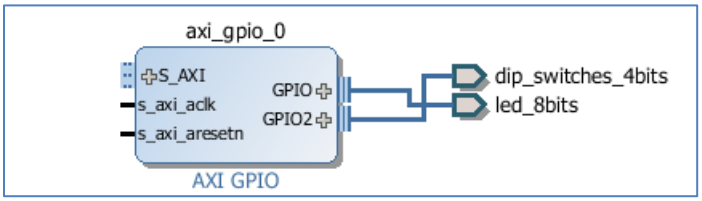


Figure 126: GPIO IP Configured to Use the Second Channel

If an external memory interface such as the linear_flash or the spi_flash is chosen, then as one of them is used the other interface becomes greyed out indicating that only one of those interfaces can be used on this board in a design.

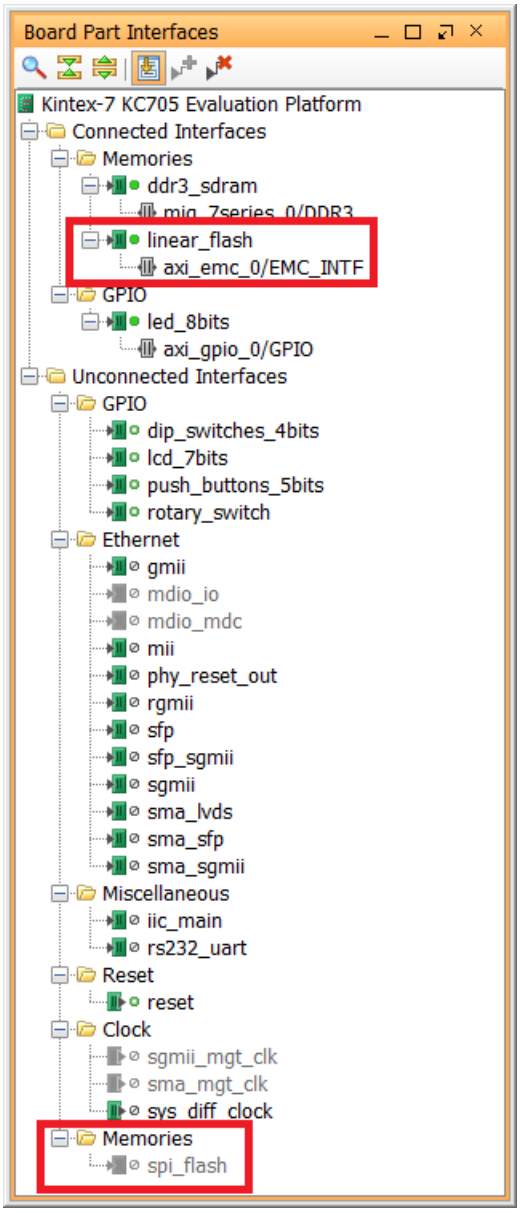


Figure 127: Board Part Interfaces Windows Showing only One External Memory Interface can be used on the KC705

Interfaces on a board can also be connected by selecting and dragging an interface from the Unconnected folder to the Connected Folder. This function also behaves like the Preferred Connection option, in that it instantiates the preferred IP and configures it for that interface. As an example, the sys_diff_clock interface was dragged from the Unconnected Interfaces folder to the Connected Interfaces folder in this example. The Clocking Wizard IP is instantiated on the canvas and the interface connection to the differential clock is made.

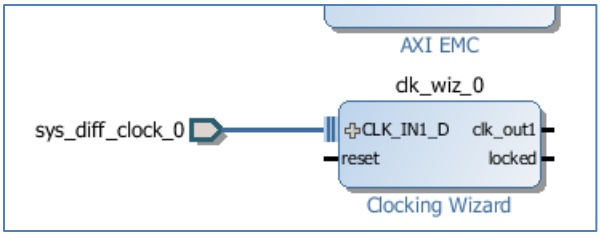


Figure 128: Dragging and Dropping an Interface from Unconnected Interfaces Folder to Connected Interfaces Folder

Double-click on the Clocking Wizard to bring up the Reconfigure IP dialog box. You will notice that the input clock frequency for the Clocking Wizard has been specified to match with the KC705 board frequency of 200 MHz.

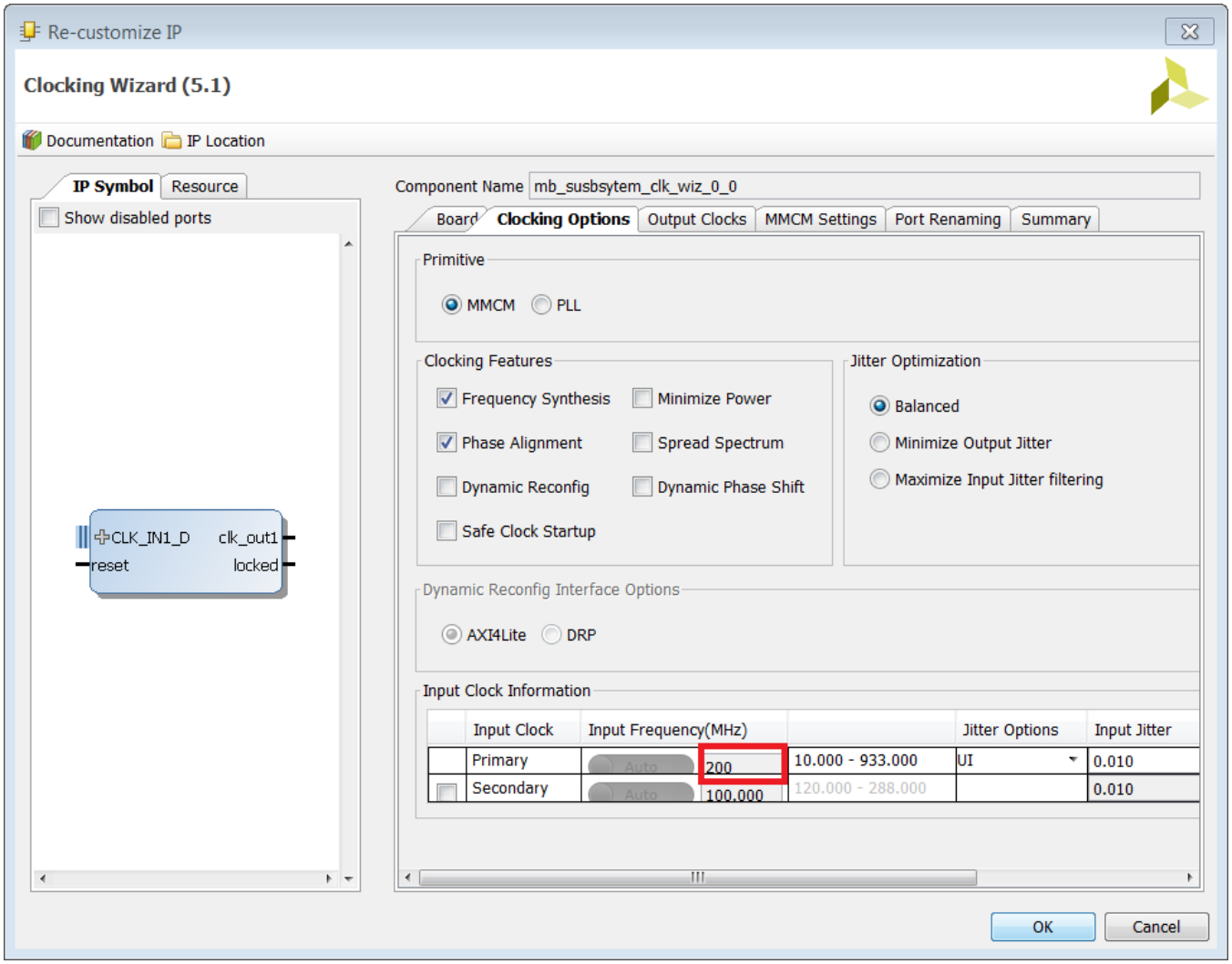


Figure 129: Clocking Wizard Configuration Dialog Box

Complete Connections in the Block Design

Once the desired interfaces have been used in the design, the next step is to instantiate a processor (in case of an processor-based design) or an AXI interconnect if this happens to be a non-embedded design to complete the design.

To do this, right-click on the canvas and select add IP. From the IP catalog choose the MicroBlaze processor, as an example.

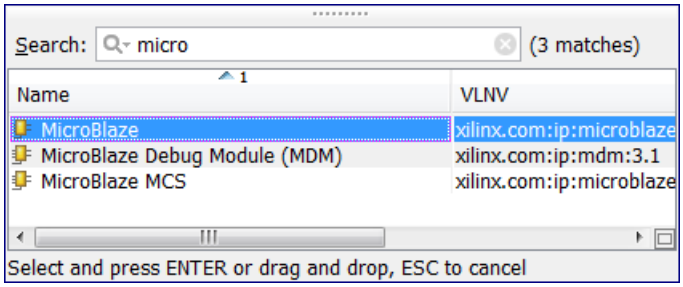


Figure 130: Instantiate a Processor to Complete the Design

As the processor is instantiated, Designer Assistance becomes available.

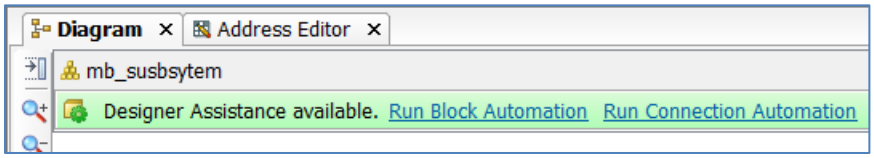


Figure 131: Use Designer Assistance to Complete Connections

Click on **Run Block Automation** to configure a basic processor sub-system. The processor sub-system is created which includes commonly used IP in a sub-system such as block memory controllers, block memory generator and a debug module. Then you can use the Connection Automation feature to connect the rest of the IP in your design to the MicroBlaze processor by selecting Run Connection Automation.

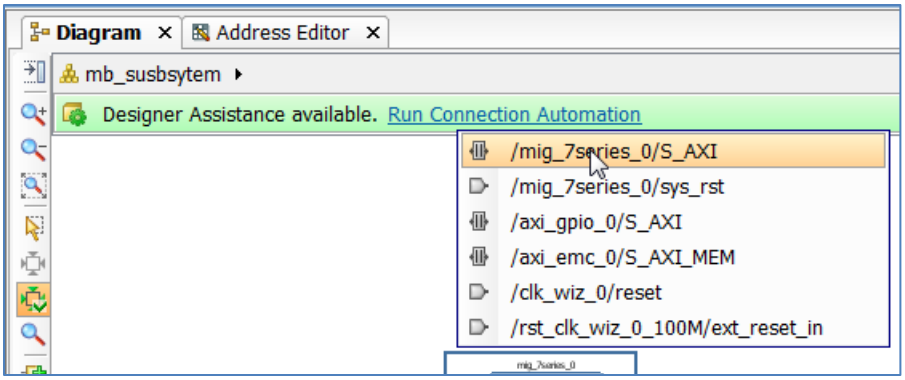


Figure 132: Use Run Connection Automation to Complete Connections to the Processor and Rest of the IP

The rest of the process from here on after is the same as needed for designing in IP integrator as described in Chapter 2 of this document.

Appendix A Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx® Support website at:

www.xilinx.com/support

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm

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.

References

Vivado® Design Suite Documentation

(**www.xilinx.com/support/documentation/dt_vivado_vivado2014-1.htm**)

Vivado Design Suite User Guides

Vivado Design Suite User Guide: System-Level Design Entry (**UG895)**

Vivado Design Suite User Guide: Design Flows Overview (**UG892**)

Vivado Design Suite User Guide: Using the Vivado IDE (**UG893**)

Vivado Design Suite User Guide: Using Tcl Scripting (**UG894**)

Vivado Design Suite User Guide: Designing with IP (**UG896**)

Vivado Design Suite User Guide: Model-Based DSP Design Using System Generator (**UG897**)

Vivado Design Suite User Guide: Embedded Hardware Design (**UG898**)

Vivado Design Suite User Guide: High-Level Synthesis (**UG902**)

Vivado Design Suite User Guide: Using Constraints (**UG903**)

Vivado Design Suite User Guide: Programming and Debugging (**UG908**)

Vivado Design Suite Tutorials

Vivado Design Suite Tutorial: Design Flows Overview (**UG888**)

Vivado Design Suite Tutorial: Designing with IP (**UG939**)

Vivado Design Suite Tutorial: Embedded Hardware Design (**UG940**)

Vivado Design Suite Tutorial: Using Constraints (**UG945**)

Vivado Design Suite Tutorial: Programming and Debugging (**UG936**)

Other Vivado Design Suite Tutorials (**www.xilinx.com/training/vivado/index.htm**)

Other Vivado Design Suite Documents

Vivado Design Suite Tcl Command Reference Guide (**UG835**)

AXI Reference Guide (**UG761**)

Large FPGA Methodology Guide (**UG872**)

Zynq-7000 All Programmable SoC PCB Design and Pin Planning Guide (**UG933**)

Zynq-7000 All Programmable SoC Software Developers Guide (**UG821**)

UltraFast Design Methodology Guide for the Vivado Design Suite (**UG949**)

Other Vivado Design Suite Documentation

(**www.xilinx.com/support/documentation/dt_vivado_vivado2014-1.htm**)

Appendix B Legal Notices

Please Read: Important Legal Notices

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

© Copyright 2014 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.