Vivado Design Suite Tutorial

Embedded Processor Hardware Design

UG940 (v 2014.3) October 1, 2014
Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
Table of Contents

Revision History ............................................................................................................................................... 2
Table of Contents ........................................................................................................................................... 3
Programming and Debugging Embedded Processors ..................................................................................... 5
  Software Requirements .................................................................................................................................. 5
  Hardware Requirements .............................................................................................................................. 5
Tutorial Design Descriptions .......................................................................................................................... 6
Locating Tutorial Design Files ....................................................................................................................... 8
Lab 1: Building a Zynq-7000 AP SoC Processor Design .................................................................................. 9
  Introduction .................................................................................................................................................. 9
  Step 1: Start the Vivado IDE and Create a Project ....................................................................................... 9
  Step 2: Create an IP Integrator Design ....................................................................................................... 11
  Step 3: Using MARK_DEBUG .................................................................................................................... 18
  Step 4: Generate HDL Design Files ........................................................................................................... 20
  Step 5: Synthesize the Design .................................................................................................................... 22
  Step 6: Assign Debug Net to an ILA Core .................................................................................................... 23
  Step 7: Implement Design and Generate Bitstream ..................................................................................... 25
  Step 8: Export Hardware to SDK .............................................................................................................. 28
  Conclusion .................................................................................................................................................. 29
Lab Files ........................................................................................................................................................... 29
Lab 2: Using SDK and the Vivado IDE Logic Analyzer .................................................................................. 30
  Introduction .................................................................................................................................................. 30
  Step 1: Start SDK and Create a Software Application ............................................................................... 30
  Step 2: Run the Software Application ....................................................................................................... 33
  Step 3: Connect to the Vivado Logic Analyzer .......................................................................................... 36
  Conclusion .................................................................................................................................................. 40
Lab 3: Zynq-7000 AP SoC Cross-Trigger Design .......................................................................................... 41
  Introduction .................................................................................................................................................. 41
  Step 1: Start the Vivado IDE and Create a Project ....................................................................................... 41
  Step 2: Create an IP Integrator Design ....................................................................................................... 42
  Step 3: Synthesize Design .......................................................................................................................... 48
  Step 4: Create an Integrated Logic Analyzer Debug Core .......................................................................... 49
<table>
<thead>
<tr>
<th>Step 5: Implement Design and Generate Bitstream</th>
<th>53</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 6: Export Hardware to SDK</td>
<td>54</td>
</tr>
<tr>
<td>Step 7: Build Application Code in SDK</td>
<td>55</td>
</tr>
<tr>
<td>Step 9: Connect to Vivado Logic Analyzer</td>
<td>62</td>
</tr>
<tr>
<td>Conclusion</td>
<td>69</td>
</tr>
<tr>
<td>Lab Files</td>
<td>69</td>
</tr>
</tbody>
</table>

Lab 4: Using the Embedded MicroBlaze Processor | 70 |

<table>
<thead>
<tr>
<th>Introduction</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1: Invoke the Vivado IDE and Create a Project</td>
<td>71</td>
</tr>
<tr>
<td>Step 2: Create an IP Integrator Design</td>
<td>72</td>
</tr>
<tr>
<td>Step 3: Memory-Mapping the Peripherals in IP Integrator</td>
<td>83</td>
</tr>
<tr>
<td>Step 4: Validate Block Design</td>
<td>84</td>
</tr>
<tr>
<td>Step 5: Generate Output Products</td>
<td>85</td>
</tr>
<tr>
<td>Step 6: Creating a Top-Level Verilog Wrapper</td>
<td>85</td>
</tr>
<tr>
<td>Step 7: Take the Design through Implementation</td>
<td>86</td>
</tr>
<tr>
<td>Step 8: Exporting the Design to SDK</td>
<td>87</td>
</tr>
<tr>
<td>Step 9: Creating a &quot;Hello World&quot; Application</td>
<td>88</td>
</tr>
<tr>
<td>Step 10: Executing the System on a KC705 Board</td>
<td>92</td>
</tr>
<tr>
<td>Conclusion</td>
<td>93</td>
</tr>
<tr>
<td>Lab Files</td>
<td>93</td>
</tr>
</tbody>
</table>

Legal Notices | 94 |

Please Read: Important Legal Notices | 94 |
Programming and Debugging Embedded Processors

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

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

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

Software Requirements

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

Hardware Requirements

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

Lab 1: Programming a Zynq-7000 AP SoC Processor

Lab 1 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

Hardware and Software Requirements

The following software is required for Lab 1.

- Vivado Design Suite 2014.3 (Embedded Version)

Lab 1 shows how to graphically build a design in the Vivado IP integrator and use the Designer Assistance feature to connect the IP to the Zynq-7000 AP SoC PS. The lab also demonstrates the Board Automation feature for the ZYNQ ZC702 Evaluation Board.

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

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

Design Files

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

- lab1.tcl

See LOCATING TUTORIAL DESIGN FILES, page 8.
Lab 2: SDK and Logic Analyzer

Lab 2 requires that you have the Software Development Kit (SDK) software installed on your machine. In Lab 2, you use the SDK software to build and debug the design software, and learn how to connect to the hardware server (\texttt{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.

**Hardware and Software Requirements**

The following hardware and software are required for Lab 2.

Hardware Requirements:

- Xilinx Zynq-7000 AP SoC ZC702 board
- One USB (Type A to Type B)
- JTAG platform USB Cable or Digilent Cable
- Power cable to the board

Software Requirements:

- Vivado Design Suite 2014.3 (Embedded Version)

Lab 3: Zynq-7000 AP SoC Cross Trigger Design

Lab 3 requires that you have the Software Development Kit (SDK) software installed on your machine. In Lab 3, you use the SDK software to build and debug the design software, and learn how to connect to the hardware server (\texttt{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.

**Hardware and Software Requirements**

The following hardware and software are required for Lab 3.

Hardware Requirements:

- Xilinx Zynq-7000 AP SoC ZC702 board
- One USB (Type A to Type B)
- JTAG platform USB Cable or Digilent Cable
- Power cable to the board

Software Requirements:

- Vivado Design Suite 2014.3 (Embedded Version)

**Design Files**

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

- \texttt{lab3.tcl}

See [LOCATING TUTORIAL DESIGN FILES](#), page 8.
Lab 4: Programming a MicroBlaze Processor

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

**Hardware and Software Requirements**

The following hardware and software are required for Lab 4.

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

Software Requirements:
- Vivado Design Suite 2014.3

**Design Files**

The following design files are included in the zip file for this guide:
- lab4.tcl

See [Locating Tutorial Design Files](#), page 8.

---

**Locating Tutorial Design Files**

Design data is in the ug940-design-files.zip file, which can be found at the following link: [https://secure.xilinx.com/webreg/clickthrough.do?cid=363328](https://secure.xilinx.com/webreg/clickthrough.do?cid=363328).
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 (FIGURE 1) by clicking the Vivado desktop icon or by typing vivado at a terminal command line.

2. From the Quick Start section, click Create New Project.

Figure 1: Vivado Quick Start Page
Step 1: Start the Vivado IDE and Create a Project

The New Project wizard opens (FIGURE 2).

3. Click Next.

4. In the Project Name dialog box, type a project name and select a location for the project files. Ensure that the Create project subdirectory check box is checked, and then click Next.

5. In the Project Type dialog box, select RTL Project, and then click Next.

6. In the Add Sources dialog box, set the Target language to your desired language, Simulator language to Mixed, and then click Next.

7. In the Add Existing IP dialog box, click Next.

8. In the Add Constraints dialog box, click Next.

9. In the Default Part dialog box:
   a. Select Boards.
   b. From the Board Rev drop-down list, select All to view all versions of the supported boards.

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

<table>
<thead>
<tr>
<th>CAUTION!</th>
</tr>
</thead>
</table>
| Choose the version of the ZYNQ-7 ZC702 Evaluation Board that you are using.

10. Review the project summary in the New Project Summary dialog box, and then click Finish to create the project.
Step 2: Create an IP Integrator Design

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

![Create Block Design Dialog Box](image)

Figure 3: Create Block Design Dialog Box

3. Click OK.
4. Right-click in the Diagram panel of the Vivado IP integrator window, and select Add IP.
   Alternatively, you can click the Add IP link in the IP integrator diagram area.

![Add IP Link in IP Integrator Canvas](image)

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.

![The IP Integrator IP Catalog](image)

Figure 5: The IP Integrator IP Catalog
Step 2: Create an IP Integrator Design

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

   In the Tcl Console, you see the following message:
   
   ```
   create_bd_cell -type ip -vlnv xilinx.com:ip:processing_system7:5.5
   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.

   The **Run Block Automation** dialog box opens, stating that the FIXED_IO, Trigger, 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 6: Zynq-7 Run Block Automation Dialog Box](image)
9. Click **OK**.

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

![Diagram of Zynq-7000 AP SoC Processing System after Running Block Automation](image1)

**Figure 7: Zynq-7000 AP SoC Processing System after Running Block Automation**

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

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

12. Similarly, add the AXI BRAM Controller.

Your Block Design window should look similar to **FIGURE 8**. 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.

![Diagram of Block Design after Instantiating IP](image2)

**Figure 8: 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** (**Figure 9**).

   ![Figure 9: Run Connection Automation](image)

2. The Run Connection Automation dialog box opens (**Figure 10**).

   ![Figure 10: Run Connection Automation Message](image)
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 11**).

![Figure 11: Connection Automation Options](image)

4. The **S_AXI** Interface of the axi_bram_ctrl_0 has a **Clock Connection (for unconnected clks)** field, which needs to be set to its default value of **Auto**. This will choose the default clock, FCLK_CLK0, generated by the PS7 for this interface.

![Figure 12: S_AXI Connection Automation Options for axi_bram_ctrl_0](image)
Step 2: Create an IP Integrator Design

5. For the axi_gpio_0 interface, to use the LEDs on the board, select the GPIO interface under axi_gpio_0 and set the Select Board Part Interface selection to leds_4bits from the drop-down list (FIGURE 12).

![Run Connection Automation Dialog Box for GPIO Interface of axi_gpio_0](image1.png)

**Figure 13: Run Connection Automation Dialog Box for GPIO Interface of axi_gpio_0**

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

![S_AXI Interface Options for axi_gpio_0](image2.png)

**Figure 14: S_AXI Interface Options for axi_gpio_0**

7. Click **OK**.
Step 2: Create an IP Integrator Design

The IP integrator subsystem looks like Figure 14. The relative positions of the IP might differ slightly.

Figure 15: Zynq-7000 AP SoC Processor System

8. 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.

9. Change the range of the AXI BRAM Controller to 64K, as shown in Figure 16.

Figure 16: Change Range of axi_bram_ctrl to 64K

10. Click on the Diagram tab to go back to the block design.

11. Click the Regerate Layout button to regenerate an optimal layout of the block design.
Step 3: Using MARK_DEBUG

You now instrument the design and add hooks 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, on the interface that you chose to mark for debug. (**FIGURE 17**).

![Design Hierarchy: Icon for Mark Debug](image.png)

2. Right-click anywhere in the IP integrator diagram and select **Create Comment** to add a title and/or comment to the design. This step is optional.

3. If you added a comment and title in step 2, you can format it. To do so:
   a. Select the comment box.
   b. Right-click and select **Format Comment** to format the comment.
4. The Format Comment dialog box opens (Figure 18).

![Format Comment Dialog Box](image)

**Figure 18: Format Comment Dialog Box**

5. Select the appropriate options and click **OK**.

6. 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.

7. Click **OK**.

8. Save the block design by selecting **File > Save Block Design** from the Vivado menu.

   Alternatively, you can press **Ctrl + S** to save your block design or click the **Save** button in the Vivado toolbar.
Step 4: Generate HDL Design Files

You now generate the HDL files for the design.

1. In the Sources window, right-click the top-level subsystem design and select **Generate Output Products**.
   
   This generates the source files for the IP used in the block design and the relevant constraints file.
   
   You can also click **Generate Block Design** in the Flow Navigator to generate the output products.

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

   ![Generate Output Products Dialog Box](image)

3. In the Sources window, right-click the top-level subsystem source and select **Create HDL Wrapper** to create an example top level HDL file.
Step 4: Generate HDL Design Files

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.

![Create HDL Wrapper Dialog Box](image)

**Figure 20: Create HDL Wrapper Dialog Box**

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

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

![Source Window After Creating the Wrapper](image)

**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 appears.

   ![Synthesis Completed Dialog Box](image)

   Figure 22: Synthesis Completed Dialog Box

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

   Vivado opens the synthesized design.

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

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

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

   ![Unassigned Debug Nets Folder](image)

   Figure 23: Unassigned Debug Nets Folder
Step 6: Assign Debug Net to an ILA Core

1. On the left-hand toolbar of the Debug window, click the **Set up Debug** button.
2. When the Set up Debug wizard opens (**Figure 24**), click **Next**.

![Figure 24: Set Up Debug Wizard](image1)

On the second panel of the wizard, notice that some of the nets in the table do not belong to a clock domain (**Figure 25**).

![Figure 25: Set Up Debug](image2)

**IMPORTANT:** Signals captured by the same ILA core must have the same clock domain selection.
3. Highlight the three signals that don’t have the clock domain association, then right-click and select Select Clock Domain (FIGURE 26).

![Figure 26: Set Up Debug: Select Clock Domain](image)

4. In the Select Clock Domain window, make sure that the selected filter is GLOBAL CLOCK and the Search hierarchically check box is checked.

5. Select the zynq_design_1_i/FCLK_CLK0 net as the clock domain (FIGURE 27).

![Figure 27: Select Clock Domain Option](image)

6. Click OK.

   The Specify Nets to Debug dialog box updates with the assigned clock domains.
Step 7: Implement Design and Generate Bitstream

7. Click **Next**.

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

   ![Figure 28: Setting Trigger and Capture Modes](image)

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

10. Click **Next**.

    The Set up Debug Summary page appears.

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

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

---

**Step 7: Implement Design and Generate Bitstream**

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

    The Save Project dialog box opens.

2. Click **Save**.

    The No Implementation Results Available dialog box opens.

3. Click **Yes**.

    During implementation flow, messages in the Log window show the implementation of the debug cores.
Step 7: Implement Design and Generate Bitstream

This step is required to synthesize the debug core modules so that they can replace the debug core black boxes that you added to the design previously (FIGURE 29).

![Messages](image1)

**Figure 29: Messages**

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

4. Notice that you can view the progress of the flow in the upper right-hand corner of the interface (FIGURE 30).

![Status Indicator](image2)

**Figure 30: Status Indicator**

After the bitstream generates, in the Bitstream Generation Completed dialog box that opens, **Open Implemented Design** is selected by default (FIGURE 31).

![Bitstream Generation Completed](image3)

**Figure 31: Bitstream Generation Completed**
5. Click OK.

6. The Vivado dialog box asks if you want to keep the synthesized design open. You can keep the synthesized design open if you want to debug more signals; otherwise close the synthesized design to save memory.

7. In the implemented design, go to the Netlist window to see the inserted ILA and Debug Hub (dbg_hub) cores in the design (FIGURE 32).

![Figure 32: Implemented Design](image)

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

9. In the Report Timing Summary dialog box, click OK.
Step 8: Export Hardware to SDK

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

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

1. From the main Vivado File menu, select **File > Export > Export Hardware**.
   
The Export Hardware dialog box opens.
   
   Ensure that the **Include Bitstream** check box is checked and that the **Export to** field is set to the default option of `<Local to Project>` (FIGURE 33).

   ![Export Hardware dialog box](image)

   **Figure 33: Export Hardware for SDK**

2. Click **OK**.
3. To launch SDK, select **File > Launch SDK**.
   
The Launch SDK dialog box opens.
4. Accept the default selections for **Exported location** and **Workspace**.

![Launch SDK Dialog Box](image)

**Figure 34: Launch SDK Dialog Box**

5. Click **OK**.

---

## 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 Design Automation and Connection Automation features.
- Marked nets for debug, to analyze them in the Vivado Integrated Logic Analyzer.
- Inserted debug probes in the design to debug it later in the Vivado Integrated Logic Analyzer.
- Synthesized, implemented, and generated the bitstream before exporting the hardware definition to SDK.

---

## Lab Files

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

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

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

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

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

Step 1: Start SDK and Create a Software Application

1. If you are doing this lab as a continuation of Lab 1 then SDK should have launched in a separate window. You can also start SDK from the Windows Start menu by clicking on Start > All Programs > Xilinx Design Tools > Vivado 2014.3 > Xilinx SDK 2014.3.

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

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

   Now you can create a software application.


   The New Project dialog box opens.
4. In the Project Name field, type **Zynq_Design** (*Figure 35*).

![Figure 35: SDK Application Project](image)

5. Click **Next**.
6. From the Available Templates, select **Peripheral Tests** *(FIGURE 36)*.

![Figure 36: SDK New Project Template](image)

7. Click **Finish**.

When the program finishes compiling, you see the following in the Console window *(FIGURE 37)*.

![Figure 37: SDK Message](image)
Step 2: Run the Software Application

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

1. Download the bitstream into the FPGA by selecting Xilinx Tools > Program FPGA.
   The Program FPGA dialog box opens.
2. Ensure that the Bitstream field shows the bitstream file that you created in Step 7 of Lab 1, and then click Program.
   \textit{Note: The DONE LED on the board turns green if the programming is successful.}
3. Select and right-click the Zynq Design application in the Project Explorer.
4. Select Debug As > Debug Configurations.
5. In the Debug Configurations dialog box, right-click Xilinx C/C++ application (GDB) and select New.
6. In the Debug Configurations dialog box, click Debug.

![Figure 38: Run Debug Configurations](image)

The Confirm Perspective Switch dialog box opens.
7. Click Yes.

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

9. Use the following settings for the ZC702 board (**FIGURE 39**).

![Terminal Settings for ZC702 Board](image)

**Figure 39: Terminal Settings for ZC702 Board**

10. Click **OK**.

11. Verify the **Terminal** connection by checking the status at the top of the tab (**FIGURE 40**).

![Terminal Connection Verification](image)

**Figure 40: Terminal Connection Verification**
12. In the **Debug** tab, expand the tree to see the processor core on which the program is running (Figure 41).

![Figure 41: Processor Core to Debug](image)

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

**Add a Breakpoint**

Next, add a breakpoint on line 103.

1. Select **Navigate > Go To Line**.
2. In the Go To Line dialog box, type **103** and click **OK**.

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

3. Double click in the blue bar to the left of line 103 to add a breakpoint on that line of source code (Figure 42).

![Figure 42: Add a Breakpoint](image)
Step 3: Connect to the Vivado Logic Analyzer

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

1. Open the Vivado project from Lab 1 if it is not already open.

2. In Vivado, select **Open Hardware Manager** from the Program and Debug folder in the Flow Navigator.

3. In the Hardware Manager window, click **Open target** and select **Open New Target** to open a connection to the Digilent JTAG cable for ZC702 (FIGURE 43).

   ![Figure 43: Launch Open New Hardware Target Wizard](image)

   The Open New Hardware Target dialog box opens.

4. Click **Next**.

5. Click **Next** on the Hardware Server Settings page.

6. Click **Next** on the Select Hardware Target page.

7. Click **Finish**.

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

   ![Figure 44: Successfully Programmed Hardware Session](image)

8. First, ensure that the ILA core is alive and capturing data. To do this, select the `hw_ila_1` core in the Hardware window.
Step 3: Connect to the Vivado Logic Analyzer

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

   Static data from the ILA core displays in the waveform window (**Figure 45**).

![Figure 45: Static Data from Set Breakpoint](image)

10. Set up a condition that triggers when the application code writes to the GPIO peripheral. To do this:
    a. Select, drag and drop the `/zynq_design_1_i/processing_system7_0_axi_periph_M00_AXI_AWVALID` signal from the Debug Probes window into the Basic Trigger Setup area.

![Figure 46: The ILA Properties window](image)
Step 3: Connect to the Vivado Logic Analyzer

b. Click the **Compare Value** column of the *WVALID row, as shown in **FIGURE 47**.

![Figure 47: ILA Properties Window](image)

<table>
<thead>
<tr>
<th><strong>ILA Properties</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Trigger Mode Settings</strong></td>
</tr>
<tr>
<td><strong>Trigger mode:</strong> BASIC_ONLY</td>
</tr>
<tr>
<td><strong>Capture Mode Settings</strong></td>
</tr>
<tr>
<td><strong>Capture mode:</strong> ALWAYS</td>
</tr>
<tr>
<td><strong>Number of windows:</strong> 1</td>
</tr>
<tr>
<td><strong>Window data depth:</strong> 1024</td>
</tr>
<tr>
<td><strong>Trigger position in window:</strong> 0</td>
</tr>
<tr>
<td><strong>General Settings</strong></td>
</tr>
<tr>
<td><strong>Refresh rate (ms):</strong> 500</td>
</tr>
<tr>
<td><strong>Trigger Capture Status</strong></td>
</tr>
<tr>
<td><strong>Core status:</strong> Idle</td>
</tr>
<tr>
<td><strong>Window 1 of 1:</strong> Window sample 0 of 1024</td>
</tr>
<tr>
<td><strong>Basic Trigger Setup</strong></td>
</tr>
<tr>
<td><strong>Operator:</strong> == (equal)</td>
</tr>
<tr>
<td><strong>Value:</strong> 1</td>
</tr>
</tbody>
</table>

![Figure 48: Change Debug Probe Settings](image)

<table>
<thead>
<tr>
<th><strong>ILA Properties</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Trigger Mode Settings</strong></td>
</tr>
<tr>
<td><strong>Trigger mode:</strong> BASIC_ONLY</td>
</tr>
<tr>
<td><strong>Capture Mode Settings</strong></td>
</tr>
<tr>
<td><strong>Capture mode:</strong> ALWAYS</td>
</tr>
<tr>
<td><strong>Number of windows:</strong> 1</td>
</tr>
<tr>
<td><strong>Window data depth:</strong> 1024</td>
</tr>
<tr>
<td><strong>Trigger position in window:</strong> 512</td>
</tr>
<tr>
<td><strong>General Settings</strong></td>
</tr>
<tr>
<td><strong>Refresh rate (ms):</strong> 500</td>
</tr>
<tr>
<td><strong>Trigger Capture Status</strong></td>
</tr>
<tr>
<td><strong>Core status:</strong> Idle</td>
</tr>
<tr>
<td><strong>Window 1 of 1:</strong> Window sample 0 of 1024</td>
</tr>
<tr>
<td><strong>Basic Trigger Setup</strong></td>
</tr>
<tr>
<td><strong>Operator:</strong> == (equal)</td>
</tr>
<tr>
<td><strong>Value:</strong> 1</td>
</tr>
</tbody>
</table>

11. Change the trigger position to the middle of the 1024 sample window by setting the **Trigger Position** for the hw_ial_1 core in the ILA Properties window to **512** and then pressing **Enter** (**FIGURE 48**).
Step 3: Connect to the Vivado Logic Analyzer

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

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

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

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

   ![Figure 49: Status of hw_ila_1](image)

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

   Alternatively, you can press F8 to resume code execution.

   The code execution stops at the breakpoint you set on line 103.

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

   This causes at least one write operation to the GPIO peripheral. These write operations cause the WVALID signal to go from 0 to 1, thereby triggering the ILA core.
As you step over the following line of code, you see the following transaction captured by looking at the waveform window in the Vivado logic analyzer feature:

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

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

**Figure 50:** Trigger Mark Goes to 1

15. If you are going on to Lab 3, close your project by selecting **File > Close Project**.
16. 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 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.
Lab 3: Zynq-7000 AP SoC Cross-Trigger Design

Introduction

In this lab, you use the cross-trigger functionality between the Zynq-7000 AP SoC processor and the fabric logic. Cross triggering is a powerful feature that you can use to co-debug software running in real time on the target hardware. This tutorial guides you from design creation in IP integrator, to marking the nets for debug and manipulating the design to stitch up the cross-trigger functionality.

Step 1: Start the Vivado IDE and Create a Project

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

1. In Vivado Flow Navigator, click **Create Block Design**.

2. In the **Create Block Design** dialog box, specify the name of the IP integrator design. For this tutorial, use `zynq_processor_system`.

3. Leave the **Directory** field set to its default value of `<Local to Project>` and the **Specify source set** field to **Design Sources**.

4. Click **OK**.

   The IP integrator diagram window opens.

5. Click **Add IP** at the top of the diagram area (**FIGURE 51**).

![Figure 51: Add IP to the Design](image)

The IP catalog opens.

6. In the Search field, type `Zynq` and select the **ZYNQ7 Processing System** IP and press **Enter**. Alternatively, double-click the **ZYNQ7 Processing System** IP to instantiate it (**FIGURE 52**).

![Figure 52: Instantiate the ZYNQ7 Processing System](image)

7. In the header at the top of the diagram, click **Run Block Automation** (**FIGURE 53**).

![Figure 53: Run Block Automation on Zynq Processing System](image)

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** (**FIGURE 54**).

**Figure 54: Run Block Automation Dialog Box**

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

**Figure 55: IP Integrator Canvas After Running Block Automation**
Step 2: Create an IP Integrator Design

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 as follows (Figure 56).

![Diagram after Instantiating IP for This Design](image)

**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.

<table>
<thead>
<tr>
<th>Connection</th>
<th>More Information</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>/axi_bram_ctrl_0/BRAM_PORTA</td>
<td>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.</td>
<td>No options.</td>
</tr>
<tr>
<td>/axi_bram_ctrl_0/BRAM_PORTB</td>
<td>Note that the Run Connection Automation dialog box offers two choices now. The first one is to use the existing Block Memory Generator from the previous step or you can chose to instantiate a new Block Memory Generator if desired. In this case we will use the existing BMG.</td>
<td>Leave the Blk_Mem_Gen field set to its default value of /Blk_Mem_Gen of BRAM_PORTA.</td>
</tr>
</tbody>
</table>
Step 2: Create an IP Integrator Design

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>/axi_bram_ctrl_0/S_AXI</td>
<td>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.</td>
<td>Leave the <strong>Clock Connection (for unconnected clks)</strong> field set to <strong>Auto</strong>.</td>
</tr>
<tr>
<td>/axi_gpio_0/GPIO</td>
<td>The Run Connection Automation dialog box shows the interfaces that are available on the ZC702 board to connect to the GPIO.</td>
<td>Select <strong>LEDs_4Bits</strong>.</td>
</tr>
<tr>
<td>/axi_gpio_0/S_AXI</td>
<td>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. 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.</td>
<td>Leave the <strong>Clock Connection (for unconnected clks)</strong> field set to <strong>Auto</strong>.</td>
</tr>
</tbody>
</table>

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

**Figure 57: Design after Running Connection Automation**
Step 2: Create an IP Integrator Design

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 (**FIGURE 58**).

![Figure 58: Memory Map the Slaves](image)

Next, you will mark some nets for debug.

14. Click the **Diagram** tab again and select the net connecting the **gpio** pin of the AXI GPIO IP to the **LEDs_4Bits** port.

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

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

You will also mark the net connecting the interfaces **TRIG_OUT** of the ila_0 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 us 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. We will monitor the net between **S_AXI** interface of AXI GPIO and **M00_AXI** interface of the AXI Interconnect (**processing_system7_0_axi_periph**). To monitor this interface, hover the cursor on the **SLOT_0_AXI** interface port of the ila_0 until it changes into a pencil, and click and drag over to the net to be monitored (**S_AXI** interface of GPIO).

16. The **clk** pin of the ILA is not yet connected either. Connect it to the **FCLK_CLK0** pin of the ZYNQ7 Processing System.
Step 2: Create an IP Integrator Design

17. 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](image)

18. 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.

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

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

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

   The Generate Output Products dialog box opens (**Figure 60**).

![Figure 60: Generate Output Products Dialog Box](image)
21. Click **Generate**.

22. Click **OK** on the Generate Output Products dialog box.

23. 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.

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

---

**Step 3: Synthesize Design**

1. From the Flow Navigator, under **Synthesis**, click **Run Synthesis**.

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

   You can also click **Synthesis > Open Synthesized Design** in Flow Navigator.
Step 4: Create an Integrated Logic Analyzer Debug Core

When the synthesized design schematic opens, the Debug window also opens. In the Debug window, notice that there is one ILA, ila_0, that has already been inserted in the design. This ILA was instantiated in the block design and will monitor the AXI transactions to the GPIO. Notice that there are the 4 bits of output leds_4bits 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. We will insert the ILA to monitor these output bits.

![Figure 63: Unassigned Debug Nets](image)

1. From the Vivado main menu, select **Tools > Set up Debug.**
   Alternatively, from the left side of the **Debug** window, click the **Set up Debug** button.
   The Set up Debug dialog box opens (**Figure 64**).
2. Click **Next**.

![Figure 64: Set Up Debug Dialog Box](image)
Step 4: Create an Integrated Logic Analyzer Debug Core

3. In the **Nets to Debug** page, select the nets that you are interested in monitoring from the list of nets that are offered (**FIGURE 65**).

4. Click the **Find Nets to Add** button at the bottom of the dialog box.

![Figure 65: Add Nets to Debug](image)

5. In the find Nets dialog box (**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 on them.

![Figure 66: Find Nets with MARK_DEBUG Attribute](image)

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`
Step 4: Create an Integrated Logic Analyzer Debug Core

- `zynq_processor_system_i/processing_system7_0_TRIGGOUT_OUT_0_TRIG`

8. Click **OK**.

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

9. Click **Next**.

![Figure 67: Add Nets Connected to Cross-Trigger Pins](image)

![Figure 68: Specify Nets for Debugging](image)
Step 4: Create an Integrated Logic Analyzer Debug Core

The ILA (Integrated Logic Analyzer) Core Options dialog box opens (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. Click **Save**.

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

2. Click **Yes**.

   When bitstream generation completes, the **Bitstream Generation Completed** dialog box opens, with the option **Open Implemented Design** option checked by default.

3. Click **OK** to open the implemented design.

4. If you see a message box that asks about closing synthesized design before opening implemented design, click **Yes**.

5. In the **Flow Navigator, under Implementation**, click Implemented Design to expand it, if it is not already expanded.

6. From the expanded selection, select **Report Timing Summary** to see if the constraints are met.

7. When the Report Timing Summary dialog box opens, click **OK**.

8. Ensure that all timing constraints are met by looking at the Timing Summary tab (**Figure 71**).

![Figure 71: Timing Summary](image-url)
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> (FIGURE 72).

3. Click OK.

4. Select File > Launch SDK. Make sure that both the Exported location and Workspace fields are set to <Local to Project> (FIGURE 73).

5. Click OK.
Step 7: Build Application Code in SDK

1. SDK launches in a separate window. Select File > New > Application Project.
2. In the New Project dialog box, specify the name for your project. For this tutorial, use the name peri_test (FIGURE 74).

![Figure 74: Name the Application Project](image)

3. Click Next.
4. From the Available Templates, select Peripheral Tests (FIGURE 75).
5. Click **Finish**.

6. Make sure that you have connected the target board to the host computer.

7. Select **Xilinx Tools > Program FPGA** to open the **Program FPGA** dialog box.
8. In the **Program FPGA** dialog box, click **Program** *(Figure 76)*.

![Figure 76: Program FPGA Dialog Box](Image)

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](image)

11. In the **Create, manage, and run configurations** screen, select the Target Setup tab and check the **Enable Cross triggering** check box.
12. Click **Debug** (**FIGURE 78**).

![Figure 78: Enable Cross-Triggering](image)

The Confirm Perspective Switch dialog box opens.

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

The Debug perspective window opens.

14. Set the terminal by selecting the **Terminal 1** tab and clicking the **Settings** button [ ].
15. Use the following settings for the ZC702 board and click **OK** (FIGURE 79).

![Figure 79: Terminal Settings](image)

16. Verify the Terminal connection by checking the status at the top of the tab (FIGURE 80).

![Figure 80: Verify Terminal Connection](image)

17. If it is not already open, select `../src/testperiph.c`, and double click to open the source file.
18. Modify the source file by inserting a while statement at line 94. Click the blue bar on the left side of the `testperiph.c` window as shown in the figure and select Show Line Numbers.

19. In line 94, add `while(1)` above in front of the curly brace as shown (Figure 81).

![Figure 81: Modify testperiph.c](image)

20. Add a breakpoint in the code so that the processor stops code execution when the breakpoint is encountered. To do so, scroll down to line 98 and double-click on the left pane, which adds a breakpoint on that line of code (Figure 82). Click Ctrl + S to save the file. Alternatively, you can select File > Save.

![Figure 82: Set a Breakpoint](image)

Now you are ready to execute the code from SDK.
Step 9: Connect to Vivado Logic Analyzer

Connect to the ZC702 board using the Vivado Logic Analyzer.

1. In the Vivado IDE session, from the **Program and Debug** drop-down list of the Vivado Flow Navigator, select **Open Hardware Manager**.

2. In the Hardware Manager window, click **Open target > Open New Target**.

   ![Figure 83: Open a New Hardware Target](image)

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

3. The Open New Hardware Target dialog box opens (**FIGURE 84**). Click **Next**.

   ![Figure 84: Open Hardware Target](image)
Step 9: Connect to Vivado Logic Analyzer

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

5. Click **Next**.

![Figure 85: Specify Server Name](image)

6. On the Select Hardware Target page (FIGURE 86), click **Next**.

![Figure 86: Select Hardware Target](image)
Step 9: Connect to Vivado Logic Analyzer

7. Ensure that all the settings are correct on the Open Hardware Target Summary dialog box (Figure 87) and click Finish.

Figure 87: Open Hardware Target Summary

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

Figure 88: Vivado Hardware Window
9. Select the ILA - hw_ila_1 tab and set the Trigger Mode Settings as follows:
   - Set Trigger mode to **TRIG_IN_ONLY**
   - Set TRIG_OUT mode to **TRIG_IN_ONLY**
   - Under Capture Mode Settings, change **Trigger position in window** to 512.

![Figure 89: Set ILA Properties for hw_ila_1](image)

10. Arm the ILA core by clicking the **Run Trigger** button.
    This arms the ILA and you should see the status “Waiting for Trigger” as shown in **FIGURE 90**.

![Figure 90: Armed ILA Core](image)

11. Select the **hw_ila_2** tab to set up trigger conditions to capture the trace. Select the **ILA - hw_ila_2** tab and set the Trigger Mode Settings as follows:
   - Set Trigger mode to **BASIC_ONLY**.
   - Under **Capture Mode Settings**, change **Trigger position in window** to 512.
12. With the **hw ila 2 tab** selected, drag and drop the **zynq_processor_system_i/processing_system7_0_TRIGGER_OUT_0_TRIG** signal from the Debug Probes window under **hw ila 2** into the Basic Trigger Setup window.

Alternatively, you can right-click the **zynq_processor_system_i/processing_system7_0_TRIGGER_OUT_0_TRIG** signal and select **Add Probes to Basic Trigger Setup**.

![Figure 91: Add zynq_processor_system_i/trig_out to Basic Trigger Setup](image)

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

![Figure 92: Set Trigger Condition](image)

14. Click **OK**.
15. Arm the ILA by clicking the **Run Trigger** button [Run Trigger] in the toolbar of the hw_ila_2 window. Just like hw_ila_1, this ILA should be “armed” and waiting for the trigger condition to happen.

16. In SDK, in the Debug window, click the **XMD Target Debug Agent** and click the **Resume** button [Resume].

Vivado displays the hw_ila_2 trigger as shown in **FIGURE 93**.

![Figure 93: PS to PL Cross Trigger Waveform](image)

Likewise, hw_ila_1 also triggers as seen in **FIGURE 94**.

![Figure 94: PS to PL Cross Trigger Waveform in hw_ila_1](image)

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

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

Alternatively, select the Breakpoints tab towards the top right corner of SDK window, right-click it, and select **Remove All**.
Step 9: Connect to Vivado Logic Analyzer

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

You will be able to see the code executing in the Terminal Window in SDK.

19. In Vivado, select the hw_ila_1 tab. Change the TRIG_OUT mode to TRIGGER_OR_TRIG_IN.

20. Select the hw_ila_2 tab, and delete the existing probe in the Basic Trigger Setup window by selecting it and clicking the Delete button to the left.

21. Drag and drop the zynq_processor_system_i/ila_0_TRIG_OUT_TRIG signal from the Debug Probes window under hw_ila_2 into the Basic Trigger Setup window.

   Alternatively, you can also right-click the zynq_processor_system_i/ila_0_TRIG_OUT_TRIG signal and select Add Probes to Basic Trigger Setup.

22. In the Basic Trigger Setup window, click the Compare Value column for the zynq_processor_system_i/ila_0_TRIG_OUT_TRIG signal, and set the Value field to 1. This essentially sets up the ILA to trigger when the trig_out transitions to a value of 1.

23. Click the Run Trigger button to “arm” the ILA. It moves into the “Waiting for Trigger” condition.

24. Select the hw_ila_1 tab again and click the Run Trigger Immediate button.

   This triggers the hw_ila_2 and the waveform window looks like Figure 95.

   Figure 95: 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 96: 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 Zynq-7000 AP SoC, which you perform in the GUI environment. Therefore, the only Tcl file provided is lab3.tcl.

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

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

The MicroBlaze system includes native Xilinx® IP including:

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

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

The application code developed in SDK prints “Hello World” in a stand-alone application mode.

This tutorial targets the Xilinx KC705 FPGA Evaluation Board.
Step 1: Invoke the Vivado IDE and Create a Project

1. Invoke the Vivado IDE by clicking the desktop icon or by typing `vivado` at a terminal command line.
2. From the Quick Start page, select Create New Project (FIGURE 97).

![Figure 97: Vivado Quick Start Page](image)

The New Project wizard opens.

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

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

```
set_property board part xilinx.com:kintex7:kc705:part0:1.1 [current_project]
```

Although Tcl commands are available for many of the actions performed in the Vivado IDE, they are not explained in this tutorial. Instead, a Tcl script is provided that can be used to recreate this entire project. See the Tcl Console for more information.
Step 2: Create an IP Integrator Design

1. From Flow Navigator, under IP integrator, select Create Block Design. The Create Block Design dialog box opens.

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

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

4. Click OK (FIGURE 98).

![Figure 98: Name Block Design](image)

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 link from the top of the diagram area.

![Figure 99: Add IP](image)

6. In the Search field, type microblaze to find the MicroBlaze IP, then select MicroBlaze and press the Enter key (FIGURE 100).

![Figure 100: Search Field](image)
Use the Board Part Interfaces Tab to Connect to Board Interfaces

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

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

![Figure 101: Using the Board Part Interfaces to Configure a MIG](image)

Figure 101: Using the Board Part Interfaces to Configure a MIG
2. Drag and drop the `ddr3_sDRAM` into the block design canvas.

   The Auto Connect dialog box opens, informing you that the MIG IP was instantiated on the block design and the interfaces for the KC705 board were connected.

   The Connect Board Part Interface dialog box opens (Figure 102).

![Auto Connect dialog box for DDR3](image)

**Figure 102: Auto-connect dialog box for DDR3**

3. Click OK.

   The block design looks like ![Block Design After Instantiating the MIG Core](image)

**Figure 103: Block Design After Instantiating the MIG Core**
4. In the Board window, notice that the ddr_sdram interface now is connected as shown by the circle and the interface symbols.

![Figure 104: DDR3 Interface Shown Under Connected Interfaces](image)

5. From the Board window, select rs232_uart under the Miscellaneous folder and drag and drop it into the block design canvas.

6. Click OK in the Auto Connect dialog box.

7. This instantiates the AXI Uartlite core on the block design, which looks like Figure 105.

![Figure 105: Block Design After Connecting the Rs232_Uart Interface](image)
Add Peripherals: AXI Timer and AXI BRAM Controller

1. By right-clicking in the IP integrator diagram area and selecting Add IP, search for and select the AXI Timer (FIGURE 106).

2. Likewise, add the AXI BRAM Controller (FIGURE 107).
Run Block Automation

1. Click **Run Block Automation** and select `/microblaze_0` (**Figure 108**).

![Figure 108: Run Block Automation](image)

2. The Run Block Automation dialog box opens (**Figure 109**).

![Figure 109: Run Block Automation Dialog Box](image)
Step 2: Create an IP Integrator Design

3. 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. Leave the **Debug Module** option to its default state of **Debug Only**.
   e. Leave the **Peripheral AXI Port** option set to its default value of **Enabled**.
   f. Check the **Interrupt Controller** option.
   g. Leave the **Clock Connection** option set to \( /\text{mig}_7\text{series}_1/\text{ui}_\text{clk} \) (100 MHz).

4. Click **OK**.

   This generates a basic **MicroBlaze** system in the IP Integrator diagram area (**FIGURE 110**).
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 (FIGURE 111).

![Figure 111: Run Connection Automation](image)

The Run Connection Automation dialog box opens.

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

![Figure 112: Run Connection Automation Dialog Box](image)
3. Now, use the **Run Connection Automation** dialog box to set the following connections.

<table>
<thead>
<tr>
<th>Connection</th>
<th>More Information</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>/axi_bram_ctrl_0/BRAM_PORTA</td>
<td>The only option for this automation is to instantiate a new Block Memory Generator as shown under options.</td>
<td>Leave the Blk_Mem_Gen option to its default value of Blk_Mem_Gen of BRAM_PORTA.</td>
</tr>
<tr>
<td>/axi_bram_ctrl_0/BRAM_PORTB</td>
<td>The Run Connection Automation dialog box opens and gives you two choices:</td>
<td>Leave it to its default value. Leave the Clock Connection (for unconnected clks) field set to its default value of Auto.</td>
</tr>
<tr>
<td></td>
<td>• Instantiate a new BMG and connect the PORTB of the AXI BRAM Controller to the new BMG IP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
<td></td>
</tr>
<tr>
<td>/axi_bram_ctrl_0/S_AXI</td>
<td>Two options are presented in this case. The Master field can be set for either cached or non-cached accesses.</td>
<td>The Run Connection Automation dialog box offers to connect this to the /microblaze_0 (Cached). Leave it to its default value. Leave the Clock Connection (for unconnected clks) field set to its default value of Auto.</td>
</tr>
<tr>
<td>/axi_timer_0/S_AXI</td>
<td>The Master field is set to its default value of /microblaze_0 (Periph).</td>
<td>Keep these default settings.</td>
</tr>
<tr>
<td></td>
<td>The Clock Connection (for unconnected clks) field is set to its default value of Auto.</td>
<td></td>
</tr>
<tr>
<td>/axi_uartlite_0/S_AXI</td>
<td>The Master field is set to its default value of /microblaze_0 (Periph).</td>
<td>Keep these default settings.</td>
</tr>
<tr>
<td></td>
<td>The Clock Connection (for unconnected clks) field is set to its default value of Auto.</td>
<td></td>
</tr>
</tbody>
</table>
Step 2: Create an IP Integrator Design

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>/mig_7series_0/S_AXI</td>
<td>The <strong>Master</strong> field is set to /microblaze_0 (Cached). Leave it to this value so the accesses to the DDR3 memory are cached accesses. The <strong>Clock Connection</strong> (for unconnected clks) field is set to its default value of <strong>Auto</strong>.</td>
<td>Keep these default settings.</td>
</tr>
<tr>
<td>/mig_7series_0/sys_rst</td>
<td>The board interface reset will be connected to the reset pin of the MIG controller.</td>
<td>Keep the default setting.</td>
</tr>
</tbody>
</table>

4. 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**. Relative placement of IP might be slightly different.

![IP Integrator Diagram](image)

**Figure 113: MicroBlaze Connected to UART and AXI Timer**

**Concatenate Interrupt Signals and Regenerate Layout**

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

1. Connect the interrupt pin of the AXI Timer to the input pin **In0[0:0]** of Concat.
2. Connect the interrupt pin of AXI Uartlite to the input pin **In1[0:0]** of Concat. The connections should look like **FIGURE 114**.
Step 2: Create an IP Integrator Design

3. 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 115**.

Figure 114: Connected Interrupt Ports

Figure 115: 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.
   b. Change the range of **mig_7_series_0** IP in both the **Data** and the **Instruction** section to **512 MB** (Figure 116).

   ![Address Editor Diagram](image)

   **Figure 116: Data and Instruction Set to 512 MB**

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

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

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

You can also use this map to manually include or exclude IP from the cacheable region or otherwise specify their addresses.
2. Double click on the MicroBlaze in the block design to re-configure it. Go to the Cache page of the Re-customize IP dialog box. On this page, for both the **Instruction Cache** and **Data Cache**:
   
   - Set the **Size in Bytes** option to **32 kB**.
   - Set the **Line length** option to **8**.
   - Set the **Base Address** to **0x80000000** by clicking on the Auto button, which enables the Base Address field.
   - Set the **High Address** to **0xFFFFFFFF** by clicking on the Auto button, which enables the High Address field.
   - Enable Use Cache for All Memory Accesses for both caches by clicking the Auto button first, and then checking the check box.

3. 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.

4. 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.

5. 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**.
   
   The Validate Design dialog box informs you that there are no critical warnings or errors in the design.

2. 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 appears.

2. Click **Generate**.

   ![Generate Output Products Dialog Box](image)

   **Figure 117: Generate Output Dialog Box**

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

Step 6: Creating a Top-Level Verilog Wrapper

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

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

2. Click **OK**.

   ![Create HDL Wrapper](image)

   **Figure 118: Creating an HDL Wrapper**
Step 7: Take the Design through Implementation

In the Flow Navigator:

1. Click **Generate Bitstream**.
   
The No Implementation Results Available dialog box asks if synthesis and implementation can be launched before generating bitstream.

2. Click **Yes**.
   
   Bitstream generation can take several minutes to complete. Once bitstream generation completes, the Bitstream Generation Completed dialog box asks you to select what to do next. Keep the default selection of **Open Implemented Design**.

3. Click **OK**.

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

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

6. Verify that all timing constraints have been met by looking at the Timing - Timing Summary window (**Figure 119**).

![Figure 119: Timing Summary](image-url)
Step 8: Exporting the Design to SDK

Next, open the design and export to SDK.

1. Select **File > Export > Export Hardware**.

2. In the Export to Hardware dialog box, select the **Include bitstream** check box (**FIGURE 120**). Make sure that the **Export to** field is set to `<Local to Project>`.

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>`.

5. Click **OK**.

SDK launches in a separate window.
Step 9: Creating a “Hello World” Application

1. In SDK, right-click `mb_subsystem_wrapper_hw_platform_0` in the Project Explorer and select **New > Project** (*FIGURE 122*).

![Figure 122: SDK New Project Selection](image)

2. In the New Project dialog box, select **Xilinx Application Project** (*FIGURE 123*).

![Figure 123: SDK New Project Wizard](image)
3. Click **Next**.

4. Type a name (**hello_world**) for your project and choose **standalone** as the OS platform (Figure 124).

![Figure 124: New Project: Application Project Wizard](image)

5. Click **Next**.
6. Select the **Hello World** application template, and click **Finish** *(FIGURE 125)*.

![Figure 125: New Project: Template Wizard](image)

SDK creates a new “Hello World” application.

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

The Generate Linker Script dialog box opens.
Step 9: Creating a “Hello World” Application

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

   To do this:
   
   a. Click in the **Assigned Memory** column in the appropriate row for heap and stack.
   
   b. Select **mig_7series_0** from the drop-down options.

   ![Figure 126: Generate Linker Script Dialog Box](image)

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

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

9. Likewise, change the Data Section Assignments and Code Section Assignments to **mig_7_series_0**.

10. Click **Generate**.

11. Click **Yes** to overwrite it.
Step 10: Executing the System on a KC705 Board

To run the design on a KC705 board:

1. Connect the board to your computer and switch on the board power.

2. Select the **Terminal 1** tab in SDK and click the **Settings** button.
   
   The Terminal Settings dialog box opens.

3. Specify the parameters as shown in **FIGURE 127**, and click **OK**.
   
   ![Terminal Settings Dialog Box](image)

   **Figure 127: Terminal Settings Dialog Box**

   The Terminal 1 tab displays confirmation that it is connected to the device (**FIGURE 128**).

   ![Terminal Connection Confirmation](image)

   **Figure 128: Terminal Connection Confirmation**

4. Start the XMD Console by selecting **Xilinx Tools > XMD Console**.
5. Program the bitstream on the board using the following command in the XMD Console.
   \[ \text{Xmd}\% \text{fpga -f } <\text{design path}>/<\text{project name}>/<\text{project name}>.runs/impl_1/<\text{ip_integrator_design name}>_wrapper.bit \]
   You can also program the FPGA by selecting \textbf{Xilinx Tools > Program FPGA}.

6. In the program FPGA dialog box, ensure the path to the bitstream is correct.

7. In the XMD Console, type: \textbf{XMD\% connect mb mdm}
   Then, type: \textbf{XMD\% mbc}

8. Reset and stop the MicroBlaze processor before running the software by using the \texttt{rst} and \texttt{stop} commands as shown:
   \[ \text{XMD}\% \text{rst} \]
   \[ \text{XMD}\% \text{stop} \]

9. Download the “Hello World” program ELF file of by typing:
   \[ \text{XMD}\% \text{dow } <\text{project path}>/<\text{project name}>/project_name.sdk/SDK/SDK.Export/hello_world/Debug/hello_world.elf \]

10. Run the program: \textbf{XMD\% run}
    The output displays in the Terminal tab as shown in \textbf{Figure 129}:

![Figure 129: Terminal Tab](image)

**Conclusion**

In this tutorial, you:

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

**Lab Files**

The \texttt{Tcl script lab4.tcl} is included with the design files to perform all the tasks in Vivado. You will also need the \texttt{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

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

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