Notice of Disclaimer

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

©Copyright 2012-2013 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>07/25/12</td>
<td>2012.2</td>
<td>Initial Xilinx release.</td>
</tr>
<tr>
<td>10/16/12</td>
<td>2012.3</td>
<td>• Added section for Synplify EDIF flow.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Various editorial fixes and updated screen shots to track changes to dialog boxes.</td>
</tr>
<tr>
<td>12/18/2012</td>
<td>2012.4</td>
<td>Revalidated for the current release. Editorial updates only; no technical content updates.</td>
</tr>
<tr>
<td>03/20/2013</td>
<td>2013.1</td>
<td>• Converted source to Word.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated DDS IP and add VIO 2.0</td>
</tr>
<tr>
<td>06/19/2013</td>
<td>2013.2</td>
<td>Editorial updates only; no technical content updates.</td>
</tr>
<tr>
<td>07/07/2013</td>
<td>2013.2</td>
<td>Updated design source files.</td>
</tr>
</tbody>
</table>
# Table of Contents

Vivado Design Suite Tutorial: Programming and Debugging ................................................................. 1

Revision History ............................................................................................................................................. 2

Debugging with Vivado® ILA 2.0 and Integrated Logic Analyzer ................................................................. 5

Introduction .................................................................................................................................................. 5

Prerequisites .................................................................................................................................................. 5

Getting Started ............................................................................................................................................ 6

Lab 1: Using the Netlist Insertion Method for Debugging a Design in Vivado® ....................................... 10

Introduction ................................................................................................................................................ 10

Step 1: Creating a Project with the Vivado New Project Wizard ............................................................. 10

Step 2: Synthesizing the Design ................................................................................................................ 11

Step 3: Probing and Adding Debug IP ..................................................................................................... 12

Step 4: Implementing and Generating Bitstream .................................................................................... 16

Lab 2: Using the HDL Instantiation Method for Debugging a Design in Vivado® ................................... 18

Introduction ................................................................................................................................................ 18

Step 1: Creating a Project with the Vivado New Project Wizard ............................................................. 18

Step 2: Synthesize, Implement and Generate Bitstream ......................................................................... 20

Lab 3: Using a VIO Core for Debugging a Design in Vivado® ................................................................. 21

Introduction ................................................................................................................................................ 21

Step 1: Creating a Project with the Vivado New Project Wizard ............................................................. 22

Step 2: Synthesize, Implement and Generate Bitstream ......................................................................... 24

Lab 4: Using the Synplify Pro Synthesis Tool and Vivado® for Debugging a Design ............................. 25

Introduction ................................................................................................................................................ 25

Step 1: Create a Synplify Pro Project ........................................................................................................ 25

Step 2: Synthesize the Synplify Project ................................................................................................... 32

Step 3: Create EDIF Netlists for the Black Box Created in Synplify Pro .................................................... 33

Step 4: Create a Post Synthesis Project in Vivado IDE ........................................................................... 34

Step 5: Add (more) Debug Nets to the Project ....................................................................................... 36

Step 6: Implementing the Design and Generating the Bitstream ............................................................ 38
Using the Vivado® Logic Analyzer to Debug the Hardware................................................................. 39
Introduction .................................................................................................................................................. 39
Step 1: Verifying Operation of the Sine Wave Generator................................................................. 39
Step 2: Debugging the Sine Wave Sequencer State Machine (Optional)........................................ 49
Verifying the VIO Core Activity (Only applicable to Lab 3)................................................................ 52
Additional Resources ............................................................................................................................... 57
Xilinx Resources ........................................................................................................................................ 57
Solution Centers ....................................................................................................................................... 57
References ............................................................................................................................................... 57
Chapter 1

Debugging with Vivado® ILA 2.0 and Integrated Logic Analyzer

Introduction

In this tutorial exercise, you will quickly learn how to debug your FPGA designs by inserting an Integrated Logic Analyzer (ILA) core using the Vivado® Integrated Design Environment (IDE). You will take advantage of integrated Vivado logic analyzer functions to debug and discover some potential root causes of your design, thereby allowing you to quickly address issues.

Example RTL designs are used to illustrate overall integration flows between Vivado logic analyzer, ILA 2.0, and Vivado IDE. In order to be successful using this tutorial, you should have some basic knowledge of Xilinx® ISE® Design Suite and Vivado Design Suite tool flows.

Prerequisites

A basic knowledge of Xilinx ISE Design Suite and Vivado Design Suite tool flows.

Objectives

This tutorial:

- Shows you how to take advantage of integrated Vivado logic analyzer features in the Vivado design environment that make the debug process faster and simpler.
- Provides specifics on how to use the Vivado Integrated Design Environment (IDE) and the Vivado logic analyzer to debug common problems in FPGA logic designs.

After completing this tutorial, you will be able to:

- Validate and debug your design using the Vivado Integrated Design Environment and the Integrated Logic Analyzer (ILA) core.
- Understand how to create an RTL project, probe your design, insert an ILA 2.0 core, and implement the design in the Vivado Integrated Design Environment.
- Generate and customize an IP core netlist in the Vivado Integrated Design Environment.
- Debug the design using Vivado logic analyzer in real-time, and iterate the design using the Vivado Integrated Design Environment and a KC705 Evaluation Kit Base Board that incorporates a Kintex™-7 device.
Getting Started

Setup Requirements

Before you start this tutorial, make sure you have and understand the hardware and software components needed to perform the steps. The following subsections list the requirements.

Software

- Vivado Design Suite 2013.2

Hardware

- Kintex-7 FPGA KC705 Evaluation Kit Base Board.

Figure 1: KC705 Board Showing Key Components

Tutorial Design Components

The design includes:

- A simple control state machine.
- Three sine wave generators using AXI-Streaming interface, native DDS Compiler.
- Common push buttons (GPIO_BUTTON).
- DIP switches (GPIO_SWITCH).
- LED displays (GPIO_LED).
**Push Button Switches:** Serve as inputs to the debounce 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 debounce circuit.

**Debounce 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 pulses from the two push button switches. Provides sine wave selection and indicator circuits, sequencing between 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.

**Board Support and Pinout Information**

Table 1: Pinout Information for the KC705 Board

<table>
<thead>
<tr>
<th>Pinout Locations</th>
<th>Function</th>
<th>Pinout Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK_N</td>
<td>AD11</td>
<td>Clock</td>
</tr>
<tr>
<td>CLK_P</td>
<td>AD12</td>
<td>Clock</td>
</tr>
<tr>
<td>GPIO_BUTTONS[0]</td>
<td>AA12</td>
<td>Reset</td>
</tr>
<tr>
<td>GPIO_BUTTONS[1]</td>
<td>AG5</td>
<td>Sine Wave Sequencer</td>
</tr>
<tr>
<td>GPIO_SWITCH</td>
<td>Y28</td>
<td>Debounce Circuit Selector</td>
</tr>
<tr>
<td>LEDS_n[0]</td>
<td>AB8</td>
<td>Sine Wave Selection[0]</td>
</tr>
<tr>
<td>LEDS_n[1]</td>
<td>AA8</td>
<td>Sine Wave Selection[1]</td>
</tr>
<tr>
<td>LEDS_n[2]</td>
<td>AC9</td>
<td>Reserved</td>
</tr>
<tr>
<td>LEDS_n[3]</td>
<td>AB9</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Design Files

In your C: drive, create a folder called /Vivado_Debug.

Find the tutorial source files.

Design file location:

*Note: The tutorial and design files might be updated or modified in between software releases on the Xilinx website, where you can download the latest version of the materials.*

Unzip the tutorial source file to the /Vivado_Debug folder. There are four 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). The files required for this lab are as shown below:

- debounce.vhd
- fsm.vhd
- rt.tcl
- sinegen.vhd
- sinegen_demo.vhd
- sine_high.xci
- sine_low.xci
- 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. The files required for this lab are as shown below:

- debounce.vhd
- fsm.vhd
- sinegen.vhd
- sinegen_demo_inst.vhd
- sine_high.xci
- sine_low.xci
- sine_mid.xci
Lab 3: You can test your design even if the hardware is not physically accessible, using a VIO core. This lab will walk you over the steps of instantiating and customizing a VIO core that you will hook to the I/Os of the design. The files needed for this lab are as follows:

- debounce.vhd
- fsm.vhd
- sinegen.vhd
- sinegen_demo_inst_vio.vhd
- sine_high.xci
- sine_low.xci
- sinde_mid.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 will walk you through the steps of marking nets for debug in Synplify tool and then using Vivado to perform the rest of the debug. The following files are needed for this lab:

- dds_compiler_v6_0xst.edn
- dds_compiler_v6_0_xst_parameterized1.edn
- dds_compiler_v6_0_xst_parameterized3.edn
- debounce.vhd
- fsm.vhd
- sine_high.xci
- sine_low.xci
- sine_mid.xci
- sinegen.edn
- sinegen_synplify.vhd
- synplify_1.sdc
- sinegen_demo_kc705.xdc
Lab 1: Using the Netlist Insertion Method for Debugging a Design in Vivado®

Introduction

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

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

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

1. Invoke the Vivado IDE.
2. In the Getting Started screen, click Create New Project to start the New Project wizard. Click Next.
3. In the Project Name screen, name the new project proj_netlist and provide the project location (C:/Vivado_Debug). Ensure that Create Project Subdirectory is checked and click Next.
4. In the Project Type screen, specify the Type of Project to create as RTL Project and click Next.
5. In the Add Sources screen:
   a. Set Target Language to VHDL.
   b. Click the Add Files button.
   c. In the Add Source Files dialog box, navigate to the /src/Lab 1 directory.
   d. Select all VHD source files, and click OK.
Step 2: Synthesizing the Design

In the Project Manager, click Project Settings.

1. In the Project Settings dialog box select Synthesis from the left pane and change flatten_hierarch option to none. Click OK.

2. Verify that the files are added, and Copy Sources into Project is checked. Click Next.

6. In the Add Existing IP (optional) dialog box:
   a. Click the Add Files button.
   b. In the Add Configurable IP dialog box, navigate to the /src directory.
   c. Select all XCI source files, and click OK.
   d. Verify that the files are added, and Copy Sources into Project is checked. Click Next.

7. In the Add Constraints (optional) dialog box, the provided XDC file, sinegen_demo_kc705.xdc should automatically appear in the main window. 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, and click Next.

9. Review the New Project Summary screen. 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 3: Probing and Adding Debug IP

To add a Vivado ILA 2.0 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.

Note: The reason for changing this setting to none is to prevent the synthesis tool from performing any boundary optimizations for this tutorial.

3. In the Vivado Flow Navigator, expand the Synthesis folder, and click the Run Synthesis button.

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

4. In the Synthesis Completed dialog box, click Cancel. You will implement the design later.
• Generate the Bitstream.

Adding Debug Nets to the Project

Following are some examples of how to add debug nets using the Vivado IDE:

• Add mark_debug attribute to the target XDC file

```bash
set_property mark_debug true [get_nets sine*]
```

**Note:** Use these attributes in synthesized designs only. Do not use them with pre-synthesis or elaborated design netlists.

• Add mark_debug attribute to HDL files

VHDL

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

Verilog

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

• Right-click and select **Mark Debug** or **Unmark Debug** on Synthesis netlist.

• Use a Tcl prompt to set the mark_debug attribute. For example,

```bash
set mark_debug true [get_nets sine*]
```

This applies the mark_debug on the current, open netlist.

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

1. From the **Synthesis** pull-down, click **Open Synthesized Design**.

**Note:** Before proceeding, make sure that the **Flow Navigator** on the left panel is enabled. Use **Ctrl-Q** to toggle it off and on.

2. Click the **Debug** window if it is not already selected.

3. Expand **Unassigned Debug Nets** folder. Figure 4 shows those debug nets that were tagged in the *sinegen_demo.vhd* with mark_debug attributes, as shown in Figure 3.
4. Select the **Netlist** tab and expand **Nets**. Select the following nets as shown in Figure 5 for debugging.
   - **GPIO_BUTTONS_IBUF[0]** and **GPIO_BUTTONS_IBUF[1]** - Nets folder under the top-level hierarchy
   - **sine(20)**- Nets folder under the **U_SINEGEN** hierarchy
   - **sel(2)** -Nets folder under the **U_SINEGEN** hierarchy

**Note:** These signals represent the significant behavior of this design and will be used to verify and debug the design in subsequent steps. Right-click the selected nets and select **Mark Debug**.
5. In the **Confirm Debug Net(s)** dialog box click **OK**.

**Note:** In the Debug window, you can see the unassigned nets you just selected. In Vivado IDE, you can also see the green bug icon next to each scalar or bus, which indicates that a net has the attribute `mark_debug = true` as shown in Figure 6.

6. From the **Debug** window or **Tools** menu, select **Set Up Debug**. The Set Up Debug wizard opens.
Figure 7: Launching the Set up Debug wizard

7. Click through the wizard to create Vivado logic analyzer debug cores, keeping the default settings.

---

**Step 4: Implementing and Generating Bitstream**

1. Click on **Generate Bitstream** from the **Program and Debug** folder in the **Flow Navigator**.

2. In the **Save Project** dialog box click on **Save**. This applies the mark debug attributes on the newly marked nets. You can see those constraints can by inspecting the `sinegen_demo_kc705.xdc` file.

3. When the **No Implementation Results Available** dialog box pops up, click **Yes**.

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. In the **Implementation is Out-of-date** dialog box, click **Yes**.

7. In the **Flow Navigator**, under **Implementation**, expand the **Implemented Design** folder and select **Report Timing Summary**.
Step 4: Implementing and Generating Bitstream

Figure 8: Analyze Timing Results

8. In the **Report Timing Summary** dialog box, click **OK**.

9. Ensure that all the specified timing constraints are met.

Figure 9: View the Timing Summary Report

10. Proceed to Chapter 6 to complete the rest of the steps for debugging the design.
Lab 2: Using the HDL Instantiation Method for Debugging a Design in Vivado®

Introduction

The HDL Instantiation method is one of the two methods supported in Vivado® Debug Probing. For this flow, you will generate an ILA 2.1 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 Getting Started screen, click Create New Project to start the New Project wizard. Click Next.
3. In the Project Name screen, name the new project proj_hdl and provide the project location (C:/Vivado_Debug). Ensure that Create Project Subdirectory is checked and click Next.
4. In the Project Type screen, specify the Type of Project to create as RTL Project and click Next.
5. In the Add Sources screen:
   a. Set Target Language to VHDL.
   b. Click the Add Files button.
   c. In the Add Source Files dialog box, navigate to the /src/Lab2 directory.
   d. Select all VHD source files, and click OK.
   e. Verify that the files are added, and Copy Sources into Project is checked. Click Next.
Step 1: Creating a Project with the Vivado New Project Wizard

6. In the **Add Existing IP (optional)** dialog box:
   a. Click the **Add Files** button.
   
   b. In the **Add Configurable IP** dialog box, navigate to the /src directory.
   
   c. Select all XCI source files, and click **OK**.
   
   d. Verify that the files are added, and **Copy Sources into Project** is checked. Click **Next**.

7. In the **Add Constraints (optional)** dialog box, the provided XDC file, `sinegen_demo_kc705.xdc` should automatically appear in the main window. 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** screen. Verify that the data appears as expected, per the steps above, click **Finish**.

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

10. In the **Sources** window in Vivado, expand **sinegen_domo_inst** to see the source files for this lab. Note that **ila_v2_1_0** core has been added to the project. Double-click the `sinegen_demo_inst.vhd` file to open it and inspect the instantiation and port mapping of the ILA core in the HDL code. Note that attributes have been placed in the source file to preserve the net names.

```
-- Attributes for keeping the net names preserved
attribute keep : string;
attribute keep of GPIO_BUTTONS_db : signal is "true";
attribute keep of GPIO_BUTTONS_dly : signal is "true";
attribute keep of GPIO_BUTTONS_re : signal is "true";
attribute keep of sineSel : signal is "true";
attribute keep of sine : signal is "true";
```

**Figure 10:** Using “keep” attribute to preserve net names

```
-- ila_v2_1_0
-------------------------------------
U_ILA : ila_v2_1_0
        port map
          ( CLK => clk,
            PROBE0 => sineSel,
            PROBE1 => sine,
            PROBE2 => GPIO_BUTTONS_db,
            PROBE3 => GPIO_BUTTONS_re,
            PROBE4 => GPIO_BUTTONS_dly,
            PROBE5 => GPIO_BUTTONS
        );
```

**Figure 11:** Hook signals that need to be debugged in the ILA
Step 2: Synthesize Implement and Generate Bitstream

1. From the Program and Debug folder, in Flow Navigator, click on Generate Bitstream. This will synthesize, implement and generate a bitstream for the design.

2. The No Implementation Results Available dialog box appears. Click Yes.

3. After bitstream generation completes, the Bitstream Generation Completed dialog box appears. Open Implemented Design is selected by default. Click OK.


5. In the Report Timing Summary dialog box, click OK. Make sure that all timing constraints are met.

6. Proceed to Chapter 6 for debugging the design in hardware.
Lab 3: Using a VIO Core for Debugging a Design in Vivado®

Introduction

The Virtual Input/Output (VIO) core is a customizable core that can both monitor and drive internal FPGA signals in real time. The number and width of the input and output ports are customizable in size to interface with the FPGA design. Because the VIO core is synchronous to the design being monitored and/or driven, all design clock constraints that are applied to your design are also applied to the components inside the VIO core. Run time interaction with this core requires the use of the Vivado® logic analyzer feature. The following figure is a block diagram of the new VIO 2.0 core.

Figure 13: VIO Block Diagram

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

The following ports are created:

- One 4-bits PROBE_IN0 port to monitor the 2-bits Sine Wave selector outputs from the finite state machine (FSM) and other two bits to mimic the state of the other two LEDs
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 screen, click Create New Project to start the New Project wizard. Click Next.
3. In the Project Name screen, name the new project proj_hdl_vio and provide the project location (C:/Vivado_Debug). Ensure that Create Project Subdirectory is checked and click Next.
4. In the Project Type screen, specify the Type of Project to create as RTL Project and click Next.
5. In the Add Sources screen:
   a. Set Target Language to VHDL.
   b. Click the Add Files button.
   c. In the Add Source Files dialog box, navigate to the /src/Lab2 directory.
   d. Select all VHD source files, and click OK.
   e. Verify that the files are added, and Copy Sources into Project is checked. Click Next.
6. In the Add Existing IP (optional) dialog box:
   a. Click the Add Files button.
   b. In the Add Configurable IP dialog box, navigate to the /src/Lab2 directory.
   c. Select all XCI source files, and click OK.
   d. Verify that the files are added and Copy Sources into Project is checked. Click Next.
7. In the Add Constraints (optional) dialog box, the provided XDC file, sinegen_demo_kc705.xdc, should automatically appear in the main window. 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, and click Next.
9. Review the New Project Summary screen. Verify that the data appears as expected, per the steps above, click Finish.
Step 1: Creating a Project with the Vivado New Project Wizard

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

10. In the Sources window in Vivado, expand sinegen_domo_inst_vio to see the source files for this lab.

Note that ila_v2_1_0 core has been added to the project. However, the vio_0 (the VIO core) is missing. You will instantiate and configure this VIO core as follows:

11. From the Flow Navigator, click IP Catalog, expand Debug & Verification, then expand Debug, and double-click on VIO 2.0. The Customize IP dialog box opens.

12. On the General Options tab, set Input Probe Count to 1, Output Probe Count to 1, and check the Enable Input Probe Activity Detectors checkbox.

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

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

15. Click OK to generate the IP. This should take less than a minute. At this point, you have finished customizing the VIO 2.0. This core has already been instantiated in the top level design as shown below:

![VIO Instantiation in the top level design](image)

16. Double-click the sinegen_demo_inst.vhd file to open it and inspect the instantiation and port mapping of the ILA core in the HDL code. Also, note that attributes have been placed in the source file to preserve the net names.

![Using “keep” attribute to preserve net names](image)
Step 2: Synthesize, Implement and Generate Bitstream

1. From the Program and Debug folder, in Flow Navigator, click on Generate Bitstream. This will synthesize, implement and generate a bitstream for the design.

2. The No Implementation Results Available dialog box pops up. Click Yes.

3. After bitstream generation completes, the Bitstream Generation Completed dialog box pops up. Open Implemented Design is selected by default. Click OK.


5. In the Report Timing Summary dialog box, click OK. Make sure that all timing constraints have been met.

6. Proceed to Chapter 6 (Verifying the VIO Core Activity section) for completing the rest of lab.
Lab 4: Using the Synplify Pro Synthesis Tool and Vivado® for Debugging a Design

Introduction

This simple tutorial shows how to do the following:

- Create a Synplify Pro project for the wave generator design.
- Mark nets for debug in the Synplify Pro constraints file as well as VHDL source files.
- Synthesize the Synplify Pro project to create an EDIF netlist.
- Create a Vivado® project based on the Synplify Pro netlist.
- Use the Vivado IDE to setup and debug the design from the synthesized design using Synplify Pro (Version 2013-3 SP1).

Step 1: Create a Synplify Pro Project

1. Launch Synplify Pro and select File > New. Set File Type to Project File (Project) as highlighted in Figure 18. In the New File Name box, enter synplify_1. Click OK.
Step 1: Create a Synplify Pro Project

2. In the left panel of the Synplify Pro window, click **Add File** as shown in Figure 19.

![Synplify Pro New Project Dialog Box](image)

**Figure 18: Synplify Pro New Project Dialog Box**

![Adding Files to a Synplify Pro Project](image)

**Figure 19: Adding Files to a Synplify Pro Project**
3. 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

Click **Add**.

**Note:** Do not select the **sinegen.vhd** file.

![Figure 20: Adding VHDL Source Files to the Synplify Pro Project](image-url)
4. In the same dialog box set **Files of type** to **Constraints File**. This shows the `synplify_1.sdc` file. Select the file and click **Add** as shown in Figure 21.

![Figure 21: Adding SDC Constraints File to the Synplify Pro Project](image-url)
5. In the same dialog box set **Files of type** to **Compiler Directives File**. This shows the `synplify_1.cdc` file. Select the file and click **Add** as shown in Figure 22. Click **OK**.

Figure 22: Adding CDC Constraints File to the Synplify Pro Project
6. Now, you need to set the implementation options. Click **Implementation Options** in the Synplify Pro window as shown in Figure 23.

![Synplify Pro window with Implementation Options highlighted](image)

**Figure 23: Opening Implementation Options in Synplify Pro**

7. This brings up the **Implementation Options** dialog box as shown in Figure 24. 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**.
Mark Nets for Debug in the Constraints Files

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

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

Figure 25: Specifying attributes to preserve net names in Synplify

9. You also can specify the `mark_debug` attributes in the source HDL files to mark the signals for debug, as shown in the snippet code from `sinegen_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";
```

Figure 26: Add `mark_debug` attribute in HDL file

10. The `synplify_1.sdc` file contains various kinds of constraints such as pin location, I/O standard, and clock definition. The `synplify_1.cdc` file contains directives for the
Step 2: Synthesize the Synplify Project

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 Figure 27.

![Attributes that are needed to mark_debug the nets that are needed to be viewed in ILM](image)

```plaintext
define_attribute -comment {Mark sinegen as black box} {v:work.sinegen} {syn_black_box} {} ;
define_attribute -comment {Set no_prune on sinegen} {v:work.sinegen} {syn_no_prune} {} ;
define_attribute -comment {Mark entire bus for debug} {i:sinegen.sin[20:0]} {mark_debug} {"true"};
define_attribute -comment {Mark entire bus for debug} {i:sinegen.sel[1:0]} {mark_debug} {"true"};
```

Figure 27: Synplify Pro Constraints in CDC Files

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

---

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

![Synthesize the design in Synplify](image)
Step 3: Create EDIF Netlists for the Black Box Created in Synplify Pro

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**. The mark_debug attributes can be seen in this netlist.

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

---

Step 3: Create EDIF Netlists 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 New Project**. This opens up the **New Project** wizard. Click **Next**.

3. Under **Project Name**, set the project name to **proj_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 checked. Click **Next**.

6. Under **Add Existing IP**, 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 **Add Constraints**, the `.sdc` files are automatically added to the project. These files are not needed for this step. Remove them from this project by clicking the **Remove Selected File** button on the right of the dialog box. Click **Next**.

8. Under **Default Part**, select **Boards** and then select the **Kintex-7 KC705 Evaluation Platform**. Click **Next**.

9. Under **New Project Summary**, ensure that all the settings are correct and click **Finish**.

10. Once the project has been created, in Vivado Flow Navigator, under the Project Manager folder, click **Project Settings**. In the pop-up dialog box, in the left panel, click **Synthesis**. From the pull down menu on the right panel, set –**flatten_hierarchy** to **none**. Click **OK**.


12. When synthesis completes the **Synthesis Completed** dialog box appears. Select **Open Synthesized Design** and click **OK**.
13. Now you need to write the netlist file for all the components used in the singen block. The four netlist files used in this tutorial are already provided as a part of the source files. However, you can overwrite them by using your own netlist files. To do this use the following Tcl command in the Tcl console of Vivado IDE:

```
write_edif -force ../Vivado_Debug/src/Lab4/sinegen.edn
```

Ensure that the path specified to the `src` folder is correct. At this point, you should see four `.edn` files in the `Vivado_Debug/src` folder as shown below:

- `dds_compiler_v6_0_xst.edn`
- `dds_compiler_v6_0_xst__parameterized1.edn`
- `dds_compiler_v6_0_xst__parameterized3.edn`
- `sinegen.edn`

14. 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 New 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/synplify_pro/rev_1` folder, and select `sinegen_demo.edf`.
6. Add the four netlist files created in the previous section. Click **Add Files** again, navigate to the `Vivado_Debug/src/Lab4` folder and select the following files:
   - `sinegen.edn`
   - `dds_compiler_v6_0_xst.edn`
   - `dds_compiler_v6_0_xst__parameterized1.edn`
   - `dds_compiler_v6_0_xst__parameterized3.edn`

Ensure that **Copy Sources into Project** is checked. Click **Next**.
7. Under **Add Constraints**, a `.sdc` file should be automatically populated. Remove this file by selecting them and clicking the **Remove Selected File** button on the right of the dialog box. Click **Add Files** and 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. Ensure that **Copy Constraints into Project** is checked. Click **Next**.
8. Under **Default Part**, select **Boards** and then select the **Kintex-7 KC705 Evaluation Platform**. Click **Next**.
9. Under **New Project Summary**, ensure that all the settings are correct and click **Finish**.
10. In the Sources window, select `sinegen_demo.edf` and select **Specify Top Module**.
Step 4: Create a Post Synthesis Project in Vivado IDE

11. In the **Specify Top Module** dialog box, click on the browse button.

![Select Top Module](image)

**Figure 30: Browse to the top module**

12. In the **Select Top Module** dialog box, select **sinegen_demo** and then click **OK**.

![Specify Top Module](image)
Step 5: Add (more) Debug Nets to the Project

13. Click OK in the Specify Top Module dialog box after ensuring that the top level module is correct.

14. In the Tcl console of Vivado type `set_property source_mgmt_mode none [current_project]`. This would make sinegen_demo the top level 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)
   - sineSel (2)

   After selecting all the nets mentioned above click Mark Debug.
Step 5: Add (more) Debug Nets to the Project

3. In the **Confirm Debug Net(s)** dialog box, click **OK**.

4. You should be able to see all the nets that are marked for debug as shown in Figure 34.

Running the Set up Debug Wizard

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

6. Click through the wizard to create Vivado logic analyzer debug cores, keeping the default settings.
Step 6: Implementing the Design and Generating the Bitstream

1. In the Flow Navigator, under the Implementation folder, click the Run Implementation button.
2. In the Save Project dialog box, click Save.
3. When the implementation process ends, an Implementation Completed dialog box opens. In the Implementation Completed dialog box, select Generate Bitstream and click OK.
4. In the Bitstream Generation Completed dialog box, Open Implemented Design is selected by default. Click OK.
5. Proceed to Chapter 6 to complete the rest of this lab.
Using the Vivado® Logic Analyzer to Debug the Hardware

Introduction

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

In this step, you learn:

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

Step 1: Verifying Operation of the Sine Wave Generator

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

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

Setting Up

If you plan to connect remotely, you will need to make sure you have KC705 hardware plugged into a machine and you are running a cse_server application on that machine. If you plan to connect locally, skip steps 1-4 below.

1. Connect the Digilent USB JTAG cable of your KC705 board to a Windows machine's USB port.
2. Ensure that the board is plugged in and powered on.
3. Assuming you are connecting your KC705 board to a 64-bit Windows machine and you will be running the cse_server from the network instead of your local drive, open a \texttt{cmd} prompt and type the following:

\texttt{<Xilinx\_Install>\ISE\_DS\ISE\bin\nt64\cse\_server}

Leave this \texttt{cmd} prompt open while the cse_server is running. Note the machine name that you are using, this will be used later when opening a connection to this instance of the cse_server application.

If you plan to connect locally, ensure that you have your KC705 hardware plugged into a Windows machine and then perform the following steps:

1. Connect the Digilent USB JTAG cable of your KC705 board to a Windows machine’s USB port.
2. Ensure that the board is plugged in and powered on.
3. Turn DIP switch positions on SW8 (Debounce Enable) to the OFF position.

**Using the Vivado Integrated Logic Analyzer**

1. In the Flow Navigator, from the Program and Debug drop-down list, select Open Hardware Session.

   \textbf{Note:} The window layout changed to look like Figure 36.

![Figure 36: Window Layout for Program and Debug](image)

2. Click on the Open a new hardware target link in the Hardware Session view.
3. The Open New Hardware Target dialog box opens. Click Next.
4. Type the name of the server (for e.g. localhost:60001) in the text field and click Next.
Figure 37: CSE Server Name

**Note:** Depending on your connection speed, this may take about 10~15 seconds
5. If there is more than one target connected to the cse_server you will see multiple entries in the **Select Hardware Target** dialog box. In this tutorial, there is only one target as shown in Figure 38. Click **Next**

---

Figure 38: Select Hardware Target
6. Leave these settings at their default values as shown in Figure 39. Click **Next**.

![Figure 39: CSE Hardware Target Parameter Settings](image)
7. In the **Open Hardware Target Summary** page, click **Finish** as shown in Figure 40.

![Figure 40: Open Hardware Summary](image)

8. Wait for the connection to the hardware to complete. The dialog in Figure 41 appears while hardware is connecting.

![Figure 41: Open Hardware Target](image)

Once the connection to the hardware target is made, the dialog shown in Figure 42 appears.

**Note:** The **Hardware** tab in the **Debug** view shows the hardware target and XC7K325T device that was detected in the JTAG chain.
9. Next, program the XC7K325T device using the .bit bitstream file that was created previously by right-clicking on the XC7K325T device and selecting Program Device as shown in Figure 43.

10. In the Program Device dialog box verify that the .bit file is correct for the lab that you are working on and click the Program button to program the device as shown in Figure 44.
Step 1: Verifying Operation of the Sine Wave Generator

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

![Hardware Detection](image)

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

12. Next, select the Debug Probes window as shown in Figure 46 and ensure that all of the debug nets added from previous steps are accounted for. This window may be slightly different for different labs.

![Debug Probes](image)

**Verifying Sine Wave Activity**

13. Click the Run Trigger Immediate button to trigger and capture data immediately as shown in Figure 47.
14. In the **Waveform** window, verify that there is activity on the 20-bit sine signal as shown in Figure 48.

15. Right-click **U_SINEGEN/sine[19:0]** signals, and select **Waveform Style > Analog** as shown in Figure 49.

   **Note:** Notice that 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.

16. Right-click on **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 Figure 50 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.

17. Cycle the sine wave sequential circuit by pressing the GPIO_SW_E push button as shown in Figure 51.

18. Click **Run Trigger Immediately** again to see the new sine selected sine wave. You should see the mid frequency as shown in Figure 52. Notice that the sel signal also changed from 0 to 1 as expected.

19. Repeat step 1 and 2 to view other sine wave outputs.
Step 2: Debugging the Sine Wave Sequencer State Machine (Optional)

As you were correcting 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, you will 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.

**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 Figure 52, Figure 53, and Figure 54.
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. Figure 55 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 (shown in Figure 1). The output signal Y connects to the top-level signal, sineSel, which selects the sine wave.

![Figure 55: Sine Wave Sequencer Button Schematic](image)

Viewing the State Machine Glitch

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

As described earlier, the glitch reveals itself as multiple low-to-high transitions on the GPIO_BUTTONS_1_IBUF signal, but it occurs intermittently. Because it could take several button presses to detect it, you will now set up the Vivado logic analyzer tool to Repetitive Trigger Run Mode. This setting makes it easier to repeat the button presses and look for the event in the Waveform viewer.

1. Set the trigger position of the ILA core to 512 (at the midway point of the 1024 sample-deep captured window shown in Figure 56).
2. Set the Compare Value for GPIO_BUTTONS_IBUF bit 1, to R. Depending on the lab this view may be different. All you have to do is ensure that the most significant bit, i.e. bit 1, should be set to R (rising edge).
3. Source the `rt.tcl` file in the Tcl command prompt. This Tcl command performs the following tasks:
   - Arms the trigger.
   - Waits for the trigger.
   - Uploads and displays waveforms.

4. On the KC705 board, press the Sine Wave Sequencer button until you see multiple transitions on the `GPIO_BUTTONS_1_IBUF` signal (this could take 10 or more tries). This is a visualization of the glitch that is occurring on the input. An example of the glitch is shown in Figure 58 and Figure 59.

   **Note:** You might not observe signal glitches 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 “debouncer” circuit is required.

5. Enable the debouncer circuit by setting DIP switch position on the KC705 board (labeled Debounce Enable in Figure 1) to the ON or UP position.

6. Repeat step 2, page 39, and step 3, page 39, to:
   - 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 on Open Hardware Session.
2. The Open New Hardware Target dialog box opens. Click **Next**.

3. In the Server Name field type in the name and port of the remote server that you want to connect to.

4. Ensure that you are connected to the right target by selecting the target from the Hardware Targets pane. If there is only one target then that will be selected by default. Click Next.

5. On the Set Hardware Target Properties page, click **Next**.

6. On the Open Hardware Target Summary Page, verify that all the information is correct and click **Finish**.

7. Program the device by selecting and right-clicking the device in the Sources window and then selecting **Program Device**.

8. In the Program Device dialog box, ensure that the bit file to be programmed is correct and click **Program**.

9. Once the FPGA has been programmed, you will see the VIO and the ILA core in the Hardware window.

10. Click on Run Trigger Immediate to capture the data immediately.
Figure 62: Run Trigger Immediate


12. Select the sine signal in the Waveform window, right-click and select **Waveform Style > Analog**.

13. 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.

14. Instead of using the GPIO_SW_3 push button to cycle through each different sine wave output frequency, we are going to use the virtual “push_button_vio” toggle switch from the VIO core.

15. In the Debug Probes window, select VIO Cores tab.

Figure 63: Select VIO Cores Tab

16. Set the “push_button_reset” output probe by right-clicking on **push_button_reset** and select **Toggle Button** type. 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.
Verifying the VIO Core Activity (Only applicable to Lab 3)

17. Follow the step above to change the push_button_vio to Toggle button as well.
18. Set “sineSel” input probes by right-clicking on sineSel and selecting LED.

19. In the LED dialog box, pick the Low Value Color and the High Value Color of the LEDs as you desire.

20. When finished, your Debug Probes window in hardware should look similar to Figure 67.
21. To cycle through each different sine wave output frequency using the virtual “push_button_vio” from the VIO core, follow 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...

   b. Select hw_ila_1 in the Hardware window and click Run Trigger to capture and display the selected sine wave signal from the previous step.
Appendix A

Additional Resources

**Xilinx Resources**

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm

**Solution Centers**

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

See the Integrated Logic Analyzer (ILA) 2.0 product page for details on the latest information on LogiCORE IP ChipScope Pro Integrated Logic Analyzer (ILA) (v2):


**References**

These documents provide supplemental material useful with this guide:

Vivado™ Design Suite 2013.1 Documentation (http://www.xilinx.com/cgi-bin/docs/rdoc?v=2013.1;t=vivado+docs)

ISE Design Suite 14.5 Documentation (http://www.xilinx.com/cgi-bin/docs/rdoc?v=14.5;t=ise+docs)