

Vivado Design Suite Tutorial

Embedded Processor Hardware Design

UG940 (v2021.2) January 11, 2022





Revision History

01-11-2022: Released with Vivado® Design Suite 2021.2 without changes from 2020.2.

Section	Revision Summary		
12/14/2020 Version 2020.2			
General Updates	Validated for release 2020.2		
07/16/2020 Version 2020.1			
General Updates	Validated for release 2020.1		



Table of Contents

Revision History2
Programming and Debugging Embedded Processors5
Introduction
Hardware and Software Requirements5
Tutorial Design Descriptions6
Locating Tutorial Design Files7
Lab 1: Building a Zynq-7000 SoC Processor Design
Introduction
Step 1: Start the Vivado IDE and Create a Project
Step 2: Create an IP Integrator Design
Step 3: Debugging the Block Design
Step 4: Generate HDL Design Files
Step 5: Implement Design and Generate Bitstream
Step 6: Export Hardware to the Vitis software platform
Step 7: Create a Software Application
Step 8: Run the Software Application
Step 9: Connect to the Vivado Logic Analyzer
Conclusion
Lab Files40
Lab 2: Zynq-7000 SoC Cross-Trigger Design41
Introduction
Step 1: Start the Vivado IDE and Create a Project41
Step 2: Create an IP Integrator Design42
Step 3: Implement Design and Generate Bitstream49
Step 4: Export Hardware to the Vitis software platform
Step 5: Build Application Code in the Vitis Software Platform
Step 6: Connect to Vivado Logic Analyzer65
Step 7: Set the Processor to Fabric Cross Trigger68
Step 8: Set the Fabric to Processor Cross-Trigger70
Conclusion71



Lab Files	71
Lab 3: Programming an Embedded MicroBlaze Processor	72
Introduction	
Step 1: Start the Vivado IDE and Create a Project	73
Step 2: Create an IP Integrator Design	74
Step 3: Memory-Mapping the Peripherals in IP Integrator	86
Step 4: Validate Block Design	87
Step 5: Generate Output Products	88
Step 6: Create a Top-Level Wrapper	89
Step 7: Take the Design through Implementation	90
Step 8: Export the Design to the Vitis software platform	90
Step 9: Create a "Peripheral Test" Application	92
Step 10: Execute the Software Application on a KC705 Board	100
Step 11: Connect to Vivado Logic Analyzer	
Step 12: Set the MicroBlaze to Logic Cross Trigger	105
Step 13: Set the Logic to Processor Cross-Trigger	
Conclusion	108
Lab Files	109
Appendix A: Additional Resources and Legal Notices	110
Xilinx Resources	110
Documentation Navigator and Design Hubs	110
References	

Please Read: Important Legal Notices...... 111



Programming and Debugging Embedded Processors

Introduction

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

In this tutorial, you use the Vivado IP integrator to build a processor design, and then debug the design with the Vitis[™] unified software platform and the Vivado Integrated Logic Analyzer.

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

Hardware and Software Requirements

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

The following platform boards and cables are also needed:

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



Tutorial Design Descriptions

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

Lab 1: Building a Zynq-7000 SoC Processor

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

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

Lab 1 shows how to graphically build a design in the Vivado IP integrator and use the Designer Assistance feature to connect the IP to the Zynq-7000 SoC PS.

After you construct the design, you mark nets for debugging the logic. Then you generate the Hardware Design Language (HDL) for the design as well as for the IP. Finally, you implement the design and generate a bitstream, then export the hardware description of the design to the Vitis[™] software platform. You will use the Vitis software platform to build and debug the software design, and learn how to connect to the hardware server (hw_server) application used to communicate with the Zynq-7000 SoC processors. Then you will perform logic analysis on the design with a connected board.

Design Files

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

• lab1.tcl

Related Information

Locating Tutorial Design Files Building a Zynq-7000 SoC Processor Design

Lab 2: Zynq-7000 SoC Cross Trigger Design

Lab 2 requires that you have the Vitis[™] software platform installed on your machine.





In Lab 2, you use the Vitis software platform to build and debug the software design, and learn how to connect to the hardware server (hw_server) application used to communicate with the Zynq-7000 SoC processors. Then, you use the cross-trigger feature of the Zynq-7000 SoC processor to perform logic analysis on the design on the target hardware.

Design Files

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

• lab2.tcl

Related Information

Locating Tutorial Design Files Zynq-7000 SoC Cross-Trigger Design

Lab 3: Programming a MicroBlaze Processor

Lab 3 uses the Xilinx MicroBlaze processor in the Vivado IP integrator to create a design and export to the Vitis[™] software platform, software design, and logic analysis as in Lab 2.

Design Files

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

• lab3.tcl

Related Information

Locating Tutorial Design Files Programming an Embedded MicroBlaze Processor

Locating Tutorial Design Files

Design data is in the associated Reference Design File.

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



Lab 1

Building a Zynq-7000 SoC Processor Design

Introduction

In this lab you create a Zynq[®]-7000 SoC processor based design and instantiate IP in the processing logic fabric (PL) to complete your design. Then you mark signals to debug in the Vivado[®] Logic Analyzer. Finally, you take the design through implementation, generate a bitstream, and export the hardware to the Vitis[™] unified software platform. You then create a Software Application that can be run on the target hardware. Breakpoints are added to the code to cross-probe between hardware and software.

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

Step 1: Start the Vivado IDE and Create a Project

- 1. Start the Vivado IDE by double-clicking the Vivado desktop icon or by typing vivado at a terminal command line.
- 2. From the Quick Start section, click **Create Project**, as shown in the following figure:



Quick Start Create Project > Open Project > Open Example Project >	
Tasks Manage IP > Open Hardware Manager > Xillinx Tcl Store >	
Learning Center Documentation and Tutorials > Quick Take Videos > Release Notes Guide >	. ∘ § ^{•.}

The New Project Wizard opens.

3. Click Next.

The Project Name dialog box opens.

new Project	
	Create a New Vivado Project This wizard will guide you through the creation of a new project.
	To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.
•	<back ned=""> Einish Cancel</back>

- 4. In the Project Name page, type a project name and select a location for the project files. Ensure that the **Create project subdirectory** check box is selected, and then click **Next**.
- 5. In the Project Type page, select **RTL Project**, and then click **Next**.
- 6. Ensure that the **Do not specify sources at this time** check box is cleared. Click **Next**.
- 7. In the Add Sources page, set the Target language to your desired language, Simulator language to **Mixed** and then click **Next**.
- 8. In the Add Constraints page, click Next.
- 9. In the Default Part page, do the following:
 - a. Select **Boards**.
 - b. From the Board Rev drop-down list, select **All** to view all versions of the supported boards.
 - c. Choose the version of the ZYNQ-7 ZC702 Evaluation Board that you are using.
 - d. Click Next.



Parts Boards						
Reset All Filters	Name: All		~	Board R	ate Board Repositori	es V
Bearch: Q- zc702	(1 matc)	b)				
Display Name		Preview	Vendor	File Version	Part	
ZYNQ-7 ZC702 Evaluation Board			xilinx.com	1.1	xc7z020clg484-1	^
ZYNQ-7 ZC702 Evaluation Board			xilinx.com	1.2	xc7z020clg484-1	I
ZYNQ-7 ZC702 Evaluation Board Add Daughter Card Connections			xilinx.com	1.3	xc7z020clg484-1	1
ZYNQ-7 ZC702 Evaluation Board Add Daughter Card Connections			xilinx.com	1.4	xc7z020clg484-1	1

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

10. Review the project summary in the New Project Summary page, and then click **Finish** to create the project.

Step 2: Create an IP Integrator Design

- 1. In the Flow Navigator \rightarrow IP Integrator, select Create Block Design.
- In the Create Block Design dialog box, specify a name for your IP subsystem design such as zynq_design_1. Leave the Directory field set to the default value of <Local to Project>, and leave the Specify source set field to its default value of Design Sources.



lease specify name	of block design.	
Design name:	zynq_design_1	0
Directory:	<local project="" to=""></local>	~
		5 mil 1

- 3. Click OK.
- 4. In the block design canvas right-click, and select Add IP.

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

This design is empty. Press the 🕂 button to add IP.

The IP catalog opens.

- 5. In the search field, type zynq to find the ZYNQ7 Processing System IP.
- 6. In the IP catalog, select the **ZYNQ7 Processing System**, and press **Enter** on the keyboard to add it to your design.

In the Tcl Console, you see the following message:

```
create_bd_cell -type ip -vlnv xilinx.com:ip:processing_system7:5.5
processing_system7_0
```

There is a corresponding Tcl command for most actions performed in the IP integrator block design. Those commands are not shown in this document; instead, the tutorial provides Tcl scripts to run each lab.

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

7. In the IP integrator window, click the **Run Block Automation** link.

```
* Designer Assistance available. Run Block Automation
```

The Run Block Automation dialog box opens, stating that the FIXED_IO and DDR interfaces will be created for the Zynq-7000 SoC IP core. Also, note that the Apply Board Preset check box is checked. This is because the selected target board is ZC702.

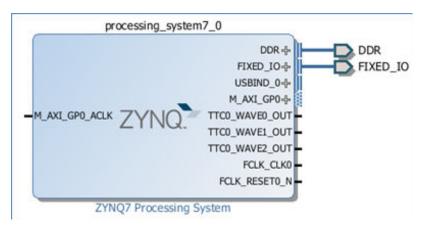
8. Ensure that both Cross Trigger In and Cross Trigger Out are disabled.



figuration options on the right.	n by checking the boxes of the blo	ocks to conne	d. Select a	block on the left to	display its
Q ¥ ♦	Description				
 All Automation (1 out of 1 selected) Processing_system7_0 	This option sets the board preset on the Processing System. All current properties will be overwritten by the board preset. This action cannot be undone. Zynq7 block automation applies current board preset and generates external connections for FIXED_IO, Trigger and DDR interfaces. NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration. Instance: /processing_system7_0 Options				
	Make Interface External:		R		
	Apply Board Preset	~			
	Apply Board Preset.	Disable	~		

9. Click OK.

After running block automation on the Zynq-7000 SoC processor, the IP integrator diagram looks as follows.



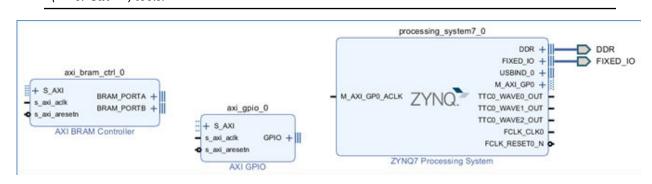
Now you can add peripherals to the processing logic (PL).

- 10. Right-click in the IP integrator diagram, and select Add IP.
- 11. In the search field, type gpi to find the AXI GPIO, and then press Enter to add it to the design.
- 12. Similarly, add the AXI BRAM Controller.



Your Block Design window will look like the following figure. The relative positions of the IP might vary.

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



Use Designer Assistance

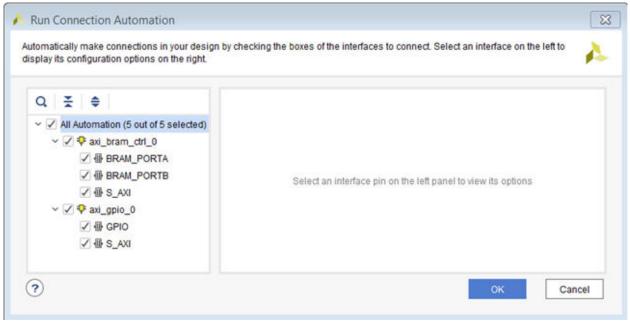
Designer Assistance helps connect the AXI GPIO and AXI block RAM Controller to the Zynq-7000 SoC PS.

1. Click Run Connection Automation as shown in the following figure.

```
Designer Assistance available. Run Connection Automation
```

The Run Connection Automation dialog box opens.

2. Select the All Automation (5 out of 5 selected) check box, as shown in the following figure.





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

3. Click the S_AXI interface of the axi_bram_ctrl_0, and ensure that its Clock Connection (for unconnected clks) field is set to the default value of Auto.

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

Q ₹ €	Description		
 ✓ All Automation (5 out of 5 selected) ✓ ♥ axi_bram_ctrl_0 ✓ ⊕ BRAM_PORTA ✓ ⊕ BRAM_PORTB ✓ ⊕ S_AXI ✓ ♥ axi_gpio_0 	Connect Slave interface (/axi_bram_ctrl_0/S_AXI) to a selected Master address space. Options		
	Master:	/processing_system7_0/M_AXI_GP0	
	Interconnect IP:	Auto ~	
☑ 昼 GPIO	Crossbar clock source of Interconnect IP:	Auto 🗸	
☑ 卧 S_AXI	Clock source for Master interface:	Auto 🗸	
	Clock source for Slave interface:	Auto 🗸	

4. For the GPIO interface of the axi_gpio_0 instance, select **leds_4bits** from the Select Board part Interface drop down list.

Q ¥ ♦	Description		
 ✓ All Automation (5 out of 5 selected) ✓ ♥ axi_bram_ctrl_0 ✓ ● BRAM_PORTA ✓ ● BRAM_PORTB ✓ ● S_AXI 	Connect Board Part Interface Interface: /axi_gpio_0/GPIO Options	to IP interface.	
✓ ✓ ♥ axi_gpio_0	Select Board Part Interface:	gpio_sw (DIP switches)	~
✓ 小 GPIO		gpio_sw (DIP switches)	
✓ 働 S_AXI		leds_4bits (LED)	6
		Custom	10

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

Q ≚ ≑	Description		
✓ III Automation (5 out of 5 selected) ✓ Ø \$\vee\$ axi_bram_ctrl_0 Ø BRAM_PORTA	Connect Slave interface (/axi_gplo_0/S_AXI) to a selected Master address space. Options		
 ☑ ⊕ BRAM_PORTB ☑ ⊕ S_AXI ✓ ☑ 幸 axi_gpio_0 	Master: Interconnect IP:	/processing_system7_0/M_AXI_GP0 New AXI Interconnect	
☑ ⓓ GPIO	Crossbar clock source of Interconnect IP:	Auto ~	
☑ ∰ S_AXI	Clock source for Master interface:	Auto	
	Clock source for Slave interface:	Auto ~	

6. Click OK.



The IP integrator subsystem looks like the following figure. The relative positions of the IP might differ slightly.

gram	? _ 🗆 ? >
(d) X № ○ 0 Ξ 0 + ∞ № B C 3 0	0
al, bram, cht, 0, ham al, bram, cht, 0, ham Block Memory Generator Block Me	FIXED_

7. Click the Address Editor window and expand the processing_system7_0 hierarchy to show the memory-map of the IP in the design.

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

8. Change the range of the axi_bram_ctrl_0 to 64K, as shown in the following figure.

Diagram 🔀 Address Edit	or ×				
Q ≚ ♦ 🖻					
Cell	Slave Interface	Slave Segment	Offset Address	Range	High Address
v Processing_system7_0)				
✓ 团 Data (32 address bit	s : 0x40000000 [1	IG])			
🚥 axi_bram_ctrl_0	S_AXI	Mem0	0x4000_0000	8к 🗸	0x4000_1FFF
🚥 axi_gpio_0	S_AXI	Reg	0x4120_0000	8K ^	0x4120_FFFF
				16K	
				32K	
				64K	
				128K 🗟	
				256K	
				512K 🗸	
				1M //	

- 9. Click the Diagram window to go back to the block design.
- 10. Click the Regenerate Layout button C to regenerate an optimal layout of the block design.



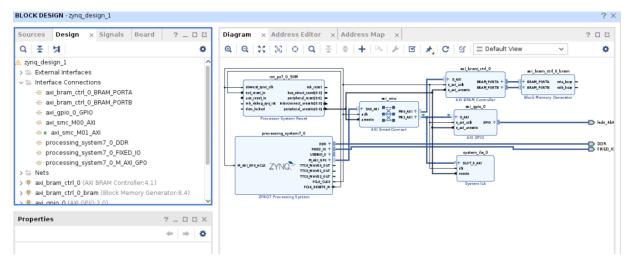
Step 3: Debugging the Block Design

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

1. To debug the master/slave interface between the AXI Interconnect IP (ps7_0_axi_periph) and the GPIO core (axi_gpio_0), in the Diagram view, select the interface, then right-click and select **Debug**.

In the Block Design canvas on the net that you selected in the previous step, a small bug icon

appears, indicating that the net has been marked for debug. You can also see this in the Design Hierarchy view, as displayed in the following figure, on the interface that you chose to mark for debug.



When a net is marked for debug, the Designer Assistance link in the banner of the block design canvas becomes active.

2. Click Run Connection Automation.

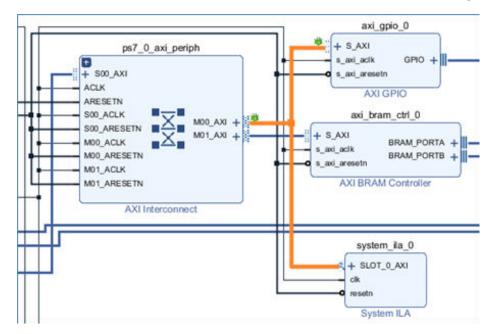
The All Automation is selected by default with the various options for AXI Read/Write signals set, as shown in the following figure.



matically make connections in your design by che figuration options on the right.					
a ž ♦	Description				
✓ All Automation (2 out of 1 selected) ✓ Interface Connections ✓ ∞ ps7_0_axi_periph_M00_AXi	Connect selected interface-connection and/or net to System ILA Core for Debugging in Hardware Manager. System ILA (Integrated Logic Analyzer) IP core is a logic analyzer which allows you to perform in-system debugging of designs and shows interface level events in the Hardware Manager in an intuitive way. Options				
	AVI Read Address: AVI Read Data: AVI Write Address: AVI Write Data:	Data and Trigger Data and Trigger Data and Trigger Data and Trigger	2 2 2 2 2 2		
	AXI Write Response: Source Clock: System ILA: AXI-MM Protocol Checker:	Data and Trigger /processing_system7 Auto ~	-0FCLK_CLK0		

3. Click OK.

A System ILA IP is instantiated on the block design which is appropriately configured to debug the AXI Interface marked for debug. The net marked for debug is connected to this System ILA IP and an appropriate clock source is connected to the clk pin of the System ILA IP. The clock source is the same clock domain to which the interface signal belongs.



4. From the toolbar, to run Design-Rules-Check (DRC), click the Validate Design button ⊠. Alternatively, you can do the same from the menu by:



- Selecting **Tools** → **Validate Design** from the menu.
- Right-clicking in the Diagram window and selecting Validate Design.

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

The Tcl Console shows the following warning.

```
WARNING: [BD 41-1781] Updates have been made to one or more nets/
interface connections marked for debug. Debug nets, which are already
connected to System ILA IP core in the block-design, will be
automatically available for debug in Hardware Manager. For unconnected
Debug nets, please open synthesized design and use 'Set Up Debug' wizard
to insert, modify or delete Debug Cores. Failure to do so could result
in critical warnings and errors in the implementation flow.
```

Block designs can use the instantiation flow, where a System ILA or ILA IP is instantiated in the block design, or they can use the netlist insertion flow, where nets are only marked for debug but the debug core is inserted post-synthesis. This warning message can be ignored if the instantiation flow is being used (as in this lab).

- 5. Click OK.
- 6. From the Vivado menu, select File \rightarrow Save Block Design

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

Step 4: Generate HDL Design Files

You now generate the HDL files for the design.

1. In the Sources window, right-click the top-level subsystem design and select **Generate Output Products**. This generates the source files for the IP used in the block design and the relevant constraints file.

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

The Generate Output Products dialog box opens, as shown in the following figure.



Generate Output Products	agram × AddressEditor × Ad	ddress Map x	
The following output products will be generated.	A A X X O A E O	+ ~ / 8 *	C 앱 = Default Ve
Preview	nt,s10,558		at bran, rol 5
Q # 0	der al general per setter al setter	-i.m	AN DRAW Card alar
∽ ▲	a ah adag men at an		-1, grin, 0
Synthesis	Free enter Tyrdam North		- 1,00 - 1,00 - 0 - 0
implementation Simulation	Construct gained orthogo	All SnarGoorer	ANIGNO
B Hw_Handoff	100 T	ļļi	ayatan jila ji
Synthesis Options	ARCHING ZYNO TOURING -	ı Ц	
) global	TTO MAR DIT - TTO MAR DIT -	ា	
Qut of context per IP	recommendation and		System ful
Out of context per glock Design	201607 Proceeding Lychem		
Run Settings			
On jocal host: Number of jobs: 20			
O On gemote hosts Configure Hosts			
O Launch rung on Cluster			
(?) Apply Cenerate Cancel	S		

- 2. Leave all the settings to their default values. Click Generate.
- 3. The Generate Output Products dialog box opens informing that Out-of-context runs were launched.
- 4. Click OK.

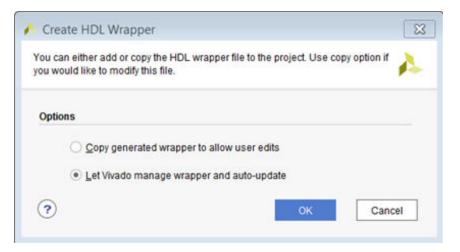
Out-of-context runs can take a few minutes to finish. You can see the status of the runs by clicking on the Design Runs window at the bottom of the Vivado IDE.

5. After the out-of-context runs are finished, in the Sources window, right-click the top-level subsystem, **zynq_design_1**, and select **Create HDL Wrapper** to create an top level HDL file that instantiates the block design.

The Create HDL Wrapper dialog box opens, as shown in the following figure, and presents you with two options:

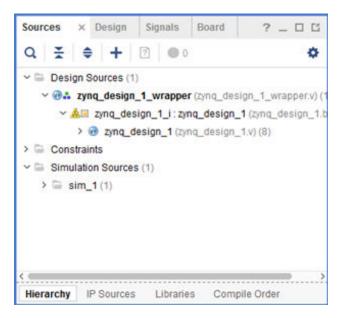
- The first option is to copy the wrapper to allow edits to the generated HDL file.
- The second option is to create a read-only wrapper file, which will be automatically generated and updated by Vivado.





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

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





Step 5: Implement Design and Generate Bitstream

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

The No Implementation Results Available dialog box opens.

2. Click Yes.

The Launch Runs dialog box opens. Here you can select various options such as the Number of Jobs, the host where the Runs are launched etc.

3. Click OK.

This will launch synthesis, implementation ,and generate the bitstream which could take a few minutes.

After the bitstream generates, the Bitstream Generation Completed dialog box opens, as shown in the following figure. **Open Implemented Design** should be checked by default.

📋 Bitstream Ge	neration successfully completed.
Open Implen	nented Design
○ View Reports	
Open <u>H</u> ardw	are Manager
O Generate Me	mory Configuration File
Don't show this	dialog again

- 4. Click OK.
- 5. When the implemented design opens, look at the **Design Timing Summary** window to ensure that all timing constraints are met.



Step 6: Export Hardware to the Vitis software platform

IMPORTANT! For the Digilent driver to install, you must power on and connect the board to the host PC before launching the Vitis software platform.

Export Hardware Platform

1. From the Vivado File menu, select File \rightarrow Export \rightarrow Export Hardware.

The Export Hardware Platform dialog box opens.

2. Click Next.



Export Hardware Platform

This wizard will guide you through the export of a hardware platform for use in the Vitis or PetaLinux software tools.

To export a hardware platform, you will need to provide a name and location for the exported file and specify the platform properties.

3. Select the Include bitstream option using the radio button in the Output view and click Next.





🕕 Export Hardware Platform	×
Output Set the platform properties to inform downstream tools of the intended use of the target platform's hardware design.	4
 Pre-synthesis This platform includes a hardware specification for downstream software tools. 	
Include bitstream This platform includes the complete hardware implementation and bitstream, in addition to the hardware specification for software tools.	
< <u>B</u> ack <u>N</u> ext > <u>F</u> inish Cance	əl

4. Leave the XSA file name field at its default value and click Next.

🙏 Export Hardwar	re Platform	×
Files		
Enter the name of	f your hardware platform file, and the directory where the XSA file will be stored.	
XSA file name:	zynq_design_1_wrapper	\otimes
Export to:	c:/ug940/lab1	⊗ …
	The XSA will be written to: c:\ug940\lab1\zynq_design_1_wrapper.xsa	
	< <u>Back</u> <u>N</u> ext > <u>F</u> inish	Cancel

5. Click Finish. This will export the hardware XSA File in the lab1 project directory.



A Export Hardware Platform	1	×
	Exporting Hardware Platform	
HLx Editions	A new fixed hardware platform named 'zynq_design_1_wrapper' will be written as 'c:\ug940\lab1\zynq_design_1_wrapper.xsa'.	
	The platform will include a post-implementation model, including a bitstream description, describing the hardware for downstream software tools.	
E XILINX.	To export the platform, click Finish.	
	< <u>Back</u> <u>N</u> ext > <u>Finish</u> Cancel	

6. To launch the Vitis software platform, select **Tools** \rightarrow **Launch Vitis IDE**.

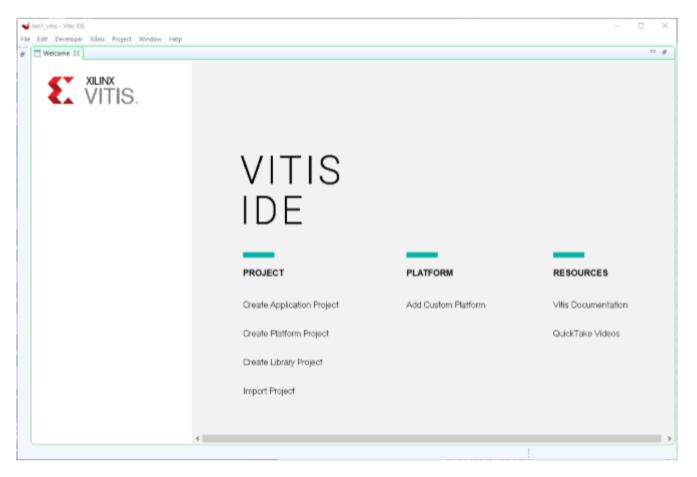
The Eclipse Launcher dialog box opens.

7. Specify the desired Workspace location such as C:\UG940\lab1_vitis and click Launch (Windows-specific example).



Step 7: Create a Software Application

The Vitis software platform launches in a separate window.



- 1. Close the Welcome screen if it appears.
- Select File → New → Application Project or under Project click Create Application Project.
 The New Application Project dialog box opens.



✓ New Application Project	_	
Create a New Application Project		
This wizard will guide you through the 4 steps of creating new application projects. 1. Choose a platform or create a platform project from Vivado exported XSA 2. Put application project in a system project , associate it with a processor 3. Prepare the application runime – domain 4. Choose a template for application to quick start development		
Platform System Project Project		
Processor Domain App		
 A platform provides hardware information and software environment settings. A system project contains one or more applications that run at the same time. A domain provides runtime for applications, such as operating system or BSP. A workspace can contain unlimited platforms and unlimited system projects. 		
Skip welcome page next time. (Can be reached with Back button)		
(?) < Back Next > Finish		Cancel

- 3. Select the **Skip welcome page next time** check box if you do not want the welcome to appear when the Vitis software platform is launched again.
- 4. Click Next.
- 5. In the Platform page, select the **Create a new platform from hardware (XSA)** tab.





Vew Application Project			_ 0	×
Platform Please select a platform to create the project				•••
Select a platform from repository				^
Hardware Specification Provide your XSA file or use a pre-built board description				
XSA File: zc706 zcu102 zed			Browse]
Platform name:				
				~
(?)	< Back Next >	Finish	Car	ncel

6. Click **Browse** to open The Create Platform from XSA window. Navigate to the directory where the XSA file was created in Vivado and click **Open**.

→ * ↑	his PC > Windows (C:) > ug940 > lab1	~ 7) Search lab1	
ganize 🔻 New fold	ler			iii 👻 🔟
ug940 ^	Name	Date modified	Туре	Size
📙 lab1	lab1.cache	7/7/2020 7:08 AM	File folder	
lab1_vitis	📊 lab1.hbs	7/7/2020 7:08 AM	File folder	
lab2	lab1.hw	7/7/2020 7:08 AM	File folder	
lab2_vitis	lab1.ip_user_files	7/7/2020 7:08 AM	File folder	
lab3	lab1.runs	7/7/2020 7:08 AM	File folder	
Users	📊 lab1.sim	7/6/2020 5:05 PM	File folder	
-	lab1.srcs	7/7/2020 7:08 AM	File folder	
versal_design_f	zynq_design_1_wrapper.xsa	7/6/2020 5:23 PM	XSA File	912 KI
Filer	name: zyng_design_1_wrapper.xsa		*.xsa;*.dsa;	

7. Ensure the **Generate boot Components** option is selected in the Platform page.



Vew Applic	ation Project					_ 0	×
latform Note: A platfor	m project will be generate	d automatically in workspace for the selected XSA. It	can be customized later.				
🔄 Select a p	olatform from repository	Create a new platform from hardware (XSA)					^
Hardware	Specification						
	C:\ug940\lab1\zynq_desig	gn_1_wrapper.xsa					
XSA File:	vck190 zc702 zc706 zcu102 zed C:\ug940\lab1\zynq_desig	n_1_wrapper.xsa				Browse]
	name: zynq_design_1_wra e boot components	pper					
							~
?			< Back	Next >	Finish	Ca	ncel

- 8. Click Next.
- 9. In the Application project name field, type the name desired, such as Zynq_Design. Leave all other fields to their default values, and click **Next**.

New Application Project			
blication Project Details ecify the application project name and	d its system project properties		
oplication project name: Zynq_Desig	n		
System Project			
	application or select an existing one from the	e workpsace 👔	
Select a system project	System project details		
Create new	System project name: Zyne	a Desian system	
	Target processor		
	Select target processor for th	ne Application project.	
	Processor	Associated applications	
	ps7_cortexa9_0	Zynq_Design	
	ps7_cortexa9_1		
	ps7_cortexa9 SMP		
	Show all processors in the h	ardware specification 🗹 👔	
	Show all processors in the h	ardware specification 🗹 👔	
	Show all processors in the h	ardware specification 🗹 👔	

10. In the Domain page leave all the fields at their default values and click **Next**.



Vew Application Project							×
Domain Select a domain for your project or create a new							••
Select the domain that the application would lin Note: New domain created by this wizard will ha			ition template sel	ected in the ne	kt step		
Select a domain Create new	Domain details						
Create new	Name:	domain_	ps7_cortexa9_0				
	Display Name:	domain_	ps7_cortexa9_0				
	Operating System:	standalo	ne	~			
	Processor:	ps7_cort	exa9_0				
	Architecture:	32-bit		~			
?	<	Back	Next >	Finish		Canc	el

11. In the Templates page, select **Peripheral Tests**.





Vew Application Project				_		Х
T emplates						••
Select a template to create your project.						
Available Templates:						
Find:	ĒĒ	Peripheral T	ests			
 SW development templates 		Simple test r	outines for all periph	erals in the hardware.		
Dhrystone						
Empty Application						
Empty Application (C++)						
Hello World						
IwIP Echo Server						
IwIP TCP Perf Client						
IwIP TCP Perf Server						
IwIP UDP Perf Client						
IwIP UDP Perf Server						
Memory Tests						
OpenAMP echo-test						
OpenAMP matrix multiplication Dem	10					
OpenAMP RPC Demo						
Peripheral Tests						
RSA Authentication App						
Zynq DRAM tests						
Zynq FSBL						
		T				
⁽)		< Back	Next >	Finish	Cance	el 👘

12. Click Finish.

13. The application project is created in the Vitis software platform. Click the hammer icon sto build the application.

When the application project finishes compiling, you will see the following in the Console window.





Step 8: Run the Software Application

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

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

The Program FPGA dialog box opens and points to the bitstream zynq_design_1_wrapper.bit which was included as a part of the XSA.

•		Program Device		
Program Devi	ce			4
Specify the bits	tream and th	e ELF files that reside in BRAM memory		
Project:	Zynq_desig	n_system 🔻		
Connection:	zc702-2 👻		New	
Device:	Auto Detect		Select	
Bitstream/PDI:	\${project_loc:Zynq_design}/_ide/bitstream/zynq_desi		Search	Browse
🗌 Partial Bitstr	eam			
BMM/MMI File:	\${project_lo	oc:Zynq_design}/_ide/bitstream/zynq_desig	Search	Browse
Software Configu	iration			
Processor ELF/MEM File to Initialize in Block R		M		

3. Ensure that the Bitstream field shows the bitstream file that you created in Step 5, and then click **Program**.

Note: The DONE LED on the board turns green if the programming is successful. You should also see an INFO message suggesting that the FPGA was configured successfully in the Vitis software platform Log window.



Console	💽 Prob	lems 📳 Vitis Log 🖇	3 Guidance	la 🕞	~ =	
			one query RDI_DATADIR one setting workspace for the tool.			^
09139109 10:17:42 10127122	INFO INFO	: Registering co : Checking for B	mmand handlers for Vitis TCF services SP changes to sync application flags for project 'Zynq_Design' srget on host '127.04.01' and port '3121'.			i
18:27:22	THEO	: 'targets -set	filter (item cable name == "Dimilent 1748-5011 2182835423654" && level==8" -in d successfully with bitstream "Ci/tutorials/2019.2/05940/labi_vitis/2yng_Design			
۲.					>	

- 4. In the Explorer, select and right-click the **Zynq_Design** application.
- 5. Select **Debug As** \rightarrow **Debug Configurations**.
- 6. In the **Debug Configurations** dialog box, right-click **Single Application Debug**, and select **New Configuration** as shown.

🖌 Debug Configurations			
Create, manage, and run Debug a program using App	-		
Type filter text	→ •		h settings from this dialo New Configuration' butte
	New Confi New Proto Export Duplicate Delete	type 😼	button to export t ate' button to copy button to remove putton to configur
	L Link Protot U Unlink Prot R Reset with	totype Prototype Values	xisting configuration nfiguration(s) and nen configuration(s) and

7. The Create, manage and run configurations page opens. Click **Debug**, as shown in the following figure.



🐋 Debug Configurations		– 0 ×
Create, manage, and run configurations Debug a program using Application Debugger		À.
Image: Single Application Debug Image: Single Application Debug <	Name Debugger_Zynq_Design-Default X Main Application Target Setup 64 Arguments Termination Symbol Files 1/2 Source Debug Type Standalone Application Debug Connection Local New Project: Zynq_Design New Debug Environment Debug Configuration Debug Project: Zynq_Design Debug Debug Environment Analysis Performance Analysis Debug Debug Debug	Browse
Filter matched 4 of 4 items	Revert	Apply
1	Debug	Close

8. The Debug perspective window opens.

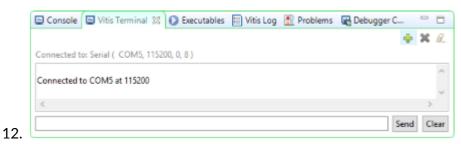
- 🖩 🕲 19 - 16 - 1 14 - 10 - 10 1 Debug 22 - 12 - 12 - 12 14 - 7 - 10 1	De III = M 2x <> .4 =, ∞ # = 0	Quick Access) Pesign to Debug
	<pre>Si #Jaclade "ktudt_header.h" Si #Jaclade "ktups.h" Si #Jaclad</pre>	(10) Yu, 22 (10) Yu, 22
Ar scugic_header.h Ar scugic_header.h Br scutimer_header.h Br scuwdt_header.h Er textpeniph.c Ar textpeniph.c	^	<pre>Ti: B _boot NactA Info: ABM Cores-AS MPCore #0 (target Info: ABM Cortes-AS MPCore #0 (target 2) St main() at/arc/testperiph.c: 61 61. Yil ToccheFashle(). </pre>

- 9. Set the terminal by selecting the **Vitis Terminal** tab and clicking the **+** button.
- 10. Use the settings shown in the following figure or the ZC702 board. The as shown in the following figure.COM Port might be different on your machine.Use the settings shown in the following figure or the ZC702 board. The

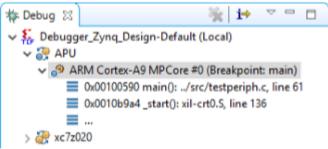


🚽 Connect to	serial port \times
Basic Settin	gs
Port:	COM5 ~
Baud Rate:	115200 ~
▼ Advance	Settings
Data Bits:	8 ~
Stop Bits:	1 ~
Parity:	None \sim
Flow Contr	ol: None 🗸
Timeout (s	ec):
ОК	Cancel

11. Click OK.



13. Verify the terminal connection by checking the status at the top of the tabln the Debug tab, expand the tree to see the processor core on which the program is running, as shown in the following figure.



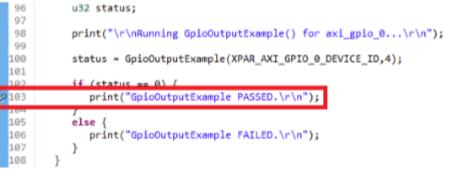


14. If testperiph.c is not already open, select .../src/testperiph.c, double-click it to open that location.

Add a Breakpoint

Next, add a breakpoint after the "if" statement.

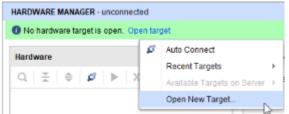
1. Double-click in the blue bar to the left of line 103 to add a breakpoint on that line of source code, as shown in the following figure:



Note: Sometimes the line number varies, so enter the breakpoint where appropriate.

Step 9: Connect to the Vivado Logic Analyzer

- 1. Go back to the Vivado session and from the Program and Debug drop-down list in the Flow Navigator → Program and Debug, click Open Hardware Manager.
- 2. In the Hardware Manager window, click **Open target**, and select **Open New Target** to open a connection to the Digilent JTAG cable for ZC702, as shown in the following figure.

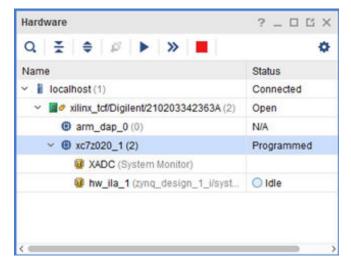


The Open New Hardware Target dialog box opens.

- 3. Click Next.
- 4. Select the appropriate options from the drop down menu for Connect to option. Click **Next** on the Hardware Server Settings page.
- 5. The hardware server should be able to identify the hardware target. Click **Next** on the Select Hardware Target page.
- 6. Click **Finish** in the Open Hardware Target Summary page.



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



- 7. First, ensure that the ILA core is active and capturing data. To do this, select the Status tab of the hw_{ila_1} in the Hardware Manager.
- 8. Click the **Run Trigger Immediate** button when hw_ila_1 window.

Waveform - hw_ila_1						? _ 0
Q + - & > > = B 0	a a 🙁 📲	H H H	ar of to of	н		
ILA Status: Idle		0				
Name	Value	°	200	400	1 ⁶⁰⁰	
Slot_0 : axi_smc_M01_A/0 : Interface	inactive			Inaciav		
> 💺 slot_0 : axi_smc_M01_AXI : AR Channel	No Read Addr Cm			No Read .ddr	Caste	
> 🛸 slot_0 : axi_smc_M01_A/0 : R Channel	No Read Data Bea			No Fand Data		
> 🛋 slot_0 : axi_smc_M01_AXI : AW Channel	No Write Addr Cm			No Write Adda		
> 🖌 slot_0 : axi_smc_M01_A00 : W Channel	No Write Data Bea			No Trite Juta		
> 🛋 slot_0 : axi_smc_M01_AXI : B Channel	No Write Respons			No Write Day	ponses	
		Dedated at 201	9-0ct-07 11:50:54			
	$\langle \rangle$	<				
ettings - hw_ila_1 Status - hw_ila_1 ×			? _ D Tri	gger Setup - hw_ila_1 × Capi	ure Setup - hw_ila_1	?.
🖢 🕨 📕 9a			9	+ - 0.		
Core status 🔵 🔘 🔘 📄 Idie						
core status						
Capture status - Window 1 of 1						
Window sample 0 of 1024				F	ress the 🕂 button to add	probes.
					-	
id ia						

Expand some of the Signal Groups by clicking the drop-down to see Static data from the System ILA core in the waveform window as shown in the following figure.



hw_Ea_1		7 🗆 🖬 X
Waveform - hw_ila_1		? _ 🗆 ×
g Q + − e > >	B & & X - N N ± ± +	f= -f H
LA Status: Idle	P	·
Q_ + - (b) >>> ELA Status: idle Name Name >> >>> Via stol_0: xol_smc_W01_ANI. intents >> >>> >>>	Value 0 (20	0,1 009 009 009 009 009 009 009
30 ✓ Mislot (): sol_smc_M01_AVI. Helm > Mislot (): sol_smc_M01_AVI. Helm > Mislot (): sol_smc_M01_AVI. > Mislot (): sol_smc_M01_AVI.	R Channel No Read Addr Cr Channel No Read Data Be V Channel No Write Addr Cr No Write Addr Cr Channel No Write Data Be Channel V WALID 0 U WALID 0 V WALID 0	Track ive No Sead Spitz Seats So Write Mark Cads IS Viewer ODD ODD ODD No Write Mark Seats So Write Marks
Settings - hw_lis_1 Status - hw_lis_		Image: Tripper Setup - trw_fia_1 x Capture Setup - trw_fia_1 ?

- 9. Set up a condition that triggers when the application code writes to the GPIO peripheral. To do this:
 - a. From the menu select **Window** \rightarrow **Debug Probes**.
 - b. Select, drag, and drop the slot_0:axi_smc_M01_AXI:AWVALID signal from the Debug Probes window into the Trigger Setup window.

•		00	•		
Debug Probes 🗆	⊡ × Iw	v_lia_1			? 🗆 🖾)
Q	٥	Waveform - hw_ila_1			? _ 🗆 ×
slot_0:axi_smc_N01_AXI:ARADDR	^ 8	Q + - * > > = D @	<u>ех</u> к и и	· 호 · · · · · · · · · · · · · · · · · ·	0
slot_0 : axi_smc_N01_AXI : ARPROT	Options	ILA Status: Idle	D		-
slot_0 : axi_smc_N01_AXI : ARREADY	, pr			2 2	
slot_0 : axi_smc_N01_AXI : ARVALID	Dashioard	Name	Value	400	
slot_0:axi_smc_N01_AXI:AW_CNT	- B	> Slot_0 : axi_smc_M01_A0 : Interface	inactive	Inactive	
slot_0:axi_smc_N01_AXI:AWADDR		> slot_0 : axi_smc_M01_AXI : AR Channel	No Read Addr Cr	No Read Addr Cude	
slot_0:asi_smc_N01_AXI:AWPROT		> slot_0 : axi_smc_W01_A01 : R Channel	No Read Data Be	No Dead Data Beats	
slot_0:ssi_smc_N01_AXI:AWREAD		> Sislot_0 : axi_smc_M01_AXI : AW Channel	No Write Add: Cri	No Write Addr Cads	
slot_0 : sxi_smc_N01_AXI : AWVALID		iii slot_0 : axi_smc_M01_AXI : W Channel	No Write Data Be	No Write Data Deats	
slot_0 : axi_smc_M01_AXI : B_CNT		16 slot_0 : axt_smc_M01_AXI : WVALID	0		
slot_0:axi_smc_N01_AXI:BREADY		\$ slot_0 : axi_smc_M01_AXI : WREADY	0		
slot_0:axi_smc_N01_AXI:BRESP		> V slot_0 : axi_smc_M01_A01 : WDATA	00000000	COCOCODO	
slot_0 : axi_smc_N01_AXI : BWALID		> V slot_0; axi_smc_M01_A0; WSTR8	•		
slot_0:axi_smc_N01_AXI:R_CNT		> Slot_0:axi_smc_W01_A00: B Channel	No Write Respon	No Write Berponses	
, slot_0:axi_smc_N01_AXI:RDATA					
slot_0:sxi_smc_N01_AXI:RREADY					
slot_0:ssi_smc_N01_AXI:RRESP			Updated at	t: 2013-0ct-07 12:50:55	
slot_0 : axi_smc_N01_AXI : RVALID			$\langle \dots \rangle \langle \dots \rangle$		
slot_0 : axi_smc_N01_AXI : WDATA					
slot_0 : axi_smc_N01_AXI : WREADY		Settings - hw_ila_1 Status - hw_ila_1 ×	? _ 🗆	Trigger Setup - hw_ila_1 × Capture Setup - hw_ila_1	? _ 🗆
slot_0:axi_smc_N01_AXI:WSTRB		🐮 🕨 🔉 📕 🗄		$Q + = D_1$	
slot_0:axi_smc_N01_AXI:WVALID					D. d.
slot_0:axi_smc_N01_AXI:W Channe	al Ev	Core status 🔵 🔘 🔘 🔘 Idle		Name Operator	Radix
zyng_design_1_i/system_ila_0/institu	it al			slot_0: axi_smc_M01_A0: AWWALID V	(8)
zyną_design_1_i/system_ila_0/institu	it sh	Capture status - Window 1 of 1			
zynq_design_1_i/system_ita_0/inst/ne	it_sh	Window sample 0 of 1024			
. zyng_design_1_i/system_ita_0/inst/ne	rt_sh	ld le			
zyng_design_1_i/system_ita_0/inst/ne	1_80 V			<	
(>				

c. Click the **Value** column of the *WVALID row, as shown in the following figure.

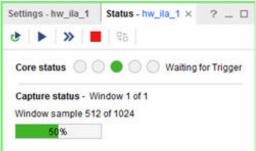


Trigger Setup - hw_ila_1 × Capture Setup -	hw_ila_1				? _ 🗆
Q + - D	Operator		Radix		Value
slot_0 : ps7_0_axi_periph_M00_AXI : AWVALID	==	•	[8]	•	X (don't ca
<					0 (logical zero) 1 (logical one) X (don't care) ⁵ R (0-to-1 transit F (1-to-0 transit B (both transition N (no transition

- d. Change the value from **X** to a **1**, from the drop down menu.
- 10. In the Waveform window or the Status window, arm the ILA core by clicking the Run Trigger

button 🕨 .

- 11. Notice that the Status window of the hw_ila_1 ILA core changes from:
 - Idle to Waiting for Trigger.
 - The Hardware window shows the Core Status as Waiting for Trigger, as shown in the following figure.



12. Go back to the Vitis software platform and continue to execute code. To do so, click the **Resume** button P on the toolbar.

Alternatively, you can press F8 to resume code execution.

The code execution stops at the breakpoint you set. By this time, at least one write operation has been done to the GPIO peripheral. These write operations cause the AWVALID signal to go from 0 to 1, thereby triggering the ILA core.

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



Waveform - hw_ila_1					? _ 🗆 >
Q + - & > > = B @	Q 🔀 📲	H H ±	tr fe er	н	0
ILA Status: Idle			7		
Name	Value	450	500	1650	600
> 🔚 slot_0 : axi_smc_M01_AXI : R Channel	No Read Data Be		No Re	ad Data Beats	
> Slot_0 : axi_smc_M01_AXI : AW Channel	-				O
18 slot_0 : axi_smc_M01_AXI : AWVALID	0				
16 slot_0 : axi_smc_M01_AXI : AWREADY	0				
> 😻 slot_0 : axi_smc_M01_AXI : AWADDR	000	000	X	004 X	000
> 😻 slot_0 : axi_smc_M01_AXI : AWPROT	Data Secure Unp	Data Secure Unpriv	ileged X	Data Secure	Privileged
> 😻 slot_0 : axi_smc_M01_AXI : AW_CNT	0	0	ж	0	0
> 🛤 slot_0 : axi_smc_M01_AXI : W Channel	_				
> Mislot_0 : axi_smc_M01_AXI : B Channel	_				
		Updated at: 2019-0c	6-07 13:05:51		

13. If you are proceeding to Lab 2, close your project by selecting File \rightarrow Close Project.

You can also close the Vitis software platform window by selecting $File \rightarrow Exit$.

Conclusion

This lab introduced you to creating a Zynq-based design in IP integrator, working with the System ILA IP to debug nets of interest, software development in the Vitis software platform and executing the code on the Zynq-7000 SoC processor.

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

In this lab, you:

- Created a Vivado project that includes a Zynq-7000 SoC processor design using the IP integrator.
- Instantiated IP in the IP integrator and made the necessary connections using the Designer Assistance feature.
- Marked and connected nets for debug using the System ILA IP, to analyze them in the Vivado Integrated Logic Analyzer.
- Synthesized, implemented, and generated the bitstream before exporting the hardware definition (XSA) to the Vitis software platform.
- Created a software application in the Vitis software platform and ran it on the target hardware, ZC702. By setting breakpoint in the application code, triggered the ILA in Vivado, thereby, demonstrating the hardware/software cross-probing ability.



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. This Tcl file only covers the Vivado portion of the design creation through bitstream generation. Subsequent steps from Step 7 and beyond must be performed manually as the intent is to demonstrate the cross-probing between hardware and software.

To use the Tcl script, launch Vivado and type <code>source lab1.tcl</code> 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 thelabl.tcl file to source the Tcl files correctly.





Lab 2

Zynq-7000 SoC Cross-Trigger Design

Introduction

In this lab, you use the cross-trigger functionality between the Zynq[®]-7000 SoC processor and the fabric logic. Cross-triggering is a powerful feature that you can use to simultaneously debug software in the Vitis software platform that is 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 double-clicking the Vivado desktop icon or by typing <code>vivado</code> at a command prompt.
- 2. From the Quick Start page, select **Create Project**.
- 3. In the New Project wizard, use the following settings:
 - a. In the Project Name page, type the project name and location.
 - b. Make sure that the **Create project subdirectory** check box is checked. Click **Next**.
 - c. In the Project Type page, select **RTL project**. Ensure that the **Do not specify sources at this time** check box is cleared. Click **Next**.
 - d. In the Add Sources page, set the Target language to either VHDL or Verilog. You can leave the Simulator language selection to Mixed. Click Next.
 - e. In Add Constraints page, click Next.
 - f. In the Default Part page, select **Boards** and choose **ZYNQ-7 ZC702 Evaluation Board** that matches the version of hardware that you have. Click **Next**.



g. Review the project summary in the New Project Summary page and click **Finish** to create the project.

Step 2: Create an IP Integrator Design

- 1. In Vivado Flow Navigator, click Create Block Design.
- 2. In the Create Block Design dialog box, specify zynq_processor_system as the name of the block design.
- 3. Leave the **Directory** field set to its default value of <Local to Project> and the Specify source set field to Design Sources.
- 4. Click OK.

The IP integrator diagram window opens.

5. Click the Add IP icon in the block design canvas, as shown in the following figure.

This design is empty. Press the 🕂 button to add IP.

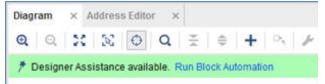
The IP catalog opens.

6. In the Search field, type Zynq, select the ZYNQ7 Processing System IP, and press Enter.

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

Bearch: Q. zyng	(1 match)	IP Details	×
ZYNQ7 Processing System System ENTER to select, ESC to cancel, Ctrl+	Vendo VLNV: Repos	n: 5.5 (Rev. 6) tes: AXI4, AXI4-Stream ption: Arm dual core SOC with Zyng tpga Production e: Included e Log: View Change Log r: Xilinx, Inc. xilinx.com/ip:processing_system7:5.5	

7. In the block design banner, click **Run Block Automation** as shown in the following figure.



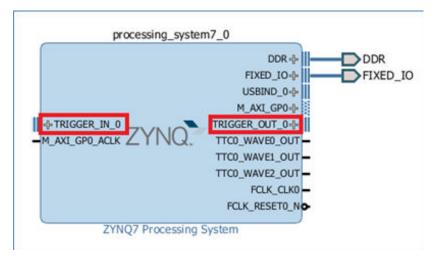


The Run Block Automation dialog box states that the FIXED_IO and the DDR pins on the ZYNQ7 Processing System 7 IP will be connected to external interface ports. Also, because you chose the ZC702 board as your target board, the **Apply Board Preset** check box is selected by default.

8. Enable the Cross Trigger In and Cross Trigger Out functionality by setting those fields to **Enable**, then click **OK**, as shown in the following figure.

figuration options on the right.	by checking the boxes of the blocks to connect. Select a block on the left to display it:	
Q ≚ ≑	Description	
 ✓ All Automation (1 out of 1 selected) ✓ ♥ processing_system7_0 	This option sets the board preset on the Processing System. All current proper will be overwritten by the board preset. This action cannot be undone. Zynq7 bl automation applies current board preset and generates external connections for FIXED_IO, Trigger and DDR interfaces. NOTE: Apply Board Preset will discard existing IP configuration - please unche this box, if you wish to retain previous configuration. Instance: /processing_system7_0 Options	ock or
	Make Interface External: FIXED_IO, DDR Apply Board Preset: Qross Trigger In: Enable ~	
	Cross <u>T</u> rigger Out Enable ~	

This enables the TRIGGER_IN_0 and TRIGGER_OUT_0 interfaces in the ZYNQ7 Processing System as show in the following figure.





9. Add the AXI GPIO and AXI block RAM Controller to the design by right-clicking anywhere in the diagram and selecting Add IP.

processing_system7_0 axi_gpio_0 DDR + DDDR FIXED_IO+ LXA 2-6 FIXED_IO axi_bram_ctrl_0 USBIND 04 axi adk GPIO --S AXI axi_aresetn M_AXI_GP0-BRAM_PORTAxi adk +TRIGGER_IN_0 TRIGGER_OUT_04 AXI GPIO BRAM_PORTB-M_AXI_GPO_ACLK TTCO WAVED OUT axi aresetn TTCO_WAVE1_OUT AXI BRAM Controller TTC0_WAVE2_OUT FCLK_CLK0 FCLK_RESETO_NO ZYNQ7 Processing System

The diagram area looks like the following figure.

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

The Run Connection Automation dialog box opens.

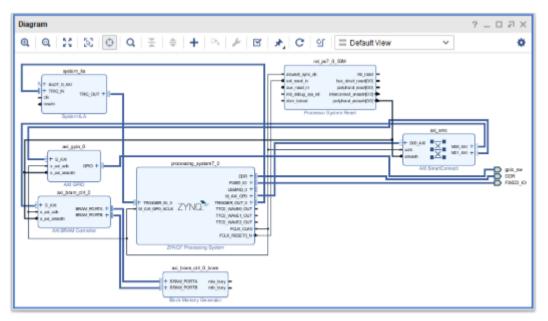
- 11. Select the **All Automation (7 out of 7 selected)** check box. This selects connection automation for all the interfaces in the design. Select each automation to see the available options for that automation in the right pane.
- 12. Make each of the following connections using the Run Connection Automation function.

Connection	More Information	Setting
axi_bram_ctrl_0 • BRAM_PORTA	The Run Connection Automation dialog box informs you that a new Block Memory Generator IP will be instantiated and connected to the AXI block RAM Controller PORTA	No options.
axi_bram_ctrl_0 • BRAM_PORTB	Note that the Run Connection Automation dialog box offers two choices now. The first one is to use the existing Block Memory Generator from the previous step or you can chose to instantiate a new Block Memory Generator if desired. In this case, use the existing BMG.	Leave the Blk_Mem_Gen field set to its default value of Blk_Mem_Gen of BRAM_PORTA.
axi_bram_ctrl_0 • S_AXI	The Run Connection Automation dialog box states that the S_AXI port of the AXI block RAM Controller will be connected to the M_AXI_GP0 port of the ZYNQ7 Processing System IP. The AXI block RAM 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 block RAM Controller.	Leave the Clock Connection (for unconnected clks) field set to Auto.



Connection	More Information	Setting
axi_gpio_0 • GPIO	The Run Connection Automation dialog box shows the interfaces that are available on the ZC702 board to connect to the GPIO.	Select LEDs_4Bits .
axi_gpio_0 • S_AXI	The Run Connection Automation dialog box states that the S_AXI pin of the GPIO IP will be connected to the M_AXI_GP0 pin of the ZYNQ7 Processing System. It also offers a choice for different clock sources that might be relevant to the design.	Leave the Clock Connection (for unconnected clks) field set to Auto.
<pre>processing_system7_0 TRIGGER_IN_0 TRIGGER_OUT_0</pre>	The Run Connection Automation dialog box states that the TRIGGER_IN_0 and TRIGGER_OUT_0 pins will be connected to the respective cross-trigger pins on the System ILA IP.	Leave the ILA option to its default value of Auto for both TRIGGER_IN_0 and TRIGGER_OUT_0 option.

Click **OK**. When these connections are complete, the IP integrator design looks like the following figure.



13. Click the Address Editor window of the design to ensure that addresses for the memorymapped slaves have been assigned properly. Expand **Data**. Change the range of the AXI block RAM Controller to **64K**, as shown below.



Q, ₹ ♦ 🖻						
Cell	Slave Interface	Slave Segment	Offset Address	Range		High Address
v • processing_system7_i	D					
✓ I Data (32 address bi	ts : 0x40000000 [1	IG])				
axi_bram_ctrl_0	S_AXI	Mem0	0x4000_0000	8K	¥	0x4000_1FF
axi_gpio_0	S_AXI	Reg	0x4120_0000	8K	^	0x4120_FFF
				16K	1	
				32K	U	
				64K		
				126K 256K		
				512K		
				111	~	

Mark Nets for Debugging

Next, you mark some nets for debugging.

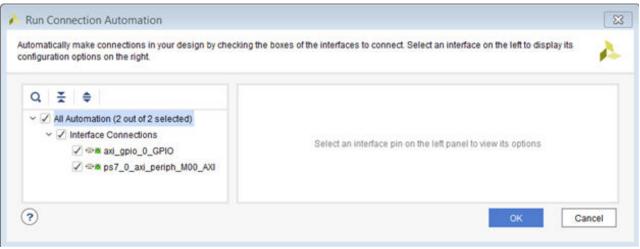
- 1. Click the Diagram window again, and select the net connecting the GPIO pin of the AXI GPIO IP to the LEDs_4Bits port.
- 2. Right-click in the block diagram area, and select **Debug**. This marks the net for debug.

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.

- 3. Similarly, select the net connecting the interface pin S_AXI of axi_gpio_0 and the M00_AXI interface pin of axi_smc instance.
- 4. Right-click in the block design and select **Debug** from the context menu.

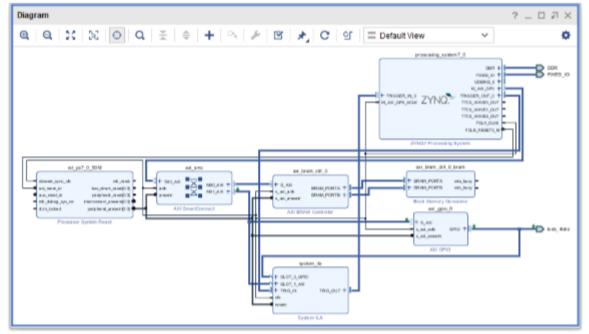
Note: When you mark a net for debugging, the Designer Assistance link at the top of the block design canvas banner becomes active.

- 5. Click Run Connection Automation.
- 6. In the Run Connection Automation dialog box, click the **All Automation (2 out of 2 selected)** check box.





- 7. Click OK.
- 8. Click the Regenerate Layout C button to generate an optimal layout of the design. The design should look like the following figure.



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

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

- 10. Click **OK**.
- 11. To save the IP integrator design, select File \rightarrow Save Block Design.

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

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

The Generate Output Products dialog box opens.



A Generate Output Products	×
The following output products will be generated.	$\mathbf{\lambda}$
Preview	
Q 🔀 🖨	
✓ ♣ ■ zynq_processor_system.bd (OOC per IP)	
Synthesis	
Implementation	
Simulation	
Hw_Handoff	
Synthesis Options	
O Global	
Out of context per IP	
Out of context per <u>B</u> lock Design	
Run Settings	
Number of jobs: 4	
Apply Generate Cane	cel

13. Click Generate.

The Generate Output Products dialog box informs you that out-of-context (OOC) module runs were launched.

- 14. Click **OK** on the Generate Output Products dialog box.
- 15. Wait until all OOC Module runs have finished running. This could take a few minutes.

Design Runs		? _ 🗆 🏹
Q ≚ ≑ I4 ≪ ▶ ≫ + %		
Name	Constraints	Status
> b synth_1 (active)	constrs_1	Not started
▷ impl_1	constrs_1	Not started
Out-of-Context Module Runs		
✓ ✓ zynq_processor_system		Submodule Runs Complete
zynq_processor_system_processing_system7_0_0_synth_1	zynq_processor_system_processing_system7_0_0	synth_design Complete!
zynq_processor_system_axi_gpio_0_0_synth_1	zyng_processor_system_axi_gpio_0_0	synth_design Complete!
zynq_processor_system_axi_bram_ctrl_0_0_synth_1	zyng_processor_system_axi_bram_ctrl_0_0	synth_design Complete!
v 🗸 zynq_processor_system_system_ila_0		Submodule Runs Complete
zynq_processor_system_system_ita_0_synth_1	zyng_processor_system_system_ila_0	synth_design Complete!
zynq_processor_system_rst_ps7_0_50M_0_synth_1	zynq_processor_system_rst_ps7_0_50M_0	synth_design Complete!
zynq_processor_system_axi_bram_ctrl_0_bram_0_synth_1	zynq_processor_system_axi_bram_ctrl_0_bram_0	synth_design Complete!
zynq_processor_system_xbar_0_synth_1	zynq_processor_system_xbar_0	synth_design Complete!
✓ zynq_processor_system_auto_pc_0_synth_1	zyng_processor_system_auto_pc_0	synth_design Complete!
zynq_processor_system_auto_pc_1_synth_1	zyng_processor_system_auto_pc_1	synth_design Complete!



16. In the Sources window, right-click zynq_processor_system, and select Create HDL Wrapper.

The Create HDL Wrapper dialog box offers two choices:

- The first choice is to generate a wrapper file that you can edit.
- The second choice is let Vivado generate and manage the wrapper file, meaning it is a read-only file.
- 17. Keep the default setting, shown in the following figure, and click OK.

	23
pper file to the project. Use copy op	tion if 🍌
allow user edits	
r and auto-update	
ок	Cancel
	allow user edits and auto-update

Step 3: Implement Design and Generate Bitstream

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

1. Click Generate Bitstream to generate the bitstream for the design.

The No Implementation Results Available dialog box opens with a message asking whether it is okay to launch synthesis and implementation.

2. Click Yes.

The Launch Runs dialog box opens.

3. Make the appropriate selections and click **OK**.

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

- 4. Click **OK** to open the implemented design.
- 5. Ensure that all timing constraints are met by looking at the Design Timing Summary tab, as shown in the following figure.



Note: The timing could be slightly different in your case.

Timing						2 - 0 2 3
Q X 0	Design Timing Summary					
General Information Timer Settings	Setup		Hold		Pulse Width	
Design Timing Summary	Worst Negative Stack (WNS):	9.610 ns	Worst Hold Stack (WHS):	0.035 ns	Worst Pulse Width Slack (WPWS):	8.750 ns
Clock Summary (2)	Total Negative Slack (TNS)	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Check Timing (4)	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints	0
> Intra-Clock Paths Inter-Clock Paths		10084	Total Number of Endpoints:	10084	Total Number of Endpoints:	5019
Cther Path Groups	All user specified timing constrain	its are met.				
Timing Summary - impl_1 (saved)						

Step 4: Export Hardware to the Vitis software platform



IMPORTANT! For the Digilent driver to install, you must power on and connect the board to the host PC before launching the Vitis software platform.

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

1. From the Vivado File menu, select File \rightarrow Export \rightarrow Export Hardware.

The Export Hardware dialog box opens.

- 2. Click Next
- 3. Select the Include bitstream option using the radio button in the Output view and click Next.



À Export Hardware Platform				×
Output Set the platform properties to inform downstream design.	tools of the intend	ed use of the targe	t platform's hardware	A
 Pre-synthesis This platform includes a hardware specifi 	cation for downstre	am software tools		
 Include bitstream This platform includes the complete hard specification for software tools. 	ware implementatio	on and bitstream, i	n addition to the hardwa	re
	< <u>B</u> ack	<u>N</u> ext >	<u>F</u> inish	Cancel

4. Leave the XSA file name field at its default value and click Next.

À Export Hardwar	re Platform	×				
Files Enter the name o	f your hardware platform file, and the directory where the XSA file will be stored.	4				
XSA file name:	zynq_processor_system_wrapper	\otimes				
Export to:	Export to: c:/ug940/lab2					
	The XSA will be written to: c:\ug940\lab2\zynq_processor_system_wrapper.xsa					
	< <u>B</u> ack <u>E</u> inish	Cancel				

5. Click Finish. This will export the hardware XSA File in the lab1 project directory.



A Export Hardware Platform	1	×
	Exporting Hardware Platform	
HLX Editions	A new fixed hardware platform named 'zynq_design_1_wrapper' will be written as 'c:\ug940\lab1\zynq_design_1_wrapper.xsa'.	
	The platform will include a post-implementation model, including a bitstream description, describing the hardware for downstream software tools.	
£ XILINX.	To export the platform, click Finish.	
	< <u>B</u> ack <u>N</u> ext > <u>Finish</u> Cancel	

- 6. To launch the Vitis software platform, select **Tools** \rightarrow **Launch Vitis IDE**.
- 7. Specify the desired Workspace location such as C:\UG940\lab2_vitis and click Launch (Windows-specific example).

Step 5: Build Application Code in the Vitis Software Platform

The Vitis software platform launches in a separate window.





-	lab1_vitis - Vitis IDE Edit Developer Xilinx Project Window Help			- 0 ×
	Welcome 23			- a
	VITIS.			
		VITIS IDE		
		PROJECT	PLATFORM	RESOURCES
		Create Application Project	Add Custom Platform	Vitis Documentation
		Create Platform Project		QuickTake Videos
		Create Library Project		
		Import Project		
		4		

- 1. Close the Welcome screen if it appears.
- 2. Select File → New → Application Project or under Project click Create Application Project.

The New Application Project dialog box opens.



✓ New Application Project	_		×
Create a New Application Project			•
This wizard will guide you through the 4 steps of creating new application projects. 1. Choose a platform or create a platform project from Vivado exported XSA 2. Put application project in a system project , associate it with a processor 3. Prepare the application runnime – domain 4. Choose a template for application to quick start development			
Platform ProjectSystem Project			
Processor Domain App			
XSA			
 A platform provides hardware information and software environment settings. A system project contains one or more applications that run at the same time. A domain provides runtime for applications, such as operating system or BSP. A workspace can contain unlimited platforms and unlimited system projects. 			
Skip welcome page next time. (Can be reached with Back button)			
(?) < Back Next >	Finish	Cance	I

- 3. Select the **Skip welcome page next time** check box if you do not want the welcome to appear when the Vitis software platform is launched again.
- 4. Click Next.
- 5. In the Platform page, select the **Create a new platform from hardware (XSA)** tab.



Vew Application Project		_	- 0	×
Platform Please select a platform to create the project				•••
Select a platform from repository 🖫 Create a new platform from hardware (XSA)				^
Hardware Specification Provide your XSA file or use a pre-built board description vck190 zc702 zc705 zcu102 zed			Browse	
٠	< Back Next >	Finish	Car	icel

6. Click **Browse** to open The Create Platform from XSA window. Navigate to the directory where the XSA file was created in Vivado and click **Open**.

→ * ↑	nis PC → Windows (C:) → ug940 → Iab2	ت ~	Search lab2	لر
rganize 🔻 🛛 New fold	er			☷ ▾ 🔟 (
teraterm ^	Name	Date modified	Туре	Size
📕 ti	lab2.cache	7/6/2020 10:56 PM	File folder	
Titus47HF7	lab2.hbs	7/6/2020 10:55 PM	File folder	
TMR_MB	lab2.hw	7/6/2020 10:55 PM	File folder	
ug	lab2.ip_user_files	7/6/2020 10:57 PM	File folder	
ug940	lab2.runs	7/6/2020 10:58 PM	File folder	
lab1	lab2.sim	7/6/2020 10:55 PM	File folder	
lab1_vitis	lab2.srcs	7/6/2020 10:55 PM	File folder	
lab2 v	zynq_processor_system_wrapper.xsa	7/7/2020 7:01 AM	XSA File	925 KB
File n	ame: zynq_processor_system_wrapper.xsa		*.xsa;*.dsa;	```
			Open	Cancel

7. Ensure the **Generate boot Components** option is selected in the Platform page.



🚽 New Applic	ation Project	— C	ו	×
Platform Note: A platfo	m project will be generated automatically in workspace for the selected XSA. It can be customized later.		••••	
Select a	alatform from repository			^
Hardware	Specification C:\ug940\lab2\zynq_processor_system_wrapper.xsa			
XSA File:	vck190 zc702 zc706 zcu102 zed	Browse.		
	C:\ug940\lab2\zynq_processor_system_wrapper.xsa			
	ame: zynq_processor_system_wrapper e boot components			~
?	< Back Next > Finish	C	ancel	

- 8. Click Next.
- 9. In the Domain page leave all the fields at their default values and click **Next**.
- 10. In the Application project name field, type the name desired, such as peri_test. Leave all other fields to their default values, and click **Next**.



New Application Project			_)
plication Project Details pecify the application project name	and its system project properties			••	•••
pplication project name: peri_test	1				
	the application or select an existing or	ne from the workpsace 🛛 👔			
Select a system project	System project details System project name: peri Target processor Select target processor for th	·			
	Processor	Associated applications			
	ps7_cortexa9_0 ps7_cortexa9_1 ps7_cortexa9 SMP	peri_test			
	Show all processors in the h	ardware specification 🗹 👔			
2		< Back Next > Fin	ish	Cance	

11. In the Domain page leave all the fields at their default values and click **Next**.





Vew Application Project					×
Domain Select a domain for your project or create a new	domain				
Select the domain that the application would lin Note: New domain created by this wizard will ha			cted in the next step	1	
Select a domain	Domain details				
-	Name:	domain_ps7_cortexa9_0			
	Display Name:	domain_ps7_cortexa9_0			
	Operating System: Processor:	standalone	~		
	Architecture:	ps7_cortexa9_0 32-bit	~		
	Architecture	J2-Dit			
?	<	Back Next >	Finish	Cance	el

12. In the Templates page, select **Peripheral Tests**.





Vew Application Project							×
Templates						••	•
Select a template to create your project							
Available Templates:							
Find:	a 🖻	Ē	Peripheral Te	sts			
✓ SW development templates			Simple test ro	utines for all perip	herals in the hardware.		
Dhrystone							
Empty Application							
Empty Application (C++)							
Hello World							
IwIP Echo Server							
IwIP TCP Perf Client							
IwIP TCP Perf Server							
IwIP UDP Perf Client							
IwIP UDP Perf Server							
Memory Tests							
OpenAMP echo-test							
OpenAMP matrix multiplication	Demo						
OpenAMP RPC Demo							
Peripheral Tests							
RSA Authentication App							
Zynq DRAM tests							
Zynq FSBL							
0			c De els	Marchin	Finish	C	
(f)			< Back	Next >	Finish	Cancel	

13. Click Finish.

14. The application project is created in the Vitis software platform. Click the hammer icon so build the application.

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

♦ ♥ 18 01 21 = 14 1 et □ • ct • = n
Î

- 15. Ensure that you have connected the target board to the host computer and it is turned on.
- 16. Select and right-click the peri_test application in the Project Explorer, and select **Debug As** → **Debug Configurations**.

The Debug Configurations dialog box opens.

17. Right-click Single Application Debug, and select New Configuration.



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

🛯 Թ 🔛 🗶 📄 🦈 🖛	Name: Debugger_peri	test-Default				
e filter text Single Application Debug Debugger_peri_test-Default	🗶 Main 🔲 Applicati	ion 🔞 Targe	t Setup 🛛 🕬 Arguments 🔚 Environment 🛛	🖡 Symbol Files 🧤	Source 🚴 Path	Map) **
👫 Single Application Debug (GDB) 🔁 SPM Analysis	Hardware Platform:	\${sdxTcfLau	nchFile:project=peri_test;fileType=hw;}	Search	Browse]
	Bitstream File:	_ide/bitstre	am/zynq_processor_system_wrapper.bit	Search	Browse	Generate
	FPGA Device:	Auto Detect	1	Select		
	PS Device:	Auto Detect	1	Select		
	Use FSBL flow for	initialization				
	Initialization File:	_ide/psinit/	ps7_init.tcl	Search	Browse]
			Summary:			
	Reset entire syste Program FPGA Skip Revision (Kun ps7_init Run ps7_post_cor Enable Cross-Trig	Check	Following operations will be performed befo 1. Resets entire system. Clears the PPGA fabe 2. Program FPGA fabric (PL). 3. Runs ps7_init to initialize PS. 4. Runs ps7_post_config. Enables level shifter only after system reset or board power ON). 5. All processors in the system will be susper following processors as specified in the Appl 1) ps7_cortexa9_0 (C/tutorials/2019.2/UG9	ic (PL). rs from PL to PS. (Rec nded, and Application lications tab.	commended to us	ided to the

19. Click the **Browse** button for Enable Cross-Triggering option.

The Cross Trigger Breakpoints dialog box opens.

- 20. Click Create.
- 21. In the Create Cross Trigger Breakpoint page, select the options as shown in the following figure.



Cross Trigger Breakpoint	,
Configuration is valid.	
Cross Trigger Signals (Input)	Cross Trigger Signals (Output)
✓ ■ CPU-0	✓ □ CPU-0
CPU0 DBGACK [0]	CPU0 debug request [0]
CPU0 PMU IRQ [1]	PTM0 EXT 0 [1]
PTM0 EXT 0 [2]	PTM0 EXT 1 [2]
PTM0 EXT 1 [3]	PTM0 EXT 2 [3]
CPU0 COMMTX [4]	PTM0 EXT 3 [4]
CPU0 COMMRX [5]	CPU0 restart request [7]
PTM0 TRIGGER [6]	✓ □ CPU-1
✓ □ CPU-1	CPU1 debug request [8]
CPU1 DBGACK [8]	PTM1 EXT 0 [9]
CPU1 PMU IRQ [9]	PTM1 EXT 1 [10]
PTM1 EXT 0 [10]	PTM1 EXT 2 [11]
PTM1 EXT 1 [11]	PTM1 EXT 3 [12]
CPU1 COMMTX [12]	CPU1 restart request [15]
CPU1 COMMRX [13]	✓
PTM1 TRIGGER [14]	ETB flush [16]
✓	ETB trigger [17]
ETB full [18]	TPIU flush [18]
ETB acquisition complete [19]	TPIU trigger [19]
ITM trigger [20]	✓ ✓ FTM
✓ □ FTM	FTM trigger 0 [24]
FTM trigger 0 [24]	FTM trigger 1 [25]
FTM trigger 1 [25]	FTM trigger 2 [26]
FTM trigger 2 [26]	FTM trigger 3 [27]
FTM trigger 3 [27]	
(2)	OK Cancel

- 22. Click **OK**. This sets up the cross trigger condition for Processor to Fabric.
- 23. In the Cross Trigger Breakpoints dialog box, click **Create**, as shown in the following figure.

•	iross Trigger Breakpoints	×
	Name	Create
Ø	Inputs (CPU0 DBGACK [0]) - Ouputs (FTM trigger 0 [24], FTM trigger 1 [25], FTM trigger 2 [26], FTM trigger 3 [Edit
		Remove
	OK	Cancel

24. In the Create Cross Trigger Breakpoint page, select the options as shown in the following figure.



V Cross Trigger Breakpoint	>
Create Cross Trigger Breakpoint Configuration is valid.	
Cross Trigger Signals (Input)	Cross Trigger Signals (Output)
CPU0 DBGACK [0]	CPU0 debug request [0]
CPU0 PMU IRQ [1]	
PTM0 EXT 0 [2]	PTM0 EXT 1 [2]
PTM0 EXT 1 [3] CPU0 COMMTX [4]	PTM0 EXT 2 [3]
	CPU0 restart request [7]
PTM0 TRIGGER [6]	CPOU restart request [7] ✓ □ CPU-1
✓ □ CPU-1	CPU1 debug request [8]
CPU1 DBGACK [8]	PTM1 EXT 0 [9]
	PTM1 EXT 1 [10]
□ PTM1 EXT 0 [10]	□ PTM1 EXT 2 [11]
□ PTM1 EXT 1 [11]	PTM1 EXT 3 [12]
CPU1 COMMTX [12]	CPU1 restart request [15]
CPU1 COMMRX [13]	✓ □ ETB and TPIU
PTM1 TRIGGER [14]	ETB flush [16]
✓ □ ETB and TPIU	ETB trigger [17]
ETB full [18]	TPIU flush [18]
ETB acquisition complete [19]	TPIU trigger [19]
ITM trigger [20]	✓ □ FTM
✓ ✓ FTM	FTM trigger 0 [24]
FTM trigger 0 [24]	FTM trigger 1 [25]
FTM trigger 1 [25]	FTM trigger 2 [26]
FTM trigger 2 [26]	FTM trigger 3 [27]
FTM trigger 3 [27]	
?	OK Cancel

- 25. Click **OK**. This sets up the cross trigger condition for Fabric to Processor.
- 26. In the Cross Trigger Breakpoints Dialog box, click **OK**.

Ø	Name Inputs (CPU0 DBGACK [0]) - Ouputs (FTM trigger 0 [24], FTM trigger 1 [25], FTM trigger 2 [26], FTM trigger 3 [Inputs (FTM trigger 0 [24], FTM trigger 1 [25], FTM trigger 2 [26], FTM trigger 3 [27]) - Ouputs (CPU0 debug re	Create Edit Remove
	OK P	Cancel

27. In the Debug Configurations dialog box, click **Debug**, as shown at the bottom of the following figure.



er und Charles and Charles and an and a supervised to the supervis	3 🖻 🕫 🔛 🗶 🗎 🖘 •	Name: Debugger_peri	test-Default				
Single Application Debug (GD8) Hardware Pletform: §(sdxTcfLaunchFile:project=peri_test;fileType=hw;) Search Browse Sight Analysis Bitstream File: _ide/bitstream/zynq_processor_system_wrapper.bit Search Browse Generate FPGA Device: Auto Detect Select Select Select Vertice: Select Use FSBL flow for initialization Initialization File: ide/psinit/ps7_init.tcl Search Browse Program FPGA Summary: Following operations will be performed before launching the debugger. Program FPGA Program FPGA fabric (PL). Program FPGA fabric (PL). <td< th=""><th>ype filter text 🖌 🛼 Single Application Debug</th><th>🗶 Main 🥅 Applicati</th><th>on 🛞 Target</th><th>Setup 🛛 60- Arguments 🚟 Environment 📷</th><th>Symbol Files 🦗 Sourc</th><th>e) 🐣 Path Map [</th><th>Common</th></td<>	ype filter text 🖌 🛼 Single Application Debug	🗶 Main 🥅 Applicati	on 🛞 Target	Setup 🛛 60- Arguments 🚟 Environment 📷	Symbol Files 🦗 Sourc	e) 🐣 Path Map [Common
Distream File ide/Eststeam/zynq_processor_system_wrapper.oit Search Drowse <	Single Application Debug (GD8)	Hardware Platform:	\$(sdxTcfLau	nchFile:project=peri_test;fileType=hw;}	Search	Browse	
PS Device: Auto Detect Select. Use FSBL flow for initialization Initialization Initialization File: ide/psinit/ps7_init.tcl Search Browse Search Browse Browse Following operations will be performed before launching the debugger. 1. Program FPGA Following operations will be performed before launching the debugger. 1. Reset entire system 1. Skip Revision Check Run ps7_init 3. Runs ps7_init in initialize PS. 1. Run ps7_init 3. Search conting 1. 1. 1. Following operating of the performed before launching the debugger. 1. 1. 1. 1. Sector Stipper beaches the operations will be performed before launching the debugger. 1. 1. 1. 1. 1. 1. 1. 2. 1. 2. 1. 2. 1. 2. 1. 2. 1. 2. 1. 2. 1. 2. 1. 2. 1. 2. 1. 2. 1. 2. 1. 2. 1. 2. 1. 2. 1. 2. 1.	🔤 SPM Analysis	Bitstream File:	_ide/bitstrea	m/zynq_processor_system_wrapper.bit	Search	Browse	Generate
Use FSBL flow for initialization Initialization File: jde/psink/ps7_init.tcl Summary: Search Program FPGA Following operations will be performed before launching the debugger. Skip Revision Check Run ps7_init Run ps7_init Sums ps7_post_config Enable Cross-Triggering Inputs (CPU0 DBGACK (0)) - Ouputs (FTM trigger 0 [24], FTM trigger 2 [26], FTM trigger 3 [27]) - Ouputs (CPU0 debug request [0) 6. All processors as specified in the Applications tab. Supplications will be suspended, and Applications will be downloaded to the following processors as specified in the Applications tab.		FPGA Device:	Auto Detect		Select		
Initialization File: jde/psinit/ps7_init.tcl Search Browse Initialization File: jde/psinit/ps7_init.tcl Search Browse Program FPGA Following operations will be performed before launching the debugger. I. Reset entire system. Clears the FPGA fabric (PL). Program FPGA Skip Revision Check Run ps7_init Num sp7_post_config Init onitialize PS. Ratus ps7_post_config Init onitialize PS. Enable Cross-Triggering Image: Plousing CPU and PGACK (D) - Ouputs (FPM trigger 0 [24], FTM trigger 1 [25], FTM trigger 2 [26], FTM trigger 3 [27]) - Ouputs (CPU0 debug request [0]) Ouputs (FTM trigger 0 [24], FTM trigger 3 [27]) - Ouputs (CPU0 debug request [0]) All processors as specified in the Applications tab. Sum specified in the Applications tab.		PS Device:	Auto Detect		Select		
Summary: Program FPGA Program FPGA Run ps7_init Run ps7_init Run ps7_init Program FPGA Init control (PL) Program FPGA Program FPGA Run ps7_init Program FPGA		Use FSBL flow for	initialization				
Reset entire system Following operations will be performed before launching the debugger. Program FPGA I. Resets entire system. Clears the PPGA fabric (PL). Skip Revision Check Program FPGA fabric (PL). Run ps7_init I. Runs ps7_iont_config. Run ps7_post_config Following cross trigger breakpoints will be added. Inputs (CPU0 DBGACK (D) - Ouputs (FTM trigger 0 [24], FTM trigger 1 [25], FTM trigger 2 [26], FTM trigger 3 [27]) - Ouputs (CPU0 dBGACK (D) - Ouputs (CPU0 dBGACK (D) - Guputs (D) - Guputs (CPU0 dBGACK (D) - Guputs (D) - Gup		Initialization File:	_ide/psinit/p	ps7_init.tcl	Search	Browse	
 Program FPGA Resets entire system. Clears the FPGA fabric (PL). Program FPGA Skip Revision Check Run ps7_init Run ps7_post_config Run ps7_post_config Enable Cross-Triggering Inputs (CPU0 DBGACK (0) - Ouputs (FTM trigger 0 [24], FTM trigger 1 [25], FTM trigger 2 [26], FTM trigger 3 [27]) Ouputs (CPU0 dBGACK (0) - Ouputs (FTM trigger 2 [26], FTM trigger 3 [27]) Ouputs (CPU0 dBGACK (0) - Support [25], FTM trigger 3 [27]) Ouputs (CPU0 debug request [0) All processors as specified in the Applications will be downloaded to the following processors as specified in the Applications tab. 				Summary			
		Program FPGA Skip Revision O Run ps7_init Run ps7_post_co	iheck nfig	Resets entire system. Clears the FPGA fabric 12. Program FPGA fabric (PL). Runs ps7_init to initialize PS. ARuns ps7_init to initialize PS. ARuns ps7_post_config. Enables level shifters 1 system reset or board power ON. Following cross trigger breakpoints will be an 1) Inputs (CPU0 DBGACK (0)) - Ouputs (FTM trigger 3 [27) 2) Inputs (FTM trigger 0 [24], FTM trigger 1 [debug request [0]) 6. All processors in the system will be suspende processors as specified in the Applications tab.	(PL). from PL to PS. (Recomm dded. I trigger 0 [24], FTM trigg 25], FTM trigger 2 [26], F ed, and Applications will	nended to use this o per 1 (25), FTM trigg TM trigger 3 (27)) - be downloaded to	ger 2 [26], FTM Ouputs (CPU0

28. The Debug Perspective Window opens.

le Edit Run Developer Xilinx Project Window	Help				
3 • 🗟 🐚 🖲 • 🗞 • 🕸 • O • 🖻 🔪	▶ 0 = ₩ 3. 0 .0 = 次 🖉 🔳 🖸		Q	ick Access	Design 🔯 Debu
🏚 Debug 🗵 🦹 🎽 👻 🗖 🗖	🚊 peri_test_system 🛠 peri_test 🔀 testperiph.c 🛙 👘		10=V., 🔅 🗣 8.,	9/ E 🛋 M	887 R
F. Debugger perites-Default (Local)	<pre>53 #include "ttos_header.h" 54 fitt with an () 53 { 54 fitt with an () 53 { 54 fitt with an () 55 fitt Xtoolic int; 55 static Xtoolic int; 56 static Xtoolic pay settier.0; 57 static Xtoolic pay static Xtoolic pay fitt () 58 static Xtoolic pay fitt,0; 59 static Xtoolic pay fitt,0; 50 fitt Status; 59 print("Entering main\n\r"); 51 fit Status; 52 fit Status == 0 { 53 static ScolicSelffestExample (PASED\r\n"); 54 } 55 else { 56 static ScolicSelffestExample FAILED\r\n"); 57 print("ScolicSelffestExample FAILED\r\n"); 57 } 58 status; 59 status; 50 status == 0 { 50 status == 0 { 51 status == 0 { 52 status == 0 { 53 status == 0 { 53 status == 0 { 54 status == 0 { 55 status =</pre>	•	> @ ps7_scutime > @ ps7_scuwdt > @ ps7_ttc_0	Type XScuGic XCanPs XScuTimer XScuWidt XStuPs	Config-OD Con Config-OD Config-OD Config-OD Config-OD Con
R icps_header.h R aspips header.h	🕒 Console 💷 🖨 Vitis Terminal 🚯 Executables 📗 Vitis Log 🖹 Problems 🙀 Debugger Console			ons 🕸 🖬 Emu	
> h scugic_header.h		•			í
> A scutimer_header.h	TCF Debug Virtual Terminal - ARM Cortex-A9 MPCore #0		A Info: AR	Cortex-A9 MPC	ore #0 (targe
> R scuwdt_header.h				t/arc/teatpe	
> h ttcps_header.h			61: X xsct%	<pre>il_ICacheEnable</pre>	01
> R xcanps_intr_example.c			<		>
/ M wanted instantibute					

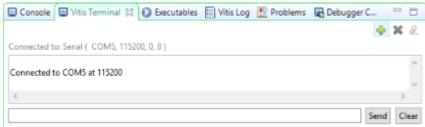
29. Set the terminal by selecting the **Vitis Terminal** tab, and then click the (+) icon.



30. Select following settings in the following figure for the ZC70 board, and click OK.

🚽 Connect to	serial port	\times
Basic Settin	gs	
Port:	COM5	~
Baud Rate:	115200	\sim
▼ Advance	Settings	
Data Bits:	8	~
Stop Bits:	1	\sim
Parity:	None	\sim
Flow Contr	ol: None	\sim
Timeout (s	ec):	
OK	Cancel	

31. Verify the terminal connection by checking the status at the top of the Vitis Terminal tab as shown in the following figure.

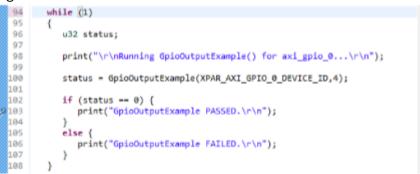


- 32. If it is not already open, select **../src/testperiph.c**, and double click to open the source file.
 - a. Modify the source file by inserting a while statement at approximately line 94.
 - b. After the else statement, add while(1) above in front of the curly brace as shown in the following figure.



```
while (1)
95
        ſ
           u32 status;
96
97
98
           print("\r\nRunning GpioOutputExample() for axi_gpio_0...\r\n");
99
100
           status = GpioOutputExample(XPAR_AXI_GPIO_0_DEVICE_ID,4);
101
           if (status == 0) {
    print("GpioOutputExample PASSED.\r\n");
102
103
104
           else {
105
              print("GpioOutputExample FAILED.\r\n");
196
           3
107
108
```

- 33. Add a breakpoint in the code so that the processor stops code execution when the breakpoint is encountered.
 - a. Scroll down to the line after the "while" statement starts, and double-click the left pane on line 103, which adds a breakpoint on that line of code, as it appears in the following figure.



- b. Click Ctrl + S to save the file. Alternatively, you can select File \rightarrow Save.
- 34. Click on the hammer icon 🍝 again to rebuild the application with the changes made to the code.

Now you are ready to execute the code from the Vitis software platform.

Step 6: Connect to Vivado Logic Analyzer

Connect to the ZC702 board using the Vivado Logic Analyzer.

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



No hardware target is open. O	pen tar	get	
Hardware Q ≚ ⊕ Ø ▶	8	Auto Connect Recent Targets Available Targets on Server	•
		Open New Target	

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

The Open New Hardware Target wizard opens, shown in the following figure.

Open New Hardwar	e Target	8
	Open Hardware Target This wizard will guide you through connecting to a hardware target. To connect to a remote hardware target, provide the host name and IP port of the remote machi instance of a Vivado Hardware Server is running.	ne on which the
•	< Back Next >	Cancel

3. Click Next.

				rver Settings	ardware Se
s an	the	Ise Local server if the		emote hardware server, then configure the hos d to the local machine; otherwise, use Remote	
			e server.	u to the local machine, otherwise, use Remoti	rgeris allacite
			~	Local server (target is on local machine)	Connect to:
			-	Local server (target is of rocal machine)	Zonneci to.
		al machine.	121) appl	launch and/or connect to the hw_server (port 3	Click Next to
		[2
3	Ca	Einish	ack	< 8	?)

4. On the Hardware Server Settings page, ensure that the Connect to field is set to **Local server** (target is on local machine) as shown in the following figure, and click Next.



	rver Settings					
	mote hardware server, then to the local machine; othen			nd port settings	Use Local server if the	- P
arger is allached	to the local machine, other	wise, use Kei	HOLE SELVEL.			
Connect to:	Local server (target is on lo	cal machine)	~			
Click Next to I	aunch and/or connect to the	hw server (p	ort 3121) appl	lication on the lo	cal machine	
2		_		Next>		

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

	are Target				
				et the appropriate JTAG clock (TCK) freque select a different target.	uency. If 🛛 卢
lardware <u>T</u> arg	jets				
Туре	e Name		JTAG Clock Fre	uency	
📓 xilinx_tcf	Digilent/210203	342363A	15000000	~	
lardware <u>D</u> evi	ces (for unknow	n devices,	specify the Instr	ction Register (IR) length)	
Name	ID Code	IR Le			
arm_dap_		4			
# xc7z020_1	03727093	6			
	er: localhost312				

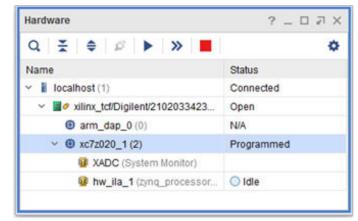
6. Ensure that all the settings are correct on the Open Hardware Target Summary page, as shown in the following figure, and click **Finish**.



Open New Hardward	open Hardware Target Summary
HLY Editions	Hardware Server Settings: Server: localhost3121
XILINX	 Target Settings: Target: xilinx_tcf/Digilent/210203342363A Frequency: 15000000
ALL PROGRAMMABLE.	To connect to the hardware described above, click Finish
?	< <u>B</u> ack <u>N</u> ext> Einish Cancel

Step 7: Set the Processor to Fabric Cross Trigger

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



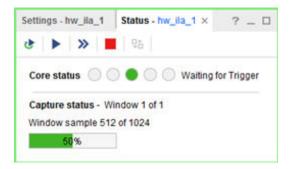
- 1. Select the ILA hw_ila_1 tab, and set the Trigger Mode Settings as follows:
 - Set Trigger mode to TRIG_IN_ONLY
 - Set TRIG_OUT mode to TRIG_IN_ONLY
 - Under Capture Mode Settings, ensure Trigger position in window is set to 512.



											200
Waveform - hw_ila_1											? _ 0)
Q + - e > 3	» 🖪 🖪 🔍 🔍	* -	H H	2 21	• I+ +I	H .					<
ILA Status Idle			0								
Name		Value	0	11	1z	(a	14	15	je.	17	10
> Mizjing_processor_system_i/											
slot_0:ps7_0_axi_periph_M0											1.1.1
> Slot_0:ps7_0_avi_periph_	M00_AVI: R Channel										
) Solot_0:ps7_0_axi_periph_	M00_A01: AW Channel										
	ang										
		$\langle \rangle$	6								
Settings , hur da t y Clubo	n hw da 4	2 0	Tripper S	etun two its	· · Canturn 1	Letter - Inv. ita -1					2
Settings - hw_Ba_1 × Statu Trigger Mode Settings	s-tw_la_1	7 - 0		etup - hw_ila_ - D	• × Capture !	ietup - tw_ila_1					? _
Trigger Mode Settings	n_only v	? - ¤				ietup low_lia_1					7 = 1
Trigger Mode Settings	N_ONLY ~	? - ¤				ietop - low_ila_1					7 = 1
Trigger Mode Settings Trigger mode: TRIG_P	N_ONLY ~	? - 0				ietup - bw_ila_1					7 = 1
Trigger Mode Settings Trigger mode: TRIG_P	N_ONLY ~	7 - 0									7 = 1
Trigger Mode Settings Trigger mode: TRIG_M TRIG_OUT mode: TRIG_P	N_ONLY ~	7 - 0					the + bullook	add protes.			? = 1
Trigger Mode Settings Trigger mode: TRIG_IP TRIG_OUT mode: TRIG_IP Capture Mode Settings Capture mode:	n_only v	? = 0					ume + bullook	add prodes.			? = 1
Trigger Mode Settings Trigger mode: TRIG_IP TRIG_OUT mode: TRIG_IP Capture Mode Settings Capture mode:		? _ 0					i the 🕂 button to	add probes.			? = 1
Trigger Mode Settings Trigger mode: TRIG_IP TRIG_OUT mode: TRIG_IP Capture Mode Settings Capture mode: Number of windows:	N_ONLY • N_ONLY • ALVEATE • 1 [1 - 1024]	? _ 0					i the 🕇 buttoo to	add prodes.			? _

2. Arm the ILA core by clicking the **Run Trigger** button 🕨 .

This arms the ILA and you should see the status "Waiting for Trigger" as shown in the following figure.



- 3. In the Vitis software platform Debug window, click the **Resume** button in the toolbar, until the code execution reaches the breakpoint set on line 103 in the testperiph.c file.
- 4. As the code hits the breakpoint, the processor sends a trigger to the ILA. The ILA has been set to trigger when it sees the trigger event from the processor. The waveform window displays the state of various signals as seen in the following figure.



Waveform - hw_ila_1		? _	
Q + - e 🕨 🕷 🖪 🛛 🔍	Q X 4	(H H 2 2 4 F 4 H	4
ILA Status: Ide			
Neme	Value	o 200 400 feco pos	ŀ
> ¥ zyng_processor_systemT_0_CPI0_Hi_0_1(3.0	0		
Sistep1: avj.amc.001_A01.Interface	Inadies	Inc. ite	
Sist_1:ai_smc_W01_A0:AR Channel	No Read Addr Cr	Be level affer Carls	
> % skit_1:avi_smc_W01_X0: R Channel	No Read Data Be	Er Dead Data Deatr	
V % sld_1: wi_smc_W01_A0. A0 Channel	No Write Add: Cri	In Wolle Loop Cade	
skt_1:ai_smc_N01_A0:AWAUD	0		
\$ skt_1: avi_smc_W01_XXI: AWREADY	0		
> 🗑 skt_1: atj.smc_N01_AXI: ANADDR	000		
	$\langle = \rangle$	Tpdatest at: 2229-049-08-09:421-68 C	

This demonstrates that when the breakpoint is encountered during code execution, the PS7 triggers the ILA that is set up to trigger. The state of a particular signal when the breakpoint is encountered can be monitored in this fashion.

Step 8: Set the Fabric to Processor Cross-Trigger

Now try the fabric to processor side of the cross-trigger mechanism. To do this remove the breakpoint that you set earlier on line 103 to have the ILA trigger the processor and stop code execution.

1. In the Vitis software platform, select the Breakpoints tab towards the top right corner of the window, right-click it, and clear the **testperiph.c** [line: 103] check box. This removes the breakpoint that you set up earlier.

Note: Alternatively, you can select the breakpoint in line 103 of the testperiph.c file, right click and select Disable Breakpoint.

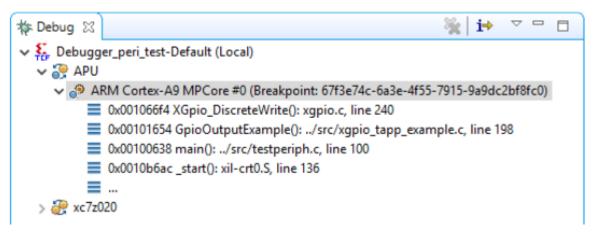
2. In the Debug window, click the Resume

You can see the code executing in the Terminal Window. icon on the toolbar. The code runs continuously because it has an infinite loop.

- 3. In Vivado, select the **Settings hw_ila_1** tab. Change the Trigger Mode to **BASIC_OR_TRIG_IN** and the TRIG_OUT mode to **TRIGGER_OR_TRIG_IN**.
- 4. Click on the + sign in the Trigger Setup window to add the slot_1:axi_smc_M01_AXI:AWVALID signal from the Add Probes window.
- 5. In the Basic Trigger Setup window, ensure that the Radix is set to [B] Binary, and change the Value for the slot_0:ps7_0_axi_periph_M00_AXI:AWVALID signal to 1. This essentially sets up the ILA to trigger when the awvalid transitions to a value of 1.
- 6. Click the Run Trigger button to "arm" the ILA. It triggers immediately as the Vitis code is running AXI transactions which causes the <code>awvalid</code> signal to toggle. This causes the <code>trigger_out</code> of the ILA to toggle which eventually will halt the processor from executing the code.



This is seen in the Vitis software platform in the highlighted area of the debug window.



Conclusion

This lab demonstrated how cross triggering works in a Zynq-7000 SoC processor based design. You can use cross triggering to co-debug hardware and software in an integrated environment.

Lab Files

This tutorial demonstrates the cross-trigger feature of the Zynq-7000 SoC processor, which you perform in the GUI environment. Therefore, the only Tcl file provided is lab2.tcl.

The lab2.tcl file helps you run all the steps all the way to exporting hardware for the Vitis software platform.

The debug portion of the lab must be carried out in the GUI; no Tcl files are provided for that purpose.





Lab 3

Programming an Embedded MicroBlaze Processor

Introduction

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

The MicroBlaze system includes native Xilinx[®] IP including:

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

Parts of the block design are constructed using the Platform Board Flow feature.

This lab also shows the cross-trigger capability of the MicroBlaze processor.

The feature is demonstrated using a software application code developed in the Vitis software platform in a stand-alone application mode.

This lab targets the Xilinx KC705 FPGA Evaluation Board.



Step 1: Start the Vivado IDE and Create a Project

- 1. Start the Vivado IDE by clicking the Vivado desktop icon or by typing vivado at a command prompt.
- 2. From the Quick Start page, select Create Project.
- 3. In the New Project dialog box, use the following settings:
 - a. In the Project Name dialog box, type the project name and location.
 - b. Make sure that the Create project subdirectory check box is selected. Click Next.
 - c. In the Project Type dialog box, select **RTL project**. Ensure that the **Do not specify sources at this time** check box is cleared. Click **Next**.
 - d. In the Add Sources dialog box, set the Target language to either **VHDL** or **Verilog**. You can leave the Simulator language selection to **Mixed**.
 - e. Click Next
 - f. In Add Constraints dialog box, click Next.
 - g. In the Default Part dialog box, select **Boards** and choose **Kintex-7 KC705 Platform** along with the current version. Click **Next**.
 - h. Review the project summary in the New Project Summary dialog box and click **Finish** to create the project.

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

```
set_property board part xilinx.com:kc705:part0:1.6
[current_project]
```

Although Tcl commands are available for many of the actions performed in the Vivado IDE, they are not explained in this tutorial. Instead, a Tcl script is provided that can be used to recreate this entire project. See the Tcl Console for more information. You can also refer to the *Vivado Design Suite Tcl Command Reference Guide* (UG835) for information about the write_bd_tcl commands.



Step 2: Create an IP Integrator Design

- 1. From Flow Navigator, under IP integrator, select Create Block Design.
- 2. Specify the IP subsystem design name. For this step, you can use mb_subsystem as the Design name. Leave the Directory field set to its default value of <Local to Project>. Leave the Specify source set drop-down list set to its default value of Design Sources.
- 3. Click OK in the Create Block Design dialog box, shown in the following figure.

🔥 Create Block De	sign	×
Please specify name	of block design.	A
<u>D</u> esign name:	mb_subsystem	8
D <u>i</u> rectory:	🛜 <local project="" to=""></local>	~
Specify source set:	Design Sources	~
?	ОК	Cancel

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

The IP integrator Catalog opens. Alternatively, you can also select the Add IP icon in the middle of the canvas.

This design is empty. Press the 🕂 button to add IP.

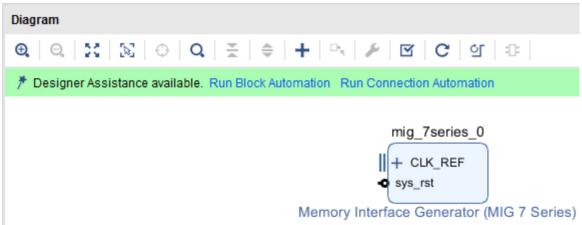
5. Type mig in the Search field to find the MIG core, then select **Memory Interface Generator** (MIG 7 Series), and press Enter.

Search: 🔍 mig 🛞 (1 match)		IP Details	×
Search: Q- mig (1 match) Memory Interface Generator (MIG 7 Series)	Name: Version: Interfaces: Description:	Memory Interface Generator (MIG 7 Series) 4.2 (Rev. 1) AXI4	
		SDRAM, QDR II+ SRAM, RLDRAMII and RLDRAMIII. Artix-7 supports DDR3 SDRAM, DDR2 SDRAM and LPDDR2 SDRAM. Zynq supports DDR3 SDRAM, DDR2 SDRAM and LPDDR2 SDRAM	
ENITED to collect. ECC to concel. Ctd., O for ID dataile	Status:	Production	
ENTER to select, ESC to cancel, Ctrl+Q for IP details	License	Included	



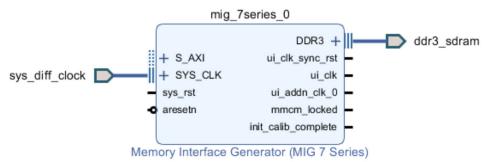
The Designer Assistance link becomes active in the block design banner.

6. Click **Run Block Automation**.



The Run Block Automation dialog box opens.

7. Click **OK**. This instantiates the MIG core and connects the I/O interfaces to the I/O interfaces for the DDR memory on the KC705 board.



8. Right-click anywhere in the block design canvas, and select Add IP.

The IP catalog opens.

9. In the Search field, type micr to find the MicroBlaze IP, then select **MicroBlaze**, and press **Enter**.

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



Search: Q- micr (3 matches)		IP Details	×
MicroBlaze	Name:	MicroBlaze	î
MicroBlaze Debug Module (MDM)	Version:	11.0 (Rev. 1)	
MicroBlaze MCS	Interfaces:	AXI4, AXI4-Stream	
	Description:	The MicroBlaze 32 and 64 bit soft processor core, providing an instruction set optimized for embedded applications with many user-configurable options. MicroBlaze has many advanced architecture features like Instruction and Data-side cache with AXI interfaces, Floating-Point unit (FPU), Memory Management Unit (MMU), and fault tolerance support. It is highly recommended to create MicroBlaze systems within Vivado IP Integrator, to enable export to the Xilinx Software Development Kit (SDK) for software development.	
	Status:	Production	
ENTER to select, ESC to cancel, Ctrl+Q for IP details	License:	Included	~

Use the Board Window to Connect to Board Interfaces

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

1. Select the **Board** window to see the interfaces present on the KC705 board.

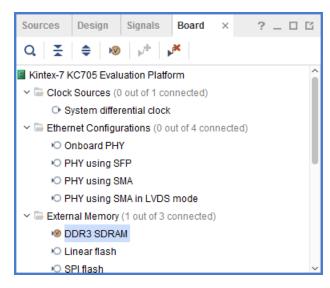




Board ? _ D 2 ×
$\mathbf{Q} \mid \mathbf{X} \mid \clubsuit \mid \mathbf{W} \mid \mathbf{p}^{\oplus} \mid \mathbf{p}^{\mathbb{N}}$
Kintex-7 KC705 Evaluation Platform
Clock Sources (0 out of 1 connected)
 System differential clock
Ethernet Configurations (0 out of 4 connected)
Onboard PHY
PHY using SFP
PHY using SMA
PHY using SMA in LVDS mode
External Memory (1 out of 3 connected)
100 DDR3 SDRAM
💫 Linear flash
© SPI flash
General Purpose Input or Output (0 out of 5 connected)
DIP switches
ICD
© LED
Push buttons
Rotary switch
 Miscellaneous (0 out of 3 connected)
© IIC
PCI Express
© UART
Reset (0 out of 1 connected)
○ FPGA Reset

In the Board window, notice that the DDR3 SDRAM interface is connected as shown by the circle ¹⁰ in the following figure. This is because you used the Block Automation feature in the previous steps to connect the MIG core to the board interfaces for DDR3 SDRAM memory.





2. From the Board window, select **UART** under the miscellaneous folder, and drag and drop it into the block design canvas.

This instantiates the AXI Uartlite IP on the block design.

3. Likewise, from the Board window, select **LED** under the General Purpose Input or Output folder, and drag and drop it into the block design canvas.

This instantiates the GPIO IP on the block design and connects it to the on-board LEDs.

The block design now should look like the following figure.

Diagram	? _ D @ X
$oldsymbol{G} \mid oldsymbol{G} \mid oldsymbol{S} \mid oldsymbol{S} \mid oldsymbol{G} \mid oldsymbol{G} \mid oldsymbol{S} \mid oldsymbol{S} \mid oldsymbol{G} \mid oldsymbol{S} \mid oldsymbol{G} \mid oldsymbol{S} \mid oldsymbol{G} \mid oldsymbol{G$	٥
* Designer Assistance available. Run Block Automation Run Connection Automation	
microblaze_0 mig_7series_0 mig_7series_0 mig_7series_0 DDR3 + + S_AXI ui_ck_sync_rst - sys_rst ui_addn_ck_0 mig_7series_0 - sys_rst ui_addn_ck_0 - aresetn mmcm_locked - init_calib_complete Memory Interface Generator (MIG 7 Series)	ddr3_sdram
axi_uartite_0 + S_AXI UART +	rs232_uart
axi_gpio_0 + S_AXI s_axi_acik GPIO + s_axi_aresetin AXI GPIO	led_8bits



Add Peripheral: AXI block RAM Controller

1. Add the AXI block RAM Controller, shown in the following figure, by right-clicking the IP integrator canvas and selecting **Add IP**.

Search: Q· axi bram (1 match)		IP Details	×
* AXI BRAM Controller	Name: Version: Interfaces: Description:	AXI BRAM Controller 4.1 (Rev. 2) AXI4 The AXI Block RAM (BRAM) Controller is a soft IP core. The core is designed as an AXI Endpoint slave IP for integration with the AXI interconnect and system master devices to communicate to local block RAM. The core supports both single and burst transactions to the block RAM and is optimized for performance.	Î
ENTER to select, ESC to cancel, Ctrl+Q for IP details	Status:	Production	~

Run Block Automation

1. Click Run Block Automation, as shown below.



The Run Block Automation dialog box opens.

- 2. On the Run Block Automation dialog box:
 - a. Leave Preset as the default value, None.
 - b. Set Local Memory to 64 KB.
 - c. Leave the Local Memory ECC as the default value, None.
 - d. Set Cache Configuration to 32 KB.
 - e. Set Debug Module to Extended Debug.
 - f. Leave the Peripheral AXI Port option as the default value, **Enabled**.
 - g. Leave the Clock Connection option set to /mig_7series_0/ui_addn_clk_0 (100 MHz).

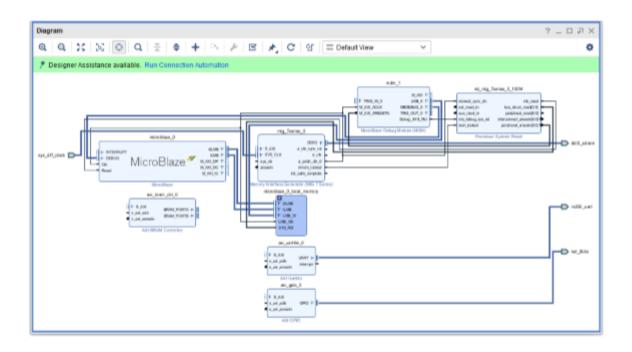


Q 素 ♦	Description	
 All Automation (1 out of 1 selected) Image: microblaze_0 	caches can be configured Interrupt Controller, a clo connected as needed. A	utomation generates local memory of selected size, and d. MicroBlaze Debug Module, Peripheral AXI Interconnect, tock source, Processor System Reset are added and preset MicroBlaze configuration can also be selected. otions can be found in the tooltips.
	Preset Local Memory Local Memory ECC Cache Configuration Debug Module Peripheral AXI Port Interrupt Controller Clock Connection	None 64KB None 32KB Extended Debug Enabled /mig_7series_0/ui_addn_clk_0 (100 MHz)

3. Click OK.

This generates a basic MicroBlaze system in the IP integrator diagram area, as shown in the following figure.

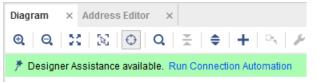




Use Connection Automation

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

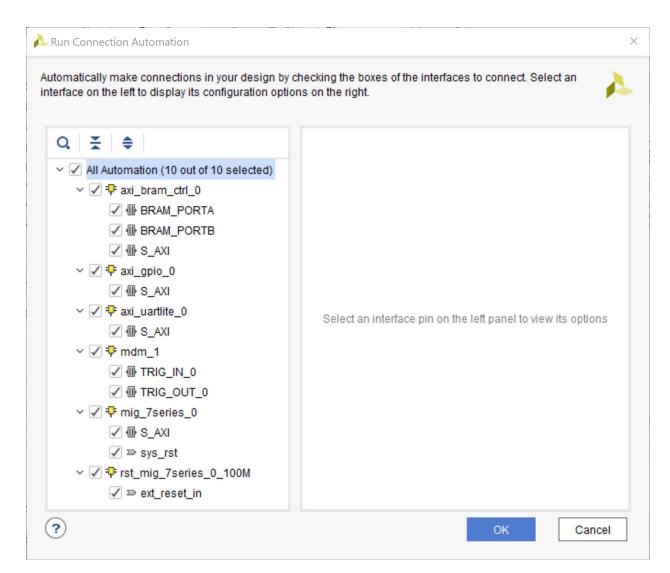
1. Click Run Connection Automation as shown in the following figure.



The Run Connection Automation dialog box opens.

2. Check the interfaces in the left pane of the dialog box as shown in the following figure.





3. Use the following table to set options in the Run Connection Automation dialog box.

Table 1: Run Connection Automation Options

Connection	More Information	Setting
axi_bram_ctrl_0 • BRAM_PORTA	The only option for this automation is to instantiate a new Block Memory Generator as shown under options.	Leave the Blk_Mem_Gen to its default option of Auto.



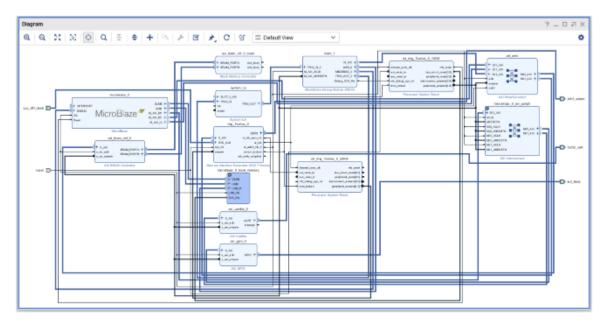
Connection	More Information	Setting		
axi_bram_ctrl_0 • BRAM_PORTB	 The Run Connection Automation dialog box opens and gives you two choices: Instantiate a new BMG and connect the PORTB of the AXI block RAM Controller to the new BMG IP Use the previously instantiated BMG core and automatically configure it to be a true dualported memory and connected to PORTB of the AXI block RAM Controller. 	Leave the Blk_Mem_Gen option to its default value of Auto. The Run Connection Automation dialog box offers to connect this to the /microblaze_0 (Cached). Leave it to its default value. In case, cached accesses are not desired this could be changed to /microblaze_0 (Periph). Leave the Clock Connection (for unconnected clks) field set to its default value of Auto. Keep these default settings.		
axi_bram_ctrl_0 • S_AXI	Two options are presented in this case. The Master field can be set for either cached or non-cached accesses.			
axi_gpio_0 • S_AXI	The Master field is set to / microblaze_0 (Periph). The Clock Connection (for unconnected clks) field is set to its default value of Auto.			
axi_uartlite_0 • S_AXI	The Master field is set to its default value of /microblaze_0 (Periph). The Clock Connection (for unconnected clks) field is set to its default value of Auto.	Keep these default settings.		
mdm_1 • TRIG_IN_0	This will be connected to a new System ILA core's TRIG_OUT pin.	Leave the ILA Connection settings to its default value of Auto.		
mdm_1 • TRIG_OUT_0	This will be connected to the System ILA core'sTRIG_IN pin.	Leave the ILA Connections settings to its default value of Auto.		
mig_7series_0 • S_AXI	The Master field is set to / microblaze_0 (Cached). Leave it to this value so the accesses to the DDR3 memory are cached accesses.	Keep these default settings.		
	The Clock Connection (for unconnected clks) field is set to its default value of Auto.			
mig_7series_0 • sys_rst	The board interface reset will be connected to the reset pin of the Memory IP.	Keep the default setting.		
Rst_mig_7_series_0_100M • ext_reset_in	The reset pin of the Processor Sys Rreset IP will be connected to the board reset pin.	Keep the default setting.		

Table 1: Run Connection Automation Options (cont'd)

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

At this point, your IP integrator diagram area should look like the following figure.





Note: The relative placement of your IP might be slightly different.

Mark Nets for Debugging

- 1. To monitor the AXI transactions taking place between the MicroBlaze and the GPIO, select the interface net connecting M00_AXI interface pin of the microblaze_0_axi_periph instance and the S_AXI interface pin of the axi_gpio_0 instance.
- 2. Right-click and select **Debug** from the context menu.

Note: The Designer Assistance is available as indicated by the Run Connection Automation link in the banner of the block design.

3. Click Run Connection Automation.

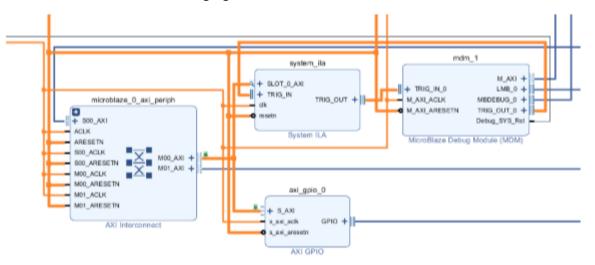


4. In the Run Connection Automation dialog box, go with the default setting as shown in the following figure.

Run Connection Automation Automatically make connections in your design by checking to	+ bases of the interfaces to connect. Select an interface on the left to display its configuration options on the right.
Q ★ All Automation (1 out of 2 selected) V ■ All Automation (1 out of 2 selected) V ♥ interface Connections V ♥ mitcroblace_0_axl_periph_N00_Axi V ■ mitcroblace_0_axl_periph_N00_Axi C ■ Mitcroblace_0_Axl_periph_N00_Axi Mitcroblace_0_Axl_periph_N00_Axi C ■ Mitcroblace_0_Axl_periph_N00_Axi C ■ Mitcroblace_0_Axl_periph_N00_Axi C ■ Mitcroblace_0_Axl_periph_N00_Axi Mitcroblace_0_Axl_periph_N00_Axi C ■ Mitcroblace_0_Axl_periph_N00_Axi C ■ Mitcroblace_0_Axl_periph_N00_Axi C ■ Mitcroblace_0_Axl_periph_N00_Axi C ■ Mitcroblace_0_Axl_periph_N00_Axi C ■ Mitcroblace_0_Axl_periph_N00_Axi C ■ Mitcroblace_0Axl_periph_N00_Axi C ■ Mitcroblace_0Axl_periph	Description Connect selected interface-connection and/or net to System ILA Core for Debugging in Hardware Manager. System ILA (Integrated Logic Analyzer) IP core is a logic analyzer which allows you to perform in-system debugging of designs and shows interface level events in the Hardware Manager in an intuitive way. Options
	AXI Read Address: Dafa and Trigger v AXI Read Dafa: Dafa and Trigger v AXI Write Address: Dafa and Trigger v AXI Write Dafa: Dafa and Trigger v AXI Write Dafa: Dafa and Trigger v AXI Write Dafa: Dafa and Trigger v AXI Write Response: Dafa and Trigger v Source Clock: /mig_Tseries_0/uL_addn_clk_0 System ILA: Auto v AXI-MM Protocol Checker:
•	OK Cancel

5. Click OK.

The cross-trigger pins of the MDM and the AXI Interface net connecting the microblaze_0_axi_periph Interconnect and axi_gpio_0 are connected to the System ILA IP as shown in the following figure.



6. Click the Regenerate Layout button C in the IP integrator toolbar to generate an optimum layout for the block design. The block diagram looks like the following figure.



Dagram	7 _ D 7 X
	0
NUM da Second and a second and	

Step 3: Memory-Mapping the Peripherals in IP Integrator

- 1. Click the Address Editor window.
- 2. In the Address Editor, do the following:
 - a. Expand the microblaze_0 instance by clicking on the Expand All icon 🗢 in the toolbar to the top of the Address Editor window.



b. Change the range of microblaze_0/mig_7_series_0 IP in both the Data and the Instruction section to **512 MB**, and mdm_1/mig_7_series_0 also to 512 MB, as shown in the following figure.

Q, , X ♦ 111						
Cell	Slave Interface	Slave Segment	Offset Address	Range		High Address
microblaze_0						
Data (32 address bits : 4G)						
microblaze_0_local_memory/dlmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	64K	•	0x0000_FFF
== axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	8K	•	0xC000_1FF
🚥 axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	Ŧ	0x4000_FFF
🚥 axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	*	0x4060_FFF
🚥 mig_7series_0	S_AXI	memaddr	0000_0008x0	512M	Ŧ	0x9FFF_FFF
✓ I Instruction (32 address bits : 4G)						
microblaze_0_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000x0	64K	Ŧ	0x0000_FFF
🚥 axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	8K	Ŧ	0xC000_1FF
🚥 mig_7series_0	S_AXI	memaddr	0000_0008x0	1G	<	0xBFFF_FFF
✓ ₱ mdm_1				32M	^	
Data (32 address bits : 4G)				64M		
microblaze_0_local_memory/dlmb_bram_if_cntlr	SLMB1	Mem	0x0000_0000x0	128M		0x0000_FFF
🚥 axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	256M		0xC000_1FF
∞ mig_7series_0	S_AXI	memaddr	0x8000_0000	512M	3	0xBFFF_FFF

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

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

Validating the design will automatically re-configure the MicroBlaze processor's cacheable address range.

Step 4: Validate Block Design

To run design rule checks on the design:

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

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

2. Click OK.



3. Save your design by pressing Ctrl+S, or select File \rightarrow Save Block Design.

Step 5: Generate Output Products

1. In the Sources window, select the block design, then right-click it and select **Generate Output Products**. Alternatively, you can click **Generate Block Design in the Flow Navigator**.

The Generate Output Products dialog box opens.

2. Click Generate.

A Generate Output Products ×
The following output products will be generated.
Preview
Q 素 ≑
✓ ▲ ■ mb_subsystem.bd (OOC per IP)
Synthesis
Implementation
Simulation
Hw_Handoff
Synthesis Options
<u>G</u> lobal <u>O</u> ut of context per IP
 Out of context per <u>B</u>lock Design
Run Settings
Number of jobs: 4 🗸
Apply Generate Cancel

The Generate Output Products dialog box informs you that Out-of-context module runs were launched.

3. Click OK.



4. Wait a few minutes for all the Out-of-Context module runs to finish as shown in the Design Runs windows.

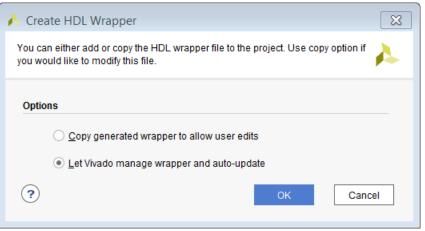
Design Runs									? _ !	7
Q ≚ ⊕ I ≪ ▶ ≫ + %										
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT
✓ ▷ synth_1 (active)	constrs_1	Not started								
▷ impl_1	constrs_1	Not started								
Out-of-Context Module Runs										
✓ ✓ mb_subsystem		Submodule Runs Complete								
mb_subsystem_microblaze_0_0_synth_1	mb_subsystem_microblaze_0_0	synth_design Complete!								36.
mb_subsystem_mig_7series_0_0_synth_1	mb_subsystem_mig_7series_0_0	synth_design Complete!								13
mb_subsystem_axi_uartiite_0_0_synth_1	mb_subsystem_axi_uartiite_0_0	synth_design Complete!								11
mb_subsystem_axi_gpio_0_0_synth_1	mb_subsystem_axi_gpio_0_0	synth_design Complete!								3
mb_subsystem_axi_bram_ctrl_0_0_synth_1	mb_subsystem_axi_bram_ctrl_0_0	synth_design Complete!								20
mb_subsystem_mdm_1_0_synth_1	mb_subsystem_mdm_1_0	synth_design Complete!								4
✓ mb_subsystem_rst_mig_7series_0_100M_0_synth_1	mb_subsystem_rst_mig_7series_0_100M_0	synth_design Complete!								
✓ ✓ mb_subsystem_axi_smc_0		Submodule Runs Complete								
mb_subsystem_axi_smc_0_synth_1	mb_subsystem_axi_smc_0	synth_design Complete!								11
mb_subsystem_rst_mig_7series_0_200M_0_synth_1	mb_subsystem_rst_mig_7series_0_200M_0	synth_design Complete!								
mb_subsystem_axi_bram_ctrl_0_bram_0_synth_1	mb_subsystem_axi_bram_ctrl_0_bram_0	synth_design Complete!								
v v mb_subsystem_system_ila_0		Submodule Runs Complete								
mb_subsystem_system_ila_0_synth_1	mb_subsystem_system_ila_0	synth_design Complete!								14
mb_subsystem_xbar_0_synth_1	mb_subsystem_xbar_0	synth_design Complete!								15
mb_subsystem_dimb_v10_0_synth_1	mb_subsystem_dimb_v10_0	synth_design Complete!								

Step 6: Create a Top-Level Wrapper

 Under Design Sources, right-click the block design mb_subsystem and click Create HDL Wrapper.

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

2. Click OK.





Step 7: Take the Design through Implementation

1. In the Flow Navigator, click Generate Bitstream.

The No implementation Results Available dialog box opens.

2. Click Yes.

The Launch Runs dialog box opens.

3. Make the appropriate choices and click **OK**.

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

- 4. Keep the default selection of Open Implemented Design and click OK.
- 5. Verify that all timing constraints have been met by looking at the Timing Design Timing Summary window, as shown in the following figure.

Timing						? _ 🗆 ?
Q ቿ ♦ ●	Design Timing Summary					
General Information Timer Settings	Setup		Hold		Pulse Width	
Design Timing Summary Clock Summary (52)	Worst Negative Slack (WNS): Total Negative Slack (TNS):	0.177 ns 0.000 ns	Worst Hold Slack (WHS): Total Hold Slack (THS):	0.051 ns 0.000 ns	Worst Pulse Width Slack (WPWS): Total Pulse Width Negative Slack (TPWS):	0.062 ns 0.000 ns
Check Timing (20) Intra-Clock Paths Intra-Clock Paths	Number of Failing Endpoints: Total Number of Endpoints:	0 118764	Number of Failing Endpoints: Total Number of Endpoints:	0 118748	Number of Failing Endpoints: Total Number of Endpoints:	0 38927
Coter Path Groups User Ignored Paths	All user specified timing constrain	nts are met.				
Unconstrained Paths						
Timing Summary - impl_1 (saved)						

Step 8: Export the Design to the Vitis software platform

IMPORTANT! For the Digilent driver to install, you must power on and connect the board to the host PC before launching the Vitis software platform.

Next, open the design and export to the Vitis software platform.

1. From the Vivado File menu, select File \rightarrow Export \rightarrow Export Hardware.

The Export Hardware dialog box opens.

2. Select the Include bitstream option using the radio button in the Output view and click Next.



🚴 Export Hardware Platform	×
Output Set the platform properties to inform downstream tools of the intended use of the target platform's hardware design.	4
Output Set the platform properties to inform downstream tools of the intended use of the target platform's hardware	
This platform includes the complete hardware implementation and bitstream, in addition to the hardware	
< <u>B</u> ack <u>N</u> ext > <u>F</u> inish Cance	el

3. Leave the XSA file name field at its default value and click **Next**. (The following figure shows Windows-specific settings.)

À Export Hardwar	re Platform	×
Files Enter the name o	f your hardware platform file, and the directory where the XSA file will be stored.	4
XSA file name:	mb_subsystem_wrapper	8
Export to:	c:/ug940/lab3	
	The XSA will be written to: c:\ug940\lab3\mb_subsystem_wrapper.xsa	
	< <u>B</u> ack <u>N</u> ext > <u>F</u> inish	Cancel



4. Click Finish. This will export the hardware XSA File in the lab1 project directory.

🝌 Export Hardware Platform		Х
VIVADO.	Exporting Hardware Platform	
HLx Editions	A new fixed hardware platform named 'mb_subsystem_wrapper' will be written as 'c:\ug940\lab3\mb_subsystem_wrapper.xsa'.	
	The platform will include a post-implementation model, including a bitstream description, describing the hardware for downstream software tasks.	
	tools.	
E XILINX.	To export the platform, click Finish.	
	< <u>B</u> ack <u>N</u> ext > <u>Finish</u> Cancel	

5. To launch the Vitis software platform, select **Tools** \rightarrow **Launch Vitis IDE**.

The Eclipse Launcher dialog box opens.

- 6. Specify the desired Workspace location such as C:\UG940\lab3_vitis (Windows-specific).
- 7. Click Launch.

Step 9: Create a "Peripheral Test" Application

The Vitis software platform launches in a separate window.



-	lab1_vitis - Vitis IDE Edit Developer Xilinx Project Window Help				- 0	×
0	Welcome 23					- #
	VITIS.					
		VI ID	TIS E			
		PROJEC	т	PLATFORM	RESOURCES	
		Create Ap	plication Project	Add Custom Platform	Vitis Documentation	
		Create Pla	itform Project		QuickTake Videos	
		Create Lit	arary Project			
		Import Pro	oject			
		<			1	```

- 1. Close the Welcome screen if it appears.





Vew Application Project	_	
Create a New Application Project		•••
This wizard will guide you through the 4 steps of creating new application projects. 1. Choose a platform or create a platform project from Vivado exported XSA 2. Put application project in a system project , associate it with a processor 3. Prepare the application runtime – domain 4. Choose a template for application to quick start development		
PlatformSystemProjectProject		
Processor Domain App		
XSA		
 A platform provides hardware information and software environment settings. A system project contains one or more applications that run at the same time. A domain provides runtime for applications, such as operating system or BSP. A workspace can contain unlimited platforms and unlimited system projects. 		
Skip welcome page next time. (Can be reached with Back button)		
? < Back Next > Finish	n	Cancel

- 3. Select the **Skip welcome page next time** check box if you do not want the welcome to appear when the Vitis software platform is launched again.
- 4. Click Next.
- 5. In the Platform page, select the **Create a new platform from hardware (XSA)** tab.



New Applic	ation Project					- C		×
Please select	t a platform to create the p	roject						•
🔚 Select a	platform from repository	Create a new platform from hardware (XSA)						^
Hardware	e Specification							
XSA File:	vck190	se a pre-built board description				Browse		
Platform r ☑ Generat	name:							
								¥
?			< Back	Next >	Finish	(Cancel	

6. Click **Browse** to open The Create Platform from XSA window. Navigate to the directory where the XSA file was created in Vivado and click **Open**.

→ × ↑ 📙 > `	This PC > Windows (C:) > ug940 > lab3	√ Ū	Search lab3	
rganize 🔻 🛛 New fo	lder			III 🔻 🔟
📙 ug940 🖌	Name	Date modified	Туре	Size
.Xil	lab3.cache	7/7/2020 12:20 AM	File folder	
lab1	lab3.hbs	7/7/2020 12:17 AM	File folder	
lab1_vitis	lab3.hw	7/7/2020 12:17 AM	File folder	
lab2	lab3.ip_user_files	7/7/2020 12:21 AM	File folder	
lab2_vitis	lab3.runs	7/7/2020 12:23 AM	File folder	
lab3	lab3.sim	7/7/2020 12:17 AM	File folder	
lab3_vitis	lab3.srcs	7/7/2020 12:18 AM	File folder	
Users	mb_subsystem_wrapper.xsa	7/7/2020 7:41 AM	XSA File	1,467 KB
File	name: mb_subsystem_wrapper.xsa		 *.xsa;*.dsa; 	

7. Ensure the Generate boot Components option is selected in the Platform page.



┥ New Applic	ation Project					_		×
Platform Note: A platfor	m project will be generate	d automatically in workspace for th	e selected XSA. It	can be customized	later.			
🔄 Select a p	olatform from repository	Create a new platform from I	hardware (XSA)					^
Hardware	Specification							
	C:\ug940\lab3\mb_subsys	stem_wrapper.xsa						
XSA File:	vck190 zc702 zc706 zcu102 zed C:\ug940\lab3\mb_subsys	tem_wrapper.xsa				Brows	e	
	aame: mb_subsystem_wra e boot components	pper						
								~
?			< Back	Next >	Finish		Cance	4

- 8. Click Next.
- 9. In the **Domain** page leave all the fields at their default values and click **Next**.
- 10. In the Application project name field, type the name desired, such as peri_test. Leave all other fields to their default values, and click **Next**.





New Application Project			_		×
pplication Project Details Specify the application project nar	ne and its system project properties	i		•	••
Application project name: peri_t System Project	est				
Create a new system project f	or the application or select an existi	ng one from the workpsace 🛛 👔			
Select a system project	System project details System project name: pe Target processor	rri_test_system			
	Select target processor for	the Application project. Associated applications			
	microblaze_0	peri_test			
	Show all processors in the	hardware specification 🗹 🛛 👔			
?	[< Back Next > Fini	sh	Cance	al

11. In the Domain page leave all the fields at their default values and click **Next**.





✓ New Application Project	– 🗆 X
Domain Select a domain for your project or create a new domain	•••
Select the domain that the application would link to or create a new do Note: New domain created by this wizard will have all the requirement Select a domain Create new	ts of the application template selected in the next step
Name:	domain_microblaze_0
Display Name:	domain_microblaze_0
Operating Syst	tem: standalone v
Processor:	microblaze_0
Architecture:	32-bit V
?	< Back Next > Finish Cancel

A new peri_test application is created. To build the application click the hammer icon sin the toolbar.

12. 16. In the Templates page, select **Peripheral Tests**.



Vew Application Project	_	o x
Templates Select a template to create your project.		
Available Templates:		
Find:	Peripheral Tests	
 ✓ SW development templates Dhrystone Empty Application Empty Application (C++) Hello World IwIP Echo Server IwIP TCP Perf Client IwIP TCP Perf Server IwIP UDP Perf Client IwIP UDP Perf Server mba_fs_boot Memory Tests Peripheral Tests SREC Bootloader SREC SPI Bootloader 	Simple test routines for all peripherals in the hardware.	
3	(Deale Names Circle	Concel
	< Back Next > Finish	Cancel

- 13. Click Finish.
- 14. A new peri_test application is created. To build the application click the hammer icon sin the toolbar.
- 15. Wait for the application to finish compiling.
- 16. Right-click the peri_test application in the Project Explorer, and select Generate Linker Script.

The Generate Linker Script dialog box opens.

17. Select the **Basic** tab, and change the Assigned Memory for Heap and Stack to **mig_7series_0**.



✔ Generate a linker script								×
Generate linker script Control your application's memory ma	ap.							0
Output Settings Project: peri_test Output Script: C:\ug940\lab3_vitis\peri_test\src\lscr Modify project build settings as follow			Browse	Place D	Advanced ode Sections in: ata Sections in: eap and Stack in:	mig_7series_0_me mig_7series_0_me mig_7series_0_me	maddr	~ ~ ~
Set generated script on all project bui	ild configurations		~	Heap Si	ze:	1 KB	0×00000400	
Hardware Memory Map				Stack Si	ze:	1 KB	0×00000400	
Memory axi_bram_ctrl_0_Mem0 microblaze_0_local_memory_ilmb mig_7series_0_memaddr Fixed Section Assignments	Base Address 0xC000000 0x0000000 0x8000000	Size 8 KB 64 KB 512						
?							Generate	Cancel

Setting these values to **mig_7series_0** ensures that the compiled code executes from the Memory IP.

- 18. Click Generate.
- 19. Click **Yes** to overwrite it in the **Linker Already Exists!** dialog box.
- 20. Click the hammer icon 🍝 in the toolbar again to rebuild the application with the modified linker script.

Step 10: Execute the Software Application on a KC705 Board

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

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

The Debug Configurations dialog box opens, as shown in the following figure.

2. Right-click Single Application Debug, and select New Configuration.



🚽 Debug Configurations							×
Create, manage, and run con Debug a program using Applicat	-					Ŕ	S.
Image: Single Application Debug Image: Single Application Debug <th></th> <th>Ner Exp Du Del Lin Un</th> <th>Press the "New Prototyne Configuration W Prototype ort plicate ete k Prototype ink Prototype et with Prototype</th> <th>from this dialog: guration' button to create a configuration of the select button to create a launch configuration prototype of to export the selected configurations. on to copy the selected configuration. to remove the selected configuration. configure filtering options. anfiguration by selecting it. on(s) and then select 'Link Prototype' menu item to u</th> <th>to link</th> <th>elected t</th> <th>type,</th>		Ner Exp Du Del Lin Un	Press the "New Prototyne Configuration W Prototype ort plicate ete k Prototype ink Prototype et with Prototype	from this dialog: guration' button to create a configuration of the select button to create a launch configuration prototype of to export the selected configurations. on to copy the selected configuration. to remove the selected configuration. configure filtering options. anfiguration by selecting it. on(s) and then select 'Link Prototype' menu item to u	to link	elected t	type,
Filter matched 3 of 3 items				ion(s) and then selectValues' menu item to reset with ive settings from the <u>'Perspectives'</u> preference page. Debug) proto	otype va Close	

3. The Create, manage and run configurations page opens. Click **Debug**.

🚽 Debug Configurations				— — ×
Create, manage, and run configuration Debug a program using Application Debug				Ť.
📑 🖻 🕼 😫 🖻 🕸 🔹	Name: Debugge	tr_peri_test-Default		
type filter text	🕈 Main 🗖	Application () Target Setup 00- Arguments	Environment	
 Single Application Debug Debugger_peri_test-Default Single Application Debug (GDB) SPM Analysis 	Debug Type: 3 Connection: 1 Project: Configuration: Performance	peri_test Debug	99 ₄	Browse
Filter matched 4 of 4 items			Revert	Apply
0			Debug	Close

The Debug perspective window opens.

4. Set the terminal by selecting the Vitis Terminal tab and clicking the 🖶 button.



5. Use the settings shown in the following figure for the KC705 board and click OK.

🚽 Connect t	Connect to serial port				
Basic Setti	ngs				
Port:	CO	M5		~	
Baud Rate	960	0		\sim	
▼ Advance	e Set	tings			
Data Bits:		8		\sim	
Stop Bits:		1		\sim	
Parity:		None		\sim	
Flow Con	trol:	None		\sim	
Timeout ((sec):				
Oł	(Cancel		

6. Verify the terminal connection by checking the status at the top of the Vitis software platform Terminal tab, as shown in the following figure.

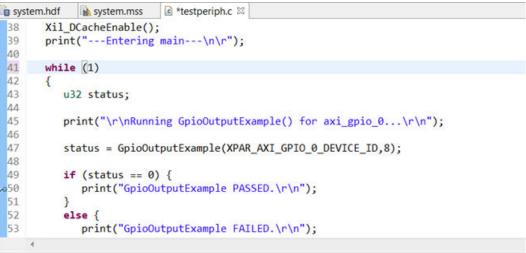


- 7. If the testperiph.c file is not already open, select ../src/testperiph.c, and double-click to open the source file.
- 8. Modify the source file by inserting a while statement at approximately line 41.
 - a. Click the blue bar on the left side of the testperiph.c window as shown in the figure, and select **Show Line Numbers**.
 - b. In line 41, add while(1) above in front of the curly brace as shown in the following figure:



i sy	stem.hdf	system.mss	🖻 *testperiph.c 🛛	
38 39 40		_DCacheEnable(); nt("Entering		
40 41	whil	Le (1)		
42	{			
43		32 status;		
44				
45	1	print("\r\nRunni	<pre>ing GpioOutputExample() for axi gpio 0\r\n");</pre>	
46				
47	-	status = GpioOut	<pre>tputExample(XPAR_AXI_GPIO 0 DEVICE_ID,8);</pre>	
48				
49	1 13	if (status == 0)) {	
50		print("GpioOu	utputExample PASSED.\r\n");	
51		}		
52		else {		
53		print("GpioOu	utputExample FAILED.\r\n");	
- 1	4			

- 9. Add a breakpoint in the code so that the processor stops code execution when the breakpoint is encountered. To do so, scroll down to line 50 and double-click on the left pane, which adds a breakpoint on that line of code, as shown in the following figure.
- 10. Press Ctrl + S to save the file. Alternatively, you can select File \rightarrow Save.



11. Click the hammer icon $\sqrt[6]{}$ to rebuild the file with the modified code.

Now you are ready to execute the code from the Vitis software platform.

Step 11: Connect to Vivado Logic Analyzer

Connect to the KC705 board using the Vivado Logic Analyzer.

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



Hardware Manager - unconnected				
In the second				
Hardware 💐 Auto Connect				
	Recent Targets			
Name Chabus	Closed Targets			
Name Status	🖻 Open New T衣get			

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

The Open New Hardware Target dialog box opens, shown in the following figure.

🔥 Open New Hardware	e Target 🛛 🕅
HLx Editions	Open Hardware Target This wizard will guide you through connecting to a hardware target. To connect to a remote hardware target, provide the host name and IP port of the remote machine on which the instance of a Vivado Hardware Server is running.
?	< <u>B</u> ack <u>N</u> ext > Cancel

- 3. Click Next.
- 4. On the Hardware Server Settings page, ensure that the Connect to field is set to **Local server** (target is on local machine) as shown in the following figure, and click Next.

🍌 Open New	Hardware Target	×
Select local or r	erver Settings emote hardware server, then configure the host name and port settings. Use Local server if the ed to the local machine; otherwise, use Remote server.	4
<u>C</u> onnect to: Click Next to	Local server (target is on local machine)	
?	< <u>B</u> ack <u>N</u> ext > <u>F</u> inish Ca	ancel

- 5. On the Select Hardware Target page, click Next.
- 6. Ensure that all the settings are correct on the Open Hardware Target Summary dialog box, as shown in the following figure, and click **Finish**.



🕕 Open New Hardware	e Target 🔀
	Open Hardware Target Summary
HLx Editions	 Hardware Server Settings: Server: localhost:3121
	 Target Settings: Target: xilinx_tcf/Digilent/210203357707A Frequency: 15000000
E XILINX ALL PROGRAMMABLE.	To connect to the hardware described above, click Finish
?	< <u>B</u> ack <u>N</u> ext > <u>Finish</u> Cancel

Step 12: Set the MicroBlaze to Logic Cross Trigger

When the Vivado Hardware Session successfully connects to the ZC702 board, you see the information shown in the following figure:

Hardware	? _ 🗆 🖒 X
$Q \mid \underbrace{\star} \mid \diamondsuit \mid \varnothing \mid \models \mid \gg \mid \blacksquare \mid$	•
Name	Status
🗠 📱 localhost (1)	Connected
✓ ✓ ✓ ✓ xilinx_tcf/Digilent/2102033577	Open
xc7k325t_0 (2)	Programmed
🖉 XADC (System Monitor)	
🦉 hw_ila_1 (mb_subsystem	Oldle

Figure 1: Vivado Hardware Window

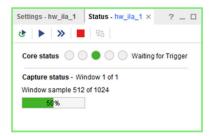
- 1. Select the settings hw_ila_1 tab and set the Trigger Mode Settings as follows:
 - a. Set Trigger mode to TRIG_IN_ONLY.
 - b. Set TRIG_OUT mode to **TRIG_IN_ONLY**.
 - c. Under Capture Mode Settings, ensure that Trigger position in window is set to **512**.



_iia_1			? 🗆 🗆
Waveform - hw_iia_1			? _ □
Q 🕂 — 🥴 🕨 👅 📴 🔍 Q	X •	I I I 12 27 + I I 0 of -	+
ILA Status: Idle			
Name	Value	0 1 2 3 4 5	6 7
slot_0: microblaze_0_axi_periph_M01_AXI : Interface % slot_0: microblaze_0_axi_periph_M01_AXI : AR Channel % slot_0: microblaze_0_axi_periph_M01_AXI : AK Channel % slot_0: microblaze_0_axi_periph_M01_AXI : AW Channel			
	$\langle \rangle$	<	
Settings - hw_ila_1 × Status - hw_ila_1 ?	_ 🗆	Trigger Setup - hw_ila_1 × Capture Setup - hw_ila_1	? _
Trigger Mode Settings	î	$\bigcirc + - \diamondsuit $	
Trigger mode: TRIG_IN_ONLY ~ TRIG_OUT mode: TRIG_IN_ONLY ~			
Capture Mode Settings	- 11		
Capture mode: ALWAYS V Number of windows: 1 [1 - 1024] Window data depth: 1024 [1 - 1024]		Press the 🕂 button to add probes.	
Trigger position in window: 512 [0 - 1023]	- 11		
General Settings Refresh rate: 500 ms	÷		

2. Arm the ILA core by clicking the Run Trigger 🕨 button.

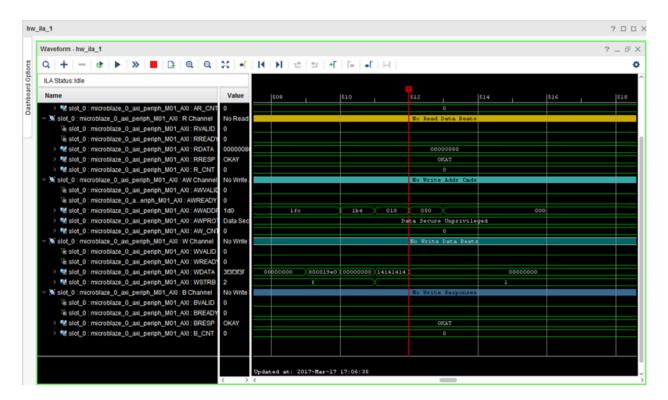
This arms the ILA. You should see the status "Waiting for Trigger" as shown in the following figure.



3. In the Vitis software platform Debug window, click **MicroBlaze #0** and then click the **Resume** button.

The code will execute until the breakpoint set on line 50 in testperiph.c file is reached. As the breakpoint is reached, this triggers the ILA, as shown in the following figure.





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

Step 13: Set the Logic to Processor Cross-Trigger

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

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

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

2. In the Debug window, right-click the **MicroBlaze #0 target** and select **Resume**.

The code runs continuously because it has an infinite loop.

You can see the code executing in the Terminal Window in the Vitis software platform.



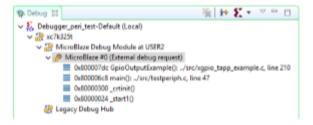
- 3. In Vivado, select the **Settings hw_ila_1** tab. Change the Trigger Mode to **BASIC_OR_TRIG_IN** and the TRIG_OUT mode to **TRIGGER_OR_TRIG_IN**.
- 4. Click on the (+) sign in the Trigger Setup window to add the slot_0:microblaze_0_axi_periph_M01:AWVALID signal from the Add Probes window.
- 5. In the Basic Trigger Setup window, for slot_0:microblaze_0_axi_periph_M00:AWVALID signal, ensure that the Radix field is set to [B] (Binary) and set the Value field to 1.

This essentially sets up the ILA to trigger when the awvalid transitions to a value of 1.

6. Click the Run Trigger button to "arm" the ILA in the Status - hw_ila_1 window.

The ILA immediately triggers as the application software is continuously performing a write to the GPIO thereby toggling the $net_slot_0_axi_awvalid$ signal, which causes the ILA to trigger. The ILA in turn, toggles the TRIG_OUT signal, which signals the processor to stop code execution.

This is seen in Vitis in the highlighted area of the debug window.



Conclusion

In this tutorial, you:

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



Lab Files

The Tcl script lab3.tcl is included with the design files to perform all the tasks in Vivado. The Vitis software platform operations must be done in the Vitis GUI. You might need to modify the Tcl script to match the project path and project name on your machine.





Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx[®] Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado[®] IDE, select **Help → Documentation and Tutorials**.
- On Windows, select Start → All Programs → Xilinx Design Tools → DocNav.
- At the Linux command prompt, enter docnav.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this guide:

Send Feedback



- 1. Vivado Design Suite User Guide: Release Notes, Installation, and Licensing (UG973)
- 2. Vivado Design Suite User Guide: Using the Vivado IDE (UG893)
- 3. Vivado Design Suite User Guide: Designing with IP (UG896)
- 4. Vivado Design Suite Tcl Command Reference Guide (UG835)
- 5. Vivado Design Suite User Guide: Design Flows Overview (UG892)
- 6. Vivado Design Suite User Guide: Logic Simulation (UG900)
- 7. Vivado Design Suite User Guide: Using Tcl Scripting (UG894)
- 8. Vivado Design Suite User Guide: Implementation (UG904)
- 9. Vivado Design Suite User Guide: Using Tcl Scripting (UG894)
- 10. Zynq-7000 SoC Technical Reference Manual (UG585)
- 11. Vitis Unified Software Platform Documentation (UG1416)

Please Read: Important Legal Notices

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



AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright

© Copyright 2013-2020 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

