

Vivado Design Suite Tutorial

Programming and Debugging

UG936 (v2020.2) February 26, 2021





Revision History

The following table shows the revision history for this document.

Section	Revision Summary		
02/26/2021 Version 2020.2			
Lab 10: Using the Vivado Serial Analyzer to Debug PS-GTR Serial Links	Reinstated lab and updated for Vivado 2020.2 release.		
06/24/2020 Version 2020.1			
General updates.	Updated for Vivado 2020.1 release.		





Table of Contents

Revision History	2
Debugging in Vivado Tutorial	6
Navigating Content by Design Process	6
Objectives	7
Getting Started	
Lab 1: Using the Netlist Insertion Method to Debug a Design	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	23
Lab 2: Using the HDL Instantiation Method to Debug a Design	24
Step 1: Creating a Project with the Vivado New Project Wizard	24
Step 2: Synthesize Implement and Generate Bitstream	26
Lab 3: Using a VIO Core to Debug a Design in Vivado Design	
Suite	27
Step 1: Creating a Project with the Vivado New Project Wizard	28
Step 2: Synthesize, Implement, and Generate the Bitstream	32
Lab 4: Using the Synplify Pro Synthesis Tool and Vivado Design	
Suite to Debug a Design	34
Step 1: Create a Synplify Pro Project	
Step 2: Synthesize the Synplify Project	40
Step 3: Create DCPs for the Black Box Created in Synplify Pro	41
Step 4: Create a Post Synthesis Project in Vivado IDE	41
Step 5: Add More Debug Nets to the Project	42
Step 6: Implementing the Design and Generating the Bitstream	

Stei	o 1: Verifvir	ng Operation	of the Sine Wave	e Generator	
000	,	ig operation		Generator	



Step 2: Debugging the Sine Wave Sequencer State Machine (Optional)57
Lab 6: Using the ECO Flow to Replace Debug Probes Post Implementation75
Lab 7: Debugging Designs Using the Incremental Compile Flow 88
Procedure.88Step 1: Opening the Example Design and Adding a Debug Core.88Step 2: Compiling the Reference Design.92Step 3: Create New Runs.93Step 4: Making Incremental Debug Changes.95Step 5: Running Incremental Compile.98Conclusion.100
Lab 8: Using the Vivado Serial Analyzer to Debug Serial Links101
Design Description
Lab 9: Using the Vivado ILA Core to Debug JTAG-AXI
Transactions
Design Description
Lab 10: Using the Vivado Serial Analyzer to Debug PS-GTR Serial
Links151IBERT PS-GTR Flow151Step 1: Generating a Zynq UltraScale+ MPSoC PS Xilinx Support Archive153Step 2: Using Xilinx Software Command-line Tool Flow to Generate a First Stage163Boot Loader163Step 3: ZCU102 Board Settings164Using FSBL with Serial I/O Analyzer to Bring Up IBERT PS-GTR165Troubleshooting172





Appendix A: Additional Resources and Legal Notices	174
Xilinx Resources	.174
Documentation Navigator and Design Hubs	174
Please Read: Important Legal Notices	175





Debugging in Vivado Tutorial

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

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

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

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

Navigating Content by Design Process

Xilinx[®] documentation is organized around a set of standard design processes to help you find relevant content for your current development task. All Versal[™] ACAP design process Design Hubs can be found on the Xilinx.com website. This document covers the following design processes:

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



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

Objectives

These tutorials:

- Show you how to take advantage of integrated Vivado[®] 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.

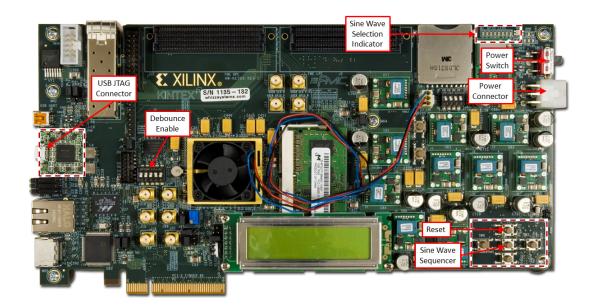
Software

Vivado[®] Design Suite 2020.2

Hardware

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

Figure 1: KC705 Board Showing Key Components



Tutorial Design Components

Labs 1 through 4 include:

• A simple control state machine



- Three sine wave generators using AXI4-Stream interface, native DDS Compiler
- Common push buttons (GPIO_BUTTON)
- DIP switches (GPIO_SWITCH)
- LED displays (GPIO_LED) VIO Core (Lab 3 only)
- **Pushbutton Switches:** Serve as inputs to the de-bounce and control state machine circuits. Pushing a button generates a high-to-low transition pulse. Each generated output pulse is used as an input into the state machine.
- DIP Switch: Enables or disables a de-bounce circuit.
- **De-bounce Circuit:** This example, when enabled, provides a clean pulse or transition from high to low. Eliminates a series of spikes or glitches when a button is pressed and released.
- Sine Wave Sequencer State Machine: Captures and decodes input from the two push buttons. Provides sine wave selection and indicator circuits, sequencing among 00, 01, 10, and 11 (zero to three).
- **LED Displays:** GPIO_LED_0 and GPIO_LED_1 display selection status from the state machine outputs, each of which represents a different sine wave frequency: high, medium, and low.

Lab 5 includes:

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

Board Support and Pinout Information

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 Reference Design Files from the Xilinx website.

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

- 3. Unzip the tutorial source file to the /Vivado_Debug folder. There are six labs that use different methodologies for debugging your design. Select the appropriate lab and follow the steps to complete them.
- Lab 1: This lab walks you through the steps of marking nets for debug in HDL as well as the post-synthesis netlist (netlist insertion method). Following are the required files:
 - debounce.vhd
 - fsm.vhd
 - sinegen.vhd
 - sinegen_demo.vhd
 - sine_high/sine_high.xci
 - sine_low/sine_low.xci
 - sine_mid/sine_mid.xci
 - sinegen_demo_kc705.xdc
- Lab 2: This lab goes over the details of marking nets for debug in the source HDL (HDL instantiation method) as well as instantiating an ILA core in the HDL. Following are the required files:
 - debounce.vhd
 - fsm.vhd
 - sinegen.vhd
 - sinegen_demo_inst.vhd
 - ila_0/ila_0.xci
 - sine_high/sine_high.xci
 - sine_low/sine_low.xci
 - sine_mid/sine_mid.xci
 - sinegen_demo_kc705.xdc
- Lab 3: You can test your design even if the hardware is not physically accessible, using a VIO core. This lab walks you through the steps of instantiating and customizing a VIO core that you will hook to the I/Os of the design. Following are the required files:
 - debounce.vhd
 - fsm.vhd



- sinegen.vhd
- sinegen_demo_inst_vio.vhd
- sine_high/sine_high.xci
- sine_low/sine_low.xci
- sine_mid/sine_mid.xci
- ila_0/ila_0.xci
- sinegen_demo_kc705.xdc
- Lab 4: Nets can also be marked for debug in a third-party synthesis tool using directives for the synthesis tool. This lab walks you through the steps of marking nets for debug in the Synplify tool and then using Vivado[®] 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: Use the IBERT UltraScale+ PS-GTR transceiver to evaluate and monitor PS-GTR transceivers in Zynq[®] UltraScale+[™] MPSoC devices. This lab is purely software-based, setting up and testing the processing system (PS) side of the Zynq UltraScale+ MPSoC device with no programmable logic (PL).

Connecting the Boards and Cables

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

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





Lab 1

Using the Netlist Insertion Method to Debug a Design

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

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

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

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

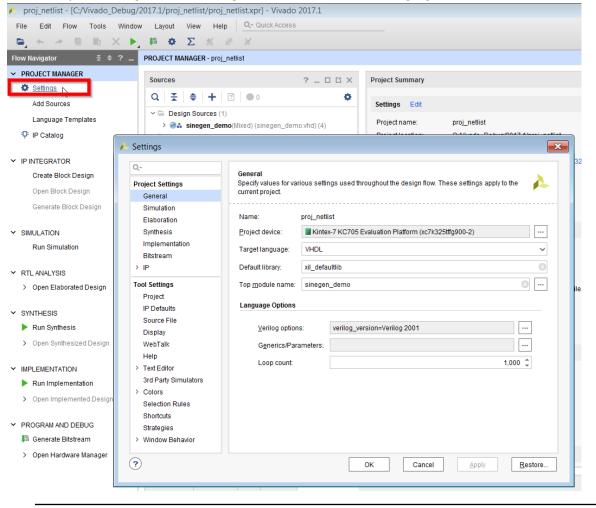


- 10. In the Add Constraints dialog box, click the "+" sign, and then click Add Files.
- 11. Navigate to /src/lab1 directory and select sinegen_demo_kc705.xdc. Click Next.
- 12. In the Default Part dialog box, specify the **xc7k325tffg900-2** part for the KC705 platform. You can also select **Boards** and then select **Kintex-7 KC705 Evaluation Platform**. Click **Next**.
- 13. Review the New Project Summary page. Verify that the data appears as expected, per the steps above, and click **Finish**.

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

Step 2: Synthesizing the Design

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



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



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

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

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

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

Step 3: Probing and Adding Debug IP

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

In this step, you will accomplish the following tasks:

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

Adding Debug Nets to the Project

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



- Add MARK_DEBUG attribute to HDL files.
 - VHDL:

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

• Verilog:

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

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

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

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

• Use a Tcl prompt to set the MARK_DEBUG attribute on a synthesized netlist.

set_property mark_debug true [get_nets -hier [list {sine[*]}]]

This applies the MARK_DEBUG on the current, open netlist.

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

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

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

Use Ctrl-Q to toggle it off and on.

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





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

attribute mark_debug : string; attribute mark_debug of GPIO_BUTTONS_db : signal is "true"; attribute mark_debug of GPIO_BUTTONS_re : signal is "true"; attribute mark_debug of GPIO_BUTTONS_re : signal is "true"; attribute mark_debug of DONT_EAT : signal is "true"; attribute mark_debug of DONT_EAT : signal is "true"; component sinegen port (clk : in std_logic; reset : in std_logic_vector(1 downto 0); sine : out std_logic_vector(19 downto 0); ; end component;					
attribute mark_debug of GPI0_BUTTONS_db : signal is "true"; attribute mark_debug of GPI0_BUTTONS_dly : signal is "true"; attribute mark_debug of GPI0_BUTTONS_re : signal is "true"; attribute mark_debug of DONT_EAT : signal is "true"; attribute mark_debug of GPI0_BUTONS_re : signal is "true"; attribute mark_debug of GPI0_BUTONS_re : signal is "true"; attribute mark_debug of GPI0_BUTONS_re : signal is "true"; attribute mark_debug of GPI0_BUTONS_db[0] : signal is "true"; attribute mark_debug of GPI0_BUTTONS_db[1] : FDRE Q image : true is std_logic_vector(19 downto 0); sine : out std_logic_vector(19 downto 0); is geno_BUTTONS_db (2) : FDRE Q image : true is std_logic_vector(19 downto 0); sine : out std_logic_vector(19 downto 0); image : true is std_logic_vector(19 downto 0); sine : out std_logic_vector(19 downto 0); image : true is std_logic_vector(19 downto	63 attribute mark_debug : string;	show debug nets :	in the synthesized		
attribute mark_debug of GPI0_BUTTONS_dly : signal is "true"; attribute mark_debug of GPI0_BUTTONS_re : signal is "true"; attribute mark_debug of DONT_EAT : signal is "true"; attribute mark_debug of construction of DONT_EAT : signal is "true"; attribute mark_debug of DONT_EAT : signal is "true"; attribute mark_debug of construction of DONT_EAT : signal construction of					
attribute mark_debug of GPI0_BUTTONS_re : signal is "true"; attribute mark_debug of DONT_EAT : signal is "true"; attribute mark_debug of donated is the signal is "true"; attribute mark_debug of DONT_EAT : signal is "true"; attribute mark_debug of donated is the signal is "true"; attribute mark_debug of donated is the signal is "true"; attribute mark_debug of donated is the signal is the signal is "true"; <		-			
attribute mark_debug of DONT_EAT : signal is "true"; component sinegen port (clk : in std_logic; sel : in std_logic_vector(1 downto 0); sine : out std_logic_vector(19 downto 0);); end component; cl Console Messages Log Reports Design Runs Debug × cl Console Messages Log Reports Debug Vector (19 downto 0); std GPI0_BUTTONS_db(2) FDRE Q cl Console Messages Log Reports Debug Vector (19 downto 0); std GPI0_BUTTONS_dly(0) FDRE Q cl Console Messages Console Messages Log Reports Debug Vector (19 downto 0); std GPI0_BUTTONS_re(2) FDRE Q cl Console Messages Console Message					
8 component sinegen port (2 (3 clk : in std_logic; reset : in std_logic, vector (1 downto 0); sine : out std_logic_vector (19 downto 0); sine : out std_logic_vector (19 downto 0); isine : out std_logic_vector (20 for Cell Isine : OPIO_BUTTONS_re(0)					
a component sinegen port (a clk : in std_logic; sel : in std_logic, vector(1 downto 0); sine : out std_logic_vector(19 downto 0); sine : out std_logic_vector(19 downto 0);); end component; Cl Console Messages Log Reports Design Runs Debug × Q ★ ★ ★ • • • • × • × • × • <td></td> <td>: signal is "tr</td> <td>ie";</td>		: signal is "tr	ie";		
0 ⇒ component sinegen 1 port 2 (3 clk : in std_logic; 4 reset : in std_logic; 5 sel : in std_logic; vector(1 downto 0); 5 sel : out std_logic_vector(19 downto 0); 6 sine : out std_logic_vector(19 downto 0); 7); 9 ← end component; Cl Console Messages Log Reports Design Runs Debug × 0 ★ ★ ★ - - × 1 ★ ★ + - - × × 9 _ end component; Design Runs Debug × × 1 ★ ★ + - - × > </td <td>69</td> <td></td> <td></td>	69				
1 port 2 (3 clk : in std_logic; 3 sel : in std_logic_vector(1 downto 0); 5 sel : in std_logic_vector(19 downto 0); 6 sine : out std_logic_vector(19 downto 0); 7); 8 end component; Cl Console Messages Log Reports Design Runs Debug × 0 ₹ € € ↑ • >	1				
a clk : in std_logic; reset : in std_logic; sel : in std_logic_vector(1 downto 0); sine : out std_logic_vector(19 downto 0););) end component; Cl Console Messages Log Reports Design Runs Debug × Q ★ ★ ★ ★ ★ × > × > × > × > × > × > × > × > × > × > × > × > × > × > × >					
4 reset : in std_logic; sel : in std_logic_vector(1 downto 0); sine : out std_logic_vector(19 downto 0);); 6 sine : out std_logic_vector(19 downto 0); sine : out std_logic_vector(19 downto 0);); 9 end component; 2 * 2 * 2 * * * 9 end component; 2 * 2 * * * <	72 (
sel : in std_logic_vector(1 downto 0); sine <td: out<="" td=""> std_logic_vector(19 downto 0) 7); 9 end component; 2 2 2 2 2 2 2 2 2 2 2 2 3 3 4 3 5 5 1 2 2 2 2 3 4 3 5 5 1 5 2 3 3 5 3 5 4 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 5 6 5 7 5 7 5 7 5 6 6 7</td:>	73 clk : in std_logic;				
sel : in std_logic_vector(1 downto 0); sine <td: out<="" td=""> std_logic_vector(19 downto 0) 7); 9 end component; 2 2 2 2 2 2 2 2 2 2 2 2 3 3 4 3 5 5 1 2 2 2 2 3 4 3 5 5 1 5 2 3 3 5 3 5 4 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 5 6 5 7 5 7 5 7 5 6 6 7</td:>	74 reset : in std_logic;				
n n	75 sel : in std_logic_vector				
Best of component; end component; Design Runs Debug × Cl Console Messages Log Reports Design Runs Debug × Q Image Image Image Driver Cell Driver Pin Iame Driver Cell Driver Pin Image Image Image Driver Cell Driver Pin Image Image Image Driver Cell Driver Pin Image Image Image Image Image Image Image Image Im		or(19 downto 0)			
Image: Problem					
Cl Console Messages Log Reports Design Runs Debug × Q ₹ € ¥ † ⇒ Driver Cell Driver Pin Iame Driver Oll Driver Cell Driver Pin Image: Unassigned Debug Nets (7) FDRE Q ✓ Image: GPIO_BUTTONS_db (2) FDRE Q 「Image: GPIO_BUTTONS_db (2) FDRE Q Image: GPIO_BUTTONS_dly (2) FDRE Q Image: GPIO_BUTTONS_dly (1) FDRE Q Image: GPIO_BUTTONS_dly (1) FDRE Q Image: GPIO_BUTTONS_re (2) FDRE Q Image: GPIO_BUTTONS_re (0) FDRE Q	78				
Ame Driver Cell Driver Pin Image: Image	/9 🖂 ena component;				
Ame Driver Cell Driver Pin Image: Image		rts Design Pu	ns Debug V		
Iame Driver Cell Driver Pin □ Unassigned Debug Nets (7) ✓ ✓ Image: Second	Ter console messages Log Repor	Design Ku	Debug A		
Image: Constraint of the constrain	Q 素 ≑ 兼 ∔ ≓				
✓ 小菜 GPIO_BUTTONS_db (2) FDRE Q 「菜 GPIO_BUTTONS_db[0] FDRE Q 「菜 GPIO_BUTTONS_db[1] FDRE Q 「菜 GPIO_BUTTONS_dly (2) FDRE Q 「菜 GPIO_BUTTONS_dly[0] FDRE Q 「菜 GPIO_BUTTONS_dly[1] FDRE Q 「菜 GPIO_BUTTONS_re (2) FDRE Q 「菜 GPIO_BUTTONS_re [0] FDRE Q	Name	Driver Cell	Driver Pin		
「☆ GPIO_BUTTONS_db[0] FDRE Q 「☆ GPIO_BUTTONS_db[1] FDRE Q ✓ 小☆ GPIO_BUTTONS_dly(2) FDRE Q 「☆ GPIO_BUTTONS_dly[0] FDRE Q 「☆ GPIO_BUTTONS_dly[1] FDRE Q 「☆ GPIO_BUTTONS_dly[1] FDRE Q 「☆ GPIO_BUTTONS_re(2) FDRE Q 「☆ GPIO_BUTTONS_re[0] FDRE Q	v 🗖 Upperiated Debug Nate (7)				
「☆ GPIO_BUTTONS_db[1] FDRE Q ✓ 小☆ GPIO_BUTTONS_dly(2) FDRE Q 「☆ GPIO_BUTTONS_dly[0] FDRE Q 「☆ GPIO_BUTTONS_dly[1] FDRE Q 「☆ GPIO_BUTTONS_re (2) FDRE Q 「☆ GPIO_BUTTONS_re[0] FDRE Q	 Onassigned Debug Nets (7) 				
✓ 小菜 GPIO_BUTTONS_dly(2) FDRE Q 「菜 GPIO_BUTTONS_dly[0] FDRE Q 「菜 GPIO_BUTTONS_dly[1] FDRE Q 「菜 GPIO_BUTTONS_re(2) FDRE Q 「菜 GPIO_BUTTONS_re[0] FDRE Q		FDRE	Q		
「☆ GPIO_BUTTONS_dly[0] FDRE Q 「☆ GPIO_BUTTONS_dly[1] FDRE Q ✓ 小☆ GPIO_BUTTONS_re (2) FDRE Q 「☆ GPIO_BUTTONS_re[0] FDRE Q	✓ Jr¤ GPIO_BUTTONS_db (2)		-		
「☆ GPIO_BUTTONS_dly[1] FDRE Q ✓ 小☆ GPIO_BUTTONS_re (2) FDRE Q 「☆ GPIO_BUTTONS_re[0] FDRE Q	✓ √f ☆ GPIO_BUTTONS_db (2) ∫ ☆ GPIO_BUTTONS_db[0]	FDRE	Q		
✓ Intra GPIO_BUTTONS_re(2) FDRE Q Intra GPIO_BUTTONS_re[0] FDRE Q	✓ √r a GPIO_BUTTONS_db (2) ∫ a GPIO_BUTTONS_db[0] ∫ a GPIO_BUTTONS_db[1]	FDRE FDRE	Q Q		
_F GPIO_BUTTONS_re[0] FDRE Q	 ✓ 「☆ GPIO_BUTTONS_db (2) ∫☆ GPIO_BUTTONS_db[0] ∫☆ GPIO_BUTTONS_db[1] ✓ 「☆ GPIO_BUTTONS_dly (2) 	FDRE FDRE FDRE	Q Q Q		
_F GPIO_BUTTONS_re[0] FDRE Q	 ✓ Jr ≈ GPIO_BUTTONS_db (2) J ≈ GPIO_BUTTONS_db[0] J ≈ GPIO_BUTTONS_db[1] ✓ Jr ≈ GPIO_BUTTONS_dly (2) J ≈ GPIO_BUTTONS_dly[0] 	FDRE FDRE FDRE FDRE	Q Q Q Q		
	 ✓ √r ☎ GPIO_BUTTONS_db (2) ∫ ☎ GPIO_BUTTONS_db[0] ∫ ☎ GPIO_BUTTONS_db[1] ✓ √r ☎ GPIO_BUTTONS_dly (2) ∫ ☎ GPIO_BUTTONS_dly[0] ∫ ☎ GPIO_BUTTONS_dly[1] 	FDRE FDRE FDRE FDRE FDRE			
	 ✓ Jr¤ GPIO_BUTTONS_db (2) J¤ GPIO_BUTTONS_db[0] J¤ GPIO_BUTTONS_db[1] ✓ Jr¤ GPIO_BUTTONS_dly (2) J¤ GPIO_BUTTONS_dly[0] J¤ GPIO_BUTTONS_dly[1] ✓ Jr¤ GPIO_BUTTONS_re (2) 	FDRE FDRE FDRE FDRE FDRE FDRE			
_ S DONT_EAT FDRE Q	 ✓ Jr ≈ GPIO_BUTTONS_db (2) J ∞ GPIO_BUTTONS_db[0] J ∞ GPIO_BUTTONS_db[1] ✓ Jr ≈ GPIO_BUTTONS_dly(2) J ∞ GPIO_BUTTONS_dly[0] J ∞ GPIO_BUTTONS_dly[1] ✓ Jr ≈ GPIO_BUTTONS_re (2) J ∞ GPIO_BUTTONS_re[0] 	FDRE FDRE FDRE FDRE FDRE FDRE FDRE			
	 ✓ J[™] GPIO_BUTTONS_db (2) J[™] GPIO_BUTTONS_db[0] J[™] GPIO_BUTTONS_db[1] ✓ J[™] GPIO_BUTTONS_dly (2) J[™] GPIO_BUTTONS_dly[0] J[™] GPIO_BUTTONS_dly[1] ✓ J[™] GPIO_BUTTONS_re (2) J[™] GPIO_BUTTONS_re[0] J[™] GPIO_BUTTONS_re[1] 	FDRE FDRE FDRE FDRE FDRE FDRE FDRE FDRE			

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

Debug Nets

Debug Cores



• sine(20): Nets folder under the U_SINEGEN hierarchy



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

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





Sources Netlist ×		? _ 🗆 🖸	Schema	tic ×	sinegen_de
¥ H		٠	Q	🔸	≁ ჯ
sinegen_demo		<u>^</u>	51	signal	GPIO_BUT
Nets (60)			52	signal	GPIO_BUT
_			53	signal	GPIO_BUT
> 📲 GPIO_BUTTONS (2)			54		
> 小☆ GPIO_BUTTONS_db			55	-	DONT_EAT
> 小電 GPIO_BUTTONS_dly	(2)		56	-	DONT_EAT DONT EAT
✓ IF GPIO_BUTTONS_IBUF	(2)		58	-	DONT EAT
	BUF[0]		59	-	DONT EAT
_ GPIO_BUTTONS_I		Net Properties		Ctrl+E	EAT
> 🖟 🛱 GPIO_BUTTONS_re	(2)	Mark Debug			deb
> 🖅 LEDS_n (4)	- 266	-			Wark_
> 「 LEDS_n_OBUF (2)		Unmark Debug			ark_
_ <const0></const0>		Assign to Debu	g Port		ark_
∫ <const1></const1>	_	Select Driver Pi	n		ark_ ark
Net Properties	H	Schematic		F4	
		Show Connecti	vity	Ctrl+T	ineg
GPIO_BUTTONS_IBUF[1]	_	Show Hierarchy	1	F6	
Name: GPIO_BUTT	ONS_	Highlight			▶ in
Type: SIGNAL	1				in
.,,,		Unhighlight			in
Bus net: 🥼 GPIO_E		Mark			 out
Route status: Has unplace	ed por	Unmark		Ctrl+Shi	ft+M
Cell pin count: 4			79 白	and	mponent;

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 in the following two figures.





	Messages	Log	Reports	Design Run	s Debug	×
Q	≑ 兼 +					
Name			Driv	ver Cell	Driver Pin	
🗠 🖨 Unass	signed Debug Ne	ets (29)				
~ -∱¤ G	PIO_BUTTONS_	db (2)	FDF	RE	Q	
œ	GPIO_BUTTON	IS_db[0]	FDF	RE	Q	
ت]	GPIO_BUTTON	IS_db[1]	FDF	RE	Q	
	PIO_BUTTONS_		FDF		Q	
	GPIO_BUTTON		FDF		Q	
_	GPIO_BUTTON		FDF		Q	
	PIO_BUTTONS_		IBU		0	
					0	
_	GPIO_BUTTON PIO_BUTTONS		IBU FDF		Q	
	.GPIQ_BUTTONS_				0	
Debug Core				NF	. 3 4	
Courses	Notlint V				7.54	
Sources	Netlist ×			? _ [
¥ H					٠	
- ,	_demo				¢	
- ,	_				¢ Î	
🛚 sinegen 🗸 🖻 Nets	_	NS (2)			¢ Î	
N sinegen ✓ ☐ Nets → ₩	s (60)		(2)		Ô	
Nisinegen ~ 📄 Nets > √fr > 小fr章	GPIO_BUTTO GPIO_BUTTO	ONS_db			¢	
N sinegen ~ 一 Nets > 婚 > 乐章 > 乐章	GPIO_BUTTO GPIO_BUTTO GPIO_BUTT GPIO_BUTT	ONS_db	(2)		¢ Î	
N sinegen ~ 一 Nets > 婚 > 乐窓 > 乐窓 ~ 乐窓	GPIO_BUTTO GPIO_BUTTO GPIO_BUTT GPIO_BUTT GPIO_BUTT	ONS_db ONS_dly ONS_IBU	(2) JF (2)		¢	
N sinegen ~	GPIO_BUTTO GPIO_BUTT GPIO_BUTT GPIO_BUTT GPIO_BUTT	ONS_db ONS_dly ONS_IBU UTTONS	(2) JF (2) _IBUF[0]		Ŷ	
N sinegen ~	GPIO_BUTTO GPIO_BUTT GPIO_BUTT GPIO_BUTT J C GPIO_B J C GPIO_B	ONS_db ONS_dly ONS_IBU UTTONS UTTONS	(2) JF (2) _IBUF[0] _IBUF[1]		¢	
N sinegen ~ 一 Nets > 小 > 小 小 小 小 小 二 、 小 二 、 小 二 、 い 本 、 小 二 、 小 二 、 小 二 、 小 二 、 小 二 、 小 二 、 小 二 、 小 二 、 小 二 、 小 二 、 、 小 二 、 、 小 二 、 、 小 二 、 、 小 二 、 、 小 二 、 、 小 二 、 、 小 二 、 、 小 二 、 、 小 二 、 、 小 二 、 、 小 二 、 、 、 小 二 、 、 、 、 、 小 二 、 、 、 、 、 、 、 、 、 、 、 、 、	GPIO_BUTTO GPIO_BUTT GPIO_BUTT GPIO_BUTT J¤ GPIO_B J¤ GPIO_B GPIO_BUTT	ONS_db ONS_dly ONS_IBU UTTONS UTTONS	(2) JF (2) _IBUF[0] _IBUF[1]		Ŷ	
N sinegen ~ 一 Nets > 小 > 小 - 小 - 小 - 小 - 小 - - - - - - - - - - - - -	GPIO_BUTTO GPIO_BUTT GPIO_BUTT GPIO_BUTT J © GPIO_B J © GPIO_B GPIO_BUTT LEDS_n (4)	ONS_db ONS_dly ONS_IBU UTTONS UTTONS ONS_re	(2) JF (2) _IBUF[0] _IBUF[1]		¢	
N sinegen ~ 一 Nets > 小 > 小 - 小 - 小 - 小 - 小 - 小 - 小 - 小 -	GPIO_BUTTO GPIO_BUTT GPIO_BUTT GPIO_BUTT J © GPIO_B J © GPIO_B GPIO_BUTT LEDS_n (4) LEDS_n_OBL	ONS_db ONS_dly ONS_IBU UTTONS UTTONS ONS_re	(2) JF (2) _IBUF[0] _IBUF[1]		Ŷ	
N sinegen ~ 一 Nets > 小 > 小 - 小 - 小 - 小 - 小 - 小 - 小 - 小 -	GPIO_BUTTO GPIO_BUTT GPIO_BUTT GPIO_BUTT J © GPIO_B J © GPIO_B GPIO_BUTT LEDS_n (4)	ONS_db ONS_dly ONS_IBU UTTONS UTTONS ONS_re	(2) JF (2) _IBUF[0] _IBUF[1]		Ŷ	

Running the Set Up Debug Wizard

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





Tcl Console Messa	ges Log Rej	ports Design	Runs Debug ×	
Q ≚ ≑ 🕷	, † ⊨			
Name	Set Up Debug	Driver Cell	Driver Pin	
👻 🚍 Unassigned Del	bug Nets (29)			
✓ - J̄r ≅ GPIO_BUTT	ONS_db (2)	FDRE	Q	
_ SPIO_BU	JTTONS_db[0]	FDRE	Q	
_∫¤ GPIO_BU	Q			
✓ - J [™] CPIO_BUTT	ONS_dly (2)	FDRE	Q	
_∫¤ GPIO_BU	JTTONS_dly[0]	FDRE	Q	
_∫¤ GPIO_BU	Q			
✓ - J̄r ≅ GPIO_BUTT	ONS_IBUF (2)	IBUF	0	
_∫¤ GPIO_BU	0			
_ SPIO_BUTTONS_IBUF[1] IBUF 0				
✓ - fr to GPIO_BUTT	ONS_re (2)	FDRE	Q	
Γ.ΰ	ITTONS, relo1	FDRF	Ω	
Debug Cores Debu	ug Nets			

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

🍌 Set Up Debug	
HLx 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.
?	< <u>Back</u> <u>Einish</u> Cancel

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

Q ¥ ♦ № M +				Contract (1998)
Vame	Clock Domain	Driver Cell	Probe Type	
IF≅ GPIO_BUTTONS_db (2)	clk	FDRE	Data and Trigger 👒	
Jr ☎ GPIO_BUTTONS_dly (2)	clk	FDRE	Data and Trigger 👒	
Intra GPIO_BUTTONS_IBUF (2)	clk	IBUF	Data and Trigger 👒	
Int GPIO_BUTTONS_re (2)	clk	FDRE	Data and Trigger 👒	
Jrt≋ U_SINEGEN/sel (2)	clk	FDRE	Data and Trigger 🛛 🗸	
√ 小☆ 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	1

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

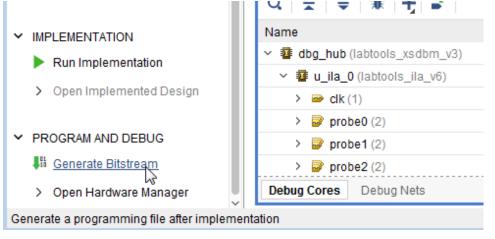
🏊 Set Up Debug	
	Set up Debug Summary
HLx Editions	I debug cores will be removed
	1 debug core will be created
	Found 1 clock
	✓ Open in Debug layout To apply the above changes, click Finish
(?)	< <u>B</u> ack <u>N</u> ext > <u>Finish</u> Cancel

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



Step 4: Implementing and Generating Bitstream

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



- In the Save Project dialog box click Save. If a dialog box appears indicating this will cause the Synthesis results to go out of date, click OK. This applies the MARK_DEBUG attributes on the newly marked nets. You can see those constraints by inspecting the sinegen_demo_kc705.xdc file.
- 3. When the No Implementation Results Available dialog box pops up, click **Yes**. In the Launch Runs dialog box, accept all of the default settings (Launch runs on local host) and click **OK**.
- 4. When the bitstream generation completes, the Bitstream Generation Completed dialog box pops up. Click **OK**.
- 5. In the dialog box asking you to closetye synthesized design before opening the implemented design. Click **Yes**.
- 6. Examine the Timing Summary report to ensure that all the specified timing constraints are met.

Tcl Console Messages Log Re	eports Design Runs Power	DRC N	lethodology	Timing \times			? _ 🗆 🗆
Q 素 ♠ ●	Design Timing Summary						
General Information Timer Settings	Setup		Hold			Pulse Width	
Design Timing Summary	Worst Negative Slack (WNS):	0.491 ns	Wors	t Hold Slack (WHS	i): 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	Num	ber of Failing Endp	ooints: 0	Number of Failing Endpoints:	0
> 🗁 Intra-Clock Paths	Total Number of Endpoints:	12755	Total	Number of Endpo	ints: 12755	Total Number of Endpoints:	6938
Inter-Clock Paths	All user specified timing constra	aints are m	et.				
> 🗁 Other Path Groups							
User Ignored Paths							
Unconstrained Paths							
Timing Summary - impl_1 (saved)							

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



Lab 2

Using the HDL Instantiation Method to Debug a Design

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

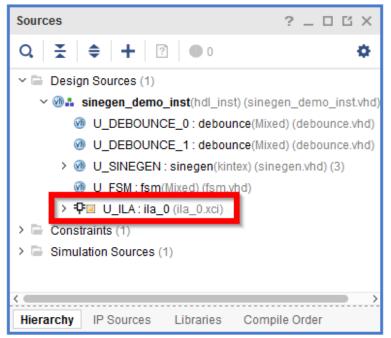
Step 1: Creating a Project with the Vivado New Project Wizard

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

- 1. Invoke the Vivado[®] IDE.
- 2. In the Quick Start tab, click Create Project to start the New Project wizard. Click Next.
- 3. In the Project Name page, name the new project proj_hdl and provide the project location (C:/Vivado_Debug). Ensure that Create project subdirectory is selected. Click Next.
- 4. In the Project Type page, specify the Type of Project to create as RTL Project. Click Next.
- 5. In the Add Sources page:
 - a. Set Target Language to VHDL.
 - b. Click the "+" sign, and then click Add Directories.
 - c. In the Add Source Directories dialog box, navigate to the /src/lab2 directory, and choose the sine_high, sine_low, sine_mid, and ila_0 directories. Click Select.
 - d. Verify that the directories are added, and Copy Sources into Project is selected.
 - e. Click the "+" sign, and then click Add File.
 - f. In the Add Source Files dialog box, navigate to the/src/lab2 directory and choose debounce.vhd, fsn.vhd, sinegen.vhd, and sinegen_demo_inst.vhd files. Click OK.
 - g. Verify that the sources and directories are added, and that **Copy Sources into Project** is selected. Click Next.



- 6. In the Add Constraints dialog box, click the "+" sign, and then click Add Files.
- 7. Navigate to /src/lab1 directory and select sinegen_demo_kc705.xdc. Click Next.
- 8. In the Default Part dialog box, specify the **xc7k325tffg900-2** part for the KC705 platform. You can also select **Boards** and then select **Kintex-7 KC705 Evaluation Platform**. Click **Next**.
- 9. Review the New Project Summary page. Verify that the data appears as expected, per the steps above. Click **Finish**.
- 10. In the Sources window in Vivado IDE, expand sinegen_demo_inst to see the source files for this lab. Note that ila_0 core has been added to the project.



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

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



Step 2: Synthesize Implement and Generate Bitstream

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

✓ PROGRAM AND DEBUG

👫 <u>Generate Bitstream</u>

> Open Hardware Manager

0.20.	
Modified:	
Copied to	D:
Copied fr	om:
Copied o	n:
<	
General	Pro

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

Q <u>∓</u> ♦ ●	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	ints: 0	Number of Failing Endpoints:	0
Intra-Clock Paths	Total Number of Endpoints:	4437	Total Number of Endpoin	ts: 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)						

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





Lab 3

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

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

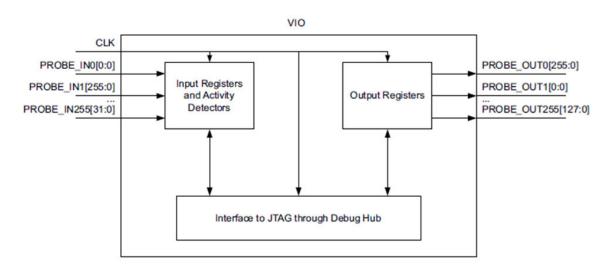


Figure 2: VIO Block Diagram

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

The following ports are created:

• One four-bit PROBE_INO port. This has two bits to monitor the two-bit Sine Wave selector outputs from the finite state machine (FSM) and other two bits to mimic the state of the other two LEDs on the board. You will configure these four-bit signals as LEDs during run time to mimic the LEDs displayed on the KC705 board.



• One two-bit PROBE_OUT0 port to drive the input buttons on the FSM. We will configure it so one bit can be used as a toggle switch during run time to mimic PUSH_BUTTON switch SW3, and the second bit will be used as PUSH_BUTTON switch SW6.

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

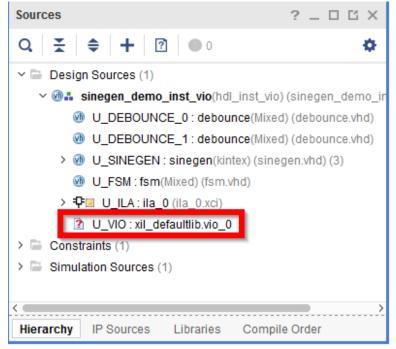
To create a project, use the New Project wizard to name the project, add RTL source files and constraints, and specify the target device.

- 1. Invoke Vivado IDE.
- 2. In the Quick Start tab, click Create Project to start the New Project wizard. Click Next.
- 3. In the Project Name page, name the new project **proj_hdl_vio** and provide the project location (C:/Vivado_Debug). Ensure that the **Create project subdirectory** is selected. Click **Next**.
- 4. In the Project Type page, specify the Type of Project to create as **RTL Project**. Click **Next**.
- 5. In the Add Sources page:
 - a. Set Target Language to VHDL.
 - b. Click Add Files.
 - c. In the Add Source Files dialog box, navigate to the /src/lab3 directory.
 - d. Select all VHD source files, and click OK.
 - e. Verify that the files are added, and **Copy Sources into Project** is selected.
- 6. Click the "+" sign, and then click **Add Directories**.
- 7. In the Add Source Directories dialog box, navigate to the /src/lab3 directory and choose the sine_high, sine_low, sine_mid, and ila_0 directories. Click Select.
- 8. Verify that the directories are added and Copy sources into project is selected. Click Next.
- 9. In the Add Constraints dialog box, click the "+" sign, and then click Add Files.
- 10. Navigate to the /src/lab3 directory and select sinegen_demo_kc705.xdc. Click Next.
- 11. In the Default Part page, specify the **xc7k325tffg900-2** platform. You can also select **Boards** and then select **Kintex-7 KC705 Evaluation Platform**. Click **Next**.
- 12. Review the New Project Summary page. Verify that the data appears as expected, in accordance with the previous steps. Click **Finish**.

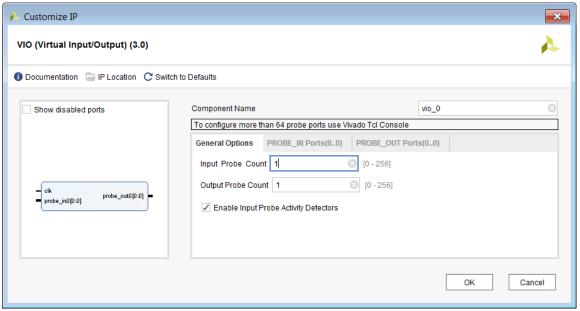
Note: It might take a moment for the project to initialize.



13. In the Sources window in Vivado IDE, expand sinegen_demo_inst_vio to see the source files for this lab. Note that the ila_0 core has been added to the project. However, vio_0 (the VIO core) is missing.



- 14. Instantiate and configure this VIO core as follows. From the Flow Navigator, click **IP Catalog**, expand **Debug & Verification**, then expand **Debug**, and double-click VIO. The Customize IP dialog box opens.
- 15. On the General Options tab, leave the Component Name as its default value of vio_0, set Input Probe Count to 1, Output Probe Count to 1, and select the **Enable Input Probe Activity Detectors** check box.





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

i Customize IP			X
VIO (Virtual Input/Output) (3.0)			4
() Documentation 📄 IP Location C	Switch to Defaults		
Show disabled ports	Component Name	vio_0	8
	To configure more than 64 probe ports u	se Vivado Tcl Console	
	General Options PROBE_IN Ports(0.	0) PROBE_OUT Ports(00)	
	Probe Port	Probe Width [1 - 256]	
	PROBE_IN0	4	0
- clk probe_in0[0:0] - probe_out0[0:0] -			
		OK	cel

17. On the PROBE_OUT Ports tab, set Probe Width to 2 and Initial Value to 0x0.

🍌 Customize IP				×					
VIO (Virtual Input/Output) (3.0)				A					
1 Documentation 📄 IP Location C Sv	vitch to Defaults								
Show disabled ports	Component Name		vio_0	8					
	To configure more than 64 probe ports use Vivado Tcl Console								
	General Options PROBE_	IN Ports(00) PROBE_OU	IT Ports(00)						
	Probe Port	Probe Width [1 - 256]	Initial Value (in hex)						
	PROBE_OUT0	2	💿 0x0	0					
olk probe_in0[3:0] probe_out0[1:0]									
			ОК	Cancel					

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



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

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

```
-- VIO

U_VIO : vio_0

port map

(

CLK => clk,

PROBE_IN0(3) => DONT_EAT,

PROBE_IN0(2) => GPIO_BUTTONS_re(1),

PROBE_IN0(1 downto 0) => sineSel,

PROBE_OUT0(1) => push_button_reset,

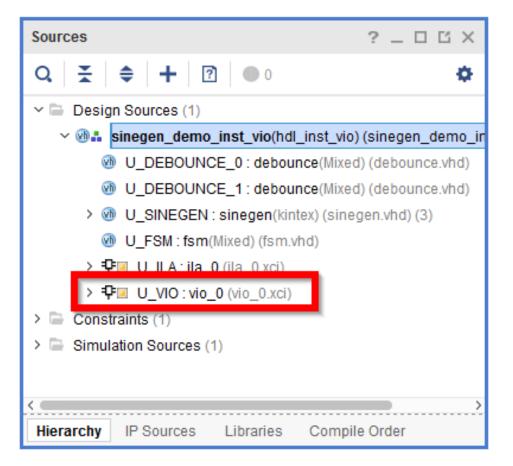
PROBE_OUT0(0) => push_button_vio

);
```

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







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

Step 2: Synthesize, Implement, and Generate the Bitstream

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





Tcl Consol	le Messages	Log	Reports	Design Runs	IP Status	Power	DRC	Methodology	Timing	×		? _ 🗆 🖸
Q	♦ ●		Desig	n Timing Summa	ry							
	eral Information er Settings		Setu	р			Hold				Pulse Width	
Desig	ign Timing Summa	ry	v	Vorst Negative Sla	ack (WNS):	0.539 ns	W	/orst Hold Slack (\	NHS):	0.044 ns	Worst Pulse Width Slack (WPWS):	1.732 ns
Clock	k Summary (4)		т	otal Negative Sla	ck (TNS):	0.000 ns	Т	otal Hold Slack (T	HS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
> 🗁 Chec	ck Timing (0)		N	lumber of Failing	Endpoints:	0	N	lumber of Failing I	Endpoints:	0	Number of Failing Endpoints:	0
> 🗁 Intra-	-Clock Paths		т	otal Number of E	ndpoints:	4703	Т	otal Number of Er	dpoints:	4703	Total Number of Endpoints:	2694
Inter-	-Clock Paths		All us	ser specified timi	na constraiı	nts are met						
> 🖹 Other	r Path Groups				-							
User	r Ignored Paths											
Unco	onstrained Paths											
Timing Su	ummary - impl_1 (s	saved)										

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





Lab 4

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

This simple tutorial shows how to do the following:

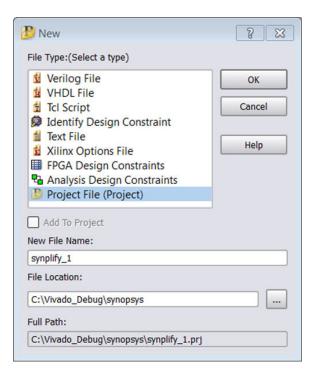
- Create a Synplify Pro project for the wave generator design.
- Mark nets for debug in the Synplify Pro constraints file as well as VHDL source files.
- Synthesize the Synplify Pro project to create an EDIF netlist.
- Create a Vivado[®] project based on the Synplify Pro netlist.
- Use the Vivado IDE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

Step 1: Create a Synplify Pro Project

- 1. Launch Synplify Pro and select **File** \rightarrow **New**.
- 2. Set File Type to Project File (Project) as highlighted in the following figure.
- 3. In the New File Name box, enter synplify_1.
- 4. Click OK.



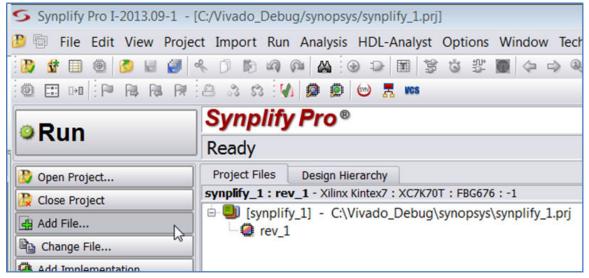




If you get a dialog box asking you to create a non-existing directory, click OK.

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

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





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

8. Click Add.

S Add Files to	Project	8
Look in:	📙 C:\Vivado_Debug\src\Lab4 🔹 🧿 🗿 📑 📰 🗏	
My Com	puter debounce.vhd fsm.vhd sinegen_demo.vhd	
File name: Files of type:	"debounce.vhd" "fsm.vhd" "sinegen_demo.vhd" HDL Files (*.vhd *.vhdl *.v *.sv *.vma)	
VHDL/Verilog lib		
-	roject: (3 file(s) selected) ♥ Use relative paths ♥ Add files to Folders Folder Options	
.\src\Lab4\de		<- Add All
.\src\Lab4\fsr .\src\Lab4\sir	n.vhd legen_demo.vhd	<- Add
		Remove All ->
		Remove ->
		ОК
		Cancel

9. In the same dialog box set Files of type to Constraints Files. This shows the synplify_1.sdc file. Select the file and click Add as shown in the following figure.





S Add Files to Project	×
Look in: C:\Vivado_Debug\src\Lab4 Computer Symplify_1.sdc	
File name: synplify_1.sdc Files of type: Constraint Files (*.sdc) VHDL/Verilog lib:	
Files to add to project: (4 file(s) selected) ✓ Use relative paths ✓ Add files to Folders	
.\src\Lab4\debounce.vhd .\src\Lab4\fsm.vhd .\src\Lab4\sinegen_demo.vhd .\src\Lab4\synplify_1.sdc	<- Add All <- Add Remove All -> Remove ->
	OK Cancel

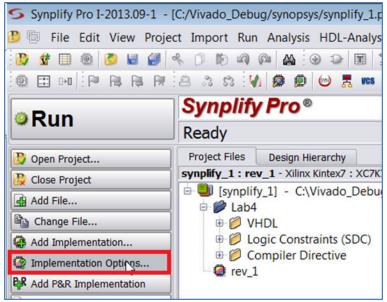
10. In the same dialog box, set Files of type to FPGA Constraint Files. This shows the synplify_1.fdc file. Select the file and click Add as shown in the following figure. Click OK.





S		Add Files to Project		×
Look in:	/proj/xcoswmktg/smitha/vivad	o_debug/lab4	- + + 1	
Comp	Name	Size Type	Date Modified	
🚞 smitha	synplify_1.fdc	468 bytes fdc File	5/6/16 9:18 AM	
File <u>n</u> ame:	synplify_1.fdc			
Files of type:	FPGA Constraint Files (*.fdc)			
VHDL/Verilog lit	b:			
Files to add to p	project: (1 file(s) selected) 🗹 Use re	elative paths 🛛 Add files	to Folders Folder Options]
./lab4/synplify_	1.fdc			<- Add All
				<- Add
				Remove All ->
				Remove ->
				ОК
				Cancel

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



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



Device Options Constra	ints Implementation Results	Timing Report H	ligh Reliability VHD	
Technology:	Part	Package:	Speed:	rev_1
Xilinx Kintex7	▼ XC7K325T	▼ FFG900	▼ -2	•
Device Mapping Options				
Option			Value	-
Fanout Guide			10000	
Disable I/O Insertion				e Josef al la companya de la companya
Disable Sequential Optim	nizations			
Update Compile Point Tir	ning Data			.
Click on an option for descrip	tion			
System Designer Board File				

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

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

15. You also can specify the MARK_DEBUG attributes in the source HDL files to mark the signals for debug, as shown in the code snippet from singen_demo.vhd file.

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

16. The synplify_1.sdc file contains various kinds of constraints such as pin location, I/O standard, and clock definition. The synplify_1.fdc file contains directives for the compiler. Here is where the nets of interest to us that are marked for debug are located. The attribute and the nets selected for debug are shown in the following figure.

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

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

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

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

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





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

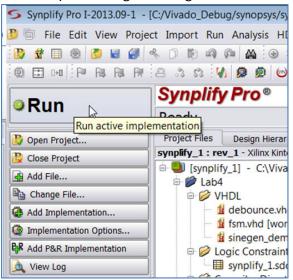
Step 2: Synthesize the Synplify Project

1. Before implementing the project, you need to set the name for the output netlist file. By default, the name of the output netlist file is synplify_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.



- 3. During synthesis, status messages appear in the Tcl Script tab. Warning messages are expected, but there should not be any Error messages. To see detailed messages, click the **Messages tab** in the bottom left-hand corner of the Synplify Pro console.
- 4. When synthesis completes, the output netlist is written to the file: rev_1/ sinegen_demo.edf

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

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



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

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

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

Step 4: Create a Post Synthesis Project in Vivado IDE

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



- 5. Under Add Netlist Sources, click Add Files, navigate to the Vivado_Debug/synopsys/ rev_1 folder, and select sinegen_demo.edf. Click OK.
- 6. Add the netlist file created in the previous section. Click Add Files again, navigate to the proj_synplify_netlist/proj_synplify_netlist.runs/synth1 folder and select sinegen.dcp.

Add the DCP files created for the sub-module IPs in the previous section. Click Add **Directories** again, navigate to the proj_synplify_netlist/

proj_synplify_netlist.srcs/sources_1/ip folder and select the following:

- sine_high
- sine_mid
- sine_low

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

- 7. Click Add Files, navigate to the Vivado_Debug/src folder, and select the sinegen_demo_kc705.xdc file. This file has the appropriate constraints needed for this Vivado project. Click OK in the Add Constraints File dialog box. In the Add Constraints (optional) dialog box ensure that Copy Constraints into Project is selected. Click Next.
- 8. Under Default Part, select **Boards** and then select **Kintex-7 KC705 Evaluation Platform** and the right version number for your hardware. Click **Next**.
- 9. Under New Project Summary, ensure that all the settings are correct and click **Finish**.

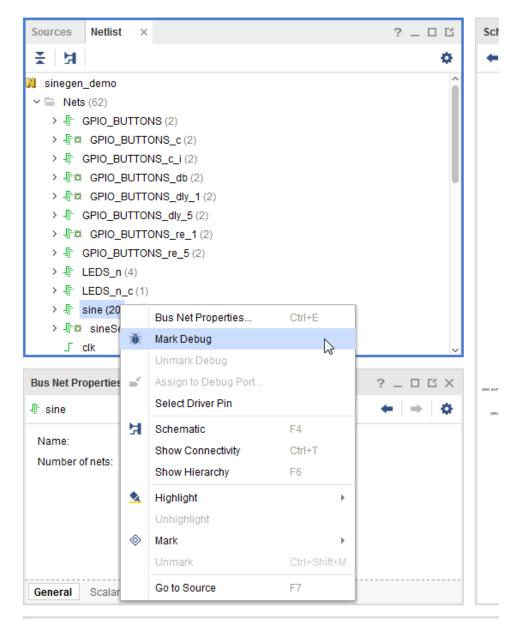
10. In the Sources window, ensure sinegen_demo.edf is selected as the top module.

Step 5: Add More Debug Nets to the Project

- 1. In Vivado[®] 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.





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 Rep	oorts Desi	gn Runs	Debug ×
> 小☆ GPIO_BUTTONS_c (2) IBUF 0 > 小☆ GPIO_BUTTONS_db (2) FDRE Q > 小☆ GPIO_BUTTONS_dly_1 (2) FDRE Q > 小☆ GPIO_BUTTONS_re_1 (2) FDRE Q > 小☆ sine (20) FDRE Q	Q 素 ♦ 兼 + ≓			
> 近並 GPIO_BUTTONS_c (2) IBUF O > 近並 GPIO_BUTTONS_db (2) FDRE Q > 近並 GPIO_BUTTONS_dly_1 (2) FDRE Q > 近並 GPIO_BUTTONS_re_1 (2) FDRE Q > 近並 sine (20) FDRE Q	Name	Driver Cell	Driver Pin	Probe Type
> 小☆ GPIO_BUTTONS_db (2) FDRE Q > 小☆ GPIO_BUTTONS_dly_1 (2) FDRE Q > 小☆ GPIO_BUTTONS_re_1 (2) FDRE Q > 小☆ sine (20) FDRE Q	 Unassigned Debug Nets (30) 			
> 小☆ GPIO_BUTTONS_dly_1 (2) FDRE Q > 小☆ GPIO_BUTTONS_re_1 (2) FDRE Q > 小☆ sine (20) FDRE Q	> Jrt GPIO_BUTTONS_c (2)	IBUF	0	
> 小☆ GPIO_BUTTONS_re_1 (2) FDRE Q > 小☆ sine (20) FDRE Q	> Jrt GPIO_BUTTONS_db (2)	FDRE	Q	
> 师章 sine (20) FDRE Q	> fra GPIO_BUTTONS_dly_1 (2)	FDRE	Q	
	> 近☎ GPIO_BUTTONS_re_1 (2)	FDRE	Q	
> √fr⊠ sineSel (2) FDRE Q	〉	FDRE	Q	
	> Jrt sineSel (2)	FDRE	Q	
	Debug Cores Debug Nets			

Running the Set Up Debug Wizard

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

Tcl Console Messages	Log Rep	oorts [Design	Runs	Debug	×
Q ¥ ♦ ¥ +						
Name Set Up	Debug	Driver C	ell C	Driver Pin	Probe	Гуре
👻 📄 Unassigned Debug Net	s (30)					
> 小☆ GPIO_BUTTONS_c	(2)	IBUF	C)		
> 小☆ GPIO_BUTTONS_d	b (2)	FDRE	G	2		
> 小☆ GPIO_BUTTONS_d	ly_1 (2)	FDRE	G	2		
> Jrt≋ GPIO_BUTTONS_re	e_1 (2)	FDRE	G	2		
> √fi≋ sine (20)		FDRE	G	2		
≻ √fr¤ sineSel (2)		FDRE	C	2		

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 the Vivado Logic Analyzer to Debug Hardware to complete the rest of this lab.





Lab 5

Using the Vivado Logic Analyzer to Debug Hardware

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

In this step, you learn:

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

Step 1: Verifying Operation of the Sine Wave Generator

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

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

Target Board and Server Set Up

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





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

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

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

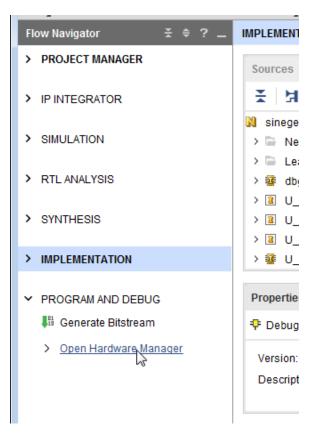
- **Connecting to the Target Board Locally:** If you plan to connect locally, ensure that the KC705 board is plugged into a Windows machine and then perform the following steps:
 - 1. Connect the Digilent USB JTAG cable of your KC705 board to a USB port on a Windows system.
 - 2. Ensure that the board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







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

HARDWARE MANAGER - unconnected						
🚯 No hardware target is open. Op	en targ	get				
Hardware	ø	Auto Connect				
		Recent Targets	•			
0, 素 ⊜ Ø ▶ 3		Available Targets on Server	→			
		Open New Target				
No conten	ıt					

- 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 remote hardware server, then configure the host name and port settings. Use Local get 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 Ca	ncel

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

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





🔥 Open New H	ardware Targe	et				×		
Select a hardwar	elect Hardware Target lect a hardware target from the list of available targets, then set the appropriate JTAG clock (TCK) quency. If you do not see the expected devices, decrease the frequency or select a different target.							
Hardware <u>T</u> arg	jets							
Туре	Name		JTAG CI	ock Frequency				
xilinx_tcf	Xilinx/Port_#000	3.Hub_#0004	6000000	×				
Hardware <u>D</u> evi Name	ces (for unknow	n devices, spe	nx Virtual C cify the Inst		er (IR) length)			
wame xc7k325t		IR Length 6						
	er: localhost:312	1	3ack	Next >	<u>F</u> inish	Cancel		

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	
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>Back</u> <u>N</u> ext > <u>Finish</u> Cano	el

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



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

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



Hardware	? _ 🗆	с×
$Q_{c} \mid \underbrace{\bigstar}_{c} \mid \diamondsuit \mid \underbrace{\varnothing}_{c} \mid \blacktriangleright \mid \underbrace{\ggg}_{c} \mid \underbrace{\blacksquare}_{c} \mid$		•
Name	Status	
V localhost (1)	Connected	
✓ ✓ ✓ ✓ xilinx_tcf/Xilinx/Port_#0003.Hu	Open	
✓ ⊕ xc7k325t_0 (1)	Not programmed	
🗿 XADC (System Monitor)		
_ (7 7		

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

Hardware			? _ 🗆	ц×	
Q ¥ ♦ ∅ ▶ ≫				•	
Name	S	tatus			
 Iocalhost (1) 	С	onnected			
✓ ✓ × xilinx_tcf/Xilinx/Port_#0003.	Hu O	pen			
 	Hardwa	are Device Pr	operties		Ctrl+l
c	Program Device Verify Device				6
		nfiguration M om Configura	-		
Hardware Device Devection	-	m BBR Key BR Key			
Hardware Device Properties	Program	m eFUSE Re	gisters		
	Export t	o Spreadshe	et		

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.



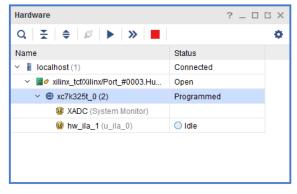


interview Program Device		×			
Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.					
Bitstre <u>a</u> m file:	C://ivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1/sinegen_demo.bit	8			
Debu <u>a</u> probes file:	C://ivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1/sinegen_demo.ltx	O			
✓ Enable end of st	artup check				
?	Program	Cancel			

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

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

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



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

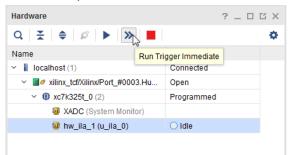




hw_	jia_1	? 🗆 🖒 X
	Waveform - hw_ila_1	? _ 🗆 ×
ous	Q + = & > > E B Q Q X • H H • F • H	ø
Dashboard Options	ILA Status:Idle	^
lboar	Name Value 0 10 20 30 40 50 6	I .
Dash	Impont_EAT > MicPio_BUTTONS_db[1:0] > MicPio_BUTTONS_db[1:0] > MicPio_BUTTONS_IBUF[1:0] > MicPio_BUTTONS_re(1:0) > MicPio_BUTTONS_re(1:0) > MicPio_BUTTONS_re(1:0) > MicPio_BUTTONS_re(1:0) > MicPio_BUTTONS_re(1:0) > MicPio_BUTTONS_re(1:0)	~ ~ ~
	Settings - hw_ila_1 Status - hw_ila_1 × ? _ D Trigger Setup - hw_ila_1 × Capture Setup - hw_ila_1	? _ 🗆
	Core status Idle Waiting for Trigger Full Capture status Window 1 of 1 Window sample 0 of 1024 Idle Idle Press the + button to add probes.	

Verifying Sine Wave Activity

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



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





Waveform - hw_ila_1							? _ 🗆
Q + − ช ► ≫	📕 📑 🔍	Θ. Χ	+ [] }	l 🗠 🖆 🕂 F	→ → →		4
ILA Status:Idle							1,023
Name	Value	0		200	400	600	800
UNT_EAT	0						
GPIO_BUTTONS_db[1:0]	0				0		
GPIO_BUTTONS_dly[1:0]	0				0		
SPIO_BUTTONS_IBUF[1:0]	0				0		
SPIO_BUTTONS_re[1:0]	0				0		
V_SINEGEN/sel[1:0]	0				0		
V_SINEGEN/sine[19:0]	05133						
		Updated	at: 2017-Ma	r-16 14:59:13			

Displaying the Sine Wave

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

Naveform - hw_ila_1		? _ 🗆 ×
Q + − ϑ ► ≫	🔁 🔍 🤆	2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
ILA Status:Idle		1,023
Name	Value	
🖫 DONT_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
V_SINEGEN/sel[1:0]	0	0
U_SINEGEN/sine[19:0]	05133	
		Updated at: 2017-Mar-16 14:59:13

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

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

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



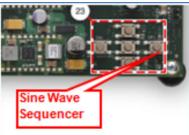


Q + - & > > ILA Status: Idle	🕒 🔍 Q	- 50 14 - 5				
ILA Status:Idle			vl 1≛ ≛r +[[🖌		
						1,02
Name	Value	°	200	400	600	800
H DONT_EAT	0					
SPIO_BUTTONS_db[1:0]	0	(0		
GPIO_BUTTONS_dly[1:0]	0			0		
SPIO_BUTTONS_IBUF[1:0]	0			0		
GPIO_BUTTONS_re[1:0]	0			0		
U_SINEGEN/sel[1:0]	0			0		
■ U_SINEGEN/sine[19:0]	20787	~~~~				
		Updated at: 2017-M	ar-16 14:59:13			

Correcting Display of the Sine Wave

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

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



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

Waveform - hw_ila_1						? _ 🗆 X
Q + − ϑ ▶ №	🕒 🔁 🤤	a ⊠ • ⊺ I4 ▶	l 12 27 4F F	⊨ ⇒ " ⊡		0
ILA Status:Idle						1,023 ^
Name	Value	°	200	400	600	800
He DONT_EAT	0					
> 🔣 GPIO_BUTTONS_db[1:0]	0			0		
Reference Stress Str	0			0		
> 🔣 GPIO_BUTTONS_IBUF[1:0]	0	0		0		
	0			0		
> Note: Singer Strategy > Note: Singer Strategy > Note: Singer Strategy > Singer	1			1		
> Pi U_SINEGEN/sine[19:0]	-83150					
	< >	Updated at: 2017-Ma	r-16 15:02:38			

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





Waveform - hw_ila_1						? _ 🗆 ×
Q + − ♂ ▶ ≫	🔁 🔁 🔍	Q 22 📲 I4 🕨	l 12 27 40 R	• • •		٥
ILA Status: Idle						1,023 ^
Name	Value	•	200	400	600	800 1 ₂ 0
	0					
> M GPIO_BUTTONS_db[1:0] > M GPIO_BUTTONS_dly[1:0]	0 0			0		
> 💐 GPIO_BUTTONS_IBUF[1:0]	0			0		
> M GPIO_BUTTONS_re[1:0] > M U_SINEGEN/sel[1:0]	0 2			0		
> = U_SINEGEN/sine[19:0]	-377487					
	011101					
		Updated at: 2017-Ma	ar-16 15:03:22			
	< >		10 10:00:00			>
Waveform - hw_ila_1						? _ 🗆 X
Q + - & > >	📕 🕞 🔍 (Q 22 - F 14)	l de ler de le	• •F [•F		0
ILA Status:Idle						1,023 ^
Name	Value	0	200	400	600	800 1,c
UDONT_EAT	0					
> M GPIO_BUTTONS_db[1:0] > M GPIO_BUTTONS_dly[1:0]	0 0	0		0		
> 📢 GPIO_BUTTONS_IBUF[1:0]	0			0		
> M GPIO_BUTTONS_re[1:0] > M U_SINEGEN/sel[1:0]	0 3	}		0		
				Ť		
> =% U_SINEGEN/sine[19:0]	75777					
- O_SINEGEN/SINE[19.0]	15/11					
		Updated at: 2017-M				
	<	Updated at: 2017-M > < ⊂	ar 16 15 US: 55			→ →

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.

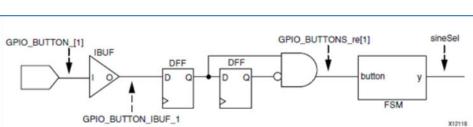


Figure 3: Sine Wave Sequence Button Schematic

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_IBUF_1 signal, but it occurs intermittently. Because it could take several button presses to detect it, you will now set up the Vivado logic analyzer tool to Repetitive Trigger Run Mode. This setting makes it easier to repeat the button presses and look for the event in the Waveform viewer.

- 1. Under the Settings tab for hw_ila_1, configure the following:
 - Trigger Mode to BASIC_ONLY
 - Capture Mode to BASIC
 - Window Data Depth to 1024



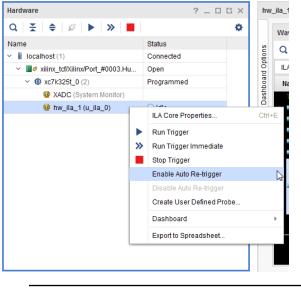
- Trigger position to 512
- Press the + button in the Trigger Setup window and add probe GPIO_BUTTONS_IBUF_1. Change the Value field to RX by selecting the value RX in the Value field, as shown in the following figure.

Waveform - hw_ila_1									?	_
Q + - & > =	D≩ ⊕, ⊖,	X - ¶ I	< ⊢ ±	±r +Γ						•
ILA Status: Idle										1,023
Name	Value	0	200		400	600		800		_ <u> </u> 1,
[™] DONT_EAT () > [™] GPIO_BUTTONS_db[1:0] ()					0					
Sector 2017 Sec		\geq			0					
Section Strategy Section Strategy Section Sect					0					
> M GPIO_BUTTONS_re[1:0] (> M U_SINEGEN/sel[1:0] ()		<u> </u>			0					
> Tu_SINEGEN/sine[19:0]	183094									
•										
					1	I		I		
<		Updated at: 2	017-Mar-16 1	5:08:22					0	
			0	T : 0 (~ -
Settings - hw_ila_1 × Status - hw_ila_	-'		? _ □	Trigger Setu		Capture Set	up - IIW_lla_			? _ [
Trigger Mode Settings				Q +	- V_	0	Death	Matura		
Trigger mode: BASIC_ONLY	\mathbf{v}			Name GPIO_BUTT	DNS_IBUF[1:0]	Operator	Radix [B]	Value • XX	Port Port	e3[1:0]
				_						
Capture Mode Settings										
Capture mode: ALWAYS	÷ •		- 11							
Number of windows: 1	[1 - 1024]									
Window data depth: 1024	✔ [1 - 1024]									
Trigger position in window: 512	[0 - 1023]		- 11							
	[0 1020]		- 11							
General Settings			- 11							
Refresh rate: 500 ms			~	<						
						_				
Trigger Setup - hw_ila_1 \times	Capture Setu	up - hw_ila_1			? _ □	1				
Q + - D,										
Name	Operator	Radix	Value		Port					
GPIO_BUTTONS_IBUF[1:0]	== •	[B] •	XX	~	probe3[1:0]					
		Value:								
		OK		ancel						
<						>				

CAUTION! For different labs the GPIO_BUTTONS_IBUF might show up differently or have a different name such as button_in4_in. This might also show up as two individual bits or two bits lumped together in a bus. Ensure that you are using bit 1 of this bus to set up your trigger condition. For example in case of a two-bit bus, you will set the Value field in the Compare Value dialog box to RX.



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

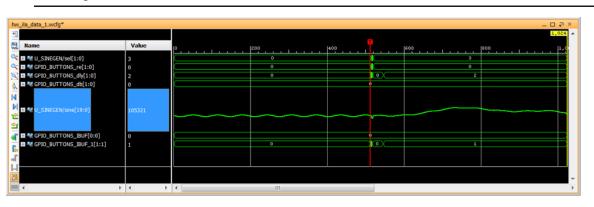


CAUTION! The ILA properties window may look slightly different for different labs.

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

- Arms the trigger.
- Waits for the trigger.
- Uploads and displays waveforms.
- 3. On the KC705 board, press the Sine Wave Sequencer button until you see multiple transitions on the GPIO_BUTTONS_IBUF_1 signal (this could take 10 or more tries). This is a visualization of the glitch that occurs on the input. An example of the glitch is shown in the following two figures.

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





hw_ila_data_1.wcfg*																_ 0	1
Name		Value		500	505	510		515	5.			520	525		530		535
TH MU_SINEGEN/sel[1:0]		3			0				1	2	х_			3			
🟹 🔢 🖬 GPIO_BUTTONS_re[1:0])			0					0 2	Х			0			
🕻 🖽 🖬 GPIO_BUTTONS_dly[1:0]		2			0		20	2		2)			0	1			
GPIO_BUTTONS_db[1:0])							(0							
4																	
U_SINEGEN/sine[19:0]		105321															
순 건								\uparrow	_		_						
E M GPIO_BUTTONS_IBUF[0:0])							(
GPIO_BUTTONS_IBUF_1[1:1]		L					101	0	\sim				0				
	Þ.	< >	٠							111							

Fixing the Signal Glitch and Verifying the Correct State Machine Behavior

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

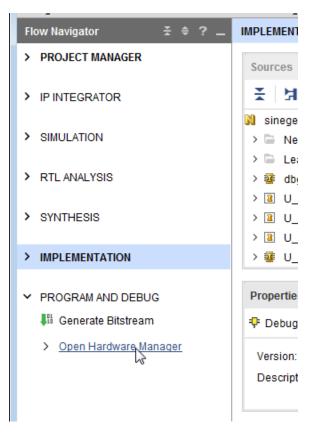
- 1. Enable the de-bouncer circuit by setting DIP switch position on the KC705 board (labeled De-bounce Enable in Figure 1: KC705 Board Showing Key Components) to the ON or UP position.
- 2. Enable the Auto-Retrigger mode on the ILA debug core and click RunTrigger on the ILA core, and
 - Ensure that 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**.







The Hardware Manager window opens.

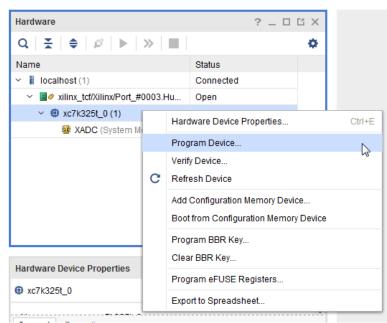
2. Click Open a new hardware target.

HARDWARE MANAGER - unconnected										
1 No hardware target is open. Open target										
Hardware	ø	Auto Connect								
0, ≚ ≑ Ø ▶ >		Recent Targets								
		Available Targets on Server	- F							
		Open New Target								
			43							
No conten	t									

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



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



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

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

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



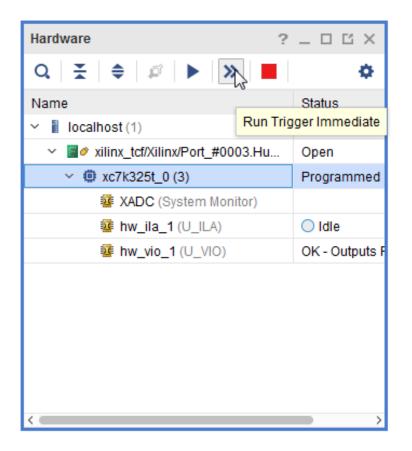
Hardware	? _ 🗆	c x
$Q_{1}\mid \underbrace{\bigstar}_{1}\mid \diamondsuit \mid \varnothing \mid \blacktriangleright \mid \gg \mid \blacksquare \mid$		•
Name	Status	
 Iocalhost (1) 	Connected	
✓ Ø xilinx_tcf/Xilinx/Port_#0003.Hu	Open	
xc7k325t_0 (3)	Programmed	
🔯 XADC (System Monitor)		
🥶 hw_ila_1 (U_ILA)	Oldle	
🥸 hw_vio_1 (U_VIO)	OK - Outputs Reset	

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

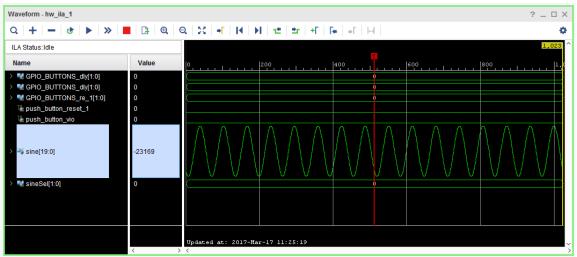
hw_ila_1							? 🗆 🗆 ×
Waveform - hw_ila_1							? _ 🗆 ×
Second 4 = 10 ≤ x	- 📕 🕒 🔍 e	a 😫 📲 H H 🖻	tr +F Te	•[[٥
Q + - Image: Constraint of the second secon							^
Name	Value	0	20	30	40	50 60	70
<pre>> M GPIO_BUTTONS_d()10) > M GPIO_BUTTONS_re_1(1:0) % push_button_resset_1 % push_button_vio > M sine(19:0) > M sine(19:0) > M sineSe((1:0)</pre>	< >>	<					ž
	/_ila_1 ×	? _			Capture Setup - hw_ila_1		? _ 🗆
Capture status	Trigger Walting for T w sample 0 of 1024 Idle			+ - Þ,	Press the 🕂 bu	tton to add probes.	

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



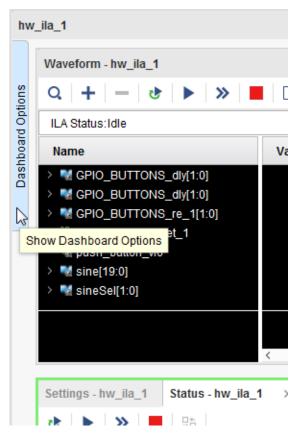


- 12. Make sure that there is activity on the sine [19:0] signal.
- 13. Select the sine signal in the Waveform window, right-click and select Waveform Style \rightarrow Analog.
- 14. Select the sine signal in the Waveform window again, right-click and select **Radix** → **Signed Decimal**. You should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





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



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



ashboard Options	Waveform - hw_ila_1	
2	Q + - & > > = B @ Q % + H H H ± ± +	Fe of D
xc7k325t_0	ILA Status: Idle	
✓ ✓ hw_ila_1 (U_ILA)	Name Value 0 200	
Status Settings	> M GPIO_BUTTONS_dly(1:0) 0	
 Settings Trigger Setup 	> M GPIO_BUTTONS_dly[1:0] 0	
Capture Setup	> M GPI0_BUTTONS_re_1(1:0) 0 (
-Z Waveform	We push_button_reset_1 0 We push_button_vio 0	
hw_vio_1 (U_VIO)		ΛΛΛΛ
XADC (System Monitor)		
	> 🍕 sine[19:0] -23169	
	> 📢 sineSel[1:0] 0	
	Updated at: 2017-Mar-17 11:25:19	
	$\langle \rangle \rangle \langle \rangle$	
	Settings - hw_ila_1 Status - hw_ila_1 × ? _ D Trigger Setup - hw_ila_1 hw_vio_1	× Capture Setup - hw_ila_1
	ở ▶ ≫ ■ 95	
	Core status	
	Idle Pre-Trigger Waiting for Trig	
	Capture status	Press the 🕂 button to add probe
	Capture status	
	Window 1 of 1 Window sample 0 of 1024 Tota	

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

18. Adjust the Trigger Setup – hw_ila_1 window and the hw_vio_1 window so that they are side by side as shown in the following figure.





Waveform - hw_ila_1							? _ 🗆 >		
Q + − & ▶ ≫	- 🔁 🔍	Q X 📲 H) 1± ±r +Γ Γ	• +F ⊨	1		0		
ILA Status: Idle							1,023		
Name	Value	0,,,,,,,,,,,,	200	400		600	800 1 ₋ 0		
> 📲 GPIO_BUTTONS_dly[1:0]	0				0				
> 號 GPIO_BUTTONS_dly[1:0] > 駴 GPIO_BUTTONS_re_1[1:0]	0				0				
W push_button_reset_1	0				ľ –				
🖟 push_button_vio	0								
		$ \land \land \land \land$	$ \land \land \land \rangle$	$ \Lambda \Lambda $	\land	ΛΛΛ.	ΛΛΛΛ		
> 🍣 sine[19:0]	-23169			$ \setminus \setminus$					
				(\setminus)	$ \setminus $				
> 🐭 sineSel[1:0]	0		ψ V V V		<u>v v</u>	VVV			
	0				ř				
		Updated at: 2017-M	or-17 11-25-19						
	<	> <							
Settings - hw_ila_1 Status - hw_ila	a 1 Trigger S	etup-hw × ? _ □	hw_vio_1 × Captu	ure Setup - hw	ila 1		? _ [
Q + - D				Q ★ ♦ + =					
	Press the 🕂 button to add probes.								
Press the 🕂 button to add probes.									
Flessule T									

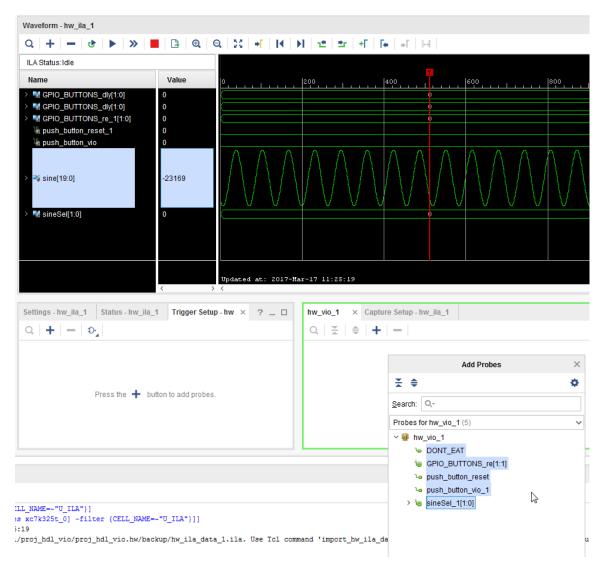
19. In the hw_vio_1 window, select the "+" button, and select all the probes under hw_vio_1.

20. Click **OK**.

Note: The initial values of all the probes.







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





Waveform - hw_ila_1								? _ 🗆 X
Q + - ♂ ▶ ≫	B Q	Q 23 - I4 D	H H H	[• →[ы			0
ILA Status:Idle								1,023
Name	Value		200	400		600	80	20 11 ₄ 0
> 💐 GPIO_BUTTONS_dly[1:0]	0							
> 📲 GPIO_BUTTONS_dly[1:0] > 📲 GPIO_BUTTONS_re_1[1:0]	0 0	<u></u>			1			
GFIO_BOTTONS_re_i[1.0] We push_button_reset_1	0	<u> </u>						
We push_button_vio	0							
> ₱% sine[19:0]	-23169				\mathbb{N}	\bigwedge	\mathbb{N}	
> 📲 sineSel[1:0]	0	(•			
	<	Updated at: 2017-Ma	ar-17 11:25:19					
Settings - hw_ila_1 Status - hw_ila	_1 Trigger Se	tup-hw × ? _ 🗆	hw_vio_1 × Ca	apture Setup -	hw_ila_1			? _ 🗆
Q + = D ₄			Q <u>∓</u> ♦	+ -				
			Name		Value	Acti Directi	VIO	
			► DONT_EAT		(B) 0	Input	hw_vio_1	
			B GPIO_BUTTO			Input	hw_vio_1	
Press the 🕂 t	outton to add probe	es.	Ve push_button		(B) 0 -	Output	hw_vio_1	
			> te push_button_		[B]0 ▼ [H]0	Output Input	hw_vio_1 hw_vio_1	
				-1	r. 1 e	input		

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 Setur	o - 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	
∿⊲ push_button_res€*		D	0.1.1	hw_vio_1	
د 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 Bu	itton			
	Toggle Buttor	n	R		
	Radix		15		? _ 🗆
	Rename				
	Name		+		
	Remove		Delete		
and 'import_hw_ila_da	Export to Spre	adsheet		Data menu :	item to impo

The Value field for push_button_reset is highlighted.

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

hw_vio_1 × Capture Setup	- hw_ila_1				? _ 🗆
Q 素 ≑ + -					
Name	Value	Activity	Direction	VIO	
Lean Pont_Eat	[B] 0		Input	hw_vio_1	
BUTTONS_re[1:1]	[B] 0		Input	hw_vio_1	
ush_button_reset	1		Output	hw_vio_1	
د∎ push_button_vio_1	0		Output	hw_vio_1	
> 🐌 sineSel_1[1:0]	[H] 0		Input	hw_vio_1	

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



					_
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	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	
	Debug Pr	robe Propertie	es Ctrl+E		
•	Text				_
	LED				
	Radix	Radix •			
	Activity Pe	ersistence	•		
n_AIO}]]	Rename.				
5_110 111	Name		•		
	Remove		Delete	ç.	
	Export to	Spreadsheet			

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

Select LED Colors						
Low Value Color:	🔘 Gray 🗸 🗸					
High Value Color:	\varTheta Red 🗸					
ОК	Cancel					

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



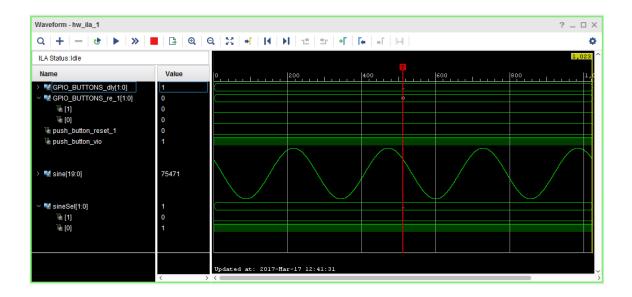
hw_vio_1 × Capture Setup	- hw_ila_1				? _ 🗆
Q ¥ € + -					
Name	Value	Activity	Direction	VIO	
ኈ DONT_EAT	[B] 0		Input	hw_vio_1	
BOPIO_BUTTONS_re[1:1]	[B] 0		Input	hw_vio_1	
ush_button_reset	1		Output	hw_vio_1	
∿ push_button_vio_1	0		Output	hw_vio_1	
> 墙 sineSel_1[1:0]	[H] 0		Input	hw_vio_1	

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

hw_vio_1 × Capture Setup	- hw_ila_1				?.
Q ¥ ♦ + -					
Name	Value	Activity	Direction	VIO	
∿ DONT_EAT	[B] 0		Input	hw_vio_1	
BCPIO_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	
	٢		Input	hw_vio_1	
	•		Input	hw_vio_1	
∿ push_button_vio_1	1		Output	hw_vio_1	

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









Lab 6

Using the ECO Flow to Replace Debug Probes Post Implementation

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

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

1. Open the Vivado[®] Design Suite, and select File \rightarrow Open Checkpoint.



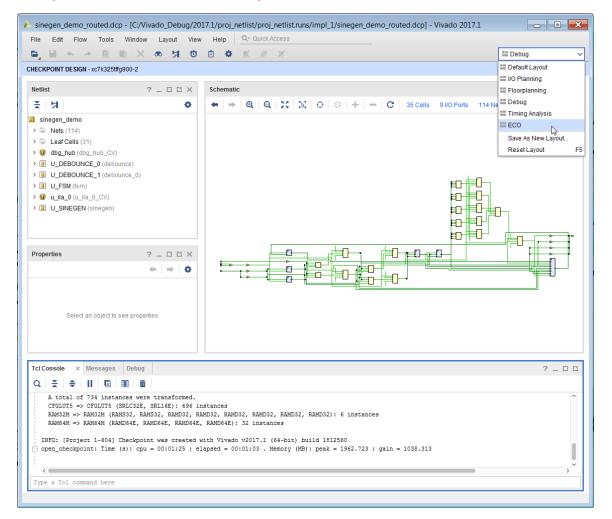
2. Open the routed checkpoint that you created in Lab 2: Using the HDL Instantiation Method to Debug a Design.





🏄 Open 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
▶ sinegen_demo_routed.dcp	File: sinegen_demo_routed.dcp Directory: C:/Vivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1 Created: 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

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







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

	[C:/Vivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1, inggen_demo_routed.dcp] - Vivado 2017.1
	X ∞ 3 1 0 0 0 0 0 X 2 X 2 X 2 X 2 X 2 X 2 X 2 X
HECKPOINT DE SIGN - xc7k325tffg900	
ECO Navigator	Scratch F Propertie Netli × ? _ □ □ Schematic × Device × Package × ? □ □
Edit	★ ★ ★ Q X X Q + - C 35 Cells 9 I/O Ports
Create Net	N sinegen_demo
Create Cell	> Sets (114) > Set Cells (31)
Create Port	> 10 dbg_hub (dbg_hub_CV)
Create Pin	> ■ U_DEBOUNCE_0 (debounce)
Connect Net	➤ I U_DEBOUNCE_1 (debounce_0) > I U_FSM (fsm)
Disconnect Net	> 顰 u_iia_0 (u_iia_0_CV)
Replace Debug Probes	➤ I U_SINEGEN (sinegen)
Place Cell	
Unplace Cell	
Run	
Check ECO	
Optimize Logical Design	
Place Design	
Optimize Physical Design	
Route Design	
Report	
Edit Timing Constraints	
Timing Summary	
Report Clock Networks	
Report Clock Interaction	
Report DRC	
Report Utilization	TclConsole × Messages Package Pins 1/0 Ports ? _ 🗆 1
Seport Power	Q X ♦ II Im Im INFO: (Project 1-111) Unisim Transformation Summary:
	A total of 734 instances were transformed.
Program	CFGLUT5 => CFGLUT5 (SRLC32E, SRL16E): 696 instances RAM32M => RAM32M (RAMS32, RAMS32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32): 6 instances
Save Checkpoint As	RAM64M => RAM64M (RAMD64E, RAMD64E, RAMD64E, RAMD64E): 32 instances
Senerate Bitstream	INFO: [Project 1-604] Checkpoint was created with Vivado v2017.1 (64-bit) build 1812560
🕒 Write Debug Probes	open_incorporation filme (a): opu = outoriza ; etabaed = outorioa : memory (no): peak = 1962.723 ; gain = 1036.313
Open Hardware Manager	Type a Icl command here

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





🔥 Replace Debug Probes				
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).				
王 ≑ ∥ ⊂ ↑ ↓ 				
Search: Q-				
Name	Probe			
• Ch 13	_	0 ^		
Oh 14	_ SINEGEN/sine[14]	0		
Oh 15	_ I € U_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		
Oh 1	_ SPIO_BUTTONS_IBUF[1]	0		
✓ ■ probe3 (2)		- 11		
Ch 0	_ SPIO_BUTTONS_db[0]	0		
🖲 Ch 1	_ SPIO_BUTTONS_db[1]	0		
✓				
🖲 Ch 0	_ SPIO_BUTTONS_dly[0]	0		
🖲 Ch 1	_ SPIO_BUTTONS_dly[1]	0		
probe5 (2)				
🖲 Ch 0	_ SPIO_BUTTONS_re[0]	0 ~		
Probes changed: 0				
	ОК Са	ancel		

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 Choose Nets dialog box.



🔥 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			
Ch 13	「≆ U_SINEGEN/sine[13]			
Ch 14	「 ¥ U_SINEGEN/sine[14]			
• Ch 15	_「 ¥ U_SINEGEN/sine[15]			
Och 16	_「 ¥ U_SINEGEN/sine[16]			
Och 17	「 ¥ U_SINEGEN/sine[17] ⊘			
Ch 18	「 ¥ U_SINEGEN/sine[18] ⊘			
Ch 19	「 ¥ U_SINEGEN/sine[19]			
✓ ■ probe2 (2)				
🖲 Ch 0	_「 * GPIO_BUTTONS_IBUF[0]			
🖲 Ch 1	_ SPIO_BUTTONS_IBUF[1] ⊘			
✓ ■ probe3 (2)				
Ch 0	_ SPIO_BUTTONS_db[0]			
Ch 1	_ SPIO_BUTTONS_db[1]			
✓				
• Ch 0	SPIO_BUTTONS_dly[0]			
• Ch 1	_ GPIO_BUTTONS_dly[1] →			
✓ m probe5 (2)				
Ch 0	「 ¥ GPIO_BUTTONS_re[0]			
Probes changed: 0				
	OK Cancel			

6. In the Choose Nets dialog box, choose the U_DEBOUNCE_0/clear net to replace the existing GPIO_BUTTONS_dly[0] probe net. Click **OK**.





Choose Nets				×
Choose nets to replace existing probe	S.			4
Properties				
NAME ~	contains 🗸 🗸	*	⊗ +	
<u>R</u> egular expression	nierarchically 🗸 Displa	av unia	ue nets	
Of objects:	,			
	l	Ein	d	
Found: 12857		- 0	Selected: 0 of 1	Ļ
_ <const0></const0>	î			
∫ <const1></const1>				
∫ clk				
∫ clk_ibufgds				×
J CLK_N J CLK_P		÷		Ŧ
J CLK_P J dbg_hub/ <const0></const0>		≓	Use the buttons on the left to copy Nets into this List.	+
∫ dbg_hub/ <const0></const0>		-		+
∫ dbg_hub/inst/BSCANID.u_xsdbm	id/ <const1></const1>			÷
∫ dbg_hub/inst/BSCANID.u_xsdbm				
∫ dbg_hub/inst/BSCANID.u_xsdbm				
			OK Cano	
			Unit Call	~

 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				
NAME V cont	ains 🗸	*cle	ar 🛛 🖌	
		_		
	rchically 🗾 Displa	av unio	lie nets	
Of objects:	ionioany <u>e b</u> iopia	i) ang		
	г			
		<u>F</u> ind	1	
Found: 68		\$	Selected: 0 of 1	Z↓
J 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			×
了 u_ila_0/inst/ila_core_inst/u_ila_regs/	_	+		Ŧ
∫ u_ila_0/inst/ila_core_inst/u_ila_regs/	UNI.CIVI_SKL		Use the buttons on the left to copy Nets into this List.	÷
∫ u_ila_0/inst/ila_core_inst/u_ila_regs/		≠		+
「u_ila_0/inst/ila_core_inst/u_ila_regs/ 「u_ila_0/inst/ila_core_inst/u_ila_regs/				+
」 u_ila_0/inst/ila_core_inst/u_ila_regs/				-
∫ u_ila_0/inst/ila_core_inst/u_ila_regs/				
<	>			
			ОК Саг	ncel



n Choose Nets	×
Choose nets to replace existing probes.	4
Properties	
NAME v contains v *clear	⊗ +
Regular expression Search hierarchically Display unique nets Of objects:	 2∗ X↓
∫ dbg_hub/inst/BSCANID.u_xsdbm_id/CORE_XSDB.U ∫ 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[2].mu ∫ u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu	2. NV
ок	Cancel

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



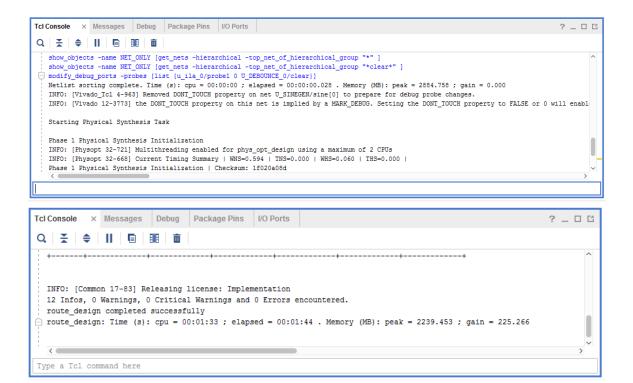


Replace Debug Probes		×
	one or more debug probes. To reflect the ger, regenerate the debug probes file (L1	
Search: Q-		
Name	Probe	
Och 13	_ I U_SINEGEN/sine[13]	0 ^
Ch 14	_ I U_SINEGEN/sine[14]	0
Ch 15	_ I ¥ U_SINEGEN/sine[15]	0
Oh 16	_ SINEGEN/sine[16]	0
Och 17	_ SINEGEN/sine[17]	0
Oh 18	_ I U_SINEGEN/sine[18]	0
Ch 19	_ I	0
probe2 (2)		
Ch 0	_ SPIO_BUTTONS_IBUF[0]	0
Och 1	_ SPIO_BUTTONS_IBUF[1]	0
✓ ■ probe3 (2)		- 11
Ch 0	_ SPIO_BUTTONS_db[0]	0
Och 1	_ SPIO_BUTTONS_db[1]	0
probe4 (2)		
Ch 0	_ U_DEBOUNCE_0/clear	C
Och 1	_ SPIO_BUTTONS_dly[1]	0
✓		_
• Ch 0	_	0 ~
Probes changed: 1		
	ОК С	ancel

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







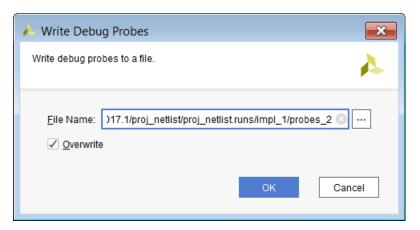
9. Save your modifications to a new checkpoint. Use the Save Checkpoint As option in the ECO Navigator to bring up the Save Checkpoint As dialog box. Specify a file name for the .dcp file and click **OK**.

Save Checkpoint 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 S	 cel

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







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

🔶 Generat	Generate Bitstream				
Create a pro	Create a programming file from the current design				
Bit File	Bit File etlist/proj_netlist.runs/impl_1/project_sinegen_demo_routed. 💿				
Options					
	w_bitfile		î		
	ask_file _binary_bitfile				
	n_file				
	adback_file gic_location_file				
	rbose		~		
Select an option above to see a description of it					
		ОК Са	ncel		

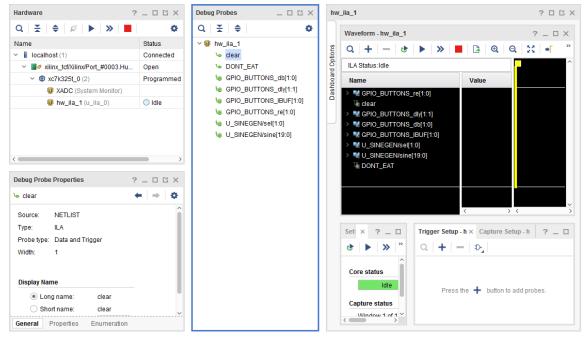
- 12. Connect to the Vivado Hardware Manager by selecting Open Hardware Manager in the ECO Navigator.
- 13. Connect to the local hardware server by following the steps in the Target Board and Server Set Up section in Lab 5: Using the Vivado Logic Analyzer to Debug Hardware.

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



🔥 Program Device	
	gramming file and download it to your hardware device. You can optionally file that corresponds to the debug cores contained in the bitstream
Bitstre <u>a</u> m file: Debu <u>q</u> probes file: ☑ <u>E</u> nable end of s	
?	Program Cancel

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.





Waveform - hw_ila_1		? _ D ×
Q + − ϑ ▶ ≫	G	Q X ➡ H ▶ ± ± + [+ → H *
ILA Status:Idle		
Name	Value	
> 🔣 GPIO_BUTTONS_re[1:0]	0	()
🖫 clear	1	
> 🔣 GPIO_BUTTONS_dly[1:1]	0	()
> 🔣 GPIO_BUTTONS_db[1:0]	0	()
> 💐 GPIO_BUTTONS_IBUF[1:0]	0	()
V_SINEGEN/sel[1:0]	0	()
V_SINEGEN/sine[19:0]	05a81	
UNT_EAT	0	
		Updated at: 2017-Mar-17 14:43:26





Lab 7

Debugging Designs Using the Incremental Compile Flow

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:

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

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

1. Start the Vivado IDE.

Load the Vivado IDE by doing one of the following:

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





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

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

Click Next.

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

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





SYNTHE SIZED DE SIG	N - constrs_	2 xc7k70tfbg676-2 (activ	re)
Sources Netlist	t ×		? _ 🗆 🖸
표 처			•
>-厅 dcpu	u_adr_cpu (32)	^
> 🖟 dcpu	u_dat_cpu (3	32)	1
> √ [r dcpu	u_dat_qmer	m (25)	
> √ [⁺ dcpu	u_sel_cpu (3	3)	
> -∰r dcpu	u_tag_dmm	u (1)	
> -∬r dcqr	mem_adr_q	mem (32)	
> 🕂 dcqr	mem_dat_q	mem (32)	
> √r dcqr	mem_sel_a		
> 🕂 dcqr		Bus Net Properties	Ctrl+E
> 🕂 dcqr	mem_ 🕷	Mark Debug	
>√∱ dcst		Unmark Debug	- 0
> √ি dcsl	b_sel 💕	Assign to Debug Port	
> 🕂 dout	(32)	Select Driver Pin	
> √r dtlb_		Schematic	F4
>_∱ du_	-1		
>_∱ du_		Show Connectivity	Ctrl+T
	dsr (1	Show Hierarchy	F6
>_∱du		Highlight	۱.
> -厅 dvr0		Unhighlight	
	dat	Mark	•
>√f dwb	_dat_	Unmark	Ctrl+Shift+M
> f E(1			
>√[r ex_i		Go to Source	F7
≻ -厅 fifo_	dat_0 (3)		~

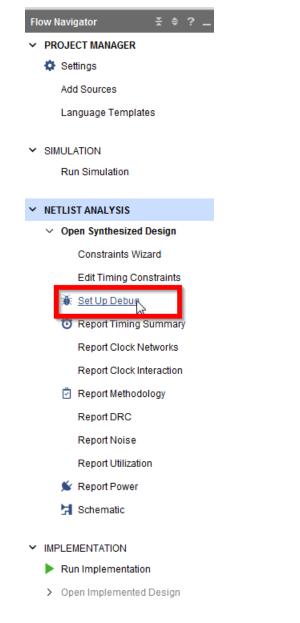
Alternatively, you can use the following Tcl command to set the MARK_DEBUG property on the signals specified.

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

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







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

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



Q 素 ≑ ㎡ Ⅲ + −					¢
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	~	
> 📲 cpuEngine/dcqmem_dat_qmem (32)	clkgen/cpuClk	FDRE	Data and Trigger	~	
> 📲 🛱 cpuEngine/dcqmem_sel_qmem (4)	clkgen/cpuClk	FDRE	Data and Trigger	~	
> √r¤ cpuEngine/du_dsr (11)	clkgen/cpuClk	FDCE	Data and Trigger	Υ.	
> √f≅ cpuEngine/dvr0_0 (6)	clkgen/cpuClk	FDCE	Data and Trigger	~	
Find Nets to Add					Nets to debug: 1

- 11. When the ILA Core Options screen appears, click Next again.
- 12. When the Set Up Debug Summary screen appears, ensure that one debug core is created and click **Finish**.
- 13. Check the Debug widow to ensure that the u_ila_0 core has been inserted into the design.

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

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

Step 2: Compiling the Reference Design

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





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

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

- 4. In the Design Runs window, right-click impl_1 and select Open Run Directory from the popup menu. This opens the run directory in a file browser as seen in the following figure. The run directory contains the routed checkpoint (top_routed.dcp) to be used later for the incremental compile flow. The location of the implementation run directory is a property of the run.
- 5. Get the location of the current run directory in the Tcl Console by typing:

```
get_property DIRECTORY [current_run]
```

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

Step 3: Create New Runs

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

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





٨	Create New R	luns			×
		lementation Runs ure one or more impleme	ntation runs using variou:	s parts, constraints, flows and strategies	4
	Create Implem	entation Runs			
	+ -				
	Name	Constraints Set	Part	Strategy	Make Active
	impl_2 💌	🛅 constrs_2 (act 🛩	xc7k70tfbg67 ¥	🏂 Vivado Implementation Defaults (Vivado Implementation 2 👻	\checkmark
				R	uns to create: 1
(?			< <u>B</u> ack <u>N</u> ext > <u>Finish</u>	Cancel

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

n Create New Runs	×
Launch Options Configure hosts for launching runs, and/or set advanced launch options	4
Launch girectory: So <default directory="" launch=""></default>	~
 ● Launch runs on local host. Number of jobs: 4 ○ Generate scripts only ○ Do not launch now 	
? ▲Back Next>	Einish Cancel

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

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

Tcl Console Messag	jes Log	Reports Package Pins	Design Runs	×	Power	Timir	ng Met	thodology D	RC							? _	0 6
Q ≚ ♦ 4	≪ ►	» + %															
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Strat
✓ impl_1 (active)	constrs_2	route_design Complete!	1.265	0.0	0.057	0.0	0.000	2.393	0	21	1	112.50	0	68	3/1	00:14:25	Viva
▷ impl_2	constrs_2	Not started															Vivad
<																	>

Step 4: Making Incremental Debug Changes

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

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

Sources Netlist ×	? _ 🗆 🖸
X H	۵
🕅 top	^
> 🖻 Nets (4564)	
> 🖻 Leaf Cells (223)	
> Clkgen (clock_generator)	
> CpuEngine (or1200_top)	
> 🦉 dbg_hub (dbg_hub_CV)	
✓ I fftEngine (fftTop)	
Nets (3331)	
≻ 师 A(16)	
> 听 C (16)	
> 「」 D (32)	
> -师☆ <mark>fifo_out (32)</mark>	
> 近 Ⅰ3 (32)	
> -∫r 14 (32)	
> 近 15 (32)	
> 「」「Ⅰ6 (32)	
> 近 17 (32)	
<u>)</u> [F 10/22)	~



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

🧼 Set Up Debug				×
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

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

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

Q					0
Name	Clock Domain	Driver Cell	Probe Type		-
> 小☆ cpuEngine/dcpu_dat_qmem (25)	clkgen/cpuClk	FDRE	Data and Trigger	~	
> 小* cpuEngine/dcqmem_adr_qmem (32)	clkgen/cpuClk	FDRE	Data and Trigger	~	
> √ ★ cpuEngine/dcqmem_dat_qmem (32)	clkgen/cpuClk	FDRE	Data and Trigger	~	
> 「 * cpuEngine/dcqmem_sel_qmem (4)	clkgen/cpuClk	FDRE	Data and Trigger	~	
> 师樂 cpuEngine/du_dsr (11)	clkgen/cpuClk	FDCE	Data and Trigger	~	
> 师業 cpuEngine/dvr00 (6)	clkgen/cpuClk	FDCE	Data and Trigger	~	
> 「fftEngine/fifo_out (32)	clkgen/fftClk	RAMB36	Data and Trigger	~	
Find Nets to <u>A</u> dd					Nets to debug: 14

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



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

Step 5: Running Incremental Compile

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

- 1. Start by defining the design checkpoint (DCP) file to use as the reference design for the incremental compile flow. This is the design from which the Vivado Design Suite draws placement and routing data.
- 2. In the Design Runs window, right-click the **impl_2 run** and select Set Incremental Implementation from the popup menu. The Set Incremental Implemenation dialog box opens.
- 3. Select Automatically use the checkpoint from the previous run.
- 4. Click **OK**. This information is stored in the INCREMENTAL_CHECKPOINT property of the selected run. Setting this property tells the Vivado Design Suite to run the incremental compile flow during implementation.
- 5. You can check this property on the current run using the following Tcl command:

get_property INCREMENTAL_CHECKPOINT [current_run]

This returns the full path to the top_routed.dcp checkpoint.

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

6. From the Flow Navigator, select Run Implementation.

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

7. Select Open Implemented Design and click **OK**. As shown in the following figure, the Design Runs window shows the elapsed time for implementation run impl_2 versus impl_1.



Tcl Console Messag	jes Log	Reports Design Runs × P	ower	DRC	Methodo	logy	Timing									1	? _ 🗆 🖒
Q <u>∓</u> ≑ I4	« >	» + %															
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 Impl
4																	

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

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

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

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





	→ X □ □ X	× // m o			Read-only
					Read-only
Copyright	t 1986-2018 Xilinx, Inc.	. All Rights Reserved.	•		
Tool Ve	ersion : Vivado v.2018.	l (win64) Build 218860	00 Wed Apr 4 18:40:38	8 MDT 2018	
	: Wed Apr 18 17:3				
	: xcosmitha32 run d : report increment	-		nlaced rot	rat
Design		ital_ieuse -iiie top_	incrementar_rease_pre_	_praced.rpc	.100
	: xc7k70t				
-	State : Fully Routed				
Increment	tal Implementation Info	rmation			
Inorement	cal implementation into	LING CION			
Table of	Contents				
1. Reuse	-				
	ence Checkpoint Informat rison with Reference Run				
-	euse Information	1			
1. Reuse	-				
	+	+	+	++	
	Matched % (of Total)		Fixed % (of Total)		
				46475	
		88.59	0.31		
+	95.69			36769	
+ Cells Nets Pins	I 95.69 I 95.80 I -	80.85 86.48	0.00 I –	189880	
+ Cells Nets Pins Ports	95.69 95.80 - 100.00	80.85 86.48 100.00	I 0.00 I - I 100.00	189880 135	
+ Cells Nets Pins Ports	I 95.69 I 95.80 I -	80.85 86.48 100.00	I 0.00 I - I 100.00	189880 135	
+ Cells Nets Pins Ports	95.69 95.80 - 100.00	80.85 86.48 100.00	I 0.00 I - I 100.00	189880 135	
+ Cells Nets Pins Ports +	95.69 95.80 - 100.00	80.85 86.48 100.00	I 0.00 I - I 100.00	189880 135	
+ Cells Nets Pins Ports +	95.69 95.80 – 100.00	80.85 86.48 100.00	I 0.00 I - I 100.00	189880 135	

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

Conclusion

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

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



Lab 8

Using the Vivado Serial Analyzer to Debug Serial Links

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

The Serial I/O Analyzer has several benefits:

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

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 using the Vivado tool.
- Interact with the design using Serial I/O Analyzer. This includes connecting to the target KC705 board, configuring the device, and interacting with the IBERT/Transceiver IP cores.
- Perform a sweep test to optimize your transceiver channel and to plot data using the IBERT sweep plot GUI feature.

Design Description

You can customize the IBERT core and use it to evaluate and monitor the functionality of transceivers for a variety of Xilinx[®] 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.

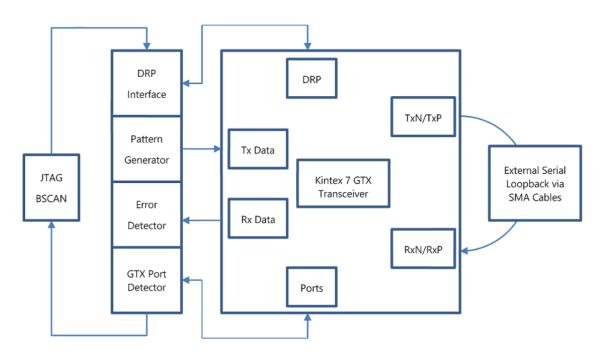


Figure 4: IBERT Design Flow

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

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

1. Invoke the Vivado IDE.



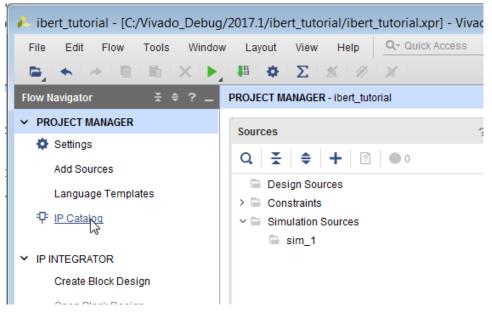
- 2. In the Quick Start screen, click **Create Project** to start the New Project wizard, and click **Next**.
- 3. In the Project Name page, name the new project ibert_tutorial and provide the project location (C:/ibert_tutorial). Ensure that **Create Project Subdirectory** is selected. Click Next.
-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

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

Step 2: Adding an IBERT Core to the Vivado Project

1. In the Flow Navigator click IP Catalog.

The IP Catalog opens.



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



Project Summary × IP Catalog ×					? 🗆 🖸
Cores Interfaces					
≚ ≑ 释 +€ ⊁ ⊘ ⊕ ᡚ Q~ IBERT	8				•
Name	∧1 AXI4	Status	License	VLNV	
 Vivado Repository 					
✓					
✓					
IBERT 7 Series GTX		Production	Included	xilinx.com:ip:ibert_7series_gtx:3.0	

- 3. Double-click IBERT 7 series GTX IP. This brings up the customization GUI for the IBERT.
- 4. In the Customize IP dialog box, choose the following options in the Protocol Definition tab:
 - a. Type the name of the component in the Component Name field. In this case, leave the name as the default name, ibert_7series_gtx_0.
 - b. Ensure that the Silicon Version is selected as General ES/Production.
 - c. Ensure that the Number of Protocols option is set to 1.
 - d. Change the LineRate (Gb/s) to 8.
 - e. Change DataWidth to 40.
 - f. Change Refclk (MHz) to 125.
 - g. Ensure that the Quad Count is set to 2.
 - h. Ensure Quad PLL box is selected.

i Customize IP										×
IBERT 7 Series GTX (3.0)									2	
Documentation 📄 IP Location C Switch to Defaults										
Show disabled ports	Component Name	ibert_7	7series_gtx_0						C	
	Protocol Definition	Pro	otocol Selection	Clock S	ettings Summa	агу				
RXN_[[3:0] RXP_[[3:0] GTREFCLK0_[[0:0] TXP_0[3:0] GTREFCLK1_[[0:0] SYSCLK_1	Silicon Version General Initial ES The maximum nu Number of Protoco Protocol	3 mber o	iduction of quads available for LineRate(Gbps)	or this d	evice is 4 DataWidth	Refclk(MHz)	Quad Count		♥ Quad PLL	
	Custom 1	•		0	40 ~	125.000 *	2	•	✓	
							[ОК	Cancel	

- 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)								λ
🗿 Documentation 🛛 IP Location C Switch to Defaults								
Show disabled ports	Component Name it	pert_7series_gtx_0						0
	Protocol Definition	Protocol Selection	Clock Settings	Summ	ary			
	Please select Proto	col-Quad combination						
	GTX Location	Protocol Sel	ected		efclk Selection		TXUSRCLK Source	
	QUAD_115	None			one	۳	Channel 0	*
RXN 1[7:0]	QUAD 116	None		▼ N			Channel 0	÷
RXP_[[7:0] TXN_0[7:0]	QUAD_117	Custom 1/8			GTREFCLK0 117	•	Channel 0	<u> </u>
 GTREFCLK0_I[1:0] TXP_0[7:0] 	QUAD_118	Custom 1 / 8	Gbps	* M	GTREFCLK0 117	۳	Channel 0	Ľ.
GTREFCLK1_[[1:0] RXOUTCLK_0 -								
							OK Can	cel

- 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.





ocumentation 📄 IP Location C Switch to Defaults										
Show disabled ports	Component Name	bert_7series_gtx_0								
	Protocol Definition	Protocol Selection	Clock Settings	Sumn	nary					
	RXOUTCLK Probe									
RXN_I(7:0)	Clock Type	Source	I/0 Standard		P Package Pin		N Package Pin		Frequency(MHz)	
 RXP_I[7:0] TXN_O[7:0] 	System Clock	External *	DIFF SSTL15	•	AD12	8	AD11	8	200.00	
GTREFCLK0_[(1:0] TXP_0[7:0] = GTREFCLK1_[(1:0] RXOUTCLK_0 =	System Clock Terr	mination Settings								
- SYSCLK_I	Enable DI	FF Term								

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

Customize IP		
IBERT 7 Series GTX (3.0)		4
ODocumentation 📄 IP Location C Switch to Defaults		
Show disabled ports	Component Name ibert_7series_gbt_0	8
	Protocol Definition Protocol Selection Clock Settings	Summary
	IBERT Design Summary	
	Number of Protocols	1
	System Clock Source	External (P Pin : AD12)
RXN_I[7:0]	System Clock Source QUAD Count	External (N Pin : AD11) 2
RXP_I[7:0] TXN_0[7:0]	MMCM Count	1
GTREFCLK0_[[1:0] TXP_0[7:0]	RefClk Sources	1
GTREFCLK1_I[1:0] RXOUTCLK_O		
- SYSCLK_I		
		OK Cancel

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





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

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





Sources			? _ 🗆 🖒 X	Project Su
Q 🛨 🕴	€ + ?	•	o 🌣	Cores
~ 🖨 Design	Sources (1)			¥ ♦
>₽ <mark>₽</mark> ≛ ib	ert_7series_gt	x 0 (i)	hert 7series atx 0.xci)	
> 📄 Constra	ints		Source Node Properties	Ctrl+E
	ion Sources (1)		Enable Core Container	D
> 🚍 sim	_1 (1)	۶	Re-customize IP	
			Generate Output Products	
			Reset Output Products	
			Upgrade IP	
			Copy IP	
			Open IP Example Design	N
			IP Documentation	× 13
			Replace File	
llianaahu	ID O		Copy File Into Project	
Hierarchy	IP Sources		Copy All Files Into Project	Alt+I
Source File Pr	operties	×	Remove File from Project	Delete
			Enable File	Alt+Equals
() :		1		
🗘 ibert_7seri	es_gtx_0.xci		Disable File	Alt+Minus
♀ ibert_7serie IP name:	es_gtx_0.xci IBERT 7 Serie		Disable File Hierarchy Update	Alt+Minus
		c		
IP name:	IBERT 7 Serie 3.0 (Rev. 16) The IBERT 7 S	c	Hierarchy Update	
IP name: Version:	IBERT 7 Serie 3.0 (Rev. 16) The IBERT 7 S customizable and monitorin	C	Hierarchy Update Refresh Hierarchy	
IP name: Version:	IBERT 7 Serie 3.0 (Rev. 16) The IBERT 7 S customizable and monitorin transceivers. 1 generators an	C	Hierarchy Update Refresh Hierarchy IP Hierarchy	
IP name: Version:	IBERT 7 Serie 3.0 (Rev. 16) The IBERT 7 S customizable and monitorin transceivers. T generators an implemented ports and the		Hierarchy Update Refresh Hierarchy IP Hierarchy Set as Top	
IP name: Version:	IBERT 7 Serie 3.0 (Rev. 16) The IBERT 7 S customizable and monitorin transceivers. T generators an implemented ports and the attributes of th	**** C	Hierarchy Update Refresh Hierarchy IP Hierarchy Set as Top Set File Type	
IP name: Version:	IBERT 7 Serie 3.0 (Rev. 16) The IBERT 7 S customizable and monitorin transceivers. T generators an implemented ports and the	C	Hierarchy Update Refresh Hierarchy IP Hierarchy Set as Top Set File Type Set Used In	

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 b placed.	e 🍌
Location	
Put example project directory here: C:/Vivado_Debug/2017.1	
✓ Overwrite existing example project	
ОК Са	ancel

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

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

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

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

2. Select Open Hardware Manager, and click OK.





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

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





ibert_7series_gtx_0_ex - [c:/	Vivado_Debug/2017/ibert_7series_gtx_0_ex/ibert_7series_gtx_0_ex.xpr] - Vivado		
File Edit Flow Tools	Window Layout View Help Q- Quick Access	write_bitstream Complete 🗸	
	🕻 🕨 🗱 🕸 🗶 🖉 💥 Dashboard	🔚 Serial I/O Analyzer 🗸 🗸	
Flow Navigator 😤 🌲 ? 🔔	HARDWARE MANAGER - unconnected	? ×	
✓ PROJECT MANAGER [^]	1 No hardware target is open. Open target		
Settings	Hardware ? _ 🗆 🗆 X		
Add Sources			
Language Templates			
👎 IP Catalog			
 IP INTEGRATOR Create Block Design Open Block Design Generate Block Design 	No content		
	Properties ? _ □ Ľ ×		
 SIMULATION Run Simulation 	$\leftarrow \rightarrow \phi$		
 RTL ANALYSIS Open Elaborated Design SYNTHESIS Run Synthesis 	Select an object to see properties		
> Open Synthesized Desigr	Tcl Console Messages Serial I/O Links × Serial I/O Scans	2 _ 0 13	
 IMPLEMENTATION Run Implementation 			
> Open Implemented Desig			
PROGRAM AND DEBUG	No content		
Senerate Bitstream			
 Open Hardware Manage Open Torget 			
Open Target			

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

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

Send Feedback





HARDWARE MANAGER - unconnected				
🚯 No hardware target is open. Op	en tar <u>o</u>	get		
Hardware	ø	Auto Connect		
		Recent Targets	•	
		Available Targets on Server	$\mathbf{F}_{\mathbf{r}}$	
		Open New Target		

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

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

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





🍌 Open New	Hardware Target	×		
Select local or i	erver Settings remote hardware server, then configure the host name and port settings. Use Local server if the target is attached to the otherwise, use Remote server.	4		
<u>C</u> onnect to:	Local server (target is on local machine)			
Click Novt to	s lounds and/or connect to the human or (and 2121) application on the 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>⊡</u> inish Can	cel		

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

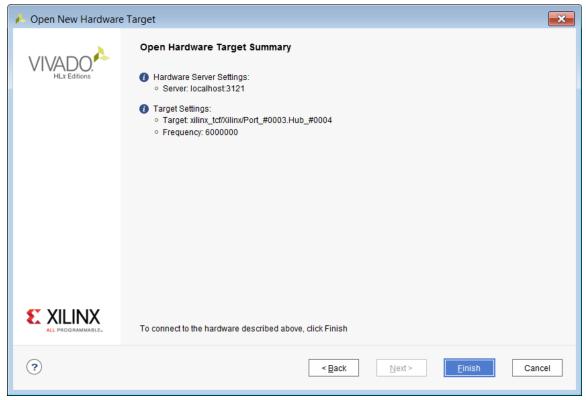
There is only one target board in this case to connect to, so that the default is selected.





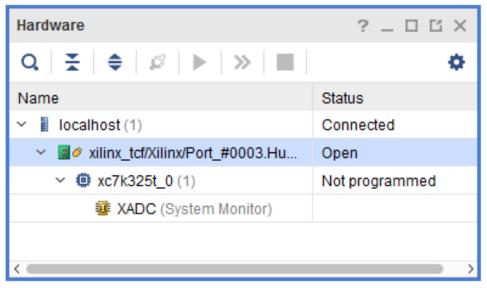
🍐 Open New H	ardware Targ	et						×
	target from the		argets, then set the t a different target.		oriate JTAG clock (TCK) frequency.	lf you do not see the	4
Hardware <u>T</u> arg	ets							
Туре	Name		JTAG Clock Free	quency				
xilinx_tcf	Xilinx/Port_#00	03.Hub_#0004	600000	~				
			Add Xilin	x Virtual	Cable (XVC)			
Hardware <u>D</u> evi	c <mark>es (</mark> for unknov	vn devices, spec	ify the Instruction	Registe	er (IR) length)			
Name	ID Code	IR Length						
@ xc7k325t_0	33651093	6						
Hardware serve	r: localhost:312	21						
?					< <u>B</u> ack	<u>N</u> ext >	<u>F</u> inish	Cancel

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

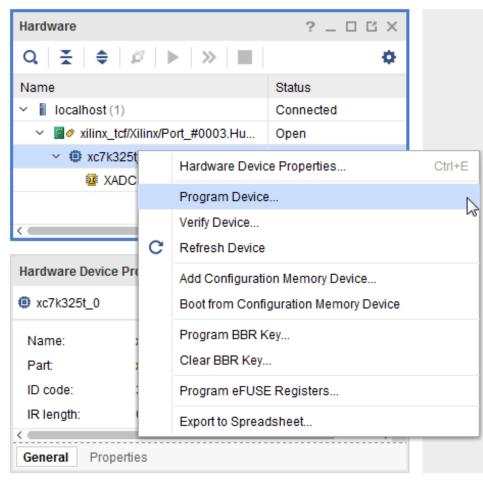




5. The Hardware window in Vivado IDE should show the status of the target FPGA on the KC705 board.



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





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

🥕 Program Device		×	
Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.			
Bitstre <u>a</u> m file: Debu <u>q</u> probes file:			
✓ Enable end of st	artup check		
?	<u>P</u> rogram Can	cel	

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 素 ♦ ∅ ▶ ≫ ■	0
Name	Status
 Iocalhost (1) 	Connected
✓ ✓ ✓ ✓ ✓ xilinx_tcf/Xilinx/Port_#0003.Hu	Open
 v (1) xc7k325t_0 (2) 	Programmed
🔯 XADC (System Monitor)	
🛩 📴 IBERT (IBERT)	
✓ 🖏 Quad_117 (5)	
COMMON_X0Y2	Locked
MGT_X0Y8	No Link
MGT_X0Y9	No Link
MGT_X0Y10	No Link
MGT_X0Y11	No Link
Quad_118 (5)	
COMMON_X0Y3	Locked
NGT_X0Y12	No Link
NGT_X0Y13	No Link
NGT_X0Y14	No Link
NGT_X0Y15	No Link
<	>

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





Hardware		? _ 🗆 🖒 X		
Q ¥ ♦ ∅ ▶ ≫		•		
Name		Status		
 Iocalhost (1) 		Connected		
✓ Ø xilinx_tcf/Xilinx/Port_#000	3.Hu	Open		
 xc7k325t_0 (2) 		Programmed		
礕 XADC (System Monit	or)			
🗸 🦉 IBERT (IBERT)		IBERT Core Properties	Ctrl+E	
V N Quad_117 (5)		Create Links		
			2	
NGT_X0Y8	MGT_X0Y8 Auto-detect Links			
MGT_X0Y9 MGT_X0Y10		Serial I/O Links		
		Serial I/O Scans		
MGT_X0Y11		Commit Properties		
V N Quad_118 (5)	с	Refresh Serial I/O Objects		
	Ŭ			
MGT_X0Y12 MGT_X0Y13		Select	•	
		Export to Spreadsheet		
NGT_X0Y14		No Link		
MGT_X0Y15		No Link		
<		>>>		

The Create Links dialog box opens.

10. Ensure the first transceiver pairs (MGT_X0Y8/TX and MGT_X0Y8/RX) are selected.





Create Links			
To create a new link select a TX GT and/or an RX GT, then 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_X0Y9/TX (xc7k325t_0/Quad_117) MGT_X0Y10/TX (xc7k325t_0/Quad_117) MGT_X0Y11/TX (xc7k325t_0/Quad_117) MGT_X0Y12/TX (xc7k325t_0/Quad_118) MGT_X0Y13/TX (xc7k325t_0/Quad_118) MGT_X0Y14/TX (xc7k325t_0/Quad_118) MGT_X0Y15/TX (xc7k325t_0/Quad_118) MGT_X0Y15/TX (xc7k325t_0/Quad_118) MGT_X0Y15/TX (xc7k325t_0/Quad_118) 	MGT_X0Y8/RX (xc7k325t_0/Quad_117) MGT_X0Y9/RX (xc7k325t_0/Quad_117) MGT_X0Y10/RX (xc7k325t_0/Quad_117) MGT_X0Y11/RX (xc7k325t_0/Quad_117) MGT_X0Y12/RX (xc7k325t_0/Quad_118) MGT_X0Y13/RX (xc7k325t_0/Quad_118) MGT_X0Y14/RX (xc7k325t_0/Quad_118) MGT_X0Y15/RX (xc7k325t_0/Quad_118)		
Press the	button to Add Link		
✓ <u>C</u> reate link group			
Link group description: Link Group 0	8		
✓ Open Serial I/O Analyzer layout			
3	OK Cancel		

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





Create Links		×		
To create a new link select a TX GT and/or an RX GT, then cl	ick the Add button on the New Links toolb	ar. 🝌		
TX GTs	RX GTs			
Search: Q-	Search: Q-			
MGT_X0Y9/TX (xc7k325t_0/Quad_117)	MGT_X0Y9/RX (xc7k325t_0/Qua)			
MGT_X0Y10/TX (xc7k325t_0/Quad_117)	MGT_X0Y10/RX (xc7k325t_0/Qua			
 MGT_X0Y11/TX (xc7k325t_0/Quad_117) MGT_X0Y12/TX (xc7k325t_0/Quad_118) 	MGT_X0Y11/RX (xc7k325t_0/Qua MGT_X0Y12/RX (xc7k325t_0/Qua MGT_X0Y12/RX (xc7k325t_0/Qua)			
MGT_X0Y13/TX (xc7k325t_0/Quad_118)	MGT_X0Y12/RX (xc7k325t_0/Qua			
MGT_X0Y14/TX (xc7k325t_0/Quad_118)	MGT_X0Y14/RX (xc7k325t_0/Qua			
MGT_X0Y15/TX (xc7k325t_0/Quad_118)	MGT_X0Y15/RX (xc7k325t_0/Qua	ad_118)		
New Links				
+ -				
Description TX	RX	Internal Loopback		
S Link 0 MGT_X0Y8/TX (xc7k325t_0/Quad_117)	MGT_X0Y8/RX (xc7k325t_0/Quad_117)	\checkmark		
✓ Create link group Link group description: Link Group SMA		8		

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

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

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



earch: Q-									
		Search: Q-							
ew Links									
W LIIKS									
+ -									
Description	TX	RX	Internal Loopback						
s Link 1	MGT_X0Y9/TX (xc7k325t_0/Quad_117)	MGT_X0Y9/RX (xc7k325t_0/Quad_117)	\checkmark						
🗞 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)	✓						
link 4	MGT_X0Y12/TX (xc7k325t_0/Quad_118)	MGT_X0Y12/RX (xc7k325t_0/Quad_118)	\checkmark						
b Link 5	MGT_X0Y13/TX (xc7k325t_0/Quad_118)	MGT_X0Y13/RX (xc7k325t_0/Quad_118)	✓						
S Link 6	MGT_X0Y14/TX (xc7k325t_0/Quad_118)	MGT_X0Y14/RX (xc7k325t_0/Quad_118)	\checkmark						
O Entro			\checkmark						
-	MGT_X0Y15/TX (xc7k325t_0/Quad_118)	MGT_X0Y15/RX (xc7k325t_0/Quad_118)	~						

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

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

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

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

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



General Prope	rties			
Tcl Console Me		Link Properties	Ctrl+E	ans
Q ₹ ♦	×	Delete	Delete	
		Create Links		_
Name		Create Link Group		us
🖨 Ungrouped I				
👻 🚳 Link Group S		Create Stran		
🗞 Link 0		Create Sweep		Gbps
👻 🚳 Link Group I		Commit Properties		
% Link 1	С	Refresh Serial I/O Objects		Gbps

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



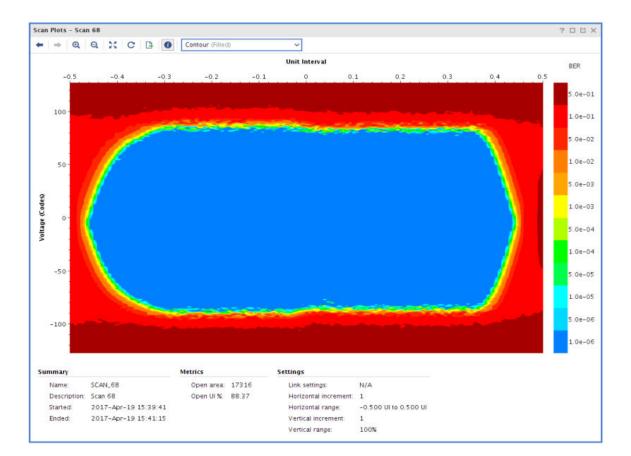


🔥 Create Scan			×							
Set the descriptio on the selected lin		er properties to create and optionally run a scan	4							
Link: Lini	k 0 (MGT_	X0Y8/TX, MGT_X0Y8/RX)								
Description: So	can O	an 0								
Scan Properties	S		_							
<u>S</u> can type:		2D Full Eyescan	•							
<u>H</u> orizontal in	crement:	8 🗸	•							
H <u>o</u> rizontal ra	nge:	-0.500 UI to 0.500 UI	·							
Vertical incre	ment:	8 🗸	·							
V <u>e</u> rtical range	e:	100% ~	•							
Dwell										
) <u>B</u> ER: 1	e-5	· · · · · · · · · · · · · · · · · · ·	•							
		0 4	-							
✓ <u>R</u> un scan										
?		OK Canc	el							

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



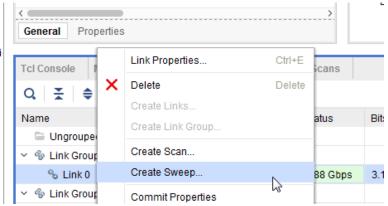




The 2D Scan Plot is a heat map of the BER value.

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

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



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



🔥 Create Swe	ер			×
Select the swe	ep propertie	s and values	to create and optionally run a set of scans on the selected link.	4
Link: L	ink 0 (MGT_	X0Y8/TX, MG	T_X0Y8/RX)	
Description:	Sweep 0			\otimes
Scan Propert	ies			
Scan type:		2D Full Eye	scan 🗸	
Horizontal	increment:	8	~	
– H <u>o</u> rizontal		-0.500 UI to	0.500 UI	
		8	×	
_		100%		
Vert <u>i</u> cal rai	ige.	100%	v	
Dwell				
• <u>B</u> ER:	1e-5		~	
O <u>T</u> ime:			0 🗘	
Sweep Prope	rtico			
Sweep m		i Custom	For each property select values to be swept. The sweep will cover all combinations of property values.	
<u>S</u> et Prop	erties & Va	ues <u>P</u> rev	iew 81 Scans	
+ -		F I		
Order	Property	Name	Values to Sweep	# of Values
°o 1	RXTERM	ı •	r 100 mV,550 mV,1100 mV	3
% 2	TXDIFFS	WING		3
% 3	TXPOST			3
°5 4	TXPRE	`	r 0.00 dB (00000),4.08 dB (01111),6.02 dB (11111)	3
Reset RX	after applyin	g Settings for	each scan	
✓ Run swee	p			
(?)			OK	Cancel
			ŬŔ.	Cancel

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.

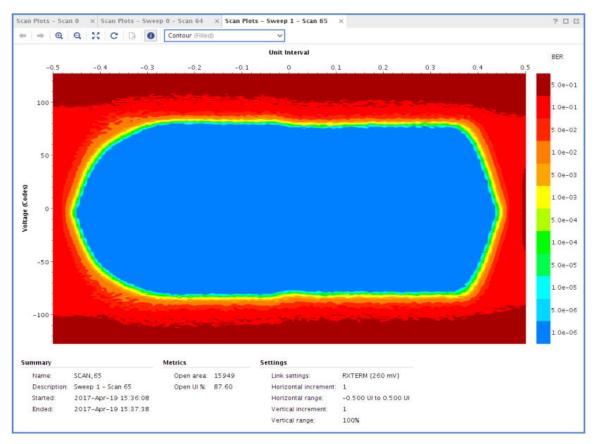
Q 꽃 ♦ ▶ ■	B	B								
Name	Link	Link Settings	Reset RX	Scan Type	Status	Progress	Open Area	Open UI %	Horz Incr	Horz Range
Scans (4)										
Sweep 0 (81)				2d_full_eye	Done				8 ~	-0.500 UI to 0.500 UI
Sweep 0 - Scan 2		RXTERM {100 mV} TXDIFFSWING (269 mV (0000)) TXPOST (0.00 dB (00000)) TXPRE {		2d_full_eye	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 V	-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 V	-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 V	-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 V	-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 V	-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 V	-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 V	-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 V	-0.500 UI to 0.500 UI
Sweep 0 - Scan 12		RXTERM {100 mV} TXDIFFSWING {741 mV (0111)} TXPOST {0.00 dB (00000)} TXPRE {		2d full eye	Done	100%	10112	77.78	8 ¥	-0.500 UI to 0.500 UI

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



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

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





Lab 9

Using the Vivado ILA Core to Debug JTAG-AXI Transactions

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

What is the JTAG to AXI Master IP core?

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

Key Features

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

Additional Documentation

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

Design Description

This section has three steps as follows:

- 1. Creating a simple design in IP integrator that includes a System ILA and JTAG-to-AXI master.
- 2. Programming the Kintex[®]-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	Boards					
Reset A	VI Filters					
Vendor:	All 🗸 Name: 🖌	All				
Search:	Qv	~				
	y Name		Preview		Vendor	File 1
Add D	aughter Card Connections				xilinx.com	1.4
	-7 KC705 Evaluation Platform aughter Card Connections				xilinx.com	1.5
	-UltraScale KCU105 Evaluation Pla aughter Card Connections	atform		and the second s	xilinx.com	1.4

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





A	New Project x
	New Project Summary
HLx Editions	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 Package: ffg900 Speed Grade: -2
E XILINX.	To create the project, click Finish
?	< <u>B</u> ack <u>N</u> ext > <u>Finish</u> Cancel

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





<u>File Edit Flow Tools Report</u>	s <u>W</u> indow La <u>v</u> out <u>V</u> iew <u>H</u> elp Q⊸o	uick Access														R	leady
															= De	fault Layout	,
Flow Navigator 😤 🌲 î																	?
PROJECT MANAGER	Sources	2		×	Projec	t Summ	any									2 5	162
Settings				ò			Dashboard									: -	
Add Sources	C Desian Sources			*	oren	i cii	Dashbuaru										
Language Templates	> Constraints				Settings Edit												
👎 IP Catalog	v 🚍 Simulation Sources				Project name: project_2												
	sim_1				-	ect locat		home/mpiazza/p	project_	2							
PINTEGRATOR	> 🚍 Utility Sources					luct fami ect part:	,	intex-7 intex-7 KC705 E	Naluatio	n Platf	orm (vc7	/k325tffa	900-2				
Create Block Design						module		lot defined						, 			
Open Block Design	Hierarchy Libraries Compile Orde	r		_		jet langu	-	erilog									
Generate Block Design	Properties	?.		\times	Simu	ilator lan	guage: M	lixed									
SIMULATION		+	$ \rightarrow $	•	Pop	d Part											
Run Simulation																	
						olay nam rd part r		itex-7 KC705 Ev nx.com:kc705:p			m						
RTL ANALYSIS	Select an object to see pr	operties			Connectors: No connections												
> Open Elaborated Design					Repository path: /proj/xbuilds/2018.3_0815_0946/installs/lin64/Vivado/2018.3									2018.3	/data/boar	ds/board_files	
 SYNTHESIS 					<			au viliny com/kc	705		_		_	_	_		>
Run Synthesis	Tcl Console Messages Log Re		cian Pu													2	
> Open Synthesized Design			Sign Ku														
	Name Constraints Status	T %	TNS	WHS	THS	TOWE	Total Dawar	Failed Routes	1117		DD AMA	UDAM	DCD	Stort	Flancad	Run Strategy	
IMPLEMENTATION	✓ ▷ synth_1 constrs_1 Not status		1145	WHO	1 HD	1 P WD	Total Power	Falleu Routes	LUT	FF.	DRUMMIS	UPDAM	Dar	Start	Elapseu	Vivado Synth	
Run Implementation	▷ impl_1 constrs_1 Not star	ted														Vivado Imple	ment
> Open Implemented Design																	
PROGRAM AND DEBUG																	
Senerate Bitstream																	
> Open Hardware Manager																	

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





Board ? _ 🗆 🖾 Diagram	
	2 素 ≑ + ⁰, ≫ ⊠ ≯, C º :
	1
Search: Q- JTAG to AXI 💿 (1 match)	
F JTAG to AXI Master	
	This design is empty. Drass the L butten to odd ID
	This design is empty. Press the 🕂 button to add IP.
roperties	
Toperne:	
Repoi	
shing IF ser IP r	
2d Vivac	46/installs/lin64/Vivado/2018.3/data/ip'.
1 = 00:C	:= 6768.984 ; gain = 7.852 ; free physical = 574
et_2/pr	id>
: Source ENTER to select, ESC to cancel, Ctrl+Q for IP details	

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





4	Re-customize IP			×
JTAG to AXI Master (1.2)				4
1 Documentation 📄 IP Location				
Show disabled ports	Component Name jtag_axi_0			
	AXI Protocol	AXI4	~	
	AXI Address Width	32	~	
	AXI Data Width	32	*	
	AXI ID Width	1	0 [1 - 4]	
	AXI4 Burst Type Support	ALL BURST TYPES	~	
aclk M_AXI + ∷	Write Transaction Queue Length		0 [1 - 16]	
	Read Transaction Queue Length	1	[1 - 16]	
			ок с	ancel

- 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-custor	mize IP			×
AXI BRAM Controller (4.0)					4
🟮 Documentation 🛛 🖨 IP Location					
Show disabled ports	Component Name axi_brar	m_ctrl_0			
	AXI Protocol Data Width		AXI4 ~		
	Memory Depth (Auto)		8192 ~		
	ID Width (Auto)	XI Narrow Bursts	0 ~ Yes ~		
+ S_AXI - s_axi_acik BRAM_PORTA +	BRAM Options BRAM_INSTANCE (Aut Number of BRAM inter		~ ~		-
	ECC Options				
	Enable ECC	No 🗸			
	ECC TYPE	Hamming 🗸 🗸			
	Enable Fault Injection	No 🗸			
	ECC Reset Value	0 ~			
				OK Cano	el

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





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

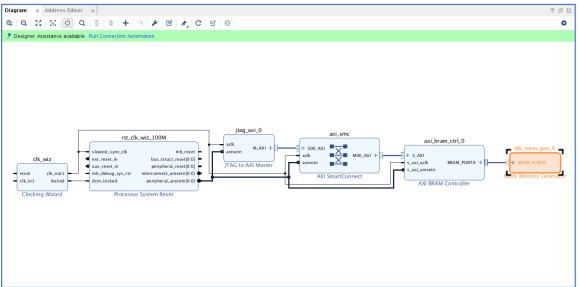
13. At this point the design should look like the following figure.





Diagram × Address Editor ×	? 0 5
Q Q X X 0 Q X 0 + 0 ≯ 8 ≯ 8 0 0	•
Designer Assistance available. Run Connection Automation	
axi_bram_ctrl_0	
jtag_axi_0	
acik M_AXI + S_AXI s_axi_acik BRAM_PORTA + BRAM_PORTA	
• aresetn	
JTAG to AXI Master AXI BRAM Controller	

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



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



16. Invoke the Customization Dialog for the Clocking Wizard by double-clicking the IP in the block diagram canvas. When the dialog appears, set CLKIN_1 to sys_diff_clk and EXT_RESET__IN to reset. Click **OK**.

Note: It is not necessary to add constraints for these ports because the project has been generated using an evaluation board as the target and the IP allows the constraint information to be selected with the sys_diff_clk.

ocumentation 📄 IP Location			
Symbol Resource	Component Name clk_wiz		
Show disabled ports	Board Clocking Options Output Clock	s MMCM Settings Summary	
	Associate IP interface with board interface		
	IP Interface	Board Interface	
	CLK_IN1	sys diff clock	•
	CLK_IN2	Custom	•
	EXT_RESET_IN	reset	
+ CLK_IN1_D clk_out1 reset locked			

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



Diagram x Address Editor x	? d 🗅
	0
reset	bik.mem.gen_0

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**.



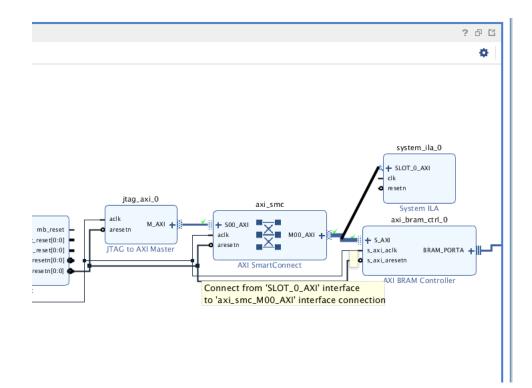


4	Re-customize IP	
System ILA (1.1)	1	
🚺 Documentation 🛛 🗁 IP Location		
IP Symbol Resources	Component Name system_ila_0	
BRAM	To configure more than 64 probe ports use Vivado Tcl Console	
Resource Estimates	General Options Interface Options	
100.0	Monitor Type	
90.0	Monitor Type INTERFACE 🗸	
80.0	Number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

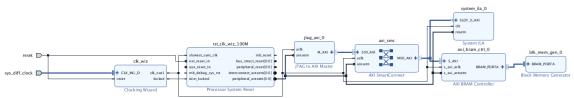
20. Now, make a connection between the System ILA SLOT_0_AXI port and the S_AXI port on the AXI BRAM Controller. Do this by clicking on the SLOT_0_AXI port and clicking again on the S_AXI port on the AXI BRAM Controller.







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



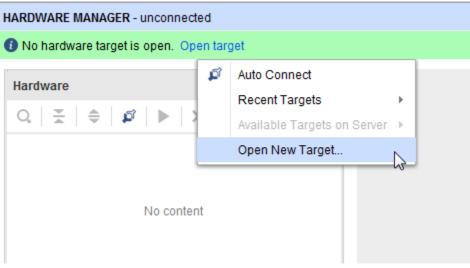
- 22. In the upper left side of the Vivado IDE, click File \rightarrow Save Block Design. Select File \rightarrow Close Block Design in the same menu to close the block design.
- 23. In the sources window, right-click on **design_1 block design** and select **Create HDL Wrapper**. Allow Vivado IDE to manage the wrapper, and click **OK**.
- 24. In the Flow Navigator on the left side of the Vivado IDE, click Generate Bitstream.
- 25. Click **OK** to implement the design.
- 26. Wait until the Vivado Status window shows write_bitstream complete.
- 27. In the Bitstream Generation Completed dialog, select **Open Hardware Manager**, and click **OK**.



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

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

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



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



🥕 Open New Hardware Target	×
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.	4
Connect to: Local server (target is on local machine)	
Click Next to launch and/or connect to the hw_server (port 3121) application on the local machine.	
(?) <back< td=""> Next> Elnish C.</back<>	ancel

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**.





٨	Open New H	ardware Targ	et				×
Se		e target from the l				oriate JTAG clock (TCK) select a different target.	4
	Hardware <u>T</u> arg	ets					
	Туре	Name		JTAG Clock Freque	ency		
	xilinx_tcf	Xilinx/Port_#00	03.Hub_#0004	6000000	Ý		
	Hardware <u>D</u> evi	C es (for unknow		nx Virtual Cable (XVC cify the Instruction R		er (IR) length)	
	Name	ID Code	IR Length				
	# xc7k325t_(Hardware serve) 33651093 er: localhost:312	6				
(?		< <u>E</u>	<u>N</u> ext	>	Einish	ncel

- 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 Target 💽					
HLX Editions	 Open Hardware Target Summary Hardware Server Settings: Server: localhost.3121 Target Settings: Target Xilinx_Lct/Xilinx/Port_#0003.Hub_#0004 Frequency: 6000000 				
E XILINX AL PROGRAMMABLE.	To connect to the hardware described above, click Finish < Back Next > Einish Cancel				

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

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

Hardware	? _ 🗆 🗹	í X
Q 素 ♣ ∅ ▶ ≫ ■		٥
Name	Status	
Y localhost (1)	Connected	
✓ ✓ ✓ ✓ xilinx_tcf/Xilinx/Port_#0003.Hu	Open	
✓ ⊕ xc7k325t_0 (1)	Not programmed	
XADC (System Monitor)		

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

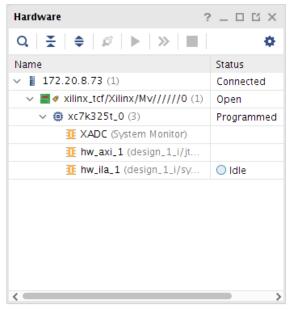




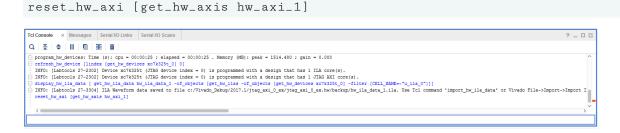
🥕 Program Device		×								
	ramming file and download it to your hardware device. You can ug probes file that corresponds to the debug cores contained in the g file.	A								
Bitstre <u>a</u> m file:	1/jtag_axi_0_ex/jtag_axi_0_ex.runs/impl_1/example_jtag_axi_0.bit									
Debu <u>q</u> probes file:	Debug probes file: 1/jtag_axi_0_ex/jtag_axi_0_ex.runs/impl_1/example_jtag_axi_0.ltx 💿 🚥									
Cookie and of a										
(?)	<u>P</u> rogram Ca	ncel								

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

9. Verify that the JTAG to AXI Master and ILA cores are detected by locating the hw_axi_1 and hw_ila_1 instances in the Hardware Manager window.



10. You can communicate with the JTAG to AXI Master core via Tcl commands only. You can issue AXI read and write transactions using the run_hw_axi command. However, before issuing these transactions, it is important to reset the JTAG to AXI Master core. Because the aresetn input port of the jtag_axi_0 core instance is not connected to anything, you need to use the following Tcl commands to reset the core:





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

set wt [create_hw_axi_txn write_txn [get_hw_axis hw_axi_1] -type WRITE address C0000000 -len 128 -data {44444444_33333333_22222222_11111111]}

where:

- write_txn is the name of the transaction.
- [get_hw_axis hw_axi_1] returns the hw_axi_1 object.
- -address C0000000 is the start address.
- -len 128 sets the AXI burst length to 128 words
- -data {4444444_3333333_2222222_11111111} is the data to be written.

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

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

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

where:

- read_txn is the name of the transaction.
- [get_hw_axis hw_axi_1] returns the hw_axi_1 object.
- -address C0000000 is the start address.
- -len 128 sets the AXI burst length to 4 words.
- 13. After creating the transaction, you can run it as a write transaction using the run_hw_axi command:

run_hw_axi \$wt

This command should return the following:

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

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

run_hw_axi \$rt

This command should return the following:

```
INFO: [Labtools 27-147] : READ DATA is : 44444443333333222222221111111...
```





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

- 1. In the ILA hw_ila_1 dashboard, locate the Trigger Mode Settings area and set Trigger mode to **ADVANCED_ONLY**.
- 2. In the Capture Mode Settings area, set the Trigger position to **512**.
- 3. In the Trigger State Machine area click the **Create new trigger state machine** link.

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

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

in New Trigger State Machine File	×
Save In: 🚚 jtag_axi_0_ex	✓ Ø S = ¥ A S = ::
Jag_axi_0_ex.cache Jag_axi_0_ex.hw Jag_axi_0_ex.hw Jag_axi_0_ex.ioplanning Jag_axi_0_ex.ip_user_files Jag_axi_0_ex.runs Jag_axi_0_ex.sim Jag_axi_0_ex.srcs	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 name: txns Files of type: Trigger State Machine Files (.tsm)	~
	Save Cancel





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

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

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

- Beginning of the read address phase
- Beginning of the read data phase
- End of the read data phase
- 6. Arm the trigger of the ILA by right-clicking the **hw_ila_1 core** in the Hardware Manager window and selecting **Run Trigger**.





Hardware					? _ 🗆	Ľ×	exa	imple_jtag_axi_0.v
Q X ♦	ø 🕨	»				•		Waveform - hw ila
Name			S	tatus			s	
👻 📱 localhost (1	1)		С	onnected			oard Options	Q + -
✓ ✓ xilinx_t	tcf/Xilinx/Port_#	ŧ0003.	.Hu 0	pen			őp	ILA Status: Idle
✓	325t_0 (3)		P	rooramme	d		Dar	Nome
🦉 XA	DC (System N		Hardwar	e Device P	roperties		C	trl+E axi_araddr[31
🦉 hv	v_axi_1 (AXI)		Program	Device				axi_arburst[1:
📴 hv	v_ila_1 (u_ila_		Verify De	vice				axi_arcache[3
			Run Trig	ger				axi_arlen[7:0]
		>>	Run Tria	- ger Immed	diate			axi_arprot[2:0
			Stop Trig	aer				axi_awaddr[3 axi_arqos[3:0
				uto Re-tric	Ider			axi_arsize[2:0
				Auto Re-tri	-			axi_awburst[1
								axi_awcache[
			Create U	ser Define	ed Probe			axi_awlen[7:0
		С	Refresh	Device				
			Add Con	figuration I	Memory Dev	ice		
			Boot from	n Configur	ation Memo	ry Device		
Hardware Device	Properties		Program	BBR Key.				igs - hw_ila_1
xc7k325t_0			Clear BE	R Key				jer Mode Sett
Name:	xc7k325t_(Program	eFUSE R	egisters			rigger mode:
Part:	xc7k325t		Export to	Spreadsh	eet			Frigger state m
ID code:	33651093							
IR length:	6							Capture Mode Set

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



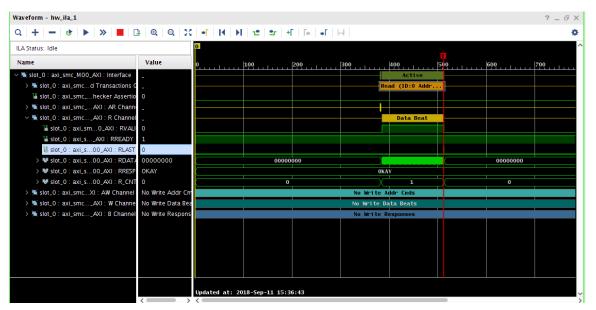


Settings - hw_ila_1 Status - hw_ila_1 ×
🔁 🕨 🗶 🗧 🕫
Core status 💿 🔵 🌑 🔵 Waiting for Trigger
Trigger State Machine - Flags: 0 1 2 3
Trigger state: wait_for_arvalid (0)
Capture status - Window 1 of 1
Window sample 512 of 1024
50%

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

run_hw_axi \$rt

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



Lab 10

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

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

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

IBERT PS-GTR transceiver does not provide these capabilities:

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

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



TIP: This is a supported feature in Vivado[®] Design Suite 2017.2 and above.

IBERT PS-GTR Flow

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

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



Tools Required

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

Board/Part/Components Required

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

Required Files

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

Assumptions

- 1. FSBL should always target Cortex[®]-A53 processor as R5 (psu_cortexr5_0) is exclusively used by IBERT PS-GTR.
- 2. Physical devices such as SATA drive, PCIe card, etc. are needed for validation.



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

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

	New Project	×
HLL Editions	New Project Create a New Vivado Project This wizard will guide you through the creation of a new project. To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.	
E XILINX.	< <u>B</u> ack <u>Finish</u> Cance	el

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



efault Part				
oose a default Xilinx part or board for your project.				
Parts Boards Reset All Filters				Install/Update Boards
Vendor: All V Name: All			~	Board Rev: Latest v
Tendor, An Tendor, An				Latest *
Search: Q. V				
Display Name	Preview	Vendor	File Version	Part
	-contraint			^
Zyng UltraScale+ ZCU102 Evaluation Board Add Companion Card Connections		xilinx.com	3,4	xczu9eg-ffvb1156-2-e
Zynq UltraScale+ ZCU104 Evaluation Board Add Companion Card Connections		xilinx.com	1.1	xczu7ev-ffvc1156-2-e
Zynq UltraScale+ ZCU106 Evaluation Platform Add Companion Card Connections	50	* xilinx.com	2.6	xczu7ev-ffvc1156-2-e
Zyng UltraScale+ ZCU111 Evaluation Platform		xilinx.com	1.4	xczu28dr-ffvg1517-2-e
Xilinx Zynq UltraScale+ RFSoC ZCU1275 Characterization Ki		xilinx.com	1.0	xczu29dr-ffvf1760-2-e
<				>

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





Flow Navigator 🗧 🔍 🤉	PROJECT MANAGER -	project_1													
PROJECT MANAGER	Sources				2		×	Project Sur	nmary						
Settings	Q = +	0					0		Dashboard						
Add Sources	Design Sources						- I	oreitien	Dashboard						
Language Templates	> Constraints							Settings	Edit						
IP Catalog		ces						Project nar	ne: p	project_1					
V IP INTEGRATOR	⇒ sim_1 > ⇒ Utility Sources							Project loca				azza/git/u	g936_des	gn_files/tmp/	project_1
Create Block Design								Product fai Project par		Zynq UltraS Zyng UltraS		J102 Eva	luation Bo	ard (xczu9eg	ffvb1156-2
Open Block Design								Top modul	e name: 🕴	Not defined					
Generate Block Design					<u> </u>		-	Target land eate Block D		/eriloa					
	Hierarchy Librarie	es Compile Orde	ir.		-		Cr	eate Block L	resign	~					
 SIMULATION 	Transformers and the second	Harmonic and the second s		Pleas	e specify	specify name of block design.									
Run Simulation	Properties										le+ 7(11)	02 Evalua	ation Boar		
 RTL ANALYSIS 					Des	ign nam	e:	design_1			102:parto			525	
> Open Elaborated Design					Dire	ectory:		Local t	o Project>	~					
					Spe	cify sour	rce set:	😑 Design	Sources	~	IS WIP/2020	2 0828	0936/inst	alls/lin64/Viva	do/2020.2/
V SYNTHESIS					?			0	Ca		m/zcu102				
Run Synthesis		Select an object t	o see prope	rties				_			le+ ZCU1	02 Evalua	ation Boan	ł	
> Open Synthesized Design					_			unanges							
 IMPLEMENTATION 								P							****
Run Implementation								<							
> Open Implemented Design	Tcl Console Mess	ages Log Re	ports De	esign Ri	uns ×										
	Q ₹ ♦ 14	$\ll > $	+ %												
PROGRAM AND DEBUG Generate Bitstream	Name Con	straints Status	WNS	TNS	WHS	THS 1	TPWS	Total Power	Failed Routes	LUT F	BRAM	URAM	DSP 9	tart Elapse	d Run St
	second and the second s	strs_1 Not start													Vivado
> Open Hardware Manager	⇒ impl_1 con	strs_1 Not start	ed												Vivado

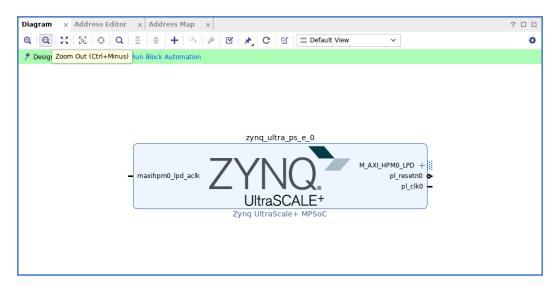
9. An empty design diagram displays. Click the Add IP button to add IP. Select the IP based on the selected board (for the ZCU102 evaluation board, search for Zynq UltraScale+ MPSoC) and double-click the selected IP.

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

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



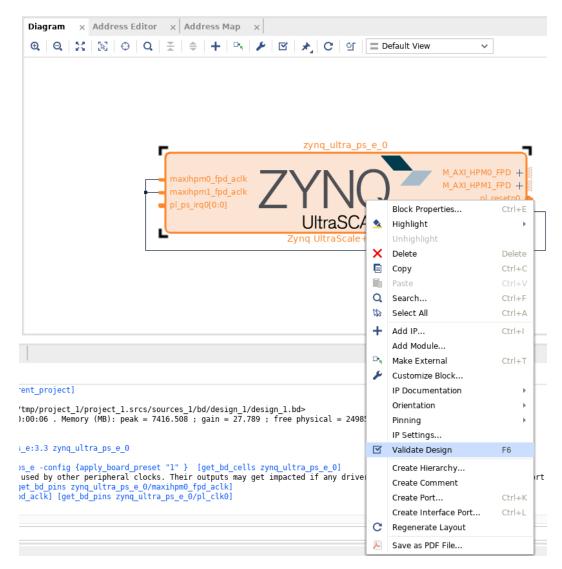




- 11. When the design diagram appears, use the following steps to validate the design:
 - a. Connect maxihpm0_fpd_aclk and maxihpm1_fpd_aclk together to pl_clk0, as shown in the following figure.
 - i. Select maxihpm0_fpd_aclk and drag it to maxihpm1_fpd_aclk.
 - ii. Select maxihpm1_fpd_aclk and drag it to pl_clk0.
 - b. Right-click the Zynq UltraScale+ MPSoC block and select **Validate Design** to validate the design. It will say validation successful. Click **OK**.







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

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

Table 2: Supported PS-GTR Connector Functionality



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

Documentation 🔅	Presets 🛛 📄 IP Location											
Page Navigator	I/O Configuration											
Switch To Adva	✓ MIO Voltage Standard											
		Bank1 [MIO 26:51]	Bank2 [MIO	52:77]	Bank3 [Dedic	ated]						
PS UltraScale+ Bloc	LVCMOS18 V	LVCMOS18 V	LVCMOS18	~	LVCMOS18	~						
I/O Configuration		Evenosio v	Evenosio	·	Evenosio							
Clock Configuration	← Q											
DDR Configuration	Search: Q-											
	Peripheral	I/O	1	Signal		I/O Type	Drive Stre					
PS-PL Configuration	> GEM											
	V USB											
	∨ USB0											
	> 🗹 USB 0	MIO 52 63	3 ~									
	> 🗹 USB 3.0	GT Lane2	~									
	> USB1											
	> USB Reset	Boot Pin	~									
	✓ 🗹 PCIe											
	> Rootport Mode Res	set MIO 31	~									
	Reset Polarity	Active Low	~									
	Lane Selection	xl	~									
	PCIe Lane0	GT Lane0										
	∨ 🕑 Display Port											
	> DPAUX	MIO 27 30) ~									
	> Lane Selection	Single Lowe	r v									
	 ✓ ✓ SATA 	L										
	SATA Lane0											
	SATA Lanel	GT Lane3	~									
	SATA Lane1 GT Lane3 Y Reference Clocks											

b. PCle-PCle - USB - SATA

Send Feedback



		Re-customiz	e IP				
Zynq UltraScale	+ MPSoC (3.3)						4
🕽 Documentation 🔅	Presets 📄 IP Location						
Page Navigator	I/O Configuration						
🗌 Switch To Adva	✓ MIO Voltage Stand	lard					
	Bank0 [MIO 0:25]	Bank1 [MIO 26:51]	Bank2 [MI	0 52:77]	Bank3 [Dedicat	ed]	
PS UltraScale+ Blo	LVCMOS18 V	LVCMOS18 V	LVCMOS1	8 🗸	LVCMOS18	~	
I/O Configuration							
Clock Configuratio	← Q 풒 ≑ ●						
DDR Configuration	Search: Q-						
DDR Configuration	Peripheral	I/O		Signal		I/O Type	Drive S
PS-PL Configuratio	· ∨ U2BU						
	> 🗹 USB 0	MIO 52 63	· ·				
	USB 3.0						
	∨ USB1						
	USB 1						
	USB 3.0						
	> USB Reset	Boot Pin	~				
	∨ 🕑 PCIe						
	> Rootport Mode Re	set MIO 31	~				
	Reset Polarity	Active Low	~				
	Lane Selection	x2	~				
	PCIe Lane0	GT Lane0					
	PCIe Lanel	GT Lanel					
	∨ □ Display Port						
	DPAUX						
	Lane Selection						
	∽ 🗹 SATA						
	SATA Lane0						
	SATA Lanel	GT Lane3	~				
	Contractor Charles						
	<						>

c. Display Port - Display Port - USB - SATA





PS-PL Configuration			Re-customi	ze IP				
Page Navigator I/O Configuration Switch To Adva × MIO Voltage Standard PS UltraScale+ Bloc Bank0 (MIO 0:25) Bank1 (MIO 26:51) Bank2 (MIO 52:77) Bank3 (Dedicated) I/O Configuration I/C MOS18 × LVCMOS18 × LVCMOS18 × LVCMOS18 × DDR Configuration Search: Q: PS:PL Configuration Search: Q: VUSB Signal VUSB Signal VUSB Subs 0 MIO 5263 × VUSB 1 VUSB 1 VUSB 1 VUSB 1 VUB 20: VUB 20: VUB 20: VUB 20: VUSB 1 VUSB 1 VUB 20: VUB 20: VUB 20:	Zynq UltraScale+	MPSoC (3.3)						
Switch To Adva × MIO Voltage Standard PS UltraScale+ Blor Bank0 [MIO 0:25] Bank1 [MIO 26:51] Bank2 [MIO 52:77] Bank3 [Dedicated] IVC Configuration ↓ UCMOS18 ↓ USMO ↓ USB ↓ USB ↓ USB 0	Documentation 🂠	Presets 🛛 📄 IP Location						
PS UltraScale+ Blor Bank0 (M0 0:253) Bank1 (MIO 26:51) Bank2 (MIO 52:77) Bank3 (Dedicated) V/O Configuration LVCMOS18 ∨ LVCMOS18 ∨ LVCMOS18 ∨ Clock Configuration € ● DDR Configuration € ● PS-PL Configuration € ● V/O USB V/O Signal I/O Type Drive S > Low Speed > USB > USB > USB 0 MIO 5263 ∨ > USB 0 GT Lane2 ∨ > USB Reset Boot Pin ∨ > USB Reset Soft Lane2 > USB Reset Boot Pin ∨ > DPAUX MIO 2730 ∨ > DPAUX MIO 2730 ∨ > DPAUX MIO 2730 ∨ > DP Lane0 GT Lane1 OP Lane1 GT Lane2	Page Navigator	I/O Configuration						
PS UltraScale+ Blor //O Configuration Clock Configuration DDR Configuration PS-PL Configuration PS-PL Configuration VUSB VUSB 0 VUSB 1	🗌 Switch To Advai	MIO Voltage Standar	rd					
I/O Configuration Clock Configuration Clock Configuration PBC Configuration PS-PL Configuration PS-PL Configuration V/O Signal V/O Type DR Configuration Ps-PL Configuration V/O Signal V/O Type Dr Speed V/O USB V/O USB 0 MIO 52 63 V/O USB 1 V/O USB 1 V/O PCIe DIAD Speed V/O USB 1 V/O USB 1 V/O PCIe Display Port V/O Display Port VLane Selection DP Lane0 GT Lane2 V/O Display Port V/O Display Port V/O Display Port V/O Display Port V/O DIAL Lower V/O SATA SATA SATA Lane0		Bank0 [MIO 0:25] E	Bank1 [MIO 26:51]	Bank2 [M	IO 52:77]	Bank3 [Dedica	ted]	
I/O Configuration Clock Configuration DDR Configuration Ps-PL Configuration Ps-PL Configuration V USB > GEM > USB0 > WIO 5263 × > WISB 0 MIO 5263 × > WISB 1 > WISB 1 > WISB 1 > WISB Reset Boot Pin × > USB Polipily Port > DPAUX MIO 2730 × > DPAUX DP Lane0 GT Lane1 DP Lane1 GT Lane0 > SATA SATA	PS UltraScale+ Bloc	LVCMOS18 V	VCMOS18 V	LVCMOS1	8 🗸	LVCMOS18	*	
DDR configuration PS-PL C	I/O Configuration							
Derived I/O Signal I/O Type Drive S Peripheral I/O Signal I/O Type Drive S > Low Speed > GEM > WSB	Clock Configuration	← Q 素 ♦ ●						
Peripheral I/O Signal I/O Type Drive S > Low Speed > > USB 0 MIO 5263 ~ Image: Section Image: Section Image: Section Image: Section I	DDR Configuration	Search: Q-						
✓ High Speed > GEM ✓ USB ✓ USB0 > Ø USB 0 MIO 52 63 ~ > Ø USB 3.0 GT Lane2 ~ > USB GT Lane2 ~ > USB Reset Boot Pin ~ > USB Porte Image: Selection < D PCle	g	Peripheral	I/O		Signal		I/O Type	Drive Stree
> GEM ✓ USB ✓ USB0 > Ø USB 0 MIO 5263 ~ > Ø USB 3.0 GT Lane2 ~ > USB1 ✓ DSB Reset Boot Pin ~ > USB Reset Boot Pin ~ ✓ PCIe Image: Selection Lane Selection Image: Selection ✓ DPAUX MIO 2730 ~ ✓ Lane Selection Image: Selection Ø Lane1 GT Lane1 Ø Lane2 Image: Selection ✓ DP Lane0 GT Lane1 Ø Lane1 Image: Selection Ø SATA Lane0 Image: Selection	PS-PL Configuration	> Low Speed						
> USB > ♥ USB 0 MIO 5263 × > ♥ USB 3.0 GT Lane2 × > ♥ USB 1.0 GT Lane2 × > USB 1		✓ High Speed						
✓ USB0 MIO 5263 ✓ ✓ USB 3.0 GT Lane2 ✓ ✓ USB Reset Boot Pin ✓ ✓ PCIe Endpoint Mode Reset Lane Selection ✓ Display Port ✓ DPAUX MIO 2730 ✓ ✓ DP Lane0 GT Lane1 DP Lane1 GT Lane0 ✓ SATA		> GEM						
> ♥ USB 0MIO 5263 ∨> ♥ USB 3.0GT Lane2 ∨> USB1> USB ResetBoot Pin ∨> USB ResetImage: Comparison of the sect		∨ USB						
> Ø USB 3.0 GT Lane2 ∨ > USB 1 > USB Reset Boot Pin ∨ > USB Reset Boot Pin ∨ > O PCIe Image: Stress of the str		V USBO						
> USB1 > USB Reset Boot Pin > USB Reset Boot Pin > PCle Image: Comparison of the section of the s		> 🗹 USB 0	MIO 52 63	~				
> USB Reset Boot Pin ✓ ✓ PCIe </td <td></td> <td>> 🕑 USB 3.0</td> <td>GT Lane2</td> <td>~</td> <td></td> <td></td> <td></td> <td></td>		> 🕑 USB 3.0	GT Lane2	~				
		> USB1						L
Endpoint Mode Reset Lane Selection ✓ ✓ Display Port > DPAUX MIO 2730 <> ✓ Lane Selection Dual Lower <> DP Lane0 GT Lane1 DP Lane1 GT Lane0 ✓ ✓ SATA		> USB Reset	Boot Pin	~				
Endpoint Mode Reset Lane Selection ✓ ✓ Display Port > DPAUX MIO 2730 <> ✓ Lane Selection Dual Lower <> DP Lane0 GT Lane1 DP Lane1 GT Lane0 ✓ ✓ SATA								
Lane Selection Image: Constraint of the selection Image: Constraint of the selection > DPAUX MIO 2730 <> Image: Constraint of the selection > DPAUX Dual Lower <> Image: Constraint of the selection DP Lane0 GT Lane1 Image: Constraint of the selection OP Lane1 GT Lane0 Image: Constraint of the selection SATA Image: Constraint of the selection Image: Constraint of the selection			t					
✓ ✓ Display Port MIO 2730 ✓ → DPAUX MIO 2730 ✓ ✓ Lane Selection Dual Lower ✓ DP Lane0 GT Lane1 DP Lane1 GT Lane0 ✓ ✓ SATA ✓								
✓ Lane Selection Dual Lower DP Lane0 GT Lane1 DP Lane1 GT Lane0 ✓ ✓ SATA								
DP Lane0 GT Lane1 DP Lane1 GT Lane0 V SATA		> DPAUX	MIO 27 30	~				
DP Lane1 GT Lane0		✓ Lane Selection	Dual Lower	~				
DP Lane1 GT Lane0			GT Lanel					
SATA Lane0								
		V♥ SATA						
SATA Lane] GT Lane3 V		SATA Lane0						
		SATA Lanel	GT Lane3	~				
					1	_		

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





		Re-customiz	ze IP				:			
Zynq UltraScale+	- MPSoC (3.3)						4			
Documentation 🌣	Presets 📮 IP Location									
Page Navigator	I/O Configuration									
🗌 Switch To Adva	✓ MIO Voltage Standard									
PS UltraScale+ Bloc	Bank0 [MIO 0:25] Ban	k1 [MIO 26:51]	Bank2 [M	IO 52:77]	Bank3 [De	dicated]				
TS OIL ASCALET BIOL	LVCMOS18 V LVC	MOS18 V	LVCMOS1	8 ~	LVCMOS18	~				
I/O Configuration										
Clock Configuration	← Q ≭ ≑ ●									
DDR Configuration	Search: Q-									
	Peripheral	I/O		Signal		I/O Type	Drive Streng			
PS-PL Configuration	> Low Speed									
	✓ High Speed									
	> GEM									
	> USB									
	 ✓ ✓ PCIe 									
	> Rootport Mode Reset	MIO 31	~							
	Reset Polarity	Active Low	~							
	Lane Selection	x4	~							
	PCIe Lane0	GT Lane0								
	PCIe Lanel	GT Lanel								
	PCIe Lane2	GT Lane2								
	PCIe Lane3	GT Lane3								
	✓ Display Port									
	DPAUX									
	Lane Selection									
	✓ SATA									
	SATA Lane0									
	SATA Lanel									
	> Reference Clocks									
	<				_		;			
						ОК	Cancel			

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

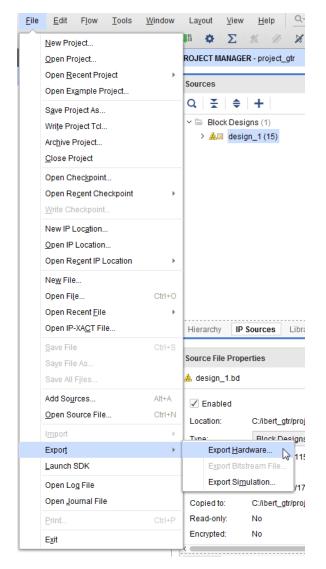


Sources			? _ 🗆 🖒 X		Project Sum	mary	
Q 素 ♦	+		٥		Settings	Edit	
V 🖨 Block Des					Project na		proj
> 🏯 des	ign_1		Source File Prop	erti		Ctrl+E	i
			Open File			Alt+O	1
			Copy Shared Log	gic	into Project		
			Report IP Status				5
			Create HDL Wra	ppe	er		
			View Instantiation	n T	emplate		
			Generate Output	Pr	oducts		
			Reset Output Pro	odu	icts		3
			Replace File				
			Copy File Into Pr	oje	ct		C
Hierarchy IP	Sources Libraries	İ.,	Copy All Files Int			Alt+I	
		×	Remove File from	n P	roject	Delete	,
Source File Prop	perties	-	Enable File			Alt+Equal	
Å design_1.bd			Disable File			Alt+Minus	
			Set File Type Set Used In				
Enabled							_
Location:	C:/ibert_gtr/project_gtr	-	Edit Constraints				
Type:	Block Designs	-	Edit Simulation S				
Part:	xczu9eg-ffvb1156-2-i	Ι.	Associate ELF Fi	lies	i		_1
Size:	43.3 KB	+	Add Sources	_		Alt+A	

- 18. Leave the option Let Vivado manage wrapper and auto-update selected. Click **OK** in the dialog to create the HDL wrapper.
- 19. Right-click **design_1_i** in the IP Sources tab, and click **Generate Output Products**.
- 20. Click Generate to generate with the default options in the panel.
- 21. After the generation is complete, click **OK**.
- 22. Select File → Export → Export Hardware.







- 23. In the Export Hardware Platform wizard, select a Fixed platform type. Click Next.
- 24. On the next page, select **Pre-Synthesis** for the platform output type.
- 25. Leave the XSA name as **design_1_wrapper**, and choose a location to store the exported XSA, preferably in a new directory.

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

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





Generating Using the Xilinx Software Command-line Tool Automated Flow

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

- 1. Copy the src/lab10/xsct_create_fsb1.tc1 script to the directory where the XSA file is located. You can modify the Tcl script if you changed the default name of the XSA file in the Vivado tool. You can also change the script if the compiler options need to be different.
- 2. Open a terminal on Linux or command prompt on Windows.
- 3. Change directory into the directory where the XSA file is located.
- 4. Call xsct from the Vitis tool install area.

% xsct xsct_create_fsbl.tcl

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

Step 3: ZCU102 Board Settings

USB Jumper Setting Requirements for HOST Mode on ZCU102

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







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

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





🍌 Open New Ha	rdware Target	×
	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
	temote server (target is on remote machine) 🗸	
Remote Server		
<u>H</u> ost name:	ibert-0 💿 🗸	
Port:	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>Next></u> Einish Can	cel

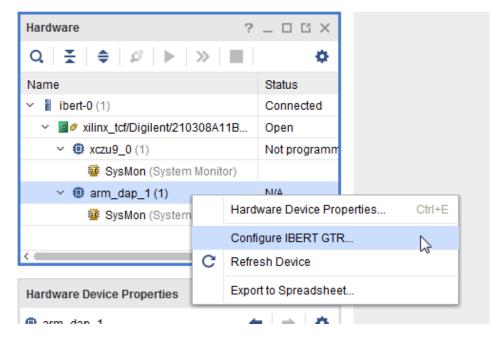
4. Verify the ARM_DAP is visible in the hardware device list and click **Next**, and then click **Finish**.

1	Open New H	ardware Targe	et					×
Se		target from the I		ble targets, then set the a select a different target.	ppropriate JTAG clock (TCK) frequency. If you	u do not see the	4
	Hardware <u>T</u> arg	ets						
	Туре	Name		JTAG Clock Frequency				
	xilinx_tcf	Digilent/210308	A11BFC	15000000 🗸 🗸				
				Add Xilinx V	irtual Cable (XVC)			
	Hardware <u>D</u> evi	ces (for unknow	n devices,	specify the Instruction R	egister (IR) length)			
	Name	ID Code	IR Length					
	xczu9_0	04738093	12					
	@ arm_dap_	1 5BA00477	4					
	Hardware serve	er: ibert-0:3121						
(?				< <u>B</u> ack	Next >	<u>F</u> inish	Cancel

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







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

🍐 Configure IBERT (GTR	×
A Zynq FSBL is required	d to configure IBERT GTR. The configuration file is optional.	4
<u>Z</u> ynq FSBL: <u>C</u> onfiguration File:	vroject_gtr/project_gtr.sdk/fsbl_design_1/Debug/fsbl_design_1.elf	
🗌 <u>R</u> eset Zyng	OK Can	cel

7. config_hw_sio_gts is executed with the selected settings. refresh_hw_device is then called to rescan the device for new debug cores. The IBERT should be configured as shown in the following example.



Vivado Lab Edition 2017.2			_	×
<u>Eile Edit T</u> ools <u>W</u> indow Layou	t <u>View H</u> elp <u>Qr Quick Access</u>			
	🚿 🖉 😹 Dashboard 🕶		📰 Serial I/O Analyzer	~
There are no serial I/O links. Auto-detect links.	inks Create links			
Hardware	? _ O C X	GT Properties	? _ 🗆 🖒 X	
Q ¥ ♦ ∅ ▶ ≫ ■	0	Pa L1	$\leftarrow \rightarrow \diamond $	
Name	Status	Name: ibert-0:3121/xilinx_tct/Digilent/210308A11BFC/1_1	O/DEDT/Ound_0/L1	
 ibert-0 (1) 	Connected	Status: Pcie 1 - Linked	_0/IBERT/QUAU_0/ET	
✓ ■● xilinx_tcf/Digilent/210308A11B	Open			
v @ xczu9_0 (1)	Not programmed	GT group: No Quad_0		
SysMon (System Monitor)		IBERT core: 19 IBERT		
@ arm_dap_1 (2)	N/A	Device: @ arm_dap_1		
SysMon (System Monitor)				
 IBERT (IBERT) 				
V N Quad_0 (4)				
Na L0	Pcie.0 - Linked			
Na L1	Pcie.1 - Linked			
Na L2	Usb0 - No Link (rx_det)			
Na L3	Sata1 - No Link (awaitcomwake)			
		General Properties PLLs		
 G IND: [Labtools 27-1455] Device current_hw_device [get_hw_devico refresh_hw_device -update_hw_pro- current_hw_device -update_hw_device 0 config_hw_sio_gts = dict [list " config_hw_sio_gts: Time (s): cp 0 refresh_hw_device -update_hw_pro- INFO: [Labtools 27-2302] Device 	es xczu9_0] obes fale [lindex [get_hw_devices xcz xczu9 (JTAG device index = 0) is not ; es arm_dap_1] obes fale [lindex [get_hw_devices arm es xczu9_0] FSBL" "C:/liert_gtr/project_gtr/projec u = 00:00:00 ; elapsed = 00:00:12 . Me obes fale [lindex [get_hw_devices arm arm_dap (JTAG device index = 1) is pr	<pre>programmed (DONE status = 0). _dap_1] 0] s_gtr.sdk/fsbl_design_1/Debug/fsbl_design_1.elf" "RESET_ mory (MS): peak = 800.699 ; gain = 0.000</pre>	? _ [^
				>
Type a Tcl command here				
T-11				

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

Hardware		? _ 🗆 🗳 ×	
$Q \mid \underbrace{\star} \mid \diamondsuit \mid \varnothing \mid \models \mid \gg \mid \blacksquare \mid$		0	
Name	Status		
✓ ■ ibert-0 (1)	Connected		
✓ ■● xilinx_tcf/Digilent/210308A11B	Open		
de xczu9_0 (1)	Not programmed		
SysMon (System Monitor)			
darm_dap_1 (2)	N/A		
SysMon (System Monitor)			
V 38 IBERT (IBERT)			
V Na Quad_0 (4)			
No.	Pcie.0 - Linked		
P⊲ L1	Pcie.1 - Linked		
Na L2	Usb0 - No Link (rx_det)		
N L3	Sata1 - No Link (awaitcominit)		
IBERT Core Properties		? _ O Ľ X	
IBERT		← → Ø	
Name: ibert-0:3121/xilinx_tcf/D Device: @ arm_dap_1 General Properties GT Groups	igilent/210308A11BFC/1_1_0/IBERT	Î	
Tcl Console Messages Serial I/O Lin	ks × Serial I/O Scans		
Q. ≍ ≑ 🛃			
Create Links Create Link Group Create Scan Create Sweep		Auto-detect links	or create links to add serial I/O links to this window.





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

Click OK when done.

TX GTs		RX GTs	
<u>S</u> earch: Q _*		<u>S</u> earch: Q-	
New Links			
+ -			
	ТХ	RX	
Description v1	TX L3/TX (arm_dap_1/Quad_0)	RX L3/RX (arm_dap_1/Quad_0)	
Description v ¹			
Description v ¹ So Link 3 So Link 2 So Link 1	L3/TX (arm_dap_1/Quad_0)	L3/RX (arm_dap_1/Quad_0)	
Description v ¹ SLink 3 Link 2	L3/TX (arm_dap_1/Quad_0) L2/TX (arm_dap_1/Quad_0)	L3/RX (arm_dap_1/Quad_0) L2/RX (arm_dap_1/Quad_0)	
Description v ¹ So Link 3 So Link 2 So Link 1	L3/TX (arm_dap_1/Quad_0) L2/TX (arm_dap_1/Quad_0) L1/TX (arm_dap_1/Quad_0)	L3/RX (arm_dap_1/Quad_0) L2/RX (arm_dap_1/Quad_0) L1/RX (arm_dap_1/Quad_0)	
Description v1 So Link 3 So Link 2 So Link 1	L3/TX (arm_dap_1/Quad_0) L2/TX (arm_dap_1/Quad_0) L1/TX (arm_dap_1/Quad_0)	L3/RX (arm_dap_1/Quad_0) L2/RX (arm_dap_1/Quad_0) L1/RX (arm_dap_1/Quad_0)	
Description v ¹ So Link 3 So Link 2 So Link 1	L3/TX (arm_dap_1/Quad_0) L2/TX (arm_dap_1/Quad_0) L1/TX (arm_dap_1/Quad_0) L0/TX (arm_dap_1/Quad_0)	L3/RX (arm_dap_1/Quad_0) L2/RX (arm_dap_1/Quad_0) L1/RX (arm_dap_1/Quad_0)	

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





Hardware				? _ 🗆 🖒 :		GT Properties					
Q ≚ ♦ ∅ ▶	· >>				¢.	№ L1			+	⇒ Φ	
Name			Status			Name: i	bert-0:3121/xilinx_	tcf/Digilent/210308	A11BFC/1_1_0/IBERT/	Quad_0/L1	
 ibert-0 (1) 			Connected			Status:	Pcie.1 - Linked				
✓ ✓ ✓ ✓ ×ilinx_tcf/Digilent/	210308/	\11B	Open			GT group:	Quad 0				
w @ xczu9_0 (1)			Not programmed			IBERT core:	IBERT				
SysMon (Sys		itor)					arm_dap_1				
@ arm_dap_1 (2)			N/A			Device.	@ ann_uap_1				
SysMon (Sys		iitor)									
Y 🦉 IBERT (IBER											
✓ № Quad_0 (4	4)										
P⊲ L0			Pcie.0 - Linked								
№ L1 № L2			Pcie.1 - Linked								
ka L3			Usb0 - No Link (nc_det) Sata1 - No Link (awaitcomini	it)							
№ L3	Seria	I I/O Lir	Sata1 - No Link (awaitcomini	lt)		General Pro	operties PLLs			? _	
Pa L3			Sata1 - No Link (awaitcomini ks × Serial I/O Scans	1)		General Pro	operties PLLs			? _	
Pa L3 Fcl Console Messages Q 궃 � ╋ ╋ Name		I I/O Lir RX	Sata1 - No Link (awaitcomini	II) TX Post-Cur:	sor	General Pro	RX PLL Status	TX PLL Status	TXUSERCLK Freq	? –	
Pa L3 Tcl Console Messages Q ★ ♦ + Name □ Ungrouped Links (0)			Sata1 - No Link (awaitcomini ks × Serial I/O Scans	TX Post-Curr		TX Diff Swing		TX PLL Status	TXUSERCLK Freq		
Roj L3 TCI Console Messages Q ¥ ♦ ↓ Name Dungrouped Links (0) Y ⊕ Link Group 0 (4)	TX	RX	Sata1 - No Link (awaitcomini aks × Serial VO Scans Status	TX Post-Curr Multiple	~	TX Diff Swing User Value 🗸	RX PLL Status			RXUSERCLK	Fred
Pg L3 Fcl Console Messages Q, ¥, ⊕ ↓ Same G Ungrouped Links (0) γ € Link Coup 0 (4) % Link 0		RX	Sata1 - No Link (awaitcomini iks x Serial I/O Scans Status Pcie.0 - Linked	TX Post-Cur Multiple User Value	~ ~	TX Diff Swing User Value ~ User Value ~	RX PLL Status	Locked	100.000	RXUSERCLK	Frec
Pg L3 Fcl Console Messages Q ★ ♦ ↓ Name P Unk Group ed Links (0) ~ ⊕ Link Group 0 (4) ~ ⊕ Link 1	TX L0/TX I L1/TX I	RX _0/RX _1/RX	Sata I - No Link (awaitcomini iks × Serial I/O Scans Status Pde.0 - Linked Pcie.1 - Linked	TX Post-Cur: Multiple User Value User Value	~ ~ ~	TX Diff Swing User Value ~ User Value ~ User Value ~	RX PLL Status Locked Not Locked	Locked Not Locked	100.000	RXUSERCLK 10 10	Fred
Pg L3 Fcl Console Messages Q, ¥, ⊕ ↓ Same G Ungrouped Links (0) γ € Link Coup 0 (4) % Link 0	TX L0/TX I L1/TX I L2/TX I	RX _0/RX _1/RX _2/RX	Sata1 - No Link (awaitcomini iks x Serial I/O Scans Status Pcie.0 - Linked	TX Post-Cur Multiple User Value	* * *	TX Diff Swing User Value ~ User Value ~	RX PLL Status Locked Not Locked Locked	Locked	100.000	RXUSERCLK	

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

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

cl Console Messages	Seri	al I/O Li	inks ×		Link Properties	CtrI+E
Q <u>∓</u> ≑ †				×	Delete Create Links	Delete
Name Dungrouped Links (0)	ТХ	RX	Status		Create Link Group	
Sink Group 0 (4)					Create Scan	le le
🗞 Link 0	L0/TX	L0/RX	Pcie.0 - L		Create Sweep	
S 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	-		

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

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



🔥 Create Sca	n		×						
Set the descrip on the selected		er properties to create and optionally run a scan	A						
Link: L	Link 1 (L1/TX, L1/RX)								
Description:	Scan 0								
Scan Propert	Scan Properties								
<u>S</u> can type:		2D Full Eyescan 🗸 🗸]						
<u>H</u> orizontal	increment:	4 ~]						
Horizontal	range:	-0.500 UI to 0.500 UI							
<u>V</u> ertical inc	crement:	4 ~							
V <u>e</u> rtical rar	nge:	100% ~]						
Dwell									
• <u>B</u> ER:	1e-5	~							
O <u>T</u> ime:		0 🌩							
<mark>√ R</mark> un scan			_						
?		OK Cance	il i						

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

Elle Edit Tools Window Layout Yiew Help Q- Outor Access Image: Comparison of the state of	
🕒 🛧 🛧 🗄 📗 🗙 🏟 🚿 🖉 💥 Dashboard -	
	Analyzer 🗸 🗸
Hardware ? _ D L X Scan Plots - Scan 0	? 🗆 🖾 ×
Q X ⇒ Q X C B 0 Contour (Filled) ✓	
Name Status Unit Interval	BER
Image: System Monitor) -1 -0.75 -0.25 0 0.25 0.76 1	DER
	7.9e-01
✓ Generation (Generation) ✓ # IBERT (BERT) 50:	1.0e-01
· • • • • • • • • • • • • • • • • • • •	
Pa L 0 Reio Da Linked 25-	5.0e-02
Ra L1 Pole.1 - United	1.0e-02
Pa L2 Usb0-No Link (nc, det)	5.0e-03
	1.0e-03
Scan Properties 7	5.0e-04
Scano + to	1.0e-04
	5.0e-05
Name: SCAN_0	1.0e-05
Description: Scan 0	-
Status: In Progress Summary Metrics Settings	
Link % Link1 Name: SCAN_0 Open area: 476 Link settings: N/A	
RX: GLIRX Description: Scan O Open U% 30.16 Horizontal increment 1	
Tx: De L/TX Stated: 2017/48/151528 Horizontal range: - 1.000 UIto 1.000 UI	
IX 2017 mar 10 1000 cm 2017 mar 2	
Lines 2017 Hep-10103142 Vence and enterior 1 Vence 100%	
General Properties	
Tcl Console Messages Serial VO Links Serial VO Scans x	? _ 🗆 🖸
	II BER Dwell
✓ E Scans (1)	
Scan 0 Link 1 2d_full_eye Done 100% 476 30.16 1 ~ 100% ~ BER 1e-5	~
<	>



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

Troubleshooting

Known Issues

- 1. By default, FSBL does not enumerate USB as that is something Linux drivers would do. To put USB in link state without Linux, a small modification is required in the FSBL C-code. This modification still does not enumerate the device, it only brings the USB into link state.
- 2. The EyeScan does not have a built-in time-out mechanism. If your link is poor (for example, if L*_TM_DIG_8.EYESURF_ENABLE != 1), the EyeScan will hang without providing a user. No results are returned in this case.
- 3. If the EyeScan progress is not moving, make sure the below parameters for all lanes are set for EyeScan to function correctly. Note that * represents the lane number (as in, for Lane 0 the parameter would be L0).

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

- a. L*_TM_MISC3.CDR_EN_FPL = 0
- b. L*_TM_MISC3.CDR_EN_FFL = 0
- c. L*_TM_DIG_8.EYESURF_ENABLE = 1

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

- d. L*_PLL_LOCK = 1
- e. L*_TM_SAMP_STATUS4.E_SAMP_PH0_CALIB_CODE is non-zero value
- f. L*_TM_SAMP_STATUS5.E_SAMP_PH180_CALIB_CODE is non-zero value

Notes

- 1. As mentioned in Assumptions, IBERT PS-GTR uses the psu_cortexr5_0 core, so no other applications should use this core.
- 2. TCMO and TCM1 memory are combined to form a unified memory for IBERT PS-GTR. Any other processor core should not access this memory while IBERT PS-GTR is running.
- 3. The error counter is 16 bits and the sample counter is 32 bits. Each sample can have 8 bits of error count. Therefore on the edges, the error counter can saturate with a sample count value of 8192. PS-GTR does not stop the sample counter even if the error counter saturates. A prescale=0 produces 8192 samples and thus a total samples of 8192 *8 (65536) and thus the outside edges of eye could show a BER of e-01 or less depending on the prescale selected.





- 4. The EyeScan assumes there is link present. If there is no link, the EyeScan may not complete. Canceling the EyeScan stops the command sequence, but the state of the previous point scan will be unknown.
- 5. If you run EyeScan and because of no link the EyeScan does not complete, set the register L*_TM_MISC_ST_0.EYE_SURF_RUN to 0 for the given lane before you run the EyeScan again.
- 6. If you run EyeScan on a lane that is either powered down or Display Port, it will immediately stop and the scan will be marked as incomplete. EyeScan will not work in either scenario.





Appendix A

Additional Resources and Legal Notices

Xilinx Resources

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

Documentation Navigator and Design Hubs

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

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

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

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

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





Please Read: Important Legal Notices

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

AUTOMOTIVE APPLICATIONS DISCLAIMER

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





Copyright

© Copyright 2012-2021 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. All other trademarks are the property of their respective owners.

