See all versions of this document

Vivado Design Suite Tutorial

Programming and Debugging

UG936 (v2019.2) December 10, 2019

XILINX®

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
12/10/2019 Version 2019.2	
General updates.	Updated for Vivado 2019.2
5/22/2019 Version 2019.1	
General updates.	Editorial updates only. No technical content changes.

Table of Contents

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)

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.

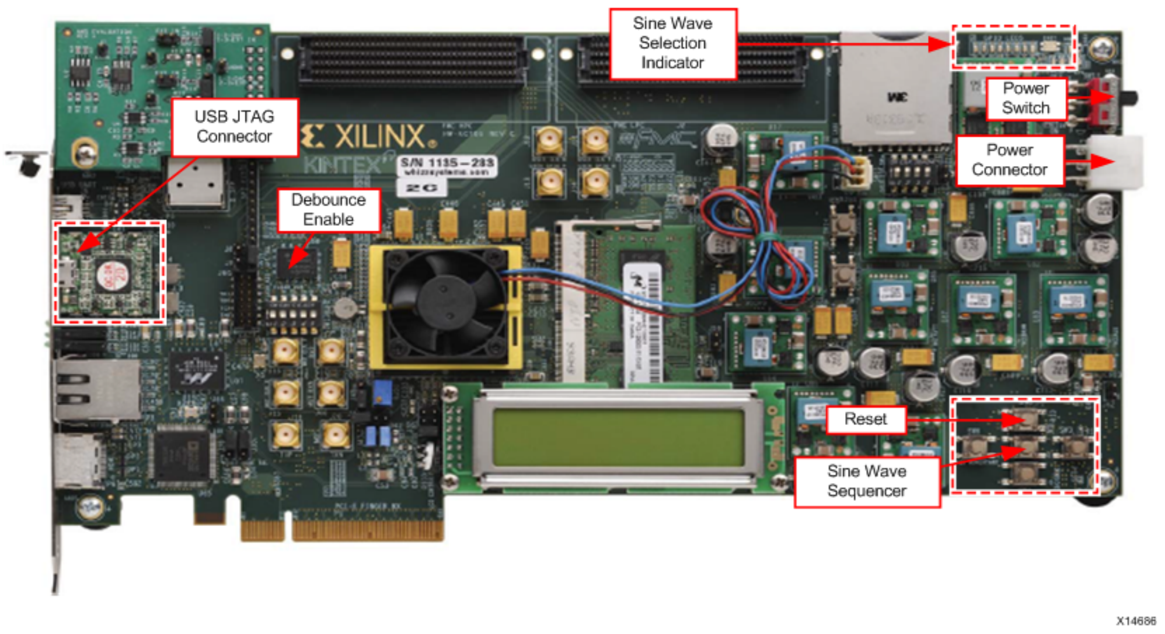
Software

Vivado® Design Suite 2019.2

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:** In this example, when enabled, provides a clean pulse or transition from high to low. Eliminates a series of spikes or glitches when a button is pressed and released.
- **Sine Wave Sequencer State Machine:** Captures and decodes input from the two push buttons. Provides sine wave selection and indicator circuits, sequencing among 00, 01, 10, and 11 (zero to three).
- **LED Displays:** GPIO_LED_0 and GPIO_LED_1 display selection status from the state machine outputs, each of which represents a different sine wave frequency: high, medium, and low.

Lab 5 includes:

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

Board Support and Pinout Information

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

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

Design Files

1. In your C: drive, create a folder called `/Vivado_Debug`.
2. Download the 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.

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.

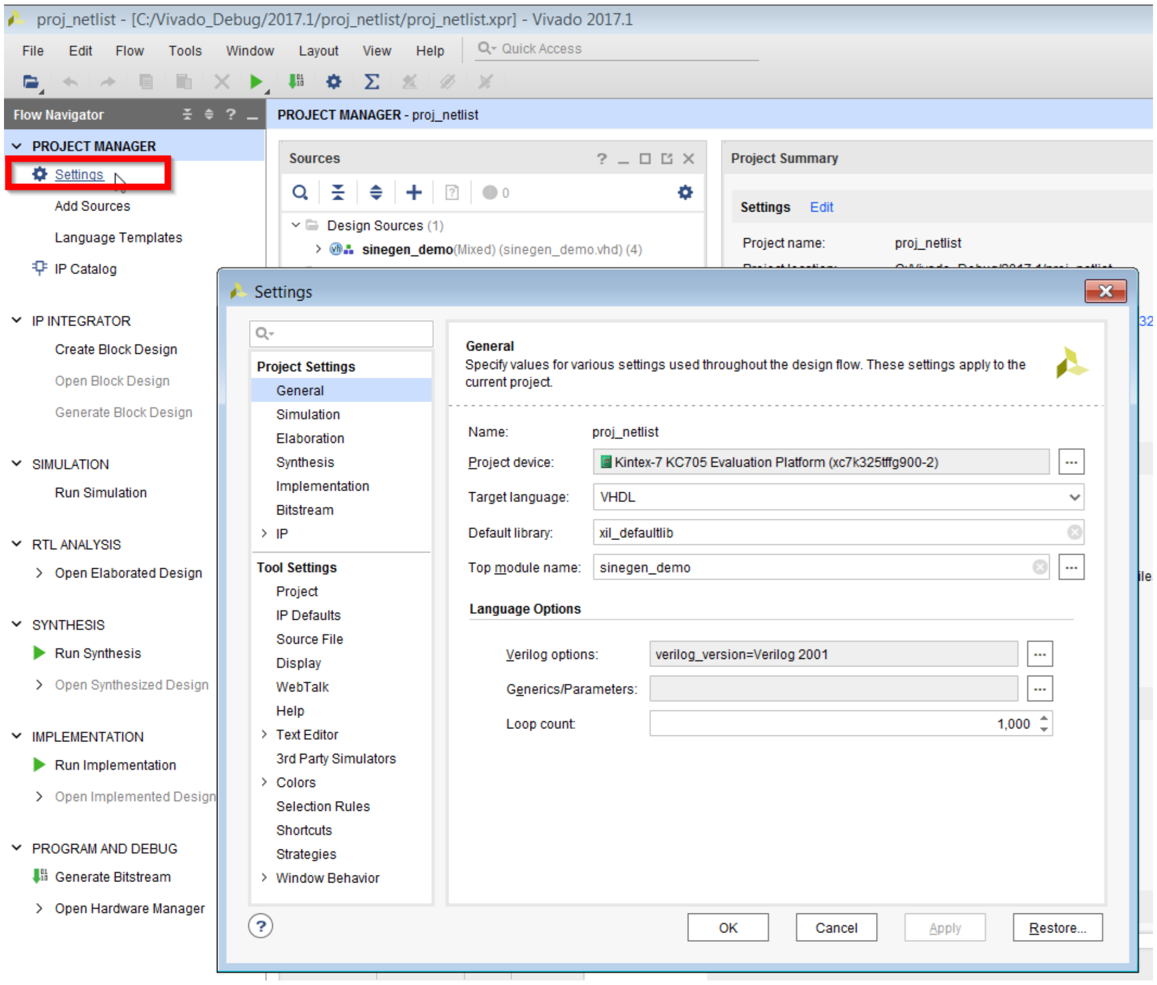
1. Invoke the Vivado® IDE.
2. In the Getting Started page, click **Create Project** to start the New Project wizard. Click **Next**.
3. In the Project Name page, name the new project proj_netlist and provide the project location (`C:/Vivado_Debug`). Ensure that **Create Project Subdirectory** is selected 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.



General
Specify values for various settings used throughout the design flow. These settings apply to the current project.

Name: proj_netlist
Project device: Kintex-7 KC705 Evaluation Platform (xc7k325tffg900-2)
Target language: VHDL
Default library: xil_defaultlib
Top module name: sinegen_demo

Language Options

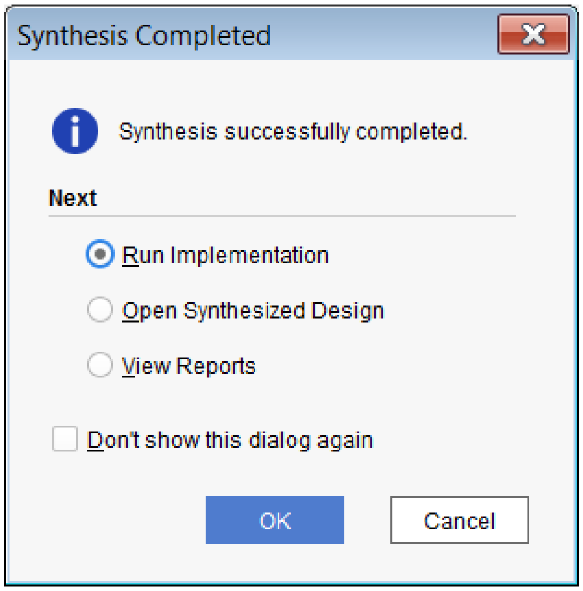
Verilog options: verilog_version=Verilog 2001
Generics/Parameters:
Loop count: 1,000

> **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.

Next

[✓ Run Implementation]

Open Synthesized Design

View Reports

☐ Don't show this dialog again

OK Cancel

Step 3: Probing and Adding Debug IP

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

In this step, you will accomplish the following tasks:

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

Adding Debug Nets to the Project

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

- Add MARK_DEBUG attribute to HDL files.
  - **VHDL:**

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

  - **Verilog:**

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

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

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

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

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

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

  This applies the MARK_DEBUG on the current, open netlist.

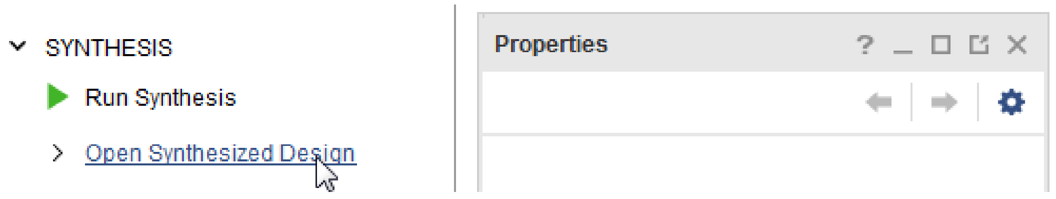
  This method is flexible since you can turn MARK_DEBUG on and off by modifying the Tcl command. In addition, this method does not require HDL source modification. However, there may be situations where synthesis does not preserve the signals due to netlist optimization involving absorption or merging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

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

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

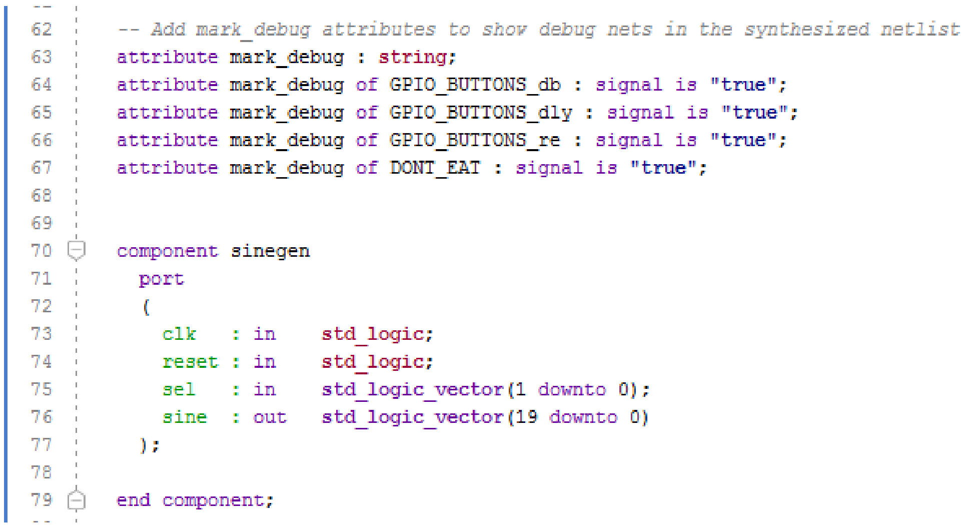
Use Ctrl-Q to toggle it off and on.

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

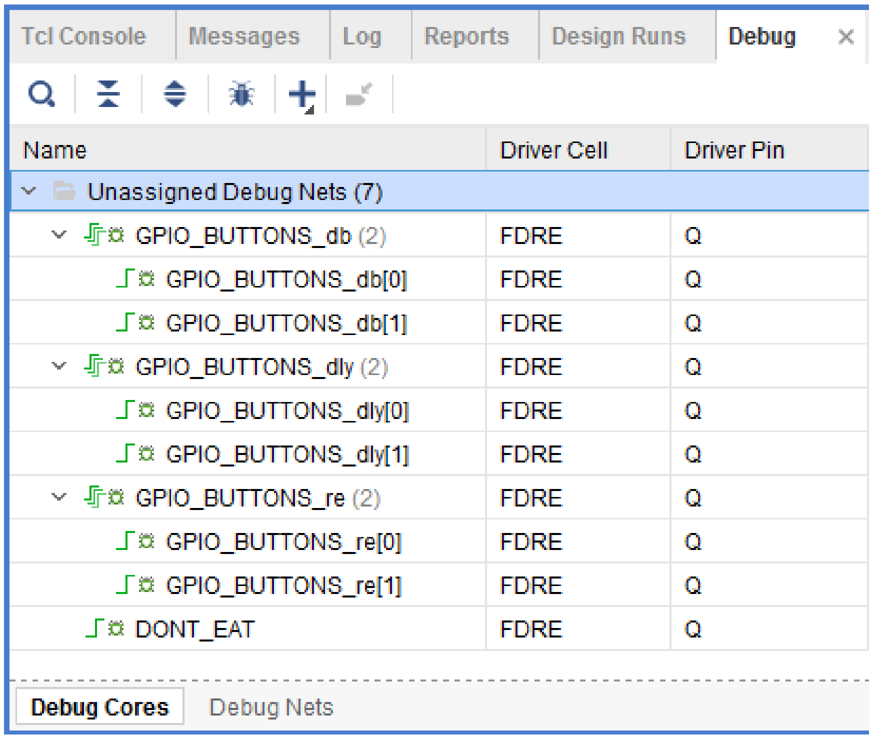


2. In the Window drop-down menu, select **Debug**. When the Debug window opens, click the window if it is not already selected.

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`.



```
-- 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";
attribute mark_debug of DONT_EAT : signal is "true";


component sinegen
  port
  (
    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;
```

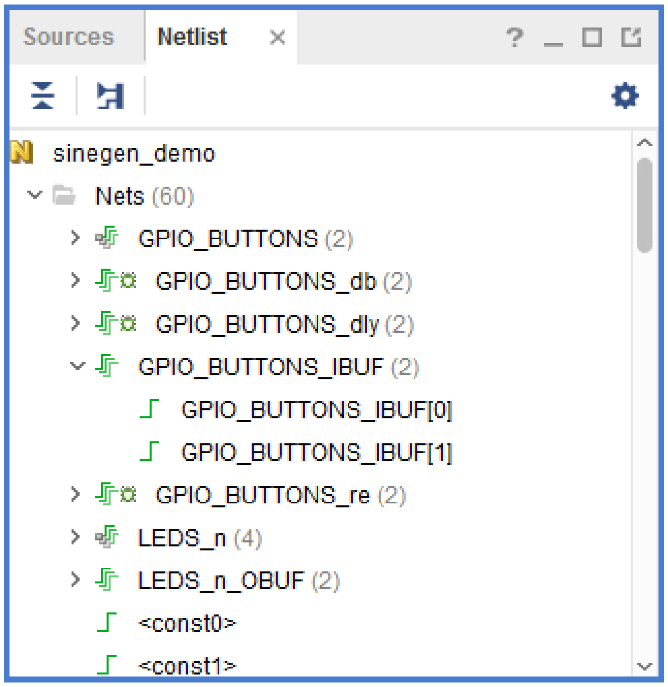


Name	Driver Cell	Driver Pin
Unassigned Debug Nets (7)		
GPIO_BUTTONS_db (2)	FDRE	Q
GPIO_BUTTONS_db[0]	FDRE	Q
GPIO_BUTTONS_db[1]	FDRE	Q
GPIO_BUTTONS_dly (2)	FDRE	Q
GPIO_BUTTONS_dly[0]	FDRE	Q
GPIO_BUTTONS_dly[1]	FDRE	Q
GPIO_BUTTONS_re (2)	FDRE	Q
GPIO_BUTTONS_re[0]	FDRE	Q
GPIO_BUTTONS_re[1]	FDRE	Q
DONT_EAT	FDRE	Q

4. In the Netlist window, select the Netlist tab and expand Nets. Select the following nets for debugging as shown in the following figure.

   - GPIO_BUTTONS_IBUF[0] and GPIO_BUTTONS_IBUF[1] - Nets folder under the top-level hierarchy
   - sel(2) - Nets folder under the U_SINEGEN hierarchy

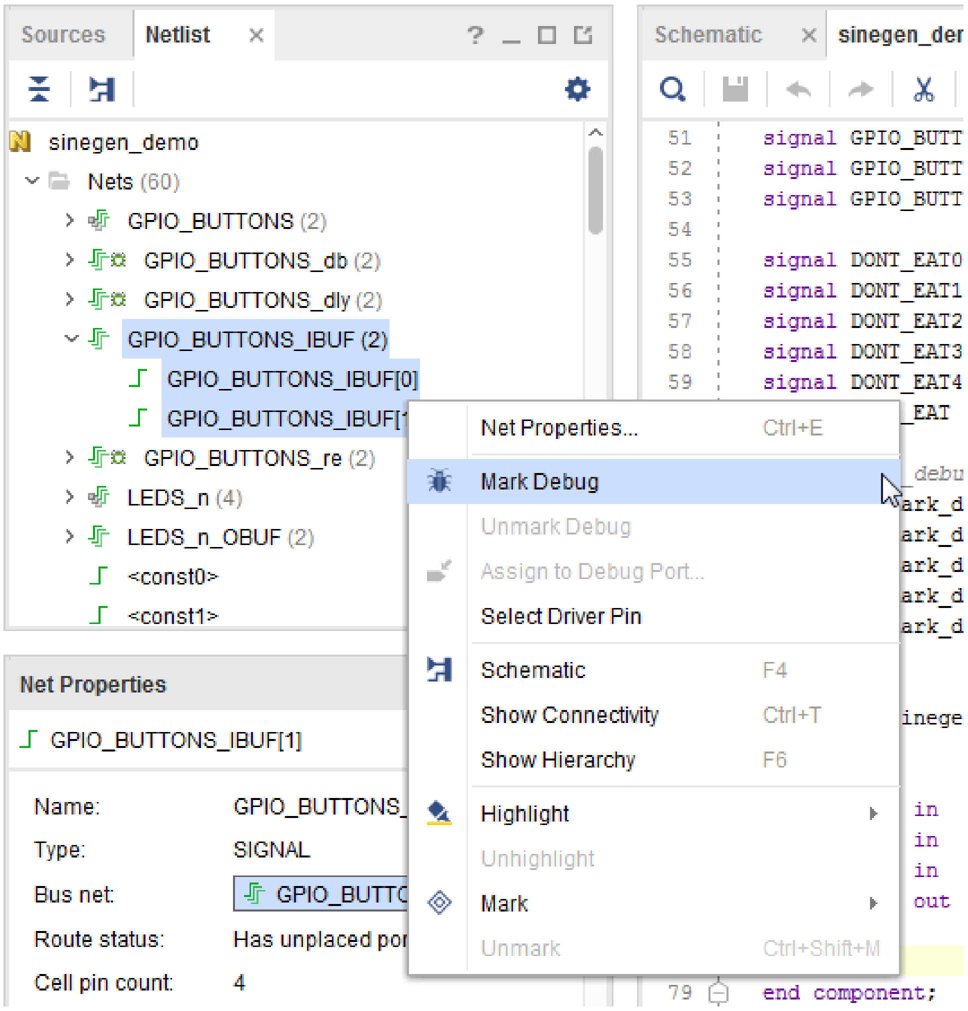
- sine(20)- Nets folder under the U_SINEGEN hierarchy



sinegen_demo
Nets (60)
GPIO_BUTTONS (2)
GPIO_BUTTONS_db (2)
GPIO_BUTTONS_dly (2)
GPIO_BUTTONS_IBUF (2)
GPIO_BUTTONS_IBUF[0]
GPIO_BUTTONS_IBUF[1]
GPIO_BUTTONS_re (2)
LEDS_n (4)
LEDS_n_OBUF (2)
<const0>
<const1>

***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.



GPIO_BUTTONS_IBUF[1]

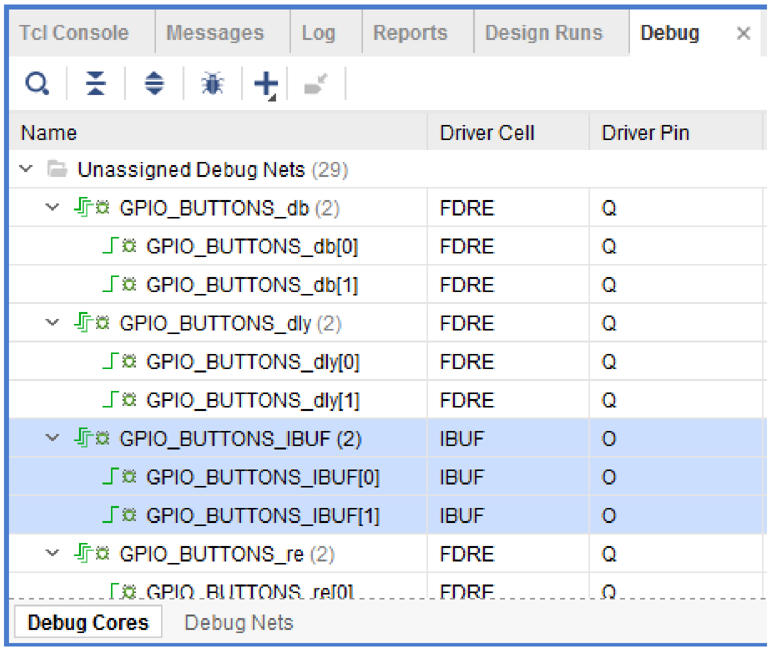
Name:	GPIO_BUTTONS_
Type:	SIGNAL
Bus net:	GPIO_BUTTC
Route status:	Has unplaced por
Cell pin count:	4

```
signal GPIO_BUTT
signal GPIO_BUTT
signal GPIO_BUTT

signal DONT_EAT0
signal DONT_EAT1
signal DONT_EAT2
signal DONT_EAT3
signal DONT_EAT4
       _EAT
...
end component;
```

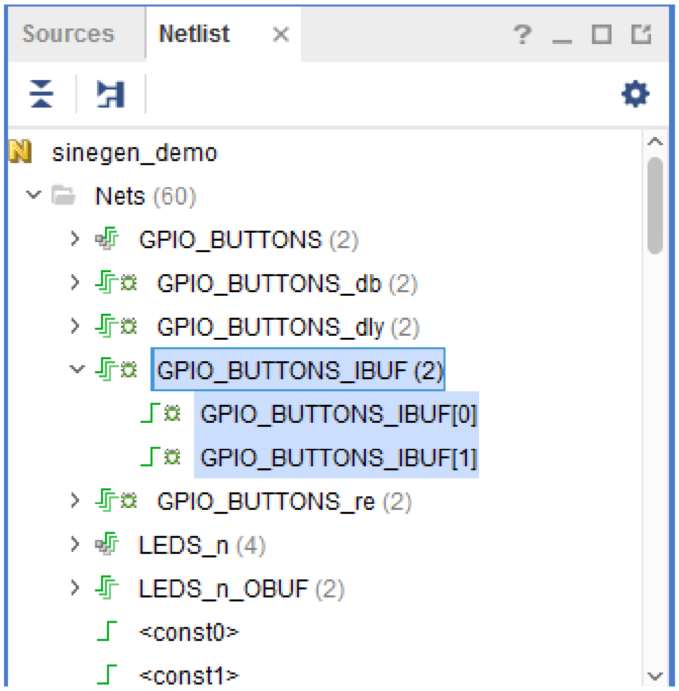
6. Next, mark nets for debug in the Tcl console. Mark nets “sine(20)” under the U_SINEGEN hierarchy for debug by executing the following Tcl command.

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

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



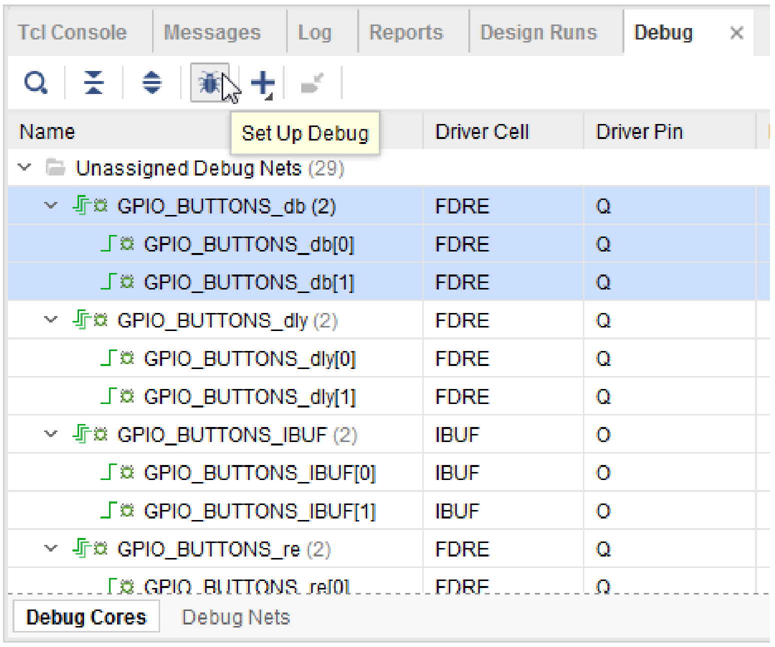
Name	Driver Cell	Driver Pin
Unassigned Debug Nets (29)		
GPIO_BUTTONS_db (2)	FDRE	Q
GPIO_BUTTONS_db[0]	FDRE	Q
GPIO_BUTTONS_db[1]	FDRE	Q
GPIO_BUTTONS_dly (2)	FDRE	Q
GPIO_BUTTONS_dly[0]	FDRE	Q
GPIO_BUTTONS_dly[1]	FDRE	Q
GPIO_BUTTONS_IBUF (2)	IBUF	O
GPIO_BUTTONS_IBUF[0]	IBUF	O
GPIO_BUTTONS_IBUF[1]	IBUF	O
GPIO_BUTTONS_re (2)	FDRE	Q
GPIO_BUTTONS_re[0]	FDRE	Q



```
sinegen_demo
  Nets (60)
    GPIO_BUTTONS (2)
    GPIO_BUTTONS_db (2)
    GPIO_BUTTONS_dly (2)
    GPIO_BUTTONS_IBUF (2)
      GPIO_BUTTONS_IBUF[0]
      GPIO_BUTTONS_IBUF[1]
    GPIO_BUTTONS_re (2)
    LEDS_n (4)
    LEDS_n_OBUF (2)
    <const0>
    <const1>
```

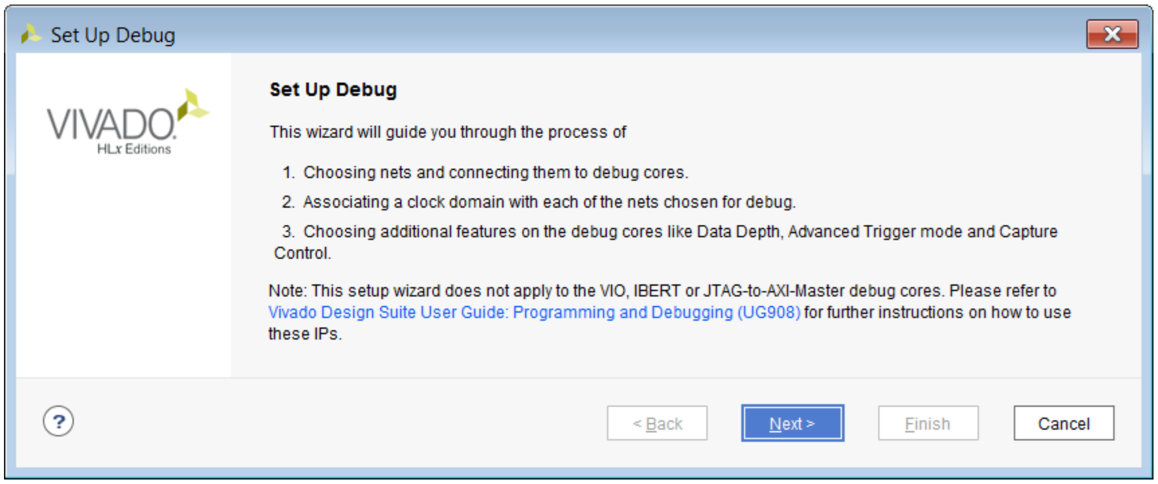
Running the Set Up Debug Wizard

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



Name	Driver Cell	Driver Pin
Unassigned Debug Nets (29)		
GPIO_BUTTONS_db (2)	FDRE	Q
GPIO_BUTTONS_db[0]	FDRE	Q
GPIO_BUTTONS_db[1]	FDRE	Q
GPIO_BUTTONS_dly (2)	FDRE	Q
GPIO_BUTTONS_dly[0]	FDRE	Q
GPIO_BUTTONS_dly[1]	FDRE	Q
GPIO_BUTTONS_IBUF (2)	IBUF	O
GPIO_BUTTONS_IBUF[0]	IBUF	O
GPIO_BUTTONS_IBUF[1]	IBUF	O
GPIO_BUTTONS_re (2)	FDRE	Q
GPIO_BUTTONS_re[0]	FDRE	Q

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



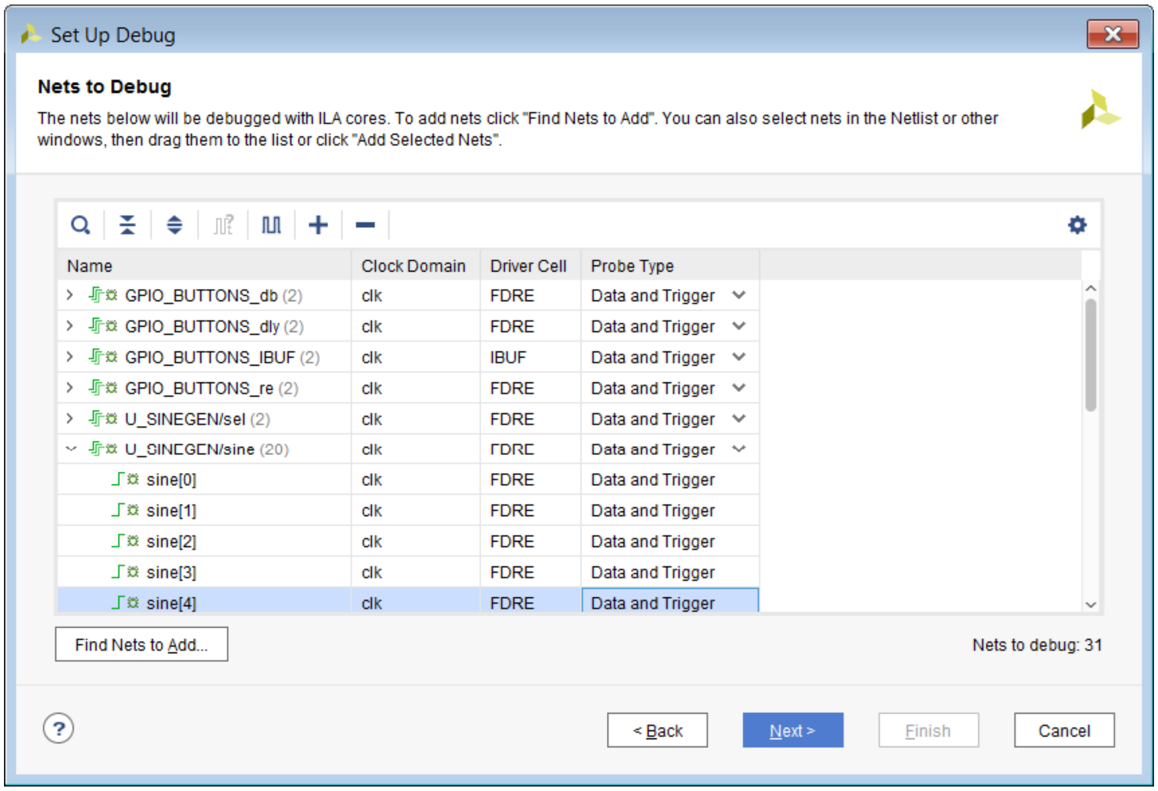
Set Up Debug

This wizard will guide you through the process of

1. Choosing nets and connecting them to debug cores.
2. Associating a clock domain with each of the nets chosen for debug.
3. 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.

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**.



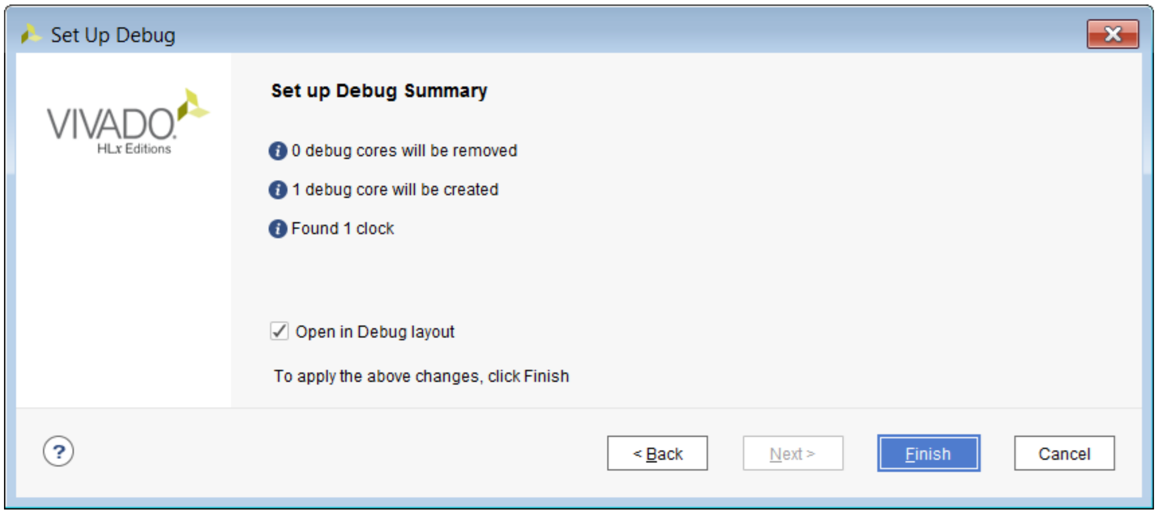
Nets to Debug

The nets below will be debugged with ILA cores. To add nets click "Find Nets to Add". You can also select nets in the Netlist or other windows, then drag them to the list or click "Add Selected Nets".

Name	Clock Domain	Driver Cell	Probe Type
GPIO_BUTTONS_db (2)	clk	FDRE	Data and Trigger
GPIO_BUTTONS_dly (2)	clk	FDRE	Data and Trigger
GPIO_BUTTONS_IBUF (2)	clk	IBUF	Data and Trigger
GPIO_BUTTONS_re (2)	clk	FDRE	Data and Trigger
U_SINEGEN/sel (2)	clk	FDRE	Data and Trigger
U_SINEGEN/sine (20)	clk	FDRE	Data and Trigger
sine[0]	clk	FDRE	Data and Trigger
sine[1]	clk	FDRE	Data and Trigger
sine[2]	clk	FDRE	Data and Trigger
sine[3]	clk	FDRE	Data and Trigger
sine[4]	clk	FDRE	Data and Trigger

Nets to debug: 31

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 Summary

0 debug cores will be removed

1 debug core will be created

Found 1 clock

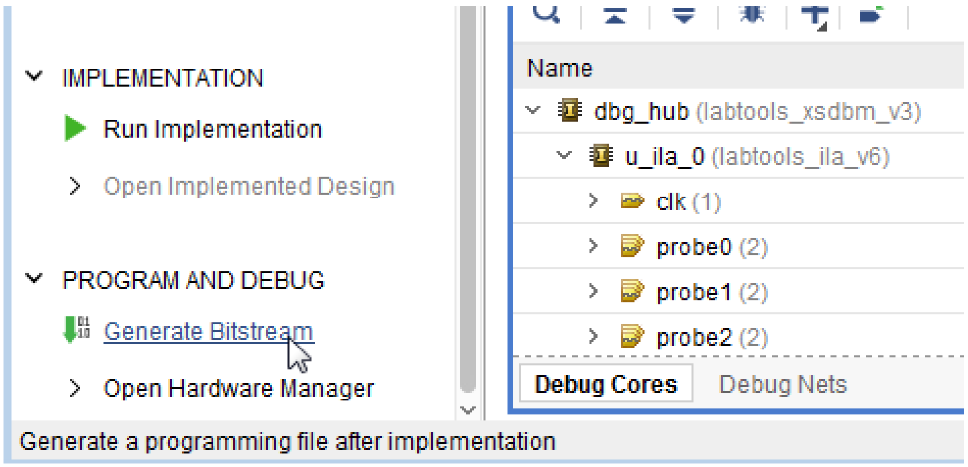
☑ Open in Debug layout

To apply the above changes, click Finish

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**.



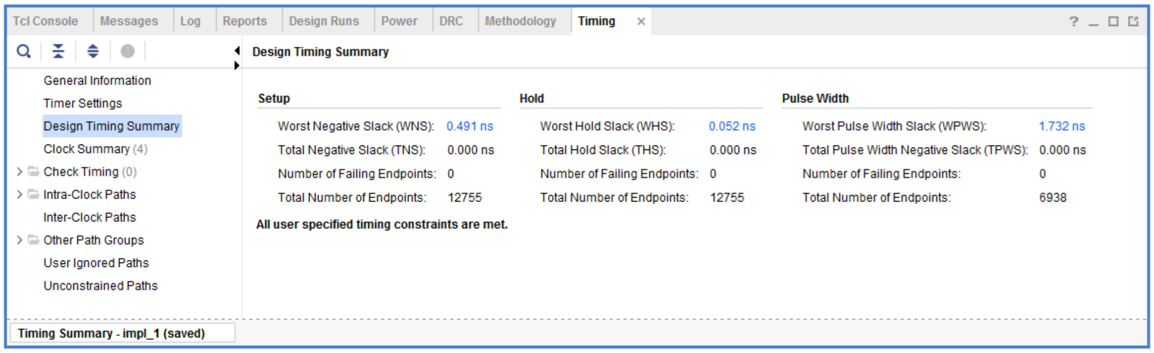
2. In the Save Project dialog box click **Save**. If a dialog box appears indicating this will cause the Synthesis results to go out of date, click **OK**. This applies the MARK_DEBUG attributes on the newly marked nets. You can see those constraints by inspecting the `sinegen_demo_kc705.xdc` file.

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 to close synthesized design before opening implemented design. Click **Yes**.

6. Examine the Timing Summary report to ensure that all the specified timing constraints are met.



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.491 ns	Worst Hold Slack (WHS):	0.052 ns	Worst Pulse Width Slack (WPWS):	1.732 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	12755	Total Number of Endpoints:	12755	Total Number of Endpoints:	6938

All user specified timing constraints are met.

Proceed to Chapter 6: 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

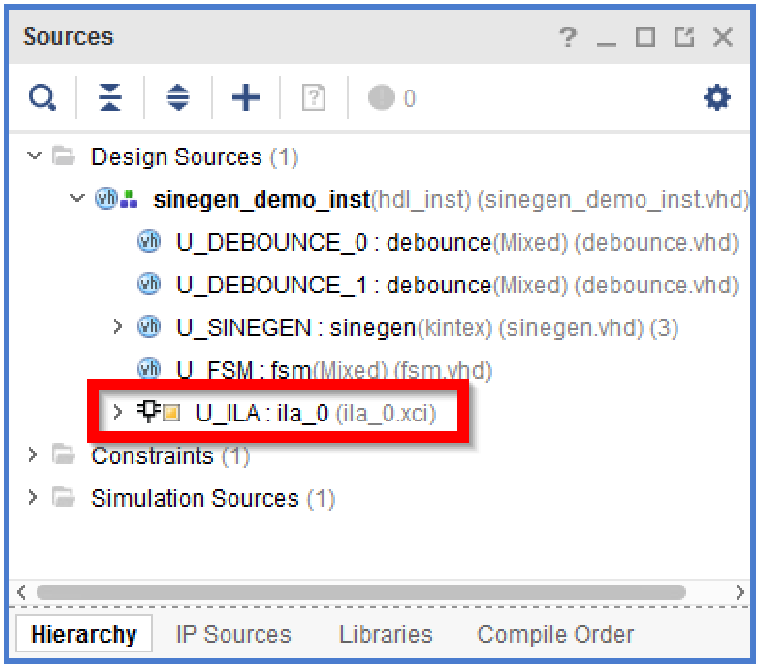
The HDL Instantiation method is one of the two methods supported in the Vivado® tool debug probing. For this flow, you will generate an ILA IP using the Vivado IP Catalog and instantiate the core in a design manually as you would with any other IP.

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.



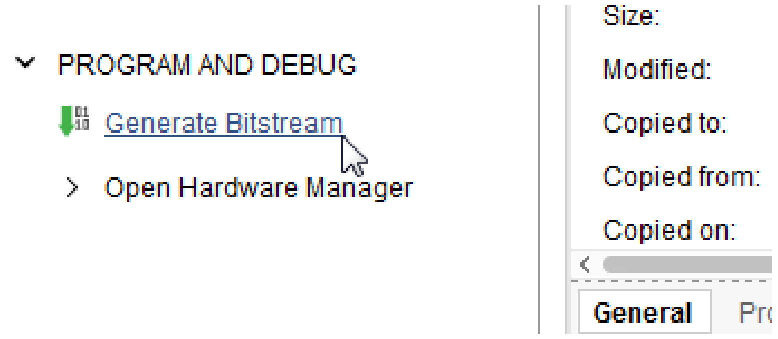
```
Design Sources (1)
  sinegen_demo_inst(hdl_inst) (sinegen_demo_inst.vhd)
    U_DEBOUNCE_0 : debounce(Mixed) (debounce.vhd)
    U_DEBOUNCE_1 : debounce(Mixed) (debounce.vhd)
    U_SINEGEN : sinegen(kintex) (sinegen.vhd) (3)
    U_FSM : fsm(Mixed) (fsm.vhd)
    U_ILA : ila_0 (ila_0.xci)
Constraints (1)
Simulation Sources (1)
```

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.

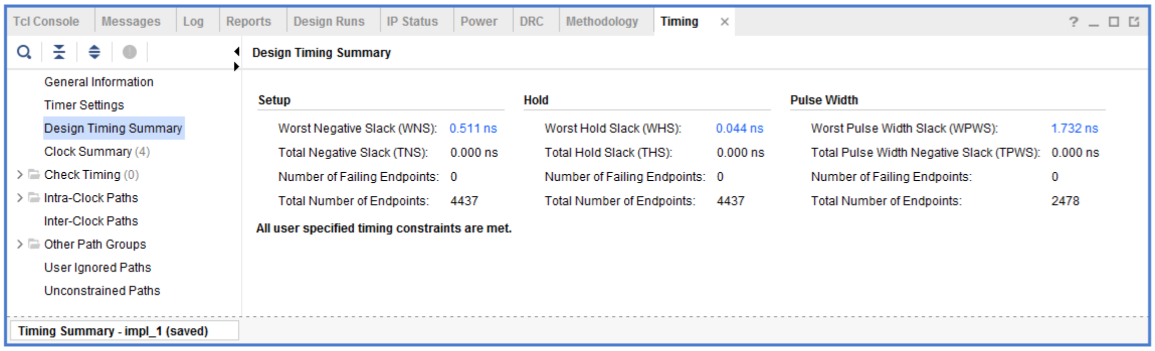
```
------------------------------------------------------------
-- ILA
------------------------------------------------------------
U_ILA : ila_0
  port map
  (
    CLK => clk,
    PROBE0 => sineSel,
    PROBE1 => sine,
    PROBE2 => GPIO_BUTTONS_db,
    PROBE3 => GPIO_BUTTONS_re,
    PROBE4 => GPIO_BUTTONS_dly,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



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.



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.511 ns	Worst Hold Slack (WHS):	0.044 ns	Worst Pulse Width Slack (WPWS):	1.732 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	4437	Total Number of Endpoints:	4437	Total Number of Endpoints:	2478

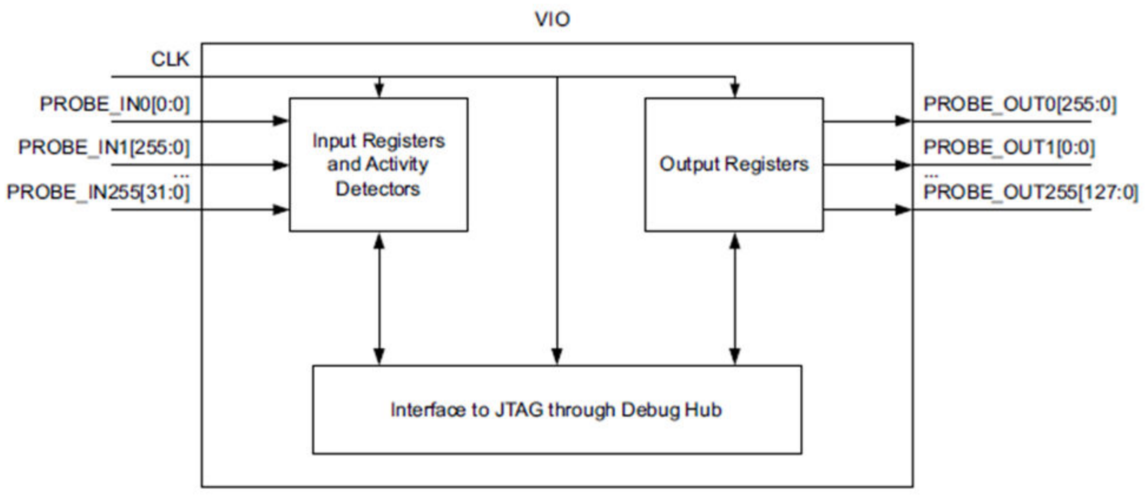
All user specified timing constraints are met.

5. Proceed to Chapter 6: 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_IN0 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. We 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.

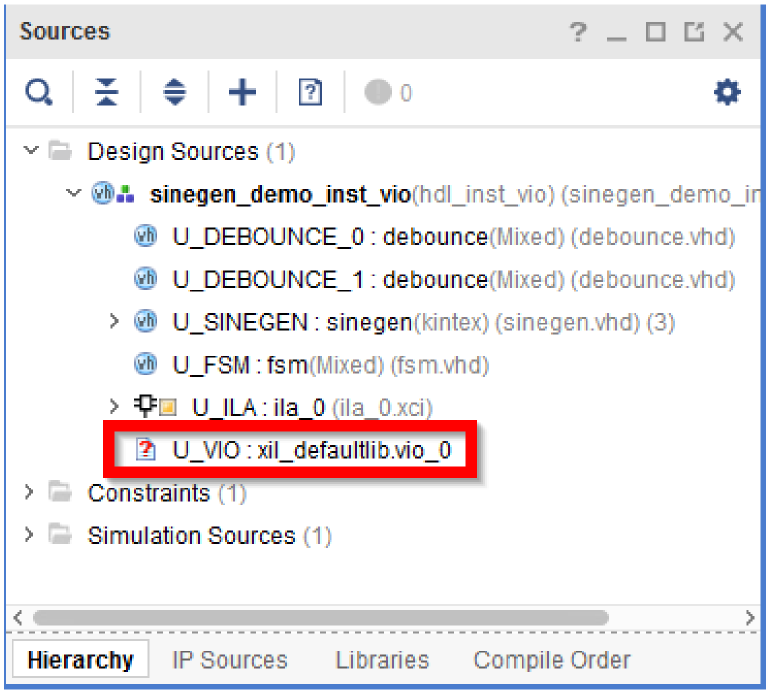
Step 1: Creating a Project with the Vivado New Project Wizard

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**.

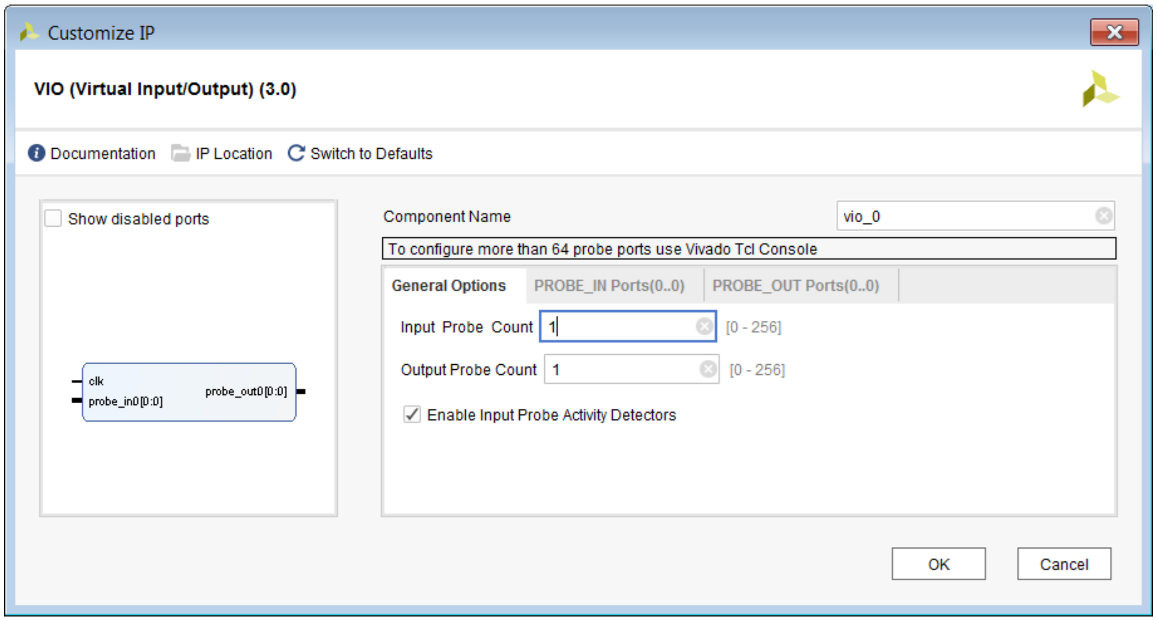
    Note: It might take a moment for the project to initialize.

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.


Sources
Design Sources (1)
sinegen_demo_inst_vio(hdl_inst_vio) (sinegen_demo_in
U_DEBOUNCE_0 : debounce(Mixed) (debounce.vhd)
U_DEBOUNCE_1 : debounce(Mixed) (debounce.vhd)
U_SINEGEN : sinegen(kintex) (sinegen.vhd) (3)
U_FSM : fsm(Mixed) (fsm.vhd)
U_ILA : ila_0 (ila_0.xci)
U_VIO : xil_defaultlib.vio_0
Constraints (1)
Simulation Sources (1)

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.

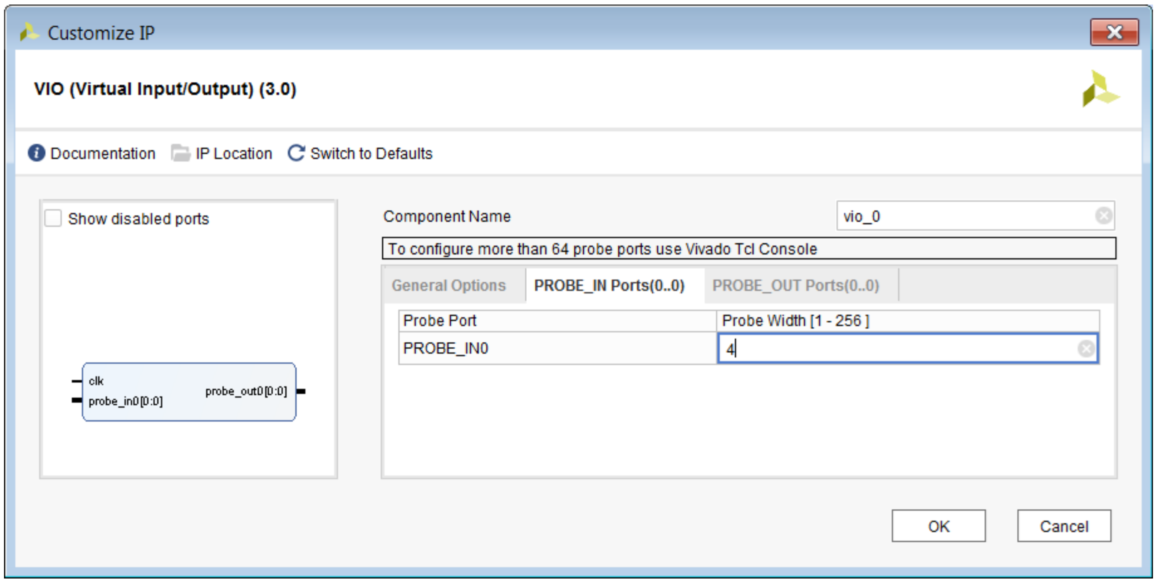
15. On the General Options tab, leave the Component Name as its default value of vio_0, set Input Probe Count to **1**, Output Probe Count to **1**, and select the **Enable Input Probe Activity Detectors** check box.


Customize IP
VIO (Virtual Input/Output) (3.0)

Component Name: vio_0
To configure more than 64 probe ports use Vivado Tcl Console

Input Probe Count: 1 [0 - 256]
Output Probe Count: 1 [0 - 256]
✓ Enable Input Probe Activity Detectors

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



Customize IP

VIO (Virtual Input/Output) (3.0)

Documentation | IP Location | Switch to Defaults

Show disabled ports

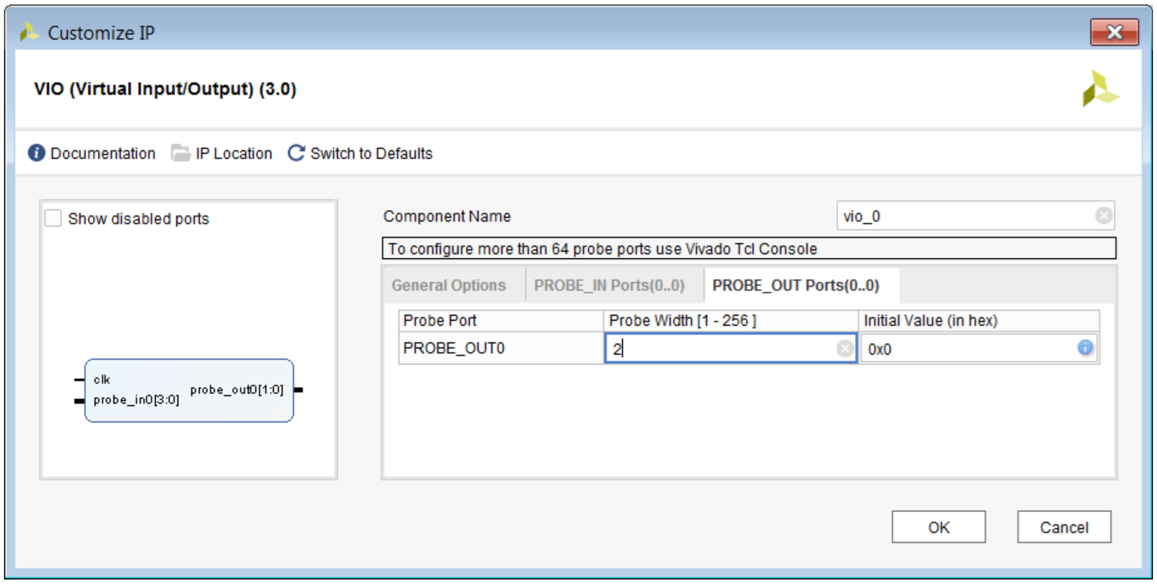
Component Name: vio_0

To configure more than 64 probe ports use Vivado Tcl Console

General Options | PROBE_IN Ports(0..0) | PROBE_OUT Ports(0..0)

Probe Port	Probe Width [1 - 256]
PROBE_IN0	4

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)

Documentation | IP Location | Switch to Defaults

Show disabled ports

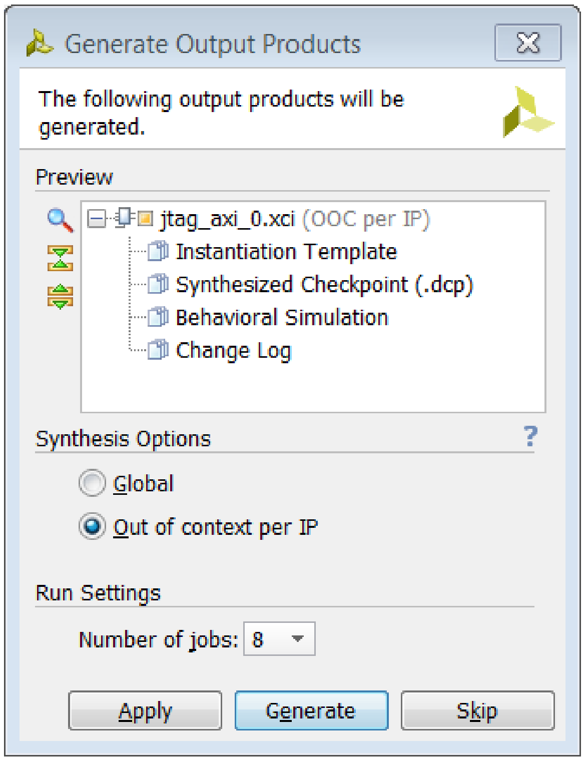
Component Name: vio_0

To configure more than 64 probe ports use Vivado Tcl Console

General Options | PROBE_IN Ports(0..0) | PROBE_OUT Ports(0..0)

Probe Port	Probe Width [1 - 256]	Initial Value (in hex)
PROBE_OUT0	2	0x0

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

jtag_axi_0.xci (OOC per IP)
- Instantiation Template
- Synthesized Checkpoint (.dcp)
- Behavioral Simulation
- Change Log

Synthesis Options

Global [✓ Out of context per IP]

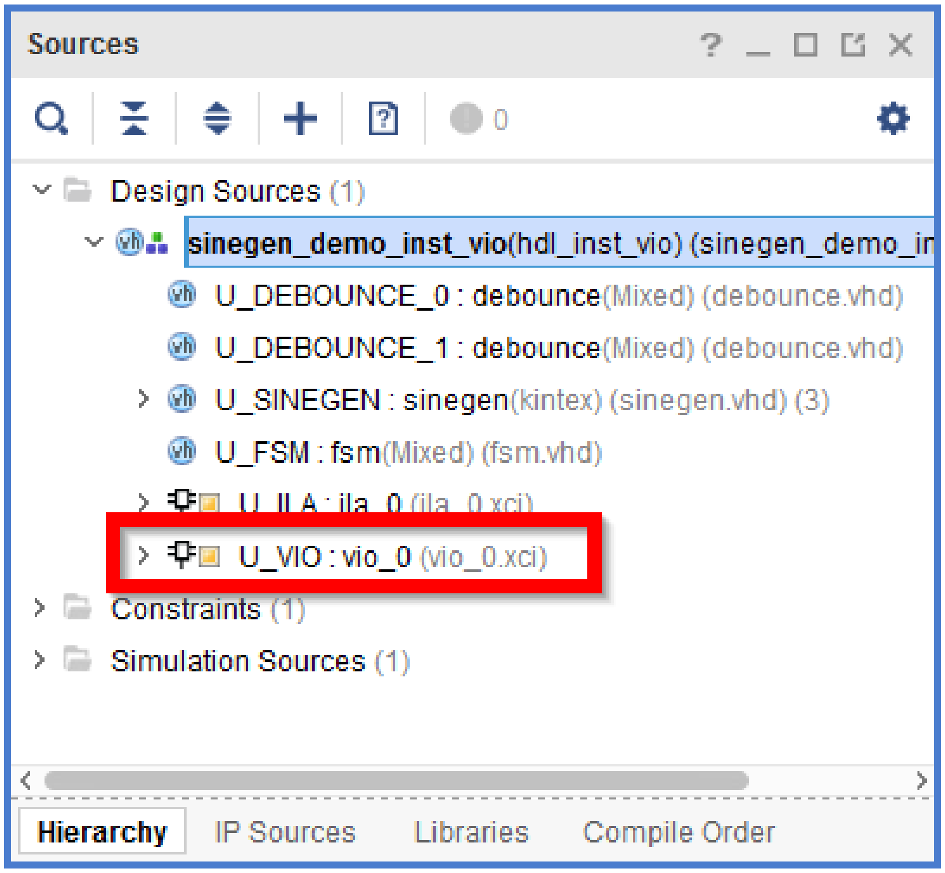
Run Settings

Number of jobs: 8

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.

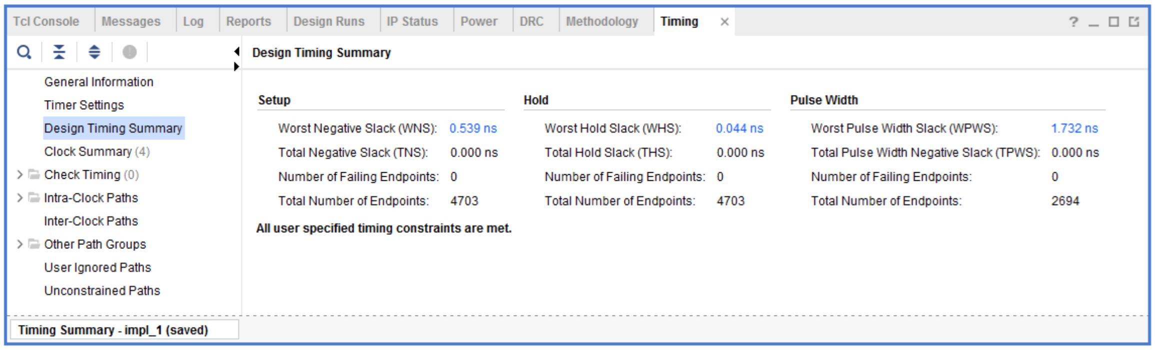


Design Sources (1)
sinegen_demo_inst_vio(hdl_inst_vio) (sinegen_demo_in
U_DEBOUNCE_0 : debounce(Mixed) (debounce.vhd)
U_DEBOUNCE_1 : debounce(Mixed) (debounce.vhd)
U_SINEGEN : sinegen(kintex) (sinegen.vhd) (3)
U_FSM : fsm(Mixed) (fsm.vhd)
U_ILA : ila_0 (ila_0.xci)
U_VIO : vio_0 (vio_0.xci)
Constraints (1)
Simulation Sources (1)

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.



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.539 ns	Worst Hold Slack (WHS):	0.044 ns	Worst Pulse Width Slack (WPWS):	1.732 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	4703	Total Number of Endpoints:	4703	Total Number of Endpoints:	2694

All user specified timing constraints are met.

5. Proceed to Chapter 6: 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.

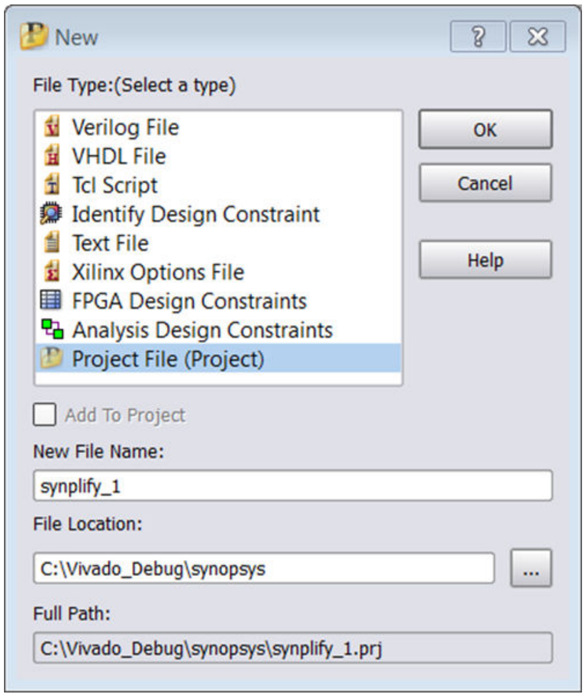
Lab 4: Using the Synplify Pro Synthesis Tool and Vivado Design Suite to Debug a Design

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→New**.
2. Set File Type to **Project File (Project)** as highlighted in the following figure.
3. In the New File Name box, enter **synplify_1**.
4. Click **OK**.

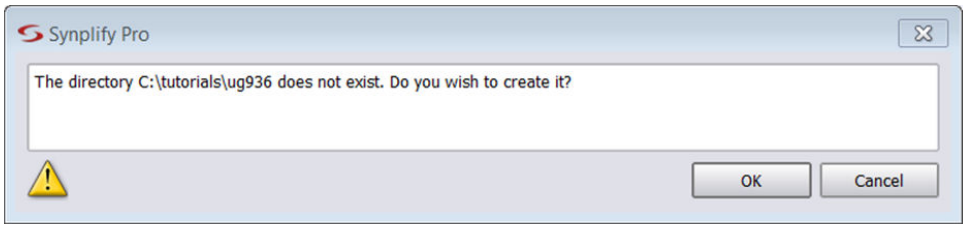


New File Name: synplify_1

File Location: C:\Vivado_Debug\synopsys

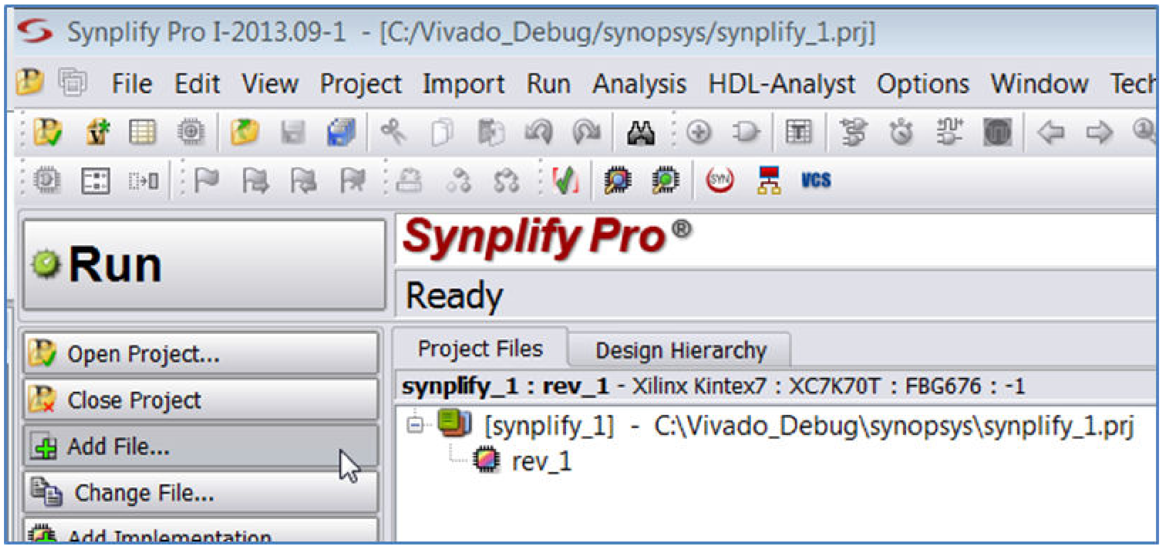
Full Path: C:\Vivado_Debug\synopsys\synplify_1.prj

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



The directory C:\tutorials\ug936 does not exist. Do you wish to create it?

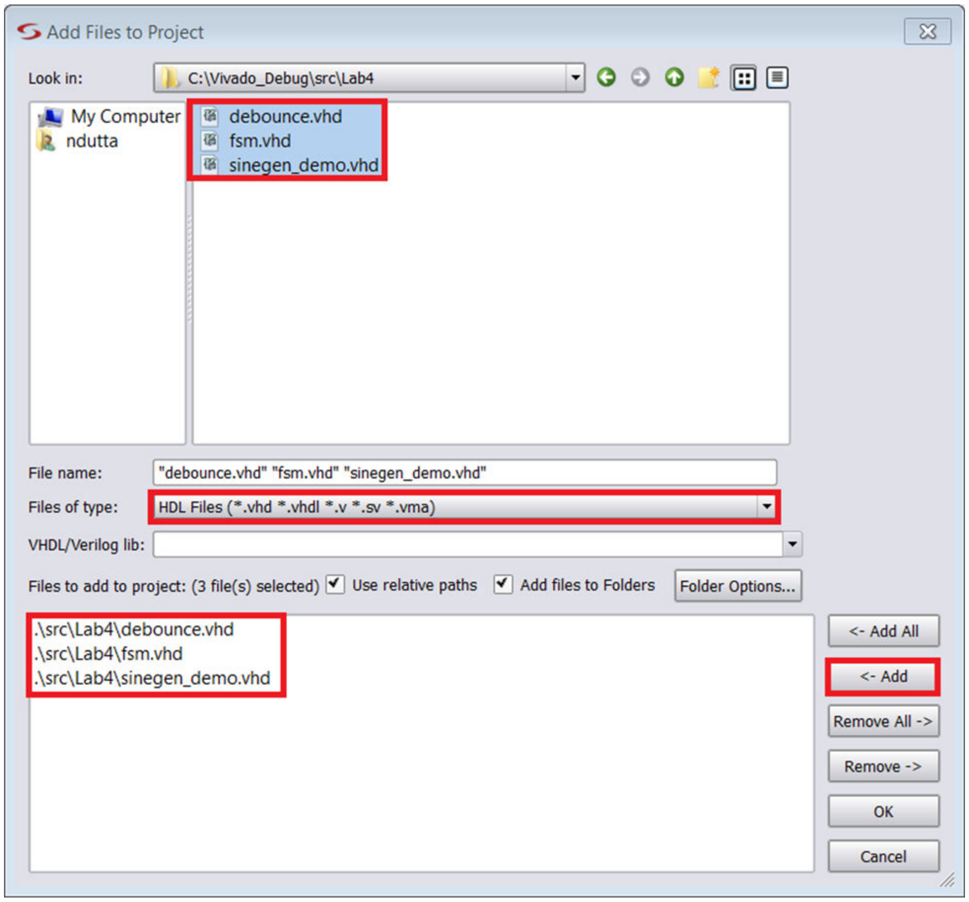
6. In the left panel of the Synplify Pro window, click **Add File** as shown in the following figure.



synplify_1 : rev_1 - Xilinx Kintex7 : XC7K70T : FBG676 : -1

[synplify_1] - C:\Vivado_Debug\synopsys\synplify_1.prj
rev_1

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**.



Add Files to Project

Look in: C:\Vivado_Debug\src\Lab4

My Computer
ndutta

debounce.vhd
fsm.vhd
sinegen_demo.vhd

File name: "debounce.vhd" "fsm.vhd" "sinegen_demo.vhd"

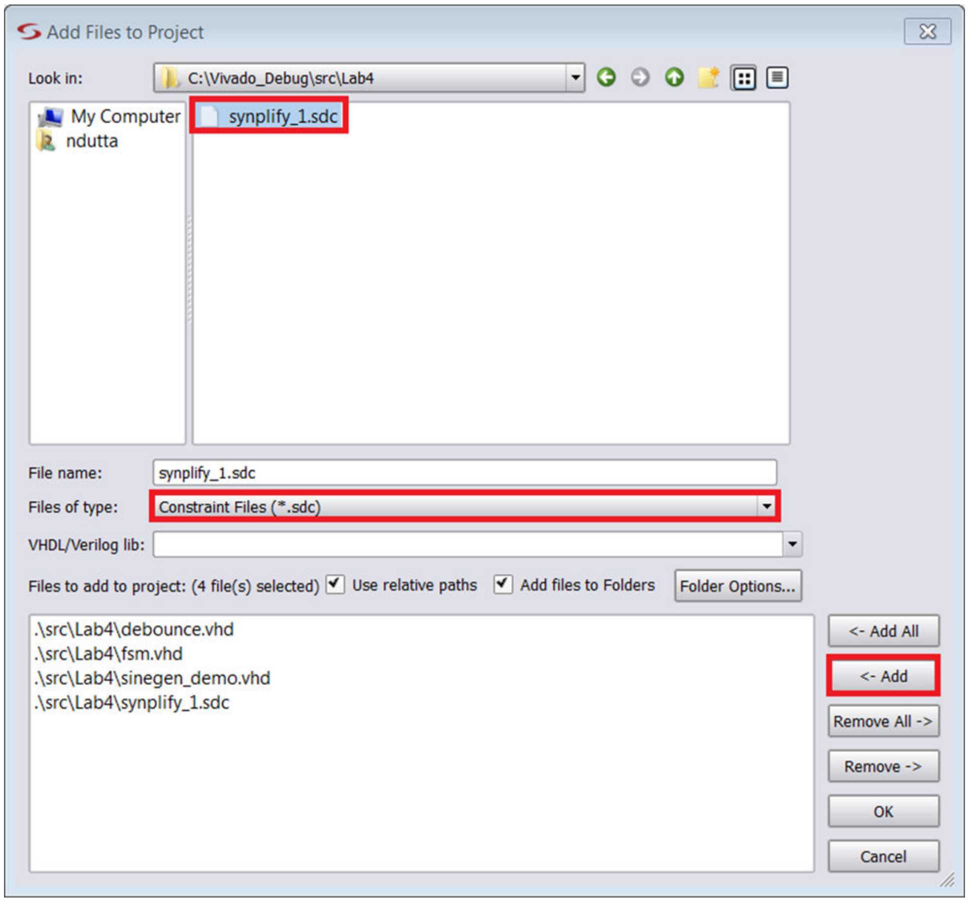
Files of type: HDL Files (*.vhd *.vhdl *.v *.sv *.vma)

VHDL/Verilog lib:

Files to add to project: (3 file(s) selected) ☑ Use relative paths ☑ Add files to Folders

.\src\Lab4\debounce.vhd
.\src\Lab4\fsm.vhd
.\src\Lab4\sinegen_demo.vhd

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.



Look in: C:\Vivado_Debug\src\Lab4

synplify_1.sdc

File name: synplify_1.sdc

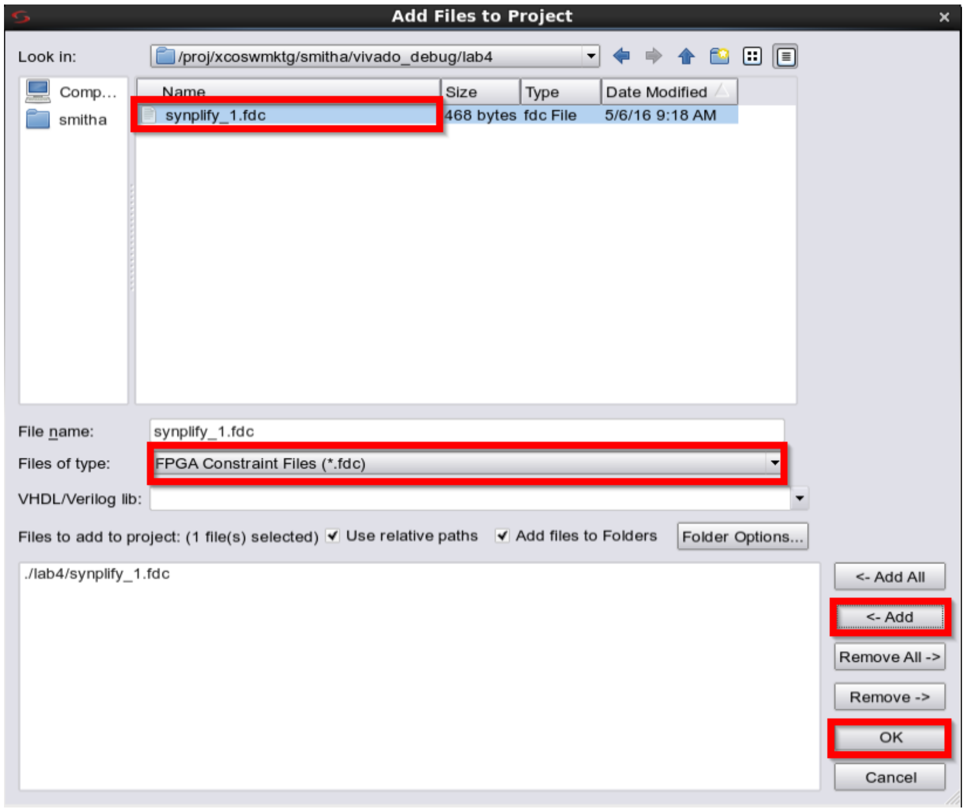
Files of type: Constraint Files (*.sdc)

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

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**.

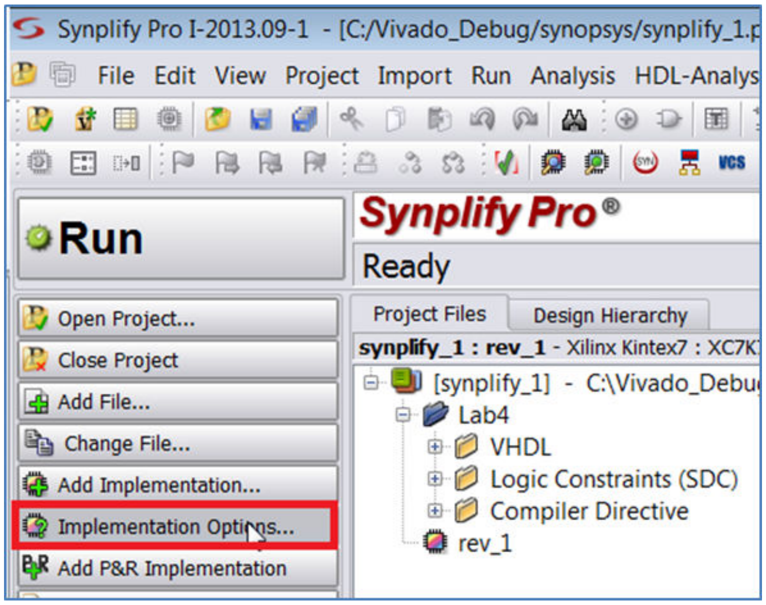


Name	Size	Type	Date Modified
synplify_1.fdc	468 bytes	fdc File	5/6/16 9:18 AM

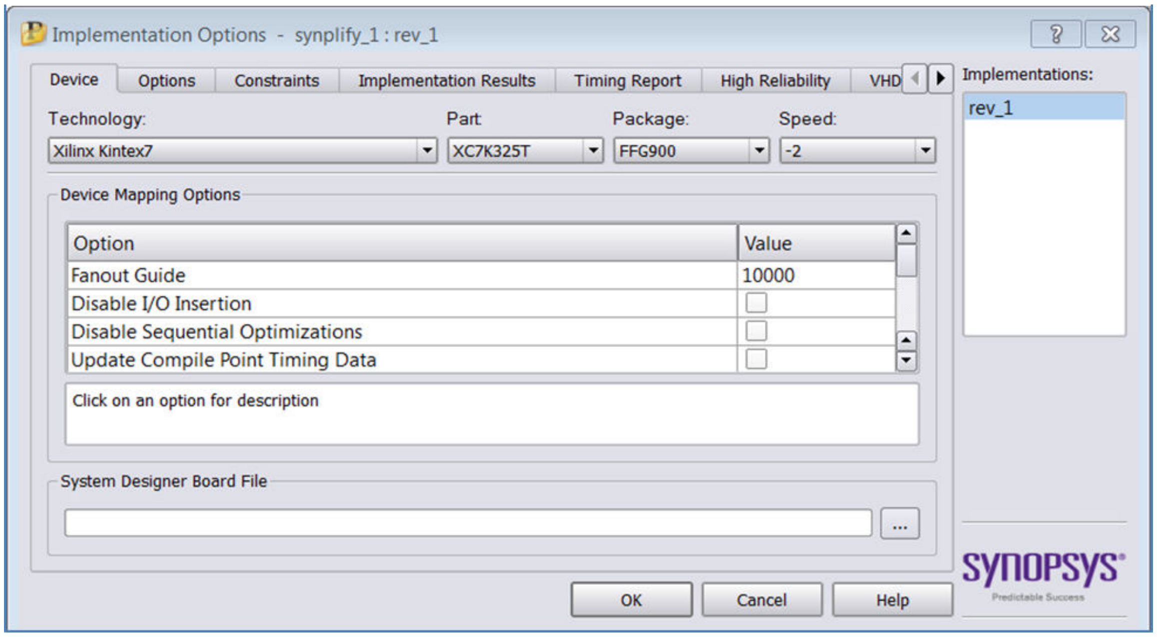
File name: synplify_1.fdc
Files of type: FPGA Constraint Files (*.fdc)
VHDL/Verilog lib:
Files to add to project: (1 file(s) selected) ✓ Use relative paths ✓ Add files to Folders

./lab4/synplify_1.fdc

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


Synplify Pro I-2013.09-1 - [C:/Vivado_Debug/synopsys/synplify_1.p
File Edit View Project Import Run Analysis HDL-Analys
Run
Synplify Pro®
Ready
Open Project...
Close Project
Add File...
Change File...
Add Implementation...
Implementation Options...
Add P&R Implementation
Project Files
Design Hierarchy
synplify_1 : rev_1 - Xilinx Kintex7 : XC7K
[synplify_1] - C:\Vivado_Debu
Lab4
VHDL
Logic Constraints (SDC)
Compiler Directive
rev_1

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**.


Implementation Options - synplify_1 : rev_1

Technology: Xilinx Kintex7 | Part: XC7K325T | Package: FFG900 | Speed: -2

Device Mapping Options

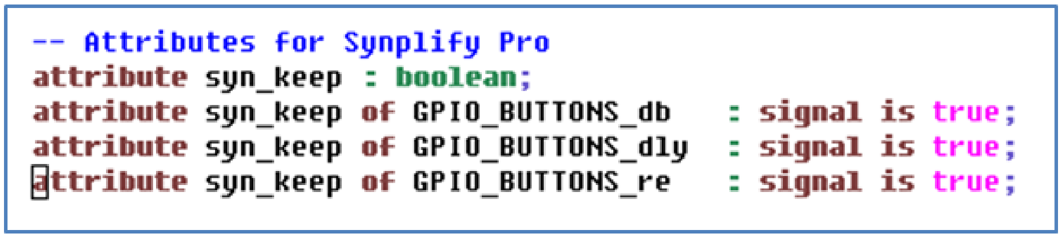
Option	Value
Fanout Guide	10000
Disable I/O Insertion	☐
Disable Sequential Optimizations	☐
Update Compile Point Timing Data	☐

Click on an option for description

System Designer Board File

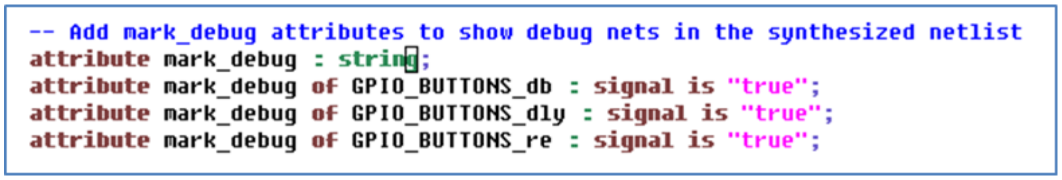
Implementations: rev_1

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.



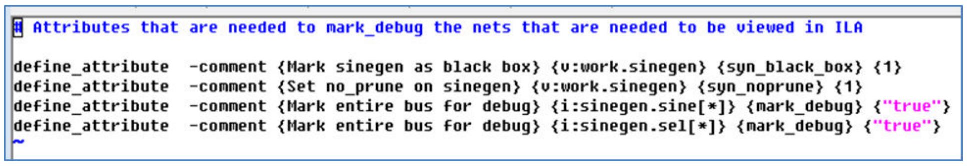
```
-- Attributes for Synplify Pro
attribute syn_keep : boolean;
attribute syn_keep of GPIO_BUTTONS_db   : signal is true;
attribute syn_keep of GPIO_BUTTONS_dly  : signal is true;
attribute syn_keep of GPIO_BUTTONS_re   : signal is true;
```

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.

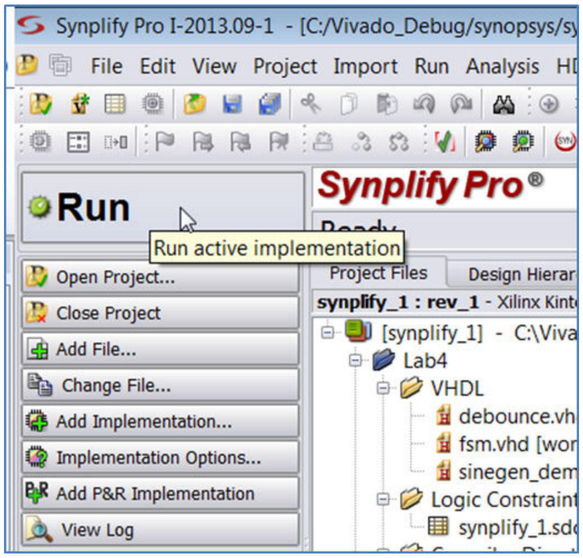
Step 2: Synthesize the Synplify Project

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→View Result File**.

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

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

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

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

Step 4: Create a Post Synthesis Project in Vivado IDE

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

5. Under Add Netlist Sources, click **Add Files**, navigate to the `Vivado_Debug/synopsys/rev_1` folder, and select `sinegen_demo.edf`. Click **OK**.
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

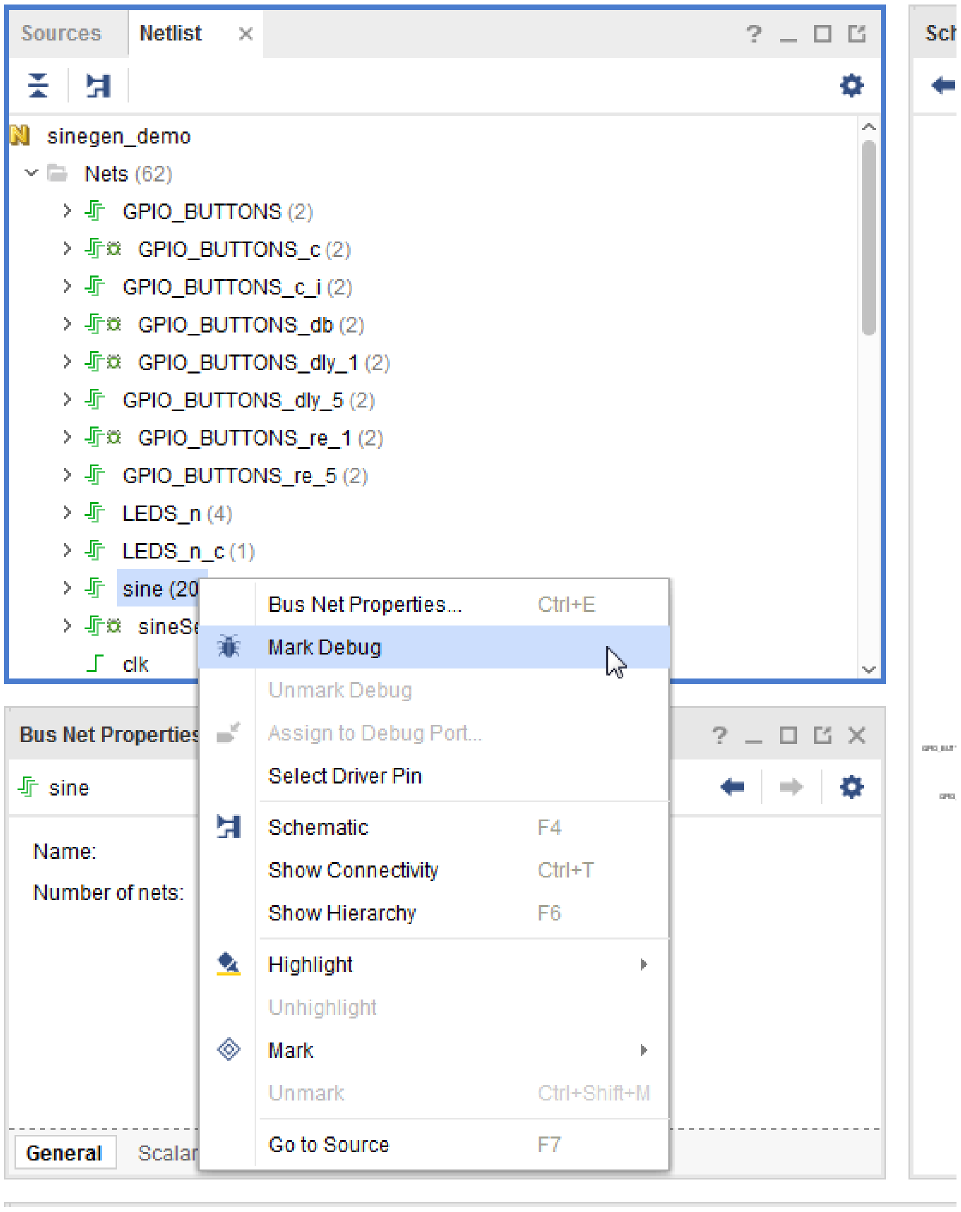
   Click **OK** in the Add Source Files dialog box. In the Add Netlist Sources dialog box ensure that Copy Sources into Project is selected. Click **Next**.
7. Click **Add Files**, navigate to the `Vivado_Debug/src` folder, and select the `sinegen_demo_kc705.xdc` file. This file has the appropriate constraints 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

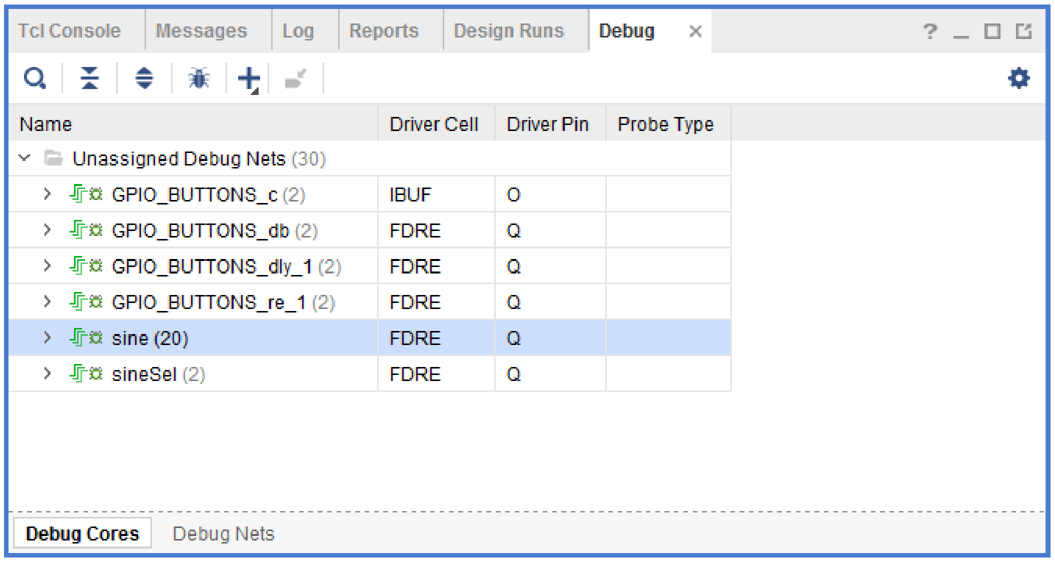
1. In Vivado® IDE, in the Flow Navigator, select **Open Synthesized Design** from the Netlist Analysis folder.
2. Select the Netlist tab in the Netlist window to expand Nets. Select the following nets for debugging:

   - 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.



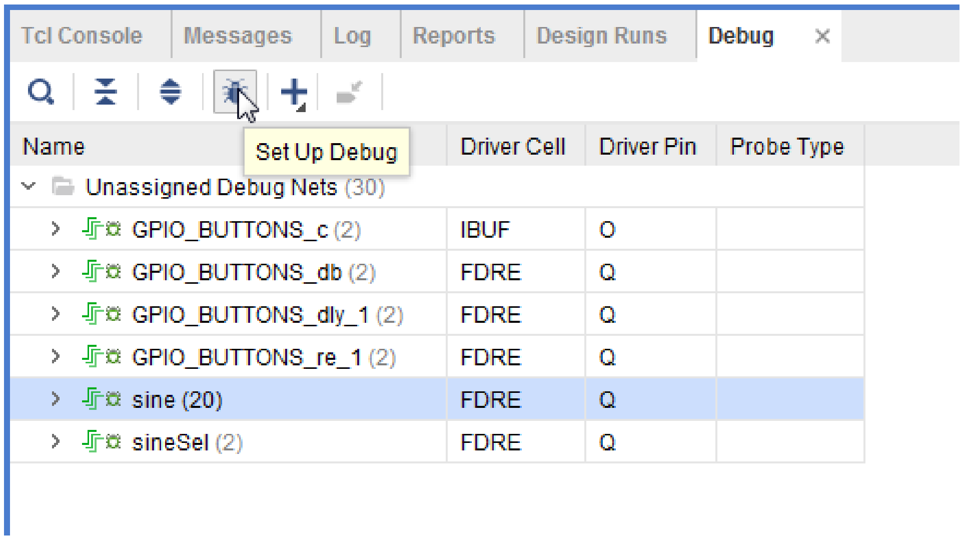
3. You should be able to see all the nets that are marked for debug, as shown in the following figure.



Name	Driver Cell	Driver Pin	Probe Type
Unassigned Debug Nets (30)			
GPIO_BUTTONS_c (2)	IBUF	O	
GPIO_BUTTONS_db (2)	FDRE	Q	
GPIO_BUTTONS_dly_1 (2)	FDRE	Q	
GPIO_BUTTONS_re_1 (2)	FDRE	Q	
sine (20)	FDRE	Q	
sineSel (2)	FDRE	Q	

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.



Name	Driver Cell	Driver Pin	Probe Type
Unassigned Debug Nets (30)			
GPIO_BUTTONS_c (2)	IBUF	O	
GPIO_BUTTONS_db (2)	FDRE	Q	
GPIO_BUTTONS_dly_1 (2)	FDRE	Q	
GPIO_BUTTONS_re_1 (2)	FDRE	Q	
sine (20)	FDRE	Q	
sineSel (2)	FDRE	Q	

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 Chapter 6: Lab 5: Using the Vivado Logic Analyzer to Debug Hardware to complete the rest of this lab.

Chapter 6

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\2019.x\bin\hw_server
   ```

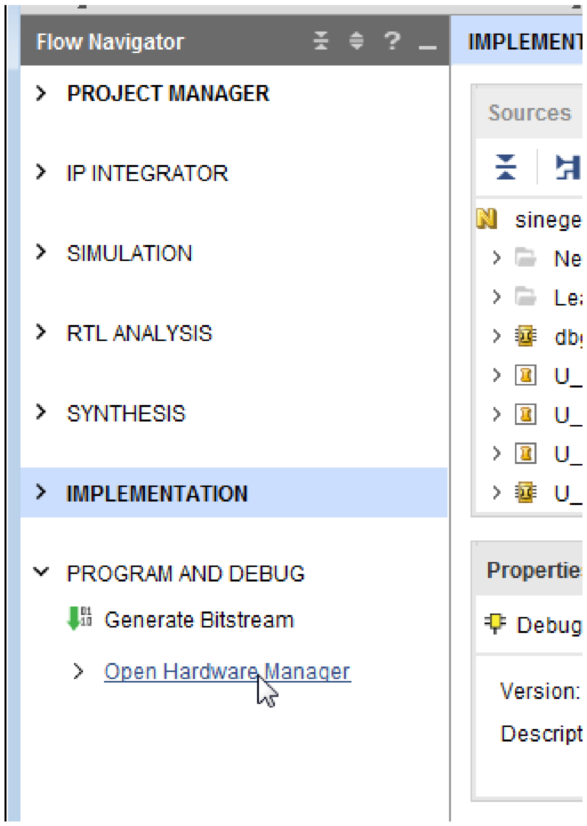
   Leave this `cmd` prompt open while the hw_server is running. Note the machine name that you are using, you will use this later when opening a connection to this instance of the hw_server application.

- **Connecting to the Target Board Locally:** If you plan to connect locally, ensure that the KC705 board is plugged into a Windows machine and then perform the following steps:

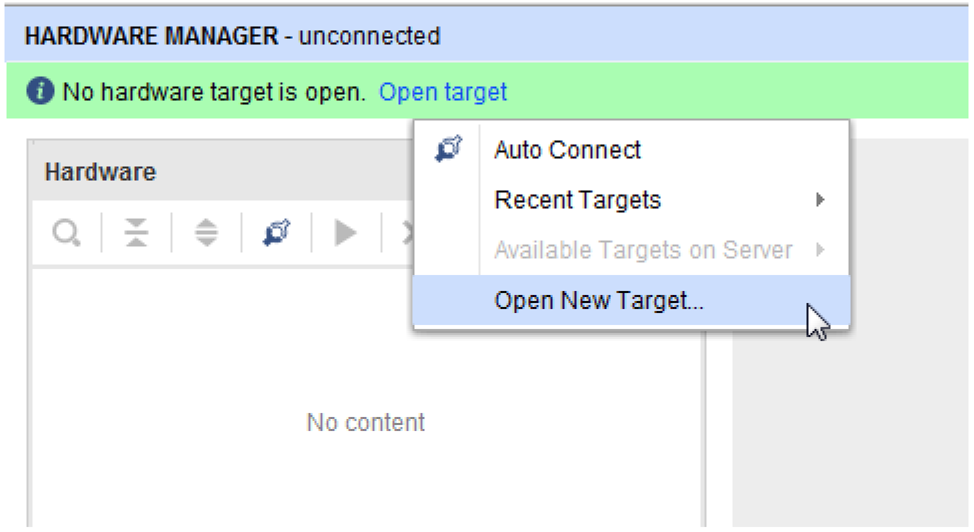
  1. Connect the Digilent USB JTAG cable of your KC705 board to a USB port on a Windows system.
  2. Ensure that the board is plugged in and powered on.
  3. Power cycle the board to clear the device.
  4. Turn DIP switch positions (pin 1 on SW13, De-bounce Enable) to the OFF position.

Using the Vivado Integrated Logic Analyzer

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

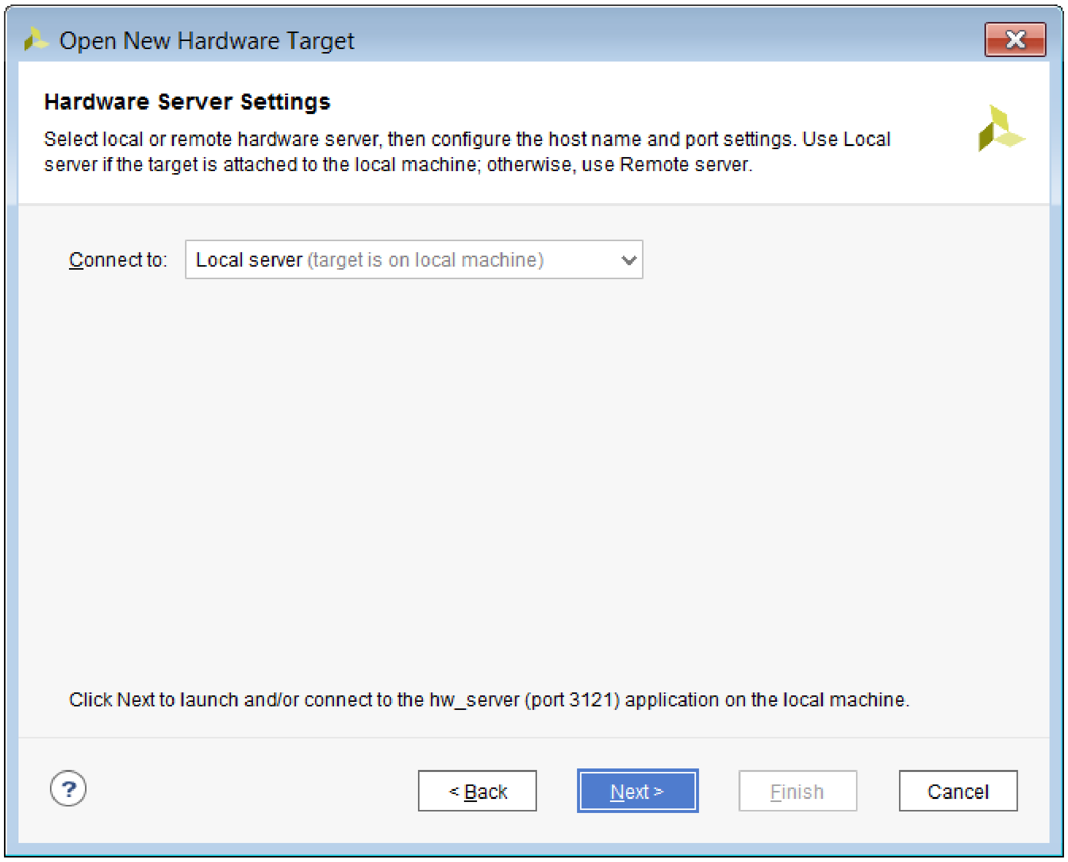


2. The Hardware Manager window opens. Click **Open Target → Open New Target**.


HARDWARE MANAGER - unconnected
No hardware target is open. Open target
Hardware
Auto Connect
Recent Targets
Available Targets on Server
Open New Target...
No content

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**.



Hardware Server Settings

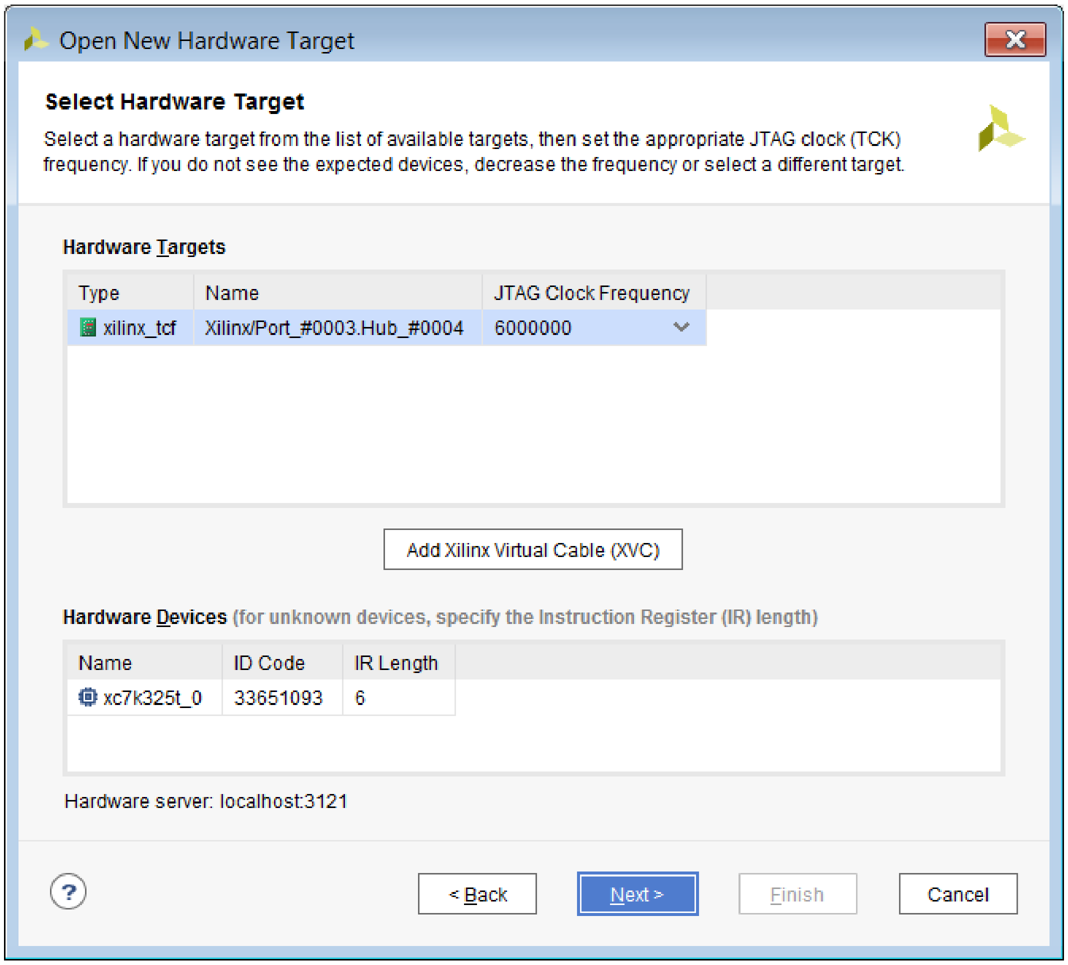
Select local or remote hardware server, then configure the host name and port settings. Use Local server if the target is attached to the local machine; otherwise, use Remote server.

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.

***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**.



Select Hardware Target

Select a hardware target from the list of available targets, then set the appropriate JTAG clock (TCK) frequency. If you do not see the expected devices, decrease the frequency or select a different target.

Hardware Targets

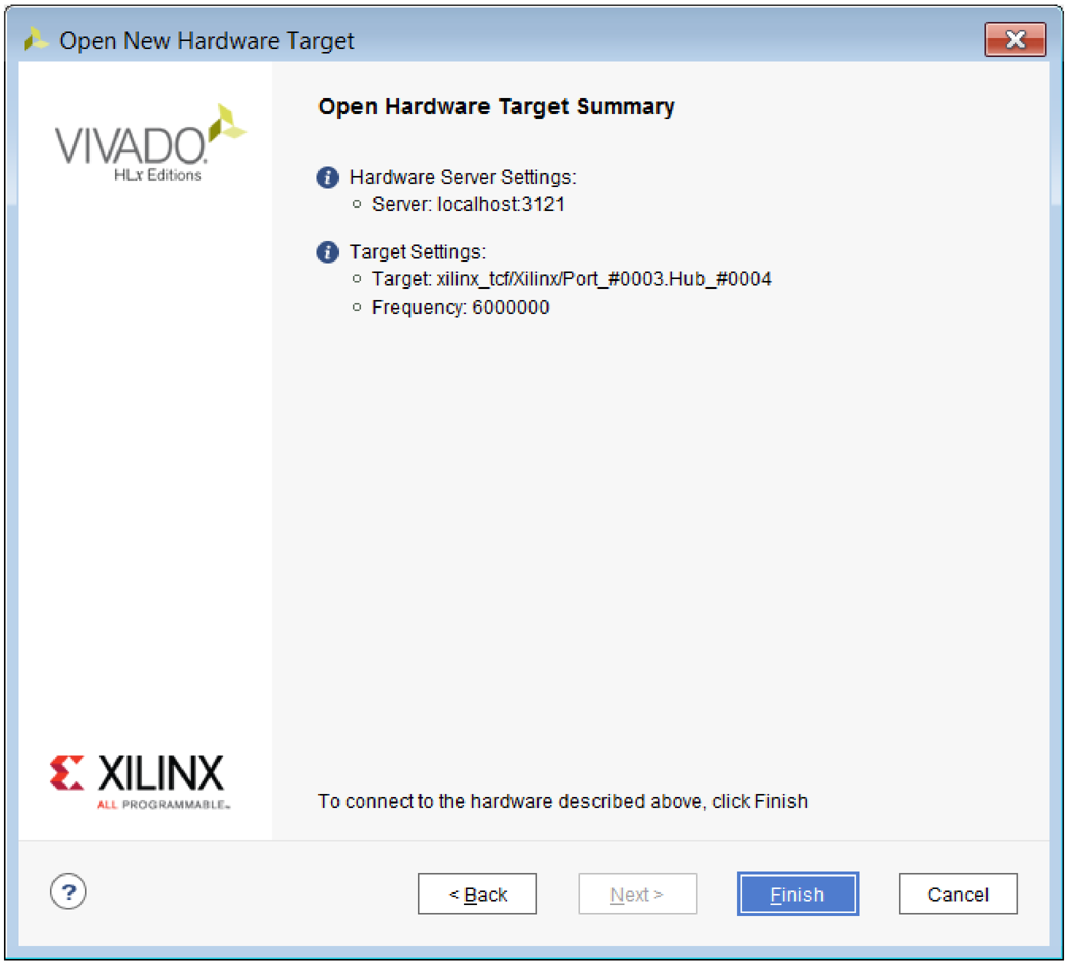
Type	Name	JTAG Clock Frequency
xilinx_tcf	Xilinx/Port_#0003.Hub_#0004	6000000

Hardware Devices (for unknown devices, specify the Instruction Register (IR) length)

Name	ID Code	IR Length
xc7k325t_0	33651093	6

Hardware server: localhost:3121

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



Open Hardware Target Summary

Hardware Server Settings:

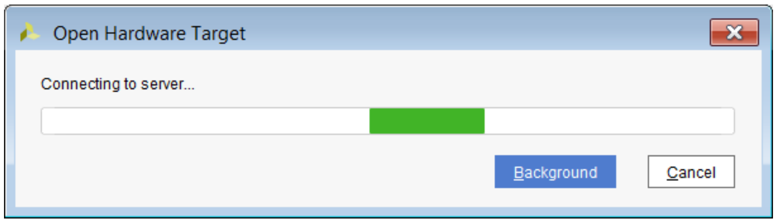
- Server: localhost:3121

Target Settings:

- Target: xilinx_tcf/Xilinx/Port_#0003.Hub_#0004
- Frequency: 6000000

To connect to the hardware described above, click Finish

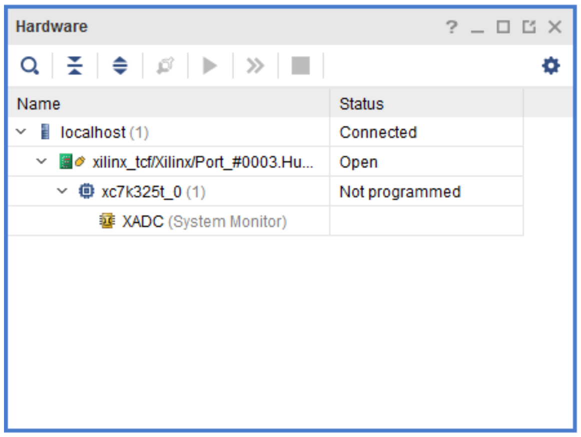
7. Wait for the connection to the hardware to complete. The dialog in following figure appears while hardware is connecting.



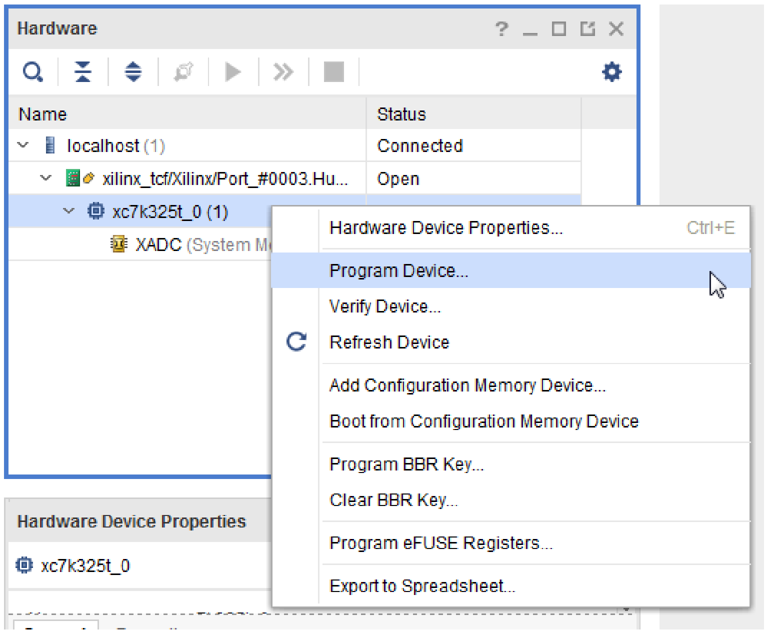
Connecting to server...

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

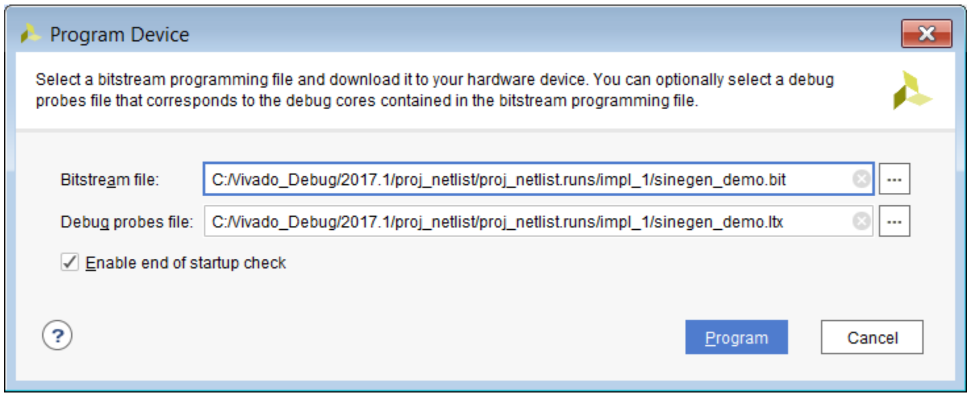
***Note*:** The Hardware tab in the Debug view shows the hardware target and XC7K325T device detected in the JTAG chain.



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


Hardware
Name
Status
localhost (1)
Connected
xilinx_tcf/Xilinx/Port_#0003.Hu...
Open
xc7k325t_0 (1)
XADC (System M
Hardware Device Properties... Ctrl+E
Program Device...
Verify Device...
Refresh Device
Add Configuration Memory Device...
Boot from Configuration Memory Device
Program BBR Key...
Clear BBR Key...
Program eFUSE Registers...
Export to Spreadsheet...
Hardware Device Properties
xc7k325t_0

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


Program Device

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.

Bitstream file: C:/Vivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1/sinegen_demo.bit

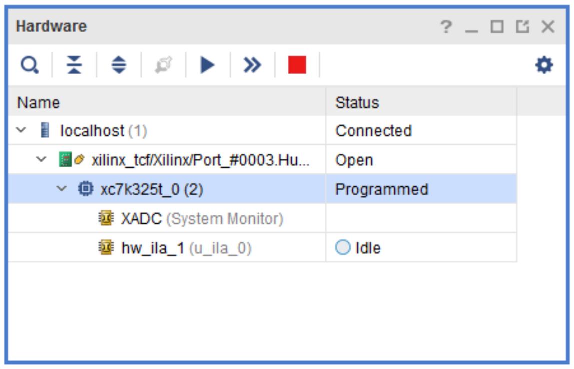
Debug probes file: C:/Vivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1/sinegen_demo.ltx

☑ Enable end of startup check

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.*

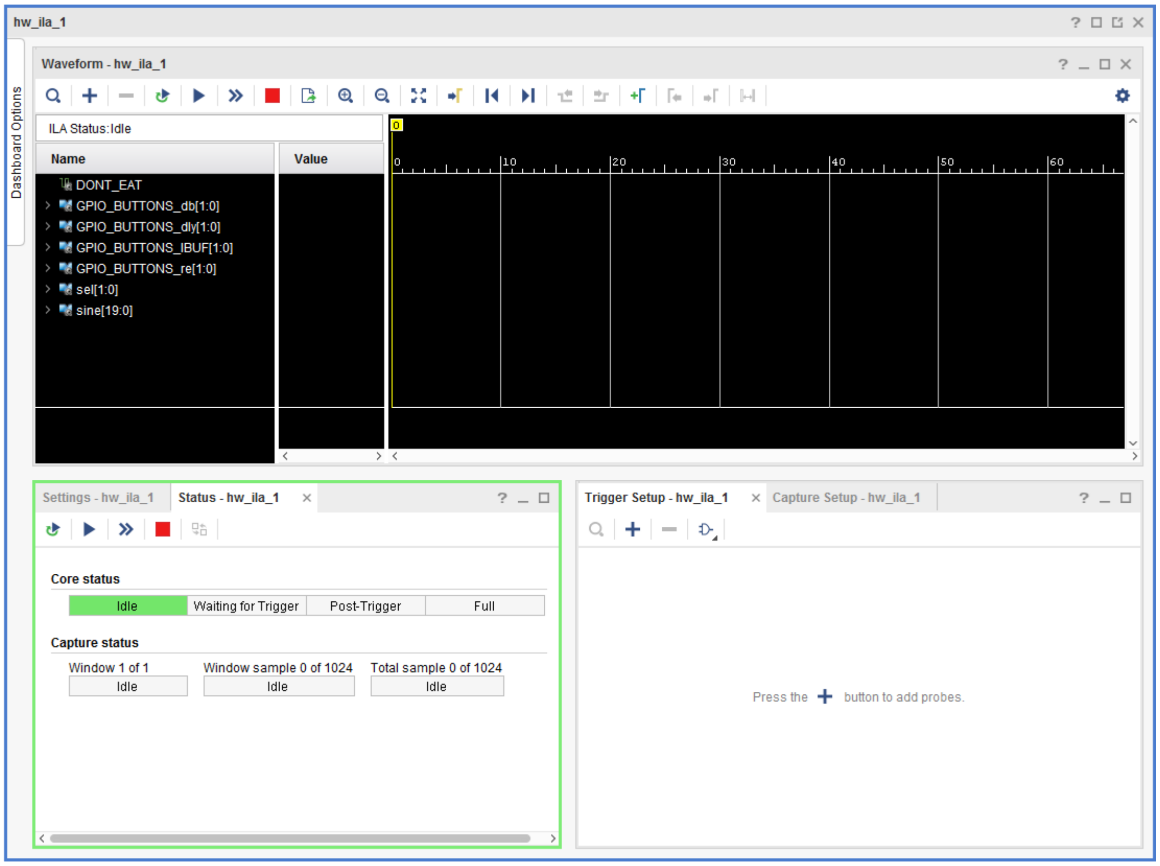
***Note*:** Wait for the program device operation to complete. This may take few minutes.

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


Hardware

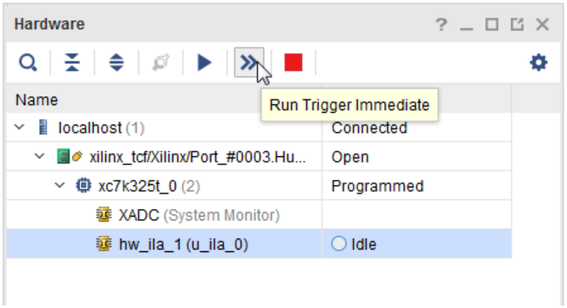
Name	Status
localhost (1)	Connected
xilinx_tcf/Xilinx/Port_#0003.Hu...	Open
xc7k325t_0 (2)	Programmed
XADC (System Monitor)	
hw_ila_1 (u_ila_0)	Idle

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

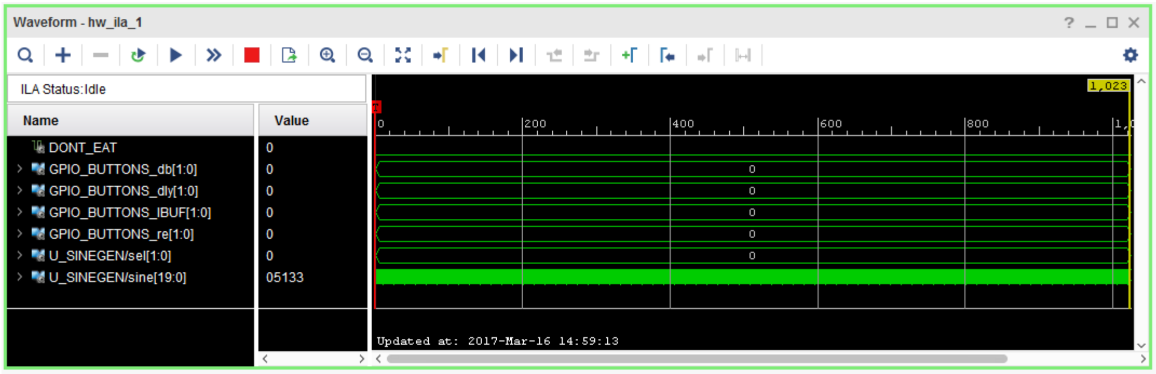


Verifying Sine Wave Activity

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.

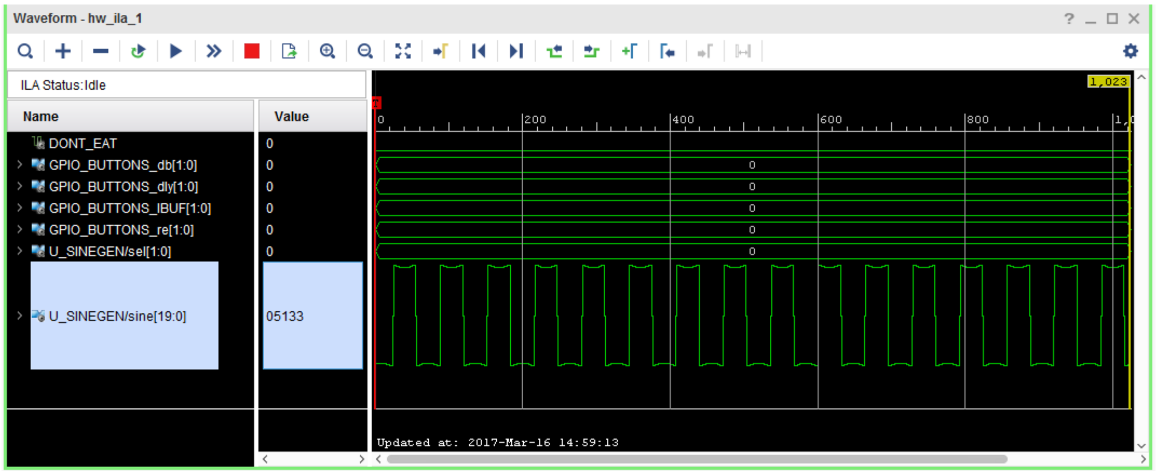


Name	Value
DONT_EAT	0
GPIO_BUTTONS_db[1:0]	0
GPIO_BUTTONS_dly[1:0]	0
GPIO_BUTTONS_IBUF[1:0]	0
GPIO_BUTTONS_re[1:0]	0
U_SINEGEN/sel[1:0]	0
U_SINEGEN/sine[19:0]	05133

```
Updated at: 2017-Mar-16 14:59:13
```

Displaying the Sine Wave

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



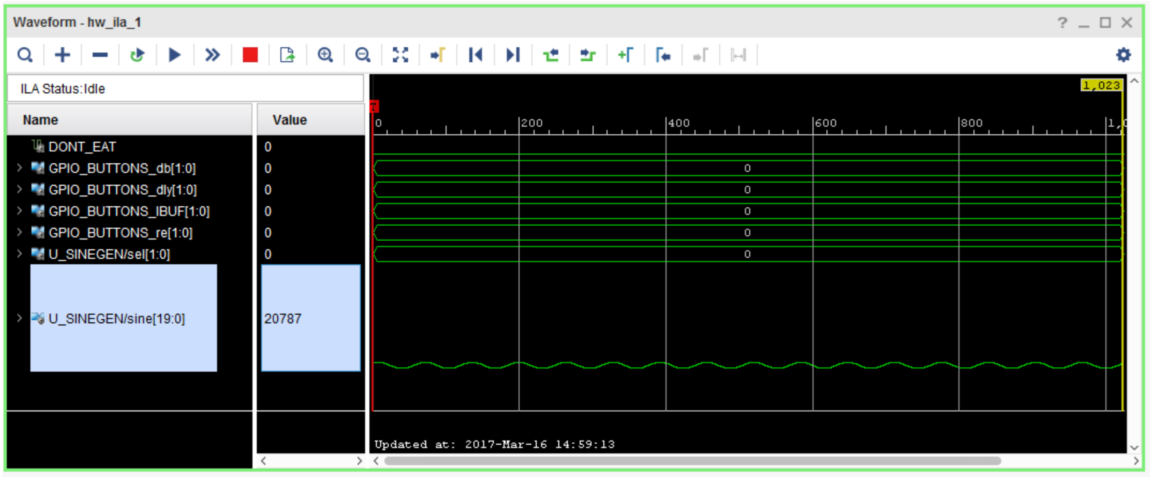
Name	Value
DONT_EAT	0
GPIO_BUTTONS_db[1:0]	0
GPIO_BUTTONS_dly[1:0]	0
GPIO_BUTTONS_IBUF[1:0]	0
GPIO_BUTTONS_re[1:0]	0
U_SINEGEN/sel[1:0]	0
U_SINEGEN/sine[19:0]	05133

```
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**.

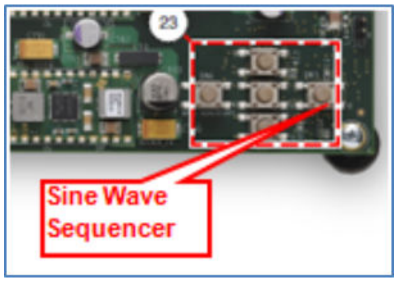
   You should now be able to see the high frequency sine wave as shown in the following figure instead of the square wave.



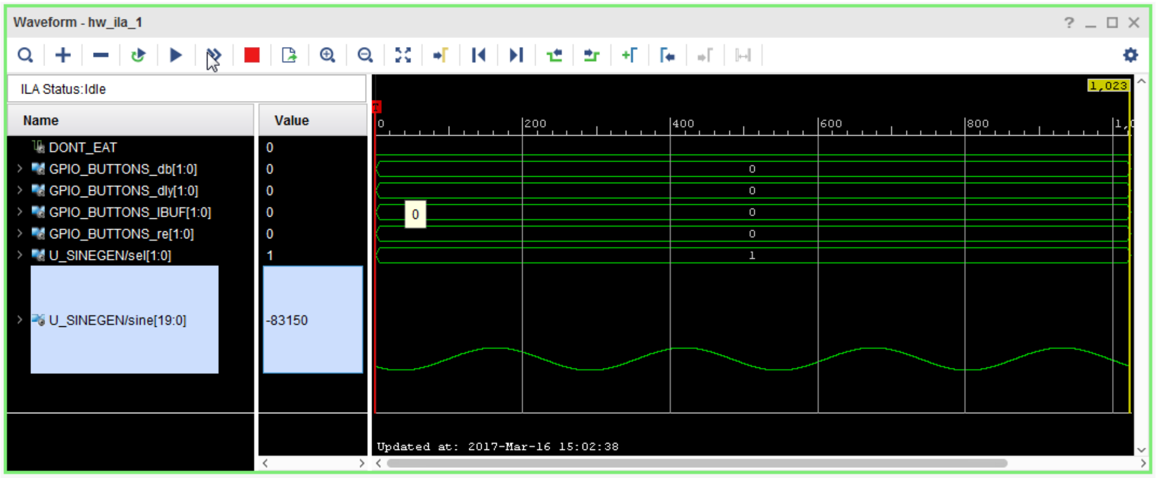
Correcting Display of the Sine Wave

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

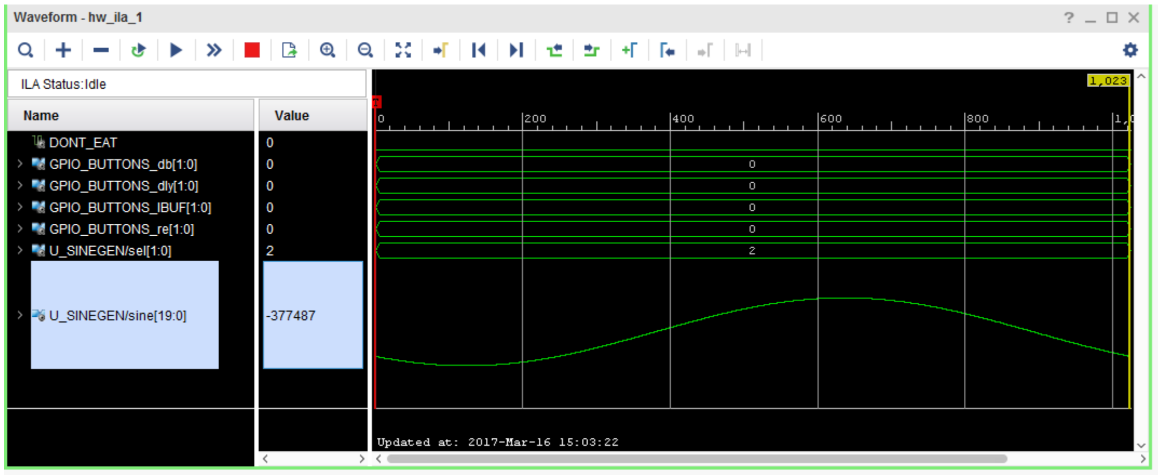
1. Cycle the sine wave sequential circuit by pressing the GPIO_SW_E push button as shown in the following figure.

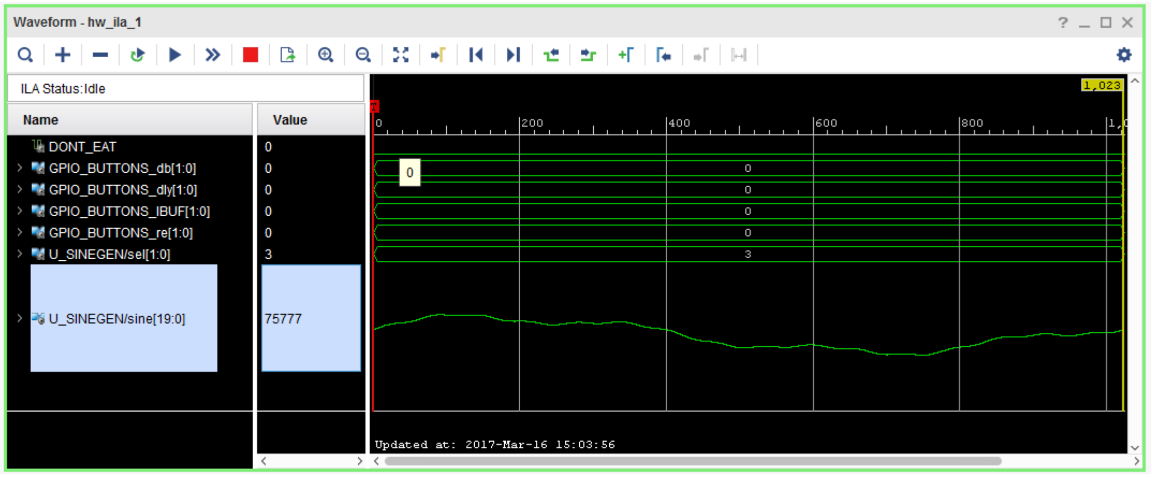


2. Click **Run Trigger Immediately** again to see the new sine selected sine wave. You should see the mid frequency as shown in the following figure. Notice that the `sel` signal also changed from 0 to 1 as expected.



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





***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.

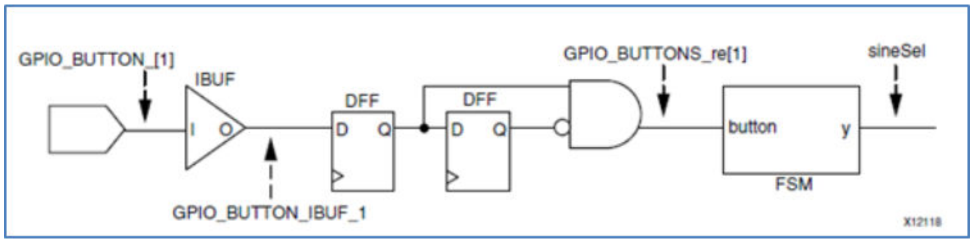
Sine Wave Sequencer State Machine Overview

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**



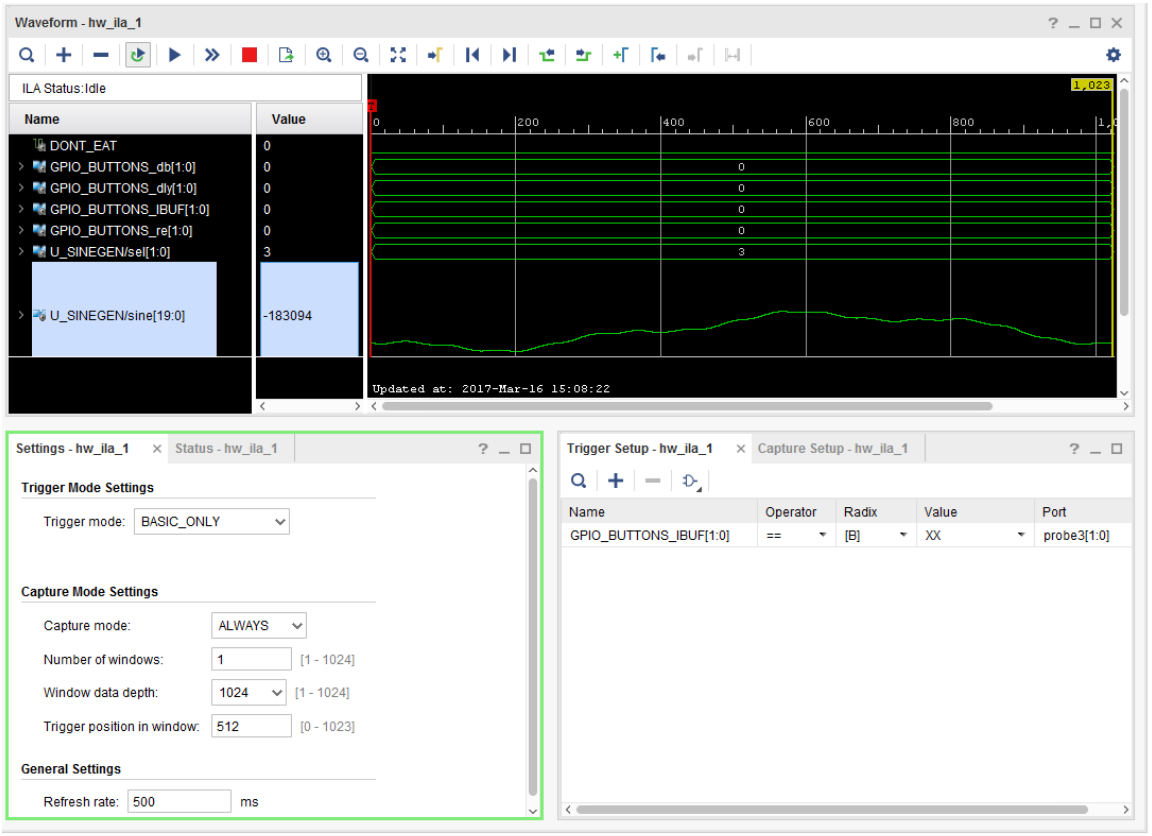
Viewing the State Machine Glitch

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

ILA Status:Idle

Name	Value
DONT_EAT	0
GPIO_BUTTONS_db[1:0]	0
GPIO_BUTTONS_dly[1:0]	0
GPIO_BUTTONS_IBUF[1:0]	0
GPIO_BUTTONS_re[1:0]	0
U_SINEGEN/sel[1:0]	3
U_SINEGEN/sine[19:0]	-183094

Updated at: 2017-Mar-16 15:08:22

Settings - hw_ila_1

Trigger Mode Settings

Trigger mode: BASIC_ONLY

Capture Mode Settings

Capture mode: ALWAYS

Number of windows: 1 [1 - 1024]

Window data depth: 1024 [1 - 1024]

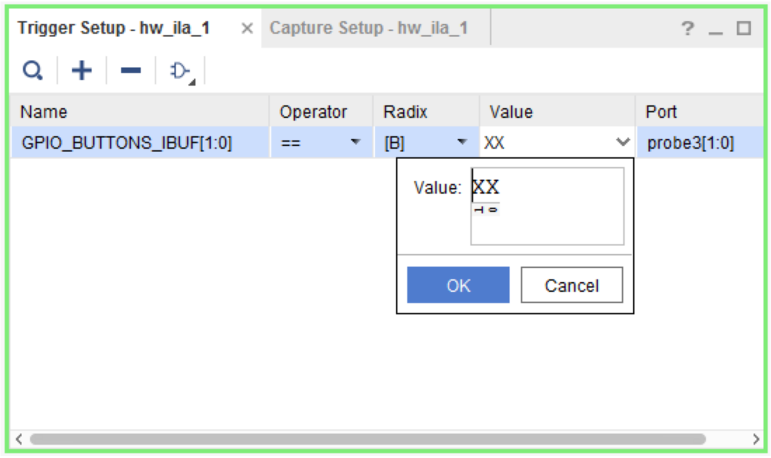
Trigger position in window: 512 [0 - 1023]

General Settings

Refresh rate: 500 ms

Trigger Setup - hw_ila_1

Name	Operator	Radix	Value	Port
GPIO_BUTTONS_IBUF[1:0]	==	[B]	XX	probe3[1:0]



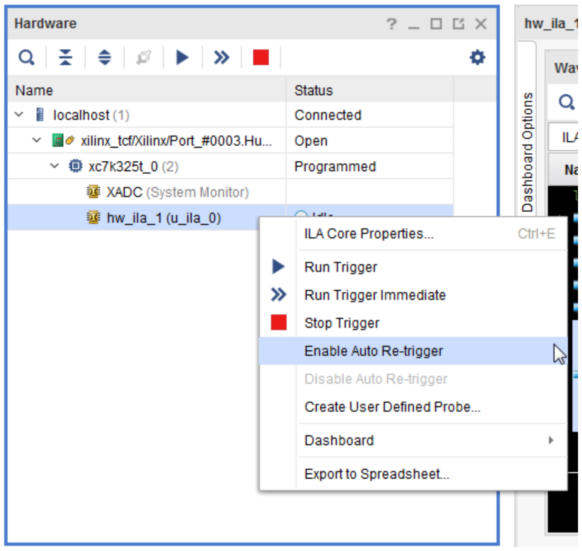
Trigger Setup - hw_ila_1

Name	Operator	Radix	Value	Port
GPIO_BUTTONS_IBUF[1:0]	==	[B]	XX	probe3[1:0]

Value: XX

CAUTION! *For different labs the GPIO_BUTTONS_IBUF may show up differently or have a different name such as button_in4_in. This may 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.



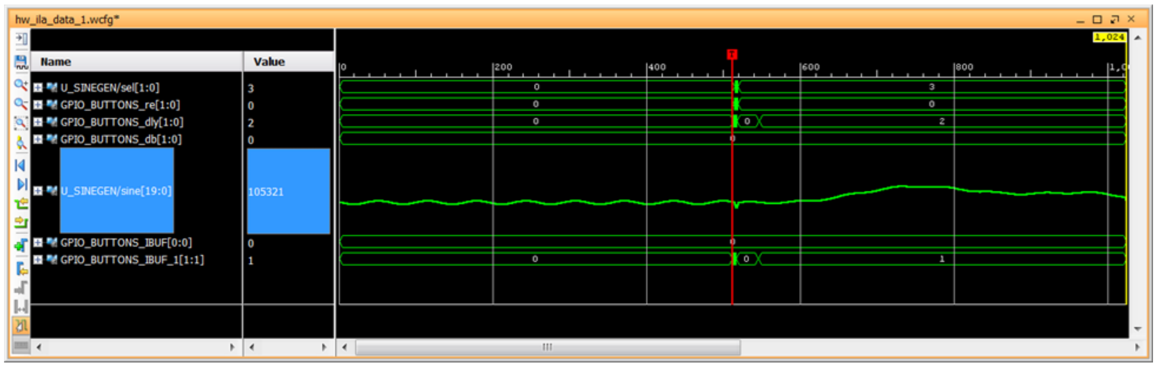
CAUTION! *The ILA properties window may look slightly different for different labs.*

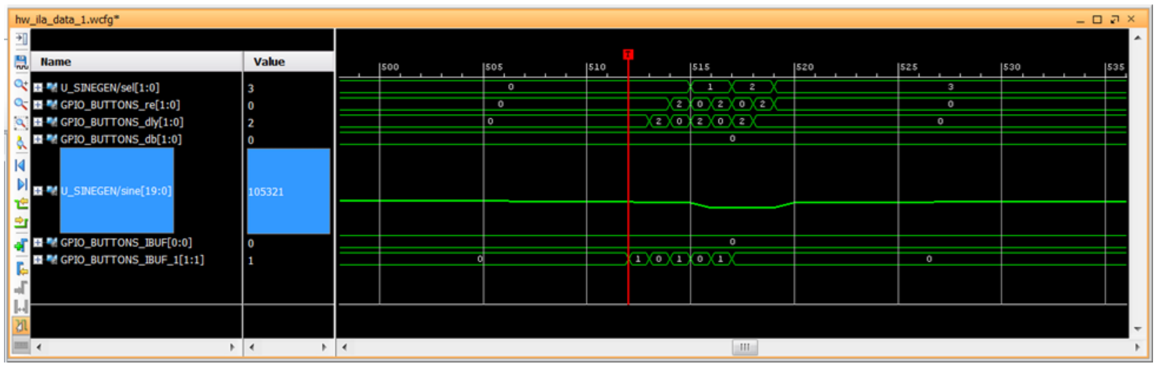
When you issue a Run Trigger or a Run Trigger Immediate command after setting the Auto Retrigger mode, the ILA core does the following repetitively until you disable the Auto Retrigger mode option.

- 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. Once you can see the glitch, you may observe that the signal glitches are not at exactly the same location as shown in the figure below.*





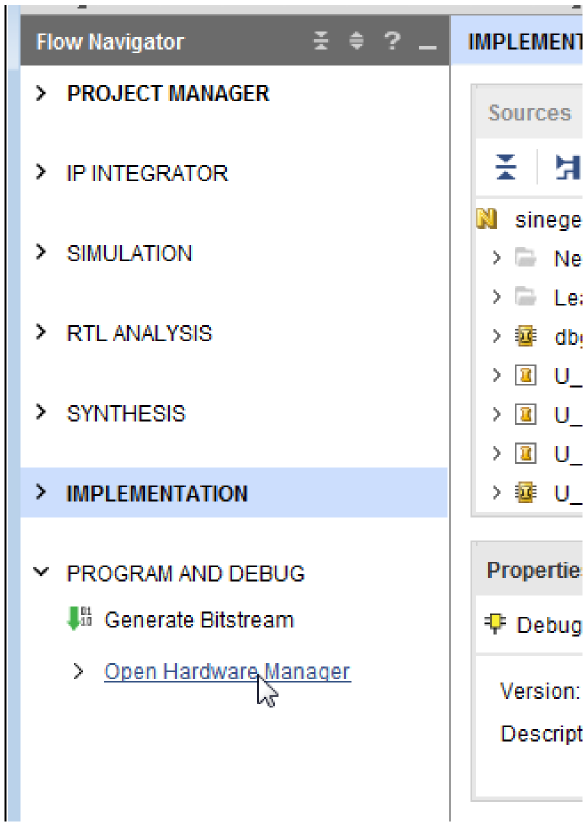
Fixing the Signal Glitch and Verifying the Correct State Machine Behavior

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.

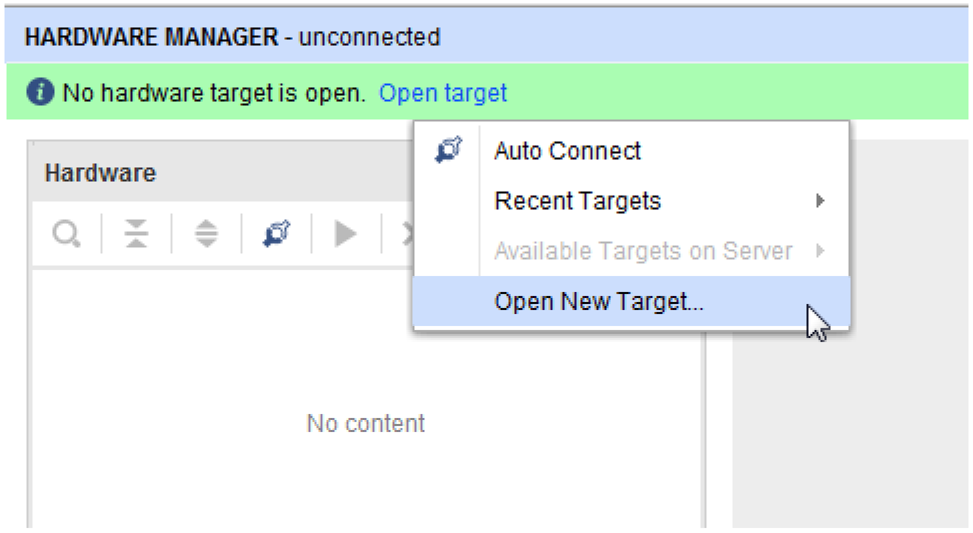
Verifying the VIO Core Activity (Only applicable to Lab 3)

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**.

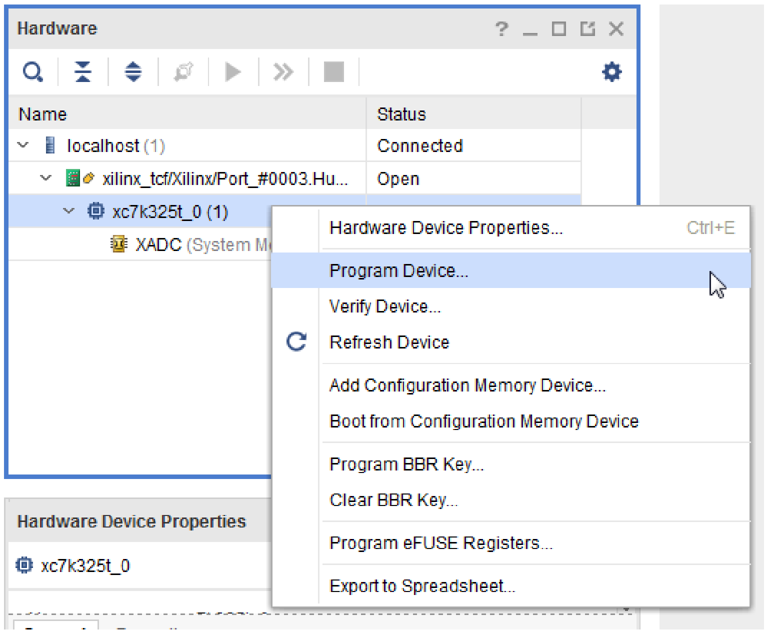


No hardware target is open. Open target

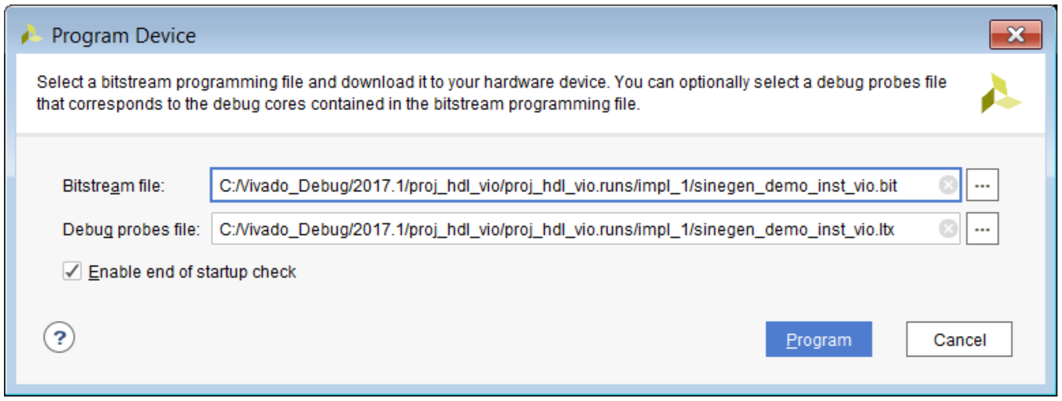
3. The Open New Hardware Target wizard opens. Click **Next**.

4. In the Hardware Server Settings page, type the name of the server (or select **Local server** if the target is on the local machine) in the Connect to field.

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


Hardware
Name
Status
localhost (1)
Connected
xilinx_tcf/Xilinx/Port_#0003.Hu...
Open
xc7k325t_0 (1)
XADC (System M
Hardware Device Properties
xc7k325t_0
Hardware Device Properties... Ctrl+E
Program Device...
Verify Device...
Refresh Device
Add Configuration Memory Device...
Boot from Configuration Memory Device
Program BBR Key...
Clear BBR Key...
Program eFUSE Registers...
Export to Spreadsheet...

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


Program Device

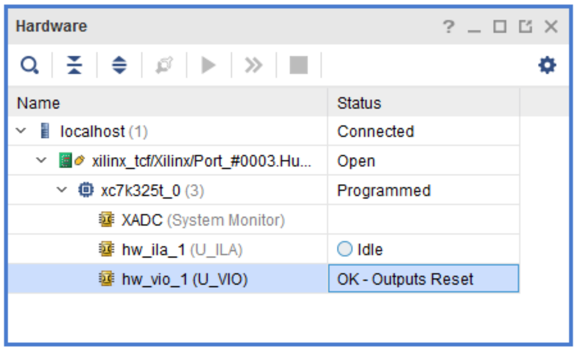
Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file: C:/Vivado_Debug/2017.1/proj_hdl_vio/proj_hdl_vio.runs/impl_1/sinegen_demo_inst_vio.bit

Debug probes file: C:/Vivado_Debug/2017.1/proj_hdl_vio/proj_hdl_vio.runs/impl_1/sinegen_demo_inst_vio.ltx

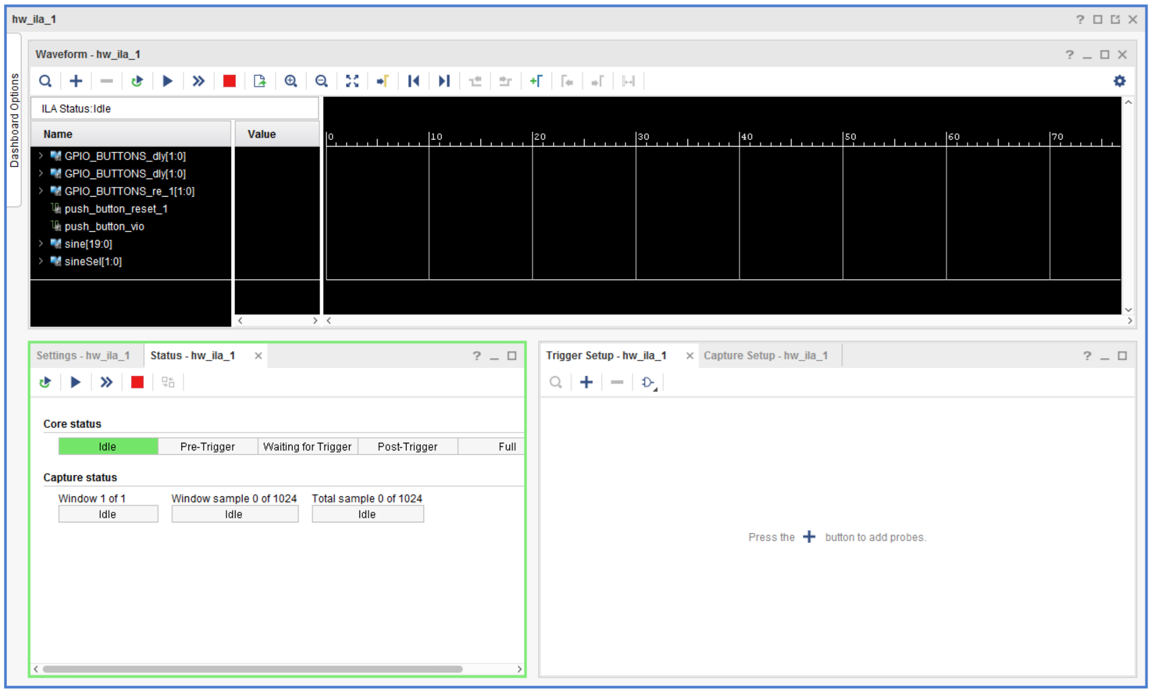
✓ Enable end of startup check

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



Name	Status
localhost (1)	Connected
xilinx_tcf/Xilinx/Port_#0003.Hu...	Open
xc7k325t_0 (3)	Programmed
XADC (System Monitor)	
hw_ila_1 (U_ILA)	Idle
hw_vio_1 (U_VIO)	OK - Outputs Reset

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



ILA Status: Idle

Name	Value
GPIO_BUTTONS_dly[1:0]	
GPIO_BUTTONS_dly[1:0]	
GPIO_BUTTONS_re_1[1:0]	
push_button_reset_1	
push_button_vio	
sine[19:0]	
sineSel[1:0]	

Core status

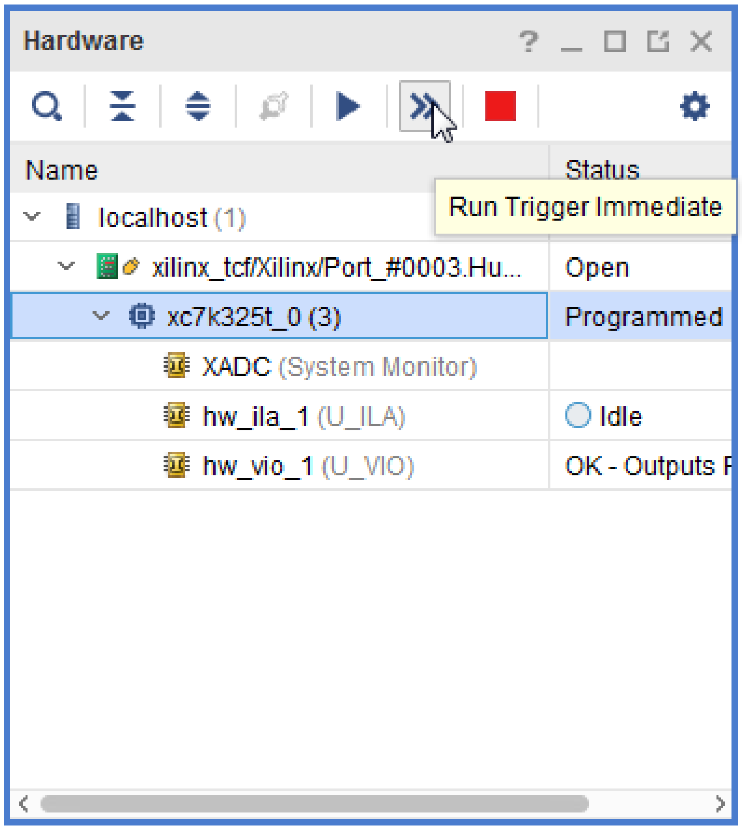
Idle	Pre-Trigger	Waiting for Trigger	Post-Trigger	Full

Capture status

Window 1 of 1	Window sample 0 of 1024	Total sample 0 of 1024
Idle	Idle	Idle

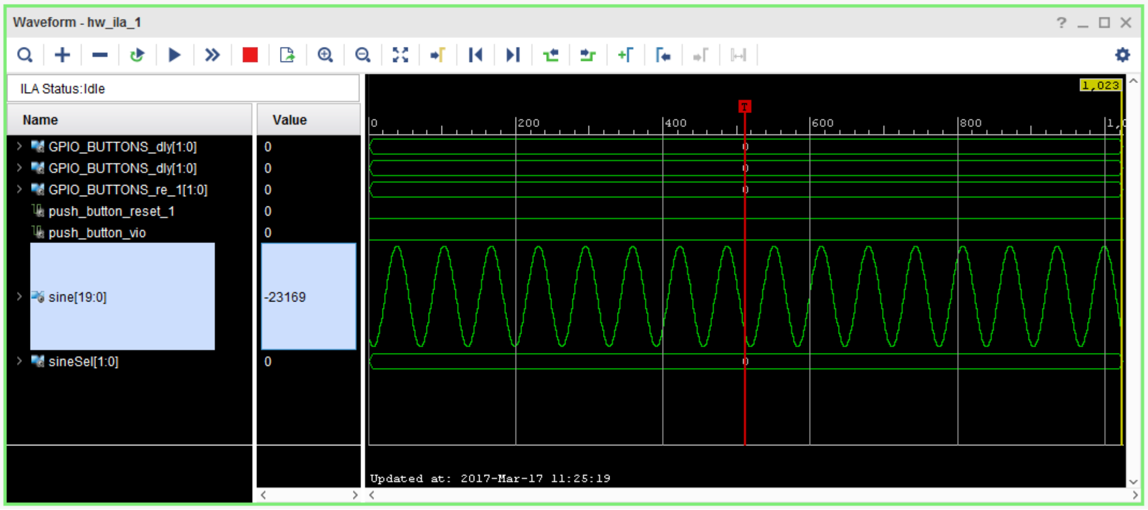
Press the + button to add probes.

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



Name	Status
localhost (1)	
xilinx_tcf/Xilinx/Port_#0003.Hu...	Open
xc7k325t_0 (3)	Programmed
XADC (System Monitor)	
hw_ila_1 (U_ILA)	Idle
hw_vio_1 (U_VIO)	OK - Outputs R

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



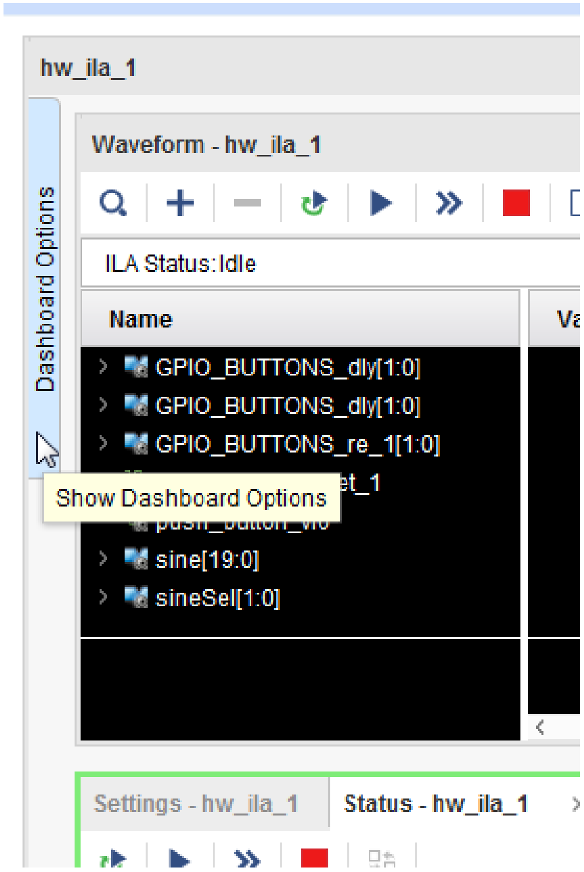
ILA Status: Idle

Name	Value
GPIO_BUTTONS_dly[1:0]	0
GPIO_BUTTONS_dly[1:0]	0
GPIO_BUTTONS_re_1[1:0]	0
push_button_reset_1	0
push_button_vio	0
sine[19:0]	-23169
sineSel[1:0]	0

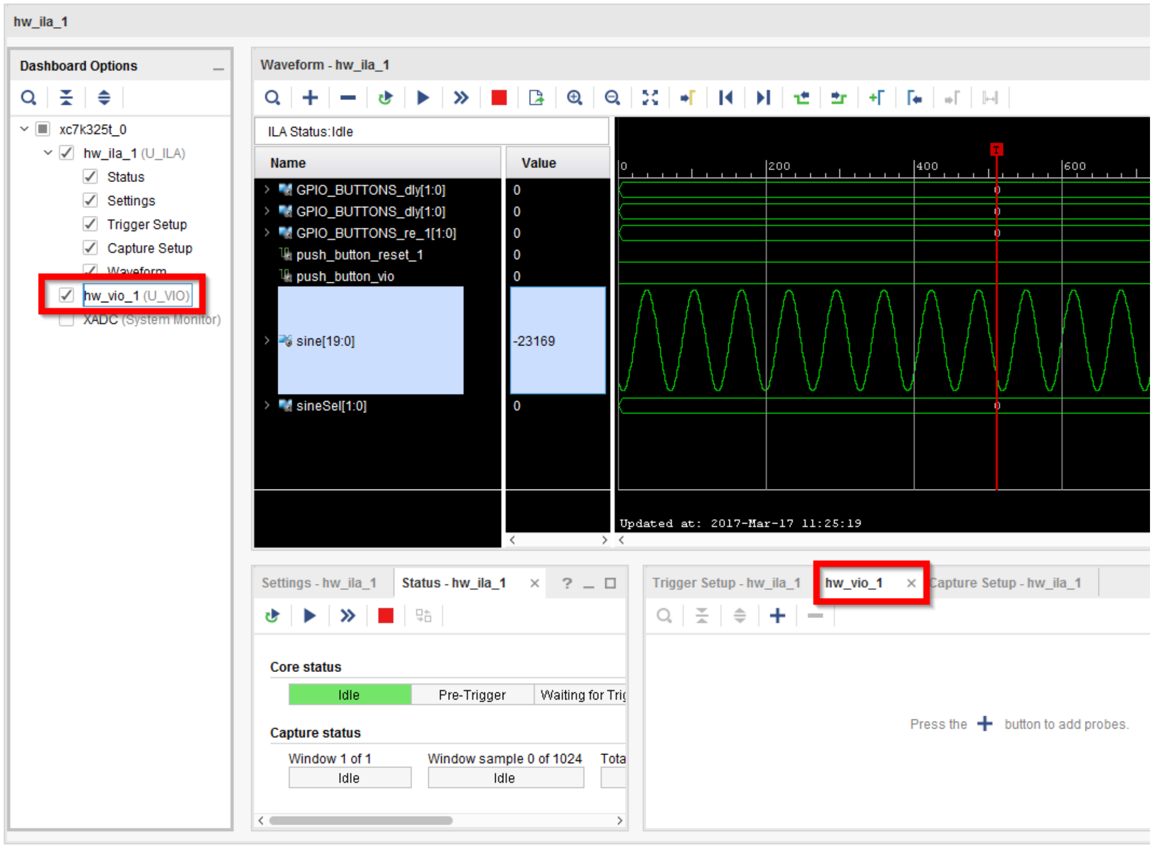
```
Updated at: 2017-Mar-17 11:25:19
```

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**.



ILA Status:Idle

Name	Value
GPIO_BUTTONS_dly[1:0]	0
GPIO_BUTTONS_dly[1:0]	0
GPIO_BUTTONS_re_1[1:0]	0
push_button_reset_1	0
push_button_vio	0
sine[19:0]	-23169
sineSel[1:0]	0

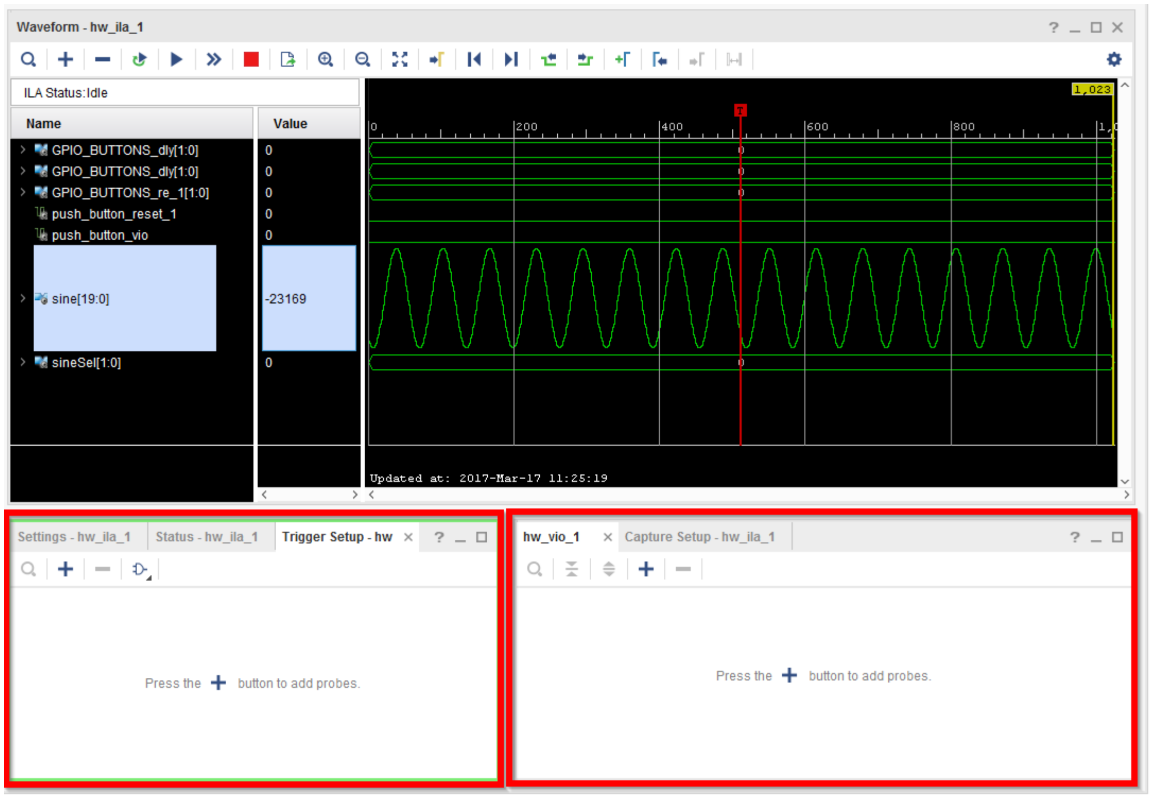
```
Updated at: 2017-Mar-17 11:25:19
```

Core status: Idle | Pre-Trigger | Waiting for Trig

Capture status: Window 1 of 1 — Idle; Window sample 0 of 1024 — Idle; Tota

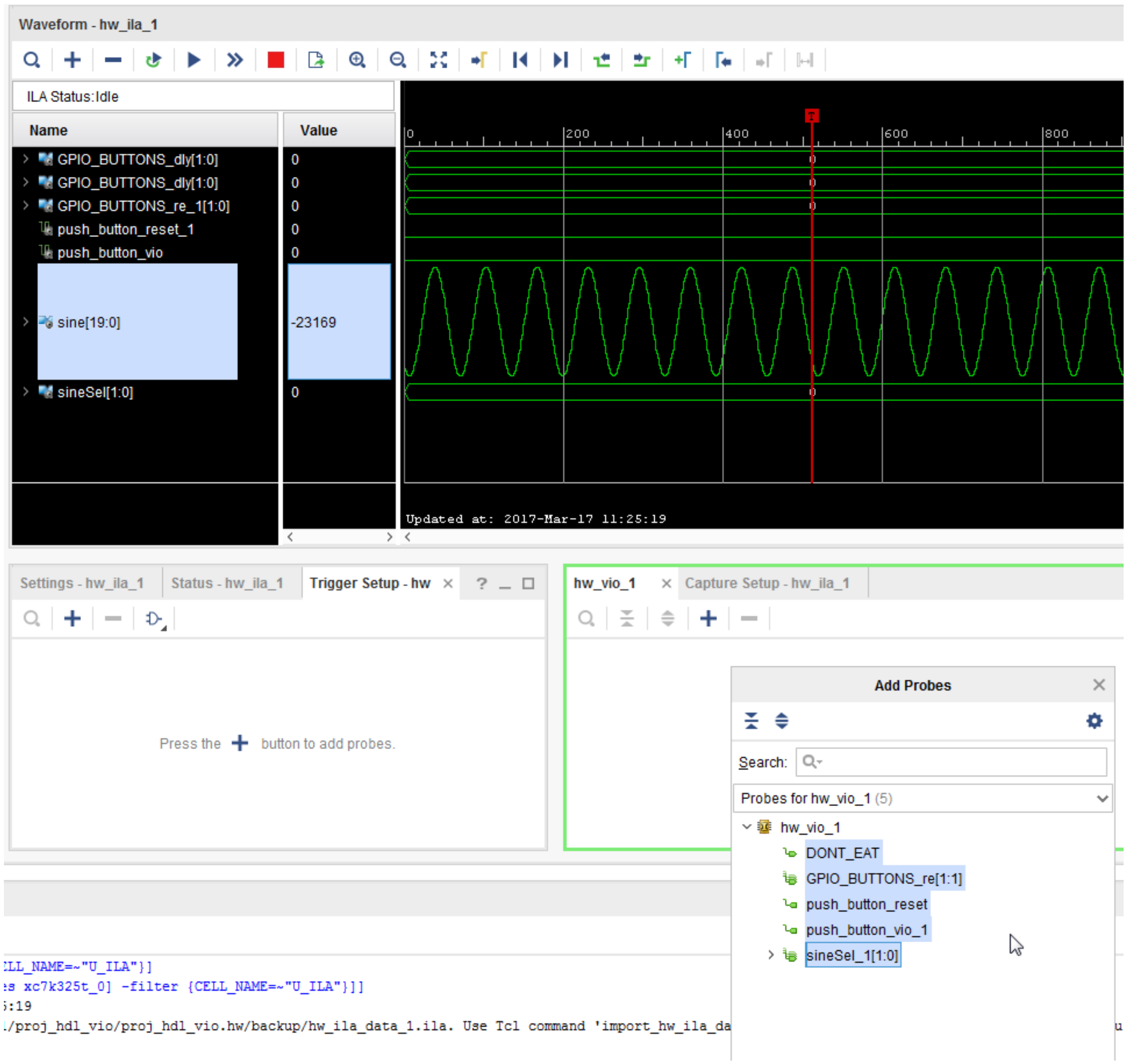
***Note*:** The ILA dashboard now contains the VIO window as well.

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.



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.

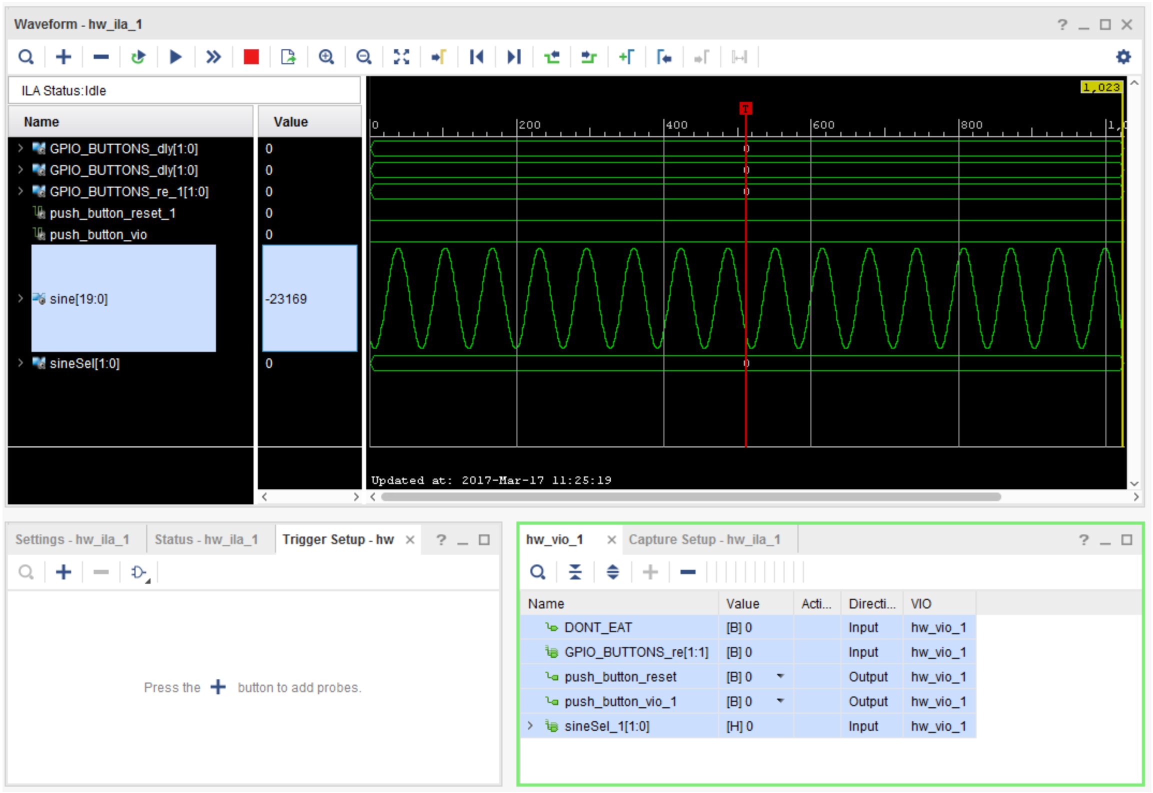


Name	Value
GPIO_BUTTONS_dly[1:0]	0
GPIO_BUTTONS_dly[1:0]	0
GPIO_BUTTONS_re_1[1:0]	0
push_button_reset_1	0
push_button_vio	0
sine[19:0]	-23169
sineSel[1:0]	0

```
Updated at: 2017-Mar-17 11:25:19
```

```
:LL_NAME=~"U_ILA"}]
:s xc7k325t_0] -filter {CELL_NAME=~"U_ILA"}]]
5:19
./proj_hdl_vio/proj_hdl_vio.hw/backup/hw_ila_data_1.ila. Use Tcl command 'import_hw_ila_da
```

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



Waveform - hw_ila_1

ILA Status: Idle

Name	Value
GPIO_BUTTONS_dly[1:0]	0
GPIO_BUTTONS_dly[1:0]	0
GPIO_BUTTONS_re_1[1:0]	0
push_button_reset_1	0
push_button_vio	0
sine[19:0]	-23169
sineSel[1:0]	0

```
Updated at: 2017-Mar-17 11:25:19
```

Settings - hw_ila_1 | Status - hw_ila_1 | Trigger Setup - hw

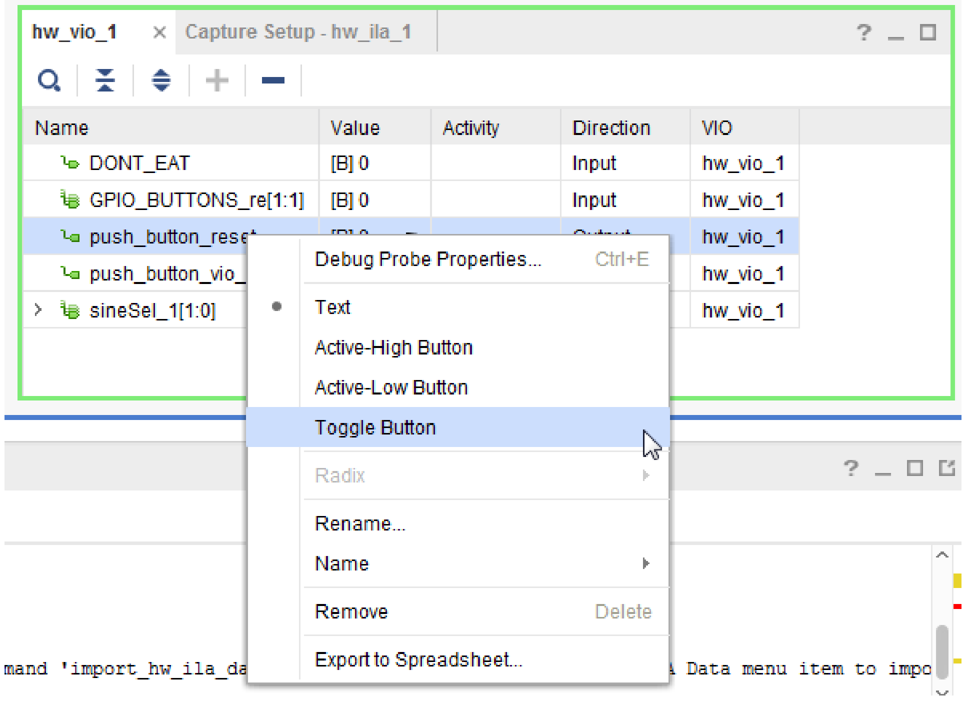
Press the + button to add probes.

hw_vio_1 | Capture Setup - hw_ila_1

Name	Value	Acti...	Directi...	VIO
DONT_EAT	[B] 0		Input	hw_vio_1
GPIO_BUTTONS_re[1:1]	[B] 0		Input	hw_vio_1
push_button_reset	[B] 0		Output	hw_vio_1
push_button_vio_1	[B] 0		Output	hw_vio_1
sineSel_1[1:0]	[H] 0		Input	hw_vio_1

22. Set the push_button_reset output probe by right-clicking **push_button_reset** and select **Toggle Button**.

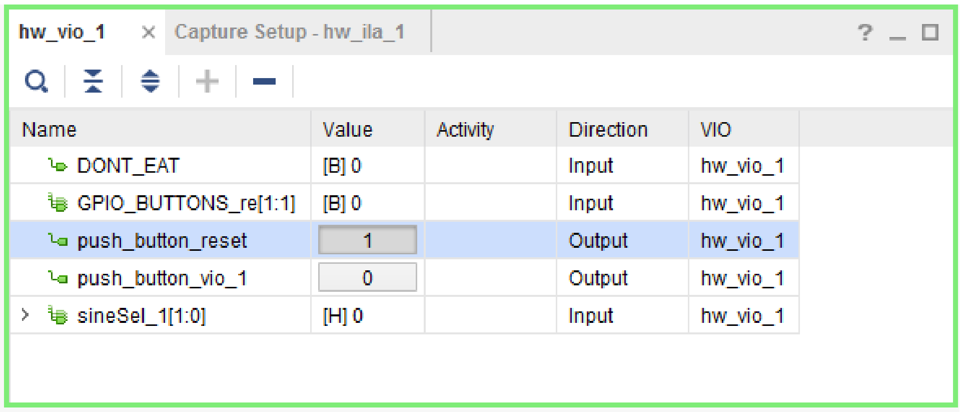
    This will toggle the output driver from logic from ‘0’ to ‘1’ to ‘0’ as you click. It is similar to the actual push button behavior, though there is no bouncing mechanical effect as with a real push button switch.



Name	Value	Activity	Direction	VIO
DONT_EAT	[B] 0		Input	hw_vio_1
GPIO_BUTTONS_re[1:1]	[B] 0		Input	hw_vio_1
push_button_rese				hw_vio_1
push_button_vio_				hw_vio_1
sineSel_1[1:0]				hw_vio_1

The Value field for push_button_reset is highlighted.

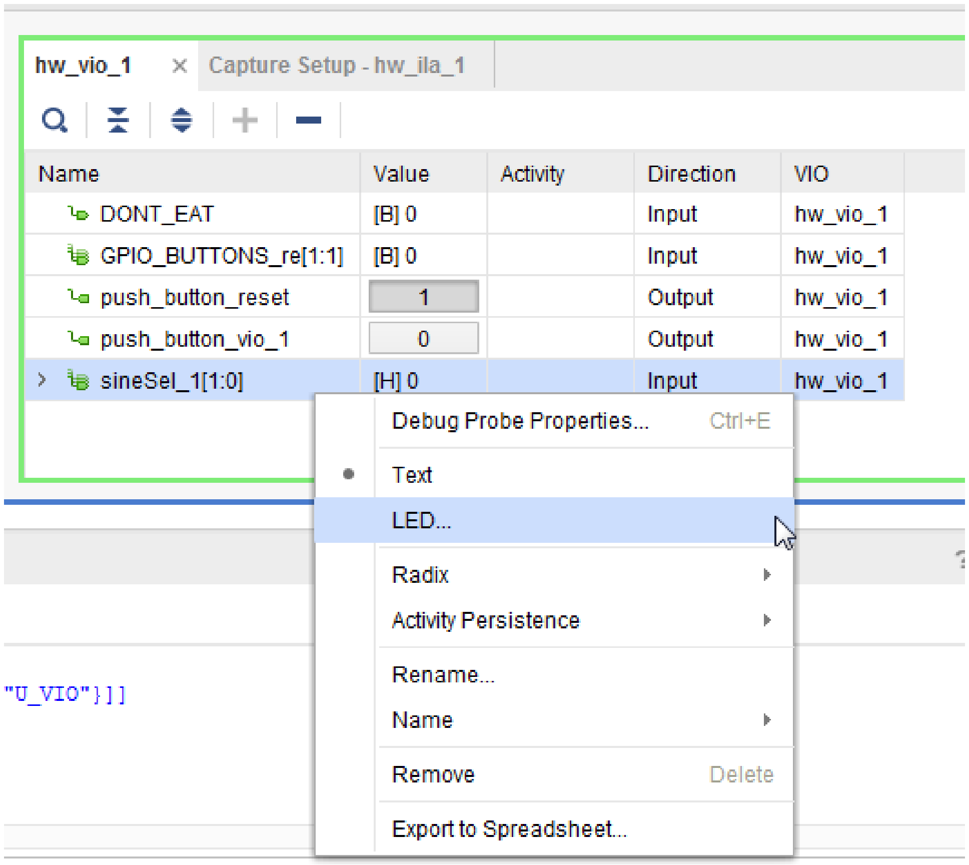
23. Click in the **Value** field to change its value to 1.



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_reset	1		Output	hw_vio_1
push_button_vio_1	0		Output	hw_vio_1
sineSel_1[1:0]	[H] 0		Input	hw_vio_1

24. Follow the step above to change the push_button_vio to Toggle button as well.

25. Set these two bits of the “sineSel” input probe by right-clicking **PROBE_IN0[0] and PROBE_IN0[1]** and selecting **LED**.

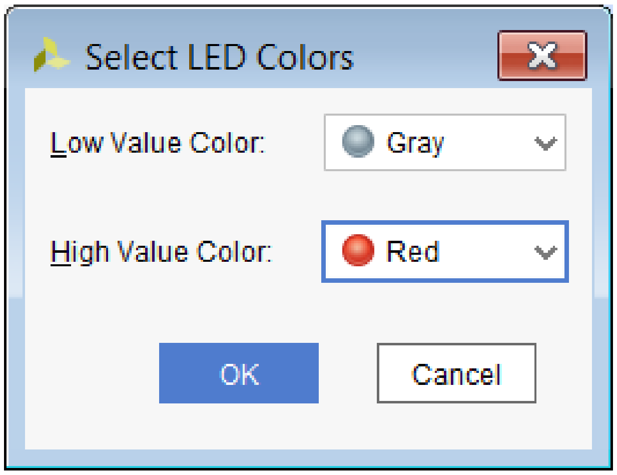


hw_vio_1 | Capture Setup - hw_ila_1

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_reset	1		Output	hw_vio_1
push_button_vio_1	0		Output	hw_vio_1
sineSel_1[1:0]	[H] 0		Input	hw_vio_1

```
"U_VIO"}]]
```

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**.

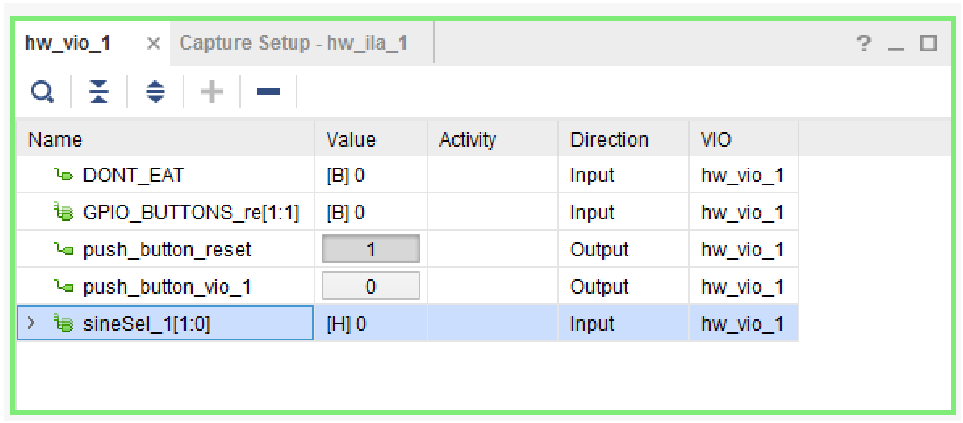


Select LED Colors

Low Value Color: Gray

High Value Color: Red

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

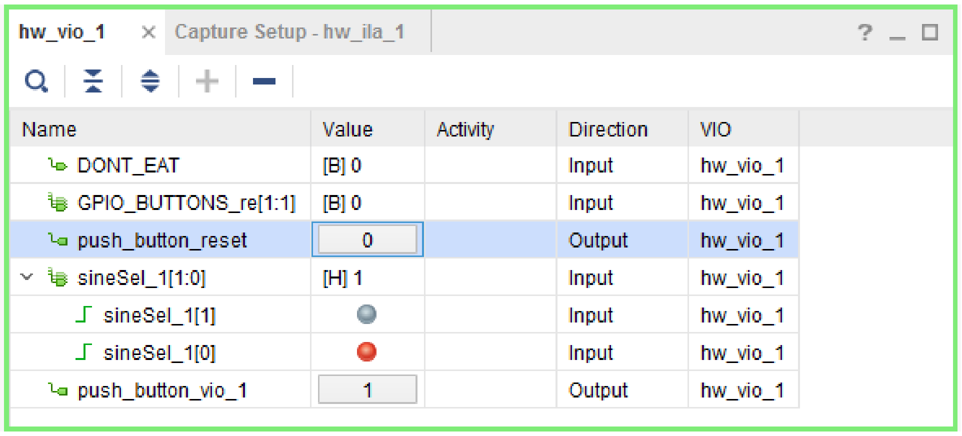


hw_vio_1 | Capture Setup - hw_ila_1

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_reset	1		Output	hw_vio_1
push_button_vio_1	0		Output	hw_vio_1
sineSel_1[1:0]	[H] 0		Input	hw_vio_1

28. To cycle through each different sine wave output frequency using the virtual "push_button_vio" from the VIO core, perform the following simple steps:

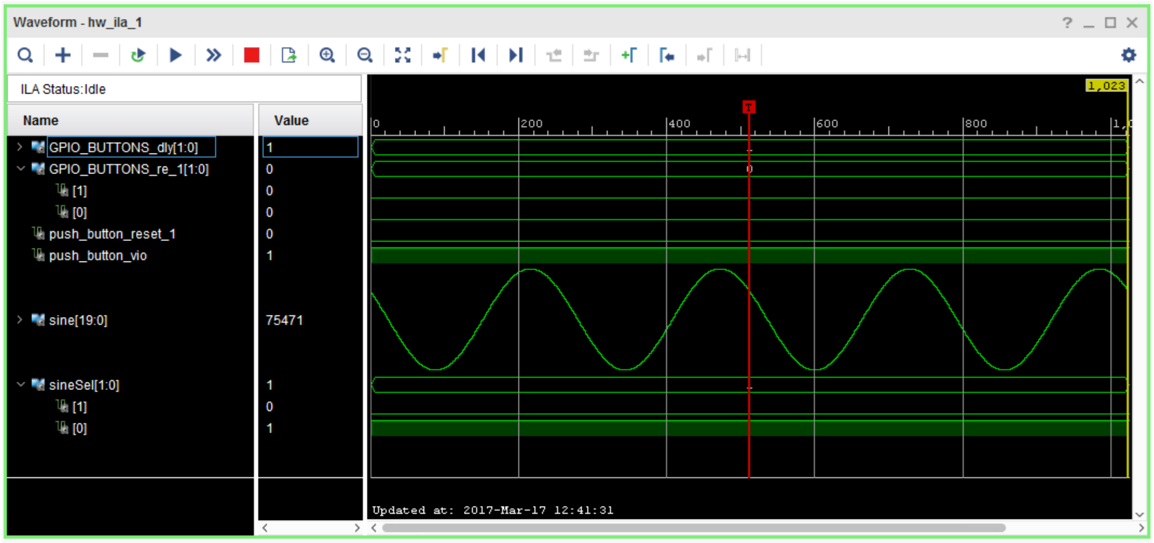
    a. Toggle the value of the "push_button_vio" output driver from 0 to 1 to 0 by clicking on the logic displayed under the Value column. You will notice the sineSel LEDs changed accordingly – 0, 1, 2, 3, 0, etc...



hw_vio_1 | Capture Setup - hw_ila_1

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_reset	0		Output	hw_vio_1
sineSel_1[1:0]	[H] 1		Input	hw_vio_1
sineSel_1[1]	●		Input	hw_vio_1
sineSel_1[0]	🔴		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.



Name	Value
GPIO_BUTTONS_dly[1:0]	1
GPIO_BUTTONS_re_1[1:0]	0
[1]	0
[0]	0
push_button_reset_1	0
push_button_vio	1
sine[19:0]	75471
sineSel[1:0]	1
[1]	0
[0]	1

```
Updated at: 2017-Mar-17 12:41:31
```

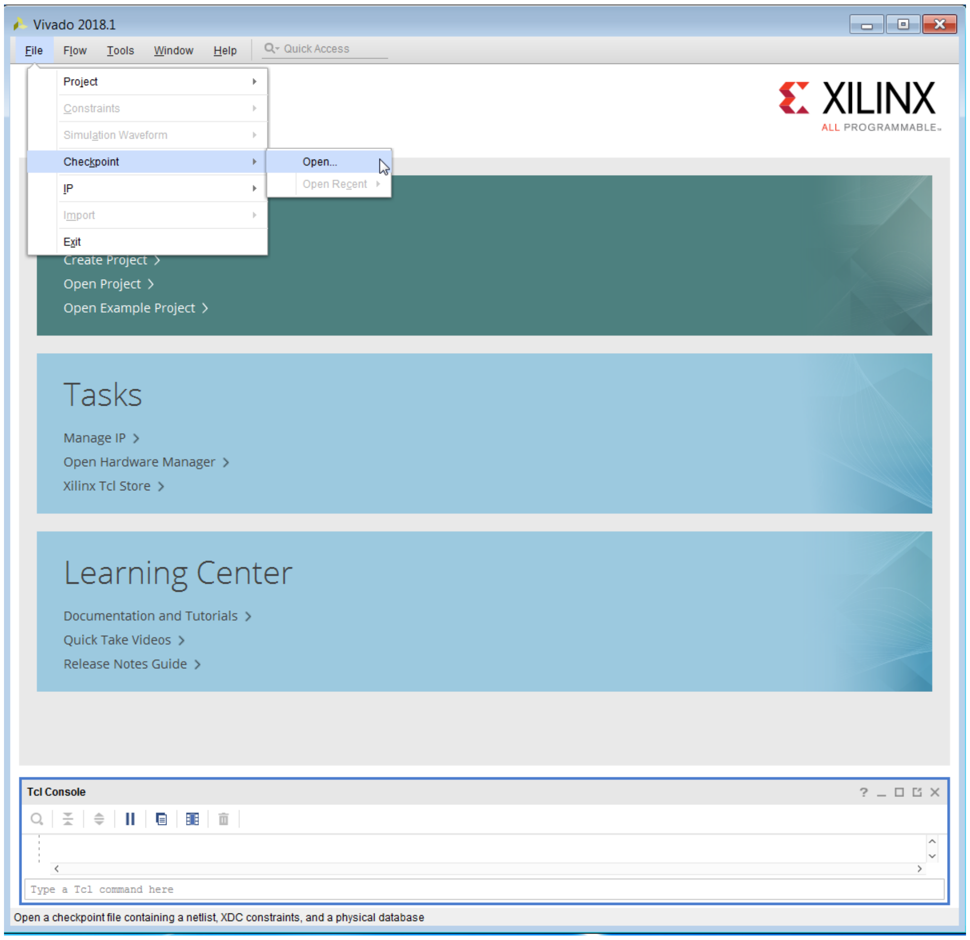
Chapter 7

Lab 6: Using the ECO Flow to Replace Debug Probes Post Implementation

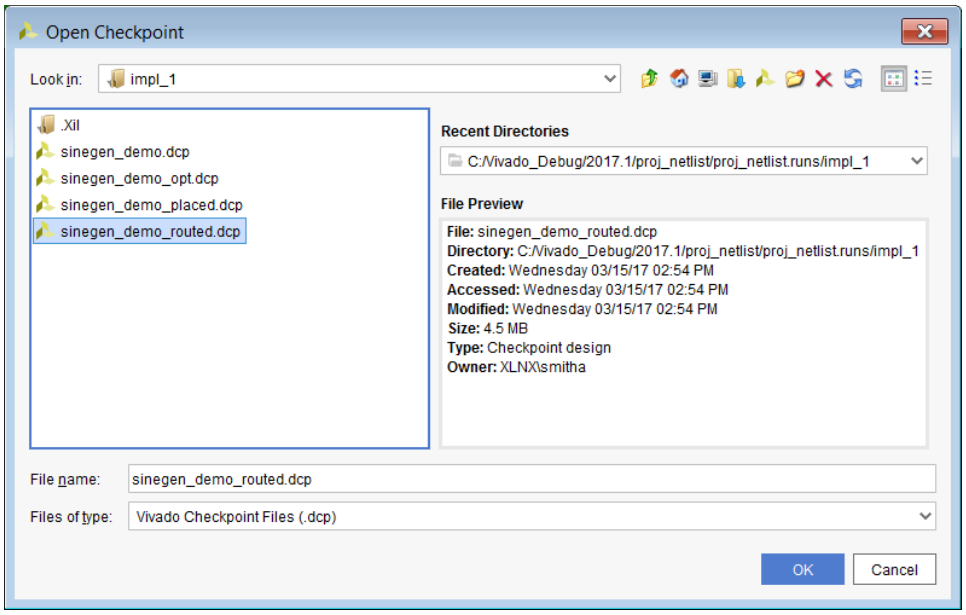
This simple tutorial shows you how to replace nets connected to an ILA core in a placed and routed design checkpoint using the Vivado® Design Suite Engineering Change Order (ECO) flow.

***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 → Open Checkpoint**.



2. Open the routed checkpoint that you created in Chapter 3: Lab 2: Using the HDL Instantiation Method to Debug a Design.

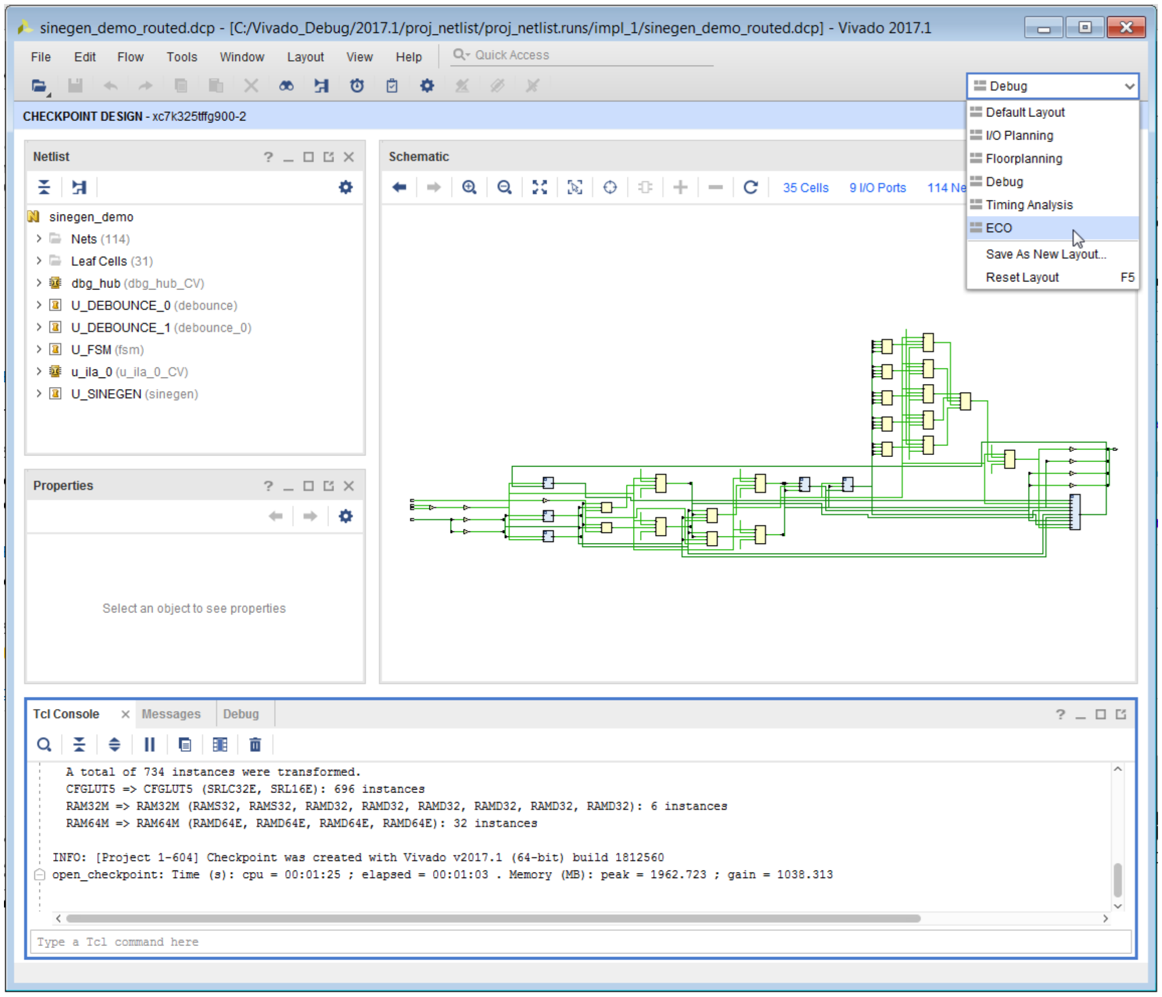


C:/Vivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1

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
Accessed: Wednesday 03/15/17 02:54 PM
Modified: Wednesday 03/15/17 02:54 PM
Size: 4.5 MB
Type: Checkpoint design
Owner: XLNX\smitha

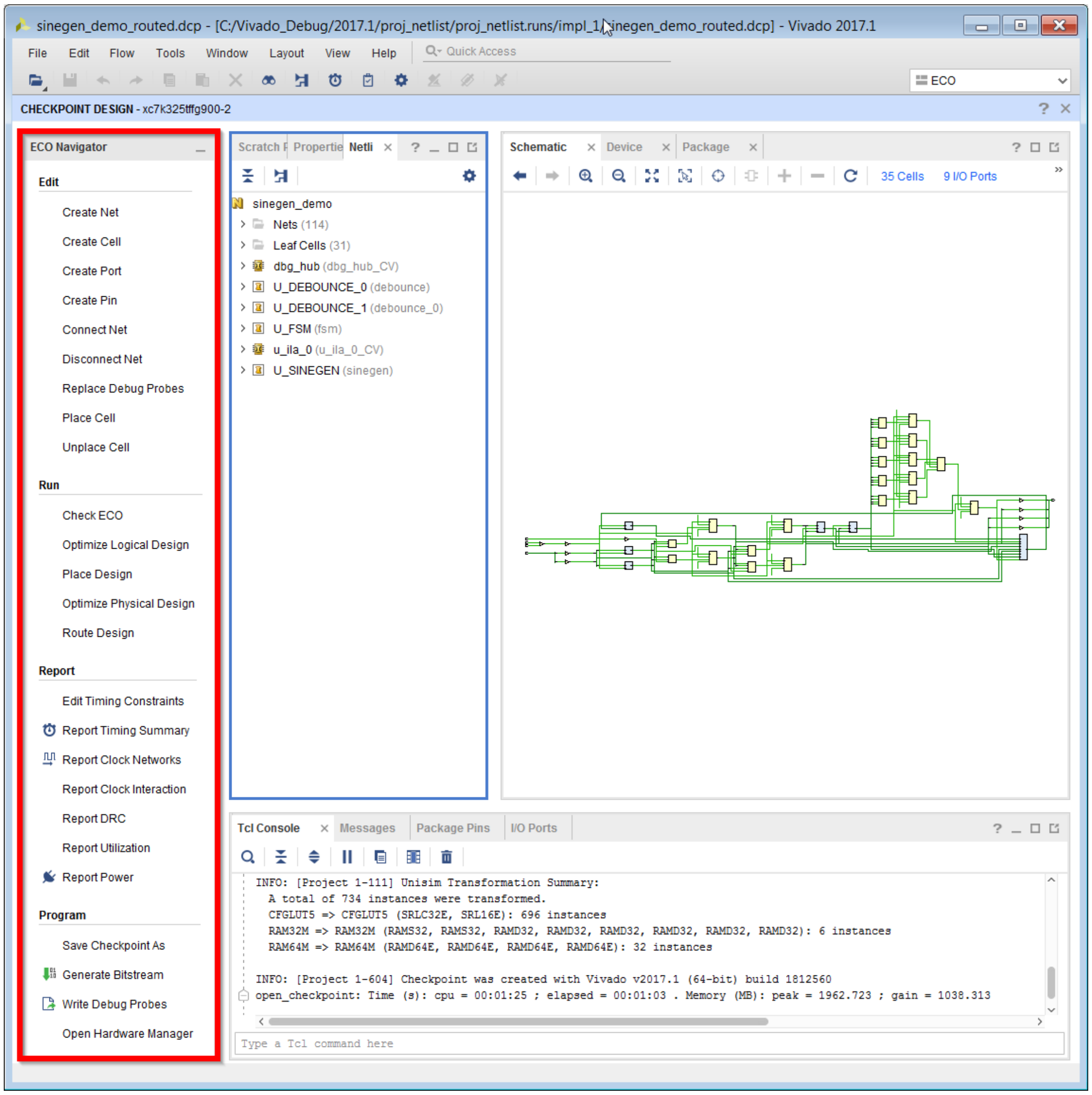
File name: sinegen_demo_routed.dcp
Files of type: Vivado Checkpoint Files (.dcp)

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



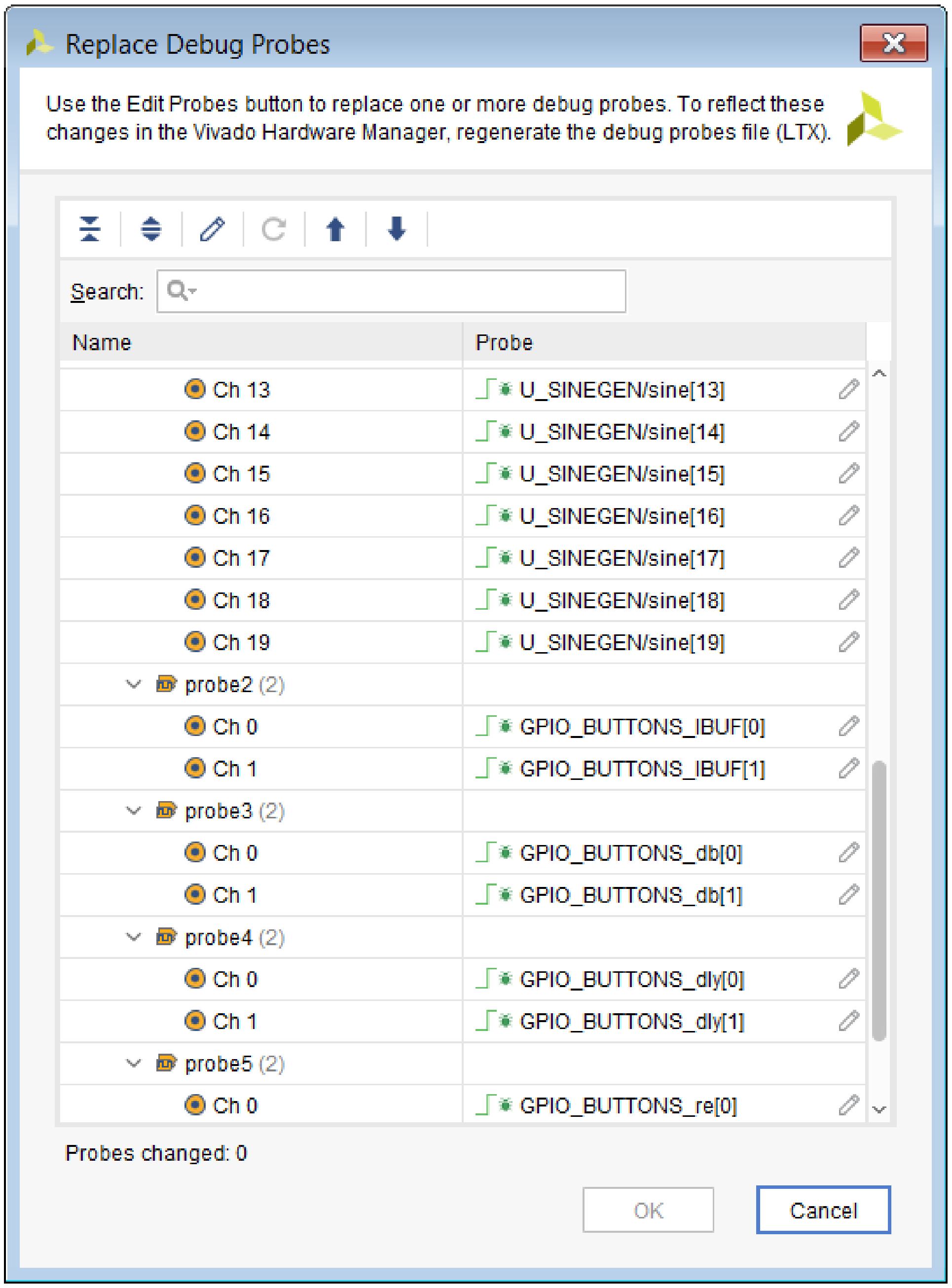
```
  A total of 734 instances were transformed.
  CFGLUT5 => CFGLUT5 (SRLC32E, SRL16E): 696 instances
  RAM32M => RAM32M (RAMS32, RAMS32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32): 6 instances
  RAM64M => RAM64M (RAMD64E, RAMD64E, RAMD64E, RAMD64E): 32 instances

INFO: [Project 1-604] Checkpoint was created with Vivado v2017.1 (64-bit) build 1812560
open_checkpoint: Time (s): cpu = 00:01:25 ; elapsed = 00:01:03 . Memory (MB): peak = 1962.723 ; gain = 1038.313
```

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



```
INFO: [Project 1-111] Unisim Transformation Summary:
  A total of 734 instances were transformed.
  CFGLUT5 => CFGLUT5 (SRLC32E, SRL16E): 696 instances
  RAM32M => RAM32M (RAMS32, RAMS32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32): 6 instances
  RAM64M => RAM64M (RAMD64E, RAMD64E, RAMD64E, RAMD64E): 32 instances

INFO: [Project 1-604] Checkpoint was created with Vivado v2017.1 (64-bit) build 1812560
open_checkpoint: Time (s): cpu = 00:01:25 ; elapsed = 00:01:03 . Memory (MB): peak = 1962.723 ; gain = 1038.313
```

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.



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).

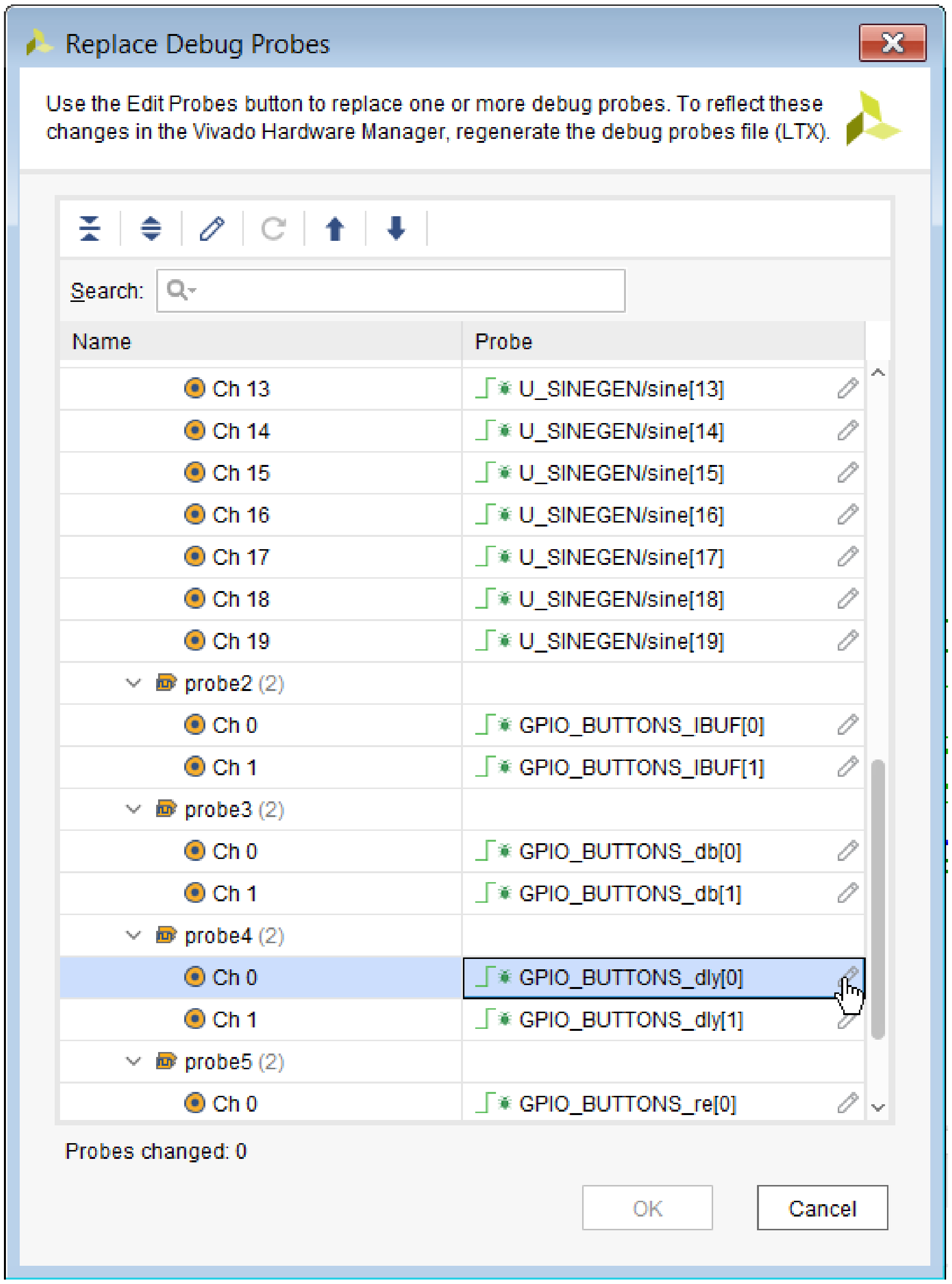
Search:

Name	Probe
Ch 13	U_SINEGEN/sine[13]
Ch 14	U_SINEGEN/sine[14]
Ch 15	U_SINEGEN/sine[15]
Ch 16	U_SINEGEN/sine[16]
Ch 17	U_SINEGEN/sine[17]
Ch 18	U_SINEGEN/sine[18]
Ch 19	U_SINEGEN/sine[19]
probe2 (2)	
Ch 0	GPIO_BUTTONS_IBUF[0]
Ch 1	GPIO_BUTTONS_IBUF[1]
probe3 (2)	
Ch 0	GPIO_BUTTONS_db[0]
Ch 1	GPIO_BUTTONS_db[1]
probe4 (2)	
Ch 0	GPIO_BUTTONS_dly[0]
Ch 1	GPIO_BUTTONS_dly[1]
probe5 (2)	
Ch 0	GPIO_BUTTONS_re[0]

Probes changed: 0

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.

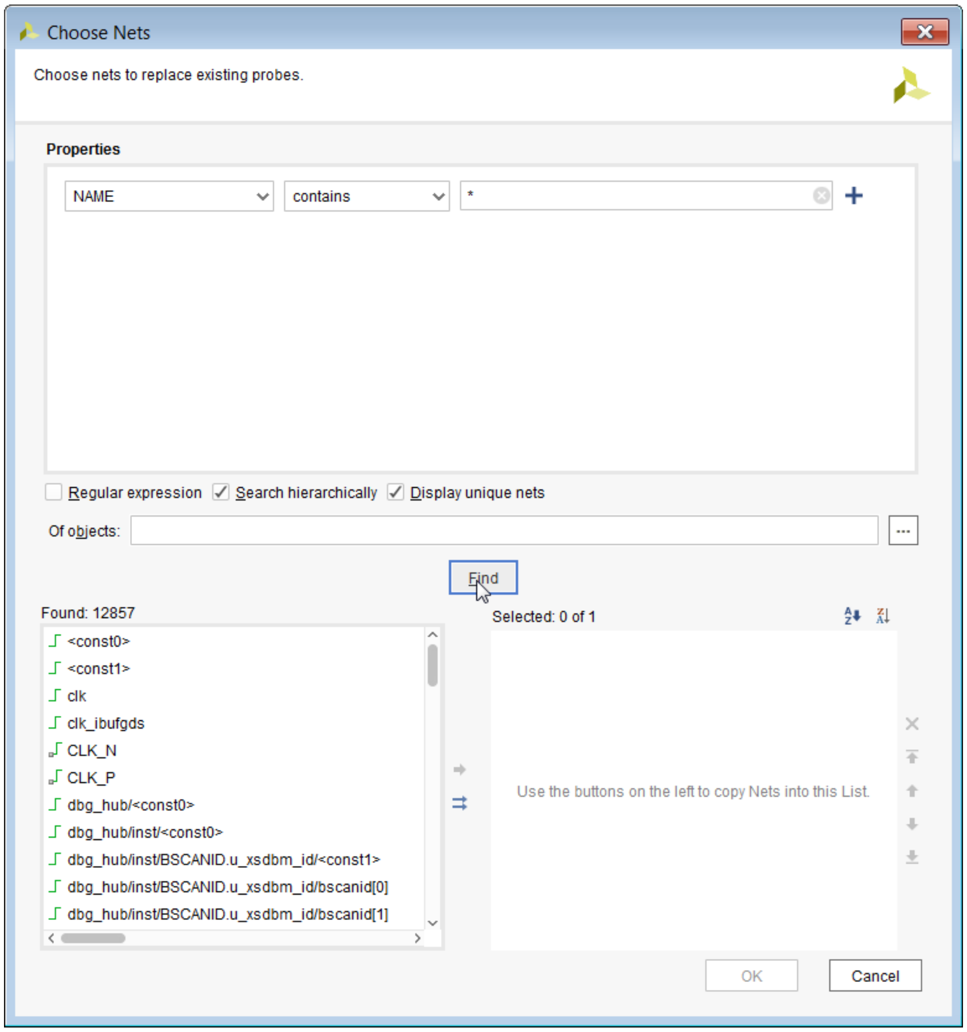


Use the Edit Probes button to replace one or more debug probes. To reflect these changes in the Vivado Hardware Manager, regenerate the debug probes file (LTX).

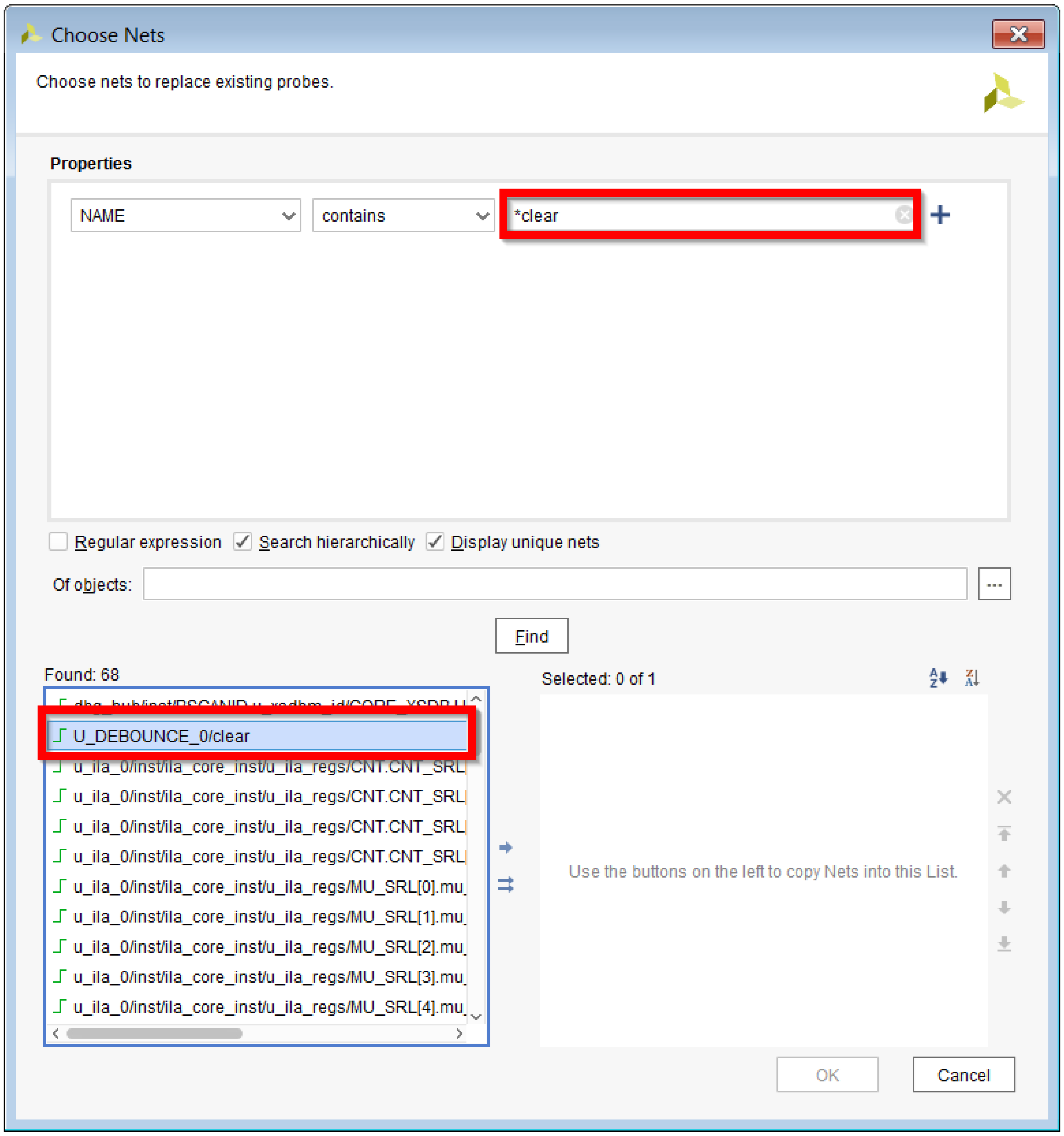
Name	Probe
Ch 13	U_SINEGEN/sine[13]
Ch 14	U_SINEGEN/sine[14]
Ch 15	U_SINEGEN/sine[15]
Ch 16	U_SINEGEN/sine[16]
Ch 17	U_SINEGEN/sine[17]
Ch 18	U_SINEGEN/sine[18]
Ch 19	U_SINEGEN/sine[19]
probe2 (2)	
Ch 0	GPIO_BUTTONS_IBUF[0]
Ch 1	GPIO_BUTTONS_IBUF[1]
probe3 (2)	
Ch 0	GPIO_BUTTONS_db[0]
Ch 1	GPIO_BUTTONS_db[1]
probe4 (2)	
Ch 0	GPIO_BUTTONS_dly[0]
Ch 1	GPIO_BUTTONS_dly[1]
probe5 (2)	
Ch 0	GPIO_BUTTONS_re[0]

Probes changed: 0

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**.



7. 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.

Properties

NAME | contains | *clear

Regular expression | ✓ Search hierarchically | ✓ Display unique nets

Of objects:

Find

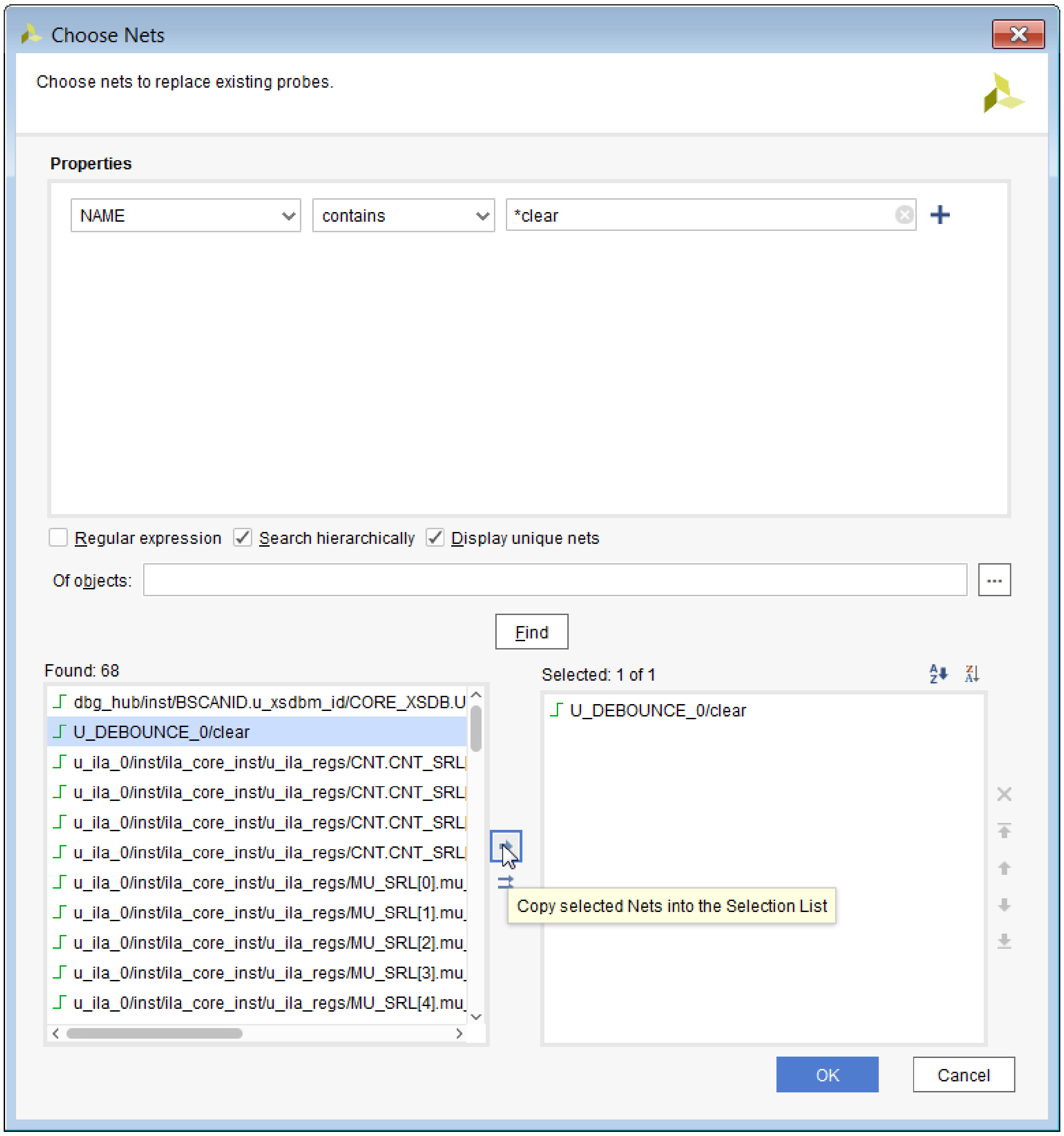
Found: 68

```
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/CNT.CNT_SRL
u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu_
u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[1].mu_
u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu_
u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[3].mu_
u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu_
```

Selected: 0 of 1

Use the buttons on the left to copy Nets into this List.

OK | Cancel



Choose Nets

Choose nets to replace existing probes.

Properties

NAME contains *clear

☐ Regular expression ☑ Search hierarchically ☑ Display unique nets

Of objects:

Find

Found: 68

```
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/CNT.CNT_SRL
u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[0].mu
u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[1].mu
u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[2].mu
u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[3].mu
u_ila_0/inst/ila_core_inst/u_ila_regs/MU_SRL[4].mu
```

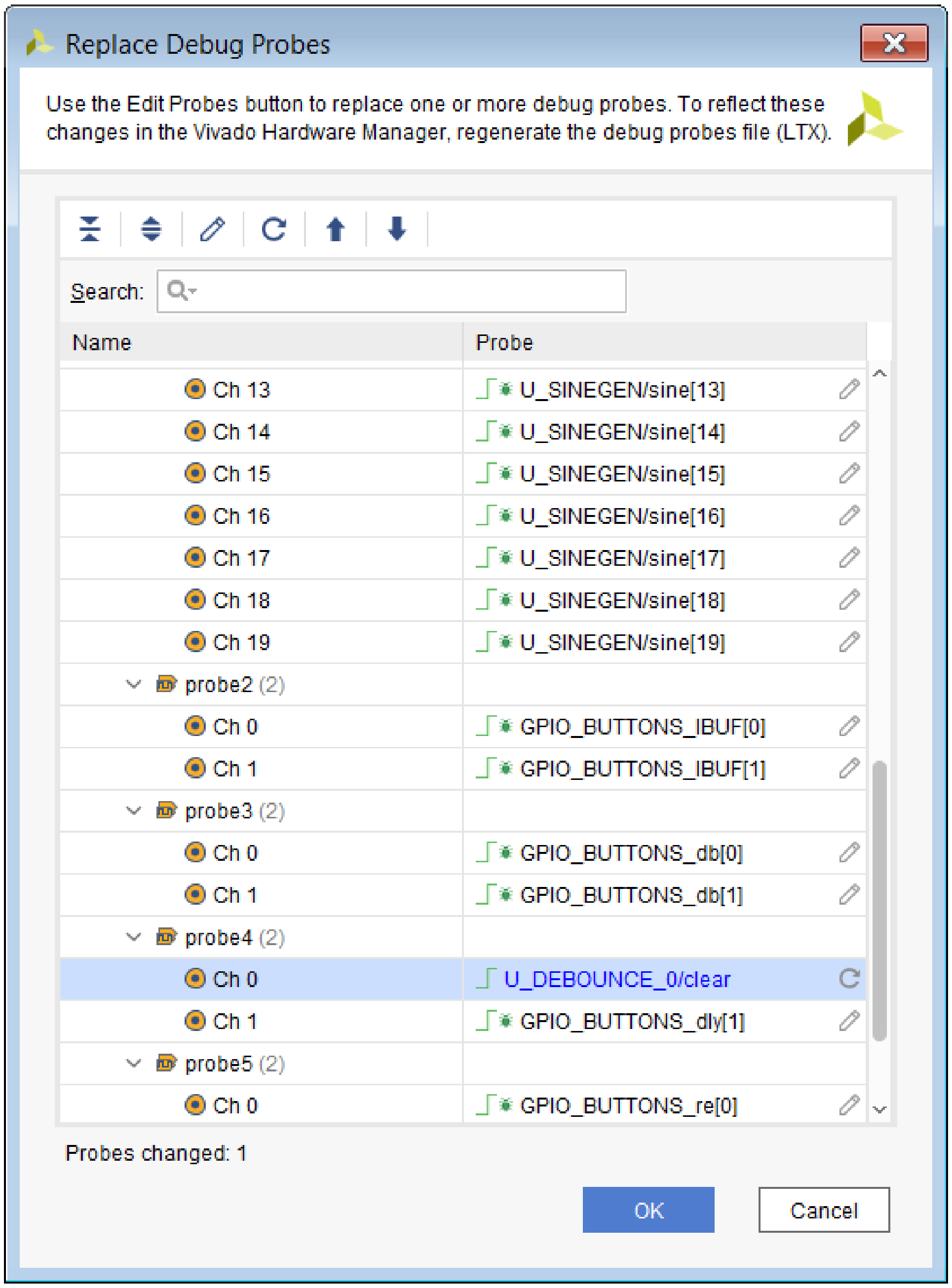
Selected: 1 of 1

U_DEBOUNCE_0/clear

Copy selected Nets into the Selection List

OK 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**.

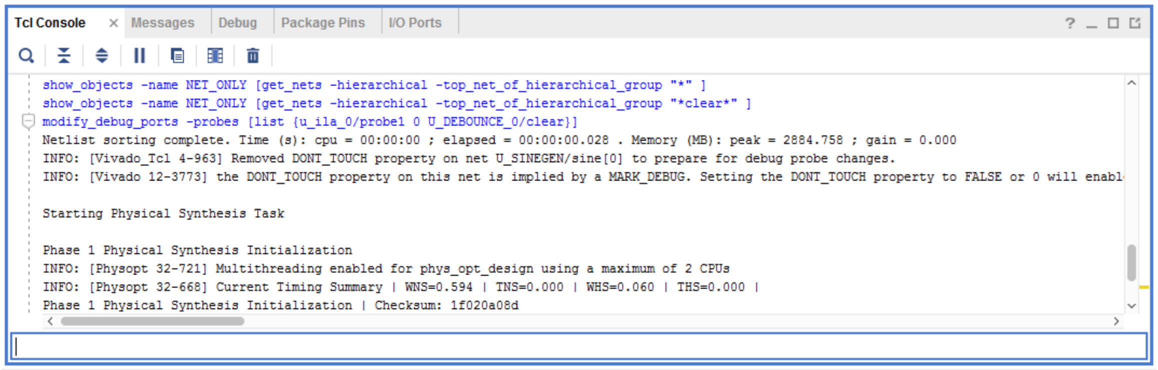


Use the Edit Probes button to replace one or more debug probes. To reflect these changes in the Vivado Hardware Manager, regenerate the debug probes file (LTX).

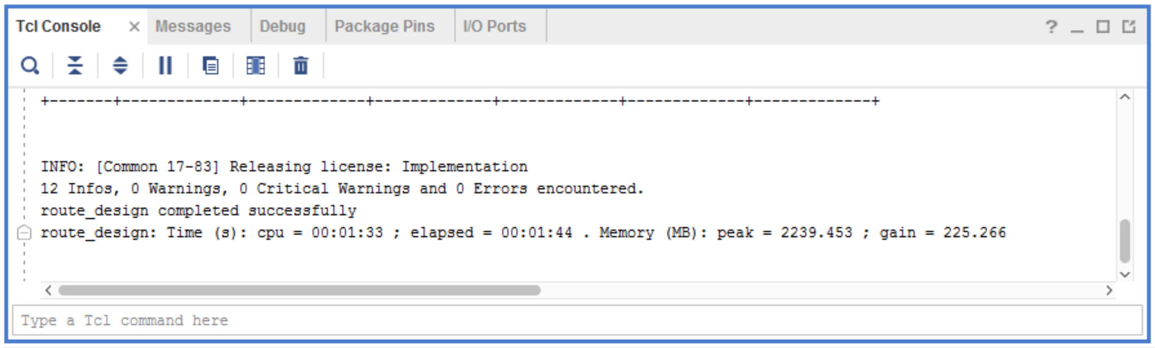
Name	Probe
Ch 13	U_SINEGEN/sine[13]
Ch 14	U_SINEGEN/sine[14]
Ch 15	U_SINEGEN/sine[15]
Ch 16	U_SINEGEN/sine[16]
Ch 17	U_SINEGEN/sine[17]
Ch 18	U_SINEGEN/sine[18]
Ch 19	U_SINEGEN/sine[19]
probe2 (2)	
Ch 0	GPIO_BUTTONS_IBUF[0]
Ch 1	GPIO_BUTTONS_IBUF[1]
probe3 (2)	
Ch 0	GPIO_BUTTONS_db[0]
Ch 1	GPIO_BUTTONS_db[1]
probe4 (2)	
Ch 0	U_DEBOUNCE_0/clear
Ch 1	GPIO_BUTTONS_dly[1]
probe5 (2)	
Ch 0	GPIO_BUTTONS_re[0]

Probes changed: 1

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

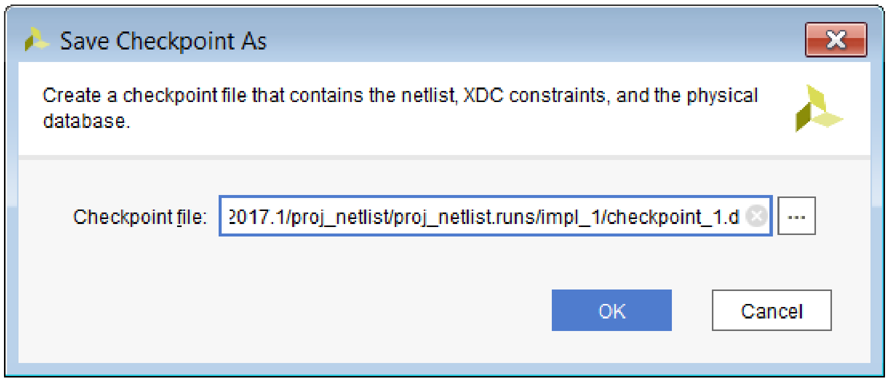


```
show_objects -name NET_ONLY [get_nets -hierarchical -top_net_of_hierarchical_group "*" ]
show_objects -name NET_ONLY [get_nets -hierarchical -top_net_of_hierarchical_group "*clear*" ]
modify_debug_ports -probes [list {u_ila_0/probe1 0 U_DEBOUNCE_0/clear}]
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.028 . Memory (MB): peak = 2884.758 ; gain = 0.000
INFO: [Vivado_Tcl 4-963] Removed DONT_TOUCH property on net U_SINEGEN/sine[0] to prepare for debug probe changes.
INFO: [Vivado 12-3773] the DONT_TOUCH property on this net is implied by a MARK_DEBUG. Setting the DONT_TOUCH property to FALSE or 0 will enabl

Starting Physical Synthesis Task

Phase 1 Physical Synthesis Initialization
INFO: [Physopt 32-721] Multithreading enabled for phys_opt_design using a maximum of 2 CPUs
INFO: [Physopt 32-668] Current Timing Summary | WNS=0.594 | TNS=0.000 | WHS=0.060 | THS=0.000 |
Phase 1 Physical Synthesis Initialization | Checksum: 1f020a08d
```



```
+--------+-------------+-------------+-------------+-------------+-------------+


INFO: [Common 17-83] Releasing license: Implementation
12 Infos, 0 Warnings, 0 Critical Warnings and 0 Errors encountered.
route_design completed successfully
route_design: Time (s): cpu = 00:01:33 ; elapsed = 00:01:44 . Memory (MB): peak = 2239.453 ; gain = 225.266
```

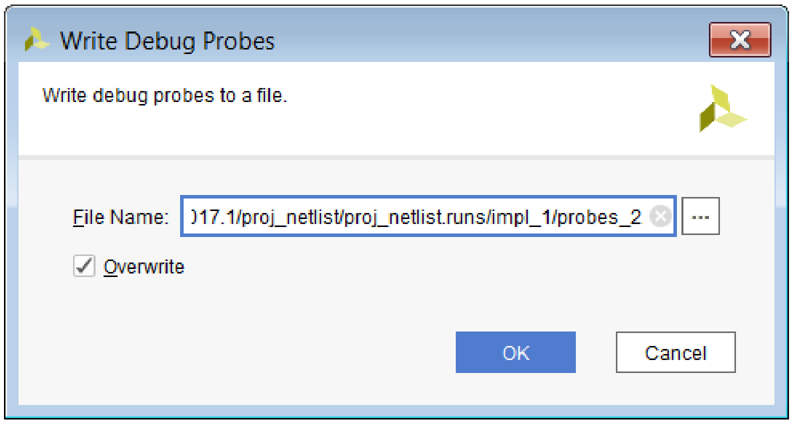
9. Save your modifications to a new checkpoint. Use the Save Checkpoint As option in the ECO Navigator to bring up the Save Checkpoint As dialog box. Specify a file name for the .dcp file and click **OK**.



Create a checkpoint file that contains the netlist, XDC constraints, and the physical database.

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

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

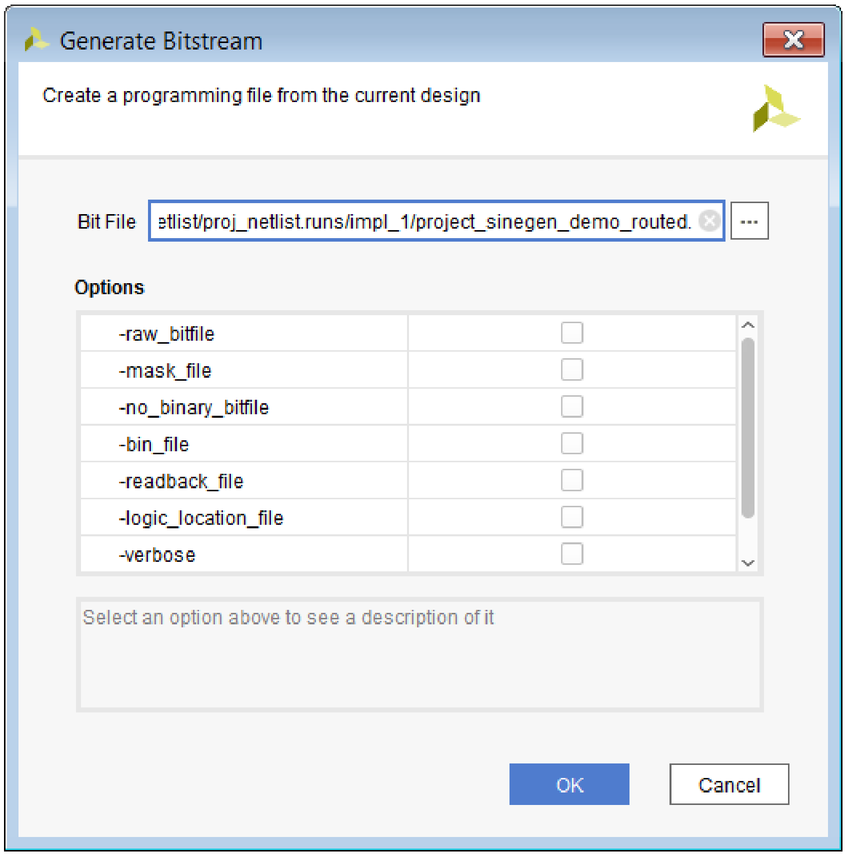

Write Debug Probes

Write debug probes to a file.

File Name:)17.1/proj_netlist/proj_netlist.runs/impl_1/probes_2

✓ Overwrite

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.


Generate Bitstream

Create a programming file from the current design

Bit File: etlist/proj_netlist.runs/impl_1/project_sinegen_demo_routed.

Options

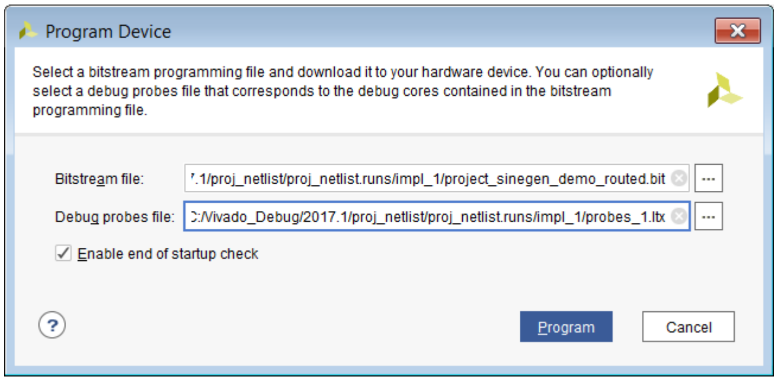
-raw_bitfile	☐
-mask_file	☐
-no_binary_bitfile	☐
-bin_file	☐
-readback_file	☐
-logic_location_file	☐
-verbose	☐

Select an option above to see a description of it

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 Chapter 6: 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.



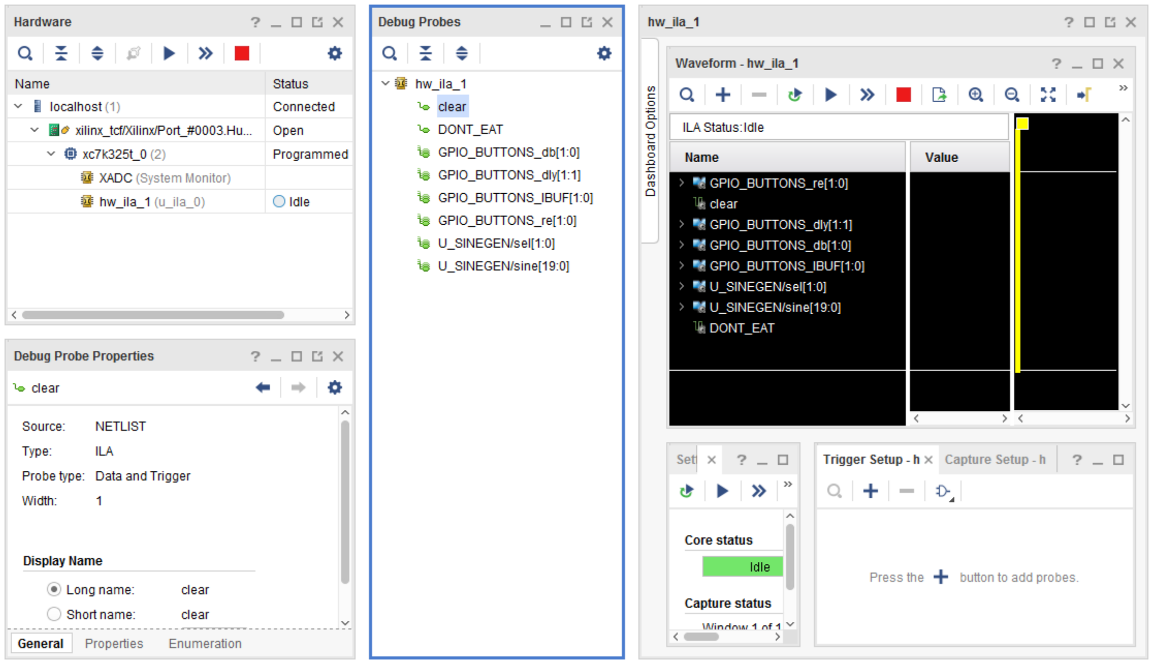
Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file: '.1/proj_netlist/proj_netlist.runs/impl_1/project_sinegen_demo_routed.bit

Debug probes file: :/Vivado_Debug/2017.1/proj_netlist/proj_netlist.runs/impl_1/probes_1.ltx

[✓] Enable end of startup check

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.



Name	Status
localhost (1)	Connected
xilinx_tcf/Xilinx/Port_#0003.Hu...	Open
xc7k325t_0 (2)	Programmed
XADC (System Monitor)	
hw_ila_1 (u_ila_0)	Idle

Debug Probes:
- hw_ila_1
  - clear
  - DONT_EAT
  - GPIO_BUTTONS_db[1:0]
  - GPIO_BUTTONS_dly[1:1]
  - GPIO_BUTTONS_IBUF[1:0]
  - GPIO_BUTTONS_re[1:0]
  - U_SINEGEN/sel[1:0]
  - U_SINEGEN/sine[19:0]

Debug Probe Properties: clear

Source: NETLIST
Type: ILA
Probe type: Data and Trigger
Width: 1

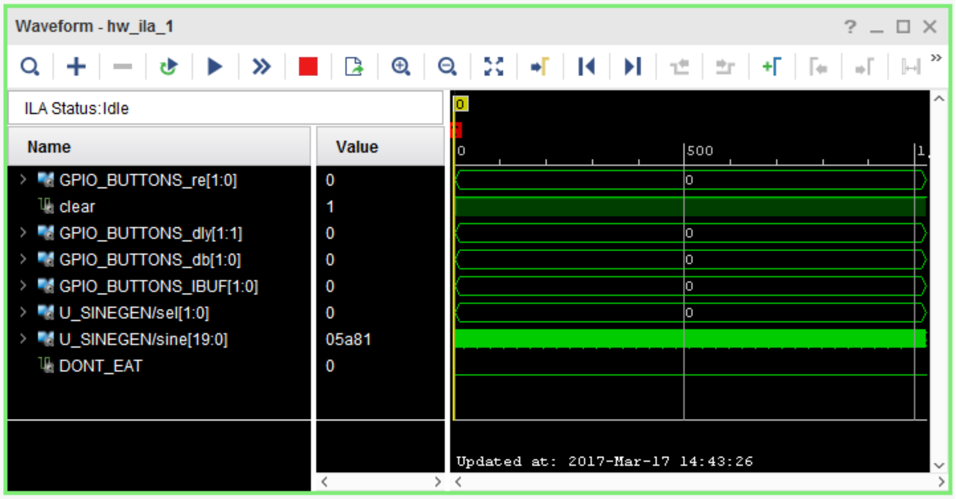
Display Name
Long name: clear
Short name: clear

ILA Status: Idle

Name	Value
GPIO_BUTTONS_re[1:0]	
clear	
GPIO_BUTTONS_dly[1:1]	
GPIO_BUTTONS_db[1:0]	
GPIO_BUTTONS_IBUF[1:0]	
U_SINEGEN/sel[1:0]	
U_SINEGEN/sine[19:0]	
DONT_EAT	

Core status: Idle
Capture status: Window 1 of 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.



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
U_SINEGEN/sel[1:0]	0
U_SINEGEN/sine[19:0]	05a81
DONT_EAT	0

```
Updated at: 2017-Mar-17 14:43:26
```

Lab 7: Debugging Designs Using the Incremental Compile Flow

This lab introduces the Vivado® 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 Vivado IDE

   Load Vivado IDE by doing one of the following:

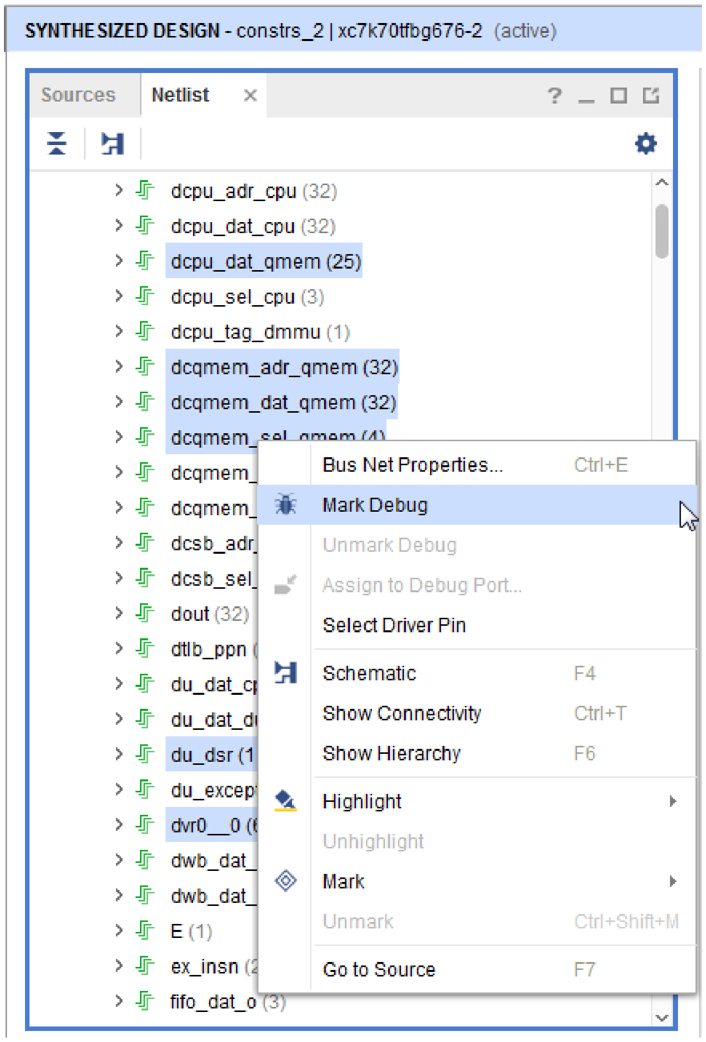
   - Double-click the Vivado IDE icon on the Windows desktop.
   - Type `vivado` in a command terminal.

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

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

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

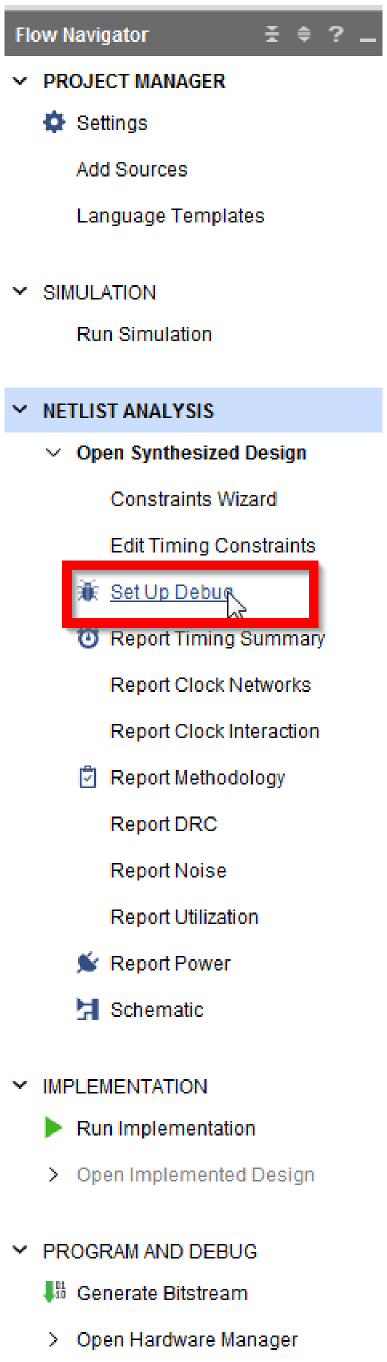
```
cpuEngine/dcqmem_dat_qmem[*],
cpuEngine/dcpu_dat_qmem[*],
cpuEngine/dcqmem_adr_qmem[*],
cpuEngine/du_dsr[*],
cpuEngine/dvr0__0[*],
cpuEngine/du_dsr[*],
cpuEngine/dcqmem_sel_qmem[*]
```



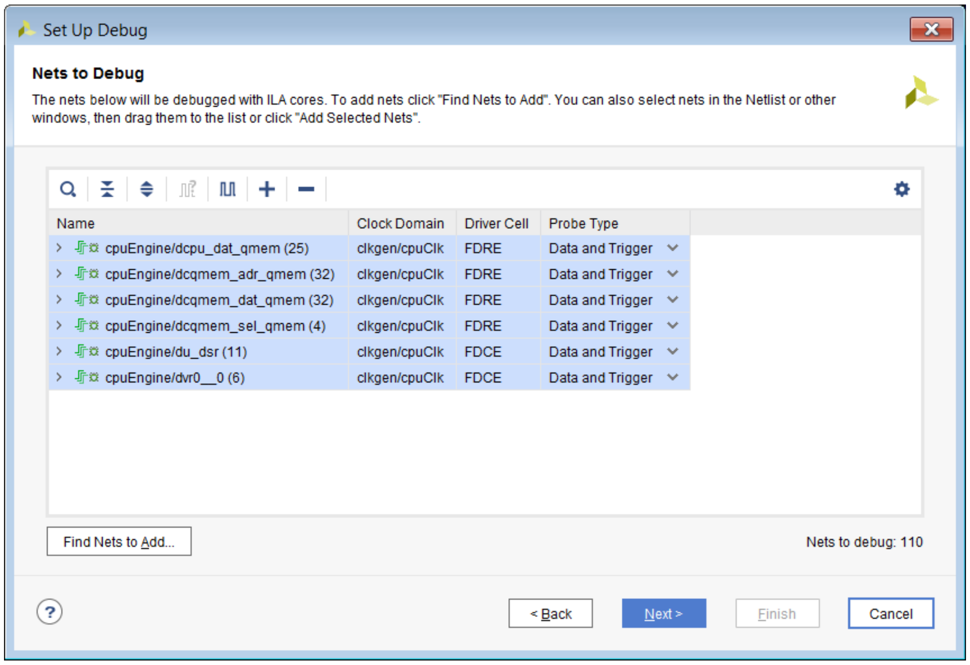
Alternatively use can use the Tcl command below to set the MARK_DEBUG property on the signals specified.

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

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



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



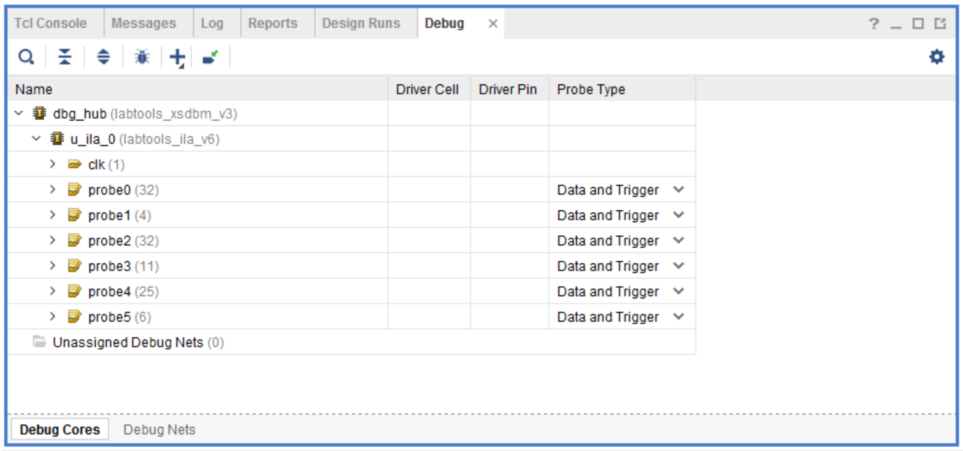
Nets to Debug

The nets below will be debugged with ILA cores. To add nets click "Find Nets to Add". You can also select nets in the Netlist or other windows, then drag them to the list or click "Add Selected Nets".

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
cpuEngine/du_dsr (11)	clkgen/cpuClk	FDCE	Data and Trigger
cpuEngine/dvr0__0 (6)	clkgen/cpuClk	FDCE	Data and Trigger

Nets to debug: 110

11. When ILA Core Options screen appears, click **Next** again.
12. When Set Up Debug Summary screen appears, ensure that 1 debug core is created and click **Finish**.
13. Check the Debug widow to ensure that u_ila_0 core has been inserted into the design.



Name	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 figure below. The run directory contains the routed checkpoint (`top_routed.dcp`) to be used later for the incremental compile flow. The location of the implementation run directory is a property of the run.
5. Get the location of the current run directory in the Tcl Console by typing:

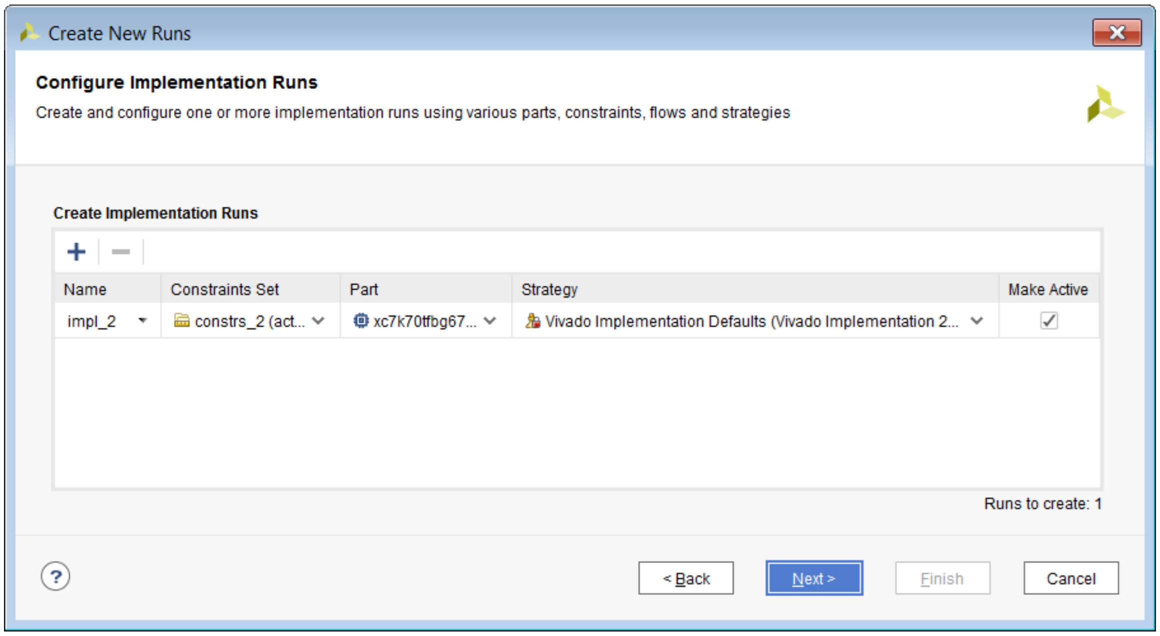
```
get_property DIRECTORY [current_run]
```

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

Step 3: Create New Runs

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

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


Create New Runs

Configure Implementation Runs

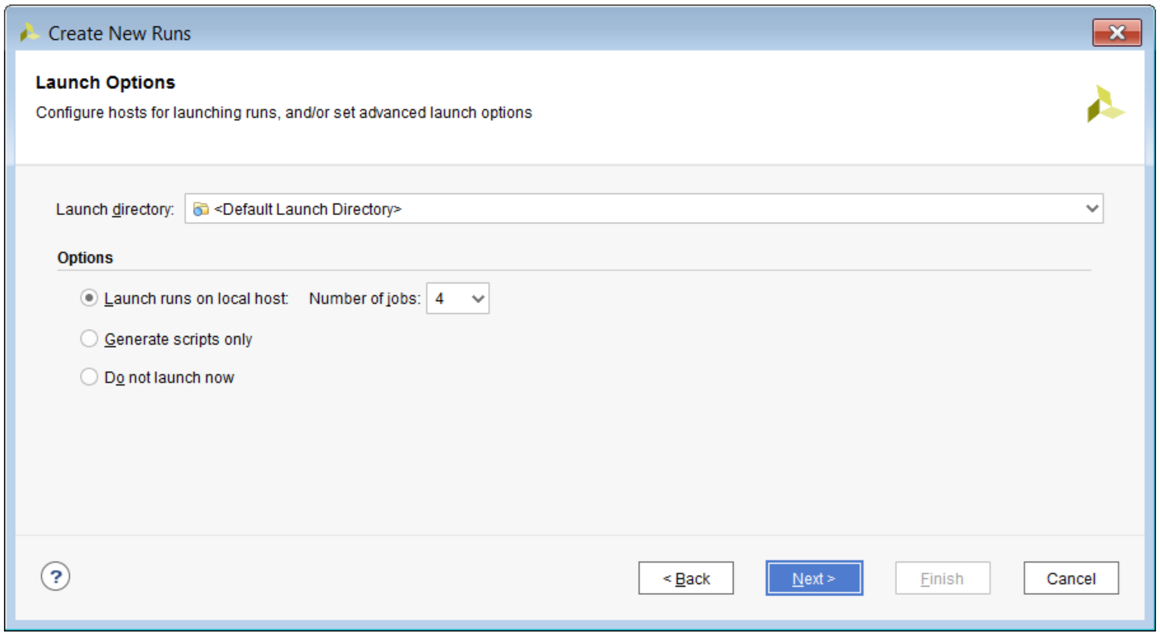
Create and configure one or more implementation runs using various parts, constraints, flows and strategies

Create Implementation Runs

Name	Constraints Set	Part	Strategy	Make Active
impl_2	constrs_2 (act...	xc7k70tfbg67...	Vivado Implementation Defaults (Vivado Implementation 2...	✓

Runs to create: 1

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


Create New Runs

Launch Options

Configure hosts for launching runs, and/or set advanced launch options

Launch directory: <Default Launch Directory>

Options

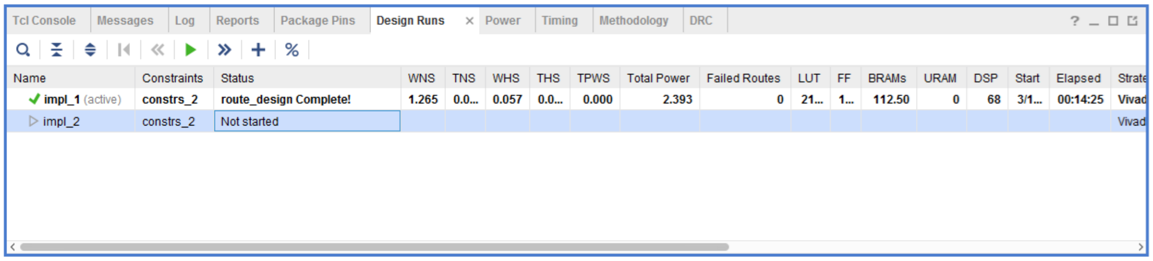
[● Launch runs on local host:] Number of jobs: 4

○ Generate scripts only

○ Do not launch now

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.

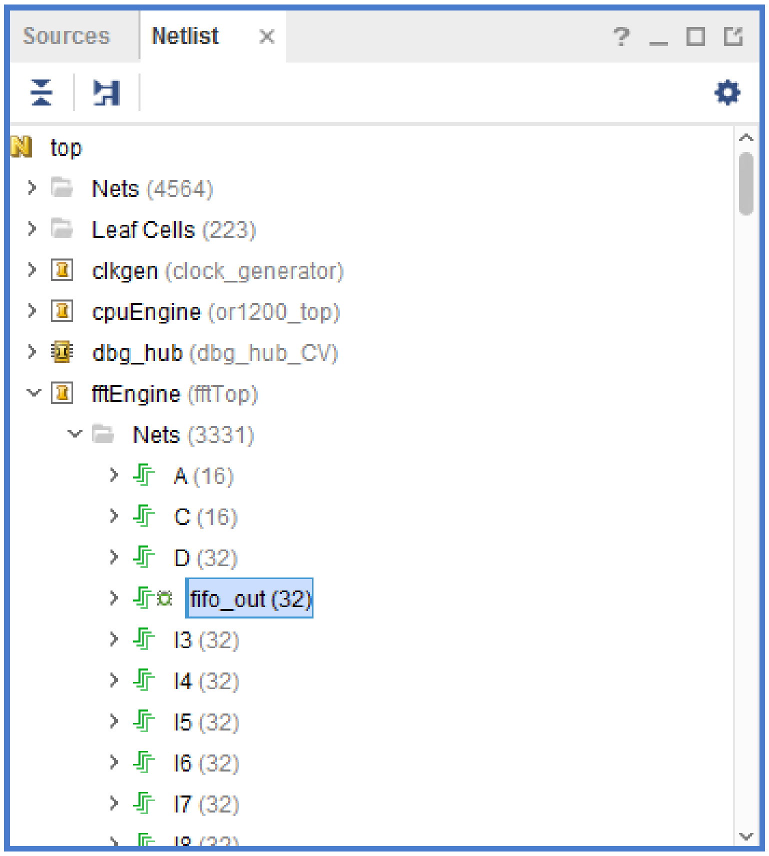


Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Strate
✓ impl_1 (active)	constrs_2	route_design Complete!	1.265	0.0...	0.057	0.0...	0.000	2.393	0	21...	1...	112.50	0	68	3/1...	00:14:25	Vivad
impl_2	constrs_2	Not started															Vivad

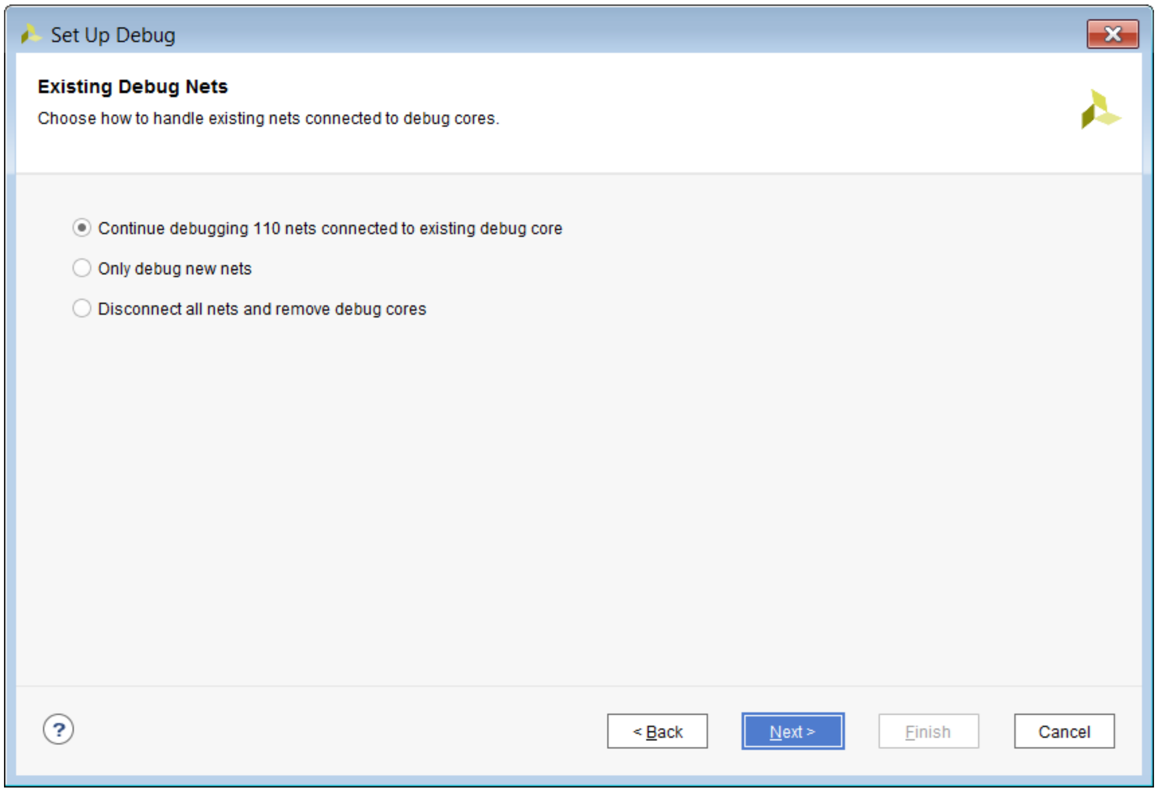
Step 4: Making Incremental Debug Changes

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.
2. For this tutorial, assume that you now need to debug some other nets in addition to the ones already being debugged. However, you want to reuse the previous place and route results. So now, you will debug the nets `fftEngine/fifo_out[*]`
3. Apply the MARK_DEBUG property to this bus in the netlist window.



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



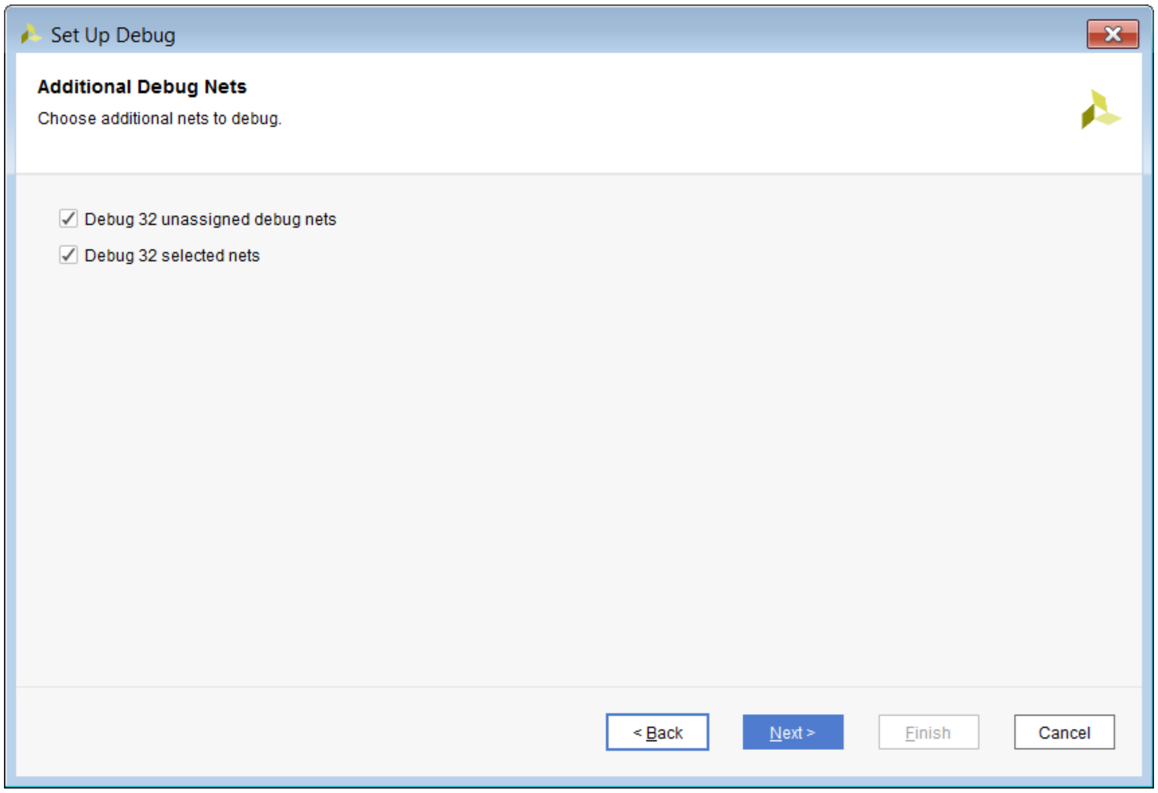
Set Up Debug

Existing Debug Nets

Choose how to handle existing nets connected to debug cores.

- [✓ Continue debugging 110 nets connected to existing debug core]
- Only debug new nets
- Disconnect all nets and remove debug cores

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

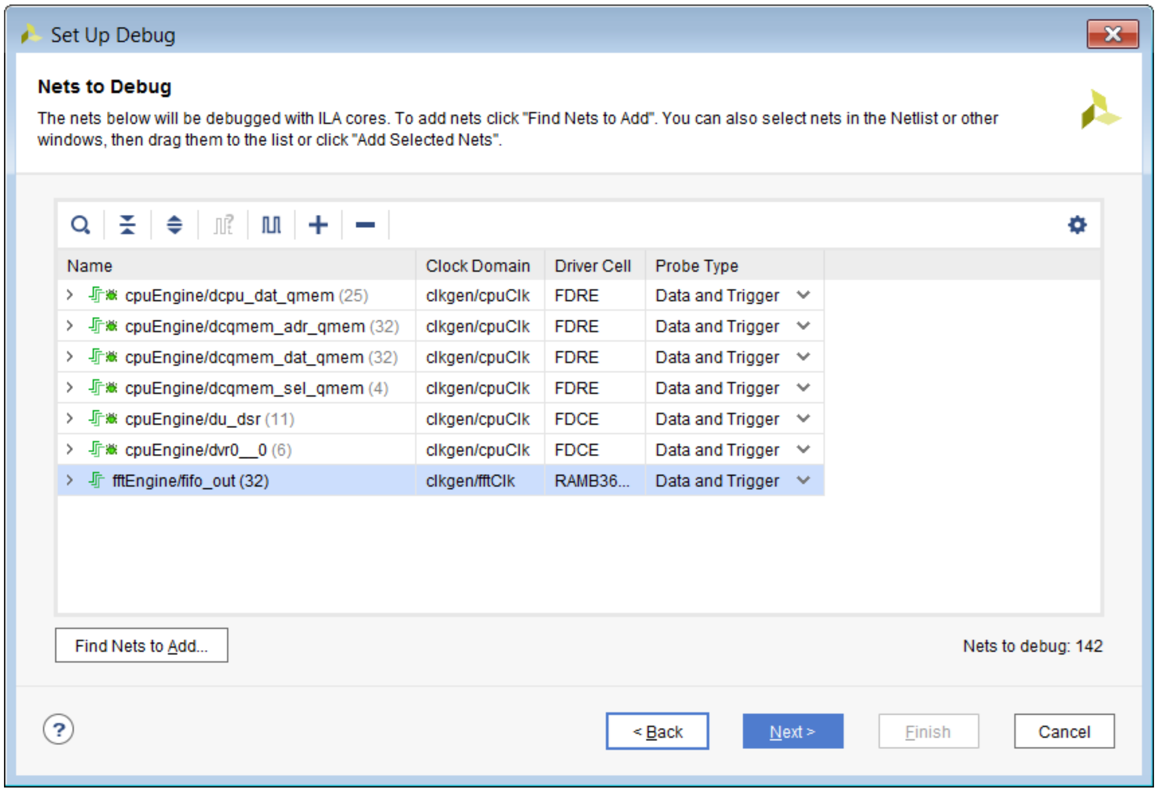

Set Up Debug

Additional Debug Nets

Choose additional nets to debug.

- ☑ Debug 32 unassigned debug nets
- ☑ Debug 32 selected nets

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


Set Up Debug

Nets to Debug

The nets below will be debugged with ILA cores. To add nets click "Find Nets to Add". You can also select nets in the Netlist or other windows, then drag them to the list or click "Add Selected Nets".

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
cpuEngine/du_dsr (11)	clkgen/cpuClk	FDCE	Data and Trigger
cpuEngine/dvr0__0 (6)	clkgen/cpuClk	FDCE	Data and Trigger
fftEngine/fifo_out (32)	clkgen/fftClk	RAMB36...	Data and Trigger

Nets to debug: 142

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

9. 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:

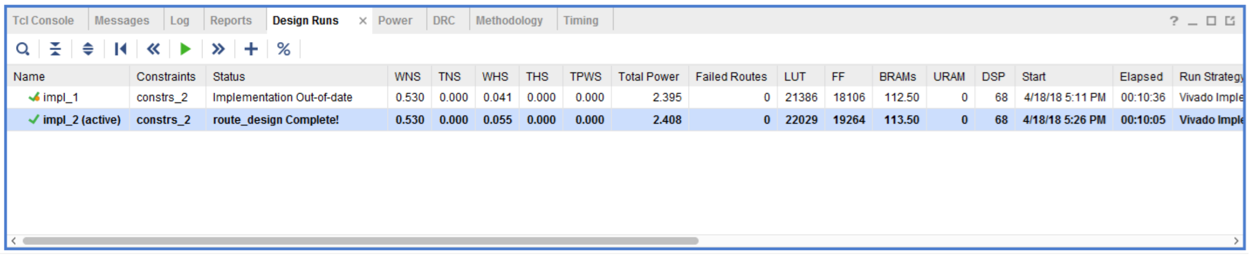
```
get_property INCREMENTAL_CHECKPOINT [current_run]
```

   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.

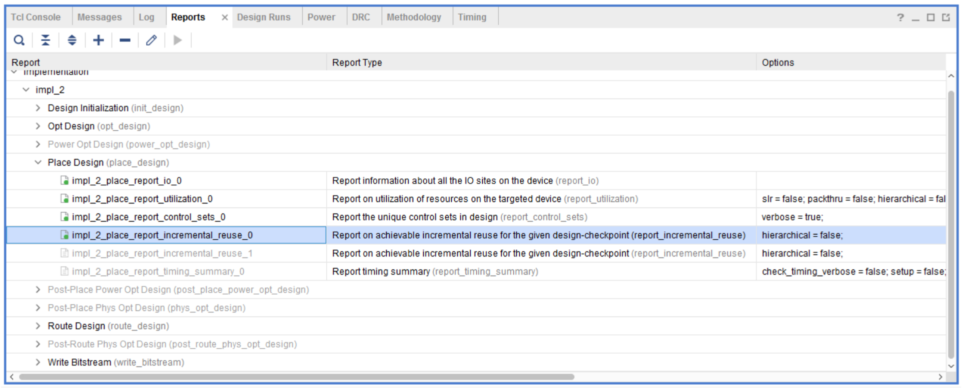
   This runs implementation on the current run, using the`top_routed.dcp` file as the reference design for the incremental compile flow. When the run is finished, the Implementation Completed dialog box opens.
7. Select Open Implemented Design and click **OK**. As shown in the following figure, the Design Runs window shows the elapsed time for implementation run impl_2 versus impl_1.



Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
✓ 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 Imple
✓ **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 Imple**

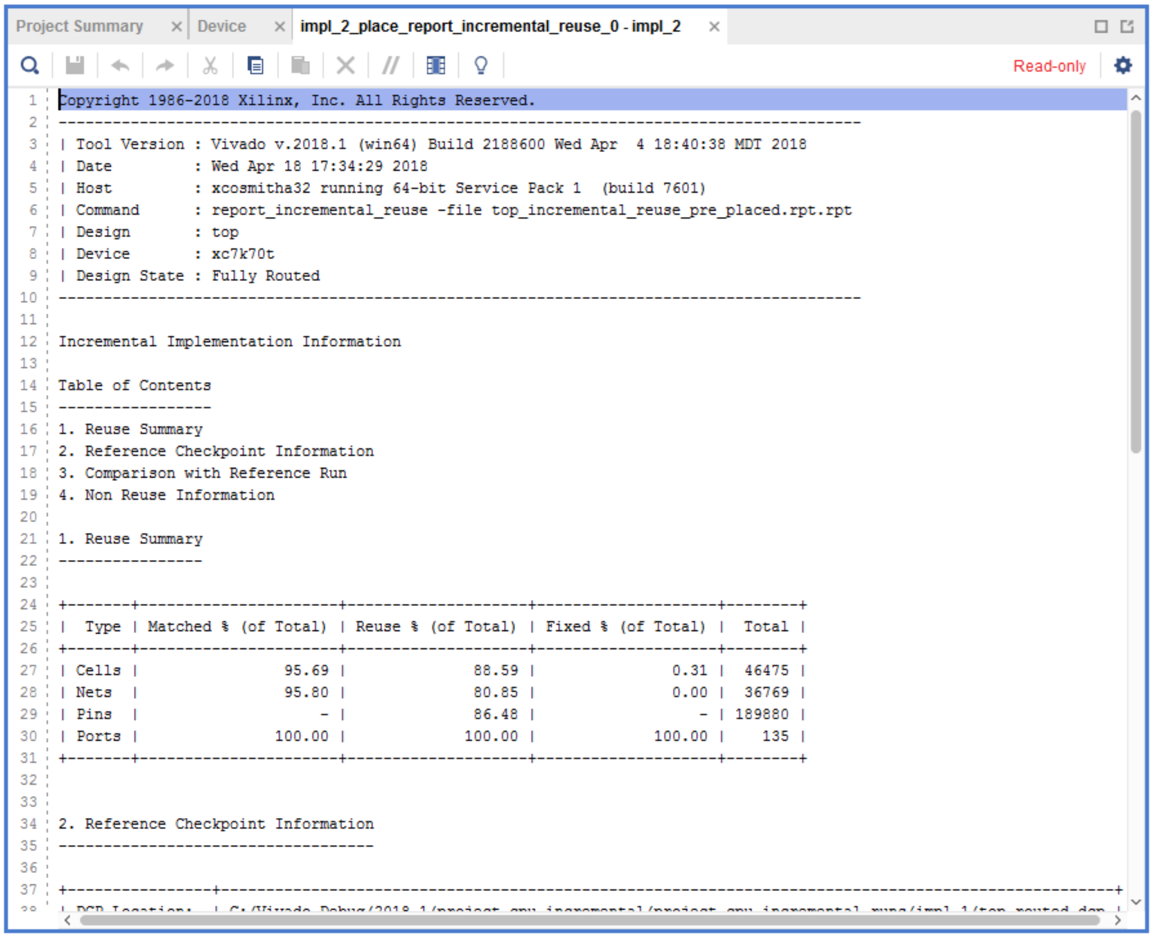
***Note*:** This is an extremely small design. The advantages of the incremental compile flow are greater and significant with larger, more complex designs.

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.



Report	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 = fal
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.



```
Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------------------------
| Tool Version : Vivado v.2018.1 (win64) Build 2188600 Wed Apr  4 18:40:38 MDT 2018
| Date         : Wed Apr 18 17:34:29 2018
| Host         : xcosmitha32 running 64-bit Service Pack 1  (build 7601)
| Command      : report_incremental_reuse -file top_incremental_reuse_pre_placed.rpt.rpt
| Design       : top
| Device       : xc7k70t
| Design State : Fully Routed
------------------------------------------------------------------------------------------

Incremental Implementation Information

Table of Contents
-----------------
1. Reuse Summary
2. Reference Checkpoint Information
3. Comparison with Reference Run
4. Non Reuse Information

1. Reuse Summary
----------------

+-------+---------------------+--------------------+--------------------+--------+
|  Type | Matched % (of Total) | Reuse % (of Total) | Fixed % (of Total) |  Total |
+-------+---------------------+--------------------+--------------------+--------+
| Cells |               95.69 |              88.59 |               0.31 |  46475 |
| Nets  |               95.80 |              80.85 |               0.00 |  36769 |
| Pins  |                   - |              86.48 |                  - | 189880 |
| Ports |              100.00 |             100.00 |             100.00 |    135 |
+-------+---------------------+--------------------+--------------------+--------+


2. Reference Checkpoint Information
-----------------------------------


+----------------+--------------------------------------------------------------------------------------+
| DCP Location:  | C:/Vivado_Debug/2018.1/project_cpu_incremental/project_cpu_incremental.runs/impl_1/top_routed.dcp |
```

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.

Chapter 9

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.

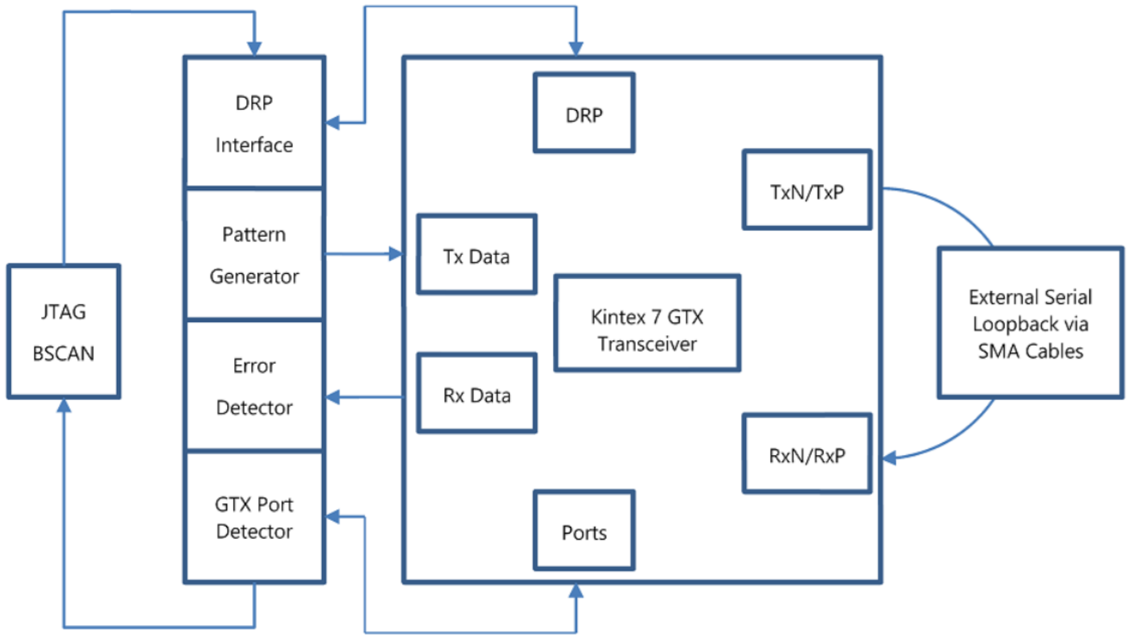
Design Description

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.

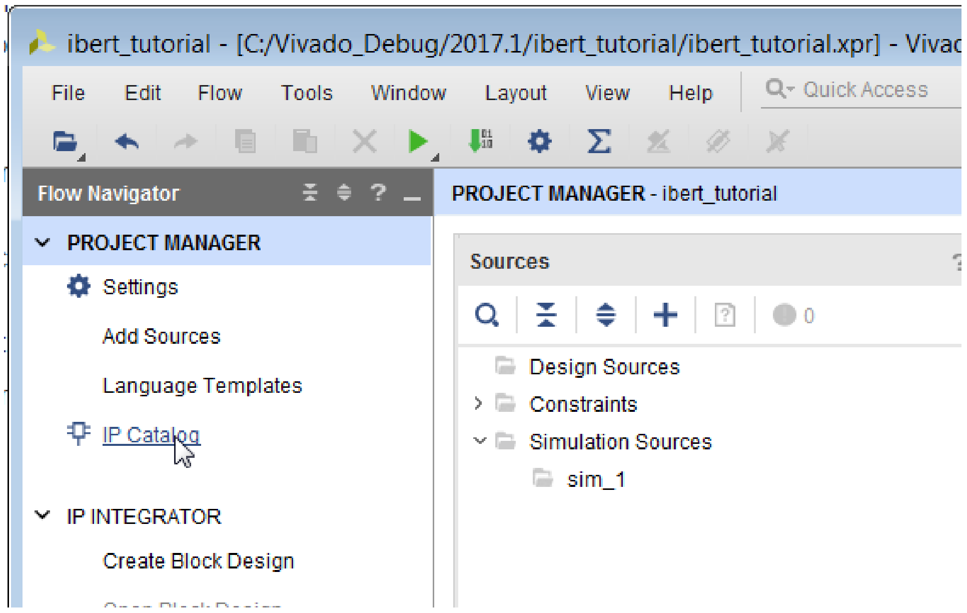
2. In the Quick Start screen, click Create Project to start the New Project wizard, and click **Next**.
3. In the Project Name page, name the new project ibert_tutorial and provide the project location (`C:/ibert_tutorial`). Ensure that **Create Project Subdirectory** is selected. Click **Next**.
4. 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.

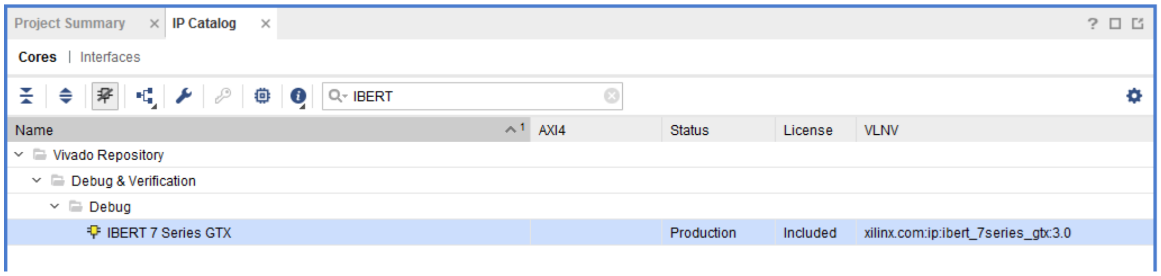
Step 2: Adding an IBERT core to the Vivado Project

1. In the Flow Navigator click **IP Catalog**.

   The IP Catalog opens.

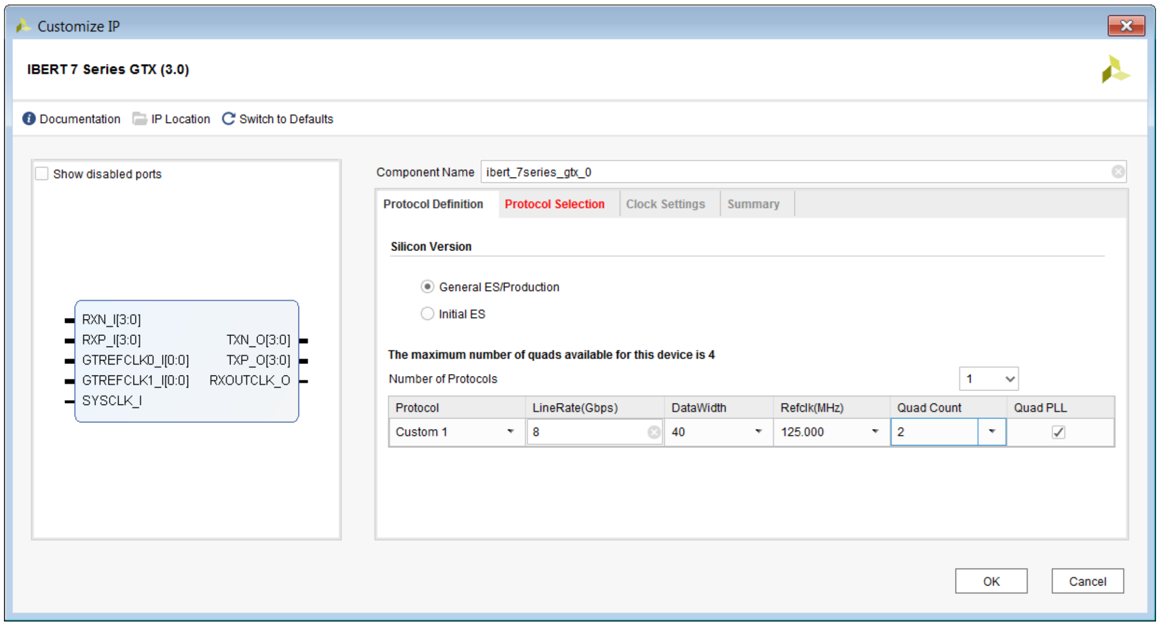


2. In the search field of the IP Catalog type `IBERT`, to display the IBERT 7 series GTX IP.



Name	AXI4	Status	License	VLNV
Vivado Repository				
Debug & Verification				
Debug				
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.



Component Name: ibert_7series_gtx_0

Silicon Version

(● General ES/Production) ○ Initial ES

The maximum number of quads available for this device is 4

Number of Protocols: 1

Protocol	LineRate(Gbps)	DataWidth	Refclk(MHz)	Quad Count	Quad PLL
Custom 1	8	40	125.000	2	✓

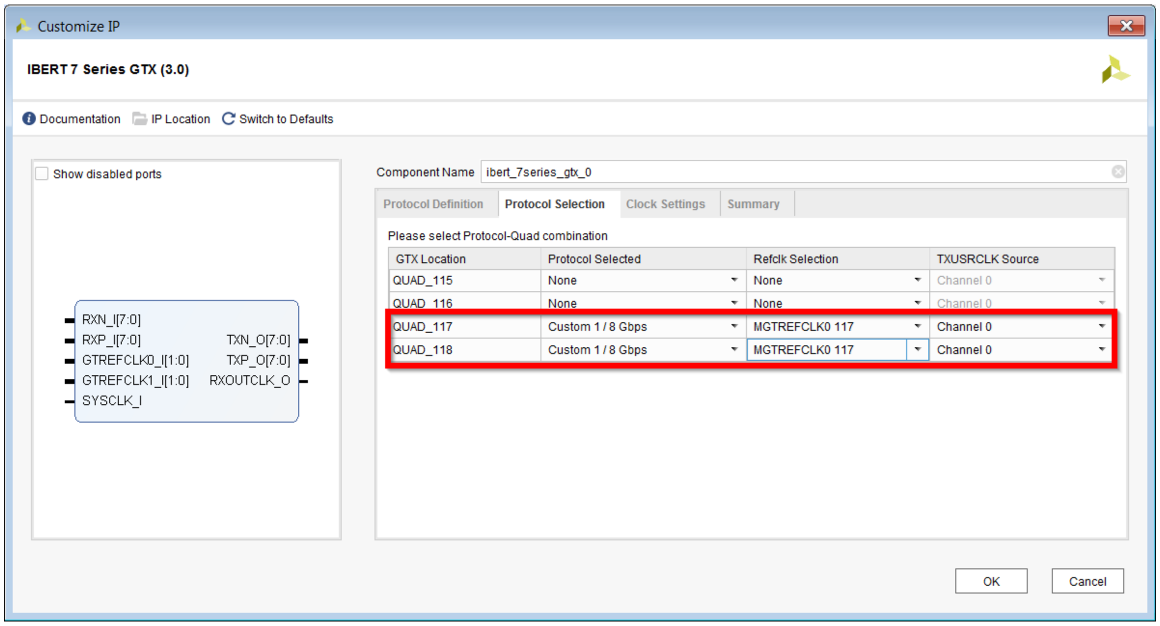
5. Under the Protocol Selection tab, update the following selections:
   a. For GTX Location QUAD_117, in the Protocol Selected column, click the pull-down menu and select **Custom 1 / 8 Gbps**. This should automatically populate Refclk Selection to MGTREFCLK0 117 and TXUSRCLK Source to Channel 0.

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.



Component Name ibert_7series_gtx_0

Please select Protocol-Quad combination

GTX Location	Protocol Selected	Refclk Selection	TXUSRCLK Source
QUAD_115	None	None	Channel 0
QUAD_116	None	None	Channel 0
QUAD_117	Custom 1 / 8 Gbps	MGTREFCLK0 117	Channel 0
QUAD_118	Custom 1 / 8 Gbps	MGTREFCLK0 117	Channel 0

6. Click the **Clock Settings tab** and make the following changes for both QUAD_117 and QUAD_118:

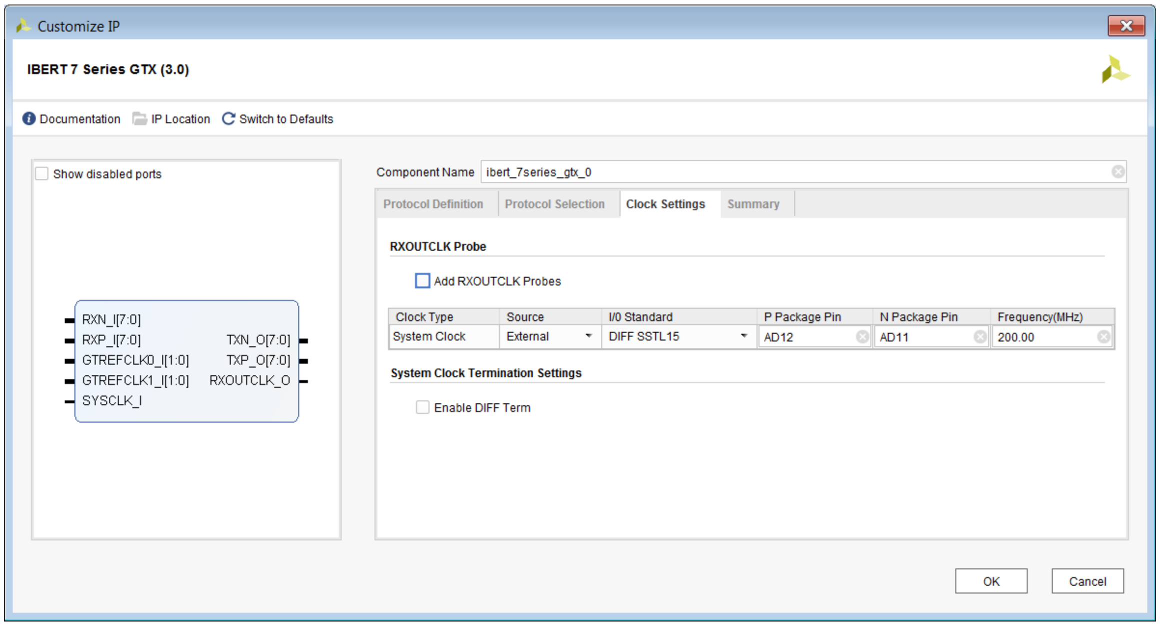
   a. Leave the Source column at its default value of External.

   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.



Component Name ibert_7series_gtx_0

RXOUTCLK Probe

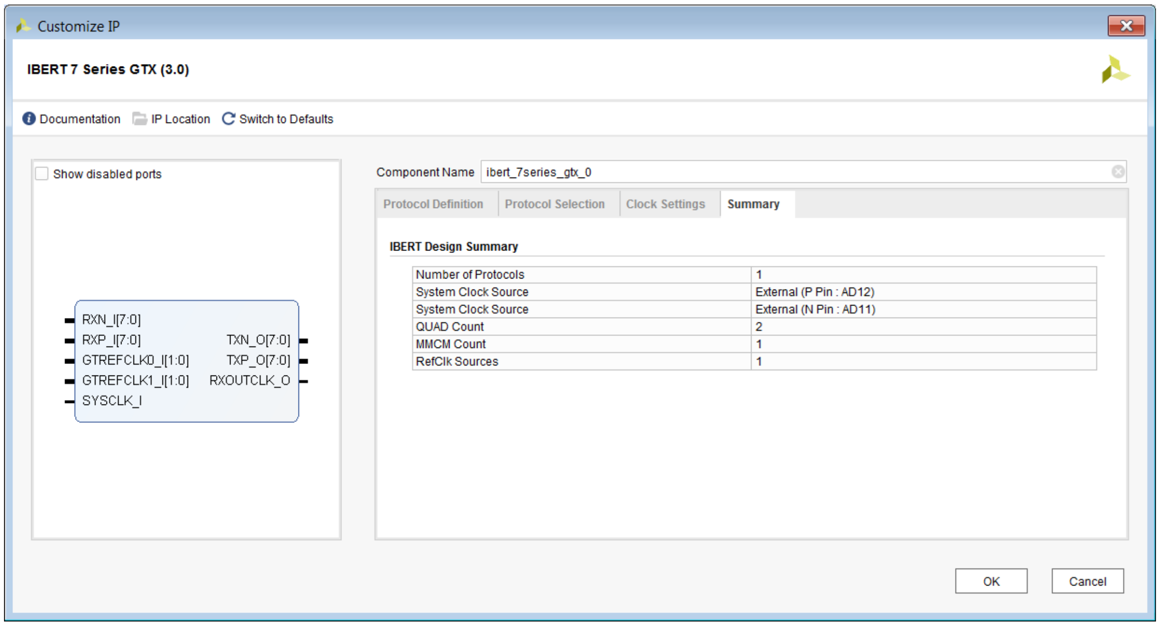
Add RXOUTCLK Probes

Clock Type	Source	I/0 Standard	P Package Pin	N Package Pin	Frequency(MHz)
System Clock	External	DIFF SSTL15	AD12	AD11	200.00

System Clock Termination Settings

Enable DIFF Term

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

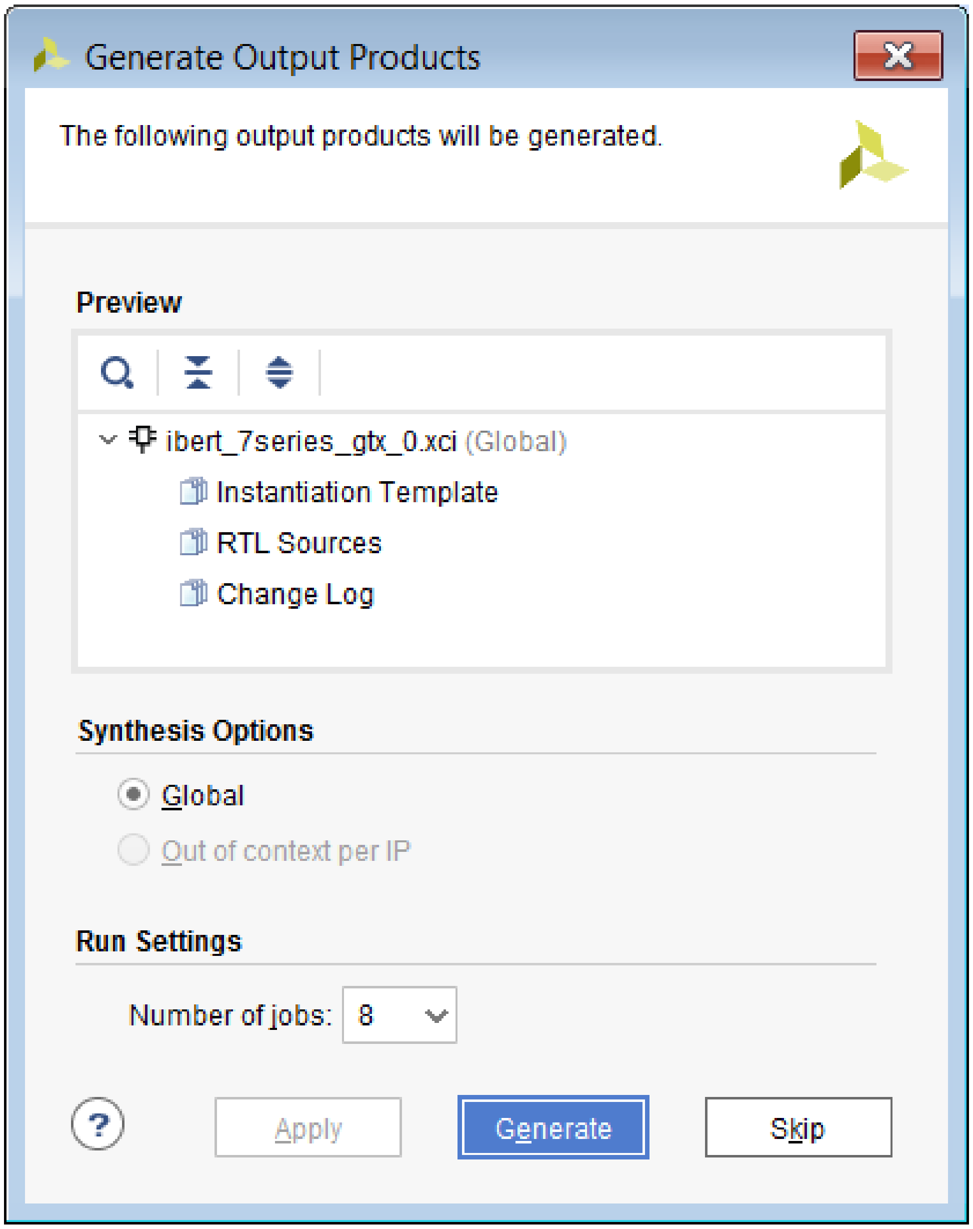


Component Name ibert_7series_gtx_0

IBERT Design Summary

Number of Protocols	1
System Clock Source	External (P Pin : AD12)
System Clock Source	External (N Pin : AD11)
QUAD Count	2
MMCM Count	1
RefClk Sources	1

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


Generate Output Products

The following output products will be generated.

Preview

ibert_7series_gtx_0.xci (Global)
- Instantiation Template
- RTL Sources
- Change Log

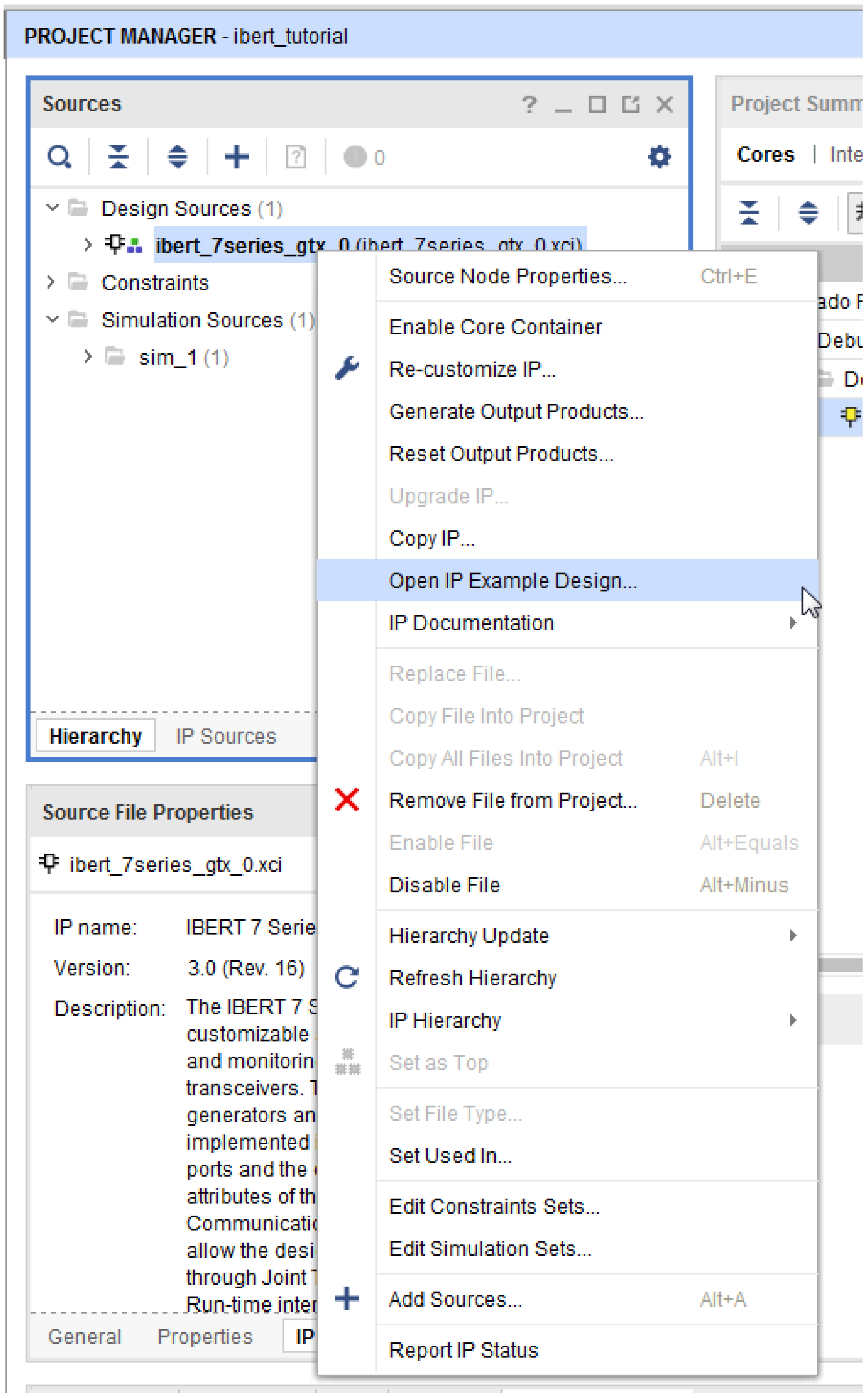
Synthesis Options

[✓ Global]
Out of context per IP

Run Settings

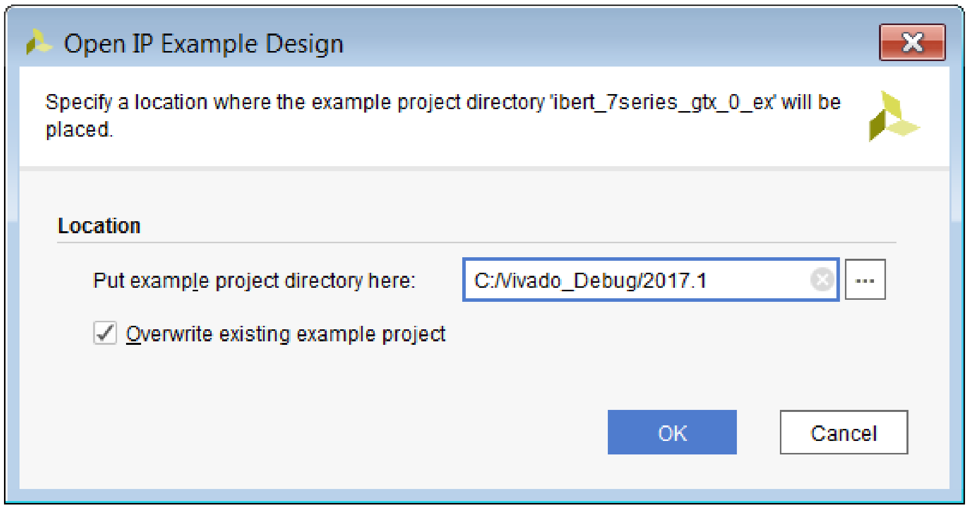
Number of jobs: 8

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



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

    Note: This opens a new instance of Vivado® IDE with the new example design opened.


Open IP Example Design

Specify a location where the example project directory 'ibert_7series_gtx_0_ex' will be placed.

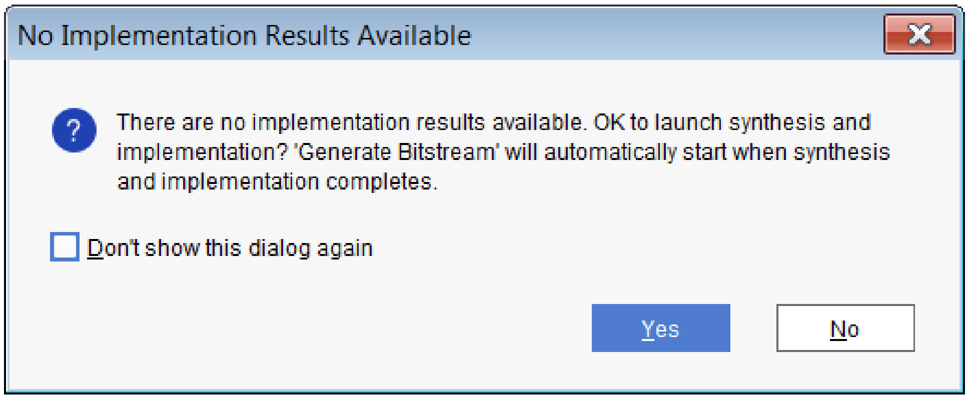
Location

Put example project directory here: C:/Vivado_Debug/2017.1

✓ Overwrite existing example project

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**.

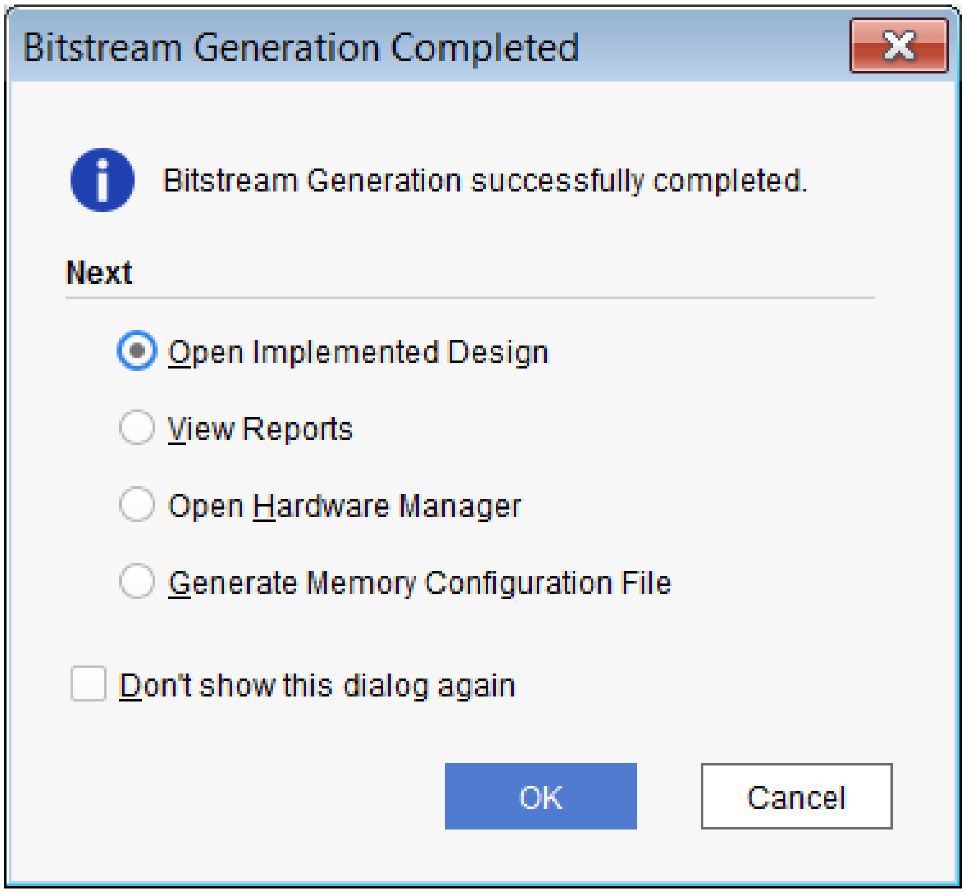

No Implementation Results Available

There are no implementation results available. OK to launch synthesis and implementation? 'Generate Bitstream' will automatically start when synthesis and implementation completes.

Don't show this dialog again

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

2. Select **Open Hardware Manager**, and click **OK**.


Bitstream Generation Completed

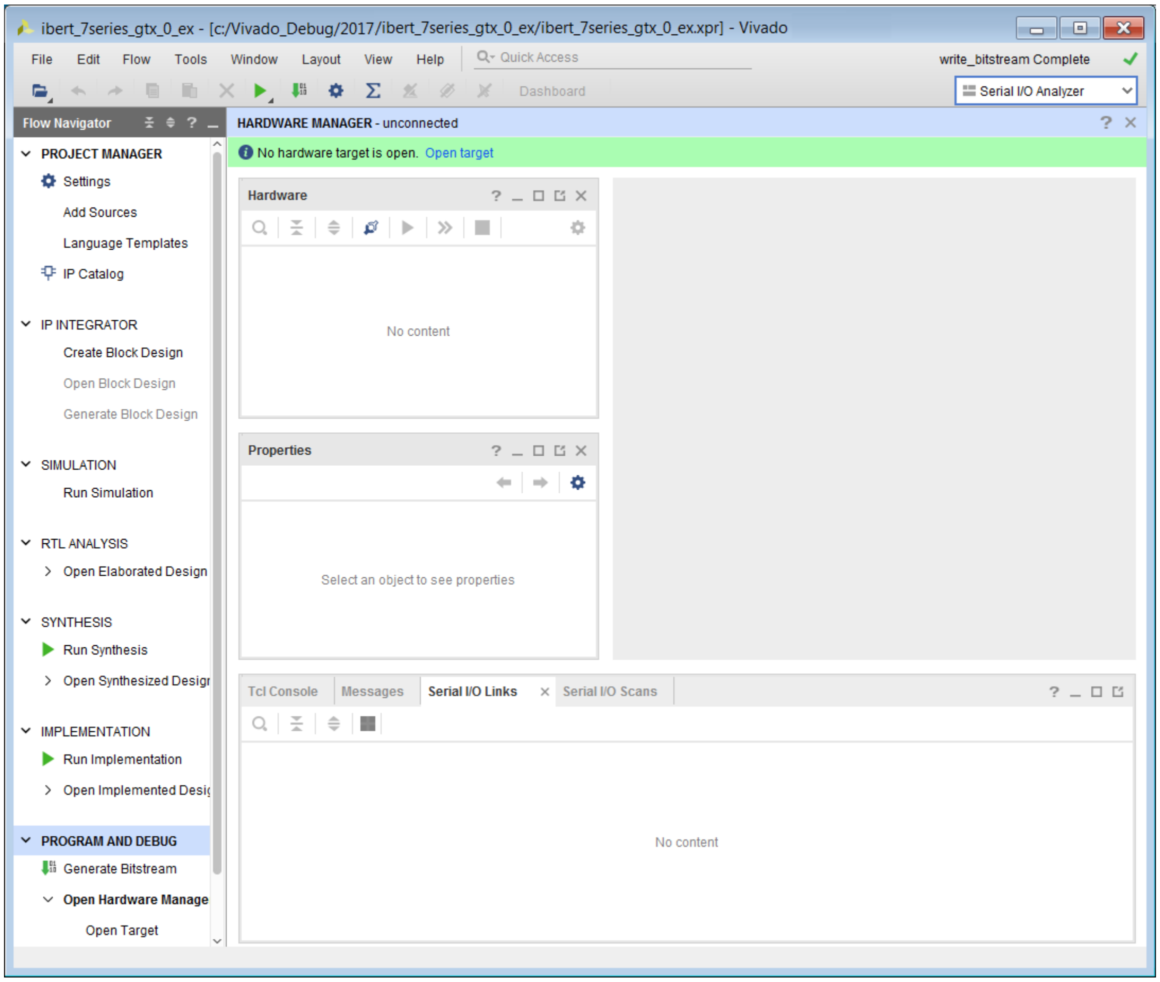
Bitstream Generation successfully completed.

Next

- [✓ Open Implemented Design]
- View Reports
- Open Hardware Manager
- Generate Memory Configuration File

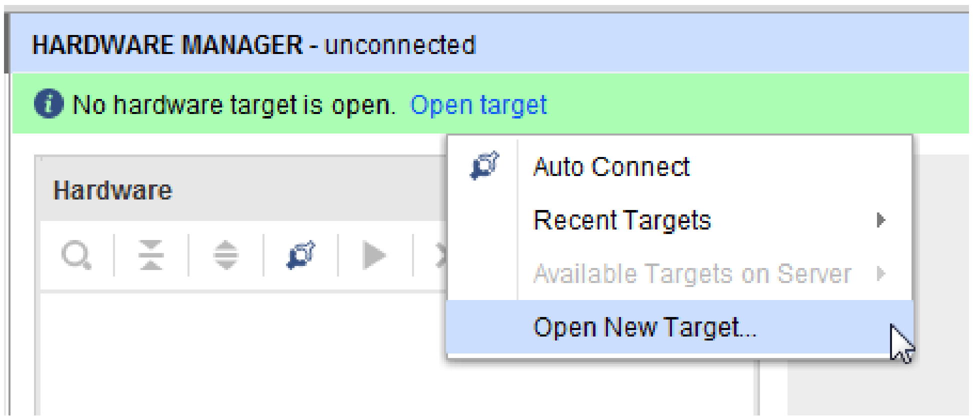
☐ Don't show this dialog again

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

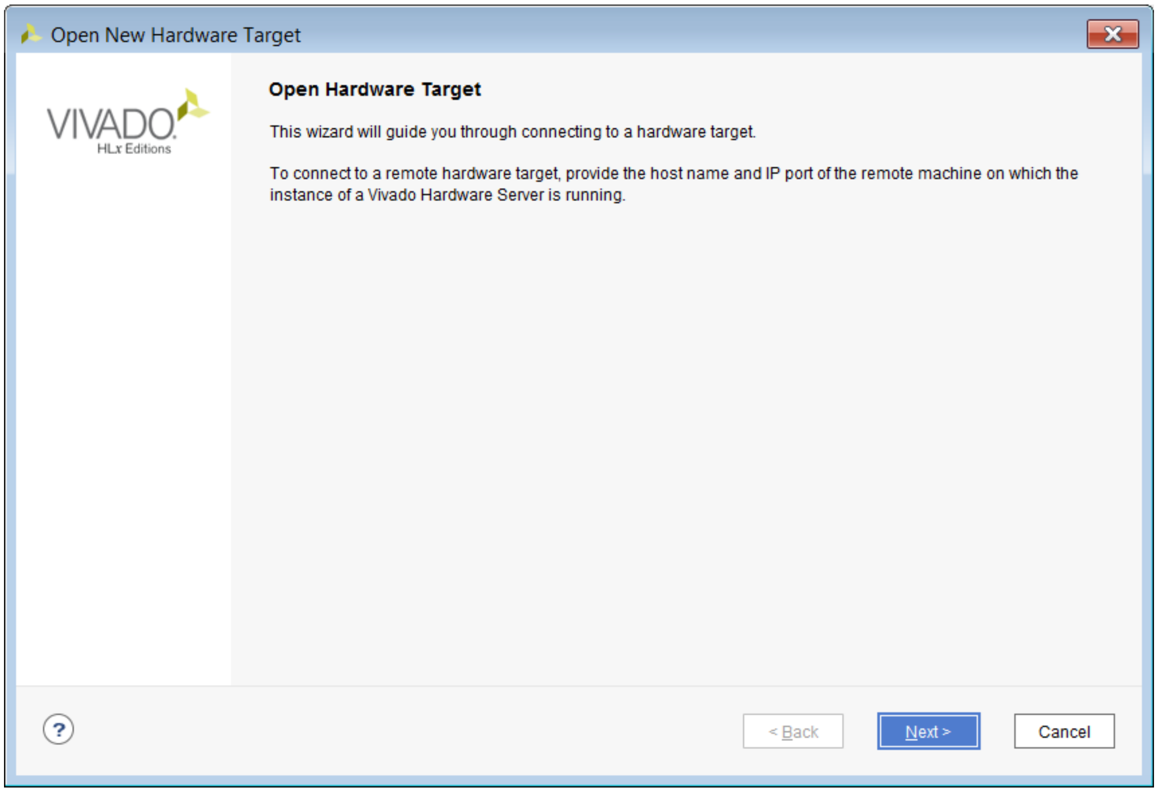


Step 4: Interact with the IBERT core using Serial I/O Analyzer

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


HARDWARE MANAGER - unconnected
No hardware target is open. Open target
Hardware
Auto Connect
Recent Targets
Available Targets on Server
Open New Target...

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

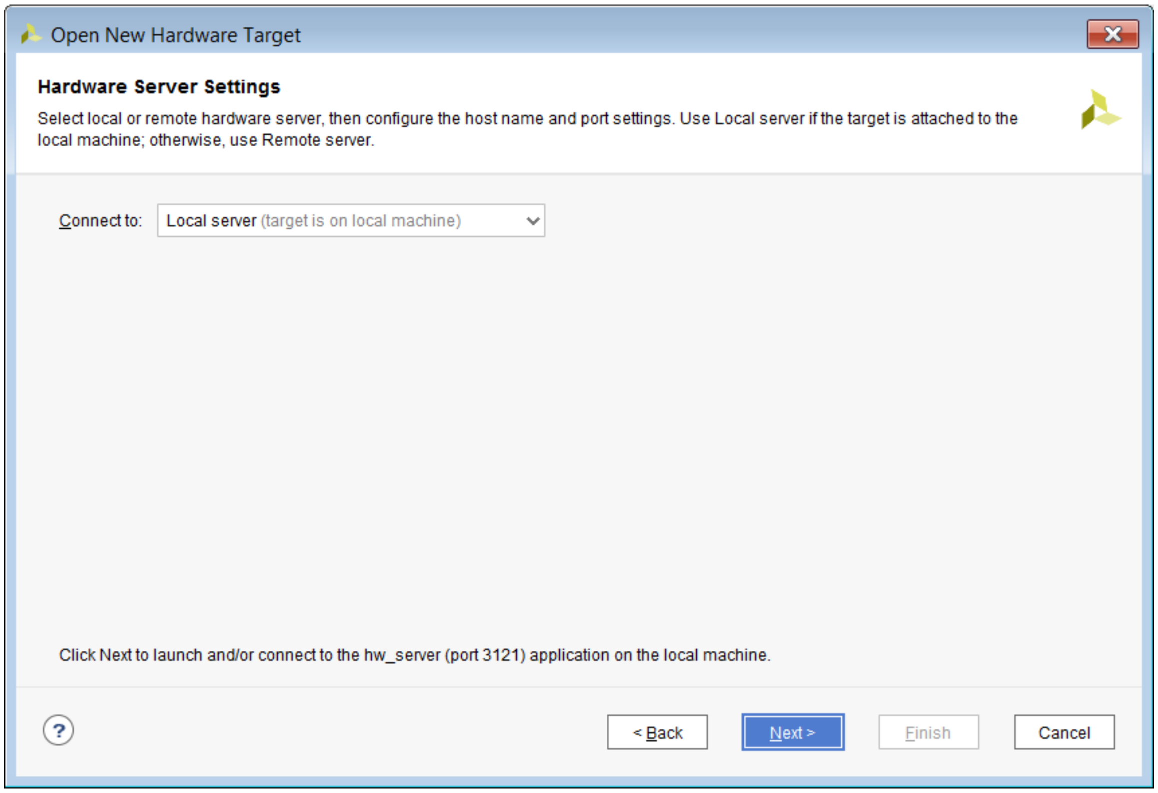


Open Hardware Target

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.

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



Hardware Server Settings

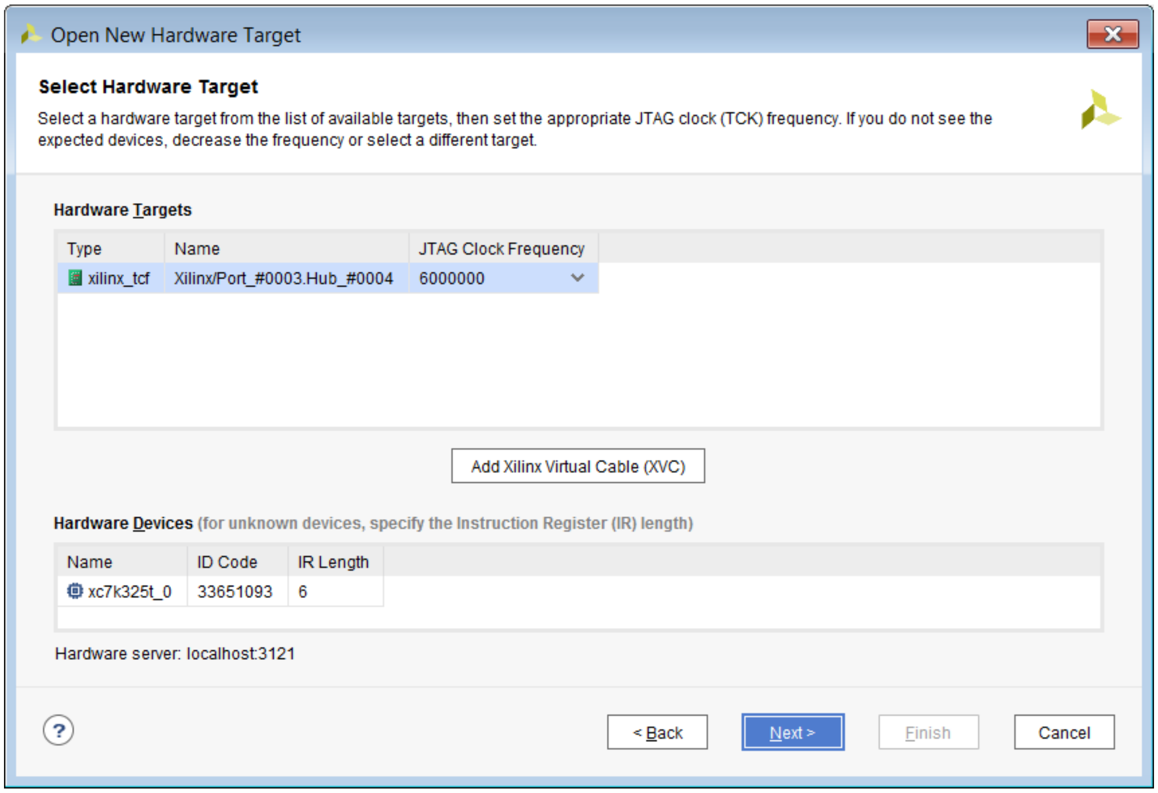
Select local or remote hardware server, then configure the host name and port settings. Use Local server if the target is attached to the local machine; otherwise, use Remote server.

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.

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 Hardware Target

Select Hardware Target

Select a hardware target from the list of available targets, then set the appropriate JTAG clock (TCK) frequency. If you do not see the expected devices, decrease the frequency or select a different target.

Hardware Targets

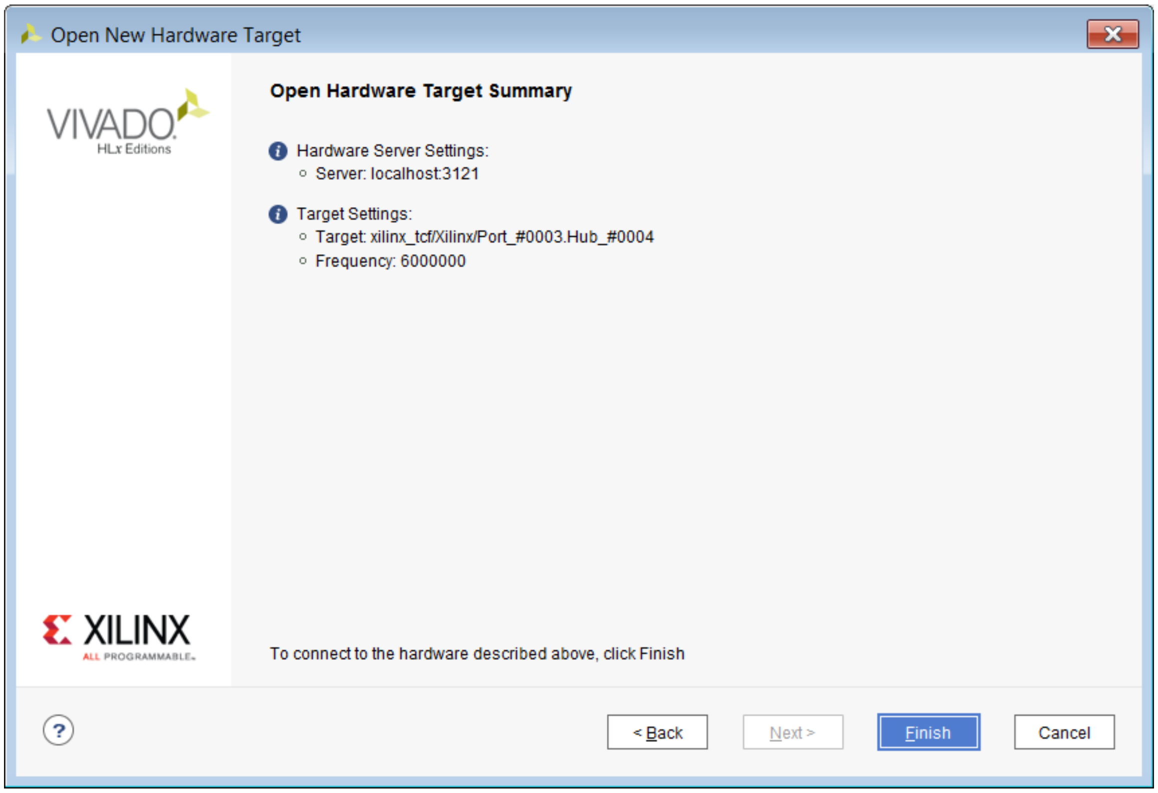
Type	Name	JTAG Clock Frequency
xilinx_tcf	Xilinx/Port_#0003.Hub_#0004	6000000

Hardware Devices (for unknown devices, specify the Instruction Register (IR) length)

Name	ID Code	IR Length
xc7k325t_0	33651093	6

Hardware server: localhost:3121

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


Open New Hardware Target

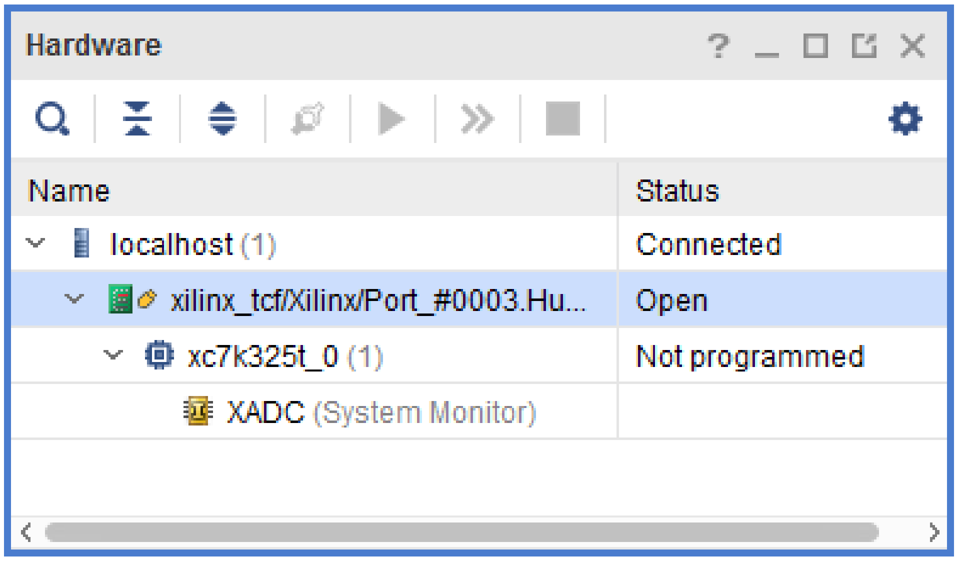
Open Hardware Target Summary

Hardware Server Settings:
- Server: localhost:3121

Target Settings:
- Target: xilinx_tcf/Xilinx/Port_#0003.Hub_#0004
- Frequency: 6000000

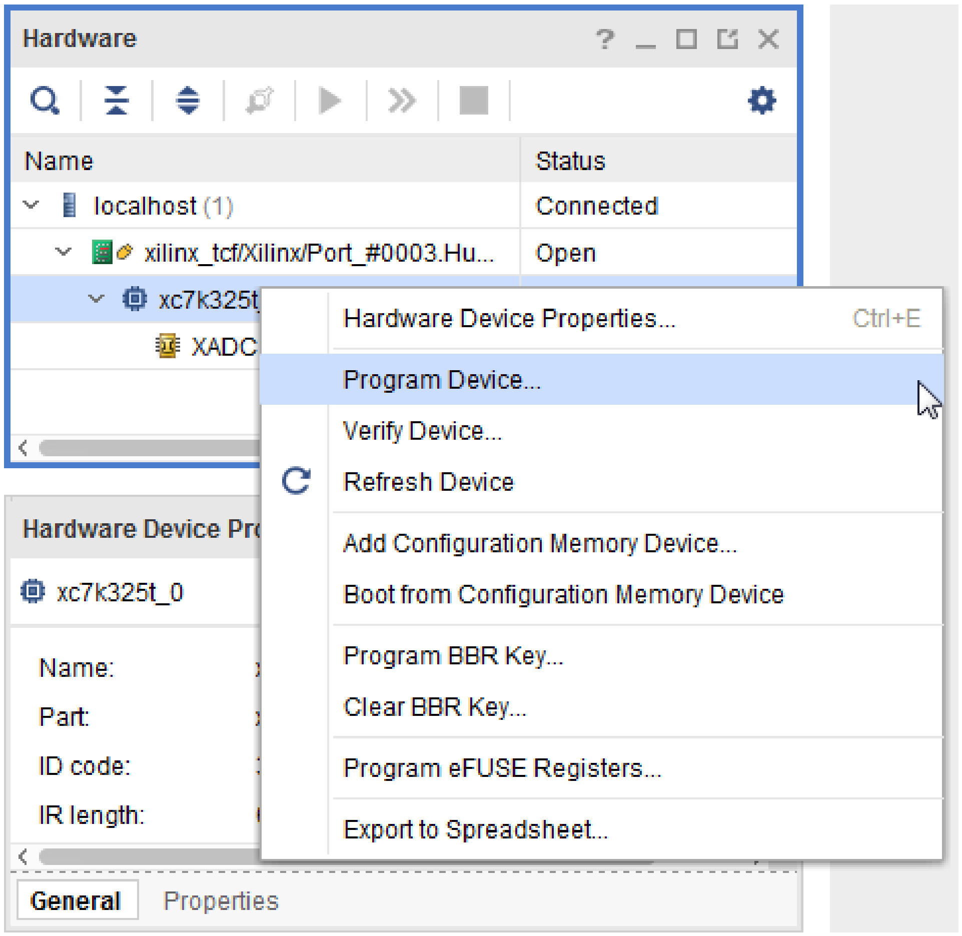
To connect to the hardware described above, click Finish

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

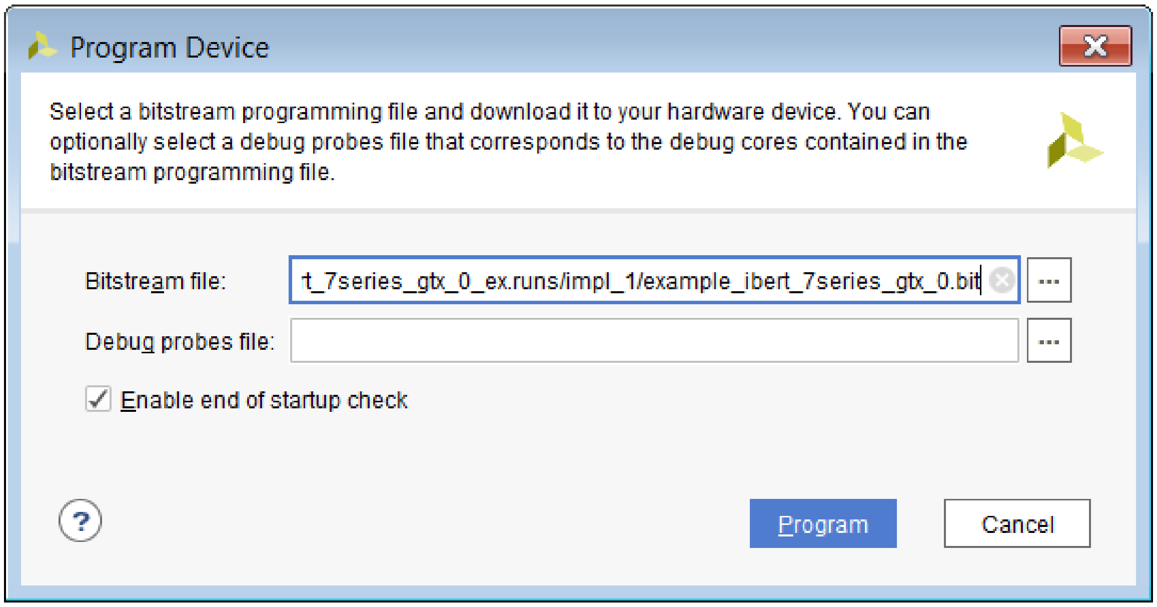


Name	Status
localhost (1)	Connected
xilinx_tcf/Xilinx/Port_#0003.Hu...	Open
xc7k325t_0 (1)	Not programmed
XADC (System Monitor)	

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

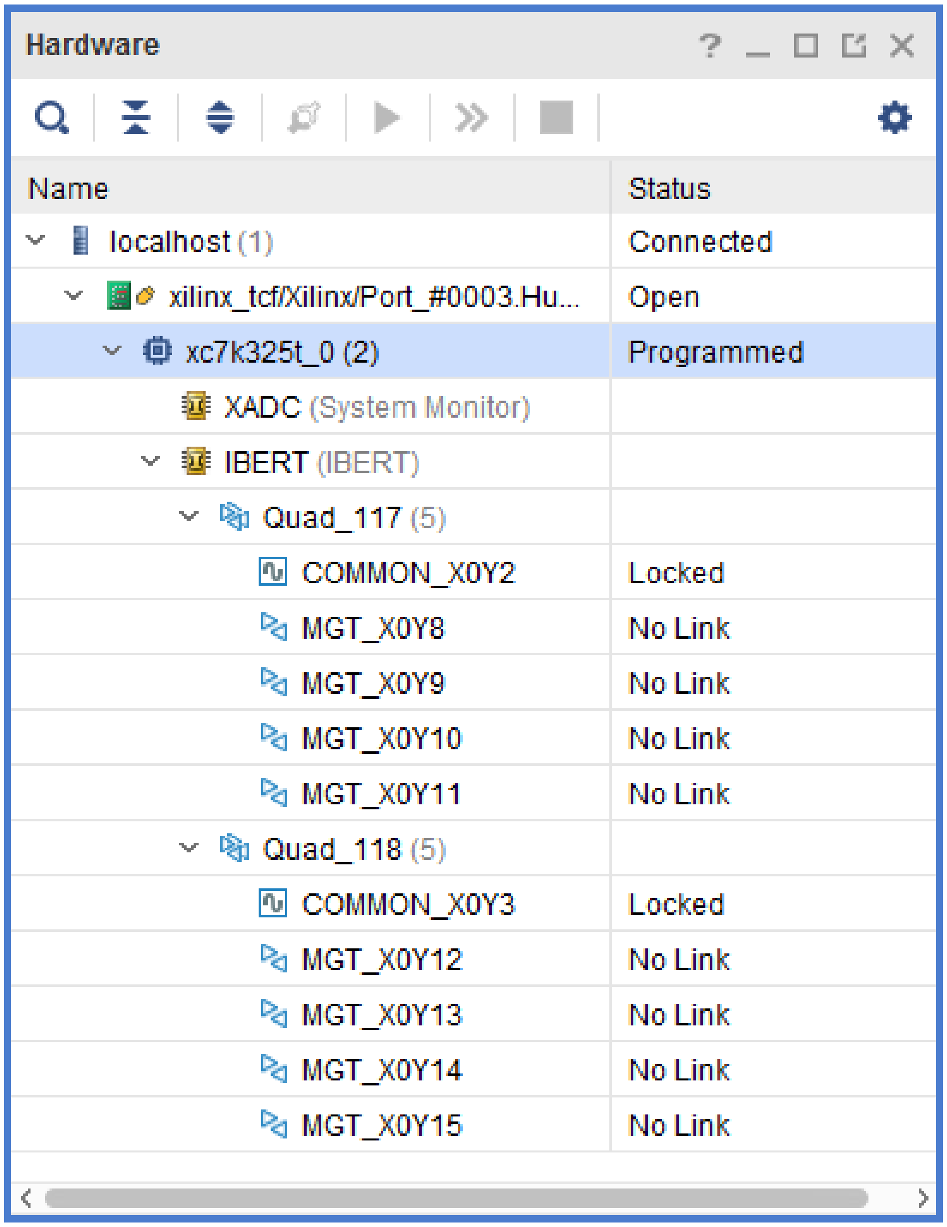
Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file: t_7series_gtx_0_ex.runs/impl_1/example_ibert_7series_gtx_0.bit

Debug probes file:

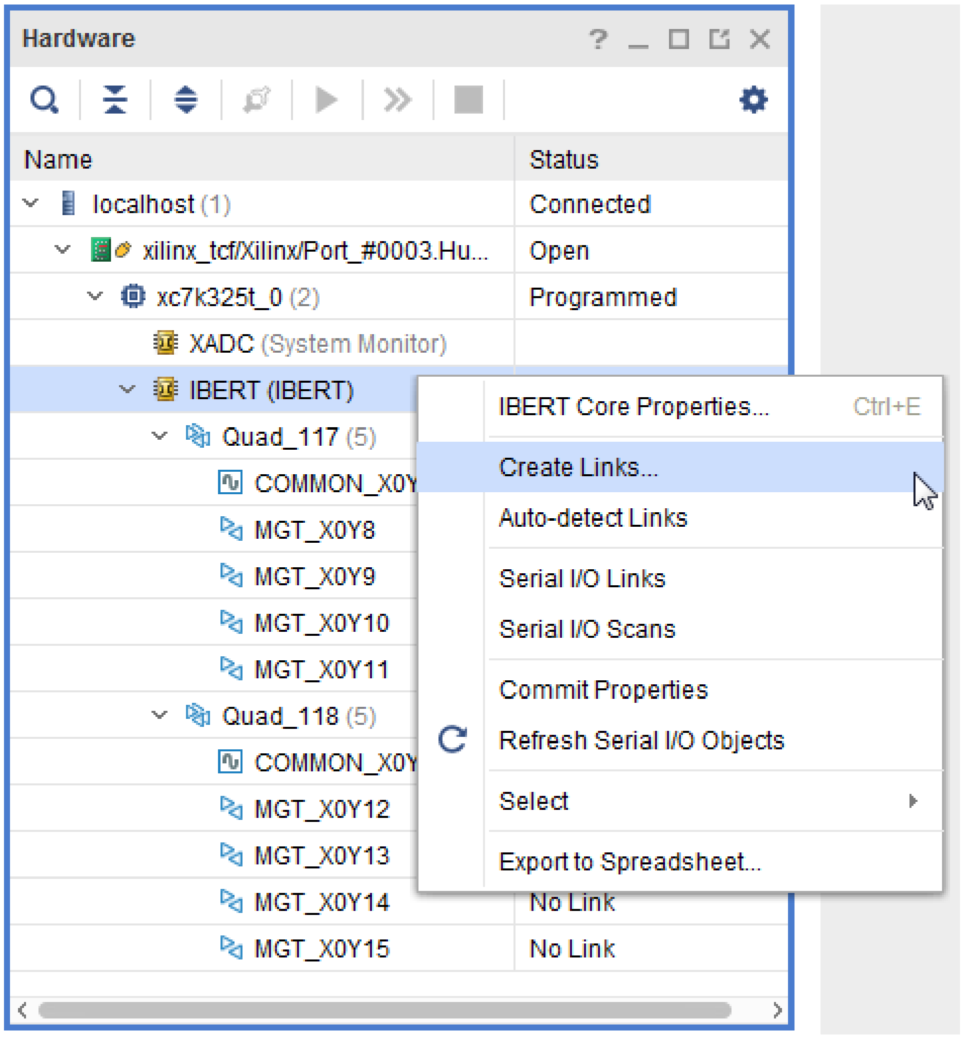
✓ Enable end of startup check

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**.



Name	Status
localhost (1)	Connected
xilinx_tcf/Xilinx/Port_#0003.Hu...	Open
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
MGT_X0Y12	No Link
MGT_X0Y13	No Link
MGT_X0Y14	No Link
MGT_X0Y15	No Link

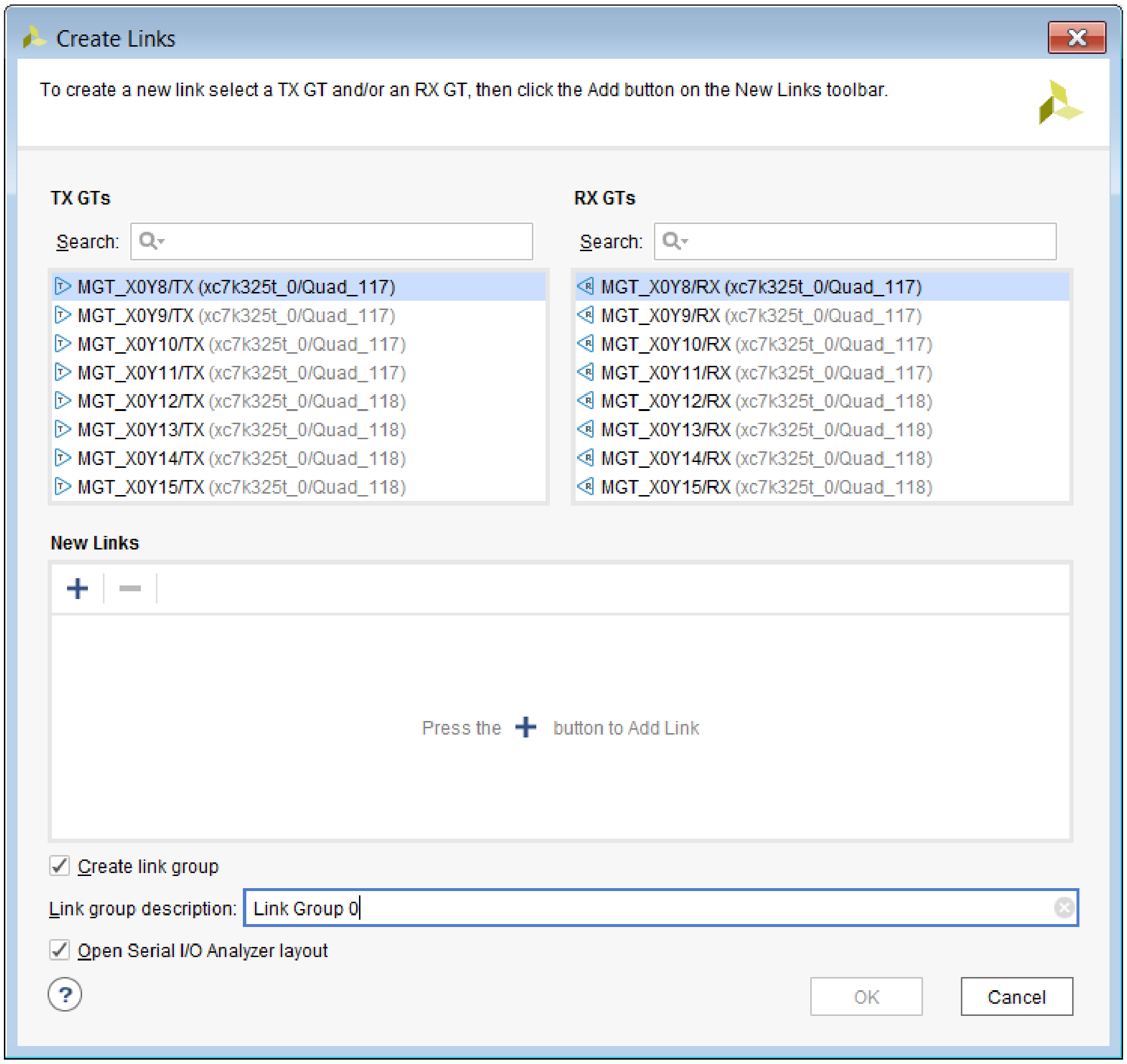
9. Next, create links for all eight transceivers. Vivado Serial I/O analyzer is a link-based analyzer, which allows users to link between any transmitter and receiver GTs within the IBERT design. For this tutorial, simply link the TX and RX of the same channel. To create a link, right-click the **IBERT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



Name	Status
localhost (1)	Connected
xilinx_tcf/Xilinx/Port_#0003.Hu...	Open
xc7k325t_0 (2)	Programmed
XADC (System Monitor)	
IBERT (IBERT)	
Quad_117 (5)	
COMMON_X0Y	
MGT_X0Y8	
MGT_X0Y9	
MGT_X0Y10	
MGT_X0Y11	
Quad_118 (5)	
COMMON_X0Y	
MGT_X0Y12	
MGT_X0Y13	
MGT_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

Search:

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)

RX GTs

Search:

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)

New Links

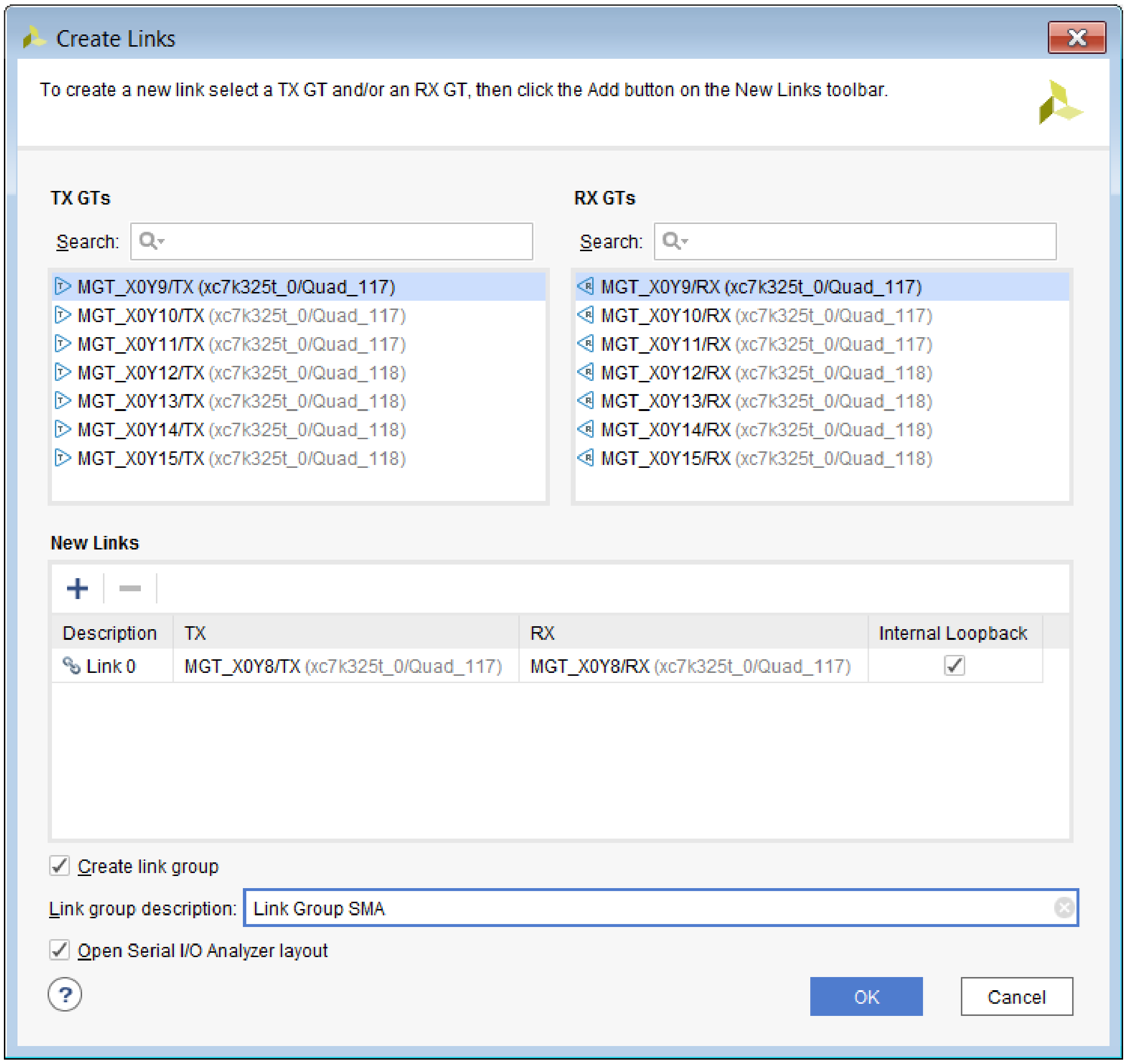
Press the + button to Add Link

✓ Create link group

Link group description: Link Group 0

✓ Open Serial I/O Analyzer layout

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 click the Add button on the New Links toolbar.
TX GTs
Search:
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)
RX GTs
Search:
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)
New Links
OK
Cancel

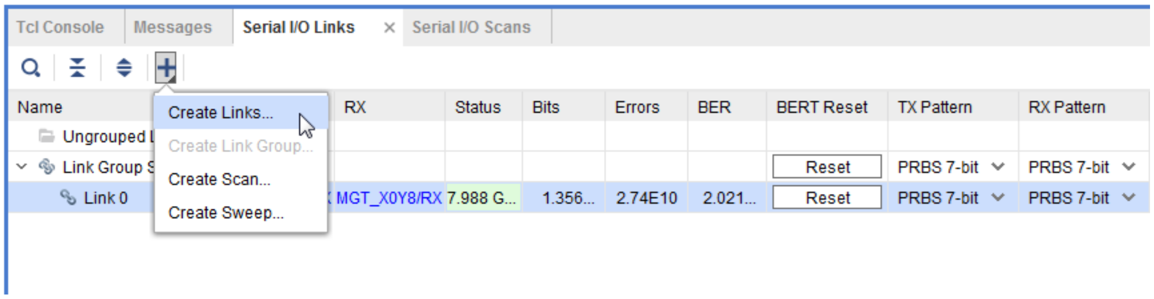
Description	TX	RX	Internal Loopback
Link 0	MGT_X0Y8/TX (xc7k325t_0/Quad_117)	MGT_X0Y8/RX (xc7k325t_0/Quad_117)	✓

☑ Create link group

Link group description: Link Group SMA

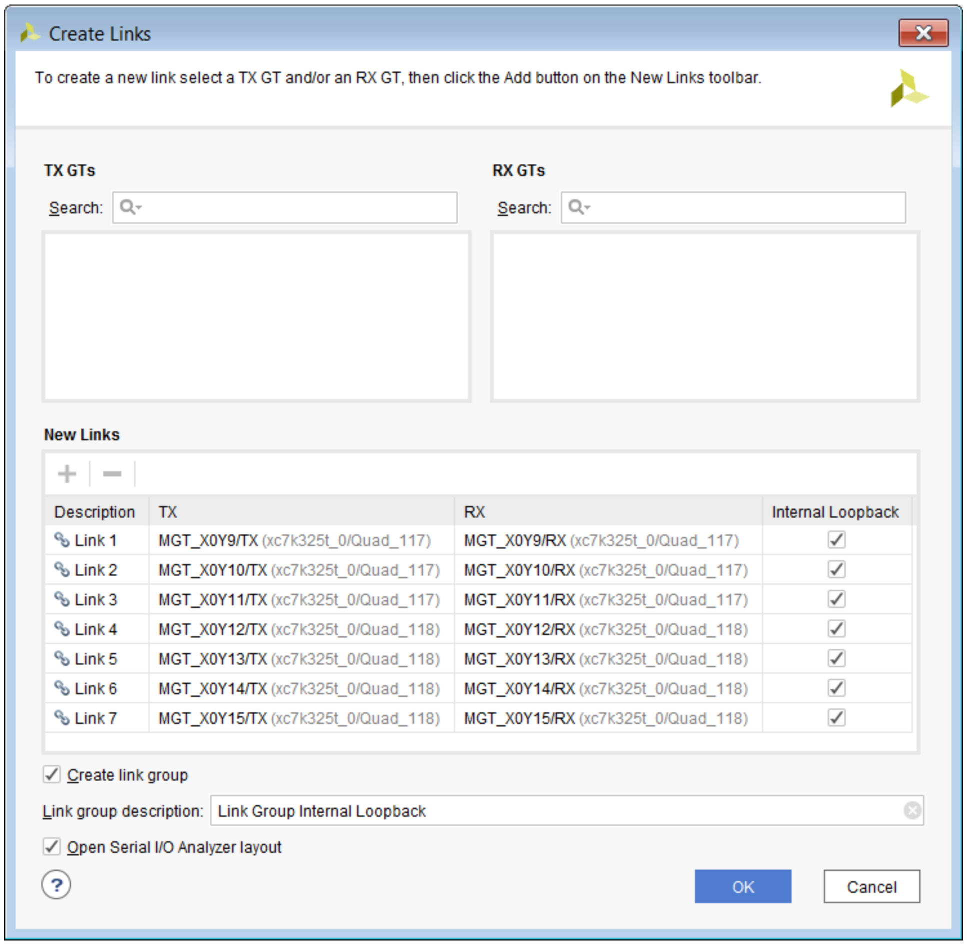
☑ Open Serial I/O Analyzer layout

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.



Name	RX	Status	Bits	Errors	BER	BERT Reset	TX Pattern	RX Pattern
Ungrouped L								
Link Group S						Reset	PRBS 7-bit	PRBS 7-bit
Link 0	MGT_X0Y8/RX	7.988 G...	1.356...	2.74E10	2.021...	Reset	PRBS 7-bit	PRBS 7-bit

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**.



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 — Search:

RX GTs — Search:

New Links

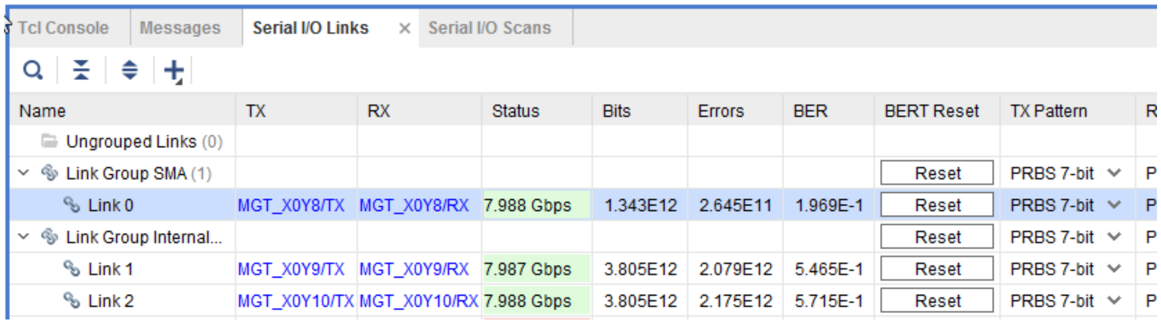
Description	TX	RX	Internal Loopback
Link 1	MGT_X0Y9/TX (xc7k325t_0/Quad_117)	MGT_X0Y9/RX (xc7k325t_0/Quad_117)	✓
Link 2	MGT_X0Y10/TX (xc7k325t_0/Quad_117)	MGT_X0Y10/RX (xc7k325t_0/Quad_117)	✓
Link 3	MGT_X0Y11/TX (xc7k325t_0/Quad_117)	MGT_X0Y11/RX (xc7k325t_0/Quad_117)	✓
Link 4	MGT_X0Y12/TX (xc7k325t_0/Quad_118)	MGT_X0Y12/RX (xc7k325t_0/Quad_118)	✓
Link 5	MGT_X0Y13/TX (xc7k325t_0/Quad_118)	MGT_X0Y13/RX (xc7k325t_0/Quad_118)	✓
Link 6	MGT_X0Y14/TX (xc7k325t_0/Quad_118)	MGT_X0Y14/RX (xc7k325t_0/Quad_118)	✓
Link 7	MGT_X0Y15/TX (xc7k325t_0/Quad_118)	MGT_X0Y15/RX (xc7k325t_0/Quad_118)	✓

✓ Create link group

Link group description: Link Group Internal Loopback

✓ Open Serial I/O Analyzer layout

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

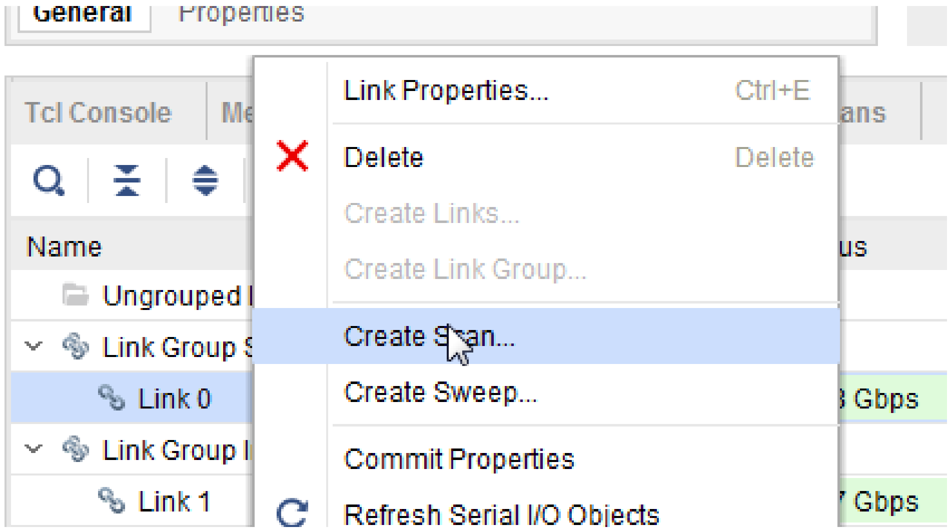


Name	TX	RX	Status	Bits	Errors	BER	BERT Reset	TX Pattern	R
Ungrouped Links (0)									
Link Group SMA (1)							Reset	PRBS 7-bit	P
Link 0	MGT_X0Y8/TX	MGT_X0Y8/RX	7.988 Gbps	1.343E12	2.645E11	1.969E-1	Reset	PRBS 7-bit	P
Link Group Internal...							Reset	PRBS 7-bit	P
Link 1	MGT_X0Y9/TX	MGT_X0Y9/RX	7.987 Gbps	3.805E12	2.079E12	5.465E-1	Reset	PRBS 7-bit	P
Link 2	MGT_X0Y10/TX	MGT_X0Y10/RX	7.988 Gbps	3.805E12	2.175E12	5.715E-1	Reset	PRBS 7-bit	P

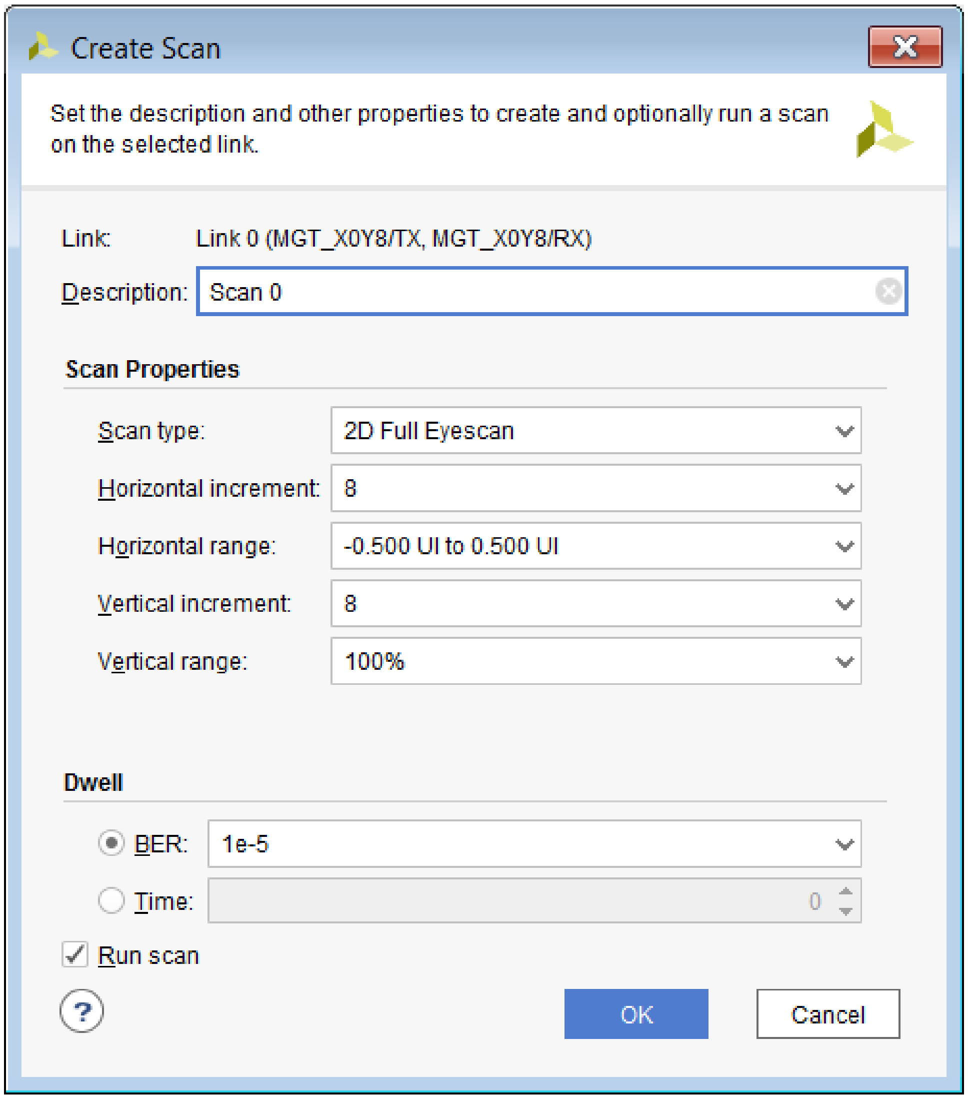
The status of the links indicate an 8.0 Gbps line rate.

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.



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 description and other properties to create and optionally run a scan on the selected link.

Link: Link 0 (MGT_X0Y8/TX, MGT_X0Y8/RX)

Description: Scan 0

Scan Properties

Scan type:	2D Full Eyescan
Horizontal increment:	8
Horizontal range:	-0.500 UI to 0.500 UI
Vertical increment:	8
Vertical range:	100%

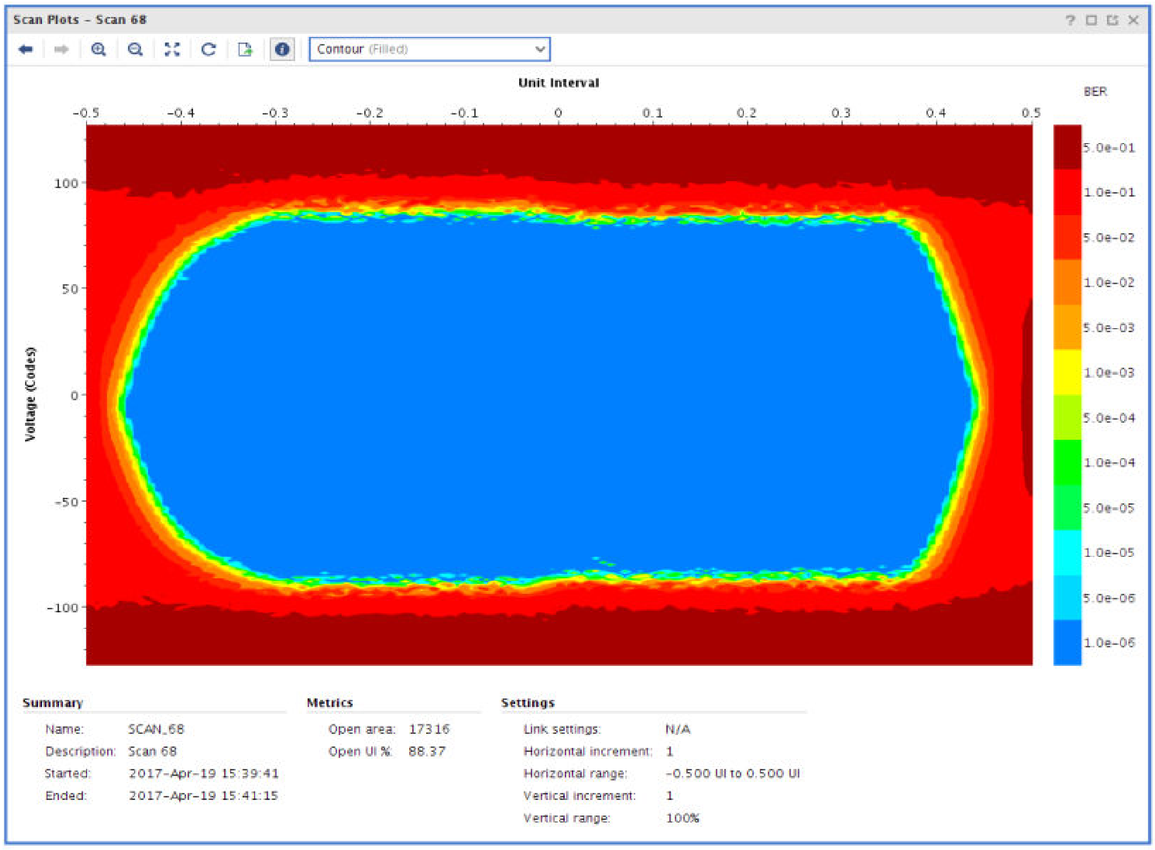
Dwell

● BER: 1e-5

○ Time: 0

✓ Run scan

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



Summary

Name:	SCAN_68
Description:	Scan 68
Started:	2017-Apr-19 15:39:41
Ended:	2017-Apr-19 15:41:15

Metrics

Open area:	17316
Open UI %:	88.37

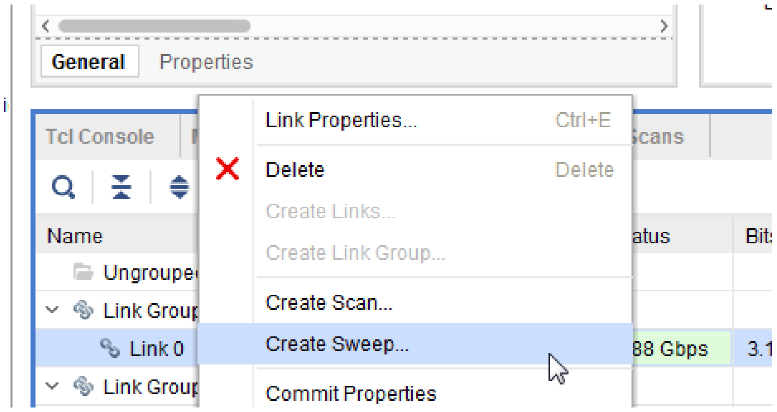
Settings

Link settings:	N/A
Horizontal increment:	1
Horizontal range:	-0.500 UI to 0.500 UI
Vertical increment:	1
Vertical range:	100%

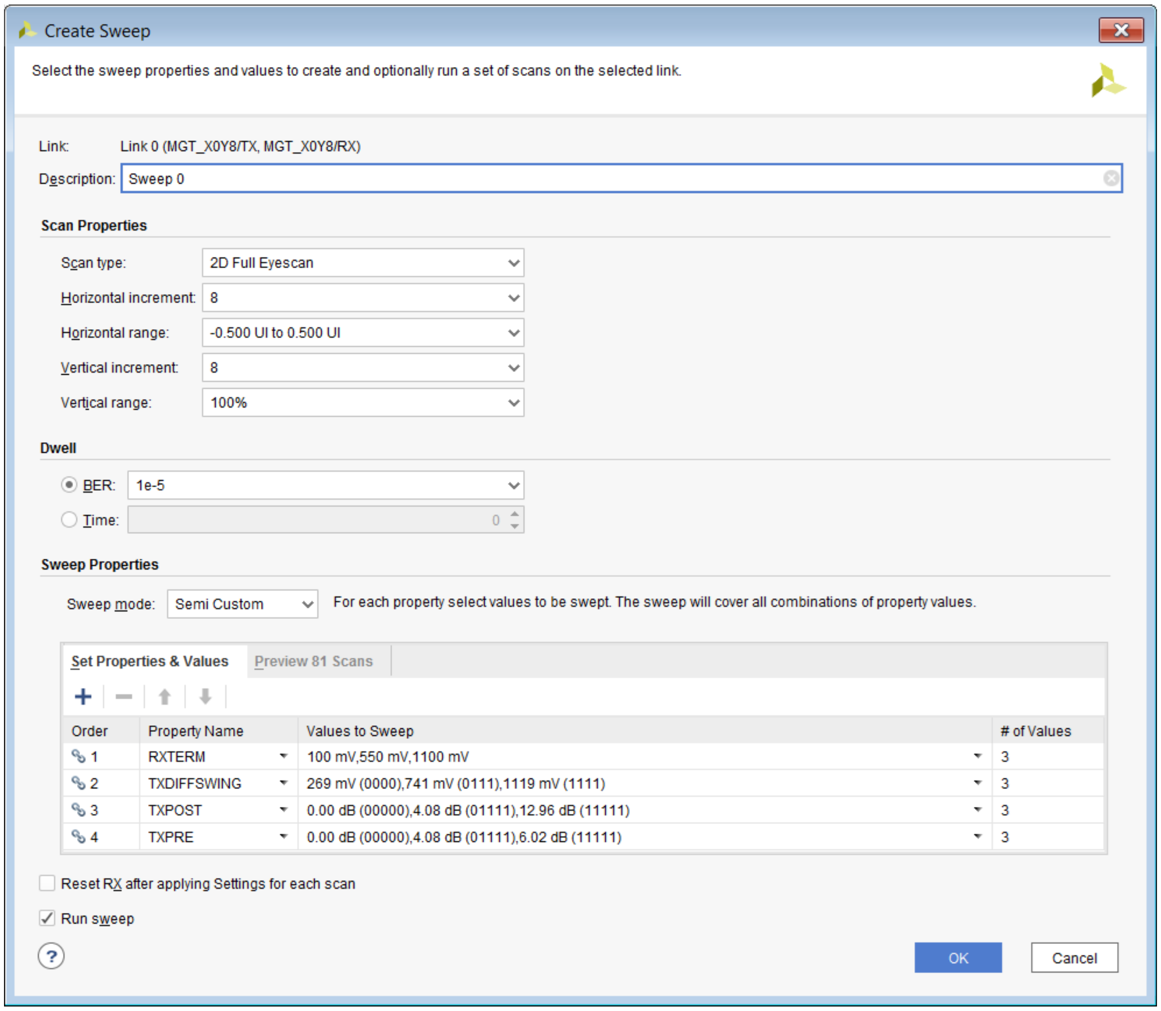
The 2D Scan Plot is a heat map of the BER value.

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 Sweep

Select the sweep properties and values to create and optionally run a set of scans on the selected link.

Link: Link 0 (MGT_X0Y8/TX, MGT_X0Y8/RX)

Description: Sweep 0

Scan Properties

Scan type:	2D Full Eyescan
Horizontal increment:	8
Horizontal range:	-0.500 UI to 0.500 UI
Vertical increment:	8
Vertical range:	100%

Dwell

[✓ BER: 1e-5]

Time: 0

Sweep Properties

Sweep mode: Semi Custom — For each property select values to be swept. The sweep will cover all combinations of property values.

Set Properties & Values | Preview 81 Scans

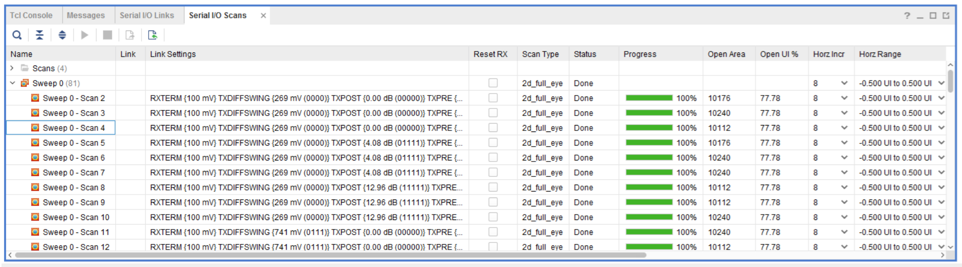
Order	Property Name	Values to Sweep	# of Values
1	RXTERM	100 mV,550 mV,1100 mV	3
2	TXDIFFSWING	269 mV (0000),741 mV (0111),1119 mV (1111)	3
3	TXPOST	0.00 dB (00000),4.08 dB (01111),12.96 dB (11111)	3
4	TXPRE	0.00 dB (00000),4.08 dB (01111),6.02 dB (11111)	3

☐ Reset RX after applying Settings for each scan

☑ Run sweep

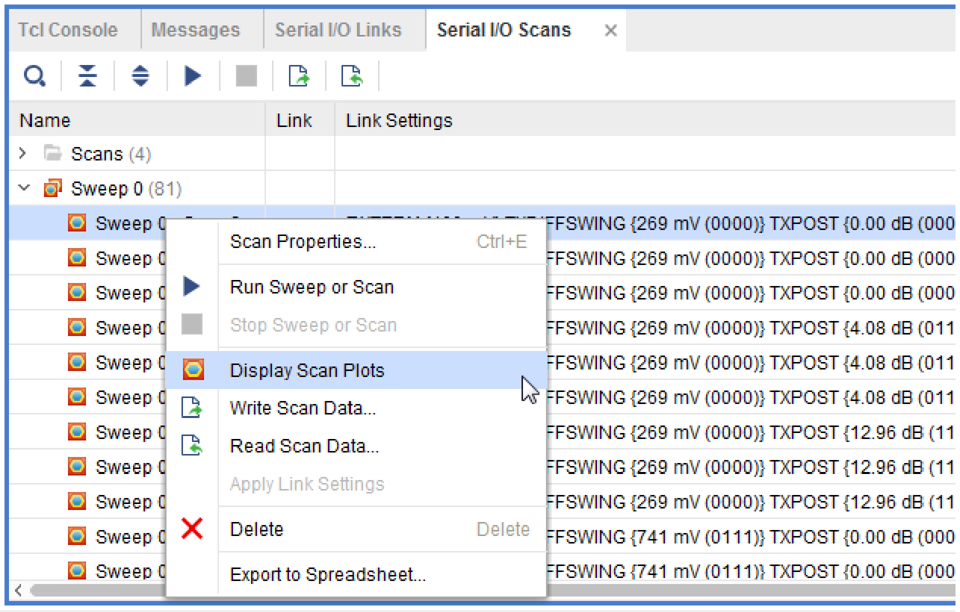
Because here are four different Sweep Properties and each of these properties has three different values (as seen in the Values to Sweep column), a total number of 81 sweep tests are carried out. The Scans window shows the results of all the scans that have been done for the selected link.

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

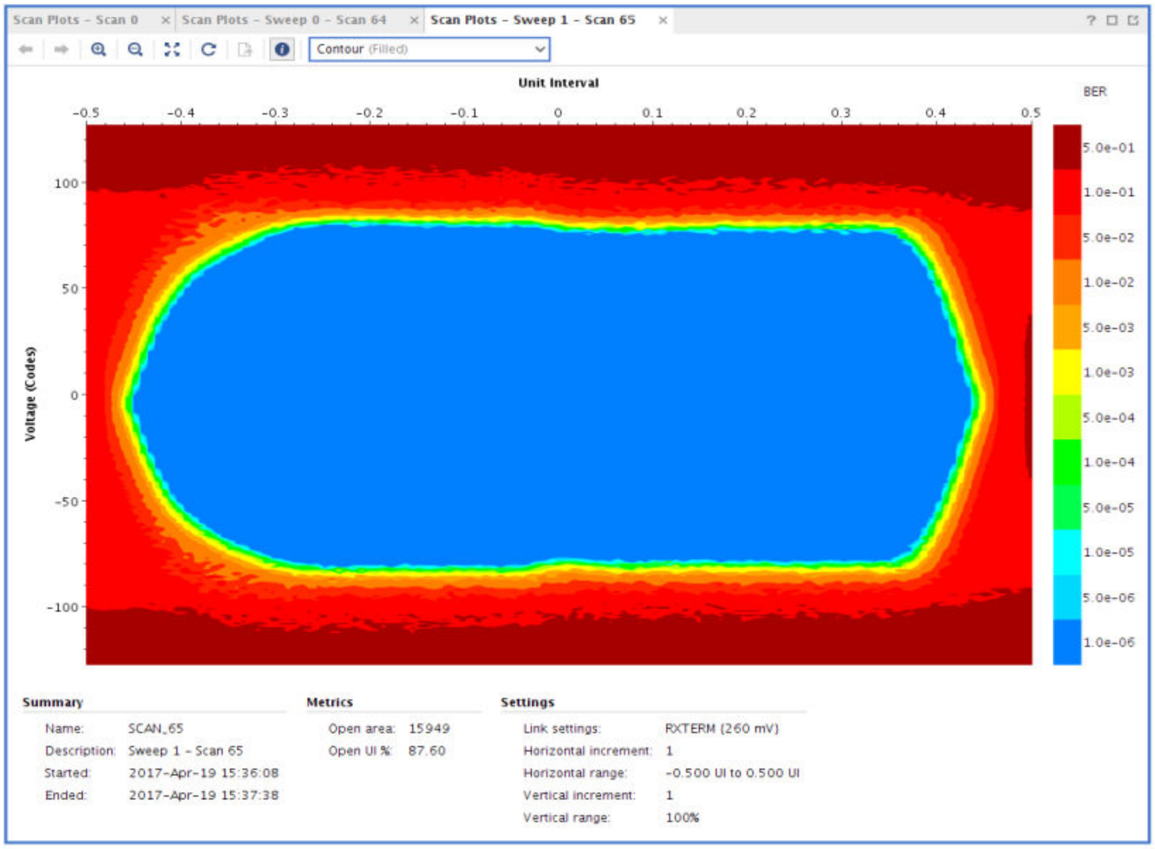


Name	Link	Link Settings	Reset RX	Scan Type	Status	Progress	Open Area	Open UI %	Horz Incr	Horz Range
Scans (4)										
Sweep 0 (81)			☐	2d_full_eye	Done				8	-0.500 UI to 0.500 UI
Sweep 0 - Scan 2		RXTERM {100 mV} TXDIFFSWING {269 mV (0000)} TXPOST {0.00 dB (00000)} TXPRE {...	☐	2d_full_eye	Done	100%	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	-0.500 UI to 0.500 UI
Sweep 0 - Scan 4		RXTERM {100 mV} TXDIFFSWING {269 mV (0000)} TXPOST {0.00 dB (00000)} TXPRE {...	☐	2d_full_eye	Done	100%	10112	77.78	8	-0.500 UI to 0.500 UI
Sweep 0 - Scan 5		RXTERM {100 mV} TXDIFFSWING {269 mV (0000)} TXPOST {4.08 dB (01111)} TXPRE {...	☐	2d_full_eye	Done	100%	10176	77.78	8	-0.500 UI to 0.500 UI
Sweep 0 - Scan 6		RXTERM {100 mV} TXDIFFSWING {269 mV (0000)} TXPOST {4.08 dB (01111)} TXPRE {...	☐	2d_full_eye	Done	100%	10240	77.78	8	-0.500 UI to 0.500 UI
Sweep 0 - Scan 7		RXTERM {100 mV} TXDIFFSWING {269 mV (0000)} TXPOST {4.08 dB (01111)} TXPRE {...	☐	2d_full_eye	Done	100%	10240	77.78	8	-0.500 UI to 0.500 UI
Sweep 0 - Scan 8		RXTERM {100 mV} TXDIFFSWING {269 mV (0000)} TXPOST {12.96 dB (11111)} TXPRE...	☐	2d_full_eye	Done	100%	10112	77.78	8	-0.500 UI to 0.500 UI
Sweep 0 - Scan 9		RXTERM {100 mV} TXDIFFSWING {269 mV (0000)} TXPOST {12.96 dB (11111)} TXPRE...	☐	2d_full_eye	Done	100%	10112	77.78	8	-0.500 UI to 0.500 UI
Sweep 0 - Scan 10		RXTERM {100 mV} TXDIFFSWING {269 mV (0000)} TXPOST {12.96 dB (11111)} TXPRE...	☐	2d_full_eye	Done	100%	10240	77.78	8	-0.500 UI to 0.500 UI
Sweep 0 - Scan 11		RXTERM {100 mV} TXDIFFSWING {741 mV (0111)} TXPOST {0.00 dB (00000)} TXPRE {...	☐	2d_full_eye	Done	100%	10240	77.78	8	-0.500 UI to 0.500 UI
Sweep 0 - Scan 12		RXTERM {100 mV} 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, right-click, and select **Display Scan Plots**.



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



Summary

Name:	SCAN_65
Description:	Sweep 1 - Scan 65
Started:	2017-Apr-19 15:36:08
Ended:	2017-Apr-19 15:37:38

Metrics

Open area:	15949
Open UI %:	87.60

Settings

Link settings:	RXTERM {260 mV}
Horizontal increment:	1
Horizontal range:	-0.500 UI to 0.500 UI
Vertical increment:	1
Vertical range:	100%

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:

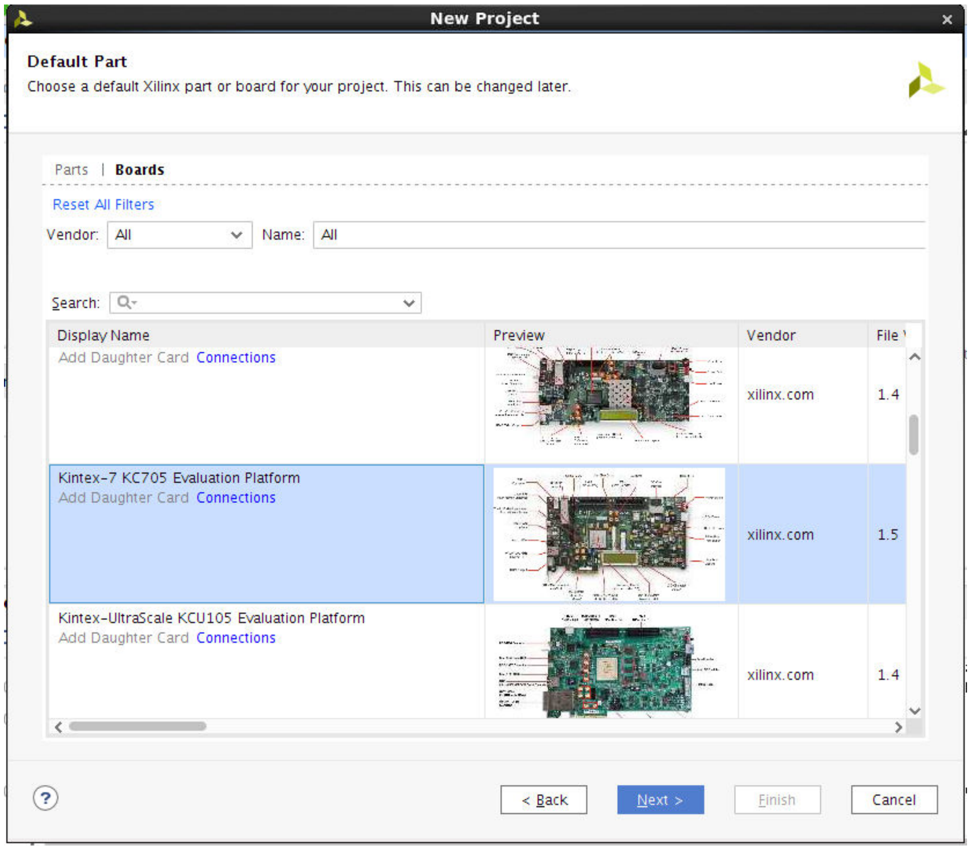
1. Creating a simple design in IP integrator that includes a System ILA and JTAG-to-AXI master.
2. Programming the Kintex®-7 FPGA KC705 Evaluation Kit Base Board and interacting with the JTAG to AXI Master IP core.

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**.



New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Parts | **Boards**

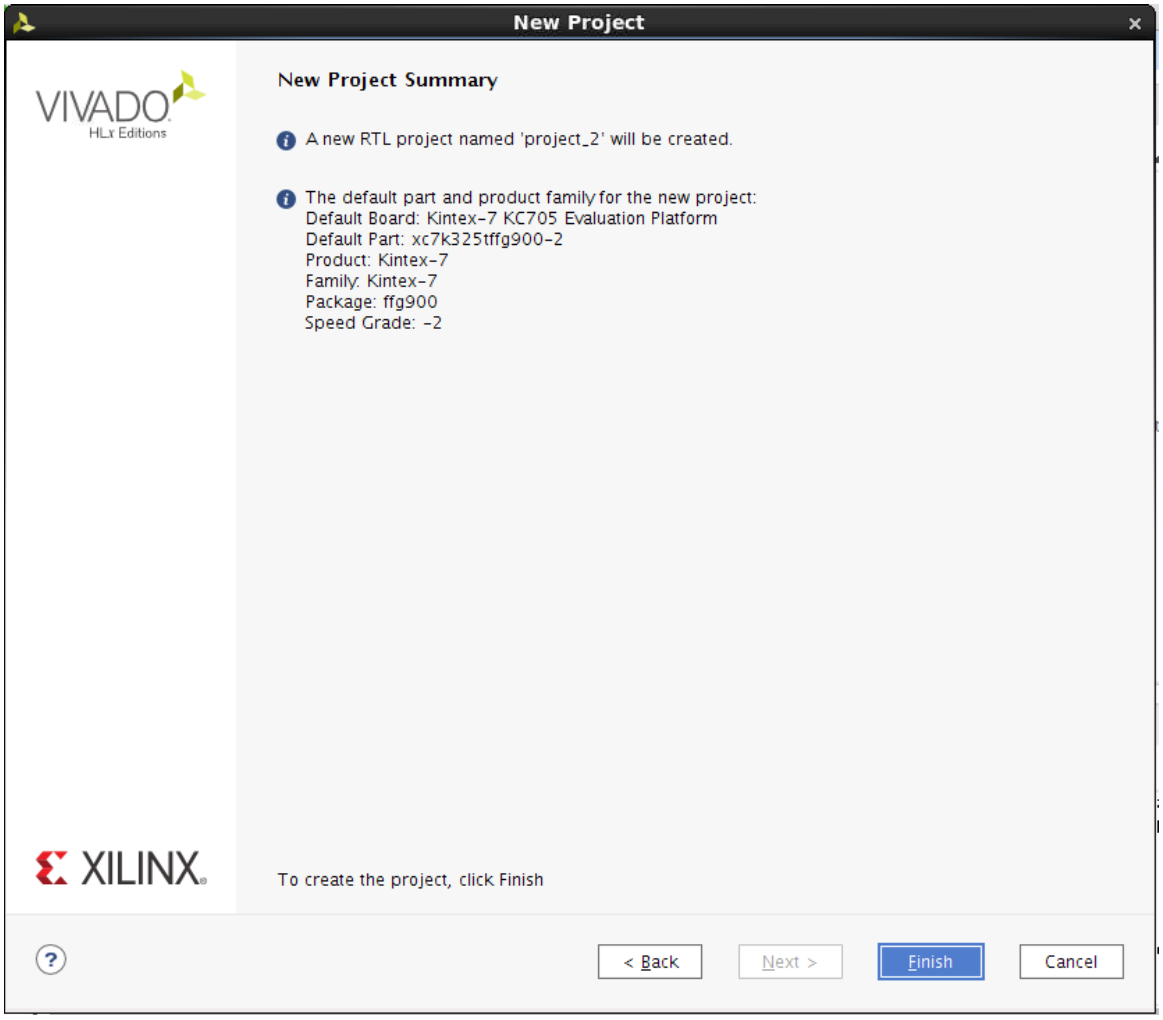
Reset All Filters

Vendor: All Name: All

Search:

Display Name	Preview	Vendor	File Version
Add Daughter Card Connections		xilinx.com	1.4
Kintex-7 KC705 Evaluation Platform Add Daughter Card Connections		xilinx.com	1.5
Kintex-UltraScale KCU105 Evaluation Platform Add Daughter Card Connections		xilinx.com	1.4

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



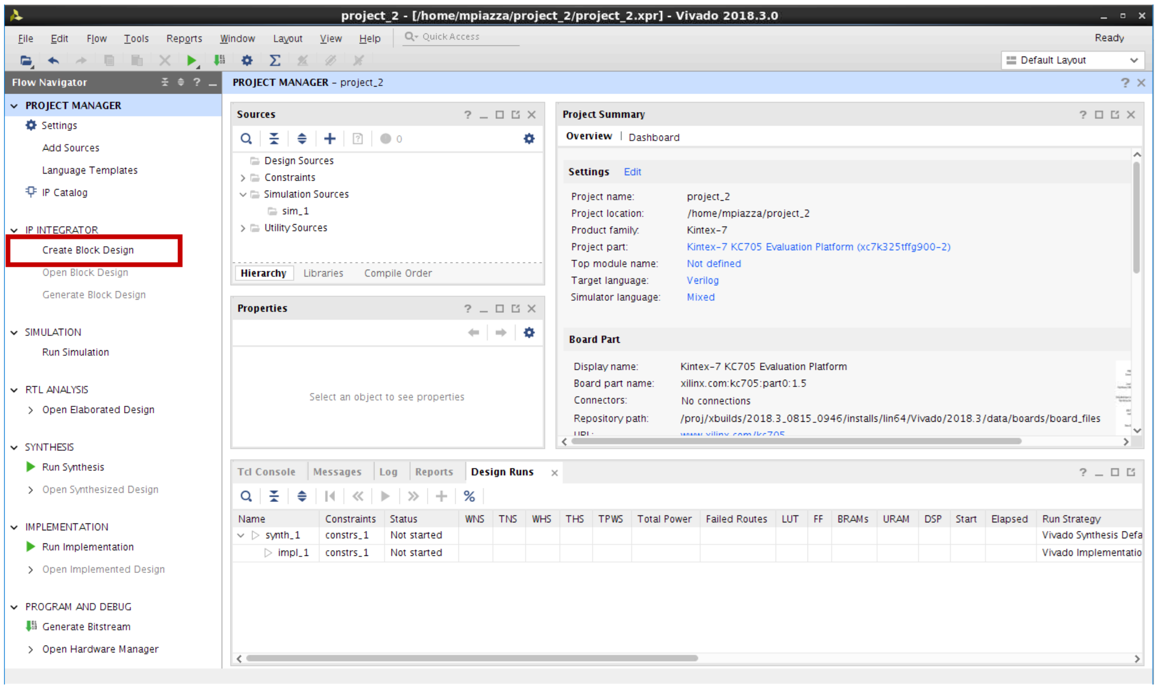
New Project Summary

A new RTL project named 'project_2' will be created.

The default part and product family for the new project:
Default Board: Kintex-7 KC705 Evaluation Platform
Default Part: xc7k325tffg900-2
Product: Kintex-7
Family: Kintex-7
Package: ffg900
Speed Grade: -2

To create the project, click Finish

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



Settings Edit

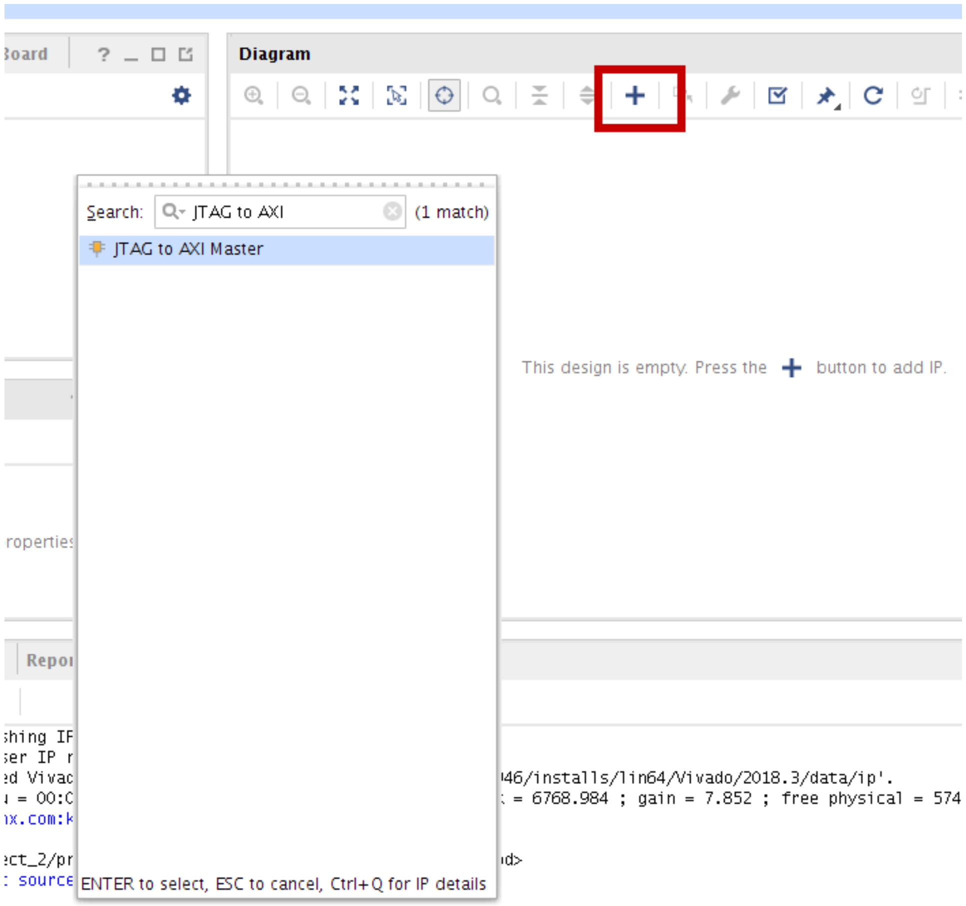
Project name:	project_2
Project location:	/home/mpiazza/project_2
Product family:	Kintex-7
Project part:	Kintex-7 KC705 Evaluation Platform (xc7k325tffg900-2)
Top module name:	Not defined
Target language:	Verilog
Simulator language:	Mixed

Board Part

Display name:	Kintex-7 KC705 Evaluation Platform
Board part name:	xilinx.com:kc705:part0:1.5
Connectors:	No connections
Repository path:	/proj/xbuilds/2018.3_0815_0946/installs/lin64/Vivado/2018.3/data/boards/board_files

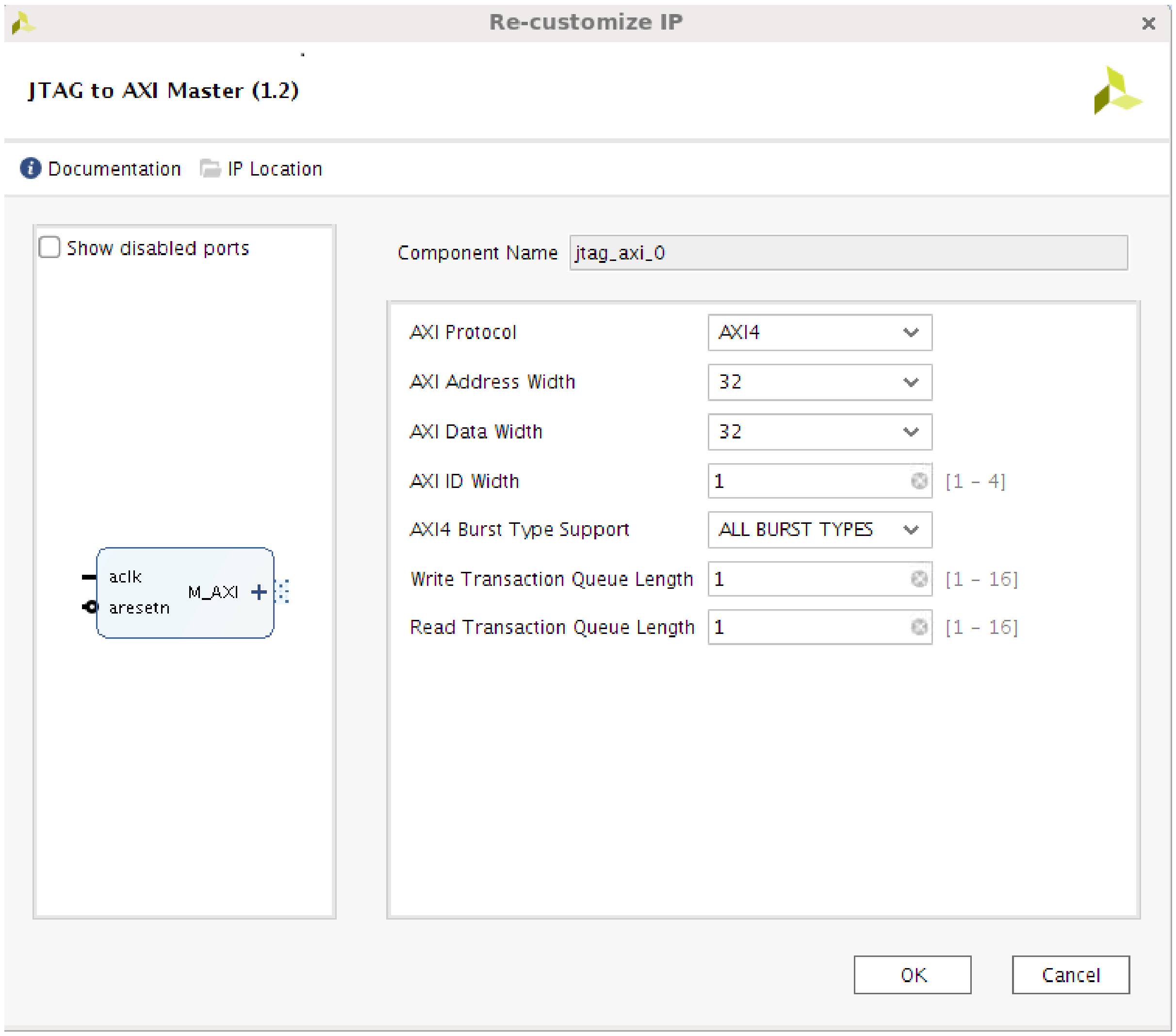
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
synth_1	constrs_1	Not started															Vivado Synthesis Defa
impl_1	constrs_1	Not started															Vivado Implementatio

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
Search: JTAG to AXI (1 match)
JTAG to AXI Master
This design is empty. Press the + button to add IP.
roperties
Repor
ENTER to select, ESC to cancel, Ctrl+Q for IP details

```
shing IF
ser IP r
ed Vivac
 = 00:C
x.com:k
ect_2/pr
: source
46/installs/lin64/Vivado/2018.3/data/ip'.
 = 6768.984 ; gain = 7.852 ; free physical = 574
d>
```

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.

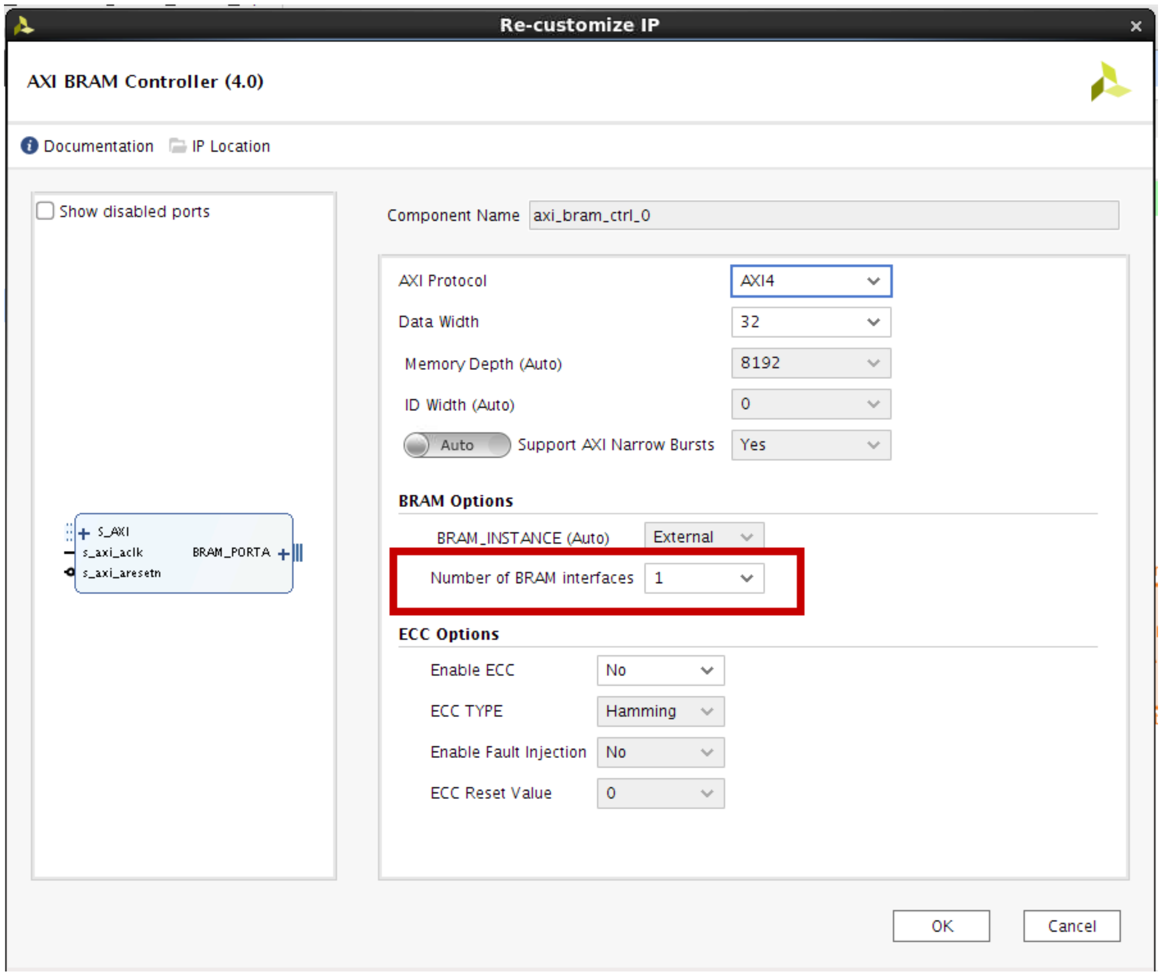


Component Name: jtag_axi_0

AXI Protocol	AXI4	
AXI Address Width	32	
AXI Data Width	32	
AXI ID Width	1	[1 - 4]
AXI4 Burst Type Support	ALL BURST TYPES	
Write Transaction Queue Length	1	[1 - 16]
Read Transaction Queue Length	1	[1 - 16]

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**.


Re-customize IP

AXI BRAM Controller (4.0)

Documentation IP Location

Show disabled ports

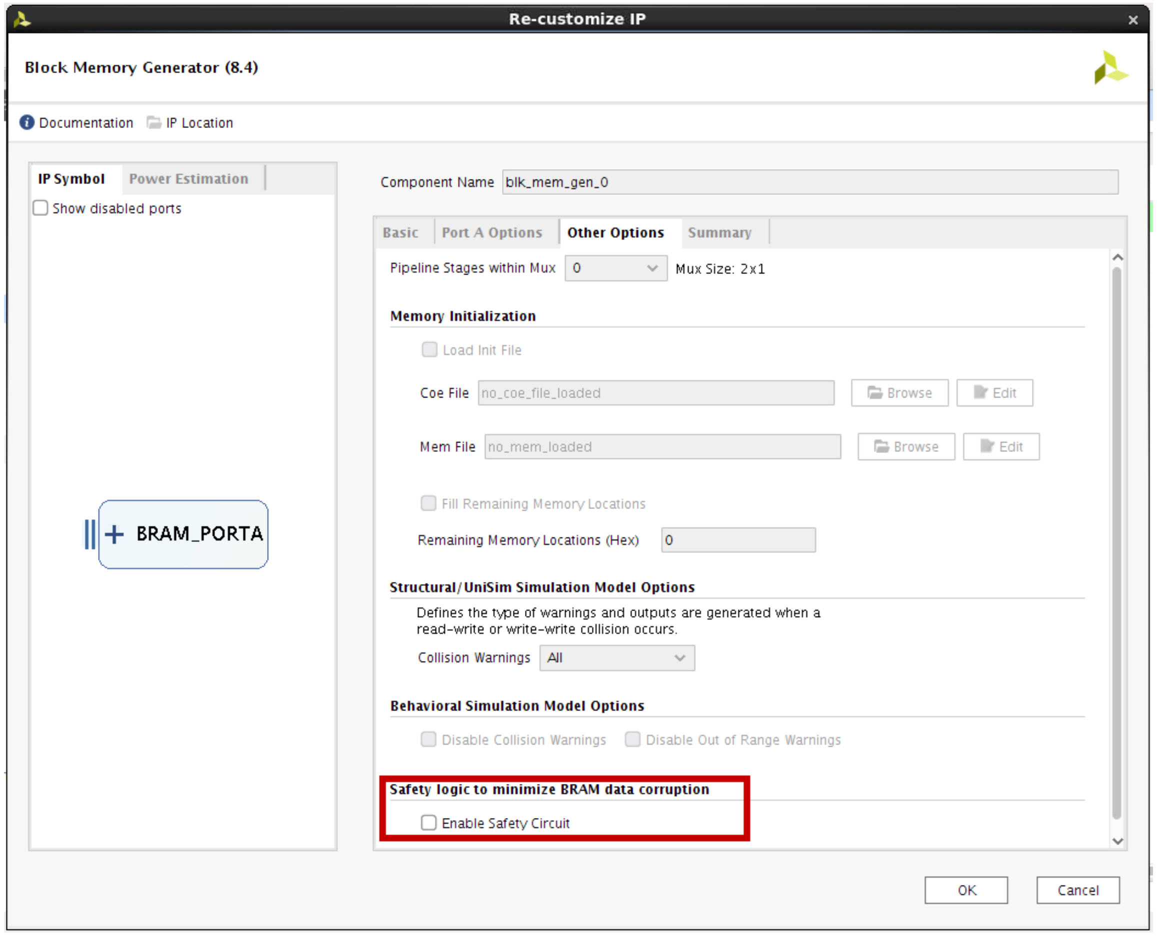
Component Name axi_bram_ctrl_0

AXI Protocol: AXI4
Data Width: 32
Memory Depth (Auto): 8192
ID Width (Auto): 0
Auto Support AXI Narrow Bursts: Yes

BRAM Options
BRAM_INSTANCE (Auto): External
Number of BRAM interfaces: 1

ECC Options
Enable ECC: No
ECC TYPE: Hamming
Enable Fault Injection: No
ECC Reset Value: 0

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**.


Re-customize IP

Block Memory Generator (8.4)

Documentation IP Location

IP Symbol Power Estimation

Show disabled ports

Component Name: blk_mem_gen_0

Basic Port A Options Other Options Summary

Pipeline Stages within Mux: 0 Mux Size: 2x1

Memory Initialization

Load Init File

Coe File: no_coe_file_loaded Browse Edit

Mem File: no_mem_loaded Browse Edit

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

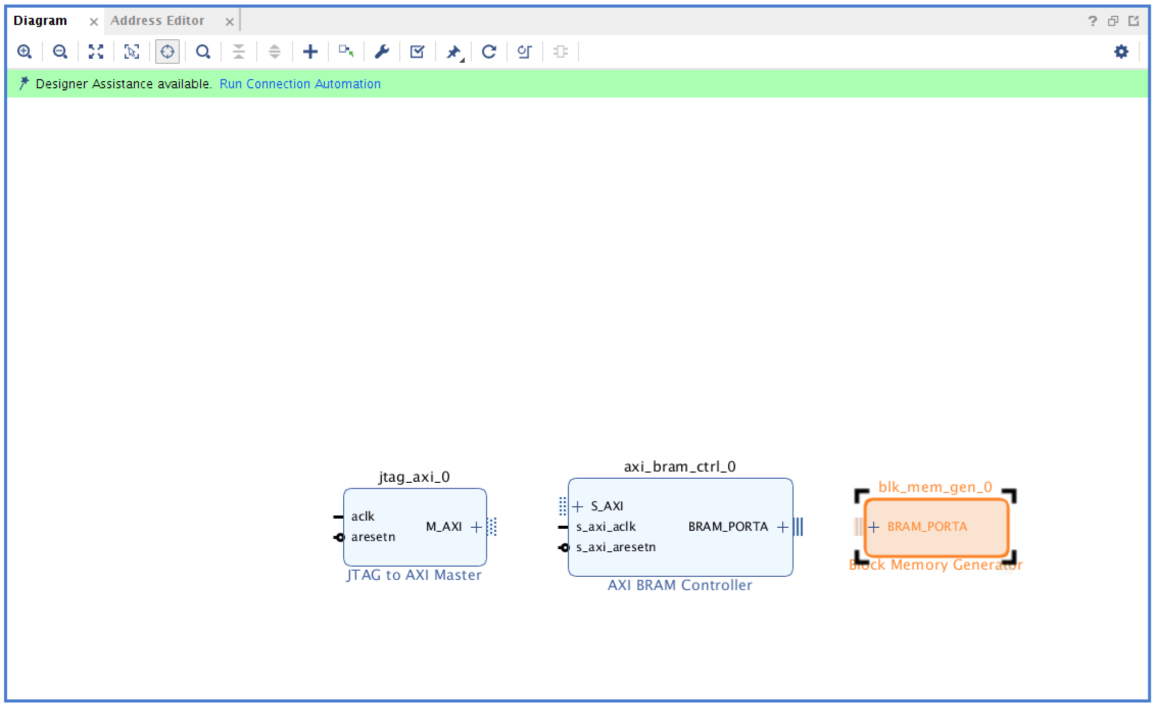
Disable Collision Warnings Disable Out of Range Warnings

Safety logic to minimize BRAM data corruption

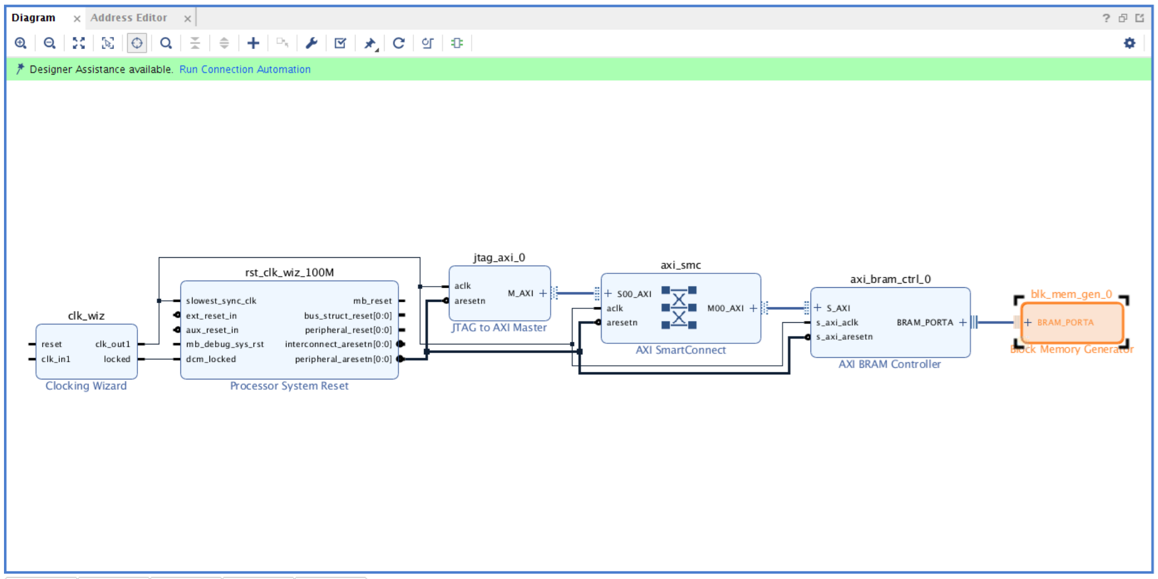
Enable Safety Circuit

OK Cancel

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



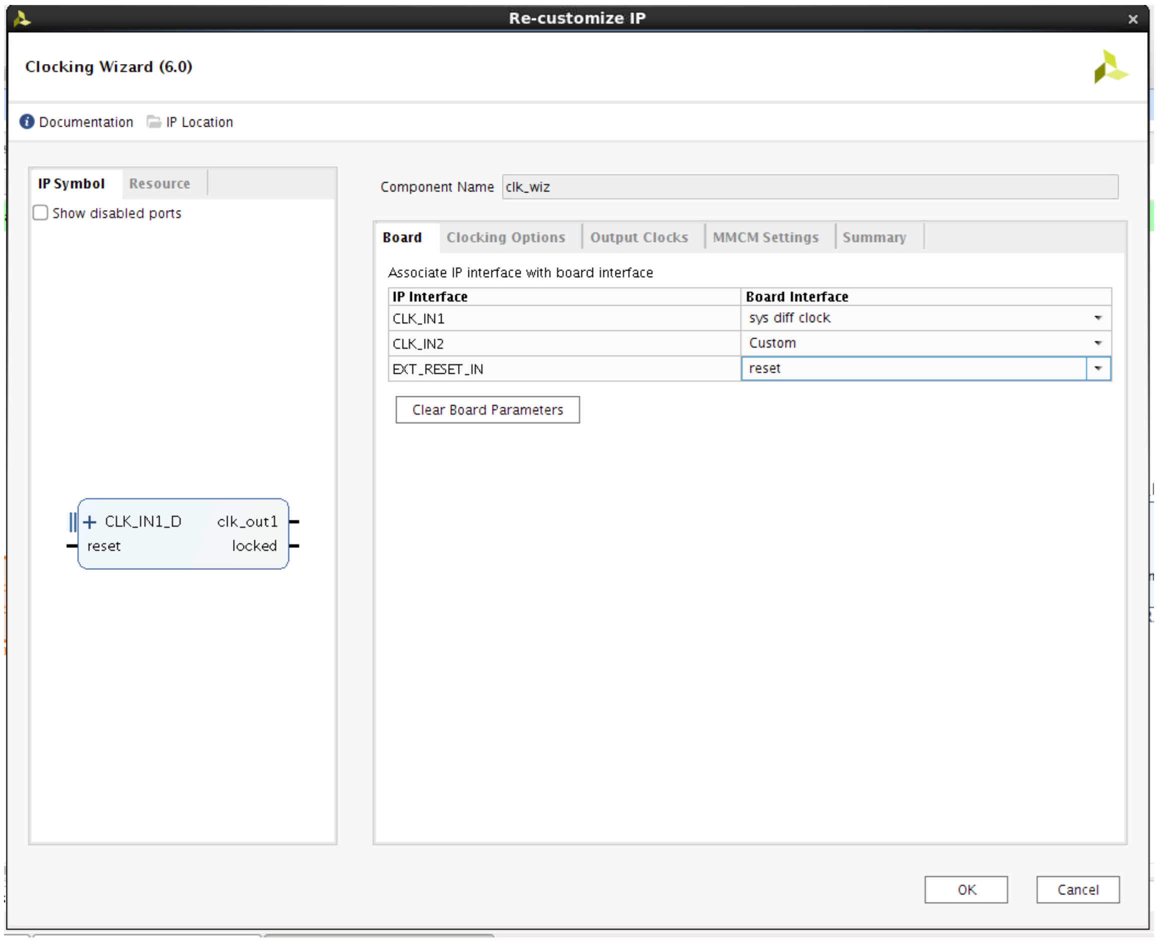
14. Notice the green banner indicating that Designer Assistance is available at the top of the block diagram canvas. Click the **Run Connection Automation** button on this banner. When the Connection Automation window appears, click the radio button for All Automation, then click **OK**.



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.



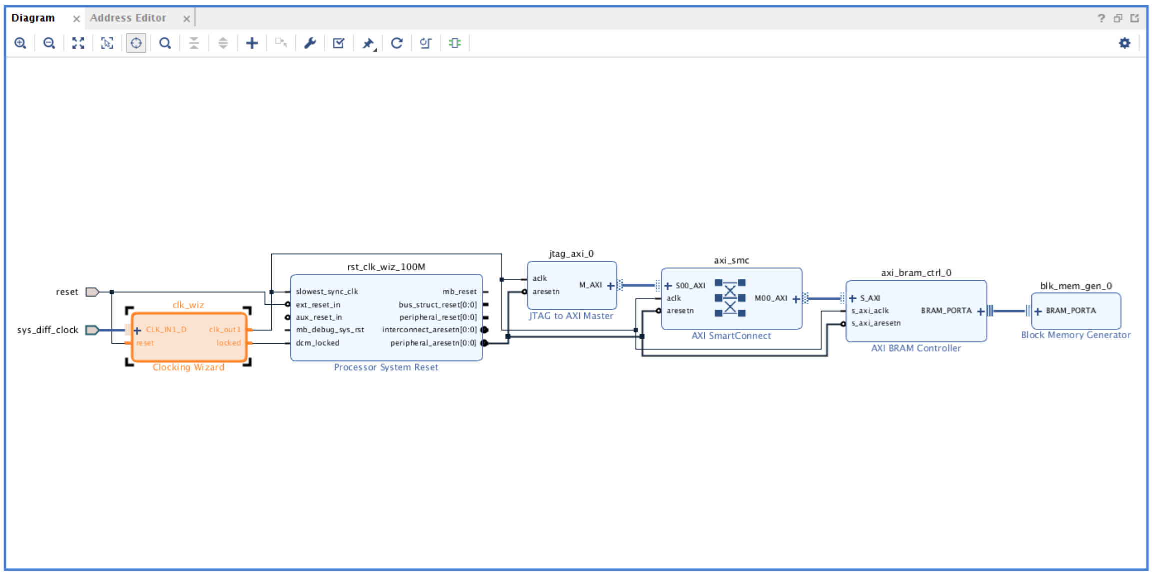
Component Name clk_wiz

Associate IP interface with board interface

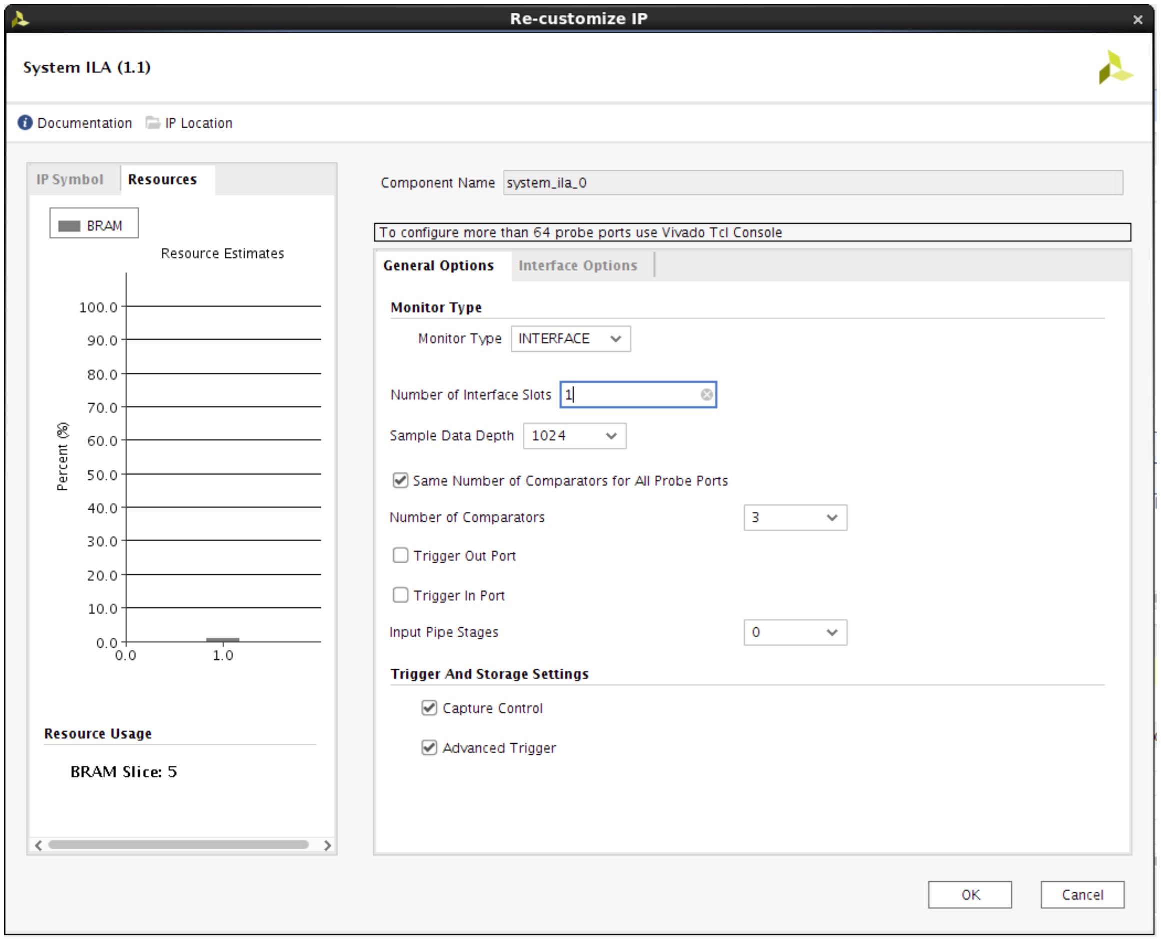
IP Interface	Board Interface
CLK_IN1	sys diff clock
CLK_IN2	Custom
EXT_RESET_IN	reset

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.



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**.



Resource Usage

BRAM Slice: 5

Component Name system_ila_0

To configure more than 64 probe ports use Vivado Tcl Console

Monitor Type

Monitor Type: INTERFACE

Number of Interface Slots: 1

Sample Data Depth: 1024

☑ Same Number of Comparators for All Probe Ports

Number of Comparators: 3

☐ Trigger Out Port

☐ Trigger In Port

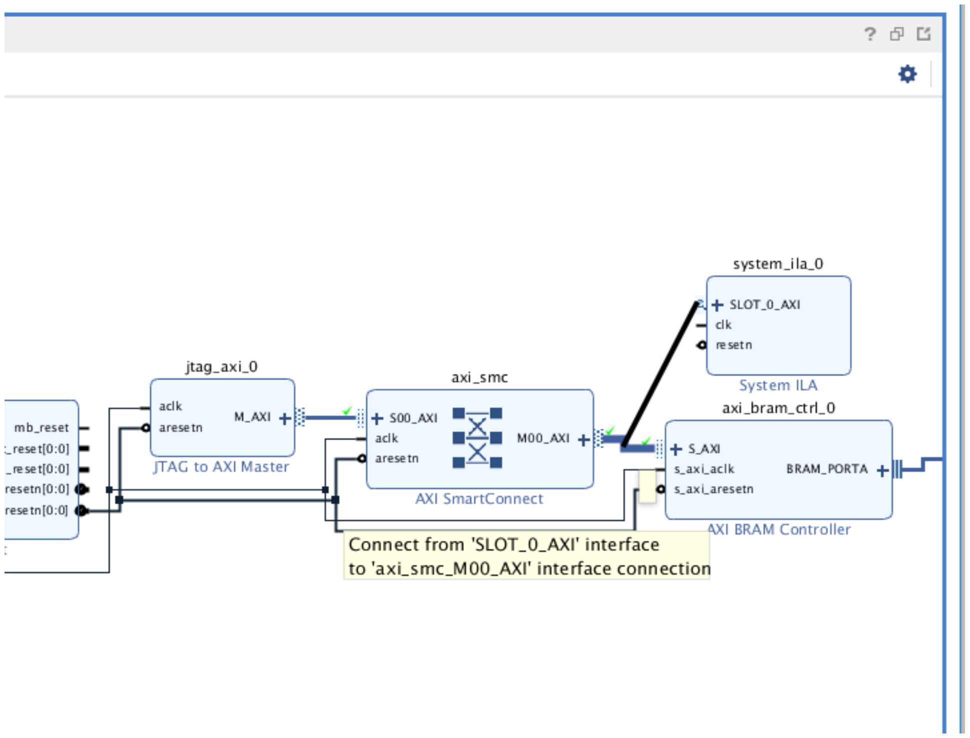
Input Pipe Stages: 0

Trigger And Storage Settings

☑ Capture Control

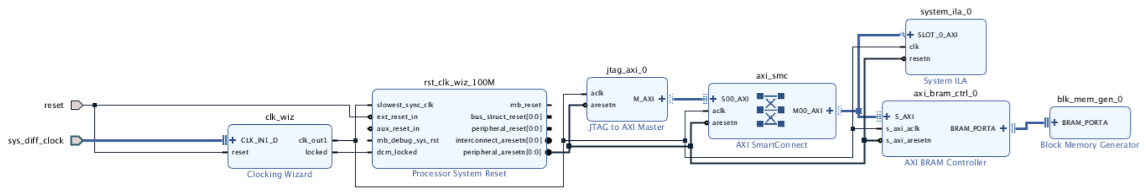
☑ Advanced Trigger

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.



Connect from 'SLOT_0_AXI' interface
to 'axi_smc_M00_AXI' interface connection

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→Save Block Design**. Select **File→Close Block Design** in the same menu to close the block design.

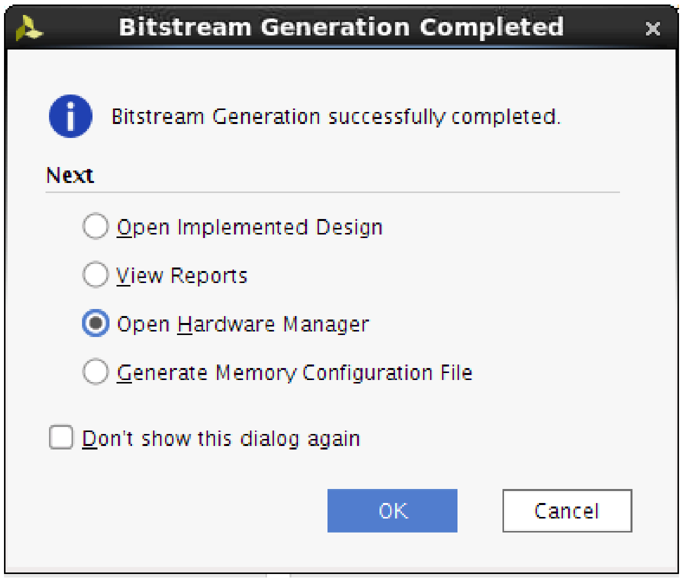
23. In the sources window, right-click on **design_1 block design** and select **Create HDL Wrapper**. Allow Vivado IDE to manage the wrapper, and click **OK**.

24. In the Flow Navigator on the left side of the Vivado IDE, click **Generate Bitstream**.

25. Click **OK** to implement the design.

26. Wait until the Vivado Status window shows write_bitstream complete.

27. In the Bitstream Generation Completed dialog, select **Open Hardware Manager**, and click **OK**.


Bitstream Generation Completed

Bitstream Generation successfully completed.

Next

Open Implemented Design

View Reports

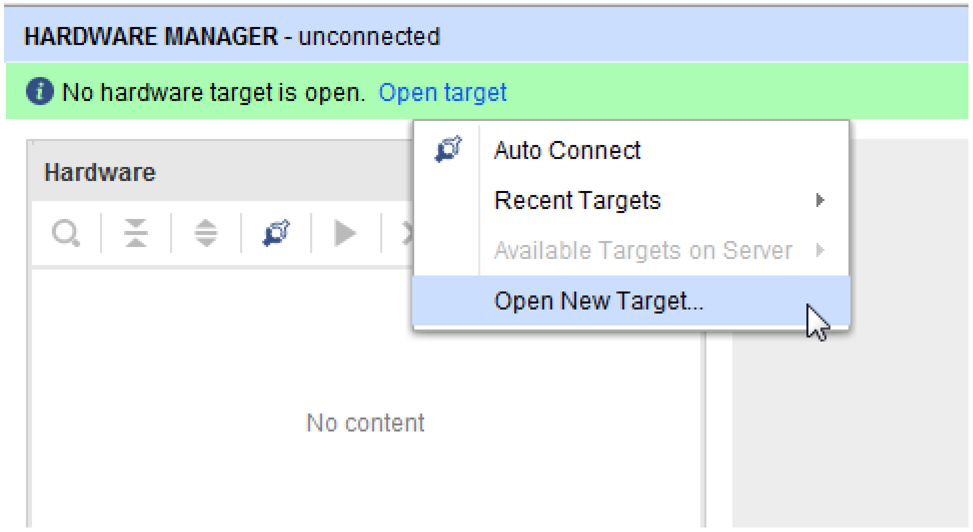
[✓ Open Hardware Manager]

Generate Memory Configuration File

Don't show this dialog again

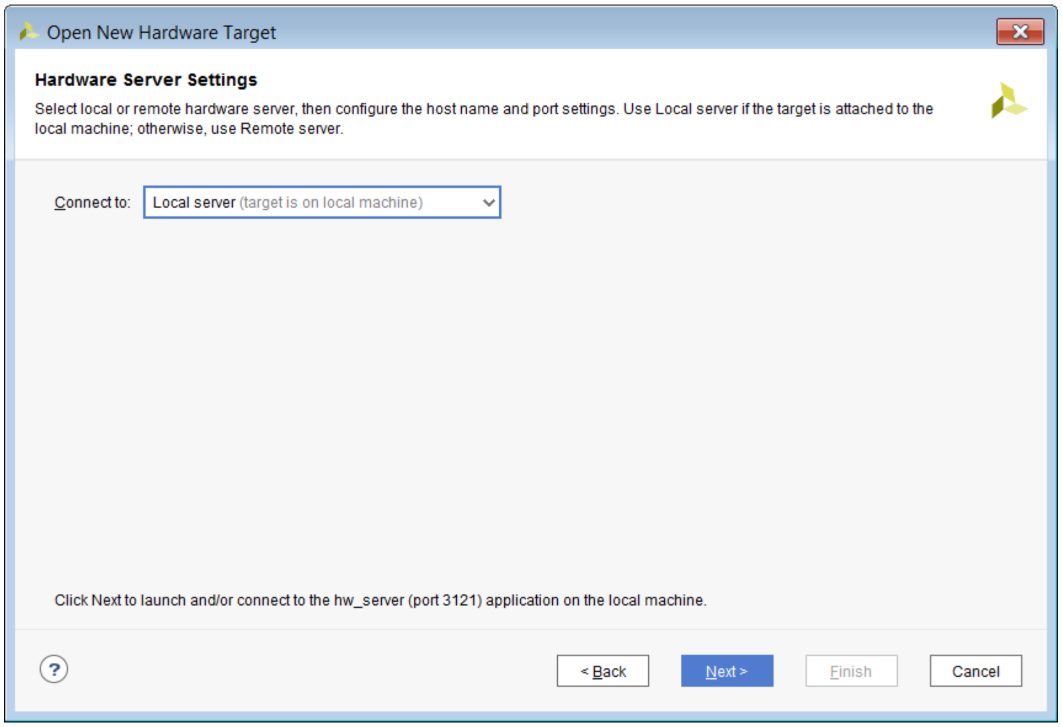
Step 2: Program the KC705 Board and Interact with the JTAG to AXI Master Core

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.


HARDWARE MANAGER - unconnected

No hardware target is open. Open target

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



Hardware Server Settings

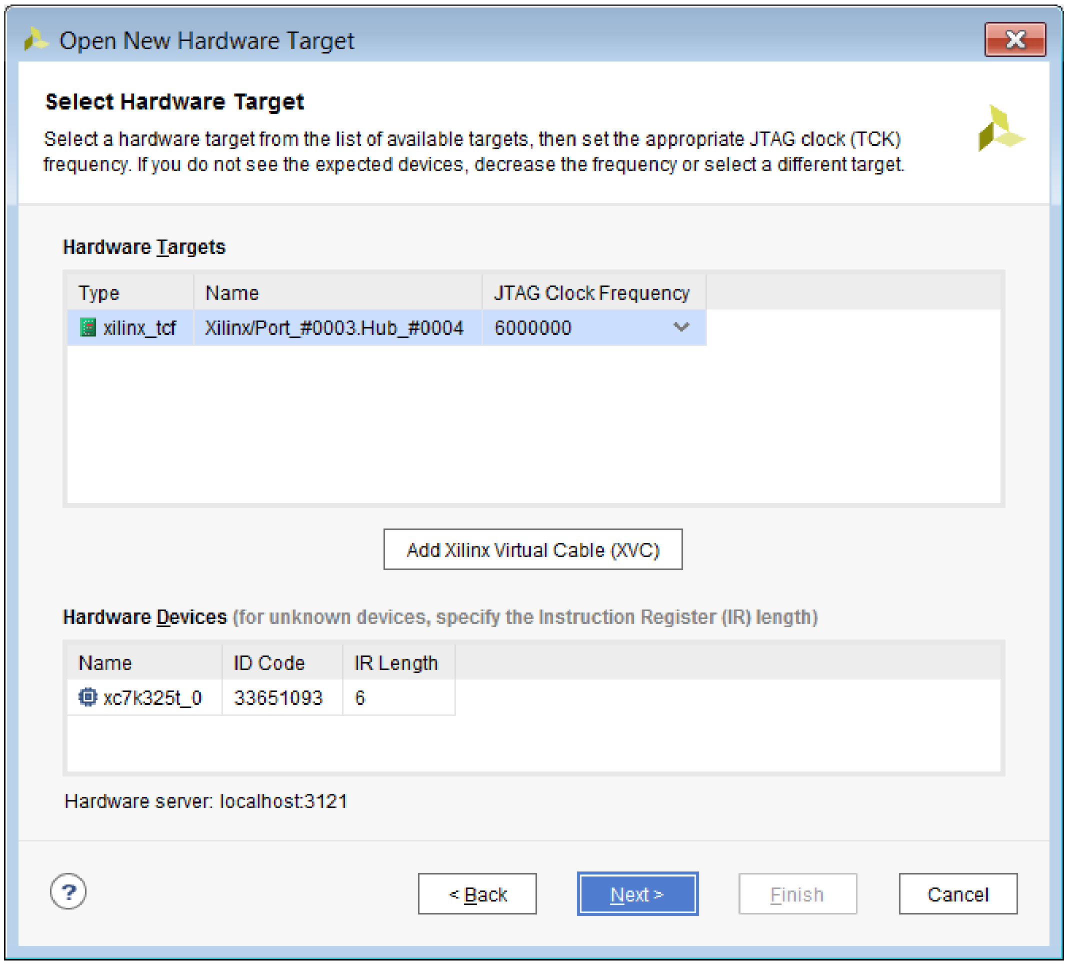
Select local or remote hardware server, then configure the host name and port settings. Use Local server if the target is attached to the local machine; otherwise, use Remote server.

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.

***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**.



Select Hardware Target

Select a hardware target from the list of available targets, then set the appropriate JTAG clock (TCK) frequency. If you do not see the expected devices, decrease the frequency or select a different target.

Hardware Targets

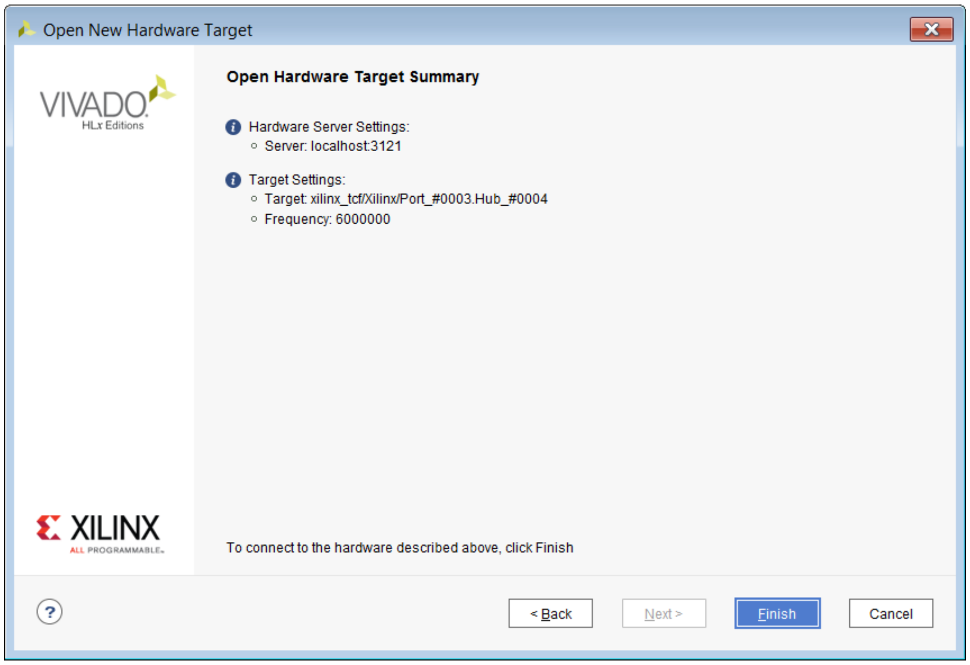
Type	Name	JTAG Clock Frequency
xilinx_tcf	Xilinx/Port_#0003.Hub_#0004	6000000

Hardware Devices (for unknown devices, specify the Instruction Register (IR) length)

Name	ID Code	IR Length
xc7k325t_0	33651093	6

Hardware server: localhost:3121

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.



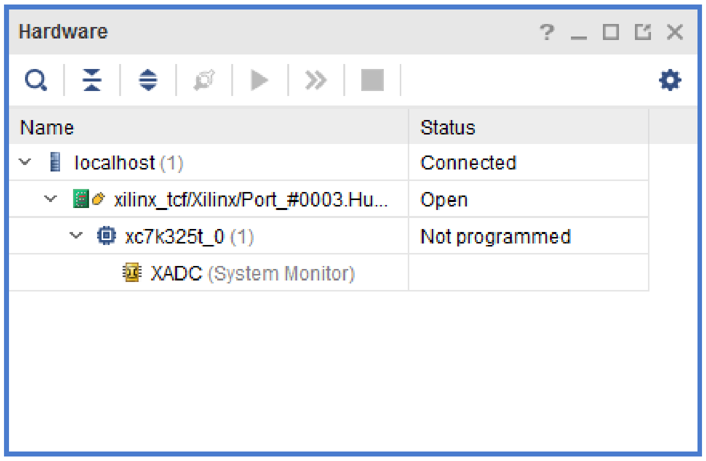
Open Hardware Target Summary

- Hardware Server Settings:
  - Server: localhost:3121
- Target Settings:
  - Target: xilinx_tcf/Xilinx/Port_#0003.Hub_#0004
  - Frequency: 6000000

To connect to the hardware described above, click Finish

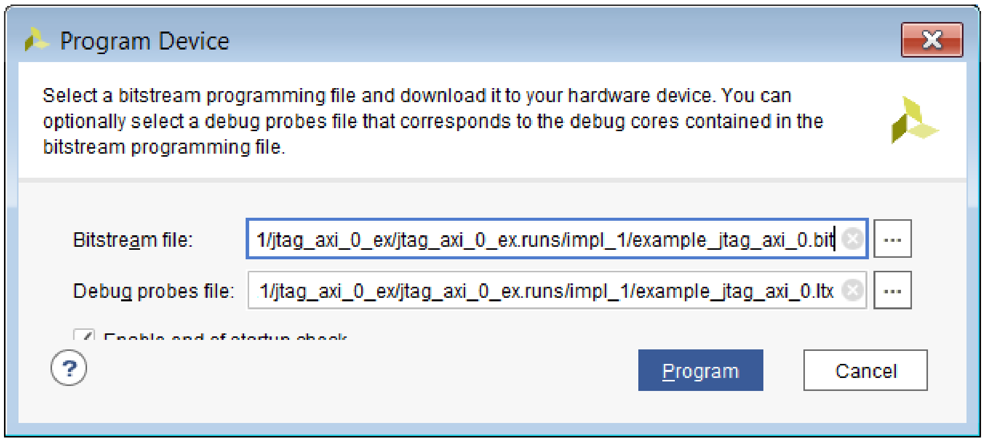
Wait for the connection to the hardware to complete. After the connection to the hardware target is made, the Hardware dialog shown in the following figure opens.

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



Name	Status
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

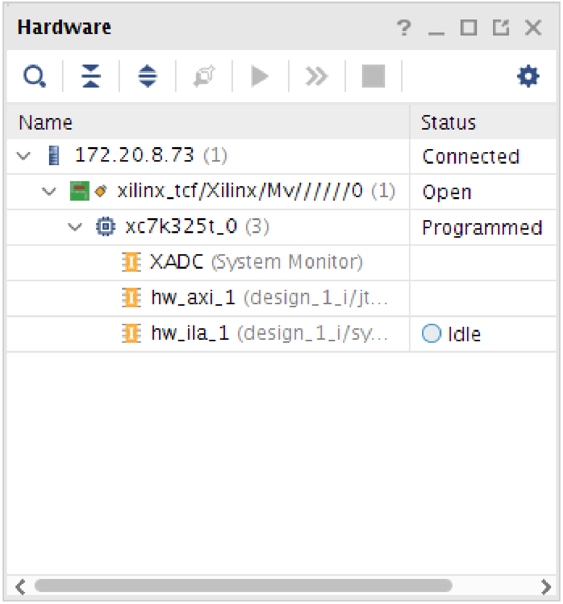
Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file: 1/jtag_axi_0_ex/jtag_axi_0_ex.runs/impl_1/example_jtag_axi_0.bit

Debug probes file: 1/jtag_axi_0_ex/jtag_axi_0_ex.runs/impl_1/example_jtag_axi_0.ltx

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

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

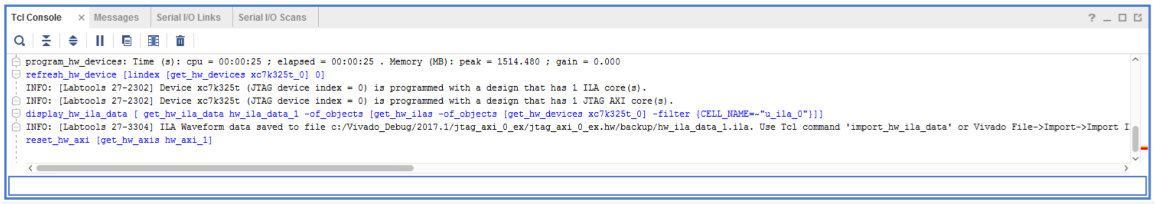


Hardware

Name	Status
172.20.8.73 (1)	Connected
xilinx_tcf/Xilinx/Mv//////0 (1)	Open
xc7k325t_0 (3)	Programmed
XADC (System Monitor)	
hw_axi_1 (design_1_i/jt...	
hw_ila_1 (design_1_i/sy...	Idle

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

```
reset_hw_axi [get_hw_axis hw_axi_1]
```



```
program_hw_devices: Time (s): cpu = 00:00:25 ; elapsed = 00:00:25 . Memory (MB): peak = 1514.480 ; gain = 0.000
refresh_hw_device [lindex [get_hw_devices xc7k325t_0] 0]
INFO: [Labtools 27-2302] Device xc7k325t (JTAG device index = 0) is programmed with a design that has 1 ILA core(s).
INFO: [Labtools 27-2302] Device xc7k325t (JTAG device index = 0) is programmed with a design that has 1 JTAG AXI core(s).
display_hw_ila_data [ get_hw_ila_data hw_ila_data_1 -of_objects [get_hw_ilas -of_objects [get_hw_devices xc7k325t_0] -filter {CELL_NAME=~"u_ila_0"}]]
INFO: [Labtools 27-3304] ILA Waveform data saved to file c:/Vivado_Debug/2017.1/jtag_axi_0_ex/jtag_axi_0_ex.hw/backup/hw_ila_data_1.ila. Use Tcl command 'import_hw_ila_data' or Vivado File->Import->Import I
reset_hw_axi [get_hw_axis hw_axi_1]
```

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 {44444444_33333333_22222222_11111111}` is the data to be written.

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

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

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

    where:

    - `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 :
    44444444333333332222222211111111…
    ```

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

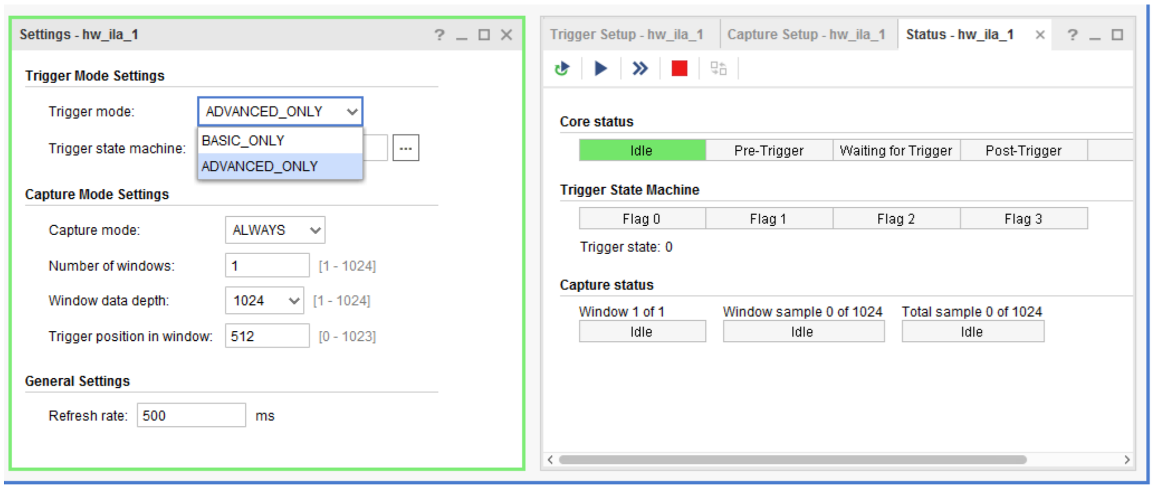
    ```
    run_hw_axi $rt
    ```

    This command should return the following:

    ```
    INFO: [Labtools 27-147] : READ DATA is :
    44444444333333332222222211111111…
    ```

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

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



Settings - hw_ila_1

Trigger Mode Settings

Trigger mode: ADVANCED_ONLY

Trigger state machine:

BASIC_ONLY
ADVANCED_ONLY

Capture Mode Settings

Capture mode: ALWAYS

Number of windows: 1 [1 - 1024]

Window data depth: 1024 [1 - 1024]

Trigger position in window: 512 [0 - 1023]

General Settings

Refresh rate: 500 ms

Trigger Setup - hw_ila_1 | Capture Setup - hw_ila_1 | Status - hw_ila_1

Core status

Idle	Pre-Trigger	Waiting for Trigger	Post-Trigger

Trigger State Machine

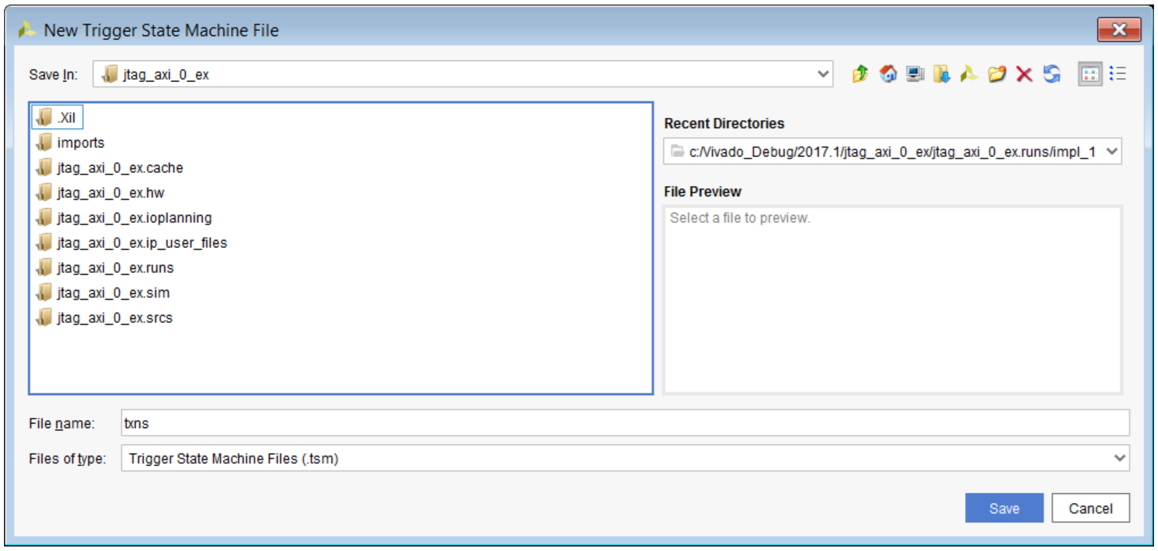
Flag 0	Flag 1	Flag 2	Flag 3

Trigger state: 0

Capture status

Window 1 of 1	Window sample 0 of 1024	Total sample 0 of 1024
Idle	Idle	Idle

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



New Trigger State Machine File

Save In: jtag_axi_0_ex

.Xil
imports
jtag_axi_0_ex.cache
jtag_axi_0_ex.hw
jtag_axi_0_ex.ioplanning
jtag_axi_0_ex.ip_user_files
jtag_axi_0_ex.runs
jtag_axi_0_ex.sim
jtag_axi_0_ex.srcs

Recent Directories

c:/Vivado_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)

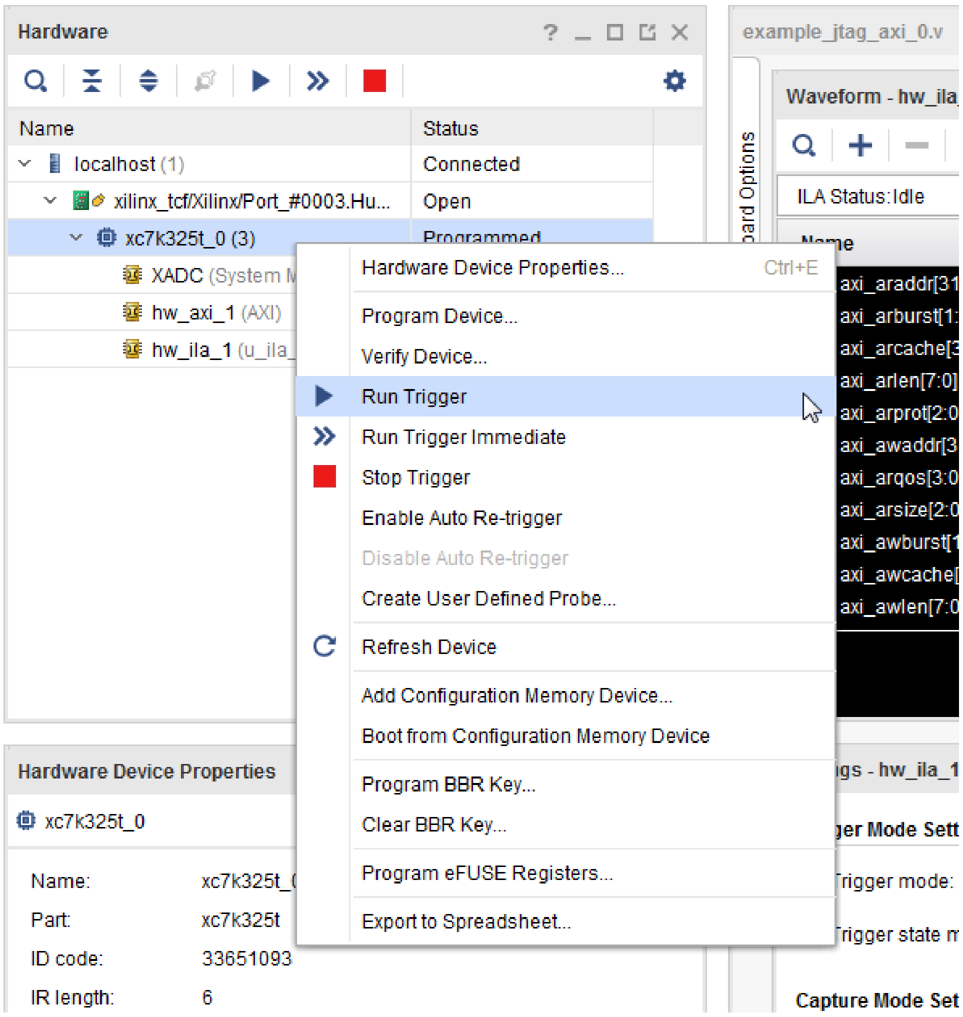
5. A basic template of the trigger state machine script is displayed in the Trigger State Machine gadget. Expand the trigger state machine gadget in the ILA dashboard. Copy the script below after line 17 of the state machine script and save the file.

```
# 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
```

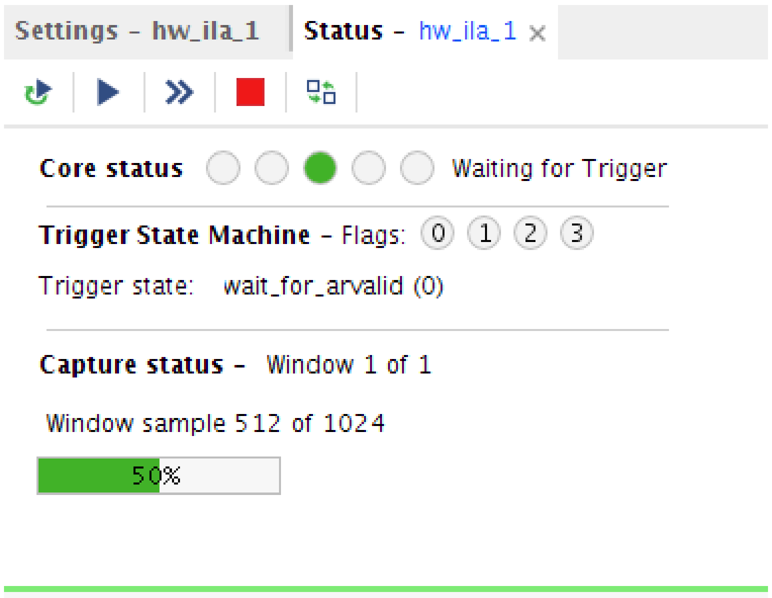
***Note*:** Use the state machine to detect the various phases of an AXI read transaction:

- 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**.



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

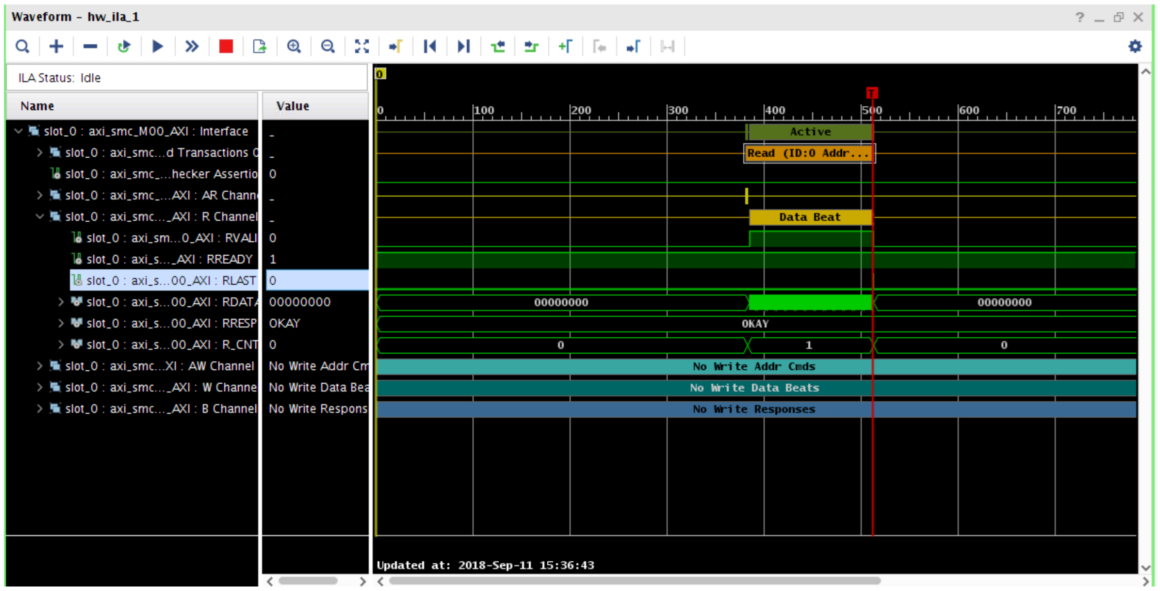
Window sample 512 of 1024

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.



Waveform – hw_ila_1

ILA Status: Idle

Name	Value
slot_0 : axi_smc_M00_AXI : Interface	_
slot_0 : axi_smc...d Transactions C	_
slot_0 : axi_smc_...hecker Assertio	0
slot_0 : axi_smc_...AXI : AR Chann	_
slot_0 : axi_smc..._AXI : R Channel	_
slot_0 : axi_sm...0_AXI : RVALI	0
slot_0 : axi_s..._AXI : RREADY	1
slot_0 : axi_s...00_AXI : RLAST	0
slot_0 : axi_s...00_AXI : RDATA	00000000
slot_0 : axi_s...00_AXI : RRESP	OKAY
slot_0 : axi_s...00_AXI : R_CNT	0
slot_0 : axi_smc...XI : AW Channel	No Write Addr Cm
slot_0 : axi_smc..._AXI : W Channe	No Write Data Bea
slot_0 : axi_smc..._AXI : B Channel	No Write Respons

Updated at: 2018-Sep-11 15:36:43

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

AUTOMOTIVE APPLICATIONS DISCLAIMER

Copyright