Vivado Design Suite Tutorial

Programming and Debugging

UG936 (v2019.1) May 22, 2019



II☐ See all versions of this document



Revision History

The following table shows the revision history for this document.

Section	Revision Summary
05/22/2019 V	Version 2019.1
General updates	Editorial updates only. No technical content updates.





Table of Contents

Revision History	2
Debugging in Vivado Tutorial	6
Introduction	6
Objectives	6
Getting Started	7
Lab 1: Using the Netlist Insertion Method for Debugging a Design	13
Introduction	13
Step 1: Creating a Project with the Vivado New Project Wizard	13
Step 2: Synthesizing the Design	14
Step 3: Probing and Adding Debug IP	15
Step 4: Implementing and Generating Bitstream.	24
Lab 2: Using the HDL Instantiation Method for Debugging a Design in Vivado	25
Introduction	25
Step 1: Creating a Project with the Vivado New Project Wizard	25
Step 2: Synthesize Implement and Generate Bitstream	27
Lab 3: Using a VIO Core for Debugging a Design in Vivado	28
Introduction	28
Step 1: Creating a Project with the Vivado New Project Wizard	29
Step 2: Synthesize, Implement, and Generate Bitstream	34
Lab 4: Using Synplify Pro Synthesis Tool and Vivado for Debugging a Design	35
Introduction	35
Step 1: Create a Synplify Pro Project	35
Step 2: Synthesize the Synplify Project	43
Step 3: Create DCPs for the Black Box Created in Synplify Pro	44
Step 4: Create a Post Synthesis Project in Vivado IDE	44
Step 5: Add More Debug Nets to the Project	46
Step 6: Implementing the Design and Generating the Bitstream	48

www.xilinx.com



Lab 5: Using Vivado Logic Analyzer to Debug Hardware	49
Introduction	49
Step 1: Verifying Operation of the Sine Wave Generator	49
Step 2: Debugging the Sine Wave Sequencer State Machine (Optional)	61
Lab 6: Using ECO Flow to Replace Debug Probes Post Implementation	81
Lab 7: Debugging Designs Using Incremental Compile Flow	97
Introduction	97
Procedure	97
Step 1: Opening the Example Design and Adding a Debug Core	97
Step 2: Compiling the Reference Design	
Step 3: Create New Runs	
Step 4: Making Incremental Debug Changes	
Step 5: Running Incremental Compile	
Conclusion	
Lab 8: Using Vivado Serial Analyzer to Debug Serial Links	
Introduction	
Design Description	114
Step 1: Creating, Customizing, and Generating an IBERT Design	
Step 2: Adding an IBERT core to the Vivado Project	116
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 Vivado ILA Core to Debug JTAG-AXI Transactions	143
Introduction	143
Design Description	144
Step 1: Creating a new Vivado Project and Generating the IP Integrator Design with JTAG System ILA	
Step 2: Program the KC705 Board and Interact with the JTAG to AXI Master Core	158
Step 3: Using ILA Advanced Trigger Feature to Trigger on an AXI Read Transaction	
Lab 10: Using Vivado Serial Analyzer to Debug GTR Serial Links	172
Introduction	172
Step 1: Generating Zynq UltraScale+ MPSoC PS Hardware Definition File (HDF)	173

www.xilinx.com



	Step 2: Using XSCT flow to generate FSBL by using HDF	189
	Step 3: ZCU102 Board Settings	190
	Using FSBL with Serial I/O Analyzer to bring up IBERT GTR	191
	Troubleshooting	202
	Using SDK Flow or XSCT Flow to Generate FSBL by Using HDF	203
Le	gal Notices	209
	Please Read: Important Legal Notices	.209





Introduction

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® 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 Vivado. The sixth lab is for debugging JTAG-AXI transactions in Vivado. 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 Vivado logic analyzer, ILA, and Vivado Integrated Design Environment (IDE). In order to be successful using this tutorial, you should have some basic knowledge of Vivado Design Suite 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)

Objectives

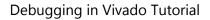
These tutorials:

Ē

- Show you how to take advantage of integrated Vivado 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®-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 as listed below.

Software

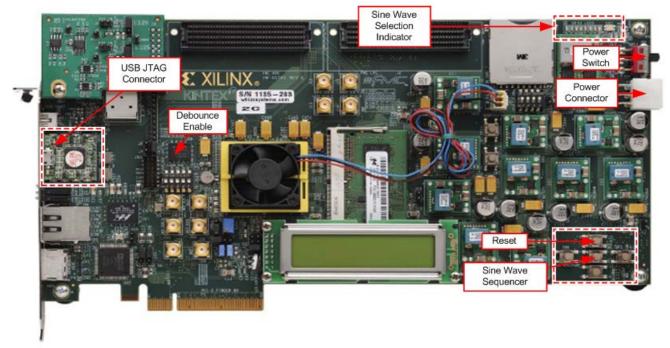
• Vivado Design Suite 2019.1

Hardware

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







X14686

Figure 1: KC705 Board Showing Key Components

Tutorial Design Components

Labs 1 through 4 include:

- A simple control state machine
- Three sine wave generators using AXI-Streaming interface, native DDS Compiler
- Common push buttons (GPIO_BUTTON)
- DIP switches (GPIO_SWITCH)
- LED displays (GPIO_LED) VIO Core (Lab 3 only)

Push Button 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: In 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

Table 1: Pinout Information for the KC705 Board

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

Design Files

- 1. In your C: drive, create a folder called /Vivado_Debug.
- 2. Download the <u>Reference Design Files</u> 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 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 postsynthesis 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 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.

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: Evaluate and Monitor IBERT UltraScale+[™] GTR (IBERT GTR) transceivers in Zynq[®] UltraScale+ MPSoC[™] devices. This lab takes you through the steps of configuring the GTR, generating the FSBL (First Stage Boot Loader file) and using the Vivado Serial Analyzer tool to debug the links.





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 Figure 1: KC705 Board Showing Key Components.





Lab 1: Using the Netlist Insertion Method for Debugging a Design

Introduction

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 ILA core and take the design through implementation. Finally, you will use Vivado® to connect to the KC705 target board and debug your design using 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 green "+" 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**.
- 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.

🏊 proj_netlist - [C:/Vivado_De	ebug/2017.1/proj_netlist/proj_n	etlist.xpr] - Vivado 2	2017.1	
File Edit Flow Tools V	Window Layout View Help	Q. ▼ Quick Access		
	< 🕨 👫 🔅 🗵 🖉	5 18		
Flow Navigator 😤 🖨 🕄	? _ PROJECT MANAGER - proj_ne	etlist		
✓ PROJECT MANAGER	Sources		? _ O Ľ X	Project Summary
Settings	Q, X ≑ + 3	2 0	0	
Add Sources	Contraction Contra			Settings Edit
Language Templates	> @ sinegen_demo	(Mixed) (sinegen_demo	o.vhd) (4)	Project name: proj_netlist
👎 IP Catalog	Settings			
✓ IP INTEGRATOR	> Settings			
Create Block Design	Q,-	General		
Open Block Design	Project Settings	Specify values for vari current project.	ious settings used	I throughout the design flow. These settings apply to the 🥼
Generate Block Design	General Simulation			
	Elaboration	Name:	proj_netlist	
	Synthesis	Project device:	Kintex-7 KC70	05 Evaluation Platform (xc7k325ttfg900-2)
Run Simulation	Implementation Bitstream	Target language:	VHDL	~
✓ RTL ANALYSIS	> IP	Default library:	xil_defaultlib	8
> Open Elaborated Design	Tool Settings	Top <u>m</u> odule name:	sinegen_demo	S
	Project	Language Options		
✓ SYNTHESIS	IP Defaults Source File	Lunguage options		
Run Synthesis	Display	Verilog options	s: verilog	_version=Verilog 2001 ····
> Open Synthesized Design	WebTalk	Generics/Para	meters:	
	Help > Text Editor	Loop count		1,000 🌲
Run Implementation	3rd Party Simulators			
> Open Implemented Design	> Colors			
	Selection Rules Shortcuts			
✓ PROGRAM AND DEBUG	Strategies			
Senerate Bitstream	> Window Behavior			
> Open Hardware Manager	?			OK Cancel Apply Restore
	\odot			

Figure 2: Configuring the Settings







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 **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.

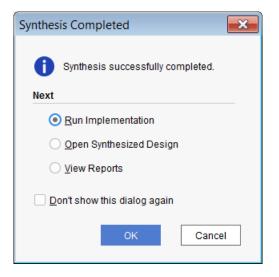


Figure 3: Synthesis Completed Dialog Box

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 since 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 may be situations where synthesis may 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 since 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 may 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.



Figure 4: Open Synthesized Design

- 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.

61	
62	Add mark_debug attributes to show debug nets in the synthesized netlist
63	attribute mark_debug : string;
64	attribute mark_debug of GPIO_BUTTONS_db : signal is "true";
65	attribute mark_debug of GPIO_BUTTONS_dly : signal is "true";
66	attribute mark_debug of GPIO_BUTTONS_re : signal is "true";
67	attribute mark_debug of DONT_EAT : signal is "true";
68	
69	
70 🖯	component sinegen
71	port
72	(
73	clk : in std_logic;
74	reset : in std_logic;
75	<pre>sel : in std_logic_vector(1 downto 0);</pre>
76	<pre>sine : out std_logic_vector(19 downto 0)</pre>
77 ;);
78	
79 🏳	end component;
80	

Figure 5: VHDL Example Using MARK_DEBUG Attributes





Lab 1: Using the Netlist Insertion Method for Debugging a Design

Tcl Console Messages Log Repo	orts Design Ru	ns Debug ×
Q 素 ≑ 兼 ∔ ≓		
Name	Driver Cell	Driver Pin
👻 🖨 Unassigned Debug Nets (7)		
✓ 「☆☆ GPIO_BUTTONS_db (2)	FDRE	Q
」 I III GPIO_BUTTONS_db[0]	FDRE	Q
」I≅ GPIO_BUTTONS_db[1]	FDRE	Q
✓ Jra GPIO_BUTTONS_dly (2)	FDRE	Q
_ I III GPIO_BUTTONS_dly[0]	FDRE	Q
_ I I GPIO_BUTTONS_dly[1]	FDRE	Q
✓ 小☆ GPIO_BUTTONS_re (2)	FDRE	Q
」 I III GPIO_BUTTONS_re[0]	FDRE	Q
」 I III GPIO_BUTTONS_re[1]	FDRE	Q
_∫ ☎ DONT_EAT	FDRE	Q
Debug Cores Debug Nets		

Figure 6: Unassigned Debug Nets Post-Synthesis

- 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 toplevel hierarchy
 - o sel(2) Nets folder under the U_SINEGEN hierarchy
 - o sine(20)- Nets folder under the U_SINEGEN hierarchy

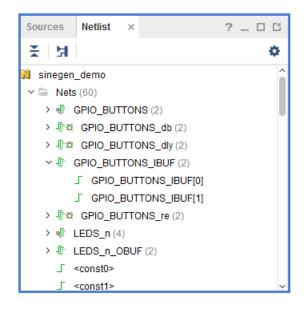


Figure 7: Add Nets for Debug from the Synthesized Netlist





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.

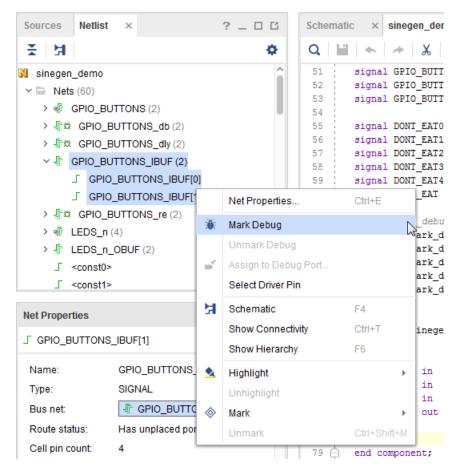


Figure 8: Adding Nets from the Netlist Tab

6. Next, 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 the following two figures.





Lab 1: Using the Netlist Insertion Method for Debugging a Design

Tcl Console Messages Log Repo	rts Design Ru	ns Debug ×
Q 素 ≑ 兼 ∔ ≓		
Name	Driver Cell	Driver Pin
🕆 📮 Unassigned Debug Nets (29)		
✓ Jr [™] GPIO_BUTTONS_db (2)	FDRE	Q
_ ☐ ☎ GPIO_BUTTONS_db[0]	FDRE	Q
_ SPIO_BUTTONS_db[1]	FDRE	Q
✓ Jr [™] GPIO_BUTTONS_dly (2)	FDRE	Q
_ SPIO_BUTTONS_dly[0]	FDRE	Q
_ SPIO_BUTTONS_dly[1]	FDRE	Q
✓ J [™] GPIO_BUTTONS_IBUF (2)	IBUF	0
_ SPIO_BUTTONS_IBUF[0]	IBUF	0
_ SPIO_BUTTONS_IBUF[1]	IBUF	0
✓ Jr [™] GPIO_BUTTONS_re (2)	FDRE	Q
Γ\$\$_GPIΩ_BUTTΩNS_rel01 Debug Cores Debug Nets	FDRF	Q

Figure 9: Newly Added Nets for Debug from the Synthesized Netlist

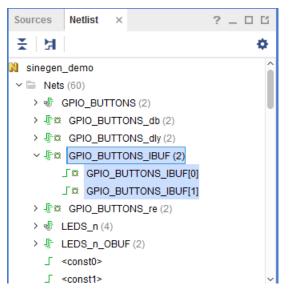


Figure 10: Netlist View of Nets Marked for Debug

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.





Lab 1: Using the Netlist Insertion Method for Debugging a Design

Tcl Console Messa	ges Log Repo	orts Design Ru	ins Debug ×
Q ≚ ♦ 🕷	↓ →		
Name	Set Up Debug	Driver Cell	Driver Pin
👻 🗎 Unassigned Det	oug Nets (29)		
✓ - J̄r ≅ GPIO_BUTT	ONS_db (2)	FDRE	Q
_∫¤ GPIO_BU	JTTONS_db[0]	FDRE	Q
_∫≅ GPIO_BU	JTTONS_db[1]	FDRE	Q
✓ - Jr ≅ GPIO_BUTT	ONS_dly (2)	FDRE	Q
_∫≅ GPIO_BU	JTTONS_dly[0]	FDRE	Q
_∫¤ GPIO_BU	JTTONS_dly[1]	FDRE	Q
✓ J [™] GPIO_BUTT	ONS_IBUF (2)	IBUF	0
_∫¤ GPIO_BU	JTTONS_IBUF[0]	IBUF	0
_∫¤ GPIO_BU	JTTONS_IBUF[1]	IBUF	0
✓ J [™] GPIO_BUTT	ONS_re (2)	FDRE	Q
Γ.ΰGPIΩBI	ITTONS, relol	FDRE	Ω
Debug Cores Debu	ig Nets		

Figure 11: Launching the Set up Debug Wizard

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

🍌 Set Up Debug		
HLL Editions	Set Up Debug This wizard will guide you through the process of Choosing nets and connecting them to debug cores. Associating a clock domain with each of the nets chosen for debug. Choosing additional features on the debug cores like Data Depth, Advanced Trigger mode and Capture control. 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.	
E XILINX ALL PROGRAMMABLE.		
 ? 	< <u>Back</u> <u>Einish</u> Cancel	

Figure 12: Set up Debug Wizard





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**.

dows, then drag them to the list or cli	ck "Add Selected Ne	ets".		
Q ¥ ≑ ㎡ Ⅲ +	-			0
Name	Clock Domain	Driver Cell	Probe Type	
> Jrt≋ 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 🛛 🗸	
> √ft≋ U_SINEGEN/sel (2)	clk	FDRE	Data and Trigger 🛛 👻	
✓ Jata U_SINEGEN/sine (20)	clk	FDRE	Data and Trigger 👒	
_f¤ sine[0]	clk	FDRE	Data and Trigger	
_f¤ sine[1]	clk	FDRE	Data and Trigger	
_f¤ sine[2]	clk	FDRE	Data and Trigger	
∫¤ sine[3]	clk	FDRE	Data and Trigger	
_∫¤ sine[4]	clk	FDRE	Data and Trigger]
Find Nets to Add				Nets to debug:

Figure 13: Specify Nets to Debug

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	🚯 0 debug cores will be removed			
	1 debug core will be created			
	Found 1 clock			
E XILINX	Open in Debug layout			
ALL PROGRAMMABLE.	To apply the above changes, click Finish			
?		< Back Next >	<u> </u>	Cancel

Figure 14: Set up Debug Summary

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.

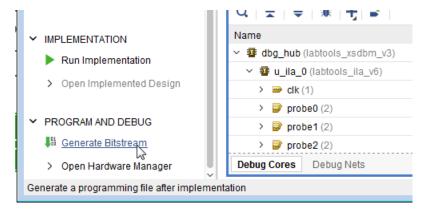


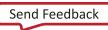
Figure 15: Implement Design and Generate Bitstream

- 2. In the **Save Project** dialog box click **Save**. 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 to close synthesized design before opening implemented design. Click Yes.
- 6. Examine the **Timing Summary** report to ensure that all the specified timing constraints are met.

Tcl Console Messages Log Re	eports Design Runs Power	DRC Meth	odology Timing ×			? _ 🗆
Q 🛨 🖨 🔍	Design Timing Summary					
General Information Timer Settings	Setup		Hold		Pulse Width	
Design Timing Summary	Worst Negative Slack (WNS):	0.491 ns	Worst Hold Slack (WHS):	0.052 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:	12755	Total Number of Endpoints:	12755	Total Number of Endpoints:	6938
Inter-Clock Paths	All user specified timing constra	ints are met.				
Other Path Groups						
User Ignored Paths						
Unconstrained Paths						
Timing Summary - impl_1 (saved)]					

Figure 16: View the Timing Summary Report

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





Lab 2: Using the HDL Instantiation Method for Debugging a Design in Vivado

Introduction

The HDL Instantiation method is one of the two methods supported in Vivado® 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 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 Files** 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 files 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**.
- 6. In the Add Constraints dialog box, click the green "+" sign, and then click Add Files.
- 7. Navigate to /src/lab1 directory and select sinegen_demo_kc705.xdc. Click Next.



XILINX. Lab 2: Using the HDL Instantiation Method for Debugging a Design in Vivado

- 8. Review the **New Project Summary** page. Verify that the data appears as expected, per the steps above. Click **Finish**.
- 9. 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.

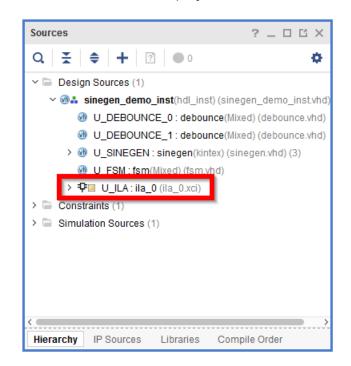


Figure 17: ILA Instantiation in HDL

10. 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	L_0 .
port map	
(
CLK =>	clk,
PROBEO	=> sineSel,
PROBE1	=> sine,
PROBE2	=> GPIO_BUTTONS_db,
PROBE 3	=> GPIO_BUTTONS_re,
PROBE 4	=> 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.

	Size:
✓ PROGRAM AND DEBUG	Modified:
Generate Bitstream	Copied to:
لک Open Hardware Manager	Copied from:
	Copied on:
	<
	General Pro

Figure 19: Generate Bitstream

- **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.

Tcl Console Messages Log R	Reports Design Runs IP Status	Power [DRC Methodology Tim	ning ×		? _ 🗆
Q ≚ ♦ ●	Design Timing Summary					
General Information Timer Settings	Setup		Hold		Pulse Width	
Design Timing Summary	Worst Negative Slack (WNS):	0.511 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 Endpo	oints: 0	Number of Failing Endpoints:	0
Intra-Clock Paths	Total Number of Endpoints:	4437	Total Number of Endpoir	nts: 4437	Total Number of Endpoints:	2478
Inter-Clock Paths	All user specified timing constrain	nts are met.				
🖻 Other Path Groups						
User Ignored Paths						
Unconstrained Paths						
Timing Summary - impl_1 (saved)						

Figure 20: Review Design Timing Summary

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





Lab 3: Using a VIO Core for Debugging a Design in Vivado

Introduction

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® logic analyzer feature. The following figure is a block diagram of the new VIO core.

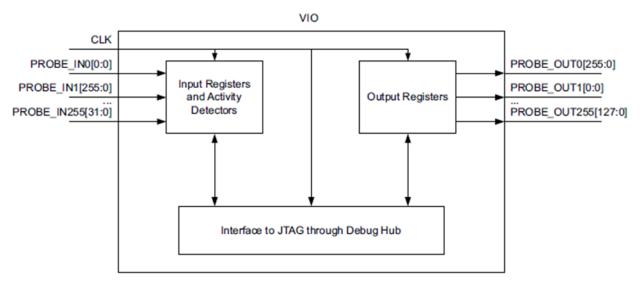


Figure 21: VIO Block Diagram

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 4-bit PROBE_IN0 port. This has two bits to monitor the 2-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. We will configure these 4-bit signals as LEDs during run time to mimic the LEDs displayed on the KC705 board.
- One 2-bit PROBE_OUT0 port to drive the input buttons on the FSM. We will configure it so one bit can be used as a toggle switch during run time to mimic the "PUSH_BUTTON", SW3, and second bit will be used as the "PUSH_BUTTON", 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, to add RTL source files and constraints, and to 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 **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. Click Next.
- 6. Click the green "+" sign, and then click **Add Files**.
- 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 files 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 /src/lab3 directory and select sinegen_demo_kc705.xdc. Click Next.
- 11. In the **Default Part** page, specify the **xc7k325tffg900-2** part for the KC705 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, per the steps above. 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 ila_0 core has been added to the project. However, vio_0 (the VIO core) is missing.

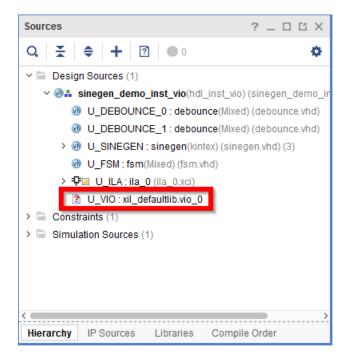


Figure 22: Missing Source for VIO Core

- 14. In this step, you will instantiate and configure this VIO core. 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** to 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.





🅕 Customize IP		×
VIO (Virtual Input/Output) (3.0)		4
Documentation 📄 IP Location C Swite	h to Defaults	
Show disabled ports	Component Name To configure more than 64 probe ports use Vivado Tcl Cons	vio_0
cik probe_in0[0:0]	General Options PROBE_IN Ports(00) PROBE_OUT Input Probe Count 1 0 - 256] Output Probe Count 1 0 - 256] Image: Count Probe Count 1 0 - 256] Image: Count Probe Count 1 0 - 256] Image: Count Probe Count 1 0 - 256]	T Ports(00)
		OK Cancel

Figure 23: Configure General Options of the VIO Core

16. On the **PROBE_IN Ports** tab, set **Probe Width** to **4** bits wide.

🏄 Customize IP					×
VIO (Virtual Input/Output) (3.0)	VIO (Virtual Input/Output) (3.0)				
1 Documentation 📄 IP Location C Switc	h to Defaults				
Show disabled ports	Component Name		vio	.0	8
	To configure more	han 64 probe ports use Vi	vado Tcl Console		
	General Options	PROBE_IN Ports(00)	PROBE_OUT Ports(0())	
	Probe Port		Probe Width [1 - 256]	[1 - 256]	
	PROBE_IN0		4		\otimes
clk probe_out0[0:0]					
				ОК	Cancel

Figure 24: Configure PROBE_IN Ports of the VIO Core

17. On the PROBE _OUT Ports, set Probe Width to 2 bits wide with an initial value of 0 in hex format.





🍐 Customize IP				×
VIO (Virtual Input/Output) (3.0)				4
1 Documentation 📄 IP Location C Switch	to Defaults			
Show disabled ports	Component Name		vio_0	8
	1	be ports use Vivado Tcl Console		
	General Options PROBE_I	N Ports(00) PROBE_OUT Po Probe Width [1 - 256]	Initial Value (in hex)	
	PROBE_OUT0 2		© 0x0	0
<pre>clk probe_out0[1:0] probe_in0[3:0]</pre>				
			ок	Cancel

Figure 25: Configure the PROBE_OUT Ports of the VIO Core

18. Click **OK** to generate the IP. The **Generate Output Products** dialog box will appear. Click **Generate**.

🚴 Generate Output Products 🛛 🔀
The following output products will be generated.
Preview
 Instantiation Template Synthesized Checkpoint (.dcp) Behavioral Simulation Change Log
Synthesis Options ?
© <u>G</u> lobal
Out of context per IP
Run Settings
Number of jobs: 8 👻
Apply Generate Skip

Figure 26: Generate Output Products for the VIO Core





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 as shown in the following figure.

VIO
U_VIO : vio_0
port map
(
CLK => clk,
<pre>PROBE_IN0(3) => DONT_EAT,</pre>
PROBE INO(2) => GPIO BUTTONS re(1),
PROBE INO(1 downto 0) => sineSel,
PROBE OUTO(1) => push button reset,
PROBE OUTO(0) => push button vio
);

Figure 27: VIO Instantiation in the Top Level Design

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

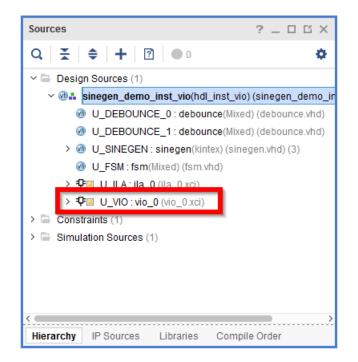


Figure 28: Instantiated VIO Core in the Sources Window





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.

```
-- ILA
U_ILA : ila_0
port map
(
    CLK => clk,
    PROBE0 => sineSel,
    PROBE1 => sine,
    PROBE2 => GPI0_BUTTONS_db,
    PROBE3 => GPI0_BUTTONS_re,
    PROBE4 => GPI0_BUTTONS_dly,
    PROBE5 => GPI0_BUTTONS
);
```

Figure 29: Hook signals that need to be debugged in the ILA

Step 2: Synthesize, Implement, and Generate 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.

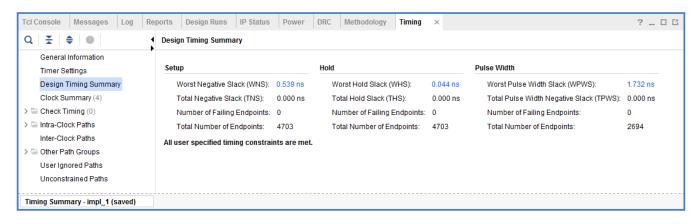


Figure 30: Report Timing Summary Dialog Box

 Proceed to Lab 5: Using Vivado Logic Analyzer to Debug Hardware chapter to complete the rest of the steps for debugging the design. Skip forward to Verifying the VIO Core Activity (Only applicable to Lab 3) section to complete the rest of this lab.





Lab 4: Using Synplify Pro Synthesis Tool and Vivado for Debugging a Design

Introduction

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® 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 > 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**.

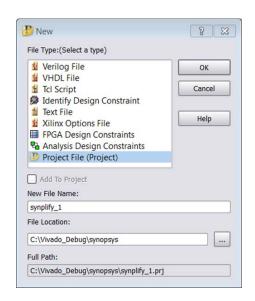


Figure 31: Synplify Pro New Project Dialog Box



XILINX.Lab 4: Using Synplify Pro Synthesis Tool and Vivado for Debugging a Design

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

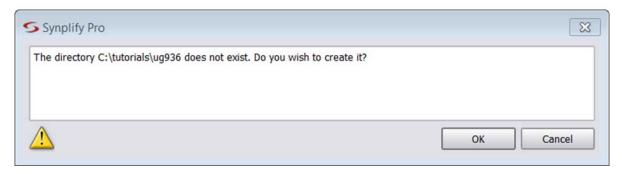


Figure 32: Synplify Pro project Confirmation Dialog Box



XILINX_•Lab 4: Using Synplify Pro Synthesis Tool and Vivado for Debugging a Design

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

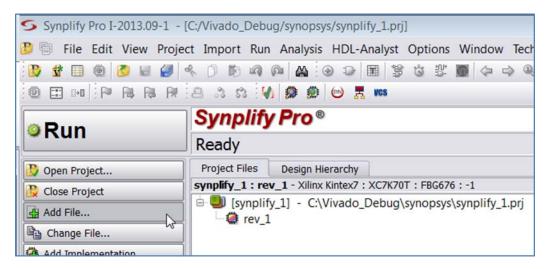


Figure 33: Adding Files to a Synplify Pro Project

- 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.

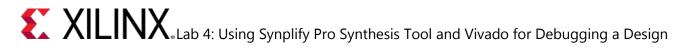


XILINX_{•Lab} 4: Using Synplify Pro Synthesis Tool and Vivado for Debugging a Design

S Add Files to Project	8
Look in: C:\Vivado_Debug\src\Lab4 C:\Vivado_Debug\src\Lab4 C:\Vivado_D	
File name: "debounce.vhd" "fsm.vhd" "sinegen_demo.vhd" Files of type: HDL Files (*.vhd *.vhd! *.v *.sv *.vma)	
VHDL/Verilog lib:	
Files to add to project: (3 file(s) selected) 🗹 Use relative paths 🗹 Add files to Folders 🛛 Folder Options	
\\src\Lab4\debounce.vhd \\src\Lab4\fsm.vhd \\src\Lab4\sinegen_demo.vhd	<- Add All <- Add Remove All -> Remove -> OK Cancel

Figure 34: Adding VHDL Source Files to the Synplify Pro Project





10. 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	8
Look in:	C:\Vivado_Debug\src\Lab4	
.\src\Lab4\deb .\src\Lab4\fsm	.vhd gen_demo.vhd	<- Add All <- Add Remove All -> Remove -> OK Cancel

Figure 35: Adding SDC Constraints File to the Synplify Pro Project



XILINX. Lab 4: Using Synplify Pro Synthesis Tool and Vivado for Debugging a Design

11. 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.

S	Add	Files to	Project				×
Look in:	proj/xcoswmktg/smitha/vivado_de	bug/lab4	-) 🗢 🔶	1	:	
💻 Comp	Name	Size	Туре	Date Mo	dified $ riangle$		
🚞 smitha	📄 synplify_1.fdc	468 byte	s 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 lib	:						
Files to add to p	oroject: (1 file(s) selected) 🗹 Use relative	e paths 🛛 🗸	Add files to	Folders	Folder	Options]
./lab4/synplify_	1.fdc						<- Add All
							<- Add
							Remove All ->
							Remove ->
							ОК
							Cancel

Figure 36: Adding FPGA Constraints File to the Synplify Pro Project



XILINX Lab 4: Using Synplify Pro Synthesis Tool and Vivado for Debugging a Design

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

Synplify Pro I-2013.09-1 -	[C:/Vivado_Debug/synopsys/synplify_1.p				
🤔 🗐 File Edit View Proje	ct Import Run Analysis HDL-Analys				
	* D D Q Q A B 0 D B 1				
© Run	Synplify Pro®				
- Kan	Ready				
🕑 Open Project	Project Files Design Hierarchy				
没 Close Project	synplify_1 : rev_1 - Xilinx Kintex7 : XC7K 				
Add File					
Change File					
Add Implementation					
🎲 Implementation Optipos	Compiler Directive				
R Add P&R Implementation	- @ rev_1				

Figure 37: Opening Implementation Options in Synplify Pro

14. 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**.

	Implementation Results		High Reliability		rev 1
Fechnology:	Part	Package:	Speed:		
Xilinx Kintex7	▼ XC7K325T	▼ FFG900	-2		
Device Mapping Options					
Option			Value	-	
Fanout Guide			10000		
Disable I/O Insertion					
Disable Sequential Optimization	S			•	1 <u>.</u>
Update Compile Point Timing Da	ata			-	
Click on an option for description					
System Designer Board File					

Figure 38: Specifying Implementation Options in Synplify Pro



XILINX_{•Lab} 4: Using Synplify Pro Synthesis Tool and Vivado for Debugging a Design

15. 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;
```

Figure 39: Specifying Attributes to Preserve Net Names in Synplify

16. 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_re : signal is "true";
```

Figure 40: Add MARK_DEBUG Attribute in HDL File

17. The symplify_1.sdc file contains various kinds of constraints such as pin location, I/O standard, and clock definition. The symplify_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 define_attribute	<pre>-comment {Mark sinegen as black box} {v:work.sinegen} {syn_black_box} {1} -comment {Set no_prune on sinegen} {v:work.sinegen} {syn_noprune} {1} -comment {Mark entire bus for debug} {i:sinegen.sine[*]} {mark_debug} {"true"} -comment {Mark entire bus for debug} {i:sinegen.sel[*]} {mark_debug} {"true"}</pre>

Figure 41: Synplify Pro Constraints in FDC Files

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 IDE for further debugging. For further information on these attributes, please refer to the Synplify Pro User Manual and Synplify Pro Reference Manual.



XILINX_{•Lab} 4: Using Synplify Pro Synthesis Tool and Vivado for Debugging a Design

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_1.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 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.

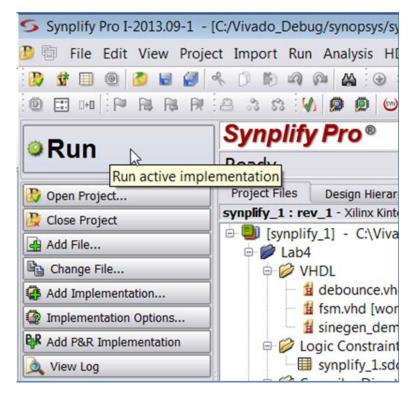


Figure 42: Synthesize the Design in Synplify

- 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.
- When synthesis completes, the output netlist is written to the file: rev_1/sinegen_demo.edf
 [Optional] To view the netlist select View > View Result File.
- 5. Click **File > Save All** to save the project, then click **File > 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 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 > 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 Vivado IDE.
- 2. Click Create Project. This opens up the New Project wizard. Click Next.
- 3. Set the Project Name to proj_synplify. 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.



XILINX Lab 4: Using Synplify Pro Synthesis Tool and Vivado for Debugging a Design

- 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 the following file:
 - o 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:

- o sine_high
- o sine_mid
- o 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 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.

Sources Netlist	t ×			? _ 🗆 🖸	Sci
포 너희				٥	+
> 鈩 GPIO_B > 鈩☆ GPIO_ > 鈩 GPIO_B > 磭 LEDS_n		ONS_c (2) NS_c_i (2) ONS_db (2) ONS_dly_1 (2) NS_dly_5 (2) ONS_re_1 (2) NS_re_5 (2)		Î	
> 小 LEDS_n > 小 sine (20					
> √r¤ sineSe		Bus Net Properties	Ctrl+E		
_ clk	*	Mark Debug	ß	~	
Bus Net Properties	μ, ,	Unmark Debug Assign to Debug Port Select Driver Pin		? _ □ Ľ × ← → ✿	040.JL3 040
Name:	Я	Schematic Show Connectivity	F4 Ctrl+T		
Number of nets:		Show Hierarchy	F6		
	\$	Highlight Unhighlight	Þ		
	۵	Mark	•		
General Scalar		Unmark Go to Source	Ctrl+Shift+M		

Figure 43: Mark Additional Signals for Debug

www.xilinx.com



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	Debug ×	? _ 🗆 🖸
Q 素 ♦ 兼 + ≓				¢
Name	Driver Cell	Driver Pin	Probe Type	
👻 🖨 Unassigned Debug Nets (30)				
> 近☎ GPIO_BUTTONS_c(2)	IBUF	0		
> Jra GPIO_BUTTONS_db (2)	FDRE	Q		
> √ft GPIO_BUTTONS_dly_1 (2)	FDRE	Q		
> Jac GPIO_BUTTONS_re_1 (2)	FDRE	Q		
> √r¤ sine (20)	FDRE	Q		
> 近☆ sineSel (2)	FDRE	Q		
Debug Cores Debug Nets				

Figure 44: Nets Added for Debug through the Synplify Pro Flow in Vivado IDE

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 Me	ssages Log I	Reports	Design Runs	Debug ×	
Q ¥ ♦	🕷 🛨 🖃				
Name	Set Up Debug	Driver	Cell Driver	Pin Probe Typ	e
🗸 🖨 Unassigned	Debug Nets (30)				
> √f¤ GPIO_B	UTTONS_c (2)	IBUF	0		
> √f¤ GPIO_B	UTTONS_db (2)	FDRE	Q		
> √f¤ GPIO_B	UTTONS_dly_1 (2)	FDRE	Q		
> Jr̃≋ GPIO_B	UTTONS_re_1 (2)	FDRE	Q		
> - ∫ r¤ sine (20)	FDRE	Q		
> √r¤ sineSel	(2)	FDRE	Q		

Figure 45: Run the Set up Debug Wizard

2. Click through the wizard to create Vivado 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 Vivado Logic Analyzer to Debug Hardware to complete the rest of this lab.





Lab 5: Using Vivado Logic Analyzer to Debug Hardware

Introduction

The final step in debugging is to connect to the hardware and debug your design using the Integrated Logic Analyzer. 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® 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.
- Some 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 SW13, 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\2019.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 SW13, 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.

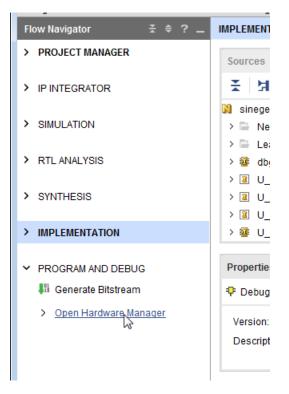


Figure 46: Open Hardware Manager

2. The Hardware Manager window opens. Click Open Target > Open New Target.





Lab 5: Using Vivado Logic Analyzer to Debug Hardware

HARDWARE MANAGER - unconnected									
1 No hardware target is open. Open target									
Hardware	ø	Auto Connect							
$ Q_{n} \stackrel{\times}{=} \Leftrightarrow \not \bowtie \models \Rightarrow$		Recent Targets Available Targets on Server							
		Open New Target	6						
No conter	ıt								

Figure 47: Connect to a Hardware Target

- 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	×
Select local or r	erver Settings emote hardware server, then configure the host name and port settings. Use Local let is attached to the local machine; 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>N</u> ext > <u>F</u> inish Can	cel

Figure 48: Hardware Server Settings

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**.





٨. (Open New H	ardware Targ	et				×
Se		target from the		-		ate JTAG clock (TCK) lect a different target	
	Hardware <u>T</u> arg	ets					
	Туре	Name		JTAG Clock F	requency		
	xilinx_tcf	Xilinx/Port_#00	03.Hub_#0004	6000000	~		
			Add Xilin	nx Virtual Cable	(XVC)		
	Hardware <u>D</u> evi	ces (for unknow	n devices, spec	cify the Instructi	ion Register	(IR) length)	
	Name	ID Code	IR Length				
	@ xc7k325t_0	33651093	6				
	Hardware serve	er: localhost:312	1				
Ċ	?		< <u>E</u>	lack	<u>N</u> ext >	<u>F</u> inish	Cancel

Figure 49: Select Hardware Target





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

🅕 Open New Hardward	i Open New Hardware Target							
	Open Hardware Target Summary							
HLx Editions	 Hardware Server Settings: Server: localhost:3121 							
	 Target Settings: Target: xilinx_tcf/Xilinx/Port_#0003.Hub_#0004 Frequency: 6000000 							
E XILINX ALL PROGRAMMABLE.	To connect to the hardware described above, click Finish							
(?)	< <u>B</u> ack <u>N</u> ext > <u>Finish</u> Cance	el						

Figure 50: Hardware Target Summary

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

🍐 Open Hardware Target		×
Connecting to server		
	<u>B</u> ackground	Cancel

Figure 51: Open Hardware Target





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.

Hardware	? _ 🗆	Ľ X
$Q \mid \underbrace{\star} \mid \diamondsuit \mid \varnothing \mid \models \mid \gg \mid \blacksquare \mid$		•
Name	Status	
✓ Iocalhost (1)	Connected	
✓ ✓ ✓ ✓ xilinx_tcf/Xilinx/Port_#0003.Hu	Open	
✓ ⊕ xc7k325t_0 (1)	Not programmed	
壅 XADC (System Monitor)		

Figure 52: Active Target Hardware

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





Hardware			? _ 🗆	с×		
Q ¥ ♦ ∅ ▶ ≫				٥		
Name		Status				
 Iocalhost (1) 		Connected				
✓ Zerver viller vil	3.Hu	Open				
 	Har	dware Device F	roperties		Ctrl+E	
C	Verit	gram Device fy Device resh Device			ß	
		Add Configuration Memory Device Boot from Configuration Memory Device				
Hardware Davice Droperties	-	Program BBR Key Clear BBR Key				
Hardware Device Properties	Prog	Program eFUSE Registers				
xc7k325t_0	Expo	ort to Spreadsh	eet			

Figure 53: Program Active Target Hardware

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.

🔥 Program Device		×
	ramming file and download it to your hardware device. You can optionally select a debug onds to the debug cores contained in the bitstream programming file.	4
Bitstre <u>a</u> m file: Debu <u>q</u> probes file: ✓ <u>E</u> nable end of st	C://ivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1/sinegen_demo.bit C://ivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1/sinegen_demo.ltx artup check	○○○○
?	<u>P</u> rogram	Cancel

Figure 54: Select Bitstream File to Download for Lab 1



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.





Lab 5: Using Vivado Logic Analyzer to Debug Hardware

	? _ 🗆	с×
		۰
Status		
Connected		
Open		
Programmed		
Oldle		
	Connected Open Programmed	Connected Open Programmed

Figure 55: ILA Core Detection

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

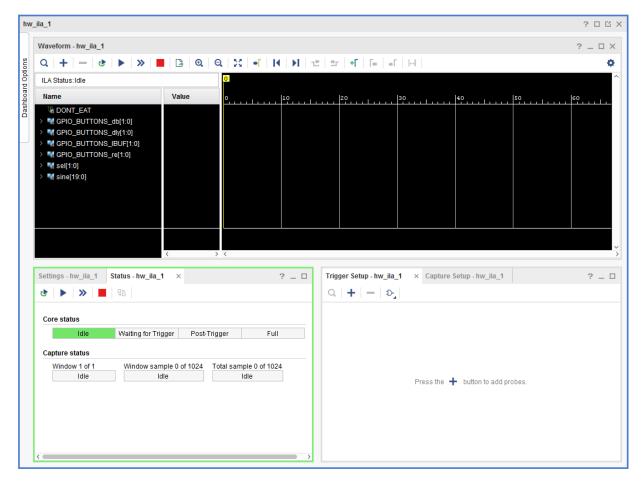


Figure 56: Vivado Integrated Logic Analyzer window

www.xilinx.com





Verifying Sine Wave Activity

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

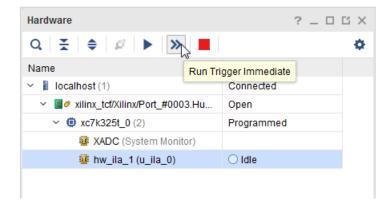


Figure 57: Run Trigger Immediate Button

13. 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							? _ 🗆 >
Q + - ♂ ▶ ≫	G	Q 🔀 📲 🛛	∢)	n± ±r +F F	⊨ ⇒ ' !⊶		¢
ILA Status: Idle							1,023
Name	Value		20	90 	400	600	800
UNT_EAT	0						
> 📢 GPIO_BUTTONS_db[1:0]	0				0		
> 📢 GPIO_BUTTONS_dly[1:0]	0				0		
> 🔣 GPIO_BUTTONS_IBUF[1:0]	0				0		
> SPIO_BUTTONS_re[1:0]	0				0		
> 🔣 U_SINEGEN/sel[1:0]	0				0		
> V_SINEGEN/sine[19:0]	05133						
		Updated at: 2	017-Mar-	16 14:59:13			

Figure 58: Output Sine Wave Displayed in Digital Format

Displaying the Sine Wave

14. Right-click **U_SINEGEN/sine[19:0]** signals, and select **Waveform Style > Analog** as shown in the following figure.



TIP: 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.





Waveform - hw_ila_1		? _ 🗆 X								
Q + − ♂ ▶ ≫	Q + - & > = B Q Q X + H H ± ± + F H H									
ILA Status:Idle		1,023								
Name	Value									
1/2 DONT_EAT	0									
> 🔣 GPIO_BUTTONS_db[1:0]	0									
> 🔣 GPIO_BUTTONS_dly[1:0]	0									
> Variable Content Stress S	0									
> Mage GPIO_BUTTONS_re[1:0]	0	0								
V V_SINEGEN/sel[1:0]	0									
> 🍕 U_SINEGEN/sine[19:0]	05133									
	<	Updated at: 2017-Mar-16 14:59:13								

Figure 59: Output Sine Wave Displayed in Analog Format - High Frequency 1

15. 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		? _ 🗆 X
Q + − ϑ ▶ ≫	🕒 🕒 🔍	Q X +
ILA Status:Idle		1,023
Name	Value	
We DONT_EAT	0	
> 📲 GPIO_BUTTONS_db[1:0]	0	
> 💐 GPIO_BUTTONS_dly[1:0]	0	
> 💐 GPIO_BUTTONS_IBUF[1:0]	0	
> Variable Control Science (1:0)	0	
> 🔣 U_SINEGEN/sel[1:0]	0	
> 🍣 U_SINEGEN/sine[19:0]	20787	
	<	Updated at: 2017-Mar-16 14:59:13

Figure 60: Output Sine Wave Displayed in Analog Format - High Frequency 2

Correcting Display of the Sine Wave

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

16. Cycle the sine wave sequential circuit by pressing the GPIO_SW_E push button as shown in the following figure.





Lab 5: Using Vivado Logic Analyzer to Debug Hardware

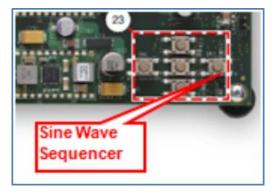


Figure 61: Sine Wave Sequencer Push Button

17. 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 + − & ▶ №	🕞 🔍 e	2 20 📲 14 🕨	l 🗠 🗠 +F F	• • • • • •		0
ILA Status:Idle						1,023
Name	Value	°	200	400	600	800
UONT_EAT	0					
> 💐 GPIO_BUTTONS_db[1:0]	0			0		
> 💐 GPIO_BUTTONS_dly[1:0]	0			0		
> 🔣 GPIO_BUTTONS_IBUF[1:0]	0	0		0		
> 💐 GPIO_BUTTONS_re[1:0]	0			0		
> 📢 U_SINEGEN/sel[1:0]	1			1		
> ➡ U_SINEGEN/sine[19:0]	-83150					
		Updated at: 2017-Ma	ar-16 15:02:38			

Figure 62: Output Sine Wave Displayed in Analog Format - Mid Frequency





18. Repeat step 17 and 18 to view other sine wave outputs.

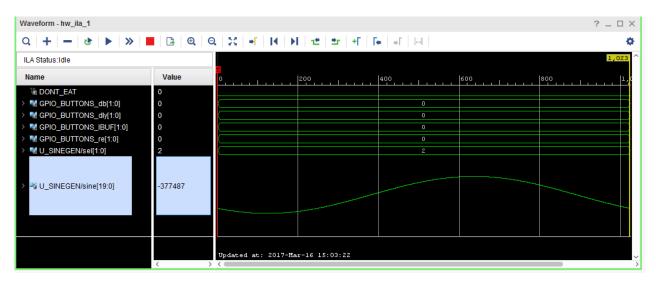


Figure 63: Output Sine Wave Displayed in Analog Format - Low Frequency

Waveform - hw_ila_1						? _ 🗆 X
Q + − ♂ ▶ ≫	🕒 🕃 🔍 (ચ ઽઽ ન I∢ મ	n± ±r +F F	• →[⊡-		•
ILA Status:Idle						1,023
Name	Value	•	200	400	600	800 1, <mark>0</mark>
₩ DONT_EAT	0					
> 📓 GPIO_BUTTONS_db[1:0]	0	0		0		
> M GPIO_BUTTONS_dly[1:0]	0			0		
> 💐 GPIO_BUTTONS_IBUF[1:0]	0			0		
> M GPIO_BUTTONS_re[1:0]	0			0		
> V_SINEGEN/sel[1:0]	3			3		
> 🍣 U_SINEGEN/sine[19:0]	75777					
	< ::	Updated at: 2017-Ma	r-16 15:03:56			

Figure 64: Output Sine Wave Displayed in Analog Format - Mixed Frequency

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.

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.

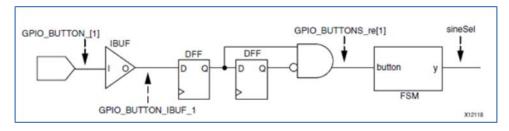


Figure 65: Sine Wave Sequencer Button Schematic

Viewing the State Machine Glitch

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_1_IBUF 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:
 - a. Trigger Mode to BASIC_ONLY
 - b. Capture Mode to BASIC
 - c. Window Data Depth to 1024
 - d. 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 + - 3 > 3	🕞 🔍 Q	X - F H H	12	ter +F ∣ F	• • •				¢
ILA Status: Idle									1,023
Name	Value	• •	200		400	eoo		800	
1 DONT_EAT	0								
> W GPIO_BUTTONS_db[1:0]	0				0				
> W GPIO_BUTTONS_dly[1:0] > W GPIO_BUTTONS_IBUF[1:0]	0 0)			0				
GPIO_BUTTONS_re[1:0]	0				0				
> NU_SINEGEN/sel[1:0]	3				3				
V_SINEGEN/sine[19:0]	-183094								
	-105094								
		Updated at: 2017-Ma	r-16 1	5:08:22					~
	< >	< 🗆							>
Settings - hw_ila_1 × Status - hw_il	la_1	? _		Trigger Setup -	hw_ila_1 ×	Capture Set	up - hw_ila_1		? _ 🗆
Trigger Mode Settings			î	a + -	Ð-				
				Name		Operator	Radix	Value	Port
Trigger mode: BASIC_ONLY	*			GPIO_BUTTON	IS_IBUF[1:0]		[B] •	XX	 probe3[1:0]
Capture Mode Settings									
Capture mode: ALWA	YS 🗸								
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	S		~	<					>

Figure 66: Trigger Setup Window





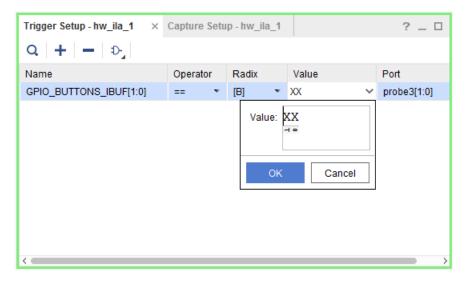


Figure 67: Setting Trigger Conditions

CAUTION! For different labs the GPIO_BUTTONS_IBUF may show up differently. This may 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.





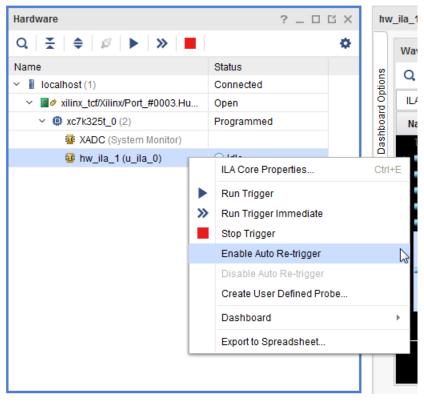


Figure 68: Enable Auto Re-trigger



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.
- o Uploads and displays waveforms.
- 3. On the KC705 board, press the Sine Wave Sequencer button until you see multiple transitions on the GPIO_BUTTONS_1_IBUF 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. Once 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.





hw_ila_data_1.wcfg*							- 0	ą×
<u>=0</u>							1,0	024 🔺
Name Name	Value	·	200	400		600	800	1,0
U_SINEGEN/sel[1:0]	3		0			3		
🔍 🖬 🖬 GPIO_BUTTONS_re[1:0]	0		0			0		
🔍 🖬 🖬 GPIO_BUTTONS_dly[1:0]	2	(0		0)	2		
GPIO_BUTTONS_db[1:0]	0							
년 1 월 월 월 U_SINEGEN/sine[19:0] 연합	105321							
🔐 🖽 📲 GPIO_BUTTONS_IBUF[0:0]	0							5
GPIO_BUTTONS_IBUF_1[1:1]	1		0		0)	1		
1								
154 155								-
	۰ F	•						

Figure 69: GPIO_BUTTONS_BUF1 Signal Glitch

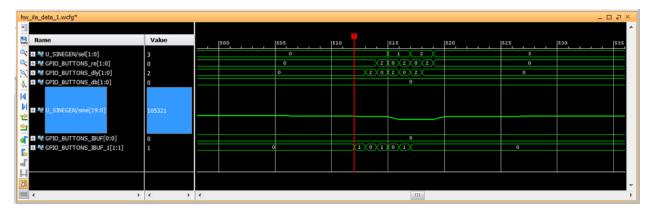


Figure 70: GPIO Buttons_1_re Signal Glitch Magnified





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 Debounce Enable in Figure 1: KC705 Board Showing Key Components) 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**.

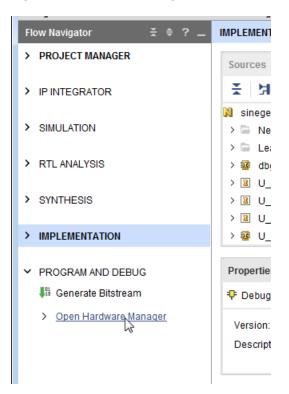


Figure 71: Open Hardware Manager

The Hardware Manager window opens.





2. Click Open a new hardware target.

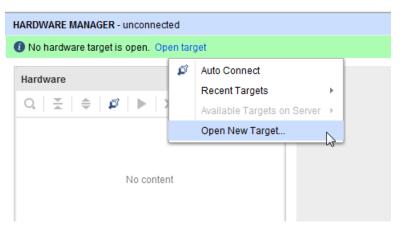


Figure 72: Connect to a New Hardware Target

- 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**.





Hardware			? _ 🗆	ц×		
Q 素 ♠ ∅ ▶ ≫				•		
Name		Status				
Iocalhost (1)		Connected				
✓ Ø xilinx_tcf/Xilinx/Port_#000	3.Hu	Open				
 	Har	dware Device F	Properties		Ctrl+E	
	Prog	gram Device				
		fy Device			63	
C		Refresh Device				
		Add Configuration Memory Device				
	Boo	Boot from Configuration Memory Device				
	Prog	gram BBR Key.				
Hardware Device Properties	Clea	Clear BBR Key				
	Prog	Program eFUSE Registers				
<pre>@ xc7k325t_0</pre>	Exp	ort to Spreadsh	neet			

Figure 73: Program FPGA

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>a</u> probes file:	C:/Vivado_Debug/2017.1/proj_hdl_vio/proj_hdl_vio.runs/impl_1/sinegen_demo_inst_vio.bit O:/Vivado_Debug/2017.1/proj_hdl_vio/proj_hdl_vio.runs/impl_1/sinegen_demo_inst_vio.ltx O:
✓ Enable end of st	artup check
?	Program Cancel

Figure 74: Program Device with the sinegen_demo_inst_vio.bit File

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





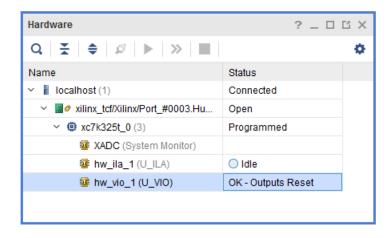


Figure 75: The ILA and VIO Cores in the Hardware Window





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

hw	/_ila_1		? 🗆 🖒 X
	Waveform - hw_ila_1		? _ 🗆 ×
suo			0
Dashboard Options	ILA Status:Idle		^
hboar	Name Value 0 10 20	30	
Das	> McGPIO_BUTTONS_dM(1:0) > McGPIO_BUTTONS_dM(1:0) > McGPIO_BUTTONS_re_[11:0] \% push_button_reset_1 \% push_button_vio > McSine[19:0] > McSineSel[1:0]		
			ž
		- hw_ila_1 × Capture Setup - hw_ila_1	? _ 🗆
		= D_	
	Core status Idle Pre-Trigger Waiting for Trigger Post-Trigger Full Capture status Vindow 1 of 1 Vindow sample 0 of 1024 Total sample 0 of 1024 Idle Idle Idle Idle	Press the 🕇 button to	add probes.

Figure 76: ILA Core and VIO Core Dashboards

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

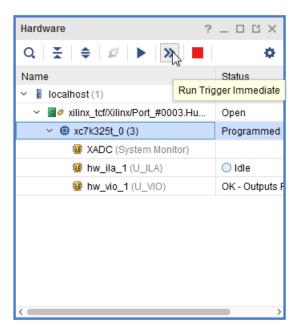


Figure 77: Run Trigger Immediate





- 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 > 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.



Figure 78: Sine Wave after Modifying the Properties of the sine [19:0] Signal

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.

hw	_ila_1	
	Waveform - hw_ila_1	
ions	Q + − & ► ≫ ■	E
Dashboard Options	ILA Status:Idle	
	Name	Va
5	A GPIO_BUTTONS_dly[1:0] GPIO_BUTTONS_dly[1:0] GPIO_BUTTONS_re_1[1:0] How Dashboard Options http://doi.org/10.00 Sine[19:0] SineSel[1:0]	
	Settings - hw ila 1 Status - hw ila 1	<

Figure 79: Invoking Dashboard Options





17. Add the VIO window to the ILA dashboard by selecting hw_vio_1.

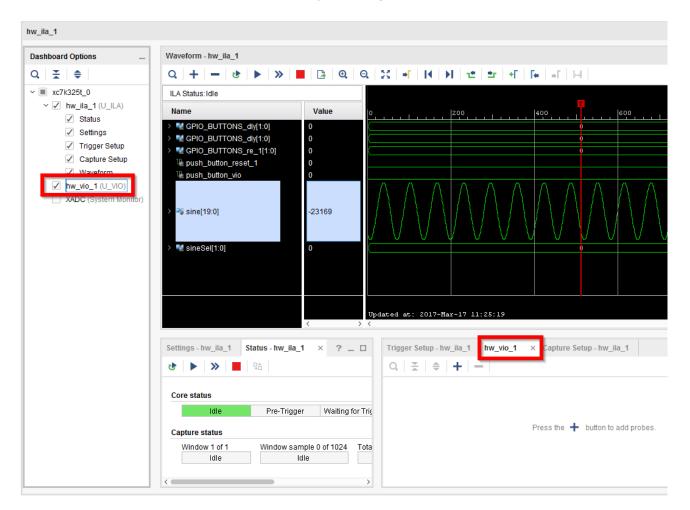


Figure 80: Dashboard Options Adding VIO

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.

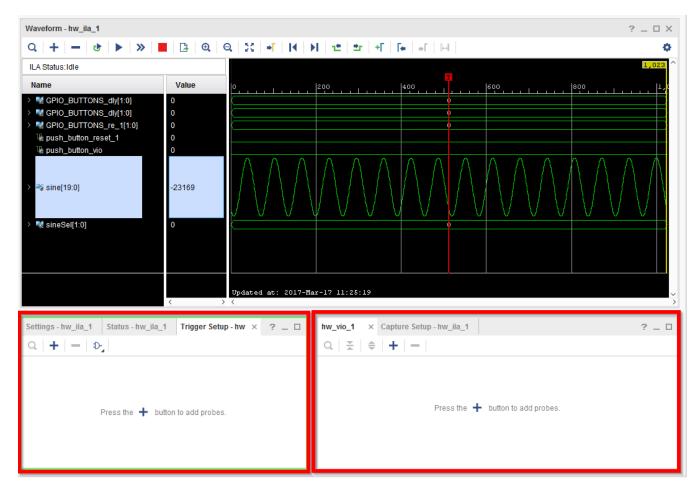


Figure 81: ILA Basic Trigger Window and VIO Window Adjustment

- 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.





Waveform - hw_ila_1								
Q + − & ▶ ≫	🕒 🔍	Q 🔀 📲 🚺	N n± l±r l+F l	• +[]	⊷			
ILA Status: Idle								
Name	Value		200	400		600		800
> Mage GPIO_BUTTONS_dly[1:0]	0							
> M GPIO_BUTTONS_dly[1:0] > M GPIO_BUTTONS_re_1[1:0]	0	[<u> </u>			
light button_reset_1	0							
⅓ push_button_vio	0						Λ	<u>^ ^</u>
≻ 📲 sine[19:0]	-23169			\bigwedge			\bigwedge	
> 📲 sineSel[1:0]	0	<			0			
		Updated at: 2017-M	ar-17 11:25:19					
	< >	× <						
Settings - hw_ila_1 Status - hw_ila_	1 Trigger Setu	p-hw × ? _ □	hw_vio_1 × Captu	ire Setup - h	w_ila_1			
Q + = D			0. ≚ ≑ +	-				
					1	Add Probes		×
				₹ ♦				•
Press the 🕂 bu	tton to add probes).		Search:	0-			
						(5)		~
				× ₩ hw	vio 1	(3)		*
					DONT_EAT			-
						TONS_re[1:1]		
					push_butto push_butto			
LL NAME=~"U ILA"}]					sineSel_1[1		S	
s xc7k325t_0] -filter {CELL_NAME=	~"U_ILA"}]]							
;:19 //proj_hdl_vio/proj_hdl_vio.hw/bac	kup/hw_ila_dat	a_1.ila. Use Tcl com	mand 'import_hw_ila_c	ia				u
		-						

Figure 82: VIO Add Probes Window





21. Note the values on all probes in the **hw_vio_1** window.

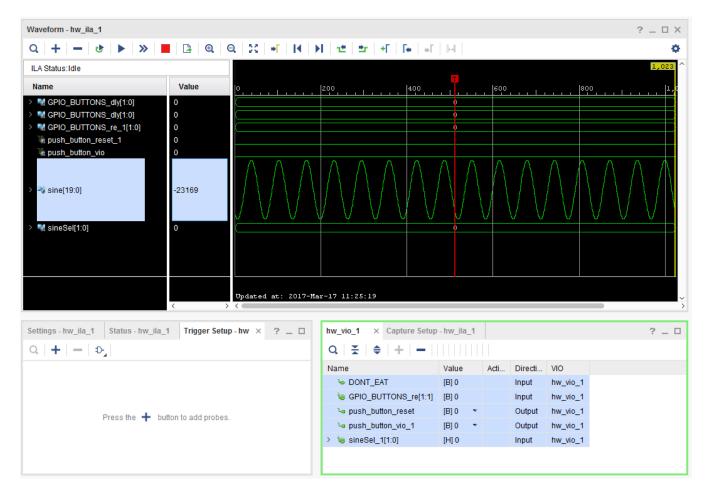


Figure 83: VIO Probes Added to hw_vio_1 Window





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	p - hw_ila_1				? _ 🗆
Q 素 ≑ + -					
Name	Value	Activity	Direction	VIO	
ኈ DONT_EAT	[B] 0		Input	hw_vio_1	
ⓑ GPIO_BUTTONS_re[1:1]	[B] 0		Input	hw_vio_1	
⊶ push_button_rese*			0	hw_vio_1	
∿a push_button_vio_	Debug Probe	Properties	Ctrl+E	hw_vio_1	
> 🐚 sineSel_1[1:0]	Text			hw_vio_1	
	Active-High B	utton			
	Active-Low B	utton			
	Toggle Butto	n			
	Radix		14		? _ 🗆 🗆
	Rename				
	Name		×		^
	Remove		Delete		
and 'import_hw_ila_da	Export to Spr	eadsheet	1	Data menu i	tem to impo

Figure 84: Toggle the push_button_reset Signal

The **Value** field for push_button_reset is highlighted.

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

hw_vio_1 × Capture Setup	- hw_ila_1				? _ 🗆
Q ¥ ♦ + -					
Name	Value	Activity	Direction	VIO	
∿ DONT_EAT	[B] 0		Input	hw_vio_1	
BCPIO_BUTTONS_re[1:1]	[B] 0		Input	hw_vio_1	
∿a push_button_reset	1		Output	hw_vio_1	
∿ push_button_vio_1	0		Output	hw_vio_1	
> 墙 sineSel_1[1:0]	[H] 0		Input	hw_vio_1	

Figure 85: Toggle the Value of push_button_reset

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**.

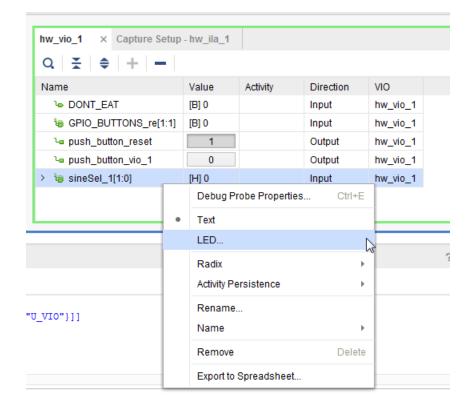


Figure 86: Change sineSel to LED

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:	🔵 Red 🗸 🗸
ОК	Cancel

Figure 87: Pick the Low Value and High Value Color of the LEDs





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	
৳ GPIO_BUTTONS_re[1:1]	[B] 0		Input	hw_vio_1	
د push_button_reset	1		Output	hw_vio_1	
∿a push_button_vio_1	0		Output	hw_vio_1	
> 🐌 sineSel_1[1:0]	[H] 0		Input	hw_vio_1	

Figure 88: Input and Output VIO Signals Displayed

- 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	
∿ DONT_EAT	[B] 0		Input	hw_vio_1	
BOPIO_BUTTONS_re[1:1]	[B] 0		Input	hw_vio_1	
∿ push_button_reset	0		Output	hw_vio_1	
✓ \u00e9 sineSel_1[1:0]	[H] 1		Input	hw_vio_1	
			Input	hw_vio_1	
∫ sineSel_1[0]	•		Input	hw_vio_1	
∿ push_button_vio_1	1		Output	hw_vio_1	

Figure 89: Toggle push_button_reset

b. Click **Run Trigger** for hw_ila_1 to capture and display the selected sine wave signal from the previous step.





Waveform - hw_ila_1		? _ □ >
Q + − & ▶ ≫	🕒 🔁	2, 23, +F H H H H H H H H H
ILA Status:Idle		1,023
Name	Value	0, , , , , , , , , , , , , , , , , , ,
> 📲 GPIO_BUTTONS_dly[1:0]	1	
V M GPIO_BUTTONS_re_1[1:0]	0	
Via [1]	0	
¹ 6 [0]	0	
₩ push_button_reset_1	0	
₩ push_button_vio	1	
> 🔩 sine[19:0]	75471	
✓ M sineSel[1:0]	1	
16 [1]	0	
¹² 4 [0]		
	<	Updated at: 2017-Mar-17 12:41:31

Figure 90: Run Trigger for hw_ila_1





Lab 6: Using 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® Design Suite Engineering Change Order (ECO) flow.



TIP: To learn more about using the ECO flow, refer to the <u>Debugging Designs Post</u> <u>Implementation Chapter</u> in the Vivado Design Suite User Guide: Programming and Debugging (UG908).

1. Open the Vivado Design Suite, and select File > Open Checkpoint.





🔶 Viva	ado 2018.1		- • ×
<u>F</u> ile	Flow <u>T</u> ools <u>W</u> indow <u>H</u> elp	Q- Quick Access	
	Project Constraints Simulation Waveform		
	Chec <u>k</u> point	> Open	
	ĮP	Open Recent	
	I <u>m</u> port		
	E <u>x</u> it		
	Create Project > Open Project > Open Example Project >		
	Tasks		
	Manage IP > Open Hardware Manager > Xilinx Tcl Store >		
	Learning Cer Documentation and Tutorials Quick Take Videos > Release Notes Guide >		
Tci C	onsole		? _ D Ľ X
	<		~ >
Туре	e a Tcl command here		
Open a	checkpoint file containing a netlist, XDC	constraints, and a physical database	

Figure 91: Opening a Checkpoint in Vivado IDE





2. Open the routed checkpoint that you created in Lab 2: Using the HDL Instantiation Method for Debugging a Design in Vivado.

🔶 Open Checkpoint	
Look in: 🕠 impl_1	- 🖉 🛠 🗐 🖳 🍋 🖉 🗙 💼 🗄
↓ Xil ▲ sinegen_demo.dcp ▲ sinegen_demo_opt.dcp ▲ sinegen_demo_placed.dcp	Recent Directories C://ivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1 File Preview File since and demonstrated d
sinegen_demo_routed.dcp	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: sinegen_demo_routed.dcp	
Files of type: Vivado Checkpoint Files (.dcp)	~
	OK Cancel

Figure 92: Open Checkpoint Dialog Box





Change the layout in the Vivado Design Suite toolbar dropdown to ECO.

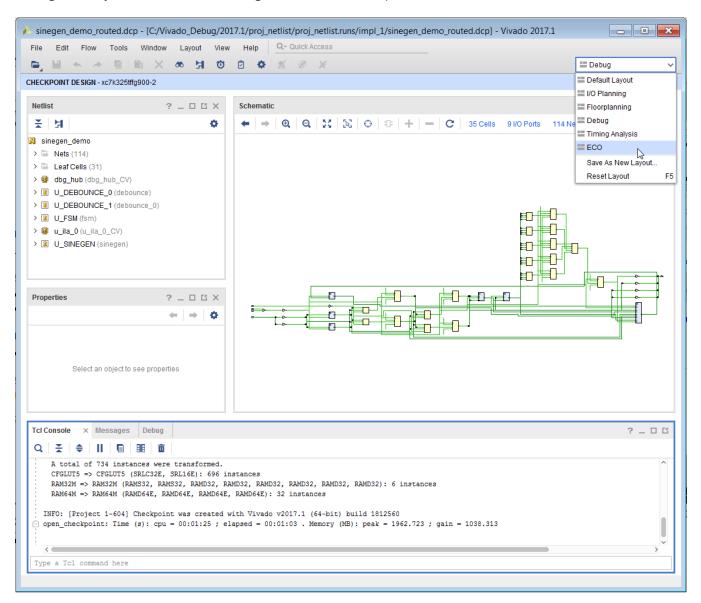


Figure 93: Change Layout to ECO

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





File Edit Flow Tools W	/indow Layout View Help Q- Quick Acc	855	
	X 👁 🔄 🕈 🖄 🖉 .	¥ ≡ ECO	
HECKPOINT DE SIGN - xc7k325tffg90	0-2		?
ECO Navigator	Scratch F Propertie Netli × ? _ 🗆 🖆	Schematic × Device × Package ×	? 🗆 🖸
Edit	조 뇌 수	$\leftarrow \rightarrow \bigcirc \bigcirc 2 2 2 0 0 0 + - 0 35 Cells 9 I/O Ports$	s [,]
Create Net Create Cell	 N sinegen_demo > ■ Nets (114) > ■ Leaf Cells (31) 		
Create Port	> 🦉 dbg_hub (dbg_hub_CV)		
Create Pin	 U_DEBOUNCE_0 (debounce) U_DEBOUNCE_1 (debounce_0) 		
Connect Net	> I U_FSM (fsm)		
Disconnect Net	> u_ila_0 (u_ila_0_CV) U_SINEGEN (sinegen)		
Replace Debug Probes			
Place Cell		₽.#₽-	
Unplace Cell			
Run			
Check ECO			
Optimize Logical Design			
Place Design			
Optimize Physical Design			
Route Design			
Report			
Edit Timing Constraints			
Report Timing Summary			
Report Clock Networks			
Report Clock Interaction			
Report DRC	Tcl Console × Messages Package Pins	I/O Ports	? _ 🗆 [
Report Utilization	Cliconsole × Messages Package Pins Q X ♦ II ■ III	IN FULS	
🛸 Report Power	INFO: [Project 1-111] Unisim Transfo:	rmation Summary:	-
Program	A total of 734 instances were trans CFGLUT5 => CFGLUT5 (SRLC32E, SRL16)	sformed.	
Save Checkpoint As		RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32): 6 instances	
Generate Bitstream		created with Vivado v2017.1 (64-bit) build 1812560	
Write Debug Probes		Created with Vivado V201.1 (04-bit) build 1812360 D1:25 ; elapsed = 00:01:03 . Memory (MB): peak = 1962.723 ; gain = 1038.313	
Open Hardware Manager			>

Figure 94: ECO Navigator Window





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		x
Use the Edit Probes button to replace one changes in the Vivado Hardware Manager,		
¥ \$ Ø C ↑ ↓		
<u>S</u> earch: Q.+		
Name	Probe	
Ch 13	「¥U_SINEGEN/sine[13] ⊘ ^	
Och 14	「¥U_SINEGEN/sine[14] ⊘	
Oh 15	「¥U_SINEGEN/sine[15] ⊘	
Oh 16	「∗ U_SINEGEN/sine[16] Ø	
• Ch 17	「¥U_SINEGEN/sine[17] ⊘	
• Ch 18	「¥U_SINEGEN/sine[18] ⊘	
Oh 19	「¥U_SINEGEN/sine[19] ⊘	
probe2 (2)		
• Ch 0	「¥ GPIO_BUTTONS_IBUF[0]	
Oh 1	「¥ GPIO_BUTTONS_IBUF[1] ⊘	
probe3 (2)		
🖲 Ch 0	「¥ GPIO_BUTTONS_db[0] ⊘	
🖲 Ch 1	「¥ GPIO_BUTTONS_db[1] ⊘	
probe4 (2)		
🖲 Ch 0	「¥ GPIO_BUTTONS_dly[0] ⊘	
🖲 Ch 1	「¥ GPIO_BUTTONS_dly[1] ⊘	
✓ probe5 (2)		
Oh 0	_「∗ GPIO_BUTTONS_re[0] // ~	
Probes changed: 0	OK Cancel	

Figure 95: Replace Debug Probes Dialog Box



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 we 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 **Replace Debug Probes** dialog box.

Replace Debug Probes		×
Use the Edit Probes button to replace one changes in the Vivado Hardware Manage		
Search: Q-		
Name	Probe	
Ch 13	_ I * U_SINEGEN/sine[13]	0
Ch 14	_	0
Och 15	_ SINEGEN/sine[15]	0
🖲 Ch 16	_ SINEGEN/sine[16]	0
Ch 17	_ SINEGEN/sine[17]	0
Ch 18	_ SINEGEN/sine[18]	0
Ch 19	_ SINEGEN/sine[19]	0
✓ ■ probe2 (2)		
Ch 0	_ SPIO_BUTTONS_IBUF[0]	0
Ch 1	_ SPIO_BUTTONS_IBUF[1]	0
✓ ■ probe3 (2)		- 11
Ch 0	_ SPIO_BUTTONS_db[0]	0
Ch 1	_ SPIO_BUTTONS_db[1]	0
✓ ■ probe4 (2)		
Ch 0	_ SPIO_BUTTONS_dly[0]	_fh
Och 1	_ SPIO_BUTTONS_dly[1]	Я
✓ ₱ probe5 (2)		
Ch 0	_ SPIO_BUTTONS_re[0]	0 ~
Probes changed: 0		
	ОК С:	ancel

Figure 96: Edit Probes Button





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**.

i Choose Nets					×
Choose nets to replace existing	probes.				4
Properties					
NAME	✓ contains	~	*	⊗ +	
<u>R</u> egular expression	earch hierarchically 📿 Di	enlav	unio	ule nets	
Of objects:		opiaj	y unitq	lue neta	
		Г			
Found: 12857		L	Fin		71
∫ <const0></const0>		^		Selected: 0 of 1	Z↓ A↓
∫ <const1></const1>					
_ Clk					
_ Clk_ibufgds 」「CLK_N					×
S CLK_P			+		Ŧ
∫ dbg_hub/ <const0></const0>		-	≠	Use the buttons on the left to copy Nets into this List	. +
∫ dbg_hub/inst/ <const0></const0>					+
_ dbg_hub/inst/BSCANID.u_) _ dbg_hub/inst/BSCANID.u_)					<u>*</u>
∫ dbg_hub/inst/BSCANID.u_)		~			
<	>				
				ок С	ancel

Figure 97: Choose Nets





 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 probes.			4
Properties	*cle	ear 🛛 🕇	
☐ Regular expression ✓ Search hierarchically ✓ Disp Of objects:	lay unio	nd	
Found: 68	1	Selected: 0 of 1	z↓
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_regs/CNT.CNT_SRL U_ila_0/inst/ila_core_inst/u_ila_regs/CNT.CNT_SRL U_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu U_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[1].mu U_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu U_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[3].mu U_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[3].mu U_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu U_ila_0/inst/ila_core_inst/U_ila_core_inst/u_ila_regs/MU_SRL[4].mu U_ila_0/inst/ila_core_i	+ ⋕	Use the buttons on the left to copy Nets into this List.	× ~ + +
		OK Can	cel

Figure 98: Choose Nets – Clear





🔥 Choose	Nets			×
Choose ne	ts to replace existing prol	Des.		4
Propertie	95			
NAME	E ~	contains ~	*clear	⊗ +
Regul	ar expression 🗸 Searc	h hierarchically 🔽 <u>D</u> ispla	y unique nets	
Of o <u>bj</u> ect				
			<u>F</u> ind	
Found: 68			Selected: 1 of 1	A∎ XI
	ub/inst/BSCANID.u_xsdb	m_id/CORE_XSDB.U	J U_DEBOUNCE_0/clear	
_	BOUNCE_0/clear 0/inst/ila_core_inst/u_ila	_regs/CNT.CNT_SRL		
∫ u_ila_	0/inst/ila_core_inst/u_ila	_regs/CNT.CNT_SRL		×
	.0/inst/ila_core_inst/u_ila .0/inst/ila_core_inst/u_ila		12	Ŧ
	0/inst/ila_core_inst/u_ila			÷
	0/inst/ila_core_inst/u_ila		Copy selected Nets into the Selection List	+
	0/inst/ila_core_inst/u_ila			±
	0/inst/ila_core_inst/u_ila			
J u_ila_	0/inst/ila_core_inst/u_ila	_regs/MU_SRL[4].mu		
		·	21	Canad
			ОК	Cancel

Figure 99: Choose Nets - Copy





8. Now click **OK** in the **Replace Debug Probes** dialog.

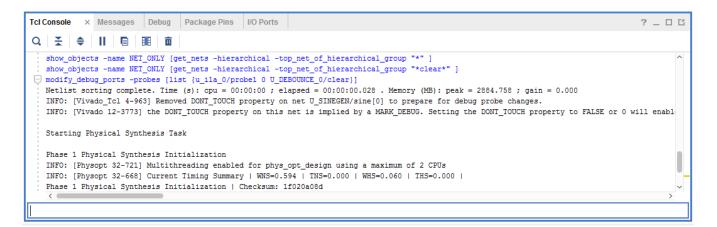
Replace Debug Probes		— ×					
Use the Edit Probes button to replace one or more debug probes. To reflect these changes in the Vivado Hardware Manager, regenerate the debug probes file (LTX).							
Search: Q.							
Name	Probe						
Och 13	_ I ¥ U_SINEGEN/sine[13]	0^					
Ch 14	_ I ¥ U_SINEGEN/sine[14]	0					
🖲 Ch 15	_ SINEGEN/sine[15]	0					
Oh 16	_ I ¥ U_SINEGEN/sine[16]	0					
Och 17	_T ¥ U_SINEGEN/sine[17]	0					
Oh 18	_ I T T U_SINEGEN/sine[18]	0					
🖲 Ch 19	_ SINEGEN/sine[19]	0					
probe2 (2)							
Ch 0	_ SPIO_BUTTONS_IBUF[0]	0					
🖲 Ch 1	_	0					
probe3 (2)		- 11					
Ch 0	_	0					
Och 1	_	0					
✓							
Och 0	_ U_DEBOUNCE_0/clear	C					
• Ch 1	_ SPIO_BUTTONS_dly[1]	0					
✓							
• Ch 0	_	0 ~					
Probes changed: 1							
	ОК С	ancel					

Figure 100: Finish Replace Debug Probes





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





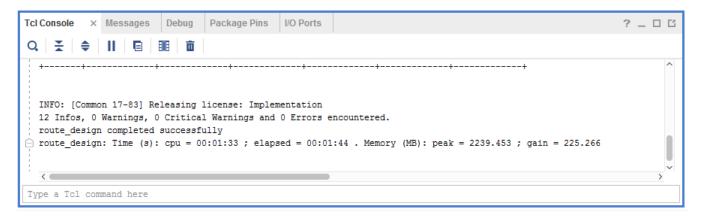


Figure 102: Route Design Messages





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.	A
Checkpoint file: 2017.1/proj_netlist/proj_netlist.runs/impl_1/checkpoint_1.d OK	

Figure 103: Save Checkpoint As Dialog Box

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.

Write Debug Probes
Write debug probes to a file.
File Name:)17.1/proj_netlist/proj_netlist.runs/impl_1/probes_2 ③
OK Cancel

Figure 104: Write Debug Probes Dialog Box





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.

A Generate Bitstream									
Create a programming file from the current	Create a programming file from the current design								
	-1								
Bit File etlist/proj_netlist.runs/impl_	1/project_sinegen_demo_routed. 📀								
Options									
-raw_bitfile		^							
-mask_file									
-no_binary_bitfile									
-bin_file									
-readback_file									
-logic_location_file									
-verbose		~							
Select an option above to see a description of it									
	ОК Саг	ncel							

Figure 105: Generate Bitstream Dialog Box

- 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 Vivado Logic Analyzer to Debug Hardware.



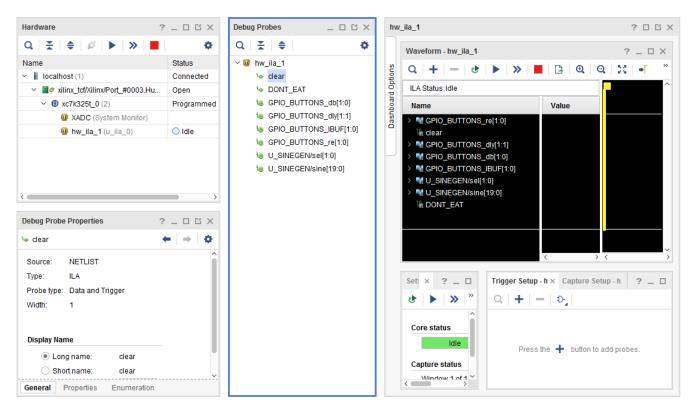


Program the device using the .bit file and .ltx files that you created in the previous steps.

in 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:							
✓ Enable end of st	artup check						
•	Program Cance	ł					

Figure 106: Program Device Dialog Box

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.

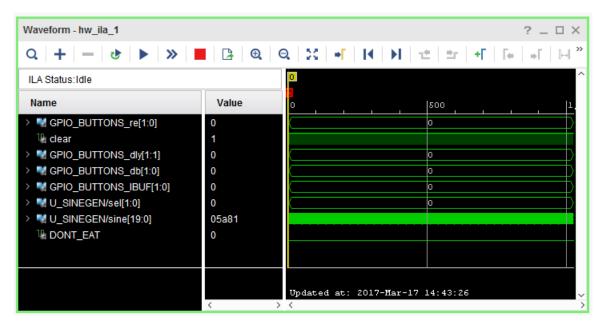


Figure 108: Running the Trigger on the ILA





Lab 7: Debugging Designs Using Incremental Compile Flow

Introduction

This lab introduces the Vivado® 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:

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

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

1. Start Vivado IDE

Load 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 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 cpuEngine 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 DE SIGN - constrs_2 xc7k70tfbg676-2 (active)								
Sources N	letlist ×		? _ 🗆 🗳					
꽃 片			•					
〉小	dcpu_adr_cpu (32)	^					
〉小	dcpu_dat_cpu (32)						
〉小	dcpu_dat_qmer	m (25)						
> 鈩	dcpu_sel_cpu (3)						
〉小	dcpu_tag_dmm	u (1)						
〉小	dcqmem_adr_q	(32) (mem						
〉小	dcqmem_dat_q	ımem (32)						
〉小	dcqmem_sel_o							
〉小	dcqmem_	Bus Net Properties	Ctrl+E					
〉①	dcqmem_ 🕷	Mark Debug	6					
〉①	dcsb_adr <u>.</u>	Unmark Debug	Ĩ					
	dcsb_sel_ 💕	Assign to Debug Port						
〉小	dout (32)	Select Driver Pin						
	dtlb_ppn (Schematic	F4					
	du_dat_c; 🎜							
	du_dat_di	Show Connectivity	Ctrl+T					
〉小		Show Hierarchy	F6					
	du_excep' 🔦	Highlight						
〉小		Unhighlight						
	dwb_dat_	Mark						
	dwb_dat_	Unmark	Ctrl+Shift+M					
	E (1)							
	ex_insn (2	Go to Source	F7					
> -î-	fifo_dat_o (3)		~					

Figure 109: Set MARK_DEBUG Property

Alternatively use can use the Tcl command below 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.

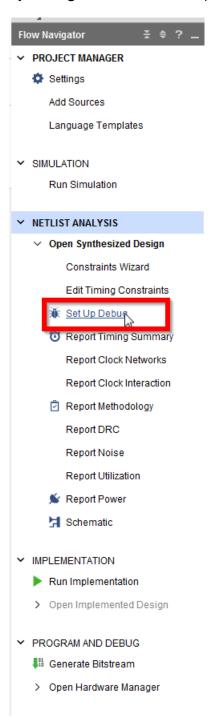


Figure 110: Setup Debug from Flow Navigator





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

Set Up Debug						×
lets to Debug The nets below will be debugged with ILA cores. To vindows, then drag them to the list or click "Add Sel		nd Nets to Ad	d". You can also sel	ect ne	ets in the Netlist or other	4
Q 素 ≑ ㎡ Ⅲ + −						٥
Name	Clock Domain	Driver Cell	Probe Type			
> Jf≅ cpuEngine/dcpu_dat_qmem (25)	clkgen/cpuClk	FDRE	Data and Trigger	~		
> 小電 cpuEngine/dcqmem_adr_qmem (32)	clkgen/cpuClk	FDRE	Data and Trigger	Υ.		
> 小章 cpuEngine/dcqmem_dat_qmem (32)	clkgen/cpuClk	FDRE	Data and Trigger	Υ.		
> 小☆ cpuEngine/dcqmem_sel_qmem (4)	clkgen/cpuClk	FDRE	Data and Trigger	Υ.		
> 小☆ cpuEngine/du_dsr (11)	clkgen/cpuClk	FDCE	Data and Trigger	Υ.		
› √ft☆ cpuEngine/dvr00 (6)	clkgen/cpuClk	FDCE	Data and Trigger	Υ.		
Find Nets to Add					Nets to debu	ua: 110
		<	Back Ne	ext >		-
?		<	Back Ne	ext >	Einish Ca	ancel

Figure 111: Set Up Debug Nets to Debug

- 11. When ILA Core Options screen appears, click Next again.
- 12. When **Set Up Debug Summary** screen appears, ensure that **1 debug core** is created and click **Finish**.





13. Check the Debug widow to ensure that **u_ila_0** core has been inserted into the design.

Tcl Console Messages Log Reports Design Ru	ins Debug	×		? _ 🗆 🛙
Q ≍ ≑ ӂ + ≤				0
Name	Driver Cell	Driver Pin	Probe Type	
 ⁴ 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)				
Debug Cores Debug Nets				

Figure 112: Check u_ila_0 Core

14. Save the new debug XDC commands by selecting **File > Save Constraints** 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.

0

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 figure below. 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 > 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**.

4	Create New Runs								
	Configure Implementation Runs Create and configure one or more implementation runs using various parts, constraints, flows and strategies								
	Create Implem	entation Runs							
	+ -								
	Name	ame Constraints Set Part Strategy							
	impl_2 🔻	🛅 constrs_2 (act 🗸	🛑 xc7k70tfbg67 🗸	🏂 Vivado Implementation Defaults (Vivado Implementation 2 👻	 Image: A start of the start of				
				R	uns to create: 1				
(?			< <u>B</u> ack <u>N</u> ext > <u>F</u> inish	Cancel				

Figure 113: Configure Implementation Runs





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

Create New Runs				×
Launch Options Configure hosts for launching runs, and/or set advanced launch options				4
Launch <u>d</u> irectory: Solution Content Content of Content				~
 Launch runs on local host: Number of jobs: 4 Generate scripts only Do not launch now 				
•	< <u>B</u> ack	Next >	Einish	Cancel

Figure 114: Launch Options

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 Messa	ges Log	Reports Package Pins De	esign Runs	×	Power	Timir	ng Met	thodology D	RC							? _	0 6
Q 素 ≑ I4	≪ ►	» + %															
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Strate
✓ 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	Vivad
▷ impl_2	constrs_2	Not started															Vivad
<																	>

Figure 115: New Design Runs





Step 4: Making Incremental Debug Changes

In this step, in order 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. So now, you will debug the nets fftEngine/fifo_out[*].
- 3. Apply the MARK_DEBUG property to this bus in the netlist window.

Sources	Netlist ×		? _ 🗆 🖸
꽃 너			¢
🕅 top			^
> 🗎 Nets	s (4564)		
🔸 🖨 Leat	f Cells (223)		
🔉 🔳 clkg	en (clock_gen	erator)	
> 🗷 cpuł	Engine (or1200)_top)	
> 📴 dbg	_hub (dbg_hut	_CV)	
🗸 🗷 fftEn	gine (fftTop)		
~ 🖻 I	Nets (3331)		
> -	🖵 A(16)		
> -	手 C (16)		
> -	🖵 D (32)		
> -	厅章 fifo_out (32)	
> -	🖵 I3 (32)		
> -	🖵 14 (32)		
> -	🖵 15 (32)		
> -	🖵 16 (32)		
> -	🖵 17 (32)		
	E 10 (22)		~

Figure 116: Netlist fifo_out

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 cores.

intersection Action Act				×
Existing Debug Nets Choose how to handle existing nets connected to debug cores.				4
 Continue debugging 110 nets connected to existing debug core Only debug new nets Disconnect all nets and remove debug cores 				
?	< <u>B</u> ack	Next >	Einish	Cancel

Figure 117: Existing Debug Nets





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

🏄 Set Up Debug				×
Additional Debug Nets Choose additional nets to debug.				4
 Debug 32 unassigned debug nets Debug 32 selected nets 				
	< <u>B</u> ack	<u>N</u> ext ≻	Einish	Cancel

Figure 118: Set Up Debug Additional Debug Nets





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

Q 素 ♦ № M + -					0
Name	Clock Domain	Driver Cell	Probe Type		
≻ 『「★ cpuEngine/dcpu_dat_qmem (25)	clkgen/cpuClk	FDRE	Data and Trigger	~	
> 「「★ cpuEngine/dcqmem_adr_qmem (32)	clkgen/cpuClk	FDRE	Data and Trigger	~	
> 师微 cpuEngine/dcqmem_dat_qmem (32)	clkgen/cpuClk	FDRE	Data and Trigger	~	
> 师樂 cpuEngine/dcqmem_sel_qmem (4)	clkgen/cpuClk	FDRE	Data and Trigger	~	
> 师樂 cpuEngine/du_dsr (11)	clkgen/cpuClk	FDCE	Data and Trigger	~	
> 师₩ cpuEngine/dvr00 (6)	clkgen/cpuClk	FDCE	Data and Trigger	~	
〉 师 fftEngine/fifo_out (32)	clkgen/fftClk	RAMB36	Data and Trigger	Υ.	
Find Nets to Add					Nets to debug: 14

Figure 119: Viewing Additional Debug Nets

- 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 > Save Constraints 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 Compile** from the popup menu. The Set Incremental Compile dialog box opens.
- 3. Click the **Browse** button in the **Set Incremental Compile** dialog box, and browse to the ./project_cpu_incremental.runs/impl_1 directory.
- 4. Select top_routed.dcp as the incremental compile checkpoint.

Set Incremental Compile	×
Enable/disable incremental compiling by choosing or clearing a checkpoint file.	A
Use checkpoint:	

Figure 120: Set Incremental Compile

- 5. 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.
- 6. 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.

7. From the Flow Navigator, select Run Implementation.

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





8. 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	ges Log	Reports Design Runs × P	ower	DRC	Methodo	logy	Timing										? _ O C
Q 素 ≑ I∢	« >	» + %															
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	Implementation 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_design 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 Imp

Figure 121: Design Runs

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

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

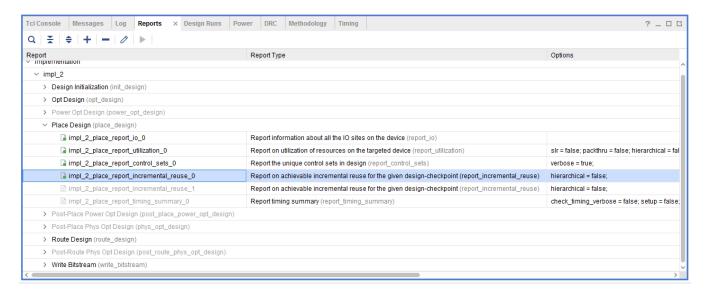


Figure 122: Opening Incremental Reuse Report

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.





Proje	ct Summar	y × Device × impl_	2_place_report_incremen	tal_reuse_0 - impl_2 ×		0 6
Q,	H 🍝	🔺 🕹 🖪 🖬 😒	× // ∎ Ω			Read-only 🏾 🖨
	Copyright	t 1986-2018 Xilinx, Inc	. All Rights Reserved	•		^
4 5 7 8 9	Date Host Command Design Device Design	d : report_increment : top	34:29 2018 nning 64-bit Service 1	-		
11 12 13	Increment	tal Implementation Info: Contents	rmation			
16 17 18	3. Compar	Summary ence Checkpoint Informa rison with Reference Rum				
20 21	4. Non Re 1. Reuse					
25	Type	Matched % (of Total)	Reuse % (of Total)	Fixed % (of Total)	Total	
27 28 29	Cells Nets Pins	95.69 95.80	88.59 80.85 86.48	0.31 0.00	46475 36769 189880	
31 32 33		+	+			
35 36 37	+	ence Checkpoint Informat		incommental (project	anu inavamantal wuna/ima	

Figure 123: Incremental Reuse Report Sample

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 Vivado Serial Analyzer to Debug Serial Links

Introduction

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 as listed below:

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

The customizable LogiCORE[™] 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 in the Vivado Integrated Design Suite.
- 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 devices. The focus for this tutorial is on Kintex®-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 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.

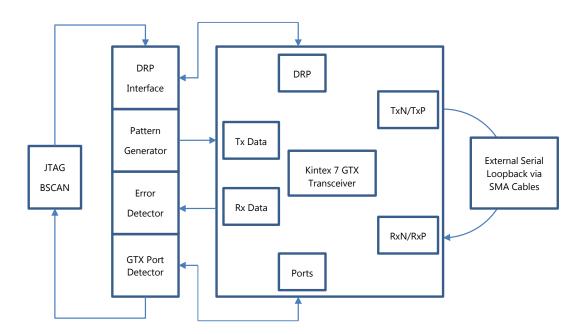


Figure 124: IBERT Design Flow





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.

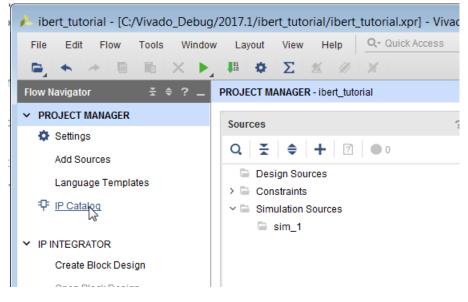


Figure 125: Opening the Vivado IP Catalog

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

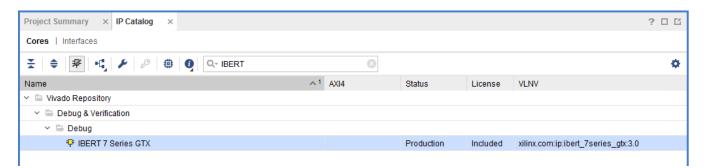


Figure 126: Instantiating the IBERT IP from the Vivado IP Catalog

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 (Gbps) 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.

n Customize IP								×
IBERT 7 Series GTX (3.0)								A
Documentation IP Location C Switch to Defaults								
Show disabled ports	Component Name ib	ert_7series_gtx_0						8
	Protocol Definition	Protocol Selection	Clock Settings	Summary				
RXN_I[3:0] RXP_I[3:0] TXN_0[3:0] - GTREFCLKD_I[0:0] TXP_0[3:0] - GTREFCLK1_I[0:0] RXOUTCLK_0 -	Silicon Version	per of quads available	for this device is	4		1		<u> </u>
	Protocol Custom 1	LineRate(Gbps)	DataW		Refclk(MHz)	Quad Count	•	Quad PLL
			<u><u> </u></u>				ОК	Cancel

Figure 127: Setting the Protocol Definition on the IBERT Core

- 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								×
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	Sun	nmary			
	Please select Proto	col-Quad combination						
	GTX Location	Protocol Sel	ected		Refclk Selection		TXUSRCLK Source	
	QUAD_115	None		•	None	•	Channel 0	Ŧ
	QUAD 116	None		v	 None 		Channel 0	Ŧ
RXN_[(7:0] RXP_[(7:0] TXN_0(7:0]	QUAD_117	Custom 1 / 8 Gbps		•	MGTREFCLK0 117		Channel 0	
GTREFCLK0_[[1:0] TXP_0[7:0]	QUAD_118	Custom 1 / 8	Custom 1 / 8 Gbps		MGTREFCLK0 117		Channel 0	
GTREFCLKI_[11:0] RXOUTCLK_O SYSCLK_I								
							ОК Са	ancel

Figure 128: Setting the Protocol Selection on the IBERT Core

- 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.





n Customize IP							×
IBERT 7 Series GTX (3.0)							4
Occumentation 📄 IP Location C Switch to Defaults							
Show disabled ports	Component Name ib	ert_7series_gtx_0 Protocol Selection	n Clock Settings Sum	mary			۲
	RXOUTCLK Probe	CLK Probes					
RXN_I[7:0]	Clock Type	Source	I/0 Standard	P Package Pin	N Package Pin	Frequency(MHz)	
RXP_[[7:0] TXN_0[7:0]	System Clock	External *	DIFF SSTL15	AD12	AD11 🛞	200.00	8
GTREFCLK0_[[1:0] TXP_0[7:0] - GTREFCLK1_[[1:0] RXOUTCLK_0 SYSCLK_I	System Clock Term			N	76	m	
					C	DK Car	icel

Figure 129: Specifying Clock Settings for the IBERT Core

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

IBERT 7 series GTX (3.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (bert_7 series gtv.0) Image: Component Name (component Name (compone)) Image	🔥 Customize IP							×	
Show disabled ports Component Name libert_7series_gtt_0 RXN_[[7:0] Protocol Definition RXN_[[7:0] TXN_0[7:0] RXP_[[7:0] TXN_0[7:0] GTREFCLK0_[(1:0) TXV_0[7:0] GTREFCLK1_[1:0] RXOUTCLK_0 SYSCLK_1 Herein (P in : AD11) MMCM Count 1 RetClk Sources External (N Pin : AD11) MMCM Count 1 RetClk Sources 1	IBERT 7 Series GTX (3.0)							4	
RXN_[[7:0] TXN_0[7:0] RXP_[[7:0] TXN_0[7:0] GTREFCLH0_[[1:0] TXP_0[7:0] GTREFCLL1_[[1:0] TXP_0[7:0] System Clock Source External (N Pin : AD12) System Clock Source External (N Pin : AD11) QUAD Count 2 MMCM Count 1 RefClk Sources 1	Occumentation IP Location C Switch to Defaults								
RXN_[[7:0] Number of Protocols 1 RXP_[[7:0] TXN_0[7:0] System Clock Source External (P Pin : AD12) System Clock Source External (V Pin : AD11) QUAD Count 2 GTREFCLK3_[[1:0] TXP_0[7:0] 1 RefClik Sources 1 System Clock Source 1 1 1 1 GTREFCLK1_[1:0] RXOUTCLK_0 1 1 1 SYSCLK_1 Image: Source Surce Su	Show disabled ports	Component Name	ibert_7series_gtx_0					8	
RXN_[[7:0] Image: Signame and the second		Protocol Definition	Protocol Selection	Clock Settings	Summary				
RXN_[[7:0] RXP_[[7:0] TXN_0[7:0] GTREFCLK0_[[1:0] TXP_0[7:0] GTREFCLK1_[[1:0] RXOUTCLK_0 System Clock Source External (P Pin : AD12) System Clock Source DUAD Count 2 MMCM Count 1 RefClk Sources 1		IBERT Design Su	nmary						
RXN_[[7:0] TXN_0[7:0] RXP_[[7:0] TXN_0[7:0] GTREFCLK0_[[1:0] TXP_0[7:0] GTREFCLK1_[(1:0] RXOUTCLK_0 System Clock Source External (N Pin : AD11) QUAD Count 2 MMCM Count 1 RefClk Sources 1									
RXN_[/7:0] TXN_0[7:0] RXP_[/7:0] TXN_0[7:0] GTREFCLK0_[[1:0] TXP_0[7:0] GTREFCLK1_[1:0] RXOUTCLK_0 SYSCLK_I RefCik Sources									
GTREFCLK0_[[1:0] TXP_O[7:0] - GTREFCLK1_[[1:0] RXOUTCLK_0 SYSCLK_1									
GTREFCLK1_[[1:0] RXOUTCLK_0		MMCM Coun	t		1				
		RefClk Source	es		1				
OK Cancel	- SYSCLK_I								
OK Cancel									
OK Cancel									
OK Cancel									
OK Cancel									
OK Cancel									
OK Cancel									
OK Cancel									
OK Cancel									
							OK	Cancel	
							·		

Figure 130: IBERT Core Summary Page





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

A Generate Output Products
The following output products will be generated.
Preview
Q ≚ ≑
 ✓ Ibert_7series_gtx_0.xci (Global) Instantiation Template Inst
Synthesis Options
● <u>G</u> lobal
Out of context per IP
Run Settings
Number of jobs: 8 🗸
Apply Generate Skip

Figure 131: Generate Output Products





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

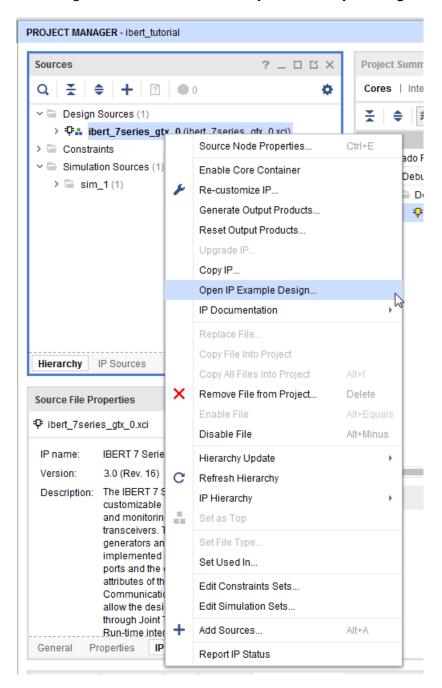


Figure 132: Open Example IP Design Menu Item





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 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	

Figure 133: Open IP Example Design Dialog Box





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**.

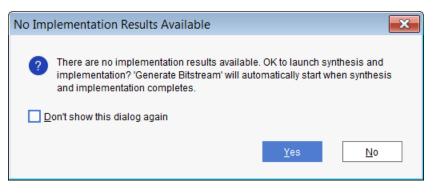


Figure 134: No Implementation Results Available Dialog Box

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

2. Select Open Hardware Manager, and click OK.

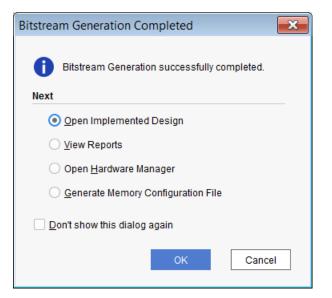


Figure 135: Bitstream Generation Completed Dialog Box





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

/ ibert_7series_gtx_0_ex - [c:/Vivado_Debug/2017/ib	pert_7series_gtx_0_ex/ibert_7series_gtx_0_ex.xpr] - Vivado	
File Edit Flow Tools Window Layout View	Help Q- Quick Access	write_bitstream Complete 🛛 🗸
	💋 🖉 🐹 Dashboard	🗮 Serial I/O Analyzer 🗸 🗸
Flow Navigator 😤 🔶 ? 💷 HARDWARE MANAGER - u	nconnected	? ×
PROJECT MANAGER O No hardware target is o	ipen. Open target	
Settings	? _ O Ľ X	
Add Sources		
Language Templates		
₽ IP Catalog		
V IP INTEGRATOR	No content	
Create Block Design		
Open Block Design		
Generate Block Design		
Properties	? _ 🗆 🗹 ×	
✓ SIMULATION	$\leftarrow \rightarrow \diamond$	
Run Simulation		
✓ RTL ANALYSIS		
> Open Elaborated Design	oject to see properties	
✓ SYNTHESIS		
Run Synthesis		
> Open Synthesized Design	es Serial I/O Links × Serial I/O Scans	2 _ 0 13
✓ IMPLEMENTATION		
Run Implementation		
> Open Implemented Desig		
Y PROGRAM AND DEBUG	No content	
↓ ^{tt} i Generate Bitstream		
✓ Open Hardware Manage		
Open Target		

Figure 136: Hardware Manager Window





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.



Figure 137: Open a New Hardware Target

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

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

Figure 138: Open New Hardware Target Wizard

2. In the Connect to field, choose Local server. Click Next.





🍌 Open New I	w 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.							
<u>C</u> onnect to:	D: 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 Lexis	Cancel					

Figure 139: Vivado CSE Server Name Page





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						×
Select Hardward Select a hardward expected devices,	e target from the				oriate JTAG clock (TCK) frequency. I	f you do not see the	A
Hardware <u>T</u> arg	ets							
Туре	Name		JTAG Clock Fre	quency				
xilinx_tcf	Xilinx/Port_#00	03.Hub_#0004	6000000	~				
			Add Vilia	v Virtual	Cable (XVC)			
			//dd /dill	ix virtual				
Hardware <u>D</u> evi	ces (for unknov	vn devices, spec	ify the Instructio	n Registe	er (IR) length)			
Name	ID Code	IR Length						
@ xc7k325t_(33651093	6						
Hardware serve	er: localhost:312	!1						
(?)					< Back	Next >	Finish	Cancel
U					Daon	<u></u>	Lunon	Ganoor

Figure 140: Select Hardware Target Page





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

🅕 Open New Hardware	e Target	×
HLx Editions	 Copen Hardware Target Summary Hardware Server Settings: Server: localhost3121 Target Settings: Target settings: Target: xilinx_tcf/Xilinx/Port_#0003.Hub_#0004 Frequency: 6000000 	
E EXILINA ALL PROGRAMMABLE.	To connect to the hardware described above, click Finish < Back Mext > Einish C	Cancel

Figure 141: Open Hardware Target Summary Dialog Box

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

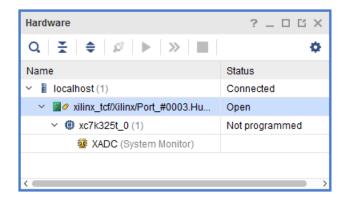


Figure 142: Hardware Window Showing the XC7K325T Device on the KC705 Board

6. Select **XC7K325T_0(0)** in the **Hardware** window, right-click and select **Program Device**.





Hardware			? _ 🗆 🖒 X	
Q ₹ \$ ¢	2	▶ ≫ ■	•	
Name			Status	
Y 📱 localhost (1)			Connected	
✓ Ø xilinx_tcf/Xi	ilinx/P	ort_#0003.Hu	Open	
✓ ∅ xc7k325t W XADC		Hardware Devic	ce Properties	Ctrl+E
		Program Device	e	
		Verify Device		hs
	С	Refresh Device		
Hardware Device Pro		Add Configurati	on Memory Device	
xc7k325t_0		Boot from Confi	guration Memory Device	
Name: p		Program BBR K	(ey	
Part: >		Clear BBR Key.		
ID code:		Program eFUSE	E Registers	
IR length:		Export to Spread	dsheet	
General Propertie	s		· · · · ·	

Figure 143: Program Target 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:	t_7series_gtx_0_ex.runs/impl_1/example_ibert_7series_gtx_0.bit	3 8				
Debu <u>q</u> probes file:						
✓ Enable end of st	tartup check					
?	<u>P</u> rogram C	ancel				

Figure 144: Program Device Dialog Box with .bit File





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**.

Hardware	? _ 🗆 🖒 X
$Q \mid \underbrace{\star} \mid \diamondsuit \mid \varnothing \mid \triangleright \mid \gg \mid \blacksquare \mid$	•
Name	Status
 Iocalhost (1) 	Connected
✓ ✓ ✓ ✓ × xilinx_tcf/Xilinx/Port_#0003.Hu	Open
 v (2) xc7k325t_0 (2) 	Programmed
🐼 XADC (System Monitor)	
 IBERT (IBERT) 	
✓ Ŋ Quad_117 (5)	
COMMON_X0Y2	Locked
NGT_X0Y8	No Link
NGT_X0Y9	No Link
NGT_X0Y10	No Link
NGT_X0Y11	No Link
✓ Ŋ Quad_118 (5)	
COMMON_X0Y3	Locked
NGT_X0Y12	No Link
NGT_X0Y13	No Link
NGT_X0Y14	No Link
NGT_X0Y15	No Link

Figure 145: The Hardware Window Showing the QUADS after Device Programming





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			? _ 🗆	Ц×	
Q ¥ ♦ ∅ ▶ ≫				ø	
Name		St	atus		
 Iocalhost (1) 		Co	onnected		
✓ ✓ × xilinx_tcf/Xilinx/Port_#000	3.Hu	. Op	pen		
 xc7k325t_0 (2) 		Pr	rogrammed		
🖉 XADC (System Monit	or)				
🗸 📴 IBERT (IBERT)		IBER	T Core Prop	erties	Ctrl+E
✓ 🖏 Quad_117 (5)					
COMMON_X0Y			te Links		L.
NGT_X0Y8		Auto-	-detect Links		
NGT_X0Y9		Seria	al I/O Links		
NGT_X0Y10		Seria	al I/O Scans		
NGT_X0Y11		Com	Commit Properties		
✓ 🖏 Quad_118 (5)	с		esh Serial I/O		
	C	Relie	esti Senari/O	Objects	
NGT_X0Y12		Sele	Select		F
NGT_X0Y13		Expo	rt to Spreads	heet	
NGT_X0Y14		N	o Link		
NGT_X0Y15		N	o Link		
<					
					1

Figure 146: Create Links

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, ther	1 click the Add button on the New Links toolbar.
TX GTs	RX GTs
Search: 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	
Press the	+ button to Add Link
✓ Create link group	
Link group description: Link Group 0	8
Open Serial I/O Analyzer layout	
(?)	OK Cancel

Figure 147: Selecting the Transceiver Pairs for Creating New Links





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

TX GTs		RX GTs				
Search: Q-		Search: Q-				
▷ MGT_X0Y9	/TX (xc7k325t_0/Quad_117)	MGT_X0Y9/RX (xc7k325t_0/Qua	d_117)			
_	0/TX (xc7k325t_0/Quad_117)	MGT_X0Y10/RX (xc7k325t_0/Qu				
MGT_X0Y11/TX (xc7k325t_0/Quad_117)		MGT_X0Y11/RX (xc7k325t_0/Quad_117)				
_	2/TX (xc7k325t_0/Quad_118)	MGT_X0Y12/RX (xc7k325t_0/Quad_118)				
_	3/TX (xc7k325t_0/Quad_118)	MGT_X0Y13/RX (xc7k325t_0/Quad_118) MGT_X0Y14/RX (xc7k325t_0/Quad_118)				
_	4/TX (xc7k325t_0/Quad_118) 5/TX (xc7k325t_0/Quad_118)	MGT_X0Y14/RX (xc7k325t_0/Qu				
lew Links						
+ -						
Description	ТХ	RX	Internal Loopback			
% Link 0	MGT_X0Y8/TX (xc7k325t_0/Quad_117)	MGT_X0Y8/RX (xc7k325t_0/Quad_117)	\checkmark			

Figure 148: Create Links Dialog Box

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 Me	essages Serial I/O Li	nks × Seri	al I/O Scan	s					
Q ¥ ♦	+								
Name	Create Links	RX	Status	Bits	Errors	BER	BERT Reset	TX Pattern	RX Pattern
🖨 Ungrouped l	Create Link Group	1							
👻 🐵 Link Group S							Reset	PRBS 7-bit 🗸	PRBS 7-bit 😒
% Link 0		MGT_X0Y8/RX	7.988 G	1.356	2.74E10	2.021	Reset	PRBS 7-bit 🐱	PRBS 7-bit 🐱
	Create Sweep								

Figure 149: Create Link Groups for Other Transceiver Pairs

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**.





RX Internal Loopback MGT_X0Y9/RX (xc7k325t_0/Quad_117) ✓ MGT_X0Y10/RX (xc7k325t_0/Quad_117) ✓			RX GTs	
MGT_X0Y9/RX (xc7k325t_0/Quad_117) ✓ MGT_X0Y10/RX (xc7k325t_0/Quad_117)	Search: Q-		Search: Q.*	
MGT_X0Y9/RX (xc7k325t_0/Quad_117) ✓ MGT_X0Y10/RX (xc7k325t_0/Quad_117)				
MGT_X0Y9/RX (xc7k325t_0/Quad_117) MGT_X0Y10/RX (xc7k325t_0/Quad_117)				
MGT_X0Y9/RX (xc7k325t_0/Quad_117) MGT_X0Y10/RX (xc7k325t_0/Quad_117)				
MGT_X0Y9/RX (xc7k325t_0/Quad_117) MGT_X0Y10/RX (xc7k325t_0/Quad_117)				
MGT_X0Y9/RX (xc7k325t_0/Quad_117) MGT_X0Y10/RX (xc7k325t_0/Quad_117)				
MGT_X0Y9/RX (xc7k325t_0/Quad_117) MGT_X0Y10/RX (xc7k325t_0/Quad_117)				
MGT_X0Y9/RX (xc7k325t_0/Quad_117) MGT_X0Y10/RX (xc7k325t_0/Quad_117)				
MGT_X0Y9/RX (xc7k325t_0/Quad_117) ✓ MGT_X0Y10/RX (xc7k325t_0/Quad_117)	ew Links			
MGT_X0Y9/RX (xc7k325t_0/Quad_117) ✓ MGT_X0Y10/RX (xc7k325t_0/Quad_117)	+ -			
) MGT_X0Y10/RX (xc7k325t_0/Quad_117)	Description	тх	RX	Internal Loopback
	% Link 1	MGT_X0Y9/TX (xc7k325t_0/Quad_117)	MGT_X0Y9/RX (xc7k325t_0/Quad_117)	\checkmark
MCT_X0Y11/BX (vc7k325t_0/0uad_117)	% Link 2	MGT_X0Y10/TX (xc7k325t_0/Quad_117)	MGT_X0Y10/RX (xc7k325t_0/Quad_117)	\checkmark
	% Link 3	MGT_X0Y11/TX (xc7k325t_0/Quad_117)	MGT_X0Y11/RX (xc7k325t_0/Quad_117)	\checkmark
) MGT_X0Y12/RX (xc7k325t_0/Quad_118)	S Link 4	MGT_X0Y12/TX (xc7k325t_0/Quad_118)	MGT_X0Y12/RX (xc7k325t_0/Quad_118)	\checkmark
MCT_V0V12/DV (vc7k225t_0/0uod_110)	% Link 5	MGT_X0Y13/TX (xc7k325t_0/Quad_118)	MGT_X0Y13/RX (xc7k325t_0/Quad_118)	\checkmark
MG1_A0113/RA (AC/A323L_0/QUB0_118)	% Link 6	MGT_X0Y14/TX (xc7k325t_0/Quad_118)	MGT_X0Y14/RX (xc7k325t_0/Quad_118)	\checkmark
			MCT X0V15/DX (xc7k225t 0/0uad 119)	\checkmark
) MGT_X0Y14/RX (xc7k325t_0/Quad_118)	% Link 7	MGT_X0Y15/TX (xc7k325t_0/Quad_118)	MG1_A0113/(AC/R5251_0/Quad_110)	
) MGT_X0Y12/RX (xc7k325t_0/Quad_118)	ବ୍ତ Link 1 ବ୍ତ Link 2 ବ୍ତ Link 3 ବ୍ତ Link 4 ବ୍ତ Link 5	MGT_X0Y10/TX (xc7k325t_0/Quad_117) MGT_X0Y11/TX (xc7k325t_0/Quad_117) MGT_X0Y12/TX (xc7k325t_0/Quad_118)	MGT_X0Y10/RX (xc7k325t_0/Quad_117) MGT_X0Y11/RX (xc7k325t_0/Quad_117) MGT_X0Y12/RX (xc7k325t_0/Quad_118)	
	-			
	ち Link 6			
			MCT_X0V15/PX (vc7k325t_0/Ouad_119)	\checkmark
) MGT_X0Y14/RX (xc7k325t_0/Quad_118)	🗞 Link 7	MG1_X0Y15/TX (XC/K325t_0/Quad_118)	MOT_AUTIONA (ACTROZOL_ORGANA_TTO)	

Figure 150: Create Link Dialog Box to Create the Second Link Group





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

Tcl Console Messages	Serial I/O Lini	ks × Serial I	/O Scans						
Q ₹ ≑ †									
Name	ТХ	RX	Status	Bits	Errors	BER	BERT Reset	TX Pattern	R
Ungrouped Links (0)									
Y 🚳 Link Group SMA (1)							Reset	PRBS 7-bit 🗸	Ρ
% Link 0	MGT_X0Y8/TX	MGT_X0Y8/RX	7.988 Gbps	1.343E12	2.645E11	1.969E-1	Reset	PRBS 7-bit 🐱	Ρ
✓ I Link Group Internal							Reset	PRBS 7-bit 🗸	Ρ
S Link 1	MGT_X0Y9/TX	MGT_X0Y9/RX	7.987 Gbps	3.805E12	2.079E12	5.465E-1	Reset	PRBS 7-bit 🗸	Ρ
% Link 2	MGT_X0Y10/TX	MGT_X0Y10/RX	7.988 Gbps	3.805E12	2.175E12	5.715E-1	Reset	PRBS 7-bit \vee	Ρ

Figure 151: Links Window after Link Groups are Created

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 <u>Vivado Design</u> <u>Suite User Guide: Programming and Debugging (UG908)</u>.

- 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 Proper	lles			
Tcl Console Me		Link Properties	Ctrl+E	ans
Q ≚ ♦	×	Delete	Delete	
· · · · ·		Create Links		
Name				us
🗅 Ungrouped I		Create Link Group		
👻 🚳 Link Group §		Create Stan		
% Link 0		Create Sweep		Gbps
👻 🚳 Link Group I		Commit Properties		
% Link 1	С	Refresh Serial I/O Objects		Gbps

Figure 152: Creating a 2D Scan for Link 1

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 <u>Vivado Design Suite User Guide: Programming and Debugging (UG908)</u>.





i Create Scan						
Set the description and other properties to create and optionally run a scan on the selected link.						
Link: L	_ink 0 (MGT_	X0Y8/TX, MGT_X0Y8/RX)				
<u>D</u> escription:	Scan 0 🙁					
Scan Propert	ties					
<u>S</u> can type:	:	2D Full Eyescan 🗸				
<u>H</u> orizontal increment:		8 ~				
H <u>o</u> rizontal range:		-0.500 UI to 0.500 UI				
<u>V</u> ertical in	crement:	8 ~				
V <u>e</u> rtical ra	nge:	100% ~				
Dwell						
• <u>B</u> ER:	1e-5	~				
◯ <u>T</u> ime:	0 \$					
<mark>√</mark> <u>R</u> un scan						
?		OK Cancel				

Figure 153: The Create Scan Dialog Box





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

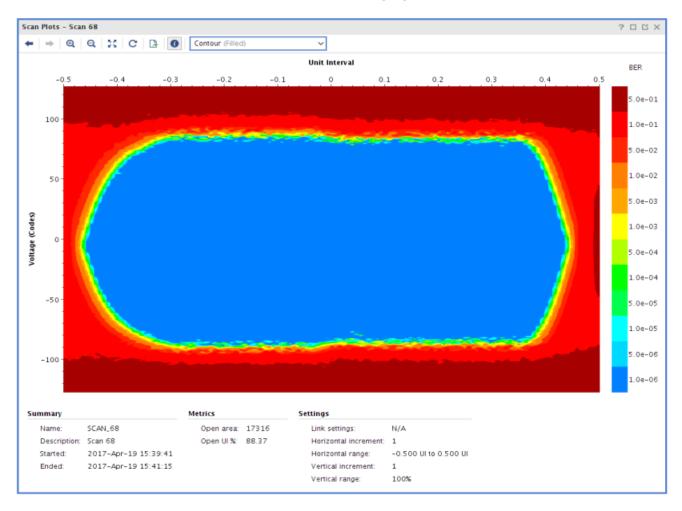


Figure 154: 2D Scan Plot

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.





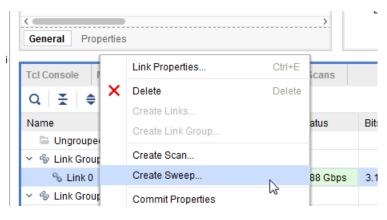


Figure 155: Create a Sweep Test

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**.





Create Sweep						
Select the sweep properties and values to create and optionally run a set of scans on the selected link.						
Link: Link 0 (MGT_X0Y8/TX, MGT_X0Y8/RX)						
cription: Sweep 0						
Scan Properties						
Scan type: 2D Full Eyescan						
Horizontal increment 8						
Horizontal range: −0.500 UI to 0.500 UI 🗸						
Vertical increment: 8						
Vertical range: 100%						
Dwell						
● BER: 1e-5 ✓						
Sweep Properties						
Sweep mode: Semi Custom V For each property select values to be swept. The sweep will cover all combinations of property values	h.					
Set Properties & Values Preview 81 Scans						
$+ - \pm \mp $						
Order Property Name Values to Sweep	# of Values					
% 1 RXTERM ▼ 100 mV,550 mV,1100 mV	• 3					
S 2 TXDIFFSWING ▼ 269 mV (0000),741 mV (0111),1119 mV (1111)	• 3					
S 3 TXPOST ▼ 0.00 dB (00000),4.08 dB (01111),12.96 dB (11111)	• 3					
S 4 TXPRE ▼ 0.00 dB (00000),4.08 dB (01111),6.02 dB (11111)	• 3					
Reset RX after applying Settings for each scan						
✓ Run sweep						
(?)	Cancel					

Figure 156: Create Sweep Dialog Box





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 complete.

Tcl Console Messages	Serial	I/O Links Serial I/O Scans ×								? _ 🗆 [
Q ≚ ♦ ▶ ■	B	B								
Name	Link	Link Settings	Reset RX	Scan Type	Status	Progress	Open Area	Open UI %	Horz Incr	Horz Range
> 🗁 Scans (4)										
🗸 🚽 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 🗸	-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 V	-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 ~	-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 ~	-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 ~	-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 🗸	-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) TXDIFESWING (741 mV (0111)) TXPOST (0.00 dB (00000)) TXPRE (2d full eve	Done	100%	10112	77.78	8 ¥	-0.500 UI to 0.500 UI

Figure 157: Sweep Test Results in the Scans Window

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

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

Figure 158: Displaying Scan Plots





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

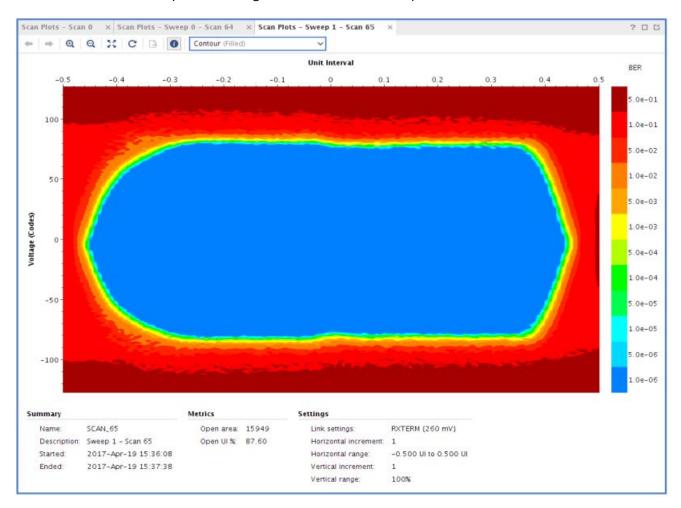


Figure 159: Analyzing the Results of Individual Scans





Lab 9: Using Vivado ILA Core to Debug JTAG-AXI Transactions

Introduction

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[™] IP JTAG-AXI core is a customizable core that can generate AXI transactions and drive AXI signals internal to 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® logic analyzer feature.

Key Features

- AXI4 master interface
- Option to select AXI4-Memory Mapped and AXI4-Lite interfaces
- User controllable AXI read and write enable
- User Selectable AXI datawidth : 32 and 64
- User Selectable AXI ID width up to four bits
- Vivado Integrated Logic Analyzer Tcl Console interface to interact with hardware

Additional Documentation

LogiCORE IP JTAG AXI Master v1.0 Product Guide (AXI) (PG174) contains more information the JTAG to AXI Master IP core.





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-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 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 B	oards				
Reset All Fil Vendor: Al		All			
<u>S</u> earch: Q		*			
Display Na Add Daugl	me hter Card Connections		Preview	Vendor xilinx.com	File 1.4
	C705 Evaluation Platform hter Card Connections			xilinx.com	1.5
	raScale KCU105 Evaluation nter Card Connections	Platform		* xilinx.com	1.4

Figure 160: Choosing the Kintex-7 KC705 Evaluation Platform Board





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

A	New Project ×
KADO	New Project × New Project Summary A new RTL project named 'project_2' will be created. The default part and product family for the new project: Default Board: Kintex-7 KC705 Evaluation Platform Default Part: xc7k325tffg900-2 Product: Kintex-7 Family. Kintex-7 Family. Kintex-7 Parkage: ffg900 Speed Grade: -2
E XILINX.	To create the project, click Finish < Back Next > Einish Cancel

Figure 161: New Project Summary





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**.

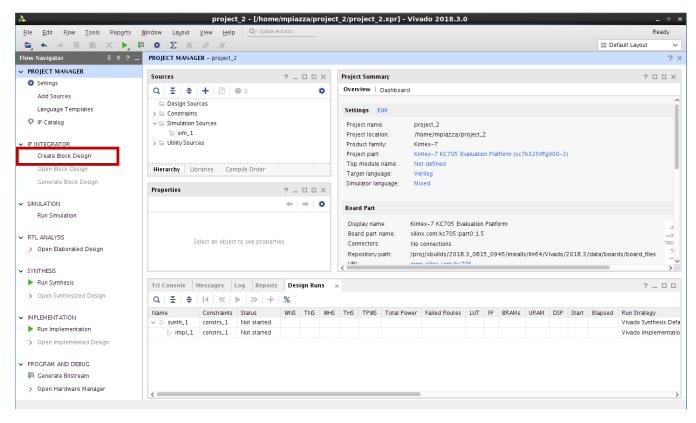


Figure 162: IP Catalog from Flow Navigator





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.

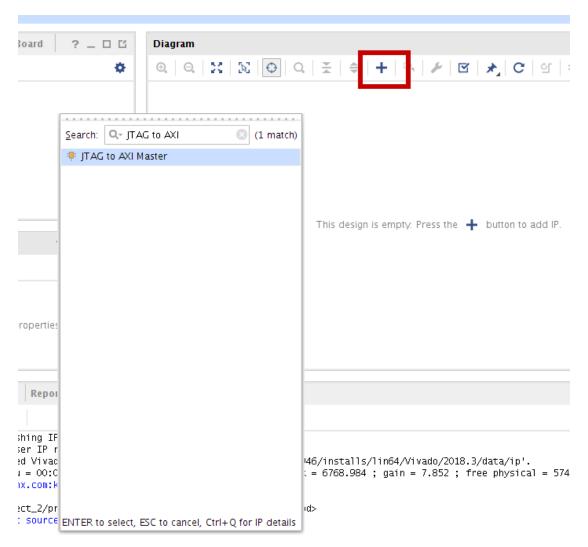


Figure 163: Adding JTAG to AXI Master IP Core to the IP Integrator Design





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)				4
🚯 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	~	
aclk ■ aresetn M_A×I + [:]	Write Transaction Queue Length	1	💿 [1 - 16]	
	Read Transaction Queue Length	1	<pre>[1 - 16]</pre>	
			ок (Cancel

Figure 164: JTAG to AXI Master Customization Dialog





- 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-customize IP	×
AXI BRAM Controller (4.0)		4
🚯 Documentation 🛛 🗁 IP Location		
Show disabled ports	Component Name axi_bram_ctrl_0	
	AXI Protocol AXI4 ~ Data Width 32 ~	
	Memory Depth (Auto) 8192 V ID Width (Auto) 0 V	
	Auto Support AXI Narrow Bursts Yes ~	
+ S_AXI s_axi_acik BRAM_PORTA +	BRAM Options BRAM_INSTANCE (Auto) External V Number of BRAM interfaces 1 V	_
	ECC Options	_
	Enable ECC No 🗸	
	ECC TYPE Hamming ~	
	Enable Fault Injection No 🗸	
	ECC Reset Value 0 V	
	OK Cano	el

Figure 165: Configuring the AXI BRAM Controller





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 checkbox. Click **OK**.

A	Re-customize IP	×
Block Memory Generator (8.4)		4
🛭 Documentation 🛛 📄 IP Location		
IP Symbol Power Estimation Show disabled ports Show disabled ports	Component Name blk_mem_gen_0 Basic Port A Options Other Options Standard Image: Stages within Mux Image: Stages within Mux Pipeline Stages within Mux Image: Mux Size: 2x1 Memory Initialization Image: Strowse Image: Docoe_file_loaded Image: Strowse <	
	Enable Safety Circuit OK C	ancel

Figure 166: Configuring the Block Memory Generator





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

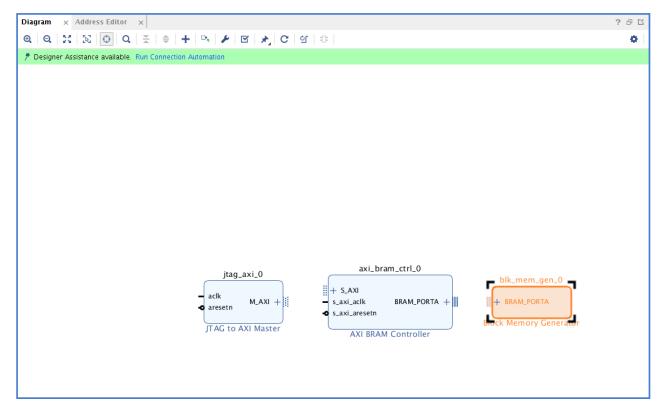


Figure 167: The Configured Design

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**.





Diagram x Address Editor x	2 6 2
	o
ck_wiz rst_ck_wiz_100M jtag_axi_0 axi_smc axi	+ BK_mem_gen_0 + BRAM_PORTA BOCK Memory Generator

Figure 168: The Design After Running Connection Automation

- 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 sys diff clk.





Symbol Resource	Component Name clk_wiz		
Show disabled ports	Board Clocking Options Outp	out Clocks MMCM Settings Summary	
	Associate IP interface with board inte	rface	
	IP Interface	Board Interface	
	CLK_IN1	sys diff clock	•
	CLK_IN2	Custom	•
	EXT_RESET_IN	reset	•
	Clear Board Parameters		
	cical board rarameters		
+ CLK_IN1_D clk_out1 -			
- reset locked -			

Figure 169: Configuring the Clocking Wizard to Connect to the Clock and Reset on the KC705

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.

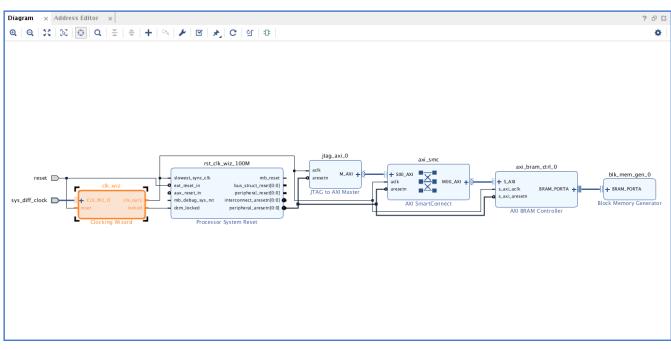


Figure 170: The Completed Design with AXI Infrastructure

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.





System ILA (1.1)	Re-customize IP		×
🚯 Documentation 🛛 📄 IP Location			
IP Symbol Resources	Component Name system_ila_0		
Resource Estimates	To configure more than 64 probe ports use V	ïvado Tcl Console	
100.0	General Options Interface Options Monitor Type		
90.0	Monitor Type INTERFACE 🗸		
80.0	Number of Interface Slots 1	8	
8 60.0 5 50.0	Sample Data Depth 1024 V		
월 50.0 40.0	Same Number of Comparators for All Pr	obe Ports	
30.0	Number of Comparators	3 🗸	
20.0	 Trigger Out Port Trigger In Port 		
	Input Pipe Stages	0 🗸	
0.0 1.0	Trigger And Storage Settings		
Resource Usage	 Capture Control Advanced Trigger 		
BRAM Slice: 5			
<			
			OK Cancel

Figure 171: System ILA Configuration Dialog

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**.





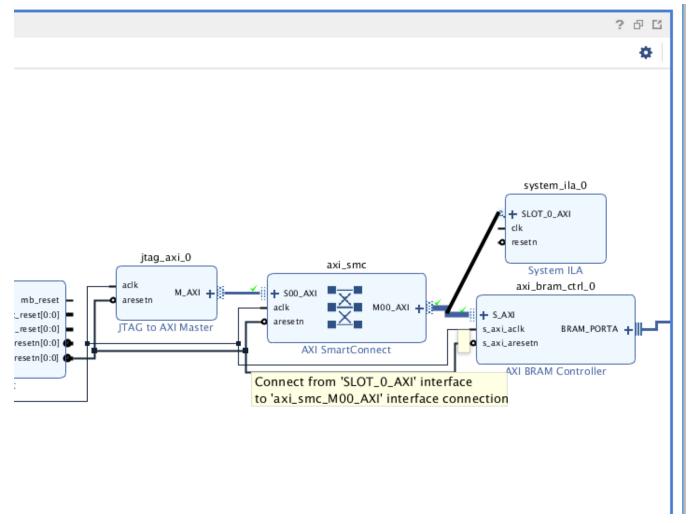


Figure 172: Probing the AXI Bus with System ILA

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.

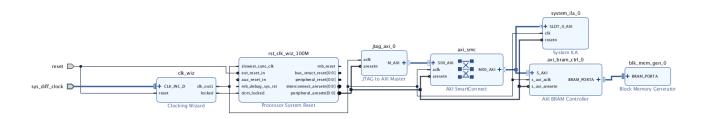
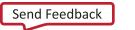


Figure 173: The Completed Design





- 22. In the upper left side of the Vivado IDE, click **File -> Save Block Design**. Select **File -> 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.

×

Figure 174: Select Open Hardware Manager

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 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.





Lab 9: Using Vivado ILA Core to Debug JTAG-AXI Transactions

HARDWARE MANAGER - unconnected						
1 No hardware target is open. Open target						
Hardware	ø	Auto Connect				
		Recent Targets Available Targets on Server	•			
		Open New Target	6			
No conter	nt					

Figure 175: Connect to a Hardware Target

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

🥕 Open New	Hardware Target			×
Select local or i	erver Settings remote hardware server, then configure the host name and port otherwise, use Remote server.	t settings. Use Local server if the ta	rget is attached to the	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>N</u> ext >	Einish	icel

Figure 176: Hardware Server Name

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 New H	ardware Targ	et				×
	e target from the l				riate JTAG clock (TCI select a different targ	
Hardware <u>T</u> arg	ets					
Туре	Name		JTAG CIO	ock Frequency		
xilinx_tcf	Xilinx/Port_#00	03.Hub_#0004	6000000	~		
		Add Ville	nx Virtual Ca			
		Add Alli	ix virtual Ca	able (XVC)		
Hardware <u>D</u> evi	ces (for unknow	n devices, spe	cify the Inst	ruction Registe	er (IR) length)	
Name	ID Code	IR Length				
@ xc7k325t_0	0 33651093	6				
Hardware serve	er: localhost:312	1				
?		< <u>E</u>	ack	<u>N</u> ext >	Einish	Cancel

Figure 177: Select Hardware Target

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.

🅕 Open New Hardware	e Target	×
	Open Hardware Target Summary () Hardware Server Settings:	
	Server: localhost3121	
	 Target Settings: Target: xilinx_tcf/Xilinx/Port_#0003.Hub_#0004 Frequency: 6000000 	
E XILINX ALL PROGRAMMABLE.	To connect to the hardware described above, click Finish	
?	< <u>B</u> ack <u>N</u> ext > <u>Finish</u> Car	icel

Figure 178: Open Hardware Summary

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.





٥

Figure 179: Hardware Target and XC7K325T Device

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.

Image: State of the second	Hardware				? _ 🗆	c ×	
 Iocalhost (1) Connected iiinx_tcf/Xilinx/Port_#0003.Hu Open iiix_crk325t_0 (1) iiiix_XADC (System M Hardware Device Properties Ctrl+E Verify Device Refresh Device Add Configuration Memory Device Boot from Configuration Memory Device Program BBR Key Clear BBR Key Clear BBR Key 	Q 素 ♦ ∅ ▶ >	>				٥	
× Image: String to the string to	Name			Status			
* (1) * xc7k325t_0 (1) * XADC (System M) Hardware Device Properties Program Device Verify Device Refresh Device Add Configuration Memory Device Boot from Configuration Memory Device Program BBR Key Clear BBR Key	 Iocalhost (1) 			Connected			
Image: ADC (System M Hardware Device Properties Ctrl+E Image: Program Device Program Device Image: Program Device Image: Program Device Refresh Device Image: Program Device Image: Program Device Refresh Device Image: Program Device Image: Program Device Properties Program BBR Key Image: Program BBR Key Image: Program BBR Key Clear BBR Key Image: Program BBR Key	✓ ✓ ✓ ✓ ✓ ✓ Ø xilinx_tcf/Xilinx/Port_#00	003.H	lu	Open			
Hardware Device Properties Program Device Verify Device Refresh Device Add Configuration Memory Device Boot from Configuration Memory Device Boot from Configuration Memory Device Program BBR Key			Hard	lware Device P	roperties		Ctrl+E
Image: Constraint of the second se				ram Device			1 contractions of the second s
Boot from Configuration Memory Device Program BBR Key Clear BBR Key	C						
Hardware Device Properties				-	-		1
	Hardware Davies Dreportion		-	-			
Program eFUSE Registers			Prog	ram eFUSE Re	egisters		
Export to Spreadsheet			Expo	ort to Spreadsh	eet		

Figure 180: Program Active Target Hardware





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		×
	ramming file and download it to your hardware device. You can Ig probes file that corresponds to the debug cores contained in the g file.	4
Bitstre <u>a</u> m file: Debug probes file:	1/jtag_axi_0_ex/jtag_axi_0_ex.runs/impl_1/example_jtag_axi_0.bit	
Proble and of all		cel

Figure 181: Select Bitstream File to Download

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

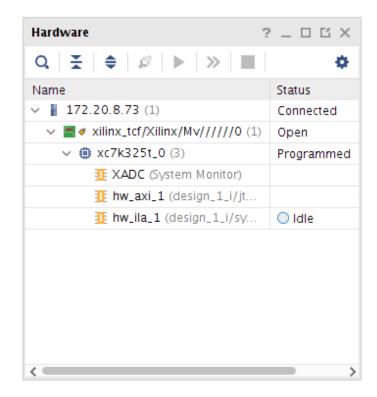


Figure 182: ILA Core Instances in the Hardware Window





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:

reset_hw_axi [get_hw_axis hw_axi_1]

Tcl Console x Messages Serial I/O Links Serial I/O Scans	? _ 🗆 🖸
<pre>program_hw_devices: Time (s): cpu = 00:00:25 ; elapsed = 00:00:25 . Memory (MB): peak = 1514.480 ; gain = 0.000 refresh_hw_devices [Lindex (get_hw_devices xc?k255_0] 0] 'INFO: [Lindox] 27-2302] Device xc?k255 (JTAB device index = 0) is programmed with a design that has 1 ILA core(s).</pre>	^
INFO: [Labtools 27-2302] Device xc7k325t (JTAG device index = 0) is programmed with a design that has 1 JTAG AXI core(s). Implay_hw_lla_data { get_hw_ila_data_1 - of_objects [get_hw_ilas - of_objects [get_hw_idevices xc7k25t_0] - filter (CELL_MAME=-"u_ila_0")]] INFO: [Labtools 27-3304] LIA Waveform data saved to file cr/Virado Debug/2017.1/jtag axi 0 ex.hw/backuphw ila data 1.ila. Use Tol command 'import hw ila data' or Virado Fbig/1.	a_\Tenort_\Tenort T
Energy instruction in the second data seven to file crystead beoug/correspondence of seven and the second data seven to file crystead beoug/correspondence of seven and the second data seven to file crystead beoug/correspondence of seven and the second data seven to file crystead beoug/correspondence of seven and the second data seven to file crystead beoug/correspondence of seven and the second data seven to file crystead beoug/correspondence of seven and the second data seven to file crystead beoug/correspondence of seven and the second data seven to file crystead beoug/correspondence of seven and the second data seven to file crystead beoug/correspondence of seven and the second data seven and the seven and the second data seven and the seven and the second data seven and the second data seven and the second data seven and the se	e->import->import i
	>

Figure 183: Reset JTAG to AXI 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:

- o write_txn is the name of the transaction.
- o [get_hw_axis hw_axi_1] returns the hw_axi_1 object.
- o -address C0000000 is the start address.
- o -len 128 sets the AXI burst length to 128 words.
- o -data {4444444_3333333_22222222_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:

- o read_txn is the name of the transaction.
- o [get_hw_axis hw_axi_1] returns the hw_axi_1 object.
- o -address C0000000 is the start address.
- o -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 : 444444433333333222222221111111...

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...





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.

Settings - hw_ila_1		? _ 🗆 ×	Trigger Setup - hw_	ila_1 Capture Setup	hw_ila_1 Status - I	nw_ila_1 × ?	_ □
Trigger Mode Settings		_	৫ ► ≫	24			
Trigger mode:	ADVANCED_ONLY ~		Core status				
ringger state machine.	BASIC_ONLY		Idle	Pre-Trigger	Waiting for Trigger	Post-Trigger	
Capture Mode Settings			Trigger State Mac	hine			
Capture mode:	ALWAYS 🗸		Flag O	Flag 1	Flag 2	Flag 3	
Number of windows:	1 [1 - 1024]		Trigger state: 0 Capture status				
Window data depth: Trigger position in windo	1024 ∨ [1 - 1024] w: 512 [0 - 1023]		Window 1 of 1	Window sample		ple 0 of 1024 Idle	
General Settings Refresh rate: 500	ms	_		,			
			<				>

Figure 184: Setting Trigger Mode to ADVANCED and Trigger Position to 512 in the ILA Dashboard



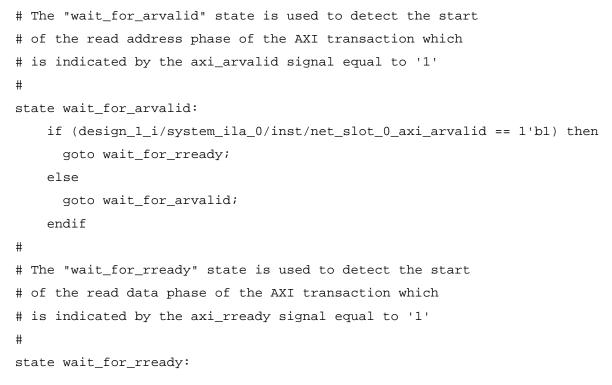


4. In the **New Trigger State Machine File** dialog box set the name of the state machine script to **txns.tsm**.

🔥 New Trigg	er State Machine File	X
Save <u>I</u> n: 🕠	jtag_axi_0_ex	 Image: Image: Ima
	_ex.hw _ex.ioplanning _ex.ip_user_files _ex.runs _ex.sim	Recent Directories C://ivado_Debug/2017.1/jtag_axi_0_ex/jtag_axi_0_ex.runs/impl_1 ♥ File Preview Select a file to preview.
File <u>n</u> ame:	txns	
Files of <u>t</u> ype:	Trigger State Machine Files (.tsm)	Save Cancel

Figure 185: Creating a New Trigger State Machine Script

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.



Programming and Debugging UG936 (v2019.1) May 22, 2019 www.xilinx.com



```
if (design_1_i/system_ila_0/inst/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/inst/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			1	? _ 🗆 [з×	exa	imple_jtag_axi_0.v
Q 素 ⊜ ∅ ►	»				٠		Waveform - hw_ila
Name			Status			s	Q + -
Y 📕 localhost (1)			Connected			otion	4 T -
✓ Ø xilinx_tcf/Xilinx/Port_	#0003	.Hu	Open			oard Options	ILA Status: Idle
 × @ xc7k325t_0 (3) 	_	1	Programmed			0al	Nome
🖉 XADC (System	N	Hard	ware Device Pro	perties		0	trl+E axi_araddr[31
🦉 hw_axi_1 (AXI)		Progr	am Device				axi_arburst[1:
🦉 hw_ila_1 (u_ila		Verify	Device				axi_arcache[3
		Run 1	Frigger				axi_arlen[7:0]
	»	Run 1	Frigger Immedia	ite			axi_arprot[2:0 axi_awaddr[3
			Trigger				axi_awaddi[3 axi_arqos[3:0
	17.		le Auto Re-trigge	or			axi_arsize[2:0
							axi_awburst[1
			ble Auto Re-trigg				axi_awcache[
		Creat	e User Defined	Probe			axi_awlen[7:0
	C	Refre	sh Device				
		Add C	Configuration Me	mory Devic	:e		
	1	Boot	from Configurati	on Memory	Device		
Hardware Device Properties		Progr	am BBR Key				igs - hw_ila_1
<pre>xc7k325t_0</pre>		Clear	BBR Key				jer Mode Sett
Name: xc7k325t_		Progr	am eFUSE Reg	isters			rigger mode:
Part: xc7k325t		Expor	t to Spreadshee	et			rigger state m
ID code: 33651093	3						
IR length: 6							Capture Mode Set

Figure 186: Run Trigger





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:

Settings - hw_ila_1 Status - hw_ila_1 ×
🔁 🕨 🗶 💶
Core status 💿 🔵 🌑 🔵 Waiting for Trigger
Trigger State Machine - Flags: 0 1 2 3
Trigger state: wait_for_arvalid (0)
Capture status - Window 1 of 1
Window sample 512 of 1024
50 <mark>%</mark>

Figure 187: Trigger Capture Status Window





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.

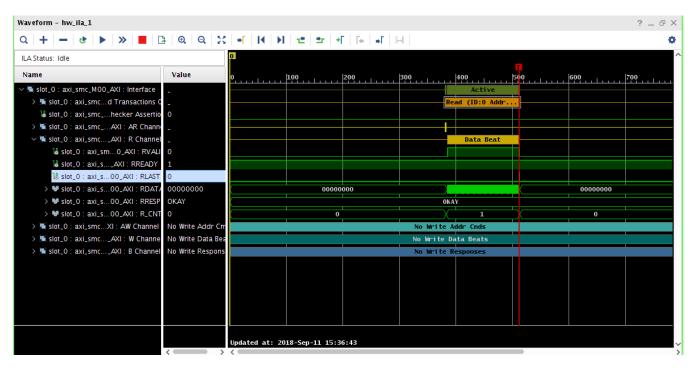


Figure 188: Waveform Window





Lab 10: Using Vivado Serial Analyzer to Debug GTR Serial Links

Introduction

IBERT UltraScale+[™] GTR (IBERT GTR) can be used to evaluate and monitor GTR transceivers in Zynq[®] UltraScale+ MPSoC devices. With this feature, a user can accomplish the following tasks:

- Perform eye scans with user data
- Change GTR settings
- View link status
- Check the "lock" status of all PLLs used by all GTR lanes

IBERT GTR does NOT provide the following capabilities:

- Perform eye scans with raw 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 of the device. This documentation will guide you through the setup of the GTR Transceivers by creating a First Stage Boot Loader (FSBL). It will then demonstrate how to load the FSBL into the Zynq UltraScale+ and use IBERT GTR.



IMPORTANT: This is a supported feature in Vivado® 2017.2 and beyond.

IBERT GTR Flow

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

- 1. Generating Zynq UltraScale+ MPSoC PS Hardware Definition File (HDF) from Vivado after configuring the GTR
- 2. Using SDK XSCT flow to generate First Stage Boot Loader file by using the Hardware Definition File (HDF)
- 3. Using First Stage Boot Loader file with Vivado Serial I/O Analyzer to bring up IBERT GTR.

Tools Required:

• Vivado





- SDK
- XSCT (Part of SDK)

Board/Part/Components required:

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

Required Files

- First Stage Boot Loader ELF File (Created using instructions below) which configures the 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 A53 processor as R5 (psu_cortexr5_0) is exclusively used by IBERT GTR.
- 2. Physical devices such as SATA drive, PCIE card, etc. are needed for validation.

Step 1: Generating Zynq UltraScale+ MPSoC PS Hardware Definition File (HDF)

- 1. Open Vivado.
- 2. Click on **Create Project** and click **Next**.





🍌 Vivado 2017.2			
<u>F</u> ile F <u>l</u> ow <u>T</u> ools <u>W</u> indow	Help Q+ Quick Access	_	
			EXILINX ALL PROGRAMMABLE.
TIEL EURONS	🍌 New Project	×	
Quick Star <u>Create Project</u> > Open Project > Open Example Project		Create a New Vivado Project This wizard will guide you through the creation of a new project. To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.	
Tasks Manage IP > Open Hardware Manag Xilinx Tcl Store >			_zcu102_gth_ES1_2016_4_ex p 11p5dB_ex s.gth_linerate_10Gbps_refclk _20dB_ex
Learning (Documentation and Tur Quick Take Videos >			ect_1
Tcl Console	(?)	< Back Next> Einish Cancel	? _ 🗆 🖒 ×
Q Z + II II II start_gui c Type a Tol command here New Project Wizard will guide you throu		gn sources and a target device for a new project.	<u>ک</u>

Figure 189: New Project Window

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





i New Project			×
Default Part Choose a default Xilinx part or board for your project. This ca	in be changed later.		A
Parts Boards			
Reset All Filters Vendor: All → Name: All Search: Q-	~]		
Display Name	Preview	Vendor	Fil
Zynq UltraScale+ ZCU102 Evaluation Board Add Daughter Card Connections	Q	xilinx.com	3.2
Zynq UltraScale+ ZCU104 Evaluation Board Add Daughter Card Connections		xilinx.com	1.0
			, ,
 ? 	< <u>B</u> ack <u>N</u> ext >	<u>F</u> inish	Cancel

Figure 190: Board Selection Window

7. The project summary displays. To create the project, click Finish.





8. Select **Create Block Design** in the Flow Navigator. 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.

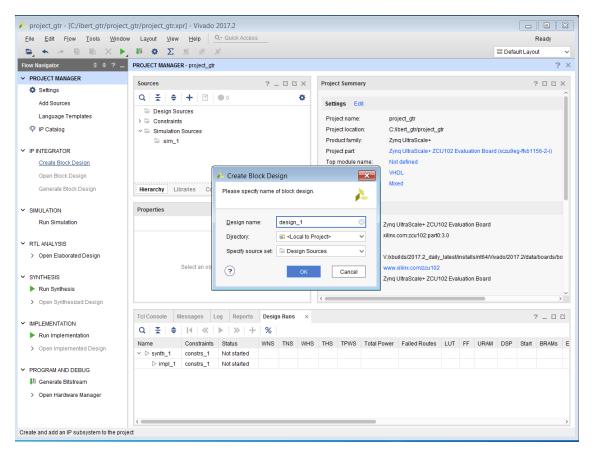


Figure 191: Create Block Design





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

BLOCK DESIGN - design_1		? ×
Sources Design × Signals Board ? _ []	Diagram	? 🗆 🖾 X
▲ design_1	Search: Q- Zy (1 match) Zyng UltraScale+ MPSoC	
Properties ? _ □ □ × ← → ◆ Select an object to see properties		
	ENTER to select, ESC to cancel, Ctrl+Q for IP details	

Figure 192: Add IP Window





10. Select **Run Block Automation** in the design diagram window. Click **OK** to continue creating the ZCU102 design.

Diagram × Address Editor ×	? 🗆 🖸
$\textcircled{\textbf{Q}} \mid \textcircled{\textbf{Q}} \mid \Huge{\fbox{\textbf{X}}} \mid \Huge{\textcircled{\textbf{Y}}} \mid \textcircled{\textbf{Q}} \mid \Huge{\textcircled{\textbf{Q}}} \mid \Huge{\textcircled{\textbf{X}}} \mid \Huge{\textcircled{\textbf{Q}}} \mid \emph{\textcircled{\textbf{Q}}} \mid \Huge{\textcircled{\textbf{X}}} \mid \Huge{\textcircled{\textbf{Q}}} \mid \emph{\textcircled{\textbf{Q}}} \mid \Huge{\textcircled{\textbf{Q}}} \mid \emph{\textcircled{\textbf{Q}}} \mid \emph{\textcircled} \mid \emph{\textcircled{\textbf{Q}}} \mid \emph{\textcircled{\textbf{Q}}} \mid \emph{\textcircled}} \mid \emph{\textcircled{\textbf{Q}}} \mid \emph{\textcircled} \mid \emph{\textcircled}} \mid \emph{\textcircled}} \mid \emph{\textcircled}$	•
★ Designer Assistance available. Run Block Automation	
zynq_ultra_ps_e_0	
- maxihpm0_lpd_aclk ZYNQ® M_AXI_HPM0_LPD pl_resetu UltraSCALE+	n0 0-
Zynq UltraScale+ MPSoC	

Figure 193: Run Block Automation





- 11. When the design diagram appears, follow the below steps to validate design:
 - a. Connect maxihpm0_fpd_aclk and maxihpm1_fpd_aclk together to pl_clk0, as shown in the figure below.
 - 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 on the Zynq UltraScale+ MPSoC block and select **Validate Design** to validate the design. It will say validation successful. Click **OK**.

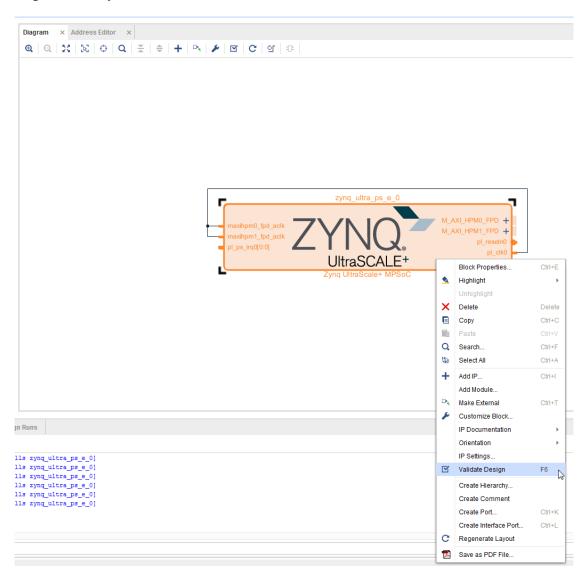


Figure 194: Validating the Design

12. Customize the design by double-clicking on the Zynq UltraScale+ MPSoC block and configuring the parameters. There are 4 valid GT configurations for ZCU102 board as shown below:





SEL (S3,2,1,0)	ICM Settings (Lane 0,1,2,3)	PCIe Connector	DP Connector	USB Connector	SATA Connector
0000	PCIe.o, 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 GTR Connector Functionality

13. Select the settings based on your requirements by double-clicking on 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 screenshots below:
 - a. PCIe Display Port USB SATA (Default Vivado preset)

ike-customize IP							×
Zynq UltraScale+ N	//PSo	C (3.0)					4
Documentation	Procot	s 📮 IR Location					
	Flesei						
Page Navigator		O Configuration					
Switch To Advar	+	MIO Voltage Standar	rd				
PS UltraScale+ Blo	Q,		Bank1 (MIO 26: LVCMOS18	51] Bank2 [N		Bank3 [Dedicated]	
I/O Configuration	*	Search: Q-					
Clock Configuration	-	Peripheral	I/O			Signal	I/O Type
DDR Configuration	— ,	> Low Speed					
		✓ High Speed					
PS-PL Configuratio		> GEM					
		✓ USB					
		~ ✓ USB 0					
		> USB 2.0		5263	~		
		> 🖉 USB 3.0	GT	Lane2	~		
		USB 1					
		✓ ✓ PCle					
		> Rootport Mode Re			~		
		Reset Polarity		ve Low	~		
		Lane Selection PCIe Lane0	x1		~		
			GIL	ane0			
		 ✓ ✓ Display Port > DPAUX 	MIC	2730	~		
		> Lane Selection			* *		
		✓ ✓ SATA	SIII	gle Lower	•		
		SATA Lane0					
		SATA Lane1	CT	Lane3	~		
		> Reference Clocks	GI	Lanes	•		
		<		-	_		>
						ОК	Cancel

Figure 195: I/O Customization Window: PCIe – Display Port – USB – SATA (Default)





b. PCIe - PCIe - USB - SATA

Documentation 🔅	Prese	ts 📄 IP Location				
Page Navigator		O Configuration				
Switch To Advar	+	 MIO Voltage Standard 				
	Q	Bank0 [MIO 0:25] Bank	1 [MIO 26:51] Bank	2 [MIO 52:77]	Bank3 [Dedicated]	
PS UltraScale+ Blo	-	LVCMOS18 🗸 LVCI	IOS18 🗸 LVCI	/OS18 🗸	LVCMOS33 V	
I/O Configuration	*	Search: Q-				
Clock Configuration	<u> </u>	Peripheral	1/0		Signal	I/O Type
	•	> Low Speed			orginar	#0 1)po
DDR Configuration		 High Speed 				
PS-PL Configuratio		> GEM				
		✓ USB				
		~ ✓ USB 0				
		> USB 2.0	MIO 5263	~		
		> 🗸 USB 3.0	GT Lane2	~		
		USB 1				
		 ✓ ✓ PCIe 				
		> Rootport Mode Reset	MIO 31	~		
		Reset Polarity	Active Low	~		
		Lane Selection	x2	~		
		PCIe Lane0	GT Lane0			
		PCIe Lane1	GT Lane1			
		> Display Port				
		∽ 🖌 SATA				
		SATA Lane0				
		🖌 SATA Lane1	GT Lane3	~		
		> Reference Clocks				
		<				

Figure 196: I/O Configuration Window: PCIe – PCIe – USB – SATA





c. Display Port - Display Port - USB - SATA

Documentation 🏟	Prese	ts 📄 IP Location						
Page Navigator	1	/O Configuration						
Switch To Advar	+	✓ MIO Voltage Standar	rd					
	Q	Bank0 [MIO 0:25]	Bank1 (MIO	26:51]	Bank2 [MIO 52	2:77]	Bank3 [Dedicated]	
PS UltraScale+ Blo	-		LVCMOS1		LVCMOS18	~	LVCMOS33 V	
/O Configuration	Ŧ	/L						
	¢	Search: Q-						
Clock Configuration		Peripheral		I/O			Signal	I/O Type
DDR Configuration	-	> Low Speed						
		✓ High Speed						
PS-PL Configuratio		> GEM						
		✓ USB						
		✓ ✓ USB 0						
		> USB 2.0		MIO 52	63	~		
		> 🗸 USB 3.0		GT Lan	e2	~		
		USB 1						
		> PCle						
		✓ ✓ Display Port						
		> DPAUX		MIO 27	30	~		
		✓ Lane Selection		Dual Lo	ower	~		
		DP Lane0	G	T Lane	1			
		DP Lane1	C	T Lane	0			
		✓ 🗹 SATA						
		SATA Lane0						
		SATA Lane1		GT Lan	e3	~		
		> Reference Clocks						

Figure 197: I/O Configuration Window: Display Port - Display Port - USB – SATA





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

Re-customize IP	C (3.0)								
Documentation 🔅 Preset									
Page Navigator —	I/O Configuration	n							
Switch To Advanced M	MIO Volta	age Standard							
	Q Bank0 [MIO 0:25] B	ank1 [MIO 26:5	1] Bank2 [MIO 5	2:77]	Bank3 [Dedicated]			
PS UltraScale+ Block Des	LVCMO	S18 🗸 L	VCMOS18	V LVCMOS18	~	LVCMOS33 V			
I/O Configuration	ž								
Clock Configuration	Peripheral		I/O			Signal	I/O Type	Drive Strength(mA)	Spe
DDR Configuration	Low Speed	I							
	✓ High Speed	đ							
PS-PL Configuration	> GEM								
	> USB								
	✓ ✓ PCle								
		ort Mode Rese		31 e Low	~				
		Polarity Selection	X4	e Low	~				
	PCle L		GT La	ne0	•				
	PCle L		GTLa						
	PCle L		GT La						
	PCIe L	ane3	GT La						
	> Displa	ay Port							
	> SATA								
	> Reference	Clocks							
	<								
								OK Cano	cel
							L	Can	

Figure 198: PCIe - PCIe - PCIe - PCIe





15. You can save customized presets using preset by selecting **Presets** > **Save Configuration** at the top of the window. This is useful if you want to save various configurations for future use.

🔥 Re-customize IP										×
Zynq UltraScale+ M	IPSo	C (3.0)								4
🚯 Documentation 🔅 P	Preset	s 📄 IP L	ocation							
Page Navigator			nfiguration							
Switch To Advar	+	~ MIO	Voltage Standa	rd						
PS UltraScale+ Blo	Q		nk0 [MIO 0:25] CMOS18 🗸 🗸	Bank1 [MI LVCMOS		Bank2 [MIO 5 LVCMOS18	2:77] ~	Bank3 [Dedicated]		
I/O Configuration	¥ ♦	Search:	Q							
Clock Configuration	-	Periphera	al		I/O			Signal	I/O	Туре
DDR Configuration	-	> Low S	peed							
		✓ High S								
PS-PL Configuratio		> GEI								
		> USE								
		~ √ P								
			ootport Mode Re	eset	MIO 31		~			
			eset Polarity		Active L	ow	~			
			ane Selection Cle Lane0		x4 GT Lane	•	~			
			Cle Laneu Cle Lane1		GT Lane	-				
			Cle Lane 1		GT Lane					
			Cle Lane3		GT Lane					
)isplay Port		OT Lane	•				
		> 🗆 s								
			ence Clocks							
		<								>
								ОК	Can	cel

Figure 199: Selecting Save Configuration





16. Saved preset can be applied by selecting **Preset** > **Apply Configuration**. A file selection dialogue will appear as shown below. Choose the preset **PCIE-PCIE-USB-SATA** and click **OK**.

4	
Look <u>i</u> n: 🥡 temp	✓ Ø S = k ≥ × S = :::
🚚 tmplpa2ty	Recent Directories
🚛 tmppqvrud	C:/temp 🗸
🐙 tmppupipm	File Preview
🗼 tmpte9mi1	proc getPresetInfo {} {
vio_0	return [dict create name {pcie_all} description {pcie
√ xinstall	3
🖲 pcie_alltcl	<pre>proc validate_preset {IPINST} { return true }</pre>
	proc apply_preset {IPINST} {
	return [dict create \
	CONFIG.PSU_PSS_REF_CLK_FREQMHZ {33.333} \ CONFIG.PSU_CAN1_PERIPHERAL_ENABLE {1} \
	CONFIG.PSU_CAN1_PERIPHERAL_IO {MIO 24 25} \
	CONFIG.PSU_DPAUX_PERIPHERAL_ENABLE {0} \
	CONFIG.PSU_DPAUX_PERIPHERAL_IO { <select>} \ CONFIG.PSU ACT DDR FREQ MHZ {1066.656006} \</select>
	CONFIG.PSU_FPGA_PLO_ENABLE {1} \
	CONFIG.PSU ENET3 PERIPHERAL ENABLE {1}
	CONFIG.PSU_ENET3_PERIPHERAL_IO {MIO 64 75}
>	
File come:	
File <u>n</u> ame: pcie_alltcl	
Files of type: Preset Tcl Files (*.tcl)	*
	OK Cancel

Figure 200: Applying a Preset Configuration

- 17. Click **OK** when finished customizing GT Lane configuration.
- 18. Do not click **Run Block Automation** again, even though the banner will reappear. It will reset the customized values if used.
- 19. Click on the **Sources** tab on the top left of the Block Design window. Under the Block Designs group, click on **IP Sources**. Right click on **design_1** and then click on **Create HDL Wrapper**.





Lab 10: Using Vivado Serial Analyzer to Debug GTR Serial Links

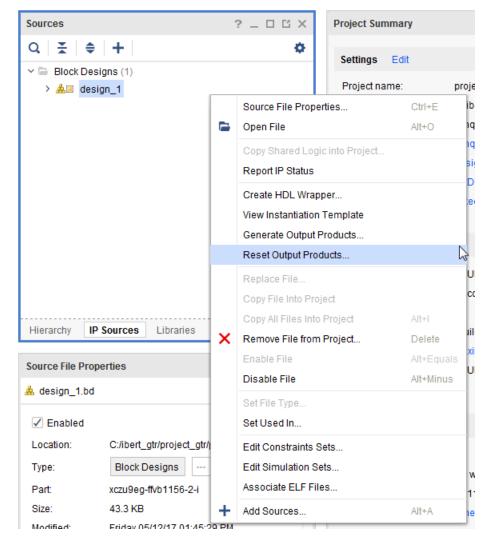


Figure 201: Create HDL Wrapper

- 20. Leave the option **Let Vivado manage wrapper and auto-update** selected. Click on **OK** in the dialogue to create HDL Wrapper.
- 21. Right click on design_1_i in IP Sources tab and click on Generate Output Products.
- 22. Click on Generate to generate with the default options in the panel.
- 23. Click **OK** after the generation is complete.





24. Select File > Export > Export Hardware.

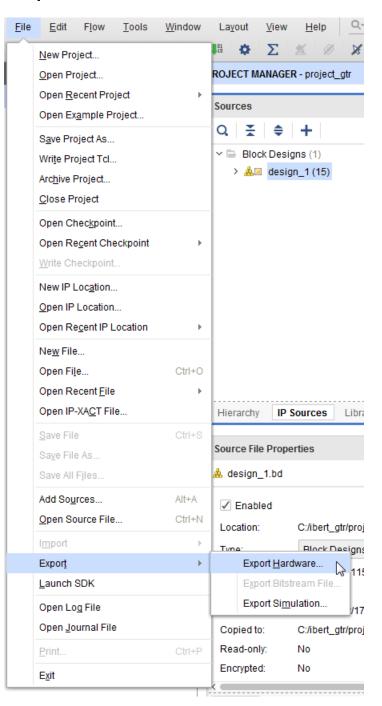


Figure 202: Export Hardware





25. You can specify an export directory or use Local to Project. Click **OK** to export the hardware.

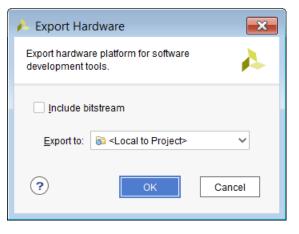


Figure 203: Export Hardware Window

26. You can see generated HDF path in the Tcl Console after the write_hwdef function call.

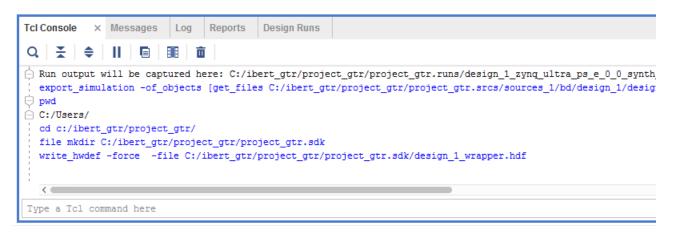


Figure 204: Running write_hwdef

Step 2: Using XSCT flow to generate FSBL by using HDF

Xilinx Software Command-line Tool (XSCT) is an interactive and scriptable command-line interface to Xilinx Software Development Kit (Xilinx SDK). The XSCT flow should be used for a more automated flow that is Tcl based and only requires running a Tcl script. There is an SDK flow that should be used for a more interactive setup of the FSBL which uses a GUI. You can use the SDK flow if you need to make changes/customizations to the FSBL. SDK flow could also be used for custom board. See "Appendix A" for the SDK or XSCT Manual flow.





Generating using XSCT Automated Flow:

To create an FSBL for the A53 #0 (64 bit) automatically (and modify the xfsbl_main.c/h files if USB is present) using the provided script, use the follow steps:

- 1. Copy the src/lab10/xsct_create_fsbl.tcl script to the directory where the HDF file is located (e.g. project_gtr/project_gtr.sdk). You can modify the Tcl script if you changed the default name of the HDF file in Vivado. 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. Cd into the directory where the HDF file is located.
- 4. Call xsct from the SDK install area:
 - % xsct xsct_create_fsbl.tcl
- 5. It prints out the location of the generated .elf file when the script completes.

Step 3: ZCU102 Board Settings

USB jumper setting requirements for HOST mode on ZCU102:

- 1. Make sure below jumpers are correctly set for USB to be in HOST mode (refer to UG1182).
 - e. J7 ON
 - f. J113 1-2
 - g. J110 2-3
- 2. Refer to the below image for jumper settings on a ZCU102 Rev 1.0 board:







Figure 205: Jumper Settings for the ZCU102 Board

Using FSBL with Serial I/O Analyzer to bring up IBERT GTR

- 1. Connect all the physical devices such as SATA Drive, PCIE 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 example below 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	×
	rer Settings ote hardware server, then configure the host name and port settings. Use Local server if the target is attached to the erwise, use Remote server.	4
<u>C</u> onnect to: F	Remote server (target 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>B</u> ack <u>N</u> ext > <u>F</u> inish C:	ancel

Figure 206: Hardware Server Settings





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

	re Target target from the I	ist of availal	ble targets, then set the a select a different target.	opropriate JTAG clock (TCK) frequency. If	you do not see the	× •
Hardware <u>T</u> arg	ets						
Туре	Name		JTAG Clock Frequency				
xilinx_tcf	Digilent/210308		15000000 🗸				
_			specify the Instruction Re	rtual Cable (XVC) egister (IR) length)			
Name	ID Code	IR Length					
xczu9_0 arm_dap_	04738093 5BA00477	12 4					
Hardware serve				< Back	Next>	Finish	Cancel

Figure 207: Selecting the Device

5. Right-click on the **ARM_DAP** device in the hardware tree and select **Configure IBERT GTR**.

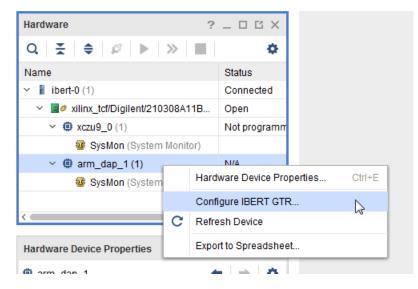


Figure 208: Selecting Configure IBERT GTR

www.xilinx.com





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 (e.g. ZU9EG es1). If that occurs, power cycle the board and keep the **Reset Zynq** option unchecked.

🔥 Configure IBERT (STR	×
A Zynq FSBL is required	d to configure IBERT GTR. The configuration file is optional.	•
Zyng FSBL:)roject_gtr/project_gtr.sdk/fsbl_design_1/Debug/fsbl_design_1.elf	1
Zyng robe.		1
Configuration File:]
Reset Zynq		
	OK Cancel	

Figure 209: Selecting the FSBL File





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 example below:

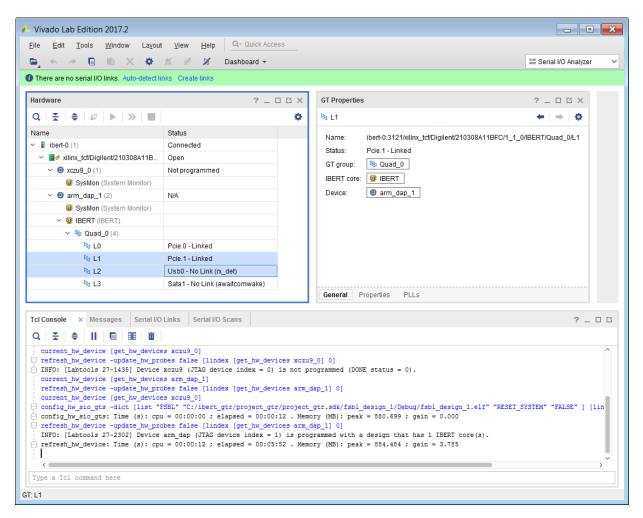


Figure 210: IBERT Configuration





8. The **Auto-detect links** option does not work for GTR. You can manually create links by using **Create Links** as shown below:

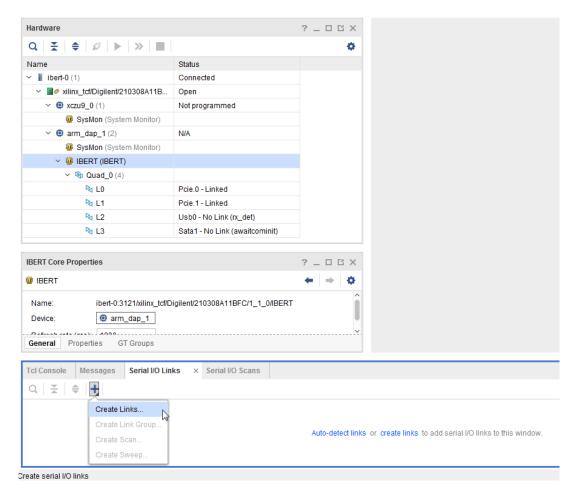


Figure 211: Selecting Create Links





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

Click **OK** when done.

🔥 Create Links		N			×
To create a new link	select a TX GT and/or an RX GT	, then click the Add b	utton on the New	/ Links toolbar.	4
TX GTs Search: Q-		RX GTs Search			
New Links					
+ -					
Description \sim ¹	ТХ	RX			
% Link 3	L3/TX (arm_dap_1/Quad_0)	L3/RX (arm_dap_	/Quad_0)		
% Link 2	L2/TX (arm_dap_1/Quad_0)	L2/RX (arm_dap_	/Quad_0)		
% Link 1	L1/TX (arm_dap_1/Quad_0)	L1/RX (arm_dap_	/Quad_0)		
% Link 0	L0/TX (arm_dap_1/Quad_0)	L0/RX (arm_dap_	/Quad_0)		
✓ Create link grou	n				
Link group descripti					U
✓ Open Serial I/O /	Analyzer layout				
?				ОК	Cancel

Figure 212: Creating Links





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

		Quick Access					
	🌣 🖄 🖉 🔀 Dashbo	oard 🔻				III Se	rial I/O Analyzer
Hardware		? _ 🗆 🖻 ×	GT Properties			?.	_ 🗆 🗆 ×
Q ≚ ≑ ∅ ▶ ≫		۰.	№ L1			+	$ \Rightarrow \diamondsuit$
Name	Status		Name:	ibert-0:3121/xilinx	tcf/Digilent/210308	A11BFC/1_1_0/IBERT/	Quad 0/L1
 ibert-0 (1) 	Connected			Pcie.1 - Linked	_ton bighten biz rooot		
✓ ✓ ✓ ✓ xilinx_tcf/Digilent/210308	A11B Open		GT group:	Quad_0			
xczu9_0 (1)	Not programmed						
🦉 SysMon (System Mo	itor)		IBERT core:	ibert			
image: mage: ma	N/A		Device:	@ arm_dap_1			
🦉 SysMon (System Mo	itor)						
IBERT (IBERT)							
✓ № Quad_0 (4)							
No. 10	Pcie.0 - Linked						
	FCIe.0 - Linked						
R₀ L1	Pcie.1 - Linked						
№ L1	Pcie.1 - Linked	init)	General Pr	operties PLLs			
Pa L1 Pa L2 Pa L3	Pcie.1 - Linked Usb0 - No Link (rx_det)	init)	General Pr	operties PLLs			? _ 🗆 [
Pa L1 Pa L2 Pa L3 Fcl Console Messages Seri	Pcie. 1 - Linked Usb0 - No Link (rx_det) Sata1 - No Link (awaitcom				TV DI L Statua		
Pa L1 Pa L2 Pa L3 Tcl Console Messages Seri Q TX ≑ +	Pcie. 1 - Linked Usb0 - No Link (rx_det) Sata1 - No Link (awaitcom	TX Post-Cursor	General Pro	operties PLLs RX PLL Status	TX PLL Status	TXUSERCLK Freq	
Pa L1 Pa L2 Pa L3 Tcl Console Messages Seri Q ≚ ♦ ↓ Name TX Digrouped Links (0)	Pcie. 1 - Linked Usb0 - No Link (rx_det) Sata1 - No Link (awaitcom	TX Post-Cursor			TX PLL Status	TXUSERCLK Freq	
Pa L1 Pa L2 Pa L3 Tcl Console Messages Seri Q I ★ I ♦ I ↓ Vame TX Dugrouped Links (0) < % Link Group 0 (4)	Pcie. 1 - Linked Usb0 - No Link (rx_det) Sata1 - No Link (awaitcom	TX Post-Cursor	TX Diff Swing	RX PLL Status	TX PLL Status	TXUSERCLK Freq	RXUSERCLK Freq
Pa L1 Pa L2 Pa L3 Tct Console Messages Seri Q I ★ I ♦ I ↓ Name TX Dungrouped Links (0) ✓ ⊕ Link Group 0 (4) Pa Link 0 L0/TX	Pcie. 1 - Linked Usb0 - No Link (rx_det) Sata1 - No Link (awaitcom) I//O Links × Serial I//O Scans RX Status	TX Post-Cursor Multiple V	TX Diff Swing User Value 🗸	RX PLL Status			RXUSERCLK Freq 100.00
Pat L1 Pat L2 Pat L3 Tcl Console Messages Serie Q ¥ ♦ H TX Dungrouped Links (0) Pat Link Group 0 (4) Pat Link 0 Lonk 0 Link 1	Pcie.1 - Linked Usb0 - No Link (rx_det) Sata1 - No Link (awaitcom) I//O Links × Serial I//O Scans RX Status O/RX Pcie.0 - Linked	TX Post-Cursor Multiple ~ User Value ~	TX Diff Swing User Value ~ User Value ~	RX PLL Status Locked Not Locked	Locked	100.000	RXUSERCLK Freq 100.00 100.00
№a L1 №a L2 №a L2 №a L3 Tcl Console Messages Seria Q ¥ ÷ + Name TX TX © Ungrouped Links (0) ~ % Link Group 0 (4) % Link 1 L1/TX % Link 1 L1/TX % Link 2 L2/2/X	Pcie.1 - Linked Usb0 - No Link (rx_det) Sata1 - No Link (awaitcom IVO Links X Status OVRX Pcie.0 - Linked _1/RX Pcie.1 - Linked	TX Post-Cursor Multiple User Value User Value User Value	TX. Diff Swing User Value & User Value & User Value & User Value & User Value &	RX PLL Status Locked Not Locked Locked	Locked Not Locked	100.000	? _ 0 0 RXUSERCLK Freq 100.00 100.00 26.00 150.00

Figure 213: Serial I/O Links View

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	nks ×		Link Properties	Ctrl+E
Q ₹ ≑ †				×	Delete Create Links	Delete
Name	ΤХ	RX	Status		Create Link Group	
Ungrouped Links (0)						
🗸 🚳 Link Group 0 (4)					Create Scan	2
% Link 0	L0/TX	L0/RX	Pcie.0 - L		Create Sweep	-0
% Link 1	L1/TX	L1/RX	Pcie.1 - L		Commit Properties	
% Link 2	L2/TX	L2/RX	Pcie.2 - L	c	Refresh Serial I/O Objects	
% Link 3	L3/TX	L3/RX	Pcie.3 - L	-		

Figure 214: Selecting Create Scan





12. Select the appropriate parameters for EyeScan and perform the EyeScan. For example, the figure below 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.

🔥 Create Scan	×			
Set the description and other properties to create and optionally run a scan on the selected link.				
Link: Link 1 (L1/T	Link 1 (L1/TX, L1/RX)			
Description: Scan 0	8			
Scan Properties				
<u>S</u> can type:	2D Full Eyescan 🗸			
Horizontal increment	4 ~			
Horizontal range:	-0.500 UI to 0.500 UI 🗸 🗸			
Vertical increment:	4 ~			
V <u>e</u> rtical range:	100% ~			
Dwell				
• <u>B</u> ER: 1e-5	~			
○ <u>T</u> ime:	0 🗘			
✓ <u>R</u> un scan				
?	OK Cancel			

Figure 215: Selecting EyeScan Parameters





13. Below is a sample EyeScan performed on Lane L1:

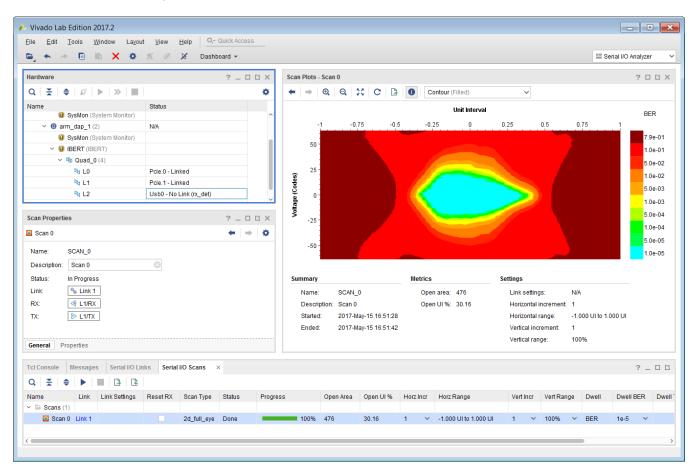


Figure 216: Sample EyeScan

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





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.
- 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), then 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 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 GTR. Any other processor core should not access this memory while IBERT 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. 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, then the EyeScan may not complete. Cancelling 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.

Using SDK Flow or XSCT Flow to Generate FSBL by Using HDF

The SDK flow should be used for a more interactive setup of the FSBL which uses a GUI as well as when creating FSBL for custom board. The XSCT flow should be used for a Tcl based flow.

Generating using SDK Flow:

 Launch SDK from Vivado with the opened Zynq project from earlier with the HDF file generated. This could be either for ZCU102 board or custom Zynq board. Launch the SDK by selecting File > Launch SDK.

<u>F</u> ile	<u>E</u> dit F <u>l</u> ow <u>T</u> ools	<u>W</u> indow	La <u>v</u> out <u>V</u> iew <u>H</u> elp Debug <u>Q</u> - Quick Acce	SS
	New Project		Φ Σ ½ ∅ ¾	
	Open Project		OJECT MANAGER project_1	
	Open <u>R</u> ecent Project	Þ	iources ?	
	Open Ex <u>a</u> mple Project			
	S <u>a</u> ve Project As			٠
	Write Project Tcl		Design Sources (1)	
	Archive Project		Orstraints Constraints	(1)
	<u>C</u> lose Project		Simulation Sources (1)	
	Open Checkpoint Open Recent Checkpoint	Þ		
	Write Checkpoint	P	Hierarchy IP Sources Libraries Compile Or	rder
	New IP Location			
	Open IP Location		Properties ? _	
	Open Recent IP Location	Þ	← (⇒ ✿
	Ne <u>w</u> File			
	Open Fi <u>l</u> e	Ctrl+0	Select an object to see properties	
	Open Recent <u>F</u> ile	Þ		
	Open IP-XA <u>C</u> T File			
	<u>S</u> ave File	Ctrl+S	Tcl Console Messages Log Reports De	esign Runs
	Sa <u>v</u> e File As		$\mathbf{Q} \mid \mathbf{X} \mid \mathbf{a} \mid $	
	Save All F <u>i</u> les		Name	Constraints
	Add So <u>u</u> rces Open Source File	Alt+A Ctrl+N	> b synth_1 (active)	constrs_1
		Cultin	▷ impl_1	constrs_1
	Import Export	×	Context Module Runs ✓ design_1	
	Launch SDK	r		
	– Open Log File			
	Open Journal File			
	<u>P</u> rint	Ctrl+P		

Figure 217: Launching the SDK





2. Provide the **Exported Location** path (where your *.hdf file was exported). The SDK **Workspace** could also be the same path. This creates the SDK workspace. Click **OK**.

Launch SDK ×
Launch software development tool.
Exported location: 🛜 <local project="" to=""> 🗸</local>
Workspace: 🕞 <local project="" to=""> 🗸 🗸 🗸</local>
OK Cancel

Figure 218: Creating the SDK Workspace

- 3. Once the SDK is open, select **File > New > Application Project** to open the New Project window. Provide a name for the FSBL project
- 4. The Target Hardware section is be populated by your generated hardware platform from Vivado along with the processor **psu_cortexa53_0**. Keep this processor for FSBL generation.

Note: The psu_cortexr5_0 is used by the IBERT GTR. No other applications should use this core.

5. For psu_cortexa53_*, select 64-bit (default) for Compiler.





Lab 10: Using Vivado Serial Analyzer to Debug GTR Serial Links

SOK	New Project 🗆 🗆 🗙			
Application Project				
Create a managed ma	ke application project.			
Project name: pcie_d	p_usb_sata			
✓ Use default location				
Location: //home	Browse			
Choose file :	system: default 🗘			
OS Platform: standa	lana			
	lone			
Target Hardware				
Hardware Platform:	design_1_wrapper_hw_platform_0 Image: New			
Processor:	psu_cortexa53_0 🗘			
Target Software				
Language:	● C ○ C++			
Compiler:	64-bit			
Hypervisor Guest:	No \$			
Board Support Packa				
Board Support Packa				
	O Use existing pcie_usb_dp_sata_fsbl_bsp 🗘			
?	< Back Next > Cancel Finish			

Figure 219: Creating a New Project

6. Leave other options set to their defaults and click **Next**.





7. Select the **Zynq MP FSBL** template.

🚾 💿 New Project	• • • • •
Templates Create one of the available t application project.	emplates to generate a fully-functioning
Available Templates:	
Empty Application Hello World IwIP Echo Server Memory Tests Peripheral Tests Zyng MP DRAM tests Zyng MP FSBL	First Stage Bootloader (FSBL) for Zynq Ultrascale+ MPSoC. The FSBL configures the FPGA with HW bit stream (if it exists) and loads the Operating System (OS) Image or Standalone (SA) Image or 2nd Stage Boot Loader image from the non- volatile memory (NAND/SD/QSPI) to RAM (DDR) and takes A53/R5 out of reset. It supports multiple partitions, and each partition can be a code image or a bit stream.
? < Back	Next > Finish Cancel

Figure 220: Selecting the New Project Template

- 8. Click **Finish** to generate the A53 FSBL. This populates the FSBL code and also builds it, along with the BSP.
- 9. Change the compiler optimization flag from 00 to 02. To change this in the SDK, follow these steps:
 - a. Right-click the FSBL application project and select C/C++ Build Settings.
 - b. On the Tool Settings tab, select Optimization (under the ARM A53 gcc compiler).
 - c. On the **Optimization Level** drop-down menu, change **None (-O0)** to **Optimize more (-O2)**.
 - d. Click **OK** to close the Properties window.
- 10. Debug prints in FSBL are now disabled by default (except for FSBL banner). To enable debug prints, you must define the symbol FSBL_DEBUG_INFO. To change this in the SDK, follow these steps:
 - a. Right-click the FSBL application project and select **C/C++ Build Settings**.
 - b. On the Tool Settings tab, select Symbols (under the ARM A53 gcc compiler).
 - c. Click the **Add** button in the **Defined symbols (-D)** section and enter the value: FSBL_DEBUG_INFO
 - d. Click **OK** to close the Properties window.





11. Skip this step if you are building FSBL for a custom board.

For building the FSBL for a ZCU102, define the additional symbol XPS_BOARD_ZCU102. Note that in this flow, the Hardware Platform (HDF) was created using the ZCU102 board preset, so the XPS_BOARD_ZCU102 symbol may be defined by default in the FSBL. If it is present, it does not need to be redefined.

To define the symbol in the SDK, follow these steps:

- a. Right-click the FSBL application project and select C/C++ Build Settings.
- b. On the Tool Settings tab, select Symbols (under ARM A53 gcc compiler).
- c. Click the **Add** button in the **Defined symbols (-D)** section and enter the value: XPS_BOARD_ZCU102
- d. Click **OK** to close the Properties window.
- 12. If the current configuration has **USB** set on one of the GTR lanes, then the FSBL source files need to be modified. This is needed to set the USB in a linked state. This step should only be performed when using an "FSBL only" setup with IBERT GTR. When Linux setup is used, USB drivers set the link appropriately and this step is not needed.
- 13. Open the files in SDK mentioned in next step, located in the applications src folder.

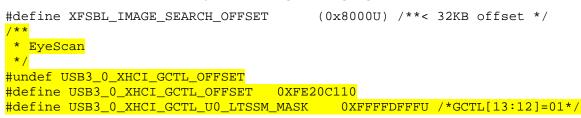
Make the appropriate changes and save the file, which automatically builds the project and generates <application_name>.elf in the Debug folder inside the applications folder.

If building for a custom board, make sure you do the appropriate changes to bring up the USB in a linked state. After this step the FSBL should be ready to be used in Vivado.

Bringing the USB into a Linked State

xfsbl_main.h

After the first line below, add the EyeScan changes as highlighted to xfsbl_main.h:







xfsbl_main.c

After the lines below, add the EyeScan changes as highlighted to xfsbl_main.c:

```
XFsbl_Printf(DEBUG_INFO,
    "============ In Stage 4 ========= \n\r");
//EyeScan
u32 RegValue;
RegValue = Xil_In32(USB3_0_XHCI_GCTL_OFFSET) &
    USB3_0_XHCI_GCTL_U0_LTSSM_MASK;
Xil_Out32(USB3_0_XHCI_GCTL_OFFSET, RegValue);
```

Generating using XSCT Manual Flow:

- 1. To create an FSBL for the A53 #0 (64 bit), type xsct from shell in the directory where the HDF file is located.
- 2. From the xsct console, issue the following commands:

```
setws
createhw -name hw0 -hwspec <path to hdf generated by you>
createapp -name fsbl_design_1 -app {Zynq MP FSBL} -proc psu_cortexa53_0 -
    hwproject hw0 -os standalone -arch 64 -lang C
configapp -app fsbl_design_1 -set compiler-optimization {Optimize more (-O2)}
configapp -app fsbl_design_1 -add define-compiler-symbols FSBL_DEBUG_INFO
```

3. If using the ZCU102, also run this command:

configapp -app fsbl_design_1 -add define-compiler-symbols XPS_BOARD_ZCU102

- 4. Make changes to xfsbl_main.c and xfsbl_main.h as shown above in Bringing the USB into a Linked State if the USB is in one of the GTR lanes. Save these files.
- 5. Run the following command:

projects -build

The FSBL ELF file, fsbl_design_1.elf, will be generated in the fsbl_design_1/Debug/ directory.





Legal Notices

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 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 2012-2019 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. PCI, PCIe and PCI Express are trademarks of PCI-SIG and used under license. All other trademarks are the property of their respective owners.

