

# Vivado Design Suite Tutorial

## **Programming and Debugging**

UG936 (v2022.2) November 16, 2022

Xilinx is creating an environment where employees, customers, and partners feel welcome and included. To that end, we're removing noninclusive language from our products and related collateral. We've launched an internal initiative to remove language that could exclude people or reinforce historical biases, including terms embedded in our software and IPs. You may still find examples of non-inclusive language in our older products as we work to make these changes and align with evolving industry standards. Follow this <u>link</u> for more information.



# AMD7 XILINX

## Table of Contents

Debugging in Vivado Tutorial	5
Navigating Content by Design Process	5
Objectives	e
Getting Started	7
Lab 1: Using the Netlist Insertion Method to Debug a Des	ign 12
Step 1: Creating a Project with the Vivado New Project Wizard	-
Step 2: Synthesizing the Design	
Step 3: Probing and Adding Debug IP	
Step 4: Implementing and Generating Bitstream	22
Lab 2: Using the HDL Instantiation Method to Debug a De	esign23
Step 1: Creating a Project with the Vivado New Project Wizard	
Step 2: Synthesize Implement and Generate Bitstream	
Lab 3: Using a VIO Core to Debug a Design in Vivado Desig	gn
Suite	
Step 1: Creating a Project with the Vivado New Project Wizard	
Step 2: Synthesize, Implement, and Generate the Bitstream	32
Lab 4: Using the Synplify Pro Synthesis Tool and Vivado D	esign
Suite to Debug a Design	
Step 1: Create a Synplify Pro Project	
Step 2: Synthesize the Synplify Project	
Step 3: Create DCPs for the Black Box Created in Synplify Pro	
Step 4: Create a Post Synthesis Project in Vivado IDE	
Step 5: Add More Debug Nets to the Project	
Step 6: Implementing the Design and Generating the Bitstream	44
Lab 5: Using the Vivado Logic Analyzer to Debug Hardwa	e45
Step 1: Verifying Operation of the Sine Wave Generator	45
Step 2: Debugging the Sine Wave Sequencer State Machine (Optional)	



#### AMD XILINX

Lab 6: Using the ECO Flow to Replace Debug Probes Post Implementation74
Lab 7: Debugging Designs Using the Incremental Compile Flow 87
Procedure
Step 1: Opening the Example Design and Adding a Debug Core
Step 3: Create New Runs
Step 4: Making Incremental Debug Changes
Step 5: Running Incremental Compile
Conclusion
Lab 8: Using the Vivado Serial Analyzer to Debug Serial Links100
Design Description
Step 1: Creating, Customizing, and Generating an IBERT Design
Step 2: Adding an IBERT Core to the Vivado Project102
Step 3: Synthesize, Implement and Generate Bitstream for the IBERT Design
Step 4: Interact with the IBERT Core Using Serial I/O Analyzer
Lab 9: Using the Vivado ILA Core to Debug JTAG-AXI
Transactions126
Design Description126
Step 1: Creating a New Vivado Project and Generating the IP Integrator Design with JTAG-to-AXI and System ILA
Step 2: Program the KC705 Board and Interact with the JTAG to AXI Master Core 140 Step 3: Using ILA Advanced Trigger Feature to Trigger on an AXI Read Transaction 146
Lab 10: Using the Vivado Serial Analyzer to Debug PS-GTR Serial Links
Links
Step 1: Generating a Zyng UltraScale+ MPSoC PS Xilinx Support Archive
Step 1: Generating a 2ynd Oltrascale+ MPSOC PS Allinx Support Archive
Boot Loader
Step 3: ZCU102 Board Settings163
Using FSBL with Serial I/O Analyzer to Bring Up IBERT PS-GTR
Troubleshooting
Appendix A: Additional Resources and Legal Notices





Xilinx Resources	173
Documentation Navigator and Design Hubs	173
Revision History	173
Please Read: Important Legal Notices	174



## Debugging in Vivado Tutorial

This document contains a set of tutorials designed to help you debug complex FPGA designs. The first four labs explain different kinds of debug flows that you can chose to use during the course of debug. These labs introduce the Vivado<sup>®</sup> Design Suite debug methodology recommended to debug your FPGA designs. The labs describe the steps involved in taking a small RTL design and the multiple ways of inserting the Integrated Logic Analyzer (ILA) core to help debug the design. The fifth lab is for debugging high-speed serial I/O links in the Vivado tool. The sixth lab is for debugging JTAG-AXI transactions in the Vivado tool. The first four labs converge at the same point when connected to a target hardware board.

Example RTL designs are used to illustrate overall integration flows between the Vivado logic analyzer, ILA, and the Vivado Integrated Design Environment (IDE). To be successful using this tutorial, you should have some basic knowledge of the Vivado tool flow.

**TRAINING:** Xilinx provides training courses that can help you learn more about the concepts presented in this document. Use these links to explore related courses:

- Designing FPGAs Using the Vivado Design Suite 1
- Designing FPGAs Using the Vivado Design Suite 2
- Designing FPGAs Using the Vivado Design Suite 3
- Designing FPGAs Using the Vivado Design Suite 4
- Vivado Design Suite User Guide: Programming and Debugging (UG908)

## **Navigating Content by Design Process**

Xilinx<sup>®</sup> documentation is organized around a set of standard design processes to help you find relevant content for your current development task. All Versal<sup>®</sup> ACAP design process Design Hubs and the Design Flow Assistant materials can be found on the Xilinx.com website. This document covers the following design processes:

- Hardware, IP, and Platform Development: Creating the PL IP blocks for the hardware platform, creating PL kernels, functional simulation, and evaluating the Vivado<sup>®</sup> timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Lab 1: Using the Netlist Insertion Method to Debug a Design
  - Lab 2: Using the HDL Instantiation Method to Debug a Design
  - Lab 3: Using a VIO Core to Debug a Design in Vivado Design Suite



- Lab 4: Using the Synplify Pro Synthesis Tool and Vivado Design Suite to Debug a Design
- Lab 5: Using the Vivado Logic Analyzer to Debug Hardware
- Lab 6: Using the ECO Flow to Replace Debug Probes Post Implementation
- Lab 7: Debugging Designs Using the Incremental Compile Flow
- Lab 9: Using the Vivado ILA Core to Debug JTAG-AXI Transactions
- **Board System Design:** Designing a PCB through schematics and board layout. Also involves power, thermal, and signal integrity considerations. Topics in this document that apply to this design process include:
  - Lab 6: Using the ECO Flow to Replace Debug Probes Post Implementation
  - Lab 7: Debugging Designs Using the Incremental Compile Flow
  - Lab 8: Using the Vivado Serial Analyzer to Debug Serial Links
  - Lab 10: Using the Vivado Serial Analyzer to Debug PS-GTR Serial Links

## Objectives

These tutorials:

- Show you how to take advantage of integrated Vivado<sup>®</sup> logic analyzer features in the Vivado design environment that make the debug process faster and simpler.
- Provide specifics on how to use the Vivado IDE and the Vivado logic analyzer to debug common problems in FPGA logic designs.
- Provide specifics on how to use the Vivado Serial I/O Analyzer to debug high-speed serial links.

After completing this tutorial, you will be able to:

- Validate and debug your design using the Vivado Integrated Design Environment (IDE) and the Integrated Logic Analyzer (ILA) core.
- Understand how to create an RTL project, probe your design, insert an ILA core, and implement the design in the Vivado IDE.
- Generate and customize an IP core netlist in the Vivado IDE.
- Debug the design using Vivado logic analyzer in real-time, and iterate the design using the Vivado IDE and a KC705 Evaluation Kit Base Board that incorporates a Kintex<sup>®</sup>-7 device.
- Analyze high-speed serial links using the Serial I/O Analyzer.



## **Getting Started**

#### **Setup Requirements**

Before you start this tutorial, make sure you have and understand the hardware and software components needed to perform the labs included in this tutorial.

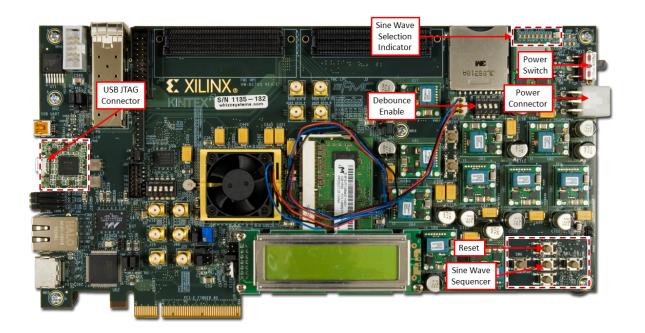
#### Software

Vivado<sup>®</sup> Design Suite 2022.1.

#### Hardware

- Kintex<sup>®</sup>-7 FPGA KC705 Evaluation Kit Base Board
- Digilent Cable
- Two SMA (Sub-miniature version A) cables

#### Figure 1: KC705 Board Showing Key Components



#### **Tutorial Design Components**

Labs 1 through 4 include:



#### AMDA XILINX

- A simple control state machine
- Three sine wave generators using AXI4-Stream interface, native DDS Compiler
- Common push buttons (GPIO\_BUTTON)
- DIP switches (GPIO\_SWITCH)
- LED displays (GPIO\_LED) VIO Core (Lab 3 only)
- **Pushbutton Switches:** Serve as inputs to the de-bounce and control state machine circuits. Pushing a button generates a high-to-low transition pulse. Each generated output pulse is used as an input into the state machine.
- **DIP Switch:** Enables or disables a de-bounce circuit.
- **De-bounce Circuit:** This example, when enabled, provides a clean pulse or transition from high to low. Eliminates a series of spikes or glitches when a button is pressed and released.
- Sine Wave Sequencer State Machine: Captures and decodes input from the two push buttons. Provides sine wave selection and indicator circuits, sequencing among 00, 01, 10, and 11 (zero to three).
- **LED Displays:** GPIO\_LED\_0 and GPIO\_LED\_1 display selection status from the state machine outputs, each of which represents a different sine wave frequency: high, medium, and low.

Lab 5 includes:

- An IBERT core
- A top-level wrapper that instantiates the IBERT core

#### **Board Support and Pinout Information**

Pin Name	Pin Location	Description
CLK_N	AD11	Clock
CLK_P	AD12	Clock
GPIO_BUTTONS[0]	AA12	Reset
GPIO_BUTTONS[1]	AG5	Sine Wave Sequencer
GPIO_SWITCH	Y28	De-bounce Circuit Selector
LEDS_n[0]	AB8	Sine Wave Selection[0]
LEDS_n[1]	AA8	Sine Wave Selection[1]
LEDS_n[2]	AC9	Reserved
LEDS_n[3]	AB9	Reserved

*Table 1:* **Pinout Information for the KC705 Board** 

#### AMD7 XILINX

#### **Design Files**

- 1. In your C: drive, create a folder called /Vivado\_Debug.
- 2. Download the Reference Design Files from the Xilinx website.



**CAUTION!** The tutorial and design files may be updated or modified between software releases. You can download the latest version of the material from the Xilinx<sup>®</sup> website.

- 3. Unzip the tutorial source file to the /Vivado\_Debug folder. There are six labs that use different methodologies for debugging your design. Select the appropriate lab and follow the steps to complete them.
- Lab 1: This lab walks you through the steps of marking nets for debug in HDL as well as the post-synthesis netlist (netlist insertion method). Following are the required files:
  - debounce.vhd
  - fsm.vhd
  - sinegen.vhd
  - sinegen\_demo.vhd
  - sine\_high/sine\_high.xci
  - sine\_low/sine\_low.xci
  - sine\_mid/sine\_mid.xci
  - sinegen\_demo\_kc705.xdc
- Lab 2: This lab goes over the details of marking nets for debug in the source HDL (HDL instantiation method) as well as instantiating an ILA core in the HDL. Following are the required files:
  - debounce.vhd
  - fsm.vhd
  - sinegen.vhd
  - sinegen\_demo\_inst.vhd
  - ila\_0/ila\_0.xci
  - sine\_high/sine\_high.xci
  - sine\_low/sine\_low.xci
  - sine\_mid/sine\_mid.xci
  - sinegen\_demo\_kc705.xdc





- Lab 3: You can test your design even if the hardware is not physically accessible, using a VIO core. This lab walks you through the steps of instantiating and customizing a VIO core that you will hook to the I/Os of the design. Following are the required files:
  - debounce.vhd
  - fsm.vhd
  - sinegen.vhd
  - sinegen\_demo\_inst\_vio.vhd
  - sine\_high/sine\_high.xci
  - sine\_low/sine\_low.xci
  - sine\_mid/sine\_mid.xci
  - ila\_0/ila\_0.xci
  - sinegen\_demo\_kc705.xdc
- Lab 4: Nets can also be marked for debug in a third-party synthesis tool using directives for the synthesis tool. This lab walks you through the steps of marking nets for debug in the Synplify tool and then using Vivado<sup>®</sup> to perform the rest of the debug. Following are the required files:
  - debounce.vhd
  - fsm.vhd
  - sign\_high.dcp
  - sign\_low.dcp
  - sine\_mid.dcp
  - sine\_high.xci
  - sine\_low.xci
  - sine\_mid.xci
  - sinegen.edn
  - sinegen\_synplify.vhd
  - synplify\_1.sdc
  - synplify\_1.fdc
  - sinegen\_demo\_kc705.xdc
- Lab 5: Take designs created from Lab 1, Lab 2, Lab 3, and Lab 4 and load them onto the KC705 board.
- Lab 6: Enhance post implementation debugging by using the ECO flow to replace debug probes.

Send Feedback



- Lab 7: Use the Incremental Compile flow to enable faster debugging flows. Using the results from a previous implementation run, this flow allows you to make debug modifications and rerun implementation.
- Lab 8: Debug high-speed serial I/O links using the Vivado Serial I/O Analyzer. This lab uses the Vivado IP example design.
- Lab 9: Use Vivado ILA core to debug JTAG-to-AXI transactions. This lab uses the Vivado IP example design.
- Lab 10: Use the IBERT UltraScale+ PS-GTR transceiver to evaluate and monitor PS-GTR transceivers in Zynq<sup>®</sup> UltraScale+<sup>™</sup> MPSoC devices. This lab is purely software-based, setting up and testing the processing system (PS) side of the Zynq UltraScale+ MPSoC device with no programmable logic (PL).

#### **Connecting the Boards and Cables**

- 1. Connect the Digilent cable from the Digilent cable connector to a USB port on your computer.
- 2. Connect the two SMA cables (for lab 5 only) as follows:
  - a. Connect one SMA cable from J19 (TXP) to J17 (RXP).
  - b. Connect the other SMA cable from J20 (TXN) to J66 (RXN).

The relative locations of SMA cables on the board are shown in Setup Requirements.





# Using the Netlist Insertion Method to Debug a Design

In this lab, you will mark signals for debug in the source HDL as well as the post synthesis netlist. Then you will create an Integrated Logic Analyzer (ILA) core and take the design through implementation. Finally, you will use the Vivado<sup>®</sup> tool to connect to the KC705 target board and debug your design with the Vivado Integrated Logic Analyzer.

### Step 1: Creating a Project with the Vivado New Project Wizard

To create a project, use the New Project wizard to name the project, to add RTL source files and constraints, and to specify the target device.

- 1. Invoke the Vivado IDE.
- 2. In the Getting Started page, click **Create Project** to start the New Project wizard. Click **Next**.
- 3. In the Project Name page, name the new project proj\_netlist and provide the project location (C:/Vivado\_Debug). Ensure that Create Project Subdirectory is selected and click Next.
- 4. In the Project Type page, specify the type of project to create as RTL Project. Click Next.
- 5. In the Add Sources page:
  - a. Set Target Language to VHDL.
  - b. Click the "+" sign, and then click Add Files.
  - c. In the Add Source Files dialog box, navigate to the /src/lab1 directory.
  - d. Select all VHD source files, and click OK.
  - e. Verify that the files are added, and Copy Sources into project is selected.
- 6. Click Add.
- 7. In the Add Directories dialog box, navigate to the /src/lab1 directory.
- 8. Select sine\_high, sine\_low, and sine\_mid directories and click Select.
- 9. Verify that the directories are added. Click **Next**.

Send Feedback

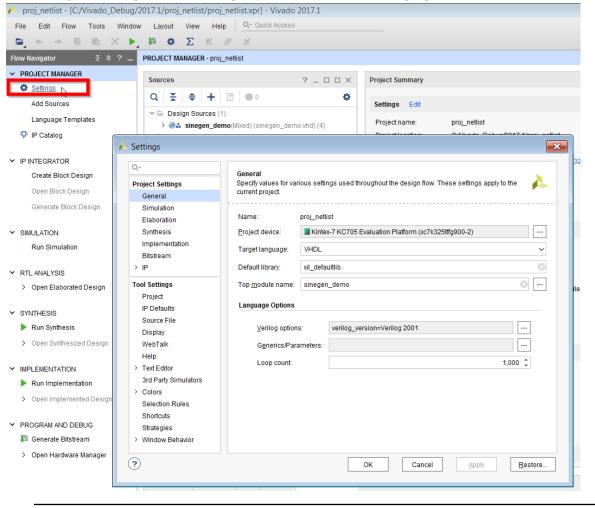


- 10. In the Add Constraints dialog box, click the "+" sign, and then click Add Files.
- 11. Navigate to /src/lab1 directory and select sinegen\_demo\_kc705.xdc. Click Next.
- 12. In the Default Part dialog box, specify the **xc7k325tffg900-2** part for the KC705 platform. You can also select **Boards** and then select **Kintex-7 KC705 Evaluation Platform**. Click **Next**.
- 13. Review the New Project Summary page. Verify that the data appears as expected, per the steps above, and click **Finish**.

Note: It could take a moment for the project to initialize.

## **Step 2: Synthesizing the Design**

1. In the Project Manager, click **Settings** as shown in the following figure.



**IMPORTANT!** As an optional step, in the Settings dialog box, select **Synthesis** from the left and change flatten hierarchy to none. The reason for changing this setting to none is to prevent the synthesis tool from performing any boundary optimizations for this tutorial.



2. In the Vivado<sup>®</sup> Flow Navigator, expand the Synthesis drop-down list, and click **Run Synthesis**. In the Launch Runs dialog box, accept all of the default settings (Launch runs on local host), and click **OK**.

*Note*: When synthesis runs, a progress indicator appears, showing that synthesis is occurring. This could take a few minutes.

3. In the Synthesis Completed dialog box, click **Cancel** as shown in the following figure. You will implement the design later.

Synthesis Completed	×
Synthesis successfully completed.	
• Run Implementation	
Open Synthesized Design	
◯ <u>V</u> iew Reports	
Don't show this dialog again	
OK Cance	I

## **Step 3: Probing and Adding Debug IP**

To add a Vivado ILA core to the design, take advantage of the integrated flows between the Vivado IDE and Vivado logic analyzer.

In this step, you will accomplish the following tasks:

- Add debug nets to the project.
- Run the Set Up Debug wizard.
- Implement and open the design.
- Generate the bitstream.

#### Adding Debug Nets to the Project

Following are some ways to add debug nets using the Vivado IDE:





- Add MARK\_DEBUG attribute to HDL files.
  - VHDL:

```
attribute mark_debug : string;
attribute mark_debug of sine : signal is "true";
attribute mark_debug of sineSel : signal is "true";
```

• Verilog:

```
(* mark_debug = "true" *) wire sine;
(* mark_debug = "true" *) wire sineSel;
```

This method lets you probe signals at the HDL design level. This can prevent optimization that might otherwise occur to that signal. It also lets you pick up the signal tagged for post synthesis, so you can insert these signals into a debug core and observe the values on this signal during FPGA operation. This method gives you the highest probability of preserving HDL signal names after synthesis.

• Right-click and select Mark Debug or Unmark Debug on a synthesized netlist.

This method is flexible because it allows probing the synthesized netlist in the Vivado IDE and allows you to add/remove MARK\_DEBUG attributes at any hierarchy in the design. In addition, this method does not require HDL source modification. However, there can be situations where synthesis might not preserve the signals due to netlist optimization involving absorption or merging of design structures.

• Use a Tcl prompt to set the MARK\_DEBUG attribute on a synthesized netlist.

set\_property mark\_debug true [get\_nets -hier [list {sine[\*]}]]

This applies the MARK\_DEBUG on the current, open netlist.

This method is flexible because you can turn MARK\_DEBUG on and off by modifying the Tcl command. In addition, this method does not require HDL source modification. However, there might be situations where synthesis does not preserve the signals due to netlist optimization involving absorption or merging of design structures.

In the following steps, you learn how to add debug nets to HDL files and the synthesized design using Vivado IDE.

TIP: Before proceeding, make sure that the Flow Navigator on the left panel is enabled.

Use Ctrl-Q to toggle it off and on.

1. In the Flow Navigator under the Synthesis drop-down list, click **Open Synthesized Design** as shown in the following figure.







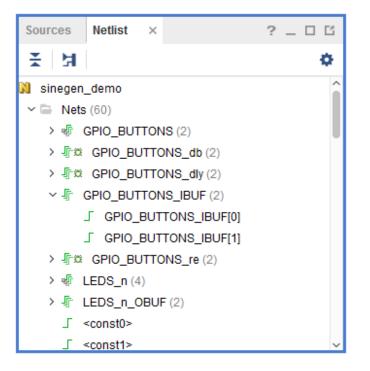
- 2. In the Window drop-down menu, select **Debug**. When the Debug window opens, click the window if it is not already selected.
- 3. Expand the Unassigned Debug Nets folder. The following figure shows those debug nets that were tagged with MARK\_DEBUG attributes in sinegen\_demo.vhd.

<ul> <li> Add mark_debug attributes to</li> <li>attribute mark_debug : string;</li> <li>attribute mark_debug of GPI0_BUI</li> <li>attribute mark_debug of GPI0_BUI</li> <li>attribute mark_debug of GPI0_BUI</li> <li>attribute mark_debug of DONT_EAR</li> <li>68</li> <li>69</li> </ul>	TTONS_db : signal TTONS_dly : signa TTONS_re : signal	<pre>is "true"; l is "true"; is "true";</pre>					
<pre>69 : 70</pre>							
Tcl Console Messages Log Repo	orts Design Ru	ns Debug ×					
Q  ጟ   ≑   兼   +   ≓							
Name	Driver Cell	Driver Pin					
✓							
✓ Jft GPIO_BUTTONS_db (2)	FDRE	Q					
_ □ C GPIO_BUTTONS_db[0]	FDRE	Q					
_      GPIO_BUTTONS_db[1]	FDRE	Q					
✓ 小窓 GPIO_BUTTONS_dly (2)	FDRE	Q					
_ ☐ ☎ GPIO_BUTTONS_dly[0]	FDRE	Q					
_ ☐ ☎ GPIO_BUTTONS_dly[1]	FDRE	Q					
✓ Jr¤ GPIO_BUTTONS_re (2)	FDRE	Q					
GPIO_BUTTONS_re[0]	FDRE	Q					
	FDRE	Q					
	FDRE	Q					
Debug Cores Debug Nets							

- 4. In the Netlist window, select the **Netlist** tab and expand Nets. Select the following nets for debugging as shown in the following figure.
  - **GPIO\_BUTTONS\_IBUF[0] and GPIO\_BUTTONS\_IBUF[1]:** Nets folder under the top-level hierarchy
  - sel(2): Nets folder under the U\_SINEGEN hierarchy



• sine(20): Nets folder under the U\_SINEGEN hierarchy



*Note*: These signals represent the significant behavior of this design and are used to verify and debug the design in subsequent steps.

5. Right-click the selected nets and select **Mark Debug** as shown in the following figure.



#### AMD**7** XILINX

Sources Netlist	×		? _ 🗆 🖾	Schemat	ic × s	inegen_der
¥ H			•	Q   🖬	🔸   /	→
🕅 sinegen_demo			^	51	signal	GPIO_BUTT
V > Nets (60)				52	-	GPIO_BUTT
> 🖑 GPIO_B	UTTONS (2)			53 54	signal	GPIO_BUTT
	BUTTONS db (2)			55	signal	DONT EATO
	BUTTONS_dly (2)			56	-	DONT_EAT1
	UTTONS_IBUF (2)			57	signal	DONT_EAT2
_				58	-	DONT_EAT3
	D_BUTTONS_IBUF[0	1		59	signal	DONT_EAT4 EAT
	D_BUTTONS_IBUF[1		Net Properties		Ctrl+E	LAI
>√f¤ GPIO_	BUTTONS_re (2)	Ť.	Mark Debug			debu
> 嵃 LEDS_n	(4)	- 19 h	-			Wark_d
〉 师 LEDS_n	_OBUF (2)		Unmark Debug			ark_d
_ <const0< th=""><th>&gt;</th><th><b></b></th><th>Assign to Debu</th><th>g Port</th><th></th><th>ark_d</th></const0<>	>	<b></b>	Assign to Debu	g Port		ark_d
_ <const1< th=""><th>&gt;</th><th></th><th>Select Driver Pi</th><th>n</th><th></th><th>ark_d ark_d</th></const1<>	>		Select Driver Pi	n		ark_d ark_d
Net Properties		Я	Schematic		F4	
			Show Connectiv	/ity	Ctrl+T	inege
_ GPIO_BUTTONS	S_IBUF[1]		Show Hierarchy		F6	
Name:	GPIO_BUTTONS		Highlight			▶ in
Type:	SIGNAL	_				in
Bus net:	F GPIO_BUTTC	~	Unhighlight			in
		$\otimes$	Mark			out
Route status:	Has unplaced por		Unmark		Ctrl+Shift	+M
Cell pin count:	4			79 🏳	end com	ponent;

6. Mark nets for debug in the Tcl Console. Mark nets "sine(20)" under the U\_SINEGEN hierarchy for debug by executing the following Tcl command.

set\_property mark\_debug true [get\_nets -hier [list {sine[\*]}]]

**TIP:** In the Debug window, you can see the unassigned nets you just selected. In the Netlist window, you can also see the green bug icon next to each scalar or bus, which indicates that a net has the attribute mark\_debug = true as shown in the following two figures.





Tcl Console	Messages	Log	Repor	rts	Design Ru	ns	Debug	×
Q	€   ¥   +							
Name				Driv	ver Cell	Dri	ver Pin	
🗠 🖨 Unassig	ined Debug Ne	ts (29)						
Y -∮r¤ GPI	O_BUTTONS_	<b>db</b> (2)		FDF	RE	Q		
_5≈(	SPIO_BUTTON	S_db[0]		FDF	RE	Q		
	SPIO_BUTTON			FDF		Q		
	O_BUTTONS_			FDF		Q		
	SPIO_BUTTON			FDF		Q		
_	SPIO_BUTTON			FDF		Q		
	O_BUTTONS_			IBU		0		
	SPIO_BUTTON	_		IBU		0		
_	-					0		
	O_BUTTONS_ 3PIQ_BUTTON							
Debug Cores					NF	Ω		
Sources	Netlist ×				? _		2	
품 처						-	≎	
sinegen_	demo						^	
~ <i>Nets (</i>								
		NS (2)						
	GPIO_BUTT		<b>Ib</b> (2)				-	
	GPIO_BUTT							
	_							
	GPIO_BUTT			_				
_	CPIO_B		_					
_F☆ GPIO_BUTTONS_IBUF[1]								
> √ŗ≋	GPIO_BUTT	ONS_r	e (2)					
> 🖑 LE	EDS_n (4)							
> J EDS_n_OBUF (2)								
)> آ	const0>							
)> آ	const1>						~	

#### **Running the Set Up Debug Wizard**

1. From the Debug window tool bar or Tools drop-down menu, select **Set Up Debug**. The Set up Debug wizard opens.



Tcl Console Messa	iges Log Rep	orts Desig	n Runs Debug	×
Q	, <b>+</b> ⊨			
Name	Set Up Debug	Driver Cell	Driver Pin	
👻 🖨 Unassigned Del	bug Nets (29)			
✓ -∰ <sup>™</sup> GPIO_BUTT	ONS_db (2)	FDRE	Q	
_ 🛱 GPIO_BU	JTTONS_db[0]	FDRE	Q	
_ SPIO_BU	FDRE	Q		
✓ J <sup>™</sup> GPIO_BUTT	ONS_dly (2)	FDRE	Q	
_∫≅ GPIO_BU	JTTONS_dly[0]	FDRE	Q	
_ Selo_BU	JTTONS_dly[1]	FDRE	Q	
∽ Jr̃¤ GPIO_BUTT	ONS_IBUF (2)	IBUF	0	
_ SPIO_BU	JTTONS_IBUF[0]	IBUF	0	
_F¤ GPIO_BU	JTTONS_IBUF[1]	IBUF	0	
✓ - J <sup>™</sup> CPIO_BUTT	ONS_re (2)	FDRE	Q	
	ITTONS, refot ug Nets	FDRF	Ω	

2. When the Set up Debug wizard opens, click **Next**.

🍌 Set Up Debug	
HLx Editions	<ul> <li>Set Up Debug</li> <li>This wizard will guide you through the process of <ol> <li>Choosing nets and connecting them to debug cores.</li> <li>Associating a clock domain with each of the nets chosen for debug.</li> <li>Choosing additional features on the debug cores like Data Depth, Advanced Trigger mode and Capture Control.</li> </ol> </li> <li>Note: This setup wizard does not apply to the VIO, IBERT or JTAG-to-AXI-Master debug cores. Please refer to Vivado Design Suite User Guide: Programming and Debugging (UG908) for further instructions on how to use these IPs.</li> </ul>
<b>?</b>	< <u>Back</u> <u>Einish</u> Cancel

3. In the Nets to Debug page, shown in the following figure, ensure that all the nets have been added for debug and click **Next**.





Q   ¥   €   №   M   +	-			
Name	Clock Domain	Driver Cell	Probe Type	
> Jra GPIO_BUTTONS_db (2)	clk	FDRE	Data and Trigger 👒	
> Jac GPIO_BUTTONS_dly (2)	clk	FDRE	Data and Trigger 👒	
> Jata GPIO_BUTTONS_IBUF (2)	clk	IBUF	Data and Trigger 👒	
> Jac GPIO_BUTTONS_re (2)	clk	FDRE	Data and Trigger 👒	
> √fr≊ U_SINEGEN/sel (2)	clk	FDRE	Data and Trigger 👒	
〜 小☆ U_SINEGEN/sine (20)	clk	FDRE	Data and Trigger 🛛 🗠	
_∫¤ sine[0]	clk	FDRE	Data and Trigger	
_∫¤ sine[1]	clk	FDRE	Data and Trigger	
_f∞ sine[2]	clk	FDRE	Data and Trigger	
_f¤ sine[3]	clk	FDRE	Data and Trigger	
_f¤ sine[4]	clk	FDRE	Data and Trigger	

- 4. In the ILA Core Options page, go to Trigger and Storage Settings section and select both **Capture Control** and **Advanced Trigger**. Click **Next**.
- 5. In the Setup Debug Summary page, make sure that all the information is correct and as expected. Click **Finish**.

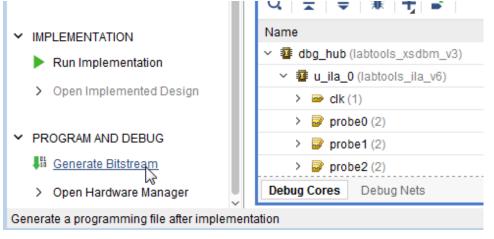
🏊 Set Up Debug		×
	Set up Debug Summary	
HLx Editions	I 0 debug cores will be removed	
	1 debug core will be created	
	1 Found 1 clock	
	✓ Open in Debug layout To apply the above changes, click Finish	
(?)	< <u>B</u> ack <u>N</u> ext > <u>Finish</u> Cancel	

Upon clicking Finish, the relevant XDC commands that insert the ILA core(s) are generated.



## Step 4: Implementing and Generating Bitstream

1. In the Flow Navigator, under Program and Debug, click Generate Bitstream.



- In the Save Project dialog box click Save. If a dialog box appears indicating this will cause the Synthesis results to go out of date, click OK. This applies the MARK\_DEBUG attributes on the newly marked nets. You can see those constraints by inspecting the sinegen\_demo\_kc705.xdc file.
- 3. When the No Implementation Results Available dialog box pops up, click **Yes**. In the Launch Runs dialog box, accept all of the default settings (Launch runs on local host) and click **OK**.
- 4. When the bitstream generation completes, the Bitstream Generation Completed dialog box pops up. Click **OK**.
- 5. In the dialog box asking you to closetye synthesized design before opening the implemented design. Click **Yes**.
- 6. Examine the Timing Summary report to ensure that all the specified timing constraints are met.

Clock Summary (4)         Total Negative Slack (TNS):         0.000 ns         Total Hold Slack (THS):         0.000 ns         Total Pulse Width Negative Slack (TPWS):         0.           > Check Timing (0)         Number of Failing Endpoints:         0         Number of Failing Endpoints:         0         Number of Failing Endpoints:         0	732 ns
Clock Summary (4) Total Negative Slack (TNS): 0.000 ns Total Hold Slack (THS): 0.000 ns Total Pulse Width Negative Slack (TPWS): 0. Clock Timing (0) Number of Failing Endpoints: 0 Number of Failing Endpoints: 0 Number of Failing Endpoints: 0 Clock Timing (0)	732 ns
Check Timing (0) Number of Failing Endpoints: 0 Number of Failing Endpoints: 0 Number of Failing Endpoints: 0	
	000 ns
Intra-Clock Paths Total Number of Endpoints: 12755 Total Number of Endpoints: 12755 Total Number of Endpoints: 166	
	938
Inter-Clock Paths All user specified timing constraints are met.	
George Contemport	
User Ignored Paths	
Unconstrained Paths	

Proceed to Lab 5: Using the Vivado Logic Analyzer to Debug Hardware to complete the rest of the steps for debugging the design.

Send Feedback



# Using the HDL Instantiation Method to Debug a Design

The HDL instantiation method is one of the two methods supported in the Vivado<sup>®</sup> tool debug probing. For this flow, you will generate an ILA IP using the Vivado IP Catalog and instantiate the core in a design manually as you would with any other IP.

## Step 1: Creating a Project with the Vivado New Project Wizard

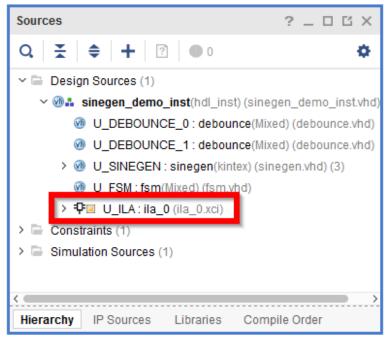
To create a project, use the New Project wizard to name the project, to add RTL source files and constraints, and to specify the target device.

- 1. Invoke the Vivado<sup>®</sup> IDE.
- 2. In the Quick Start tab, click **Create Project** to start the New Project wizard. Click **Next**.
- 3. In the Project Name page, name the new project proj\_hdl and provide the project location (C:/Vivado\_Debug). Ensure that Create project subdirectory is selected. Click Next.
- 4. In the Project Type page, specify the Type of Project to create as RTL Project. Click Next.
- 5. In the Add Sources page:
  - a. Set Target Language to VHDL.
  - b. Click the "+" sign, and then click Add Directories.
  - c. In the Add Source Directories dialog box, navigate to the /src/lab2 directory, and choose the sine\_high, sine\_low, sine\_mid, and ila\_0 directories. Click Select.
  - d. Verify that the directories are added, and Copy Sources into Project is selected.
  - e. Click the "+" sign, and then click Add File.
  - f. In the Add Source Files dialog box, navigate to the/src/lab2 directory and choose debounce.vhd, fsn.vhd, sinegen.vhd, and sinegen\_demo\_inst.vhd files. Click OK.
  - g. Verify that the sources and directories are added, and that **Copy Sources into Project** is selected. Click Next.

Send Feedback



- 6. In the Add Constraints dialog box, click the "+" sign, and then click Add Files.
- 7. Navigate to /src/lab1 directory and select sinegen\_demo\_kc705.xdc. Click Next.
- 8. In the Default Part dialog box, specify the **xc7k325tffg900-2** part for the KC705 platform. You can also select **Boards** and then select **Kintex-7 KC705 Evaluation Platform**. Click **Next**.
- 9. Review the New Project Summary page. Verify that the data appears as expected, per the steps above. Click **Finish**.
- 10. In the Sources window in Vivado IDE, expand sinegen\_demo\_inst to see the source files for this lab. Note that ila\_0 core has been added to the project.



11. Double-click the sinegen\_demo\_inst.vhd file, shown in the following figure to open it and inspect the instantiation and port mapping of the ILA core in the HDL code.

```
-- ILA
U_ILA : ila_0 .
port map
(
    CLK => clk,
    PROBE0 => sineSel,
    PROBE1 => sine,
    PROBE2 => GPIO_BUTTONS_db,
    PROBE3 => GPIO_BUTTONS_re,
    PROBE3 => GPIO_BUTTONS_re,
    PROBE4 => GPIO_BUTTONS_dly,
    PROBE5 => GPIO_BUTTONS
);
```



## Step 2: Synthesize Implement and Generate Bitstream

1. From the Program and Debug drop-down list, in Flow Navigator, click **Generate Bitstream**. This will synthesize, implement and generate a bitstream for the design.



- 👫 <u>Generate Bitstream</u>
- > Open Hardware Manager

0.20.
Modified:
Copied to:
Copied from:
Copied on:
<
General Pro

- 2. The No Implementation Results Available dialog box appears. Click **Yes**. In the Launch Runs dialog box, accept all of the default settings (Launch runs on local host) and click **OK**.
- 3. After bitstream generation completes, the Bitstream Generation Completed dialog box appears. Open Implemented Design is selected by default. Click **OK**.
- 4. In the Design Timing Summary window, ensure that all timing constraints are met.

TCI Console   Messages   Log   R Q   ★   ♦   ●	pports Design Runs IP Status Power DRC Methodology Timing ×	? _ 0 0
General Information Timer Settings	Setup Hold	Pulse Width
Design Timing Summary Clock Summary (4) Check Timing (0) Check Timing (0) Inter-Clock Paths Other Path Groups	Worst Negative Slack (WNS):         0.511 ns         Worst Hold Slack (WHS):         0.044 ns           Total Negative Slack (TNS):         0.000 ns         Total Hold Slack (THS):         0.000 ns           Number of Failing Endpoints:         0         Number of Failing Endpoints:         0           Total Number of Endpoints:         4437         Total Number of Endpoints:         4437           All user specified timing constraints are met.	Worst Pulse Width Slack (WPWS):     1.732 ns       Total Pulse Width Negative Slack (TPWS):     0.000 ns       Number of Failing Endpoints:     0       Total Number of Endpoints:     2478
User Ignored Paths Unconstrained Paths		

5. Proceed to Lab 5: Using the Vivado Logic Analyzer to Debug Hardware chapter to complete the rest of this lab.



## Lab 3

# Using a VIO Core to Debug a Design in Vivado Design Suite

The Virtual Input/Output (VIO) core is a customizable core that can both monitor and drive internal FPGA signals in real time. The number and width of the input and output ports are customizable in size to interface with the FPGA design. Because the VIO core is synchronous to the design being monitored and/or driven, all design clock constraints that are applied to your design are also applied to the components inside the VIO core. Run time interaction with this core requires the use of the Vivado<sup>®</sup> tool's logic analyzer feature. The following figure is a block diagram of the new VIO core.

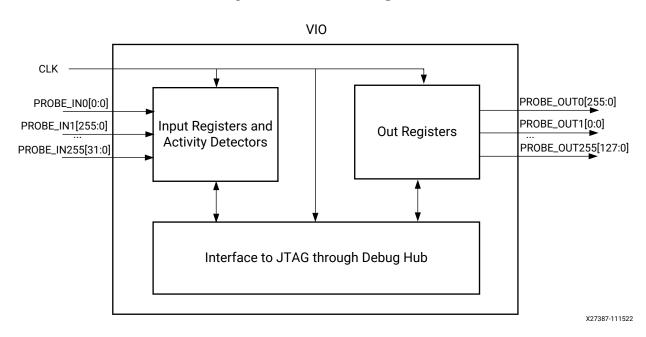


Figure 2: VIO Block Diagram

This lab walks you through the steps of instantiating and configuring the VIO core. It walks you through the steps of connecting the I/Os of the design to the VIO core. This way, you can debug your design when you do not have access to the hardware or the hardware is remotely located.

The following ports are created:



- One four-bit PROBE\_INO port. This has two bits to monitor the two-bit Sine Wave selector outputs from the finite state machine (FSM) and other two bits to mimic the state of the other two LEDs on the board. You will configure these four-bit signals as LEDs during run time to mimic the LEDs displayed on the KC705 board.
- One two-bit PROBE\_OUT0 port to drive the input buttons on the FSM. You will configure it so one bit can be used as a toggle switch during run time to mimic PUSH\_BUTTON switch SW3, and the second bit will be used as PUSH\_BUTTON switch SW6.

## Step 1: Creating a Project with the Vivado New Project Wizard

To create a project, use the New Project wizard to name the project, add RTL source files and constraints, and specify the target device.

- 1. Invoke Vivado IDE.
- 2. In the Quick Start tab, click Create Project to start the New Project wizard. Click Next.
- 3. In the Project Name page, name the new project **proj\_hdl\_vio** and provide the project location (C:/Vivado\_Debug). Ensure that the **Create project subdirectory** is selected. Click **Next**.
- 4. In the Project Type page, specify the Type of Project to create as **RTL Project**. Click **Next**.
- 5. In the Add Sources page:
  - a. Set Target Language to VHDL.
  - b. Click Add Files.
  - c. In the Add Source Files dialog box, navigate to the /src/lab3 directory.
  - d. Select all VHD source files, and click **OK**.
  - e. Verify that the files are added, and **Copy Sources into Project** is selected.
- 6. Click the "+" sign, and then click **Add Directories**.
- 7. In the Add Source Directories dialog box, navigate to the /src/lab3 directory and choose the sine\_high, sine\_low, sine\_mid, and ila\_0 directories. Click Select.
- 8. Verify that the directories are added and Copy sources into project is selected. Click Next.
- 9. In the Add Constraints dialog box, click the "+" sign, and then click Add Files.
- 10. Navigate to the /src/lab3 directory and select sinegen\_demo\_kc705.xdc. Click Next.
- 11. In the Default Part page, specify the **xc7k325tffg900-2** platform. You can also select **Boards** and then select **Kintex-7 KC705 Evaluation Platform**. Click **Next**.

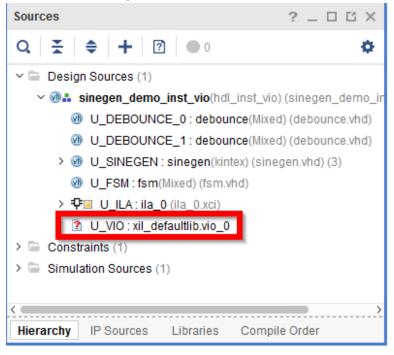




12. Review the New Project Summary page. Verify that the data appears as expected, in accordance with the previous steps. Click **Finish**.

*Note*: It might take a moment for the project to initialize.

13. In the Sources window in Vivado IDE, expand sinegen\_demo\_inst\_vio to see the source files for this lab. Note that the ila\_0 core has been added to the project. However, vio\_0 (the VIO core) is missing.



- 14. Instantiate and configure this VIO core as follows. From the Flow Navigator, click **IP Catalog**, expand **Debug & Verification**, then expand **Debug**, and double-click VIO. The Customize IP dialog box opens.
- 15. On the General Options tab, leave the Component Name as its default value of vio\_0, set Input Probe Count to 1, Output Probe Count to 1, and select the **Enable Input Probe Activity Detectors** check box.





in Customize IP			×
VIO (Virtual Input/Output) (3.0)			A
🚯 Documentation 📄 IP Location  C Switch	n to Defaults		
Show disabled ports	Component Name	vio_0	8
	To configure more than 64 probe ports use Viv General Options PROBE_IN Ports(00)	PROBE_OUT Ports(00)	
	Input Probe Count 1	[0 - 256]	
clk probe_in0[0:0]		0 - 256]	
	Enable Input Probe Activity Detectors		
		ОК	Cancel

16. On the PROBE\_IN Ports tab, set Probe Width to 4.

🍌 Customize IP					×
VIO (Virtual Input/Output) (3.0)				1	
Occumentation IP Location C Sw	itch to Defaults				
Show disabled ports	Component Name		vio_0		8
	To configure more than	64 probe ports use Viv	vado Tcl Console		
	General Options PF	ROBE_IN Ports(00)	PROBE_OUT Ports(00)		
	Probe Port		Probe Width [1 - 256 ]		
	PROBE_IN0		4	8	
- olk probe_out0[0:0] - probe_out0[0:0]					_
				OK Cancel	

17. On the PROBE\_OUT Ports tab, set Probe Width to **2** and Initial Value to **0x0**.



🖊 Customize IP				×
VIO (Virtual Input/Output) (3.0)				4
() Documentation 📄 IP Location C Swit	ch to Defaults			
Show disabled ports	Component Name		vio_0	8
	To configure more than 64 p	probe ports use Vivado Tcl (	Console	
	General Options PROB	E_IN Ports(00) PROBE	_OUT Ports(00)	
	Probe Port	Probe Width [1 - 256 ]	Initial Value (in hex)	
	PROBE_OUT0	2	🛞 0x0	0
olk probe_in0[3:0]				
			ОК	Cancel

18. Click **OK** to generate the IP. The Generate Output Products dialog box appears. Click **Generate**. An additional dialog box may appear indicating that an out-of-context module run has been launched, if so click **OK**.

👃 Generate Output Products	×
The following output products will be generated.	4
Preview	
<ul> <li>Instantiation Template</li> <li>Synthesized Checkpoint (.dcp)</li> <li>Behavioral Simulation</li> <li>Change Log</li> </ul>	
Synthesis Options	?
─ <u>G</u> lobal	
Out of context per IP	
Run Settings	
Number of jobs: 8 💌	
Apply Generate Sk	ip

Output product generation should take less than a minute. At this point, you have finished customizing the VIO. This core has already been instantiated in the top level design.



```
--- VIO

U_VIO : vio_0

port map
(

CLK => clk,

PROBE_IN0(3) => DONT_EAT,

PROBE_IN0(2) => GPIO_BUTTONS_re(1),

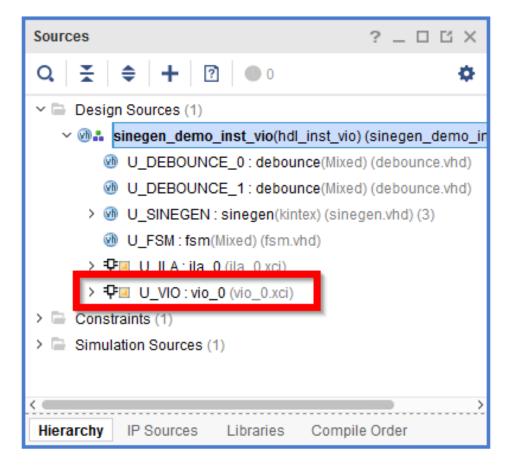
PROBE_IN0(1 downto 0) => sineSel,

PROBE_OUT0(1) => push_button_reset,

PROBE_OUT0(0) => push_button_vio

);
```

At this point, the Sources window should look as shown in the following figure.



19. Double-click **sinegen\_demo\_inst.vhd** in the Sources window to open it, and inspect the instantiation and port mapping of the ILA core in the HDL code.

# Step 2: Synthesize, Implement, and Generate the Bitstream

- 1. From the Program and Debug drop-down list in Flow Navigator, click **Generate Bitstream**. This synthesizes, implements, and generates a bitstream for the design
- 2. The Missing Implementation Results dialog box appears. Click OK.
- 3. After bitstream generation completes, the Bitstream Generation Completed dialog box appears. Open Implemented Design is selected by default. Click **OK**.
- 4. Inspect the Timing Summary report and make sure that all timing constraints have been met.

Tcl Console Messages Log	Reports Design Runs IP Status Por	er DRC Methodology Timing	×		? _ 🗆 🖸
Q 🛣 🖨 🜑	Design Timing Summary				
General Information Timer Settings	Setup	Hold		Pulse Width	
Design Timing Summary	Worst Negative Slack (WNS): 0.539	ns Worst Hold Slack (WHS):	0.044 ns	Worst Pulse Width Slack (WPWS):	1.732 ns
Clock Summary (4)	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 (0)	Number of Failing Endpoints: 0	Number of Failing Endpoints	: 0	Number of Failing Endpoints:	0
> 🗁 Intra-Clock Paths	Total Number of Endpoints: 4703	Total Number of Endpoints:	4703	Total Number of Endpoints:	2694
Inter-Clock Paths	All user specified timing constraints an	met.			
> 🗁 Other Path Groups	. ,				
User Ignored Paths					
Unconstrained Paths					
Timing Summary - impl_1 (saved)					

5. Proceed to Lab 5: Using the Vivado Logic Analyzer to Debug Hardware to complete the rest of the steps for debugging the design. Then proceed to the Verifying the VIO Core Activity (Only Applicable to Lab 3) section in Lab 5 Step 2 to complete the rest of this lab.



#### AMDA XILINX

### Lab 4

## Using the Synplify Pro Synthesis Tool and Vivado Design Suite to Debug a Design

This simple tutorial shows how to do the following:

- Create a Synplify Pro project for the wave generator design.
- Mark nets for debug in the Synplify Pro constraints file as well as VHDL source files.
- Synthesize the Synplify Pro project to create an EDIF netlist.
- Create a Vivado<sup>®</sup> project based on the Synplify Pro netlist.
- Use the Vivado IDE to setup and debug the design from the synthesized design using Synplify Pro.

## Step 1: Create a Synplify Pro Project

- 1. Launch Synplify Pro and select **File**  $\rightarrow$  **New**.
- 2. Set File Type to Project File (Project) as highlighted in the following figure.
- 3. In the New File Name box, enter **synplify\_1**.
- 4. Click OK.



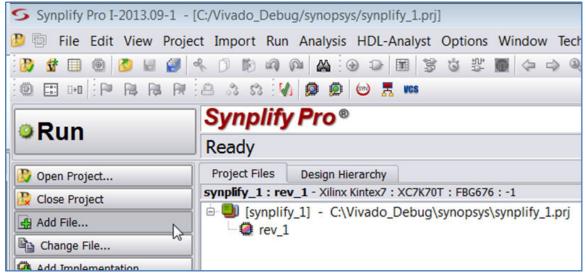


🕑 New	8 8
File Type:(Select a type)	
Verilog File	ОК
VHDL File Tcl Script	Cancel
Design Constraint	
<ul> <li>Text File</li> <li>Xilinx Options File</li> </ul>	Help
FPGA Design Constraints	
Analysis Design Constraints Project File (Project)	
Add To Project	
New File Name:	
synplify_1	
File Location:	
C:\Vivado_Debug\synopsys	
Full Path:	
C:\Vivado_Debug\synopsys\synplify_1.prj	

5. If you get a dialog box asking you to create a non-existing directory, click **OK**.

Synplify Pro		×
The directory C:\tutorials\ug936 does not exist. Do you wish to create it?		
	ОК	Cancel

6. In the left panel of the Synplify Pro window, click Add File as shown in the following figure.



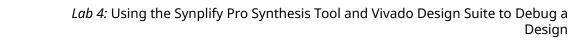


- 7. In the Add Files to Project dialog box, change the Files of Type to HDL File. Navigate to C:\Vivado\_Debug\src\lab4, which shows all the VHDL source files needed for this lab. Select the following three files by pressing the Ctrl key and clicking on them.
  - debounce.vhd
  - fsm.vhd
  - sinegen\_demo.vhd

#### 8. Click Add.

S Add Files to	o Project	83
Look in:	🗼 C:\Vivado_Debug\src\Lab4 🔹 🖸 🕤 🚺 📰 🗏	
My Com	puter debounce.vhd fsm.vhd sinegen_demo.vhd	
File name:	"debounce.vhd" "fsm.vhd" "sinegen_demo.vhd"	
Files of type:	HDL Files (*.vhd *.vhdl *.v *.sv *.vma)	
VHDL/Verilog lib		
Files to add to p	project: (3 file(s) selected) 🗹 Use relative paths 🗹 Add files to Folders 🛛 Folder Options	
.\src\Lab4\de		<- Add All
.\src\Lab4\fs .\src\Lab4\sir	n.vhd negen_demo.vhd	<- Add
		Remove All ->
		Remove ->
		ОК
		Cancel

9. In the same dialog box set Files of type to Constraints Files. This shows the synplify\_1.sdc file. Select the file and click Add as shown in the following figure.





S Add Files to Project	×
Look in: C:\Vivado_Debug\src\Lab4	
File name:     synplify_1.sdc       Files of type:     Constraint Files (*.sdc)       VHDL/Verilog lib:	
Files to add to project: (4 file(s) selected) 🗹 Use relative paths 🗹 Add files to Folders Folder Options	<- Add All <- Add Remove All -> Remove -> OK Cancel

10. In the same dialog box, set Files of type to FPGA Constraint Files. This shows the synplify\_1.fdc file. Select the file and click Add as shown in the following figure. Click OK.

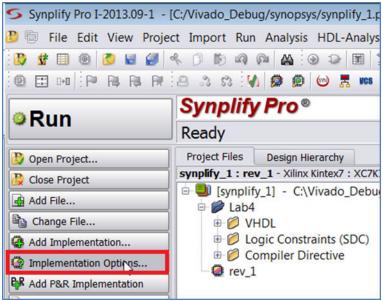


Design



5	Add	l Files to P	roject				×
Look in:	proj/xcoswmktg/smitha/vivado_de	ebug/lab4	-	• 🗢	1	:: 🔳	
Comp	Name	Size	Туре	Date Mo	dified $ riangle$		
🚞 smitha	synplify_1.fdc	468 bytes	fdc File	5/6/16 9:	18 AM		
		_					
File <u>n</u> ame:	synplify_1.fdc					_	
Files of type:	FPGA Constraint Files (*.fdc)					-	
VHDL/Verilog lit	b:					•	
Files to add to p	project: (1 file(s) selected) 🗹 Use relativ	e paths 🖌	Add files to	Folders	Folder	Options	
./lab4/synplify_	1.fdc						<- Add All
							<- Add
							Remove All ->
							Remove ->
							ОК
							Cancel

- 11. Now, you need to set the implementation options.
- 12. Click Implementation Options in the Synplify Pro window as shown in the following figure.



13. This brings up the Implementation Options dialog box as shown in the following figure. In the Device tab, set Technology to Xilinx Kintex7, Part to XC7K325T, Package to FFG900 and Speed to -2. Leave all the other options at their default values. Click **OK**.



Device Options Constraints	Implementa	ation Results	Timi	ng Report	High Re	liability	VHD I	Implementations:
Fechnology:		Part		Package:		Speed:		rev_1
Xilinx Kintex7	•	XC7K325T	•	FFG900	•	-2	•	
Device Mapping Options								
Option			_		Valu	Je		
Fanout Guide					100	10000		
Disable I/O Insertion								
Disable Sequential Optimizat	tions							
Update Compile Point Timing	g Data						•	
Click on an option for description								
System Designer Board File								

14. You need to preserve the net names that you want to debug by putting attributes in the HDL files. These attributes are already placed in the sinegen\_demo.vhd, file of this tutorial. Open the sinegen\_demo.vhd file and inspect the lines shown.

```
-- Attributes for Synplify Pro
attribute syn_keep : boolean;
attribute syn_keep of GPIO_BUTTONS_db : signal is true;
attribute syn_keep of GPIO_BUTTONS_dly : signal is true;
attribute syn_keep of GPIO_BUTTONS_re : signal is true;
```

15. You also can specify the MARK\_DEBUG attributes in the source HDL files to mark the signals for debug, as shown in the code snippet from singen\_demo.vhd file.

```
-- Add mark_debug attributes to show debug nets in the synthesized netlist
attribute mark_debug : string;
attribute mark_debug of GPIO_BUTTONS_db : signal is "true";
attribute mark_debug of GPIO_BUTTONS_dly : signal is "true";
attribute mark_debug of GPIO_BUTTONS_re : signal is "true";
```

16. The synplify\_1.sdc file contains various kinds of constraints such as pin location, I/O standard, and clock definition. The synplify\_1.fdc file contains directives for the compiler. Here is where the nets of interest to us that are marked for debug are located. The attribute and the nets selected for debug are shown in the following figure.

```
Attributes that are needed to mark_debug the nets that are needed to be viewed in ILA

define_attribute -comment {Mark sinegen as black box} {v:work.sinegen} {syn_black_box} {1}

define_attribute -comment {Set no_prune on sinegen} {v:work.sinegen} {syn_noprune} {1}

define_attribute -comment {Mark entire bus for debug} {i:sinegen.sine[*]} {mark_debug} {"true"}

define_attribute -comment {Mark entire bus for debug} {i:sinegen.sel[*]} {mark_debug} {"true"}
```





In the above constraints, sinegen has been defined as a black box by using the syn\_black\_box attribute. Second, the syn\_no\_prune attribute has been used so that the I/Os of this block are not optimized away. Finally, two nets, sine[20:0] and sel[1:0], have been assigned the MARK\_DEBUG attribute such that these two nets should show up in the synthesized design in Vivado<sup>®</sup> IDE for further debugging. For further information on these attributes, please refer to the Synplify Pro User Manual and Synplify Pro Reference Manual.

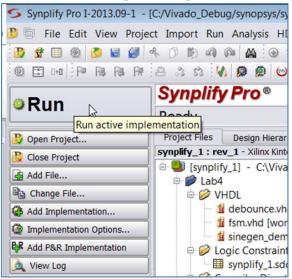
## Step 2: Synthesize the Synplify Project

1. Before implementing the project, you need to set the name for the output netlist file. By default, the name of the output netlist file is synplify\_l.edf. To change the name of the output file, type the following command at the Tcl command prompt:

%project -result\_file "./rev\_1/sinegen\_demo.edf"

You will use this file in Vivado<sup>®</sup> IDE.

2. With all the settings in place, click the **Run** button in the left panel of the Synplify Pro window to start synthesizing the design.



- 3. During synthesis, status messages appear in the Tcl Script tab. Warning messages are expected, but there should not be any Error messages. To see detailed messages, click the **Messages tab** in the bottom left-hand corner of the Synplify Pro console.
- 4. When synthesis completes, the output netlist is written to the file: rev\_1/ sinegen\_demo.edf

[Optional] To view the netlist select  $View \rightarrow View$  Result File.

5. Click File  $\rightarrow$  Save All to save the project, then click File  $\rightarrow$  Exit.



## Step 3: Create DCPs for the Black Box Created in Synplify Pro

The black box, sinegen, created in the Synplify Pro project, contains the Direct Digital Synthesizer IP. You need to create a synthesized design for this block. To do this, create an RTL type project in Vivado<sup>®</sup> IDE by following the steps outlined below.

- 1. Launch Vivado IDE.
- 2. Click Create Project. This opens up the New Project wizard. Click Next.
- 3. Under Project Name, set the project name to proj\_synplify\_netlist. Click Next.
- 4. Under Project Type, select RTL Project. Click Next.
- 5. Under Add Sources, click Add Files, navigate to the Vivado\_Debug/src/lab4 folder and select the sinegen.vhd file. Set Target Language to VHDL. Ensure that Copy sources into project box is selected. Click Next.
- 6. Click Add Files, navigate to the Vivado\_Debug/src/lab4 folder and select the sine\_high.xci, sine\_low.xci, and sine\_mid.xci files. Click Next.
- 7. Under Default Parts, select Boards and then select the **Kintex-7 KC705 Evaluation Platform** and correct version for your hardware. Click **Next**.
- 8. Under New Project Summary, ensure that all the settings are correct. Click Finish.
- 9. Once the project has been created, in Vivado Flow Navigator, under the Project Manager folder, click **Settings**. In the dialog box, in the left panel, click **Synthesis**. From the pull-down menu on the right panel, set -flatten\_hierarchy to none. Click **OK**.
- 10. In Vivado IDE Flow Navigator, under Synthesis Folder, click Run Synthesis.
- 11. When synthesis completes the Synthesis Completed dialog box appears. Select **Open Synthesized Design** and click **OK**.
- 12. Click File  $\rightarrow$  Exit in Vivado IDE. When the OK to exit dialog box pops up, click OK.

## Step 4: Create a Post Synthesis Project in Vivado IDE

- 1. Launch the Vivado IDE.
- 2. Click Create Project. This opens up the New Project wizard. Click Next.
- 3. Set the Project Name to proj\_symplify. Click Next.
- 4. Under Project Type, select Post-synthesis Project. Click Next.





- 5. Under Add Netlist Sources, click Add Files, navigate to the Vivado\_Debug/synopsys/ rev\_1 folder, and select sinegen\_demo.edf. Click OK.
- 6. Add the netlist file created in the previous section. Click Add Files again, navigate to the proj\_synplify\_netlist/proj\_synplify\_netlist.runs/synth1 folder and select sinegen.dcp.

Add the DCP files created for the sub-module IPs in the previous section. Click Add Directories again, navigate to the proj\_synplify\_netlist/ proj\_synplify\_netlist.srcs/sources\_1/ip folder and select the following:

- sine\_high
- sine\_mid
- sine\_low

Click **OK** in the Add Source Files dialog box. In the Add Netlist Sources dialog box ensure that Copy Sources into Project is selected. Click **Next**.

- 7. Click Add Files, navigate to the Vivado\_Debug/src folder, and select the sinegen\_demo\_kc705.xdc file. This file has the appropriate constraints needed for this Vivado project. Click OK in the Add Constraints File dialog box. In the Add Constraints (optional) dialog box ensure that Copy Constraints into Project is selected. Click Next.
- 8. Under Default Part, select **Boards** and then select **Kintex-7 KC705 Evaluation Platform** and the right version number for your hardware. Click **Next**.
- 9. Under New Project Summary, ensure that all the settings are correct and click Finish.

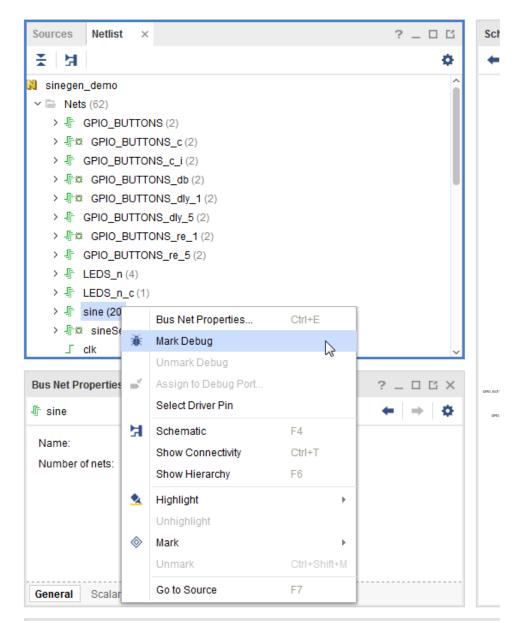
10. In the Sources window, ensure sinegen\_demo.edf is selected as the top module.

#### **Step 5: Add More Debug Nets to the Project**

- 1. In Vivado<sup>®</sup> IDE, in the Flow Navigator, select **Open Synthesized Design** from the Netlist Analysis folder.
- 2. Select the Netlist tab in the Netlist window to expand Nets. Select the following nets for debugging:
  - GPIO\_BUTTONS\_c(2)
  - sine (20)

After selecting all the specified nets, right-click the nets and click **Mark Debug**, as shown in the following figure.

#### AMD7 XILINX



3. You should be able to see all the nets that are marked for debug, as shown in the following figure.





Tcl Console Messages Log Re	ports Desi	gn Runs I	Debug ×	
Q   ≚   ♦   ₩   <b>+</b>   ≓				
Name	Driver Cell	Driver Pin	Probe Type	
<ul> <li>Unassigned Debug Nets (30)</li> </ul>				
> JFt GPIO_BUTTONS_c (2)	IBUF	0		
> Jrt GPIO_BUTTONS_db (2)	FDRE	Q		
> <b>近窓 GPIO_BUTTONS_dly_1</b> (2)	FDRE	Q		
> Jrt GPIO_BUTTONS_re_1 (2)	FDRE	Q		
〉 √f章 sine (20)	FDRE	Q		
> 近☆ sineSel (2)	FDRE	Q		
Debug Cores Debug Nets				

#### Running the Set Up Debug Wizard

1. Click the **Set up Debug** icon in the Debug window or select the Tools menu, and select **Set up Debug**. The Set up Debug wizard opens.

Tcl Console Messages Log	Reports	Design Runs	Debug ×	
Q   ¥   ♦   ¥   ⊨				
Name Set Up Debug	Driver (	Cell Driver Pin	Probe Type	
<ul> <li>Unassigned Debug Nets (30)</li> </ul>	,			
> Jfr ☎ GPIO_BUTTONS_c (2)	IBUF	0		
> 师篇 GPIO_BUTTONS_db (2)	FDRE	Q		
> 小窓 GPIO_BUTTONS_dly_1 (2)	FDRE	Q		
> Jftஜ GPIO_BUTTONS_re_1 (2)	FDRE	Q		
> - <b>√</b> ≊ sine (20)	FDRE	Q		
> -师章 sineSel (2)	FDRE	Q		

2. Click through the wizard to create Vivado<sup>®</sup> logic analyzer debug cores, keeping the default settings.

*Note*: In the Specify Nets to Debug dialog box, ensure that all the nets marked for debug have the same clock domain.

# Step 6: Implementing the Design and Generating the Bitstream

- 1. In the Flow Navigator, under the Program and Debug drop-down list, click **Generate Bitstream**.
- 2. In the Save Project dialog box, click **Save**.
- 3. When the Bitstream generation finishes, the Bitstream Generation Completed dialog box pops up and Open Implemented Design is selected by default. Click **OK**.
- 4. If you get a dialog box asking to close the synthesized design before opening the implemented design, click **Yes**.
- 5. Proceed to Lab 5: Using the Vivado Logic Analyzer to Debug Hardware to complete the rest of this lab.





#### Lab 5

# Using the Vivado Logic Analyzer to Debug Hardware

The final step in debugging is to connect to the hardware and debug your design using the Integrated Logic Analyzer (ILA). Before continuing, make sure you have the KC705 hardware plugged into a machine.

In this step, you learn:

- How to debug the design using the Vivado<sup>®</sup> logic analyzer.
- How to use the currently supported Tcl commands to communicate with your target board (KC705).
- How to discover and correct a circuit problem by identifying unintended behaviors of the push-button switch.
- Useful techniques for triggering and capturing design data.

# Step 1: Verifying Operation of the Sine Wave Generator

After doing some setup work, you will use Vivado logic analyzer to verify that the sine wave generator is working correctly. Your two primary objectives are to verify that:

- All sine wave selections are correct.
- The selection logic works correctly.

#### **Target Board and Server Set Up**

- Connecting to the Target Board Remotely: If you plan to connect remotely, you need to make sure that the KC705 board is plugged into a machine and you are running an hw\_server application on that machine. If you plan to connect locally, skip steps 1-5 below and go directly to the Connecting to the Target Board Locally section.
  - 1. Connect the Digilent USB JTAG cable of your KC705 board to a USB port on a Windows system.



- 2. Ensure that the board is plugged in and powered on.
- 3. Power cycle the board to clear the device.
- 4. Turn DIP switch positions (pin 1 on SW11, De-bounce Enable) to the OFF position.
- 5. Assuming you are connecting your KC705 board to a 64-bit Windows machine and you will be running the hw\_server from the network instead of your local drive, open a cmd prompt and type the following:

```
<Xilinx_Install>\Vivado\2020.x\bin\hw_server
```

Leave this cmd prompt open while the hw\_server is running. Note the machine name that you are using, you will use this later when opening a connection to this instance of the hw\_server application.

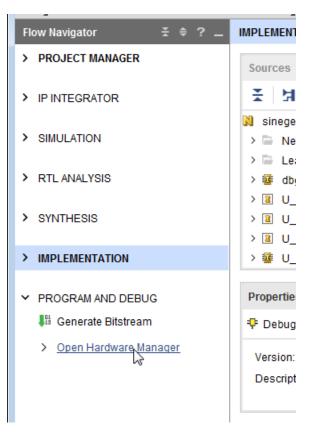
- **Connecting to the Target Board Locally:** If you plan to connect locally, ensure that the KC705 board is plugged into a Windows machine and then perform the following steps:
  - 1. Connect the Digilent USB JTAG cable of your KC705 board to a USB port on a Windows system.
  - 2. Ensure that the board is plugged in and powered on.
  - 3. Power cycle the board to clear the device.
  - 4. Turn DIP switch positions (pin 1 on SW11, De-bounce Enable) to the OFF position.

#### Using the Vivado Integrated Logic Analyzer

1. In the Flow Navigator, under Program and Debug, select **Open Hardware Manager**.



#### AMD**Z** XILINX



2. The Hardware Manager window opens. Click **Open Target**  $\rightarrow$  **Open New Target**.

HARDWARE MANAGER - unconnected								
🚯 No hardware target is open. Op	1 No hardware target is open. Open target							
Hardware	ø	Auto Connect						
		Recent Targets	*					
		Available Targets on Server	- F					
		Open New Target						
No conten	nt		0,					

- 3. The Open New Hardware Target wizard opens. Click Next.
- 4. In the Hardware Server Settings page, type the name of the server (or select **Local server** if the target is on the local machine) in the Connect to field. Click **Next**.



🅕 Open New Hardware Target	×
Hardware Server Settings Select local or remote hardware server, then configure the host name and port settings. Use Local server if the target is attached to the local machine; otherwise, use Remote server.	A
<u>C</u> onnect to: Local server (target is on local machine) ✓	
Click Next to launch and/or connect to the hw_server (port 3121) application on the local machine.	
	ncel

*Note*: Depending on your connection speed, this may take about 10 to 15 seconds.

5. If there is more than one target connected, you will see multiple entries in the Select **Hardware Target** page. In this tutorial, there is only one target, as shown in the following figure. Click **Next**.





🔥 Open New H	ardware Targ	et				×		
Select a hardware	Select Hardware Target Select a hardware target from the list of available targets, then set the appropriate JTAG clock (TCK) frequency. If you do not see the expected devices, decrease the frequency or select a different target.							
Hardware <u>T</u> arg	jets							
Туре	Name		JTAG Clock Fre	equency				
xilinx_tcf	Xilinx/Port_#00	03.Hub_#0004	6000000	~				
Hardware <u>D</u> evi	ces (for unknow		nx Virtual Cable () and the Instruction		R) length)			
Name	ID Code	IR Length						
@ xc7k325t_	0 33651093	6						
Hardware serv	er: localhost:312	1						
?		< <u>B</u>	ack	lext >	Einish	Cancel		

6. In the Open Hardware Target Summary page, click **Finish** as shown in the following figure.



#### AMDA XILINX

🥕 Open New Hardware	e Target	×
	Open Hardware Target Summary	
HLx Editions	<ul> <li>Hardware Server Settings:</li> <li>Server: localhost:3121</li> </ul>	
	<ul> <li>Target Settings:         <ul> <li>Target: xilinx_tcf/Xilinx/Port_#0003.Hub_#0004</li> <li>Frequency: 6000000</li> </ul> </li> </ul>	
E XILINX ALL PROGRAMMABLE.	To connect to the hardware described above, click Finish	
?	< <u>B</u> ack <u>N</u> ext > <u>Finish</u> Can	icel

7. Wait for the connection to the hardware to complete. The dialog in following figure appears while hardware is connecting.



After the connection to the hardware target is made, the Hardware window appears as in the following figure.

*Note*: The Hardware tab in the Debug view shows the hardware target and XC7K325T device detected in the JTAG chain.





? _ 🗆	с×
	•
Status	
Connected	
Open	
Not programmed	
	Status Connected Open

8. Next, program the XC7K325T device using the previously created .bit bitstream by rightclicking the XC7K325T device and selecting **Program Device** as shown in the following figure.

Hardware			? _ 🗆	с×	
Q   素   ♦   ∅   ▶   ≫				•	
Name		Status			
<ul> <li>Iocalhost (1)</li> </ul>		Connected			
✓ Ø xilinx_tcf/Xilinx/Port_#000	)3.Hu	Open			
✓	Hard	dware Device F	Properties		Ctrl+E
C		gram Device fy Device esh Device			G
		Configuration t from Configur	-		
		Program BBR Key Clear BBR Key			
Hardware Device Properties	Prog	gram eFUSE R	egisters		
<pre>     xc7k325t_0 </pre>	Expo	ort to Spreadsh	ieet		

9. In the Program Device dialog box verify that the .bit and .ltx files are correct for the lab that you are working on and click **Program** to program the device as shown in the following figure.



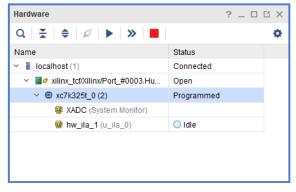
#### AMD7 XILINX

🍌 Program Device		<b>×</b>				
Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.						
Bitstre <u>a</u> m file: Debu <u>q</u> probes file: ☑ <u>E</u> nable end of st	C:/Vivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1/sinegen_demo.bit C:/Vivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1/sinegen_demo.ttx artup check	<ul><li>○</li><li>···</li></ul>				
•	<u>P</u> rogram	Cancel				

**CAUTION!** The file paths of the bitstream and debug probes to be programmed will be different for different labs. Ensure that the relative paths are correct.

Note: Wait for the program device operation to complete. This may take few minutes.

10. Ensure that an ILA core was detected in the Hardware panel of the Debug view.



11. The Integrated Logic Analyzer dashboard opens, as shown in the following figure.

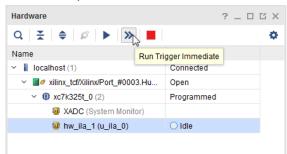




hw	_ila_1	? 🗆 Ľ X
	Waveform - hw_ila_1	? _ 🗆 ×
ions	Q   +   −   ♂   ▶   ≫   ■   ⊉   @   Q   ∷   ≠	12   27   <b>+</b> [   [4   4]   [4   ]
Dashboard Options	ILA Status:Idle	^
hboai	Name Value 0	20   20   30   40   50   60   60   1   20   50   50   50   50   50   50   50
Das	Image: Dont_EAT         Image: Mark Control (100)         Image: Mark Control (100)	
	Settings - hw_ila_1 Status - hw_ila_1 × ? _	Trigger Setup - hw_ila_1 × Capture Setup - hw_ila_1 ? _ □
	년   ▶   ≫   ■   무습	$ \alpha  +  -   \phi_{\lambda} $
	Core status           Idle         Waitling for Trigger         Post-Trigger         Full           Capture status	Press the 🕂 button to add probes.

#### **Verifying Sine Wave Activity**

1. In the Hardware window, click **Run Trigger Immediate** to trigger and capture data immediately as shown in shown in the following figure.



2. In the Waveform window, verify that there is activity on the 20-bit sine signal as shown in the following figure.





Waveform - hw_ila_1						? _ 🗆 X
Q   <b>+</b>   <b>-</b>   ϑ   ▶   ≫	<b>e</b>	Q 🔀 📲 🕅	▶   1≝   ±r   +F   F	←   →    ⊡-		0
ILA Status: Idle						1,023
Name	Value	0	200	400	600	800  1 <sub>.</sub> 0
₩ DONT_EAT	0					
> 🔣 GPIO_BUTTONS_db[1:0]	0			0		
> 🔣 GPIO_BUTTONS_dly[1:0]	0			0		
> 🔣 GPIO_BUTTONS_IBUF[1:0]	0			0		
> M GPIO_BUTTONS_re[1:0]	0			0		
> 🔣 U_SINEGEN/sel[1:0]	0			0		
> 🔣 U_SINEGEN/sine[19:0]	05133					
		Updated at: 2017-	Mar-16 14:59:13			

#### **Displaying the Sine Wave**

1. Right-click U\_SINEGEN/sine[19:0] signals, and select Waveform Style → Analog as shown in the following figure.

Vaveform - hw_ila_1		? _ 🗆 X
Q   <b>+</b>   <b>−</b>   ϑ   ▶   ≫	🔁 🔍	Q   X   •     •   •   •   •   •   •     □   •   □   •   □   □
ILA Status:Idle		1,023
Name	Value	0,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
W DONT_EAT	0	
M GPIO_BUTTONS_db[1:0]	0	
GPIO_BUTTONS_dly[1:0]	0	0
GPIO_BUTTONS_IBUF[1:0]	0	0
GPIO_BUTTONS_re[1:0]	0	0
🛛 🔣 U_SINEGEN/sel[1:0]	0	0
➡ U_SINEGEN/sine[19:0]	05133	
	<	Updated at: 2017-Mar-16 14:59:13

**CAUTION!** The waveform does not look like a sine wave. This is because you must change the radix setting from Hex to Signed Decimal, as described in the following subsection.

2. Right-click U\_SINEGEN/sine[19:0] signals, and select Radix → Signed Decimal.

You should now be able to see the high frequency sine wave as shown in the following figure instead of the square wave.

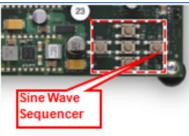


Waveform - hw_ila_1		? _ 🗆
Q   <b>+</b>   <b>−</b>   ♂   ▶   ≫	<b>e</b>	Q   X   ■
ILA Status:Idle		1,023
Name	Value	0
UDONT_EAT	0	
> 💐 GPIO_BUTTONS_db[1:0]	0	0
> 💐 GPIO_BUTTONS_dly[1:0]	0	0
> 📲 GPIO_BUTTONS_IBUF[1:0]	0	0
> Mage: GPIO_BUTTONS_re[1:0]	0	0
> 📲 U_SINEGEN/sel[1:0]	0	0
> ₱\$ U_SINEGEN/sine[19:0]	20787	
		Updated at: 2017-Mar-16 14:59:13

#### **Correcting Display of the Sine Wave**

To view the mid, and low frequency output sine waves, perform the following steps:

1. Cycle the sine wave sequential circuit by pressing the GPIO\_SW\_E push button as shown in the following figure.



2. Click **Run Trigger Immediately** again to see the new sine selected sine wave. You should see the mid frequency as shown in the following figure. Notice that the sel signal also changed from 0 to 1 as expected.

Waveform - hw_ila_1		? _ 🗆 X
Q + - & > 8	🕞 🔍	Q   X   ••     •   •   •   •   •   •
ILA Status:Idle		1,023
Name	Value	2 0
₩ DONT_EAT	0	
> 🔣 GPIO_BUTTONS_db[1:0]	0	0
> 💐 GPIO_BUTTONS_dly[1:0]	0	0
> 🔣 GPIO_BUTTONS_IBUF[1:0]	0	
> Variable Control Strate S	0	
> 🔣 U_SINEGEN/sel[1:0]	1	1
> 🍽 U_SINEGEN/sine[19:0]	-83150	
		Updated at: 2017-Mar-16 15:02:38

3. Repeat step 1 and 2 to view other sine wave outputs.



Waveform - hw_ila_1						? _ 🗆 ×
Q + - ♂ ► ≫	🕒 🖪	Q 23 - IA N	ter er	Fe   of   Dot		0
ILA Status: Idle						1,023 ^
Name	Value		200	400	eoo	800  1, <mark>0</mark>
1 DONT_EAT	0					
GPIO_BUTTONS_db[1:0]	0			0		
> W GPIO_BUTTONS_dly[1:0] > W GPIO_BUTTONS_IBUF[1:0]	0 0			0		
GPIO_BUTTONS_IBUF[1:0]	0			0		
> U_SINEGEN/sel[1:0]	2			2		
V_SINEGEN/sine[19:0]	-377487					
	<	Updated at: 2017-Max > < C	-16 15:03:22			~ >
Waveform - hw_ila_1						? _ 🗆 ×
Q   +   −   &   ►   ≫	🕒 🕞	Q   22   <b>-</b>	nt tr H	Fe   +F   I=I		٥
ILA Status: Idle						1,023 ^
Name	Value	• · · · · · · · · · · · ·	200	400	600	800  1,0
We DONT_EAT	0					
> 📲 GPIO_BUTTONS_db[1:0] > 📲 GPIO_BUTTONS_dly[1:0]	0 0	0		0		
GPIO_BUTTONS_div[1:0]	0			0		
> GPIO_BUTTONS_re[1:0]	0			0		
> 💐 U_SINEGEN/sel[1:0]	3			3		
> 🍣 U_SINEGEN/sine[19:0]	75777					
	<	Updated at: 2017-Max	r-16 15:03:56			

**Note:** As you sequence through the sine wave selections, you may notice that the LEDs do not light up in the expected order. You will debug this in the next section of this tutorial. For now, verify for each LED selection, that the correct sine wave displays. Also, note that the signals in the Waveform window have been re-arranged in the previous three figures.

### Step 2: Debugging the Sine Wave Sequencer State Machine (Optional)

As you corrected the sine wave display, the LEDs might not have lit up in sequence as you pressed the Sine Wave Sequencer button. With each push of the button, there should be a single, cycle-wide pulse on the GPIO\_BUTTONS\_re[1] signal. If there is more than one, the behavior of the LEDs becomes irregular. In this section of the tutorial, use Vivado logic analyzer to probe the sine wave sequencer state machine, and to view and repair the root cause of the problem.

Before starting the actual debug process, it is important to understand more about the sine wave sequencer state machine.

Send Feedback

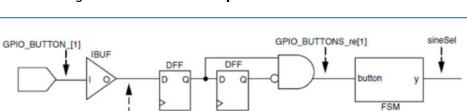


#### Sine Wave Sequencer State Machine Overview

The sine wave sequencer state machine selects one of the four sine waves to be driven onto the sine signal at the top-level of the design. The state machine has one input and one output. The following figure shows the schematic elements of the state machine. Refer to this diagram as you read the following description and as you perform the steps to view and repair the state machine glitch.

- The input is a scalar signal called "button". When the button input equals "1", the state machine advances from one state to the next.
- The output is a 2-bit signal vector called "Y", and it indicates which of the four sine wave generators is selected.

The input signal button connects to the top-level signal  $GPIO_BUTTONS_re[1]$ , which is a low-to-high transition indicator on the Sine Wave Sequencer button. The output signal Y connects to the top-level signal, sineSel, which selects the sine wave.



#### Figure 3: Sine Wave Sequence Button Schematic

#### Viewing the State Machine Glitch

GPIO\_BUTTON\_IBUF\_1

You cannot troubleshoot the issue identified above by connecting a debug probe to the GPIO\_BUTTON [1] input signal itself. The GPIO\_BUTTON [1] input signal is a PAD signal that is not directly accessible from the FPGA fabric. Instead, you must trigger on low-to-high transitions (rising edges) on the GPIO\_BUTTON\_IBUF signal, which is connected to the output of the input buffer of the GPIO\_BUTTON [1] input signal.

As described earlier, the glitch reveals itself as multiple low-to-high transitions on the GPIO\_BUTTONS\_IBUF\_1 signal, but it occurs intermittently. Because it could take several button presses to detect it, you will now set up the Vivado logic analyzer tool to Repetitive Trigger Run Mode. This setting makes it easier to repeat the button presses and look for the event in the Waveform viewer.

- 1. Under the Settings tab for hw\_ila\_1, configure the following:
  - Trigger Mode to BASIC\_ONLY
  - Capture Mode to BASIC
  - Window Data Depth to 1024

X12118



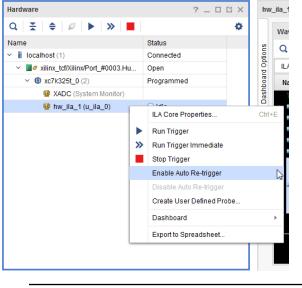
- Trigger position to 512
- Press the + button in the Trigger Setup window and add probe GPIO\_BUTTONS\_IBUF\_1. Change the Value field to RX by selecting the value RX in the Value field, as shown in the following figure.

Waveform - hw_ila_1										? _ 🗆 ×
Q + = ♂ ► ≫ ■	👍 🔍 Q	20   <b>+</b> [   ]	<   ▶    ±	:   ±r   +Γ						٠
ILA Status: Idle										1,023
Name	Value	°	200		400	600		800		1 <mark>,</mark> 1
	0	/			0					
> W GPIO_BUTTONS_db[1:0] () > W GPIO_BUTTONS_dly[1:0] ()					0					
> M GPIO_BUTTONS_IBUF[1:0]	D				0					
> M GPIO_BUTTONS_re[1:0] () > M U_SINEGEN/sel[1:0] (3)					0					
> = U_SINEGEN/sine[19:0]	-183094									
	103094									
		Updated at: 20	017-Mar-16	15:08:22						~
		`	_	r						
Settings - hw_ila_1 × Status - hw_ila_	_1		? _ □		ıp - hw_ila_1 ×	Capture Set	up - hw_ila	_1		? _ 🗆
Trigger Mode Settings			î	Q +	- Þ.					
Trigger mode: BASIC_ONLY	~			Name		Operator	Radix	Value		Port
				GPIO_BUTT	ONS_IBUF[1:0]	•	(B)	▼ XX	•	probe3[1:0]
Capture Mode Settings										
Capture mode: ALWAYS										
Number of windows: 1	[1 - 1024]									
Window data depth: 1024	✓ [1 - 1024]									
Trigger position in window: 512	[0 - 1023]									
General Settings										
Refresh rate: 500 ms										
			~	<						> >
Trigger Setup - hw_ila_1 ×	Capture Setu	.ip - hw_ila_1			? _ □	1				
Q + - D										
Name	Operator	Radix	Value		Port					
GPIO_BUTTONS_IBUF[1:0]	== *	[B] •	RX	~	probe3[1:0]					
				]						
		Value:	RX							
		ОК	0	Cancel						
<						>				

**CAUTION!** For different labs the GPIO\_BUTTONS\_IBUF might show up differently or have a different name such as button\_in4\_in. This might also show up as two individual bits or two bits lumped together in a bus. Ensure that you are using bit 1 of this bus to set up your trigger condition. For example in case of a two-bit bus, you will set the Value field in the Compare Value dialog box to RX.



2. Select Enable Auto Re-trigger mode on the ILA debug core as shown below.

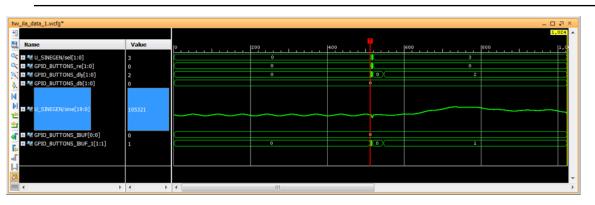


CAUTION! The ILA properties window may look slightly different for different labs.

When you issue a Run Trigger or a Run Trigger Immediate command after setting the Auto Retrigger mode, the ILA core does the following repetitively until you disable the Auto Retrigger mode option:

- Arms the trigger.
- Waits for the trigger.
- Uploads and displays waveforms.
- 3. On the KC705 board, press the Sine Wave Sequencer button until you see multiple transitions on the GPIO\_BUTTONS\_IBUF\_1 signal (this could take 10 or more tries). This is a visualization of the glitch that occurs on the input. An example of the glitch is shown in the following two figures.

**CAUTION!** You may have to repeat the previous two steps repeatedly to see the glitch. After you can see the glitch, you may observe that the signal glitches are not at exactly the same location as shown in the figure below.







nw_ila_data_1.wcfg*											-	50
Name	Value		\$00	505	510		515		520	525	530	535
U_SINEGEN/sel[1:0]	3			0			( i	<u>x</u> z <u>x</u>		3		
GPIO_BUTTONS_re[1:0]	0			0				02		0		
H GPIO_BUTTONS_dly[1:0]	2			0		20	20			0		
GPIO_BUTTONS_db[1:0]	0							0				
U_SINEGEN/sine[19:0]	105321											
1												
H GPIO_BUTTONS_IBUF[0:0]	0							0				
■ ■ GPIO_BUTTONS_IBUF_1[1:1] 	1			0		101	0 1	х		0		
l												
( )	4	- F	(									

#### Fixing the Signal Glitch and Verifying the Correct State Machine Behavior

The multiple transition glitch or "bounce" occurs because the mechanical button is making and breaking electrical contact just as you press it. To eliminate this signal bounce, a "de-bouncer" circuit is required.

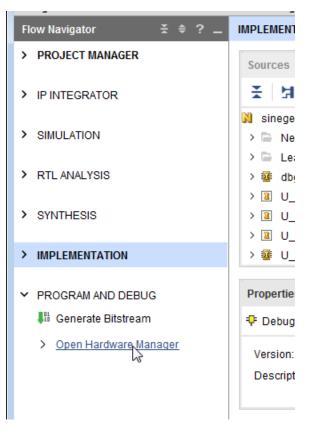
- 1. Enable the de-bouncer circuit by setting DIP switch position on the KC705 board (labeled De-bounce Enable in Figure 1) to the ON or UP position.
- 2. Enable the Auto-Retrigger mode on the ILA debug core and click RunTrigger on the ILA core, and
  - Ensure that you no longer see multiple transitions on the GPIO\_BUTTON\_re[1] signal on a single press of the Sine Wave Sequencer button.
  - Verify that the state machine is working correctly by ensuring that the sineSel signal transitions from 00 to 01 to 10 to 11 and back to 00 with each successive button press.

## Verifying the VIO Core Activity (Only Applicable to Lab 3)

1. From the Program and Debug section in Flow Navigator, click **Open Hardware Manager**.



#### AMD**7** XILINX



The Hardware Manager window opens.

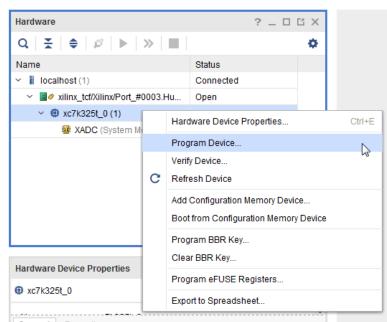
2. Click Open a new hardware target.

HARDWARE MANAGER - unconnected	ed		
🚯 No hardware target is open. Ope	en targ	get	
Hardware	ø	Auto Connect	
		Recent Targets	<b>•</b>
		Available Targets on Server	
		Open New Target	
			N
No conten	t		
	-		

- 3. The Open New Hardware Target wizard opens. Click Next.
- 4. In the Hardware Server Settings page, type the name of the server (or select **Local server** if the target is on the local machine) in the Connect to field.



- 5. Ensure that you are connected to the right target by selecting the target from the Hardware Targets page. If there is only one target, that target is selected by default. Click **Next**.
- 6. In the Set Hardware Target Properties page, click Next.
- 7. In the Open Hardware Target Summary page, verify that all the information is correct, and click **Finish**.
- 8. Program the device by selecting and right-clicking the device in the Sources window and then selecting **Program Device**.



9. In the Program Device dialog box, ensure that the bit file to be programmed is correct. Click **OK**.

🍌 Program Device		×
	ramming file and download it to your hardware device. You can optionally select a debug probes file debug cores contained in the bitstream programming file.	•
Bitstre <u>a</u> m file: Debu <u>q</u> probes file:	C://ivado_Debug/2017.1/proj_hdl_vio/proj_hdl_vio.runs/impl_1/sinegen_demo_inst_vio.bit C://ivado_Debug/2017.1/proj_hdl_vio/proj_hdl_vio.runs/impl_1/sinegen_demo_inst_vio.ltx	]
✓ Enable end of st	artup check	
?	<u>P</u> rogram Cancel	

10. After the FPGA device is programmed, you see the VIO and the ILA core in the Hardware window.



Hardware	? _ 🗆	с ×
$Q_{1}\mid \underbrace{\texttt{A}}_{1}\mid \diamondsuit \mid \varnothing \mid \mathrel{\blacktriangleright} \mid \gg \mid \blacksquare \mid$		•
Name	Status	
<ul> <li>Iocalhost (1)</li> </ul>	Connected	
✓ Ø xilinx_tcf/Xilinx/Port_#0003.Hu	Open	
xc7k325t_0 (3)	Programmed	
🔯 XADC (System Monitor)		
🦉 hw_ila_1 (U_ILA)	Oldle	
🦉 hw_vio_1 (U_VIO)	OK - Outputs Reset	

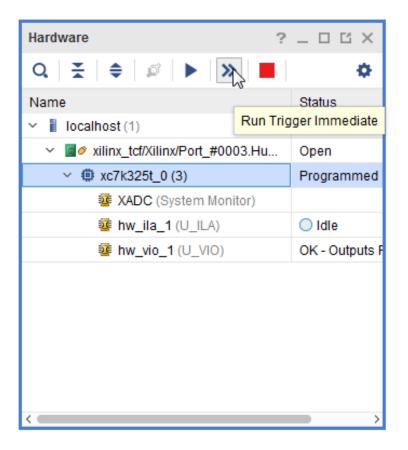
You now have a debug dashboard for the ILA core as shown in the following figure.

hw.	ia_1		? 🗆 🖒 🗙
	Waveform - hw_ila_1		? _ 🗆 ×
ions	Q   +   −   ♂   ▶   ≫   ■   ⊡   @   Q   X   •               ±   ±   +		0
Dashboard Options	ILA Status: Idle		^
shboa		20	
Das	> M GPIO_BUTTONS_dM(10) > M GPIO_BUTTONS_dM(10) M GPIO_BUTTONS_re_1(10) ↓ push_button_vest_1 ↓ push_button_vio > M sime(19:0) > M sineSel(10) 		~
	Settings - hw_ila_1 Status - hw_ila_1 × ? _ D	Trigger Setup - hw_ila_1 × Capture Setup - hw_ila_1	? _ 🗆
	년 🕨 🗶 📕 무급	$ c  +   =  b_{\lambda} $	
	Core status           Idle         Pre-Trigger         Waiting for Trigger         Post-Trigger         Full           Capture status         Window 1 of 1         Window sample 0 of 1024         Total sample 0 of 1024         Idle	Press the 🔸 button to add probes.	

11. Click Run Trigger Immediate to capture the data immediately.



#### AMD7 XILINX



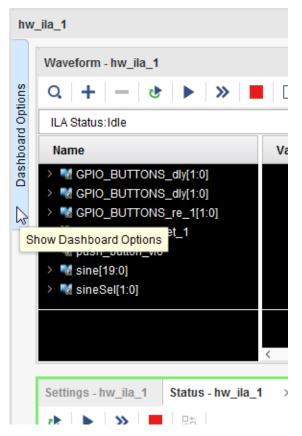
- 12. Make sure that there is activity on the sine [19:0] signal.
- 13. Select the sine signal in the Waveform window, right-click and select Waveform Style  $\rightarrow$  Analog.
- 14. Select the sine signal in the Waveform window again, right-click and select **Radix** → **Signed Decimal**. You should be able to see the sine wave in the Waveform window.







- 15. Instead of using the GPIO\_SW push button to cycle through each different sine wave output frequency, you are going to use the virtual "push\_button\_vio" toggle switch from the VIO core.
- 16. You can now customize the ILA dashboard options to include the VIO window. This allows you to toggle the VIO output drivers and observe the impact on the ILA waveform window all in one dashboard. Slide out the Dashboard Options window.



17. Add the VIO window to the ILA dashboard by selecting hw\_vio\_1.





hw_ila_1											
Dashboard Options _	Waveform - hw_ila_1										
Q ≚ ≑	Q   +   −   &   ▶   ≫	🕞 🔍	Q.   ;	¢   +[	I	) I	t i tr	+	F#   +F   -F	,	
✓ ■ xc7k325t_0	ILA Status: Idle										
✓ ✓ hw_ila_1 (U_ILA)	Name	Value	0			200			400	T	600
<ul> <li>✓ Status</li> <li>✓ Settings</li> <li>✓ Trigger Setup</li> <li>✓ Capture Setup</li> <li>✓ Wavaform</li> <li>✓ hw_vio_1 (U_VIO)</li> <li>✓ XADC (system Monitor)</li> </ul>	<pre>&gt; ** GPIO_BUTTONS_div[1:0] &gt; ** GPIO_BUTTONS_div[1:0] &gt; ** GPIO_BUTTONS_re_1[1:0]  ** grush_button_reset_1  ** push_button_vio &gt; ** sine[19:0] &gt; ** sineSet[1:0]</pre>	0 0 0 0 -23169 0									
		<	Upda > <	ted at:	2017-№	ar-17 1	11:25:1	19			
	Settings - hw_ila_1 Status - hw_ila_	1 × ? _		Trigger S	Setup - hv	v_ila_1	hw_	vio_1	× Capture S	etup - hw_il	la_1
	🕑 🕨 🔉 📕 🚓			Q	\$	+	-				
			r Triç Tota						Press the 🕂	button to	add probes.
	<		>								

Note: The ILA dashboard now contains the VIO window as well.

18. Adjust the Trigger Setup – hw\_ila\_1 window and the hw\_vio\_1 window so that they are side by side as shown in the following figure.





Waveform - hw_ila_1							? _ 🗆 X		
Q + - ♂ ► ≫	<b>D</b>	Q 23 • I4	N 1± ± 1+	Fe   +F   F	+		0		
ILA Status: Idle							1,023		
Name	Value	0	200	400	T	600	800 1.,0		
> V GPIO_BUTTONS_dly[1:0]	0				0				
> Variable GPIO_BUTTONS_dly[1:0] > Variable GPIO_BUTTONS_re_1[1:0]	0 0				1				
<sup>10</sup> push_button_reset_1	0								
₩ push_button_vio	0	0 0 0		0 0					
			$  \land \land \land $	$   \wedge    \wedge$	$\Lambda$	$\Lambda \Lambda \Lambda$			
> 🍣 sine[19:0]	-23169						$/ \land / \land / \land / \land / \land$		
		$\downarrow \downarrow \downarrow \downarrow$	$\mathbb{V} \setminus \mathbb{V}$	V = V	V V				
> 📲 sineSel[1:0]	0				0				
	<	Updated at: 2017-M	far-17 11:25:19						
Settings - hw_ila_1 Status - hw_ila_	_1 Ingger Se	tup-hw × ? _ □		oture Setup - hv	v_iia_1		? _ 🗆		
Q   <b>+</b>   −   ⊅ <sub>4</sub>			0,   素   ≑   •						
Press the 🕂 b	utton to add prob	a.c.	Press the 🕂 button to add probes.						
11035 410	attor to add prob								

19. In the hw\_vio\_1 window, select the "+" button, and select all the probes under hw\_vio\_1.

#### 20. Click **OK**.

*Note*: The initial values of all the probes.





ຊ   <b>+</b>   <b>−</b>   ອ   ▶   ≫															
ILA Status: Idle															
Name	Value		0			2	200			400			600		.  800
GPIO_BUTTONS_dly[1:0]	0											)			
Version Superson Street	0 0		1)									)			
line grid_borroids_re_r[r.o]	0											,			
₩ push_button_vio	0														
₩ sine[19:0]	-23169		$\int$		$\mathbb{A}$		$\bigwedge$	$\bigwedge$	$\bigwedge$	$\bigwedge$	$\bigwedge$	$\bigwedge$	$\bigwedge$	$\mathbb{N}$	$\mathbb{N}$
™ sineSel[1:0]	0											)			
	<	>	Upda	ated a	t: 2017 <sup>.</sup>	-Mar·	-17 11:	25:19							
	<a>a_1 Trigg</a>		> <		t: 2017- ? 0		-17 11: hw_vio_	_1 ×	Captu	ıre Setu	ıp - hw	_ila_1			
	< a_1 Trigg		> <				hw_vio_	_1 ×			ıp - hw <u>.</u>		Add Probes		
$ \mathbf{a}  +   -  \mathbf{b} $		jer Setu	> < ıp - hw				hw_vio_	_1 ×					Add Probes		
		jer Setu	> < ıp - hw				hw_vio_	_1 ×		-			Add Probes		
$ \mathbf{a}  +   -  \mathbf{b} $		jer Setu	> < ıp - hw				hw_vio_	_1 ×			¢ ch: C				
$ \mathbf{a}  +   -  \mathbf{b} $		jer Setu	> < ıp - hw				hw_vio_	_1 ×		 ∑ear Prot	♦ ch: □ bes for hw_v	l∽ hw_vio_1 ( io_1	(5)		
$ \mathbf{a}  +   -  \mathbf{b} $		jer Setu	> < ıp - hw				hw_vio_	_1 ×		 ∑ear Prot	♦ ch: O bes for  hw_v hw_v	ly hw_vio_1 ( io_1 DONT_EAT			
ettings - hw_ila_1 Status - hw_ila $Q + = D_1$ Press the +		jer Setu	> < ıp - hw				hw_vio_	_1 ×		 ∑ear Prot	♦ rch: □ Des for I hw_v • [ <p< td=""><td>لب hw_vio_1 ا io_1 DONT_EAT SPIO_BUT</td><td>5) TONS_re[1:</td><td></td><td></td></p<>	لب hw_vio_1 ا io_1 DONT_EAT SPIO_BUT	5) TONS_re[1:		
$ \mathbf{a}  +   -  \mathbf{b} $		jer Setu	> < ıp - hw				hw_vio_	_1 ×		 ∑ear Prot	♦ ch: C bes for hw_v № C % C % C	ly hw_vio_1 ( io_1 DONT_EAT	(5) • • TONS_re[1: n_reset	1]	<
$ \mathbf{a}  +   -   \mathbf{b}_{\mathbf{a}} $		jer Setu	> < ıp - hw				hw_vio_	_1 ×		∑ Sear Prot ~ 33	♦ ch: O bes for hw_v bes for hw_v 0 10	k- hw_vio_1 f io_1 DONT_EAT SPIO_BUT uush_butto	(5) • • TONS_re[1: n_reset n_vio_1		<

21. Note the values on all probes in the hw\_vio\_1 window.





Waveform - hw_ila_1							? _ 🗆 :
Q   <b>+</b>   <b>−</b>   ♂   ▶   ≫	📕 📴 🔍	Q   X   <b>-f</b>   <b>H</b>   <b>→</b>	l   12   2r   +F   Fe	.   al'   lal   -			
ILA Status:Idle							1,023
Name	Value	•	200 4	100 T	600	800	·
🛛 💐 GPIO_BUTTONS_dly[1:0]	0						
GPIO_BUTTONS_dly[1:0]	0	{					
GPIO_BUTTONS_re_1[1:0]	0	(					P
⅓ push_button_reset_1 ⅓ push_button_vio	0 0						
> <b>™</b> sine[19:0]	-23169						
sineSel[1:0]	0						
	<	Updated at: 2017-Ma	r-17 11:25:19				
iettings - hw_ila_1 Status - hw_ila	_1 Trigger Set	tup-hw × ? _ 🗆	hw_vio_1 × Captur	e Setup - hw_ila_1			? _ [
0   <b>+</b>   =   <b>D</b> _			Q   ₹   ≑   +				
			Name	Value	Acti Directi.	VIO	
			I DONT_EAT	[B] 0	Input	hw_vio_1	
			I GPIO_BUTTONS_	re[1:1] [B] 0	Input	hw_vio_1	
Dense the start	outton to add probe		🛥 push_button_rese	et [B] 0 •	• Output	hw_vio_1	
Pressine 🕇 t	outton to add probe	-5.	∿= push_button_vio_	1 [B] 0 •	• Output	hw_vio_1	
			> 🍓 sineSel_1[1:0]	[H] 0	Input	hw_vio_1	

22. Set the push\_button\_reset output probe by right-clicking **push\_button\_reset** and select **Toggle Button**.

This will toggle the output driver from logic from 0 to 1 to 0 as you click. It is similar to the actual push button behavior, though there is no bouncing mechanical effect as with a real push button switch.





hw_vio_1 × Capture Setu	o - hw_ila_1				? _ 🗆
Q   풒   <b>≑</b>   <b>+</b>   <b>-</b>					
Name	Value	Activity	Direction	VIO	
🕒 DONT_EAT	[B] 0		Input	hw_vio_1	
৳ GPIO_BUTTONS_re[1:1]	[B] 0		Input	hw_vio_1	
∿⊲ push_button_rese*			A	hw_vio_1	
∿a push_button_vio_	Debug Probe	Properties	CtrI+E	hw_vio_1	
> 墙 sineSel_1[1:0]	Text			hw_vio_1	
	Active-High B	utton			
	Active-Low B	utton			
	Toggle Butto	n	L.		
	Radix		6		? _ 🗆 🗆
	Rename				
	Name		+		^
	Remove		Delete		
and 'import_hw_ila_da	Export to Spr	eadsheet		Data menu	item to impo

The Value field for push\_button\_reset is highlighted.

23. Click in the **Value** field to change its value to 1.

Q       Xame       Value       Activity       Direction       VIO         Image: Dont_EAT       [B] 0       Input       hw_vio_1         Image: GPIO_BUTTONS_re[1:1]       [B] 0       Input       hw_vio_1         Image: GPIO_BUTTONS_re[1:1]       [B] 0       Input       hw_vio_1         Image: GPIO_BUTTONS_relocation       1       Output       hw_vio_1         Image: GPIO_BUTTONS_relocation       0       Output       hw_vio_1         Image: GPIO_BUTTONS_relocation       0       Image: GPIO_BUTTONS_relocation       Image: GPIO_BUTTONS_relocation         Image: GPIO_BUTTONS_relocation       1       Output       hw_vio_1         Image: GPIO_BUTTONS_relocation       0       Image: GPIO_BUTTONS_relocation         Image: GPIO_BUTTON_FORTOR       0       Image: GPIO_BUTTONS_relocation	hw_vio_1 × Capture Setup	- hw_ila_1				? _ □
Image: Second	Q   ¥   ♦   +   -					
Image: GPIO_BUTTONS_re[1:1]     [B] 0     Input     hw_vio_1       Image: GPIO_BUtton_reset     1     Output     hw_vio_1       Image: GPIO_Button_vio_1     0     Output     hw_vio_1	Name	Value	Activity	Direction	VIO	
La push_button_reset     1     Output     hw_vio_1       La push_button_vio_1     0     Output     hw_vio_1	∿ DONT_EAT	[B] 0		Input	hw_vio_1	
ta push_button_vio_1 0 Output hw_vio_1	BOPIO_BUTTONS_re[1:1]	[B] 0		Input	hw_vio_1	
	ush_button_reset	1		Output	hw_vio_1	
S in sine Cal 4[4:0] [11] 0 [anut ] but via 4	∿a push_button_vio_1	0		Output	hw_vio_1	
resinesei_i(i.oj [H]o input nw_vio_i	> 🐌 sineSel_1[1:0]	[H] 0		Input	hw_vio_1	

- 24. Follow the step above to change the push\_button\_vio to Toggle button as well.
- 25. Set these two bits of the "sineSel" input probe by right-clicking **PROBE\_IN0[0]** and **PROBE\_IN0[1]** and selecting **LED**.

#### AMD7 XILINX

hw_vio_1 × Capture Setur	o - hw_ila_1				
$Q \mid \Xi \mid \clubsuit \mid + \mid - \mid$					
Name	Value	Activity	Direction	VIO	
∿ DONT_EAT	[B] 0		Input	hw_vio_1	
BOPIO_BUTTONS_re[1:1]	[B] 0		Input	hw_vio_1	
∿ push_button_reset	1		Output	hw_vio_1	
∿ push_button_vio_1	0		Output	hw_vio_1	
> 🐚 sineSel_1[1:0]	[H] 0		Input	hw_vio_1	
	Debug P	robe Propertie	es Ctrl+E		
· · ·	Text				
	LED				
	Radix		•		
	Activity Pe	ersistence	Þ		
U VIO"}]]	Rename				
0_010 }11	Name		Þ		
	Remove		Delete	9	
	Export to	Spreadsheet			

26. In the Select LED Colors dialog box, pick the **Low Value Color** and the High Value Color of the LEDs as you desire and click **OK**.

🔥 Select LED Cold	ors 🔀
Low Value Color:	Gray 🗸
<u>H</u> igh Value Color:	\varTheta Red 🗸 🗸
ОК	Cancel

27. When finished, your VIO Probes window in the Hardware Manager should look similar to the following figure.



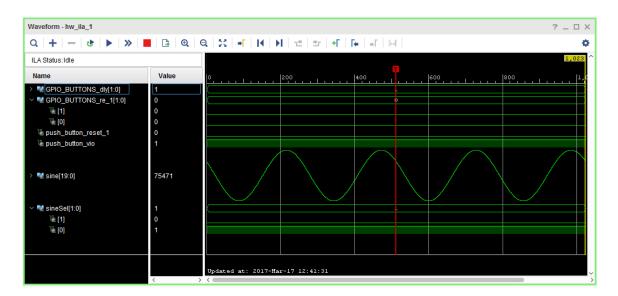
hw_vio_1 × Capture Setup	- hw_ila_1				? _ 🗆
Q   ¥   €   +   -					
Name	Value	Activity	Direction	VIO	
∿ DONT_EAT	[B] 0		Input	hw_vio_1	
BOPIO_BUTTONS_re[1:1]	[B] 0		Input	hw_vio_1	
ush_button_reset	1		Output	hw_vio_1	
∿ push_button_vio_1	0		Output	hw_vio_1	
> 🐌 sineSel_1[1:0]	[H] 0		Input	hw_vio_1	

- 28. To cycle through each different sine wave output frequency using the virtual "push\_button\_vio" from the VIO core, perform the following simple steps:
  - a. Toggle the value of the "push\_button\_vio" output driver from 0 to 1 to 0 by clicking on the logic displayed under the Value column. You will notice the sineSel LEDs changed accordingly 0, 1, 2, 3, 0, etc.

hw_vio_1 × Capture Setup	- hw_ila_1				? _
Q   ¥   ♦   +   -					
Name	Value	Activity	Direction	VIO	
Ъ DONT_EAT	[B] 0		Input	hw_vio_1	
BUTTONS_re[1:1]	[B] 0		Input	hw_vio_1	
∿ push_button_reset	0		Output	hw_vio_1	
✓ iie sineSel_1[1:0]	[H] 1		Input	hw_vio_1	
	٢		Input	hw_vio_1	
	•		Input	hw_vio_1	
∿a push_button_vio_1	1		Output	hw_vio_1	

b. Click **Run Trigger** for hw\_ila\_1 to capture and display the selected sine wave signal from the previous step.









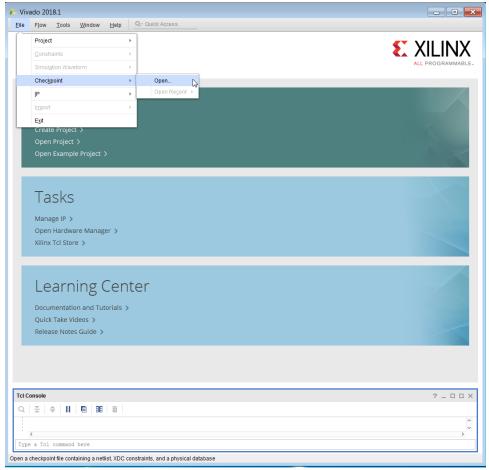
#### Lab 6

## Using the ECO Flow to Replace Debug Probes Post Implementation

This simple tutorial shows you how to replace nets connected to an ILA core in a placed and routed design checkpoint using the Vivado<sup>®</sup> Design Suite Engineering Change Order (ECO) flow.

**Note:** To learn more about using the ECO flow, refer to the *Debugging Designs Post Implementation* chapter in the Vivado Design Suite User Guide: Programming and Debugging (UG908).

1. Open the Vivado<sup>®</sup> Design Suite, and select File  $\rightarrow$  Open Checkpoint.



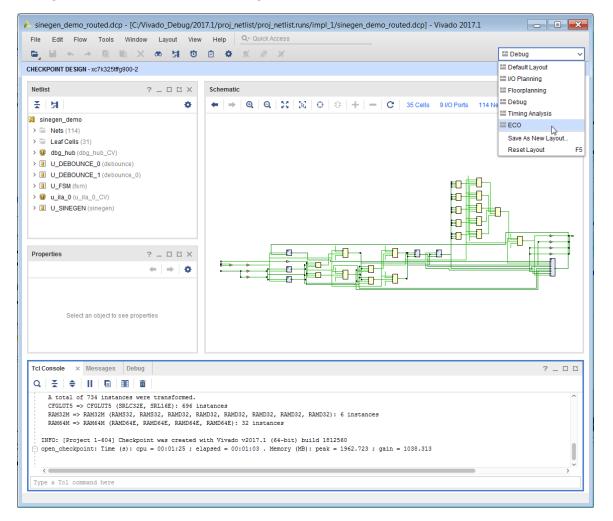
2. Open the routed checkpoint that you created in Lab 2: Using the HDL Instantiation Method to Debug a Design.





🔥 Open Che	eckpoint	
Look <u>i</u> n: 🕠	impl_1	- 😥 😒 🗙 🎸 🐒 🕲 🗞
<ul> <li>Xil</li> <li>sinegen_demo.dcp</li> <li>sinegen_demo_opt.dcp</li> <li>sinegen_demo_placed.dcp</li> <li>sinegen_demo_routed.dcp</li> </ul>		Recent Directories         C://ivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1         File Preview         File: sinegen_demo_routed.dcp         Directory: C://ivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1
		Created: Wednesday 03/15/17 02:54 PM Accessed: Wednesday 03/15/17 02:54 PM Modified: Wednesday 03/15/17 02:54 PM Size: 4.5 MB Type: Checkpoint design Owner: XLNX\smitha
File <u>n</u> ame: Files of type:	sinegen_demo_routed.dcp Vivado Checkpoint Files (.dcp)	~
		OK Cancel

Change the layout in the Vivado Design Suite toolbar drop-down to ECO.







Note: The Flow Navigator window now changes to ECO Navigator with a different set of options.

	[C:/Vivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1], "inegen_demo_routed.dcp] - Vivado 2017.1
ECKPOINT DESIGN - xc7k325tffg90	
ECKPOINT DESIGN - XC7K325uig90	U-2
ECO Navigator	Scratch Propertie Netti X ? _ C C Schematic X Device X Package X ? C
Edit	★       ★       ◆       ●       Q       X       N       ●       +       -       C       35 Cells       9 I/O Ports
Create Net	N sinegen_demo
Create Cell	> ≧ Nets (114) > ≧ Leaf Cells (31)
Create Port	> <b>2 dbg_hub</b> (dbg_hub_CV)
Create Pin	> U_DEBOUNCE_0 (debounce)
Connect Net	U_DEBOUNCE_1(debounce_0)     U_FSM (fsm)
Disconnect Net	> ﷺ u_ila_0 (u_ila_0_CV)
Replace Debug Probes	I U_SINEGEN (sinegen)
Place Cell	
Unplace Cell	
Run	
Check ECO	
Optimize Logical Design	
Place Design	
Optimize Physical Design	
Route Design	
Report	
Edit Timing Constraints	
Ӧ Report Timing Summary	
A Report Clock Networks	
Report Clock Interaction	
Report DRC	
Report Utilization	Tcl Console × Messages Package Pins VO Ports ? _ □
≸ Report Power	Q     ★     II     Image: Second state       INFO: [Project 1-111] Unisim Transformation Summary:
	A total of 734 instances were transformed.
Program	CFGLUT5 => CFGLUT5 (SRLC32E, SRL16E): 696 instances RAM32M => RAM32M (RAMS32, RAMS32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32): 6 instances
Save Checkpoint As	RAM64M => RAM64M (RAMD64E, RAMD64E, RAMD64E, RAMD64E): 32 instances
Senerate Bitstream	INFO: [Project 1-604] Checkpoint was created with Vivado v2017.1 (64-bit) build 1812560
🕒 Write Debug Probes	Gen_incopping. Time (a). Ga = 00.01.25 , elapsed = 00.01.05 . Hemoly (hb). peak = 1902.725 , gain = 1000.015
Open Hardware Manager	Type a Tcl command here

3. In the ECO Navigator window, click **Replace Debug Probes** to bring up the Replace Debug Probes dialog box. Note the Debug Hub and ILA cores in the design.





Replace Debug Probes	<b>×</b>
Use the Edit Probes button to replace one o changes in the Vivado Hardware Manager, r	
Search: Q-	
Name	Probe
Ch 13	「 ¥ U_SINEGEN/sine[13]
• Ch 14	「 ¥ U_SINEGEN/sine[14] ⊘
Och 15	「¥U_SINEGEN/sine[15] ⊘
Och 16	_「 ¥ U_SINEGEN/sine[16]
Och 17	_「 ¥ U_SINEGEN/sine[17]
Ch 18	「 ¥ U_SINEGEN/sine[18] ∥
Ch 19	「 ¥ U_SINEGEN/sine[19]
✓ ■ probe2 (2)	
Ch 0	_「 * GPIO_BUTTONS_IBUF[0]
🖲 Ch 1	_「 * GPIO_BUTTONS_IBUF[1]
probe3 (2)	
Ch 0	_ SPIO_BUTTONS_db[0]
Och 1	_ ★ GPIO_BUTTONS_db[1] 🖉
✓	
Och 0	_ SPIO_BUTTONS_dly[0]
• Ch 1	_ SPIO_BUTTONS_dly[1]
✓	
Ch 0	_ SPIO_BUTTONS_re[0]
Probes changed: 0	
	OK Cancel

**IMPORTANT!** Xilinx strongly recommends that you do not replace the clock nets associated with ILA and Debug Hub cores.

- 4. In the Replace Debug Probes dialog box, highlight the probes whose nets you want to change. In this lab you will replace the GPIO\_BUTTONS\_dly[0] net that is being probed.
- 5. Click the **Edit Probes** button to the right of the GPIO\_BUTTONS\_dly[0] probe net to bring up the Choose Nets dialog box.



🍐 Replace Debug Probes	<b>×</b>
Use the Edit Probes button to replace one of changes in the Vivado Hardware Manager,	
Search: Q-	
Name	Probe
• Ch 13	_ ¥ U_SINEGEN/sine[13]
• Ch 14	_ SINEGEN/sine[14] ⊘
Ch 15	_ ¥ U_SINEGEN/sine[15] ∥
Ch 16	_「∗ U_SINEGEN/sine[16] //
Ch 17	「∗U_SINEGEN/sine[17] //
Ch 18	「∗ U_SINEGEN/sine[18] /
Ch 19	_ SINEGEN/sine[19]
✓	
Ch 0	「★ GPIO_BUTTONS_IBUF[0]
• Ch 1	_ SPIO_BUTTONS_IBUF[1]  ⊘
probe3 (2)	
Och 0	_ SPIO_BUTTONS_db[0]  ⊘
• Ch 1	_ SPIO_BUTTONS_db[1]
✓ ➡ probe4 (2)	-
• Ch 0	GPIO_BUTTONS_dly[0]
Ch 1	「∗ GPIO_BUTTONS_dly[1]
✓ Improbe5 (2)	Example a second second
Ch 0	「* GPIO_BUTTONS_re[0] ⊘ ∨
Probes changed: 0	
	OK Cancel

6. In the Choose Nets dialog box, choose the U\_DEBOUNCE\_0/clear net to replace the existing GPIO\_BUTTONS\_dly[0] probe net. Click **OK**.





Choose Nets				×
Choose nets to replace existing probes.				4
Properties				
NAME v contains	~	*	⊗ +	
Regular expression Search hierarchically	/ ✓ <u>D</u> ispl	ay uni	que nets	
Of objects:				
		Ein		
Found: 12857	A		Selected: 0 of 1	(
_∫ <const0></const0>				
∫ <const1></const1>				
∫ clk_ibufgds				×
		+		Ŧ
J CLK_P		⇒	Use the buttons on the left to copy Nets into this List.	+
∫ dbg_hub/ <const0></const0>		-		+
∫ dbg_hub/inst/ <const0></const0>				÷
∫ dbg_hub/inst/BSCANID.u_xsdbm_id/ <const1< td=""><td></td><td></td><td></td><td>*</td></const1<>				*
_f dbg_hub/inst/BSCANID.u_xsdbm_id/bscanid				
J dbg_hub/inst/BSCANID.u_xsdbm_id/bscanid	[1] >			
	,			
			OK Can	cel

 Type for "\*clear net" in the Name field and Click Find. Notice the U\_DEBOUNCE\_0 net in the Found nets area. Select U\_DEBOUNCE\_0/clear net using the "->" arrow and click OK. The U\_DEBOUNCE\_0/clear net to replaces the existing GPIO\_BUTTONS\_dly[0] probe net.





🍐 Choose Nets				×
Choose nets to replace existing prob	es.			4
Properties				
NAME 🗸	contains ~	*cle	ar 🛛 🕹 🕇	
	hiararchically 📿 Dian	ov unio	ue note	
Of objects:	nierarchically 🕑 <u>D</u> ispi	ay unic	lue nets	<b></b>
Found: 68		Ein		71
Found, 68			Selected: 0 of 1	Ă↓
_ U_DEBOUNCE_0/clear				
_ u_ila_0/inst/ila_core_inst/u_ila_	regs/CNT.CNT_SRL			
」 u_ila_0/inst/ila_core_inst/u_ila_				×
∫ u_ila_0/inst/ila_core_inst/u_ila_		+		Ŧ
∫ u_ila_0/inst/ila_core_inst/u_ila_		⇒	Use the buttons on the left to copy Nets into this List	÷
J u_ila_0/inst/ila_core_inst/u_ila_     S u_ila_0/inst/ila_core_inst/u_ila_		-		+
∫ u_ila_0/inst/ila_core_inst/u_ila_				+
∫ u_ila_0/inst/ila_core_inst/u_ila_				Ť
∫ u_ila_0/inst/ila_core_inst/u_ila_ ∫ u_ila_0/inst/ila_core_inst/u_ila_				
	>			
			ОК	ancel





Choose nets to replace existing probes.	r	Cho	oose Nets					×
NAME       contains       'dear         MAME       contains       'dear         Image: Contains       'dear       +         Image: Contains       'dear       +         Image: Contains       'dear       +         Image: Contains       'dear       +         Image: Contains       Image: Contains       +       +         Image: Contains       Image: Contains       +       +         Image: Contains       Image: Contains       +       Image: Contains       +         Image: Contains       Image: Contains       -       +       Image: Contains       +         Image: Contains       Image: Contains       -       -       +       Image: Contains       +         Image: Contains       Image: Contains       -       -       -       -       -       -       -       -	С	hoo	se nets to replace existing pr	obes.				4
Begular expression       Search hierarchically       Display unique nets         Of objects:          Found: 68          I ubg/nubl/inst/BSCANID.u_xsdbm_id/CORE_XSDBU          J ubg/nubl/inst/BSCANID.u_xsdbm_id/CORE_XSDBU          I ubg/nubl/inst/BSCANID.u_xsdbm_id/CORE_XSDBU          J ubg/nubl/inst/Bacore_inst/u_ila_regs/CNT.CNT_SRL          J uia_0inst/ila_core_inst/u_ila_regs/NU_SRL[0].mu          J uia_0inst/ila_core_inst/u_ila_regs/NU_SRL[0].mu		Pro	perties					
Of objects:          Found: 68       Eind         J dbg_hub/inst/BSCANID.u_xsdbm_id/CORE_XSDB.U       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu       Copy selected Nets into the Selection List         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu       L         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu       L         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu       L			NAME ~	contains	~	*clear	8	F I
Of objects:          Found: 68       Eind         J dbg_hub/inst/BSCANID.u_xsdbm_id/CORE_XSDB.U       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu       Copy selected Nets into the Selection List         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu       L         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu       L         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu       L								
Of objects:          Found: 68       Eind         J dbg_hub/inst/BSCANID.u_xsdbm_id/CORE_XSDB.U       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu       Copy selected Nets into the Selection List         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu       L         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu       L         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu       L								
Of objects:          Found: 68       Eind         J dbg_hub/inst/BSCANID.u_xsdbm_id/CORE_XSDB.U       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu       Copy selected Nets into the Selection List         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu       L         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu       L         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu       L								
Of objects:          Found: 68       Eind         J dbg_hub/inst/BSCANID.u_xsdbm_id/CORE_XSDB.U       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu       Copy selected Nets into the Selection List         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu       L         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu       L         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu       L								
Of objects:          Found: 68       Eind         J dbg_hub/inst/BSCANID.u_xsdbm_id/CORE_XSDB.U       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu       J         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu       Copy selected Nets into the Selection List         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu       L         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu       L         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu       L			Regular expression 📝 Sear	ch hierarchically 📝 Disr	nla	v unique nets		
Found: 68       Selected: 1 of 1       Image: the selection construction of t								
I dbg_hub/inst/BSCANID.u_xsdbm_id/CORE_XSDB.U         J U_DEBOUNCE_0/clear         I u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[1].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu						<u>F</u> ind		
J U_DEBOUNCE_0/clear         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[1].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu		Four	nd: 68			Selected: 1 of 1	A.	↓ <mark>Z</mark> ↓
J U_DEBOUNCE_0/clear         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL         J u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[1].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[3].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu		Ъ	lbg_hub/inst/BSCANID.u_xsc	lbm_id/CORE_XSDB.U		LU DEBOUNCE 0/dear		
J       u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL         J       u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL         J       u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL         J       u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu         J       u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[1].mu         J       u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J       u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J       u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[3].mu         J       u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu         V       V		зu	J_DEBOUNCE_0/clear					
<ul> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL</li> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu</li> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[1].mu</li> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu</li> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu</li> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu</li> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu</li> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu</li> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu</li> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu</li> <liu_ila_0 ila_core_inst="" inst="" mu_sr<="" td="" u_ila_regs=""><td></td><td>Лu</td><td>ı_ila_0/inst/ila_core_inst/u_il</td><td>a_regs/CNT.CNT_SRL</td><td></td><td></td><td></td><td></td></liu_ila_0></ul>		Лu	ı_ila_0/inst/ila_core_inst/u_il	a_regs/CNT.CNT_SRL				
<ul> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL</li> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu</li> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[1].mu</li> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu</li> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[3].mu</li> <li>u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu</li> <liu_ila_0 ila_core_inst="" inst="" mu_sr<="" td="" u_ila_regs=""><td></td><td>Лu</td><td>ı_ila_0/inst/ila_core_inst/u_il</td><td>a_regs/CNT.CNT_SRL</td><td></td><td></td><td></td><td>×</td></liu_ila_0></ul>		Лu	ı_ila_0/inst/ila_core_inst/u_il	a_regs/CNT.CNT_SRL				×
J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[1].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu         J u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu		Лu	ı_ila_0/inst/ila_core_inst/u_il	a_regs/CNT.CNT_SRL	L	_		-
		Лu	ı_ila_0/inst/ila_core_inst/u_il	a_regs/CNT.CNT_SRL	Ŀ			
L u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu L u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[3].mu L u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu C		Лu	ı_ila_0/inst/ila_core_inst/u_il	a_regs/MU_SRL[0].mu				т
└ u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu └ u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu <		Лu	ı_ila_0/inst/ila_core_inst/u_il	a_regs/MU_SRL[1].mu		Copy selected Nets into the Selection List		+
J     u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu.        >		Лu	ı_ila_0/inst/ila_core_inst/u_il	a_regs/MU_SRL[2].mu				±
		Гu	ı_ila_0/inst/ila_core_inst/u_il	a_regs/MU_SRL[3].mu				
		Лu	ı_ila_0/inst/ila_core_inst/u_il	a_regs/MU_SRL[4].mu				
OK Cancel		<		>				
						ок	(	Cancel

8. Now click **OK** in the Replace Debug Probes dialog. An additional dialog box may appear if the nets were marked with DONT\_TOUCH indicating that it must be removed to proceed. If so, click **Unset Property and Continue**.



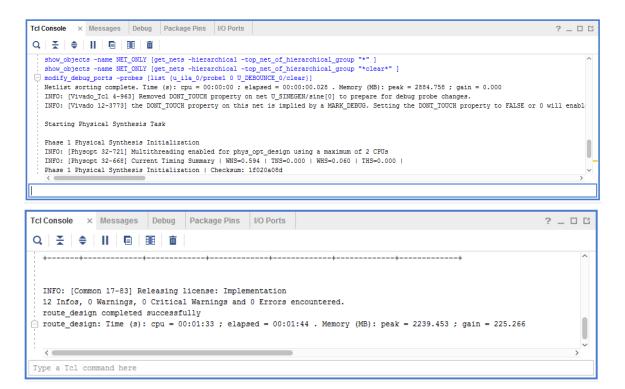


Replace Debug Probes		<b>X</b>
	ne or more debug probes. To reflect these ger, regenerate the debug probes file (LTX)	1
Ξ		
Search: Q-		
Name	Probe	
Och 13	_	0 ^
Ch 14	_	0
Oh 15	_	0
Ch 16	_	0
Och 17	_	0
Ch 18	_	0
Oh 19	_	0
probe2 (2)		
Ch 0	_	0
Oh 1	_	0
probe3 (2)		
Ch 0	_	0
Oh 1	_	0
probe4 (2)		
Ch 0	_ U_DEBOUNCE_0/clear	C
Oh 1	_	0
probe5 (2)		
Ch 0	_	0 ~
Probes changed: 1		
	OK Can	col
	Cal	Cei

**IMPORTANT!** Check the Tcl Console to ensure that there are no Warnings/Errors.





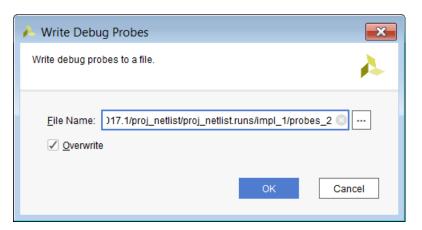


9. Save your modifications to a new checkpoint. Use the Save Checkpoint As option in the ECO Navigator to bring up the Save Checkpoint As dialog box. Specify a file name for the .dcp file and click **OK**.

int As	×
Create a checkpoint file that contains the netlist, XDC constraints, and the physical database.	4
Checkpoint file: 2017.1/proj_netlist/proj_netlist.runs/impl_1/checkpoint_1.d	 cel

10. Click **Write Debug Probes** in the ECO Navigator. When the Write Debug Probes dialog appears, click **OK** to generate a new .ltx file for the debug probes.





11. Click Generate Bitstream in the ECO navigator. When the Generate Bitstream dialog appears, change the bit file name to project\_sinegen\_demo\_routed\_debug\_changes.bit in the Bit File field and click OK to generate a new .bit file that reflects the debug probe changes.

🔶 Generat	e Bitstream		×
Create a pro	ogramming file from the curr	ent design	4
Bit File	ətlist/proj_netlist.runs/impl	_1/project_sinegen_demo_routed.	
Options			
-ra	w_bitfile		^
-m	ask_file		
-no	_binary_bitfile		
-bi	n_file		
-re	adback_file		
-lo	gic_location_file		
-ve	rbose		~
Select	an option above to see a des	scription of it	
		ОК С	ancel

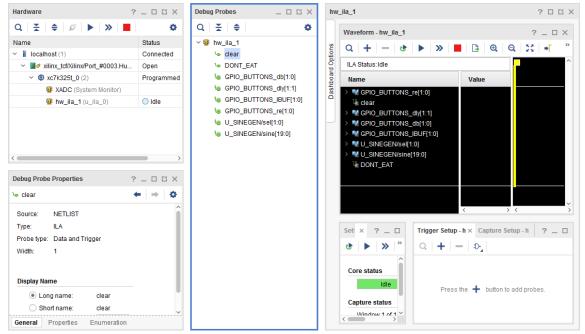
- 12. Connect to the Vivado Hardware Manager by selecting Open Hardware Manager in the ECO Navigator.
- 13. Connect to the local hardware server by following the steps in the Target Board and Server Set Up section in Lab 5: Using the Vivado Logic Analyzer to Debug Hardware.

Program the device using the <code>.bit</code> file and <code>.ltx</code> files that you created in the previous steps.



🔶 Program Device		×
	ramming file and download it to your hardware device. You can optionally file that corresponds to the debug cores contained in the bitstream	•
Bitstre <u>a</u> m file: Debu <u>o</u> probes file: ✓ <u>E</u> nable end of st	<ul> <li>'.1/proj_netlist/proj_netlist.runs/impl_1/project_sinegen_demo_routed.bit </li> <li></li> <li>://ivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1/probes_1.ltx </li> <li></li> <li>artup check</li> </ul>	
?	Program	]

14. Select **Window** → **Debug Probes** from the Vivado Design Suite toolbar. Ensure that the probes that were replaced in step 8 and 9 above are reflected in the probes associated with hw\_ila\_1.



15. Run the Trigger on the ILA. Ensure the probes that were replaced in step 8 and 9 above are reflected in the Waveform window as well.





Vaveform - hw_ila_1		? _ 🗆 >
Q   <b>+</b>   −   ϑ   ▶   ≫	• • • • • • • • • • • • • • • • • • •	
ILA Status:Idle		0
Name	Value	
GPIO_BUTTONS_re[1:0]	0	(
¼ clear	1	
GPIO_BUTTONS_dly[1:1]	0	0
GPIO_BUTTONS_db[1:0]	0	
GPIO_BUTTONS_IBUF[1:0]	0	
U_SINEGEN/sel[1:0]	0	
U_SINEGEN/sine[19:0]	05a81	
U DONT_EAT	0	
		Updated at: 2017-Mar-17 14:43:26





Lab 7

## Debugging Designs Using the Incremental Compile Flow

This lab introduces the Vivado<sup>®</sup> Incremental Compile Flow to add/edit/delete debug cores to an earlier implementation of the design.

#### Procedure

This lab consists of five generalized steps followed by general instructions and supplementary detailed steps that allow you to make choices based on your skill level as you progress through the lab.

If you need help completing a general instruction, go to the detailed steps below it, or if you are ready, simply skip the step-by-step directions and move on to the next general instruction.

The lab has five primary steps as follows:

- 1. Step 1: Opening the Example Design and Adding a Debug Core
- 2. Step 2: Compiling the Reference Design
- 3. Step 3: Create New Runs
- 4. Step 4: Making Incremental Debug Changes
- 5. Step 5: Running Incremental Compile

## Step 1: Opening the Example Design and Adding a Debug Core

1. Start the Vivado IDE.

Load the Vivado IDE by doing one of the following:

- Double-click the Vivado IDE icon on the Windows desktop.
- Type vivado in a command terminal.





From the Getting Started page, click **Open Example Project.** 

- 2. In the Open Example Project dialog box, click Next.
- 3. Select the CPU (Synthesized) design template, and click Next.
- 4. In the Project Name dialog box, specify the following:
  - **Project name:** project\_cpu\_incremental
  - **Project location:** <Project\_Dir>

Click Next.

- 5. In the Default Part screen, select **xc7k70tfbg676-2** and click **Next**.
- 6. The New Project Summary screen appears, displaying project details. Reviewed these and click **Finish**.
- 7. When the Vivado IDE opens with the default view, open the Synthesized design.
- 8. In the Netlist window, select the set of signals specified below in the <code>cpuEngine</code> hierarchy and apply the MARK\_DEBUG property by right-clicking and selecting **Mark Debug** from the dialog.

```
cpuEngine/dcqmem_dat_qmem[*],
cpuEngine/dcpu_dat_qmem[*],
cpuEngine/dcqmem_adr_qmem[*],
cpuEngine/du_dsr[*],
cpuEngine/dvr0__0[*],
cpuEngine/du_dsr[*],
cpuEngine/dcqmem_sel_qmem[*]
```





SYNTHE SIZED D	DESIGN - constrs	_2 xc7k70tfbg676-2 (activ	ve)
Sources N	letlist ×		? _ 🗆 🖸
꽃 눩			•
> 鈩	dcpu_dat_qme dcpu_sel_cpu	(32) m (25) (3)	Î
〉小	dcpu_tag_dmn dcqmem_adr_ dcqmem_dat_ dcqmem_set	qmem (32) qmem (32)	
	dcqmem_ dcqmem_ 🕷	Bus Net Properties Mark Debug	Ctrl+E
> 먗	dcsb_adr dcsb_sel_ dout (32)	Unmark Debug Assign to Debug Port Select Driver Pin	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
> 乐 > 乐	dtlb_ppn ( du_dat_c;	Schematic Show Connectivity	F4 Ctrl+T
> 鈩	du_excepi	Unhighlight	F6
> 鈩	dwb_dat_ E (1) ex_insn (2	Mark Unmark Go to Source	► Ctrl+Shift+M F7
> 小	fifo_dat_o (3)		~

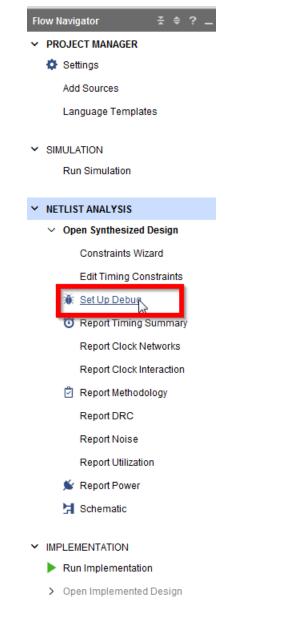
Alternatively, you can use the following Tcl command to set the MARK\_DEBUG property on the signals specified.

```
set_property mark_debug true [get_nets [list {cpuEngine/
dcqmem_dat_qmem[*]}
  {cpuEngine/dcpu_dat_qmem[*]} {cpuEngine/dcqmem_adr_qmem[*]}
  {cpuEngine/du_dsr[*]} {cpuEngine/dvr0__0[*]} {cpuEngine/du_dsr[*]}
  {cpuEngine/dcqmem_sel_qmem[*]}]
```

9. In the Flow Navigator, click **Set Up Debug** to invoke the Set Up Debug wizard.



#### AMDA XILINX



- PROGRAM AND DEBUG
  - 👫 Generate Bitstream
  - > Open Hardware Manager

10. When the Set Up Debug Wizard appears, click Next.





Q					0
Name	Clock Domain	Driver Cell	Probe Type		
> 小章 cpuEngine/dcpu_dat_qmem (25)	clkgen/cpuClk	FDRE	Data and Trigger		
> fre cpuEngine/dcqmem_adr_qmem (32)	clkgen/cpuClk	FDRE	Data and Trigger 🚿		
> Jre cpuEngine/dcqmem_dat_qmem (32)	clkgen/cpuClk	FDRE	Data and Trigger 🚿		
> 「## cpuEngine/dcqmem_sel_qmem (4)	clkgen/cpuClk	FDRE	Data and Trigger 🚿		
› 「☆ cpuEngine/du_dsr (11)	clkgen/cpuClk	FDCE	Data and Trigger 🚿	-	
> 「宽 cpuEngine/dvr00 (6)	clkgen/cpuClk	FDCE	Data and Trigger 🚿		

- 11. When the ILA Core Options screen appears, click Next again.
- 12. When the Set Up Debug Summary screen appears, ensure that one debug core is created and click **Finish**.
- 13. Check the Debug widow to ensure that the u\_ila\_0 core has been inserted into the design.

cl Console Messages Log Reports Desi	gn Runs Debug	×		? _ 🗆
Q.   풒   ≑   兼   <b>士</b>   ≝				
lame	Driver Cell	Driver Pin	Probe Type	
dbg_hub (labtools_xsdbm_v3)				
u_ila_0 (labtools_ila_v6)				
> 🗃 clk (1)				
> 📄 probe0 (32)			Data and Trigger 👒	
> probe1 (4)			Data and Trigger 👒	
> probe2 (32)			Data and Trigger 🛛 👻	
> 📄 probe3 (11)			Data and Trigger 👒	
> probe4 (25)			Data and Trigger 👒	
> 📄 probe5 (6)			Data and Trigger 👒	
Unassigned Debug Nets (0)				

14. Save the new debug XDC commands by selecting **File** → **Constraints** → **Save** or clicking the **Save Constraints** button.

#### **Step 2: Compiling the Reference Design**

The following are the steps to run implementation on the reference design.





- 1. From the Flow Navigator, select **Run Implementation**.
- 2. After implementation finishes, the Implementation Complete dialog box opens. Click Cancel.
- 3. In a project-based design, the Vivado Design Suite saves intermediate implementation results as design checkpoints in the implementation runs directory. You will use one of the saved design checkpoints from the implementation in the incremental compile flow.

**TIP:** When you re-run implementation, the previous results will be deleted. Save the intermediate implementation results to a new directory or create a new implementation run for your incremental compile to preserve the reference implementation run directory.

- 4. In the Design Runs window, right-click impl\_1 and select Open Run Directory from the popup menu. This opens the run directory in a file browser as seen in the following figure. The run directory contains the routed checkpoint (top\_routed.dcp) to be used later for the incremental compile flow. The location of the implementation run directory is a property of the run.
- 5. Get the location of the current run directory in the Tcl Console by typing:

```
get_property DIRECTORY [current_run]
```

This returns the path to the current run directory that contains the design checkpoint. You can use this Tcl command, and the DIRECTORY property, to locate the DCP files needed for the incremental compile flow.

#### **Step 3: Create New Runs**

In this step, you define new synthesis and implementation runs to preserve the results of the current runs. Then you make debug related changes to the design and rerun synthesis and implementation. If you do not create new runs, Vivado overwrites the current results.

- 1. From the Vivado tool bar, select  $Flow \rightarrow Create Runs$  to invoke the Create New Runs wizard.
- 2. In the Create New Runs screen, click Next.
- 3. The Configure Implementation Runs screen opens, as shown in the figure below. Select the Make Active check box, and click **Next**.





٨	Create New R	luns			×
			ntation runs using variou	s parts, constraints, flows and strategies	4
	reate and comig		mation runs using variou	s paris, consulantis, nows and sublegies	
	Create Implem	entation Runs			
	+   -				
	Name	Constraints Set	Part	Strategy	Make Active
	impl_2 💌	🛅 constrs_2 (act 🗸	🛑 xc7k70tfbg67 🗸	🏂 Vivado Implementation Defaults (Vivado Implementation 2 👻	$\checkmark$
				R	uns to create: 1
(	?			< <u>B</u> ack <u>N</u> ext> <u>F</u> inish	Cancel

4. From the Launch Options window, select Do not launch now and click **Next**.

n Create New Runs				×
Launch Options Configure hosts for launching runs, and/or set advanced launch options				4
Launch girectory: 💿 < Default Launch Directory>				*
<ul> <li>● Launch runs on local host Number of jobs: 4 </li> <li>○ Generate scripts only</li> <li>○ Do not launch now</li> </ul>				
(?)	< <u>B</u> ack	Next >	<u>F</u> inish	Cancel

5. In the Create New Runs Summary screen, click **Finish** to create the new runs.

The Design Runs window displays the new active runs in bold.



Tcl Console Messag	ges Log	Reports Package Pins	Design Runs	×	Power	Timir	ng Met	hodology D	RC							? _	0 6
Q   素   ♣   I4	≪ ►	<b>» +</b> %															
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Strat
✓ impl_1 (active)	constrs_2	route_design Complete!	1.265	0.0	0.057	0.0	0.000	2.393	0	21	1	112.50	0	68	3/1	00:14:25	Viva
▷ impl_2	constrs_2	Not started															Vivad
<																	>

#### **Step 4: Making Incremental Debug Changes**

In this step, to add/delete/edit debug cores, you need to reopen the synthesized netlist. Make debug related changes to the design using the Set Up Debug wizard.

- 1. If you have closed the synthesized netlist, go back to the synthesized design using the Flow Navigator.
- For this tutorial, assume that you now need to debug some other nets in addition to the ones already being debugged. However, you want to reuse the previous place and route results. Now, you will debug the nets fftEngine/fifo\_out[\*].
- 3. Apply the MARK\_DEBUG property to this bus in the netlist window.

Sources Netlist ×	?	_	Ľ
X H			٥
🕅 top			^
> 🖻 Nets (4564)			
> 🖻 Leaf Cells (223)			
> Clkgen (clock_generator)			
> CpuEngine (or1200_top)			
> 🦉 dbg_hub (dbg_hub_CV)			
✓ I fftEngine (fftTop)			
Nets (3331)			
≻ 近 A(16)			
> 近 C (16)			
> 「f D (32)			
> <b>师¤</b> <mark>fifo_out (32)</mark>			
> 近 13 (32)			
> Jr 14 (32)			
> Jr 15 (32)			
> 「」 16 (32)			
> - √r 17 (32)			
N TE 10 (22)			$\sim$





- 4. Click **Set Up** Debug to invoke the Set Up Debug wizard in the Flow Navigator.
- 5. In the Existing Debug Nets tab, select **Continue debugging 110 nets connected to existing debug core**.

i Set Up Debug				×
Existing Debug Nets Choose how to handle existing nets connected to debug cores.				4
<ul> <li>Continue debugging 110 nets connected to existing debug core</li> <li>Only debug new nets</li> <li>Disconnect all nets and remove debug cores</li> </ul>				
•	< <u>B</u> ack	Next >	Einish	Cancel

6. Click **Next** to debug the new unassigned debug nets.





🍌 Set Up Debug				×
Additional Debug Nets Choose additional nets to debug.				4
<ul> <li>✓ Debug 32 unassigned debug nets</li> <li>✓ Debug 32 selected nets</li> </ul>				
	< <u>B</u> ack	<u>N</u> ext ≻	<u>F</u> inish	Cancel

7. Click **Next** and ensure the new nets are in the list of Nets to Debug.

Q.   ★   ♦   №   Ⅲ   +   −				0
Name	Clock Domain	Driver Cell	Probe Type	
> 小* cpuEngine/dcpu_dat_qmem (25)	clkgen/cpuClk	FDRE	Data and Trigger 🗸	
> J & cpuEngine/dcqmem_adr_qmem (32)	clkgen/cpuClk	FDRE	Data and Trigger 🗸	
> 『# cpuEngine/dcqmem_dat_qmem (32)	clkgen/cpuClk	FDRE	Data and Trigger 🔍	
→ 「「★ cpuEngine/dcqmem_sel_qmem (4)	clkgen/cpuClk	FDRE	Data and Trigger 🔍	
≻ <b>师撇 cpuEngine/du_dsr</b> (11)	clkgen/cpuClk	FDCE	Data and Trigger 🔍	
> <b>师業 cpuEngine/dvr00</b> (6)	clkgen/cpuClk	FDCE	Data and Trigger 🗸 🗸	•
> 🕂 fftEngine/fifo_out (32)	clkgen/fftClk	RAMB36	Data and Trigger 🛛 🗸	
Find Nets to Add				Nets to debug: 1

8. Click **Next** and ensure that two debug cores are created and click **Finish**.



 Save the new debug XDC commands by clicking the Save Constraints button or selecting File → Constraints → Save from the main Vivado toolbar.

#### **Step 5: Running Incremental Compile**

In the previous steps, you have updated the design with debug changes. You could run implementation on the new netlist, to place and route the design and work to meet the timing requirements. However, with only minor changes between this iteration and the last, the incremental compile flow lets you reuse the bulk of your prior debug, placement and routing efforts. This can greatly reduce the time it takes to meet timing on design iterations. For more information, refer to *Vivado Design Suite User Guide: Implementation* (UG904).

- 1. Start by defining the design checkpoint (DCP) file to use as the reference design for the incremental compile flow. This is the design from which the Vivado Design Suite draws placement and routing data.
- 2. In the Design Runs window, right-click the **impl\_2 run** and select **Set Incremental Implementation** from the popup menu. The Set Incremental Implemenation dialog box opens.
- 3. Select Automatically use the checkpoint from the previous run.
- 4. Click **OK**. This information is stored in the INCREMENTAL\_CHECKPOINT property of the selected run. Setting this property tells the Vivado Design Suite to run the incremental compile flow during implementation.
- 5. You can check this property on the current run using the following Tcl command:

get\_property INCREMENTAL\_CHECKPOINT [current\_run]

This returns the full path to the top\_routed.dcp checkpoint.

**TIP:** To disable Incremental Compile for the current run, clear the INCREMENTAL\_CHECKPOINT property. This can be done using the Set Incremental Compile dialog box, or by editing the property directly through the Properties window of the design run, or through the *reset\_property* command.

6. From the Flow Navigator, select Run Implementation.

This runs implementation on the current run, using thetop\_routed.dcp file as the reference design for the incremental compile flow. When the run is finished, the Implementation Completed dialog box opens.

7. Select **Open Implemented Design** and click **OK**. As shown in the following figure, the Design Runs window shows the elapsed time for implementation run impl\_2 versus impl\_1.



Tcl Console Messag	jes Log	Reports	Design Runs	K Power	DRC	Methodo	ology	Timing										? _ 🗆 🖸
Q   <u>∓</u>   <b>≑</b>   I4	« <b>&gt;</b>	» +	%															
Name	Constraints	Status		WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strateg
🎺 impl_1	constrs_2	Implement	tation Out-of-date	0.530	0.000	0.041	0.000	0.000	2.395	0	21386	18106	112.50	0	68	4/18/18 5:11 PM	00:10:36	Vivado Impl
✓ impl_2 (active)	constrs_2	route_des	ign Complete!	0.530	0.000	0.055	0.000	0.000	2.408	0	22029	19264	113.50	0	68	4/18/18 5:26 PM	00:10:05	Vivado Impl
4																		
·																		

*Note:* This is an extremely small design. The advantages of the incremental compile flow are greater and significant with larger, more complex designs.

8. Select the Reports tab in the Results window area and under Place Design, double-click **Incremental Reuse Report** as shown in the following figure.

Tcl Console Messages Log Reports × Design Runs Pow	ver DRC Methodology Timing	? _ 🗆		
Q ≍ ≑ + - ∅ ►				
Report Y imprementation	Report Type	Options		
✓ impl_2				
> Design Initialization (init_design)				
> OptDesign (opt_design)				
<ul> <li>Power Opt Design (power_opt_design)</li> </ul>				
<ul> <li>Place Design (place_design)</li> </ul>				
impl_2_place_report_io_0	Report information about all the IO sites on the device (report_io)			
impl_2_place_report_utilization_0	Report on utilization of resources on the targeted device (report_utilization)	slr = false; packthru = false; hierarchical =		
impl_2_place_report_control_sets_0	Report the unique control sets in design (report_control_sets)	verbose = true;		
impl_2_place_report_incremental_reuse_0	Report on achievable incremental reuse for the given design-checkpoint (report_incremental_reuse)	hierarchical = false;		
impl_2_place_report_incremental_reuse_1	Report on achievable incremental reuse for the given design-checkpoint (report_incremental_reuse)	hierarchical = false;		
impl_2_place_report_timing_summary_0	Report timing summary (report_timing_summary)	check_timing_verbose = false; setup = false		
> Post-Place Power Opt Design (post_place_power_opt_design)				
<ul> <li>Post-Place Phys Opt Design (phys_opt_design)</li> </ul>				
> Route Design (route_design)				
<ul> <li>Post-Route Phys Opt Design (post_route_phys_opt_design)</li> </ul>				
> Write Bitstream (write_bitstream)				

The Incremental Reuse Report opens in the Vivado IDE text editor. This report shows the percentage of reused Cells, Ports, and Nets. A higher percentage indicates more effective reuse of placement and routing from the incremental checkpoint.





- H							1
	→   X   🖬   🛍   X	×   //   ⊞≣   ♀				Read-only	1
	t 1986-2018 Xilinx, Inc	-					
1	ersion : Vivado v.2018.						
1001 V			00 Wed Apr 4 16:40:5	6 MDI 2010			
Host	: xcosmitha32 ru	nning 64-bit Service	Pack 1 (build 7601)				
1.1.1	d : report_increment	ntal_reuse -file top_	incremental_reuse_pre	placed.rg	ot.rpt		
Design	: top			-			
Device	: xc7k70t						
-	State : Fully Routed						
1							
Theremon	tal Implementation Info:	wation					
Incremen	cai impiementation inio.	rmation					
Table of	Contents						
1. Reuse	Summary						
2. Refer	ence Checkpoint Informat	tion					
-	rison with Reference Ru	n					
4. Non R	euse Information						
	<b>5</b>						
1. Reuse	-						
1. Reuse	-						
1. Reuse	-	+	+	+	÷		
1. Reuse  +   Type	 +	Reuse % (of Total)	Fixed % (of Total)	Total	l		
1. Reuse  +   Type +	 +   Matched % (of Total) +	Reuse % (of Total) +	Fixed % (of Total)	Total   +	+		
1. Reuse  +   Type +   Cells	   Matched % (of Total) +   95.69	Reuse % (of Total) +	Fixed % (of Total) +	Total   +	 + 		
1. Reuse    Type +   Cells   Nets	   Matched % (of Total)    95.69   95.80	Reuse % (of Total)    88.59   80.85	Fixed % (of Total)    0.31   0.00	Total   +   46475     36769	 + 		
<ol> <li>Reuse</li> <li>Type</li> <li>Type</li> <li>Cells</li> <li>Nets</li> <li>Pins</li> </ol>	   Matched % (of Total)    95.69   95.80   -	Reuse % (of Total)   88.59   80.85   86.48	Fixed % (of Total)   0.31   0.00   -	Total   +   46475     36769     189880	 -   		
1. Reuse    Type +   Cells   Nets   Pins   Ports	   Matched % (of Total)    95.69   95.80   -	Reuse % (of Total)   88.59   80.85   86.48   100.00	Fixed % (of Total)   0.31   0.00   -   100.00	Total     46475     36769     189880     135	 -   		
Type +   Cells   Nets   Pins   Ports	   Matched % (of Total) +   95.69   95.80   -   100.00	Reuse % (of Total)   88.59   80.85   86.48   100.00	Fixed % (of Total)   0.31   0.00   -   100.00	Total     46475     36769     189880     135	 -   		
1. Reuse    Type +   Cells   Nets   Pins   Ports +	   Matched % (of Total) +   95.69   95.80   -   100.00	Reuse % (of Total)   88.59   80.85   86.48   100.00	Fixed % (of Total)   0.31   0.00   -   100.00	Total     46475     36769     189880     135	 -   		
1. Reuse    Type +   Cells   Nets   Pins   Ports +	   Matched % (of Total) +   95.69   95.80   -   100.00	Reuse % (of Total)   88.59   80.85   86.48   100.00	Fixed % (of Total)   0.31   0.00   -   100.00	Total     46475     36769     189880     135	 -   		
1. Reuse    Type +   Cells   Nets   Pins   Ports + 2. Refer	Matched % (of Total) 95.69 95.80 1 - 100.00	Reuse % (of Total)   88.59   80.85   86.48   100.00 +	Fixed % (of Total)   0.31   0.00   -   100.00	Total     46475     36769     189880     135	 -   		

In the report, fully reused nets indicate that the entire routing of the nets is reused from the reference design. Partially reused nets indicate that some of the routing of the nets reuses routing from the reference design. Some segments re-route due to changed cells, changed cell placements, or both. Non-reused nets indicate that the net in the current design was not matched in the reference design.

#### Conclusion

This concludes the lab. You can close the current project and exit the Vivado IDE.

In this lab, you learned how to run the Incremental Compile Debug flow, using a checkpoint from a previously implemented design. You inserted a new debug core using the Set Up Debug wizard on the synthesized netlist. You examined the similarity between a reference design checkpoint and the current design by examining the Incremental Reuse Report.



#### Lab 8

# Using the Vivado Serial Analyzer to Debug Serial Links

The Serial I/O analyzer is used to interact with IBERT debug IP cores contained in a design. It is used to debug and verify issues in high speed serial I/O links.

The Serial I/O Analyzer has several benefits:

- Tight integration with Vivado<sup>®</sup> IDE.
- Ability to script during netlist customization/generation and serial hardware debug.
- Common interface with the Vivado Integrated Logic Analyzer (ILA).

The customizable LogiCORE<sup>™</sup> IP Integrated Bit Error Ratio Tester (IBERT) core for 7 series FPGA GTX transceivers is designed for evaluating and monitoring the GTX transceivers. This core includes pattern generators and checkers that are implemented in FPGA logic, and provides access to ports and the dynamic reconfiguration port attributes of the GTX transceivers. Communication logic is also included to allow the design to be run time accessible through JTAG.

In the course of this tutorial, you:

- Create, customize, and generate an Integrated Bit Error Ratio Tester (IBERT) core design using the Vivado tool.
- Interact with the design using Serial I/O Analyzer. This includes connecting to the target KC705 board, configuring the device, and interacting with the IBERT/Transceiver IP cores.
- Perform a sweep test to optimize your transceiver channel and to plot data using the IBERT sweep plot GUI feature.

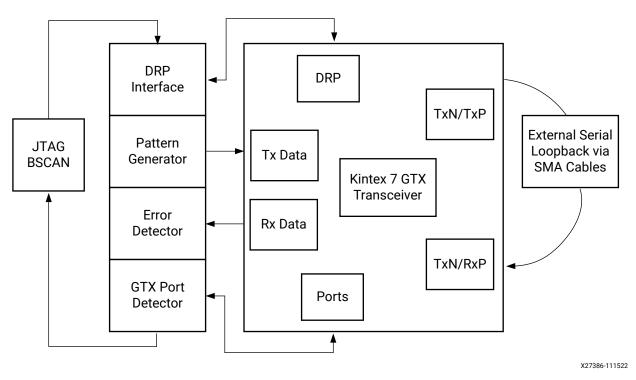
#### **Design Description**

You can customize the IBERT core and use it to evaluate and monitor the functionality of transceivers for a variety of Xilinx<sup>®</sup> devices. The focus for this tutorial is on Kintex<sup>®</sup>-7 GTX transceivers. Accordingly, the KC705 target board is used for this tutorial.

The following figure shows a block diagram of the interface between the IBERT Kintex-7 GTX core interfaces with Kintex-7 transceivers.



- DRP Interface and GTX Port Registers: IBERT provides you with the flexibility to change GTX transceiver ports and attributes. Dynamic reconfiguration port (DRP) logic is included, which allows the runtime software to monitor and change any attribute in any of the GTX transceivers included in the IBERT core. When applicable, readable and writable registers are also included. These are connected to the ports of the GTX transceiver. All are accessible at run time using the Vivado<sup>®</sup> logic analyzer.
- **Pattern Generator:** Each GTX transceiver enabled in the IBERT design has both a pattern generator and a pattern checker. The pattern generator sends data out through the transmitter.
- **Error Detector:** Each GTX transceiver enabled in the IBERT design has both a pattern generator and a pattern checker. The pattern checker takes the data coming in through the receiver and checks it against an internally generated pattern.



#### Figure 4: IBERT Design Flow

# Step 1: Creating, Customizing, and Generating an IBERT Design

To create a project, use the New Project wizard to name the project, to add RTL source files and constraints, and to specify the target device.

1. Invoke the Vivado IDE.



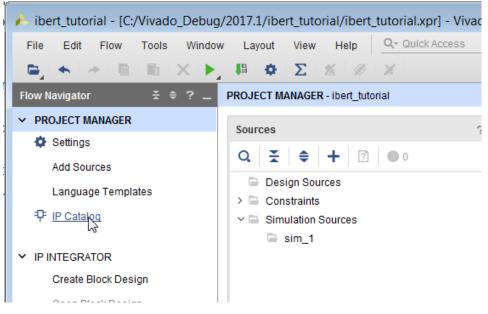
- 2. In the Quick Start screen, click **Create Project** to start the New Project wizard, and click **Next**.
- 3. In the Project Name page, name the new project ibert\_tutorial and provide the project location (C:/ibert\_tutorial). Ensure that **Create Project Subdirectory** is selected. Click Next.
- 4. In the Project Type page, specify the Type of Project to create as RTL Project. Click Next.
- 5. In the Add Sources page, click Next.
- 6. In the Add Existing IP page, click Next.
- 7. In the Add Constraints page, click Next.
- 8. In the Default Part page, select Boards and then select Kintex-7 KC705 Evaluation Platform. Click **Next**.
- 9. Review the New Project Summary page. Verify that the data appears as expected, per the steps above. Click **Finish**.

Note: It might take a moment for the project to initialize.

#### Step 2: Adding an IBERT Core to the Vivado Project

1. In the Flow Navigator click IP Catalog.

The IP Catalog opens.



2. In the search field of the IP Catalog type IBERT, to display the IBERT 7 series GTX IP.



Project Summary × IP Catalog ×					202
Cores   Interfaces					
≚   ≑   释 +€   ⊁   ⊘   ⊕   ᡚ   Q~ IBERT	8				٥
Name	∧1 AXI4	Status	License	VLNV	
<ul> <li>Vivado Repository</li> </ul>					
✓					
✓					
IBERT 7 Series GTX		Production	Included	xilinx.com:ip:ibert_7series_gtx:3.0	

- 3. Double-click IBERT 7 series GTX IP. This brings up the customization GUI for the IBERT.
- 4. In the Customize IP dialog box, choose the following options in the Protocol Definition tab:
  - a. Type the name of the component in the Component Name field. In this case, leave the name as the default name, ibert\_7series\_gtx\_0.
  - b. Ensure that the Silicon Version is selected as General ES/Production.
  - c. Ensure that the Number of Protocols option is set to 1.
  - d. Change the LineRate (Gb/s) to 8.
  - e. Change DataWidth to 40.
  - f. Change Refclk (MHz) to 125.
  - g. Ensure that the Quad Count is set to 2.
  - h. Ensure Quad PLL box is selected.

Customize IP									×
IBERT 7 Series GTX (3.0)								I	2
Occumentation IP Location C Switch to Defaults									
Show disabled ports	Component Name	ibert_7	series_gtx_0						0
	Protocol Definition	Pro	tocol Selection Clock	Settings Summar	у				
RXN_[3:0] RXP_[3:0] TXN_0[3:0] GTREFCLK0_[0:0] TXP_0[3:0] GTREFCLK1_[0:0] RXOUTCLK_0 SYSCLK_I	Number of Protoc	mber of	f quads available for this d	DataWidth	Refclk(MHz)	1 Quad Count	_	V Quad PLL	
	Custom 1	•	8	40 •	125.000 •	2	•		
							ОК	Cancel	I

- 5. Under the Protocol Selection tab, update the following selections:
  - a. For GTX Location QUAD\_117, in the Protocol Selected column, click the pull-down menu and select Custom 1 / 8 Gbps. This should automatically populate Refclk Selection to MGTREFCLK0 117 and TXUSRCLK Source to Channel 0.



- b. For GTX Location QUAD\_118, do the following:
  - i. In the Protocol Selected column, click the pull-down menu and select **Custom 1 / 8 Gbps**.
  - ii. In the Refclk Selection column, change the value to MGTREFCLK0 117.
  - iii. In the TXUSRCLK Source column, change the value to Channel 0.

🔶 Customize IP							<b>—</b>
IBERT 7 Series GTX (3.0)							4
Documentation 📄 IP Location C Switch to Defaults							
Show disabled ports	Component Name it	pert_7series_gtx_0					8
	Protocol Definition	Protocol Selection	Clock Settings	Summar	y		
	Please select Proto	col-Quad combination					
	GTX Location	Protocol Sel	ected	Refo	dk Selection		TXUSRCLK Source
	QUAD_115	None		▼ Non		۳	Channel 0 👻
RXN 1(7:0)	QUAD 116	None		▼ Non		۰	Channel 0 👻
RXP_[[7:0] TXN_0[7:0]	QUAD_117	Custom 1/8			REFCLK0 117	•	Channel 0 🔹
GTREFCLK0_[[1:0] TXP_0[7:0]	QUAD_118	Custom 1/8	Gbps	<ul> <li>MGT</li> </ul>	REFCLK0 117	۳	Channel 0 🔹
GTREFCLK1_[[1:0] RXOUTCLK_O							
11							OK Cancel

- 6. Click the **Clock Settings tab** and make the following changes for both QUAD\_117 and QUAD\_118:
  - a. Leave the Source column at its default value of External.
  - b. Change the I/O Standard column to DIFF SSTL15.
  - c. Change the P Package Pin to AD12.
  - d. Change the N Package Pin to AD11.
  - e. Leave the Frequency (MHz) at its default value of 200.00.





Customize IP										×
IBERT 7 Series GTX (3.0)										4
Documentation 📄 IP Location C Switch to Defaults										
Show disabled ports	Component Name ibe	ert_7series_gtx_0								8
	Protocol Definition	Protocol Selection	Clock Settings	Sumn	nary					
	RXOUTCLK Probe		I/0 Standard		P Package Pin		N Package Pin		Frequency(MHz)	-
RXN_I(7:0) RXP_I(7:0) TXN_0(7:0)	System Clock		DIFF SSTL15	•	AD12	Θ	AD11	Θ	200.00	0
GTREFCLK0_[[1:0] TXP_0[7:0] GTREFCLK1_[[1:0] RXOUTCLK_0 SYSCLK_I	System Clock Termi									
								0	K Car	icel

7. Click the Summary tab and ensure that the content matches the following figure, then click **OK**.

🔥 Customize IP		<b>X</b>
IBERT 7 Series GTX (3.0)		4
Ocumentation 📄 IP Location C Switch to Defaults		
Show disabled ports	Component Name ibert_7series_gtx_0	8
RXN_[7:0]         RXP_[7:0]       TXN_0[7:0]         GTREFCLK0_[[1:0]       TXP_0[7:0]         GTREFCLK1_[[1:0]       RXOUTCLK_0         SYSCLK_1       SYSCLK_1	Protocol Definition Protocol Selection Clock Settings Su IBERT Design Summary Number of Protocols System Clock Source QUAD Count MMCM Count RefClk Sources	1           External (P Pin : AD12)           External (N Pin : AD11)           2           1           1

8. When the Generate Output Products dialog box opens, click Generate.



interview Contracts And Contra	3
The following output products will be generated.	•
Preview	
Q	
<ul> <li>✓ ₱ ibert_7series_gtx_0.xci (Global)</li> <li>⑦ Instantiation Template</li> <li>⑦ RTL Sources</li> <li>⑦ Change Log</li> </ul>	
Synthesis Options	
<ul> <li><u>Global</u></li> <li><u>O</u>ut of context per IP</li> </ul>	
Run Settings	
Number of jobs: 8 🗸	
Apply     Generate     Skip	

9. In the Sources window, right-click the IP, and select **Open IP Example Design**.





Sources	?_0ĽX	Project Su
Q   ¥   ≑   +   ⊠   ●	0 🔅	Cores
✓ Design Sources (1) > ₽ ibert_7series_gt× ∩	(ibert 7series atv 0.vci)	<b>₹</b> ♦
> Constraints	Source Node Properties	Ctrl+E
Simulation Sources (1)	Enable Core Container	ас
> 🖻 sim_1 (1)		P
	Generate Output Products	
	Reset Output Products	
	Upgrade IP	
	Copy IP	
	Open IP Example Design	
	IP Documentation	, Co
	Replace File	
	Copy File Into Project	
Hierarchy IP Sources	Copy All Files Into Project	Alt+I
Source File Properties	Remove File from Project	Delete
	Enable File	Alt+Equals
♀ ibert_7series_gtx_0.xci	Disable File	Alt+Minus
IP name: IBERT 7 Serie	Hierarchy Update	•
Version: 3.0 (Rev. 16)	Refresh Hierarchy	
Description: The IBERT 7 S customizable	IP Hierarchy	
and monitorin	set as Top	
transceivers. 1 generators an	Set File Type	
implemented ports and the (	Set Used In	
attributes of th Communication	Edit Constraints Sets	
allow the desi	Edit Simulation Sets	
through Joint 1 Run-time inter	Add Sources	Alt+A

10. In the Open IP Example Design dialog box, and specify the location of your project directory. Ensure that the Overwrite existing example project is selected and click **OK**.

*Note*: This opens a new instance of Vivado<sup>®</sup> IDE with the new example design opened.





Open IP Example Design
Specify a location where the example project directory 'ibert_7series_gtx_0_ex' will be placed.
Location
Put example project directory here: C:/Vivado_Debug/2017.1 💿
Qverwrite existing example project
OK Cancel

#### Step 3: Synthesize, Implement and Generate Bitstream for the IBERT Design

1. In the newly opened instance of Vivado IDE, click **Generate Bitstream** in the Flow Navigator. When the No Implementation Results Available dialog box appears. Click **Yes**.

No Imp	elementation Results Available
?	There are no implementation results available. OK to launch synthesis and implementation? 'Generate Bitstream' will automatically start when synthesis and implementation completes.
	on't show this dialog again
	Yes <u>N</u> o

When the bitstream generation is complete, the Bitstream Generation Completed dialog box opens.

2. Select Open Hardware Manager, and click OK.



Bitstream Generation Completed						
Bitstream Generation successfully completed. Next						
Open Implemented Design						
◯ <u>V</u> iew Reports						
Open <u>H</u> ardware Manager						
Generate Memory Configuration File						
Don't show this dialog again						
OK Cancel						

3. The Hardware Manager window appears as shown in the following figure.





ibert_7series_gtx_0_ex - [c:/'	Vivado_Debug/2017/ibert_7series_gtx_0_ex/ibert_7series_gtx_0_ex.xpr] - Vivado	_ 0 🗙
File Edit Flow Tools	Window Layout View Help Qr Quick Access	write_bitstream Complete 🗸
	Σ 👂 👔 🕸 Σ 🗶 🖉 🕷 Dashboard	🔚 Serial I/O Analyzer 🗸 🗸
Flow Navigator	- HARDWARE MANAGER - unconnected	? ×
✓ PROJECT MANAGER <sup>^</sup>	No hardware target is open. Open target	
Settings	Hardware ? _ □ Ľ ×	
Add Sources		
Language Templates		
👎 IP Catalog		
<ul> <li>IP INTEGRATOR</li> <li>Create Block Design</li> <li>Open Block Design</li> <li>Generate Block Design</li> </ul>	No content	
	Properties ? _ 🗆 🖾 🗙	
✓ SIMULATION Run Simulation	$\leftrightarrow   \Rightarrow   \diamond$	
<ul> <li>RTL ANALYSIS</li> <li>Open Elaborated Design</li> <li>SYNTHESIS</li> <li>Run Synthesis</li> </ul>	Select an object to see properties	
> Open Synthesized Desigr	Tcl Console Messages Serial I/O Links × Serial I/O Scans	? _ 0 6
<ul> <li>IMPLEMENTATION</li> <li>Run Implementation</li> <li>Open Implemented Desig</li> </ul>		
✓ PROGRAM AND DEBUG	No content	
👫 Generate Bitstream		
✓ Open Hardware Manage		
Open Target 🗸 🗸		

## Step 4: Interact with the IBERT Core Using Serial I/O Analyzer

In this tutorial step, you connect to the KC705 target board, program the bitstream created in the previous step, and then use the Serial I/O Analyzer to interact with the IBERT design that you created in Step 1. You perform some analysis using various input patterns and loopback modes, while observing the bit error count.





HARDWARE MANAGER - unconnected							
Open target is open. Open target							
Hardware	ø	Auto Connect					
		Recent Targets	•				
		Available Targets on Server	× .				
		Open New Target					
			- VU				

1. Click **Open New Target**. When the Open Hardware Target wizard opens, click **Next**.

🥕 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>Back</u> Cancel

2. In the Connect to field, choose Local server. 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 target is attached to the otherwise, use Remote server.	4
<u>C</u> onnect to:	Local server (target is on local machine)	
Click Next to	launch and/or connect to the hw_server (port 3121) application on the local machine.	
?	< <u>B</u> ack <u>Next</u> > <u>Finish</u> Can	cel

3. In the Select Hardware Target page, and click **Next**.

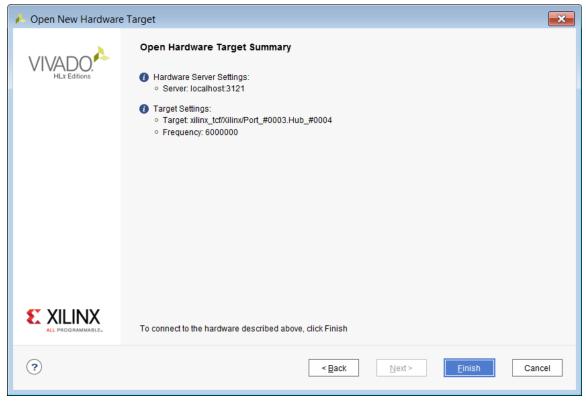
There is only one target board in this case to connect to, so that the default is selected.





Open New H	ardware Targ	et						×
	e target from the l		argets, then set the a t a different target.	ppropriate JTA	\G clock (TC	CK) frequency. If	you do not see the	4
Hardware <u>T</u> arg	jets							
Туре	Name		JTAG Clock Freque	ency				
ilinx_tcf	Xilinx/Port_#00	03.Hub_#0004	600000	×				
Hardware <u>D</u> evi Name	ces (for unknow	/n devices, spec	Add Xilinx V	irtual Cable (X egister (IR) ler				
@ xc7k325t_0	0 33651093	6						
Hardware serve	er: localhost:312	1						
?				< <u>B</u> ;	ack	<u>N</u> ext >	Einish	Cancel

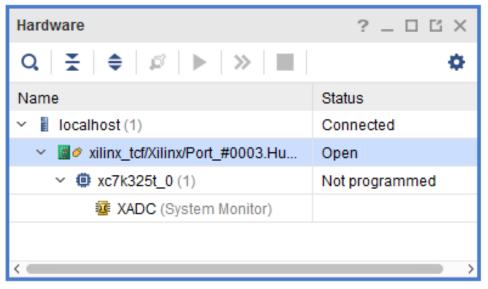
4. In the Open Hardware Target Summary page, review the options that you selected. Click **Finish**.



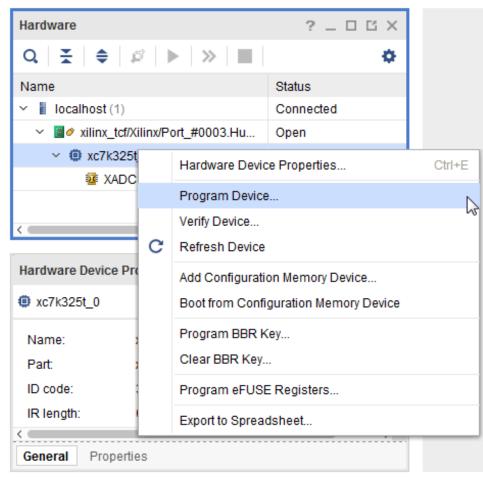




5. The Hardware window in Vivado IDE should show the status of the target FPGA on the KC705 board.



6. Select XC7K325T\_0(0) in the Hardware window, right-click and select Program Device.





7. The Program Device dialog box opens. Make sure that the correct .bit file is selected, and click **Program**.

Program Device							
Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.							
Bitstre <u>a</u> m file: Debu <u>q</u> probes file: ✔ <u>E</u> nable end of st	t_7series_gtx_0_ex.runs/impl_1/example_ibert_7series_gtx_0.bit S						
?	<u>P</u> rogram Car	icel					

8. The Hardware window now shows the IBERT IP that you customized and implemented from the previous steps. It contains two QUADS each of which has four GTX transceivers. These components of the IBERT were detected while scanning the device after downloading the bitstream. If you do not see the QUADS then select the **XC7K325 device**, right-click and select **Refresh Device**.



## AMD XILINX

Hardware	? _ 🗆 🖒 X
Q   素   ♦   ∅   ▶   ≫   ■	•
Name	Status
<ul> <li>Iocalhost (1)</li> </ul>	Connected
✓ Ø xilinx_tcf/Xilinx/Port_#0003.Hu	Open
<ul> <li>xc7k325t_0 (2)</li> </ul>	Programmed
🗿 XADC (System Monitor)	
<ul> <li>IBERT (IBERT)</li> </ul>	
Quad_117 (5)	
COMMON_X0Y2	Locked
NGT_X0Y8	No Link
NGT_X0Y9	No Link
NGT_X0Y10	No Link
NGT_X0Y11	No Link
Quad_118 (5)	
COMMON_X0Y3	Locked
NGT_X0Y12	No Link
NGT_X0Y13	No Link
NGT_X0Y14	No Link
NGT_X0Y15	No Link

9. Next, create links for all eight transceivers. Vivado Serial I/O analyzer is a link-based analyzer, which allows users to link between any transmitter and receiver GTs within the IBERT design. For this tutorial, simply link the TX and RX of the same channel. To create a link, right-click the **IBERT Core** in the Hardware window and click **Create Links**.



Hardware		? _ 🗆 🖒 ×	
Q   ¥   ♦   ∅   ▶   ≫	•		
Name		Status	
<ul> <li>Iocalhost (1)</li> </ul>		Connected	
✓ I vilinx_tcf/Xilinx/Port_#000	3.Hu	Open	
<ul> <li>xc7k325t_0 (2)</li> </ul>		Programmed	
🖉 XADC (System Monit	or)		
🗸 🖉 IBERT (IBERT)		IBERT Core Properties	Ctrl+E
Wave Quad_117 (5)			
COMMON_X0Y		Create Links	A.
NGT_X0Y8		Auto-detect Links	, in the second s
NGT_X0Y9		Serial I/O Links	
NGT_X0Y10		Serial I/O Scans	
NGT_X0Y11		Commit Properties	
V 🖏 Quad_118 (5)	с		
COMMON_X0Y	C	Refresh Serial I/O Objects	
NGT_X0Y12		Select	Þ
NGT_X0Y13		Export to Spreadsheet	
NGT_X0Y14		No Link	
NGT_X0Y15		No Link	
<		>	

The Create Links dialog box opens.

10. Ensure the first transceiver pairs (MGT\_X0Y8/TX and MGT\_X0Y8/RX) are selected.





🏄 Create Links	×
To create a new link select a TX GT and/or an RX GT, the	en click the Add button on the New Links toolbar.
TX GTs	RX GTs
<u>S</u> earch: Q-	Search: Q-
MGT_X0Y8/TX (xc7k325t_0/Quad_117)	MGT_X0Y8/RX (xc7k325t_0/Quad_117)
MGT_X0Y9/TX (xc7k325t_0/Quad_117)	MGT_X0Y9/RX (xc7k325t_0/Quad_117)
MGT_X0Y10/TX (xc7k325t_0/Quad_117)	MGT_X0Y10/RX (xc7k325t_0/Quad_117)
MGT_X0Y11/TX (xc7k325t_0/Quad_117)	MGT_X0Y11/RX (xc7k325t_0/Quad_117)
MGT_X0Y12/TX (xc7k325t_0/Quad_118)	MGT_X0Y12/RX (xc7k325t_0/Quad_118)
MGT_X0Y13/TX (xc7k325t_0/Quad_118)	MGT_X0Y13/RX (xc7k325t_0/Quad_118)
MGT_X0Y14/TX (xc7k325t_0/Quad_118)	MGT_X0Y14/RX (xc7k325t_0/Quad_118)
MGT_X0Y15/TX (xc7k325t_0/Quad_118)	MGT_X0Y15/RX (xc7k325t_0/Quad_118)
New Links	
+   -	
· · · ·	
Proce the	e 🕂 button to Add Link
Fless ut	
✓ Create link group	
Link group description: Link Group 0	$\otimes$
✓ Open Serial I/O Analyzer layout	
(?)	OK
	Caller

11. Click the "+" button add a new link. In the Link group description field, type Link Group SMA. Select the **Internal Loopback check box**.



Create Link	s		×
To create a new	link select a TX GT and/or an RX GT, then	click the Add button on the New Links toolb	var.
TX GTs		RX GTs	
Search: Q.		<u>S</u> earch: Q-	
MGT_X0Y9	/TX (xc7k325t_0/Quad_117)	MGT_X0Y9/RX (xc7k325t_0/Quad	d_117)
_	0/TX (xc7k325t_0/Quad_117)	MGT_X0Y10/RX (xc7k325t_0/Qu;	
	1/TX (xc7k325t_0/Quad_117)	MGT_X0Y11/RX (xc7k325t_0/Qu: 2 MOT_X0Y10/DX (xc7k325t_0/Qu: 2 MOT_X0Y10/DX (xc7k325t_0/Qu: 2 MOT_X0Y10/DX (xc7k325t_0/Qu: 2 MOT_X0Y11/RX (xc7k325t_0/QU))	
_	2/TX (xc7k325t_0/Quad_118) 3/TX (xc7k325t_0/Quad_118)	MGT_X0Y12/RX (xc7k325t_0/Qu: MGT_X0Y13/RX (xc7k325t_0/Qu: MGT_X0Y13/RX (xc7k325t_0/Qu: MGT_X0Y13/RX (xc7k325t_0/Qu: MGT_X0Y12/RX	
	4/TX (xc7k325t_0/Quad_118)	MGT_X0Y14/RX (xc7k325t_0/Qu	
_	5/TX (xc7k325t_0/Quad_118)	MGT_X0Y15/RX (xc7k325t_0/Qu	
New Links			
+  -			
Description	ТХ	RX	Internal Loopback
% Link 0	MGT_X0Y8/TX (xc7k325t_0/Quad_117)	MGT_X0Y8/RX (xc7k325t_0/Quad_117)	$\checkmark$
	group scription: Link Group SMA al I/O Analyzer layout		K Cancel

For the first link group, call this Link Group SMA as this is the only transceiver channel that is linked through the SMA cables. The new link shows up in the Links window.

Tcl Console Messages Serial I/O Links × Serial I/O Scans									
Name	Create Links	RX	Status	Bits	Errors	BER	BERT Reset	TX Pattern	RX Pattern
🖴 Ungrouped l	Create Link Group								
👻 🐵 Link Group S							Reset	PRBS 7-bit 💙	PRBS 7-bit \vee
🗞 Link 0	Create Sweep	MGT_X0Y8/RX	7.988 G	1.356	2.74E10	2.021	Reset	PRBS 7-bit 💉	PRBS 7-bit 💉
	Greate Sweep	ļ							

12. Click **Create Link** again to create link groups for the rest of the transceiver pairs. To do this ensure that the transceiver pairs are selected, and click the + sign icon (add new link) repeatedly, until all the links have been added to the new link group called Link Group Internal Loopback. Click **OK**.



X GTs		RX GTs	
Bearch: Q-		Search: Q.	
lew Links			
Description	ТХ	RX	Internal Loopback
	TX MGT_X0Y9/TX (xc7k325t_0/Quad_117)	RX MGT_X0Y9/RX (xc7k325t_0/Quad_117)	Internal Loopback
Description			
Description	MGT_X0Y9/TX (xc7k325t_0/Quad_117)	MGT_X0Y9/RX (xc7k325t_0/Quad_117)	
Description & Link 1 & Link 2	MGT_X0Y9/TX (xc7k325t_0/Quad_117) MGT_X0Y10/TX (xc7k325t_0/Quad_117)	MGT_X0Y9/RX (xc7k325t_0/Quad_117) MGT_X0Y10/RX (xc7k325t_0/Quad_117)	
Description S Link 1 Link 2 Link 3	MGT_X0Y9/TX (xc7k325t_0/Quad_117) MGT_X0Y10/TX (xc7k325t_0/Quad_117) MGT_X0Y11/TX (xc7k325t_0/Quad_117)	MGT_X0Y9/RX (xc7k325t_0/Quad_117) MGT_X0Y10/RX (xc7k325t_0/Quad_117) MGT_X0Y11/RX (xc7k325t_0/Quad_117)	
Description Link 1 Link 2 Link 3 Link 4	MGT_X0Y9/TX (xc7k325t_0/Quad_117) MGT_X0Y10/TX (xc7k325t_0/Quad_117) MGT_X0Y11/TX (xc7k325t_0/Quad_117) MGT_X0Y12/TX (xc7k325t_0/Quad_118)	MGT_X0Y9/RX (xc7k325t_0/Quad_117) MGT_X0Y10/RX (xc7k325t_0/Quad_117) MGT_X0Y11/RX (xc7k325t_0/Quad_117) MGT_X0Y12/RX (xc7k325t_0/Quad_118)	
Description Sulink 1 Link 2 Link 3 Link 4 Link 5	MGT_X0Y9/TX (xc7k325t_0/Quad_117) MGT_X0Y10/TX (xc7k325t_0/Quad_117) MGT_X0Y11/TX (xc7k325t_0/Quad_117) MGT_X0Y12/TX (xc7k325t_0/Quad_118) MGT_X0Y13/TX (xc7k325t_0/Quad_118)	MGT_X0Y9/RX (xc7k325t_0/Quad_117) MGT_X0Y10/RX (xc7k325t_0/Quad_117) MGT_X0Y11/RX (xc7k325t_0/Quad_117) MGT_X0Y12/RX (xc7k325t_0/Quad_118) MGT_X0Y13/RX (xc7k325t_0/Quad_118)	V V V V
Description S Link 1 S Link 2 Link 3 Link 4 Link 5 Link 5 Link 6	MGT_X0Y9/TX (xc7k325t_0/Quad_117) MGT_X0Y10/TX (xc7k325t_0/Quad_117) MGT_X0Y11/TX (xc7k325t_0/Quad_117) MGT_X0Y12/TX (xc7k325t_0/Quad_118) MGT_X0Y13/TX (xc7k325t_0/Quad_118) MGT_X0Y14/TX (xc7k325t_0/Quad_118) MGT_X0Y15/TX (xc7k325t_0/Quad_118)	MGT_X0Y9/RX (xc7k325t_0/Quad_117)           MGT_X0Y10/RX (xc7k325t_0/Quad_117)           MGT_X0Y11/RX (xc7k325t_0/Quad_117)           MGT_X0Y12/RX (xc7k325t_0/Quad_118)           MGT_X0Y13/RX (xc7k325t_0/Quad_118)           MGT_X0Y14/RX (xc7k325t_0/Quad_118)	V V V V

13. After the links have been created, they are added to the Links window as shown.

Tcl Console Messages	Serial I/O Link	ks × Serial I	/O Scans									
Q   <u>X</u>   <del>\$</del>   <del>1</del>												
Name	ТХ	RX	Status	Bits	Errors	BER	BERT Reset	TX Pattern	R			
Ungrouped Links (0)												
👻 🏐 Link Group SMA (1)							Reset	PRBS 7-bit 🗸	Р			
🗞 Link 0	MGT_X0Y8/TX	MGT_X0Y8/RX	7.988 Gbps	1.343E12	2.645E11	1.969E-1	Reset	PRBS 7-bit 🐱	Р			
👻 🚳 Link Group Internal							Reset	PRBS 7-bit 🗸	Р			
🗞 Link 1	MGT_X0Y9/TX	MGT_X0Y9/RX	7.987 Gbps	3.805E12	2.079E12	5.465E-1	Reset	PRBS 7-bit 🗸	Р			
% Link 2	MGT_X0Y10/TX	MGT_X0Y10/RX	7.988 Gbps	3.805E12	2.175E12	5.715E-1	Reset	PRBS 7-bit \vee	Ρ			

The status of the links indicate an 8.0 Gbps line rate.

For more information about the different columns of the Links windows, see the Vivado Design Suite User Guide: Programming and Debugging (UG908).

- 14. Change the GT properties of the rest of the transceivers as described above.
- 15. Next, create a 2D scan. Click **Create Scan** in the Links window.



General Prope	rties			
Tcl Console Me		Link Properties	Ctrl+E	ans
Q ≚ ♦	×	Delete	Delete	
Name		Create Links Create Link Group		us
Ungrouped I		Create San		-
✓ S Link Group S		43		-
% Link 0		Create Sweep		} Gbps
👻 🚳 Link Group I		Commit Properties		
% Link 1	С	Refresh Serial I/O Objects		Gbps

The Create Scan dialog box opens. In this dialog box, you can change the various scan properties. In this case, leave everything to its default value and click **OK**. For more information on the scan properties, see *Vivado Design Suite User Guide: Programming and Debugging* (UG908).



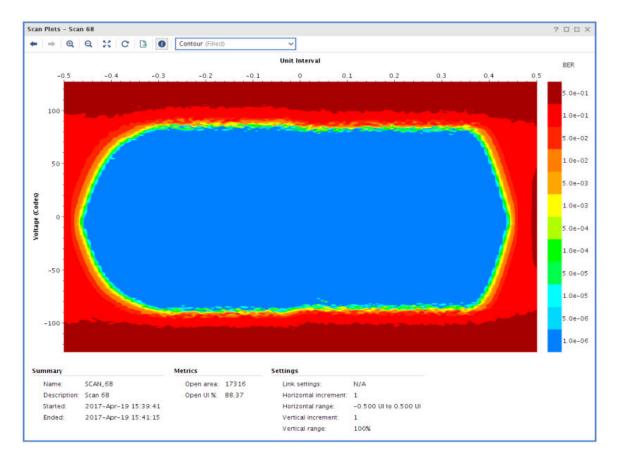
AMD**7** XILINX

🍐 Create Sca	an		×									
	Set the description and other properties to create and optionally run a scan on the selected link.											
Link:	Link: Link 0 (MGT_X0Y8/TX, MGT_X0Y8/RX)											
Description:	Scan 0	Scan 0 🛛 🛞										
Scan Proper	ties		_									
<u>S</u> can type	:	2D Full Eyescan	*									
<u>H</u> orizonta	l increment:	8	•									
H <u>o</u> rizonta	I range:	-0.500 UI to 0.500 UI	•									
<u>V</u> ertical in	crement:	8	*									
V <u>e</u> rtical ra	inge:	100%	*									
Dwell												
• <u>B</u> ER:	1e-5	\	•									
O <u>T</u> ime:		0	- -									
✓ <u>R</u> un scan												
?		OK Canc	el									

The Scan Plot window opens as shown in the following figure.



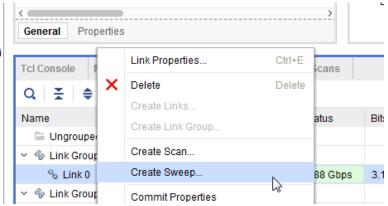




The 2D Scan Plot is a heat map of the BER value.

You can also perform a Sweep test on the links that you created earlier.

16. In the Links window, highlight Link 0 under the Link called Link Group SMA, right-click and select **Create Sweep**.



17. The Create Sweep dialog box opens, as shown below. Various properties for the Sweep test can be changed in this dialog box. Leave all the values to its default state and click **OK**.



reate Swe	еер					
ect the swe	ep propertie	s and values	s to c	reate and optionally run a set of scans on the selected link.		4
ik: L	_ink 0 (MGT_	_X0Y8/TX, MG	GT_X	0Y8/RX)		
scription:	Sweep 0					8
an Propert	ties					
Scan type:		2D Full Eye	escar			
<u>H</u> orizontal	increment:	8		~		
H <u>o</u> rizontal	range:	-0.500 UI to	0 0.50			
Vertical ind	crement:	8		~		
Vertical rai	nae:	100%		~		
-	-					
vell						
• <u>B</u> ER:	1e-5			~		
○ <u>T</u> ime:				0 🌩		
veep Prope	erties					
Sweep <u>m</u>	ode: Sem	ni Custom	~	For each property select values to be swept. The sweep will cover all combinations of property values.		
Set Prop	erties & Val	lues Prev	view	81 Scans		
+   -	-   -	4   T				
Order	Property	Name	١	/alues to Sweep	# of Values	
°o 1	RXTERM	ı -	w i	100 mV,550 mV,1100 mV	3	
°o 2	TXDIFFS	WING	• 1	269 mV (0000),741 mV (0111),1119 mV (1111)	3	
<b>%</b> 3	TXPOST		<b>v</b> (	0.00 dB (00000),4.08 dB (01111),12.96 dB (11111)	<sup>7</sup> 3	
°s 4	TXPRE		<b>v</b> (	0.00 dB (00000),4.08 dB (01111),6.02 dB (11111)	3	
	after applyin	g Settings fo			r 3	icel

Because here are four different Sweep Properties and each of these properties has three different values (as seen in the Values to Sweep column), a total number of 81 sweep tests are carried out. The Scans window shows the results of all the scans that have been done for the selected link.

CAUTION! Since there are 81 scans to be done, it could be a few minutes before all the scans are

	con	complete.											
Tcl Console	Messages	Serial	I/O Links Serial I/O Scans	×									? _ 0 0
Q													. = 0 0
Name		Link	Link Settings		Reset RX	Scan Type	Status	Progress	Open Area	Open UI %	Horz I	ncr	Horz Range
🔸 🗎 Scans (	4)												
Sweep i	0 (81)					2d_full_eye	Done				8	~	-0.500 UI to 0.500 UI 🗸
Swee	ep 0 - Scan 2		RXTERM (100 mV) TXDIFFS	WING (269 mV (0000)) TXPOST (0.00 dB (00000)) TXPRE (.		2d_full_eye	Done	100%	10176	77.78	8	~	-0.500 UI to 0.500 UI 🗸
C Ourse	0.0000.0		DYTEDM (400 m) 0 TYDIEEC	NAME (2000 WW/ (00000) TYPOOT (0.00 4D (000000) TYPOF (		04 641 444	Dene	1000	10040	77.70	0		0.5001846.0.50018

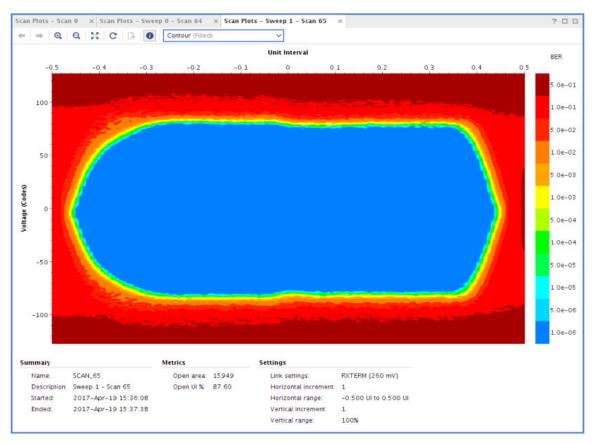
Sweep 0 (81)		2d_full_eye	Done				8	~	-0.500 UI to 0.500 UI 🗸
Sweep 0 - Scan 2	RXTERM {100 mV} TXDIFFSWING (269 mV (0000)) TXPOST {0.00 dB (00000)} TXPRE {	2d_full_eye	Done	100%	10176	77.78	8	$\sim$	-0.500 UI to 0.500 UI 🗸
Sweep 0 - Scan 3	RXTERM (100 mV) TXDIFFSWING (269 mV (0000)) TXPOST (0.00 dB (00000)) TXPRE (	2d_full_eye	Done	100%	10240	77.78	8	$\sim$	-0.500 UI to 0.500 UI 🗸
Sweep 0 - Scan 4	RXTERM (100 mV) TXDIFFSWING (269 mV (0000)) TXPOST (0.00 dB (00000)) TXPRE (	2d_full_eye	Done	100%	10112	77.78	8	$\sim$	-0.500 UI to 0.500 UI 🗸
Sweep 0 - Scan 5	RXTERM (100 mV) TXDIFFSWING (269 mV (0000)) TXPOST (4.08 dB (01111)) TXPRE (	2d_full_eye	Done	100%	10176	77.78	8	$\sim$	-0.500 UI to 0.500 UI 🗸
Sweep 0 - Scan 6	RXTERM (100 mV) TXDIFFSWING (269 mV (0000)) TXPOST (4.08 dB (01111)) TXPRE (	2d_full_eye	Done	100%	10240	77.78	8	$\sim$	-0.500 UI to 0.500 UI 🗸
Sweep 0 - Scan 7	RXTERM (100 mV) TXDIFFSWING (269 mV (0000)) TXPOST (4.08 dB (01111)) TXPRE (	2d_full_eye	Done	100%	10240	77.78	8	$\sim$	-0.500 UI to 0.500 UI 🗸
Sweep 0 - Scan 8	RXTERM (100 mV) TXDIFFSWING (269 mV (0000)) TXPOST (12.96 dB (11111)) TXPRE	2d_full_eye	Done	100%	10112	77.78	8	~	-0.500 UI to 0.500 UI 🗸
📴 Sweep 0 - Scan 9	RXTERM {100 mV} TXDIFFSWING {269 mV (0000)} TXPOST {12.96 dB (11111)} TXPRE	2d_full_eye	Done	100%	10112	77.78	8	~	-0.500 UI to 0.500 UI 🗸
Sweep 0 - Scan 10	RXTERM {100 mV} TXDIFFSWING {269 mV (0000)} TXPOST {12.96 dB (11111)} TXPRE	2d_full_eye	Done	100%	10240	77.78	8	~	-0.500 UI to 0.500 UI 🗸
Sweep 0 - Scan 11	RXTERM {100 mV} TXDIFFSWING {741 mV (0111)} TXPOST {0.00 dB (00000)} TXPRE {	2d_full_eye	Done	100%	10240	77.78	8	~	-0.500 UI to 0.500 UI 🗸
Sweep 0 - Scan 12	RXTERM {100 mV} TXDIFFSWING {741 mV (0111)} TXPOST {0.00 dB (00000)} TXPRE {	2d full eye	Done	100%	10112	77.78	8	×	-0.500 UI to 0.500 UI 💙 🗸

To see the results of any of the scans that have been performed, highlight the scan, rightclick, and select **Display Scan Plots**.



Tcl Console Messag	es Serial I/O Links Serial I/O Scans ×
Q   ¥   ♦   ▶	
Name	Link Link Settings
> 🗁 Scans (4)	
~ 🗿 Sweep 0 (81)	
Sweep 0	Scan Properties Ctrl+E
Sweep C	Scan Properties Ctrl+E FSWING {269 mV (0000)} TXPOST {0.00 dB (000
🔄 Sweep ( 🕨	Run Sweep or Scan FFSWING (269 mV (0000)) TXPOST (0.00 dB (000
Sweep C	Stop Sweep or Scan FFSWING (269 mV (0000)) TXPOST (4.08 dB (011
🖸 Sweep 🕻 👩	Display Scan Plots FFSWING (269 mV (0000)) TXPOST (4.08 dB (011
🖸 Sweep (	Write Scan Data FFSWING {269 mV (0000)} TXPOST {4.08 dB (011
Sweep C	FSWING {269 mV (0000)} TXPOST {12.96 dB (11
Sweep C	FFSWING {269 mV (0000)} TXPOST {12.96 dB (11
Sweep C	Apply Link Settings FSWING {269 mV (0000)} TXPOST {12.96 dB (11
🔄 Sweep ( 🗙	Delete Delete FFSWING {741 mV (0111)} TXPOST {0.00 dB (000
Sweep C	Export to Spreadsheet FSWING {741 mV (0111)} TXPOST {0.00 dB (000

#### The Scan Plots window opens showing the details of the scan performed.







Lab 9

## Using the Vivado ILA Core to Debug JTAG-AXI Transactions

This lab illustrates how to insert an ILA core into the JTAG to AXI Master IP core example design, using the ILA's advanced trigger and capture capabilities.

#### What is the JTAG to AXI Master IP core?

The LogiCORE<sup>™</sup>LogiCORE IP JTAG-AXI core is a customizable core that can generate AXI transactions and drive AXI signals internal to the FPGA at run-time. This supports all memory-mapped AXI interfaces (except AXI4-Stream) and Lite protocol and can be selected using a parameter. The width of the AXI data bus is customizable. This IP can drive any AXI4-Lite or Memory-Mapped Slave directly. It can also be connected as master to the interconnect. Run-time interaction with this core requires the use of the Vivado<sup>®</sup> logic analyzer feature.

#### **Key Features**

- AXI4 master interface
- Option to select AXI4 and AXI4-Lite interfaces
- User controllable AXI read and write enable
- User Selectable AXI data width: 32 and 64
- Vivado Integrated Logic Analyzer Tcl Console interface to interact with hardware

#### **Additional Documentation**

JTAG to AXI Master LogiCORE IP Product Guide (PG174) contains additional information

## **Design Description**

This section has three steps as follows:

- 1. Creating a simple design in IP integrator that includes a System ILA and JTAG-to-AXI master.
- 2. Programming the Kintex<sup>®</sup>-7 FPGA KC705 Evaluation Kit Base Board and interacting with the JTAG to AXI Master IP core.



3. Using the ILA Advanced Trigger Feature to Trigger on an AXI Read Transaction.

## Step 1: Creating a New Vivado Project and Generating the IP Integrator Design with JTAG-to-AXI and System ILA

To create a project, use the New Project wizard to name the project, add RTL source files and constraints, and specify the target device.

- 1. Invoke the Vivado<sup>®</sup> IDE.
- 2. In the Quick Start tab, click Create Project to start the New Project wizard. Click Next.
- 3. In the Project Name page, name the new project jtag\_2\_axi\_tutorial and provide the project location (C:/jtag\_2\_axi\_tutorial). Ensure that Create Project Subdirectory is selected. Click Next.
- 4. In the Project Type page, specify the Type of Project to create as RTL Project. Ensure that Do not specify sources at this time is checked. Click **Next**.
- 5. In the Default Part page, choose **Boards** and choose the **Kintex-7 KC705 Evaluation Platform**. Click **Next**.



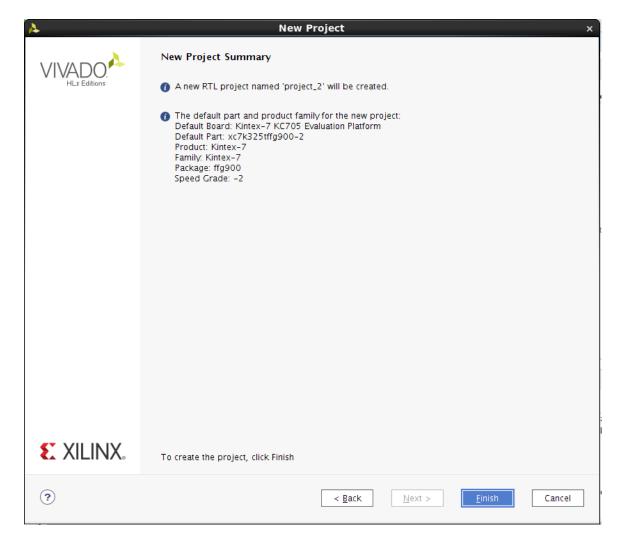


Parts	Boards			
Reset A	ll Filters			
Vendor:	All 🗸 Name: All			
Search:	Q	~		
Display		Preview	Vendor	File 1
Add Da	aughter Card Connections		xilinx.com	1.4
	-7 KC705 Evaluation Platform aughter Card Connections		xilinx.com	1.5
	UltraScale KCU105 Evaluation Platform aughter Card Connections		xilinx.com	1.4

6. In the New Project Summary page, click **Finish**.







7. In the leftmost panel of the Flow Navigator under Project Manager, click **Create Block Diagram**. A dialog box appears that allows you to specify a block diagram name. You can choose to specify a custom name or take the default. Click **OK**.





<u>File Edit Flow Tools Report</u>	s <u>W</u> indow La⊻out	View Help	Q+ Quick	Access														F	Ready
🖕 🔸 🖉 🖻 🖄 🔺 🕨	III 🗘 Σ 🖄	11 ×															II De	fault Layout	~
Flow Navigator 😤 🖨 🕈	PROJECT MANA	ER - project_2	2																?
PROJECT MANAGER	Sources			? _		i X	Proje	ct Summ	ary									? [	) C X
Settings	9	+ 2	•			ø	Over	view	Dashboard										
Add Sources	Desian So		÷ ·																
Language Templates		> Constraints					Setti	ings E	dit										
👎 IP Catalog		Sources					Proj	ect name	e: 1	project_2									
		□ sim_1 > □ Utility Sources						ect locat		/home/mpiazza/p	project_	2							
IP INTEGRATOR	> 🚍 Utility Sour	ces						duct fami ect part:	·	Kintex-7 Kintex-7 KC705 E	Voluetia	n Plat	larm (vc)	14225+66	000.3				
Create Block Design								module		Not defined	valualit	ni riau	unn (xc)	KSZDUIY	900-2	,			
Open Block Design	Hierarchy Li	oraries Cor	npile Order				Targ	get langu	age:	/erilog									
Generate Block Design	Properties			? _		i ×	Simu	ulator lan	iguage:	Mixed									
SIMULATION				-															
Run Simulation							Boar	rd Part											
							Dis	play nam	ie: Ki	ntex-7 KC705 Ev	aluation	Platfo	rm						
<ul> <li>RTL ANALYSIS</li> </ul>		elect on object	to see propert	loc			Board part name: xilinx: com:kc705;part0:1.5 Connectors: No connections Repository path: /proj/xbuilds/2018.3_0815_0946/installs/lin64/Vivado/2018.3/												
> Open Elaborated Design		elect all objec	to see propert	ies.										(data /bear	dr/board file	10000			
							1101			www.viliny.com/kcT		5_054	ro/instan	5/1110-4/1	ivau0/.	2010.5	(data) boai	us/board_nie:	
<ul> <li>SYNTHESIS</li> </ul>							<										_		>
Run Synthesis	Tcl Console	Messages	Log Reports	De	sign R	uns >	< l											?.	
> Open Synthesized Design	Q	€   ≪	▶   ≫   +	%															
<ul> <li>IMPLEMENTATION</li> </ul>	Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Powe	r Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy	
Run Implementation	∨ ▷ synth_1	constrs_1	Not started															Vivado Synth	
> Open Implemented Design	▷ impl_1	constrs_1	Not started															Vivado Impl	ementati
<ul> <li>open impremented oresign</li> </ul>																			
PROGRAM AND DEBUG																			
👫 Generate Bitstream																			
> Open Hardware Manager																			

8. In the far right of the window is an empty block diagram design window labeled Diagram. Click the + sign in the middle of the pane or the + toolbar button to bring up a search window. In the Search field, type "JTAG to AXI" and double-click it to add the JTAG to AXI Master to the block diagram.





Board ? _ 🗆 🖆 Diagram	
<ul> <li></li></ul>	Q   ≍   ≑ <mark>  +   <sup>1</sup> ∖   ≫   ⊠   ★</mark>   C   थ   :
	-
Search: Q- JTAG to AXI (1 match)	1
👎 JTAG to AXI Master	
	This design is empty. Press the ∔ button to add IP.
roperties	
Repoi	
shing IF	
ser IP r	
2d Vivac 1 = 00:0	<pre>46/installs/lin64/Vivado/2018.3/data/ip'.     = 6768.984 ; gain = 7.852 ; free physical = 574</pre>
1X. Comtk	( = 0,00.004 , gain = 7.052 , free physical = 5.4
<pre>ect_2/pr</pre>	\d>
: source ENTER to select, ESC to cancel, Ctrl+Q for IP details	

9. The JTAG to AXI Master core appears on the IP integrator canvas. Double-click the core to view the Customization dialog. Review the available settings and click **OK** to accept the default core settings.





4	Re-customize	IP		×
JTAG to AXI Master (1.2)				A
1 Documentation 📄 IP Location				
Show disabled ports	Component Name jtag_axi	.0		
	AXI Protocol	AXI4	~	
	AXI Address Width	32	~	
	AXI Data Width	32	~	
	AXI ID Width	1	0 [1 -	4]
	AXI4 Burst Type Support	ALL BURST TYPES	~	
acik M_AXI +	Write Transaction Queue Le	ength 1	0 [1 -	16]
• aresetn	Read Transaction Queue Le	ength 1	0 [1 -	16]
			ОК	Cancel

- 10. Following the same process from the previous step, add the additional IP to the block diagram: AXI BRAM controller and Block Memory Generator. This creates a design using a simple AXI infrastructure to create AXI transactions that demonstrate the debugging capabilities of the System ILA core.
- 11. Before continuing, you need to customize AXI BRAM Controller and Block Memory Generator. Begin by locating the AXI BRAM Controller in the block diagram canvas and double-clicking on it. This invokes the Customization Dialog for the IP. Locate the Number of BRAM interfaces and set the value to 1. Click **OK**.





A	Re-custo	mize IP			×
AXI BRAM Controller (4.0)					4
1 Documentation 🕒 IP Location					
Show disabled ports	Component Name axi_brar	m_ctrl_0		1	
	AXI Protocol Data Width		AXI4 ~	]	
	Memory Depth (Auto)		8192 ~		
	ID Width (Auto)	XI Narrow Bursts	0 ~ Yes ~	]	
+ S_AXI s_axi_acik BRAM_PORTA +	BRAM Options BRAM_INSTANCE (Au Number of BRAM inter		~ ~		-
	ECC Options				
	Enable ECC	No 🗸			
	ECC TYPE	Hamming 🗸 🗸			
	Enable Fault Injection	No v			
	ECC Reset Value	0 ~			
				ОК (	Iancel

12. Next, locate the Block Memory Generator in the block diagram and double-click as in the previous step to invoke the Customization dialog. Clear Enable Safety Circuit check box. Click **OK**.





umentation 📄 IP Location	
mbol Power Estimation	Component Name blk_mem_gen_0
ow disabled ports	Basic Port A Options Other Options Summary
	Pipeline Stages within Mux 0 V Mux Size: 2x1
	Memory Initialization
	Coe File no_coe_file_loaded
	Mem File no_mem_loaded
	Fill Remaining Memory Locations
🛛 🕂 BRAM_PORTA	Remaining Memory Locations (Hex) 0
	Structural/UniSim Simulation Model Options
	Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.
	Collision Warnings All
	Behavioral Simulation Model Options
	Disable Collision Warnings
	Safety logic to minimize BRAM data corruption

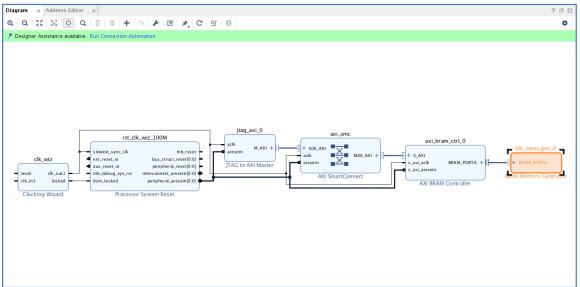
13. At this point the design should look like the following figure.





Diagram × Address Editor ×	265
$\textcircled{0} \bigcirc \textcircled{0} \bigcirc \fbox{0} \bigcirc \textcircled{0} \bigcirc \textcircled{1} \bigcirc @{1} @{1} \bigcirc @{1} @{1} @{1} @{1} @{1} @{1} @{1} @{1}$	•
axi_bram_ctrl_0	
jtag_axi_0	
aclk M_AXI + S_AXI s_axi_aclk BRAM_PORTA +	
• aresetn	
JTAG to AXI Master AXI BRAM Controller	

14. Notice the green banner indicating that Designer Assistance is available at the top of the block diagram canvas. Click the **Run Connection Automation** button on this banner. When the Connection Automation window appears, click the radio button for All Automation, then click **OK**.



15. Notice, that the Clocking Wizard and Processor System Reset as well as an AXI SmartConnect are auto-inserted into the design. Also, take note that the Clocking Wizard clock and reset inputs are not connected and the Run Connection Automation banner persists. These inputs will be connected to physical input ports on the FPGA, wired to buttons on the KC705 board though customization of the Clocking Wizard.



16. Invoke the Customization Dialog for the Clocking Wizard by double-clicking the IP in the block diagram canvas. When the dialog appears, set CLKIN\_1 to sys\_diff\_clk and EXT\_RESET\_\_IN to reset. Click **OK**.

**Note:** It is not necessary to add constraints for these ports because the project has been generated using an evaluation board as the target and the IP allows the constraint information to be selected with the <code>sys\_diff\_clk</code>.

ocking Wizard (6.0)			, e e e e e e e e e e e e e e e e e e e
ocumentation 🕒 IP Location			
Symbol Resource	Component Name clk_wiz		
Show disabled ports	Board Clocking Options Output Clock	cks MMCM Settings Summary	
	Associate IP interface with board interface		
	IP Interface	Board Interface	
	CLK_IN1	sys diff clock	*
	CLK_IN2	Custom	•
	EXT_RESET_IN	reset	v
	Clear Board Parameters		
+ CLK_IN1_D clk_out1 -			
reset locked			

- 17. Just as before, locate the green banner indicating that Designer Assistance is Available and click **Run Connection Automation**. When the Run Connection Automation dialog appears select the button for All Automation. Click **OK**.
- 18. Now, sys\_diff\_clk and reset are connected to external ports. Examine the connectivity of the design and notice that it might be necessary to monitor AXI transactions between the JTAG to AXI master and the AXI BRAM Controller slave. This is possible if a System ILA is added to probe the AXI bus between the AXI BRAM Controller and the JTAG to AXI master.



Diagram × Address Editor ×	? & Ľ
	۰
reset ck_wiz_town_cik bi_struct.exet(0) bi_str	_mem_gen_0 AML_RORTA Remory Generator

19. To add a System ILA to the design, click the Add IP (+) button as in previous steps. Search for System ILA, and double click to add it to the block diagram. When it appears in the block diagram canvas, double-click on it to invoke the Customization Dialog. Ensure that both Capture Control and Advanced Trigger are selected. Also, set the Number of Comparators to the value 3. Click **OK**.



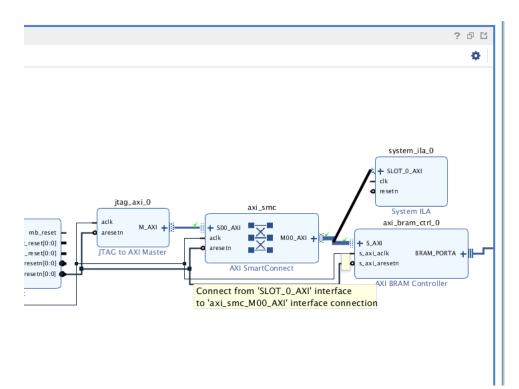


Δ	Re-customize IP	>
System ILA (1.1)		1
1 Documentation 📄 IP Location		
IP Symbol Resources	Component Name system_ila_0	
BRAM	To configure more than 64 probe ports use Vivado Tcl Console	
Resource Estimates	General Options Interface Options	
100.0	- Monitor Type	
90.0	Monitor Type INTERFACE V	
80.0	Number of Interface Slots 1	
8 60.0 E 50.0	Sample Data Depth 1024 V	
	Same Number of Comparators for All Probe Ports	
40.0	Number of Comparators 3 V	
30.0	Trigger Out Port	
20.0	Trigger In Port	
0.0 1.0	Input Pipe Stages 0 ~	
0.0 1.0	Trigger And Storage Settings	
	Capture Control	
Resource Usage BRAM Slice: 5	Advanced Trigger	
DRAM SILE D		
	OK Can	icel

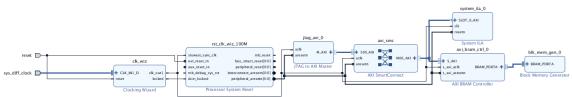
20. Now, make a connection between the System ILA SLOT\_0\_AXI port and the S\_AXI port on the AXI BRAM Controller. Do this by clicking on the SLOT\_0\_AXI port and clicking again on the S\_AXI port on the AXI BRAM Controller.



## AMD7 XILINX



21. When the Run Connection Automation banner appears, click it and select **All Automation**. Then click **OK**. Notice that the clk and resetn ports on the System ILA are connected to the AXI clock and the AXI reset.



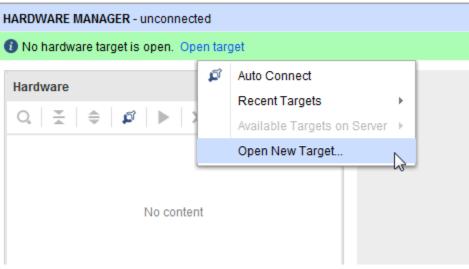
- 22. In the upper left side of the Vivado IDE, click File  $\rightarrow$  Save Block Design. Select File  $\rightarrow$  Close Block Design in the same menu to close the block design.
- 23. In the sources window, right-click on **design\_1 block design** and select **Create HDL Wrapper**. Allow Vivado IDE to manage the wrapper, and click **OK**.
- 24. In the Flow Navigator on the left side of the Vivado IDE, click Generate Bitstream.
- 25. Click **OK** to implement the design.
- 26. Wait until the Vivado Status window shows write\_bitstream complete.
- 27. In the Bitstream Generation Completed dialog, select **Open Hardware Manager**, and click **OK**.



Bitstream Generation Completed ×
i Bitstream Generation successfully completed.
Next
O Open Implemented Design
○ <u>V</u> iew Reports
⊙ Open <u>H</u> ardware Manager
O Generate Memory Configuration File
Don't show this dialog again
OK Cancel

# Step 2: Program the KC705 Board and Interact with the JTAG to AXI Master Core

- 1. Connect your KC705 board's USB-JTAG interface to a machine that has Vivado<sup>®</sup> IDE and cable drivers installed and power up the board.
- 2. The Hardware Manager window opens. Click **Open New Target**. The Open New Hardware Target dialog opens.



3. In the Connect to field choose **Local server**, and click **Next**.





🔥 Open New	Hardware Target	×
Select local or r	erver Settings remote hardware server, then configure the host name and port settings. Use Local server if the target is attached to the otherwise, use Remote server.	A
<u>C</u> onnect to:	Local server (target is on local machine)	
Click Next to	launch and/or connect to the hw_server (port 3121) application on the local machine.	
?	< Back Next > Einish Ca	ncel

*Note*: Depending on your connection speed, this may take about 10 to 15 seconds.

4. If there is more than one target connected to the hardware server, you see multiple entries in the Select **Hardware Target** page. In this tutorial, there is only one target as shown in the following figure. Leave these settings at their default values, and click **Next**.





🍌 Open Ne	ew Ha	ardware Targe	et				<b>X</b>
Select a har	dware					riate JTAG clock ( select a different ta	
Hardware	e <u>T</u> arg	ets					
Туре		Name		JTAG CIO	ock Frequency		
📓 xilinx	_tcf	Xilinx/Port_#000	)3.Hub_#0004	6000000	· · · ·		
			Add Xili	nx Virtual C	able (XVC)		
Hardware	e <u>D</u> evi	c <b>es</b> (for unknow	n devices, spe	cify the Inst	truction Registe	er (IR) length)	
Name		ID Code	IR Length				
🙂 xc7k3	325t_0	33651093	6				
Hardware	serve	er: localhost:312	1				
?			< <u>E</u>	ack	<u>N</u> ext ≻	<u>F</u> inish	Cancel

- 5. Leave these settings at their default values as shown. Click Next.
- 6. In the Open Hardware Target Summary page, click **Finish** as shown in the following figure.



## AMDA XILINX

🍐 Open New Hardware	a Target	
	Open Hardware Target Summary	
VIVADO. HLx Editions	Hardware Server Settings:         • Server: localhost3121	
	<ul> <li>Target Settings:         <ul> <li>Target: xilinx_tdtXilinx/Port_#0003.Hub_#0004</li> <li>Frequency: 6000000</li> </ul> </li> </ul>	
EXILINX ALL PROGRAMMABLE.	To connect to the hardware described above, click Finish	
?	< <u>B</u> ack <u>Next</u> > <u>Finish</u> Cancel	

Wait for the connection to the hardware to complete. After the connection to the hardware target is made, the Hardware dialog shown in the following figure opens.

**Note:** The Hardware tab in the Debug view shows the hardware target and XC7K325T device that was detected in the JTAG chain.

Hardware	? _ 🗆 🗹	×
$Q \mid \underbrace{\star} \mid \diamondsuit \mid \varnothing \mid \models \mid \gg \mid \blacksquare \mid$		٥
Name	Status	
V localhost (1)	Connected	
✓ ✓ ✓ ✓ × xilinx_tcf/Xilinx/Port_#0003.Hu	Open	
ki k	Not programmed	
📴 XADC (System Monitor)		

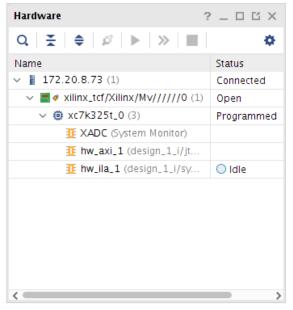
- 7. Next, program the previously created XC7K325T device using the .bit bitstream file by right-clicking the XC7K325T device, and selecting **Program Device** as shown in the following figure.
- 8. In the Program Device dialog verify that the .bit file is correct for the lab that you are working on. Click **Program** to program the device.



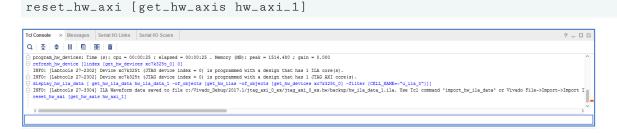
🔶 Program Device		×		
Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.				
Bitstre <u>a</u> m file:	1/jtag_axi_0_ex/jtag_axi_0_ex.runs/impl_1/example_jtag_axi_0.bit			
Debu <u>q</u> probes file:	1/jtag_axi_0_ex/jtag_axi_0_ex.runs/impl_1/example_jtag_axi_0.ltx 😒			
Cancel     Cancel				

Note: Wait for the program device operation to complete. This can take a few minutes.

9. Verify that the JTAG to AXI Master and ILA cores are detected by locating the hw\_axi\_1 and hw\_ila\_1 instances in the Hardware Manager window.



10. You can communicate with the JTAG to AXI Master core via Tcl commands only. You can issue AXI read and write transactions using the run\_hw\_axi command. However, before issuing these transactions, it is important to reset the JTAG to AXI Master core. Because the aresetn input port of the jtag\_axi\_0 core instance is not connected to anything, you need to use the following Tcl commands to reset the core:





11. The next step is to create a 4-word AXI burst transaction to write to the first four locations of the BRAM:

set wt [create\_hw\_axi\_txn write\_txn [get\_hw\_axis hw\_axi\_1] -type WRITE address C0000000 -len 128 -data {44444444\_33333333\_22222222\_11111111]}

where:

- write\_txn is the name of the transaction.
- [get\_hw\_axis hw\_axi\_1] returns the hw\_axi\_1 object.
- -address C0000000 is the start address.
- -len 128 sets the AXI burst length to 128 words
- -data {4444444\_3333333\_2222222\_11111111} is the data to be written.

**Note:** The data direction is MSB to the left (i.e., address 3) and LSB to the right (i.e., address 0). Also note that the data will be repeated from the LSB to the MSB to fill up the entire burst.

12. The next step is to set up a 128-word AXI burst transaction to read the contents of the first four locations of the AXI-BRAM core:

```
set rt [create_hw_axi_txn read_txn [get_hw_axis hw_axi_1] -type READ -
address C0000000 -len 128]
```

where:

- read\_txn is the name of the transaction.
- [get\_hw\_axis hw\_axi\_1] returns the hw\_axi\_1 object.
- -address C0000000 is the start address.
- -len 128 sets the AXI burst length to 4 words.
- 13. After creating the transaction, you can run it as a write transaction using the run\_hw\_axi command:

run\_hw\_axi \$wt

This command should return the following:

INFO: [Labtools 27-147] : WRITE DATA is : 444444433333332222222211111111...

14. After creating the transaction, you can run it as a read transaction using the run\_hw\_axi command:

run\_hw\_axi \$rt

This command should return the following:

```
INFO: [Labtools 27-147] : READ DATA is : 444444433333333222222221111111...
```



## AMD7 XILINX

## Step 3: Using ILA Advanced Trigger Feature to Trigger on an AXI Read Transaction

- 1. In the ILA hw\_ila\_1 dashboard, locate the Trigger Mode Settings area and set Trigger mode to ADVANCED\_ONLY.
- 2. In the Capture Mode Settings area, set the Trigger position to **512**.
- 3. In the Trigger State Machine area click the **Create new trigger state machine** link.

ettings - hw_ila_1	? _ 🗆 X	Trigger Setup - hw_ila_1		- hw_ila_1 Status - h	w_ila_1 × 1	? _
rigger Mode Settings		শ্ৰ 🕨 💌	<b>1</b>			
Trigger mode: ADVANCED_ONLY V		Core status				
Trigger state machine: BASIC_ONLY ADVANCED_ONLY		Idle	Pre-Trigger	Waiting for Trigger	Post-Trigger	
apture Mode Settings		Trigger State Machine				
Capture mode: ALWAYS V		Flag O	Flag 1	Flag 2	Flag 3	
Number of windows: 1 [1 - 1024]		Trigger state: 0				
Window data depth: 1024 🗸 [1 - 1024]		Capture status Window 1 of 1	Window sample	0 of 1024 Total sam	ple 0 of 1024	
Trigger position in window: 512 [0 - 1023]		Idle	Idle	1	dle	
eneral Settings						
Refresh rate: 500 ms						
		<				

4. In the New Trigger State Machine File dialog box, set the name of the state machine script to **txns.tsm**.

ika 🔥 New Trigo	ger State Machine File	
Save <u>I</u> n:	jtag_axi_0_ex	✓ Ø S = ¥ A Z
	)_ex.hw )_ex.ioplanning )_ex.ip_user_files )_ex.runs )_ex.sim	Recent Directories Cr/livado_Debug/2017.1/jtag_axi_0_ex/jtag_axi_0_ex.runs/impl_1 v File Preview Select a file to preview.
File <u>n</u> ame:	bins	
Files of type:	Trigger State Machine Files (.tsm)	~
		Save





5. A basic template of the trigger state machine script is displayed in the Trigger State Machine gadget. Expand the trigger state machine gadget in the ILA dashboard. Copy the script below after line 17 of the state machine script and save the file.

```
# The "wait_for_arvalid" state is used to detect the start
# of the read address phase of the AXI transaction which
# is indicated by the axi_arvalid signal equal to '1'
state wait_for_arvalid:
    if (design_1_i/system_ila_0/U0/net_slot_0_axi_arvalid == 1'b1) then
      goto wait_for_rready;
    else
      goto wait_for_arvalid;
    endif
#
# The "wait_for_rready" state is used to detect the start
# of the read data phase of the AXI transaction which
# is indicated by the axi_rready signal equal to '1'
state wait_for_rready:
  if (design_1_i/system_ila_0/U0/net_slot_0_axi_rready == 1'b1) then
    goto wait_for_rlast;
  else
    goto wait_for_rready;
  endif
# The "wait_for_rlast" state is used to detect the end
\# of the read data phase of the AXI transaction which
# is indicated by the axi_rlast signal equal to '1'.
# Once the end of the data phase is detected, the ILA core
# will trigger.
state wait_for_rlast:
  if (design_1_i/system_ila_0/U0/net_slot_0_axi_rlast == 1'b1) then
   trigger;
  else
   goto wait_for_rlast;
  endif
```

Note: Use the state machine to detect the various phases of an AXI read transaction:

- Beginning of the read address phase
- Beginning of the read data phase
- End of the read data phase
- 6. Arm the trigger of the ILA by right-clicking the **hw\_ila\_1 core** in the Hardware Manager window and selecting **Run Trigger**.

Hardware				? _ 🗆 🖸	×	exam	ple_jtag_axi_0.v
Q ₹ ♦	ø 🕨	<b>»</b>			•	v	Vaveform - hw_ila
Name			Status			s a	
🗸 📱 localhost (1	)		Connected			oard Options	Q + -
✓ ✓ xilinx_t	cf/Xilinx/Port_#	0003.H	lu Open			ğ	ILA Status: Idle
✓	25t_0 (3)		Programm	ha		oar 🗌	Nome
🗿 XA	DC (System N		Hardware Device	Properties		Ctrl	+E axi_araddr[31
📴 hw	/_axi_1 (AXI)		Program Device				axi_arburst[1:
📴 hw	/_ila_1 (u_ila_		Verify Device				axi_arcache[3
			Run Trigger				axi_arlen[7:0]
		-	Run Trigger Imme	diate			axi_arprot[2:0
			Stop Trigger	anato			axi_awaddr[3 axi_arqos[3:0
		_					axi_arsize[2:0
			Enable Auto Re-tri				axi_awburst[1
			Disable Auto Re-tr	rigger			axi_awcache[
			Create User Defin	ed Probe			axi_awlen[7:0
		С	Refresh Device				
			Add Configuration	Memory Device			
		1	Boot from Configu	ration Memory (	Device		
Hardware Device	Properties		Program BBR Key	·			igs - hw_ila_1
xc7k325t_0			Clear BBR Key				jer Mode Sett
Name:	xc7k325t_(		Program eFUSE F	Registers			rigger mode:
Part:	xc7k325t		Export to Spreads	heet			Trigger state m
ID code:	33651093						
IR length:	6						Capture Mode Set

7. In the Trigger Capture Status window, note that the ILA core is waiting for the trigger to occur, and that the trigger state machine is in the wait\_for\_a\_valid state. Note that the pre-trigger capture of 512 samples has completed successfully:

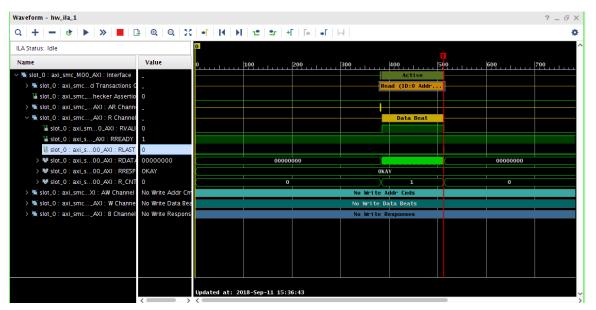




8. In the Tcl Console, run the read transaction that you set up in the previous section of this tutorial.

run\_hw\_axi \$rt

*Note*: The ILA core has triggered and the trigger mark is on the sample where the axi\_rlast signal is equal to '1', just as the trigger state machine program intended.



## Lab 10

# Using the Vivado Serial Analyzer to Debug PS-GTR Serial Links

IBERT UltraScale+<sup>™</sup> PS-GTR (IBERT PS-GTR) transceiver can be used to evaluate and monitor PS-GTR transceivers in Zynq<sup>®</sup> UltraScale+<sup>™</sup> MPSoC devices. With this feature, you can accomplish these tasks:

- Perform eye scans with user data
- Change PS-GTR settings
- View link status
- Check the "lock" status of all phase-locked loops (PLLs) used by all PS-GTR lanes

IBERT PS-GTR transceiver does not provide these capabilities:

- Perform eye scans with raw pseudo-random binary sequence (PRBS) data patterns
- Measure Bit Error Ratio (no bit or error counters)

Note that this solution is purely software based, meaning that no IP or logic is required in the programmable logic (PL) of the device. This documentation guides you through the setup of the PS-GTR Transceivers by creating a first stage boot loader (FSBL). It then demonstrates how to load the FSBL into the Zynq UltraScale+ MPSoC and use IBERT PS-GTR.



**TIP:** This is a supported feature in Vivado<sup>®</sup> Design Suite 2017.2 and above.

## **IBERT PS-GTR Flow**

The IBERT PS-GTR Bring-up and subsequent EyeScan involves three different components:

- 1. Generating Zynq UltraScale+ MPSoC PS Xilinx<sup>®</sup> Support Archive (XSA) file from the Vivado<sup>®</sup> tool after configuring the PS-GTR.
- 2. Using the Vitis<sup>™</sup> Xilinx Software Command-line Tool (XSCT) flow to generate a FSBL file using the XSA file.
- 3. Using the FSBL file with Vivado Serial I/O Analyzer to bring up IBERT PS-GTR.



#### **Tools Required**

- Vivado
- Vitis
- XSCT (Part of the Vitis tool)

#### **Board/Part/Components Required**

- ZCU102 Rev 1.0 board
- XCZU9EG-FFVB1156 production device
- PCle<sup>®</sup>
  - A PCIe card which has at least x4 lanes
  - PCI Express 4x Male to PCIe 16x Female Riser Cable if PCIe card is larger than x4
- SATA
  - 。 SanDisk 128 GB SATA SSD Drive
  - 。 SATA connector cable
  - 4 Pin Molex to SATA Power Cable Adapter
- USB
  - 。 SanDisk Ultra 32 GB USB 3.0 Flash Drive
  - 。 USB 3.0 Type A Female to Micro Male Adapter

#### **Required Files**

- FSBL executable and linkable format file (ELF File) (Created using the following instructions) which configures the PS-GTR
- Configuration Bitstream File (Optional file that may be needed to custom configure the FPGA depending on the board setup)
- Tcl script to generate the FSBL and modify C-source for USB Support (when available)

#### Assumptions

- 1. FSBL should always target Cortex<sup>®</sup>-A53 processor as R5 (psu\_cortexr5\_0) is exclusively used by IBERT PS-GTR.
- 2. Physical devices such as SATA drive, PCIe card, etc. are needed for validation.

## Step 1: Generating a Zynq UltraScale+ MPSoC PS Xilinx Support Archive

- 1. Open the Vivado IDE.
- 2. Click Create Project, and click Next.

	New Project	×
HLx Editions	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.	λf
E XILINX.		
<b>?</b>	< <u>B</u> ack <u>Next</u> <u>Finish</u> Cance	el

- 3. Set your project name, and specify the project directory. Click Next.
- 4. Select Project type as RTL project.
- 5. Select do not specify sources at this time checked, then click Next.
- 6. To choose the board, click the board icon, and select **Zynq UltraScale+ ZCU102 Evaluation board**, with Board Rev 1.0. Click **Next**.



efault Part pose a default Xilinx part or board for your project.				
Parts   Boards				
Reset All Filters				Install/Update Boards
Vendor: All ~ Name: All			~	Board Rev: Latest 🗸
Search: Q.	Bassien		File Manalas	-
Display Name	Preview	Vendor	File Version	Part
Zyng UltraScale+ 2CU102 Evaluation Board Add Companion Card Connections		xilinx.com	3.4	xczu9eg-Ħvb1156-2-e
Zyng UltraScale+ ZCU104 Evaluation Board Add Companion Card Connections		xilinx.com	1.1	xczu7ev-ffvc1156-2-e
Zyng UltraScale+ ZCU106 Evaluation Platform Add Companion Card Connections	600	xilinx.com	2.6	xczu7ev-ffvc1156-2-e
Zyng UltraScale+ ZCU111 Evaluation Platform		xilinx.com	1.4	xczu28dr-ffvg1517-2-e
Xilinx Zyng UltraScale+ RFSoC ZCU1275 Characterization Ki		xilinx.com	1.0	xczu29dr-ffvf1760-2-e
				>

- 7. The project summary displays. To create the project, click **Finish**.
- 8. In the Flow Navigator, select **Create Block Design**. You can specify the design name and directory, but it is not necessary for a local project directory. Click **OK** to create the block design.





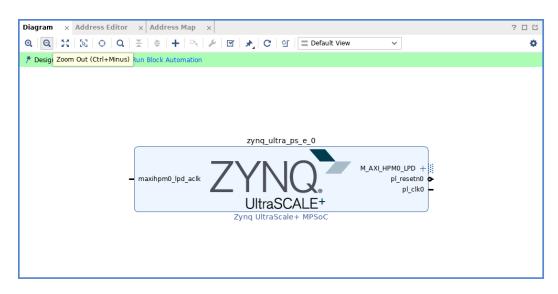
Flow Navigator 🗄 0 ?	PROJECT MANAGER - project_1											
V PROJECT MANAGER	Sources		7 _ 0 6 X	Project Sur	mmary							
Settings Add Sources	Q ₹ ♦ + 10 ● °		0	Overview	Dashboard							
Language Templates	Design Sources Constraints			Settings	Edit							
👎 IP Catalog				Project nar Project loc		roject_1 proj/xcoswr	nktg/mpia	izza/git/u	g936_d	esign_file	s/tmp/pro	ject_1
IP INTEGRATOR <u>Create Block Design</u>	>  Utility Sources			Product fa Project par Top modul	rt: Z	lyng UltraS lyng UltraS lot defined		J102 Eval	luation 8	Board (xc	zu9eg-ffv	b1156-
Open Block Design				Target lan		eriloa						
Generate Block Design				Create Block D	Design	×						
<ul> <li>SIMULATION</li> </ul>	Hierarchy Libraries Compile Order		Please specify name of block design.									
Run Simulation	Properties									5.04 T		
✓ RTL ANALYSIS			Design name:	design_1		0	le+ ZCU10 102:part0		ation Bo	ard		
> Open Elaborated Design			Directory: Specify source	et: Design		~	5					
✓ SYNTHESIS			?	o congr		_	WIP/2020.	2_0828_	0936/in	stalls/line	54/Vivado	/2020.2
Run Synthesis	Select an object to s	ee properties	C	· · ·	Car	ncel	le+ ZCU1	02 Evalua	ation Bo	ard		
> Open Synthesized Design				unanges								
<ul> <li>IMPLEMENTATION</li> </ul>				P						ter e te		
Run Implementation				<								
> Open Implemented Design	Tcl Console Messages Log Repo		ans ×									
Y PROGRAM AND DEBUG	Q ≚ ♦ 14 ≪ ▶ ≫ +											
👫 Generate Bitstream	Name         Constraints         Status           V         > synth 1         constrs 1         Not started	WNS TNS	WHS THS TPW	S Total Power	Failed Routes	LUT FF	BRAM	URAM	DSP	Start	Elapsed	Run S Vivado
> Open Hardware Manager	b impl 1 constrs 1 Not started											Vivado

9. An empty design diagram displays. Click the Add IP button to add IP. Select the IP based on the selected board (for the ZCU102 evaluation board, search for Zynq UltraScale+ MPSoC) and double-click the selected IP.

BLOCK DESIGN - design_1	
Sources Design x Signals Board ? _ [ ]	Diagram
Q   ¥   ⅓	Q       Q       Search: Q: zynq       Q       I match)         Imatch       Zynq UltraScale+ MPSoC       Imatch       Imatch         Imatch       Imatch       Imatch
Properties     ? _ □ □ ×       ←   ⇒   ♦       Select an object to see properties	
	ENTER to select, ESC to cancel, Ctrl+Q for IP details

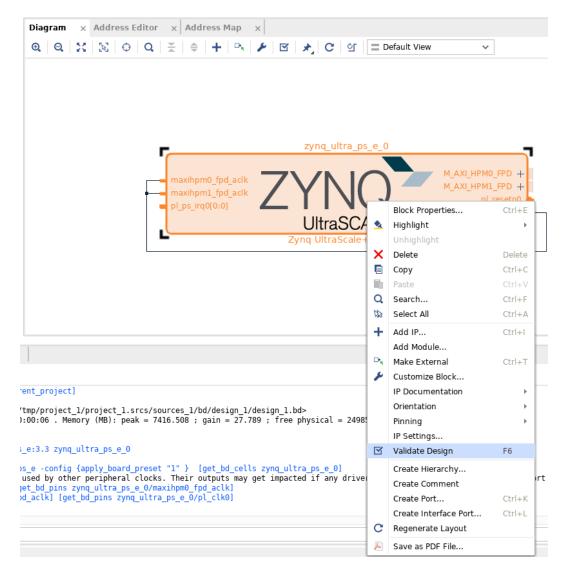
10. In the design diagram window, select **Run Block Automation**. Click **OK** to continue creating the ZCU102 design.





- 11. When the design diagram appears, use the following steps to validate the design:
  - a. Connect maxihpm0\_fpd\_aclk and maxihpm1\_fpd\_aclk together to pl\_clk0, as shown in the following figure.
    - i. Select maxihpm0\_fpd\_aclk and drag it to maxihpm1\_fpd\_aclk.
    - ii. Select maxihpm1\_fpd\_aclk and drag it to pl\_clk0.
  - b. Right-click the Zynq UltraScale+ MPSoC block and select **Validate Design** to validate the design. It will say validation successful. Click **OK**.





12. Customize the design by double-clicking the Zynq UltraScale+ MPSoC block and configuring the parameters. There are four valid GT configurations for ZCU102 board as shown in the following table.

SEL (S3,2,1,0)	ICM Settings (Lane 0,1,2,3)	PCIe Connector	DP Connector	USB Connector	SATA Connector
0000	PCIe.0, PCIe.1, PCIe.2, PCIe.3	PCIe Gen2 x4	N.C.	N.C.	N.C.
1111	DP.1, DP.0, USB, SATA	N.C.	DP.0, DP.1	USB0	SATA1
1100	PCIe.0, PCIe.1, USB, SATA	PCIe Gen2 x2	N.C.	USB0	SATA1
1110	PCIe.0, DP.0, USB, SATA	PCIe Gen2 x1	DP.0	USB0	SATA1

#### Table 2: Supported PS-GTR Connector Functionality





- 13. Select the settings based on your requirements by double-clicking the Zynq UltraScale+ MPSoC block to customize GT Lane configuration.
- 14. Select I/O Configuration → High Speed. Select one of the four combinations using the settings in the following screenshots.
  - a. PCle Display Port USB SATA (Default Vivado preset)

Zynq UltraScale+							
Documentation 🌣	Presets 🛛 📄 IP Location						
Page Navigator	I/O Configuration						
Switch To Adva	✓ MIO Voltage Standa	ard					
	Bank0 [MIO 0:25]	Bank1 [MIO 26:51]	Bank2 [MIO	52:77]	Bank3 [Dedi	cated]	
PS UltraScale+ Bloc	LVCMOS18 V	LVCMOS18 V	LVCMOS18	~	LVCMOS18	*	
I/O Configuration			210110010		210110010	-	
	← Q ≍ ≑ ●						
Clock Configuration							
DDR Configuration	Search: Q-						
	Peripheral	I/O		Signal		I/O Type	Drive Stren
PS-PL Configuration	> GEM						
	∨ USB						
	∨ USB0	1					
	> 🕑 USB 0	MIO 52 63	×				
	> 🕑 USB 3.0	GT Lane2	~				
	> USB1						
	> USB Reset	Boot Pin	~				
	∨ 🗹 PCIe						
	> Rootport Mode Res	et MIO 31	~				
	Reset Polarity	Active Low	~				
	Lane Selection	xl	~				
	PCIe Lane0	GT Lane0					
	<ul> <li>✓ ✓ Display Port</li> </ul>						
	> DPAUX	MIO 27 30	) v				
	> Lane Selection	Single Lowe	r v				
	<ul> <li>✓ ✓ SATA</li> </ul>						
	SATA Lane0						
	SATA Lanel	GT Lane3	~				
	SATA Lane     SATA Lane     SATA Lane	OT Lanes	¥				
	<				_		>

b. PCle-PCle - USB - SATA





		Re-customize	e IP				×
Zynq UltraScale+	⊢ MPSoC (3.3)						4
🗊 Documentation 🔅	Presets 🛛 🕞 IP Location						
Page Navigator	I/O Configuration						
Switch To Adva	MIO Voltage Standard						
PS UltraScale+ Blo					Bank3 [Dedic LVCMOS18	ated]	
I/O Configuration							
Clock Configuration	← Q ≭ ¢ ●						
DDR Configuration	<u>S</u> earch: Q-						
PS-PL Configuratio	Peripheral	I/O		Signal		I/O Type	Drive S
	> 🗹 USB 0	MIO 52 63	~				
	🗌 USB 3.0						
	∨ USB1						
	🗌 USB 1						
	USB 3.0						
	> USB Reset	Boot Pin	~				
	∨ 🖉 PCIe						
	> Rootport Mode Reset	MIO 31	~				
	Reset Polarity	Active Low	~				
	Lane Selection	x2	~				
	PCIe Lane0	GT Lane0					
	PCIe Lanel	GT Lanel					
	∨ 🗌 Display Port						
	DPAUX						
	Lane Selection						
	V 🗹 SATA	Image: Additional system of the system of					
	SATA Lane0						
	🗹 SATA Lanel	GT Lane3	~				
	Contractor Charles						`

c. Display Port - Display Port - USB - SATA





ynq UltraScale+	• MPSoC (3.3)	Re-customize IP				2		
Documentation 🔅	Presets 📄 IP Location							
Page Navigator	I/O Configuration							
Switch To Adva	MIO Voltage Standard							
	Bank0 [MIO 0:25] Bank	k1 [MIO 26:51] Bank2	[MIO 52:77]	Bank3 [Dedica	ated1			
S UltraScale+ Bloc		MOS18 V LVCM	-	LVCMOS18				
/O Configuration			0510 +	EVCHOSIS	•			
lock Configuration	← Q ¥ ≑ ●							
lock comgutation			_					
DR Configuration	Search: Q-	110				<b>D</b> : 01		
S-PL Configuration	Peripheral Low Speed	I/O	Signal		I/O Type	Drive Stre		
- · · · · · · · · · · · · · · · · ·	<ul> <li>High Speed</li> </ul>							
	> GEM							
	V USB							
	V USB0							
	> 🕑 USB 0	MIO 52 63	*					
	> 🖉 USB 3.0	GT Lane2	*					
	> USB1	OT Lanez	•					
	> USB Reset	Boot Pin	~					
		BOOL PIN	~					
	✓ PCle							
	Endpoint Mode Reset Lane Selection							
	✓ ✓ Display Port							
	> DPAUX	MIO 27 30	~					
	✓ Lane Selection	Dual Lower	*					
	DP Lane0	GT Lanel	-					
	DP Lanel	GT Lane0						
	<ul> <li>✓ ✓ SATA</li> </ul>							
	SATA Lane0							
	SATA Lanel	GT Lane3	~					

d. PCIe-PCIe - PCIe - PCIe (PCIe x4)



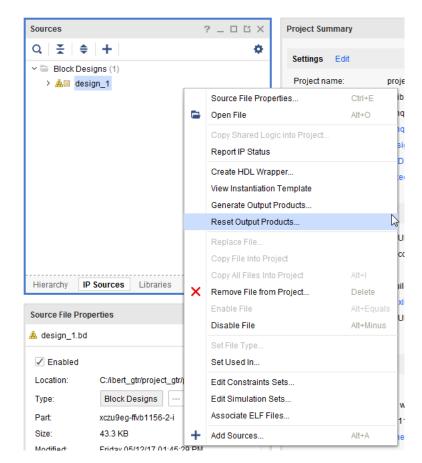


		Re-customize I	Р			×						
Zynq UltraScale+	+ MPSoC (3.3)					4						
🚺 Documentation 🔅	Presets 🕒 IP Location											
Page Navigator	I/O Configuration											
Switch To Adva	MIO Voltage Standard											
PS UltraScale+ Bloc		k1 [MIO 26:51] Ba		Bank3 [Dedicat	:ed]							
I/O Configuration	LVCMOS18 V LVC	MOSI8 V	CMOS18 V	LVCMO518	×							
Clock Configuration	← Q ≚ ≑ ●											
	Search: Q-											
DDR Configuration	Peripheral	I/O	Signal		I/O Type	Drive Strengt						
PS-PL Configuration	> Low Speed		5									
	✓ High Speed											
	> GEM											
	> USB											
	√ 🕑 PCle											
	> Rootport Mode Reset	MIO 31	~									
	Reset Polarity	Active Low	*									
	Lane Selection	x4	*									
	PCIe Lane0	GT Lane0										
	PCIe Lanel	GT Lanel										
	PCIe Lane2	GT Lane2										
	PCIe Lane3	GT Lane3										
	✓ Display Port											
	DPAUX											
	Lane Selection											
	✓ SATA											
	SATA Lane0											
	SATA Lanel											
	> Reference Clocks											
	<			_		>						
				0	к	Cancel						

- 15. Click **OK** when finished customizing the GT Lane configuration.
- 16. Do not click Run Block Automation again, even though the banner will reappear. If used, the customized values will reset.
- 17. Click the **Sources** tab on the top left of the Block Design window.
  - a. Under the Block Designs group, click **IP Sources**.
  - b. Right-click design\_1 and then click Create HDL Wrapper.



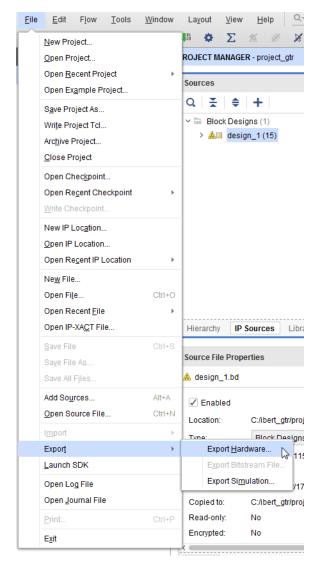




- 18. Leave the option Let Vivado manage wrapper and auto-update selected. Click **OK** in the dialog to create the HDL wrapper.
- 19. Right-click design\_1\_i in the IP Sources tab, and click Generate Output Products.
- 20. Click Generate to generate with the default options in the panel.
- 21. After the generation is complete, click **OK**.
- 22. Select File → Export → Export Hardware.



### AMD7 XILINX



- 23. In the Export Hardware Platform wizard, select a Fixed platform type. Click Next.
- 24. On the next page, select **Pre-Synthesis** for the platform output type.
- 25. Leave the XSA name as **design\_1\_wrapper**, and choose a location to store the exported XSA, preferably in a new directory.

## Step 2: Using Xilinx Software Command-line Tool Flow to Generate a First Stage Boot Loader

XSCT is an interactive and scriptable command-line interface to the Vitis tool. The XSCT flow requires running a Tcl script.





#### Generating Using the Xilinx Software Command-line Tool Automated Flow

To create a FSBL for the Cortex-A53 #0 (64-bit) automatically (and modify the xfsbl\_main.c/h files if a USB is present) using the provided script, use the following steps:

- 1. Copy the src/lab10/xsct\_create\_fsbl.tcl script to the directory where the XSA file is located. You can modify the Tcl script if you changed the default name of the XSA file in the Vivado tool. You can also change the script if the compiler options need to be different.
- 2. Open a terminal on Linux or command prompt on Windows.
- 3. Change directory into the directory where the XSA file is located.
- 4. Call xsct from the Vitis tool install area.

% xsct xsct\_create\_fsbl.tcl

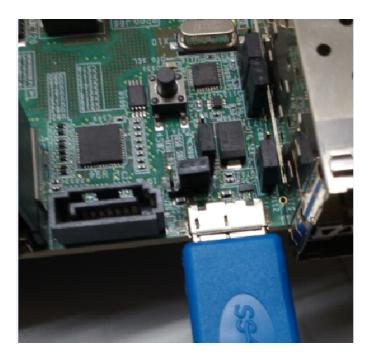
5. The location of the generated ELF File prints out when the script completes.

## Step 3: ZCU102 Board Settings

#### USB Jumper Setting Requirements for HOST Mode on ZCU102

- 1. Make sure the following jumpers are correctly set for USB to be in HOST mode (refer to *ZCU102 Evaluation Board User Guide* (UG1182)).
  - a. J7 ON
  - b. J113 1-2
  - c. J110 2-3
- 2. Refer to the following image for jumper settings on a ZCU102 Rev 1.0 board.





## Using FSBL with Serial I/O Analyzer to Bring Up IBERT PS-GTR

- 1. Connect all the physical devices such as SATA Drive, PCIe<sup>®</sup> card, and USB device based on your selection from the four valid GT configurations for ZCU102 prior to loading the FSBL. Hot swap or hot plug is not supported.
- 2. Open Vivado.
- 3. Open hardware manager and connect to a board with a Zynq UltraScale+ device. The following example shows connecting to a board on a remote machine, so hw\_server needs to be running on the remote machine before it can connect.





🔥 Open New Ha	rdware Target		×
Hardware Serv Select local or reme local machine; othe	ote hardware serve	r, then configure the host name and port settings. Use Local server if the target is attached to the s server.	4
<u>C</u> onnect to: R	emote server (targ	et is on remote machine)	
Remote Server			
<u>H</u> ost name:	ibert-0	◎ ∽	
<u>P</u> ort:	3121	[default is 3121]	
Click Next to lau	nch and/or connect	to the hw_server (port 3121) application on the remote machine 'lentinus14'.	
•		< <u>Back</u> <u>Next&gt;</u> Einish Car	ncel

4. Verify the ARM\_DAP is visible in the hardware device list and click **Next**, and then click **Finish**.

	target from the	list of availa			ppropriate JTAG clock (	TCK) frequency. I	f you do not see the	×
expected devices,	decrease the fre	equency or	select a different ta	irget.				
Hardware <u>T</u> arg	ets							
Туре	Name		JTAG Clock Freq	uency				
xilinx_tcf	Digilent/210308	BA11BFC	15000000	~				
			Add	Xilinx Vir	tual Cable (XVC)			
Hardware <u>D</u> evi	ces (for unknow	n devices,	specify the Instru	ction Re	gister (IR) length)			
Name	ID Code	IR Lengt	1					
xczu9_0	04738093	12						
@ arm_dap_	1 5BA00477	4						
Hardware serve	er ibert-0:3121							
Than and to both								
_								
?					< <u>B</u> ack	<u>N</u> ext >	<u>F</u> inish	Cancel

5. Right-click the **ARM\_DAP device** in the hardware tree and select **Configure IBERT GTR**.





Hardware	?	_ 🗆 🖒 ×			
$Q \mid \underbrace{\star} \mid \diamondsuit \mid \wp \mid \Vdash \mid \gg$		•			
Name		Status			
<ul> <li>ibert-0 (1)</li> </ul>	<ul> <li>ibert-0 (1)</li> </ul>				
✓ Ø xilinx_tcf/Digilent/21030	Open				
million xczu9_0 (1)	Not programm				
📴 SysMon (System Mo					
		NI/A			
📴 SysMon (System	Hard	Iware Device Propertie	s Ctrl+E		
	Cont	figure IBERT GTR			
< (	Refr	esh Device	~		
Hardware Device Properties	Expo	ort to Spreadsheet			
🗭 arm, dan, 1					

6. When the dialog box opens, you must provide the FSBL ELF file created in the previous steps and optionally a configuration file (a bitstream, if your design requires one). You can also reset the system before configuring with the Reset Zynq option checked. Click **OK** when done.

**Note:** The Reset Zynq option leaves the ARM\_DAP in a bad state on early versions of Zynq UltraScale+ devices (for example, ZU9EG es1). If that occurs, power cycle the board and keep the Reset Zynq option unchecked.

🔥 Configure IBERT (	GTR	×
A Zynq FSBL is required	d to configure IBERT GTR. The configuration file is optional.	A
<u>Z</u> ynq FSBL: <u>C</u> onfiguration File:	<pre>yroject_gtr/project_gtr.sdk/fsbl_design_1/Debug/fsbl_design_1.eff</pre>	
🗌 <u>R</u> eset Zynq	ОК Са	incel

7. config\_hw\_sio\_gts is executed with the selected settings. refresh\_hw\_device is then called to rescan the device for new debug cores. The IBERT should be configured as shown in the following example.





Vivado Lab Edition 2017.2			_	×
<u>File Edit T</u> ools <u>W</u> indow Layout	t <u>V</u> iew <u>H</u> elp <u>Q</u> → Quick Access			
_ <b>⊜_</b>   ←   ≁   <b>⊕</b>   ∥   ×   <b>¢</b>	🕺 🖉 😹 Dashboard 🕶		🔚 Serial I/O Analyzer	~
There are no serial I/O links. Auto-detect links.	nks Create links			
Hardware	? _ 🗆 🗆 X	GT Properties	? _ 🗆 🖒 ×	
Q   ¥   ≑   ∅   ▶   ≫   ■	0	№ L1	←   ⇒   Φ	
Name	Status	Name: ibert-0:3121/xilinx tct/Digilent/210308A11BFC/1 1	OURERTIONED ON 1	
✓ ibert-0 (1)	Connected	Status: Pcie.1 - Linked	UNDERTIGUAU_UNET	
✓ ■ xilinx_tcf/Digilent/210308A11B	Open	GT group: No Quad 0		
v	Not programmed			
SysMon (System Monitor)		IBERT core: 19 IBERT		
w	N/A	Device: i arm_dap_1		
SysMon (System Monitor)				
BERT (IBERT)				
✓ № Quad_0 (4)				
P⊲ L0	Pcie.0 - Linked			
P⊲ L1	Pcie.1 - Linked			
№ L2	Usb0 - No Link (rx_det)			
P⊲ L3	Sata1 - No Link (awaitcomwake)			
		General Properties PLLs		
Tcl Console × Messages Serial I/O	Links Serial I/O Scans		? _ [	0 6
current_hw_device [get_hw_device refresh_hw_device -update_hw_pro	es xczu9_0] obes false [lindex [get_hw_devices xczu xczu9 (JTAG device index = 0) is not p			^
current_hw_device [get_hw_device refresh_hw_device -update_hw_pro current_hw_device [get_hw_device	es arm_dap_1] obes false [lindex [get_hw_devices arm_ es xczu9_0]	dap_1] 0]		
<pre>config_hw_sio_gts: Time (s): cpu refresh_hw_device -update_hw_pro</pre>	u = 00:00:00 ; elapsed = 00:00:12 . Mem obes false [lindex [get_hw_devices arm_	_gtr.adk/fsbl_design_1/Debug/fsbl_design_1.elf" "RESET_ ory (MB): peak = 880.699 ; gain = 0.000 dap_1] 0] grammed with a design that has 1 IBERT core(s).	SYSTEM" "FALSE" ] [1	in
		younnet Hield C Clay Hield in a 1 Joint Cort(5).		Ĵ
Type a Tcl command here				- I
GT: L1				

8. The Auto-detect links option does not work for PS-GTR. You can manually create links by using Create Links as shown in the following figure.

Hardware		? _ 🗆 🖒 ×	
$Q_{i}\mid \underbrace{\bigstar}_{i}\mid \diamondsuit \mid \not \bowtie \mid \underbrace{\bowtie}_{i}\mid \underbrace{\bigstar}_{i}\mid \underbrace{\blacksquare}_{i}\mid \underbrace{\blacksquare}_{$		٥	
Name	Status		
<ul> <li>ibert-0 (1)</li> </ul>	Connected		
✓ Image: Value >> ✓ Yange: State >> ✓ Yange:	Open		
czu9_0 (1)	Not programmed		
SysMon (System Monitor)			
@ arm_dap_1 (2)	N/A		
🖉 SysMon (System Monitor)			
V IBERT (IBERT)			
✓ № Quad_0 (4)			
▶a L0	Pcie.0 - Linked		
P⊲ L1	Pcie.1 - Linked		
P⊲ L2	Usb0 - No Link (rx_det)		
Pa L3	Sata1 - No Link (awaitcominit)		
IBERT Core Properties		? _ □ Ľ ×	
		♥ ♥ ♥	
Name: ibert-0:3121/xilinx_tcf/D	igilent/210308A11BFC/1_1_0/IBERT		
Device: @ arm_dap_1			
Definals colo (mo). 4000.		·····×	
General Properties GT Groups			
Tcl Console Messages Serial I/O Lin			
	ks × Serial I/O Scans		
0,   ≚   ≑   ╋			
Create Links			
Create Link Group			
Create Scan		Auto-detect links	s or create links to add serial I/O links to this window
Create Sweep			





9. Create links for all four lanes with each lane's TX connected to the same lane's RX, as shown in the following figure.

Click **OK** when done.

Create I		elect a TX GT and/or an RX GT		6					X
Fo create a	new link s	elect a TX GT and/or an RX GT	, then click	the Add but	ton on the	New Links	s toolbar.		1
TX GTs				RX GTs					
Search:	Q-			Search:	Q				
New Links	S								
+ -	_								
	-								
Descript		TX	RX						
% Link 3	3	L3/TX (arm_dap_1/Quad_0)	L3/RX (ar	rm_dap_1/0	Quad_0)				
% Link 2	2	L2/TX (arm_dap_1/Quad_0)	L2/RX (ar	rm_dap_1/0	Quad_0)				
% Link 1	I	L1/TX (arm_dap_1/Quad_0)	L1/RX (ar	rm_dap_1/0	Quad_0)				
% Link 0	)	L0/TX (arm_dap_1/Quad_0)	L0/RX (ar	rm_dap_1/0	Quad_0)				
✓ <u>C</u> reate	link group	)							
		on: Link Group 0							8
									-
✓ Open s	senaí I/O A	nalyzer layout							
									1
(?)							OK	Ca	ancel

10. The following figure shows the Serial I/O Links view where Status shows all the four lanes as linked.





Vivado Lab Edition 20 File Edit Tools Wir	17.2 ndow	Layo	ut View Help Q-Qu	ick Access							
	×	-	🗶 🖉 🗶 Dashboa	rd 👻					III Se	rial I/O Analyzer	•
Hardware				? _ 0 6 3	<	GT Properties			?.	_ 0 6 ×	
Q   ₹   ♦   ∅   ▶	-   »				×	№ L1			+	$  \rightarrow   \phi$	
Name			Status			Name:	ibort 0:2121/viliov	tcf/Digilant/210209	A11BFC/1_1_0/IBERT/	Ouad 0/L1	
<ul> <li>ibert-0 (1)</li> </ul>			Connected				Pcie.1 - Linked	tenergilentez 10300		ddad_orc i	
✓ I vilinx_tcf/Digilent/	210308	3A11B	Open								
<ul> <li>xczu9_0 (1)</li> </ul>			Not programmed			GT group:	Note: Contract Contract Note:				
😨 SysMon (Sys	tem Mo	nitor)				IBERT core:	· IBERT				
<ul> <li>image: mail of the second secon</li></ul>			N/A			Device:	@ arm_dap_1				
🦉 SysMon (Sys	tem Ma	nitor)									
Y 🦉 IBERT (IBER	T)										
V 🖏 Quad_0 (4)	4)										
Na 10			Pcie.0 - Linked								
No 1			Pcie.1 - Linked								
Na 12			Usb0 - No Link (rx_det)								
Na 13			Sata1 - No Link (awaitcomin	it)							
Tcl Console Messages Q 😤 ♦ 🕂	Seri	ial I/O Li	nks × Serial I/O Scans			General Pr	operties PLLs			? _	
Name	TX	RX	Status	TX Post-Cur	sor	TX Diff Swing	RX PLL Status	TX PLL Status	TXUSERCLK Freq	RXUSERCLK	Freq
Ungrouped Links (0)											
Subscription (4)				Multiple	$\sim$	User Value 😽					
% Link 0	L0/TX	L0/RX	Pcie.0 - Linked	User Value	~	User Value 🗸	Locked	Locked	100.000	1(	00.00
% Link 1	L1/TX	L1/RX	Pcie.1 - Linked	User Value	~	User Value 🗸	Not Locked	Not Locked	100.000	10	00.00
% Link 2	L2/TX	L2/RX	Usb0 - No Link (rx_det)	User Value	$\sim$	User Value 🗸	Locked	Locked	26.000	:	26.00
% Link 3	L3/TX	L3/RX	Sata1 - No Link (awaitcominit)	3.5	$\sim$	User Value 😽	Locked	Locked	150.000	1	50.00

*Note*: The Link 1 PLL Status shows Not Locked, because it uses the Link 0 PLL as required by PCIe protocol.

11. Right-click on any link and select **Create Scan**.

cl Console Messages	Seri	al I/O Li	inks ×		Link Properties	Ctrl+E
Q   <u>∓</u>   ≑   <del>1</del>				×	Delete Create Links	Delete
Name Dungrouped Links (0)	ΤХ	RX	Status		Create Link Group	
<ul> <li>S Link Group 0 (4)</li> </ul>					Create Scan	6
🗞 Link 0	L0/TX	L0/RX	Pcie.0 - L		Create Sweep	
🗞 Link 1	L1/TX	L1/RX	Pcie.1 - L		Commit Properties	
% Link 2	L2/TX	L2/RX	Pcie.2 - L	С	Refresh Serial I/O Objects	
% Link 3	L3/TX	L3/RX	Pcie.3 - L	-		

12. Select the appropriate parameters for EyeScan and perform the EyeScan. For example, the following figure is performing EyeScan on Lane L1 (Link 1). Once the EyeScan completes, the eye from -1UI to +1UI will be displayed.

**Note:** Although the Create Scan pop up shows -0.5UI to +0.5UI, the EyeScan displayed is from -1UI to +1UI.

Send Feedback



🔥 Create Sca	an		×							
	Set the description and other properties to create and optionally run a scan on the selected link.									
Link:	Link 1 (L1/TX	ink 1 (L1/TX, L1/RX)								
Description:	Scan 0	Scan 0 💿								
Scan Proper	ties		_							
<u>S</u> can type	C	2D Full Eyescan	·							
<u>H</u> orizonta	l increment:	4	~							
Horizonta	l range:	-0.500 UI to 0.500 UI	~							
<u>V</u> ertical in	crement:	4	~							
V <u>e</u> rtical ra	inge:	100%	<b>~</b>							
Dwell										
• <u>B</u> ER:	1e-5		~							
◯ <u>T</u> ime:		0	* *							
✓ <u>R</u> un scan										
(?)		OK Can	cel							

13. The following figure is a sample EyeScan performed on Lane L1.

Vivado Lab Edition 2017.2											x
<u>File Edit T</u> ools <u>W</u> indow La <u>v</u> out	View Help Q- Quic	k Access									
□ = 1 + 1 + 1 = 1 = 1 + 1 + 1 + 1 + 1 + 1	🥢 🔀 Dashboard	*							III Se	rial I/O Analyzer	~
Hardware	?	_ 🗆 🖾 × 🛛 S	ican Plots - Scan	0						? 🗆	$\square \times$
$Q \mid \Xi \mid \Leftrightarrow \mid \wp \mid \models \mid \gg \mid \blacksquare \mid$		•	<b>←</b> ⇒ ⊙,	Q 🔀 C 🖪	<ul> <li>Co</li> </ul>	ontour (Filled)	~				
Name S SysMon (System Monitor)	Status					Unit Interval				BE	R
	J/A		-1	-0.75 -0.	-0.2	25 0	0.25	0.5 0	.75	1	
SysMon (System Monitor)										7.9	le-01
# IBERT (IBERT)			50 -							10	le-01
✓ <sup>1</sup> %1 Quad_0 (4)											le-02
	cie.0 - Linked		25-								
№ L1 P	cie.1 - Linked		des			F					le-02
Na L2 U	Jsb0 - No Link (rx_det)		<u> </u>								le-03
		~	Voltage (Codes)							1.0	le-03
Scan Properties	?	_ 🗆 🖾 ×	\$ .25							5.0	le-04
Scan 0		+ I → I O								1.0	le-04
			-50 -							5.0	le-05
Name: SCAN_0			-30							1.0	le-05
Description: Scan 0	8										
Status: In Progress			Summary		Metrics		Settings				
Link: 💊 Link 1			Name:	SCAN_0	Oper	n area: 476	Link settin	os: N/	A		
RX: ILI/RX			Description:	-		n UI %: 30.16		increment 1			
TX: D L1/TX			Started:	2017-May-15 16:51:28			Horizontal		000 UI to 1.0	00.11	
D. CON			Ended:	2017-May-15 16:51:42			Vertical inc	-			
			Ended.	2017 may 10 10.01.42			Vertical rar		0%		
General Properties							veracarra	196. 10			
Tcl Console Messages Serial I/O Links	Serial I/O Scans ×									? _	
Q ¥ ♦ ▶ ■ B B											
	eset RX Scan Type Sta	tus Progress	Ope	n Area Open UI %	Horz Incr	Horz Range	Vert Incr	Vert Range	Dwell	Dwell BER	Dwell '
Scans (1)											
Scan 0 Link 1	2d_full_eye Do	ne	100% 476	30.16	1 👻	-1.000 UI to 1.000 UI	1 🗸	100% 🗸	BER	1e-5 🗸 🗸	
< -											>



*Note*: The value reported by Open UI % is a percentage of the entire horizontal axis, which is 2UI wide for the PS-GTR transceiver.

## Troubleshooting

### **Known Issues**

- 1. By default, FSBL does not enumerate USB as that is something Linux drivers would do. To put USB in link state without Linux, a small modification is required in the FSBL C-code. This modification still does not enumerate the device, it only brings the USB into link state.
- 2. The EyeScan does not have a built-in time-out mechanism. If your link is poor (for example, if L\*\_TM\_DIG\_8.EYESURF\_ENABLE != 1), the EyeScan will hang without providing a user. No results are returned in this case.
- 3. If the EyeScan progress is not moving, make sure the below parameters for all lanes are set for EyeScan to function correctly. Note that \* represents the lane number (as in, for Lane 0 the parameter would be L0).

Click on the lane in the hardware tree and then click on the properties tab. There's a search button you can use to find the properties below.

- a. L\*\_TM\_MISC3.CDR\_EN\_FPL = 0
- b. L\*\_TM\_MISC3.CDR\_EN\_FFL = 0
- c. L\*\_TM\_DIG\_8.EYESURF\_ENABLE = 1

Also check below parameters values which ensures Eye Scan circuit is operational.

- d. L\*\_PLL\_LOCK = 1
- e. L\*\_TM\_SAMP\_STATUS4.E\_SAMP\_PH0\_CALIB\_CODE is non-zero value
- f. L\*\_TM\_SAMP\_STATUS5.E\_SAMP\_PH180\_CALIB\_CODE is non-zero value

## Notes

- 1. As mentioned in Assumptions, IBERT PS-GTR uses the psu\_cortexr5\_0 core, so no other applications should use this core.
- 2. TCM0 and TCM1 memory are combined to form a unified memory for IBERT PS-GTR. Any other processor core should not access this memory while IBERT PS-GTR is running.
- 3. The error counter is 16 bits and the sample counter is 32 bits. Each sample can have 8 bits of error count. Therefore on the edges, the error counter can saturate with a sample count value of 8192. PS-GTR does not stop the sample counter even if the error counter saturates. A prescale=0 produces 8192 samples and thus a total samples of 8192 \*8 (65536) and thus the outside edges of eye could show a BER of e-01 or less depending on the prescale selected.





- 4. The EyeScan assumes there is link present. If there is no link, the EyeScan may not complete. Canceling the EyeScan stops the command sequence, but the state of the previous point scan will be unknown.
- 5. If you run EyeScan and because of no link the EyeScan does not complete, set the register L\*\_TM\_MISC\_ST\_0.EYE\_SURF\_RUN to 0 for the given lane before you run the EyeScan again.
- 6. If you run EyeScan on a lane that is either powered down or Display Port, it will immediately stop and the scan will be marked as incomplete. EyeScan will not work in either scenario.



## AMD7 XILINX

## 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<sup>®</sup> 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<sup>®</sup> 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.

## **Revision History**

The following table shows the revision history for this document.



Section	Revision Summary
11/16/2022 Version 2022.2	
Design Files	Updated design files.
05/20/2022 Version 2022.1	
Design Files	Updated design files.

## **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 2012-2022 Advanced Micro Devices, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Kria, Spartan, Versal, Vitis, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. The DisplayPort Icon is a trademark of the Video Electronics Standards Association, registered in the U.S. and other countries. All other trademarks are the property of their respective owners.

