Vivado Design Suite Tutorial

Embedded Processor Hardware Design

UG940 (v2015.3) September 30, 2015

Revision History

The following table shows the revision history for this document.

Date	Version	Changes
09/30/2015	2015.3	Validated with release.
06/24/2015	2015.2	Validated with release.
04/01/2015	2015.1	Validated and updated for Vivado® Design Suite 2015.1 release. Updated figures to reflect 2015.1. Modified Lab 4 to include creating a cross-trigger for MicroBlaze.

Table of Contents

Overview

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 tool to build a processor design, and then debug the design with the Xilinx® Software Development Kit (SDK) and the Vivado Integrated Logic Analyzer.

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

Hardware and Software Requirements

This tutorial requires that the 2015.2 Vivado Design Suite software (System Edition) release is installed. See the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973) for a complete list and description of the system and software requirements.

The following Platform Boards and cables are also needed:

- Xilinx Zynq-7000 AP SoC ZC702 board for Lab 1, Lab 2 and Lab 3
- Xilinx Kintex – 7 KC705 board for Lab4
- One USB (Type A to Type B)
- JTAG platform USB Cable or Digilent Cable
- Power cable to the board

Tutorial Design Descriptions

No design files are required for these labs, if step-by-step instructions are followed as outlined. However, for subsequent iterations of the design or to build the design quickly, TCL files for these labs are provided. For cross-probing hardware and software, manual interaction with Vivado and Platform boards is necessary. No TCL files are provided for that purpose.

Lab 1: Programming a Zynq-7000 AP SoC Processor

Lab 1: Programming a Zynq-7000 AP SoC Processor uses the Zynq-7000 AP SoC 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

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-7000 AP SoC PS.

After you construct the design, you mark nets for debugging to enable debug of the logic. Then you generate the Hardware Design Language (HDL) for the design as well as for the IP. Finally, you implement the design and generate a bitstream, export the hardware description of the design to the Software Development Kit (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.

Lab 2: SDK and Logic Analyzer

Lab 2: Using SDK and the Vivado IDE Logic Analyzer 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 AP SoC processors. Then you perform logic analysis on the design with a connected board.

Lab 3: Zynq-7000 AP SoC Cross Trigger Design

Lab 3: Zynq-7000 AP SoC Cross Trigger Design 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 AP SoC processors. Then, use the cross-trigger feature of the Zynq-7000 AP SoC processor to perform logic analysis on the design on the target hardware.

Design Files

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

- `lab3.tcl`

See Locating Tutorial Design Files.

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.

Design Files

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

- `lab4.tcl`

See Locating Tutorial Design Files.

Locating Tutorial Design Files

Design data is in the associated Reference Design File.

This document refers to the design data as `<Design_Files>`.

Introduction

In this lab you create a Zynq®-7000 AP SoC 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 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**, as shown in the figure below:

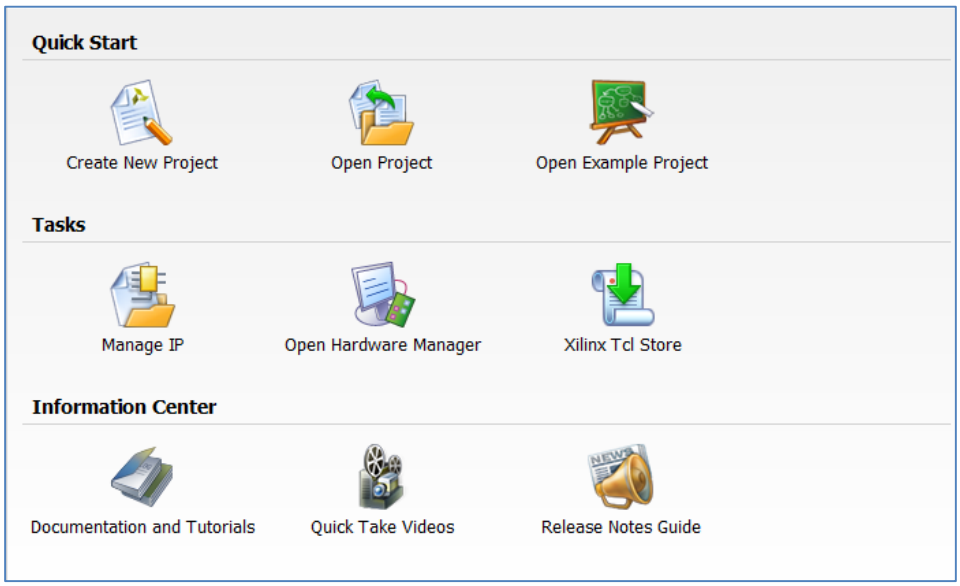

Figure 1: Vivado Quick Start Page

The New Project wizard opens.

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:
10. Select **Boards.**
11. 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.*

12. Choose the version of the **ZYNQ-7 ZC702 Evaluation Board** that you are using.
13. 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> IP Integrator**, select **Create Block Design**.
2. In the Create Block Design dialog box, specify a name for your IP subsystem design or type the design name `zynq_design_1`. Leave the Directory field set to the default value of <Local to Project>, and leave the Specify source set field to its default value of Design Sources.

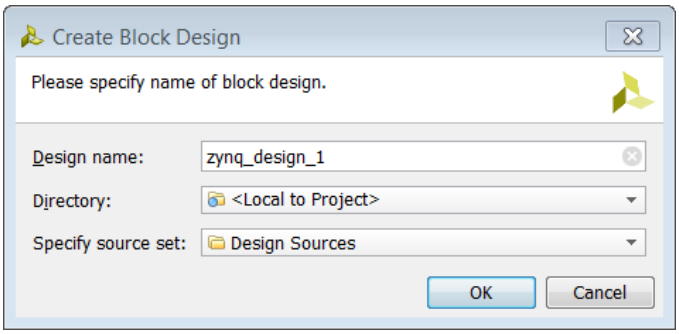

Figure 3: Create Block Design Dialog Box

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

   Alternatively, you can click the **Add IP** icon in the IP integrator canvas.

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.

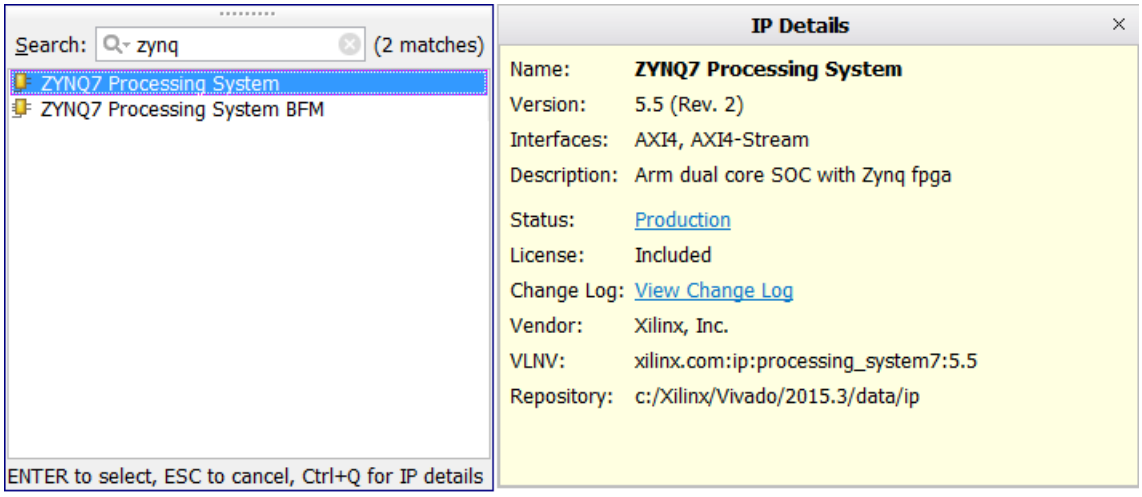

Figure 5: The IP Integrator IP Catalog

6. In the IP Catalog, select the ZYNQ7 Processing System, 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_system7:5.5
   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.

Figure 6: Run Block Automation link

   The Run Block Automation dialog box opens, stating that the `FIXED_IO`, and `DDR` interfaces will be created for the Zynq-7000 AP SoC 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.

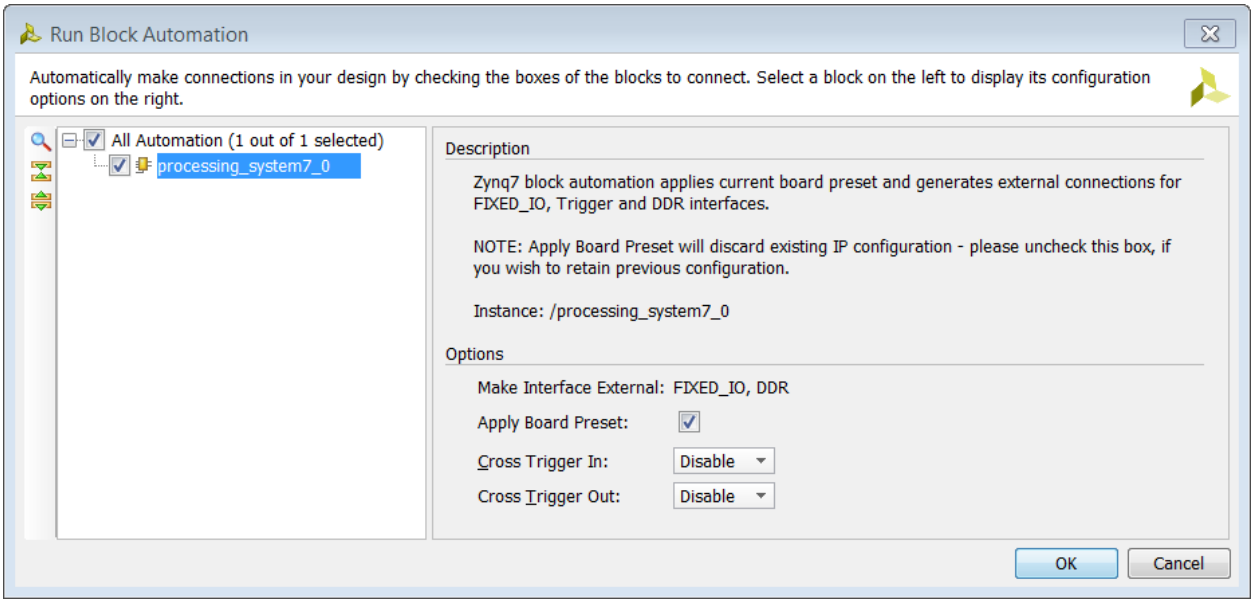

Figure 7: Zynq-7 Run Block Automation Dialog Box

9. Click **OK**.

   After running block automation on the Zynq-7000 AP SoC processor, the IP integrator diagram should look as follows:

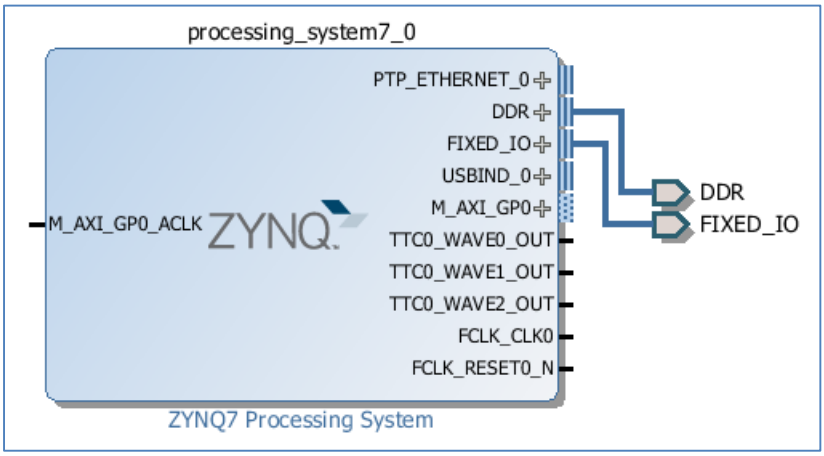

Figure 8: Zynq-7000 AP SoC Processing System after Running Block Automation

   Now you can add peripherals to the processing logic (PL). To do this, right-click in the IP integrator diagram, and select Add IP.

10. In the search field, type `gpi` to find the **AXI GPIO IP**, and then press **Enter** to add it to the design.
11. Similarly, add the **AXI BRAM Controller**.

Your Block Design window should look 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.*

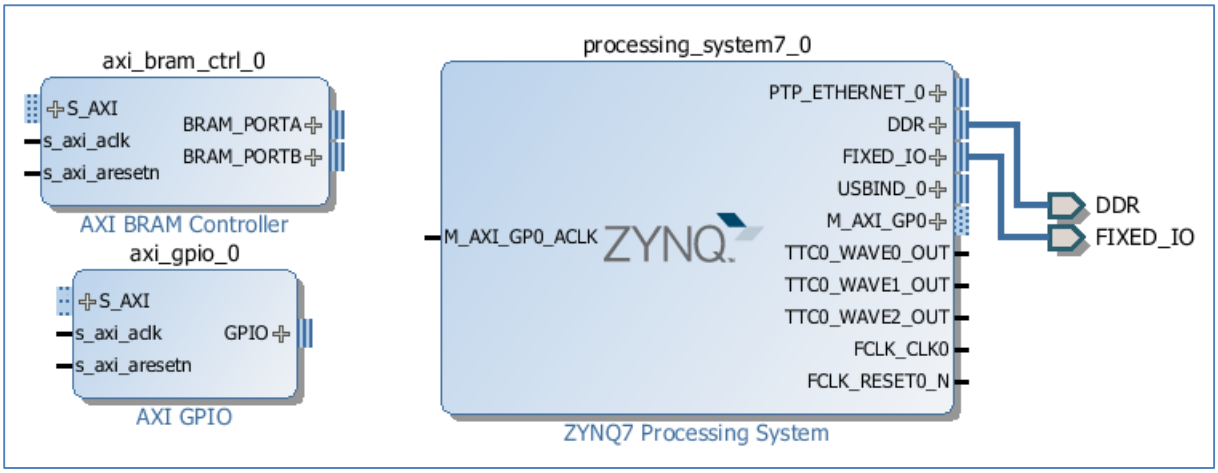

Figure 9: Block Design after Instantiating IP

Use Designer Assistance

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

1. Click Run Connection Automation as shown in Figure 10.

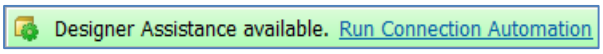

Figure 10: Run Connection Automation

2. The Run Connection Automation dialog box opens.

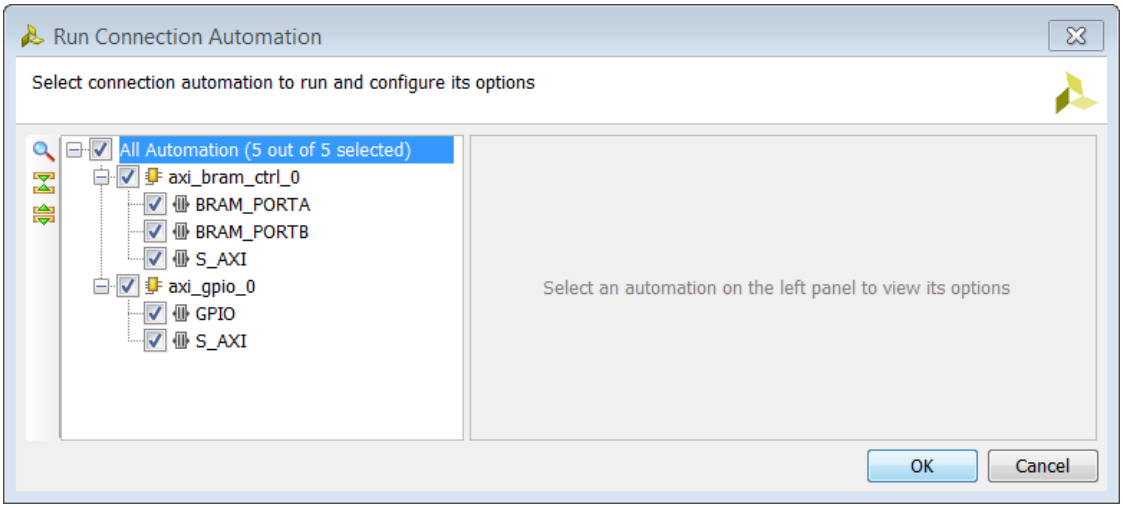

Figure 11: Run Connection Automation Message

3. Select the **All Automation** (5 out of 5 selected) check box.

   As you select each interface for which connection automation is to be run, the description and options available for that interface appear in the right pane.

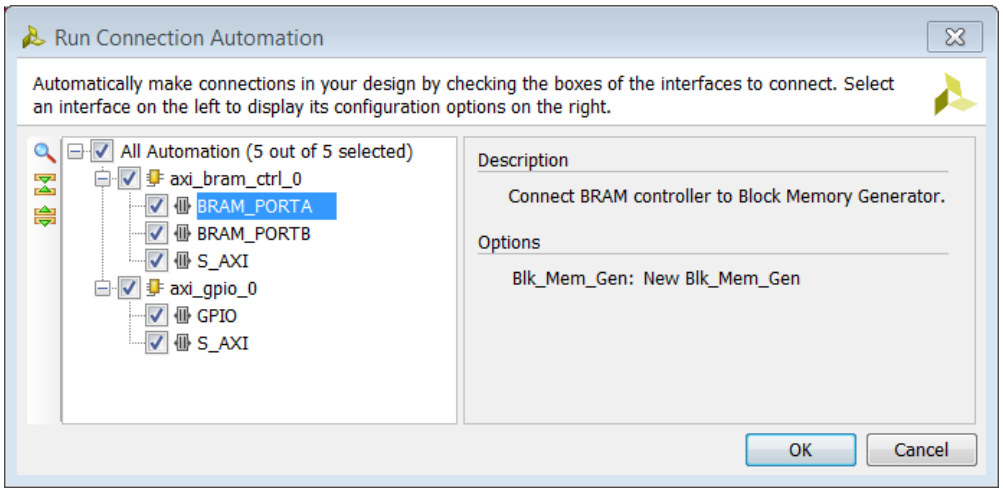

Figure 12: Connection Automation Options

   The `S_AXI` Interface of the `axi_bram_ctrl_0` has a **Clock Connection (for unconnected clks) field**.

4. Ensure that this is set to its default value of **Auto**.

   This value selects the default clock, `FCLK_CLK0`, generated by the PS7 for this interface.

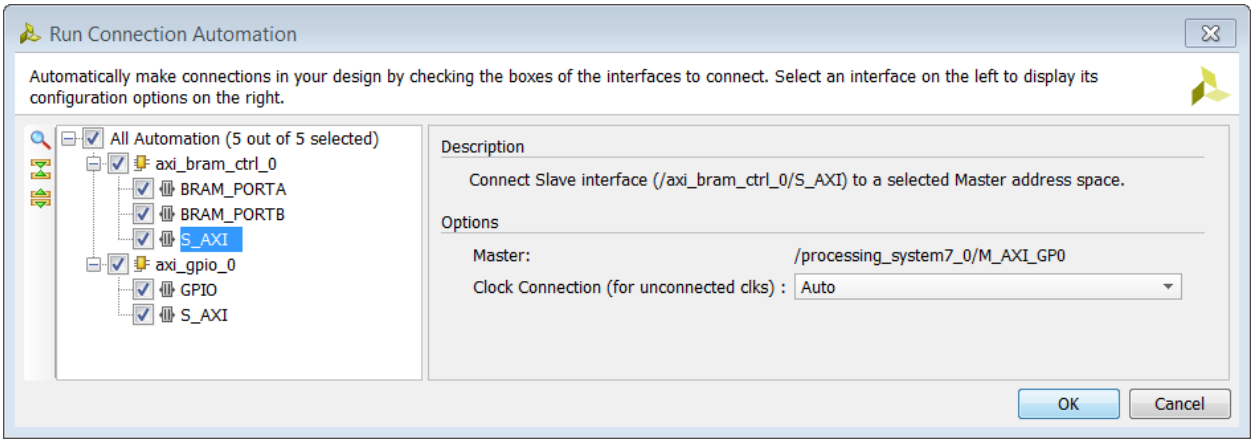

Figure 13: S_AXI Connection Automation Options for axi_bram_ctrl_0

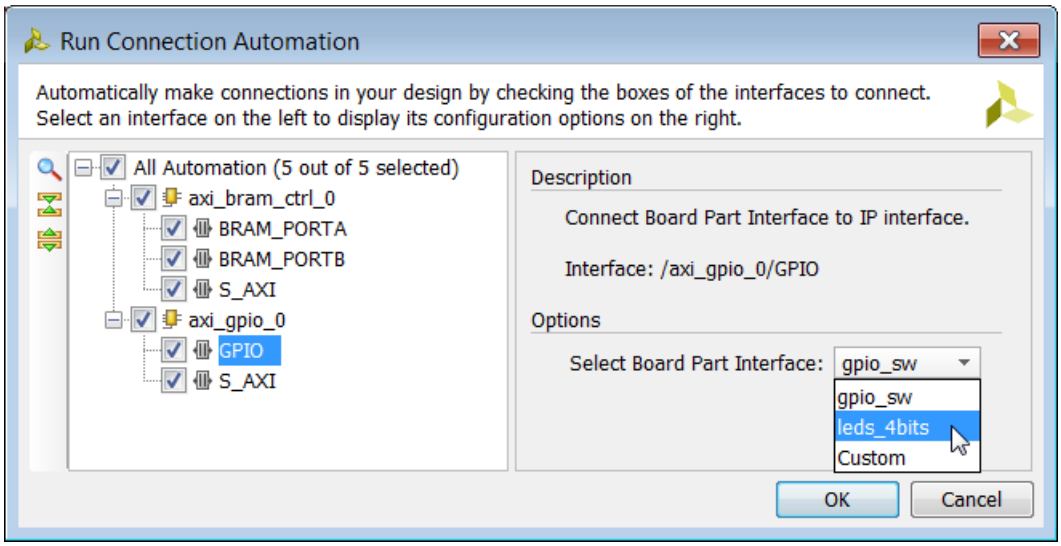

Figure 14: Run Connection Automation Dialog Box for GPIO Interface of axi_gpio_0

5. For the `S_AXI` interface of `axi_gpio_0` instance, leave the **Clock Connection** (for unconnected clks) field to **Auto**.

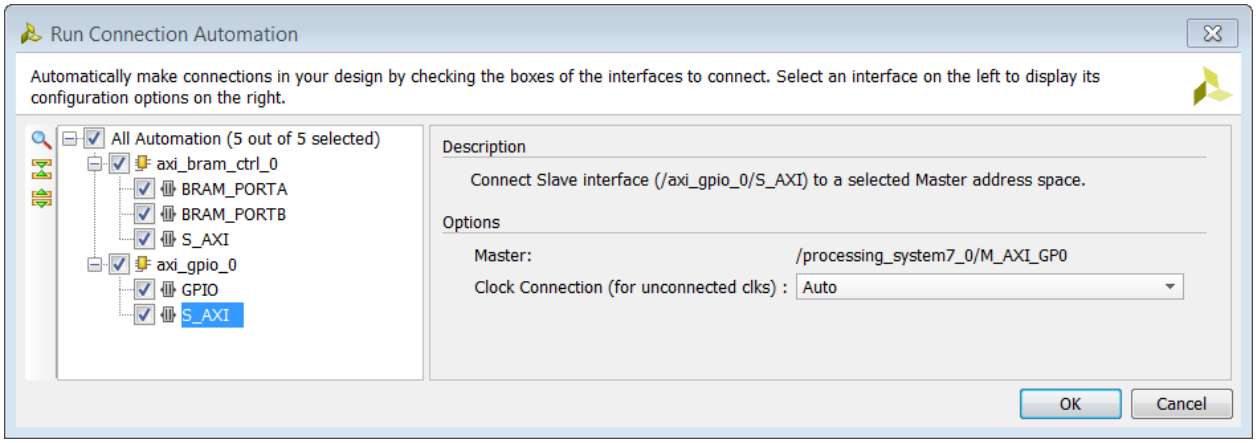

Figure 15: S_AXI Interface Options for axi_gpio_0

6. Click **OK**.

The IP integrator subsystem looks like Figure 16. The relative positions of the IP might differ slightly.

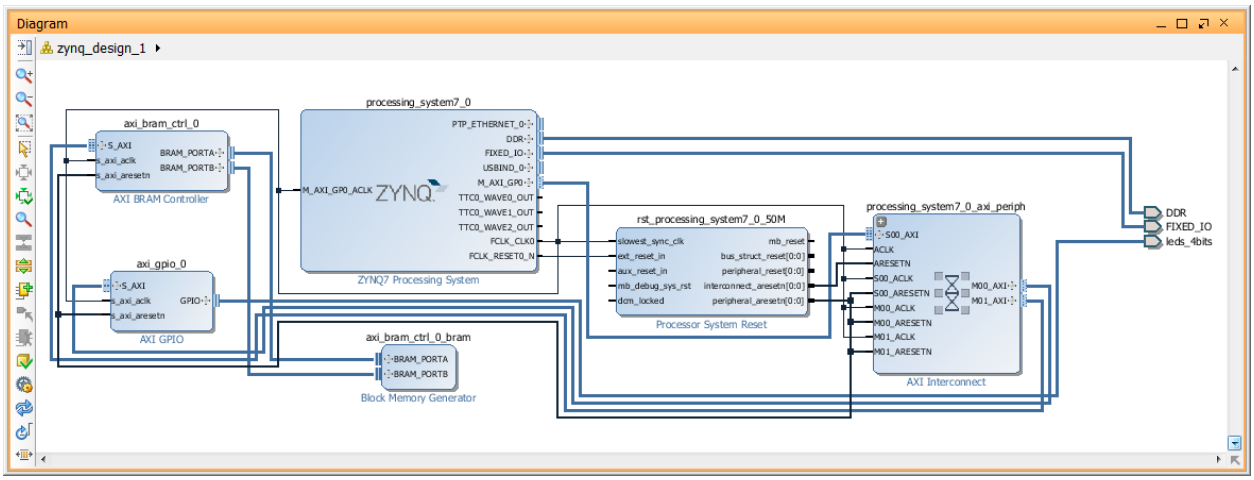

Figure 16: Zynq-7000 AP SoC Processor System

7. 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.

8. Change the range of the AXI BRAM Controller to **64K**, as shown below.

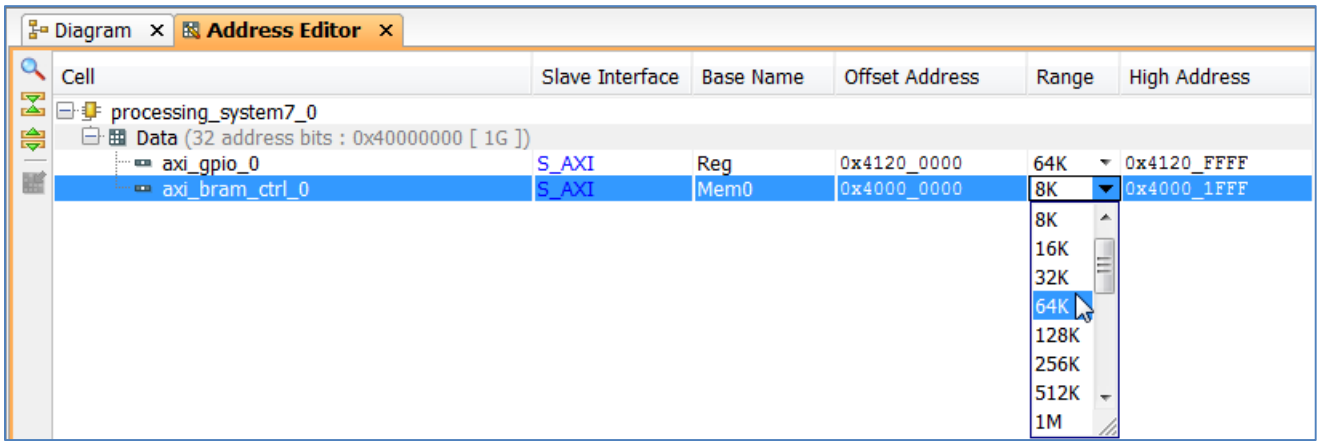

Figure 17: Change Range of axi_bram_ctrl to 64K

9. Click the Diagram tab to go back to the block design.

10. Click the **Regenerate Layout** button  to regenerate an optimal layout of the block design.

Step 3: Using MARK_DEBUG

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

1. 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, as displayed in the figure below, on the interface that you chose to mark for debug.

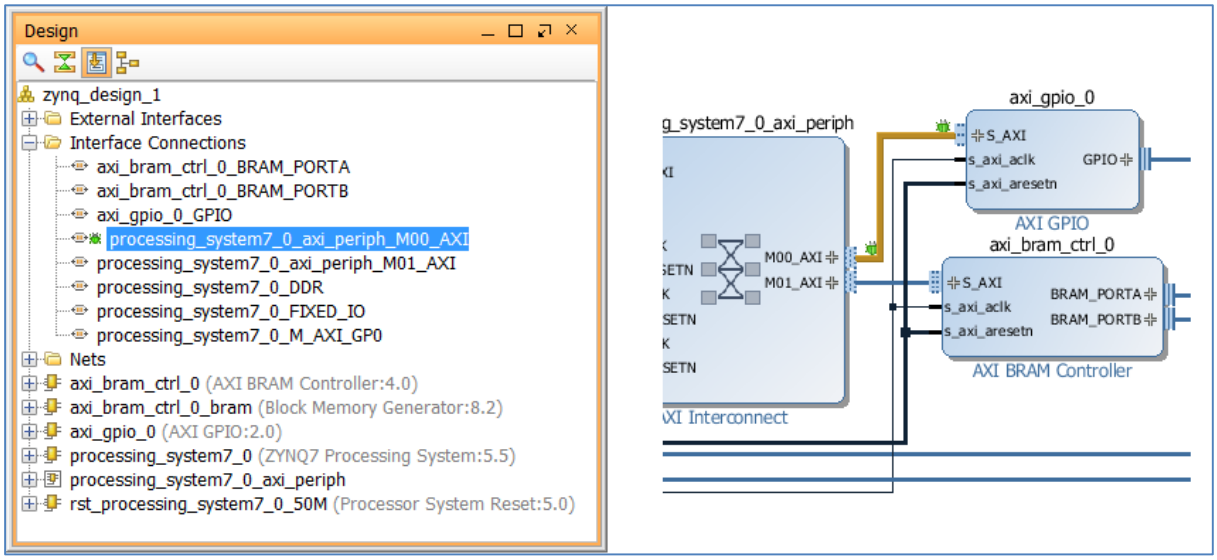

Figure 18: Design Hierarchy: Icon for Mark Debug

2. 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.

   The Tcl console shows the following warning.

```
WARNING: [BD 41-1634] Updates have been made to one or more nets marked for
debug. It is required to open the synthesized design and use the Set Up Debug
wizard to insert, modify or delete Debug Cores. Failure to do so could result in
critical warnings and errors in the implementation flow.
```

This warning message informs you that you must insert a debug core after synthesizing the design. Likewise, after the debug activities have been done and the nets have been "unmarked" for debug, then you must delete any existing debug cores from the synthesized netlist.

3. Click **OK**.
4. From the Vivado menu, save the block design by selecting **File > Save Block Design**.

   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.

   The Generate Output Products dialog box opens, as shown in Figure 19.
2. Leave all the settings to their default values. Click **Generate**.

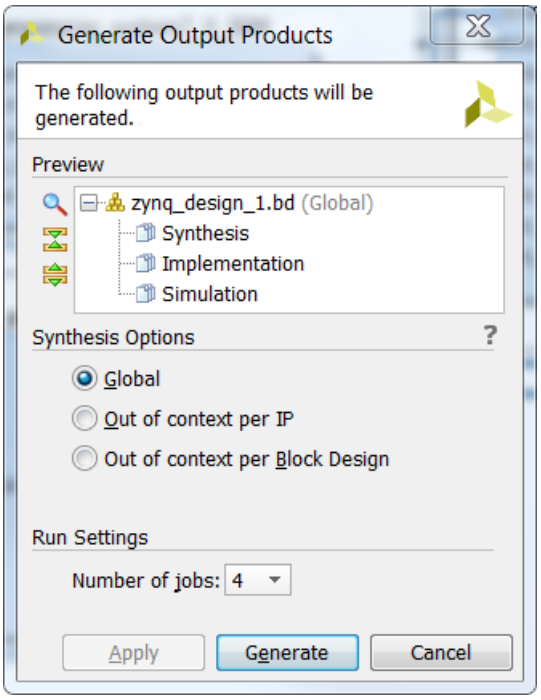

Figure 19: Generate Output Products Dialog Box

3. The Generate Output Products dialog box opens after the output products are generated.
4. Click **OK**.
5. In the Sources window, right-click the top-level subsystem, **zynq_design_1**, 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.

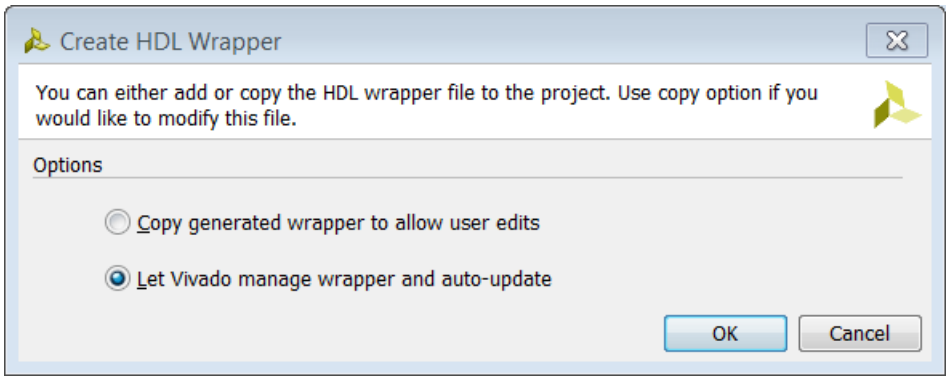

Figure 20: Create HDL Wrapper Dialog Box

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

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

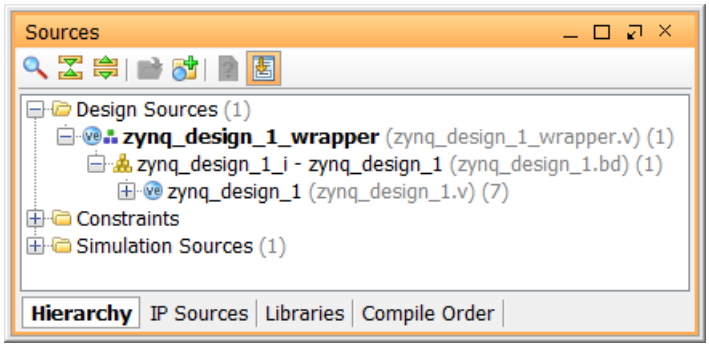

Figure 21: Source Window After Creating the Wrapper

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 opens.

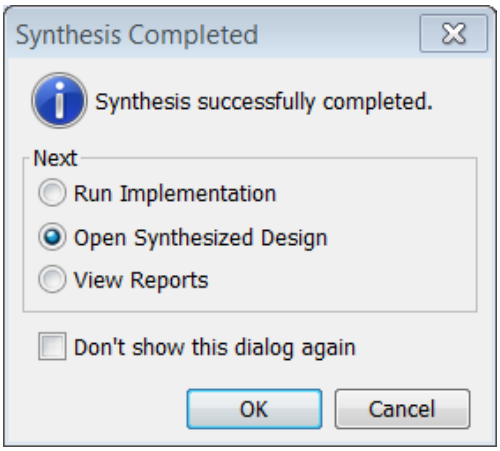

Figure 22: Synthesis Completed Dialog Box

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

   Vivado opens the synthesized design.

   The Debug window opens as you open the synthesized design. You can open this window manually by selecting ***Window* > *Debug***, or by choosing the ***Debug*** window layout.

   The Debug window displays a list of nets in the `Unassigned Debug Nets` folder, as shown in Figure 23. These nets correspond to the various signals that make up the interface connection that you marked for debug in the IP block design.

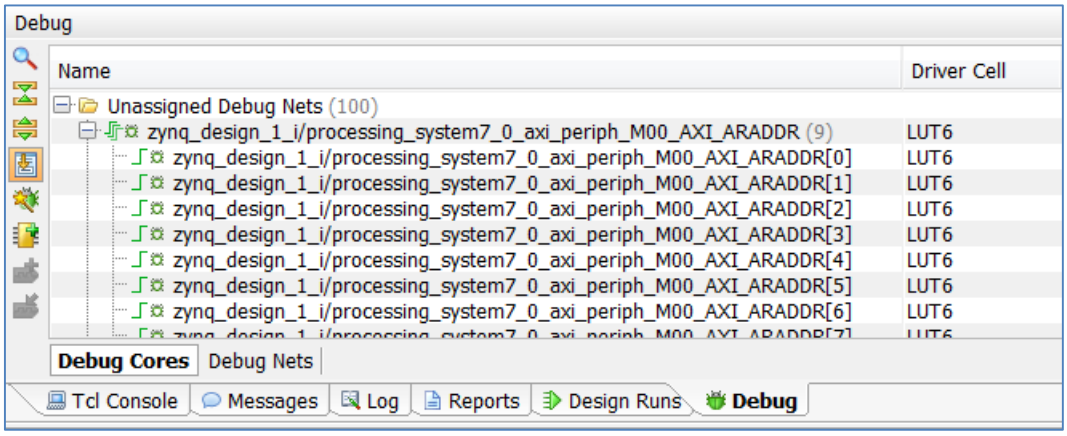

Figure 23: Unassigned Debug Nets Folder

Step 6: Insert an ILA Core and Assign Nets to Debug

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

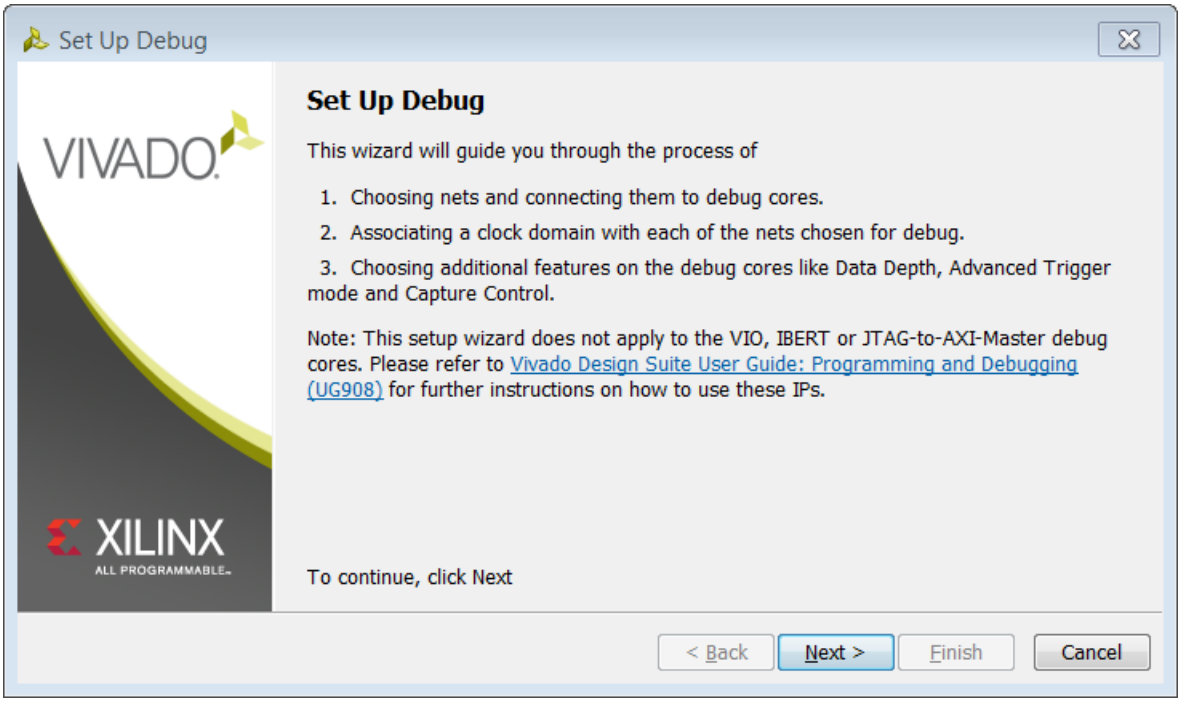

Figure 24: Set Up Debug Wizard

On the second panel of the wizard, notice that some of the nets in the table do not belong to a clock domain. These nets are highlighted below in Figure 25.

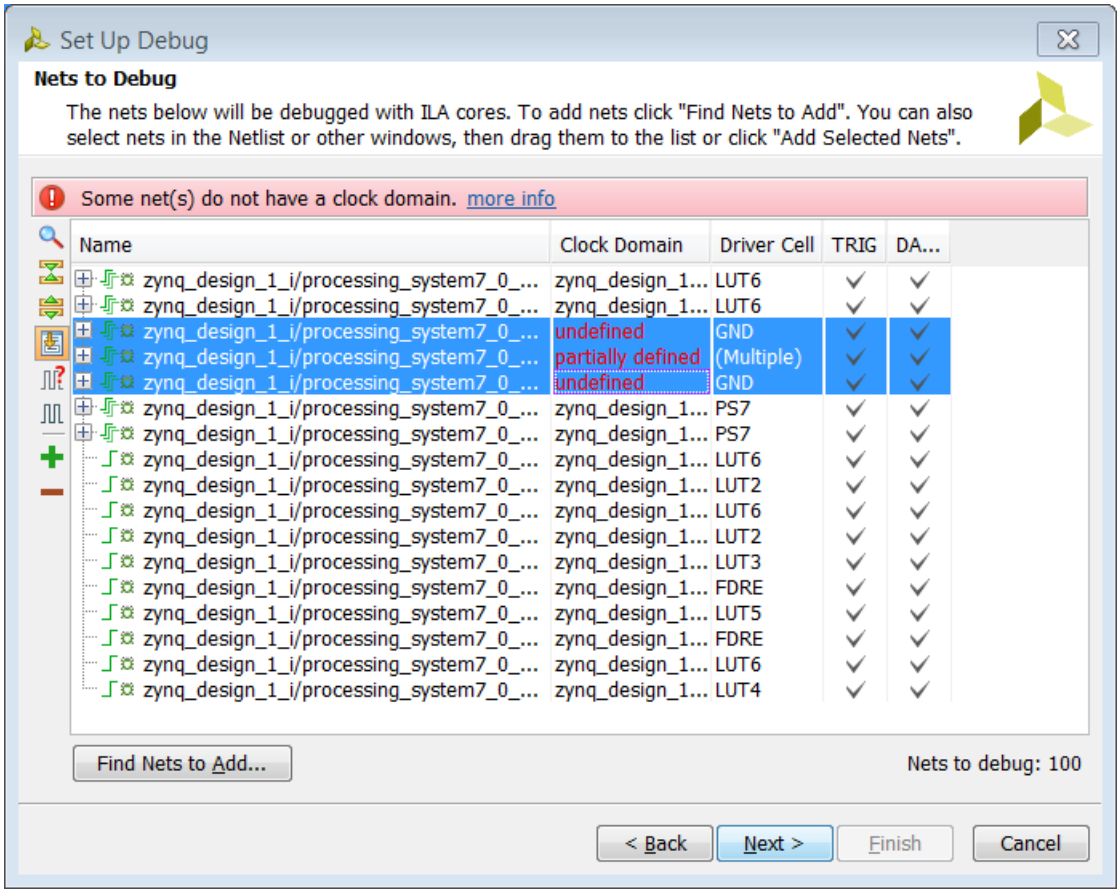

Figure 25: Set Up Debug

IMPORTANT: All signals captured by a single ILA core must have the same clock domain selection.

3. Click the **Filter** button in the left toolbar to show only the nets that do not have a clock domain associated with them.

4. Select all the nets that are shown in the dialog box after filtering, then right-click on your selection and select **Select Clock Domain**.

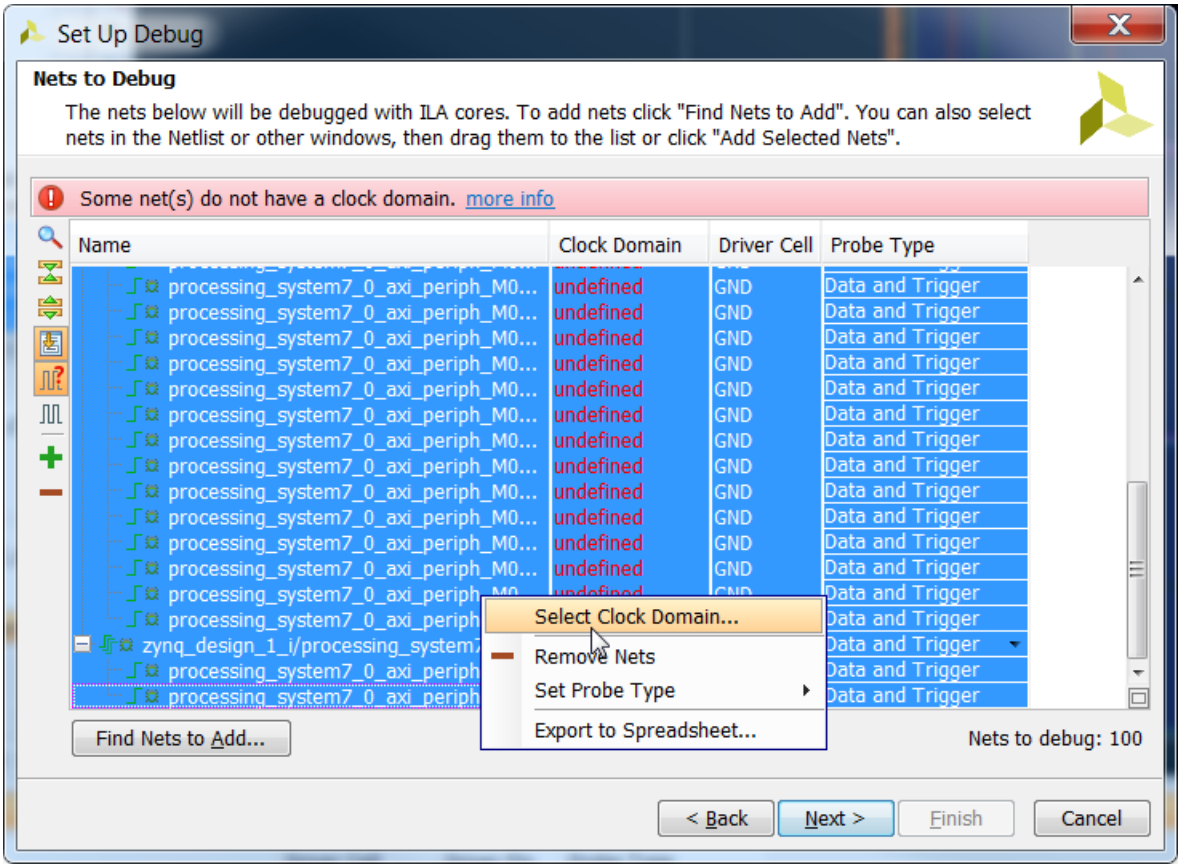

Figure 26: Set Up Debug: Select Clock Domain

5. In the Select Clock Domain window, make sure that the selected filter is GLOBAL_CLOCK, the **Search hierarchically** check box is checked.
6. Select the `zynq_design_1_i/processing_system7_0/inst/FCLK_CLK0` net as the clock domain.

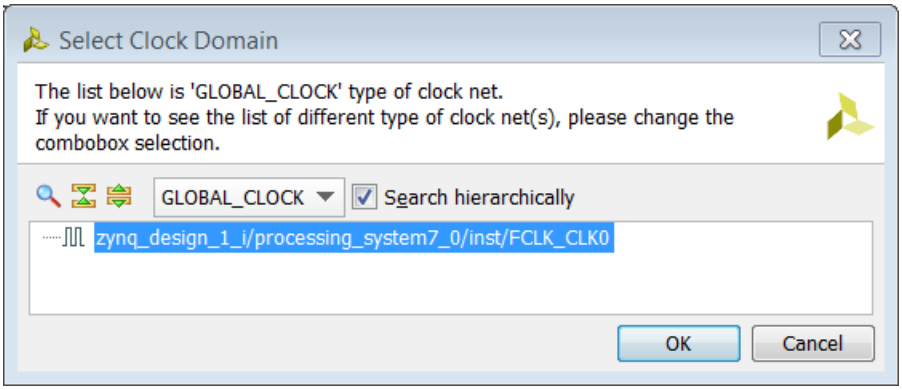

Figure 27: Select Clock Domain Option

7. Click **OK**.

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

8. Click **Next**.

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

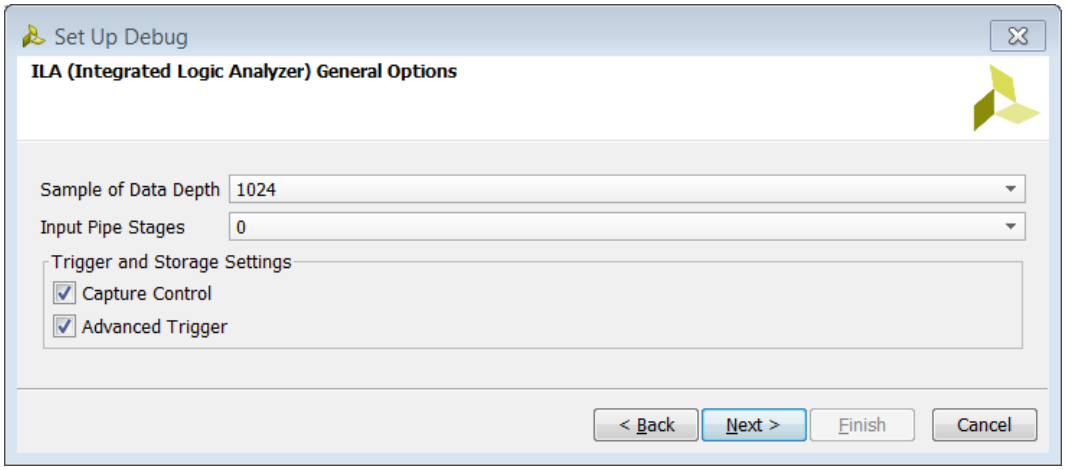

Figure 28: Setting Trigger and Capture Modes

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

    The Set up Debug Summary page opens.

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 > 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 Out of Date Design dialog box opens informing you that saving the new debug related constraints could make your synthesis out of date.

3. Click **OK**.

   The No Implementation Results Available dialog box opens.

4. Click **Yes**.

   Notice that you can view the progress of the flow in the upper right-hand corner of the interface, as shown below.

Figure 29: Status Indicator

5. After the bitstream generates, the Bitstream Generation Completed dialog box opens. In it, the **Open Implemented Design** option is selected by default.

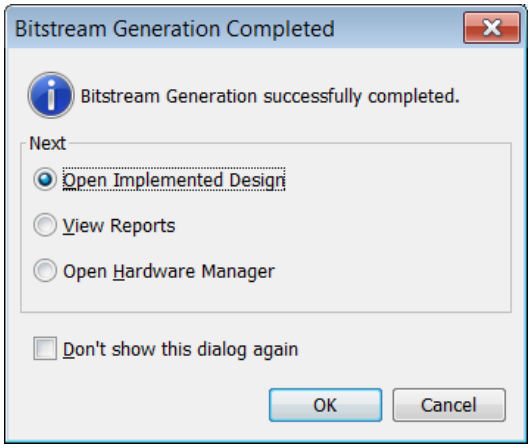

Figure 30: Bitstream Generation Completed

6. Click **OK**.

   The Vivado dialog box prompts you to close the synthesized design before opening the implemented design.

   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, as displayed in Figure 31, to see the inserted ILA and Debug Hub (`dbg_hub`) cores in the design ().

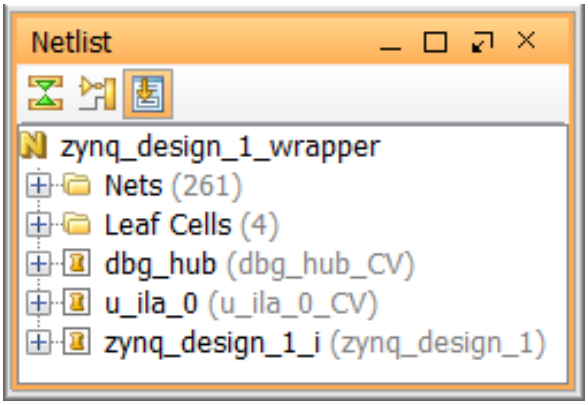

Figure 31: Implemented Design

8. When the implemented design opens, look at the timing window to ensure that all timing constraints were met.

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: SDK and Logic Analyzer.

***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.

2. 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> as shown in Figure 32.

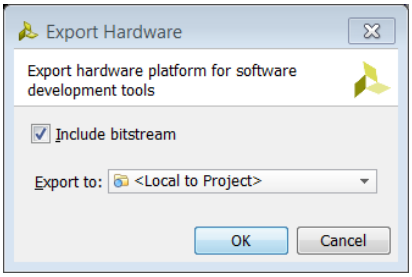

Figure 32: Export Hardware for SDK

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

   The Launch SDK dialog box opens.
5. Accept the default selections for **Exported location** and **Workspace** and click **OK**.

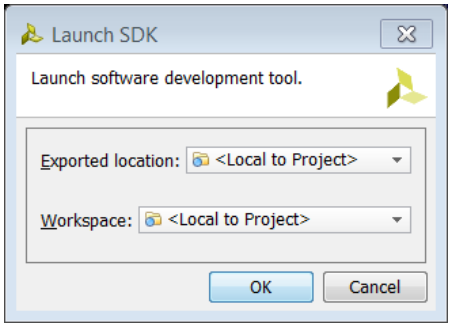

Figure 33: Launch SDK Dialog Box

Conclusion

In this lab, you:

- Created a Vivado project that includes a Zynq-7000 AP SoC processor design using the IP Integrator tool.
- Instantiated IP in the IP Integrator tool and made the necessary connections utilizing the Designer Assistance feature.
- 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 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 2015.2 > Xilinx SDK 2015.2**.

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

   Note: Changing the workspace relaunches the SDK tool to load the appropriate data.

   Now you can create a software application.

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

   The New Project dialog box opens.

3. In the Project Name field, type the name desired, or type `Zynq_Design`.

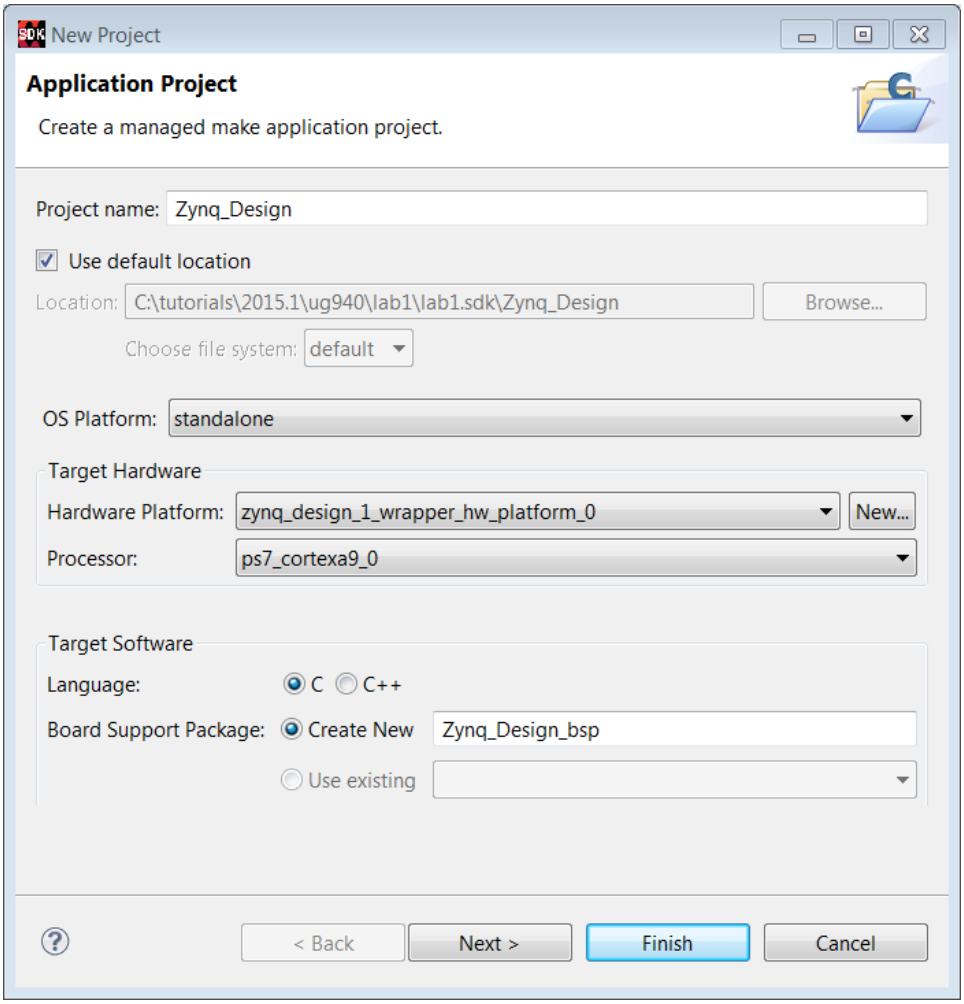

Figure 34: SDK Application Project

4. Click **Next**.

5. From the Available Templates, select Peripheral Tests as shown in Figure 35.

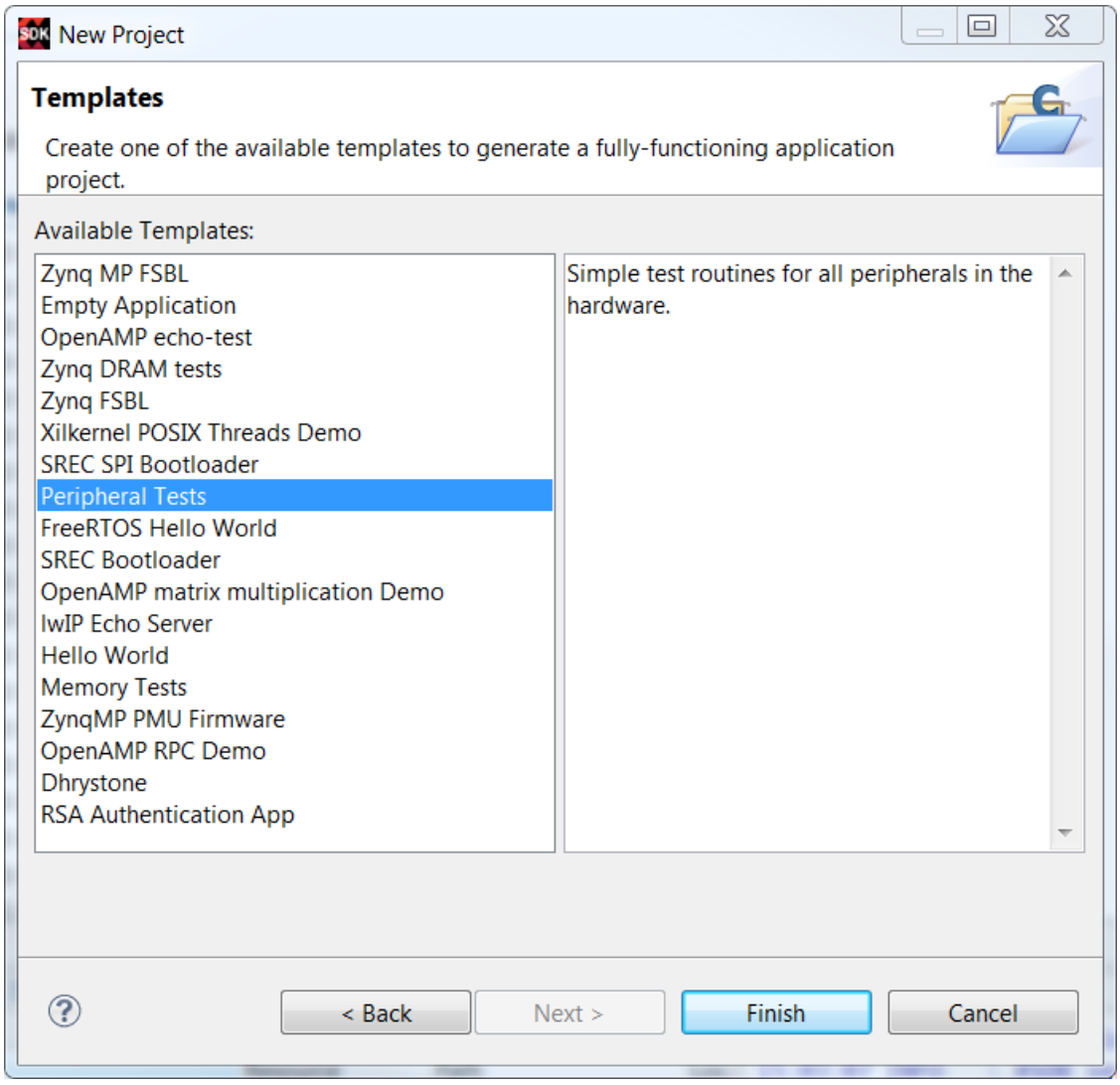

Figure 35: SDK New Project Template

6. Click **Finish.**

   When the program finishes compiling, you see the following in the Console window.

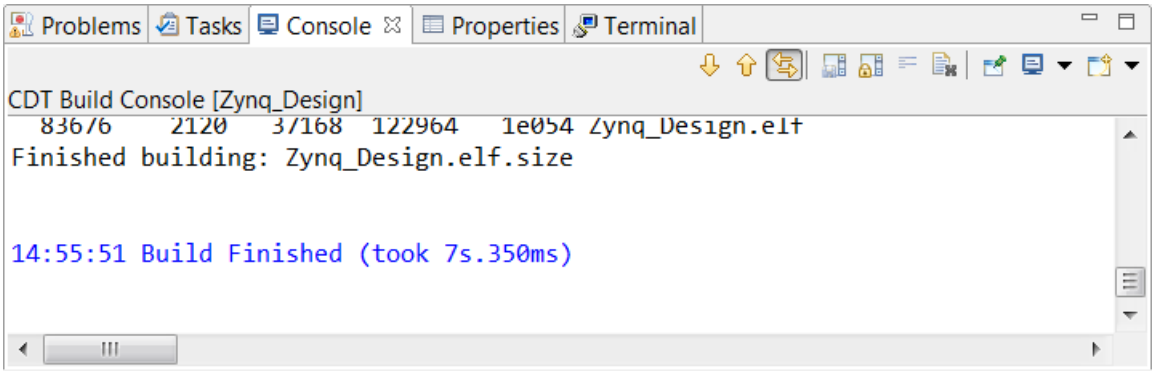

Figure 36: SDK Message

Step 2: Run the Software Application

Now, run the peripheral test application on the ZC702 board. To do so, you need to configure the JTAG port.

1. Ensure that your hardware is powered on and a Digilent Cable or the USB Platform Cable is connected to the host PC. Also, ensure that you have a USB cable connected to the UART port of the ZC702 board.
2. Download the bitstream into the FPGA by selecting **Xilinx Tools > Program FPGA**.

   The Program FPGA dialog box opens.
3. 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.*
4. In the Project Explorer, select and right-click the **Zynq_Design** application.
5. Select **Debug As > Debug Configurations.**
6. In the Debug Configurations dialog box, right-click **Xilinx C/C++ application (GDB)** and select **New.**
7. In the Debug Configurations dialog box, click **Debug**, as shown in the following figure:

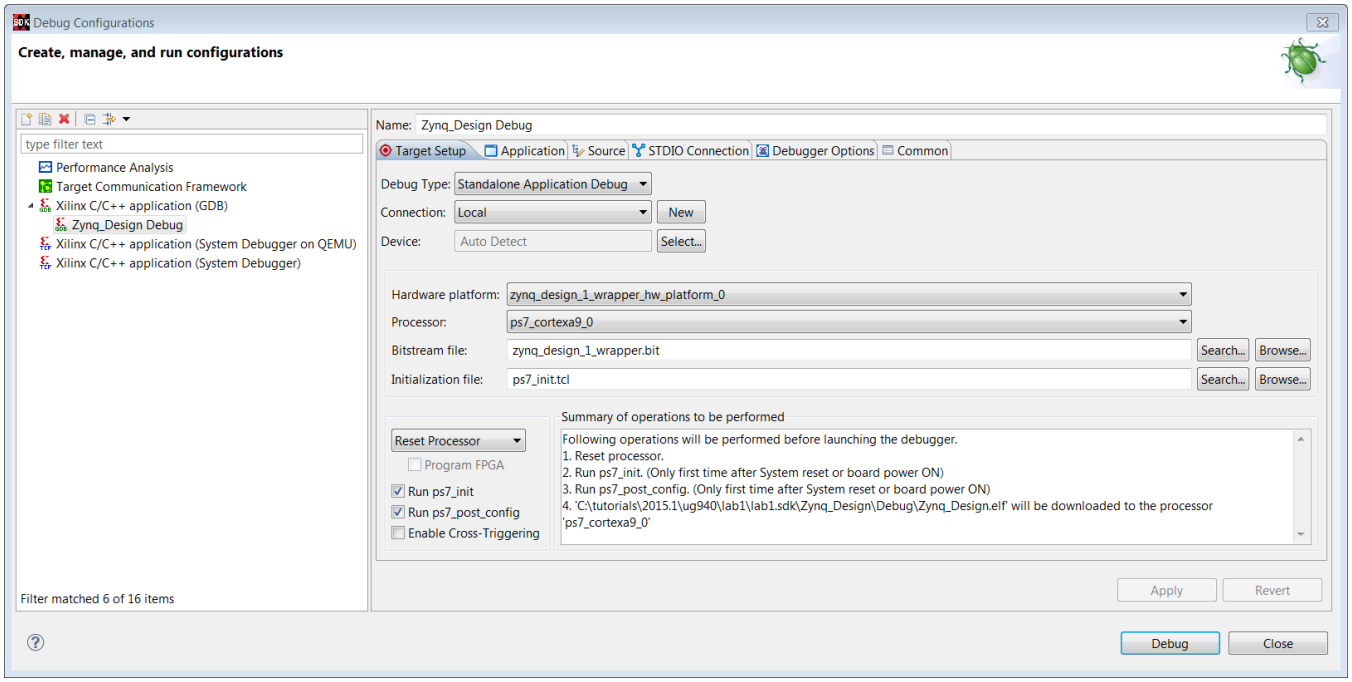

Figure 37: Run Debug Configurations

The Confirm Perspective Switch dialog box opens.

8. Click **Yes**.
9. Set the terminal by selecting the **Terminal** tab and clicking the icon.
10. Use the settings in Figure 38 for the ZC702 board. The COM Port might be different on your machine.

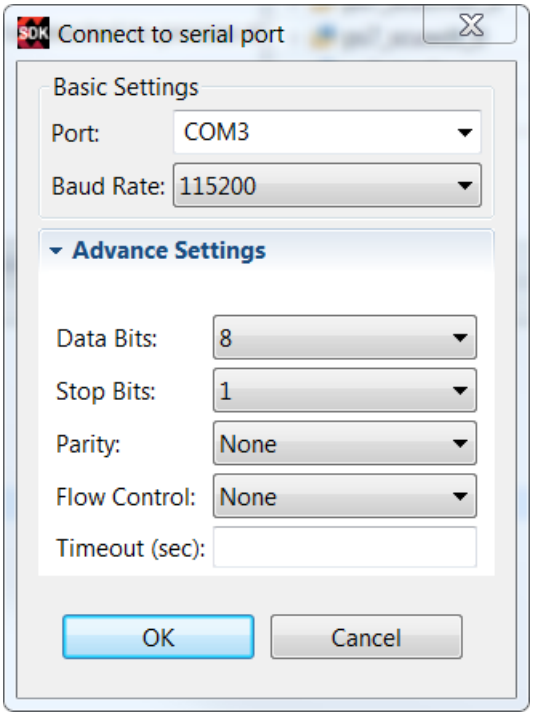

Figure 38: Terminal Settings for ZC702 Board

11. Click **OK**.
12. Verify the **Terminal** connection by checking the status at the top of the tab.

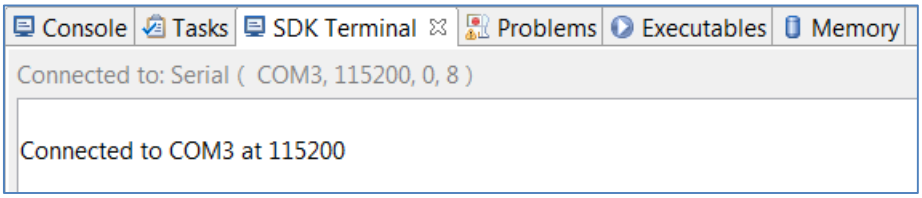

Figure 39: Terminal Connection Verification

13. In the Debug tab, expand the tree to see the processor core on which the program is running, as shown in Figure 40.

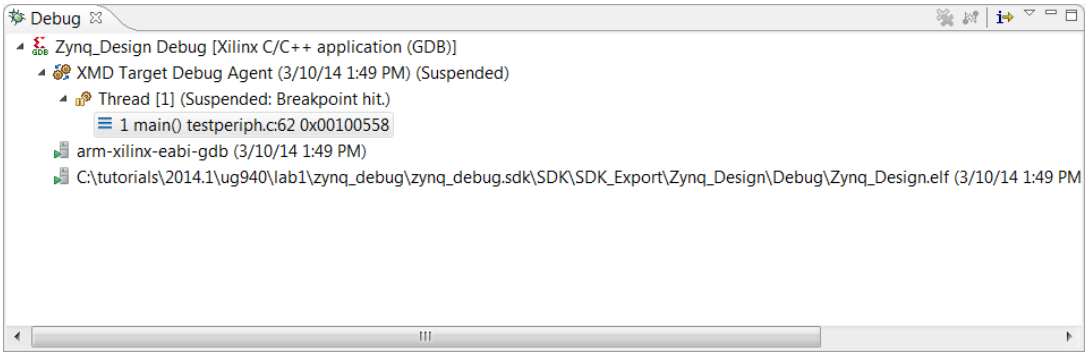

Figure 40: Processor Core to Debug

14. If `testperiph.c` is not already open, select `../src/testperiph.c`, double-click it to open that location

Add a Breakpoint

Next, add a breakpoint on line 106.

1. From the main menu, select **Navigate > Go To Line**.
2. In the Go To Line dialog box, type **106** 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 106 to add a breakpoint on that line of source code, shown in Figure 41.

```
101       print("\r\nRunning GpioOutputExample() for axi_gpio_0...\r\n");
102
103       status = GpioOutputExample(XPAR_AXI_GPIO_0_DEVICE_ID,4);
104
105       if (status == 0) {
106          print("GpioOutputExample PASSED.\r\n");
107       }
108       else {
109          print("GpioOutputExample FAILED.\r\n");
110       }
```

Figure 41: Add a Breakpoint

Step 3: Connect to the Vivado Logic Analyzer

1. Connect to the ZC702 board using the Vivado® logic analyzer.
2. Open the Vivado project from Lab 1: Programming a Zynq-7000 AP SoC Processor if it is not already open.
3. From the Program and Debug drop-down list in the **Flow Navigator > Program and Debug**, select **Open Hardware Manager**.
4. In the Hardware Manager window, click **Open target** and select **Open New Target** to open a connection to the Digilent JTAG cable for ZC702, as shown below in Figure 42.

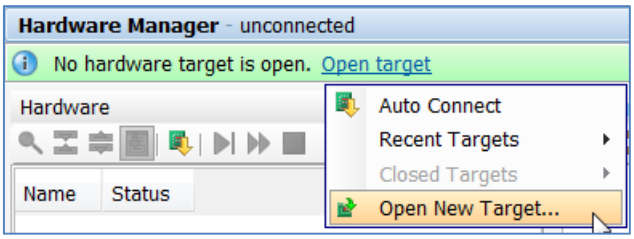

Figure 42: Launch Open New Hardware Target Wizard

The Open New Hardware Target dialog box opens.

5. Click **Next**.
6. Click **Next** on the Hardware Server Settings page.
7. Click **Next** on the Select Hardware Target page.
8. Click **Finish**.

When the Vivado hardware session successfully connects to the ZC702 board, the Hardware window shows the following information:

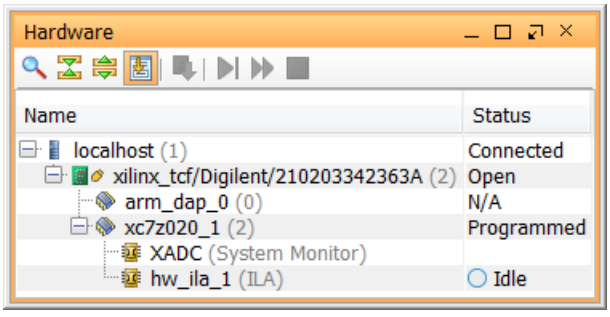

Figure 43: Successfully Programmed Hardware Session

9. First, ensure that the ILA core is active and capturing data. To do this, select the Status tab of the `hw_ila_1` in the Hardware Manager.

10. Click the **Run Trigger Immediate** button  .

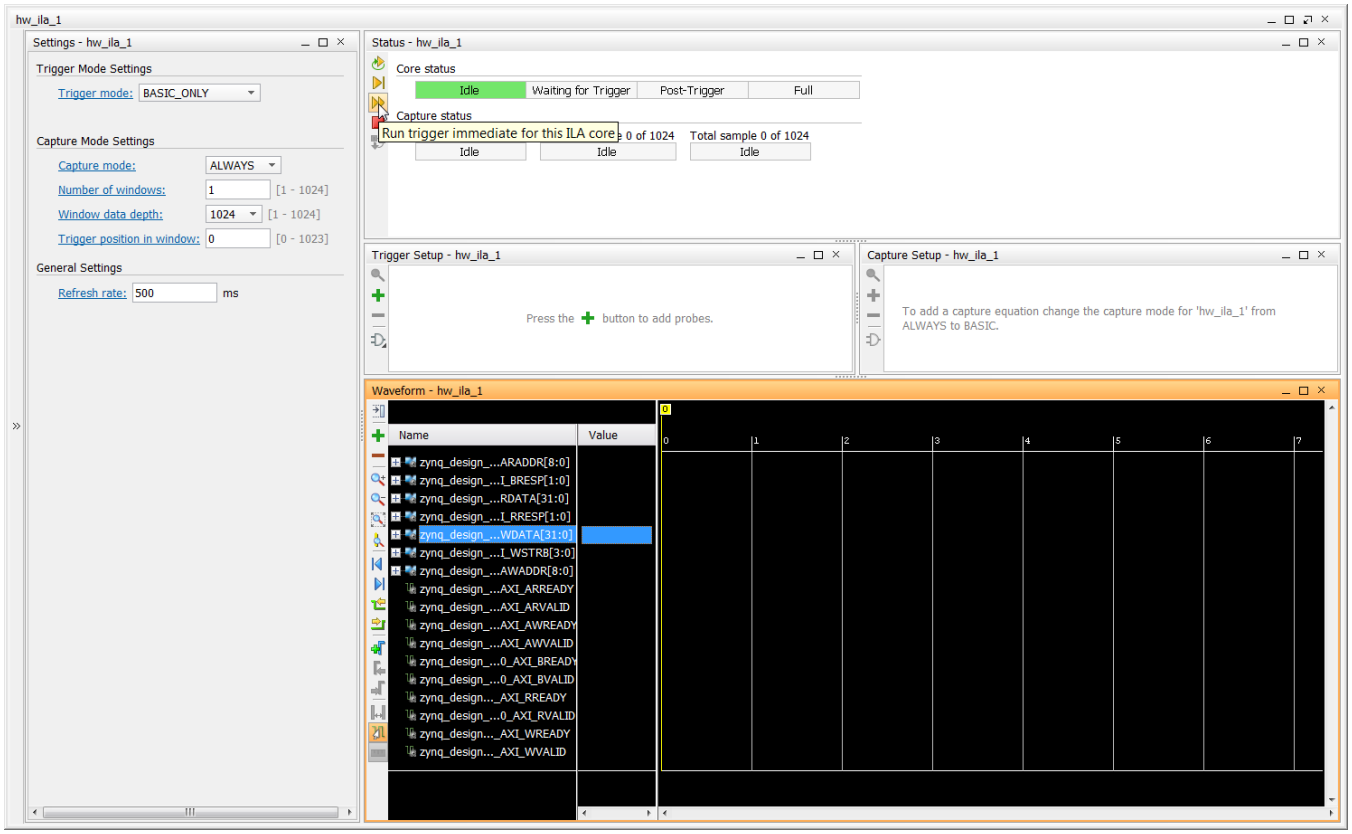

Figure 44: Run Trigger Immediate to capture static data

Static data from the ILA core displays in the waveform window as shown in Figure 45.

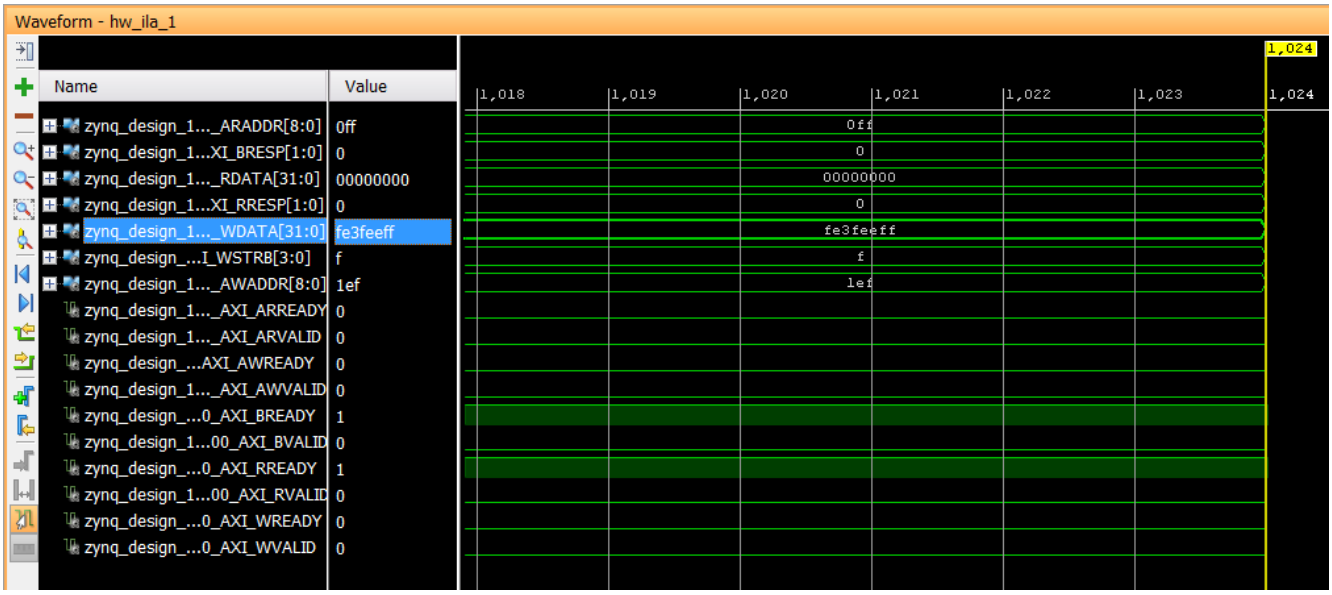

Figure 45: Static Data from the hardware

11. Set up a condition that triggers when the application code writes to the GPIO peripheral. To do this:
    a. From the menu select **Window > Debug Probes**.
    b. 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 Trigger Setup window.

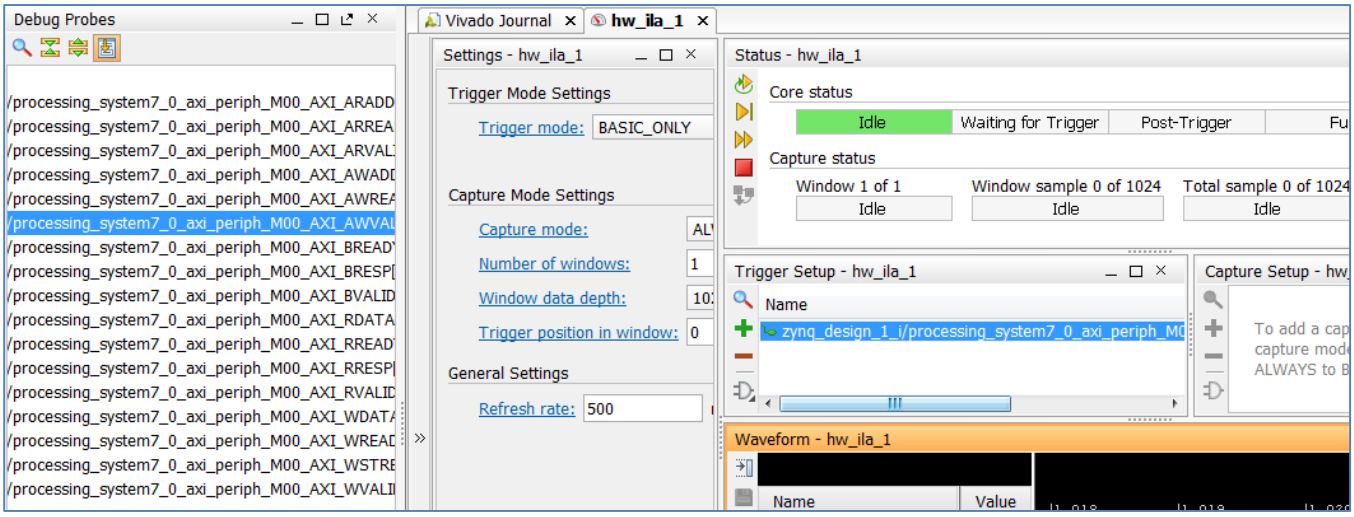

Figure 46: The ILA Properties window

c. Click the **Compare Value** column of the *WVALID row, as shown in Figure 47.

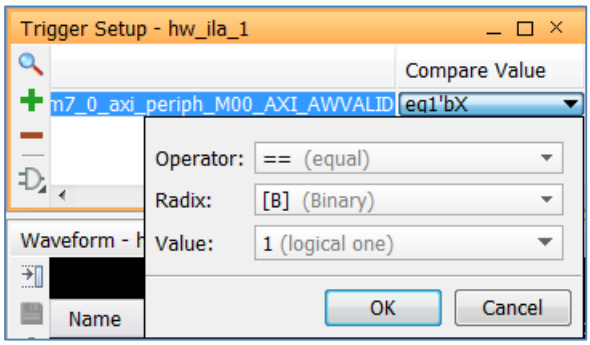

Figure 47: ILA Properties Window

d. Change the **Radix** to [B] binary.

e. Change the value from an X (don't care) to a 1, and click **OK**.

12. You also want to see several samples of the captured data before and after the trigger condition. Change the trigger position to the middle of the 1024 sample window by setting the **Trigger Position in window** for the hw_ial_1 core in the ILA Properties window to 512 and then pressing Enter (Figure 48).

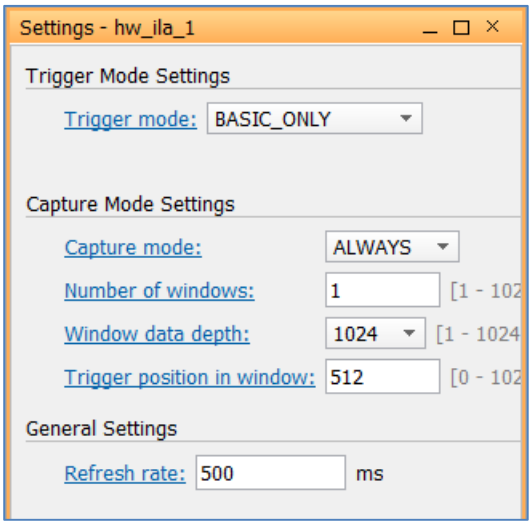

Figure 48: Change Debug Probe Settings

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

13. 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.

14. Notice that the Status window of the `hw_ila_1` ILA core changes from:
    - **Idle** to **Waiting for Trigger**.
    - Likewise, the Hardware window shows the Core Status as **Waiting for Trigger**, as shown in Figure 49.

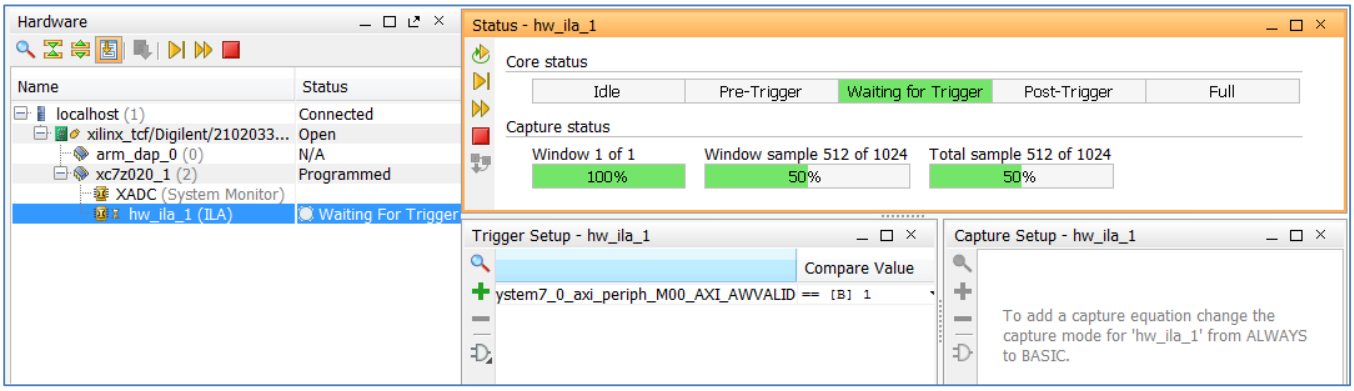

Figure 49: Status of hw_ila_1

15. Go back to SDK and continue to execute code. To do so, 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 106. By this time, at least one write operation has been done to the GPIO peripheral. These write operations cause the WVALID signal to go from 0 to 1, thereby triggering the ILA core.

    ***Note**: The trigger mark occurs at the first occurrence of the AWVALID signal going to a 1, as shown in Figure 50.*

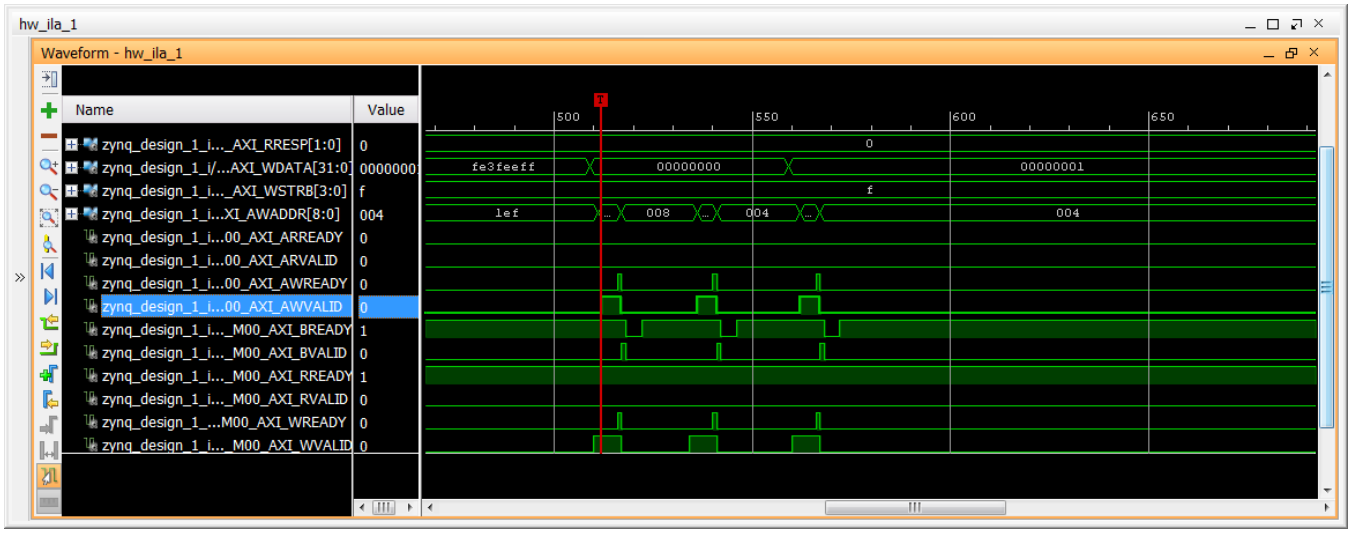

Figure 50: Trigger Mark Goes to 1

- If you are going on to Lab 3, close your project by selecting **File > Close Project.**
- You can also close the SDK window by selecting **File > Exit**.

Conclusion

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

This lab also introduced you to the 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®-7000 AP SoC processor and the fabric logic.

IMPORTANT: *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.
   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. Click **Next**.
   e. In **Add Existing IP** dialog box, click **Next**.
   f. In **Add Constraints** dialog box, click **Next**.
   g. 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**.
   h. 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 `zynq_processor_system` as the name of the block design .
3. Leave the Directory field set to its default value of **<Local to Project>** and the Specify source set field to **Design Sources**.
4. Click **OK**.

   The IP integrator diagram window opens.
5. Click the **Add IP** icon in the block design canvas, as shown in Figure 51.

Figure 51: Add IP to the Design

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

   Alternatively, double-click the **ZYNQ7 Processing System** IP to instantiate it as shown in Figure 52.

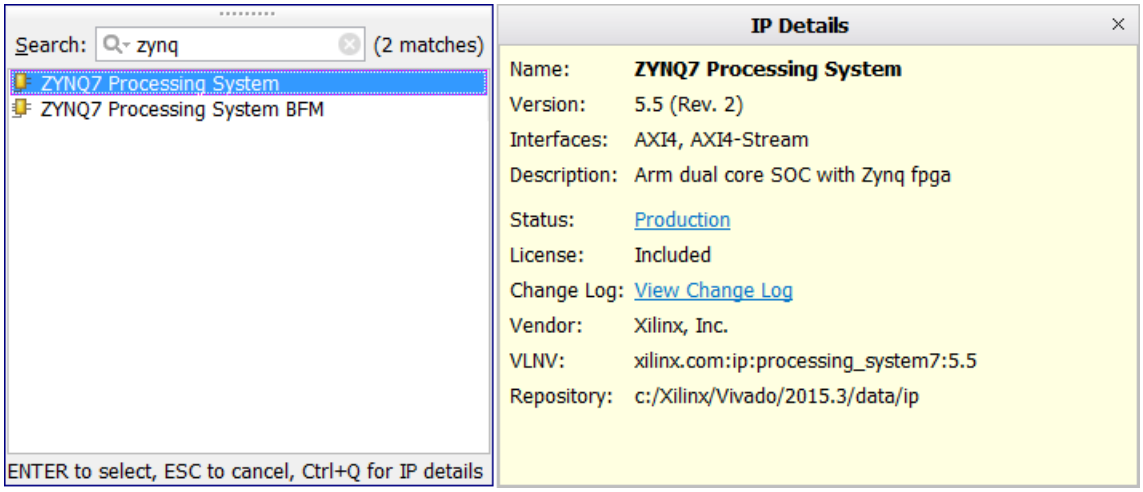

Figure 52: Instantiate the ZYNQ7 Processing System

7. In the header at the top of the diagram, click **Run Block Automation** as shown in Figure 53.

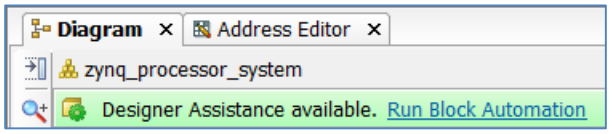

Figure 53: Run Block Automation on Zynq Processing System

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.

8. Enable the Cross Trigger In and Cross Trigger Out functionality by setting those fields to **New ILA**, then click **OK**, as shown in Figure 54.

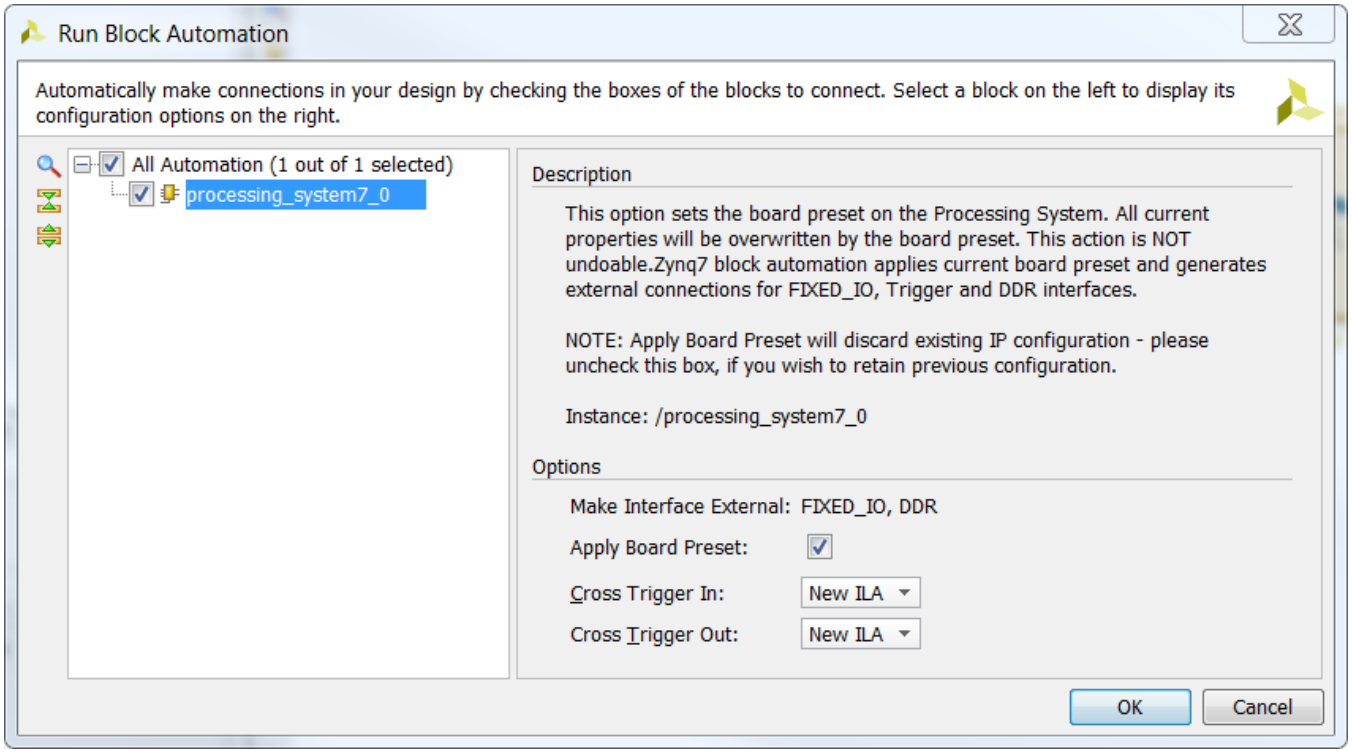

Figure 54: Run Block Automation Dialog Box

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 as show in Figure 55.

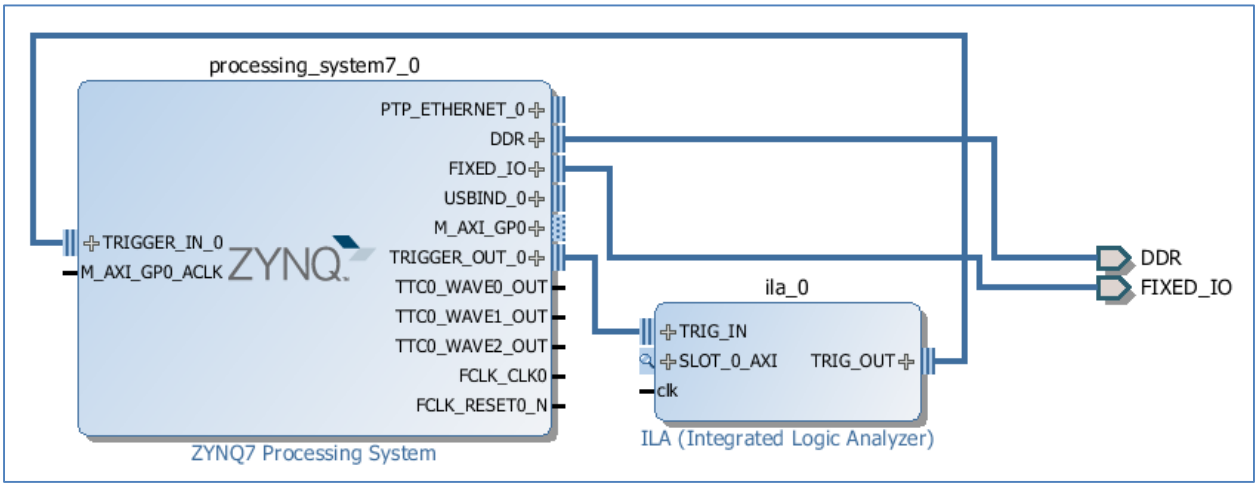

Figure 55: IP Integrator Canvas After Running Block Automation

9. 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 like Figure 56.

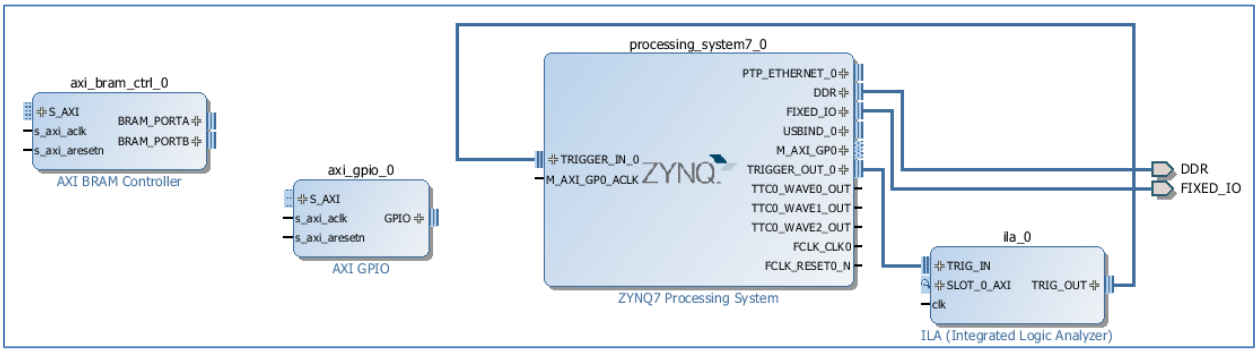

Figure 56: Diagram after Instantiating IP for This Design

10. Click the **Run Connection Automation** link at the top of the Diagram window.

    The Run Connection Automation dialog box opens.

11. Select the **All Automation** (5 out of 5 selected) checkbox. This selects connection automation for all the interfaces in the design. Select each automation to see the available options for that automation in the right pane.

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

Connection	More Information	Setting
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, use the existing BMG.	Leave the Blk_Mem_Gen field set to its default value of **Blk_Mem_Gen of BRAM_PORTA**.
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 (for unconnected clks)** field 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_gpio_0 • S_AXI	The Run Connection Automation dialog box states that the S_AXI pin of the GPIO IP will be connected to the 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.	Leave the **Clock Connection (for unconnected clks)** field set to **Auto**.

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

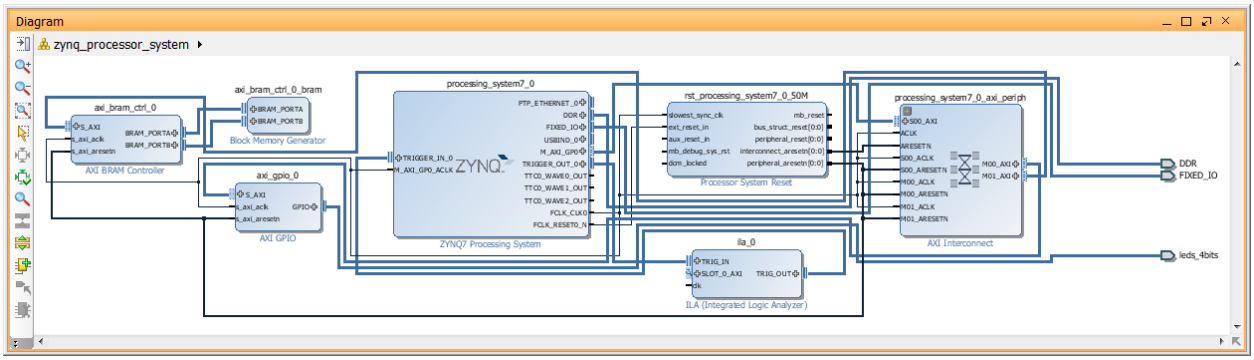

Figure 57: Design after Running Connection Automation

13. 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 as shown in Figure 58.

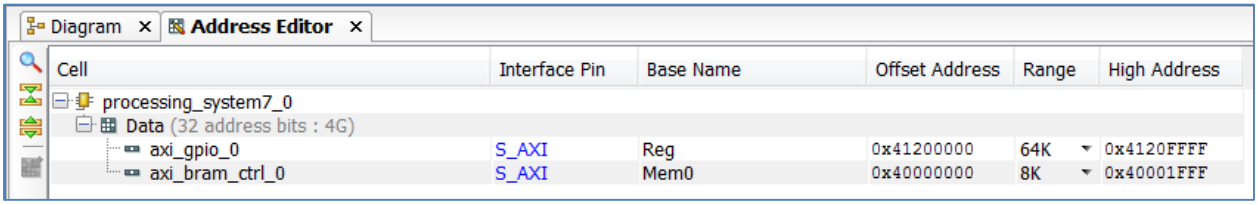

Figure 58: Memory Map the Slaves

Mark Nets for Debug

Next, you will mark some nets for debug.

1. Click the **Diagram** tab again and select the net connecting the gpio pin of the AXI GPIO IP to the LEDs_4Bits port.
2. 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.
3. 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.
4. You will also mark the net connecting the interfaces `TRIG_OUT` and `TRIGGER_IN_0` of the processing_`system7_0`. Likewise, you will mark the net connecting the interfaces `TRIG_IN` of the `ila_0` and `TRIGGER_OUT_0` of processing_system7_0. Marking these cross-trigger nets for debug will enable you to view these signals in the ILA.

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. You 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).

5. The clk pin of the ILA is not yet connected either. Connect it to the FCLK_CLK0 pin of the ZYNQ7 Processing System.
6. Click the **Regenerate Layout** button to generate an optimal layout of the design. The design should look like Figure 59.

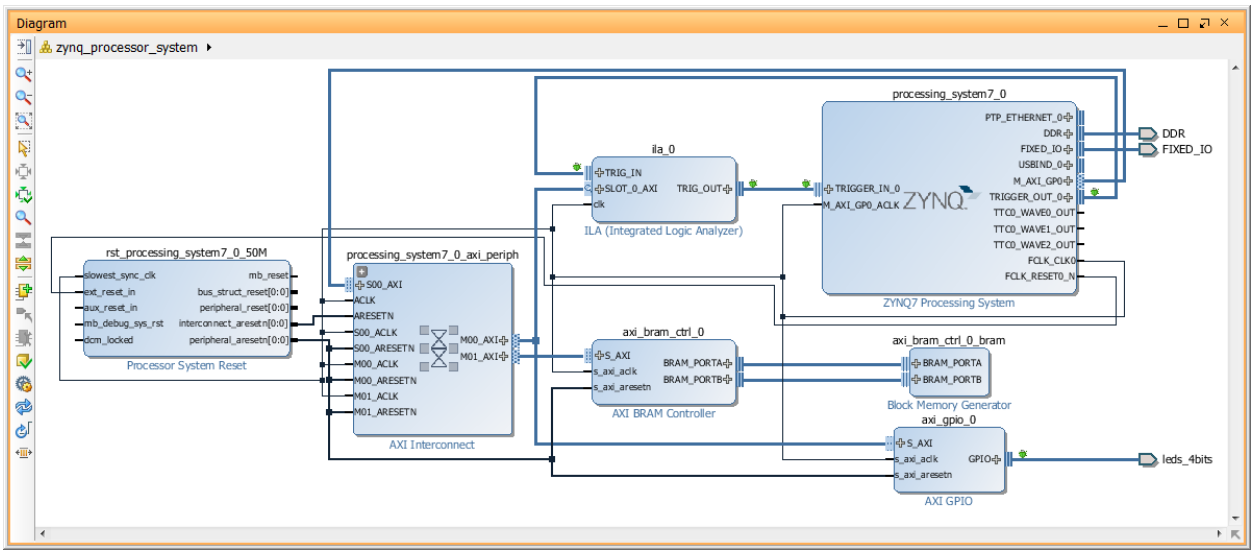

Figure 59: Block Design after Running Regenerate Layout

7. 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.
8. Click **OK**.
9. Select **File > Save Block** Design to save the IP integrator design.

   Alternatively, press **Ctrl + S** to save the design.
10. In the Sources window, right-click the block design, zynq_processor_system, and **select Generate Output Products.**

The Generate Output Products dialog box opens.

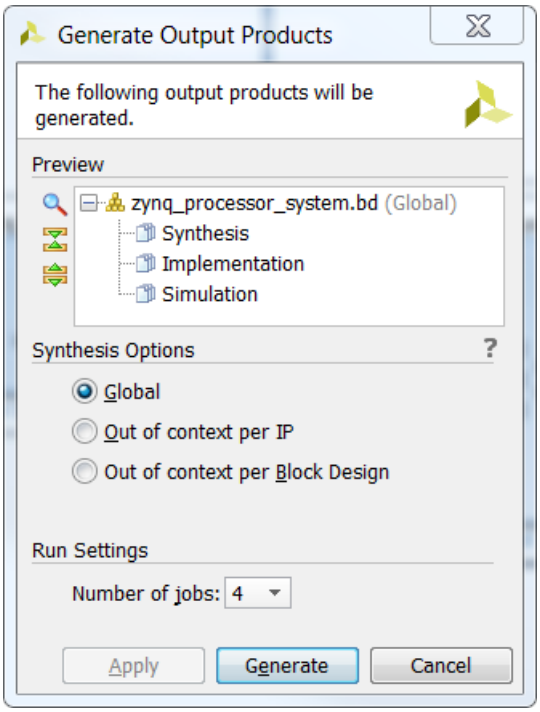

Figure 60: Generate Output Products Dialog Box

11. Click **Generate.**
12. Click **OK** on the Generate Output Products dialog box.
13. 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.

14. Keep the default setting, shown in Figure 61, and click **OK**.

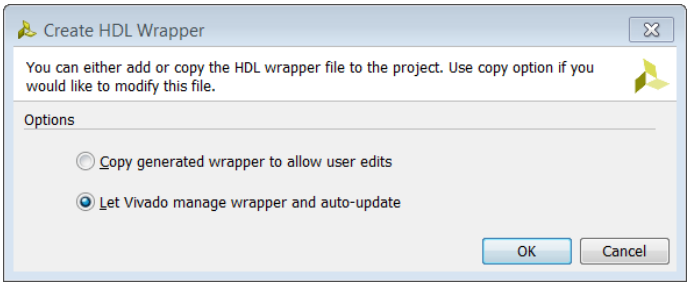

Figure 61: Create HDL Wrapper Dialog Box

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** as shown in Figure 62.

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

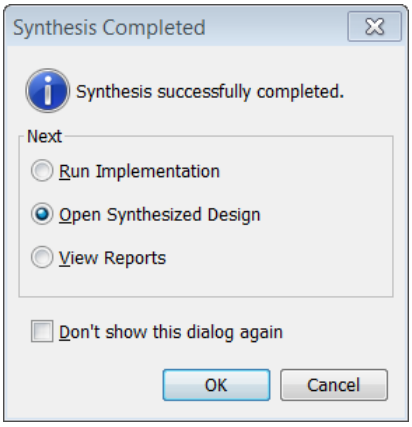

Figure 62: Open Synthesized Design

Step 4: Insert 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 and the cross-trigger pins that were also marked for debug in the block design. These outputs have not yet been assigned to an ILA. You now insert the ILA to monitor these output bits.

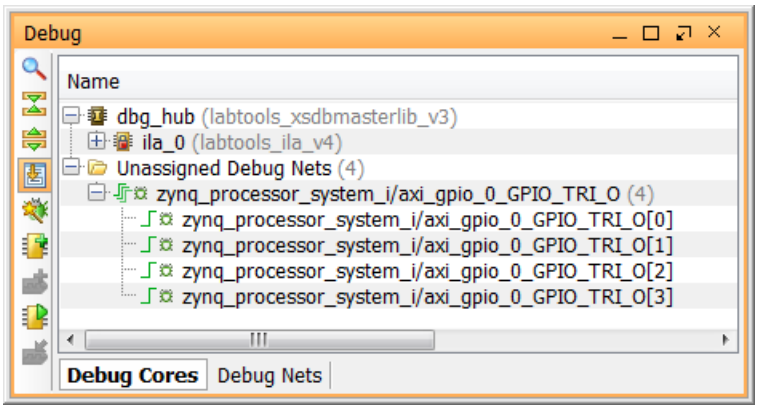

Figure 63: Unassigned Debug Nets

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 as shown in Figure 64.

2. Click **Next**.

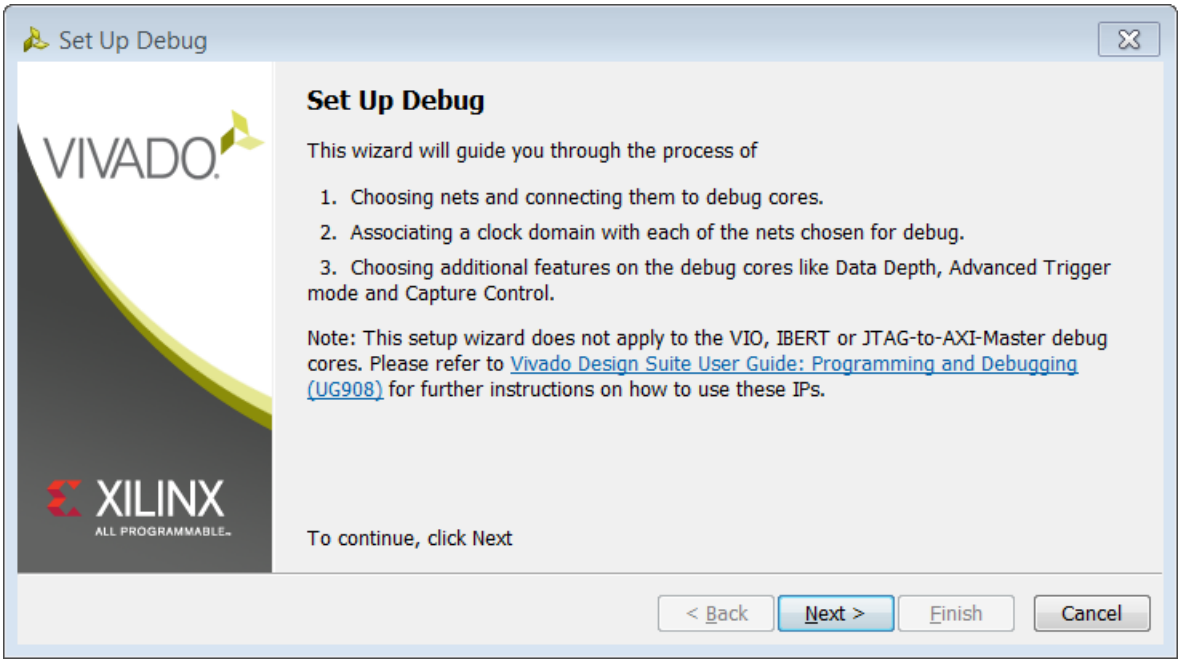

Figure 64: Set Up Debug Dialog Box

3. In the Nets to Debug page, select the nets that you are interested in monitoring from the list of nets offered, as shown in Figure 65.
4. Click the **Find Nets to Add** button at the bottom of the dialog box.

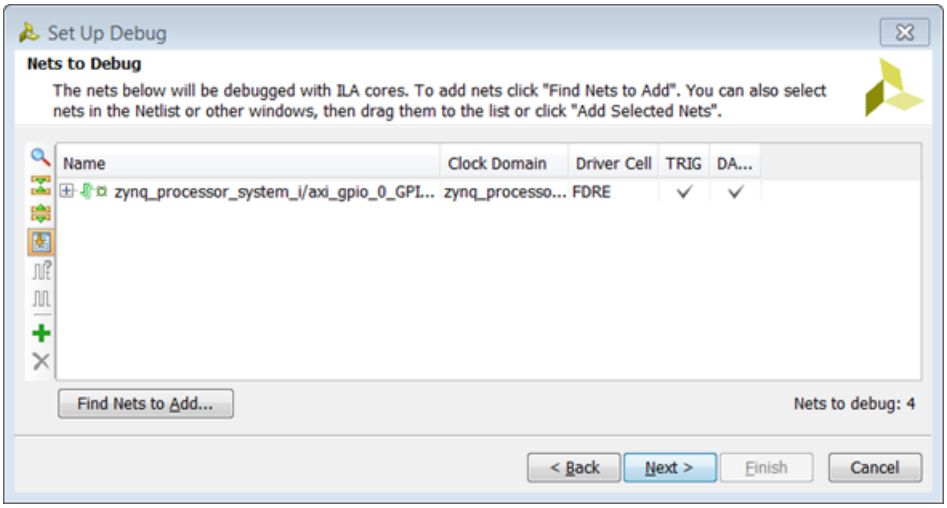

Figure 65: Add Nets to Debug

5. In the find Nets dialog box, shown in Figure 66, change the Properties to select **MARK_DEBUG** from the first drop-down list. This will show all the nets that have the `MARK_DEBUG` attribute set.

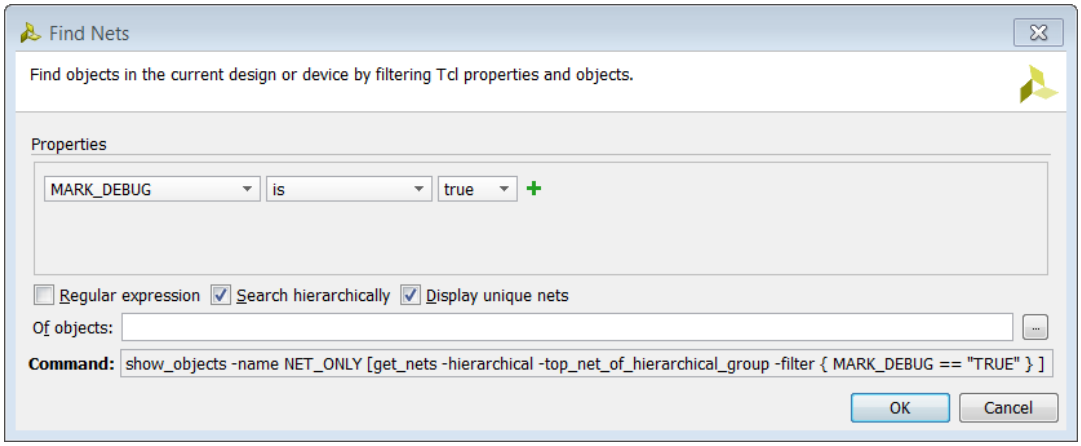

Figure 66: Find Nets with MARK_DEBUG Attribute

6. Click **OK**. The Add Nets to Debug dialog box opens.
7. Select the following nets, as shown in Figure 67:
   - zynq_processor_system_i/ila_0_TRIG_OUT_ACK
   - zynq_processor_system_i/ila_0_TRIG_OUT_TRIG
   - zynq_processor_system_i/processing_system7_0_TRIGGER_OUT_0_ACK
   - zynq_processor_system_i/processing_system7_0_TRIGGOUT_OUT_0_TRIG

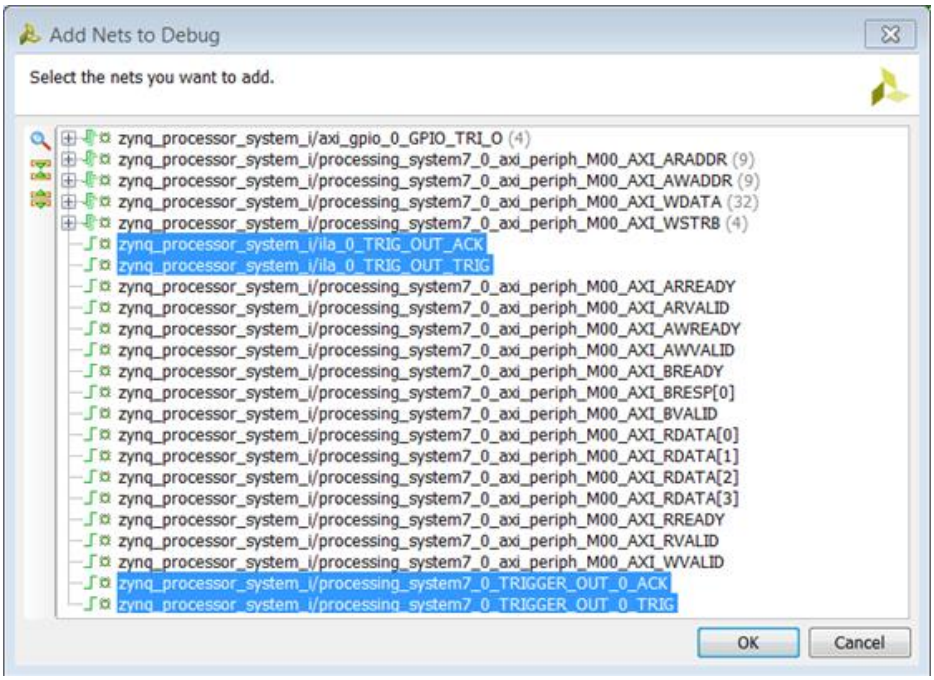

Figure 67: Add Nets Connected to Cross-Trigger Pins

8. Click **OK.**

   The additional cross-trigger nets are added to the ILA in the Set Up Debug dialog box.

9. Click **Next**.

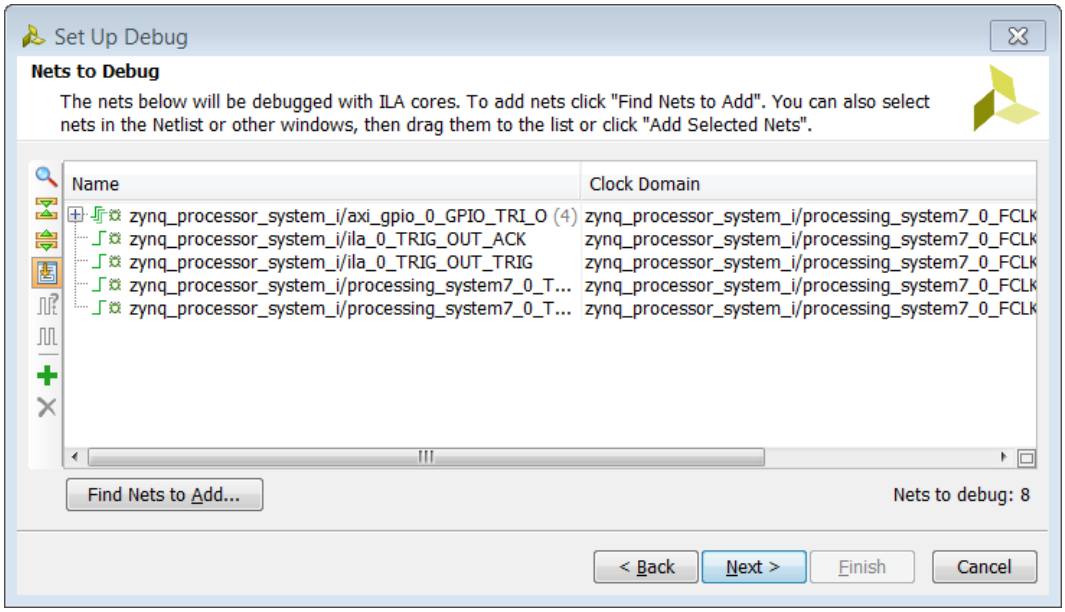

Figure 68: Specify Nets for Debugging

The ILA (Integrated Logic Analyzer) Core Options dialog box opens as shown in Figure 69.

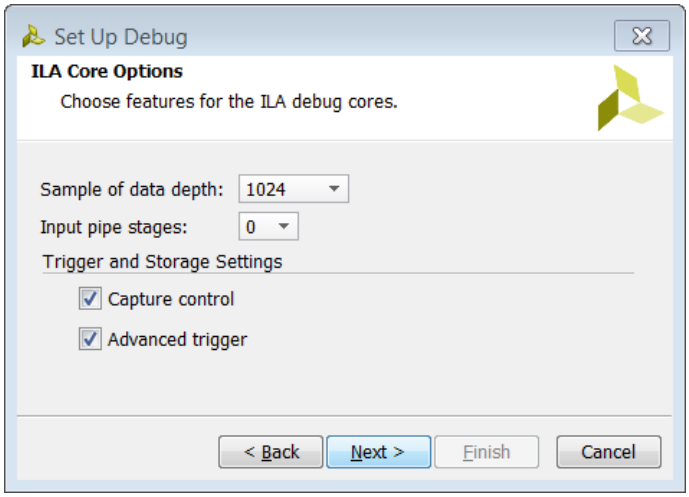

Figure 69: Enable Advanced Trigger and Capture Modes

10. In the ILA (Integrated Logic Analyzer) Core 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**.
11. 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 70. Notice that there are now no nets under the Unassigned Debug Nets folder.

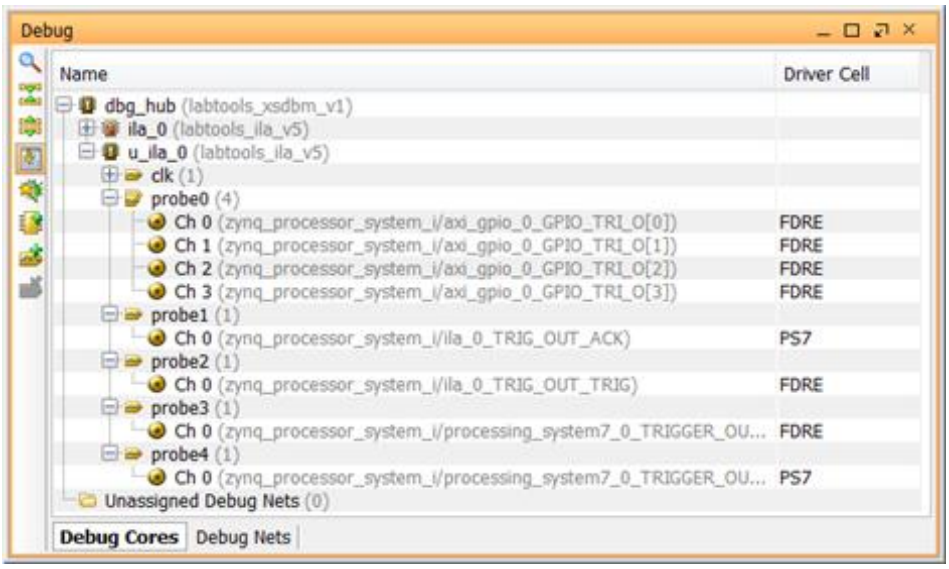

Figure 70: Debug Nets Assigned to Debug Core

Step 5: 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.
2. Click **Save**.

   The Out of Date Design dialog box opens informing you that saving the new debug related constraints could make your synthesis out of date.
3. Click **OK**.

   The **No Implementation Results Available** dialog box asks if it is okay to implement the design before generating the bitstream.
4. Click **Yes**.

   When bitstream generation completes, the **Bitstream Generation Completed** dialog box opens, with the option **Open Implemented Design** option checked by default.
5. Click **OK** to open the implemented design.
6. You will see a message box that asks about closing synthesized design before opening implemented design. Click **Yes**.
7. Ensure that all timing constraints are met by looking at the Timing Summary tab, as shown in Figure 71. Note that timing could be slightly different in your case.

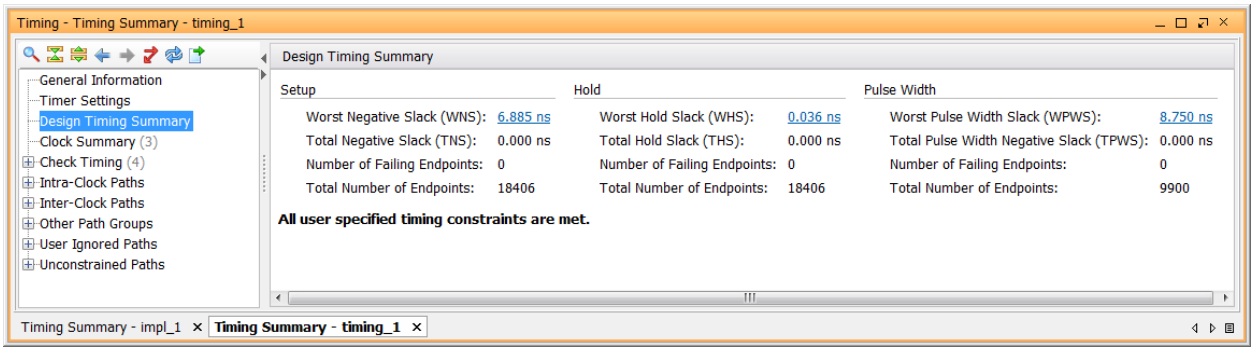

Figure 71: Timing Summary

Step 6: 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>**, as shown below in Figure 72.

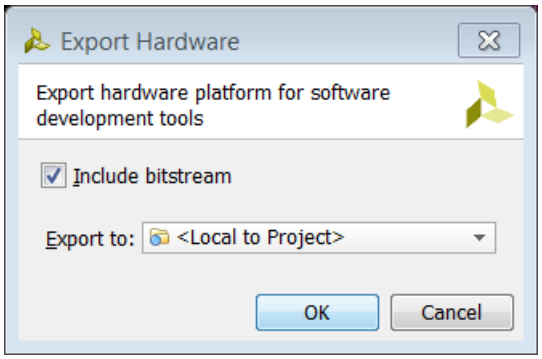

Figure 72: Export Hardware for SDK Dialog Box

3. Click **OK**.
4. Select **File > Launch SDK**. Make sure that both the **Exported location** and **Workspace** fields are set to **<Local to Project>**, as shown below in Figure 73.

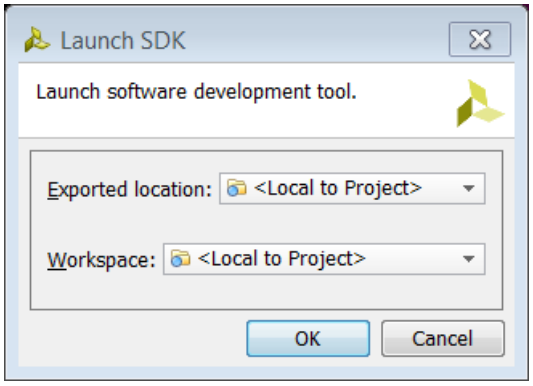

Figure 73: Launch SDK Dialog Box

5. Click **OK**.

Step 7: Build Application Code in SDK

SDK launches in a separate window.

1. After the project has been loaded, select **File > New > Application Project**.

   In the New Project dialog box, as it appears in Figure 74, specify the name for your project. For this lab, you can use the name **peri_test**.

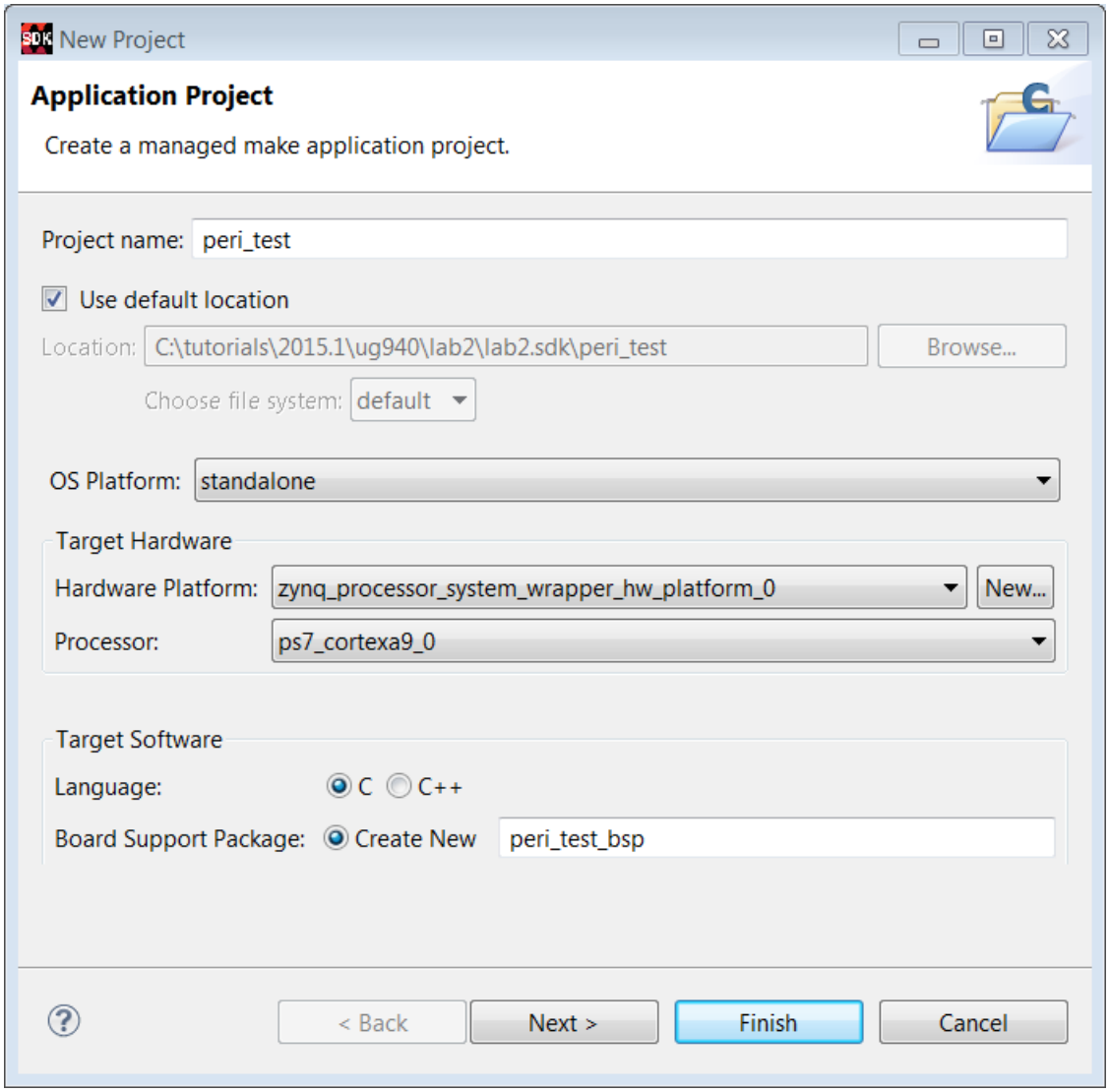

Figure 74: Name the Application Project

2. Click **Next**.

3. From the Available Templates, select **Peripheral Tests**.

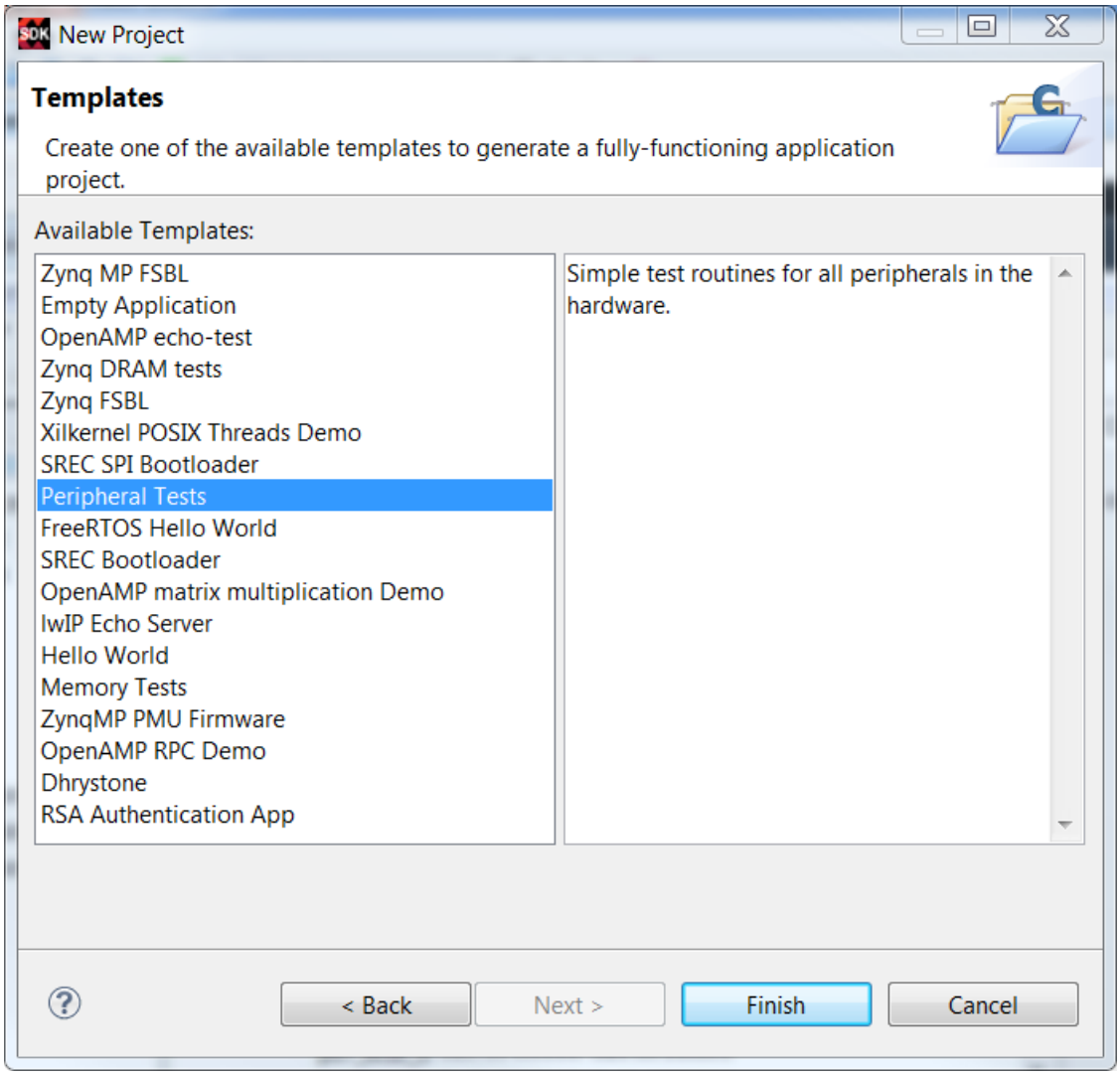

Figure 75: Select the Peripheral Tests Template

4. Click **Finish**.
5. Wait for the application to compile.
6. Make sure that you have connected the target board to the host computer and it is turned on.
7. After the application has finished compiling, select **Xilinx Tools > Program FPGA** to open the Program FPGA dialog box.

8. In the Program FPGA dialog box, click **Program**.

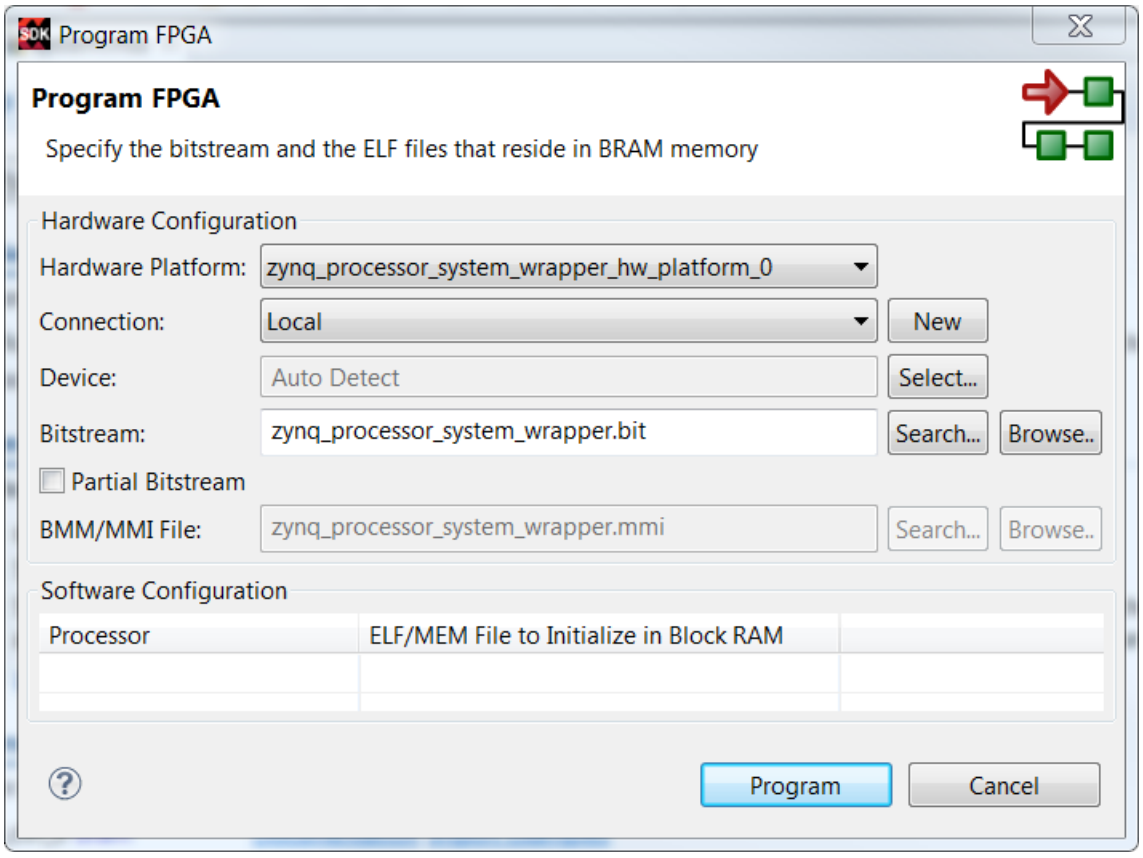

Figure 76: Program FPGA Dialog Box

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 (System Debugger**), and select **New.**

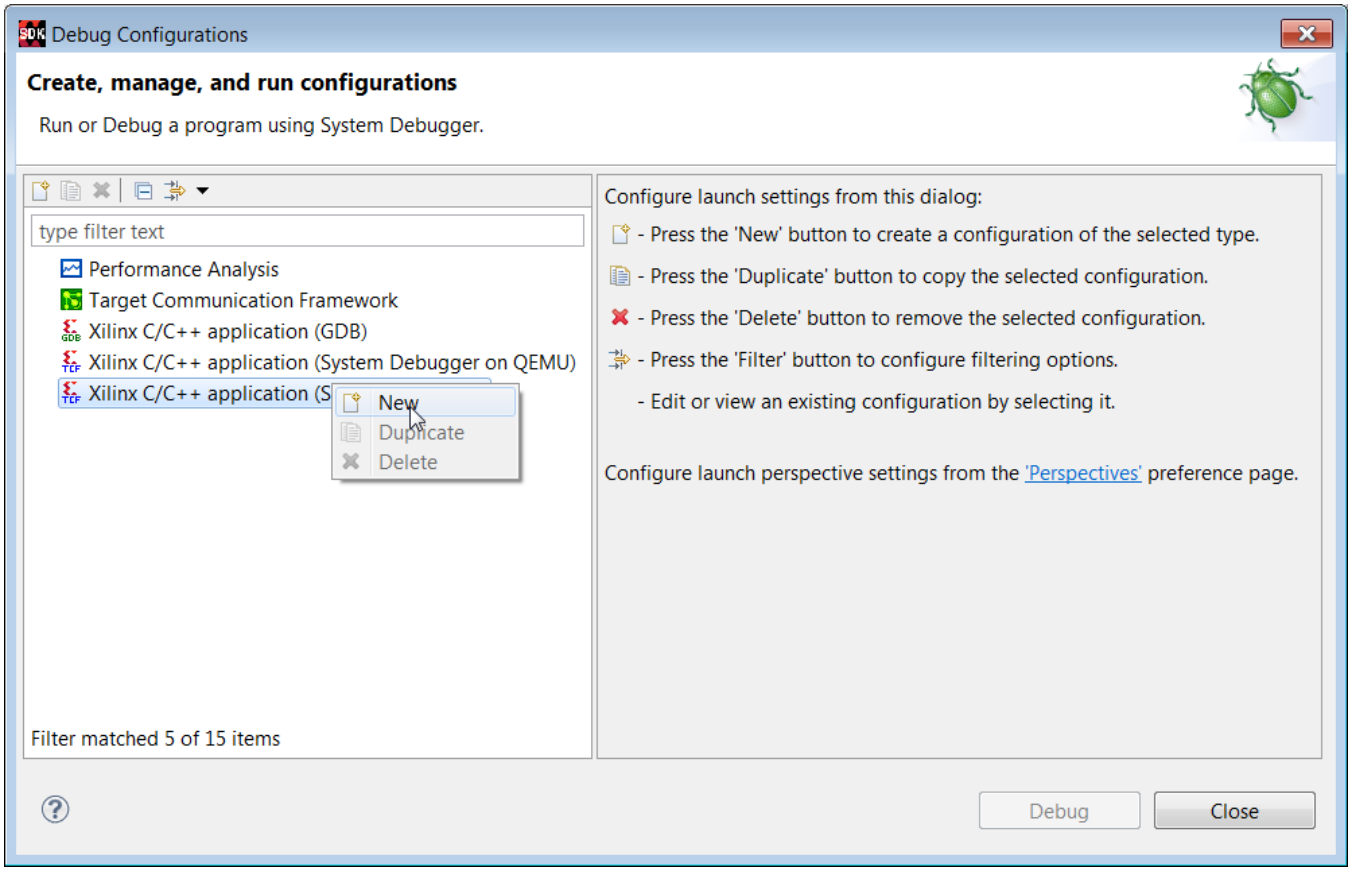

Figure 77: Create New Debug Configuration

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**, as shown at the bottom of Figure 78.

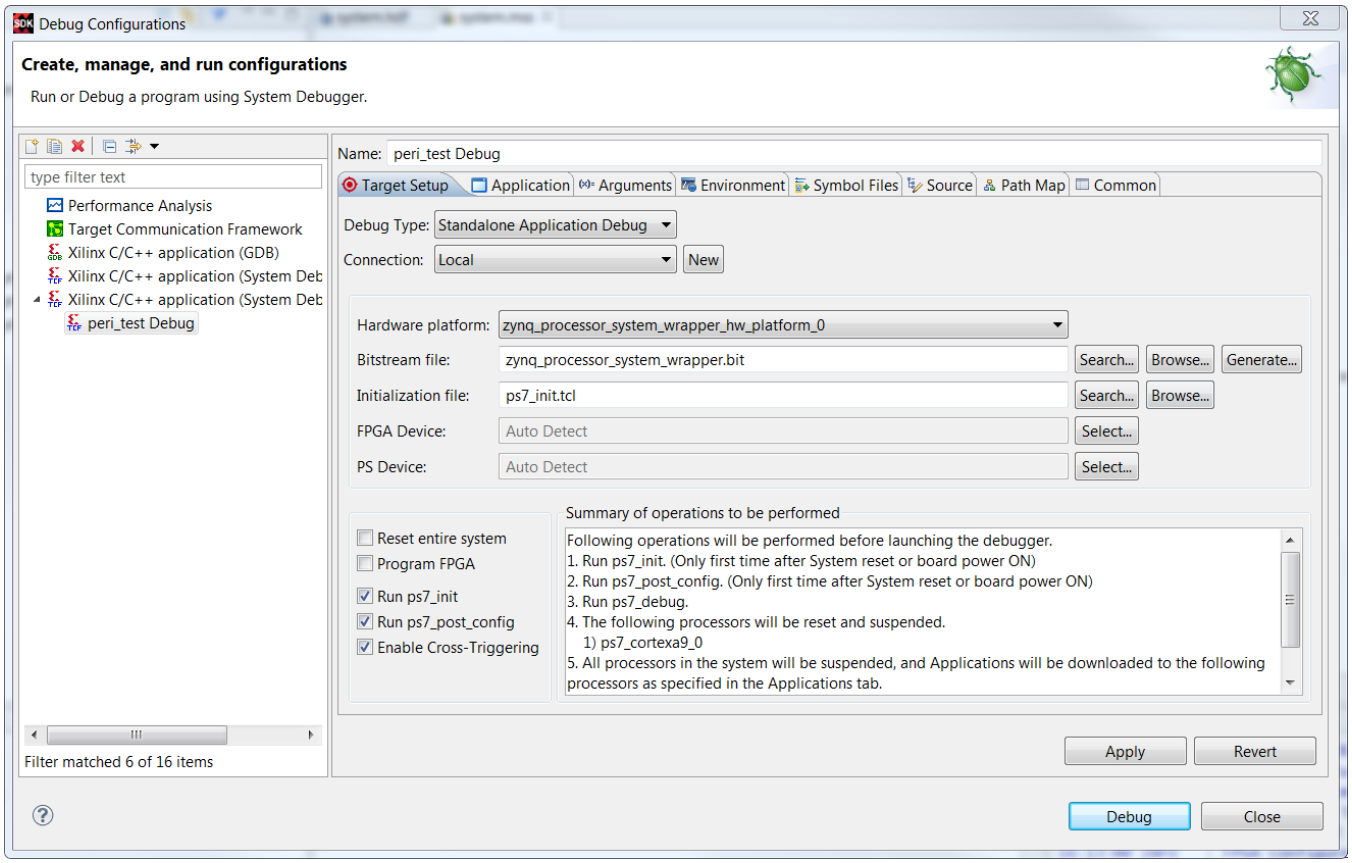

Figure 78: Enable Cross-Triggering

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 icon.

Use the following settings in Figure 79 for the ZC702 board and click **OK**.

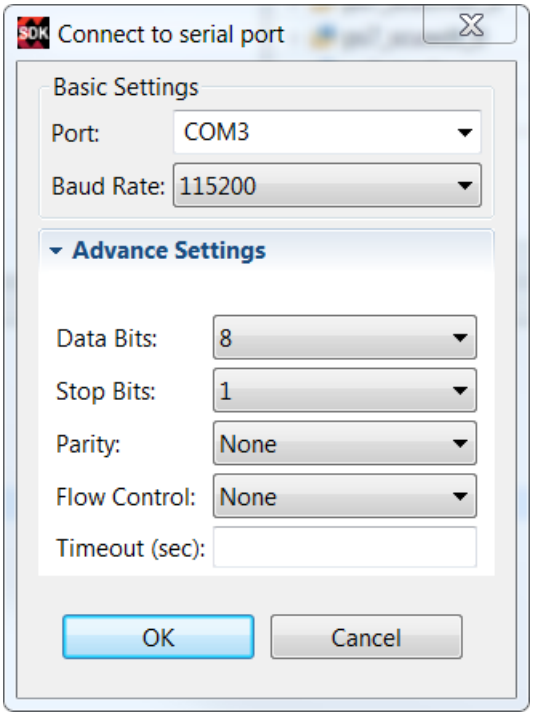

Figure 79: Terminal Settings

15. Verify the Terminal connection by checking the status at the top of the tab as shown in Figure 80.

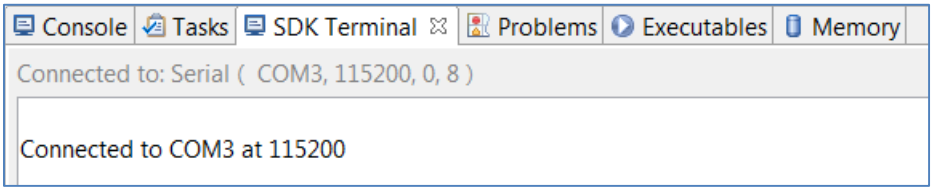

Figure 80: Verify Terminal Connection

16. If it is not already open, select `../src/testperiph.c`, and double click to open the source file.
17. Click the blue bar on the left side of the `testperiph.c` window as shown in the figure and select **Show Line Numbers**.
18. Modify the source file by inserting a while statement at line 97.
19. In line 97, add `while(1)` above in front of the curly brace as shown in Figure 81.

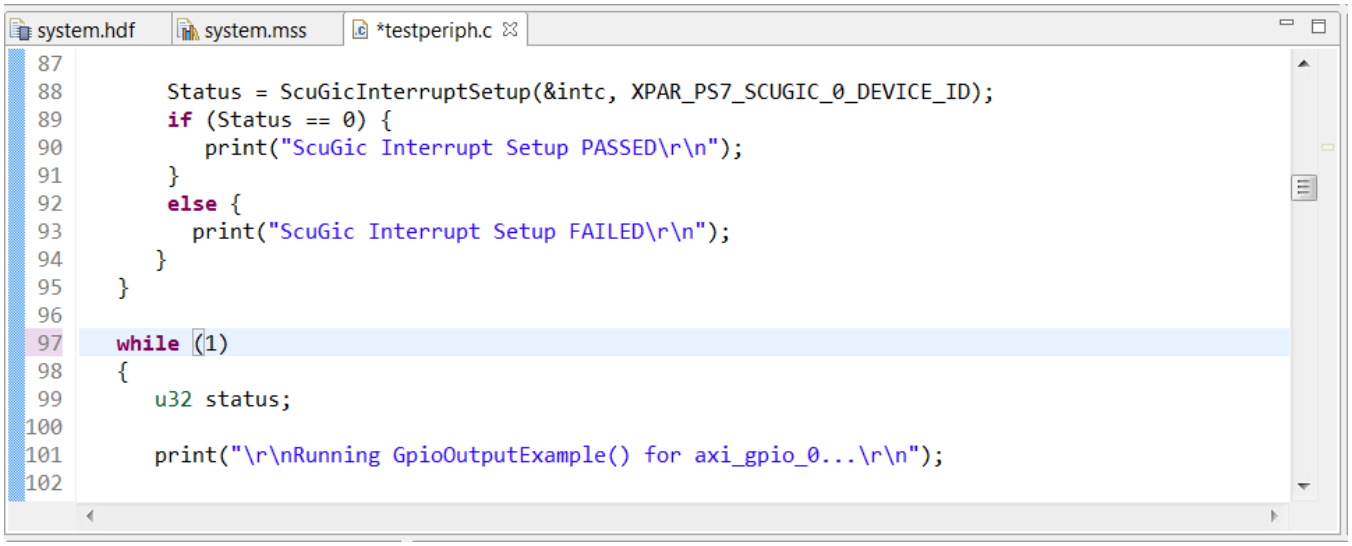

Figure 81: Modify testperiph.c

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 106 and double-click on the left pane, which adds a breakpoint on that line of code, as it appears in Figure 82. Click **Ctrl + S** to save the file. Alternatively, you can select **File > Save**.

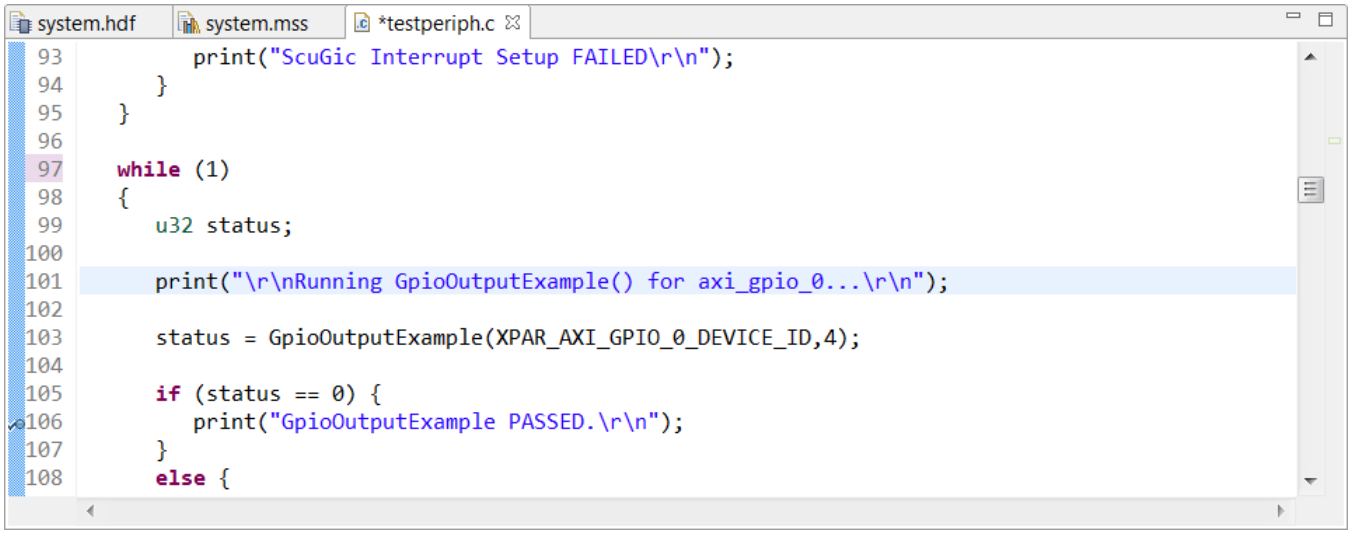

Figure 82: Set a Breakpoint

Now you are ready to execute the code from SDK.

Step 8: 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 target > Open New Target**.

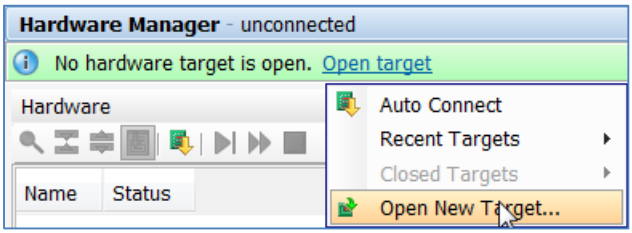

Figure 83: Open a New Hardware Target

Note: *You can also use the Auto Connect option to connect to the target hardware.*

The **Open New Hardware Target** dialog box opens, shown in Figure 84.

3. Click **Next**.

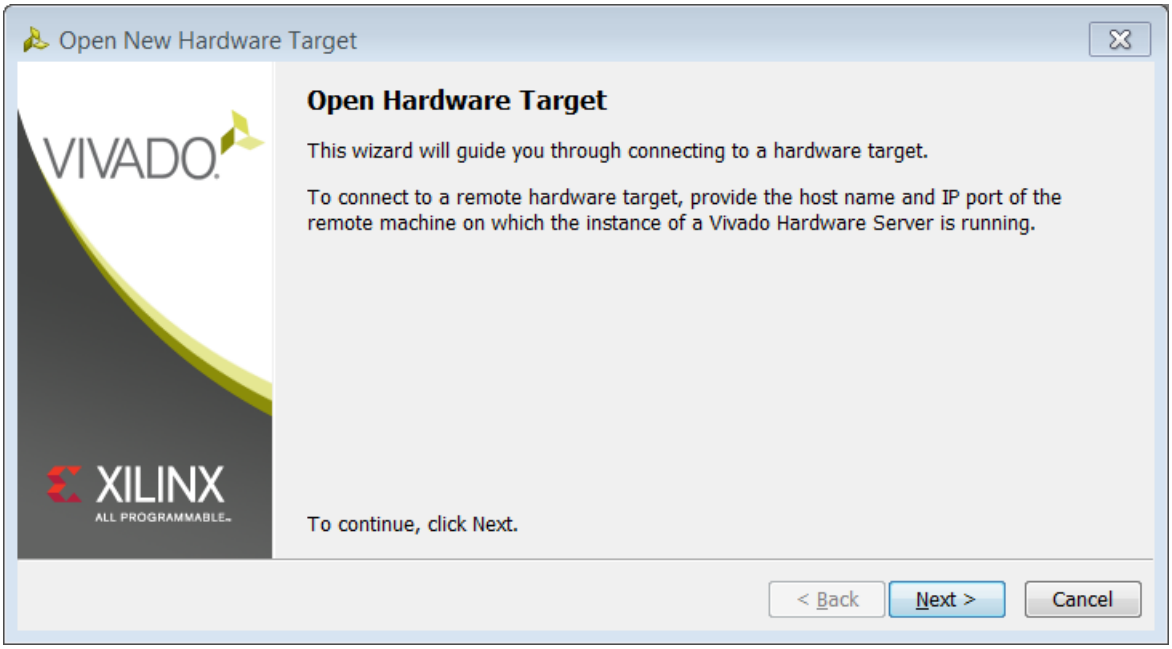

Figure 84: Open Hardware Target

On the Hardware Server Settings page, ensure that the Connect to field is set to **Local server (target is on local machine)** as shown in Figure 85.

4. Click **Next**.

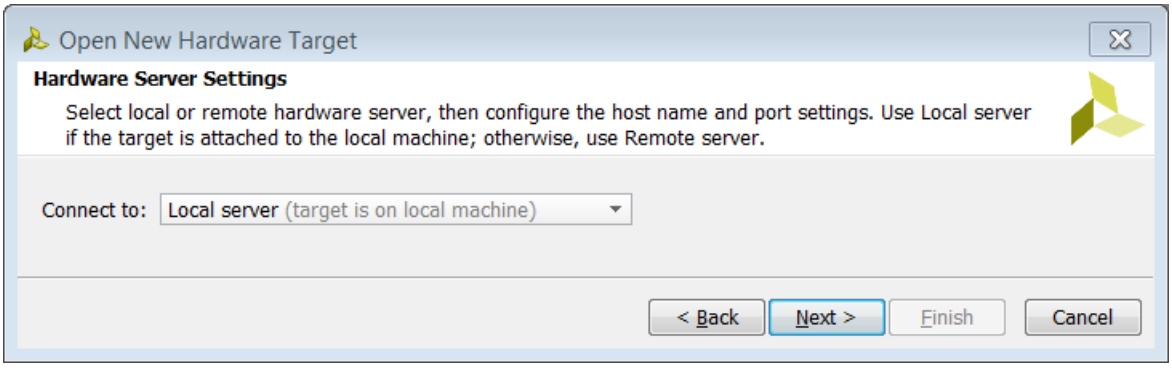

Figure 85: Specify Server Name

5. On the Select Hardware Target page, click **Next**.

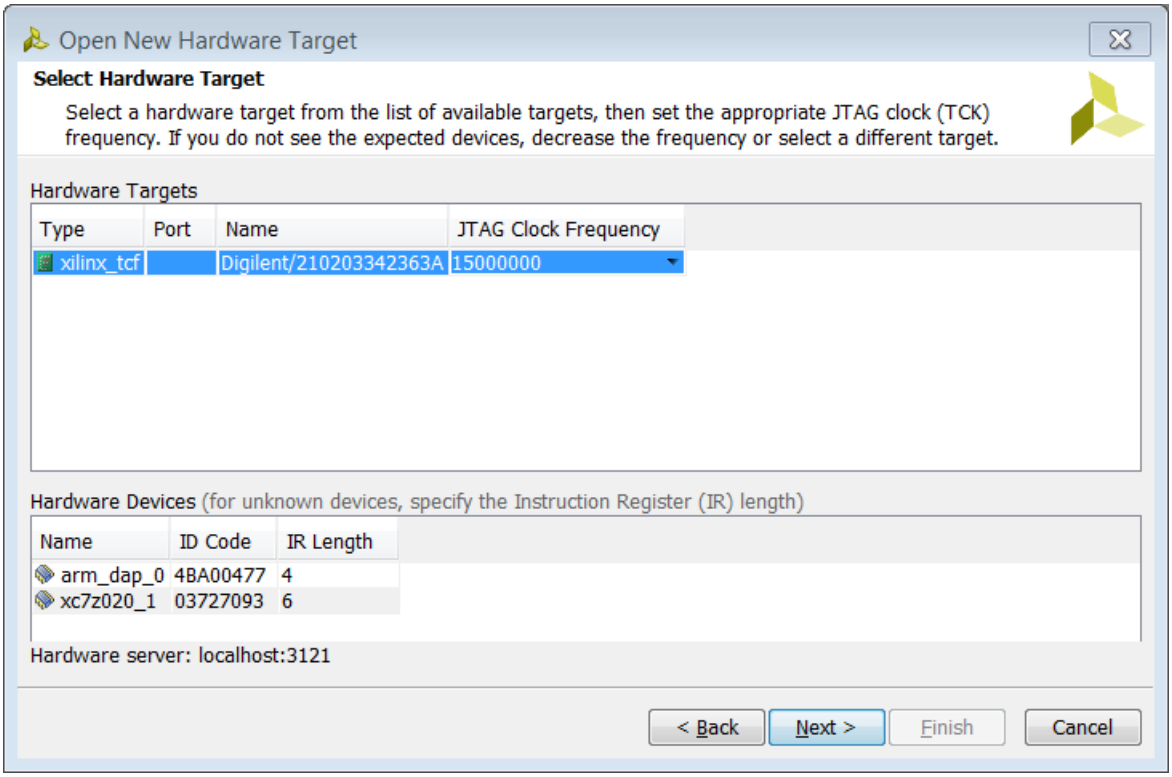

Figure 86: Select Hardware Target

6. Ensure that all the settings are correct on the **Open Hardware Target Summary** dialog box, as shown in Figure 87 and click **Finish**.

Figure 87: Open Hardware Target Summary

Step 9: Setting the Processor to Fabric Cross Trigger

When the Vivado Hardware Session successfully connects to the ZC702 board, you see the information shown in Figure 88.

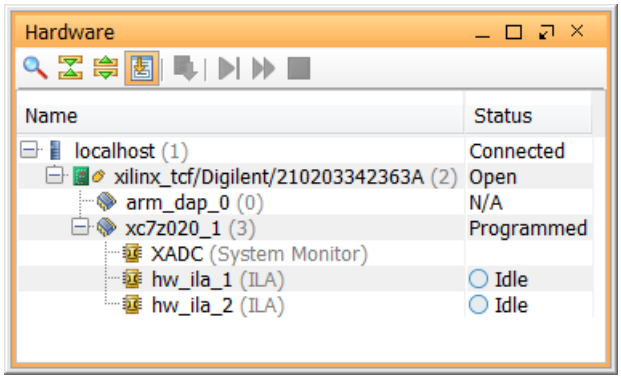

Figure 88: Vivado Hardware Window

1. 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.

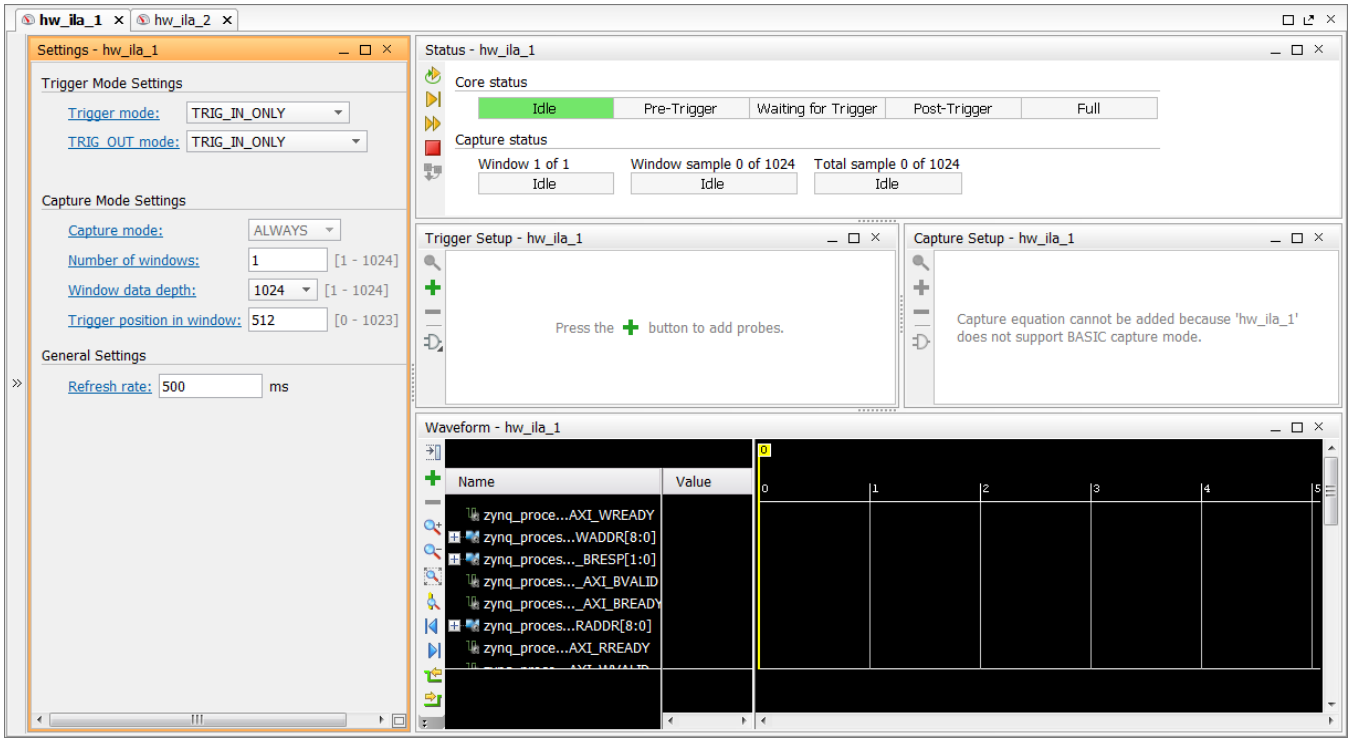

Figure 89: Set ILA Properties for hw_ila_1

2. 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 90.

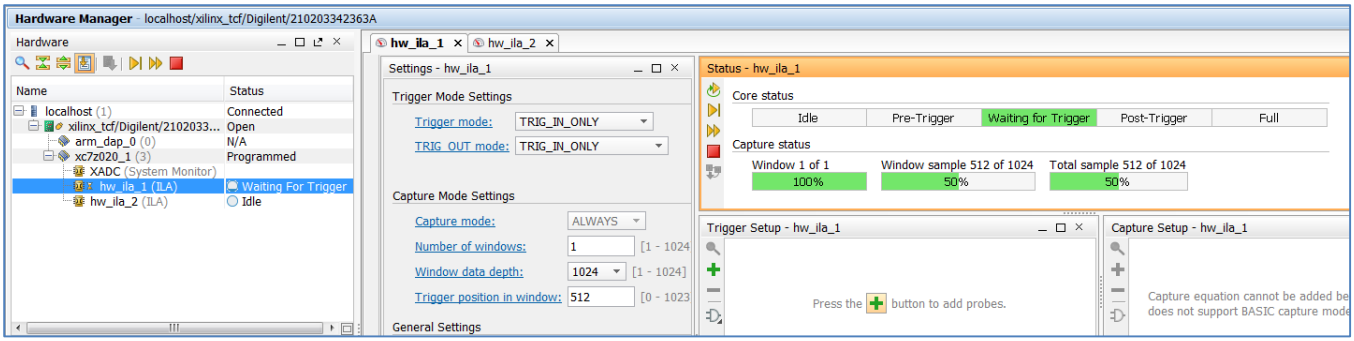

Figure 90: Armed ILA Core

3. Select the hw_ila_2 tab to set up trigger conditions to capture the trace, 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**.
   - With the hw_ila_2 tab selected, click on the **+** sign in the Trigger Setup window.

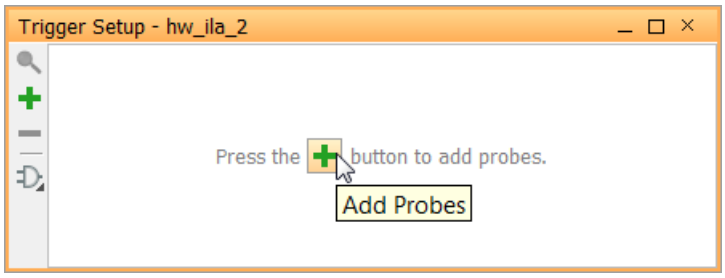

Figure 91: Add Probes to setup trigger

   - Select the `zynq_processor_system_i/processing_system7_0_TRIGGER_OUT_0_TRIG` signal from the Debug Probes window under hw_ila_2 into the **Trigger Setup** window.

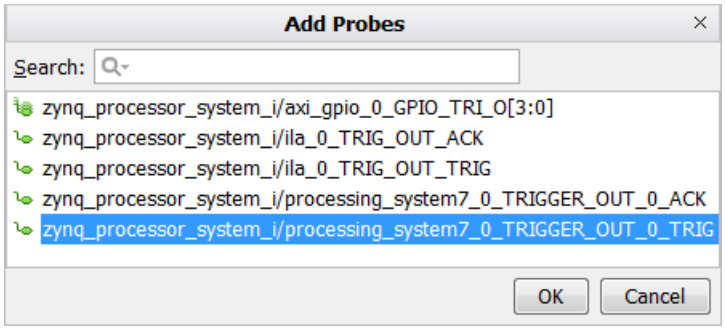

Figure 92: Add Probes window

4. Click **OK**.
5. In the Trigger Setup window, change the **Radix** to [B] Binary and **Compare Value** for the `zynq_processor_system_i/processing_system7_0_TRIGGER_OUT_0_TRIG` signal to **1**. This essentially sets up the ILA to trigger when the `trig_out` transitions to a value of 1.

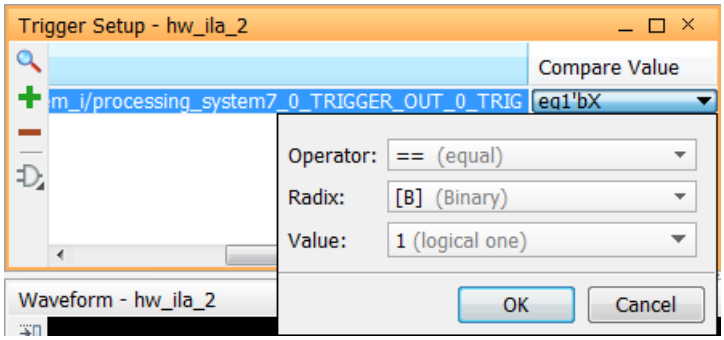

Figure 93: Set Trigger Condition

6. Click **OK**.
7. 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.
8. In SDK, in the Debug window, click the **XMD Target Debug Agent** and click the **Resume** button until the code execution reaches the breakpoint set on line 106 in the testperiph.c file. Vivado displays the hw_ila_2 trigger as shown in Figure 94.

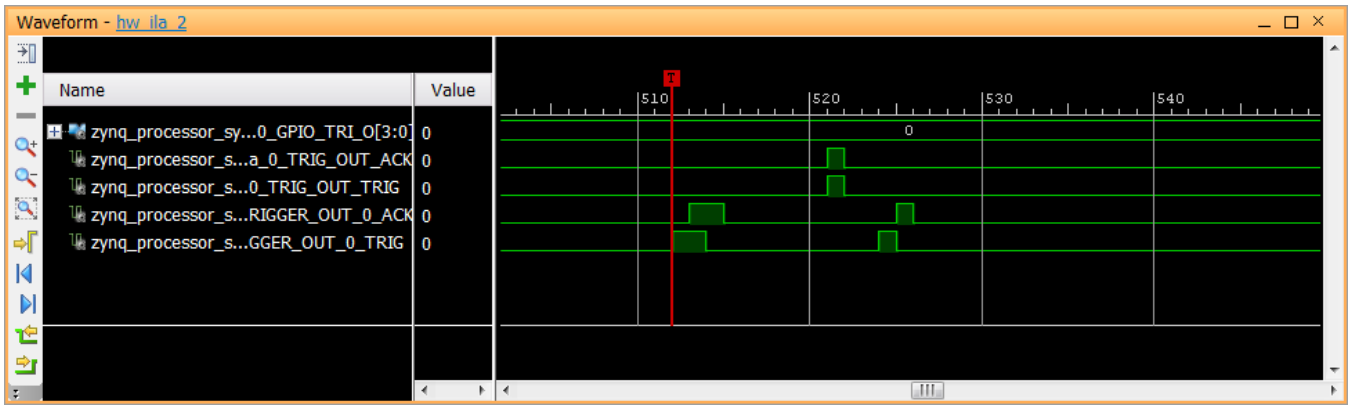

Figure 94: PS to PL Cross Trigger Waveform

Likewise, hw_ila_1 also triggers as seen in Figure 95.

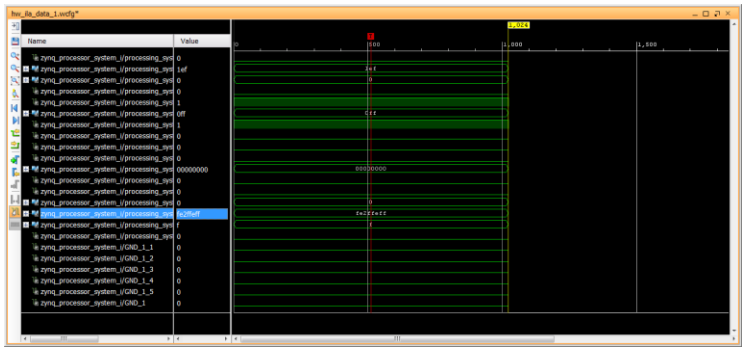

Figure 95: PS to PL Cross Trigger Waveform in hw_ila_1

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.

Step 10: Setting the Fabric to Processor Cross-Trigger

Now try the fabric to processor side of the cross-trigger mechanism. In other words, remove the breakpoint that you set earlier on line 98 to have the ILA trigger the processor and stop code execution. To do this:

1. Select the Breakpoints tab towards the top right corner of SDK window, right-click it, and uncheck the testperiph.c [line: 106] checkbox. This removes the breakpoint that you set up earlier.

   Note: Alternatively, you can select the breakpoint in line 106 of the testperiph.c file, right click and select **Disable Breakpoint**.

2. In the Debug window, right-click the **XMD Debug Target Agent** and select **Resume**. The code runs continuously because it has an infinite loop.

   You can see the code executing in the Terminal Window in SDK.

3. In Vivado, select the hw_ila_1 tab. Change the Trigger Mode to **BASIC_OR_TRIG_IN** and the TRIG_OUT mode to **TRIGGER_OR_TRIG_IN**.

4. Select the hw_ila_2 tab, and delete the existing probe in the Basic Trigger Setup window by selecting it and clicking the Remove Selected Probe "**-**" button to the left.

5. Click on the + sign in the Trigger Setup window to add the `zynq_processor_system_i/ila_0_TRIG_OUT_TRIG` signal from the Add Probes window.

6. In the Basic Trigger Setup window, change the **Radix** to [B] Binary and the **Compare Value** for the `zynq_processor_system_i/ila_0_TRIG_OUT_TRIG` signal to **1**. This essentially sets up the ILA to trigger when the trig_out transitions to a value of 1.

7. Click the **Run Trigger** button to "arm" the ILA. It moves into the "`Waiting for Trigger`" condition.

8. 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 96.

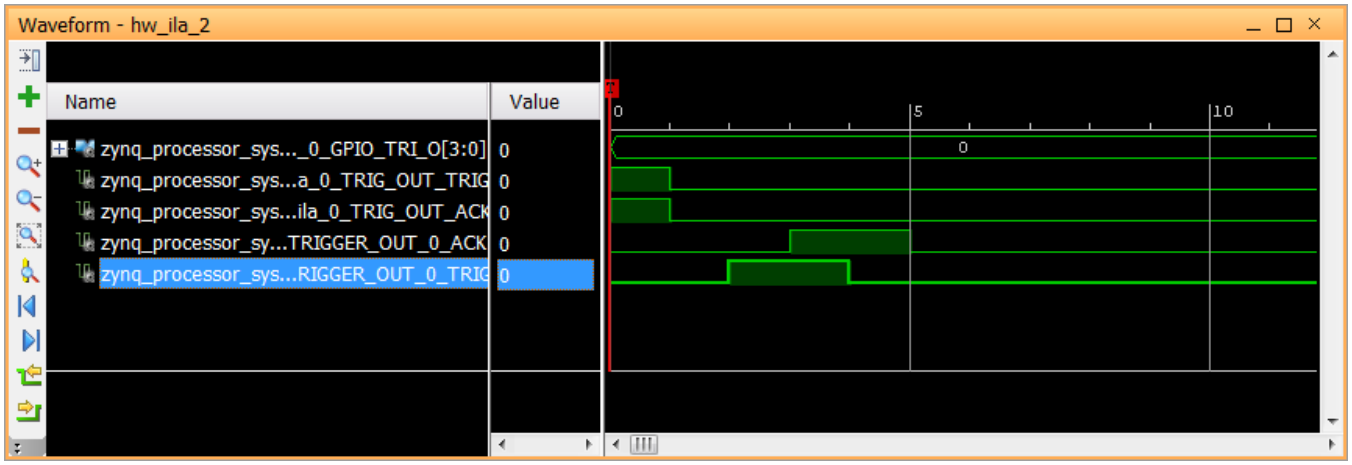

Figure 96: Waveform Demonstrating PS to PL Trigger

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.

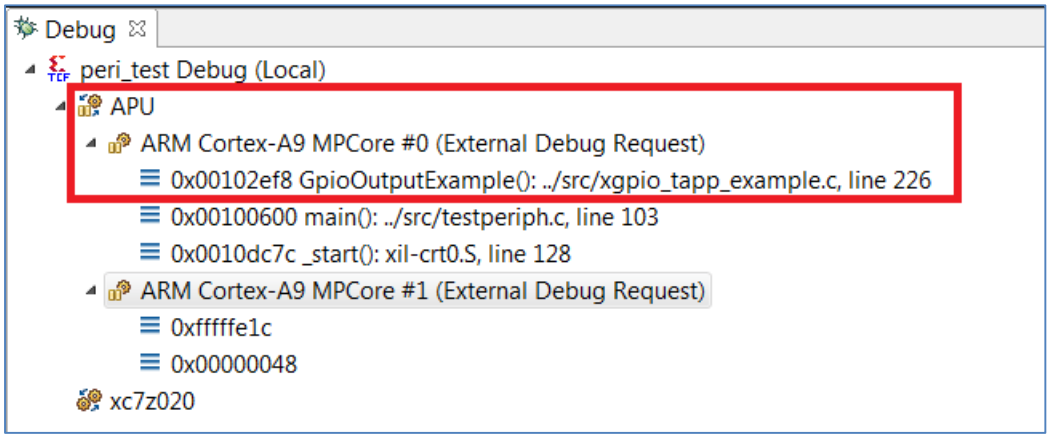

Figure 97: Verify that the Processor Has Been Interrupted in SDK

Conclusion

This lab demonstrated how cross triggering works in a Zynq-7000 AP SoC processor 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 the Zynq-7000 AP SoC processor, 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.

Lab 4: Using the Embedded MicroBlaze Processor

Introduction

In this tutorial, you create a simple MicroBlaze™ system for a Kintex®-7 FPGA using Vivado® IP integrator.

The MicroBlaze system includes native Xilinx® IP including:

- MicroBlaze processor
- AXI block RAM
- Double Data Rate 3 (DDR3) memory
- UARTLite
- GPIO
- Debug Module (MDM)
- Proc Sys Reset
- Local memory bus (LMB)

Parts of the block design are constructed using the Platform Board Flow feature. This lab also shows the cross-trigger capability of the MicroBlaze processor. The feature is demonstrated using a software application code developed in SDK in a stand-alone application mode.

This lab targets the Xilinx KC705 FPGA Evaluation Board.

Step 1: Invoke the Vivado IDE and Create a Project

1. Open the Vivado IDE by clicking the desktop icon or by typing **vivado** at a terminal command line.

   From the Quick Start page, select **Create New Project**.

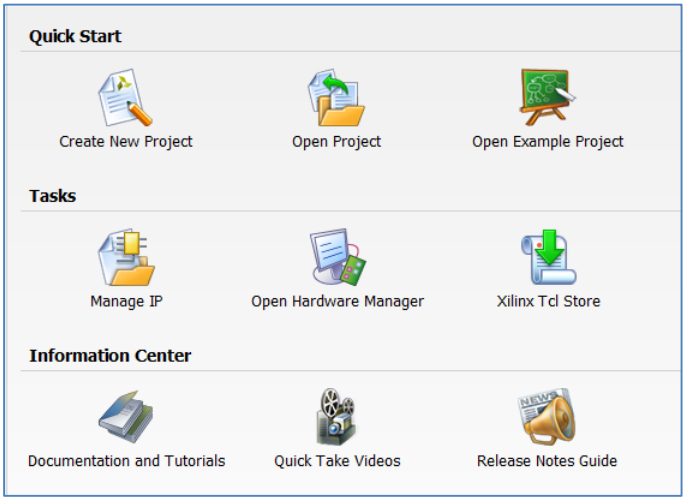

Figure 98: Vivado Quick Start Page

   The New Project wizard opens.

2. In the **Project Name** dialog box, type the project name and location. Make sure that **Create project subdirectory** is checked. Click **Next**.
3. In the **Project Type** dialog box, select **RTL Project**. Click **Next**.
4. 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**.
5. Click **Next**.
6. In **Add Existing IP** dialog box, click **Next**.
7. In **Add Constraints** dialog box, click **Next**.
8. In the Default Part dialog box, select **Boards** and choose the **Kintex-7 KC705 Evaluation Platform** along with the correct version. Click **Next**.
9. 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 in the Tcl Console:

```
set_property board_part xilinx.com:kc705:part0:1.2 [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. You can also refer to the *Vivado Design Suit Tcl Command Reference Guide* (UG835) for additional information about Tcl commands.

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. For this step, you can use mb_subsystem as the Design name. Leave the Directory field set to its default value of <**Local to Project**>.

3. Leave the Specify source set drop-down list set to its default value of **Design Sources**.

4. Click **OK** in the Create Block Design dialog box, shown in Figure 99.

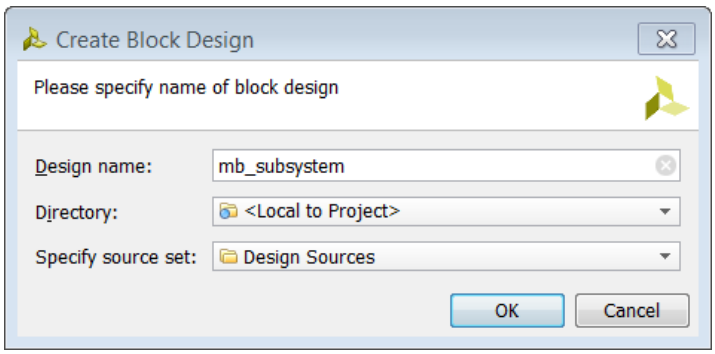

Figure 99: Name Block Design

5. 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** icon in the middle of the canvas.

Figure 100: Add IP

6. As shown in Figure 101, type **micr** in the Search field to find the MicroBlaze IP, then select **MicroBlaze** and press the **Enter** key.

   Note: The IP Details window can be displayed by clicking **CTRL+Q** key on the keyboard.

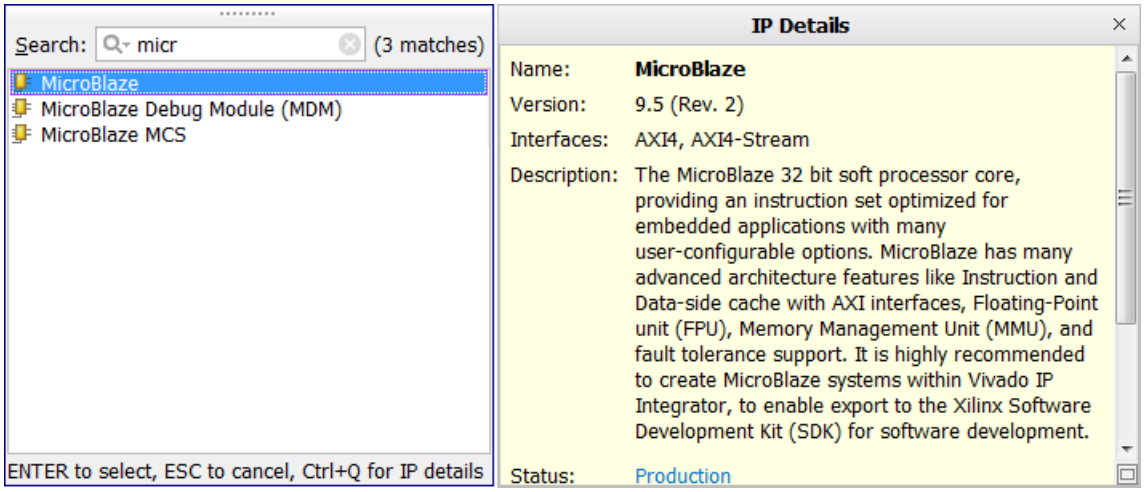

Figure 101: Search Field

Use the Board Tab to Connect to Board Interfaces

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

1. Click the **Board** tab. You can see that there are several components listed in that tab. These components are present on the KC705 board. These components are all listed under different categories in the Board window.

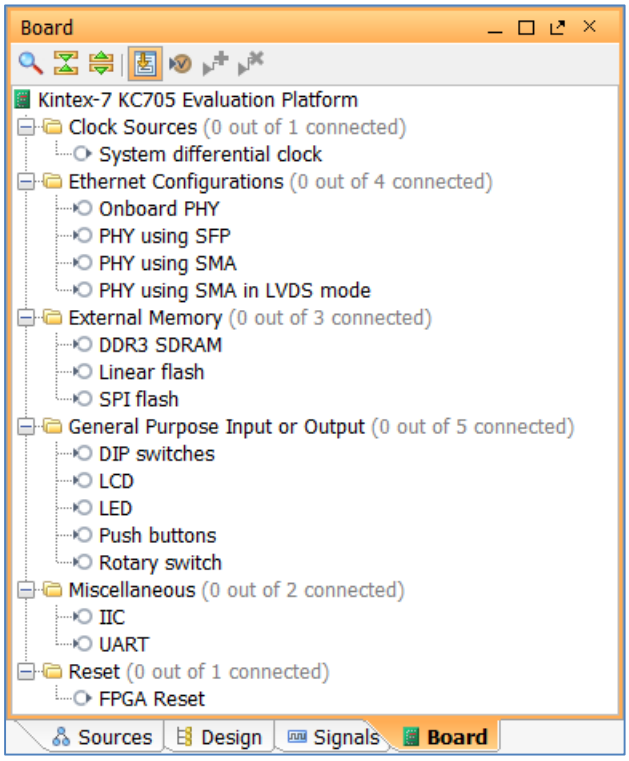

Figure 102: Using the Board Part Interfaces to Configure a MIG

2. From the **External Memory** folder, drag and drop the DDR3 SDRAM component into the block design canvas.

   The Auto Connect dialog box opens, as shown in Figure 103, informing you that the MIG IP was instantiated on the block design and then connected to DDR3 SDRAM component on the board.

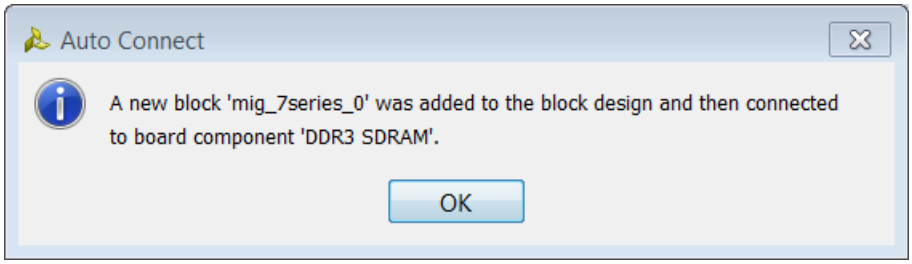

Figure 103: Auto-connect dialog box for DDR3

   Note: The order of instantiation of these IP are important as they affect the options available with Designer Assistance. As an example, when the DDR3 component is instantiated, the Block Automation option for the MicroBlaze will enable caching by default.

3. Click **OK**.

The block design looks like Figure 104.

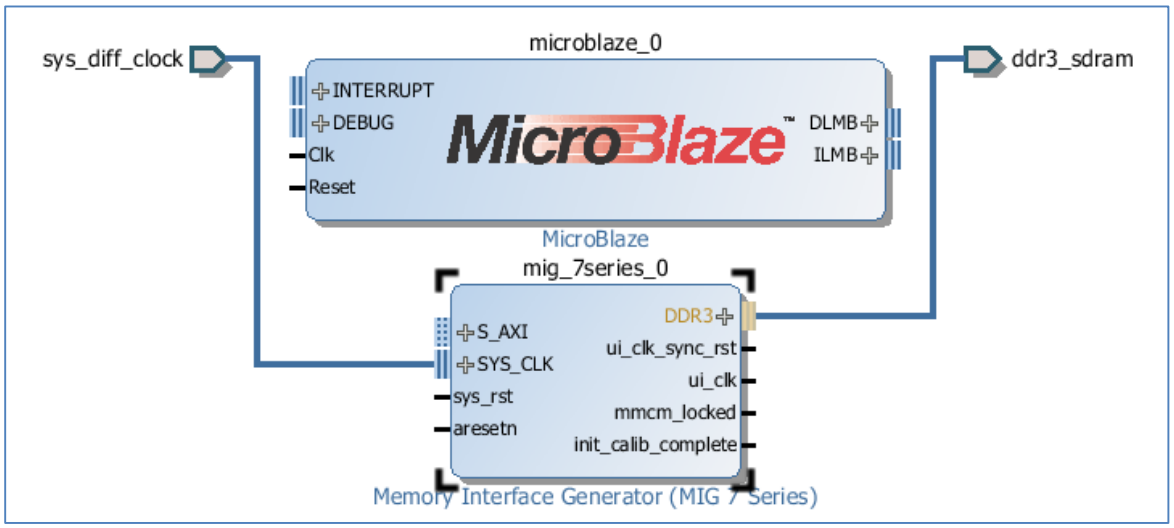

Figure 104: Block Design After Instantiating the MIG Core

4. In the Board window, notice that the DDR3 SDRAM interface now is connected as shown by the circle .

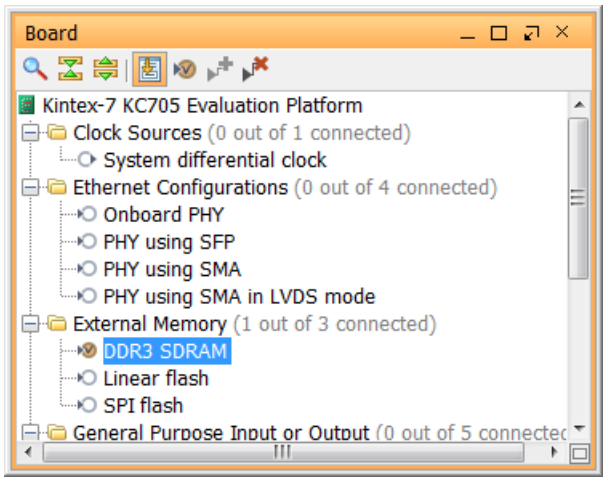

Figure 105: DDR3 Interface Shown Under Connected Interfaces

5. From the Board window, select **UART** under the miscellaneous folder and drag and drop it into the block design canvas.
6. Click **OK** in the Auto Connect dialog box.

   This instantiates the AXI Uartlite IP on the block design.

7. Click **OK** in the Auto Connect dialog box.
8. Likewise, from the Board window, select **LED** under the General Purpose Input or Output folder and drag and drop it into the block design canvas.
9. Click **OK** in the Auto Connect dialog box.

   This instantiates the GPIO IP on the block design and connects it to the on-board LEDs.
10. The block design now should look like Figure 106.

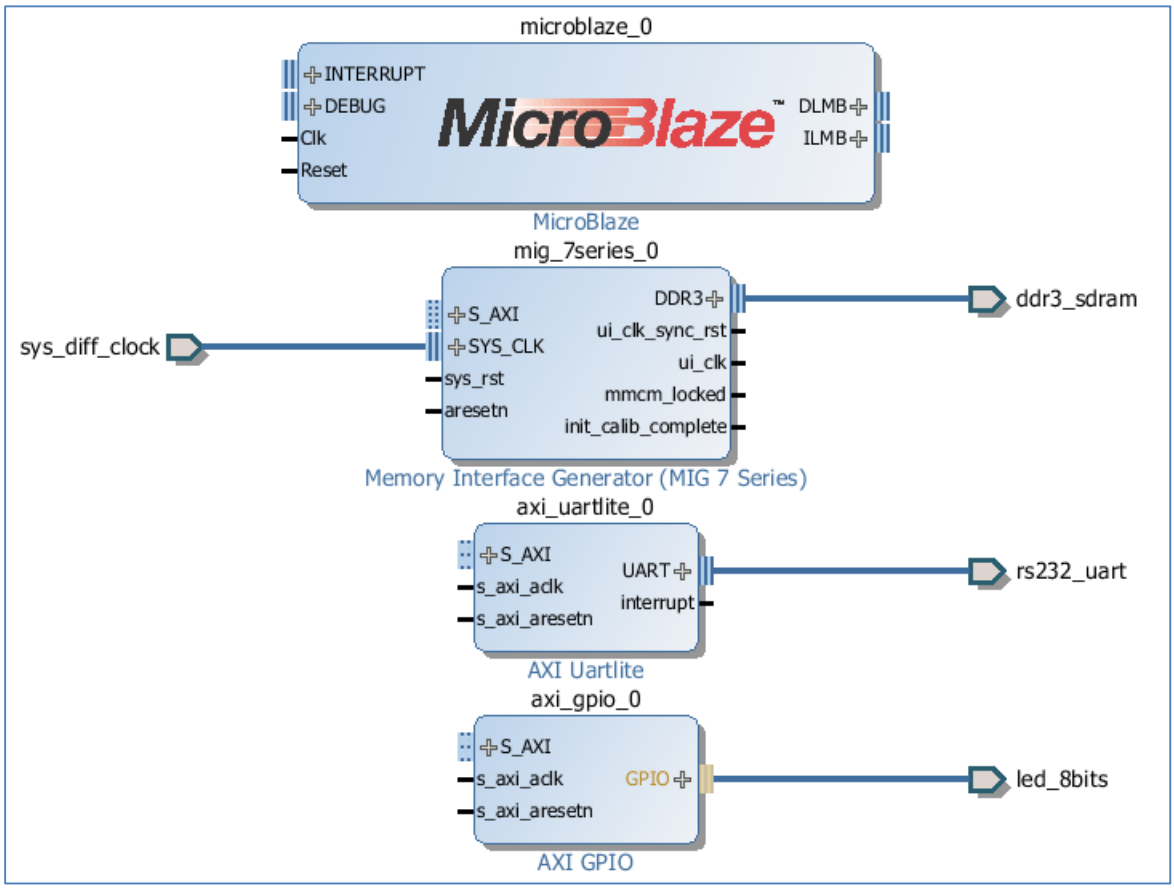

Figure 106: Block Design After Connecting the Rs232_Uart Interface

Add Peripheral: AXI BRAM Controller

1. Add the AXI BRAM Controller shown in Figure 107 by right-clicking on the IPI canvas and selecting **Add IP**.

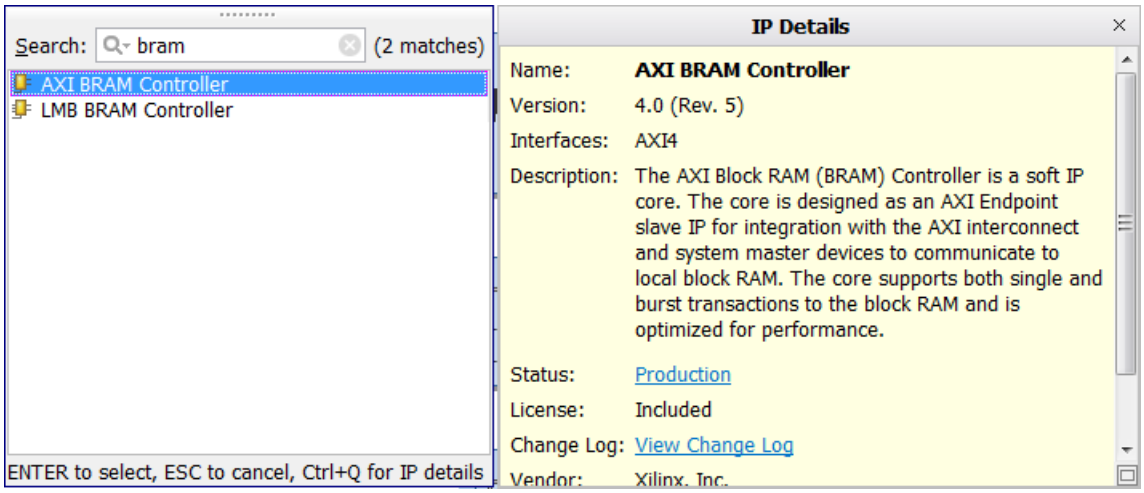

Figure 107: Add BRAM Controller

Run Block Automation

1. Click Run Block Automation, displayed in Figure 108.

Figure 108: Run Block Automation

The Run Block Automation dialog box opens, as shown in Figure 109.

The values of the fields shown in this figure show the values that you will set in the next step.

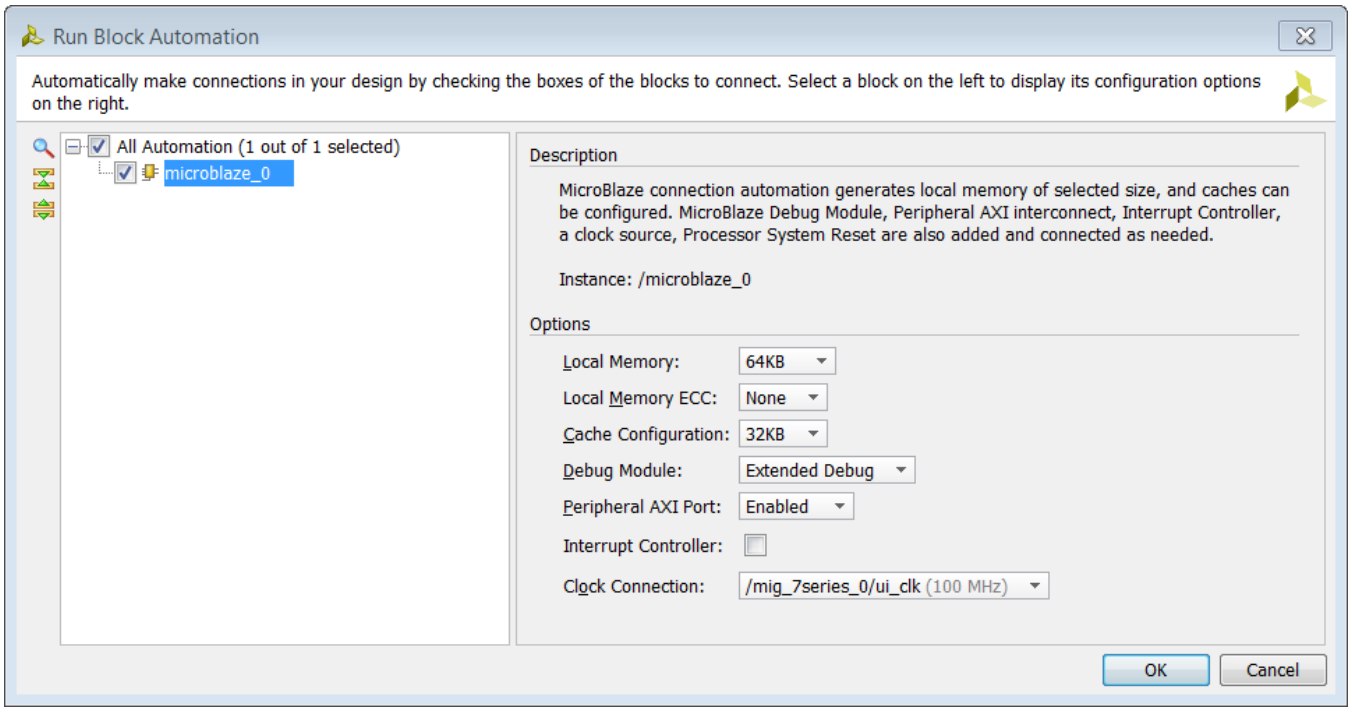

Figure 109: Run Block Automation Dialog Box

2. On the Run Block Automation page:
   a. Set **Local Memory** to **64 KB**.
   b. Leave the **Local Memory ECC** to its default value of **None**.
   c. Change the **Cache Configuration** to **32 KB**.
   d. Change the **Debug Module** option to **Extended Debug**.
   e. Leave the Peripheral AXI Port option set to its default value of **Enabled**.
   f. Leave the **Clock Connection** option set to `/mig_7series_0/ui_clk (100 MHz)`.
3. Click **OK**.

   This generates a basic **MicroBlaze** system in the IP Integrator diagram area, shown in Figure 110.

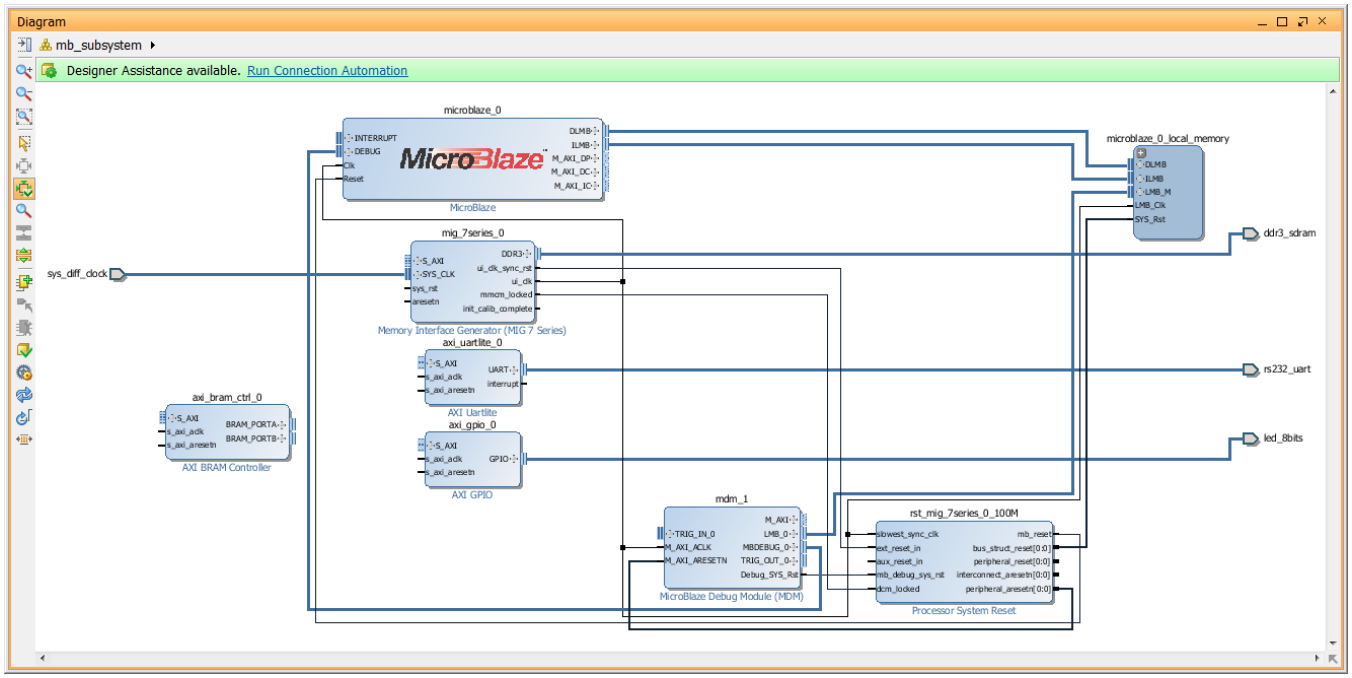

Figure 110: MicroBlaze System

Use Connection Automation

Run Connection Automation provides several options that you can select to make connections. This section will 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** as shown in Figure 111.

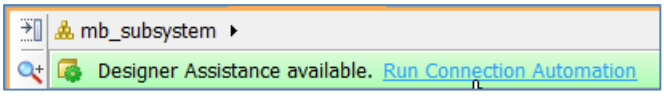

Figure 111: Run Connection Automation

The Run Connection Automation dialog box opens.

2. Check the interfaces in the left pane of the dialog box as shown in Figure 112.

Note: You will connect the MDM interfaces later.

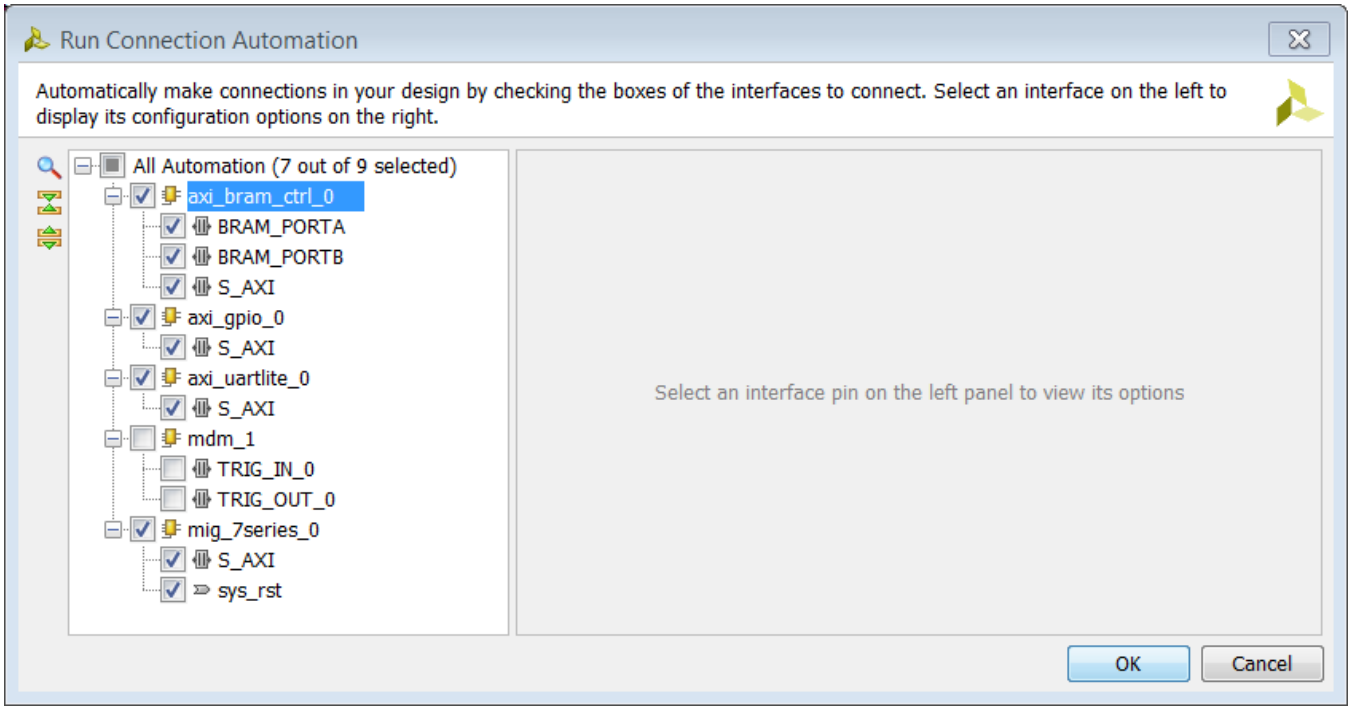

Figure 112: Run Connection Automation Dialog Box

Now, use the table below to set options in the Run Connection Automation dialog box.

Connection	More Information	Setting
axi_bram_ctrl_0 • BRAM_PORTA	The only option for this automation is to instantiate a new Block Memory Generator as shown under options.	
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.	Leave the Blk_Mem_Gen option to its default value of **Blk_Mem_Gen of BRAM_PORTA**.
axi_bram_ctrl_0 • S_AXI	Two options are presented in this case. The **Master** field can be set for either cached or non-cached accesses.	The Run Connection Automation dialog box offers to connect this to the **/microblaze_0 (Cached)**. Leave it to its default value. In case, cached accesses are not desired this could be changed to **/microblaze_0 (Periph)**. Leave the Clock Connection (for unconnected clks) field set to its default value of **Auto**.
axi_gpio • S_AXI	The **Master** field is set to **/microblaze_0 (Periph)**. The **Clock Connection (for unconnected clks)** field is set to its default value of **Auto**.	Keep these default settings.
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 its default value of **Auto**.	Keep these default settings.
mig_7series_0 • S_AXI	The **Master** field is set to **/microblaze_0 (Cached)**. Leave it to this value so the accesses to the DDR3 memory are cached accesses. The **Clock Connection (for unconnected clks)** field is set to its default value of **Auto**.	Keep these default settings.

mig_7series_0 • sys_rst	The board interface reset will be connected to the reset pin of the MIG controller.	Keep the default setting.

3. After setting the appropriate options as shown in the table above, click **OK**.

   At this point, your IP integrator diagram area should look like Figure 113. The relative placement of your IP might be slightly different.

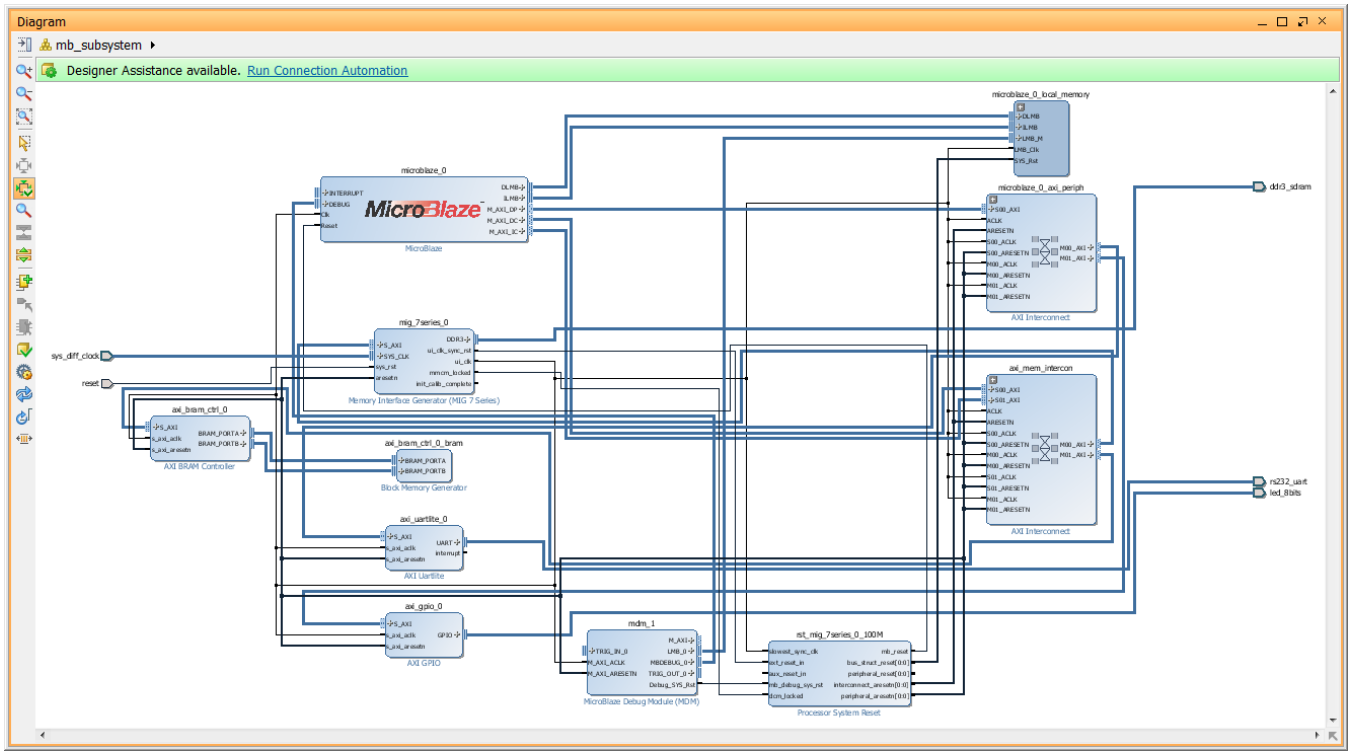

Figure 113: MicroBlaze Connected to UART, GPIO and AXI Timer

Connect the Cross-Trigger pins of the MDM

1. Click on the **Run Connection Automation** link in the banner at the top of the block design canvas. The Run Connection Automation dialog box opens.
2. Check the TRIG_IN_0 interface under mdm_1. Click **OK**.

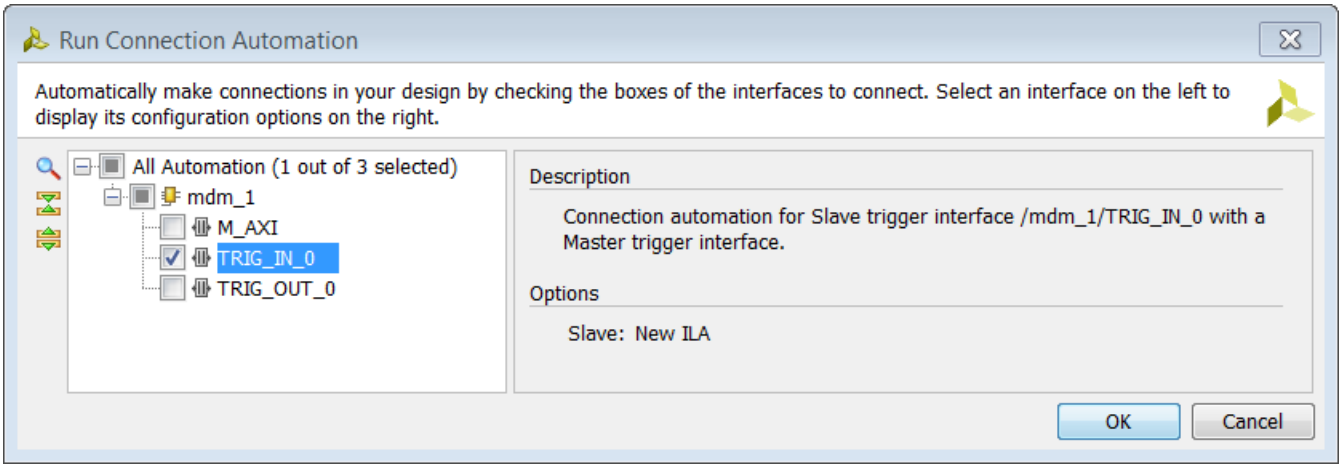

Figure 114: Run Connection Automation to connect the TRIG_IN_0 interface pin of the MDM

This instantiates an Integrated Logic Analyzer core in the block design and connects its TRIG_OUT pin to the TRIG_IN_0 interface pin of the MDM. In the next step, you will reuse this ILA to connect the TRIG_OUT_0 interface pin of the MDM.

3. Double click on the ila_0 instance to open the Re-customize IP dialog box. Check the **Trigger In Port** option in the dialog box.

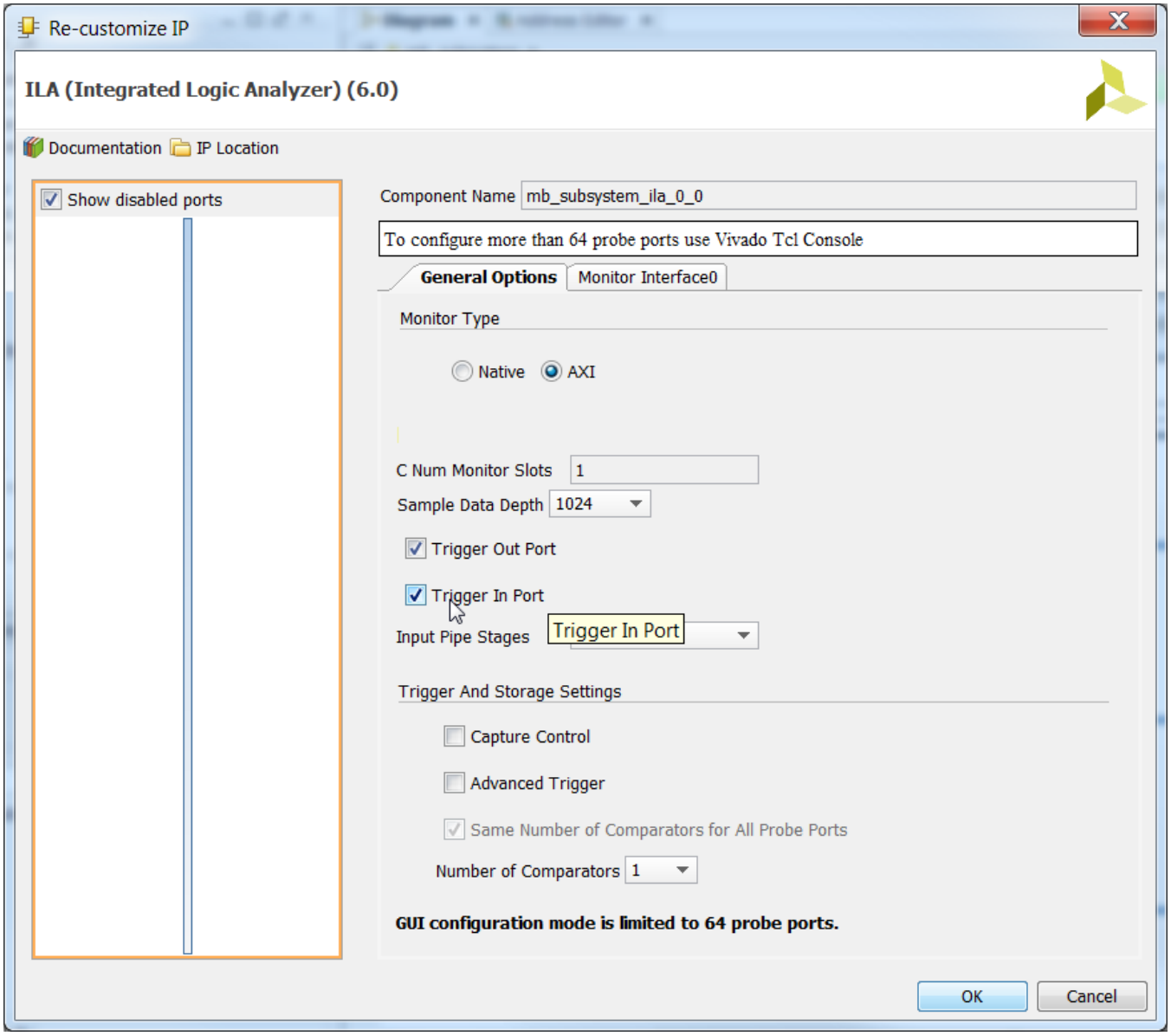

Figure 115: Enable Trigger In Port by recustomizing the ILA

4. Click **OK**.
5. Click on the **Run Connection Automation** link at the top of block design canvas again.
6. In the Run Connection Automation dialog box select TRIG_OUT_0 interface under mdm_1. Leave the Slave field set to **/ila_0/TRIG_IN**.

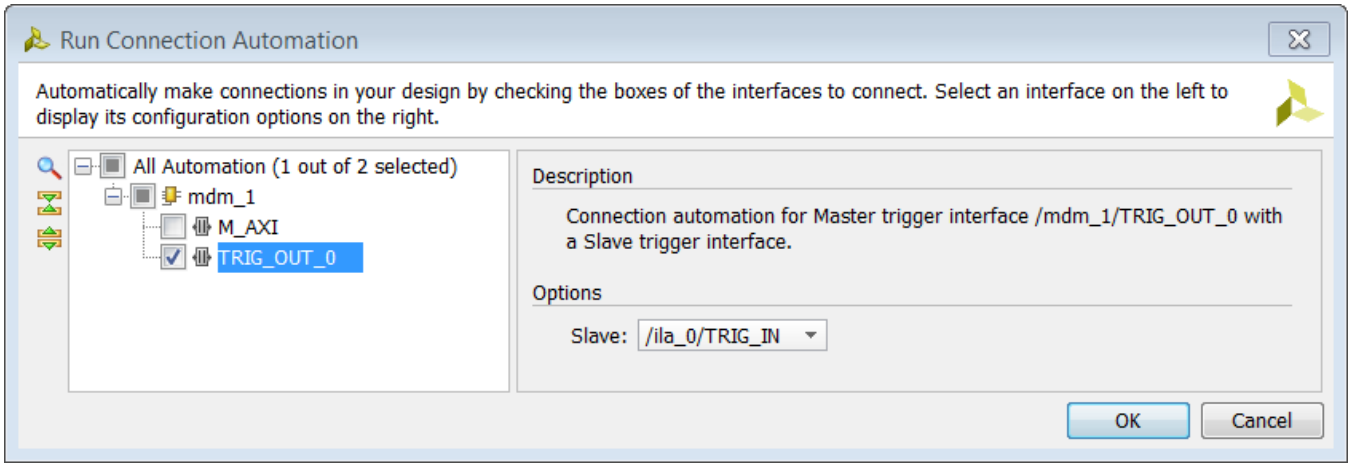

Figure 116: Run Connection Automation to connect the TRIG_OUT_0 interface pin of the MDM

7. Click **OK**.
8. To monitor the cross trigger pins in the ILA, you need to mark the nets connecting these pins for debug. To do this, select the net connecting TRIG_OUT_0 interface pin of the MicroBlaze Debug Module mdm_0 and the TRIG_IN interface pin of the ILA instance ila_0, right-click and select **Mark Debug** from the context menu.
9. Likewise, select the net connecting interface pin TRIG_OUT of ila_0 and TRIG_IN_0 interface pin of mdm_1, right-click and select **Mark Debug** from the context menu. A tiny bug symbol appears on the nets marked for debug.
10. Next, connect the clk pin of ila_0 to ui_clk pin of the mig_7series_0.
11. To monitor the AXI transactions taking place between the MicroBlaze and the GPIO, connect the net connecting the interface pins M01_AXI of the microblaze_0_axi_periph instance and the S_AXI interface pin of the axi_gpio_0 instance, to the SLOT_0_AXI pin of the ILA instance, ila_0.
12. The Cross-Trigger interface connections should look as follows.

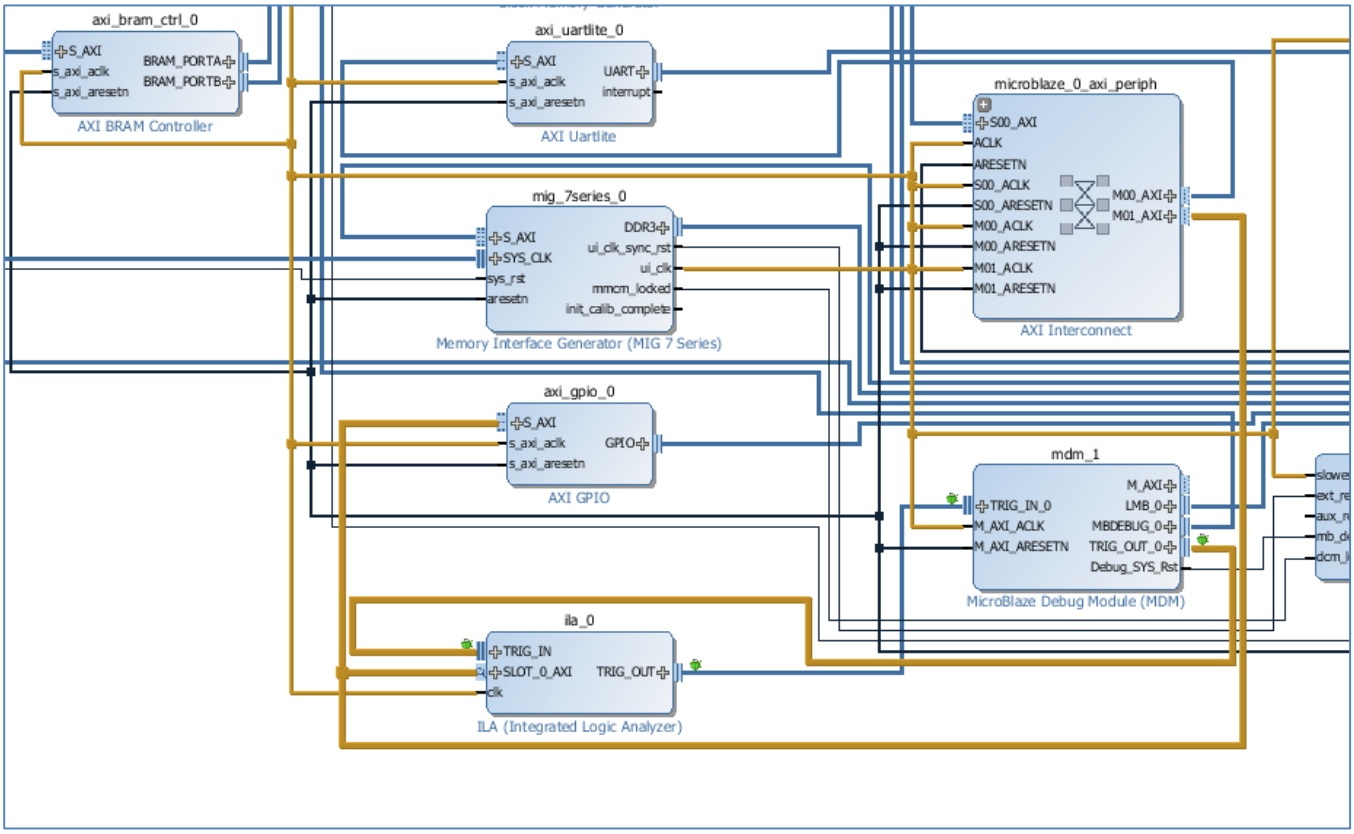

Figure 117: ILA connections to the Cross-Trigger Interface

13. Click the **Regenerate Layout** button in the IP Integrator toolbar to generate an optimum layout for the block design. The block diagram looks like Figure 118.

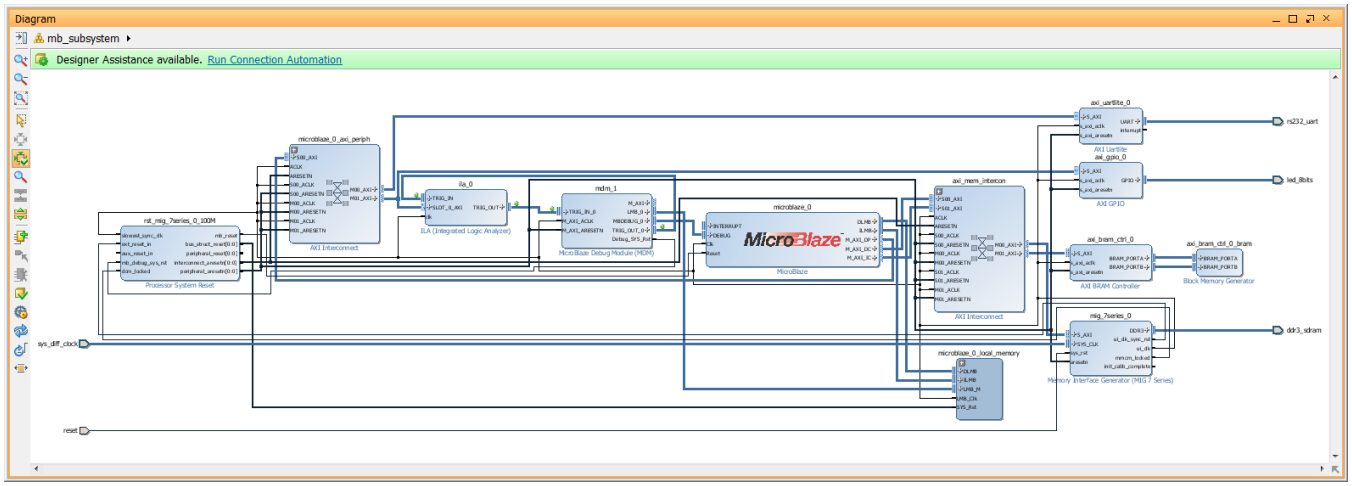

Figure 118: Block Diagram After Regenerating the Layout

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 by clicking on the Expand All icon in the toolbar to the left of the Address Editor window.

   b. Change the range of **mig_7series_0** IP in both the Data and the Instruction section to **512 MB**, as shown in Figure 119.

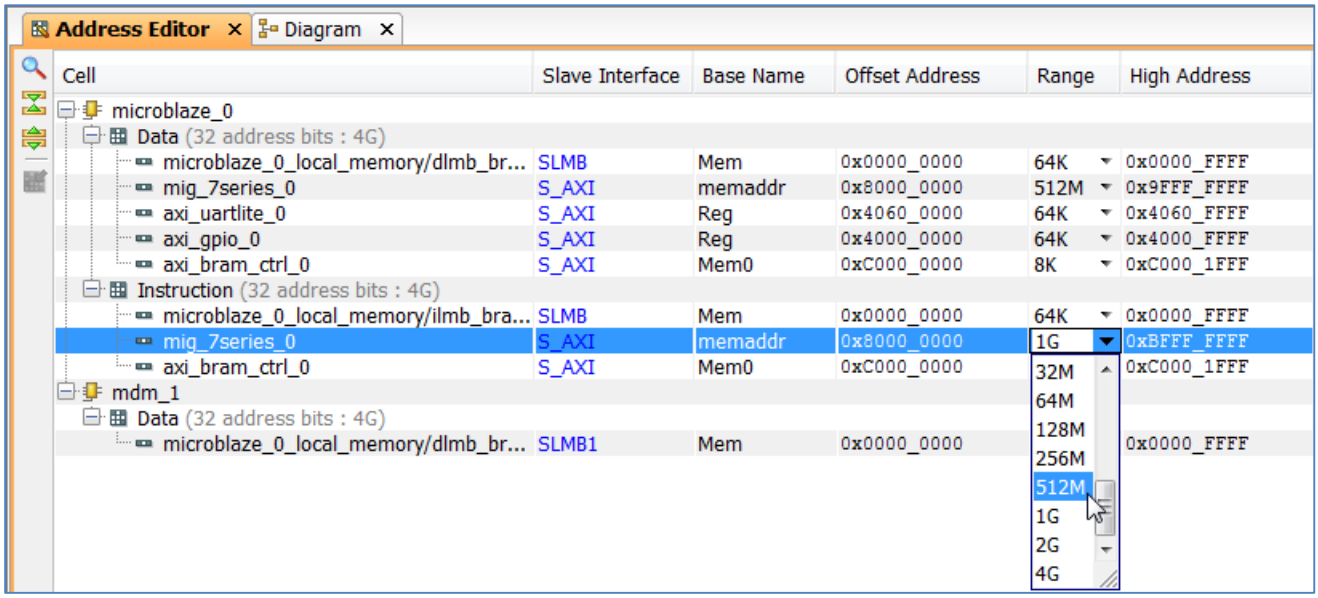

Figure 119: Data and Instruction Set to 512 MB

You must also ensure that the memory in which you are going to run and store your software is within the cacheable address range. This occurs when you enable Instruction Cache and Data Cache, while running the Block Automation for 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. The following step demonstrates how to set the cacheable region.

2. Double click on the MicroBlaze in the block design to re-configure it. Go to the Cache page (page 3) of the **Re-customize IP** dialog box, as shown in Figure 120. On this page, for both the **Instruction Cache** and **Data Cache**:
   a. The **Size in Bytes** option should be set to **32 kB**. Leave it set to this value.
   b. Set the **Line length** option to **8**.
3. Set the **Base Address** to **0x80000000** by clicking on the **Auto** button so that it changes to **Manual**, which enables the Base Address field.
4. Set the **High Address** to **0xFFFFFFFF** by clicking on the **Auto** button so that it changes to **Manual**, which enables the High Address field.
5. Enable **Use Cache for All Memory Accesses** for both caches by clicking the Auto button first to change it to **Manual**, and then checking the check box.
6. Next, verify that the size of the cacheable segment of memory (that is, the memory space between the **Base** and **High** addresses of the **Instruction Cache** and **Data Cache**) is a power of **2**, which it should be if the options were set as specified. Additionally, ensure that the Base address and the High address of both Data Cache and Instruction Cache are the same.
7. Ensure that all IP that are slaves of the Instruction Cache, and that the Data Cache buses fall within this cacheable segment. Otherwise, the MicroBlaze processor cannot access those IP.

   Note: *For any IP connected only to the Instruction Cache and Data Cache bus, you must enable the* ***Use Cache for All Memory Access*** *option. In this example, the Instruction Cache and Data Cache buses are the sole masters of DDR and block RAM; therefore, you must enable this option. In other configurations, you must decide whether to enable this option per the design requirements.*

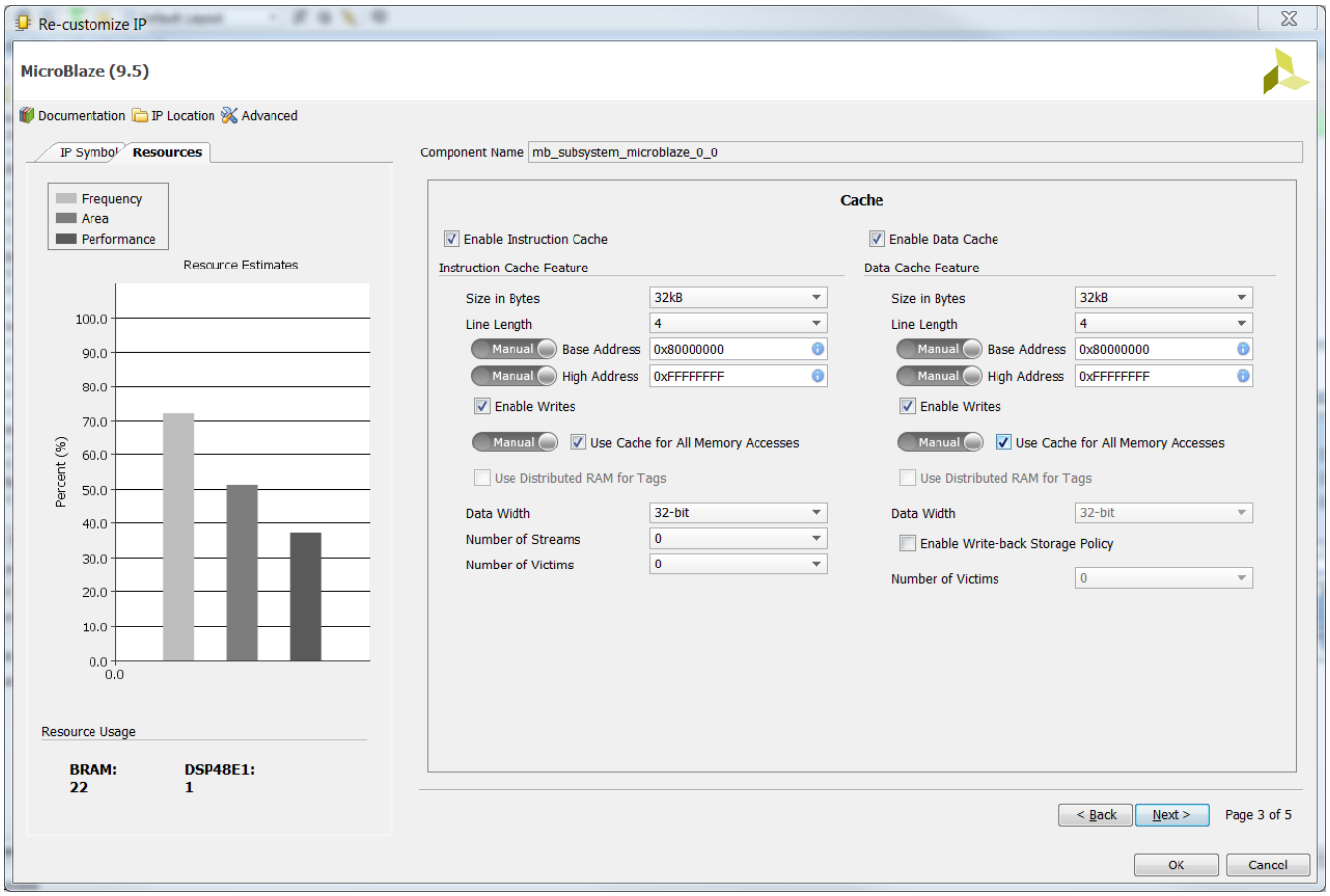

Figure 120: Setting Cache option for MicroBlaze

8. Click **OK.**

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**.
2. The Validate Design dialog box informs you that there are no critical warnings or errors in the design. Click **OK**.
3. 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 opens.

2. Click **Generate**.

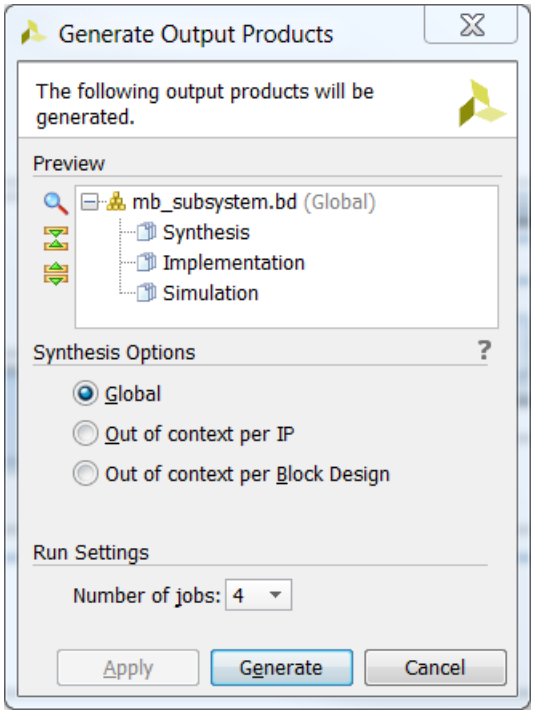

Figure 121: Generate Output Dialog Box

3. Click **OK** in the Generate Output Products dialog box.

Step 6: Create a Top-Level Verilog Wrapper

1. Under Design Sources, right-click your design and click **Create HDL Wrapper**.
2. In the Create HDL Wrapper dialog box, **Let Vivado manage wrapper and auto-update** is selected by default.
3. Click **OK**.

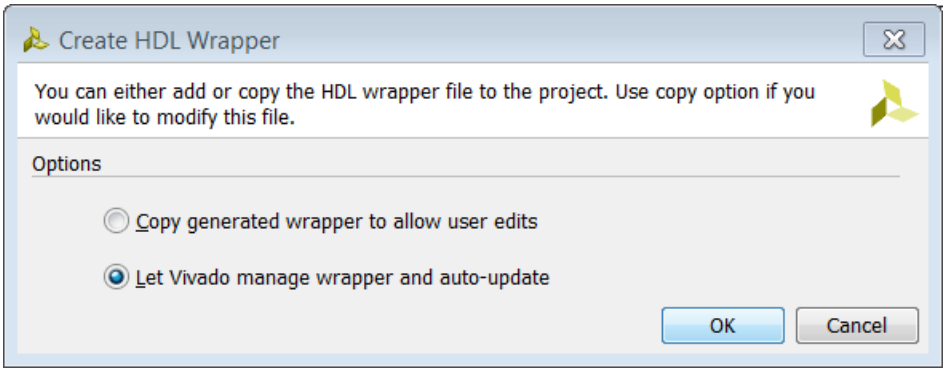

Figure 122: Creating an HDL Wrapper

Step 7: Synthesize the design

1. From the Flow Navigator, click on **Run Synthesis**.
2. After synthesis finishes, open the synthesized design by selecting the **Open Synthesized Design** option in the Synthesis Completed dialog box.

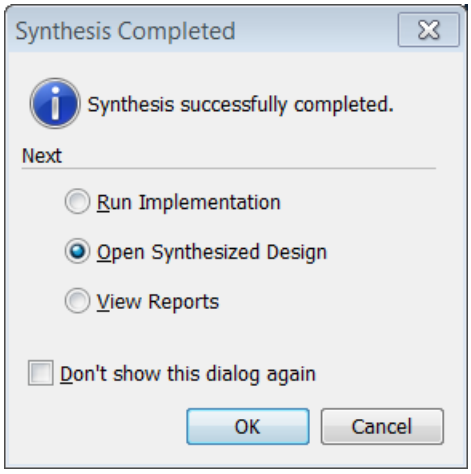

Figure 123: Synthesis Completed dialog box

3. Click **OK**.

Step 8: Insert an Integrated Logic Analyzer (ILA) 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. You will insert another ILA in the design to monitor the activity on cross-trigger signals.

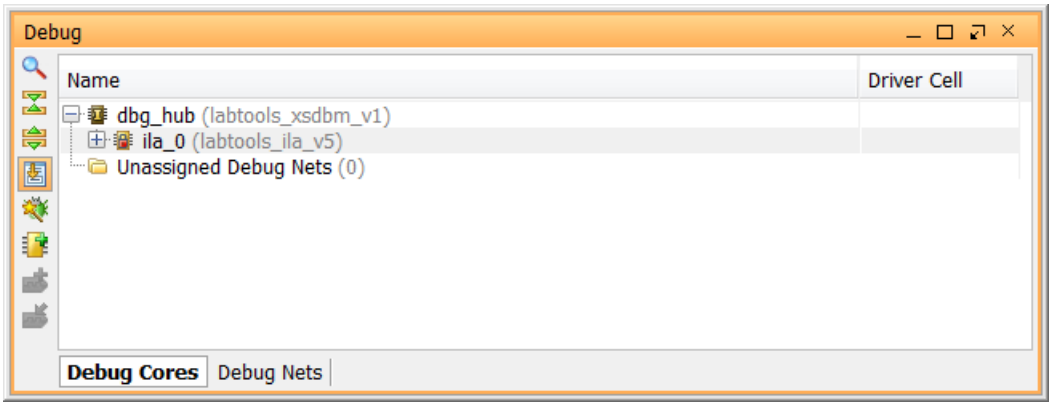

Figure 124: Unassigned Debug Nets

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, shown in Figure 125.

2. Click **Next**.

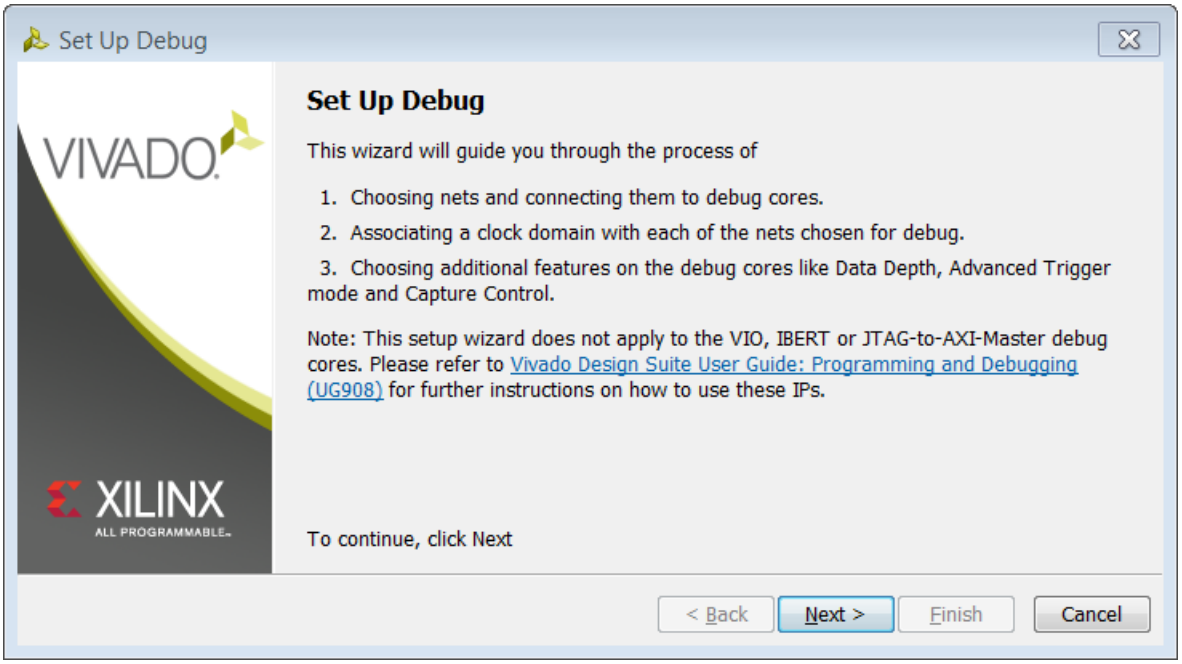

Figure 125: Set Up Debug Dialog Box

3. In the Nets to Debug page, click the **Find Nets to Add** button at the bottom of the dialog box.

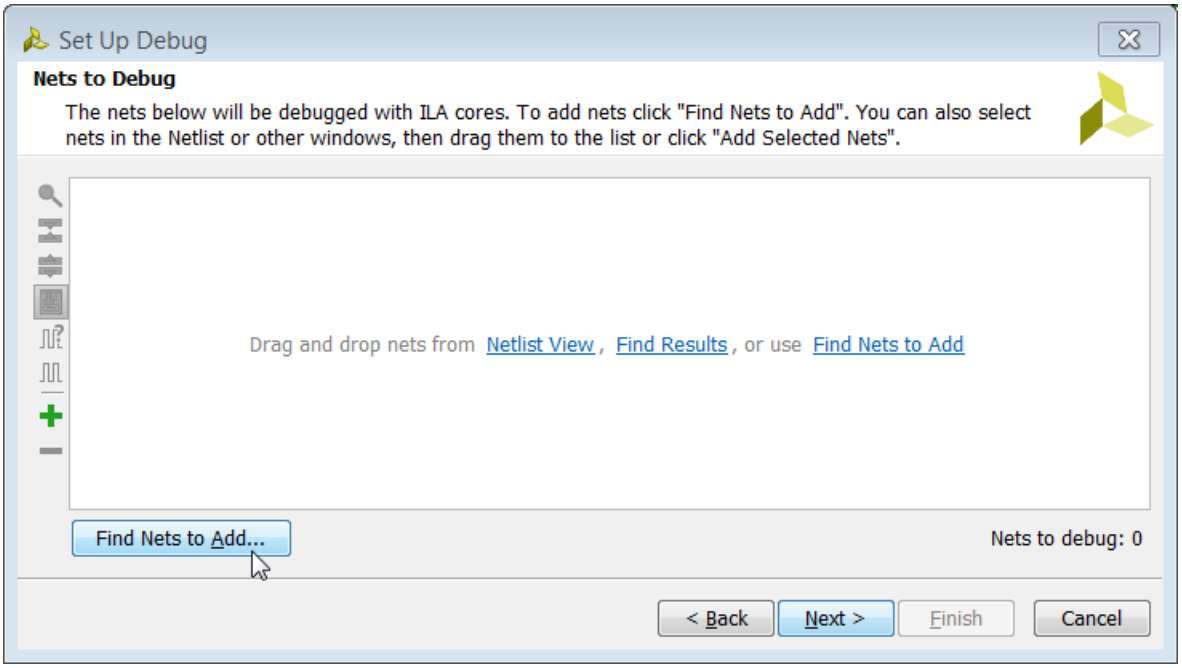

Figure 126: Add Nets to Debug

4. In the find Nets dialog box (Figure 127), change the Properties to select **MARK_DEBUG** from the first drop-down list. This will show all the nets that have the `MARK_DEBUG` attribute set.

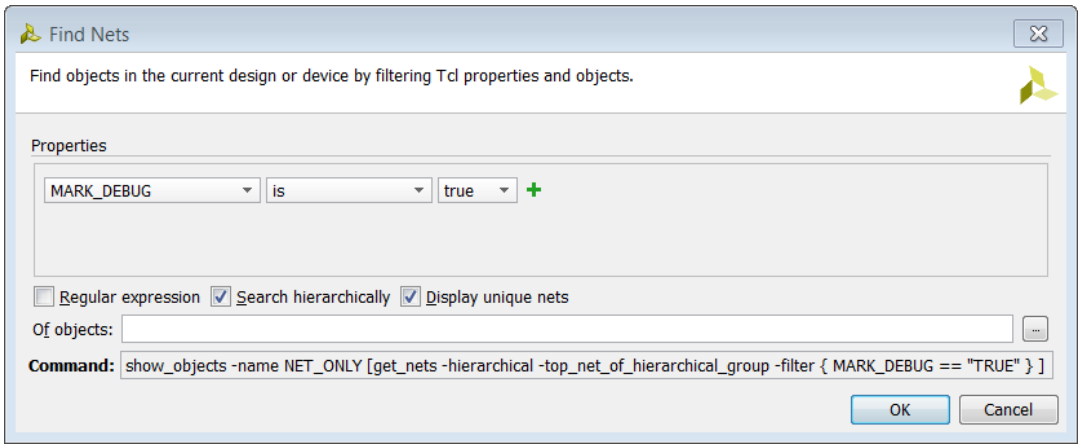

Figure 127: Find Nets with MARK_DEBUG Attribute

5. Click **OK**. The Add Nets to Debug dialog box opens.
6. Select the following nets, as shown in Figure 128:
   - mb_subsystem_i/ila_0_TRIG_OUT_ACK

- mb_subsystem_i /ila_0_TRIG_OUT_TRIG
- mb_subsystem_i /mdm_1_TRIG_OUT_0_ACK
- mb_subsystem_i /mdm_1_TRIG_OUT_0_TRIG

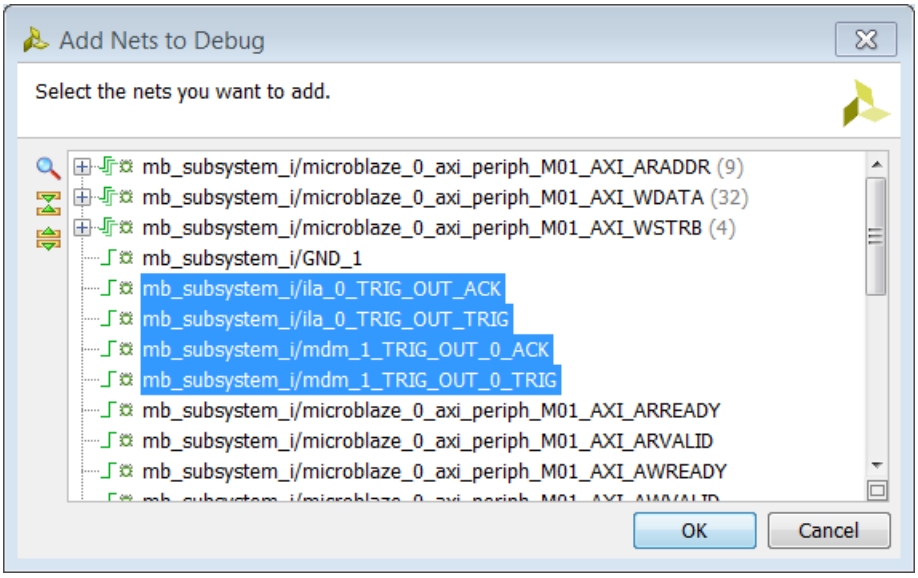

Figure 128: Add Nets Connected to Cross-Trigger Pins

7. Click **OK.**

   The additional cross-trigger nets are added to a new ILA in the Set Up Debug dialog box.

8. In the Set Up Debug dialog box select all the newly added nets, right-click and select **Select Clock Domain**.

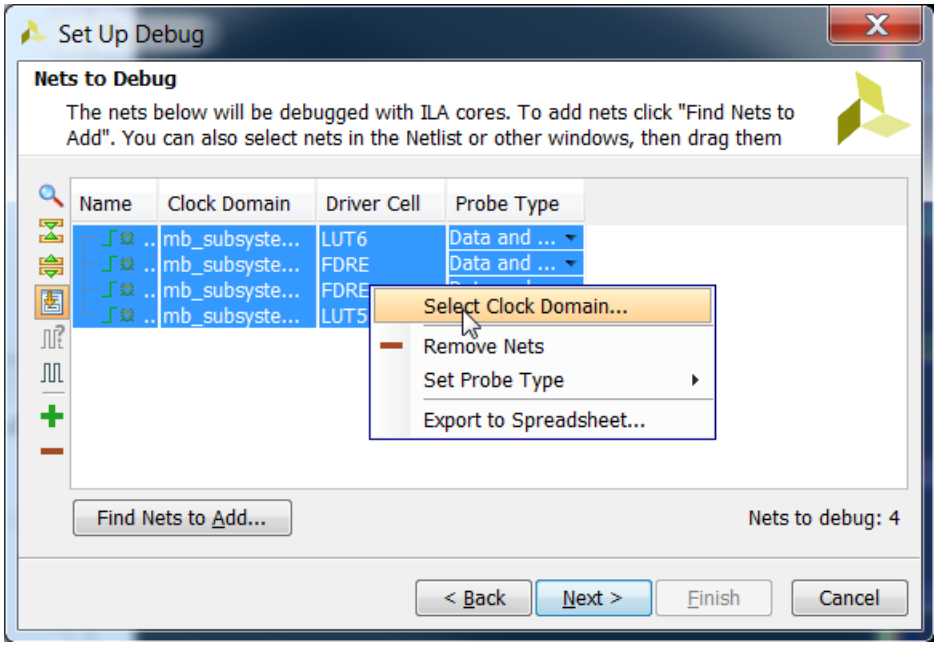

Figure 129: Select Clock Domain for the newly added nets

9. Select the `mb_subsystem_i/mig_7series_0/u_mb_subsystem_mig_7series_0_0_mig/u_ddr3_infrastructure/rstdiv0_sync_r1_reg_rep__28_0` clock as the Clock Domain for all the cross-trigger signals.

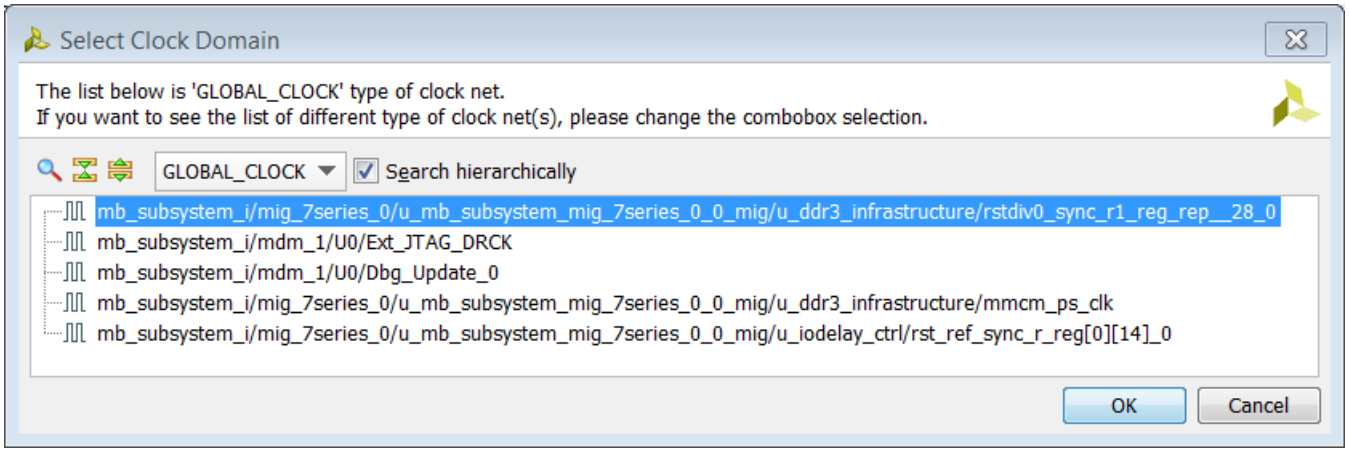

Figure 130: Select the appropriate clock for the newly added cross-trigger signals

10. Click **OK**.
11. Click **Next** on the Nets to Debug page of the Set Up Debug wizard.

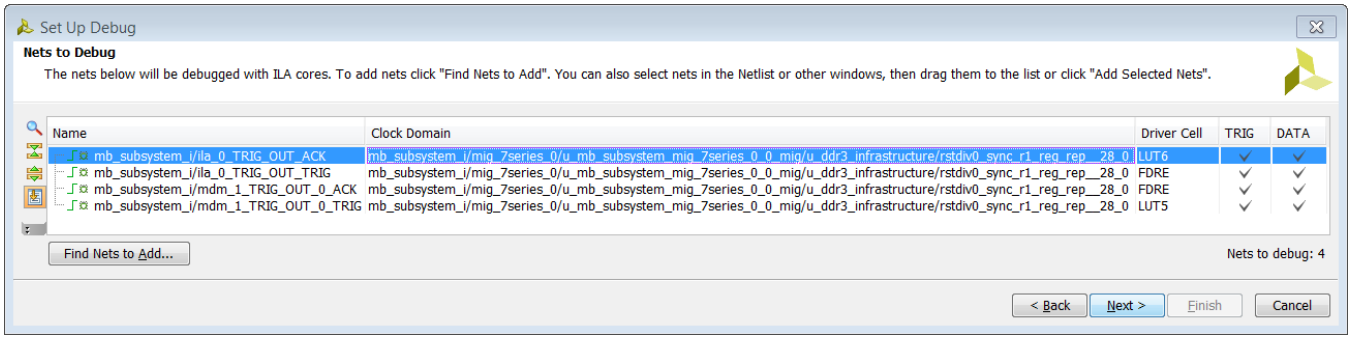

Figure 131: Specify Nets for Debugging

12. The ILA (Integrated Logic Analyzer) Core Options dialog box opens, as shown in Figure 132.

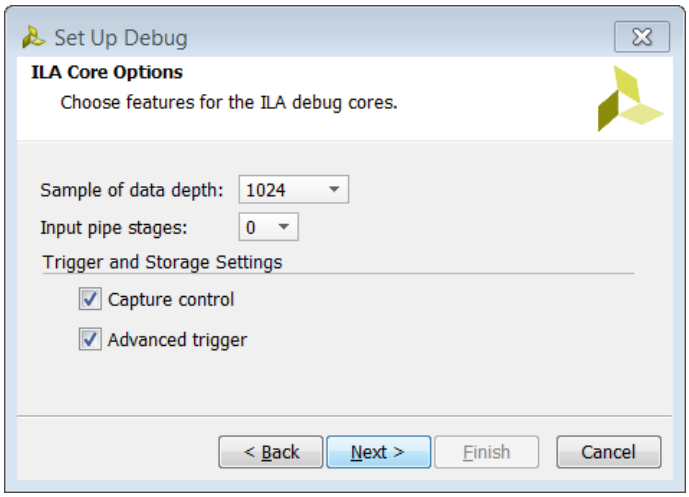

Figure 132: Enable Advanced Trigger and Capture Modes

In the ILA (Integrated Logic Analyzer) Core 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**.

13. In the Set up Debug Summary page verify that the debug core was added and click **Finish**.

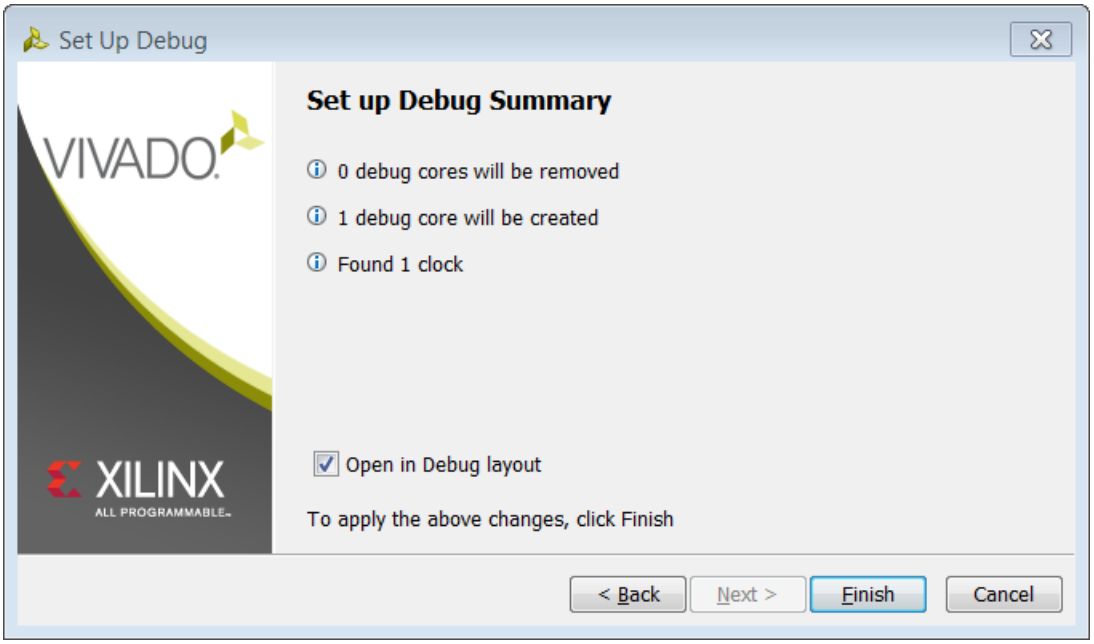

Figure 133: Set up Debug Summary page

The debug nets are now assigned to the `ILA u_ila_0` debug core, as shown in Figure 134. Notice that a new ILA called u_ila_0 has been added.

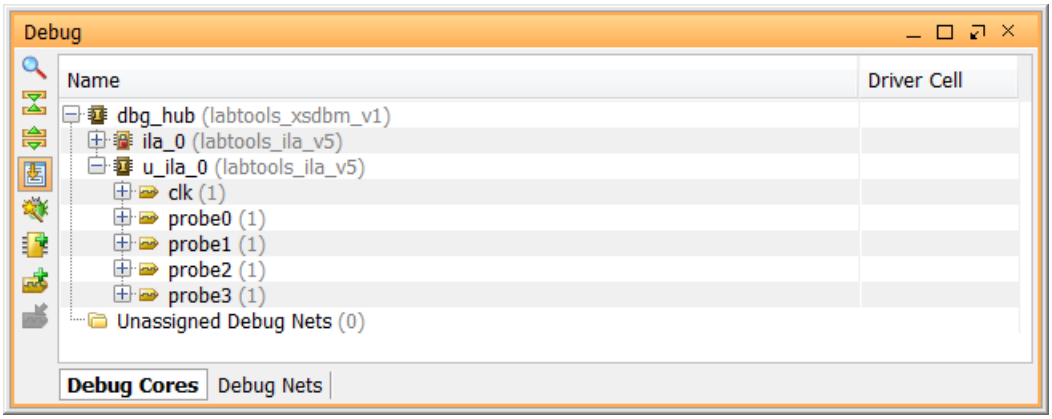

Figure 134: Debug Nets Assigned to Debug Core

Step 9: Take the Design through Implementation

In the Flow Navigator:

1. Click **Generate Bitstream**.

The Save Project dialog box opens.

2. Click **Save**.
3. The Out of Date Design dialog box opens informing that synthesis may go out-of-date as the constraints related to ILA were saved.
4. Click **OK**.
5. The No Implementation Results Available dialog box asks if synthesis and implementation can be launched before generating bitstream.
6. Click **Yes**.

   Bitstream generation can take several minutes to complete. Once it finishes, the Bitstream Generation Completed dialog box asks you to select what to do next.
7. Keep the default selection of **Open Implemented Design** and click **OK**.
8. The Vivado dialog box opens and asks you if you want to close the Synthesized Design before opening the Implemented Design.
9. Click **Yes**.
10. Verify that all timing constraints have been met by looking at the Timing - Timing Summary window, as shown in Figure 135.

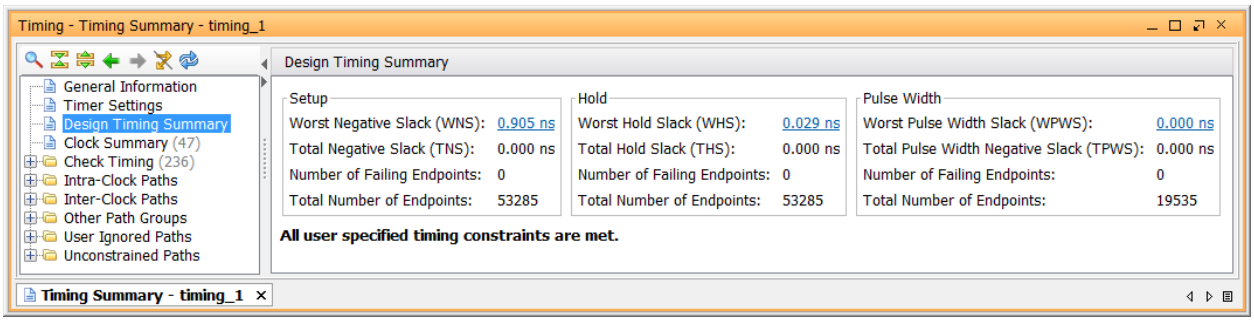

Figure 135: Timing Summary

Step 10: 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, shown in Figure 136. Make sure that the **Export to** field is set to **<Local to Project>**.

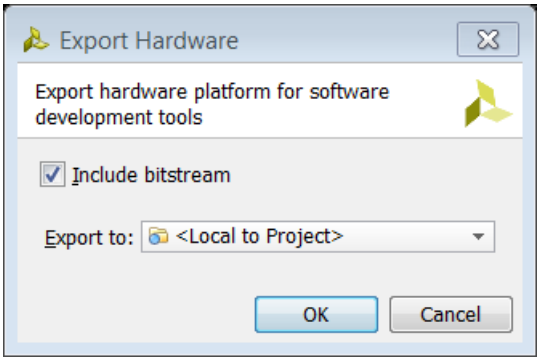

Figure 136: Export to Hardware Dialog Box

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>**.

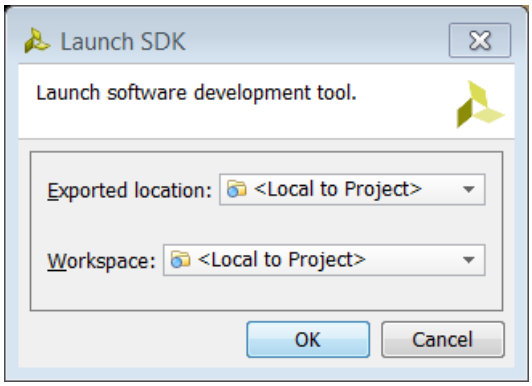

Figure 137: Launch SDK Dialog Box

5. Click **OK**.

   SDK launches in a separate window.

Step 11: Create a "Peripheral Test" Application

1. In SDK, right-click **mb_subsystem_wrapper_hw_platform_0** in the Project Explorer and select **New > Project**, as shown in Figure 138.

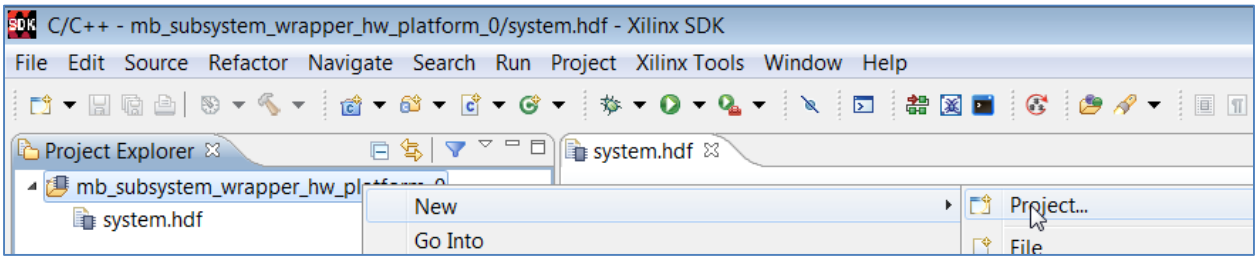

Figure 138: SDK New Project Selection

2. In the New Project dialog box, select **Xilinx Application Project**.

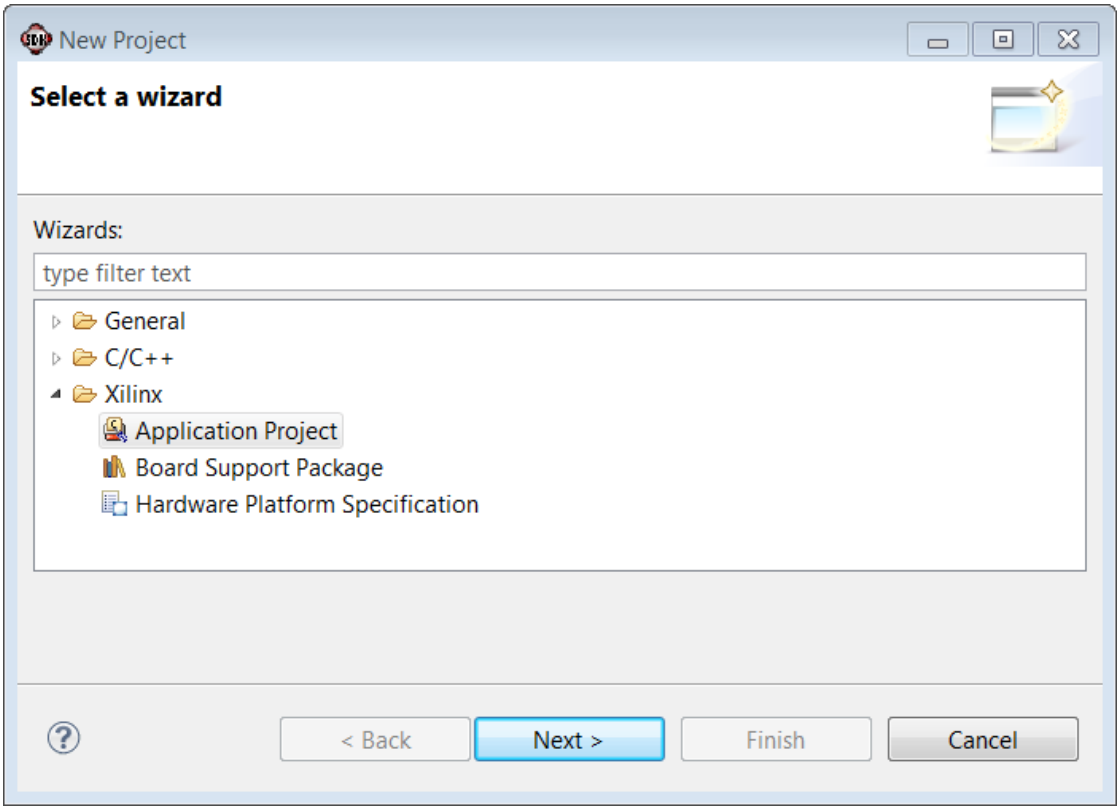

Figure 139: SDK New Project Wizard

3. Click **Next**.
4. Type a name (such as `peri_test`) for your project and choose standalone as the OS platform, as displayed below in Figure 140.

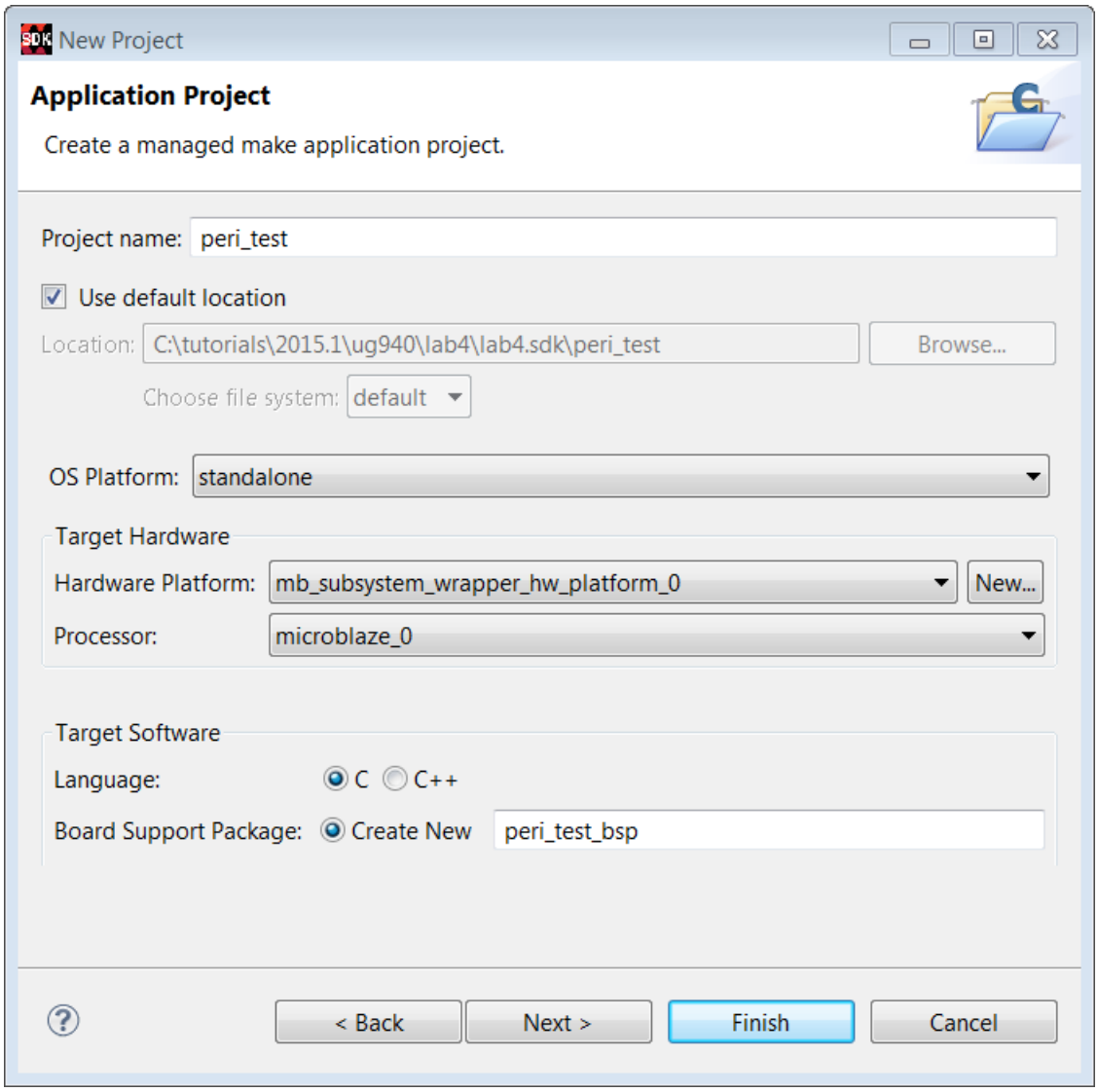

Figure 140: New Project: Application Project Wizard

5. Click **Next**.

6. Select the `Peripheral Tests` application template, and click **Finish**.

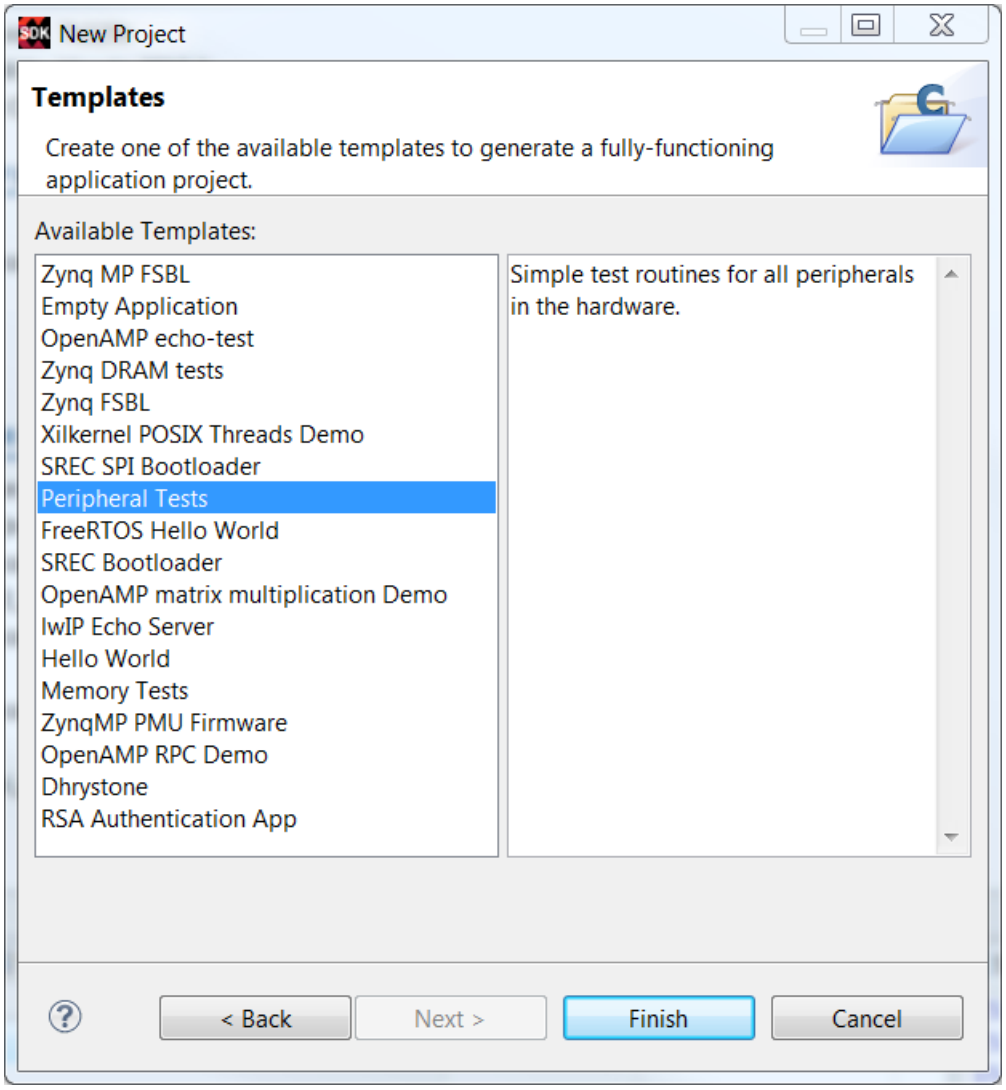

Figure 141: New Project: Template Wizard

SDK creates a new "peri_test" application.

7. Right-click the `peri_test` 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. In the Basic tab, change the **Place Code Sections in** to mig_7_series_0 using the drop-down list.
   b. Likewise, change the **Place Data Sections in** and **Place Heap and Stack in** to mig_7series_0.

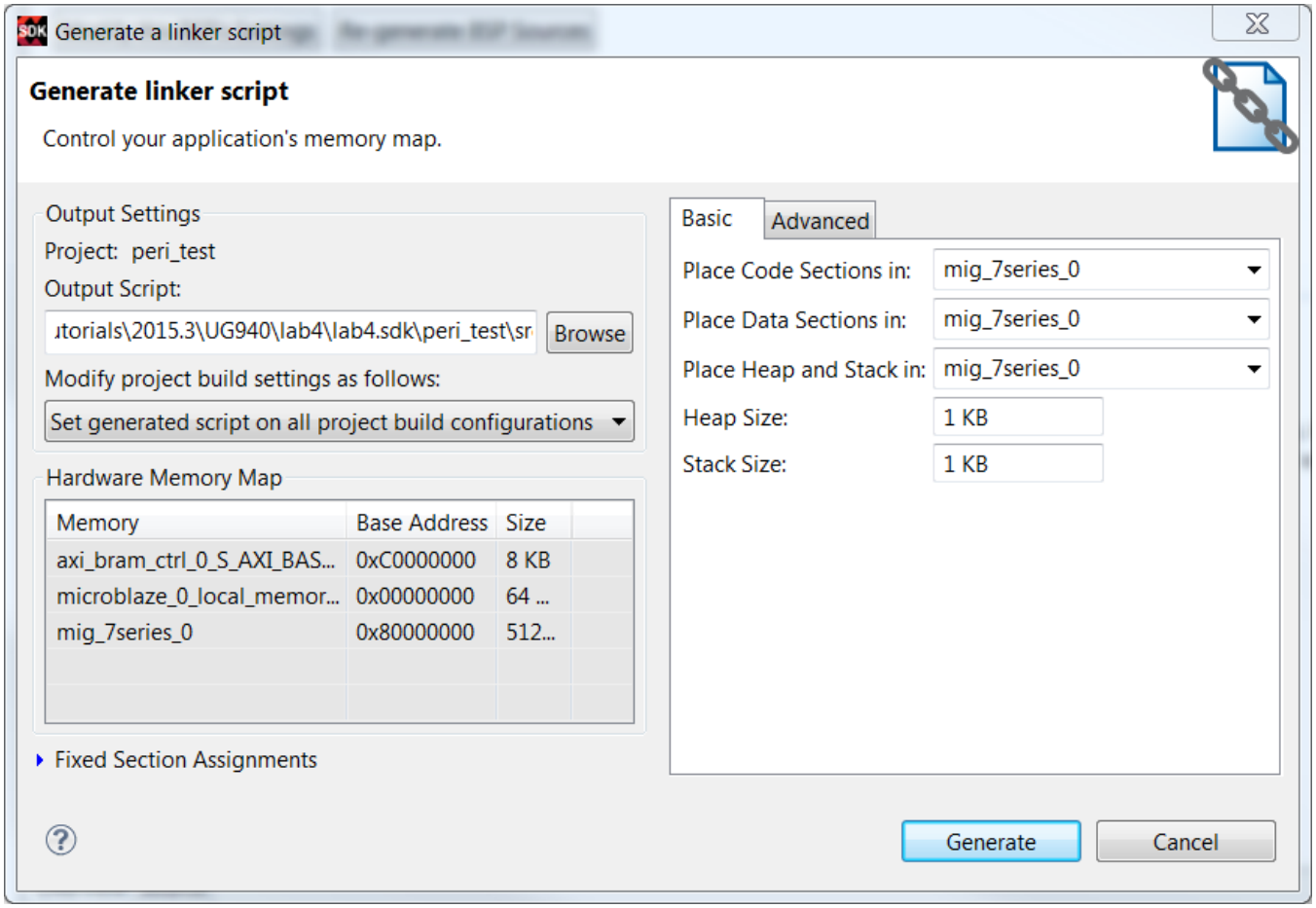

Figure 142: Basic Tab of the Generate a linker script dialog box

   c. The Advanced options all change to mig_7_series_0 as shown below.

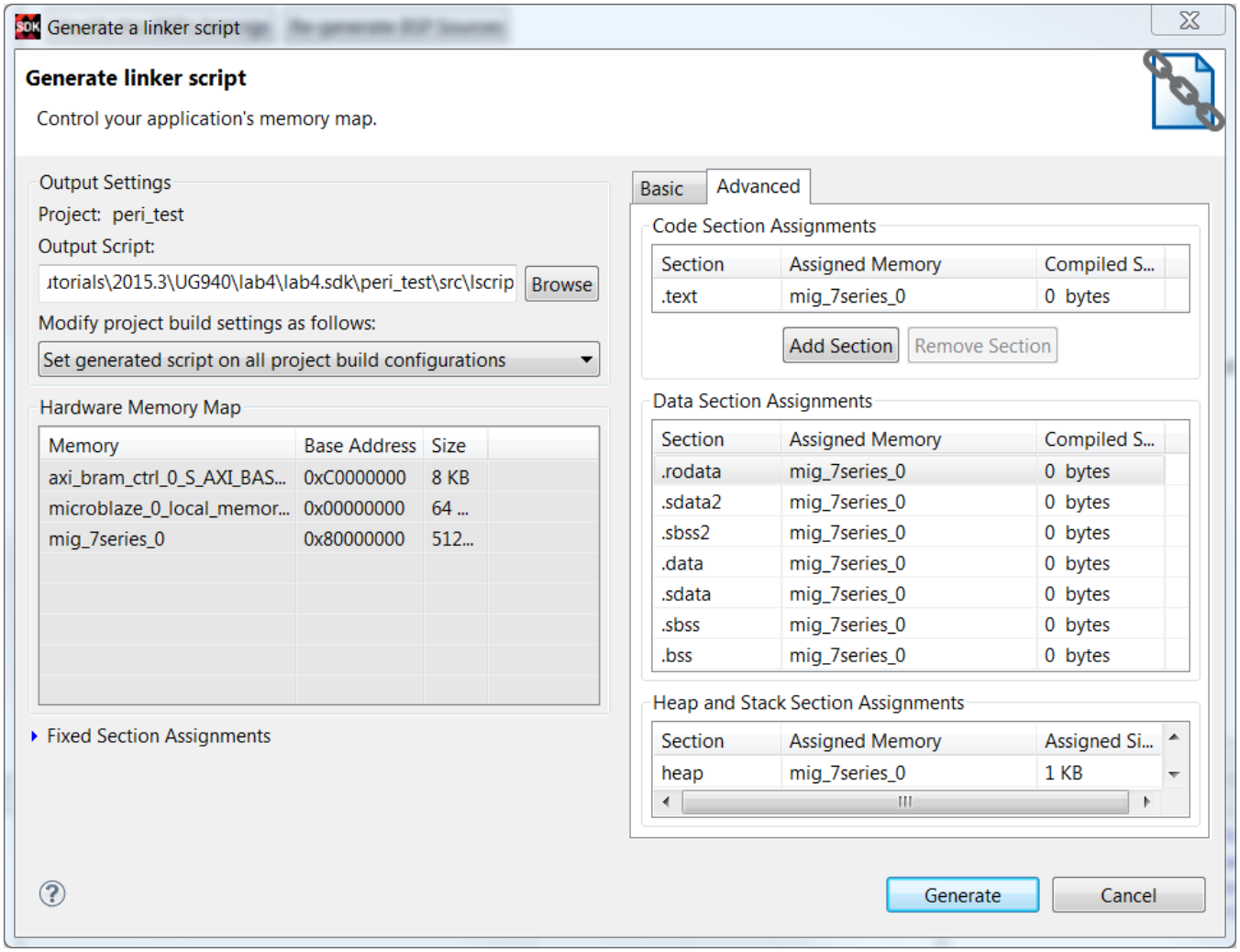

Figure 143: Generate Linker Script Dialog Box

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

9. Click **Generate**.
10. Click **Yes** to overwrite it in the Linker Script Already Exists! dialog box.

Step 12: Executing the Software Application on a KC705 Board

Make sure that you have connected the target board to the host computer and it is turned on.

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

2. In the Program FPGA dialog box, click **Program,** as shown in Figure 144.

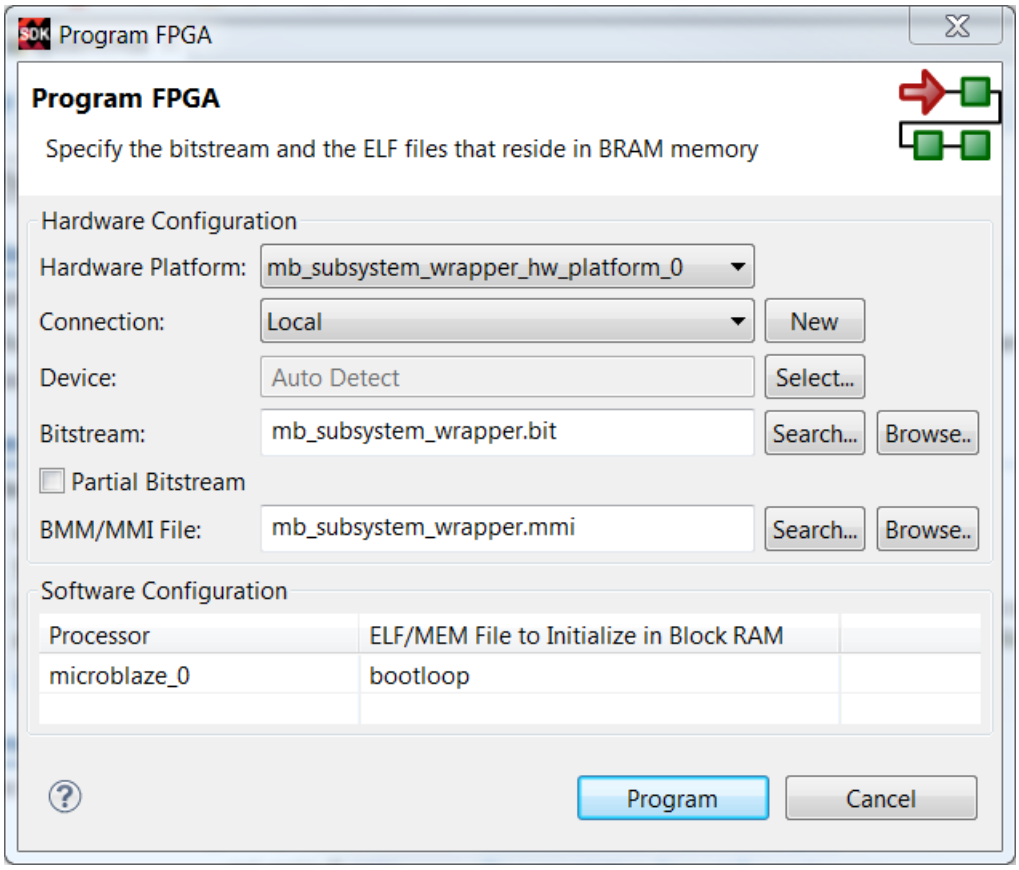

Figure 144: Program FPGA Dialog Box

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

   The Debug Configurations dialog box opens.

4. Right-click **Xilinx C/C++ application (System Debugger**), and select **New.**

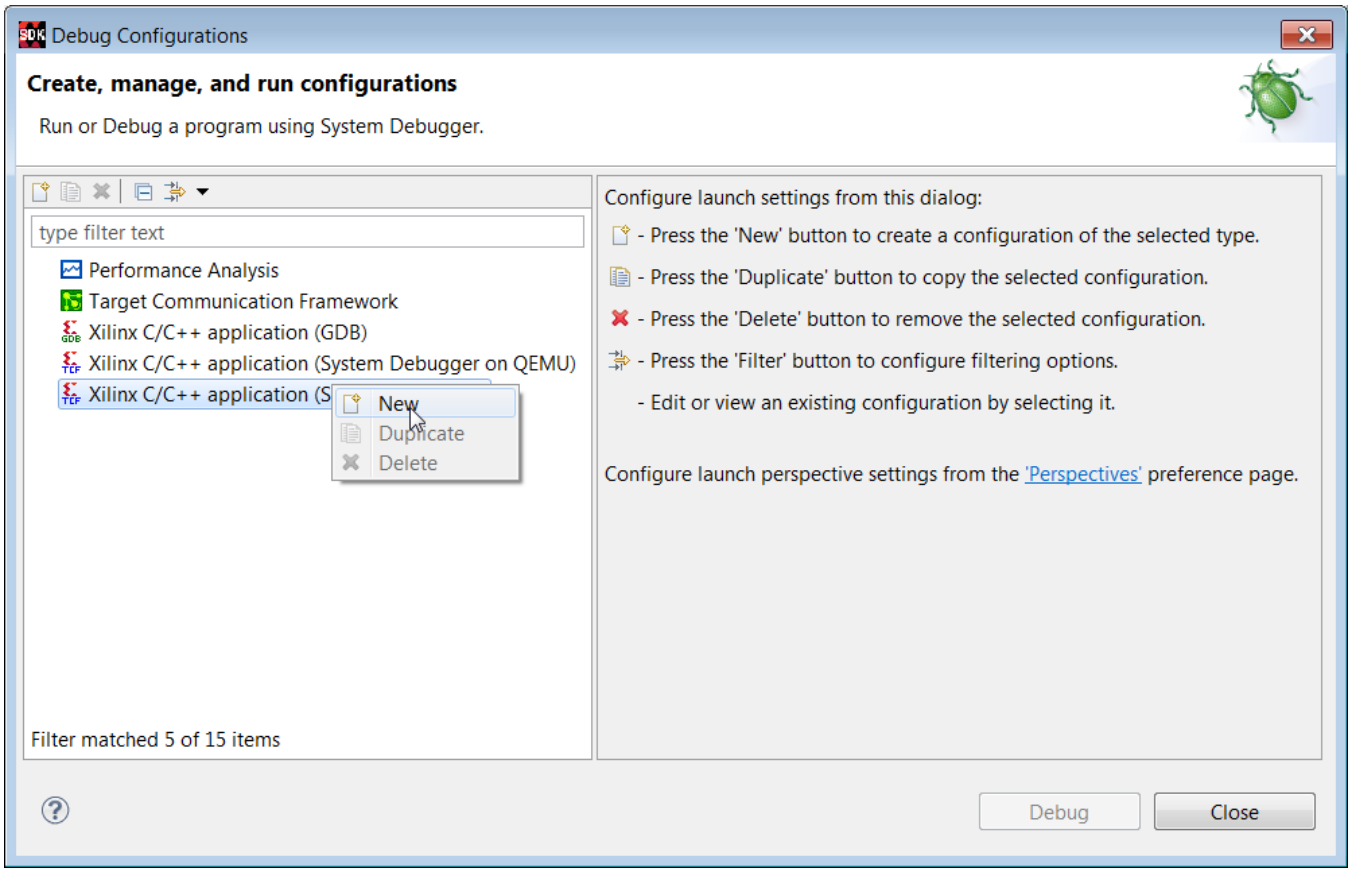

Figure 145: Create New Debug Configuration

5. Click **Debug**.

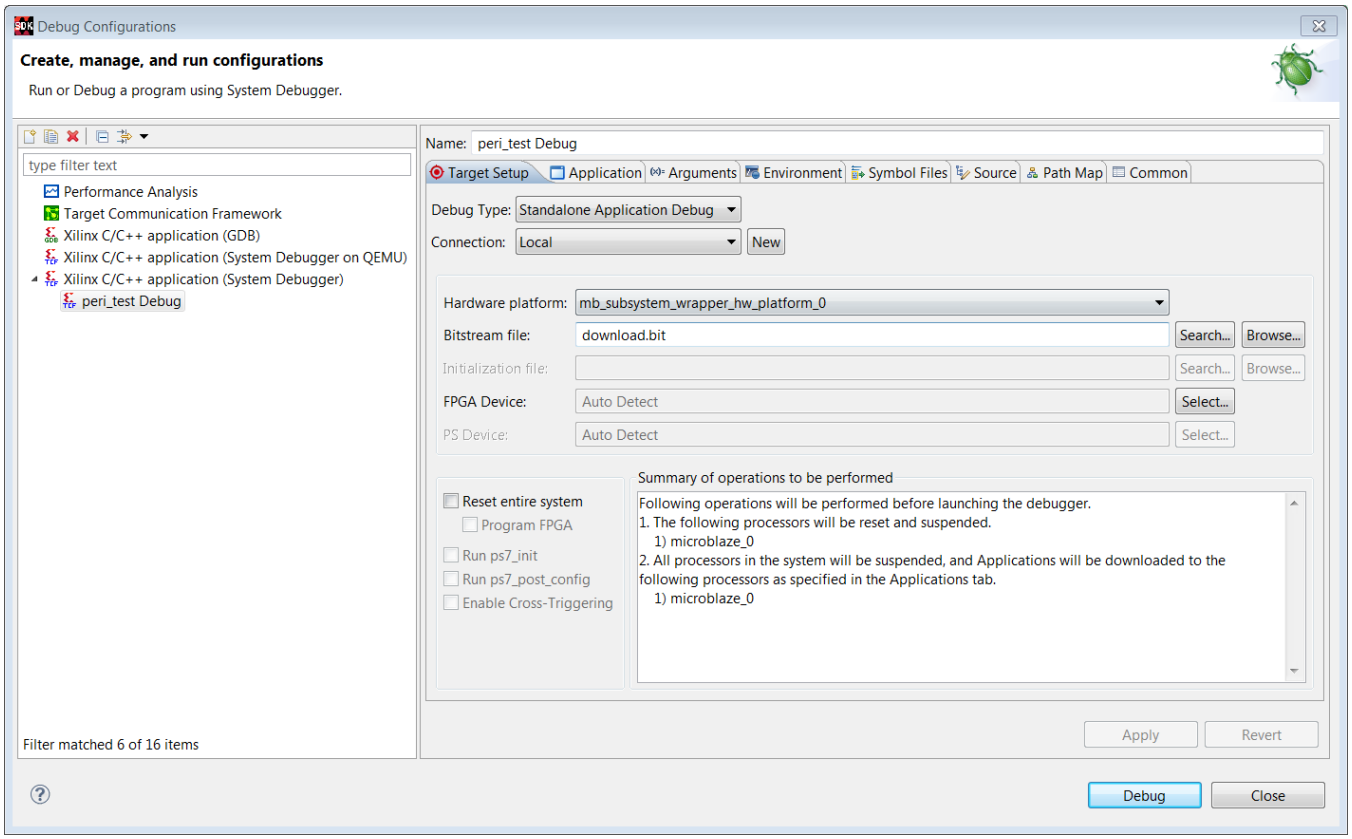

Figure 146: Debug Configurations dialog box

The Confirm Perspective Switch dialog box opens.

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

The Debug perspective window opens.

7. Set the terminal by selecting the **SDK Terminal** tab and clicking the icon.
   a. Use the settings shown in Figure 147 for the KC705 board and click **OK**.

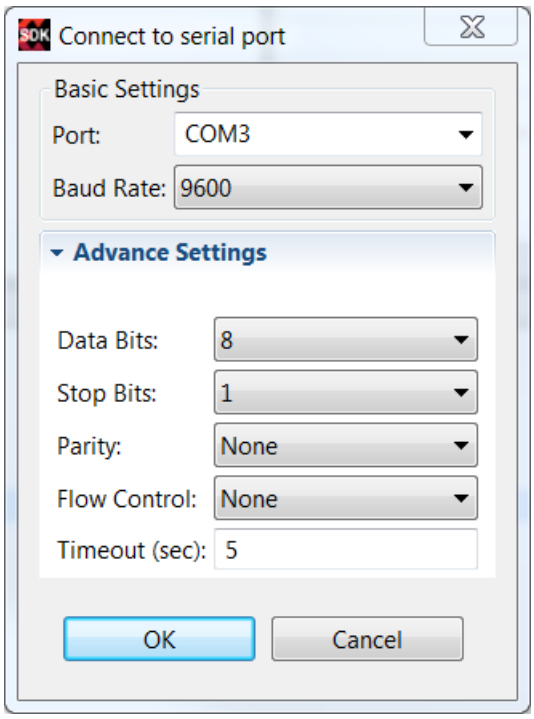

Figure 147: Terminal Settings

8. Verify the Terminal connection by checking the status at the top of the tab as shown in Figure 148.

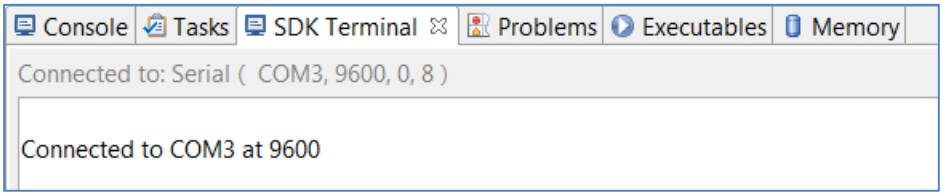

Figure 148: Verify Terminal Connection

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

10. Modify the source file by inserting a while statement at line 41.

    a. Click the blue bar on the left side of the `testperiph.c` window as shown in the figure and select **Show Line Numbers**.

    b. In line 41, add `while(1)` above in front of the curly brace as shown in Figure 149.

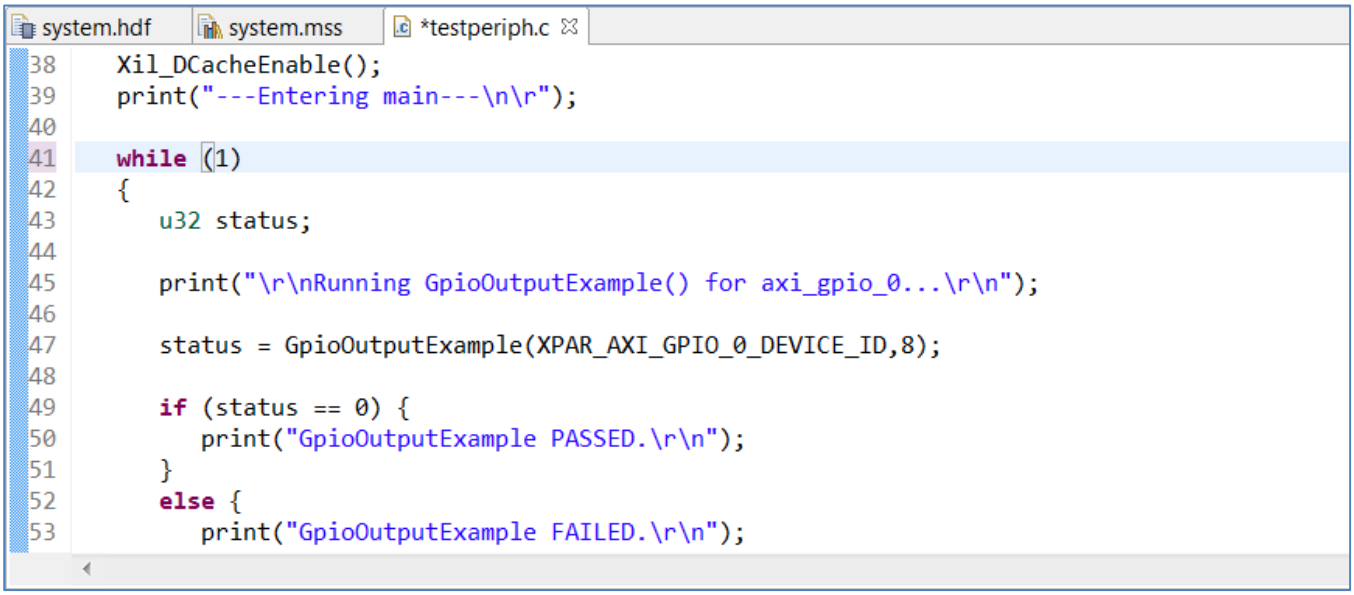

Figure 149: Modify testperiph.c

11. 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 50 and double-click on the left pane, which adds a breakpoint on that line of code, as shown in Figure 150. Click **Ctrl + S** to save the file. Alternatively, you can select **File > Save**.

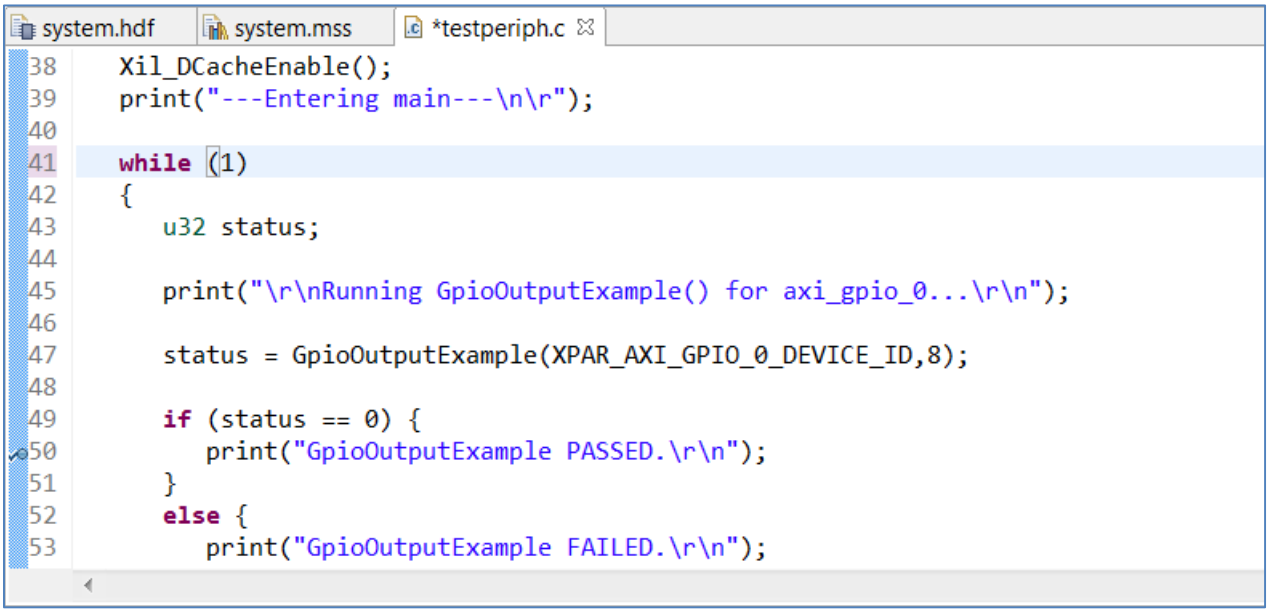

Figure 150: Set a Breakpoint

Now you are ready to execute the code from SDK.

Step 13: Connect to Vivado Logic Analyzer

Connect to the KC705 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 target > Open New Target**.

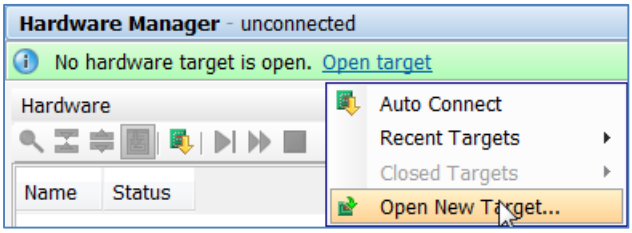

Figure 151: Open a New Hardware Target

Note: *You can also use the Auto Connect option to connect to the target hardware.*

The Open New Hardware Target dialog box opens.

3. Click **Next**.

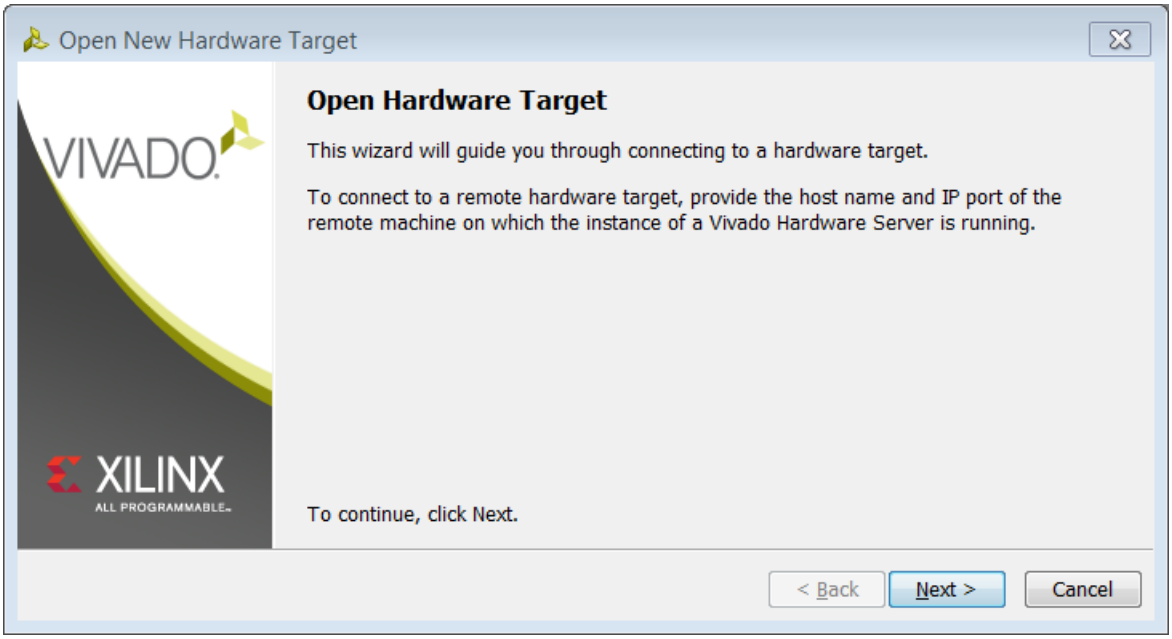

Figure 152: Open Hardware Target

On the Hardware Server Settings page, ensure that the Connect to field is set to **Local server (target is on local machine)**, as shown in Figure 153.

4. Click **Next**.

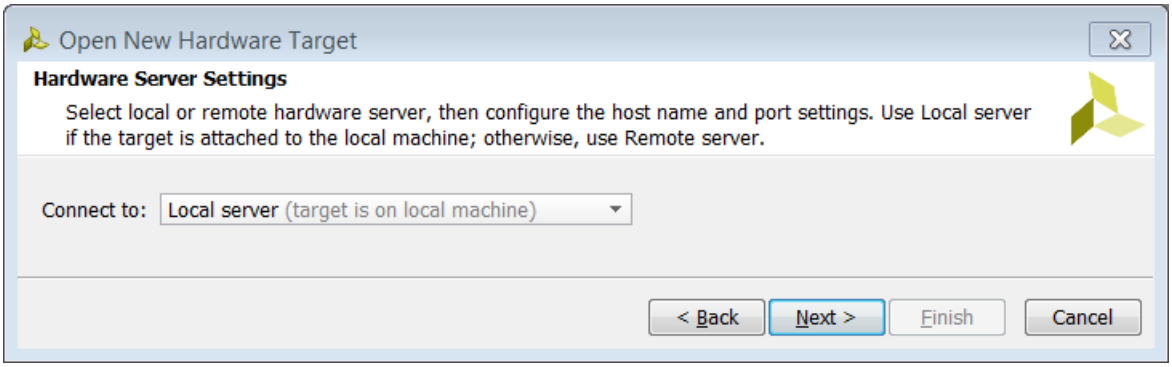

Figure 153: Specify Server Name

5. On the Select Hardware Target page, shown below, click **Next**.

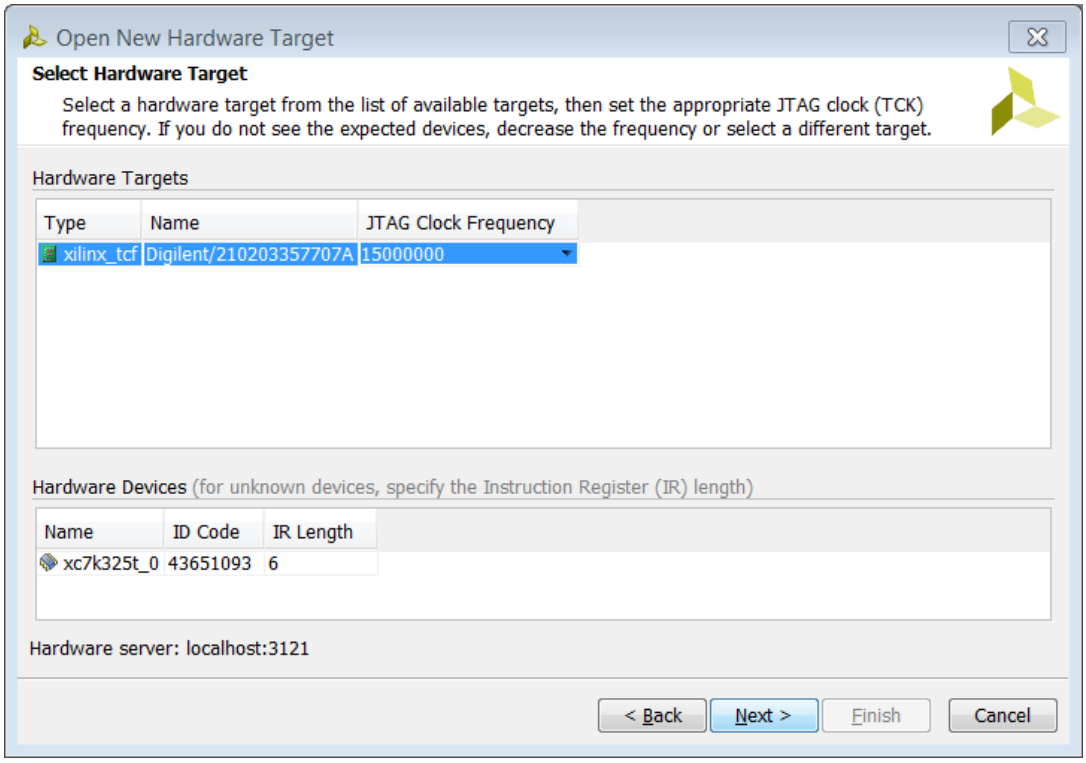

Figure 154: Select Hardware Target

6. Ensure that all the settings are correct on the Open Hardware Target Summary dialog box and click **Finish**.

Figure 155: Open Hardware Target Summary

Step 14: Setting the MicroBlaze to Logic Cross Trigger

When the Vivado Hardware Session successfully connects to the ZC702 board, you see the information shown in Figure 156.

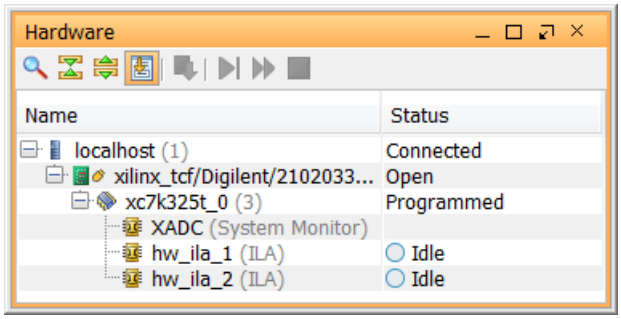

Figure 156: Vivado Hardware Window

1. 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**.

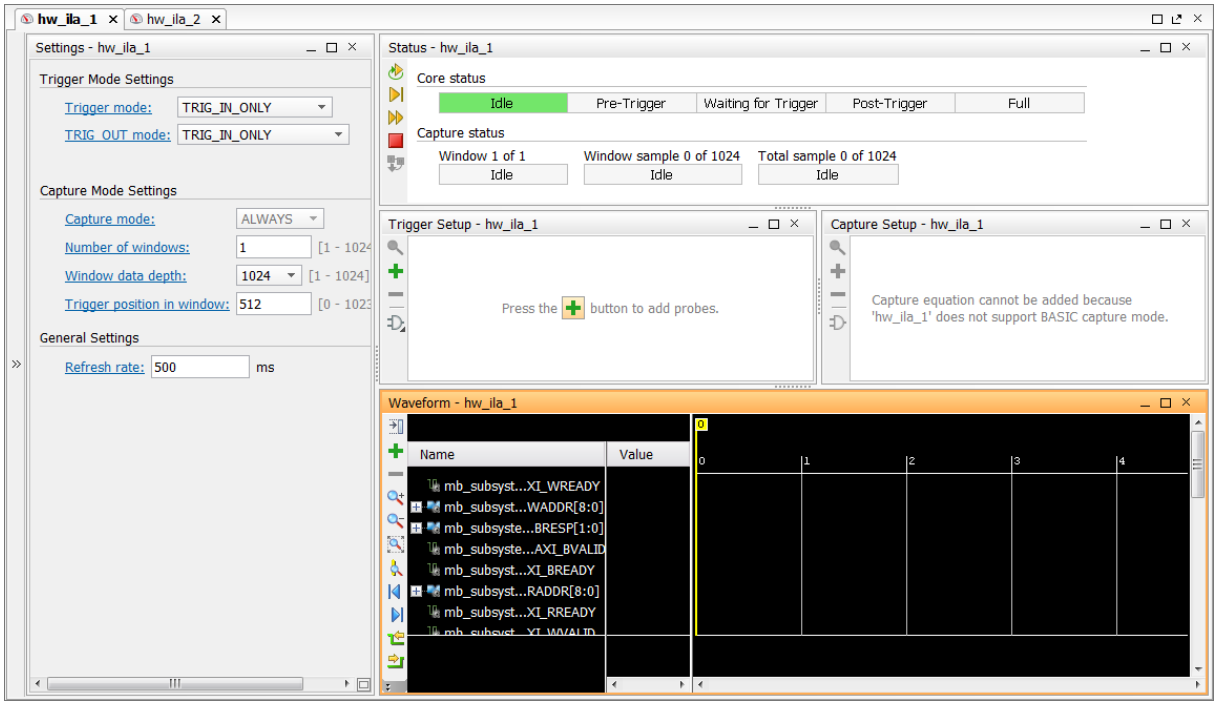

Figure 157: Set ILA Properties for hw_ila_1

2. 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 158.

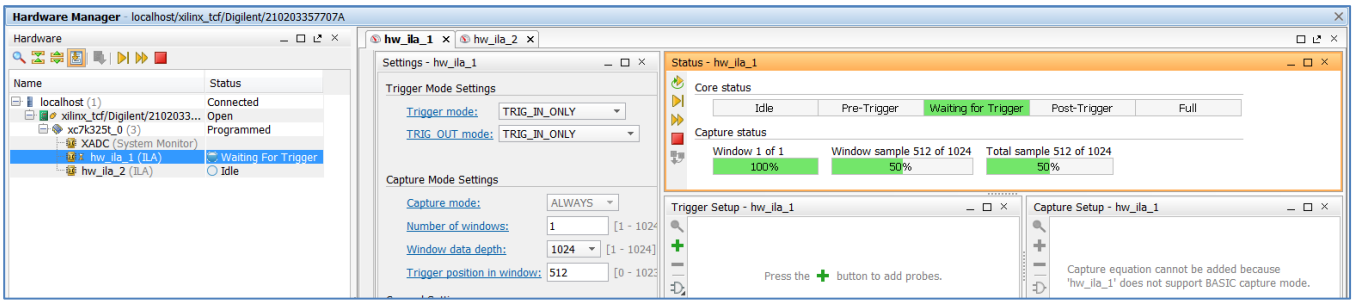

Figure 158: Armed ILA Core

3. Select the hw_ila_2 tab to set up trigger conditions to capture the trace, 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**.
   - With the hw_ila_2 tab selected, click on the + sign in the Trigger Setup window.

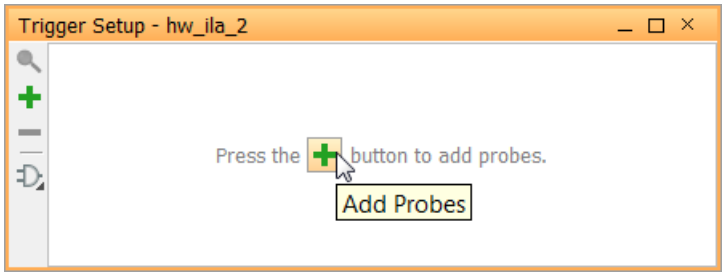

Figure 159: Add Probes to setup trigger

- Select the `mb_subsystem_i/mdm_1_TRIG_OUT_0_TRIG` signal from the Debug Probes window under hw_ila_2 into the **Trigger Setup** window.

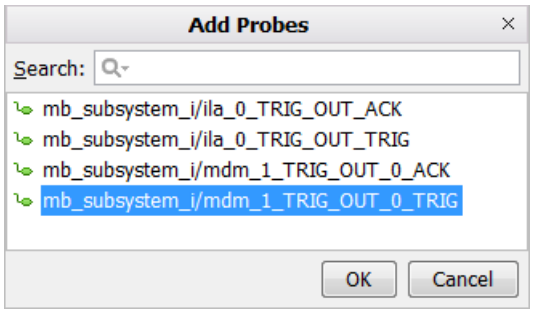

Figure 160: Add Probes window

4. Click **OK**.
5. In the Trigger Setup window, click the **Compare Value** field for the `mb_subsystem/mdm_1_TRIG_OUT_0_TRIG` signal, and set the Radix field to B and the **Value** field to **1**. This essentially sets up the ILA to trigger when the `trig_out` transitions to a value of 1.

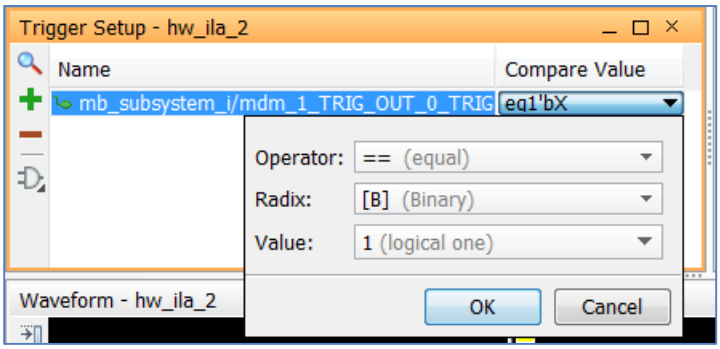

Figure 161: Set Trigger Condition

6. Click **OK**.
7. 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.
8. In SDK, in the Debug window, click the **MicroBlaze #0** in the Debug window and click the **Resume** button .

   Vivado displays the hw_ila_2 trigger as shown in Figure 162.

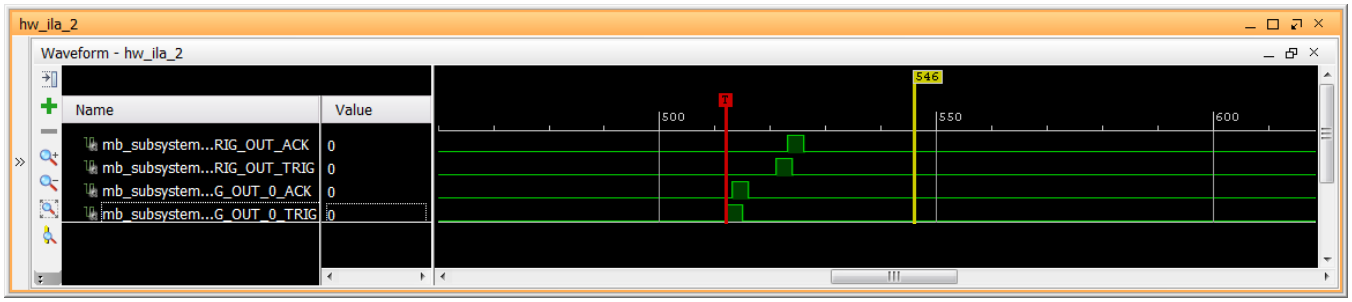

Figure 162: MicroBlaze to Logic Cross Trigger Waveform

Likewise, hw_ila_1 also triggers as seen in Figure 163.

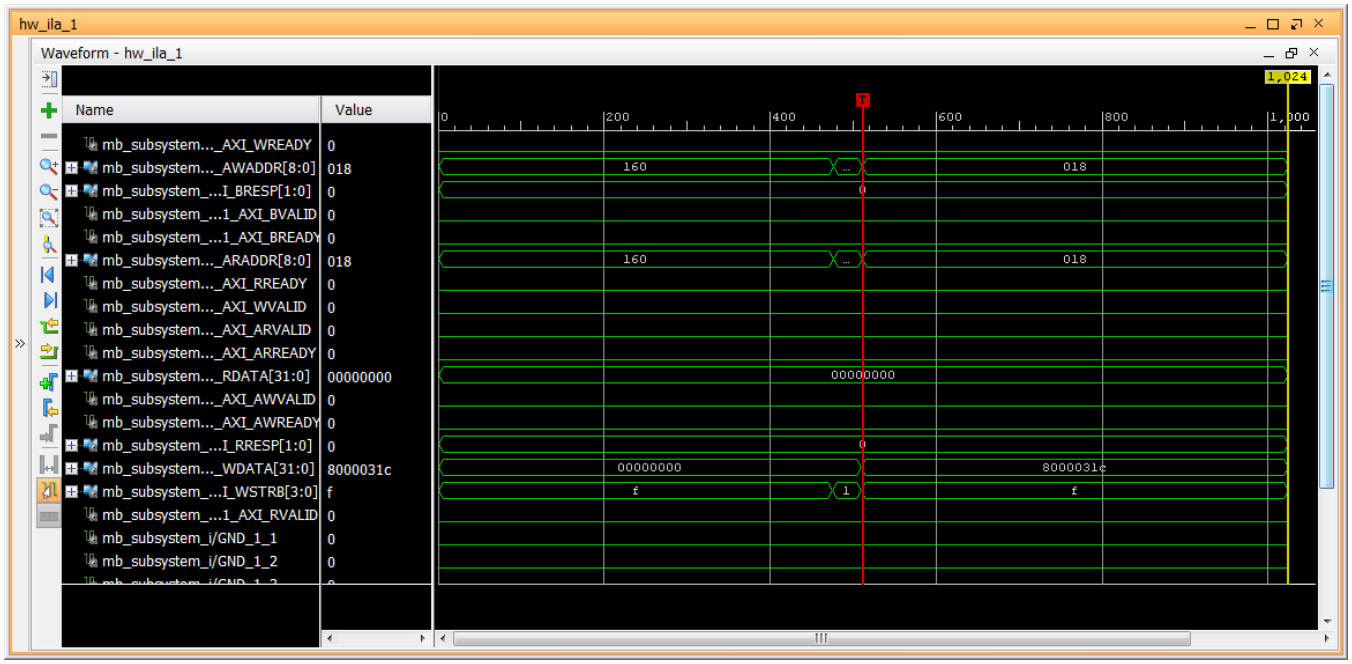

Figure 163: MicroBlaze to Logic Cross Trigger Waveform in hw_ila_1

This demonstrates that when the breakpoint is encountered during code execution, the MicroBlaze 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.

Step 15: Setting the Logic to Processor Cross-Trigger

Now try the logic to processor side of the cross-trigger mechanism. In other words, remove the breakpoint that you set earlier on line 50 to have the ILA trigger the processor and stop code execution. To do this:

1. Select the Breakpoints tab towards the top right corner of SDK window, and uncheck the testperiph.c [line: 50] checkbox. This removes the breakpoint that you set up earlier.

   Note: Alternatively, you can also click on the breakpoint in the testperiph.c file and select Disable Breakpoint.

2. In the Debug window, right-click the **MicroBlaze #0 target** and select **Resume**. The code runs continuously because it has an infinite loop.

   You can see the code executing in the Terminal Window in SDK.

3. In Vivado, select the hw_ila_1 tab. Change the Trigger Mode to **BASIC_OR_TRIG_IN** and the TRIG_OUT mode to **TRIGGER_OR_TRIG_IN**.

4. Select the hw_ila_2 tab, and delete the existing probe in the Basic Trigger Setup window by selecting it and clicking the Remove Selected Probe "**-**" button to the left.

5. Click on the + sign in the Trigger Setup window to add the `mb_subsystem_i/ila_0_TRIG_OUT_TRIG` signal from the Add Probes window.

6. In the Basic Trigger Setup window, click the **Compare Value** column for the `mb_subsystem_i/ila_0_TRIG_OUT_TRIG` signal, and set the **Radix** field to **[B] (Binary)** and the **Value** field to **1**. This essentially sets up the ILA to trigger when the trig_out transitions to a value of 1.

7. Click **OK**.

8. Click the **Run Trigger** button  to "arm" the ILA. It moves into the "`Waiting for Trigger`" condition.

9. 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 164.

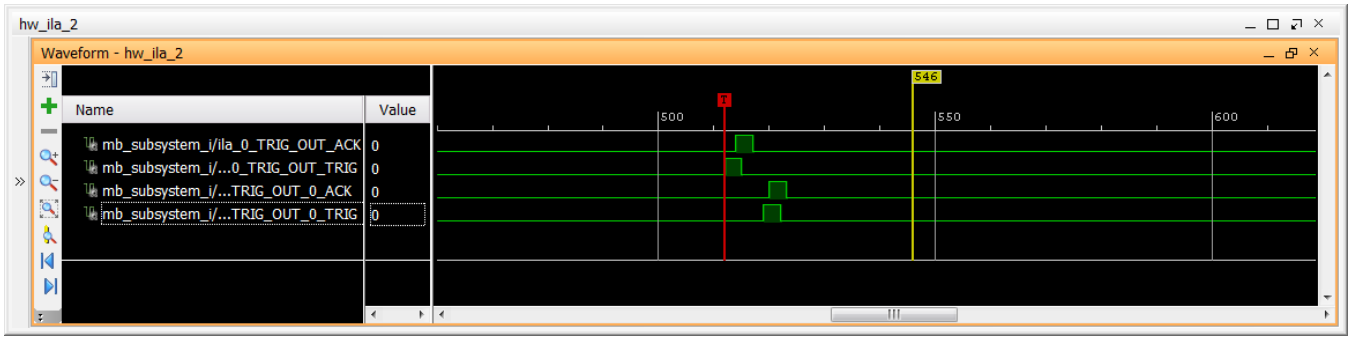

Figure 164: Waveform Demonstrating PS to PL Trigger

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

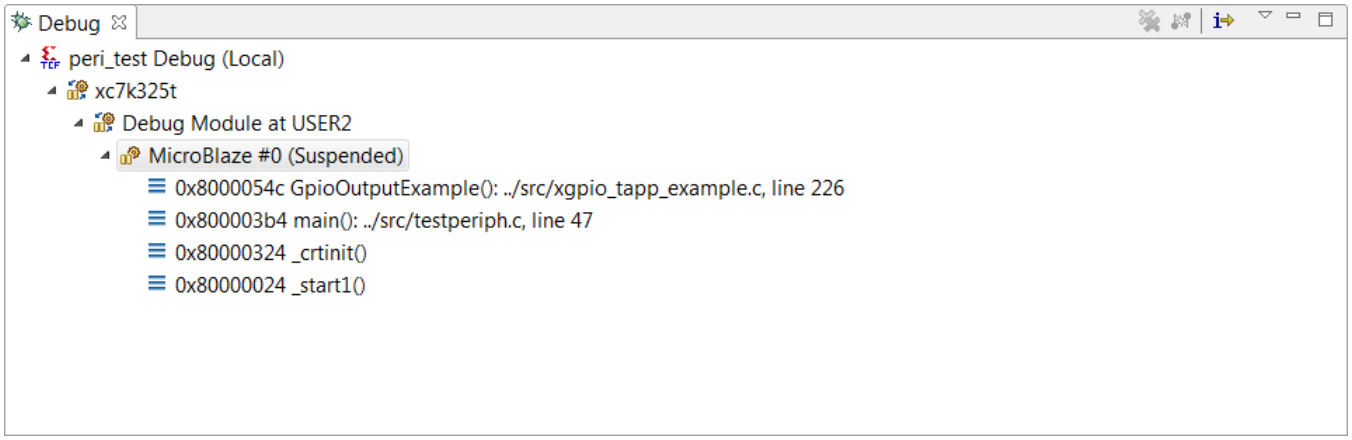

Figure 165: Verify that the Processor Has Been Interrupted in SDK

Conclusion

In this tutorial, you:

- Stitched together a design in the Vivado IP integrator tool
- Brought the design through implementation and bitstream generation
- Exported the hardware to SDK
- Created and modified application code that runs on a Standalone Operating System
- Modified the linker script so that the code executes from the DDR3
- Verified cross-trigger functionality between the MicroBlaze processor executing code and the design logic

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.

Please Read: Important Legal Notices