Notice of Disclaimer

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, Vivado 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.

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>
<tbody>
<tr>
<td>06/19/2013</td>
<td>2013.2</td>
<td>Updated for Vivado Design Suite version 2013.2.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added a new lab demonstrating Zynq Cross Trigger capability.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added a new lab showing how to migrate EDK IP.</td>
</tr>
<tr>
<td>10/02/2013</td>
<td>2013.3</td>
<td>Updated tutorials for 2013.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added Connect BRAM Controller to Block Memory Generator, page 135.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added additional steps to Lab 4: Using the Embedded MicroBlaze Processor</td>
</tr>
<tr>
<td>02/07/2014</td>
<td>2013.3</td>
<td>Correction to link to design files .zip file on page 9.</td>
</tr>
</tbody>
</table>
# Table of Contents

Revision History......................................................................................................................................................... 2

Programming and Debugging Embedded Processors Using Vivado Design Suite........................................ 6

Introduction............................................................................................................................................................. 6

Software Requirements........................................................................................................................................ 6

Hardware Requirements........................................................................................................................................ 6

Tutorial Design Descriptions ................................................................................................................................ 7

Lab 1: Programming a Zynq-7000 Processor............................................................................................ 7

Lab 2: SDK and Logic Analyzer ....................................................................................................................... 7

Lab 3: Zynq Cross Trigger Design.................................................................................................................. 8

Lab 4: Programming a MicroBlaze Processor............................................................................................ 9

Lab 5: Migrating EDK IP to the Vivado Design Suite .............................................................................. 9

Locating Tutorial Design Files.............................................................................................................................. 9

Lab 1: Building a Zynq-7000 Processor Design................................................................................................ 10

Introduction............................................................................................................................................................. 10

Step 1: Start the Vivado IDE and Create a Project .................................................................................... 10

Step 2: Create an IP Integrator Design.......................................................................................................... 12

  Customize Instantiated IP.............................................................................................................................. 16

  Use Designer Assistance................................................................................................................................. 17

Step 3: Using MARK_DEBUG ............................................................................................................................. 21

Step 4: Generate HDL Design Files ................................................................................................................. 25

Step 5: Synthesize the Design........................................................................................................................... 27

Step 6: Assign Debug Net to an ILA Core ......................................................................................................... 29

Step 7: Implement Design and Generate Bitstream..................................................................................... 33

Step 8: Export Hardware to SDK ...................................................................................................................... 36

  Export to SDK..................................................................................................................................................... 36

Conclusion................................................................................................................................................................ 37

Lab Files................................................................................................................................................................ 37

Lab 2: Using SDK and the Vivado IDE Logic Analyzer.................................................................................... 38
Programming and Debugging Embedded Processors Using Vivado Design Suite

Introduction

This tutorial shows how to build a basic Zynq\textsuperscript{\textregistered}-7000 All Programmable (AP) SoC processor and a MicroBlaze\textsuperscript{\textregistered} processor design using the Vivado\textsuperscript{\textregistered} 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\textsuperscript{\textregistered} Software Development Kit (SDK) and the Vivado logic analyzer.

IMPORTANT: The Vivado IP integrator is the replacement for Xilinx Platform Studio (XPS) for embedded processor designs, including designs targeting Zynq-7000 devices and MicroBlaze\textsuperscript{\textregistered} processors. XPS only supports designs targeting MicroBlaze processors, not Zynq devices. Both IP integrator and XPS are available from the Vivado IDE.

Software Requirements

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

Hardware Requirements

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

Lab 1: Programming a Zynq-7000 Processor

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

- IP in the PL are:
  - 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 2013.3 (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 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.

Lab 2: SDK and Logic Analyzer

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

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

Hardware and Software Requirements

The following hardware and software are required for Lab 2.
Hardware Requirements:
  - Xilinx Zynq ZC702 board
  - One USB (Type A to Type B)
  - JTAG platform USB Cable or Digilent Cable
  - Power cable to the board

Software Requirements:
  - Vivado Design Suite 2013.3 (Embedded Version)

**Lab 3: Zynq Cross Trigger Design**

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

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

**Hardware and Software Requirements**

The following hardware and software are required for Lab 3.

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

Software Requirements:
  - Vivado Design Suite 2013.3 (Embedded Version)

**Required Design Files**

- `top_ref.xdc` - constraints file
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 3.

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 2013.3

Required Design Files

- freeRTOS folder that contains the operating system needed in SDK
- mig_7_series_pin_layout.ucf

Lab 5: Migrating EDK IP to the Vivado Design Suite

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

Software Requirements

- Vivado Design Suite 2013.3

Required Design Files

- axi_gpio_v1_01_b

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=343198.
Lab 1: Building a Zynq-7000 Processor Design

Introduction

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

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

Step 1: Start the Vivado IDE and Create a Project

1. Start the Vivado IDE (FIGURE 1) by clicking the Vivado desktop icon or by typing vivado at a terminal command line.

![Figure 1: Vivado Getting Started Page](image)

2. From the Getting Started page, select Create New Project.
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 the project name and location. 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 VHDL, 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, select Boards and choose ZYNQ-7 ZC702 Evaluation Board. Make sure that you have selected the proper Board Version to match your hardware, because multiple versions of hardware are supported in the Vivado IDE. Click Next.

   **CAUTION!** Multiple versions of boards are supported in Vivado. Ensure that you are targeting the design to the right hardware. The versions of the boards supported can be seen by setting the pull-down menu for Version to All.

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, select **Create Block Design**.

![Create Block Design from Flow Navigator](image)

**Figure 3: Create Block Design from Flow Navigator**

2. In the **Create Block Design** dialog box, specify a name for your IP subsystem design.

![Create Block Design Dialog Box](image)

**Figure 4: Create Block Design Dialog Box**
3. Right-click in the **Diagram** panel of the Vivado IP integrator window, and select **Add IP**.

![Add IP Option](image)

**Figure 5: Add IP Option**

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

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

**Figure 6: Add IP Link in IP Integrator Canvas**

The IP Catalog opens.
4. In the search field, type **zynq** to find the ZYNQ7 Processing System IP

![Figure 7: The IP Integrator IP Catalog](image)

5. Press **Enter** on the keyboard to add the ZYNQ7 Processing System IP to your design.

Because you selected the ZC702 board when you created the project, the Vivado IP integrator configures the design appropriately.

In the Tcl Console, you see the following message:

```tcl
create_bd_cell -type ip -vlnv xilinx.com:ip:processing_system7:5.3 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).

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

![Figure 8: Run Block Automation on Zynq](image)
The **Run Block Automation** dialog box opens, stating that the FIXED_IO and DDR interfaces will be created for the Zynq core. Also, note that the Apply Board Preset checkbox is checked. This is because the selected target board is ZC702. Click **OK**.

![Run Block Automation Dialog Box](image)

**Figure 9: Zynq-7 Run Block Automation Dialog Box**

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

![IP Integrator Diagram](image)

**Figure 10: Zynq Processing System After Running Block Automation**

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

8. In the search field, type `gpi` to find the AXI GPIO IP, and then press **Enter** to add the AXI GPIO IP to the design. Repeating the action, add the AXI BRAM Controller and Block Memory Generator.
Your Block Design window is similar to Figure 11. 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 11: Block Design After Instantiating IP](image)

**Customize Instantiated IP**

1. Double-click the Block Memory Generator IP, or right-click and select **Customize Block** (Figure 12).

![Figure 12: Customize Block Option](image)

The Re-customize IP dialog box opens.
2. On the Basic tab of the dialog box, set:
   - **Mode** to **BRAM Controller**
   - **Memory Type** to **True Dual Port RAM**

   ![Component Name](Component Name)

   **Figure 13: Set Mode and Memory Type**

3. Click **OK**.

**Use Designer Assistance**

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

1. Click **Run Connection Automation** and then select `/axi_gpio_1/s_axi` to connect the S_AXI interface of the GPIO IP to the Zynq PS. *(FIGURE 14)*

   ![Diagram](Diagram)

   **Figure 14: Run Connection Automation**

   The Run Connection Automation dialog box opens and states that it will connect the master AXI interface to a slave interface.
In this case, the master is the Zynq Processing System IP (FIGURE 15).

![Figure 15: Run Connection Automation Message](image)

2. Click **OK**.

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

3. Select **Run Connection Automation** again, and select the `/axi_gpio_1/gpio` port as shown in **FIGURE 16**.

![Figure 16: axi_gpio Selection](image)

The Run Connection Automation dialog box includes options to hook up to the GPIO port.
4. Select LEDs_4Bits (FIGURE 17).

![Run Connection Automation](image)

**Figure 17: Select Board Interface Options**

5. Click **OK**.

This step also configures the IP so that during netlist generation, the IP creates the necessary Xilinx Design Constraints (XDC).

6. Click **Run Connection Automation** again, and select the option `/axi_bram_ctrl_0/S_AXI` (FIGURE 18).

![axi_bram_ctrl Selection](image)

**Figure 18: axi_bram_ctrl Selection**

In the Run Connection Automation dialog box, click **OK** to connect the axi_bram_ctrl_0/S_AXI interface to the Master address space.

This completes the connection between the Zynq7 Processing System and the AXI BRAM Controller.
7. Select the other two remaining options for running Connection automation. These automations will connect the AXI BRAM controller to the two ports of the Block Memory Generator.

![Figure 19: Connecting the Block Memory Generator to AXI BRAM Controller](image1)

When you choose these options, the Run Connection Automation dialog box asks for confirmation to connect the AXI BRAM Controller to either the existing Block Memory Generator or instantiate a new Block Memory Generator. Select the existing Block Memory Generator /blk_mem_gen_0.

![Figure 20: Select Existing Block Memory Generator](image2)

The IP integrator subsystem looks like ![Figure 21: Zynq Processor System](image3). Again, the relative positions of the IP can differ slightly.
8. Click the **Address Editor** tab 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 22**.

![Figure 22: Change Range of axi_bram_ctrl to 64K](image_url)

---

**Step 3: Using MARK_DEBUG**

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

1. To debug the interface between the AXI Interconnect IP named `processing_system7_0_axi_periph` and the GPIO core (named `axi_gpio_1`), in the Diagram view, select the interface, then right-click and select **Mark Debug** (**FIGURE 23**).

![Figure 23: Mark Debug Option](image_url)
2. In the Block Design canvas on the net that you selected in the previous step, a small bug icon appears to indicate 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 24).

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

Figure 24: Design Hierarchy: Icon for Mark Debug

3. Right-click anywhere in the IP integrator diagram and add a title and/or comment to the design using Create Comment (FIGURE 25).

![Create Comment Option](image)

Figure 25: Create Comment Option
4. Add a title to your design, and then right-click and format the comment.

![Format Comment Menu Item](image)

Figure 26: Format Comment Menu Item

5. The Format Comment dialog box opens. Select the appropriate options and click OK.

![Format Comment Dialog Box](image)

Figure 27: Format Comment Dialog Box

6. Save your design by pressing Ctrl+S, or select File > Save Block Design.
Step 3: Using MARK_DEBUG

7. From the toolbar, run Design-Rules-Check (DRC) by clicking the **Validate Design** button (FIGURE 28). 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**.
   - Clicking the **Validate Design** button in the IP integrator toolbar.

![Validate Design Button](image)

**Figure 28: Validate Design Button**

The Validate Design Successful dialog box opens (FIGURE 29).

![Validate Design Message](image)

**Figure 29: Validate Design Message**

8. Click **OK**.
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 (FIGURE 30). This generates the source files for the IP used in the block design and the relevant constraints file.

![Figure 30: Generate Output Products Option](image)

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

![Figure 31: Generate Block Design Button](image)
2. The **Generate Output Products** dialog box opens. Click **Generate**.

![Generate Output Products Dialog Box](image)

**Figure 32: Manage Output Products Dialog Box**

3. In the Sources window, select the top-level subsystem source, and select **Create HDL Wrapper** to create an example top level HDL file (**Figure 33**).

![Create HDL Wrapper](image)

**Figure 33: Create HDL Wrapper**

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

![Create HDL Wrapper Dialog Box](image)

**Figure 34: Create HDL Wrapper Dialog Box**

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

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

**Figure 35: Source Window After Creating the Wrapper**

---

**Step 5: Synthesize the Design**

Now assign the signals to debug in the hardware.

1. After generating the IP Integrator design, from the **Flow Navigator** click **Run Synthesis** (FIGURE 36).

![Run Synthesis](image)

**Figure 36: Run Synthesis**

*Note: Running synthesis could take several minutes.*
2. After synthesis completes, in the **Synthesis Completed** dialog box, check the **Open Synthesized Design** option, and click **OK**.

![Synthesis Completed Dialog Box](image)

**Figure 37: Synthesis Completed Dialog Box**

3. The Debug window opens.

   **Note**: You can also open this window by selecting **Window > Debug** from the menu.

   In the Debug window, you see 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 38**).

![Unassigned Debug Nets Folder](image)

**Figure 38: Unassigned Debug Nets Folder**

Next, you assign the debug nets to an Integrated Logic Analyzer (ILA) debug core.
Step 6: Assign Debug Net to an ILA Core

1. On the left-hand toolbar of the Debug window, click the **Set up Debug** wizard button (FIGURE 39).

2. When the Set up Debug wizard opens, click **Next** (FIGURE 40).
On the second panel of the wizard, notice that some of the nets in the table do not belong to a clock domain (FIGURE 41).

**Figure 41: Set Up Debug**

**IMPORTANT:** Signals captured by the same ILA core must have the same clock domain selection.
3. Highlight the three signals that don’t have the clock domain association, then right-click and select **Select Clock Domain** (*FIGURE 42*).
4. In the Select Clock Domain window, select the `zynq_design_1_i/FCLK_CLK0` net as the clock domain, and click **OK** (FIGURE 43).

![Select Clock Domain Option](image)

**Figure 43: Select Clock Domain Option**

5. In the Set up Debug dialog box, click **Next** to bring up the Trigger and Capture Modes dialog box.

6. Select both the **Enable advanced trigger mode** and **Enable basic capture mode** check boxes.
7. Click Next.

![Figure 44: Setting Trigger and Capture Modes](image)

8. The Set up Debug Summary page appears. 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, 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**.

   **Note:** If the system requests to re-synthesize the design before implementing, click No.

   The previous step of saving the constraints caused the flow to mark synthesis out-of-date.
Ordinarily, you might want to re-synthesize the design if you manually changed the constraints, but for this tutorial, it is safe to ignore this condition (FIGURE 45).

Figure 45: Generate Bitstream

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

This step is required to synthesize the debug core modules so that they can replace the debug core black boxes that you added to the design previously (FIGURE 47).

Figure 47: 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. In the Bitstream Generation Completed dialog box that opens, click **Open Implemented Design**, (FIGURE 48).

   ![Bitstream Generation Completed](image)

   **Figure 48: Bitstream Generation Completed**

5. You can keep the synthesized design open if you want to debug more signals; otherwise close the synthesized design to save memory (FIGURE 49).

   ![Close Synthesized Design Dialog Box](image)

   **Figure 49: Close Synthesized Design Dialog Box**

6. You might get a warning that the implementation is out of date. Click **Yes**.

   ![Implementation Is Out-of-date Dialog Box](image)

   **Figure 50: Implementation Is Out-of-date Dialog Box**
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 51**).

![Figure 51: Implemented Design](image)

---

**Step 8: Export Hardware to SDK**

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

The IP integrator block design, and the Implemented design, must be open to export the design to SDK.

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

---

**Export to SDK**

1. From the main Vivado File menu, select Export Hardware for SDK (**FIGURE 52**).

![Figure 52: Export Hardware for SDK](image)

The Export Hardware for SDK dialog box opens.
If you want to go on to Lab 2 then ensure that Export Hardware, Include Bitstream, and Launch SDK are checked (Figure 53). Otherwise, you can leave the Launch SDK option unchecked.

![Figure 53: Export Hardware for SDK](image)

**Conclusion**

In this lab, you:

- Created a Vivado project that includes a Zynq processor design using the IP integrator tool.
- Instantiated IP in the IP integrator tool and made the necessary connections utilizing the Design Automation and Connection Automation features.
- Inserted debug probes in the design to debug it later in the Vivado 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 correctly source the Tcl files.
Lab 2: Using SDK and the Vivado IDE Logic Analyzer

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 (if you checked the Launch SDK option while exporting hardware). You can also start SDK from the Windows Start menu by clicking on **Start > All Programs > Xilinx Design Tools > Vivado 2013.3 > SDK > Xilinx SDK 2013.3**.

   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 peripheral test application.

3. Select **File > New > Application Project** (**FIGURE 54**).

   ![Figure 54: File > New > Application Project](image)

   The New Project dialog box opens.
4. In the Project Name field, type **Zynq_Design**, and click **Next** (FIGURE 55).

![Figure 55: SDK Application Project](image)

5. From the Available Templates, select **Peripheral Test** (FIGURE 56) and click **Finish**.

![Figure 56: SDK New Project Template](image)
When the program finishes compiling, you see the following (FIGURE 57).

![Figure 57: 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. Select **Xilinx Tools > Configure JTAG Settings** (FIGURE 58).

![Figure 58: Configure JTAG Settings](image)
2. In the Configure JTAG Settings dialog box, select the **Type** as **Auto Detect**, and click **OK** *(FIGURE 59).*

![Figure 59: Configure JTAG Settings](image)

3. Next, download the bitstream into the FPGA by selecting Xilinx **Tools > Program FPGA** *(FIGURE 60).*

![Figure 60: Program FPGA](image)

The Program FPGA dialog box opens.
4. Ensure that the path to the bitstream that you created in Step 7 of Lab 1 is correct and then click **Program**.

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

5. Select and right-click the **Zynq Design** application.

6. Select **Debug As** and **Debug Configurations** ([FIGURE 61](#)).

![Figure 61: Launch on Hardware](image-url)
7. In the Debug Configurations dialog box, right-click **Xilinx C/C++ Application (GDB)** and select **New**.

![Debug Configurations Dialog Box](image1)

Figure 62: Debug Configurations Dialog Box

8. In the Debug Configurations dialog box, click **Debug**.

![Run Debug Configurations](image2)

Figure 63: Run Debug Configurations
9. The Confirm Perspective Switch dialog box opens. Click **Yes**.

![Confirm Perspective Switch Dialog Box]

**Figure 64: Confirm Perspective Switch Dialog Box**

10. Set the terminal by selecting the Terminal 1 tab and clicking the Settings button (**Figure 65**).

![Settings Button]

**Figure 65: Settings Button**
11. Use the following settings for the ZC702 board (Figure 66). Click OK.

![Terminal Settings for ZC702 Board](image)

**Figure 66: Terminal Settings for ZC702 Board**

12. Verify the **Terminal** connection by checking the status at the top of the tab (Figure 67).

![Terminal Connection Verification](image)

**Figure 67: Terminal Connection Verification**
13. In the **Debug** tab, expand the tree, and select the processor core on which the program is to be run (**FIGURE 68**).

![Figure 68: Processor Core to Debug](image)

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

**Add a Breakpoint**

You add a breakpoint on line 104.

1. Select **Navigate > Go To Line** (**FIGURE 69**).

![Figure 69: Go to Line](image)

2. In the Go To Line dialog box, type **108**.
3. Double click on the left pane of line 104, which adds a breakpoint on that line of source code (Figure 70).

![Figure 70: Add a Breakpoint]

---

**Step 3: Connect to the Vivado Logic Analyzer**

Connect to the ZC702 board using the Vivado logic analyzer.

1. Open the Vivado project from Lab 1 if not already open. Select **Vivado Flow Navigator > Program and Debug > Open Hardware Manager** (Figure 71).

![Figure 71: Open Hardware Session]

2. In the Hardware Manager window, click **Open a new hardware target** to open a connection to the Digilent JTAG cable for ZC702 (Figure 72).

![Figure 72: Launch Open New Hardware Target Wizard]

The Open New Hardware Target dialog box opens.
3. Click **Next**.

   The server name should be already populated by default to **localhost:60001** (Figure 73).

![Figure 73: Open New Hardware Target](image)

4. Click **Next**.

   You now see the **xilinx_tcf** hardware target and the **ARM_DAP_0** and **XC7Z020_1** hardware devices as shown in **FIGURE 74**.

![Figure 74: Select Target Hardware](image)
5. Click **Next** again to invoke the next dialog box (FIGURE 75).

![Figure 75: Type and Device](image)

6. Click **Next** again.

The Open Hardware Target Summary dialog box is displayed (FIGURE 76).

![Figure 76: Open Hardware Target Summary](image)
7. Click **Finish** to connect to the target.

When the Vivado hardware session successfully connects to the ZC702 board, you see the information, shown in **Figure 77**.

![Figure 77: Successfully Programmed Hardware Session](image)

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

9. On the Hardware window toolbar, click the **Run Trigger Immediate** button, as shown in **Figure 78**.

![Figure 78: Run Trigger Immediate on hw_ila_1](image)
At this point, you should see static data from the ILA core in the waveform window (Figure 79). The data is static is because the processor is halted at the breakpoint you set earlier.

Figure 79: Static Data from Set Breakpoint

10. If the Debug Probes window is not already open, select **Tools > Debug Probes**.
This opens the Debug Probes window as shown below.

![Debug Probes window](image1)

**Figure 81: Debug Probes window**

11. 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 in the ILA Properties window.

![ILA Properties window](image2)

**Figure 82: The ILA Properties window**
Step 3: Connect to the Vivado Logic Analyzer

b. Click the **Compare Value** column and the **WVALID** row, as shown in (FIGURE 83).

![Figure 83: Debug Probe View](image)

Figure 83: Debug Probe View

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

12. 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 (FIGURE 84).

![Figure 84: Change Debug Probe Settings](image)

Figure 84: 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 (FIGURE 85).

![Figure 85: Run Trigger](image)

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 86.

![Figure 86: Status of hw_ila_1](image)

14. Go back to SDK and continue to execute the code after you hit the breakpoint.

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 (FIGURE 87).

![Figure 87: Step Over Write Operation](image)
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 88)*.

![Figure 88: Trigger Mark Goes to 1](image)

## Conclusion

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

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

**Note:** Tcl files are not provided for this lab as the intent is for the user to see the power of cross-probing between the hardware and software in the GUI environment.
Chapter 4

Lab 3: Zynq Cross-Trigger Design

Introduction

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

Step 1: Start the Vivado IDE and Create a Project

1. Start the Vivado IDE by clicking the Vivado desktop icon or by typing `vivado` at a command prompt.
2. From the Getting Started page, select Create New Project.
3. In the New Project dialog box, use the following settings:
   d. In the Project Name dialog box, type the project name and location. For this project, use the name zynq_x_trigger.
   e. Make sure that the Create project subdirectory check box is checked. Click Next.
   f. In the Project Type dialog box, select RTL project. Click Next.
   g. In the Add Sources dialog box, ensure that the Target language is set to VHDL. 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.

In the Create Block Design dialog box, specify the name of the IP integrator design. For this tutorial, use zynq_processor_system. The IP integrator diagram opens.

2. Click Add IP at the top of the diagram area (FIGURE 89).

![Figure 89: Add IP to the Design](image)

The IP catalog opens.

3. 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 90).

![Figure 90: Instantiate Zynq Processing System](image)

4. In the header at the top of the diagram, click Run Block Automation and select /processing_system7_1 (FIGURE 91).

![Figure 91: 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. Click **OK** (FIGURE 92).

![Figure 92: Run Block Automation Dialog Box](image)

5. 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 93).

![Figure 93: Diagram after Instantiating IP for This Design](image)
6. Click **Run Connection Automation** and select `/axi_gpio_1/S_AXI` *(Figure 94)*.

![Figure 94: Run Connection Automation on /axi_gpio_1/s_axi](image)

The Run Connection Automation dialog box states that the S_AXI pin of the GPIO IP will be connected to M_AXI_GP0 pin of the ZYNQ7 Processing System.

7. Click **OK** *(Figure 95)*.

![Figure 95: Run Connection Automation Dialog Box for /axi_gpio_1/S_AXI](image)

8. Click **Run Connection Automation** again and select `/axi_gpio_0/GPIO` *(Figure 98)*

![Figure 96: Run Connection Automation on /axi_gpio_0/S_AXI](image)
Step 2: Create an IP Integrator Design

The Run Connection Automation dialog box shows the interfaces that are available on the ZC702 board to connect to the GPIO. Select LEDs_4Bits and click OK.

![Select LEDs_4Bits to Connect to the GPIO](image)

9. Next, click Run Connection Automation again and select /axi_bram_ctrl_1/S_AXI (Figure 98).

![Run Connection Automation on /axi_bram_ctrl_1/S_AXI](image)

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. Click OK (Figure 99).

![Run Connection Automation Dialog Box for /processing_syste7_1/M_AXI_GP0](image)
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.

10. Click **Run Connection Automation** and select `/axi_bram_ctrl_0/BRAM_PORTA`.

![Figure 100: Run Connection Automation on AXI BRAM Controller PORTA](image)

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.

11. Click **OK**.

![Figure 101: Run Connection Automation Dialog Box](image)

12. Repeat the above step for PORT B.

![Figure 102: Run Connection Automation on AXI BRAM Controller Port B](image)

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, so select `/axi_bram_ctrl_0_bram` and click **OK**.
13. The IP integrator design should look as follows (FIGURE 104).

![Figure 104: Design after Running Connection Automation](image)

14. Click **Regenerate Layout** to generate an optimal layout of the design (FIGURE 105).

![Figure 105: Regenerate Layout](image)
The design should look as follows (FIGURE 106).

15. 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 107).

16. Now we will mark some nets for debug. Click the Diagram tab again and select the net connecting the gpio pin of the AXI GPIO IP and the LEDs_4Bits port.
17. Right-click in the IP integrator diagram area and select **Mark Debug** ([Figure 108](#)). This marks the net for debug using the Integrated Logic Analyzer.

![Figure 108: Mark axi_gpio_1_gpio Net for Debug](#)

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.

18. Similarly, highlight the net connected between S_AXI interface of AXI GPIO and M00_AXI interface of the AXI Interconnect (`processing_system7_0_axi_periph`) of the AXI GPIO, right-click, and select **Mark Debug**.

19. Click **Validate Design** to run Design Rule Checks on the design ([Figure 109](#)).

![Figure 109: Run Validate Design](#)
After design validation is complete, the **Validate Design** dialog box opens (FIGURE 110).

![Validate Design Dialog Box](image)

**Figure 110: Validate Design Dialog Box**

20. Save the IP integrator design by clicking on the **File** menu and selecting **Save Block Design** (FIGURE 111).

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

![Save Block Design](image)

**Figure 111: Save Block Design**
21. In the Sources window, right-click zynq_processor_system and select **Generate Output Products** *(FIGURE 112).*

![Figure 112: Generate Output Products](image)

The Generate Output Products dialog box opens.

22. Click **Generate** *(FIGURE 113).*

![Figure 113: Manage Output Products Dialog Box](image)
23. In the Sources window, right-click `zynq_processor_system`, and select **Create HDL Wrapper** (Figure 114).

![Create HDL Wrapper](image)

**Figure 114: 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 115).

![Create HDL Wrapper Dialog Box](image)

**Figure 115: Create HDL Wrapper Dialog Box**
Step 3: Synthesize Design

1. From the Flow Navigator, click Synthesis > Run Synthesis (FIGURE 116).

![Figure 116: Synthesize Design](image1)

2. When synthesis completes, select Open Synthesized Design from the Synthesis Completed dialog box and click OK (Figure 117).

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

![Figure 117: Open Synthesized Design](image2)
Step 4: Create an Integrated Logic Analyzer Debug Core

1. From the Vivado main menu, select **Tools > Setup Debug** (FIGURE 118).

![Figure 118: Setup Debug Core](image)

Alternatively, from the left side of the **Debug** window, click the **Set up Debug** button (FIGURE 119).

![Figure 119: Setting Debug Core from the Debug Window](image)
Step 4: Create an Integrated Logic Analyzer Debug Core

The Set up Debug dialog box opens (Figure 120).

2. Click Next.

![Set up Debug Dialog Box](image)

Figure 120: Set Up Debug Dialog Box

3. In the Specify Nets to Debug page, make sure that all the nets selected for debug have a Clock Domain associated with them.

Here, some nets do not have a clock domain associated with them as can be seen in the Clock Domain column (marked as undefined). (Figure 121).
4. To assign the clock domains to these nets, select and highlight the three nets that do not have a clock domain associated with them, right-click, and click **Select Clock Domain** (Figure 122).

![Figure 122: Select Clock Domain](image-url)
5. In the **Select Clock Domain** dialog box, select `zynq_processor_system_i/FCLK_CLK0` as shown (FIGURE 123). If the clock name is different in your design, select the same clock as that specified for all the other nets. Click **OK**.

![Select Clock Domain Dialog Box](image)

**Figure 123: Select Clock Domain Dialog Box**

6. Click **Next** in the Specify Nets to Debug page.
7. In the Trigger and Capture Modes page, select both **Enable advanced trigger mode** and **Enable basic capture mode** check boxes.

8. Click **Next**.

![Figure 124: Enable Advanced Trigger and Capture Modes](image)

9. Click **Finish** in the Set up Debug Summary dialog box (**Figure 125**).

![Figure 125: Set Up Debug Summary](image)
Step 5: Create Debug Ports and Probes in the ILA Core

The debug nets are now assigned to the ILA v2.1 debug core, as shown below (Figure 126).

![Debug Nets Assigned to Debug Core](image)

**Figure 126: Debug Nets Assigned to Debug Core**

**Step 5: Create Debug Ports and Probes in the ILA Core**

1. In the Debug window, click the **Debug Cores** tab (Figure 127).

![The Debug Cores Tab](image)

**Figure 127: The Debug Cores Tab**
2. Select `u ila_0 (labtools ila_v3)`, right-click it, and select **Create Debug Port** (FIGURE 128).

![Create Debug Ports](image1.png)

**Figure 128: Create Debug Ports**

Alternatively, you can also click the **Create Debug Port** button (FIGURE 129).

![Create Debug Port Button](image2.png)

**Figure 129: Create Debug Port Button**

In the Create Debug Port dialog box, `trig_in` is selected in the **Type** field by default (FIGURE 130).

3. Click **OK**.

![Create TRIG_IN Debug Port](image3.png)

**Figure 130: Create TRIG_IN Debug Port**
4. Repeat the previous step to include `trig_in_ack`, `trig_out`, and `trig_out_ack` from the pull-down menu in the **Type** field.

5. Next, create four debug probes to which these pins will be connected by clicking on **u ila_0 (labtools ila_v3)** core and selecting **Create Debug Port** (FIGURE 131).

![Figure 131: Create Debug Probes](image)

In the **Create Debug Port** dialog box **Type** field, **PROBE** is selected by default (FIGURE 132).

6. Click **OK**.

![Figure 132: Create Debug Port Dialog Box](image)

7. Repeat the previous steps three more times to create three more probes.
8. After the debug ports and probes have been created, the Debug window should look as follows (FIGURE 133). Look at the highlighted signals to make sure that all have been added correctly.

![Figure 133: New Debug Ports in the ILA Core](image)

*Figure 133: New Debug Ports in the ILA Core*

You can also verify that the debug ports and probes have been added properly in the netlist schematic.
9. In the Netlist window, select `u_ila_0 (u_ila_0.CV)`, right-click and select Schematic (Figure 134).

Figure 134: View Schematic to Verify Changes
The Schematic window opens, showing the newly created ports and probes in the ILA core (FIGURE 135).

![Figure 135: ILA Core Schematic](image)

Step 6: Connect Zynq Cross Trigger Signals to the ILA Core

Like those created in the ILA core in Step 5, the PS7 primitive includes a set of cross-trigger pins:

- FTMT_F2P_TRIG
- FTMT_F2P_TRIGACK
- FTMT_P2F_TRIG
- FTMT_P2F_TRIGACK

You can see these pins in the Netlist window by expanding the hierarchy as shown (FIGURE 136).

![Netlist window showing Zynq Cross-Trigger Pins](image-url)
For cross triggering to work correctly, these pins must be connected to the corresponding sets of pins of the ILA core created in Step 5.

However, the input pins FTMT_F2P_TRIG and FTMT_P2F_TRIGACK are connected to ground after synthesis as shown in the following figure (FIGURE 137).

These pins must be disconnected and reconnected to the corresponding pins of the ILA core. You cannot perform this action in the GUI; you must use Tcl scripts.

The following procedure describes how to use a Tcl script that removes the PS7 instance and disconnects the FTMT_F2P_TRIG and FTMT_P2F_TRIGACK input pins from the PS7 primitive. It then connects the two input pins FTMT_F2P_TRIG and FTMT_P2F_TRIGACK and the two output pins FTMT_F2P_TRIGACK and FTMT_P2F_TRIG to the trig_in, trig_out_ack, trig_in_ack, and trig_out pins of the ILA, respectively.

![Figure 137: Zynq Cross-Trigger Inputs Connected to Ground](image)

From the Vivado Tcl Console, run the Tcl file supplied with this tutorial.

1. In the Tcl Console, type:

   ```tcl
   source <path_to_tcl_file>/connectZynqCrossTrigger.tcl
   ```

2. Verify that the Tcl file makes the appropriate connections between the PS7 primitive and ILA core by looking at the connectivity between the ILA core and PS7 primitive in the Schematic window.
The recently connected cross-trigger pins \texttt{trig\_in}, \texttt{trig\_out\_ack}, \texttt{trig\_in\_ack}, and \texttt{trig\_out} must be connected to the probes created in \textbf{Step 4}: \texttt{PROBE18}, \texttt{PROBE19}, \texttt{PROBE20}, and \texttt{PROBE21}.

To do this:

a. In the Netlist window, select the \texttt{u\_ila\_0 (u\_ila\_0\_CV)} instance.

b. Right-click the instance and select \textbf{Schematic}.

The Schematic window opens with the ILA instance, and the \texttt{PROBE18}, \texttt{PROBE19}, \texttt{PROBE20} and \texttt{PROBE21} can be seen as no-connects (\textit{FIGURE 138}).
3. Click the `trig_in` pin of the `uila_0` instance and select **Assign to Debug Port** (Figure 139).

![Figure 139: Assign Cross-Trigger Pins to Debug Probes](image)

4. In the **Assign to Debug Port** dialog box, select `PROBE18`, and click **OK** (Figure 140).

![Figure 140: Connect TRIG_IN to PROBE18](image)
5. Repeat the previous step to connect trig_out_ack, trig_in_ack, and trig_out to PROBE19, PROBE20, and PROBE21 respectively.

6. See the changes made to the netlist by selecting the u ila_0 instance and pressing the F4 key. This opens a new schematic window with the connections as shown (FIGURE 141). You need to double click on the ports to see the connections.

![Figure 141: Verify Connections in Schematic](image)

### Step 7: Implement Design and Generate Bitstream

Now that the cross-trigger functionality is implemented, you can complete the design through the rest of the flow.

1. Click **Generate Bitstream** to generate the bitstream for the design. The **Save Project** dialog box opens with a message asking whether the project should be saved at this point. Click **Save**.

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

3. When bitstream generation completes, the **Bitstream Generation Completed** dialog box opens, with the option **Open Implemented Design** option checked by default. Click **OK** to open the implemented design.
4. If you see a message box that asks about closing Synthesized Design before opening Implemented Design, click Yes (Figure 142).

![Figure 142: Pop-Up Option for Closing Synthesized Design](image)

5. You might see the **Implementation is Out-of-date** dialog box stating that the implemented design is out of date (Figure 143). This occurs if you manipulate the netlist and save the constraints. However, nothing has changed in the design, so it is safe to open.

   Click **Yes** to open the implemented design.

![Figure 143: Implementation Out-of-Date Message](image)

6. From **Flow Navigator > Implementation**, click **Implemented Design** (Figure 144).

![Figure 144: Open Implemented Design Folder](image)
7. From the expanded selection, select **Report Timing Summary** to see if the constraints are met (FIGURE 145).

![Figure 145: Run Report Timing Summary](image)

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

9. Ensure that all timing constraints are met by looking at the Timing tab (FIGURE 146).

![Figure 146: Timing Summary](image)
Step 8: Export Hardware to SDK

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

1. Make sure that the `zynq_processor_system` block design is open before you export the hardware.

2. Click File and select Export Hardware for SDK (FIGURE 147).

![Figure 147: Export Hardware for SDK](image-url)
3. In the **Export Hardware for SDK** dialog box, make sure all three options are checked as shown (FIGURE 148).

![Figure 148: Export Hardware for SDK Dialog Box](image)

**Step 9: Build Application Code in SDK**

1. SDK launches in a separate window. Click **File**, select **New**, and then select **Application Project** (FIGURE 149).

![Figure 149: Create a New Application Project](image)
2. In the **New Project** dialog box, specify the name for your project. For this tutorial, use the name `peri_test` (**FIGURE 150**). Click **Next**.
3. From the **Available Templates**, select **Peripheral Tests** (FIGURE 151) and click **Finish**.

![Figure 151: Select the Peripheral Tests Template](image)

4. To configure the JTAG port, click **Configure JTAG Settings** (FIGURE 152).

![Figure 152: Configure JTAG Settings](image)
5. Leave the Type as **Auto Detect** and click **OK** (FIGURE 153).

![Figure 153: Configure JTAG Settings Dialog Box](image)

---

**XILINX**

Step 9: Build Application Code in SDK

Embedded Processor Hardware Design

UG940 (v 2013.3) February 7, 2014

www.xilinx.com
6. Click Xilinx Tools and select Program FPGA to open the Program FPGA dialog box (FIGURE 154).

![Figure 154: Program FPGA](image)

7. In the Program FPGA dialog box, make sure that the path to the bitstream is specified correctly (FIGURE 155) and click Program.

![Figure 155: Program FPGA Dialog Box](image)
8. Select and right-click the **peri_test** application, select **Debug As**, and then select **Debug Configurations** (Figure 156).

![Figure 156: Debug Configurations](image)

The Debug Configurations dialog box opens.
9. Right-click **Xilinx C/C++ application (GDB)**, and select **New** *(FIGURE 157).*

![Figure 157: Debug Configurations Dialog Box](image)

10. In the **Create, manage, and run configurations** screen, select the **Debugger Options** tab and check the **Enable Cross triggering** check box, and then click **Debug** *(FIGURE 158).*

![Figure 158: Enable Cross-Triggering](image)
Step 9: Build Application Code in SDK

The Confirm Perspective Switch dialog box opens.

11. Click **Yes** *(FIGURE 159)*.

![Confirm Perspective Switch](image)

**Figure 159: Confirm Perspective Switch**

The Debug perspective window opens.

12. Set the terminal by selecting the **Terminal 1** tab and clicking the **Settings** button *(FIGURE 160)*.

![Set the Terminal Settings](image)

**Figure 160: Set the Terminal Settings**
13. Use the following settings for the ZC702 board and click **OK** (**FIGURE 161**).

![Terminal Settings](image)

**Figure 161: Terminal Settings**

14. Verify the Terminal connection by checking the status at the top of the tab (**FIGURE 162**).

![Verify Terminal Connection](image)

**Figure 162: Verify Terminal Connection**

15. If it is not already open, select `./src/testperiph.c`, and double click to open the source file.
16. Modify the source file by inserting a while statement at line 101. Click the left side of the `testperiph.c` window as shown in the figure and select Show Line Numbers (FIGURE 163).

![Figure 163: Show Line Numbers](image)

17. In line 102, add `while(1)` in front of the curly brace as shown (FIGURE 164).

![Figure 164: Modify testperiph.c](image)

18. Add a breakpoint in the code so that the processor stops code execution when the breakpoint is encountered.
19. Scroll down to line 106 and double click on the left pane of line 104, which adds a breakpoint on that line of code (*FIGURE 165*). Click 
\textbf{Ctrl + S} to save the file. Alternatively, you can select \textbf{File > Save} from the menu.

At the same time, the Vivado Integrated Logic Analyzer should also trigger demonstrating Processor to Programmable Logic cross-trigger functionality.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure165.png}
\caption{Set a Breakpoint}
\end{figure}

Now you are ready to execute the code from SDK.

\section*{Step 10: Connect to Vivado Logic Analyzer}

Connect to the ZC702 board using the Vivado Logic Analyzer.

1. In the Vivado IDE session, from the \textbf{Program and Debug} folder of the Vivado Flow Navigator, select \textbf{Open Hardware Manager} (*FIGURE 166*).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure166.png}
\caption{Open Hardware Session in Vivado}
\end{figure}
2. In the Hardware Manager window, click **Open a new hardware target** (Figure 167).

   ![Figure 167: Open a New Hardware Target](image)

3. The Open New Hardware Target dialog box opens (Figure 168). Click **Next**.

   ![Figure 168: Open Hardware Target](image)
4. On the Vivado CSE Server Name screen, ensure that the Server name `<host[:port]>` field is set to `localhost:60001` (FIGURE 169). Click Next.

![Figure 169: Specify Server Name and Port](image)

5. On the Select Hardware Target screen (FIGURE 170), click Next.

![Figure 170: Select Hardware Target](image)
6. On the Set Hardware Target Properties screen (Figure 171), click **Next**.

![Figure 171: Set Hardware Target Properties](image)

7. Ensure that all the settings are correct on the Open Hardware Target Summary dialog box (Figure 172) and click **Finish**.

![Figure 172: Open Hardware Target Summary](image)
8. When the Vivado Hardware Session successfully connects to the ZC702 board, you see the information shown (Figure 173).

Figure 173: Vivado Hardware Session Window

9. If the Debug Probes window is not open already, select Window > Debug Probes.

10. In the Debug Probes window, find the trig_in signal, select it and drag and drop it into the ILA Properties window in the Basic Trigger Setup pane.

11. Click the Compare Value column for TRIG_IN. In the pop-up box, enter a value of 1 (Figure 174). Click OK.

Figure 174: Set Trigger

12. Arm the ILA core by selecting hw_ila_1 in the Hardware window and clicking on Run Trigger (Figure 175).

Figure 175: Arm the ILA Core
Step 10: Connect to Vivado Logic Analyzer

This arms the ILA and you should see the status “Waiting for Trigger” as shown (Figure 176).

![Figure 176: Armed ILA Core](image)

Notice that **Waiting for Trigger** is highlighted in the Trigger Capture Status pane of the ILA properties window.

13. In SDK, in the Debug window, right-click the **XMD Debug Target Agent** and select **Resume** (Figure 177).

![Figure 177: Resume Code Execution on XMD Target Debug Agent](image)
In Vivado, you should see the ILA trigger as shown below (Figure 178).

![Figure 178: PS to PL Cross Trigger Waveform](image_url)

This demonstrates that when the breakpoint is encountered during code execution, the PS7 triggers the ILA that is set up to trigger when TRIG_IN goes high.

14. Remove the breakpoint that you set earlier on line 104. Select the Breakpoints tab, right-click testperiph.c [line: 104] and select Remove All (Figure 179).

![Figure 179: Remove All Breakpoints](image_url)
15. In the Remove All Breakpoints dialog box, click Yes (FIGURE 180).

![Figure 180: Remove All Breakpoints Dialog Box](image)

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

17. In Vivado, click Run Trigger Immediate (FIGURE 181).

![Figure 181: Run Trigger Immediate](image)

The waveform window should look as follows (FIGURE 182).
Figure 182: Waveform Demonstrating PS to PL Trigger

This should stop the Processor from executing code because the ILA triggers the TRIG_OUT port, which in turn interrupts the processor. This can be seen in SDK the in the highlighted area of the debug window (Figure 183).

Figure 183: Verify that the Processor Has Been Interrupted in SDK

Conclusion

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

Lab Files

This tutorial demonstrates the Cross Trigger feature of Zynq, which you perform in the GUI environment. Therefore, the only provided Tcl file is connectZynqCrossTrigger.tcl.
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. You can use a machine on any operating system for this tutorial.

The MicroBlaze system includes native Xilinx IP including:

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

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

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

This tutorial targets the Xilinx KC705 FPGA Evaluation Board, and uses the 2013.3 version of Vivado Design Suite. To test your system on a KC705 board, you must use a terminal emulation program such as TeraTerm or Hyperterminal.

You must also ensure that you have the device drivers for the board installed correctly.
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 Getting Started page, select Create **New Project** (FIGURE 184).

Figure 184: Vivado Getting Started
Step 1: Invoke the Vivado IDE and Create a Project

The New Project wizard opens (Figure 185).

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 **Verilog**. 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:

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

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

1. From Flow Navigator > IP Integrator, select Create Block Design (FIGURE 186).

![Create Block Design](image)

**Figure 186: Create Block Design**

The Create Block Design dialog box opens.

2. Specify the IP subsystem design name, and click OK (Figure 187).

![Name Block Design](image)

**Figure 187: Name Block Design**
3. In the IP integrator diagram area, right-click and select **Add IP** (Figure 188).

![Figure 188: Add IP](image)

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

4. In the Search field, type **microblaze** to find the MicroBlaze IP, then select MicroBlaze and click **Enter** (Figure 189).

![Figure 189: Search Field](image)
Customize the MicroBlaze Processor

1. In the IP diagram area, double-click the MicroBlaze Processor diagram. The Re-customize IP dialog box opens.
2. On page 1 of the Re-customize IP dialog box, check the **Use Instruction and Data Caches** options box, and click **Next** (Figure 190).

![Figure 190: MicroBlaze Configuration Wizard](image)

3. On page 2 of the Re-customize IP dialog box (Figure 191):
   a. Check the **Enable Barrel Shifter** option.
   b. From the pulldown menu, in Enable Floating Point Unit, select **BASIC**.
   c. From the pulldown menu, in Enable Integer Multiplier, select **MUL32 (32-bit)**.
   d. Check the **Enable Integer Divide** option.
   e. Check the **Enable Branch Target Cache** option.
   f. Click **Next**.
4. On page 3 of the MicroBlaze Re-customize IP dialog box (FIGURE 192):
   a. 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 High Address to 0xFFFFFFFF by clicking on the Auto button, which enables the High Address field.
      - Set the Base Address to 0x80000000 by clicking on the Auto button, which enables the Base Address field.
      - Enable Use Cache for All Memory Accesses for both Instruction Cache as well as Data Cache by clicking the Auto button first then checking the check box.
   b. 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.
   c. Additionally, ensure that the Base address and the High address of both Data Cache and Instruction Cache are the same.
d. 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.

e. Click **Next**.

**Figure 192: Page 3: Re-customize IP**
5. On Page 4 of the Re-customize IP dialog box, ensure that the MicroBlaze Debug Module is enabled by checking the **Enable MicroBlaze Debug Module Interface** check box, and click **Next** (FIGURE 193).

![Figure 193: Page 4: Re-customize IP](image-url)
6. On Page 5 of the Re-customize IP dialog box, ensure that the **Enable Peripheral AXI Data Interface** option is checked, and click OK to re-configure the MicroBlaze processor (Figure 194).

**Figure 194: Re-customize IP**

The MicroBlaze processor diagram should look like **Figure 195**.

**Note:** Two extra ports, M_AXI_DC and M_AXI_IC are added to the MicroBlaze processor.

**Figure 195: MicroBlaze Diagram**
Instantiate and Customize the MIG

1. On the diagram area, right-click and select **Add IP** to instantiate the MIG 7 Series (DDR3 Interface Controller) (FIGURE 196).

   ![Figure 196: MIG Selection](image)

2. Configure the MIG core by double-clicking it.

3. Double-click the **MIG 7 Series** IP. The Memory Interface Generator wizard opens.

   ![Figure 197: Memory Interface Generator (MIG)](image)

4. Click **Next**. By default, the IP integrator enables the **Create Design** option, and sets the **Number of Controllers** to 1.
5. Ensure that the AXI4 interface check box is checked, and click Next (FIGURE 198).

![Xilinx Memory Interface Generator](image)

**Figure 198: Page 2: MIG Options**

6. In the Pin Compatible FPGAs page, click Next.

7. In the next page, DDR3 SDRAM memory is checked by default, click Next.

8. In the Options for Controller 0 – DDR3 SDRAM page set the value of the Clock Period to 2500 ps (400.00 MHz), and select the Memory Type as SODIMMs. This changes the Memory Part to MT8JTF12864HZ-1G6 (Figure 199).
9. Click **Next**.

![Figure 199: MIG page: Controller Options](image)

**Figure 199: MIG page: Controller Options**
10. In the AXI Parameter page (FIGURE 200):
   a. Set **Data Width** to **32**.
   b. Click **Next**.

**Figure 200: MIG Page: AXI Parameters Options**
11. In the Memory Options page (FIGURE 201):
   a. Set the **Input Clock Period** to **5000 ps** (200 MHz)
   b. Set RTT (nominal) - **On Die Termination (ODT)** to **RZQ6**.
   c. Click **Next**.

![Figure 201: MIG Page: Memory Options](image)

12. On the FPGA Options page, set the **Reference Clock** to **Use System Clock**.
13. Change System Reset Polarity to **ACTIVE HIGH**, and click **Next** ([FIGURE 202]).

![Figure 202: MIG Page: System Clock](image-url)
14. In the Extended FPGA Options page, select the **DCI Cascade** check box. Click **Next** (Figure 203).

![Figure 203: MIG Page: DCI Cascade](image)

15. In the Pin/Bank Selection Mode page, select **Fixed Pin Out** and click **Next** (FIGURE 204).

![Figure 204: MIG Page: Pin/Bank Selection Mode](image)

16. Next, import the pin configurations for a specified user constraint file (UCF). In the MIG Pin Selection for Controller page, click **Read XDC/UCF**.

17. Navigate to the folder where you unzipped and placed the `mig_7_series_pin_layout.ucf` (see Required Design Files to locate this file), and click **Open** to import the file.
18. Click Validate to validate the pinout (FIGURE 205).

![Figure 205: MIG Page: Pin Selection](image)

19. The DRC Validation Log dialog box opens, stating that the Current Pinout is valid (Figure 206).

20. Click OK.

![Figure 206: DRC Validation Dialog Box](image)

21. After reviewing the message, click Next to show the Systems Signals Selection page.

22. Click Next to open a summarized report.
23. Click **Next** to open the Simulation Model License Agreement.

24. Click **Accept** to accept the license agreement.

25. Click **Next** to open a **PCB configuration** note, and then click **Next** again. The design notes open.

26. Click **Generate** to generate the **mig_7_series DDR3 IP**.

You see the following design diagram in the IP integrator diagram area (**FIGURE 207**).

![Figure 207: mig_7series_1 Diagram](image)

Use IP integrator Run Block Automation to complete the block design.

**Run Block Automation**

1. Click **Run Block Automation** (**FIGURE 208**).
2. In the Run Block Automation dialog box (Figure 209):
   a. Set Local Memory to 64 KB.
   b. Leave the Local Memory ECC to its default value None.
   c. Change the Cache Configuration to 32 KB.
   d. Leave the Debug Module option to its default state Debug Only.
   e. Leave the Peripheral AXI Interconnect option checked.
   f. Check the Interrupt Controller option.
   g. Select the Clock Connection option of /mig_7series_1/ui_clk (100 MHz).
   h. Click OK.
This generates a basic MicroBlaze system in the IP Integrator diagram area (Figure 210).

Figure 209: Run Block Automation Dialog Box

Figure 210: MicroBlaze System
Add Peripherals: AXI Timer, and AXI Uartlite

1. By right-clicking in the IP integrator diagram area and selecting Add IP, search for and select the AXI Timer (FIGURE 211).

![Figure 211: AXI Timer](image)

2. By following the same step, add AXI UartLite (FIGURE 212).

![Figure 212: AXI UartLite](image)

Make a connection between the /mig_7series_1/ui_clk_sync_rst port and the /proc_sys_reset_1/ext_reset_in port (FIGURE 213).

![Figure 213: Port Connections](image)

Use Connection Automation

Run Connection Automation provides several options that you can select to make connections.

1. Click Run Connection Automation and select /axi_timer_0/S_AXI (FIGURE 214).
You can make the connections to the MIG core later.

Figure 214: Run Block Automation on the Design
Step 2: Create an IP Integrator Design

The Run Connection Automation dialog box opens (FIGURE 215).

2. Click **OK**.

![Figure 215: Run Connection Automation Dialog Box](image)

Figure 215: Run Connection Automation Dialog Box

This connects the `s_axi` port of the timer to the AXI Interconnect peripheral.

3. Click **Run Connection Automation** again, and select `/axi_uartlite_0/S_AXI` (FIGURE 216).

![Figure 216: axi_uartlite_1](image)

Figure 216: axi_uartlite_1

The Run Connection Automation dialog opens, indicating the connection that can be made.
4. Click OK.

![Run Connection Automation Dialog Box for /axi_uartlite_i/S_AXI](image)

**Figure 217: Run Connection Automation Dialog Box for /axi_uartlite_i/S_AXI**

5. Click **Run Connection Automation** and select `/axi_uartlite_0/UART`. (Figure 218).

![axi_uartlite_1](image)

**Figure 218: axi_uartlite_1**

6. The Run Connection Automation dialog box opens (Figure 219). Click OK.

![Run Connection Automation Dialog Box for /axi_uartlite_0/UART](image)

**Figure 219: Run Connection Automation Dialog Box for /axi_uartlite_0/UART**

This connects the **uart** interface of the **Uartlite** to external ports.
At this point, your IP integrator diagram area looks like **Figure 220.**

**Figure 220: MicroBlaze Connected to UART and AXI Timer**

### Concatenate Interrupt Signals

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

1. Connect the interrupt pin of the AXI Timer to the input pin `In0[0:0]` of Concat.
2. Connect the interrupt pin of AXI Uartlite to the input pin `In1[0:0]` of Concat. The connections should look like **Figure 221.**

**Figure 221: Connected Interrupt Ports**
3. Add an AXI BRAM Controller by right-clicking the IP integrator diagram area, selecting **Add IP**, and typing the name in the IP Catalog Search box (**FIGURE 222**).

![AXI BRAM Controller](image)

**Figure 222: AXI BRAM Controller**

4. Click **Run Connection Automation**, and select `/mig_7series_0/S_AXI` (**FIGURE 223**).

![Run Connection Automation](image)

**Figure 223: mig_7series_1/S_AXI**

The Run Connection Automation dialog box opens. You have two options:

- Select the **Peripheral AXI Interconnect** to connect to the MicroBlaze processor.
- Select the **Cached AXI Interconnect** to connect to the MicroBlaze processor.

5. `/microblaze_1 (Cached)` is selected by default. Click **OK**. (**FIGURE 224**).

![Run Connection Automation](image)

**Figure 224: Run Connection Automation for /microblaze_0 (cached)**
This instantiates another AXI Interconnect, and makes the appropriate connections between the MIG and the MicroBlaze using the AXI Interconnect.

6. Click **Run Connection Automation** again to connect the BRAM Controller to the MicroBlaze processor. Select `/axi_bram_ctrl_0/S_AXI` (**FIGURE 225**).

![Figure 225: axi_bram_ctrl_1/S_AXI](image)

You have two options:

- Connect the AXI BRAM Controller to the MicroBlaze processor using the Peripheral AXI Interconnect.
- Use a different AXI Interconnect to connect to the MicroBlaze processor in cached mode.

The `/microblaze_0(Cached)` option is selected by default.

7. Click **OK** (**FIGURE 226**).

![Figure 226: Run Connection Automation on AXI BRAM Controller](image)
Connect BRAM Controller to Block Memory Generator

You need to connect the AXI BRAM Controller to Block Memory Generator. As you will notice, the Block Memory Generator IP has not been instantiated in the Block Design. As you run connection automation, the Block Memory Generator will be instantiated and connected to the appropriate pins of the AXI BRAM Controller.

1. Click Run Connection Automation and select /axi_bram_ctrl_0/BRAM_PORTA.

   ![Figure 227: Run Connection Automation on /axi_bram_ctrl_0/BRAM_PORTA](image)

   The Run Connection Automation dialog box opens.

2. Click OK.

   ![Figure 228: Run Connection Automation Dialog Box for BRAM Controller](image)

   ![Figure 229: Bmg Instantiated After Running Connection Automation on AXI BRAM Controller](image)
3. Click Run Connection Automation and select `/axi_bram_ctrl_0/BRAM_PORTB`.

![Figure 230: Run Connection Automation on /axi_bram_ctrl_0/BRAM_PORTB](image)

The Run Connection Automation dialog box opens and gives you two choices. You can either instantiate a new BMG and connect the PORTB of the AXI BRAM Controller to the new BMG IP or you can 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.

![Figure 231: Run Connection Automation dialog box for /axi_bram_ctrl_0_bram](image)

**Connect MIG Input and Reset Ports**

Now you need to make connections for the input clock and reset ports of the MIG core.

1. Highlight SYS_CLK interface of the MIG core, then right-click it and select **Create Interface Port**. In the Create Interface Port dialog box, leave everything at its default value and click **OK** (Figure 232).

![Figure 232: Create MIG Clock Interface Port](image)
2. Click **Run Connection Automation** and select `/mig_7series_0/sys_rst` (FIGURE 233).

![Figure 233: Run Connection Automation on /mig_7series_0/sys_rst](image)

The Run Connection Automation dialog box offers to connect the sys_rst port to the on-board reset for KC705 or connect to a custom reset. In this case, you chose the default option of on-board reset.

3. Click **OK**.

![Figure 234: Run Connection Automation Dialog Box for /mig_7series_0/sys_rst](image)

4. Select the **DDR3** port of the MIG core, right-click it, and select **Make External** (FIGURE 235).

![Figure 235: DDR3 Port of the MIG as an External Port](image)
In the IP integrator tool, click the **Regenerate** button. The IP integrator diagram area looks like [FIGURE 236].

**Step 3: Create Constraints**

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

1. In the Sources window, expand the Constraints folder ([FIGURE 237]).
2. Open the **Constraints** folder, right-click the **constrs_1** folder, then select **Edit Constraints Set** *(FIGURE 238).*

**Figure 238: Edit Constraints Sets Option**

The Edit Constraints Sets dialog box opens.

3. Click **Create File** *(FIGURE 239).*

**Figure 239: Edit Constraints Set Dialog Box**
4. Name the file, and click **OK** (Figure 240).

![Create Constraints File](image)

**Figure 240: Name Constraint Set File**

5. Click **OK**.

6. Expand the **constrs_1** folder and double-click the constraints file.

   The file specifies the local constraints for some of the ports created in the block design (Figure 241).

![Local Constraints File](image)

**Figure 241: Local Constraints File**

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

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

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

Save the file by pressing the **Ctrl+S** keys, or use **File > File Save**.
Step 4: Memory-Mapping the Peripherals in IP Integrator

1. In the Sources window, double-click the design name under Design Sources to open the block design if not already open (Figure 242).

![Figure 242: Design Name and Address Editor Location](image)

2. Click the **Address Editor** tab. In the Address Editor:
   a. Expand the MicroBlaze IP.
   b. Change the range of mig_7_series IP in both the **Data** and the **Instruction** section to **512 MB** (Figure 243).

![Figure 243: Data and Instruction Set to 512 MB](image)
3. Save your design by pressing Ctrl+S, or from the File menu, select Save Block Design.
   You must also ensure that the memory in which you are going to run and store your software is within in the cacheable address range that you specified when you assigned values to the cache(s) Base address and cache(s) High address.
   This occurs when you enable Instruction Cache and Data Cache, and when you re-configure the MicroBlaze processor.
   To use either MIG DDR or AXI block RAM, those IP must be in the cacheable area; otherwise, the MicroBlaze processor cannot read from or write to them.
   You can also use this map to manually include or exclude IP from the cacheable region or otherwise specify their addresses.

---

**Step 5: Creating a Top-Level Verilog Wrapper**

1. Under Design Sources, right-click your design and click Create HDL Wrapper (FIGURE 244).

   ![Create HDL Wrapper](image)

   **Figure 244: Create HDL Wrapper**

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

2. Click OK.

   ![Creating an HDL Wrapper](image)

   **Figure 245: Creating an HDL Wrapper**
Step 6: Validate Block Design

To run design rule checks on the design, click the **Validate Design** button on the toolbar, or select **Tools > Validate Design**.

![Figure 246: Running Design Rule Checks by Clicking on Validate Design](image)

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

![Figure 247: The Validate Design Dialog Box](image)

Step 7: Generate Output Products

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

![Figure 248: Generate Output Products](image)
Alternatively, you can click **Generate Block Design** in the Flow Navigator.

![Generate Block Design](image)

**Figure 249: Generate Block Design**

The Generate Output Products dialog box appears.

2. Click **Generate**.

![Generate Output dialog box](image)

**Figure 250: Generate Output dialog box**
Step 8: Take the Design through Implementation

In the Flow Navigator:

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

2. Click Yes.
   
The Bitstream Generation Completed dialog box asks you to select what to do next.

3. Keep the default selection of Open Implemented Design, and click OK.

   ![Bitstream Generation Completed Dialog Box](image)

   Figure 251: Bitstream Generation Completed Dialog Box

4. After the implanted design opens, expand the Implemented Design drop-down list in the Flow Navigator and click on Report Timing Summary (FIGURE 252).

   ![Report Timing Summary](image)

   Figure 252: Verify Timing Constraints

5. In the Report Timing Summary dialog box, click OK (FIGURE 253).
6. Verify that all timing constraints have been met by looking at the Timing – Timing Summary window.

![Timing Summary](image)

**Figure 253: Timing Summary**

---

**Step 9: Exporting the Design to SDK**

Next, open the design and export to SDK.

1. Select **File > Export > Export Hardware for SDK**.
2. In the Export to Hardware dialog box, select the **Launch SDK** check box (FIGURE 254).

![Export to Hardware Dialog Box](image)

**Figure 254: Export to Hardware Dialog Box**

**CAUTION!** If you get an error message stating Failed to run “export_hardware” for design, make sure that you open the block design by double-clicking the IP integrator design from the Sources Pane.

3. Specify the workspace to which you want to export your hardware design. If you do not specify the workspace, your project exports to:  
   `<project_name>.sdk/SDK/SDK_Export/` You can leave this field to its default state.
4. Click **OK**. SDK launches in a separate window.
Step 10: Configuring FreeRTOS

1. In SDK, select **Xilinx Tools > Repositories** (Figure 255).

   ![Figure 255: Xilinx Tools Repositories](image)

2. Click **New** next to the local repository box to add a new local repository (Figure 256).

   The freeRTOS repository is in the path `<design_path>/sysrtos/lib`.

   ![Figure 256: SDK Preferences Page](image)
3. Add the path and re-scan the repositories.
4. Click **Apply** and then click **OK**.

---

**Step 11: Creating a freeRTOS “Hello World” Application**

1. In SDK, right-click **hw_platform_0** and select **New > Project** (FIGURE 257).

![Figure 257: SDK New Project Selection](image)

Figure 257: SDK New Project Selection

2. In the New Project dialog box, select **Xilinx Tools > Application Project** (FIGURE 258).
3. Click **Next**.

![Figure 258: SDK New Project Wizard](image)

Figure 258: SDK New Project Wizard
4. Type a name (**hello_world**) for your project and choose **FreeRTOS** as the OS platform (**FIGURE 259**).

![Figure 259: New Project: Application Project Wizard](image)

5. Click **Next**.
6. Select the **freeRTOS Hello World** application template, and click **Finish** (**FIGURE 260**).

![Figure 260: New Project: Template Wizard](image)

SDK creates a new “Hello World” application.
Step 12: Executing the System on a KC705 Board

To run the design on a KC705 board:

1. Connect the board to your computer.
2. Select the **Terminal 1** tab in SDK and click the **Settings** button (**FIGURE 261**).

![Figure 261: SDK Terminal Tab and Settings Button](image)

The Terminal Settings dialog box opens.

3. Specify the parameters as shown in the Terminal Settings dialog box, and click **OK** (**FIGURE 262**).

![Figure 262: Terminal Settings Dialog Box](image)

You see the confirmation in the Terminal 1 tab that it is connected to the device (**FIGURE 263**).
Step 12: Executing the System on a KC705 Board

4. Start the XMD Console by selecting **Xilinx Tools > XMD Console** *(FIGURE 264)*.

5. Program the bitstream on the board using the following command in the XMD Console.

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

You can also program the FPGA by selecting **Xilinx Tools > Program FPGA** *(FIGURE 265)*.

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

7. In the XMD Console, type:

```
XMD% connect mb mdm
```

Then, type:
8. Reset and stop the MicroBlaze processor before running the software by using the **rst** and **stop** commands as shown:

```
XMD% rst
XMD% stop
```

9. Download the freeRTOS “Hello World” program ELF file by typing:

```
XMD% dow
<project_path>/<project_name>/project_name.sdk/SDK/SDK_Export/hello_world/Debug/hello_world.elf
```

10. Run the program:

```
XMD% run
```

The output displays in the Terminal tab as shown in **FIGURE 266**:

![Figure 266: 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 Real Time Operating System.

**Lab Files**

The **Tcl script lab4.tcl** is included with the design files to perform all the tasks in Vivado. The SDK operations must be done in the SDK GUI. You will need to modify the Tcl script to match the project path and project name on your machine.
Lab 5: Converting Legacy EDK IP to Use in IP Integrator

Introduction

You might at some point need to use a legacy core from XPS in Vivado. In this lab, you will learn how to convert an XPS processor core, or Pcore, to a Vivado Design Suite native IP for use in IP Integrator.

Step 1: Managing IP

1. Launch the Vivado Design Suite IDE:

   Start > All Programs > Xilinx Design Tools > Vivado 2013.3 > Vivado 2013.3

   As an alternative, click the Vivado 2013.3 Desktop icon to start the Vivado IDE.

   The Vivado IDE Getting Started page, shown in FIGURE 267, contains links to open or create projects and to view documentation.

---

1 Your Vivado Design Suite installation may be called something different from Xilinx Design Tools on the Start menu.
2. Select **Manage IP** from the Getting Started page and select **New IP Location** from the drop down menu (**FIGURE 268**).

**Figure 268: Manage IP**

The Create a New Customized IP Location dialog box displays, as seen in **FIGURE 269**.

**Figure 269: Manage IP**
3. Click **Next**.

![Figure 269: Open IP Catalog](image)

**Figure 269: Open IP Catalog**

4. On the Manage IP Settings page, click the browse button to the right of the **Part** field (**FIGURE 270**).

![Figure 270: Manage IP Settings Page](image)

**Figure 270: Manage IP Settings Page**
5. The Select Device dialog box opens. Ensure that **Parts** is selected under **Specify** and type **xc7k325** in the search field. From the list of available devices, select the **xc7k325tffg900-1** part, and then click **OK** *(FIGURE 271).*

![Figure 271: Select Part](image-url)
6. Leave the default setting of Target Language as VHDL and Target Simulator as Vivado Simulator. Leave the Simulator language to Mixed. Change the Default IP Location to: <Extract_Dir>/lab_5. Click Finish (Figure 270).

The Vivado IDE loads the IP Catalog view layout (Figure 272).

Figure 272: IP Catalog View Layout

Because this is a Managed IP session and not an RTL project, you cannot perform behavioral simulation, elaborate the design, or launch synthesis or implementation. This view is strictly for exploring the IP Catalog, and for packaging designs for use as IP.
Step 2: Running the Package IP Wizard

1. Select **Tools > Create and Package IP** from the main menu.

   The Package New IP wizard displays and provides information about the process you are starting (FIGURE 273).

![Figure 273: Package IP](image)

2. In the Package New IP dialog box, in the Welcome to the Create and Package IP page, press **Next** to continue (FIGURE 274).

![Figure 274: IP Packager Dialog Box](image)

3. In the Choose IP Source Location dialog box, **Package a specified directory** is the only valid option. In the IP Definition Location, select **...** and browse to `<Extract_Dir>/lab_5` (FIGURE 275).

![Figure 275](image)
4. Click **Next**.

   The Begin IP Creation page displays, showing what work is performed during the IP packaging flow. Click **Finish** (Figure 276).

   ![Figure 276: Begin IP Creation Page](image)

   The Package IP wizard collects the available information from the specified IP location, and the **IP Packager Summary** screen summarizes what is gathered and populated into the `component.xml` file, a newly created IP-XACT definition of the IP.
**Note:** If a data/*.pao file is present in the specified IP location, this file is read, and the file and associated library information are used, as is the case for this EDK Pcore IP.

5. Examine the contents of the **IP Packager Summary**, as seen in **FIGURE 277**.

![IP Packager Summary](Image)

**Figure 277: IP Packager Summary**

6. Click **OK** to close the IP Packager Summary dialog box.
7. Review the various sections of the Package IP window, as seen in Figure 278, and update information as required. The Package IP window shows the current IP identification, including the Vendor, Library, Name, and Version (VLNV) attributes of the newly packaged core.

Make the following changes:

a. Display Name: my_axi_gpio

b. Description: AXI general purpose IO block

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

![Figure 278: Package IP – IP Identification](image-url)
8. Click **IP Compatibility** to see the family of device supported for this project (FIGURE 279).

![Figure 279: IP Compatibility Page](image)

9. Click **IP File Groups** to see the source files included for synthesis as well as simulation (FIGURE 280).

![Figure 280: IP File Groups Page](image)
10. Click IP Customization Parameters.

This page shows all the parameters that can be customized for this IP. Closely examine the IP Customization Parameters page (Figure 281).

Consider the address range for the core: C_S_AXI_ADDR_WIDTH.

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

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

![Figure 281: IP Customization Parameters Page](image)
Step 2: Running the Package IP Wizard

11. Click **IP Ports** to see all the I/O ports in the design (Figure 282).

Figure 282: IP Ports Page

12. Click **IP Interfaces** to see the interfaces in the IP (Figure 283).

Figure 283: IP Interfaces Page
13. Click **IP Addressing and Memory** to see the memory mapping of any registers in the IP (Figure 284).

![Figure 284: IP Addressing and Memory Page](image)

14. Click **IP GUI Customization Layout** to see a preview of how the IP will look during customization and when added to an IP Integrator subsystem design (Figure 285).

![Figure 285: IP GUI Customization Layout](image)
15. Click IP Licensing and Security to see the settings for adding any license features and any payment that might be required for purchasing the IP (FIGURE 286).

![Figure 286: IP Licensing and Security](image)

16. Click Review and Package for a summary of the IP and a list of possible missing information, such as product guide, example, or change log (FIGURE 287).

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

When you exit Vivado or close the Package IP window, you are asked if you want to save the project. Click **Package IP**.

![Figure 287: Review and Package IP Page](image)
Adding the IP to the catalog results in any unsaved edits written to `component.xml`, the IP definition file, being saved to the location of the existing Pcore, which is now also an IP Integrator IP.

The Add to Catalog command only adds the IP repository to the IP Catalog in the current project, or Manage IP session. To use the IP repository in other designs, you must add the IP repository to the IP Settings using the **Tools > Project Settings** command from the main menu. Other users can also use the IP from the repository you just created by adding it to the IP Settings of their own projects.

---

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

17. Select the **IP Catalog** tab and look for `my_axi_gpio` in the **BaseIP** category, and review the details (**Figure 288**).

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

![Figure 288: IP Catalog – my_axi_gpio](image-url)
Conclusion

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

You could:

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

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

Lab Files

You can run the lab5.tcl file to perform this tutorial using a scripted flow.