Vivado Design Suite Tutorial

Embedded Processor Hardware Design

UG940 (v 2014.2) June 4, 2014

Revision History

The following table shows the revision history for this document.

Date	Version	Changes
06/04/2014	2014.2	Validated and updated for Vivado® Design Suite 2014.2 release.
04/23/2014	2014.1	Validated for Vivado Design Suite 2014.1 release.

Table of Contents

This tutorial shows how to build a basic Zynq®-7000 All Programmable (AP) SoC processor and a MicroBlaze™ processor design using the Vivado® Integrated Development Environment (IDE).

In this tutorial, you use the Vivado IP integrator to build a processor design, and then debug the design with the Xilinx® Software Development Kit (SDK) and the Vivado logic analyzer.

IMPORTANT: The Vivado IP integrator is the replacement for Xilinx Platform Studio (XPS) for embedded processor designs, including designs targeting Zynq-7000 devices and MicroBlaze™ processors. XPS only supports designs targeting MicroBlaze processors, not Zynq devices.

Software Requirements

Before starting the tutorial, ensure that the Vivado Design Suite Enterprise Edition is operational, and that you have installed the relevant tutorial design data. For installation instructions and information, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973).

Hardware Requirements

Xilinx recommends a minimum of 2 GB of RAM when using the Vivado Design Suite on larger devices.

Tutorial Design Descriptions

Lab 1: Programming a Zynq-7000 Processor

Lab 1 uses the Zynq-7000 Processing Subsystem (PS) IP, and two peripherals that are instantiated in the Programmable Logic (PL) and connected using the AXI Interconnect. The Lab uses the following IP in the PL:

- A General Purpose IO (GPIO)
- A Block Memory
- An AXI BRAM Controller

Hardware and Software Requirements

The following software is required for Lab 1.

- Vivado Design Suite 2014.1 (Embedded Version)

Lab 1 shows how to graphically build a design in the Vivado IP integrator and use the Designer Assistance feature to connect the IP to the Zynq PS. The lab also demonstrates the Board Automation feature for the `ZYNQ ZC702` Evaluation Board.

After you construct the design, you generate the Hardware Design Language (HDL) for the design as well as for the IP. Then you compile the design and generate a bitstream.

You use the **MARK_DEBUG** properties on the hardware to enable debug of the PL. Then, you export the hardware description of the design to the SDK for software debug.

Design Files

The following design files are included in the zip file for this guide:

- `lab1.tcl`

See **LOCATING TUTORIAL DESIGN FILES**, page 8.

Lab 2: SDK and Logic Analyzer

Lab 2 requires that you have the Software Development Kit (SDK) software installed on your machine.

In Lab 2, you use the SDK software to build and debug the design software, and learn how to connect to the hardware server (`hw_server`) application that SDK uses to communicate with the Zynq-7000 processors. Then you perform logic analysis on the design with a connected board.

Hardware and Software Requirements

The following hardware and software are required for Lab 2.

Hardware Requirements:

- Xilinx Zynq ZC702 board
- One USB (Type A to Type B)
- JTAG platform USB Cable or Digilent Cable
- Power cable to the board

Software Requirements:

- Vivado Design Suite 2014.1 (Embedded Version)

Lab 3: Zynq Cross Trigger Design

Lab 3 requires that you have the Software Development Kit (SDK) software installed on your machine.

In Lab 3, you use the SDK software to build and debug the design software, and learn how to connect to the hardware server (`hw_server`) application that SDK uses to communicate with the Zynq-7000 processors. Then, use the cross-trigger feature of the Zynq processor to perform logic analysis on the design on the target hardware.

Hardware and Software Requirements

The following hardware and software are required for Lab 3.

Hardware Requirements:

- Xilinx Zynq ZC702 board
- One USB (Type A to Type B)
- JTAG platform USB Cable or Digilent Cable
- Power cable to the board

Software Requirements:

- Vivado Design Suite 2014.1 (Embedded Version)

Design Files

The following design files are included in the zip file for this guide:

- `lab3.tcl`

See **LOCATING TUTORIAL DESIGN FILES**, page 8.

Lab 4: Programming a MicroBlaze Processor

Lab 4 uses the Xilinx MicroBlaze processor in the Vivado IP integrator to create a design and perform the same export to SDK, software design, and logic analysis.

Hardware and Software Requirements

The following hardware and software are required for Lab 4.

Hardware Requirements:

- Xilinx Kintex-7 KC705 board
- One USB (Type A to Type B)
- JTAG platform US Cable or Digilent Cable
- Power cable to the board

Software Requirements:

- Vivado Design Suite 2014.1

Design Files

The following design files are included in the zip file for this guide.:

- `lab4.tcl`
- `system.xdc`

See **LOCATING TUTORIAL DESIGN FILES**, page 8.

Lab 5: Migrating EDK IP to the Vivado Design Suite

Lab 5 shows you how to migrate EDK IP for use in the Vivado Design Suite.

Software Requirements

- Vivado Design Suite 2014.1

Design Files

The following design folders are included in the zip file for this guide:

- `axi_gpio_v1_01_b`
- `axi_lite_ipif_v1_01_a`
- `interrupt_control_v2_01_a`
- `proc_common_v3_00_a`

See **LOCATING TUTORIAL DESIGN FILES**.

Locating Tutorial Design Files

Design data is in the `ug940-design-files.zip` file, which can be found at the following link: **https://secure.xilinx.com/webreg/clickthrough.do?cid=358946**.

Introduction

In this lab you create a Zynq®-7000 processor based design and instantiate IP in the processing logic fabric (PL) to complete your design. Then you mark signals to debug in the Vivado® Logic Analyzer (Lab 2). Finally, you take the design through implementation, generate a bitstream, and export the hardware to SDK.

If you are not familiar with the Vivado Integrated Development Environment Vivado (IDE), see the *Vivado Design Suite User Guide: Using the Vivado IDE* (**UG893**).

Step 1: Start the Vivado IDE and Create a Project

1. Start the Vivado IDE (**FIGURE 1**) by clicking the Vivado desktop icon or by typing **vivado** at a terminal command line.
2. From the **Quick Start** section, click **Create New Project**.

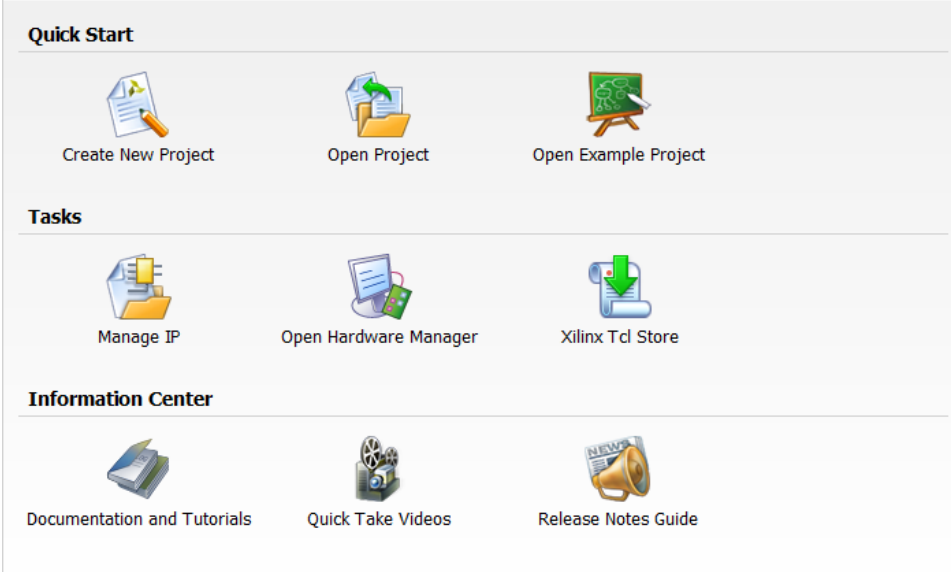

Figure 1: Vivado Quick Start Page

The New Project wizard opens (FIGURE 2).

Figure 2: Create New Project Wizard

3. Click **Next**.
4. In the **Project Name** dialog box, type a project name and select a location for the project files. Ensure that the **Create project subdirectory** check box is checked, and then click **Next**.
5. In the **Project Type** dialog box, select **RTL Project**, and then click **Next**.
6. In the **Add Sources** dialog box, set the **Target language** to your desired language, **Simulator language** to **Mixed**, and then click **Next**.
7. In the **Add Existing IP** dialog box, click **Next**.
8. In the **Add Constraints** dialog box, click **Next**.
9. In the **Default Part** dialog box:
   a. Select **Boards.**
   b. From the **Board Rev** drop-down list, select **All** to view all versions of the supported boards.

CAUTION! Multiple versions of boards are supported in Vivado. Ensure that you are targeting the design to the right hardware.

   c. Choose the version of the **ZYNQ-7 ZC702** Evaluation Board that you are using.
10. Review the project summary in the **New Project Summary** dialog box, and then click **Finish** to create the project.

Step 2: Create an IP Integrator Design

1. In the Flow Navigator, under IP Integrator, select **Create Block Design**.
2. In the **Create Block Design** dialog box, specify a name for your IP subsystem design. Leave the **Directory** field set to the default value of **<Local to Project>**.

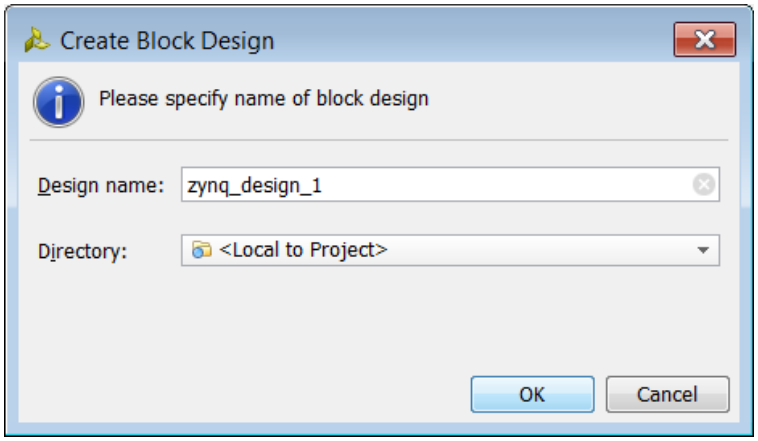

Figure 3: Create Block Design Dialog Box

3. Click **OK**.
4. Right-click in the **Diagram** panel of the Vivado IP integrator window, and select **Add IP**.

   Alternatively, you can click the **Add IP** link in the IP integrator diagram area.

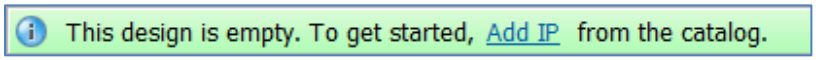

Figure 4: Add IP Link in IP Integrator Canvas

   The IP Catalog opens.

5. In the search field, type **zynq** to find the ZYNQ7 Processing System IP.

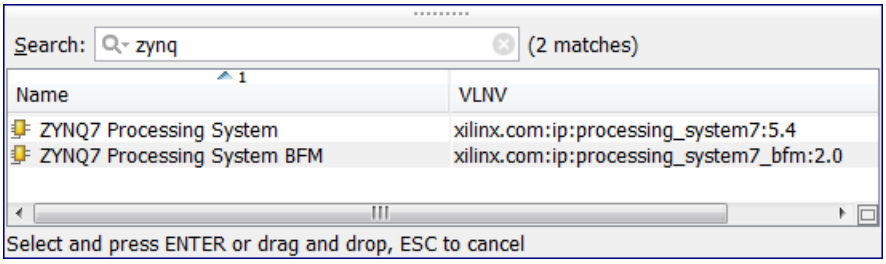

Figure 5: The IP Integrator IP Catalog

6. Select the **ZYNQ7 Processing System** in the IP Catalog and press **Enter** on the keyboard to add it to your design.

   In the Tcl Console, you see the following message:

   ```
   create_bd_cell -type ip -vlnv xilinx.com:ip:processing_system7:5.4
   processing_system7_0
   ```

   There is a corresponding Tcl command for all actions performed in the IP integrator block design. Those commands are not shown in this document. Instead, Tcl scripts to run each labs are provided.

   Note: *Tcl commands are documented in the Vivado Design Suite Tcl Command Reference Guide (**UG835**).*

7. In the IP Integrator window, click the **Run Block Automation** link, and select **/processing_system7_0**.

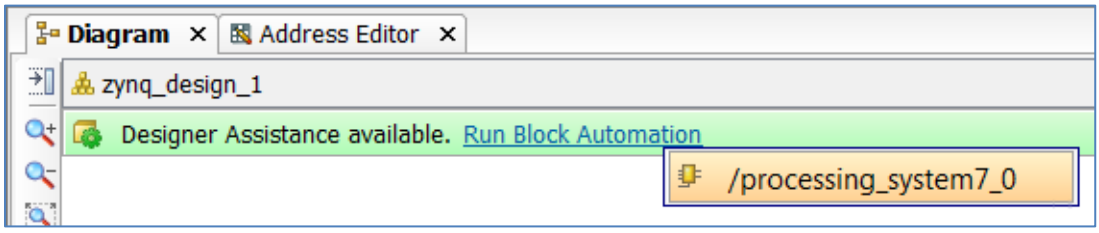

Figure 6: Run Block Automation on Zynq

   The **Run Block Automation** dialog box opens, stating that the FIXED_IO, Trigger, and DDR interfaces will be created for the Zynq core. Also, note that the **Apply Board Preset** check box is checked. This is because the selected target board is ZC702.

8. Make sure that both **Cross Trigger In** and **Cross Trigger Out** are disabled.

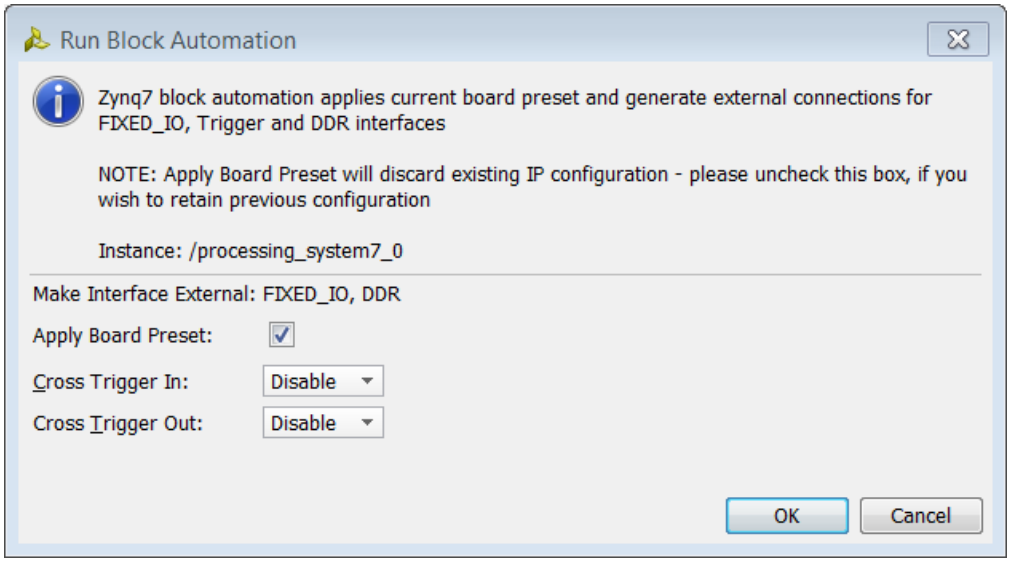

Figure 7: Zynq-7 Run Block Automation Dialog Box

9. Click **OK**.

   After running block automation on the Zynq processor, the IP integrator diagram should look as follows:

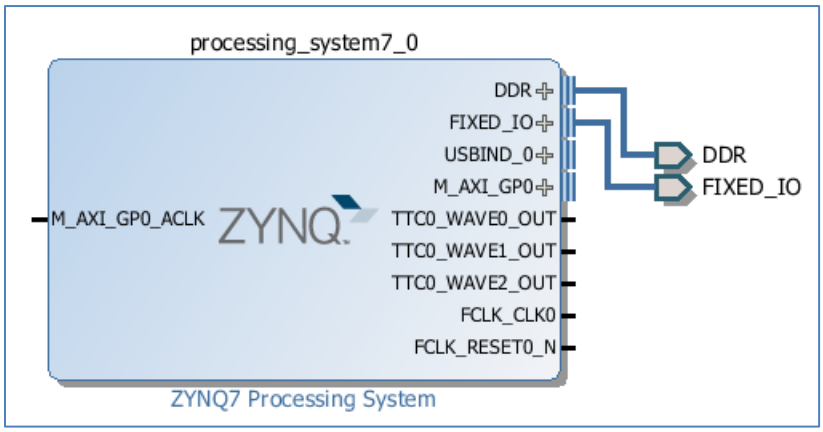

Figure 8: Zynq Processing System after Running Block Automation

10. Now you can add peripherals to the processing logic (PL). To do this, right-click in the IP integrator diagram, and select **Add IP**.
11. In the search field, type `gpi` to find the AXI GPIO IP, and then press **Enter** to add it to the design.
12. Similarly, add the AXI BRAM Controller.

    Your Block Design window is similar to **FIGURE 9**. The relative positions of the IP might vary.

TIP: You can zoom in and out in the Diagram Panel using the **Zoom In** (or **Ctrl**+=) and **Zoom Out** (or **Ctrl**+-) tools.

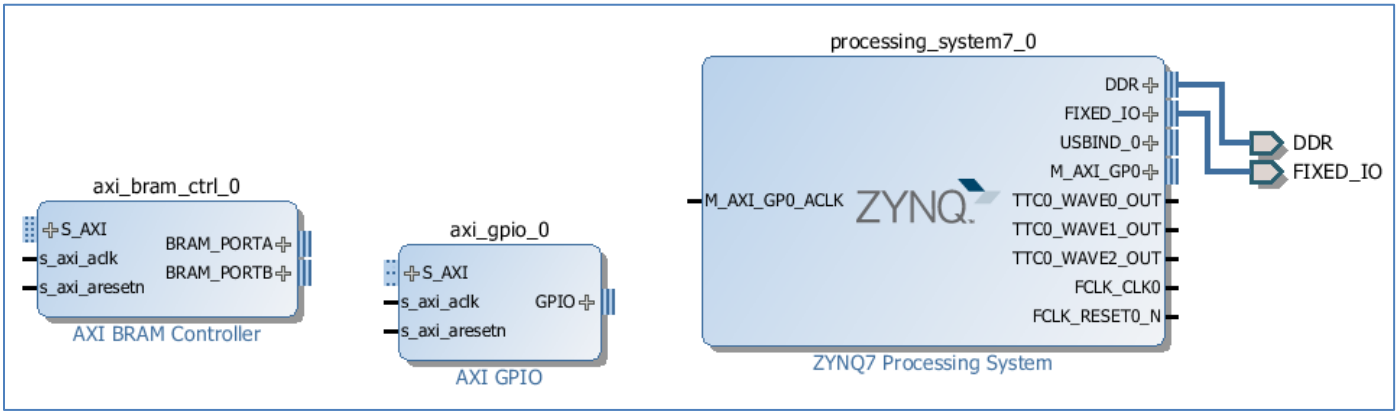

Figure 9: Block Design after Instantiating IP

Use Designer Assistance

Designer Assistance helps connect the AXI GPIO and AXI BRAM Controller to the Zynq-7000 PS.

1. Click **Run Connection Automation** and then select **/axi_gpio_0/S_AXI** to connect the S_AXI interface of the GPIO IP to the Zynq PS (FIGURE 10).

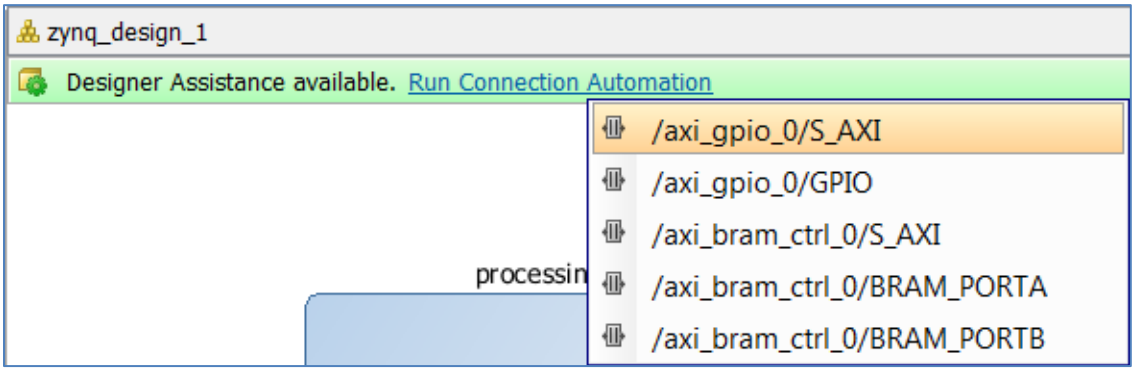

Figure 10: Run Connection Automation

The Run Connection Automation dialog box opens and states that it will connect the AXI Slave Interface of the GPIO to the AXI Master Interface of the Zynq Processor. The dialog box also offers you a selection of clocks that might be available for your design.

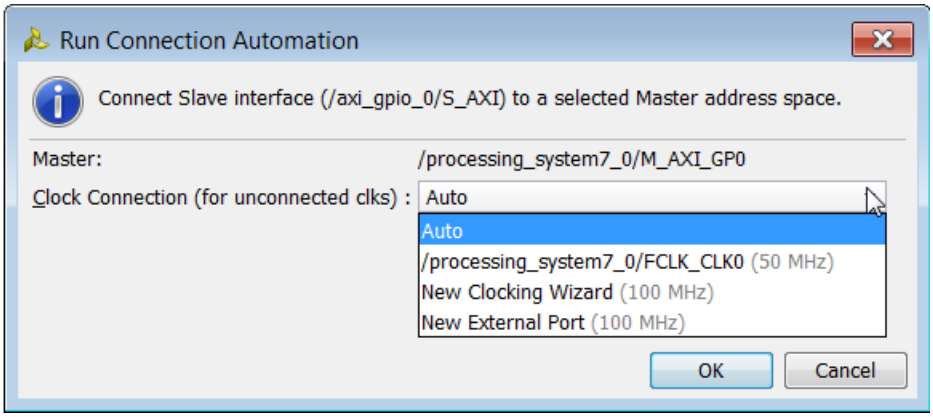

Figure 11: Run Connection Automation Message

In this case we will leave the **Clock Connection (for unconnected clks)** field set to **Auto**.

Notice that the master is the Zynq Processing System IP (FIGURE 11).

2. Click **OK**.

This action instantiates an AXI Interconnect IP as well as a Proc Sys Reset IP and makes the interconnection between the AXI interface of the GPIO and the Zynq-7000 PS.

3. Next, use the same method to add the remaining connections with the following settings:

Connection	*More Information*	*Setting*
/axi_gpio_0/GPIO	The Run Connection Automation dialog box includes options to hook up to the GPIO port. This step also configures the IP appropriately and creates the necessary Xilinx® Design Constraints (XDC) for the GPIO IP.	Select Board Part Interface: **leds_4bits**
/axi_bram_ctrl_0/S_AXI	This completes the connection between the Zynq7 Processing System and the AXI BRAM Controller.	Clock: Auto
/axi_bram_ctrl_0/BRAM_PORTA	This connects the BRAM_PORTA interface of the AXI BRAM Controller to the BRAM_PORTA interface of the Block Memory Generator.	No option
/axi_bram_ctrl_0/BRAM_PORTB	In the Run Connection Automation dialog box, you are presented with two choices. You can use the existing Block Memory Generator (which was instantiated in the above step), or you can instantiate a new Block Memory Generator to connect to this port. For this lab, we are using the existing Block Memory Generator.	Blk_Mem_Gen: **/axi_bram_ctrl_0_bram**

The IP integrator subsystem looks like **FIGURE 12**. Again, the relative positions of the IP can differ slightly.

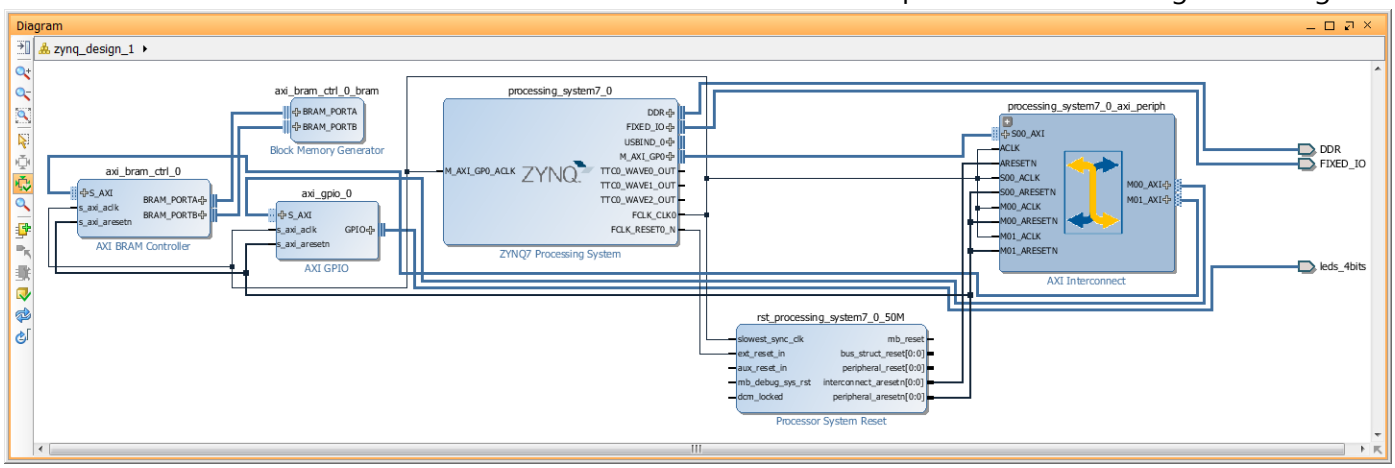

Figure 12: Zynq Processor System

4. Click the **Address Editor** tab and expand the processing_system7_0 hierarchy to show the memory map of all the IP in the design.

   In this case, there are two IP: the AXI GPIO and the AXI BRAM Controller. The IP integrator assigns the memory maps for these IP automatically. You can change them if necessary.

5. Change the range of the AXI BRAM Controller to **64K**, as shown in **FIGURE 13**.

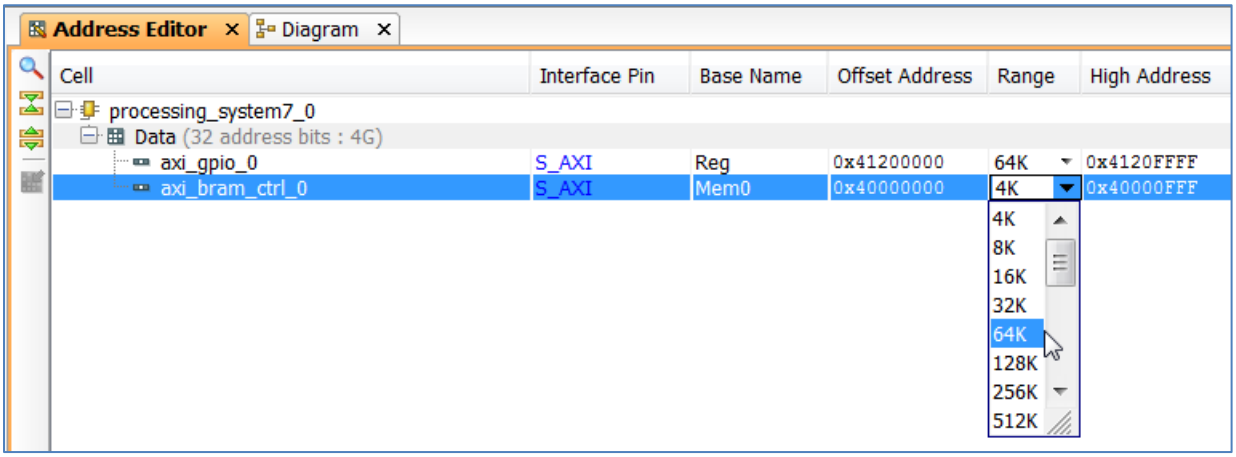

Figure 13: Change Range of axi_bram_ctrl to 64K

Step 3: Using MARK_DEBUG

You now instrument the design and add hooks to debug nets of interest.

1. Click the **Diagram** tab to show the Diagram view.
2. To debug the master/slave interface between the AXI Interconnect IP (`processing_system7_0_axi_periph`) and the GPIO core (`axi_gpio_0`), in the Diagram view, select the interface, then right-click and select **Mark Debug**.

   In the Block Design canvas on the net that you selected in the previous step, a small bug icon appears, indicating that the net has been marked for debug. You can also see this in the Design Hierarchy view, on the interface that you chose to mark for debug. (FIGURE 14).

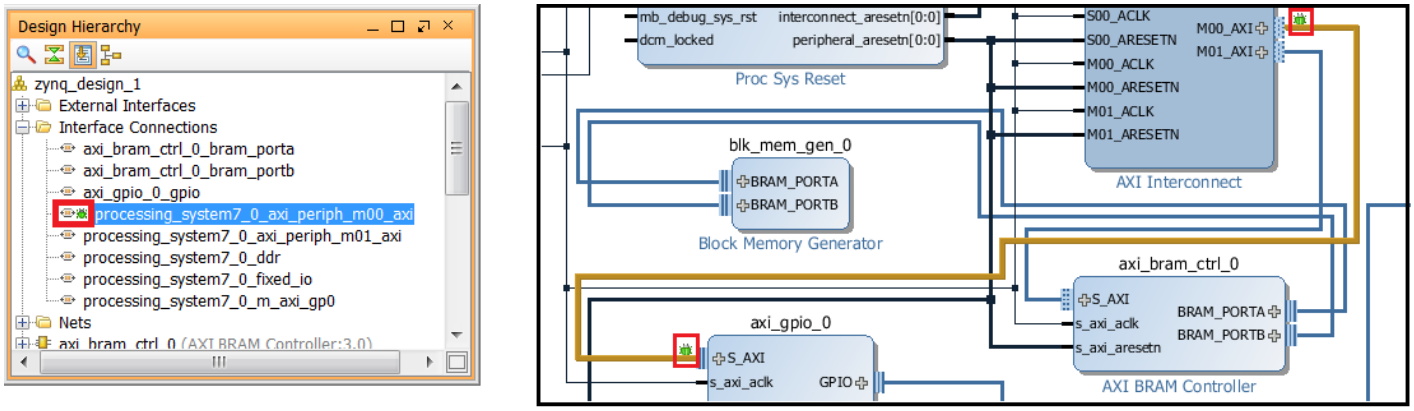

Figure 14: Design Hierarchy: Icon for Mark Debug

3. Right-click anywhere in the IP integrator diagram and select **Create Comment** to add a title and/or comment to the design. This step is optional.
4. If you added a comment and title in step 3, you can format it. To do so:
   a. Select the comment box.
   b. Right-click and select **Format Comment** to format the comment.
5. The **Format Comment** dialog box opens (FIGURE 15).

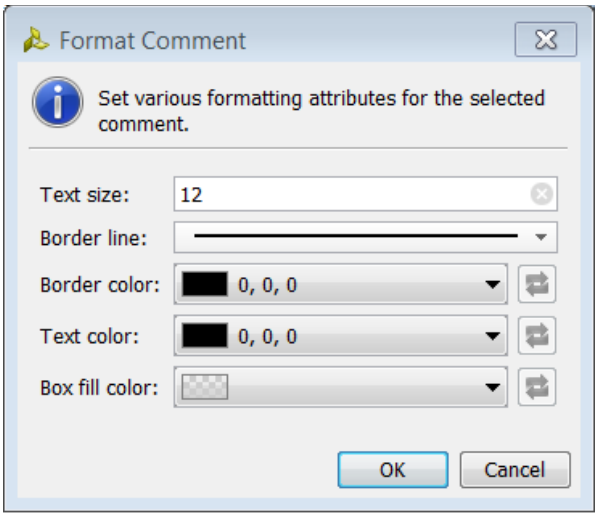

Figure 15: Format Comment Dialog Box

6. Select the appropriate options and click **OK**.
7. On the IP Integrator Diagram toolbar, click the **Regenerate Layout** button .

   IP Integrator optimizes the placement of the IP in the block design.
8. From the toolbar, run Design-Rules-Check (DRC) by clicking the **Validate Design** button . Alternatively, you can do the same from the menu by:
   - Selecting **Tools** > **Validate Design** from the menu.
   - Right-clicking in the Diagram window and selecting **Validate Design**.

   The Validate Design dialog box opens to notify you that there are no errors or critical warnings in the design.
9. Click **OK**.
10. Save the block design by selecting **File > Save Block Design** from the Vivado menu.

    Alternatively, you can press **Ctrl + S** to save your block design or click the **Save** button in the Vivado toolbar.

Step 4: Generate HDL Design Files

You now generate the HDL files for the design.

1. In the Sources window, right-click the top-level subsystem design and select **Generate Output Products**.

   This generates the source files for the IP used in the block design and the relevant constraints file.

   You can also click **Generate Block Design** in the Flow Navigator to generate the output products.

2. The Generate Output Products dialog box opens. Click **Generate**.

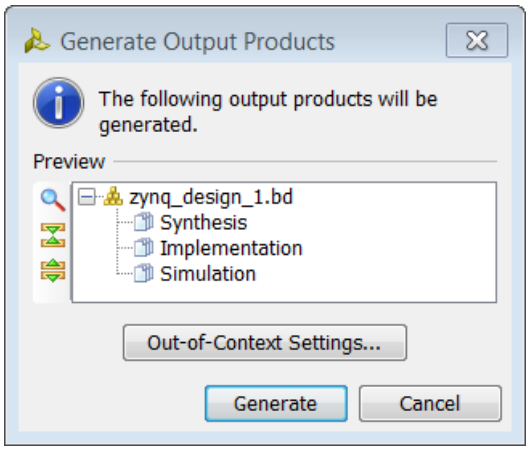

Figure 16: Manage Output Products Dialog Box

3. In the Sources window, right-click the top-level subsystem source and select **Create HDL Wrapper** to create an example top level HDL file.

   The Create HDL Wrapper dialog box presents you with two options. The first option is to copy the wrapper to allow edits to the generated HDL file. The second option is to create a read-only wrapper file, which will be automatically generated and updated by Vivado.

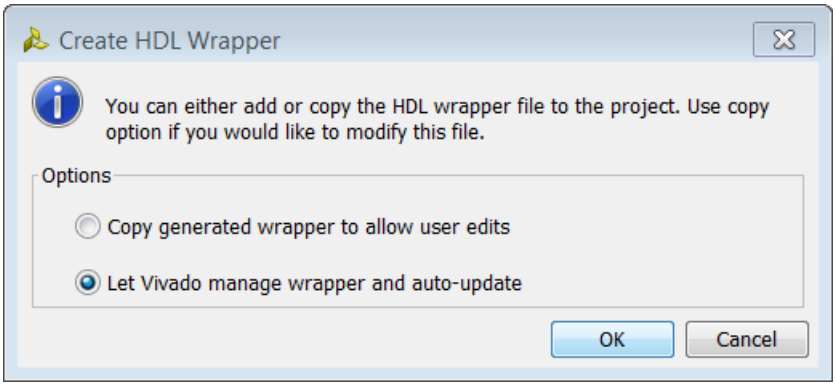

Figure 17: Create HDL Wrapper Dialog Box

4. Select the default option of **Let Vivado manage wrapper and auto-update**.
5. Click **OK**.

   Once the wrapper has been created, the Sources window looks as follows.

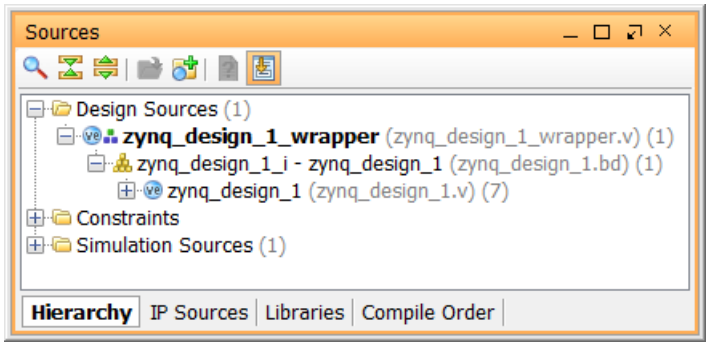

Figure 18: Source Window After Creating the Wrapper

Step 5: Synthesize the Design

1. After generating the IP Integrator design, in the **Flow Navigator**, click **Run Synthesis**.

   ***Note**: Running synthesis could take several minutes.*

   When synthesis completes, the Synthesis Completed dialog box appears.

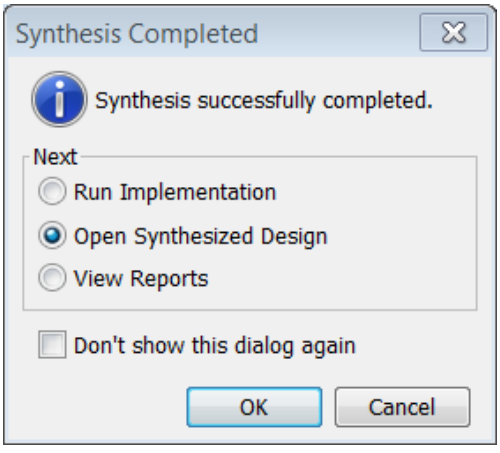

Figure 19: Synthesis Completed Dialog Box

2. Select the **Open Synthesized Design** option, and click **OK**.

   Vivado opens the synthesized design.

3. The Debug window opens as you open the synthesized design.

   Note: *You can open this window manually by selecting* ***Window > Debug****.*

   The Debug window displays a list of nets in the `Unassigned Debug Nets` folder. These nets correspond to the various signals that make up the interface connection that you marked for debug in the IP block design (**FIGURE 20**).

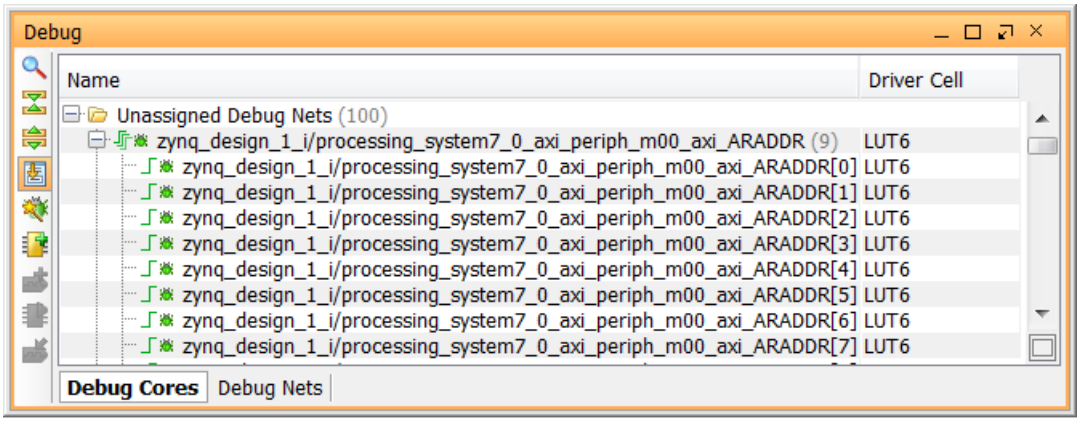

Figure 20: Unassigned Debug Nets Folder

Next, you assign the debug nets to an Integrated Logic Analyzer (ILA) debug core.

Step 6: Assign Debug Net to an ILA Core

1. On the left-hand toolbar of the Debug window, click the **Set up Debug** button  .
2. When the Set up Debug wizard opens (FIGURE 21), click **Next**.

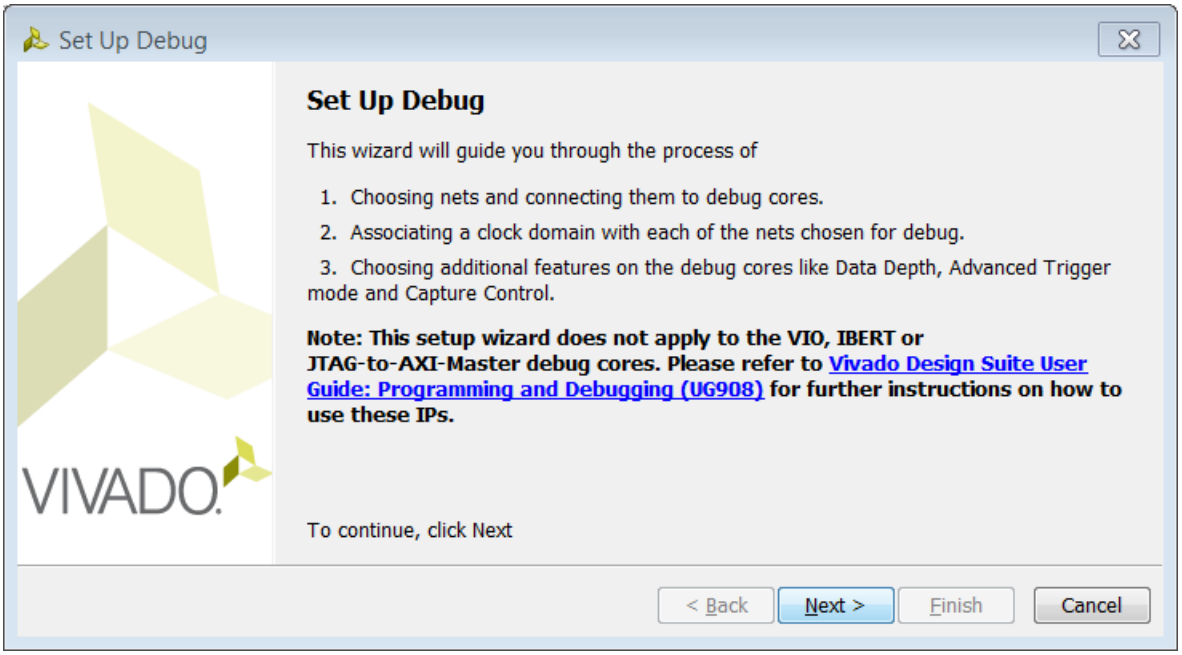

Figure 21: Set Up Debug Wizard

On the second panel of the wizard, notice that some of the nets in the table do not belong to a clock domain (FIGURE 22).

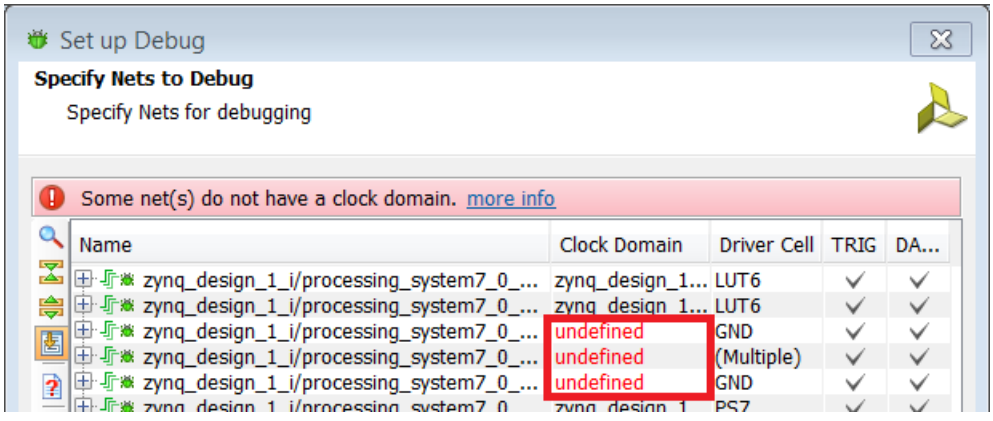

Figure 22: Set Up Debug

IMPORTANT: Signals captured by the same ILA core must have the same clock domain selection.

3. Highlight the three signals that don't have the clock domain association, then right-click and select **Select Clock Domain** (FIGURE 23).

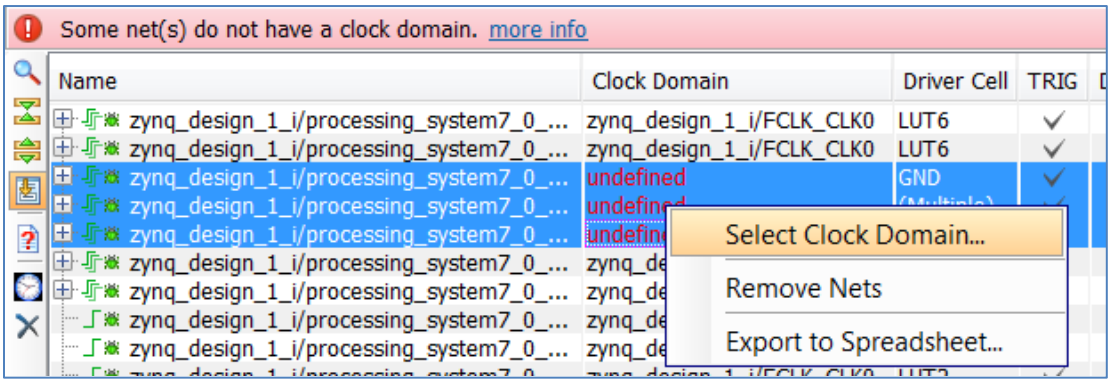

Figure 23: Set Up Debug: Select Clock Domain

4. In the Select Clock Domain window, make sure that the selected filter is **GLOBAL CLOCK** and the **Search hierarchically** check box is checked.
5. Select the `zynq_design_1_i/processing_system7_0/FCLK_CLK0` net as the clock domain (FIGURE 24).

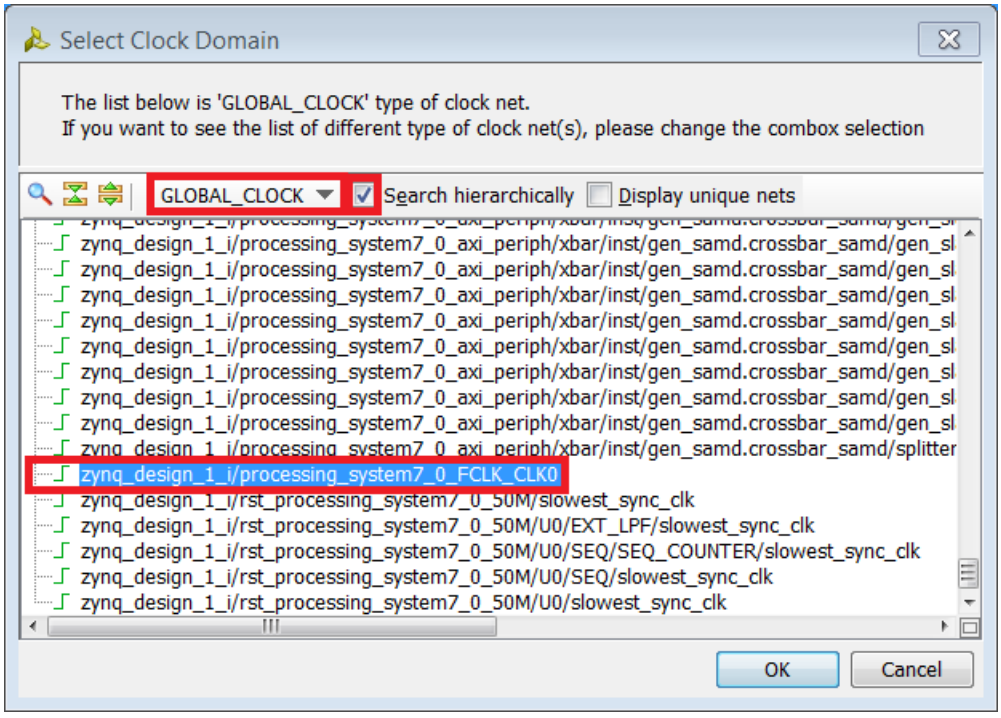

Figure 24: Select Clock Domain Option

6. Click **OK**.

The Specify Nets to Debug dialog box updates with the assigned clock domains.

7. Click **Next**.
8. In the Set up Debug dialog box, click **Next** to open the ILA General Options page.

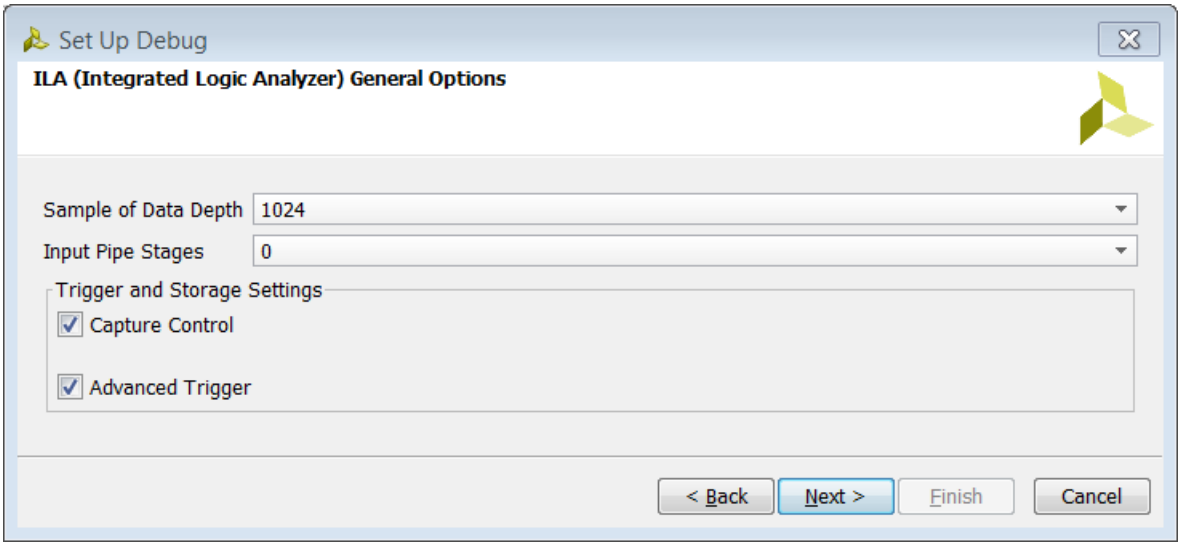

Figure 25: Setting Trigger and Capture Modes

9. Select both the **Capture Control** and **Advanced Trigger** check boxes.
10. Click **Next**.

    The Set up Debug Summary page appears.

11. Verify all the information and click **Finish**.

    Note: It takes approximately 30 seconds for Vivado to add the ILA and Debug Hub cores as black boxes into the synthesized design. During this time, you see several Tcl commands executing in the Tcl Console.

Step 7: Implement Design and Generate Bitstream

1. In Flow Navigator, under Program and Debug, click **Generate Bitstream** to implement the design and generate a BIT file.

   The Save Project dialog box opens.

2. Click **Save**.

   The No Implementation Results Available dialog box opens.

3. Click **Yes**.

   During implementation flow, messages in the Log window show the implementation of the debug cores.

   This step is required to synthesize the debug core modules so that they can replace the debug core black boxes that you added to the design previously (FIGURE 26).

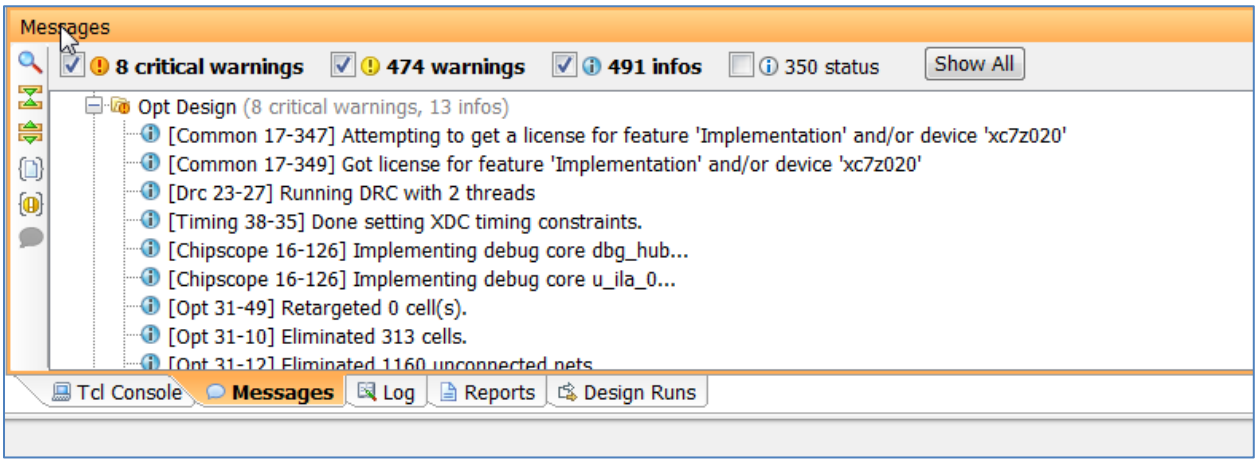

Figure 26: Messages

   After the debug cores are implemented, the rest of the implementation flow (commands such as `opt_design`, `place_design`, and `route_design`) follow as usual.

4. Notice that you can view the progress of the flow in the upper right-hand corner of the interface (FIGURE 27).

Figure 27: Status Indicator

After the bitstream generates, in the Bitstream Generation Completed dialog box that opens, **Open Implemented Design** is selected by default (FIGURE 28).

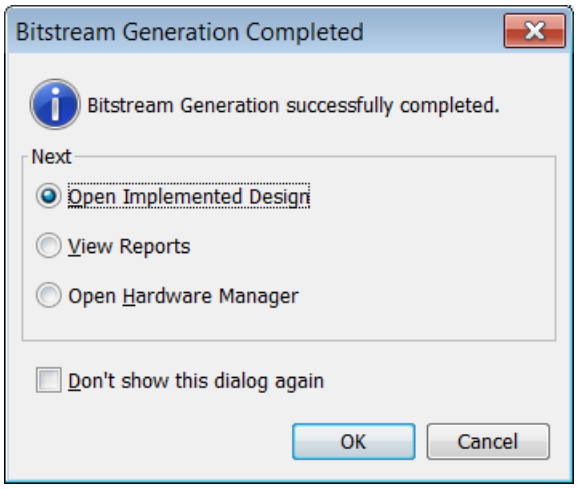

Figure 28: Bitstream Generation Completed

5. Click **OK**.
6. The Vivado dialog box asks if you want to keep the synthesized design open. You can keep the synthesized design open if you want to debug more signals; otherwise close the synthesized design to save memory.
7. In the implemented design, go to the **Netlist** window to see the inserted ILA and Debug Hub (`dbg_hub`) cores in the design (FIGURE 29).

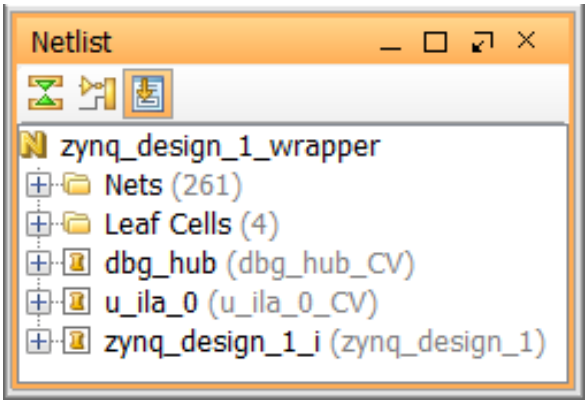

Figure 29: Implemented Design

8. In the Flow Navigator, under Implementation, click **Report Timing Summary** to generate a report that will show you whether all timing constraints were met.
9. In the Report Timing Summary dialog box, click **OK**.

Step 8: Export Hardware to SDK

In this step, you export the hardware description to SDK. You will use this hardware description in Lab 2.

***IMPORTANT**: For the Digilent driver to install, you must power on and connect the board to the host PC before launching SDK.*

1. From the main Vivado File menu, select **File > Export > Export Hardware**.

   The Export Hardware dialog box opens.

   Ensure that the **Include Bitstream** check box is checked and that the **Export to** field is set to the default option of **<Local to Project>** (FIGURE 30).

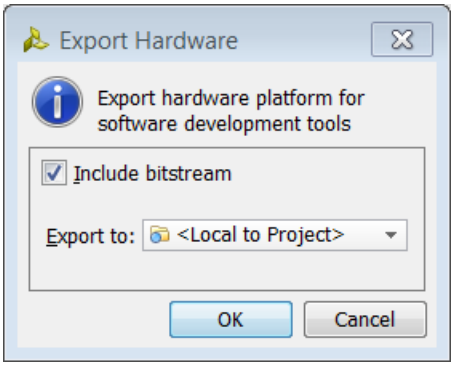

Figure 30: Export Hardware for SDK

2. Click **OK**.
3. To launch SDK, select **File > Launch SDK**.

   The Launch SDK dialog box opens.

4. Accept the default selections for **Exported location** and **Workspace**.

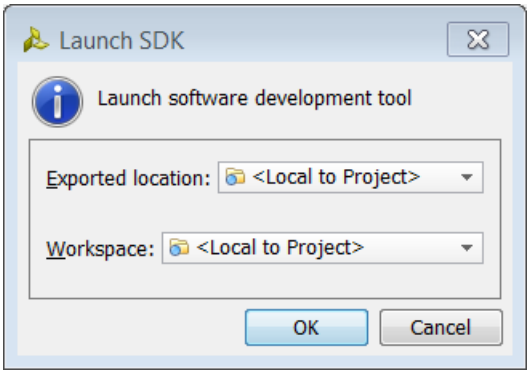

Figure 31: Launch SDK Dialog Box

5. Click **OK**.

Conclusion

In this lab, you:

- Created a Vivado project that includes a Zynq processor design using the IP integrator tool.
- Instantiated IP in the IP integrator tool and made the necessary connections utilizing the Design Automation and Connection Automation features.
- Marked nets for debug, to analyze them in the Vivado Integrated Logic Analyzer.
- Inserted debug probes in the design to debug it later in the Vivado Integrated Logic Analyzer.
- Synthesized, implemented, and generated the bitstream before exporting the hardware definition to SDK.

Lab Files

You can use the Tcl file Lab1.tcl that is included with this tutorial design files to perform all the steps in this lab.

To use the Tcl script, launch Vivado and type **`source Lab1.tcl`** in the Tcl console.

Alternatively, you can also run the script in the batch mode by typing **`Vivado -mode batch -source Lab1.tcl`** at the command prompt.

> ***Note***: *You must modify the project path in the lab1.tcl file in order to source the Tcl files correctly.*

Introduction

You can run this lab after Lab 1. Make sure that you followed all the steps in Lab 1 before proceeding.

Step 1: Start SDK and Create a Software Application

1. If you are doing this lab as a continuation of Lab 1 then SDK should have launched in a separate window. You can also start SDK from the Windows Start menu by clicking on **Start > All Programs > Xilinx Design Tools > Vivado 2014.2 > Xilinx SDK 2014.2**.

   When starting SDK in this manner you need to ensure that you in the correct workspace.

2. You can do that by clicking on **File > Switch Workspace > Other** in SDK. In the Workspace Launcher dialog box in the Workspace field, point to the SDK_Export folder where you had exported your hardware from lab 1. Usually, this is located at `..\project_name\project_name.sdk\SDK\SDK_Export`.

   Now you can create a software application.

3. Select **File > New > Application Project**.

   The New Project dialog box opens.

4. In the Project Name field, type **Zynq_Design** (FIGURE 32).

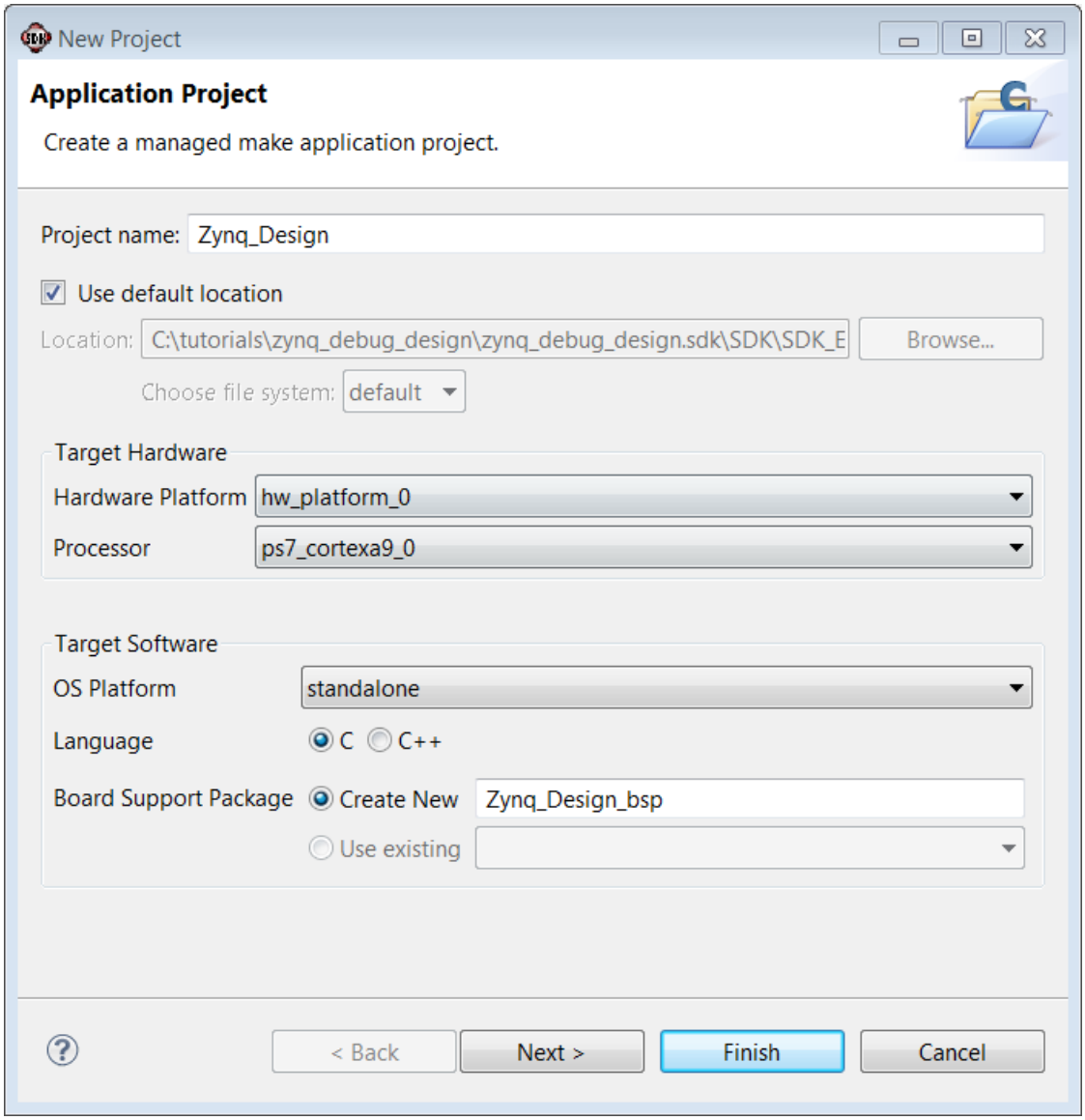

Figure 32: SDK Application Project

5. Click **Next**.
6. From the Available Templates, select **Peripheral Tests** (FIGURE 33).

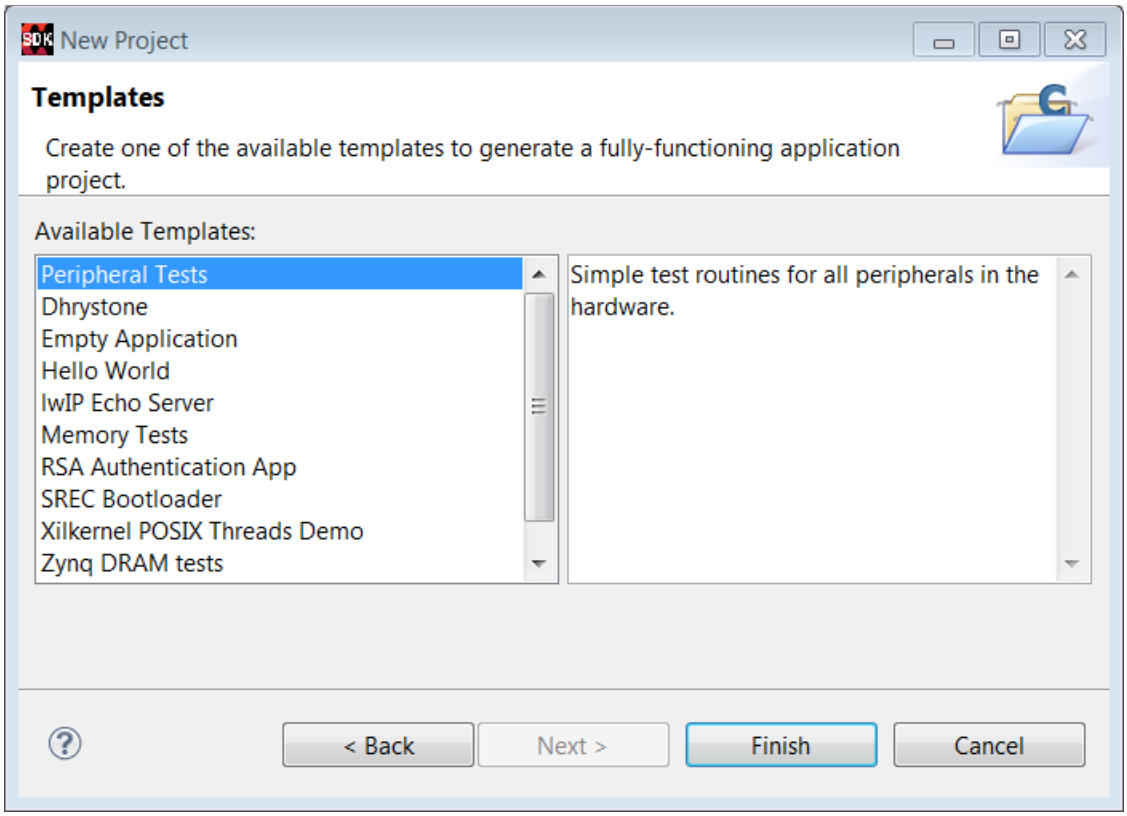

Figure 33: SDK New Project Template

7. Click **Finish**.

   When the program finishes compiling, you see the following in the Console window (**FIGURE 34**).

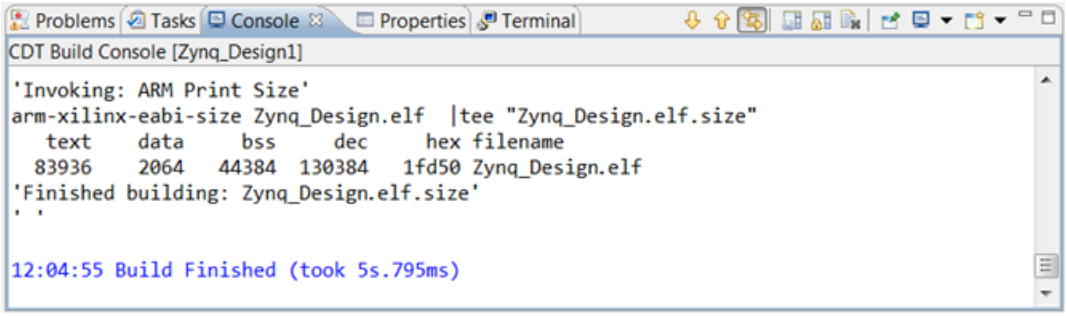

Figure 34: SDK Message

Step 2: Run the Software Application

Now, you must run the peripheral test application on the ZC702 board. To do so, you need to configure the JTAG port. Make sure that your hardware is powered on and a Digilent Cable is connected to the host PC. Also, ensure that you have a USB cable connected to the UART port of the ZC702 board.

1. Download the bitstream into the FPGA by selecting **Xilinx Tools > Program FPGA**.

   The Program FPGA dialog box opens.

2. Ensure that the **Bitstream** field shows the bitstream file that you created in Step 7 of Lab 1, and then click **Program**.

   Note: *The DONE LED on the board turns green if the programming is successful.*

3. Select and right-click the **Zynq_Design** application.
4. Select **Debug As > Debug Configurations**.
5. In the Debug Configurations dialog box, right-click **Xilinx C/C++ application (GDB)** and select **New**.
6. In the Debug Configurations dialog box, click **Debug**.

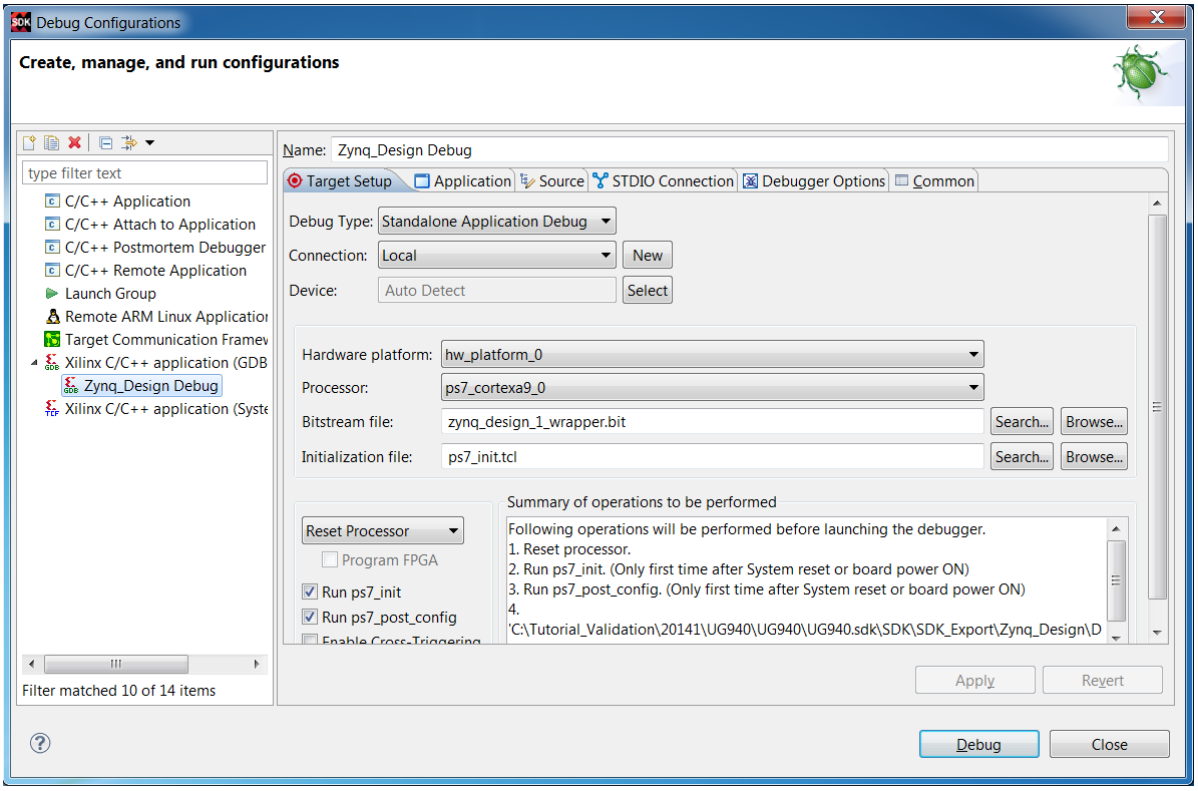

Figure 35: Run Debug Configurations

The Confirm Perspective Switch dialog box opens.

7. Click **Yes**.
8. Set the terminal by selecting the **Terminal** tab and clicking the **Settings** button .
9. Use the following settings for the ZC702 board (FIGURE 36).

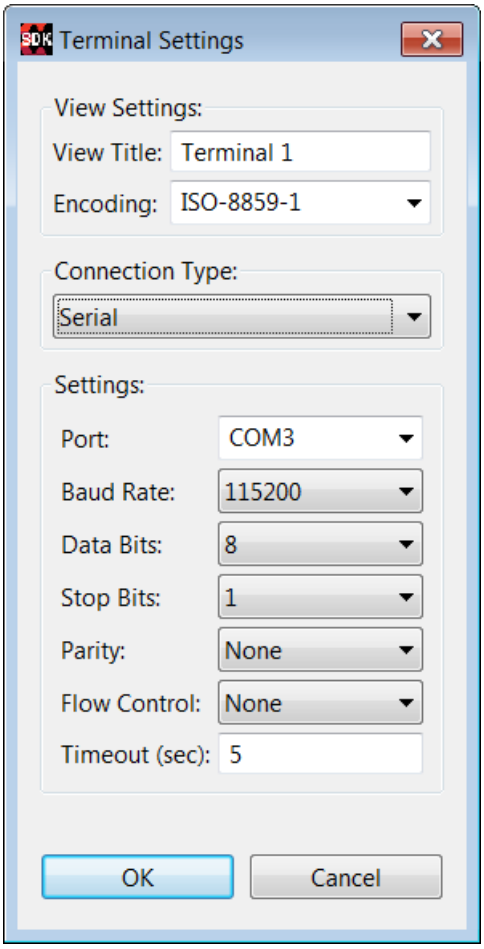

Figure 36: Terminal Settings for ZC702 Board

10. Click **OK**.
11. Verify the **Terminal** connection by checking the status at the top of the tab (FIGURE 37).

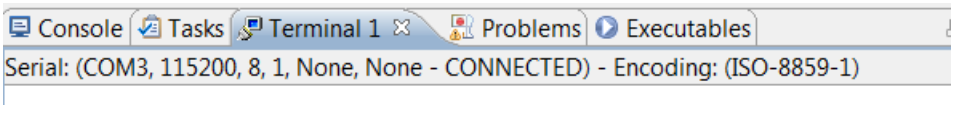

Figure 37: Terminal Connection Verification

12. In the **Debug** tab, expand the tree to see the processor core on which the program is running (FIGURE 38).

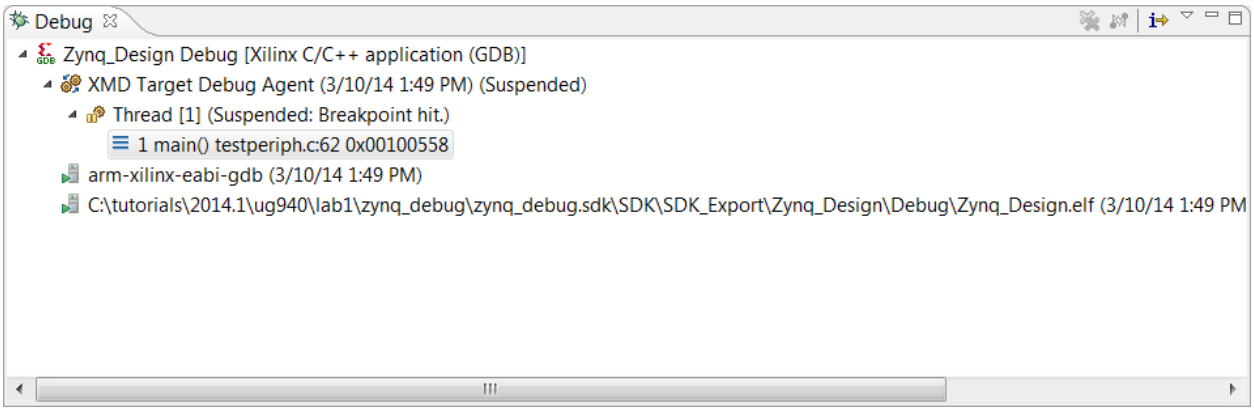

Figure 38: Processor Core to Debug

13. If `testperiph.c` is not already open, select `../src/testperiph.c`, then double-click it to open it.

Add a Breakpoint

You add a breakpoint on line 98.

1. Select **Navigate > Go To Line**.
2. In the Go To Line dialog box, type **98** and click **OK**.

TIP: If line numbers are not visible, right-click in the blue bar on the left side of the window and select **Show Line Numbers**.

3. Double click in the blue bar to the left of line 103 to add a breakpoint on that line of source code (FIGURE 39).

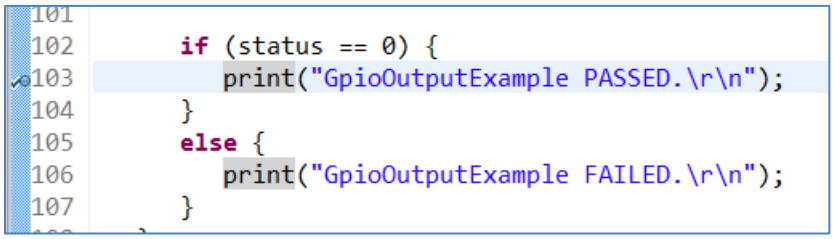

Figure 39: Add a Breakpoint

Step 3: Connect to the Vivado Logic Analyzer

Connect to the ZC702 board using the Vivado® logic analyzer.

1. Open the Vivado project from Lab 1 if not already open.
2. In Vivado, select **Flow Navigator** > **Program and Debug** > **Open Hardware Manager**.
3. In the Hardware Manager window, click **Open a new hardware target** to open a connection to the Digilent JTAG cable for ZC702 (**FIGURE 40**).

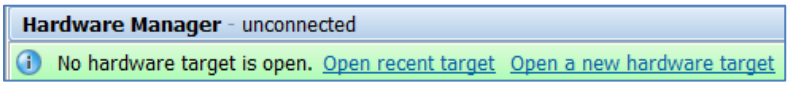

Figure 40: Launch Open New Hardware Target Wizard

The Open New Hardware Target dialog box opens.

4. Click **Next**.

The server name should be already populated by default to **Local server** (**FIGURE 41**).

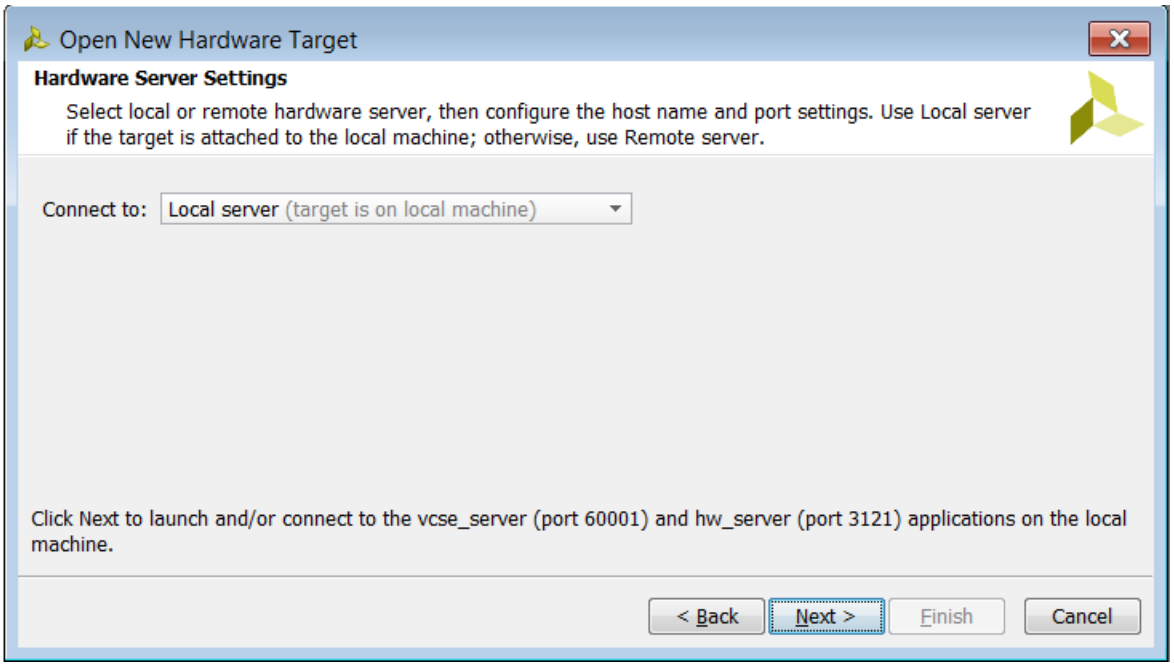

Figure 41: Open New Hardware Target

5. Click **Next.**

You now see the `xilinx_tcf` hardware target and the `arm_dap_0` and `xc7z020_1` hardware devices as shown in **FIGURE 42**.

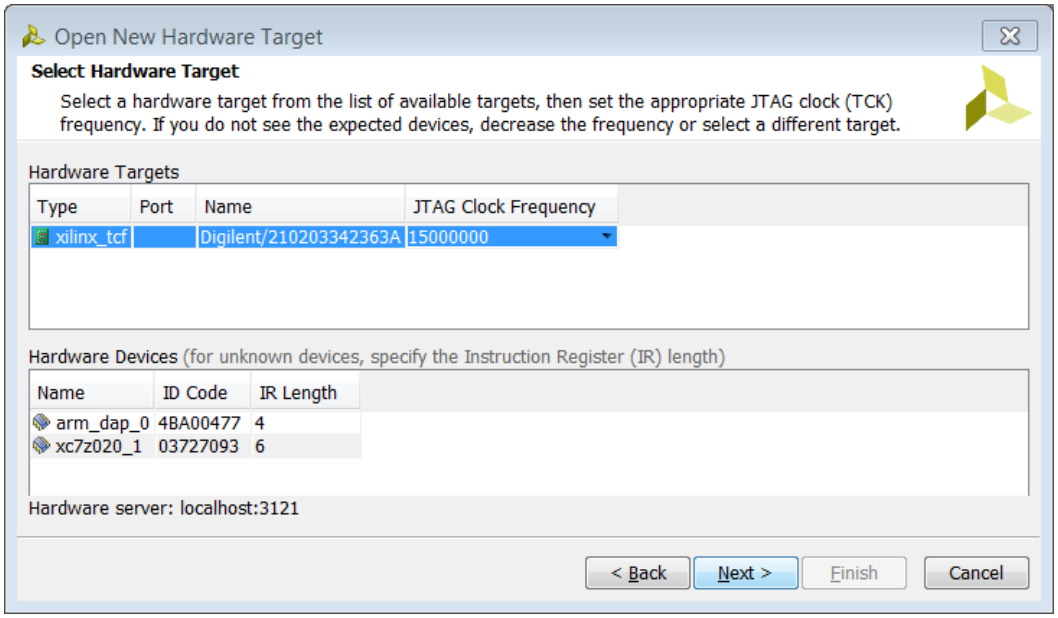

Figure 42: Select Target Hardware

6. Click **Next**.

   The Open Hardware Target Summary dialog box displays. (**FIGURE 43**).

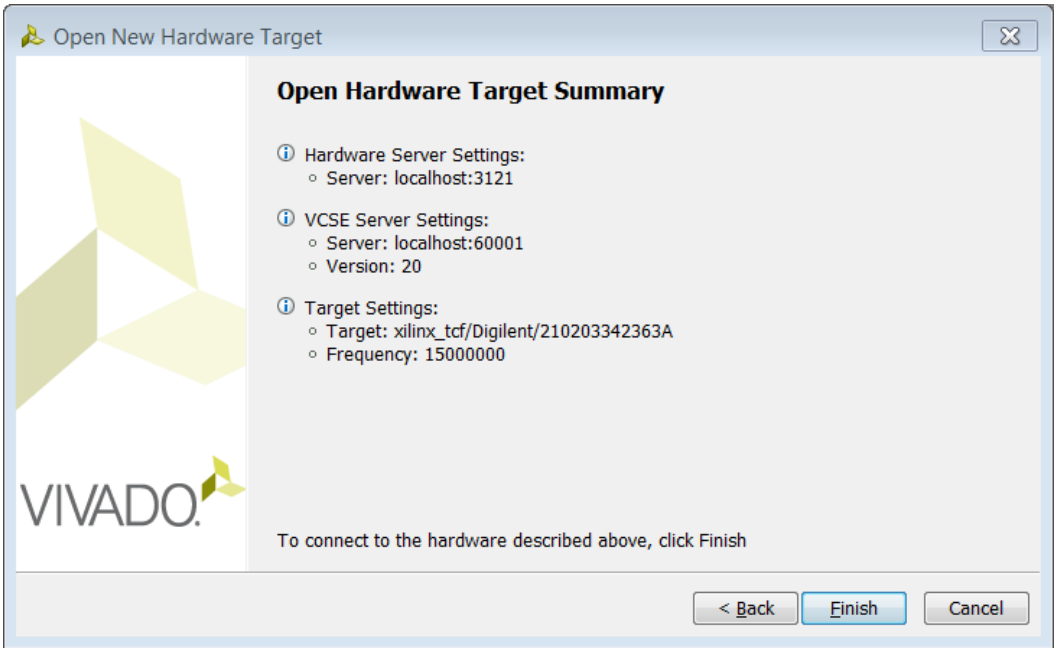

Figure 43: Open Hardware Target Summary

7. Click **Finish** to connect to the target.

   When the Vivado hardware session successfully connects to the ZC702 board, the Hardware window shows the following information (FIGURE 44).

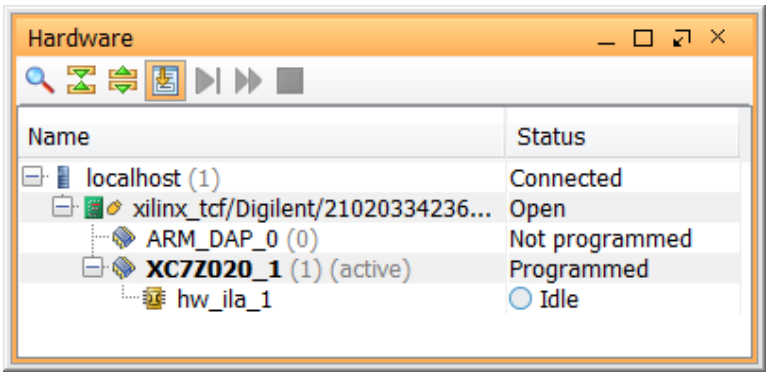

Figure 44: Successfully Programmed Hardware Session

8. First, ensure that the ILA core is alive and capturing data. To do this, select the `hw_ila_1 core` in the Hardware window.

9. On the Hardware window toolbar, click the **Run Trigger Immediate** button.

   At this point, you should see static data from the ILA core in the waveform window (FIGURE 45).

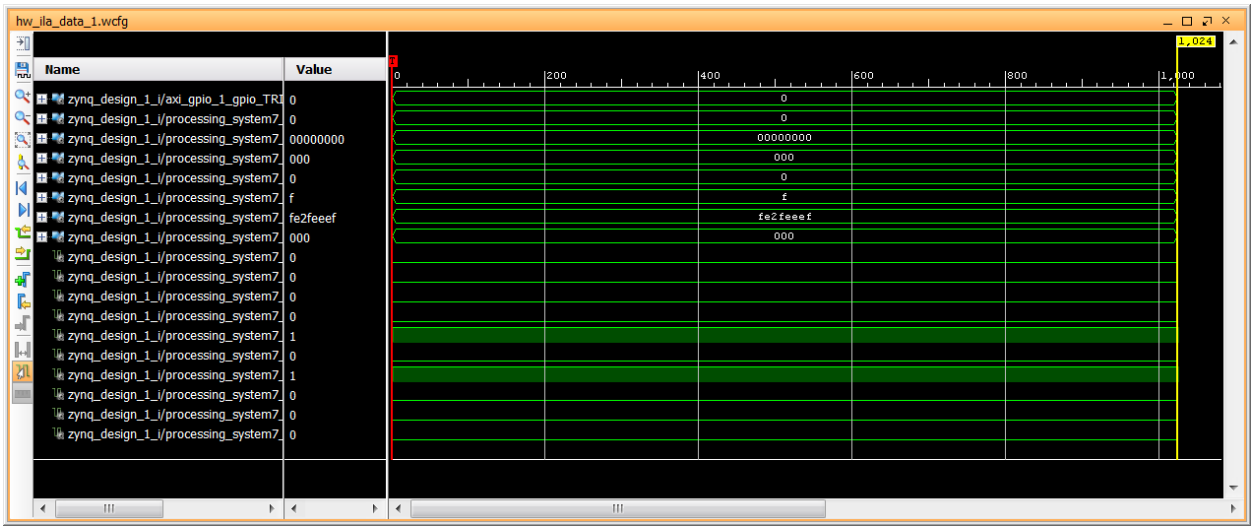

Figure 45: Static Data from Set Breakpoint

10. Set up a condition that triggers when the application code writes to the GPIO peripheral. To do this:

    a. Select, drag and drop the `/zynq_design_1_i/processing_system7_0_axi_periph_M00_AXI_AWVALID` signal from the Debug Probes window into the Basic Trigger Setup area.

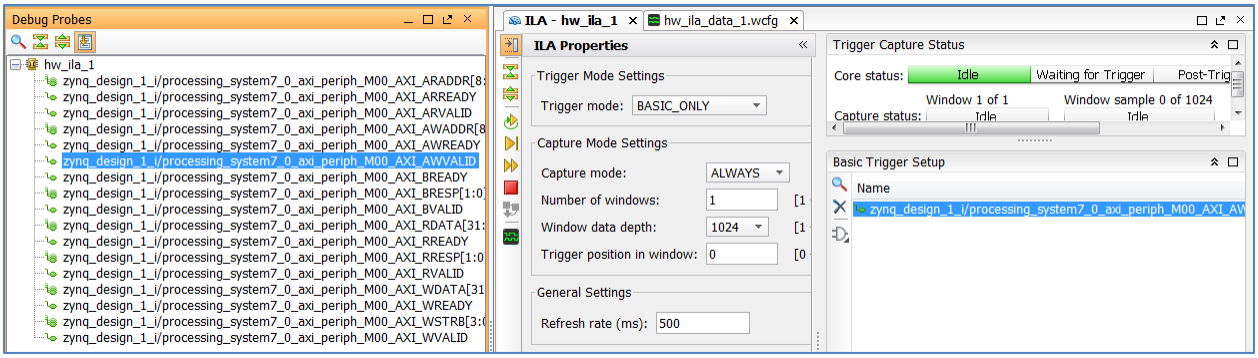

Figure 46: The ILA Properties window

    b. Click the **Compare Value** column of the ***WVALID** row, as shown in [FIGURE 47](#).

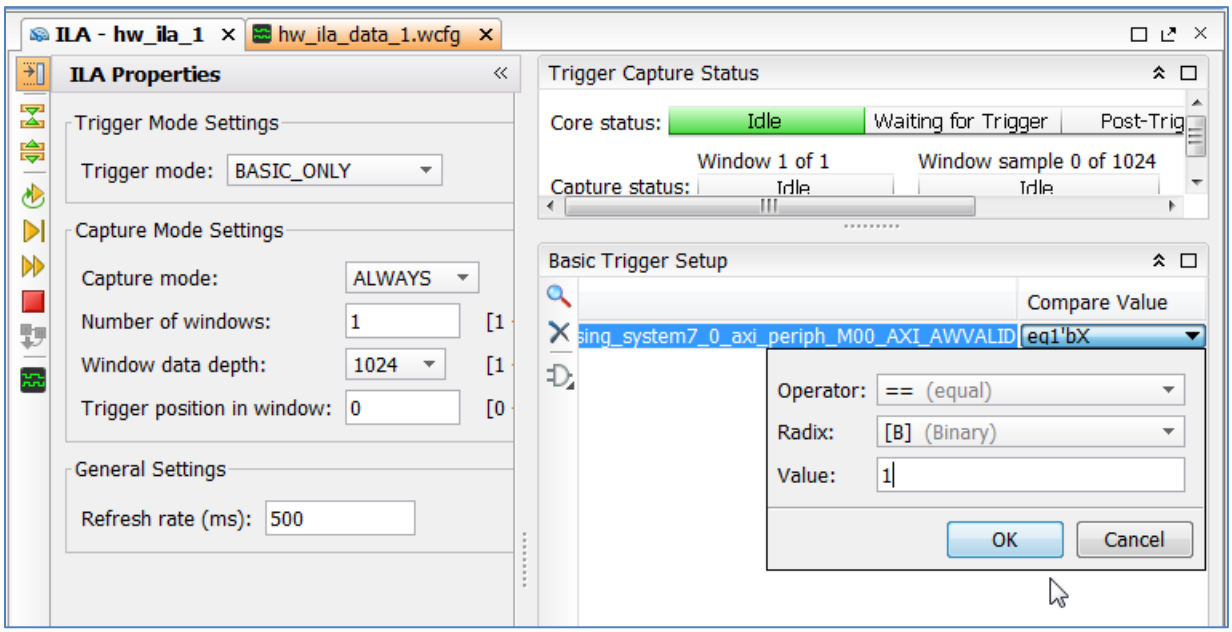

Figure 47: ILA Properties Window

    c. Change the value from an X (don't care) to a **1**, and click **OK**. You also want to see several samples of the captured data before and after the trigger condition.

11. Change the trigger position to the middle of the 1024 sample window by setting the **Trigger Position** for the `hw_ial_1` core in the ILA Properties window to **512** and then pressing **Enter** (FIGURE 48).

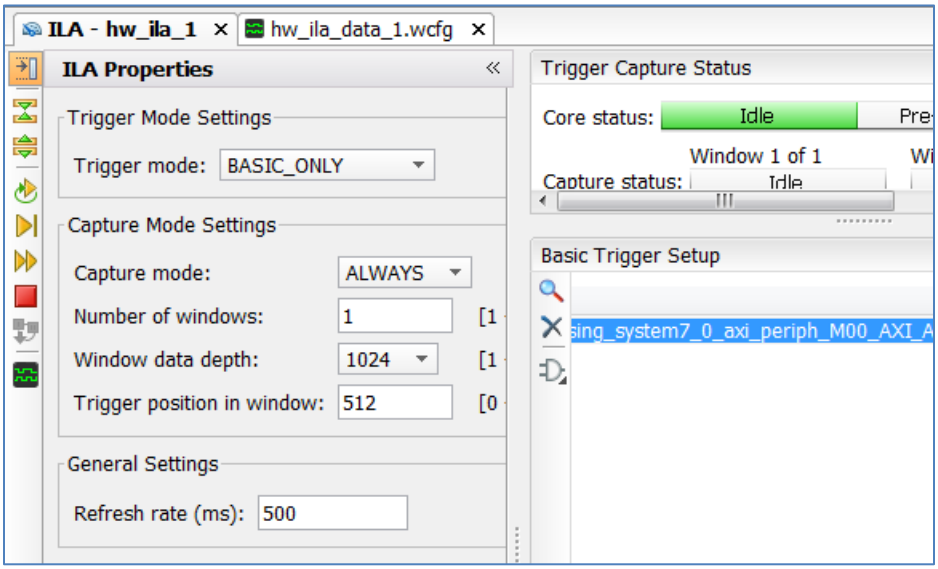

Figure 48: Change Debug Probe Settings

After setting up the compare value and the trigger position, you can arm the ILA core.

12. In the Hardware window, select the `hw_ila_1` core, and then click the **Run Trigger** button.

    Alternatively, you can arm the ILA core directly from the ILA Properties window by clicking the **Run Trigger** button.

    Notice that the Status column of the **hw_ila_1** ILA core changes from Idle to **Waiting for Trigger**. Likewise, the ILA Properties window shows the Core Status as **Waiting for Trigger**. as shown in FIGURE 49.

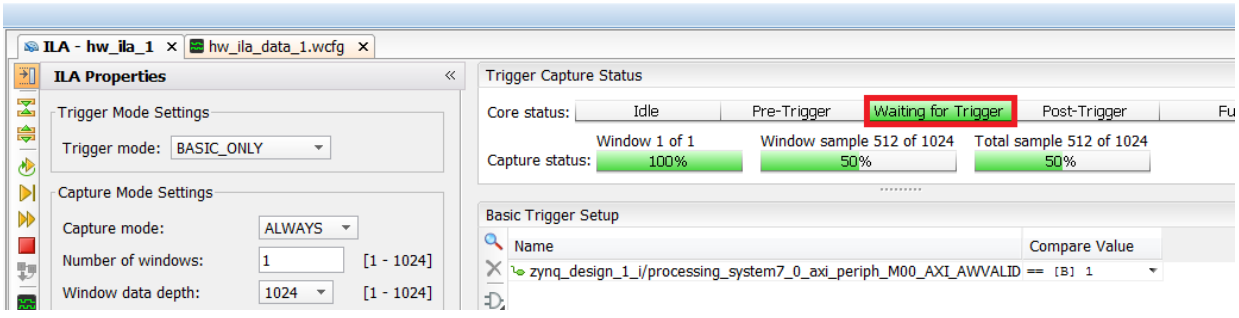

Figure 49: Status of hw_ila_1

13. Go back to SDK and continue to execute code. To do this, click the **Resume** button on the SDK toolbar.

    Alternatively, you can press **F8** to resume code execution.

    The code execution stops at the breakpoint you set on line 98.

14. Use the **Step Over** button to run past the `GpioOutputExample()` function.

    This causes at least one write operation to the GPIO peripheral. These write operations cause the `WVALID` signal to go from 0 to 1, thereby triggering the ILA core.

    As you step over the following line of code, you see the following transaction captured by looking at the waveform window in the Vivado logic analyzer feature:

    ```
    status = GpioOutputExample(XPAR_AXI_GPIO_1_DEVICE_ID,4);
    ```

    ***Note**: The trigger mark occurs at the first occurrence of the `WVALID` signal going to a 1 (FIGURE 50).*

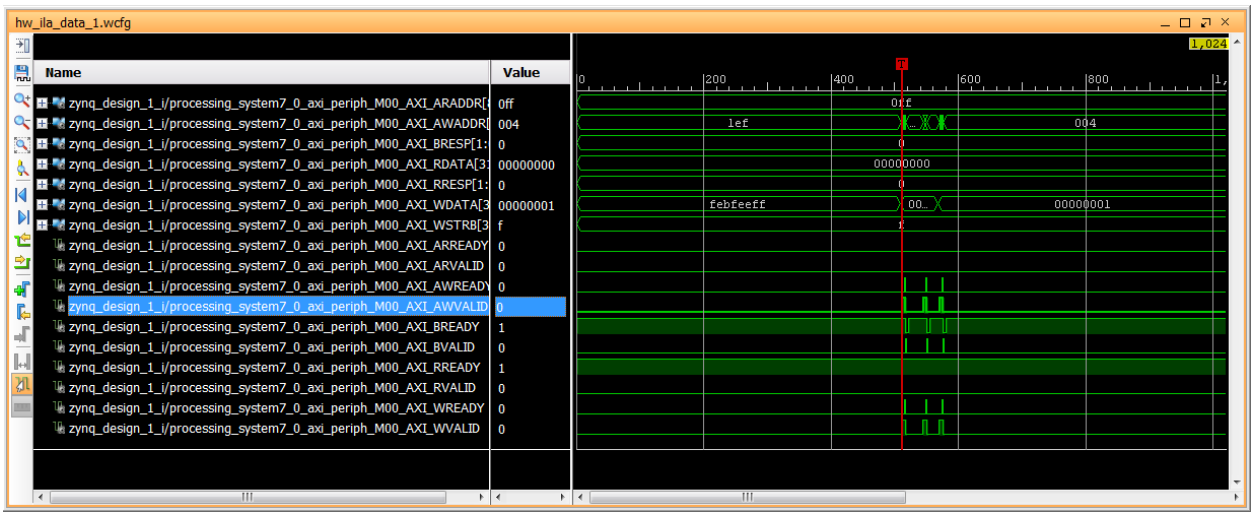

Figure 50: Trigger Mark Goes to 1

15. If you are going on to Lab 3, close your project by selecting **File > Close Project**.

Conclusion

This lab introduced you to software development in SDK and executing the code on the Zynq-7000 processor.

This lab also introduced you to Vivado Logic Analyzer and analyzing the nets that were marked for debug in Lab 1 and cross-probing between hardware and software

> ***Note:** Tcl files are not provided for this lab as the intent is for the user to see the power of cross-probing between the hardware and software in the GUI environment.*

Introduction

In this lab, you use the cross-trigger functionality between the Zynq processor and the fabric logic. Cross triggering is a powerful feature that you can use to co-debug software running in real time on the target hardware. This tutorial guides you from design creation in IP integrator, to marking the nets for debug and manipulating the design to stitch up the cross-trigger functionality.

Step 1: Start the Vivado IDE and Create a Project

1. Start the Vivado® IDE by clicking the Vivado desktop icon or by typing **vivado** at a command prompt.
2. From the Quick Start page, select **Create New Project**.
3. In the New Project dialog box, use the following settings:
   a. In the **Project Name** dialog box, type the project name and location. For this project, use the name zynq_x_trigger.
   b. Make sure that the **Create project subdirectory** check box is checked. Click **Next**.
   c. In the **Project Type** dialog box, select **RTL project**. Click **Next**.
   d. In the **Add Sources** dialog box, set the **Target language** to either **VHDL** or **Verilog**. You can leave the **Simulator language** selection to **Mixed**.
   e. Click **Next**.
   f. In **Add Existing IP** dialog box, click **Next**.
   g. In **Add Constraints** dialog box, click **Next**.
   h. In the **Default Part** dialog box, select **Boards** and choose **ZYNQ-7 ZC702 Evaluation Board** that matches the version of hardware that you have. Click **Next**.
4. Review the project summary in the **New Project Summary** dialog box before clicking **Finish** to create the project.

Step 2: Create an IP Integrator Design

1. In Vivado Flow Navigator, click **Create Block Design**.
2. In the **Create Block Design** dialog box, specify the name of the IP integrator design. For this tutorial, use `zynq_processor_system`.
3. Leave the Directory field set to its default value of <Local to Project> and click **OK**.

   The IP integrator diagram opens.
4. Click **Add IP** at the top of the diagram area (FIGURE 51).

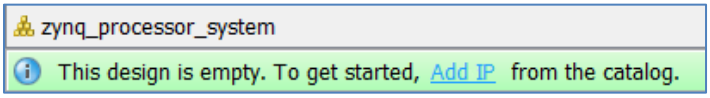

Figure 51: Add IP to the Design

   The IP catalog opens.
5. In the Search field, type `Zynq` and select the **ZYNQ7 Processing System** IP and press **Enter**.

   Alternatively, double-click the **ZYNQ7 Processing System** IP to instantiate it (FIGURE 52).

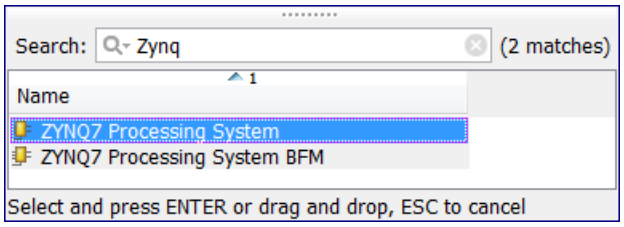

Figure 52: Instantiate Zynq Processing System

6. In the header at the top of the diagram, click **Run Block Automation** and select **/processing_system7_0** (FIGURE 53).

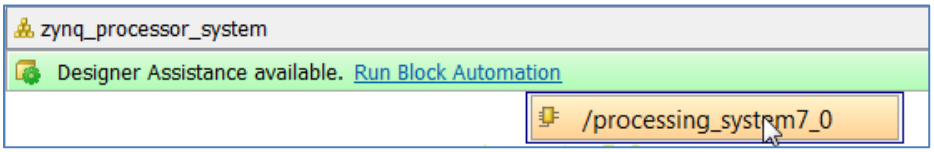

Figure 53: Run Block Automation on Zynq Processing System

   The Run Block Automation dialog box states that the FIXED_IO and the DDR pins on the ZYNQ7 Processing System 7 IP will be connected to external interface ports. Also, because you chose the ZC702 board as your target board, the **Apply Board Preset** checkbox is checked by default.

7. Enable the Cross Trigger In and Cross Trigger Out functionality by setting those fields to **New ILA**, then click **OK** (FIGURE 54).

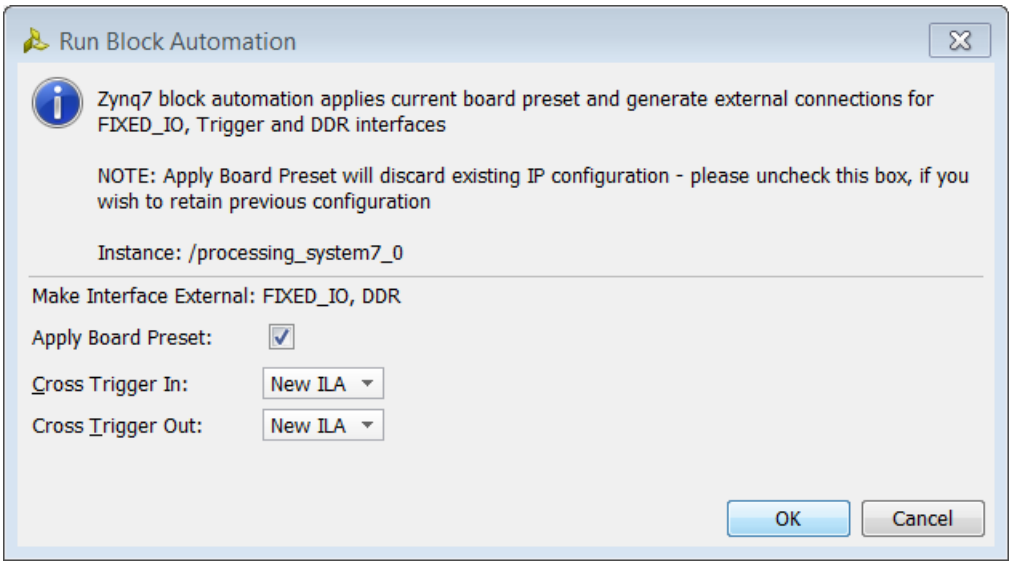

Figure 54: Run Block Automation Dialog Box

This instantiates an ILA core in the design and also connects the TRIGGER_IN_0 and TRIGGER_OUT_0 interface pins of the PS7 to the ILA. The automation also connects the DDR and FIXED_IO interface pins to external I/O ports (FIGURE 54).

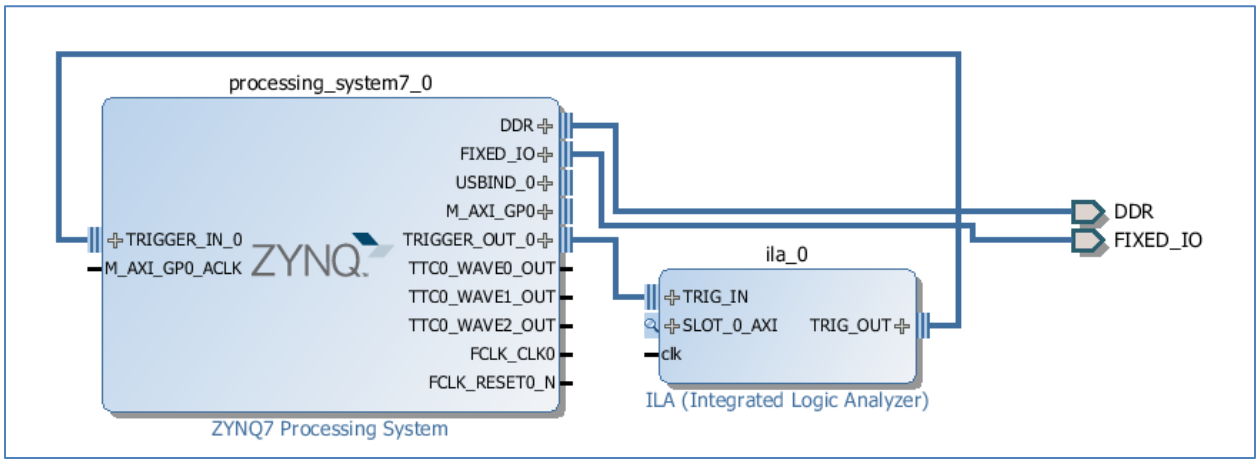

Figure 55: IP Integrator Canvas After Running Block Automation

8. Add the AXI GPIO and AXI BRAM Controller to the design by right-clicking anywhere in the diagram and selecting **Add IP**.

   The diagram area should look as follows (**FIGURE 56**).

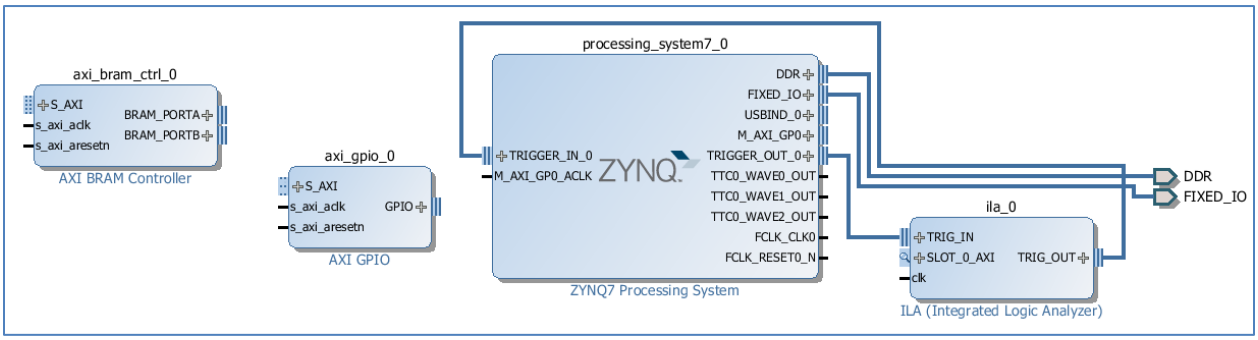

Figure 56: Diagram after Instantiating IP for This Design

9. Make each of the following connections using the **Run Connection Automation** function.

Connection	*More Information*	*Setting*
/axi_gpio_0/S_AXI	The Run Connection Automation dialog box states that the S_AXI pin of the GPIO IP will be connected to M_AXI_GP0 pin of the ZYNQ7 Processing System. It also offers a choice for different clock sources that might be relevant to the design.	**Clock Connection (for unconnected clks)** field: leave it set to **Auto**.
/axi_gpio_0/GPIO	The Run Connection Automation dialog box shows the interfaces that are available on the ZC702 board to connect to the GPIO.	Select **LEDs_4Bits**.
/axi_bram_ctrl_0/S_AXI	The Run Connection Automation dialog box states that the S_AXI port of the AXI BRAM Controller will be connected to the M_AXI_GP0 port of the ZYNQ7 Processing System IP. The AXI BRAM Controller needs to be connected to a Block Memory Generator block. The connection automation feature offers this automation by instantiating the Block Memory Generator IP and making appropriate connections to the AXI BRAM Controller.	Leave the **Clock Connection** field set to **Auto**.

/axi_bram_ctrl_0/ BRAM_PORTA	The Run Connection Automation dialog box informs you that a new Block Memory Generator IP will be instantiated and connected to the AXI BRAM Controller PORTA.	No options.
/axi_bram_ctrl_0/ BRAM_PORTB	Note that the Run Connection Automation dialog box offers two choices now. The first one is to use the existing Block Memory Generator from the previous step or you can chose to instantiate a new Block Memory Generator if desired. In this case we will use the existing BMG	Leave the Blk_Mem_Gen field set to its default value of **/axi_bram_ctrl_0_bram**.

When these connections are complete, the IP integrator design looks like **FIGURE 57**.

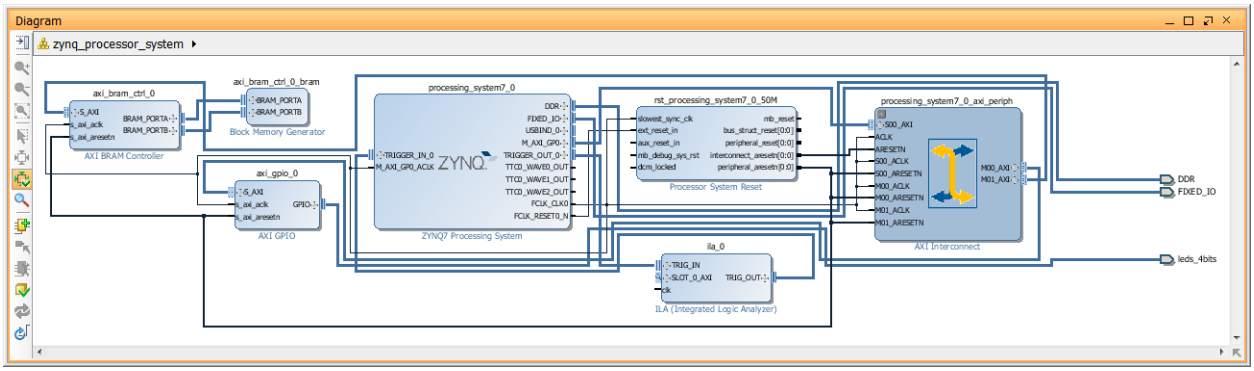

Figure 57: Design after Running Connection Automation

10. Click the **Address Editor** tab of the design to ensure that addresses for the memory-mapped slaves have been assigned properly. Expand **Data** by clicking the **+** sign (**FIGURE 58**).

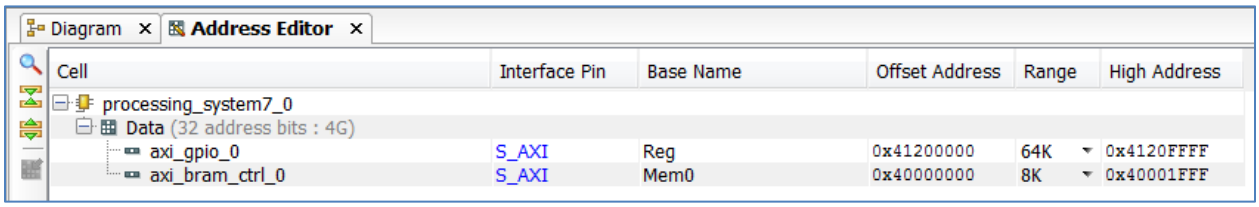

Figure 58: Memory Map the Slaves

11. Now you will mark some nets for debug. Click the **Diagram** tab again and select the net connecting the `gpio` pin of the AXI GPIO IP to the `LEDs_4Bits` port.

12. Right-click in the IP integrator diagram area and select **Mark Debug**. This marks the net for debug. The ILA insertion for debugging this net will be done after the design is synthesized.

    Notice that a bug symbol appears on the net to be debugged. You can also see this bug symbol in the Design Hierarchy window on the selected net.

    Notice also that the SLOT_0_AXI interface port of the ILA is yet to be connected. These interfaces are designed to monitor an AXI interface. We will monitor the net between S_AXI interface of AXI GPIO and M00_AXI interface of the AXI Interconnect (`processing_system7_0_axi_periph`). To monitor this interface, hover the cursor on the SLOT_0_AXI interface port of the ila_0 until it changes into a pencil, and click and drag over to the net to be monitored (S_AXI interface of GPIO).

13. The clk pin of the ILA is not yet connected either. Connect it to the FCLK_CLK0 pin of the ZYNQ7 Processing System.

14. Click the **Regenerate Layout** button to generate an optimal layout of the design. The design should look as follows (**FIGURE 59**).

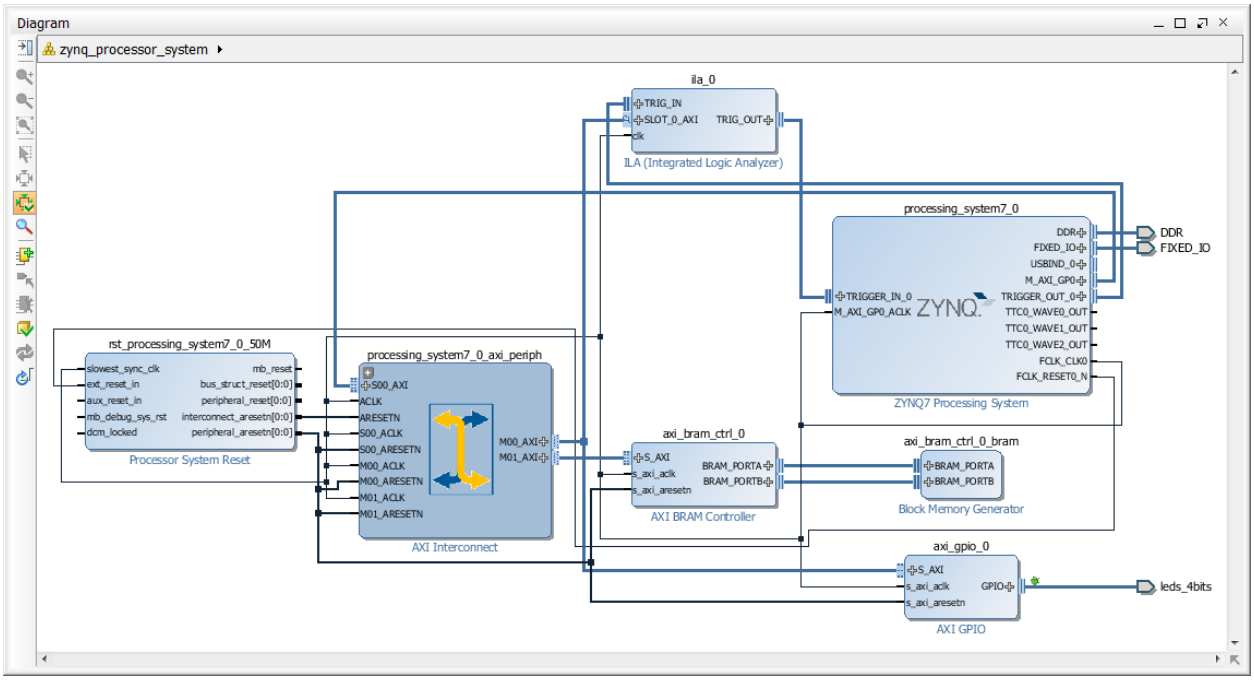

Figure 59: Block Design after Running Regenerate Layout

15. Click the **Validate Design** button to run Design Rule Checks on the design.

    After design validation is complete, the **Validate Design** dialog box opens to verify that there are no errors or critical warnings in the design.

16. Select **File > Save Block Design** to save the IP integrator design.

    Alternatively, press **Ctrl + S** to save the design.

17. In the Sources window, right-click **zynq_processor_system** and select **Generate Output Products**.

    The Generate Output Products dialog box opens (FIGURE 60).

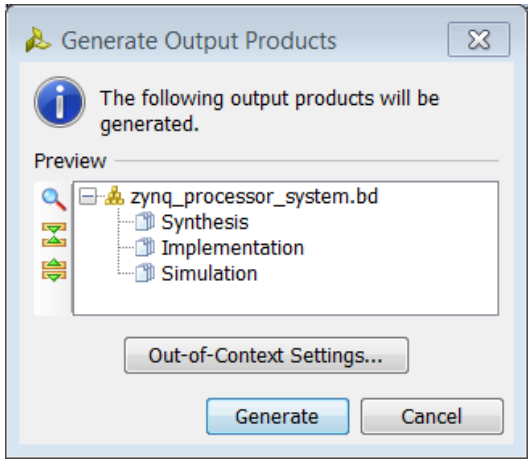

Figure 60: Generate Output Products Dialog Box

18. Click **Generate**.
19. In the Sources window, right-click **zynq_processor_system**, and select **Create HDL Wrapper**.

    The Create HDL Wrapper dialog box offers two choices. The first choice is to generate a wrapper file that you can edit. The second choice is let Vivado generate and manage the wrapper file, meaning it is a read-only file.

20. Keep the default setting and click **OK** (FIGURE 61).

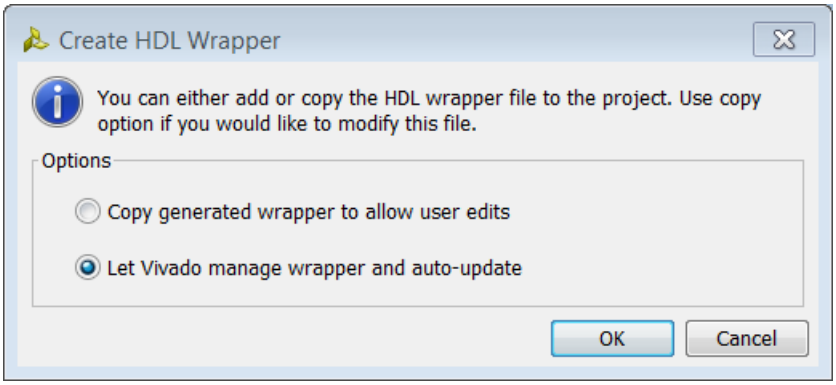

Figure 61: Create HDL Wrapper Dialog Box

Step 3: Synthesize Design

1. From the Flow Navigator, under **Synthesis**, click **Run Synthesis**.
2. When synthesis completes, select **Open Synthesized Design** from the Synthesis Completed dialog box and click **OK** (FIGURE 62).

   You can also click **Synthesis > Open Synthesized Design** in Flow Navigator.

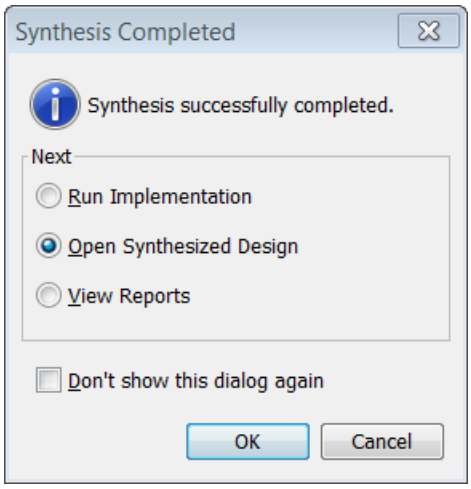

Figure 62: Open Synthesized Design

Step 4: Create an Integrated Logic Analyzer Debug Core

When the synthesized design schematic opens, the Debug window also opens. In the Debug window, notice that there is one ILA, ila_0, that has already been inserted in the design. This ILA was instantiated in the block design and will monitor the AXI transactions to the GPIO. Notice that there are the 4-bits of output leds_4bits that were also marked for debug in the block design. These outputs have not yet been assigned to an ILA. We will insert the ILA to monitor these output bits in the following steps.

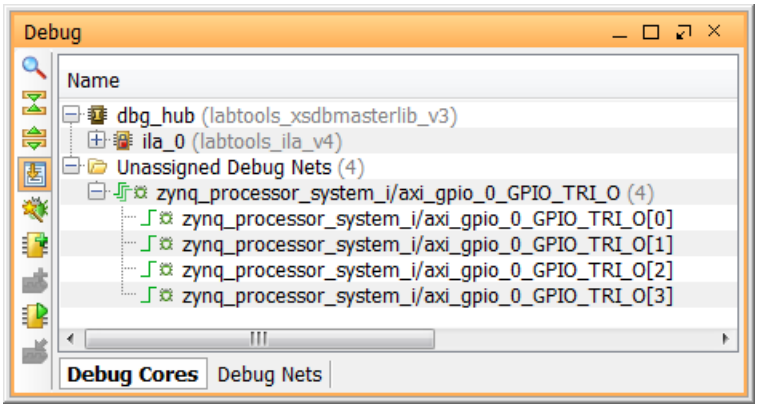

Figure 63: Unassigned Debug Nets

1. From the Vivado main menu, select **Tools** > **Set up Debug**.

   Alternatively, from the left side of the **Debug** window, click the **Set up Debug** button .

   The Set up Debug dialog box opens (FIGURE 64).

2. Click **Next**.

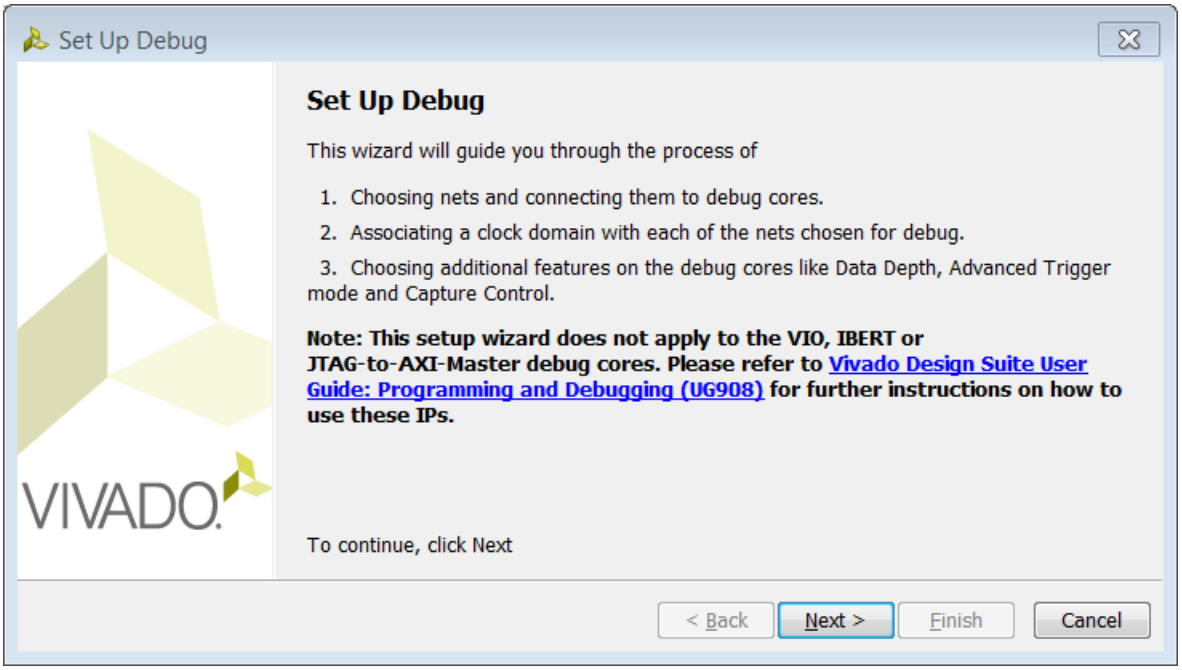

Figure 64: Set Up Debug Dialog Box

3. In the **Specify Nets to Debug** page, select the nets that you are interested in monitoring from the list of nets that are offered (FIGURE 65). Select everything except the following nets:
   - zynq_processor_system_i/axi_gpio_0_GPIO_TRI_O (4)
   - zynq_processor_system_i/processing_system7_0_TRIGGER_OUT_0_ACK
   - zynq_processor_system_i/processing_system7_0_TRIGGER_OUT_0_TRIG
   - zynq_processor_system_i/ila_0_TRIG_OUT_ACK
   - zynq_processor_system_i/ila_0_TRIG_OUT_TRIG

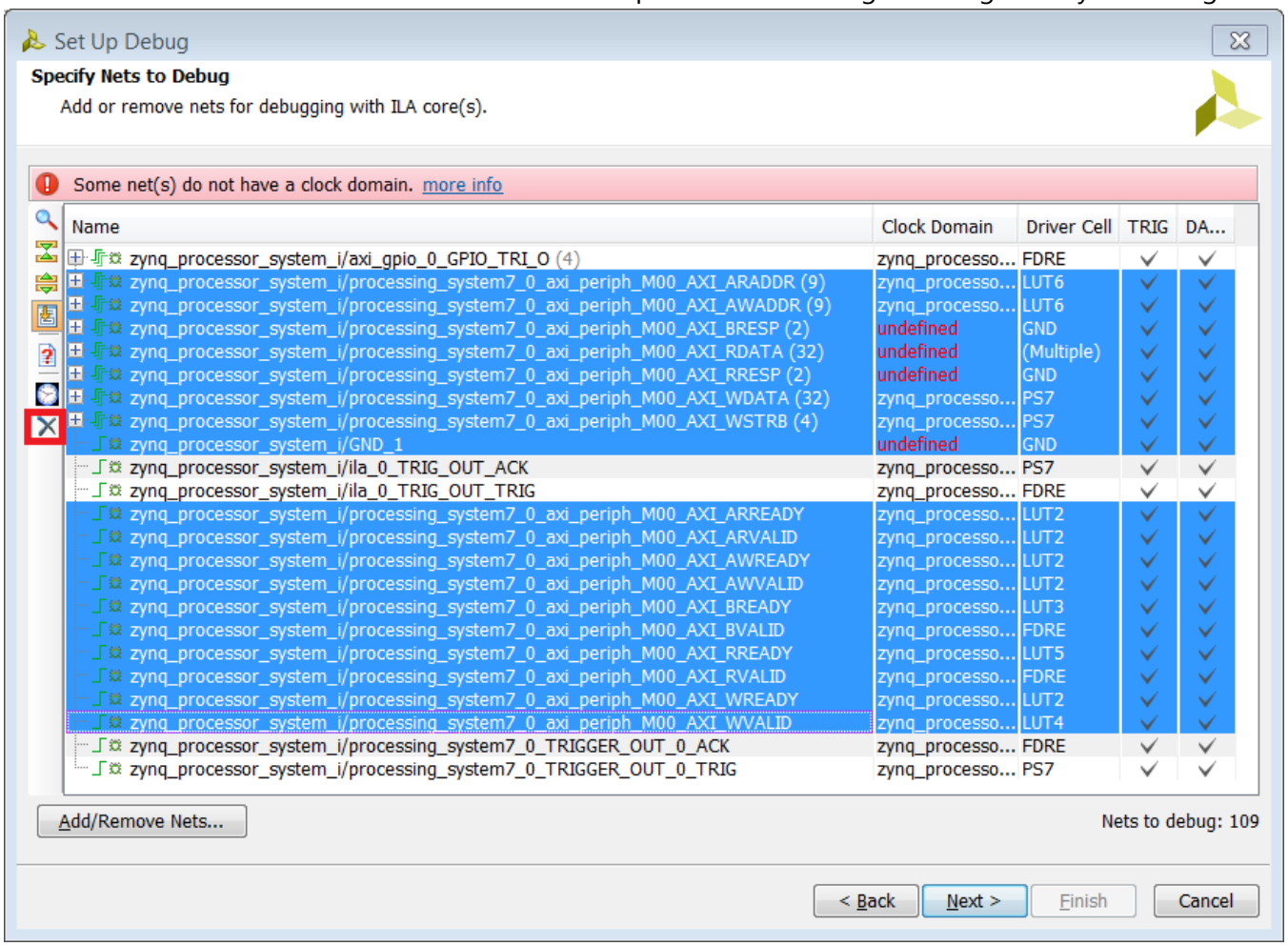

Figure 65: Specify Nets for Debugging

> ***CAUTION!*** You might see a different net name than that shown in Figure 64 for the cross trigger signals after the design is synthesized. Make sure that you select the nets connected to the `trig_in`, `trig_in_ack`, `trig_out`, and `trig_out_ack` pins of the ILA instance. On the Processing System 7 instance the corresponding pins are called `FTMT_F2P_TRIG_0`, `FTMT_F2P_TRIGACK_0`, `FTMT_P2F_TRIG_0`, and `FTMT_P2F_TRIGACK_0`.

4. Click the **Delete** button, highlighted in FIGURE 65.
5. Click **Next**.

The ILA (Integrated Logic Analyzer) General Options dialog box opens (FIGURE 66)

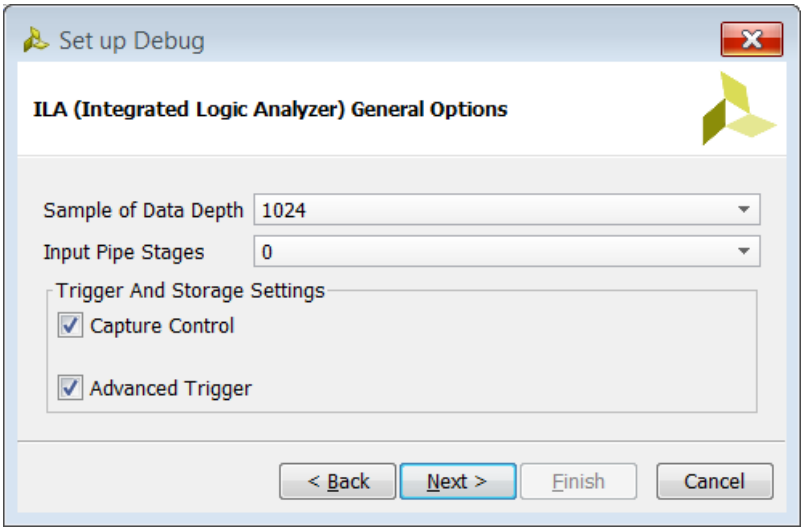

Figure 66: Enable Advanced Trigger and Capture Modes

6. In the **ILA (Integrated Logic Analyzer) General Options** page, do the following:
   a. leave the **Sample of Data Depth** and **Input Pipe Stages** options to their default values of 1024 and 0, respectively.
   b. Select both **Capture Control** and **Advanced Trigger** check boxes.
   c. Click **Next**.
7. Review the Set up Debug Summary dialog box and click **Finish**.

   The debug nets are now assigned to the ILA u_ila_0 debug core, as shown in FIGURE 67. Notice that there are now no nets under the Unassigned Debug Nets folder.

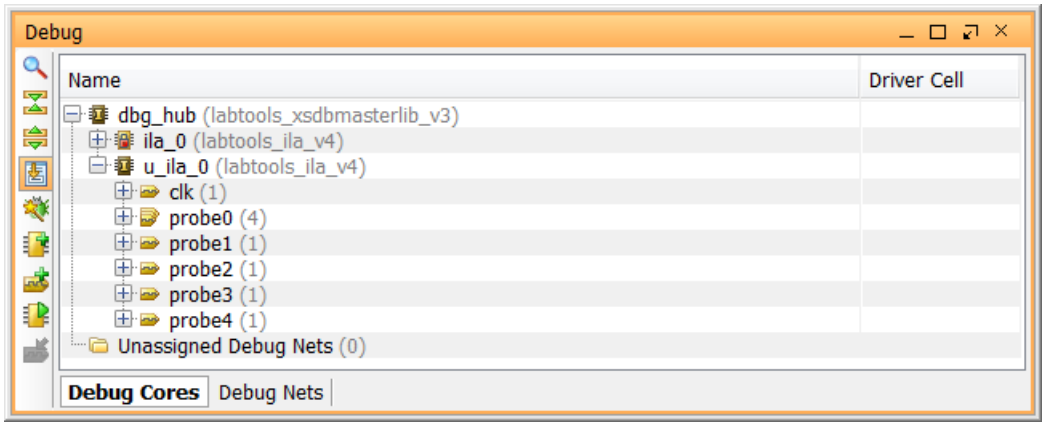

Figure 67: Debug Nets Assigned to Debug Core

Step 5: Verify that the Appropriate Connections Have Been Made in the Netlist Schematic

1. In the Netlist window, select and then right-click **u_ila_0 (u_ila_0_CV)** and select **Schematic**.

   The schematic opens. You can double-click on the different probes of the ILA to see the connections. Verify that all the connections are made appropriately to the endpoints that you are interested in debugging.

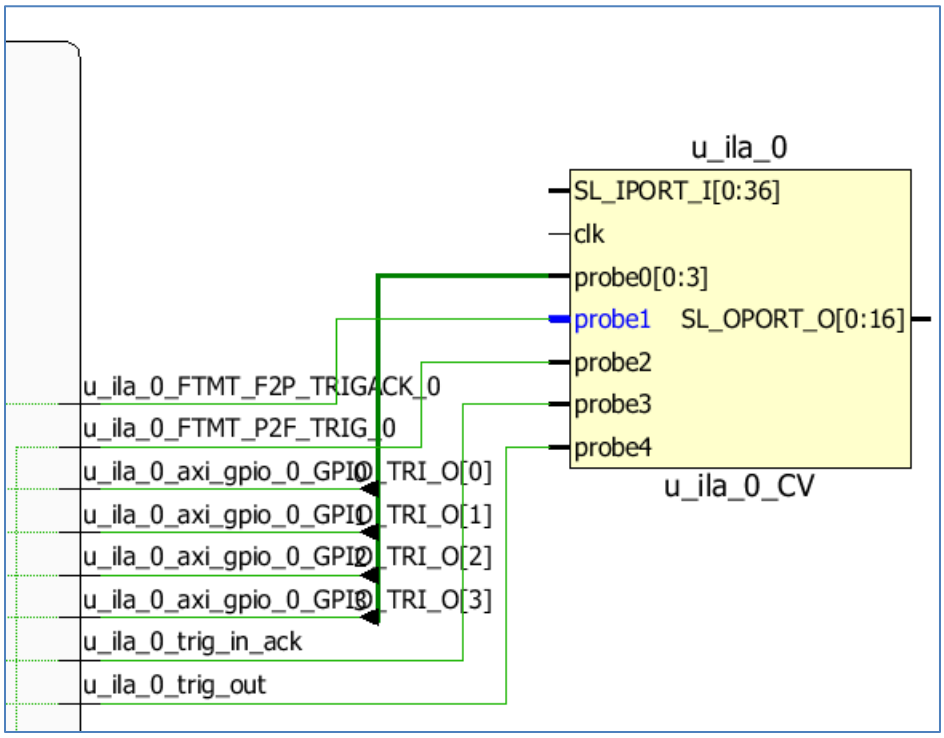

Figure 68: Verify connections in schematic

TIP: Your design might look slightly different than the figure above, depending on the net names of the cross-trigger signals.

Step 6: Implement Design and Generate Bitstream

Now that the cross-trigger signals have been connected to the ILA for monitoring, you can complete the design through the rest of the flow.

1. Click **Generate Bitstream** to generate the bitstream for the design. The **Save Project** dialog box opens with a message asking whether the project should be saved at this point. Click **Save**.
2. The **No Implementation Results Available** dialog box asks if it is okay to implement the design before generating the bitstream. Click **Yes**.
3. When bitstream generation completes, the **Bitstream Generation Completed** dialog box opens, with the option **Open Implemented Design** option checked by default. Click **OK** to open the implemented design.
4. If you see a message box that asks about closing synthesized design before opening implemented design, click **Yes**.
5. In the **Flow Navigator, under Implementation**, click Implemented Design to expand it, if it is not already expanded.
6. From the expanded selection, select **Report Timing Summary** to see if the constraints are met.
7. When the Report Timing Summary dialog box opens, click **OK**.
8. Ensure that all timing constraints are met by looking at the Timing Summary tab (FIGURE 69).

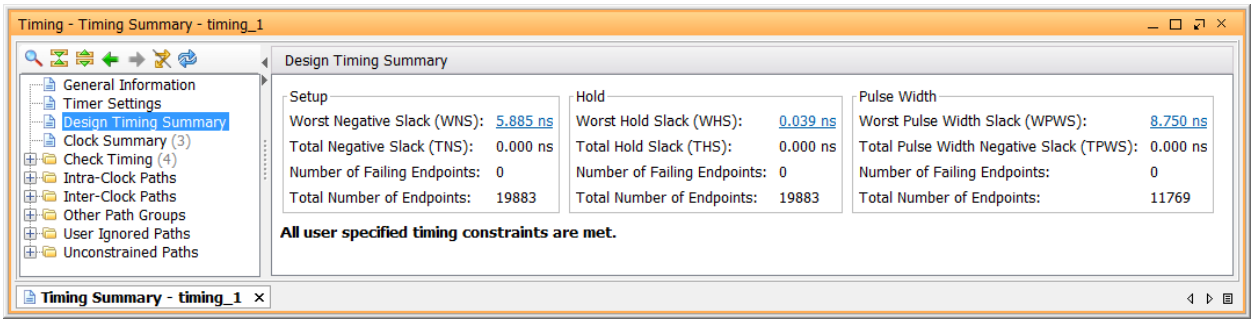

Figure 69: Timing Summary

Step 7: Export Hardware to SDK

After you generate the bitstream, you must export the hardware to SDK and generate your software application.

1. Select **File > Export > Export Hardware.**
2. In the Export Hardware for SDK dialog box, make sure that the **Include bitstream** check box is checked, and **Export to field** is set to **<Local to Project>** (FIGURE 70).

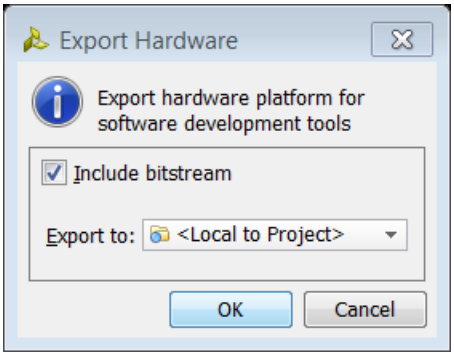

Figure 70: Export Hardware for SDK Dialog Box

3. Click **OK**.
4. Select **File < Launch SDK**. Make sure that both the **Exported location** and **Workspace** fields are set to **<Local to Project>** (FIGURE 71).

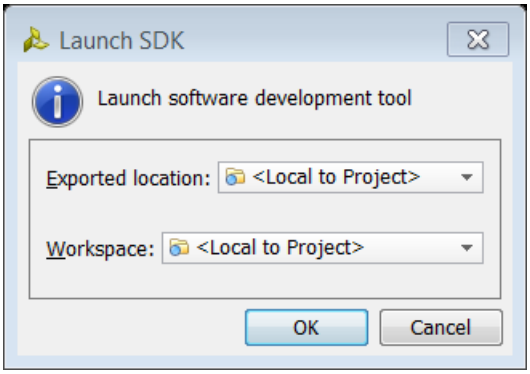

Figure 71: Launch SDK Dialog Box

5. Click **OK**.

Step 8: Build Application Code in SDK

1. SDK launches in a separate window. Select **File > New > Application Project**.
2. In the **New Project** dialog box, specify the name for your project. For this tutorial, use the name **peri_test** (FIGURE 72).
3. Click **Next**.

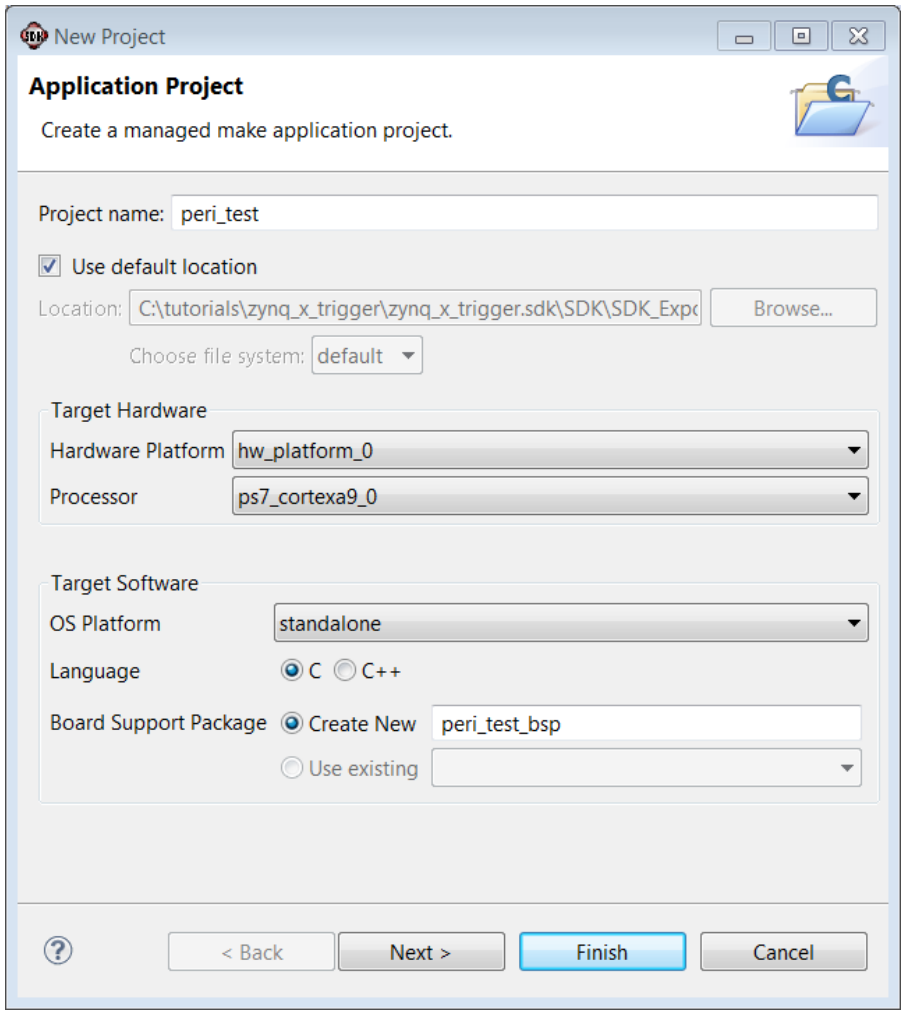

Figure 72: Name the Application Project

4. From the **Available Templates**, select Peripheral Tests (**FIGURE 73**).
5. Click **Finish**.

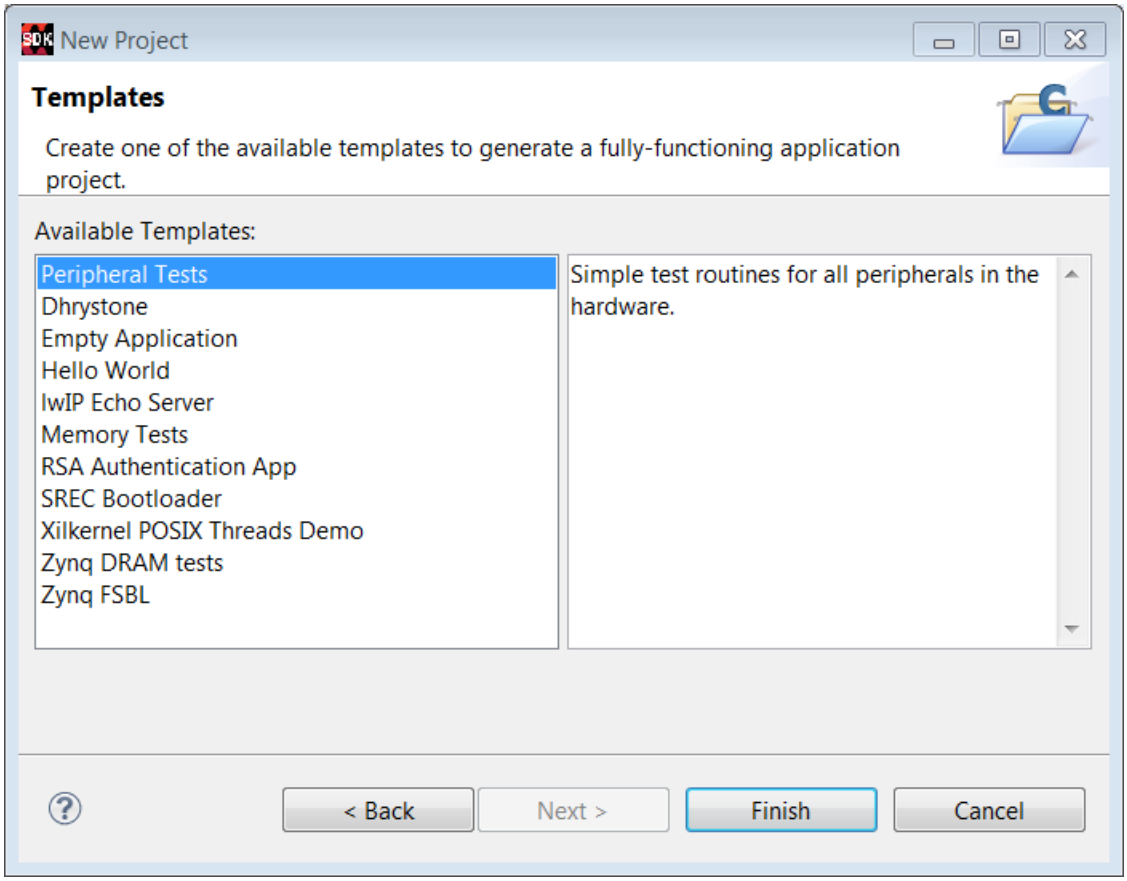

Figure 73: Select the Peripheral Tests Template

6. Select **Xilinx Tools > Program FPGA** to open the **Program FPGA** dialog box.

7. In the **Program FPGA** dialog box, make sure that the path to the bitstream is specified correctly (FIGURE 74).
8. Click **Program**.

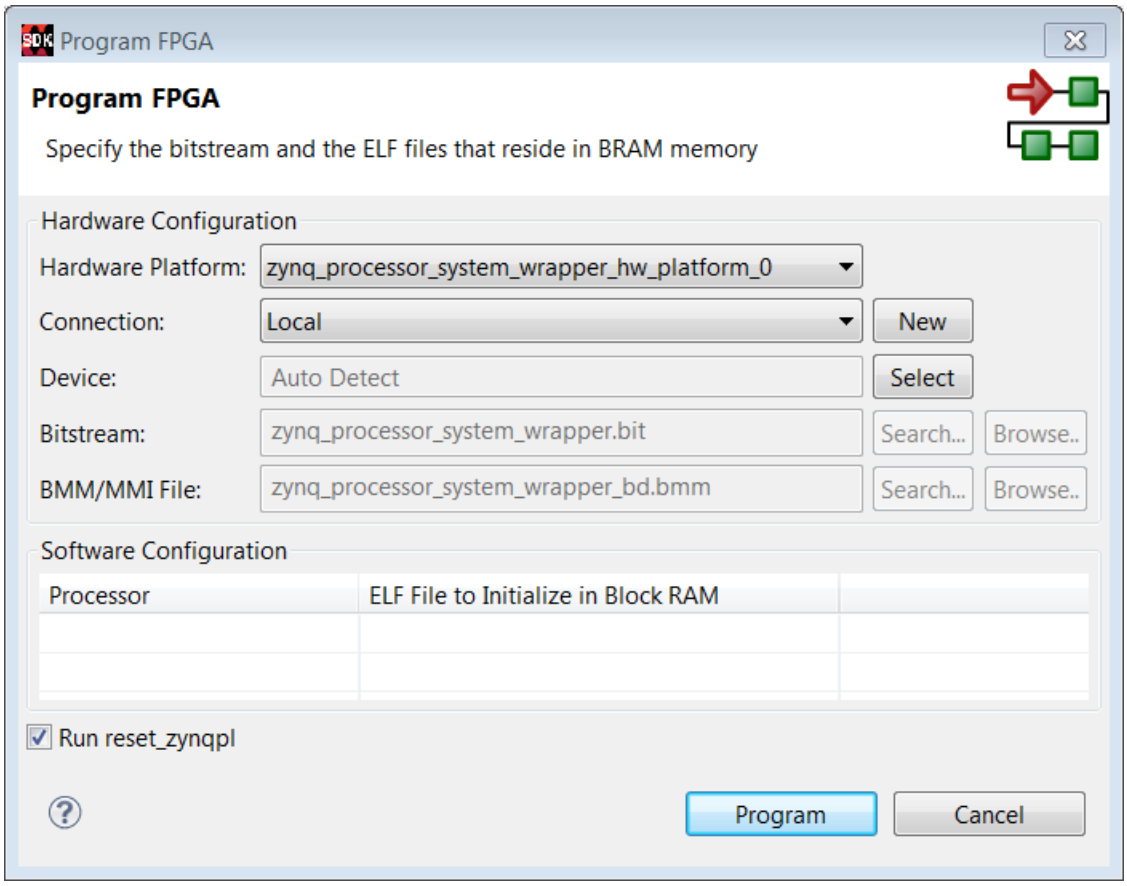

Figure 74: Program FPGA Dialog Box

9. Select and right-click the **peri_test** application in the Project Explorer, and select **Debug As** > **Debug Configurations**.

   The Debug Configurations dialog box opens.
10. Right-click **Xilinx C/C++ application (GDB)**, and select **New**.

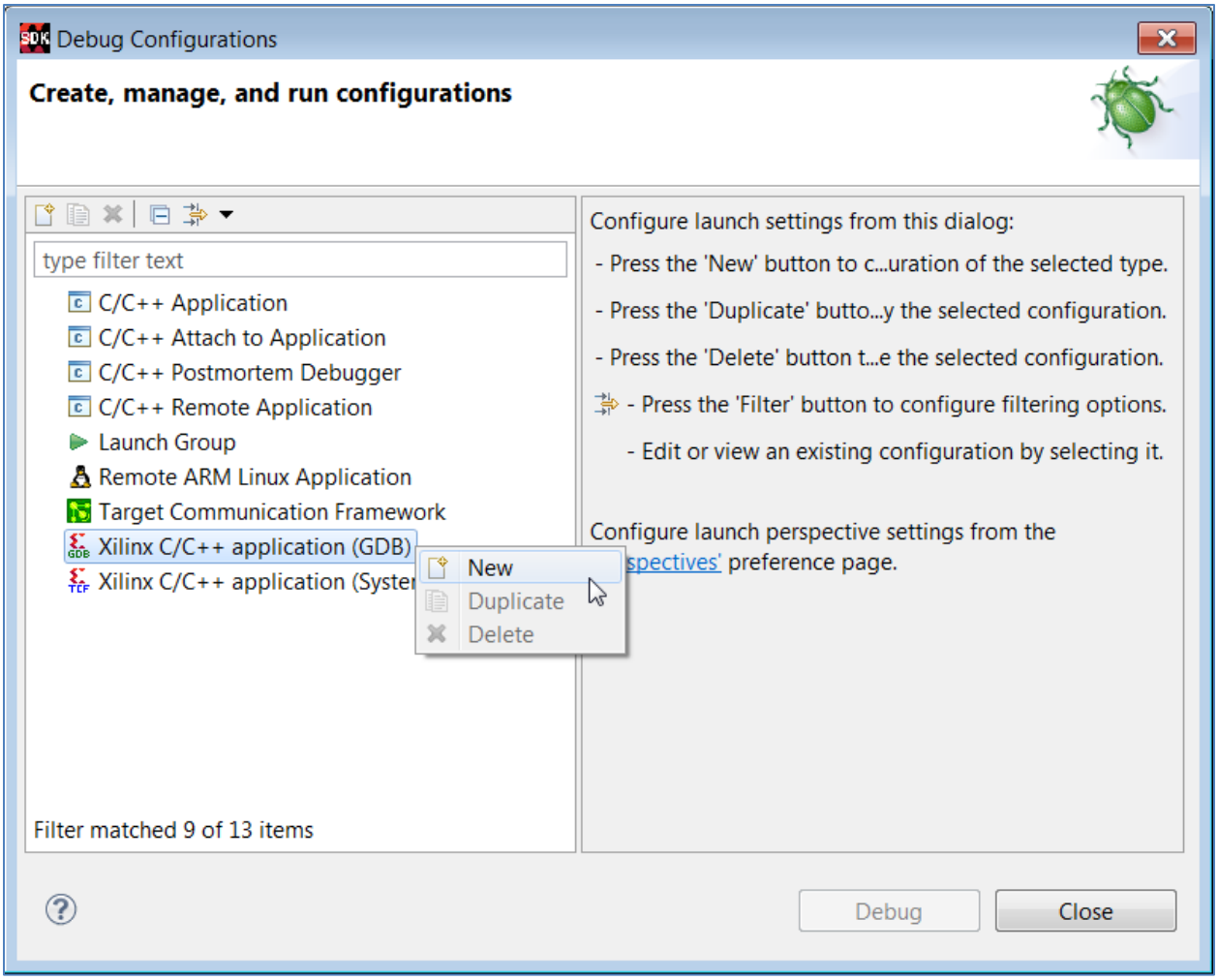

Figure 75: Create New Debug Configuration

11. In the **Create, manage, and run configurations** screen, select the Target Setup tab and check the Enable Cross triggering check box.

12. Click **Debug** (FIGURE 76).

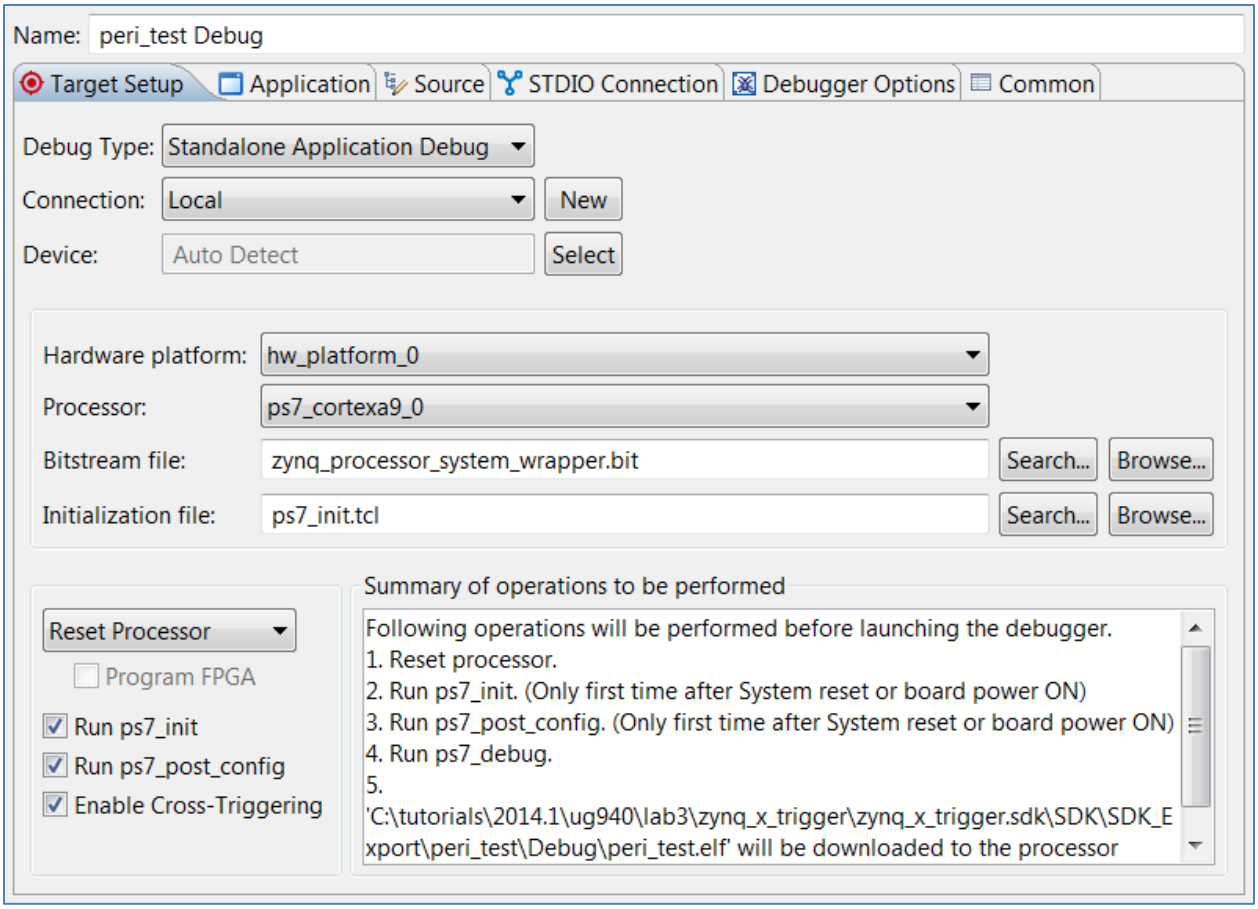

Figure 76: Enable Cross-Triggering

The Confirm Perspective Switch dialog box opens.

13. Click **Yes** to confirm the perspective switch.

    The Debug perspective window opens.

14. Set the terminal by selecting the **Terminal 1** tab and clicking the **Settings** button .

15. Use the following settings for the ZC702 board and click **OK** (FIGURE 77).

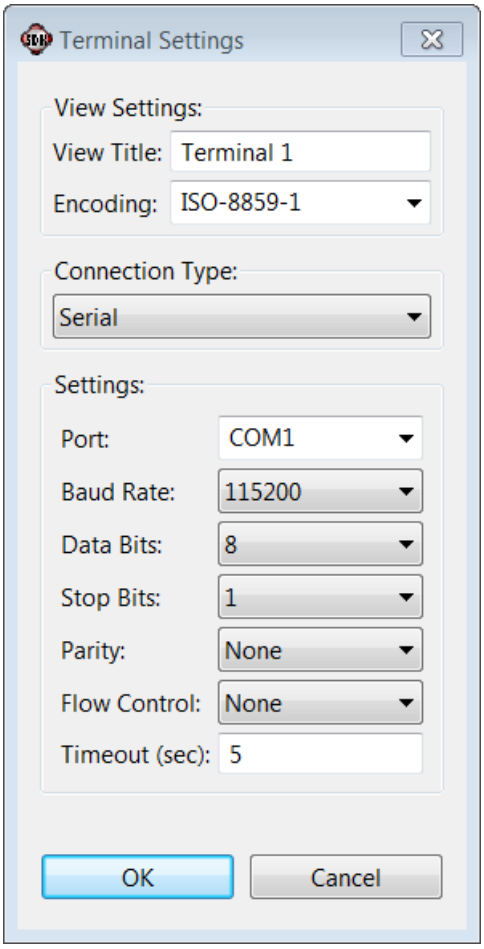

Figure 77: Terminal Settings

16. Verify the Terminal connection by checking the status at the top of the tab (FIGURE 78).

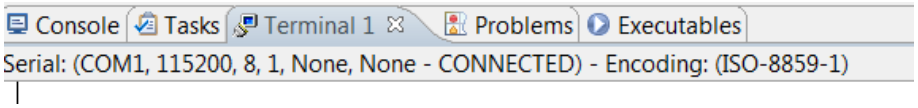

Figure 78: Verify Terminal Connection

17. If it is not already open, select `../src/testperiph.c`, and double click to open the source file.

18. Modify the source file by inserting a while statement at line 94. Click the blue bar on the left side of the **testperiph.c** window as shown in the figure and select **Show Line Numbers**.

19. In line 94, add **while(1)** above in front of the curly brace as shown (FIGURE 79).

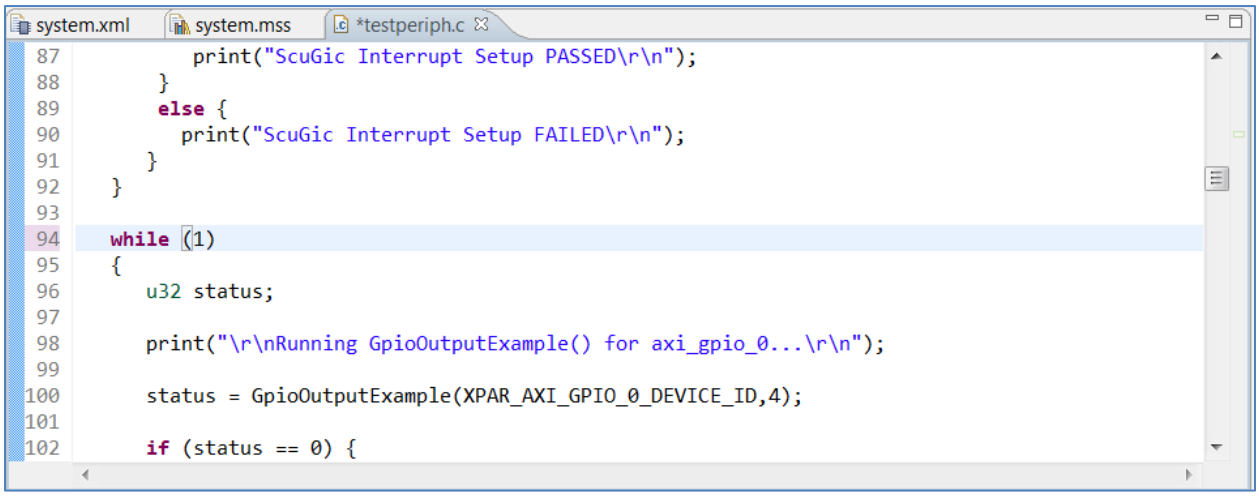

Figure 79: Modify testperiph.c

20. Add a breakpoint in the code so that the processor stops code execution when the breakpoint is encountered. To do so, scroll down to line 98 and double-click on the left pane, which adds a breakpoint on that line of code (FIGURE 80). Click **Ctrl + S** to save the file. Alternatively, you can select **File > Save**.

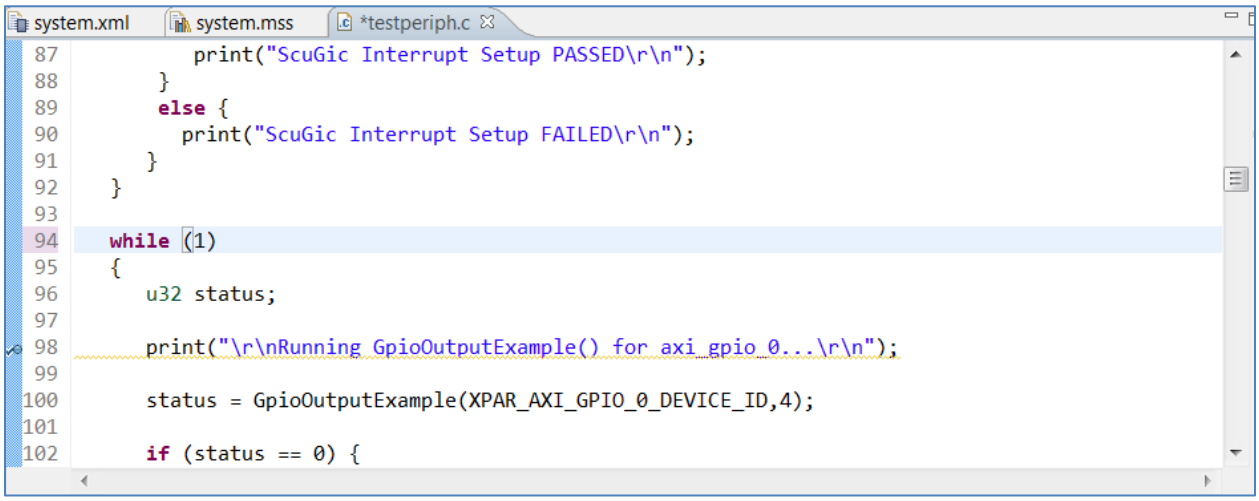

Figure 80: Set a Breakpoint

Now you are ready to execute the code from SDK.

Step 9: Connect to Vivado Logic Analyzer

Connect to the ZC702 board using the Vivado Logic Analyzer.

1. In the Vivado IDE session, from the **Program and Debug** drop-down list of the Vivado Flow Navigator, select **Open Hardware Manager**.
2. In the Hardware Manager window, click **Open a new hardware target**.

Figure 81: Open a New Hardware Target

3. The Open New Hardware Target dialog box opens (FIGURE 82). Click **Next**.

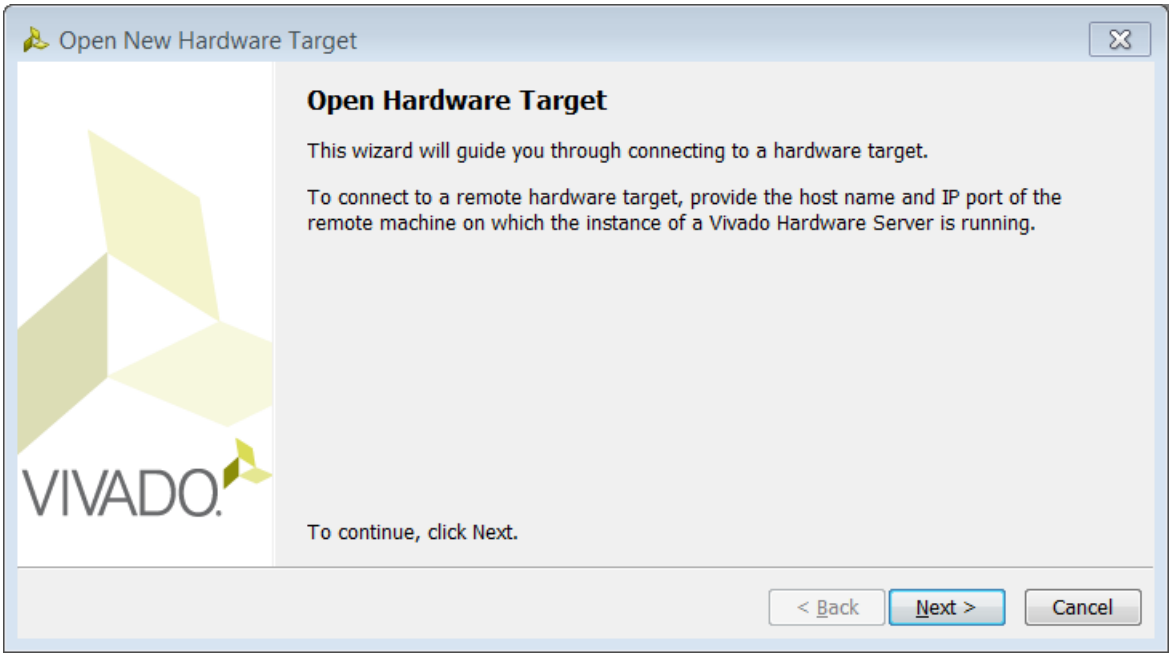

Figure 82: Open Hardware Target

4. On the Hardware Server Settings page, ensure that the **Connect to** field is set to **Local server (target is on local machine)** (FIGURE 83).
5. Click **Next**.

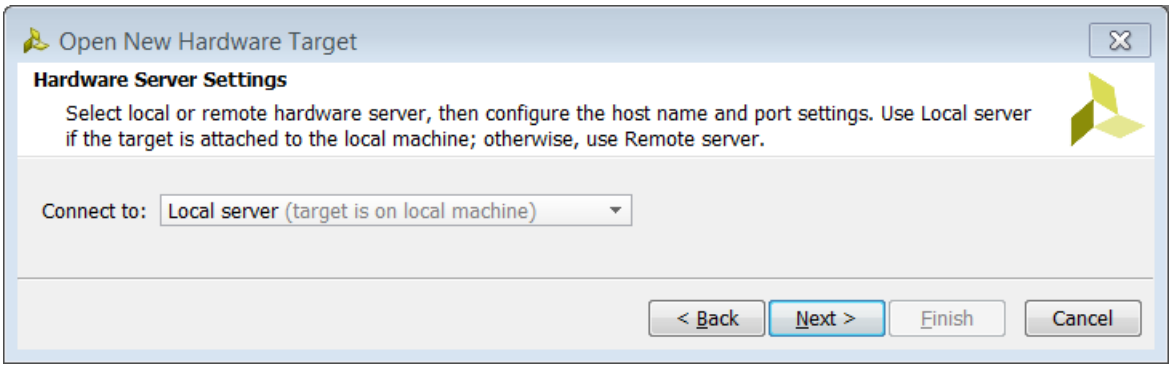

Figure 83: Specify Server Name

6. On the Select Hardware Target screen (FIGURE 84), click **Next**.

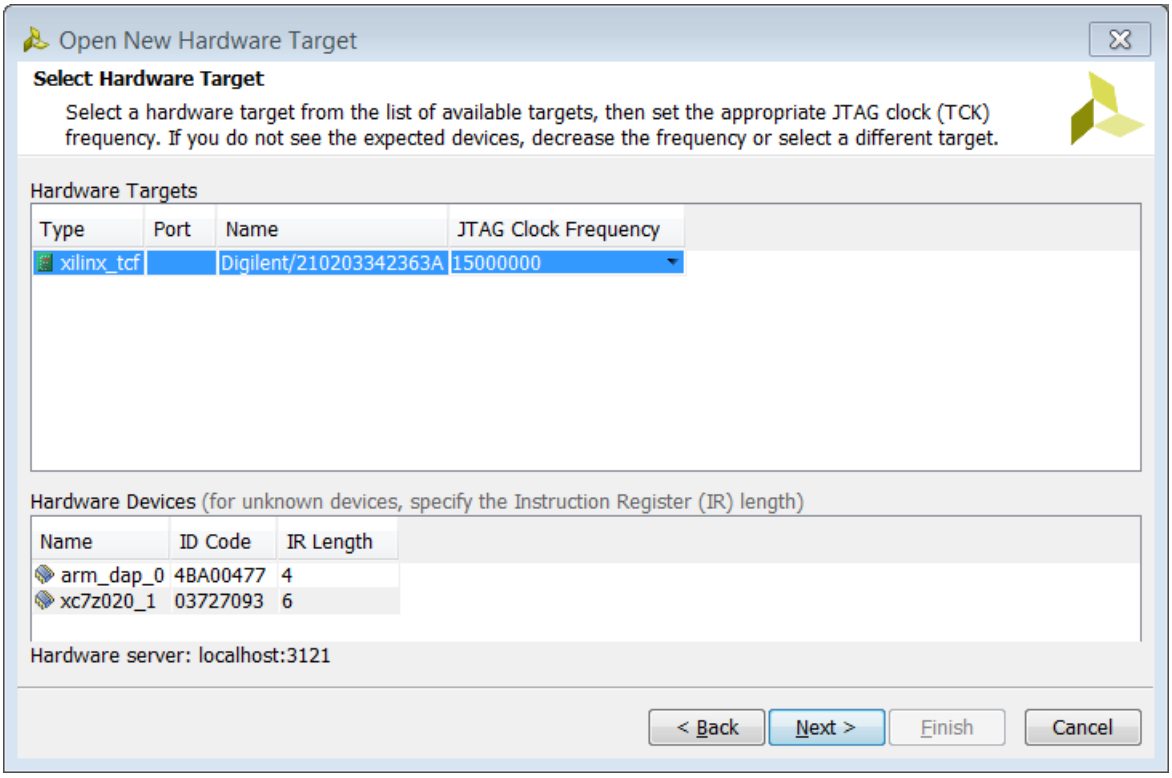

Figure 84: Select Hardware Target

7. Ensure that all the settings are correct on the Open Hardware Target Summary dialog box (FIGURE 85) and click **Finish**.

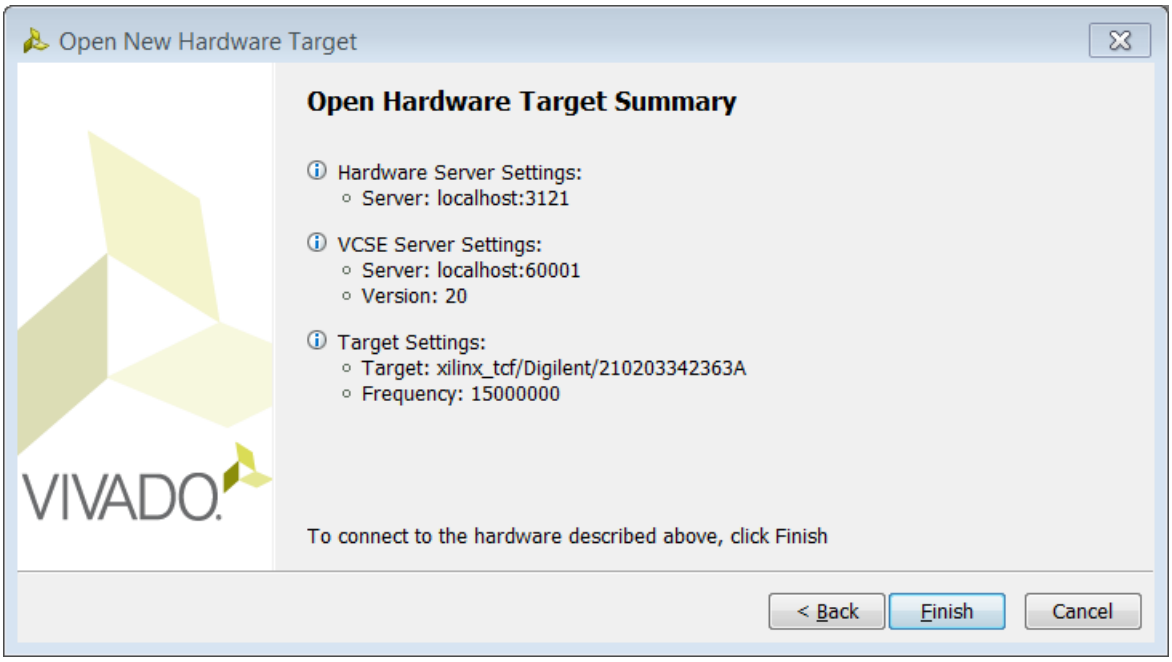

Figure 85: Open Hardware Target Summary

8. When the Vivado Hardware Session successfully connects to the ZC702 board, you see the information shown (FIGURE 86).

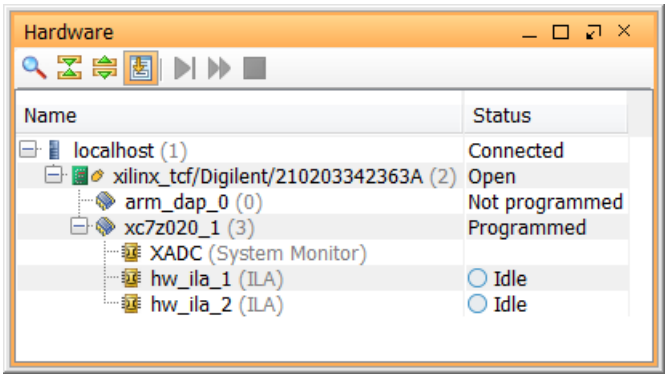

Figure 86: Vivado Hardware Window

9. Select the ILA - hw_ila_1 tab and set the Trigger Mode Settings as follows:
   - Set Trigger mode to **TRIG_IN_ONLY**
   - Set TRIG_OUT mode to **TRIG_IN_ONLY**
   - Under Capture Mode Settings, change **Trigger position in window** to **512**.

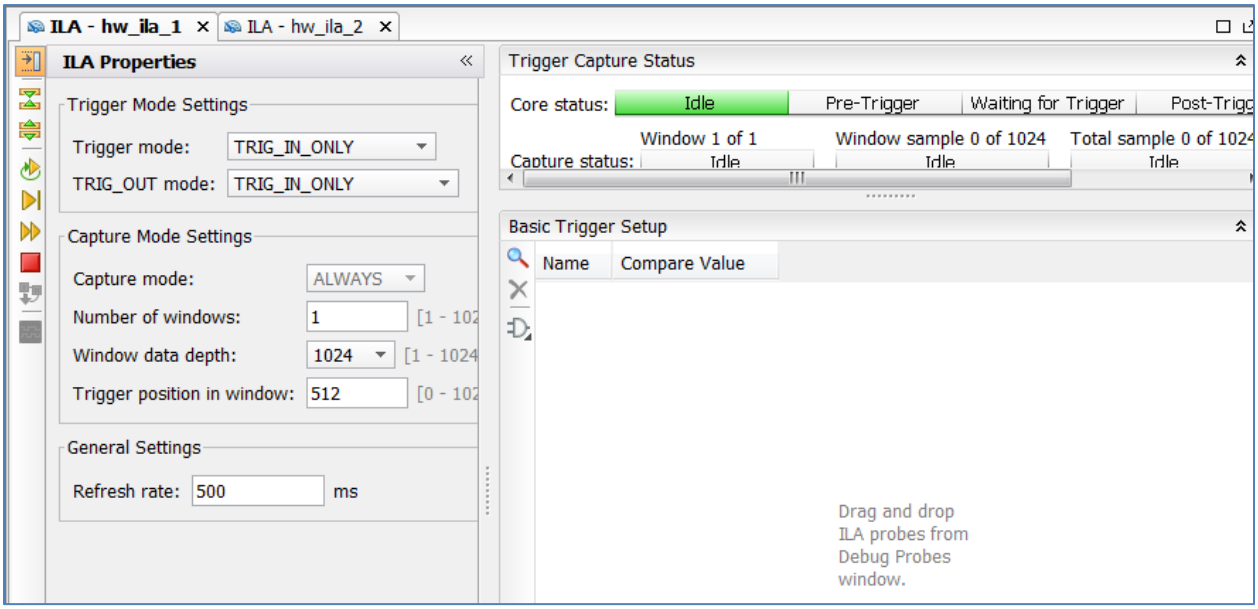

Figure 87: Set ILA Properties for hw_ila_1

10. Arm the ILA core by clicking the **Run Trigger** button .

    This arms the ILA and you should see the status "Waiting for Trigger" as shown in **FIGURE 88**.

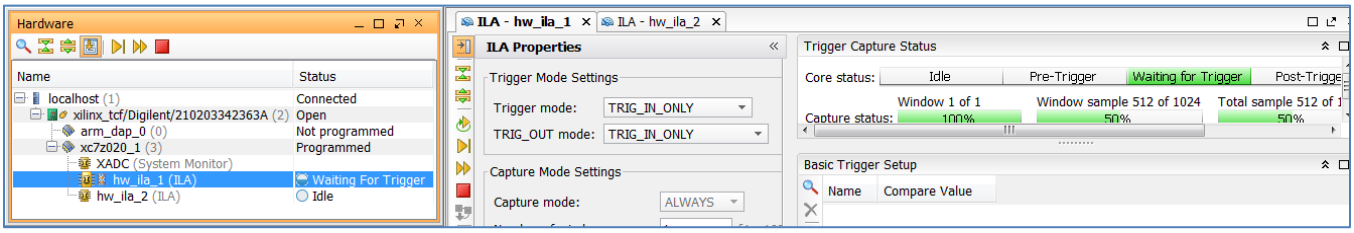

Figure 88: Armed ILA Core

11. Select the **hw_ila_2** tab to set up trigger conditions to capture the trace. Select the **ILA - hw_ila_2** tab and set the Trigger Mode Settings as follows:
    - Set Trigger mode to **BASIC_ONLY**.
    - Under **Capture Mode Settings**, change **Trigger position in window** to **512**.
12. With the **hw_ila_2 tab** selected, drag and drop the **zynq_processor_system_i/processing_system7_0_TRIGGER_OUT_0_TRIG** signal from the Debug

Probes window under **hw_ila_2** into the Basic Trigger Setup window. Alternatively, you can right-click the **zynq_processor_system_i/trig_out** signal and select **Add Probes to Basic Trigger Setup**.

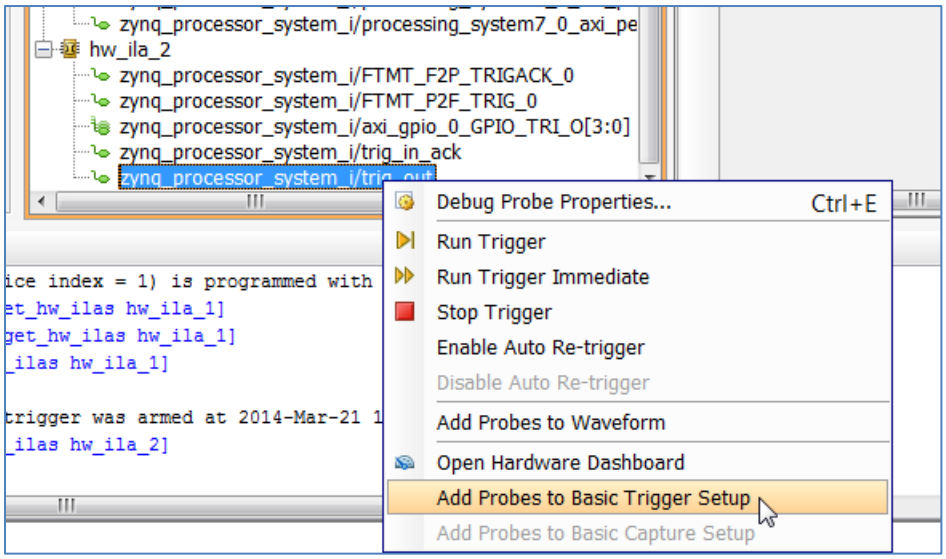

Figure 89: Add zynq_processor_system_i/trig_out to Basic Trigger Setup

13. In the Basic Trigger Setup window, click the **Compare Value** column for the zynq_processor_system_i/FTMT_P2F_TRIG_0 signal, and set the **Value** field to **1**. This essentially sets up the ILA to trigger when the trig_out transitions to a value of 1.

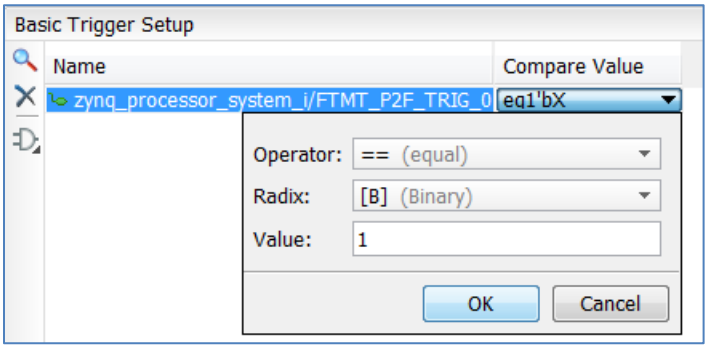

Figure 90: Set Trigger Condition

14. Click **OK**.

15. Arm the ILA by clicking the **Run Trigger** button in the toolbar of the hw_ila_2 window. Just like hw_ila_1, this ILA should be "armed" and waiting for the trigger condition to happen.

16. In SDK, in the Debug window, right-click the **XMD Target Debug Agent** and select **Resume**.

    Vivado displays the hw_ila_2 trigger as shown in [FIGURE 91].

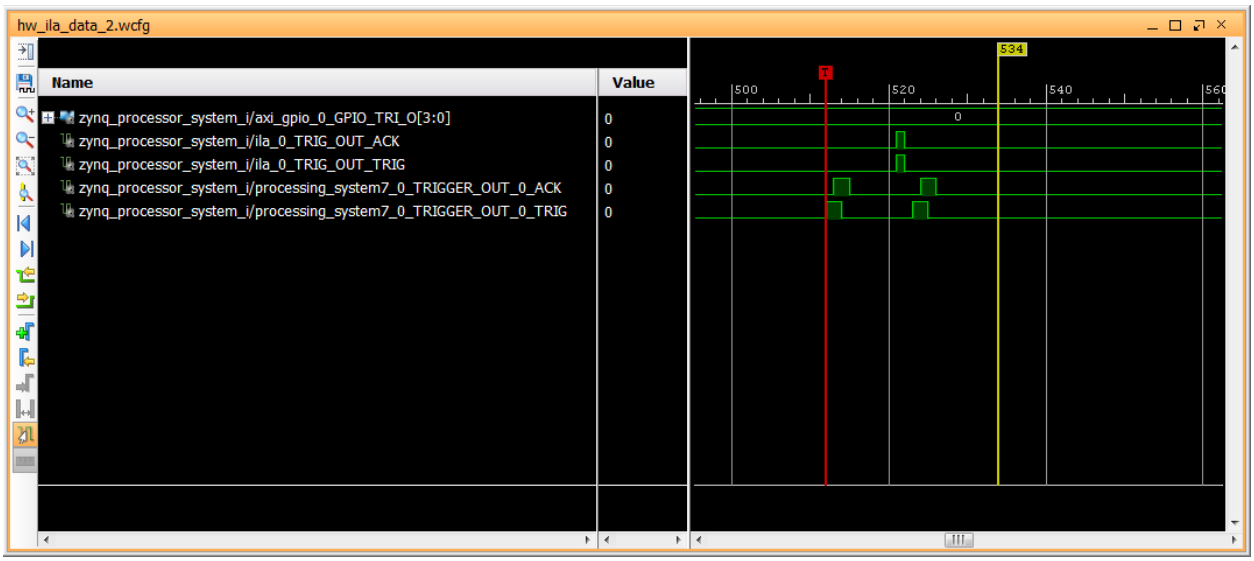

Figure 91: PS to PL Cross Trigger Waveform

    Likewise, hw_ila_1 also triggers as seen in [FIGURE 92].

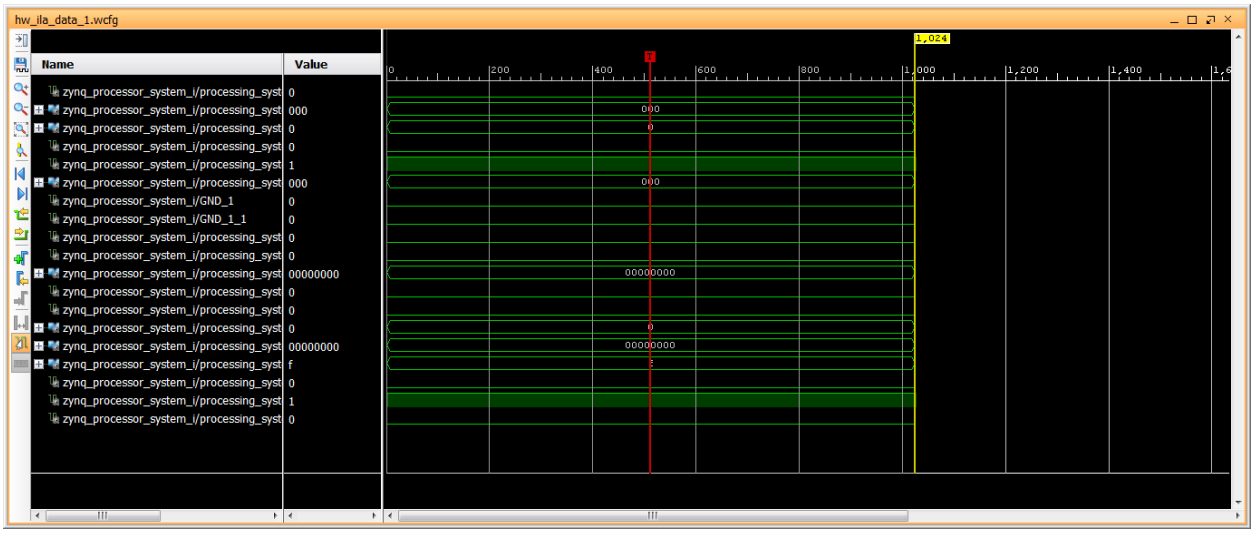

Figure 92: PS to PL Cross Trigger Waveform in hw_ila_1

    This demonstrates that when the breakpoint is encountered during code execution, the PS7 triggers the ILA that is set up to trigger. Between the two waveform windows, you can monitor the state of the hardware at a certain point of code execution.

17. Now we'll try the fabric to processor side of the cross-trigger mechanism. In other words we will remove the breakpoint that you set earlier on line 98 to have the ILA trigger the processor and stop code execution. To do this, double-click the blue bar on Line 98 where the breakpoint is set.

Alternatively, select the Breakpoints tab towards the top right corner of SDK window, right-click it, and select **Remove All**.

18. In the Debug window, right-click the XMD Debug Target Agent and select **Resume**. The code runs continuously because it has an infinite loop.
19. In Vivado, select the **hw_ila_1** tab. Change the **TRIG_OUT** mode to **TRIGGER_OR_TRIG_IN**.
20. Select the **hw_ila_2** tab, and delete the existing probe by selecting it and clicking the **Delete** button to the left.
21. Drag and drop the **zynq_processor_system_i/ila_0_TRIG_OUT_TRIG** signal from the Debug Probes window under **hw_ila_2** into the Basic Trigger Setup window.

    Alternatively, you can also right-click the **zynq_processor_system_i/trig_out** signal and select **Add Probes to Basic Trigger Setup**.
22. In the Basic Trigger Setup window, click the **Compare Value** column for the zynq_processor_system_i/ila_0_TRIG_OUT_TRIG signal, and set the **Value** field to **1**. This essentially sets up the ILA to trigger when the trig_out transitions to a value of 1.
23. Click the **Run Trigger** button to "arm" the ILA. It moves into the "Waiting for Trigger" condition.
24. Select the **hw_ila_1** tab again and click the **Run Trigger Immediate** button .

    This triggers the **hw_ila_2** and the waveform window looks like **FIGURE 93**.

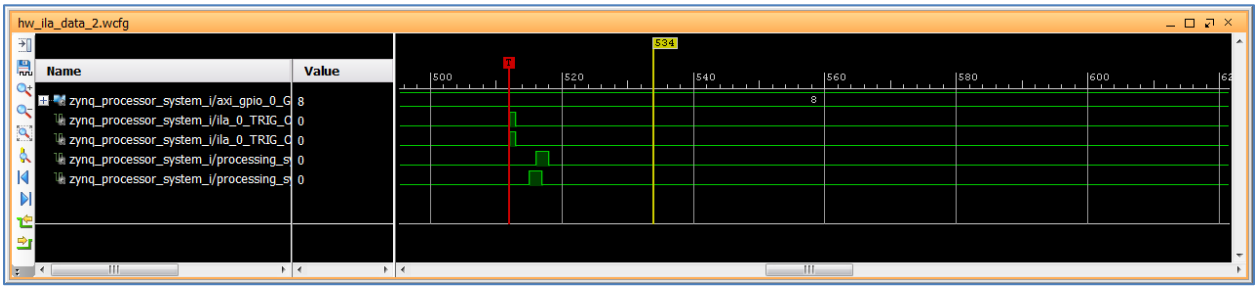

Figure 93: Waveform Demonstrating PS to PL Trigger

This also stops the Processor from executing code because the ILA triggers the TRIG_OUT port, which in turn interrupts the processor. This is seen in SDK the in the highlighted area of the debug window.

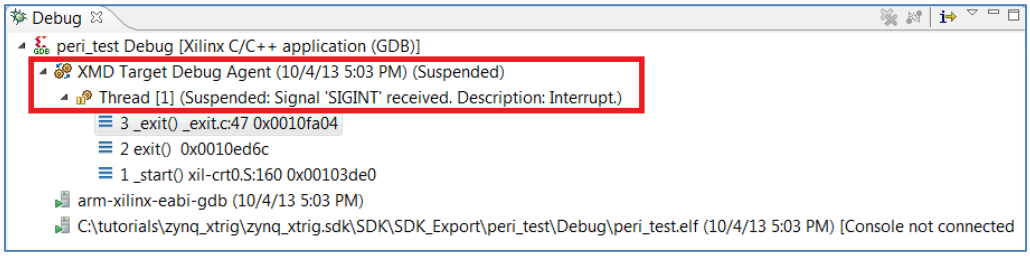

Figure 94: Verify that the Processor Has Been Interrupted in SDK

Conclusion

This lab demonstrated how cross triggering works in a Zynq based design. You can use cross triggering to co-debug hardware and software in an integrated environment.

Lab Files

This tutorial demonstrates the Cross Trigger feature of Zynq, which you perform in the GUI environment. Therefore, the only Tcl file provided is lab3.tcl.

The `lab3.tcl` file helps you run all the steps all the way to exporting hardware for SDK. You might need to modify the net names of the `TRIG_OUT_ACK`, `TRIG_OUT`, `TRIG_IN`, and `TRIG_IN_ACK` signals in the tcl file as these net names might be different after synthesis. The debug portion of the lab must be carried out in the GUI; no Tcl files are provided for that purpose.

Introduction

In this tutorial, you create a simple MicroBlaze™ system for a Kintex®-7 FPGA using Vivado® IP integrator. You can use a machine on any operating system for this tutorial.

The MicroBlaze system includes native Xilinx® IP including:

- MicroBlaze processor
- AXI Timer
- AXI block RAM
- Double Data Rate 3 (DDR3) memory
- UARTLite
- Debug Module (MDM)
- Proc Sys Reset
- Interrupt Controller
- local memory bus (LMB)

These are the basic building blocks used in a typical MicroBlaze system.

In addition to creating the system described above, this tutorial also describes porting an operating system on a Kintex device that you develop in the Xilinx Software Development Kit (SDK) in the Vivado Design Suite. The application code developed in SDK prints "**Hello World**" in a stand-alone application mode.

This tutorial targets the Xilinx KC705 FPGA Evaluation Board, and uses the 2014.2 version of Vivado Design Suite.

Step 1: Invoke the Vivado IDE and Create a Project

1. Invoke the Vivado IDE by clicking the desktop icon or by typing **vivado** at a terminal command line.
2. From the Quick Start page, select Create **New Project** (FIGURE 95).

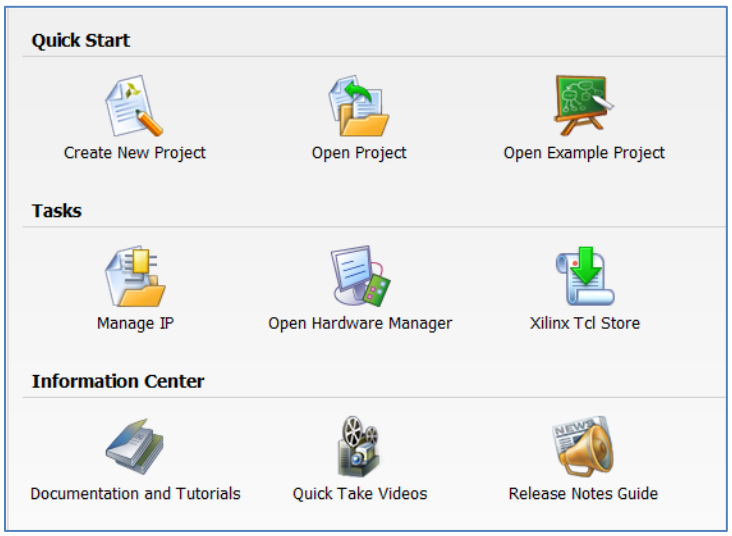

Figure 95: Vivado Quick Start Page

   The New Project wizard opens.

3. In the **Project Name** dialog box, type the project name and location. Make sure that **Create project subdirectory** is checked. Click **Next**.
4. In the **Project Type** dialog box, select **RTL Project**. Click **Next**.
5. In the **Add Sources** dialog box, ensure that the **Target language** is set to **VHDL** or **Verilog**. Leave the **Simulator language** set to its default value of **Mixed**.
6. Click **Next**.
7. In **Add Existing IP** dialog box, click **Next**.
8. In **Add Constraints** dialog box, click **Next**.
9. In the **Default Part** dialog box, select Boards and choose the **Kintex-7 KC705 Evaluation Platform along with the correct version**. Click **Next.**
10. Review the project summary in the **New Project Summary** dialog box before clicking **Finish** to create the project.

    Because you selected the KC705 board when you created the Vivado IDE project, you see the following message:

```
set_property board_part xilinx.com:kintex7:kc705:1.0 [current_project]
```

    Although Tcl commands are available for many of the actions performed in the Vivado IDE, they are not explained in this tutorial. Instead, a Tcl script is provided that can be used to recreate this entire project. See the Tcl Console for more information.

Step 2: Create an IP Integrator Design

1. From **Flow Navigator, under IP integrator**, select **Create Block Design**.

   The Create Block Design dialog box opens.

2. Specify the IP subsystem design name. Leave the Directory field set to its default value of <**Local to Project**>.

3. Click **OK** (FIGURE 96).

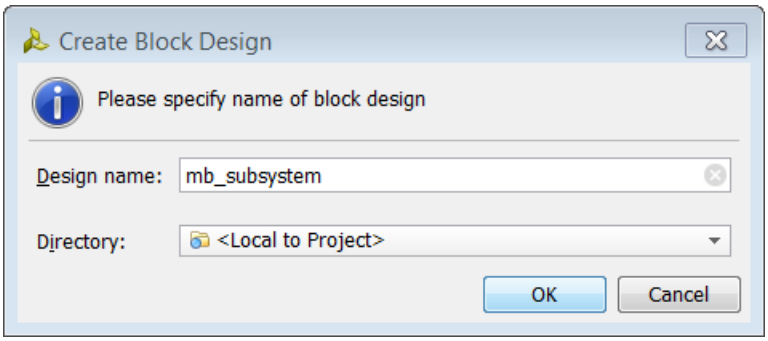

Figure 96: Name Block Design

4. In the IP integrator diagram area, right-click and select **Add IP**.

   The IP integrator Catalog opens. Alternatively, you can also select the **Add IP** link from the top of the diagram area.

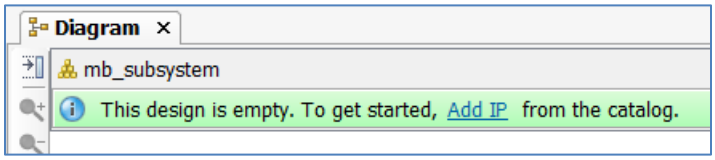

Figure 97: Add IP

5. In the Search field, type **microblaze** to find the MicroBlaze IP, then select MicroBlaze and press the **Enter** key (FIGURE 98).

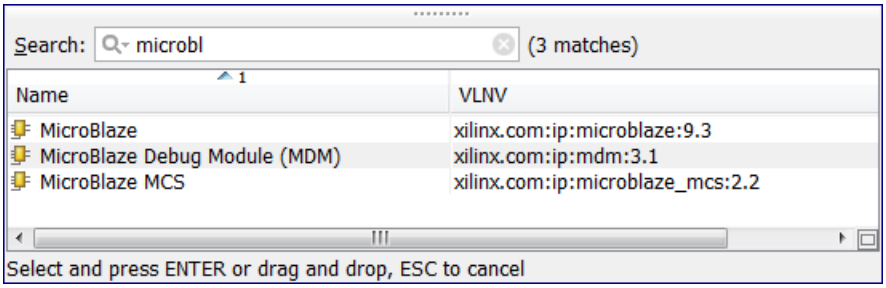

Figure 98: Search Field

Use the Board Part Interfaces Tab to Connect to Board Interfaces

There are several ways to use an existing interface in IP Integrator. We will use the Board Part Interfaces tab to instantiate some of the interfaces that are present on the KC705 board.

1. Click the **Board Part Interfaces** tab. You can see that there are several interfaces that are present on the KC705 board. These interfaces are all listed under **Unconnected Interfaces** in the Board Part Interfaces windows.

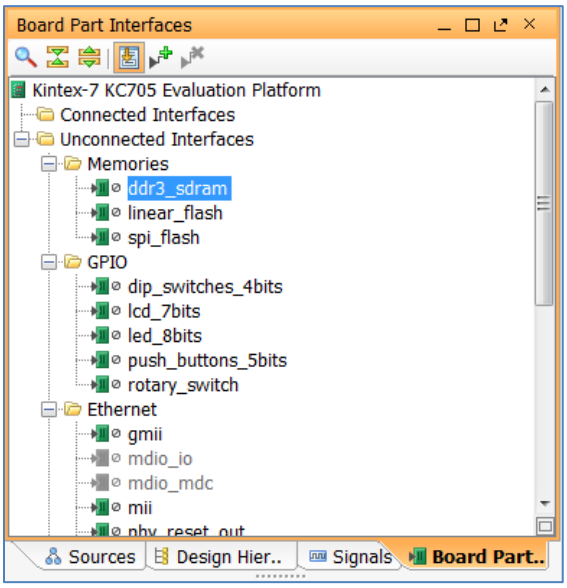

Figure 99: Using the Board Part Interfaces to Configure a MIG

2. Double click **ddr3_sdram**.

   The Connect Board Part Interface dialog box opens (FIGURE 100).

3. Click **New**.

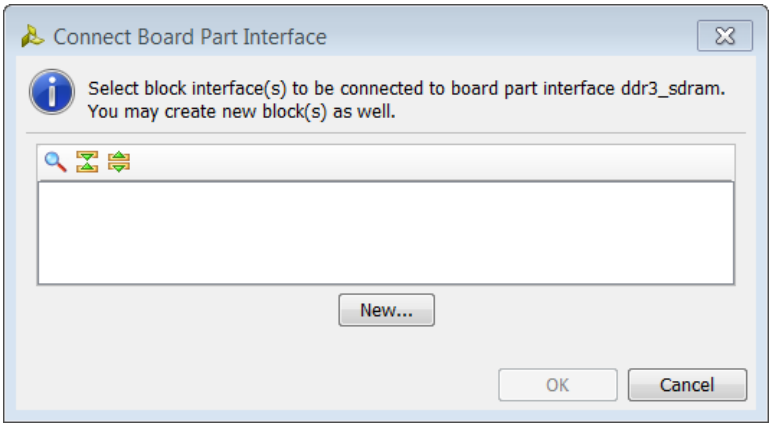

Figure 100: Connect Board Part Interface Dialog Box

The IP Catalog opens, showing the MIG controller as a possible IP to connect to this interface.

4. Select and double-click the Memory Interface Generator.

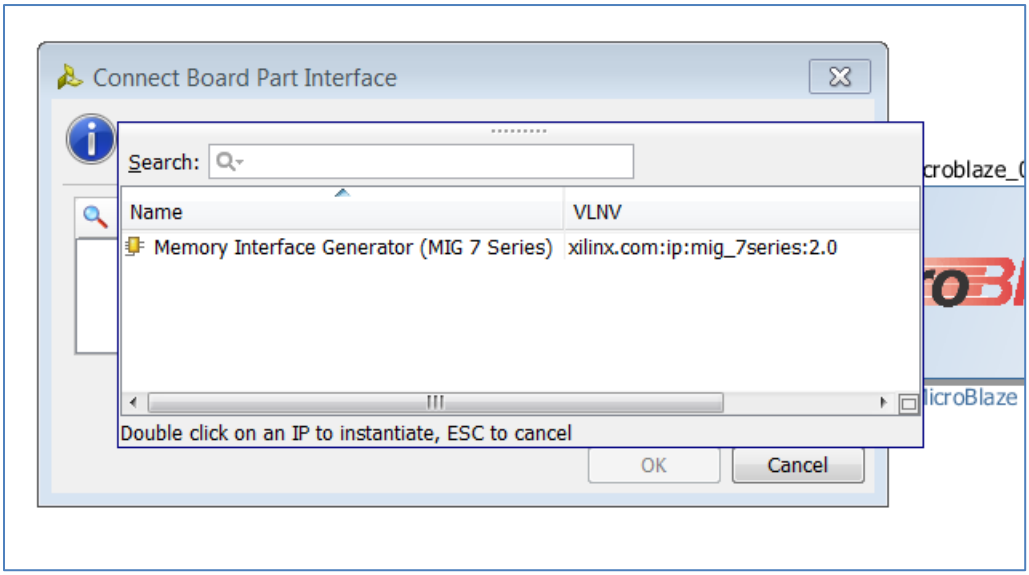

Figure 101: IP Catalog Showing the MIG as a Possible IP to Connect to the DDR3 Interface

Notice that the Connect Board Part Interface dialog box now shows the MIG interface, which is selected by default.

5. Click **OK**.

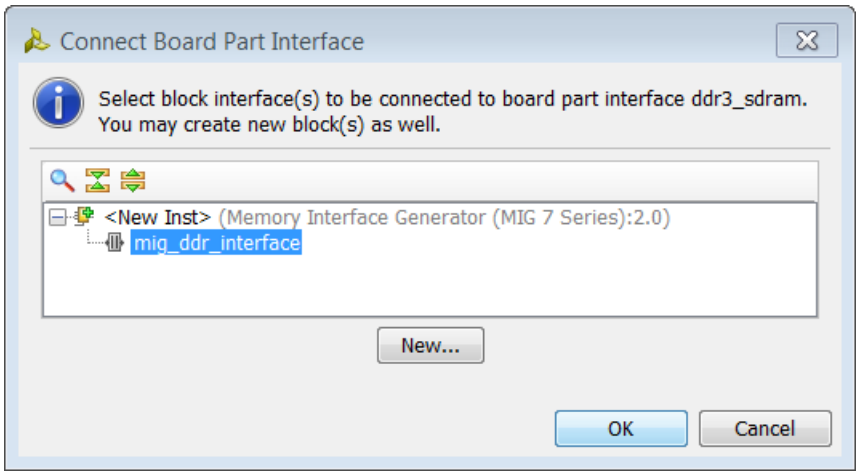

Figure 102: Select the MIG core to connect to the DDR3 Interface

This will instantiate the MIG core and connects the DDR3 interface and clock to the MIG as shown in **FIGURE 103**.

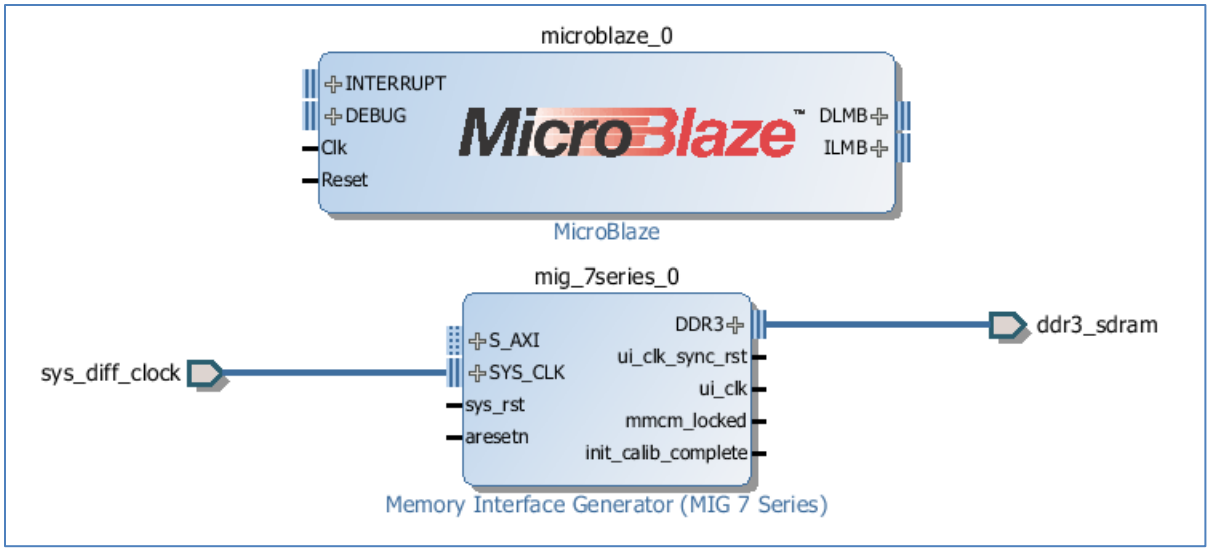

Figure 103: Block Diagram After Instantiating the MIG Core

6. In the Board Part Interfaces tab, notice that the ddr_sdram interface now appears under the **Connected Interfaces** folder.

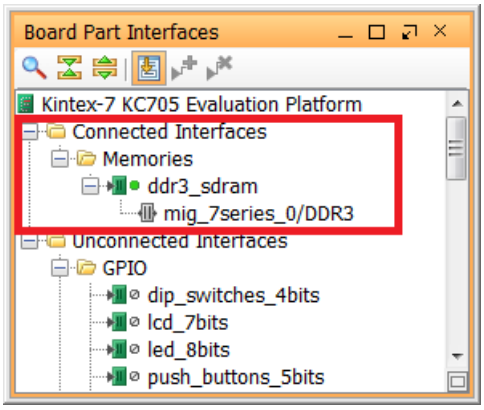

Figure 104: DDR3 Interface Shown Under Connected Interfaces

7. In the Board Part Interfaces tab, select and double-click **rs232_uart** under the **Miscellaneous** folder.
8. When the Connect Board Part Interfaces dialog box opens, click **New**.
9. In the IP Catalog that opens, select and double-click **AXI Uartlite** to instantiate and connect to the **rs232_uart** interface.
10. In the Connect Board Part Interface dialog box, click **OK**.

Run Block Automation

1. Click **Run Block Automation** and select **/microblaze_0** (FIGURE 105).

Figure 105: Run Block Automation

2. The Run Block Automation dialog box opens (FIGURE 106).

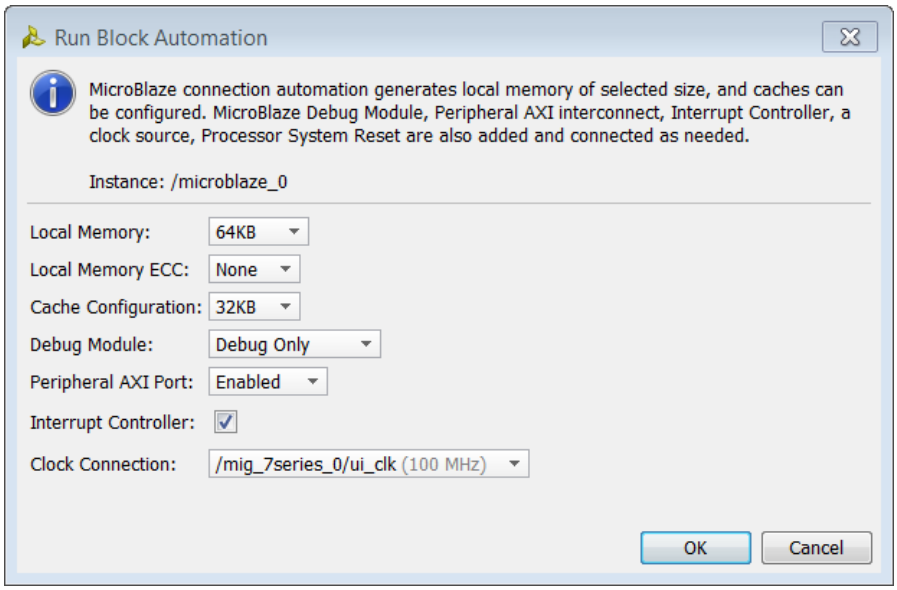

Figure 106: Run Block Automation Dialog Box

3. On the Run Block Automation page:
   a. Set Local Memory to **64 KB**.
   b. Leave the Local Memory ECC to its default value **None**.
   c. Change the Cache Configuration to **32 KB**.
   d. Leave the **Debug Module** option to its default state **Debug Only**.
   e. Leave the **Peripheral AXI Port** option set to its default value of **Enabled**.
   f. Check the **Interrupt Controller** option.
   g. Select the **Clock Connection** option of `/mig_7series_1/ui_clk (100 MHz)`.
4. Click **OK**.

This generates a basic **MicroBlaze** system in the IP Integrator diagram area (FIGURE 107).

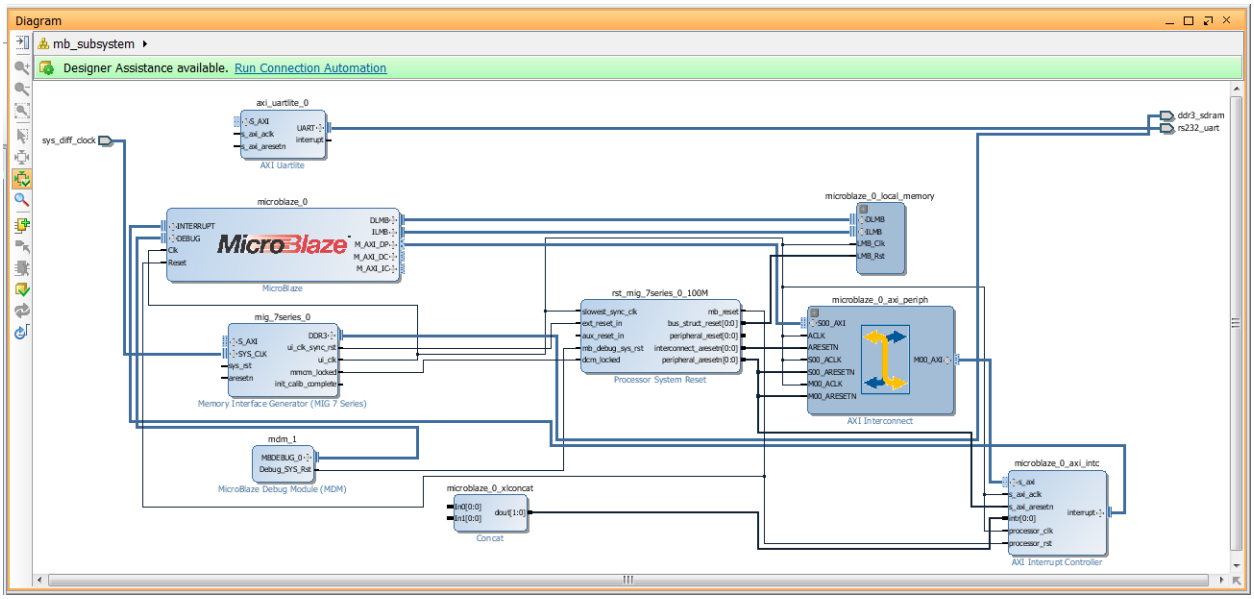

Figure 107: MicroBlaze System

Add Peripherals: AXI Timer and AXI BRAM Controller

1. By right-clicking in the IP integrator diagram area and selecting **Add IP**, search for and select the **AXI Timer** (FIGURE 108).

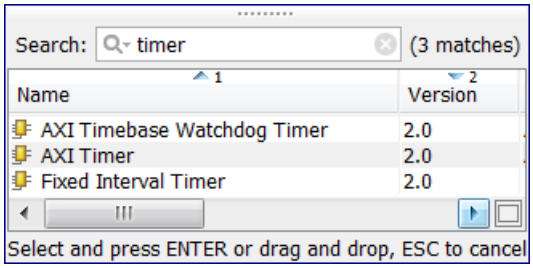

Figure 108: AXI Timer

2. Likewise, add the AXI BRAM Controller (FIGURE 109).

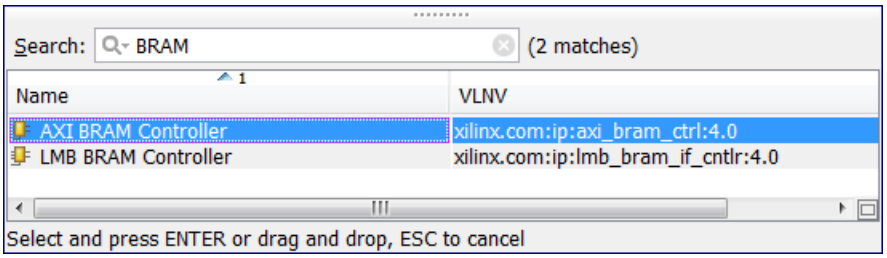

Figure 109: Add BRAM Controller

Use Connection Automation

Run Connection Automation provides several options that you can select to make connections. In this section, we'll walk you through the first connection, and then you will use the same procedure to make the rest of the required connections for this tutorial.

1. Click **Run Connection Automation** and select **/mig_7series_0/S_AXI** (FIGURE 110).

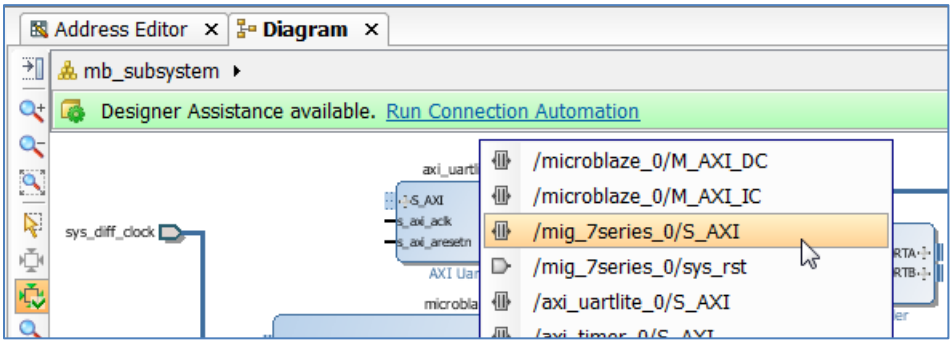

Figure 110: Run Connection Automation on /mig_7series_0/S_AXI

The Run Connection Automation dialog box opens. For /mig_7series_0/S_AXI, you have two options:

- Select the **Peripheral AXI Interconnect (/microblaze_0/Periph)** to connect to the MicroBlaze processor.
- Select the **Cached AXI Interconnect (/microblaze_0/Cached)** to connect to the MicroBlaze processor. This option is selected by default.

2. Leave the Clock Connection field to its default value of **Auto** and click **OK**.
3. Now, use the **Run Connection Automation** option to set the following connections.

Connection	*More Information*	*Setting*
/mig_7series_0/sys_rst	The Run Connection Automation dialog box opens and shows that the board interface reset will be connected to the reset pin of the MIG controller.	Click **OK** to acknowledge this connection.
/axi_uartlite_0/S_AXI	The **Master** field is set to its default value of **/microblaze_0 (Periph)**. The **Clock Connection** (for unconnected clks) field is set to **Auto**.	Keep these default settings and click **OK**.
/axi_timer_0/S_AXI	The Clock Connection field is set to its default value of **Auto**.	Keep this default setting and click **OK**.

Connection	*More Information*	*Setting*
/mig_7series_0/sys_rst	The Run Connection Automation dialog box opens and shows that the board interface reset will be connected to the reset pin of the MIG controller.	Click **OK** to acknowledge this connection.
/axi_bram_ctrl_0/S_AXI	The Run Connection Automation dialog box offers to connect this to the **/microblaze_0 (Cached).**	Leave the Master and the Clock Connection fields to their default values and click **OK**.
/axi_bram_ctrl_0/ BRAM_PORTA	The Run Connection Automation dialog box opens stating that the BRAM Controller will be connected to a new instance of Block Memory Generator.	Click **OK** to acknowledge this connection.
/axi_bram_ctrl_0/ BRAM_PORTB	The Run Connection Automation dialog box opens and gives you two choices: • Instantiate a new BMG and connect the PORTB of the AXI BRAM Controller to the new BMG IP • Use the previously instantiated BMG core and automatically configure it to be a true dual-ported memory and connected to PORTB of the AXI BRAM Controller. The **/axi_bram_ctrl_0_bram** is selected by default.	Click **OK** to accept the default setting.

At this point, your IP integrator diagram area should look like **FIGURE 111**. Relative placement of IP might be slightly different.

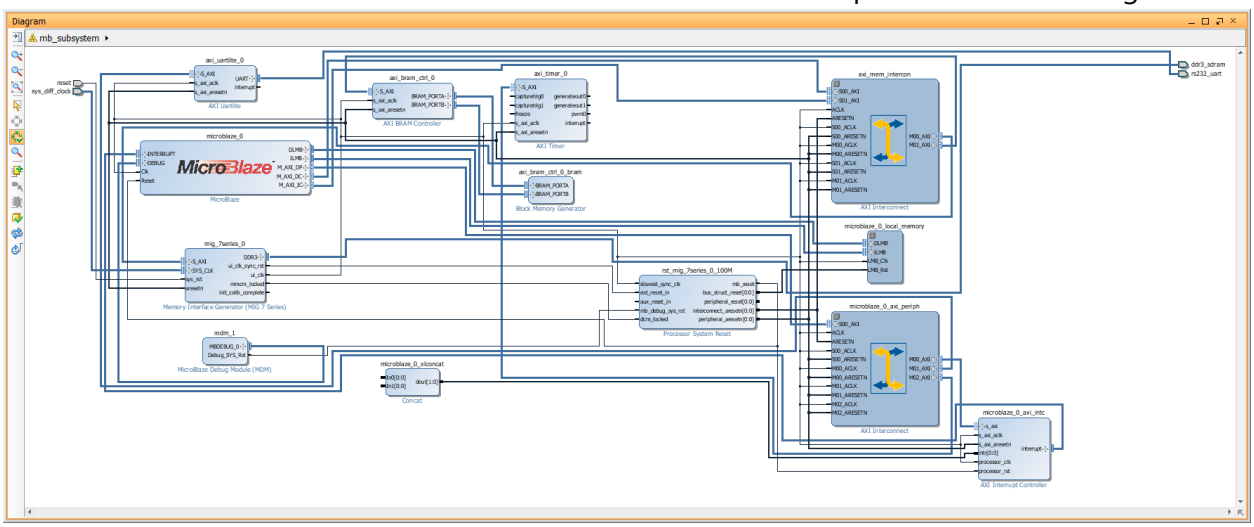

Figure 111: MicroBlaze Connected to UART and AXI Timer

Concatenate Interrupt Signals

Now, complete the input connections to the Concat IP, which *concatenates* the interrupt signal generated from the AXI Timer and the AXI Uartlite.

1. Connect the interrupt pin of the AXI Timer to the input pin **In0[0:0]** of Concat.
2. Connect the interrupt pin of AXI Uartlite to the input pin **In1[0:0]** of Concat. The connections should look like **FIGURE 112**.

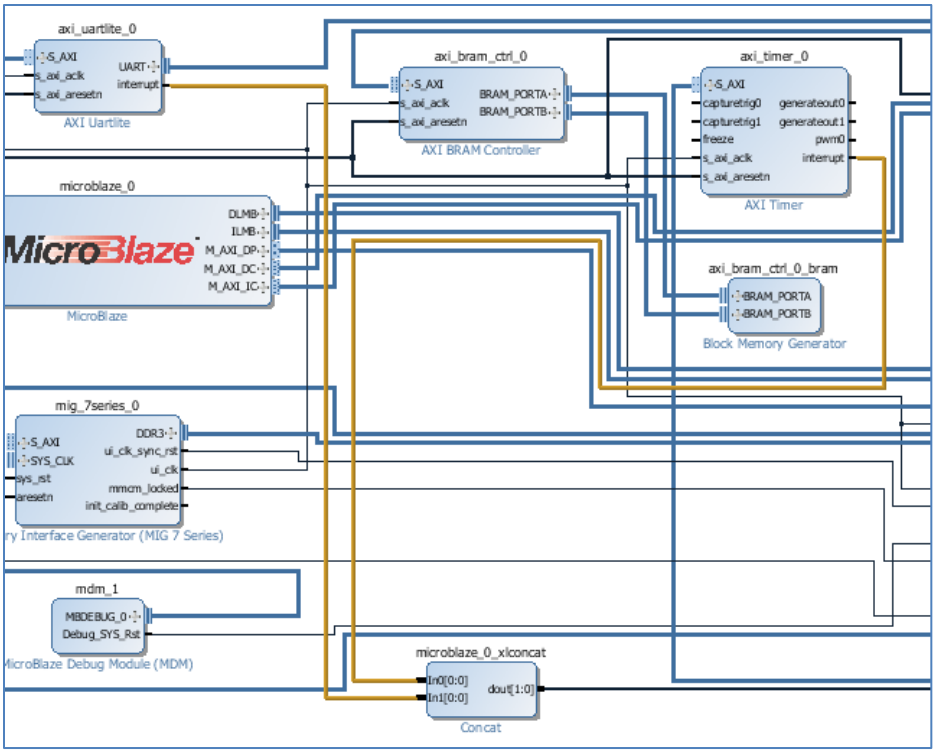

Figure 112: Connected Interrupt Ports

Regenerate Layout

3. Click the **Regenerate Layout** button in the IP Integrator toolbar to generate an optimum layout for the block design. The block diagram should look like **FIGURE 113**.

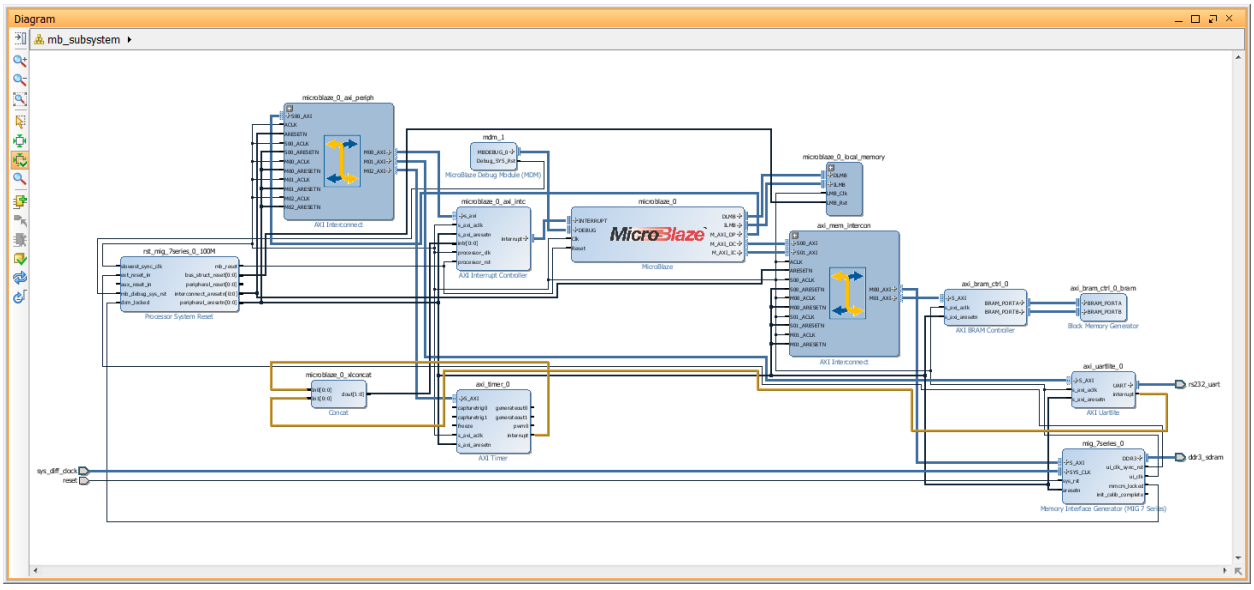

Figure 113: Block Diagram After Regenerating the Layout

Step 3: Memory-Mapping the Peripherals in IP Integrator

1. Click the **Address Editor** tab. In the Address Editor:
   a. Expand the **microblaze_0** instance.
   a. Change the range of `mig_7_series_0` IP in both the **Data** and the **Instruction** section to **512 MB** (FIGURE 114).

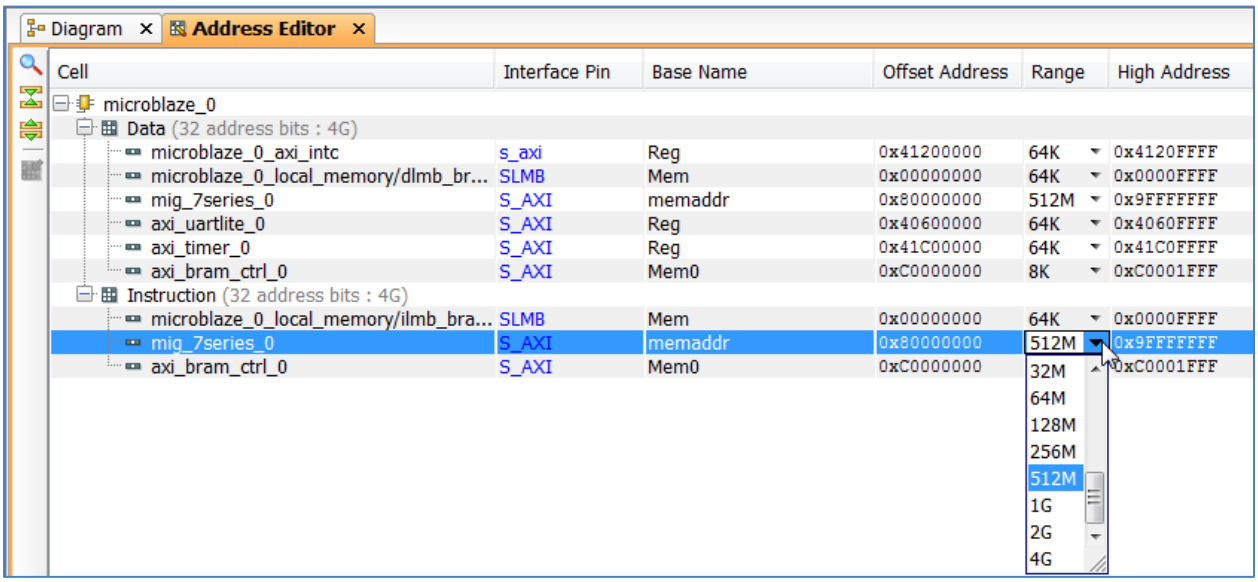

Figure 114: Data and Instruction Set to 512 MB

You must also ensure that the memory in which you are going to run and store your software is within in the cacheable address range that you specified when you assigned values to the cache(s) Base address and cache(s) High address.

This occurs when you enable Instruction Cache and Data Cache, and when you re-configure the MicroBlaze processor.

To use either MIG DDR or AXI block RAM, those IP must be in the cacheable area; otherwise, the MicroBlaze processor cannot read from or write to them.

You can also use this map to manually include or exclude IP from the cacheable region or otherwise specify their addresses.

Step 4: Validate Block Design

To run design rule checks on the design:

1. Click the **Validate Design** button on the toolbar, or select **Tools > Validate Design**.

   The Validate Design dialog box informs you that there are no critical warnings or errors in the design.

2. Save your design by pressing **Ctrl+S**, or select **File > Save Block Design**.

Step 5: Generate Output Products

1. In the Sources window, select the block design, then right-click it and select **Generate Output Products**.

   Alternatively, you can click **Generate Block Design** in the Flow Navigator.

   The Generate Output Products dialog box appears.

2. Click **Generate**.

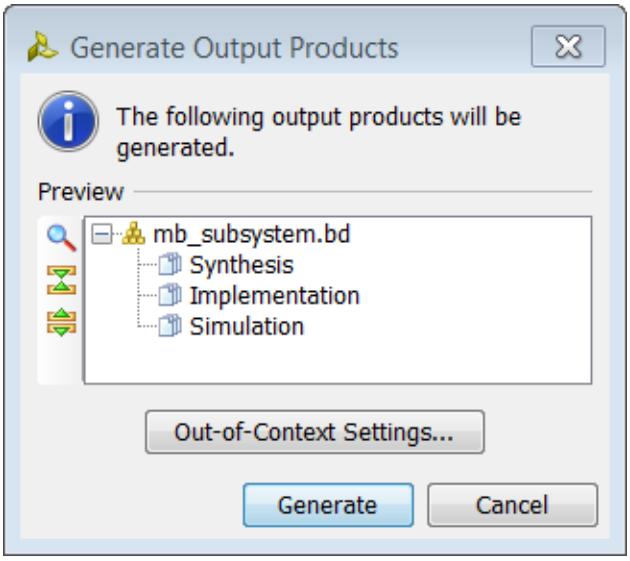

Figure 115: Generate Output Dialog Box

Step 6: Creating a Top-Level Verilog Wrapper

1. Under **Design Sources**, right-click your design and click **Create HDL Wrapper**.

   In the Create HDL Wrapper dialog box, **Let Vivado manage wrapper and auto-update** is selected by default.

2. Click **OK**.

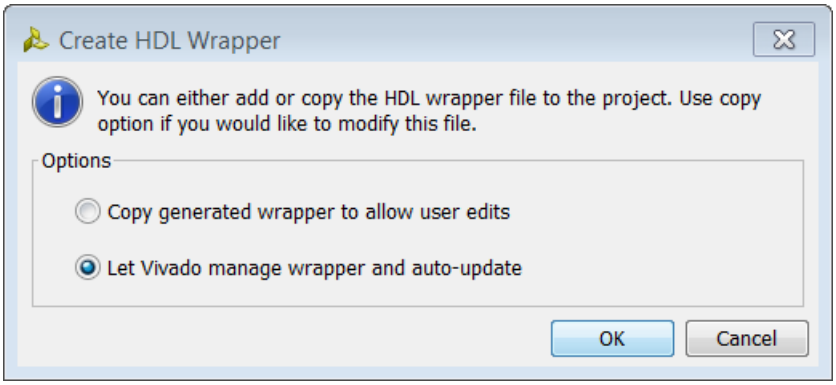

Figure 116: Creating an HDL Wrapper

Step 7: Create Constraints

To complete the IP integrator design, you must create constraints.

1. In the Sources window, expand the **Constraints** folder (FIGURE 117).

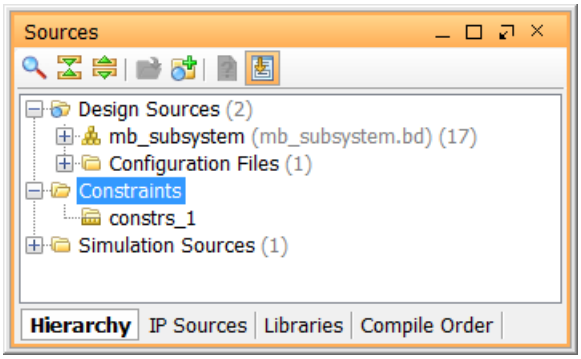

Figure 117: Project Manager

2. Open the **Constraints** folder, right-click the **constrs_1** folder, then select **Edit Constraints Sets**.

   The Edit Constraints Sets dialog box opens.

3. Click **Create File** (FIGURE 118).

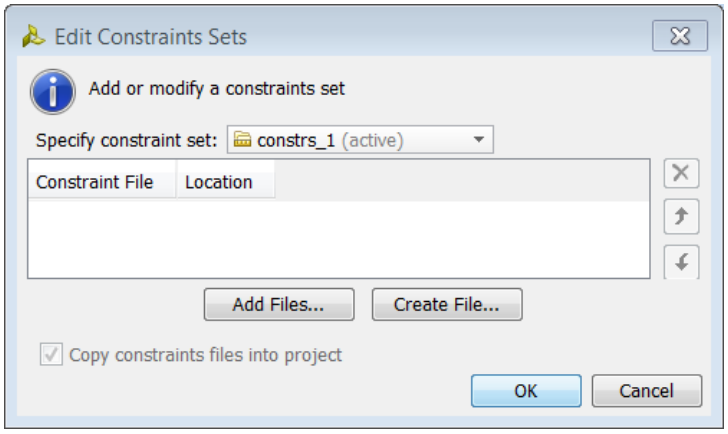

Figure 118: Edit Constraints Set Dialog Box

4. Name the file **system**, and click **OK** (FIGURE 119).

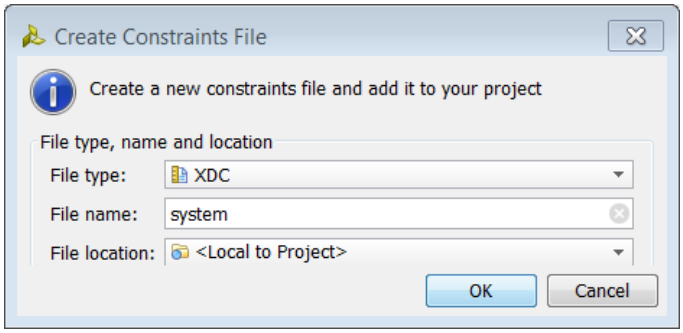

Figure 119: Name Constraint Set File

5. In the Edit Constraints Set dialog box, click **OK**.
6. Expand the **constrs_1** folder and double-click the constraints file.

   The constraints file opens. This file specifies the local constraints for some of the ports created in the block design (FIGURE 120).

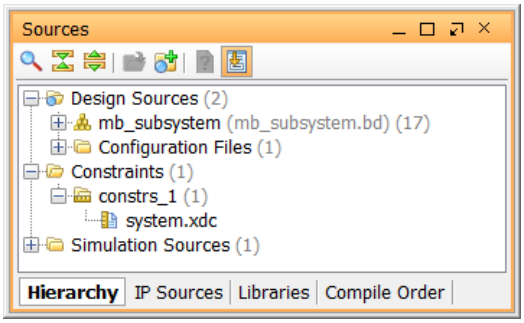

Figure 120: Local Constraints File

7. In the constraints file, add the following lines of code for the KC705 board with the Kintex-7 FPGA.

```
# Added for RevC board
set_property slave_banks {32 34} [get_iobanks 33]
```

TIP: You might have noticed that the constraints for the RS232_Uart port are not included in the above constraints file. That is because Vivado automatically generates the appropriate constraints for those ports because you selected the target board as the KC705 board. This is the board automation feature of the Vivado IDE IP integrator that you can use to hook up ports such as clocks, resets, GPIOS, and UART to the pins on the target board.

8. Save the file by pressing **Ctrl + S**, or select **File > File Save**.

Step 8: Take the Design through Implementation

In the Flow Navigator:

1. Click **Generate Bitstream**.

   The No Implementation Results Available dialog box asks if synthesis and implementation can be launched before generating bitstream.

2. Click **Yes**.

   Bitstream generation can take several minutes to complete. Once bitstream generation completes, the Bitstream Generation Completed dialog box asks you to select what to do next.

3. Keep the default selection of **Open Implemented Design**.

4. Click **OK**.

5. After the implemented design opens, expand the Implemented Design drop-down list in the Flow Navigator and click on **Report Timing Summary**.

6. In the Report Timing Summary dialog box, click **OK.**

7. Verify that all timing constraints have been met by looking at the Timing - Timing Summary window (FIGURE 121).

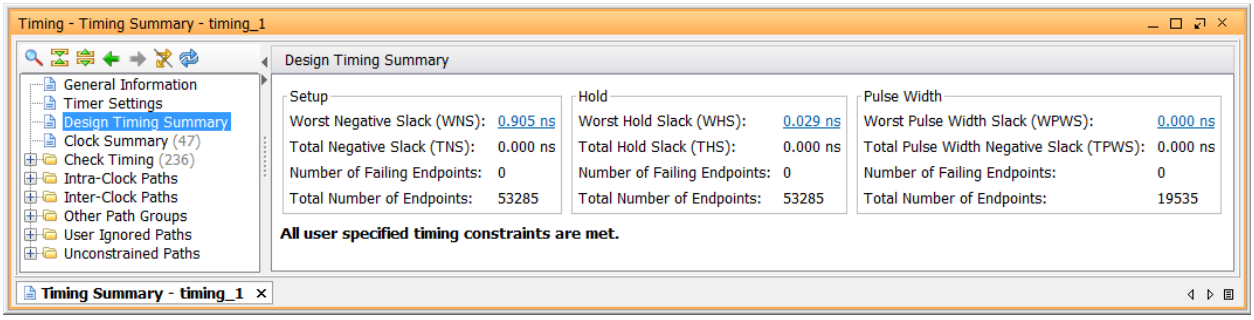

Figure 121: Timing Summary

Step 9: Exporting the Design to SDK

Next, open the design and export to SDK.

1. Select **File > Export > Export Hardware.**
2. In the Export to Hardware dialog box, select the **Include bitstream** check box (FIGURE 122). Make sure that the **Export to** field is set to **<Local to Project>**.

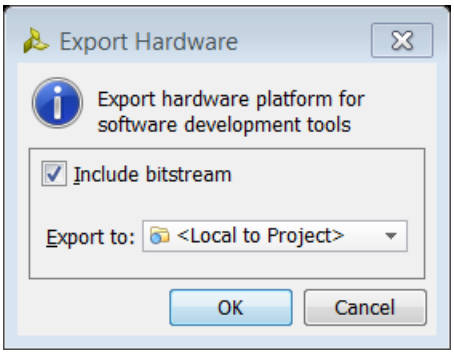

Figure 122: Export to Hardware Dialog Box

3. Click **OK**.
4. Select **File > Launch SDK**. In the Launch SDK dialog box, make sure that both the **Exported location** and the **Workspace** drop-down lists are set to **<Local to Project>**.

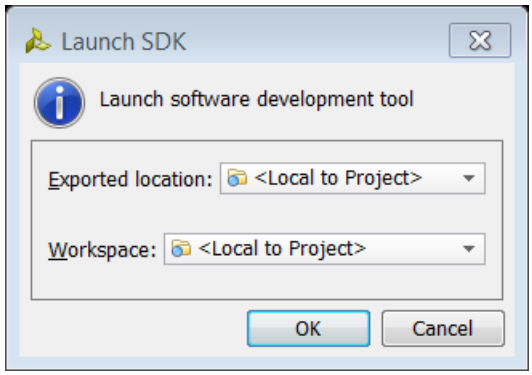

Figure 123: Launch SDK Dialog Box

5. Click **OK**.

   SDK launches in a separate window.

Step 10: Creating a "Hello World" Application

1. In SDK, right-click **mb_subsystem_wrapper_hw_platform_0** and select **New > Project** (FIGURE 124).

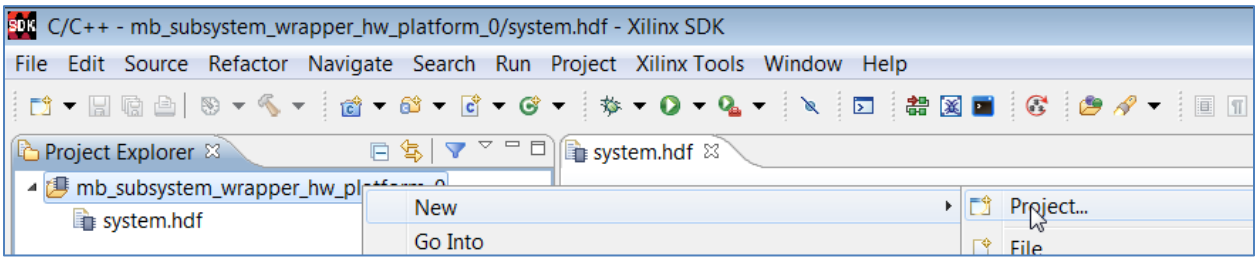

Figure 124: SDK New Project Selection

2. In the New Project dialog box, select **Xilinx Application Project** (FIGURE 125).
3. Click **Next**.

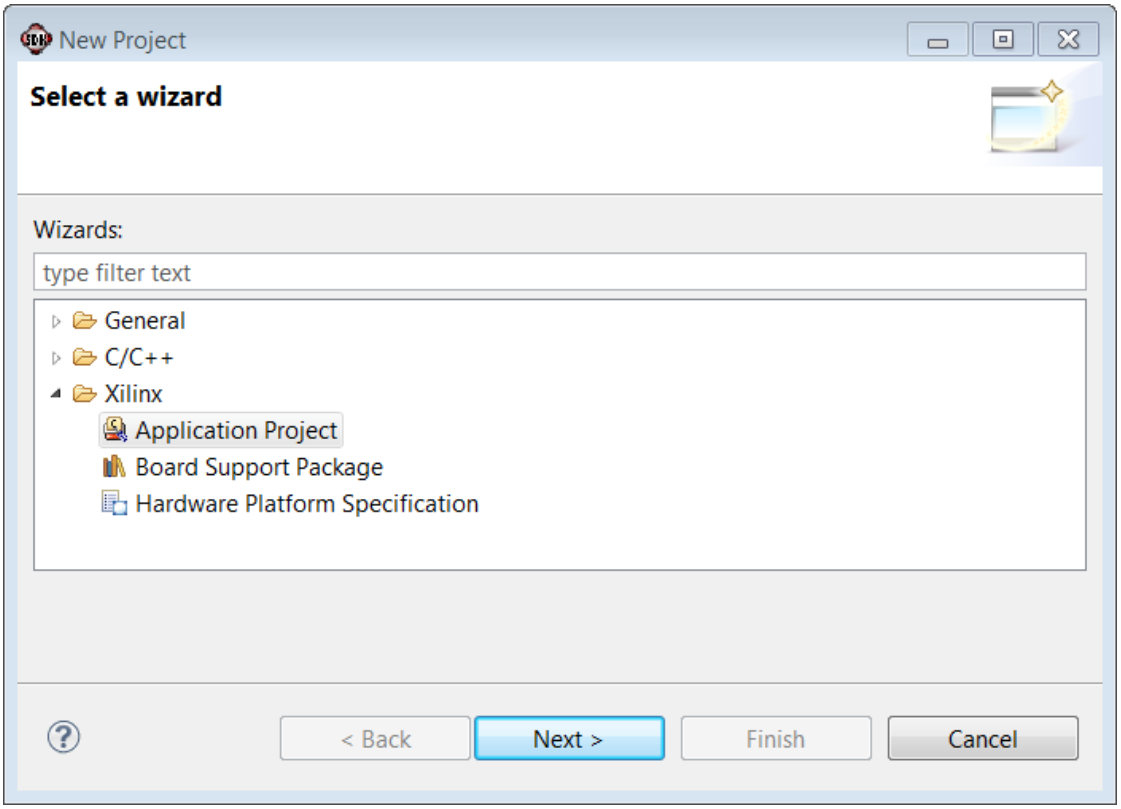

Figure 125: SDK New Project Wizard

4. Type a name (**hello_world**) for your project and choose **standalone** as the OS platform (FIGURE 126).

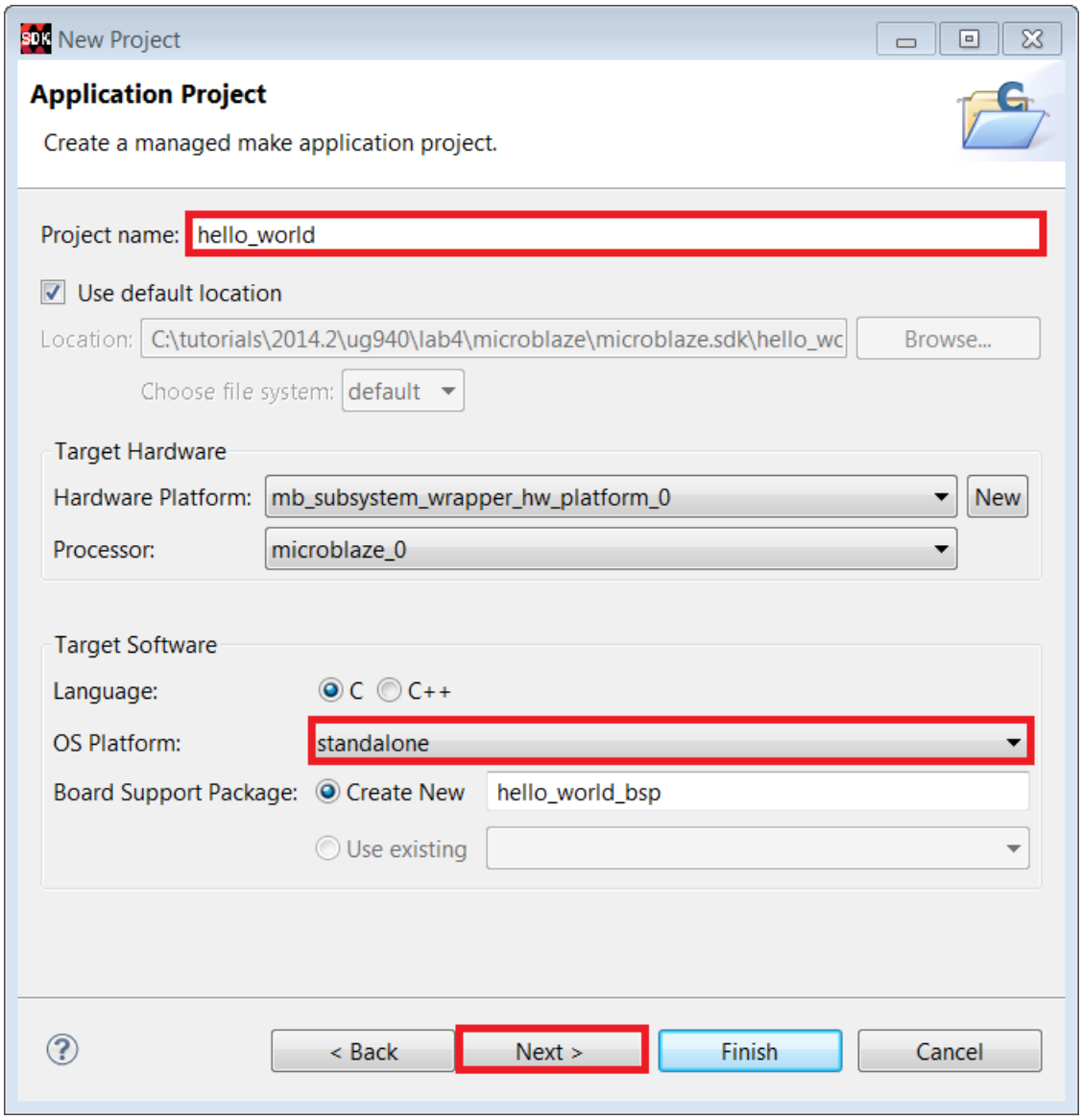

Figure 126: New Project: Application Project Wizard

5. Click **Next**.

6. Select the **Hello World** application template, and click **Finish** (FIGURE 127).

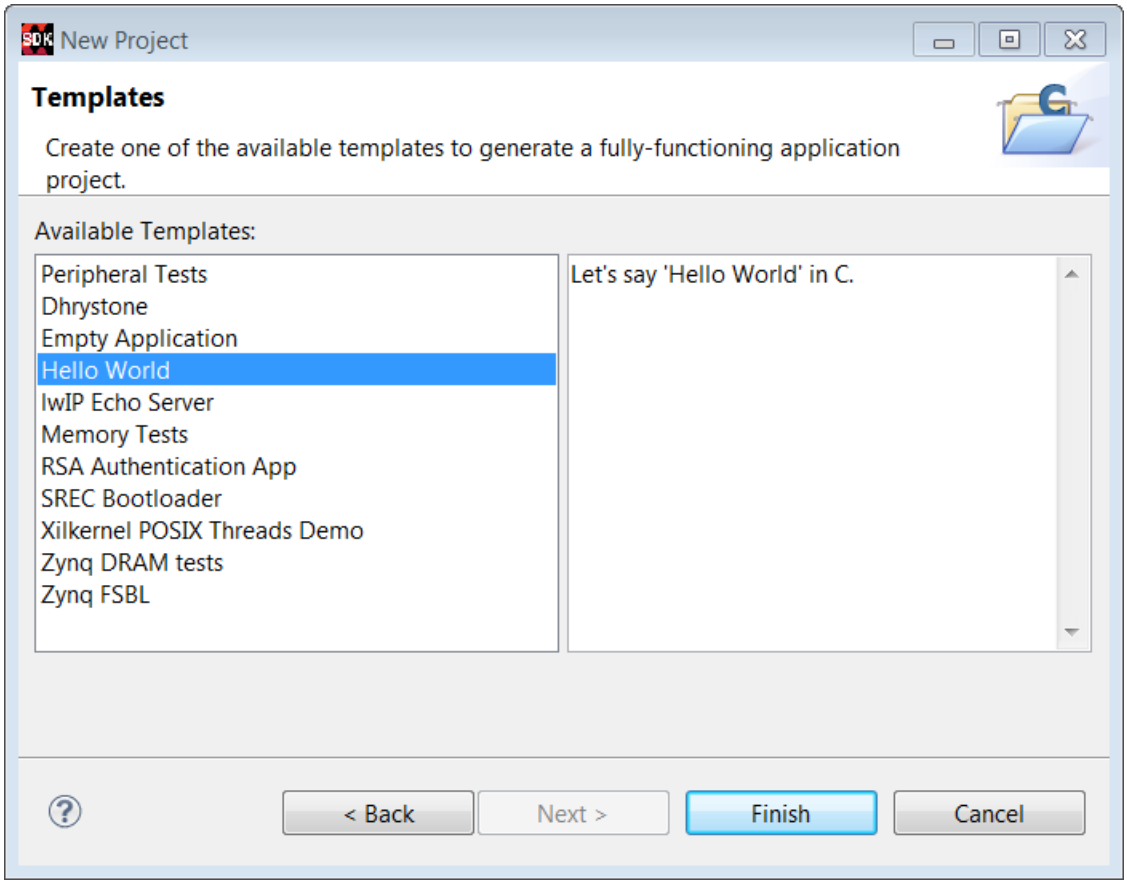

Figure 127: New Project: Template Wizard

SDK creates a new "Hello World" application.

7. Right-click the hello_world application in the Project Explorer and select **Generate Linker Script**.

   The Generate Linker Script dialog box opens.

8. Select the **Advanced** tab and change the Assigned Memory for Heap and Stack to **mig_7series_0**. To do this:

   a. Click in the **Assigned Memory** column in the appropriate row for heap and stack.

   b. Select **mig_7series_0** from the drop-down options.

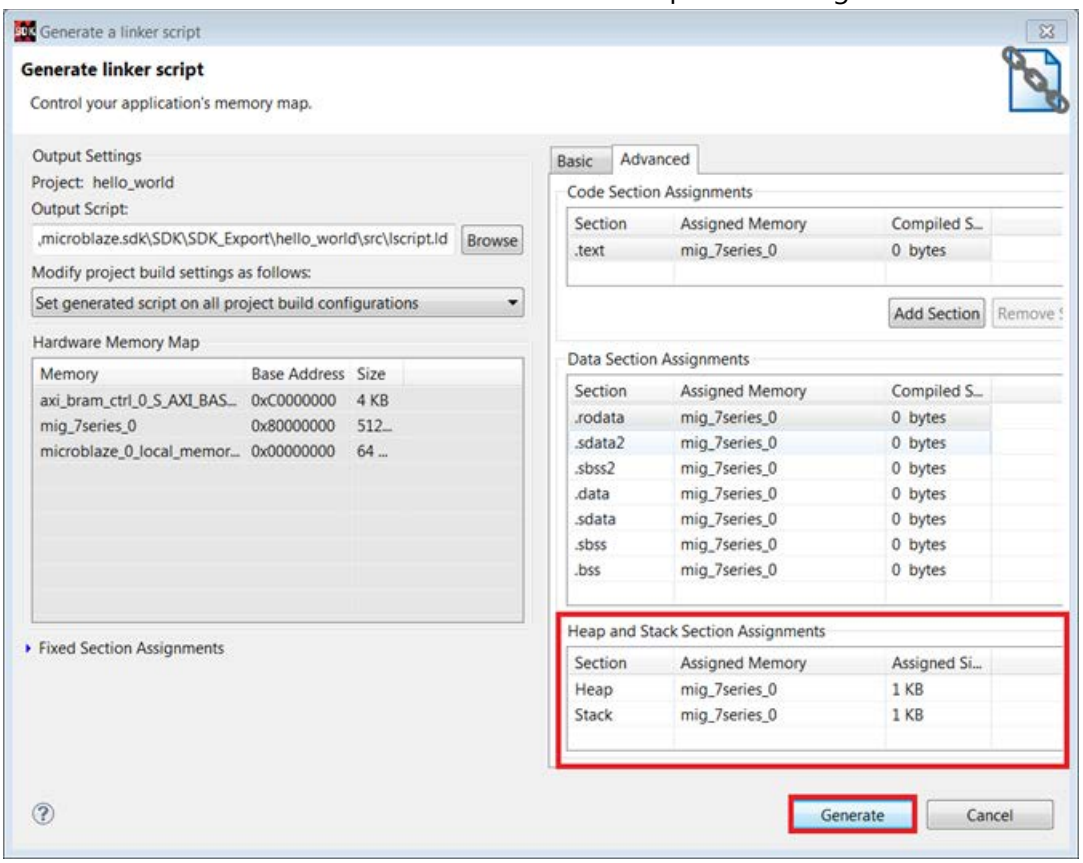

Figure 128: Generate Linker Script Dialog Box

Setting these values to mig_7series_0 ensures that the compiled code executes from the MIG.

A dialog box opens to notify you that the linker script already exists.

9. Likewise, change the Data Section Assignments and Code Section Assignments to **mig_7_series_0**.
10. Click **Generate**.
11. Click **Yes** to overwrite it.

Step 11: Executing the System on a KC705 Board

To run the design on a KC705 board:

1. Connect the board to your computer and switch on the board power.
2. Select the **Terminal 1** tab in SDK and click the **Settings** button .

   The Terminal Settings dialog box opens.
3. Specify the parameters as shown in **FIGURE 129**, and click **OK**.

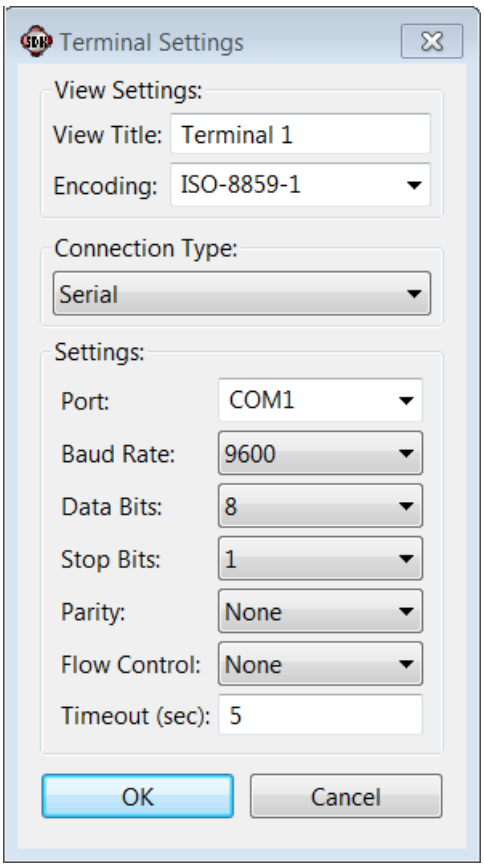

Figure 129: Terminal Settings Dialog Box

The Terminal 1 tab displays confirmation in the Terminal 1 tab that it is connected to the device (**FIGURE 130**).

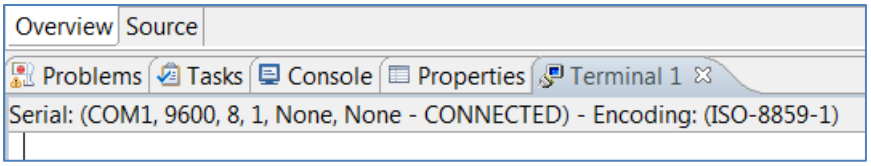

Figure 130: Terminal Connection Confirmation

1. Start the XMD Console by selecting **Xilinx Tools > XMD Console**.
2. Program the bitstream on the board using the following command in the XMD Console.

   ```
   Xmd% fpga -f <design path>/<project_name>/<project_name>.runs/
   impl_1/<ip_integrator_design_name>_wrapper.bit
   ```

   You can also program the FPGA by selecting **Xilinx Tools > Program FPGA**.
3. In the program FPGA dialog box, ensure the path to the bitstream is correct.
4. In the XMD Console, type: **`XMD% connect mb mdm`**

   Then, type: **`XMD% mbc`**
5. Reset and stop the MicroBlaze processor before running the software by using the **rst** and **stop** commands as shown:

   ```
   XMD% rst
   XMD% stop
   ```
6. Download the "`Hello World`" program ELF file of by typing:

   ```
   XMD% dow <project_path>/<project_name>/project_name.sdk/SDK/SDK_Export/
   hello_world/Debug/hello_world.elf
   ```
7. Run the program:

   ```
   XMD% run
   ```

   The output displays in the Terminal tab as shown in **FIGURE 131**:

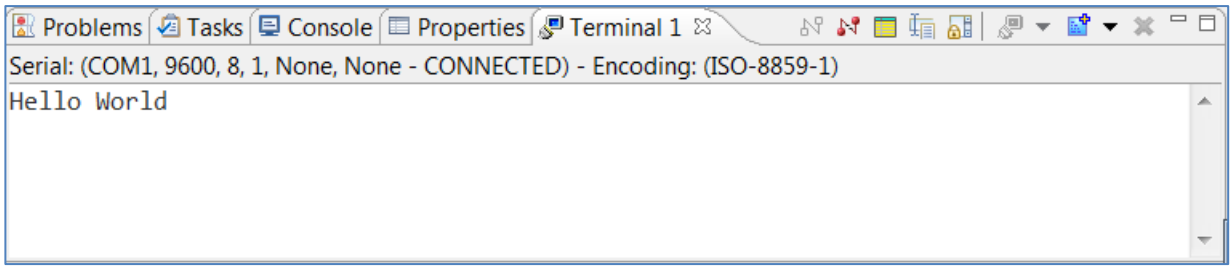

Figure 131: Terminal Tab

Conclusion

In this tutorial, you:

- Stitched together a moderately complex design in the Vivado IDE IP integrator tool
- Taken the design through implementation and bitstream generation
- Exported the hardware to SDK
- Created and modified an application code that runs on a Standalone Operating System.

Lab Files

The Tcl script `lab4.tcl` is included with the design files to perform all the tasks in Vivado. You will also need the `system.xdc` file to run in tcl mode. The SDK operations must be done in the SDK GUI. You might need to modify the Tcl script to match the project path and project name on your machine.

Lab 5: Converting Legacy EDK IP to Use in IP Integrator

Introduction

You might at some point need to use a legacy core from XPS in Vivado®. In this lab, you will learn how to convert an XPS processor core, or Pcore, to a Vivado Design Suite native IP for use in IP Integrator. To migrate a legacy core, you need all the subcores as well as libraries that the main core is dependent upon. This lab uses a simple GPIO pcore from an EDK project. The core has several dependencies on the following libraries:

- `proc_common_v3_00_a`
- `axi_lite_ipif_v1_01_a`
- `interrupt_control_v2_01_a`
- `axi_gpio_v1_01_b`

In order to migrate this pcore properly, you need to figure out all the files that are needed for the GPIO core, package them as sub-cores or libraries, add the sub-cores and/or libraries to the IP catalog and then finally, package the GPIO core.

Step 1: Managing IP

1. Launch the Xilinx® Vivado Design Suite IDE:

   Start > All Programs > Xilinx Design Tools > Vivado 2014.2 > Vivado 2014.2[1]

   As an alternative, click the **Vivado 2014.2** Desktop icon to start the Vivado IDE.

   The Vivado IDE Quick Start page, shown in **FIGURE 132**, contains links to open or create projects and to view documentation.

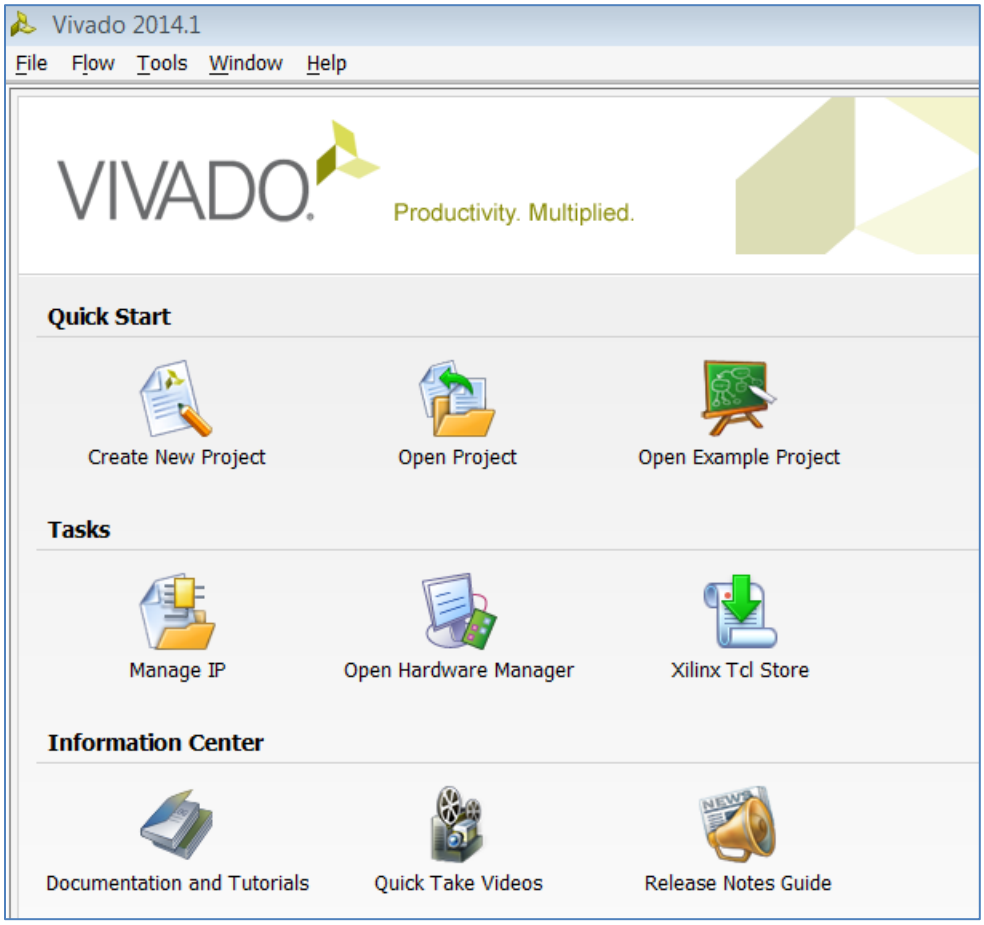

Figure 132: Vivado IDE - Quick Start Page

1 Your Vivado Design Suite installation might appear slightly differently on the **Start** menu.

2. Select **Manage IP** from the Quick Start page and select **New IP Location** from the drop-down menu.

   The Create a New Customized IP Location dialog box displays, as shown in FIGURE 133.

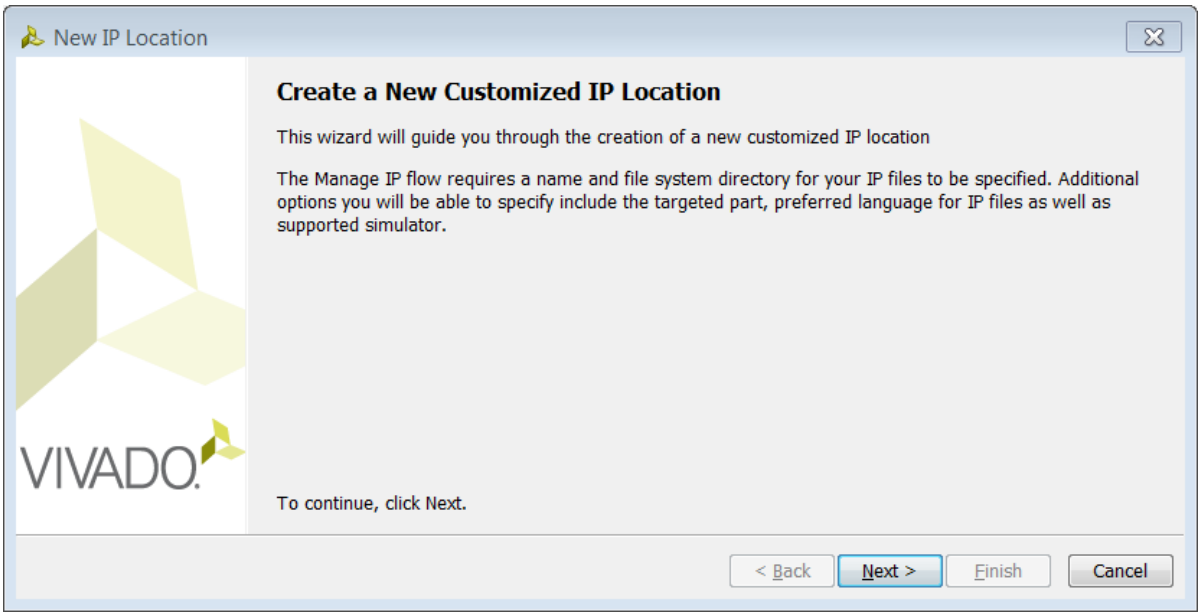

Figure 133: Open IP Catalog

3. Click **Next**.
4. On the Manage IP Settings page, click the **Browse** button ... to the right of the **Part** field.

5. The Select Device dialog box opens. Ensure that **Parts** is selected under **Specify** and type **xc7k325** in the search field. From the list of available devices, select the **xc7k325tffg900-1** part, and then click **OK** (FIGURE 134).

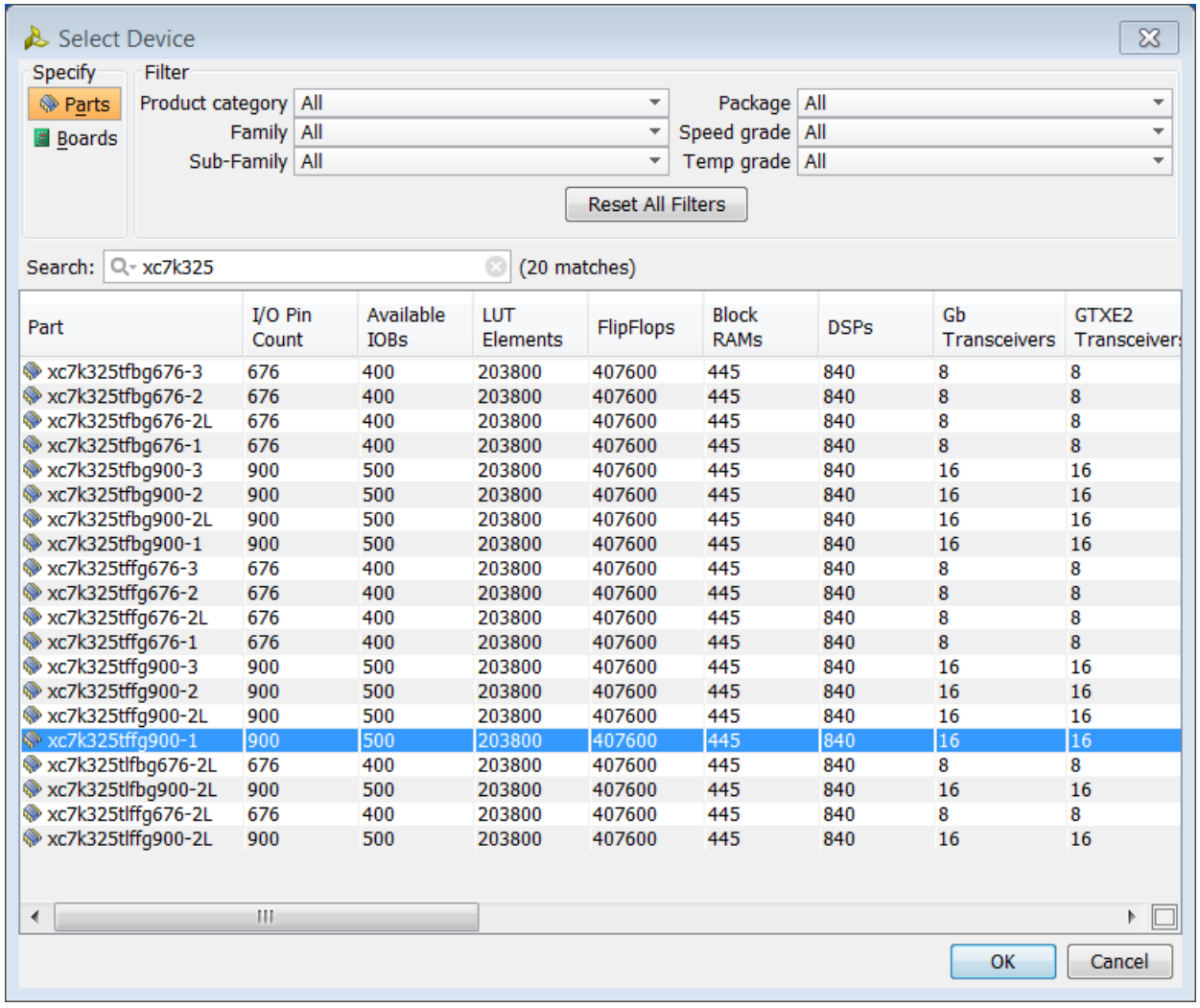

Figure 134: Select Part

6. Change the Target Language to **VHDL** (if not already selected) and Target Simulator as **Vivado Simulator**. Leave the Simulator language to **Mixed**. Change the Default IP Location to: **<Extract_Dir>/lab5** (FIGURE 135).

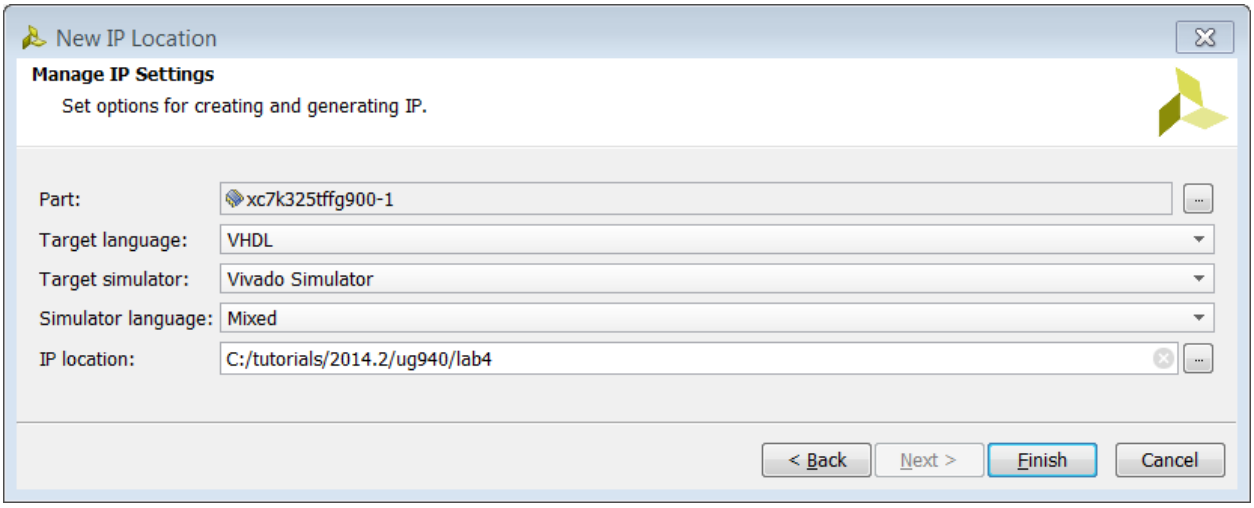

Figure 135: New IP Location Dialog Box

7. Click **Finish.**

   The Vivado IDE loads the IP Catalog view layout (FIGURE 136).

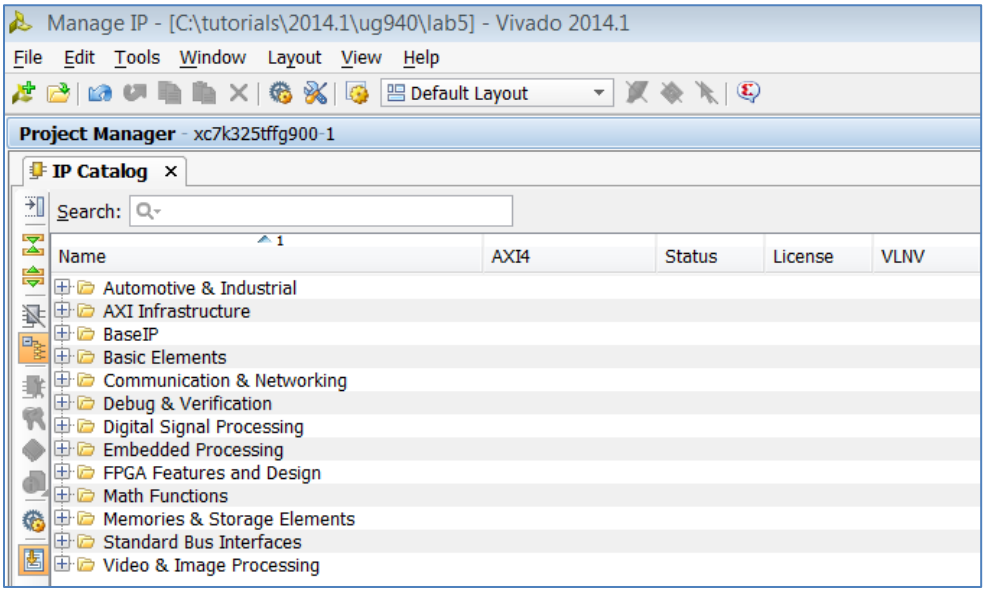

Figure 136: IP Catalog View Layout

Because this is a Managed IP session and not an RTL project, you cannot perform behavioral simulation, elaborate the design, or launch synthesis or implementation. This view is strictly for exploring the IP Catalog, and for packaging designs for use as IP.

Step 2: Packaging a Sub-Core

As mentioned earlier in this guide, the GPIO core needs several sub-core references. We start by packaging the sub-cores and libraries before we get to packaging the GPIO core.

1. Select **Tools > Create and Package IP**.

   The Create and Package New IP dialog box opens (FIGURE 137).

Figure 137: Create and Package IP dialog box

2. Click **Next**.

   The Choose Create Peripheral or Package IP page opens (FIGURE 138). Package a specified directory is selected by default.

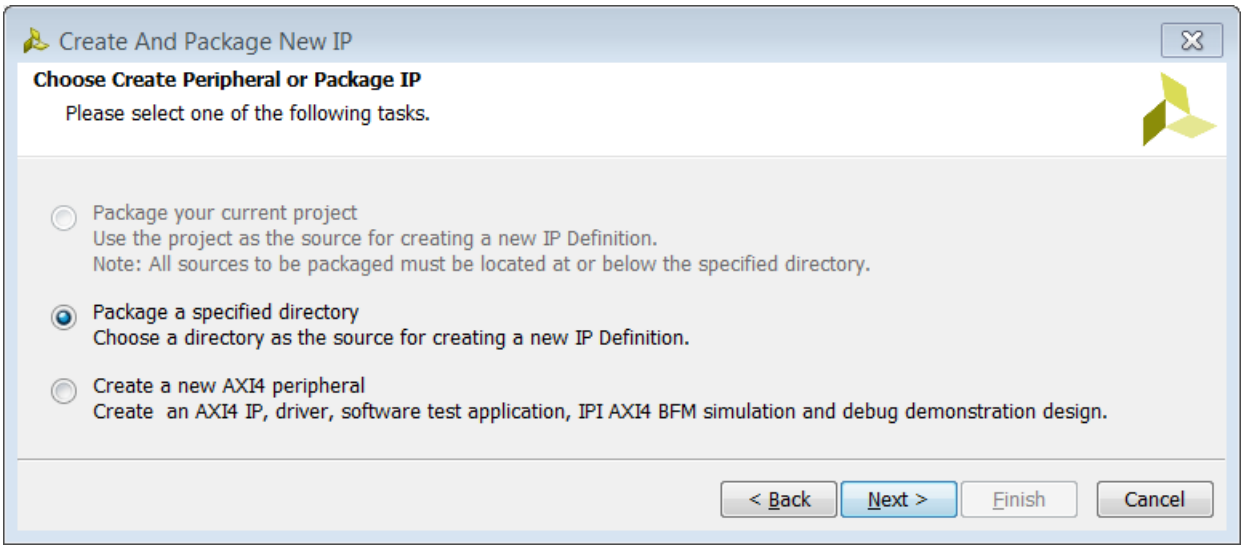

Figure 138: Package a Specified Directory as a Library Core

3. Click **Next**.
4. In the next page of the Create and Package New IP dialog box, Package Your Current Project page, browse to the location of the IP to be packaged (`<extracted_folder>/lab5/proc_commmon_v3_00_a`), and check the **Package as a library core** option.
5. Click **Next**.

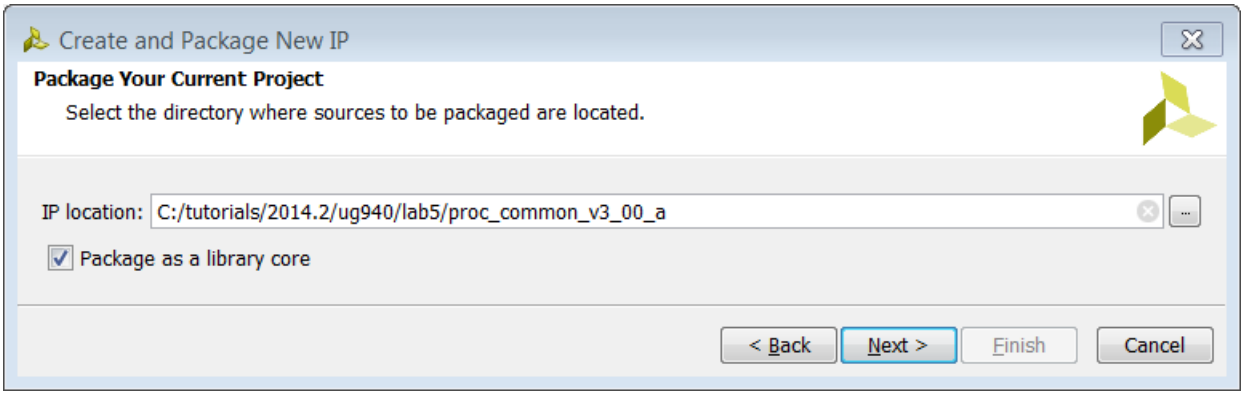

Figure 139: Package Your Current Project Page

6. In the Edit in IP Packager Project Name, leave everything to its default value and click **Next**.

   The Edit in IP Packager Project Name window opens (FIGURE 140).

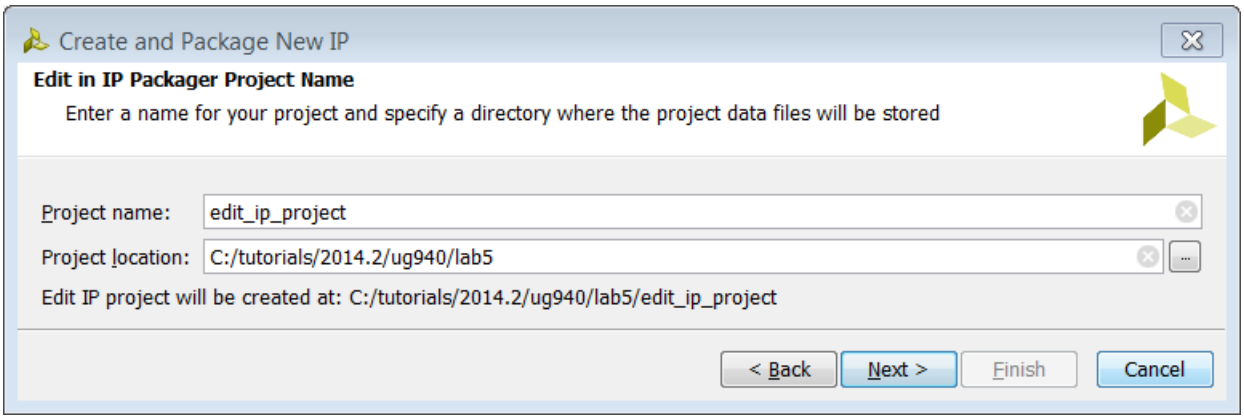

Figure 140: Edit in IP Packager Project Name page

7. In the Begin IP Creation page (FIGURE 141), review the information and click **Finish**.

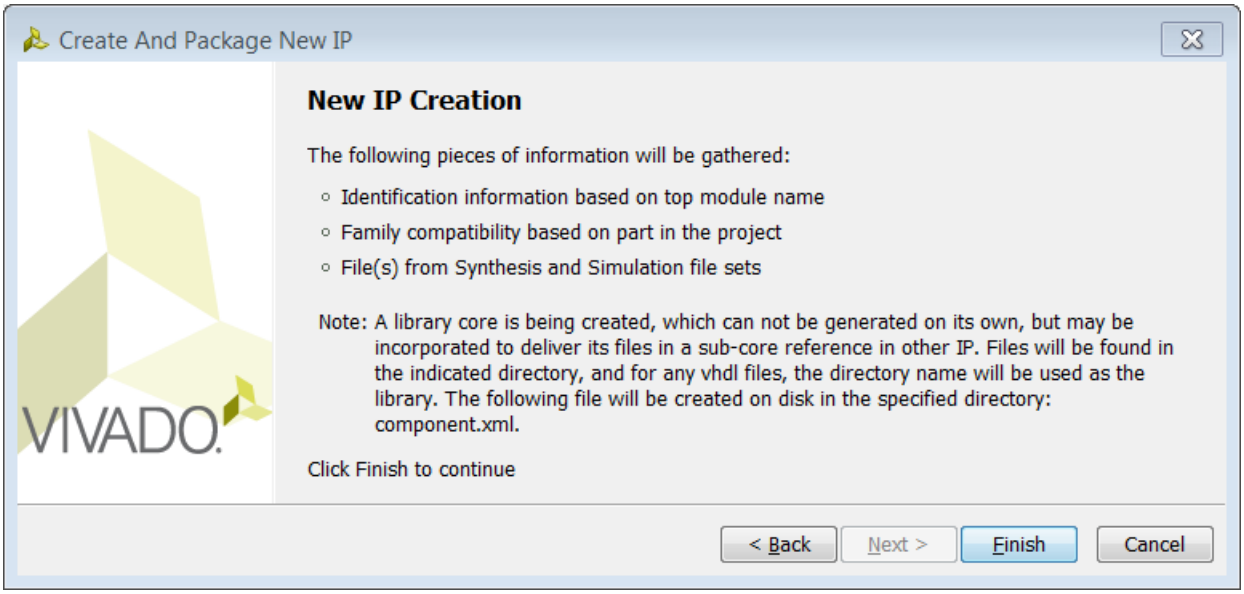

Figure 141: Begin IP Creation page

The Package IP Summary page opens.

8. Click **OK**.

The edit_ip project opens in a separate window.

9. Select the **Package IP** tab (FIGURE 142).

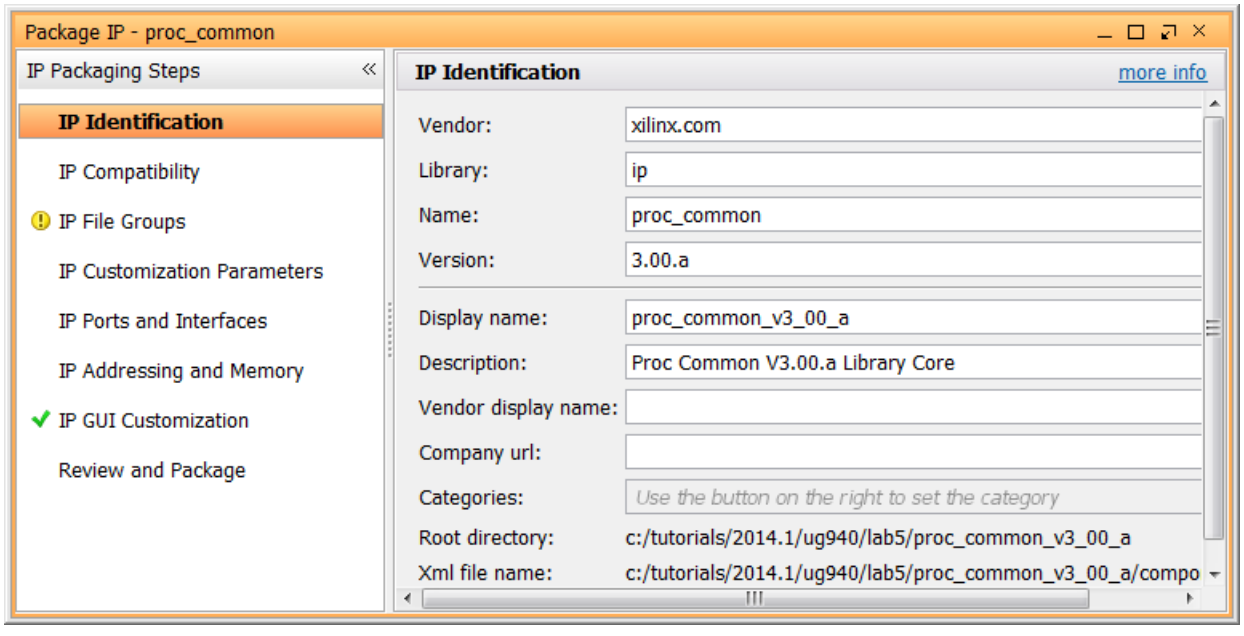

Figure 142: Package IP Tab

10. Select IP identification (if not already selected) and fill in the following fields:
    - **Display name**: proc_common_v3_00_a
    - **Description**: Proc Common V3.00.a Library Core
11. On the left side of the window, click on **Review and Package**.
12. Click Package IP (FIGURE 143).

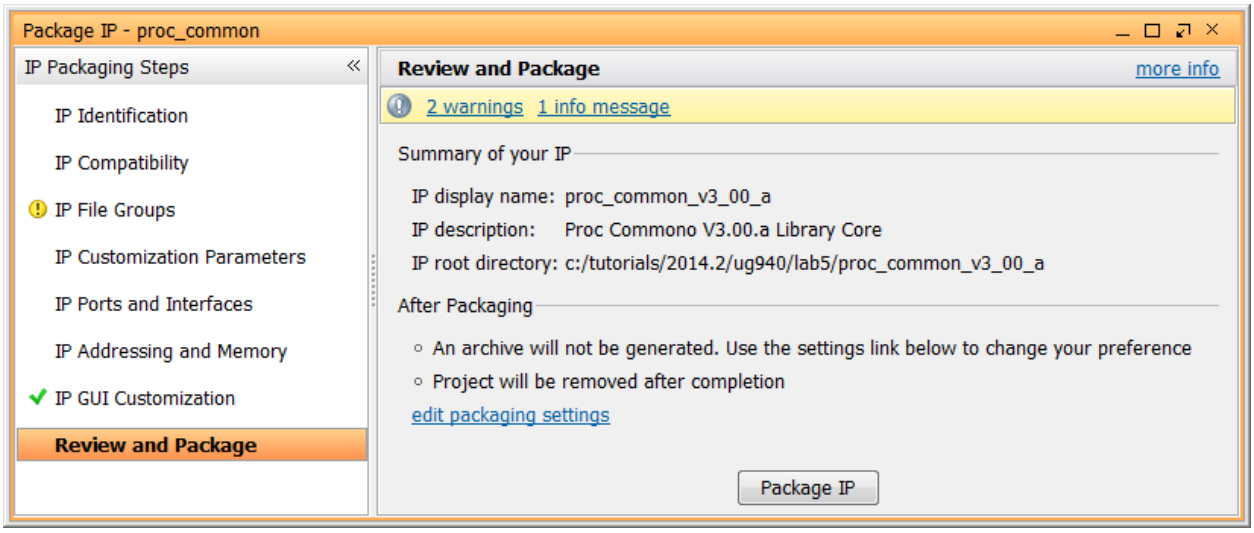

Figure 143: Review and Package the Library Core

The edit_ip_1 project closes and takes you back to the Manage IP session in Vivado.

13. Right-click anywhere in the **IP Catalog** and select **IP Settings** (FIGURE 144).

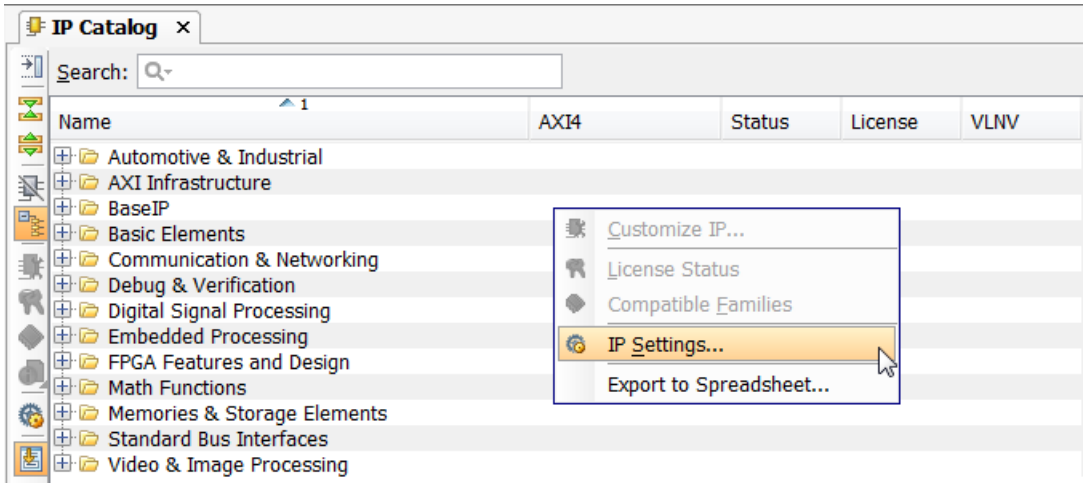

Figure 144: Ensure that IP Repository has been Updated with the Packaged Library Core

The Project Settings dialog box opens (FIGURE 145).

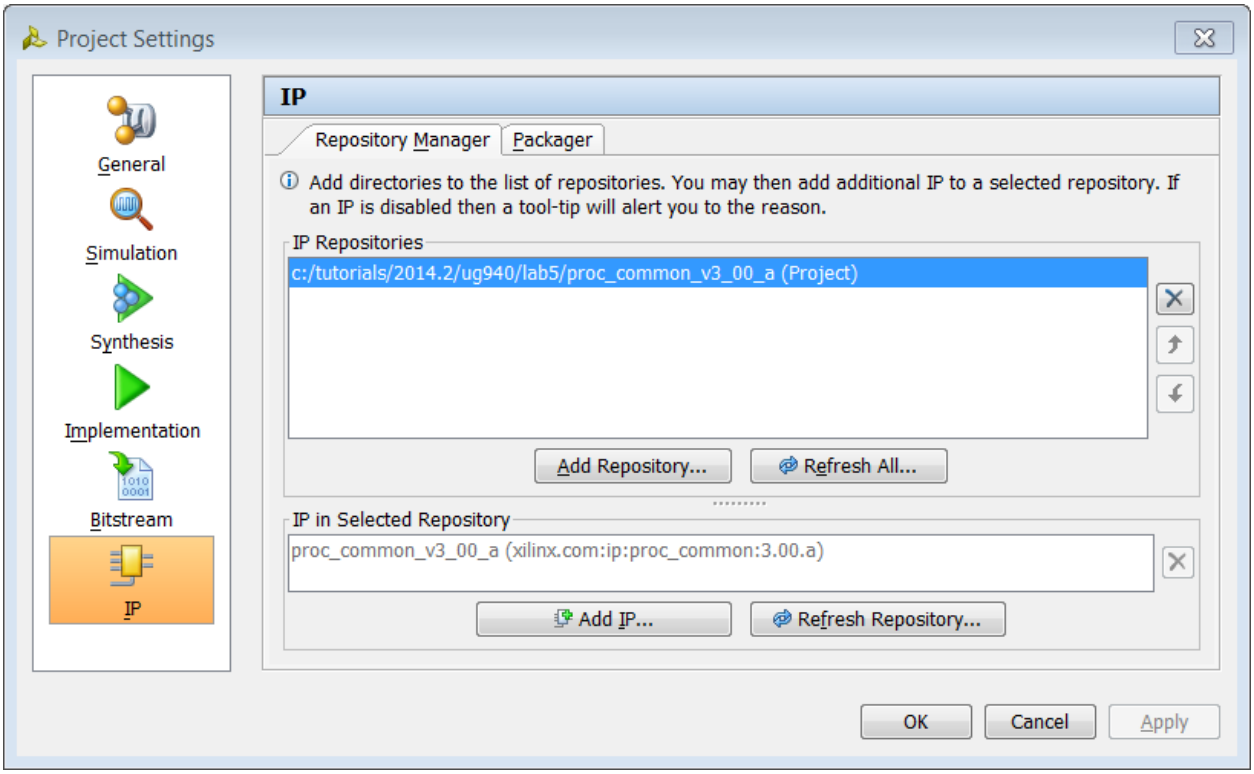

Figure 145: Ensure that IP Repository is Set Correctly and Packaged Library is Found

14. Ensure that the IP Repositories points to the newly packaged library folder and that the IP proc_common_v3_00_a is found in the IP in Selected Repository field.
15. Click **OK** to close the dialog box.
16. Repeat these steps to package the axi_lite_ipif_v1_01_library and the interrupt_control_v2_01_a library. Make sure to specify the Display Name and Description for each of these subcores as follows:

Subcore	Name	Description
axi_lite_ipif	axi_lite_ipif_v1_01_a	AXI Lite IPIF v1.01.a Library Core
interrupt_control	interrupt_control_v2_01_a	Interrupt Control V2.01.a Library Core

Step 3: Package the GPIO Core

Now that all the libraries have been packaged up properly, you can package the GPIO core.

1. Select **Tools > Create and Package IP**.

   The Package New IP wizard displays and provides information about the process you are starting.

2. In the Welcome to the Create and Package IP page, click **Next** to continue (FIGURE 146).

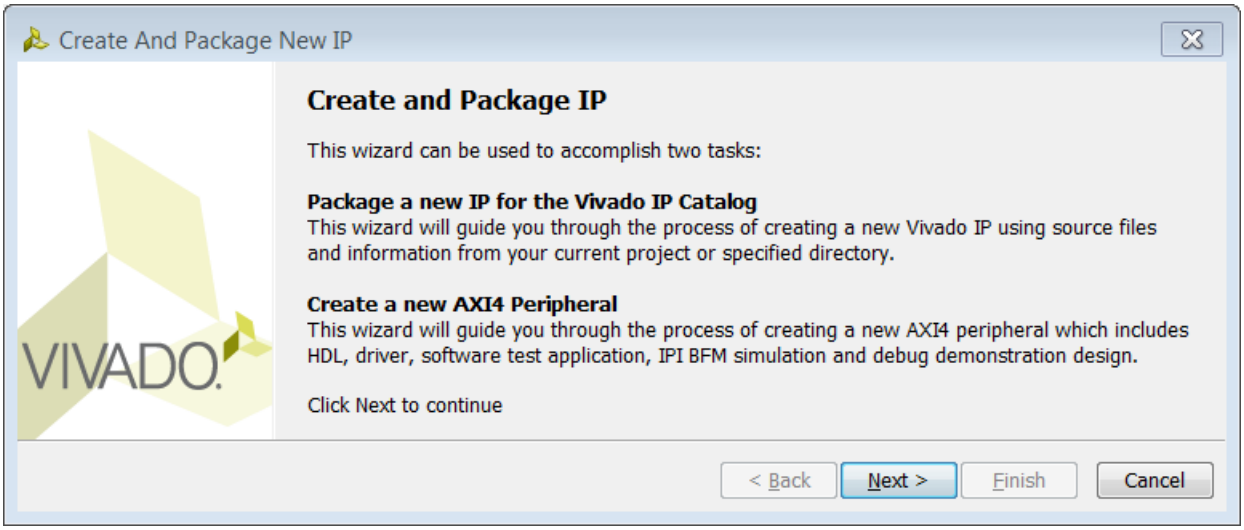

Figure 146: IP Packager Dialog Box

The Choose Create Peripheral or Package IP window opens (FIGURE 147).

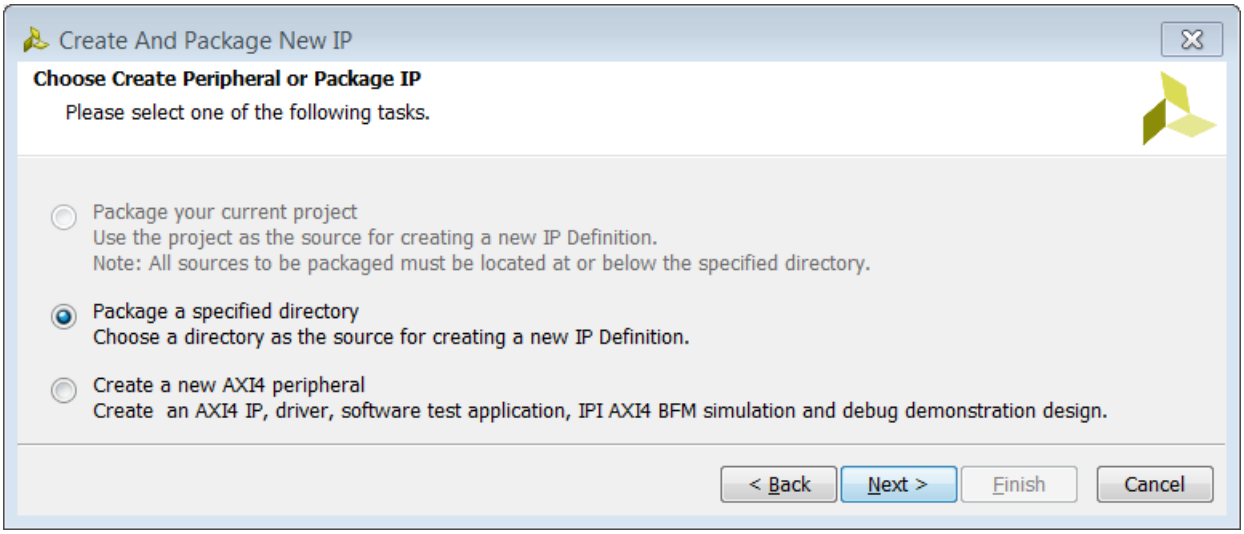

Figure 147: Setting the IP Source Location

3. In the Choose IP Source Location dialog box, select **Package a specified directory**.

4. Click **Next**.
5. In the Package Your Current Project page, in the IP Definition Location, click the **Browse** button and browse to `<Extract_Dir>/lab_5/axi_gpio_v1_01_b`, as shown in FIGURE 147.
6. Click **Next**.

   Note that this core is not being packaged as library core, so the **Package as a library core** check box is not checked.

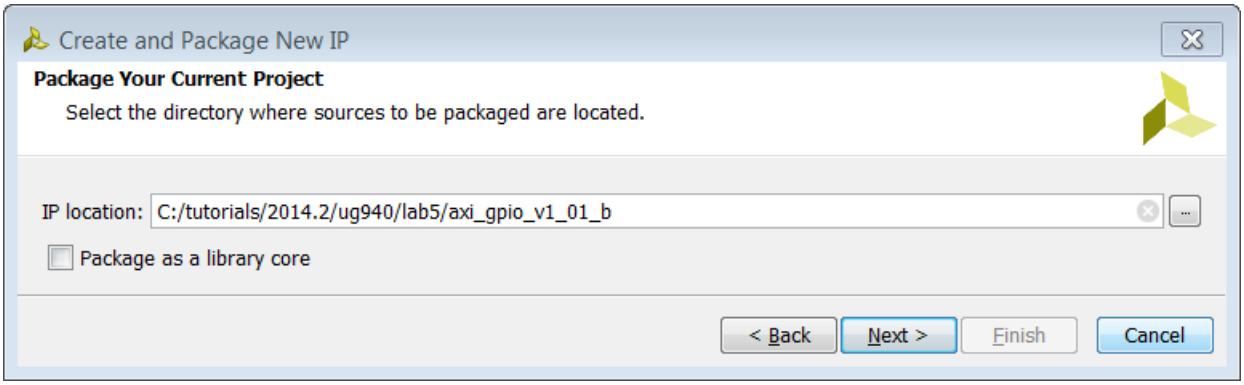

Figure 148: Package Your Current Project Dialog Box

   The Edit in IP Packager Project Name dialog box opens. By default the project name is **edit_ip_project**. You can choose to rename your project here or use the default name.

7. Click **Next**.

   The New IP Creation page displays, showing what work is performed during the IP packaging flow (FIGURE 149).

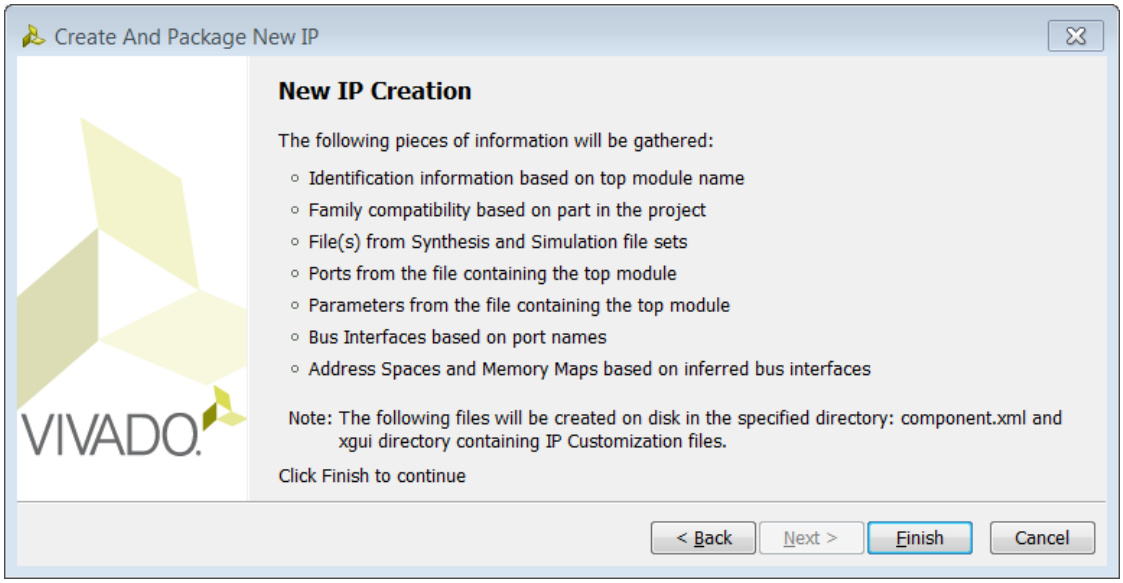

Figure 149: Begin IP Creation Page

8. Click **Finish**.

   The Package IP wizard collects the available information from the specified IP location, and the **IP Packager Summary** screen summarizes what is gathered and populated into the `component.xml` file, a newly created IP-XACT definition of the IP.

   Note: *If a `data/*.pao` file is present in the specified IP location, this file is read, and the file and associated library information are used, as is the case for this EDK Pcore IP.*

9. Click **OK** to close the Package IP dialog box.

10. Review the various sections of the Package IP window, as seen in **FIGURE 150**, and update information as required. The Package IP window shows the current IP identification, including the Vendor, Library, Name, and Version (VLNV) attributes of the newly packaged core.

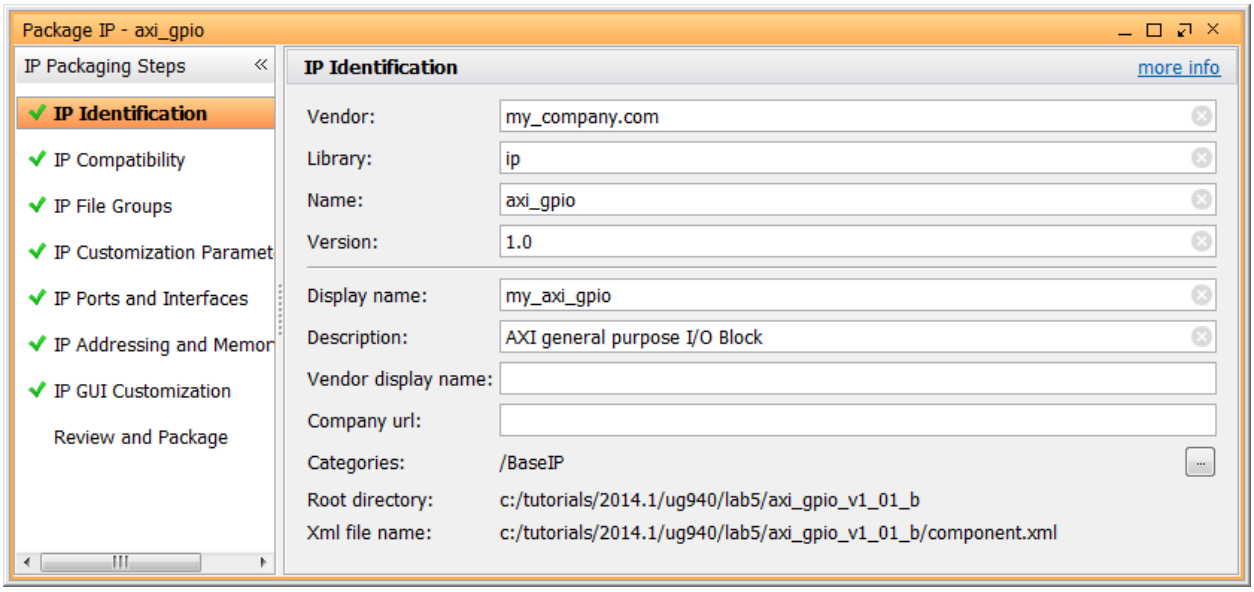

Figure 150: Package IP - IP Identification

11. Make the following changes:
    - **Vendor**: my_company.com
    - **Display Name**: my_axi_gpio
    - **Description**: AXI general purpose I/O block

    As with any design that you package as an IP, you can provide company identification such as name and web site, as well as add supporting documentation, examples, testbench, and so on.

CAUTION! Make sure that Vendor, Library, Name and Version fields are correct. If there is a more recent version of the IP in the repository, the packaged IP will not show up in the IP Catalog.

---

12. Click IP Compatibility to see the family of device supported for this project (FIGURE 151).

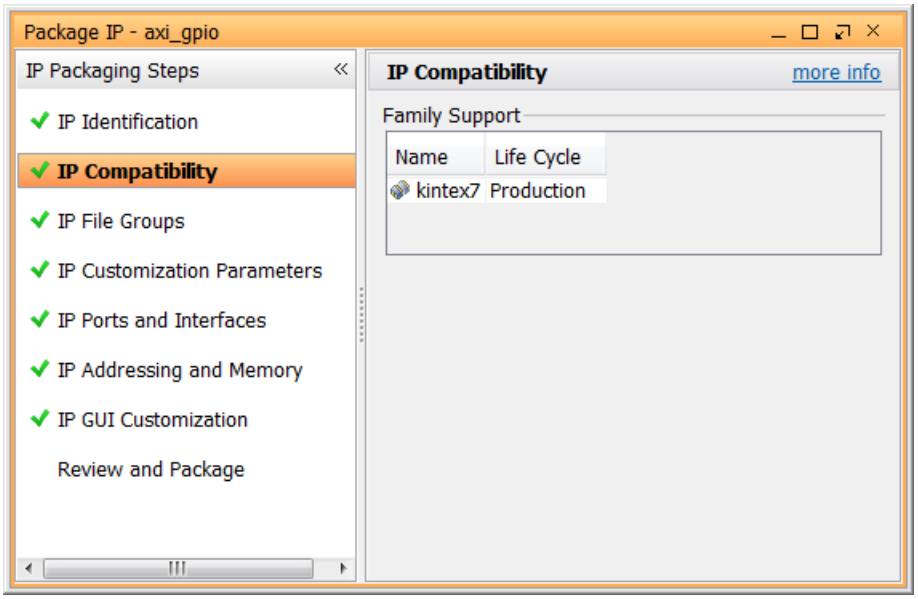

Figure 151: IP Compatibility Page

13. Click **IP File Groups** to see the source files included for synthesis as well as simulation (FIGURE 152).

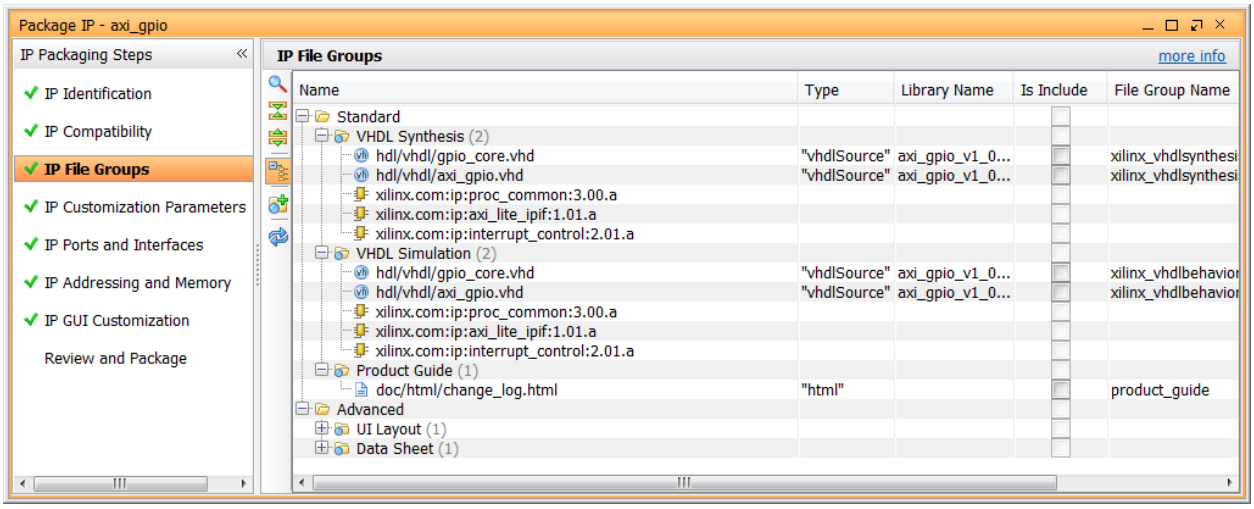

Figure 152: IP File Groups Page

14. Click IP Customization Parameters to open the IP Customization Parameters page (FIGURE 153).

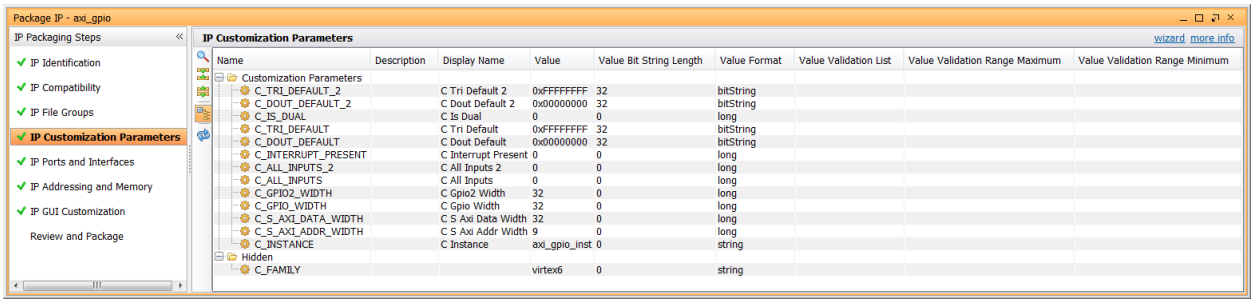

Figure 153: IP Customization Parameters Page

Closely examine the IP Customization Parameters page. This page shows all the parameters that can be customized for this IP.

Consider the address range for the core: **C_S_AXI_ADDR_WIDTH**.

Is the address range specified as a fixed value, or is it configurable at the time the IP is customized? If it is configurable, what is the reason? Is the IP acting like a bridge with a variable-sized memory window on the slave side? Or, does it have a variable-sized memory buffer for storing information?

Most IP have a fixed set of registers, and this parameter is meaningless in the context of the IP Integrator because the HDL does not accommodate a variable memory range. This parameter made sense in XPS/EDK because of how that software worked. However, in IP integrator, the interconnect handles this, and the bits are directly related to the range.

15. Click **IP Ports** to see all the I/O ports in the design (FIGURE 154).

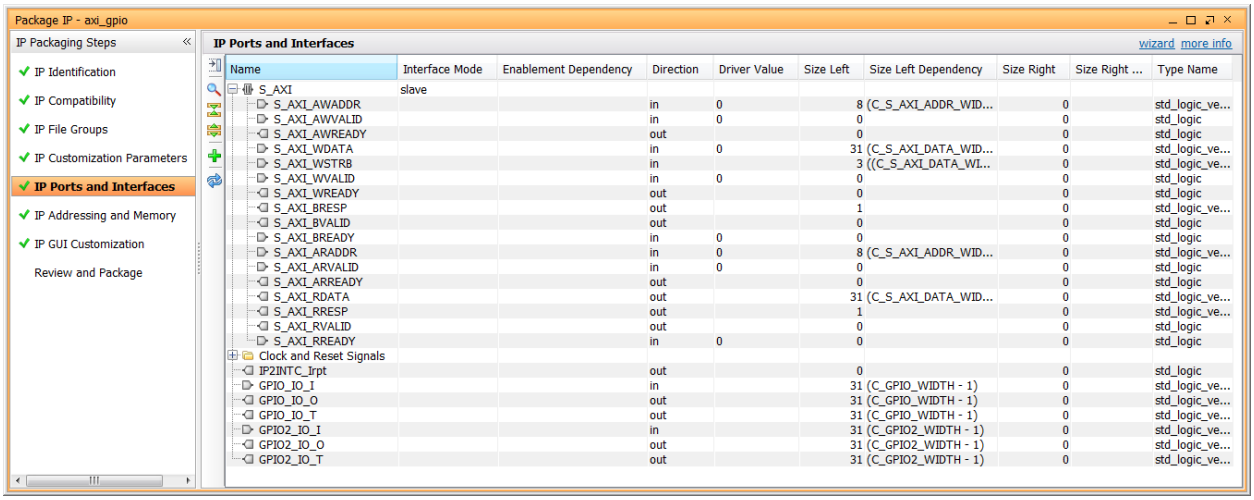

Figure 154: IP Ports Page

16. Click **IP Addressing and Memory** to see the memory mapping of any registers in the IP (**FIGURE 155**).

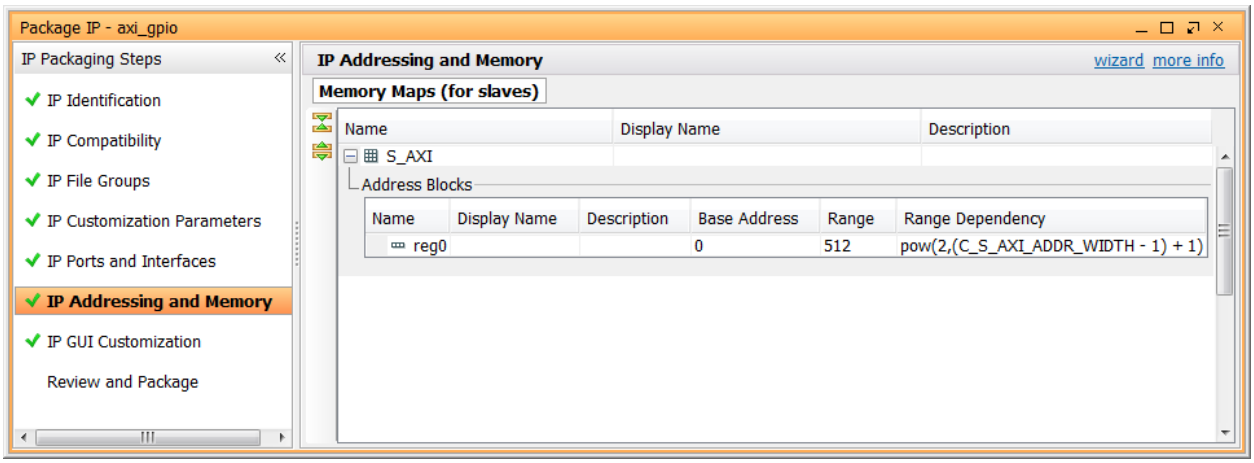

Figure 155: IP Addressing and Memory Page

17. Click **IP GUI Customization** to see a preview of how the IP will look during customization and when added to an IP integrator subsystem design (**FIGURE 156**).

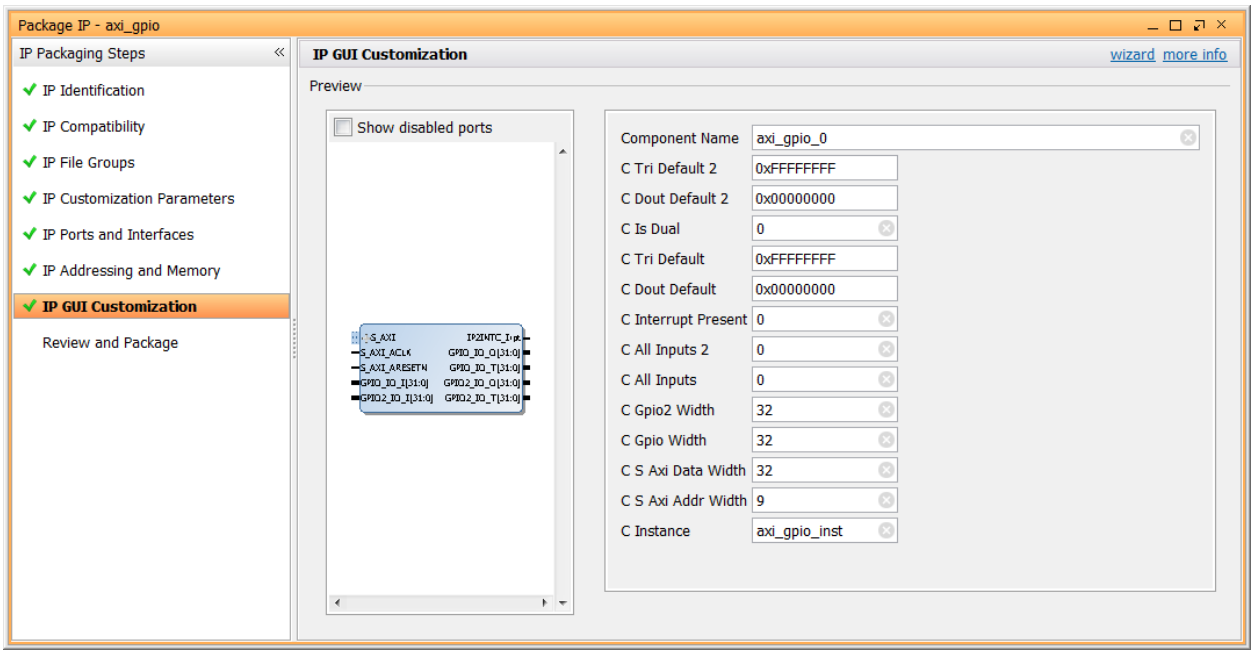

Figure 156: IP GUI Customization Layout

18. Click **Review and Package** for a summary of the IP (**FIGURE 157**).

19. Click **Package IP**.

    The IP definition file, `component.xml`, is saved automatically as you make changes to the different pages of the Package IP window.

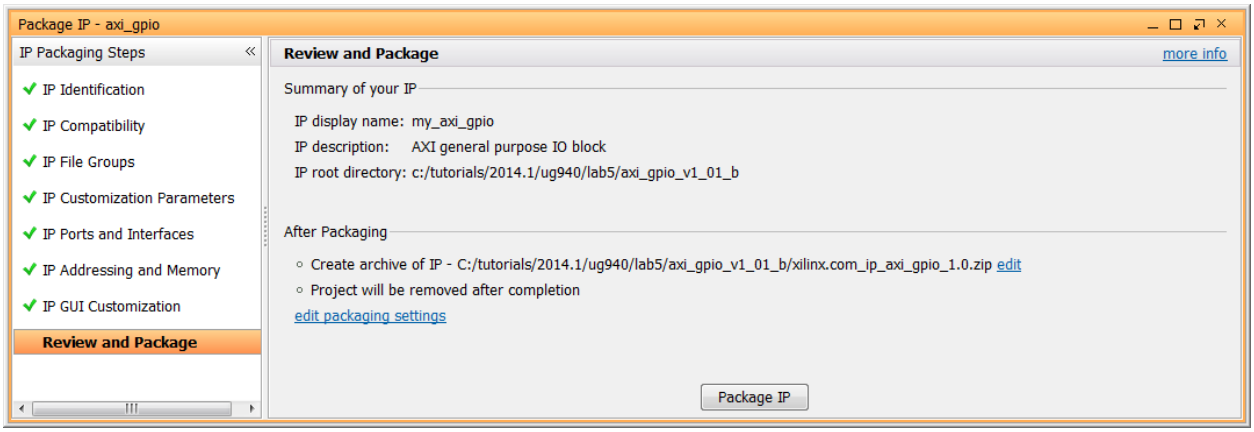

Figure 157: Review and Package IP Page

RECOMMENDED: *To use a custom IP repository across multiple design projects, or for use by multiple designers, you can add the repository when the Vivado tool launches by adding it to your init.tcl script. Refer to the Vivado Design Tcl Command Reference Guide (**UG835**) for more information on the init.tcl script.*

---

20. Click the **IP Catalog** tab and look for **my_axi_gpio** in the **BaseIP** category. Review the details (FIGURE 158).

    You have created an IP-XACT definition file for the IP, which can now be used in a Vivado IP Integrator subsystem design.

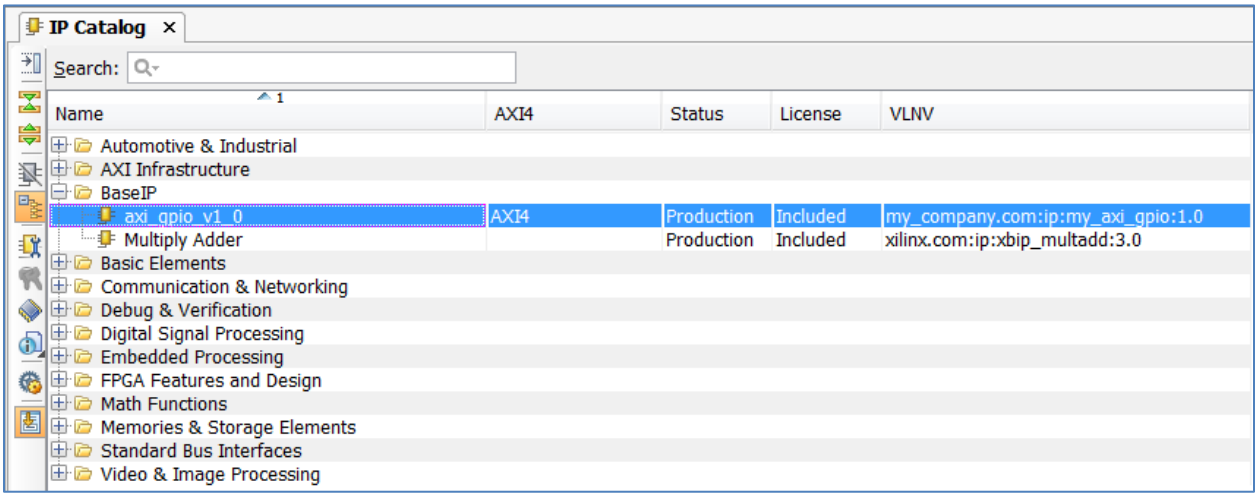

Figure 158: IP Catalog - my_axi_gpio

Conclusion

In this exercise, you learned how to create an IP Integrator IP from an existing EDK Pcore. The process involved creating an IP-XACT definition file, `component.xml`, via the Package IP wizard. You can complete the through the Manage IP flow, working directly with the Pcore, or within your design project.

In a different approach to this exercise, you could edit a Vivado project that included an EDK Pcore, and then create a native Vivado Design Suite IP from the Pcore for use in the Vivado IP Integrator.

You could:

- Launch the Package IP wizard from within the current project
- Reference the location of the existing Pcore as the IP repository
- Follow the steps of this lab to package the IP
- Add the new IP repository to the IP Catalog
- Use the new IP in your current project to replace the existing Pcore

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 the Limited Warranties which can be viewed at **http://www.xilinx.com/warranty.htm**; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: **http://www.xilinx.com/warranty.htm#critapps**.