Vivado Design Suite
Tutorial:

Logic Simulation

UG937 (v 2013.1) March 20, 2013

XILINX®
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; 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.

©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

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>03/20/2013</td>
<td>2013.1</td>
<td>Validated with Release.</td>
</tr>
</tbody>
</table>
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revision History</td>
<td>2</td>
</tr>
<tr>
<td>Vivado Simulator Overview</td>
<td>5</td>
</tr>
<tr>
<td>Introduction</td>
<td>5</td>
</tr>
<tr>
<td>Tutorial Description</td>
<td>5</td>
</tr>
<tr>
<td>Software and Hardware Requirements</td>
<td>7</td>
</tr>
<tr>
<td>Locating Tutorial Design Files</td>
<td>7</td>
</tr>
<tr>
<td>Lab 1:</td>
<td>8</td>
</tr>
<tr>
<td>Running the Simulator in Vivado IDE</td>
<td>8</td>
</tr>
<tr>
<td>Step 1: Creating a New Project</td>
<td>8</td>
</tr>
<tr>
<td>Step 2:</td>
<td>13</td>
</tr>
<tr>
<td>Adding IP from the IP Catalog</td>
<td>13</td>
</tr>
<tr>
<td>Step 3:</td>
<td>17</td>
</tr>
<tr>
<td>Generating Simulation Outputs for IP</td>
<td>17</td>
</tr>
<tr>
<td>Step 4: Running Behavioral Simulation</td>
<td>19</td>
</tr>
<tr>
<td>Conclusion</td>
<td>21</td>
</tr>
<tr>
<td>Lab 2:</td>
<td>22</td>
</tr>
<tr>
<td>Debugging the Design</td>
<td>22</td>
</tr>
<tr>
<td>Step 1: Opening the Project</td>
<td>22</td>
</tr>
<tr>
<td>Step 2: Displaying Signal Waveforms</td>
<td>23</td>
</tr>
<tr>
<td>Step 3: Using the Analog Wave Viewer</td>
<td>24</td>
</tr>
<tr>
<td>Step 4: Working with the Waveform Window</td>
<td>26</td>
</tr>
<tr>
<td>Step 5: Changing Signal Properties</td>
<td>30</td>
</tr>
<tr>
<td>Step 6: Saving the Waveform Configuration</td>
<td>32</td>
</tr>
<tr>
<td>Step 7: Re-Simulating the Design</td>
<td>34</td>
</tr>
<tr>
<td>Step 8: Using Cursors, Markers, and Measuring Time</td>
<td>35</td>
</tr>
<tr>
<td>Step 9: Debugging with Breakpoints</td>
<td>38</td>
</tr>
<tr>
<td>Step 10: Re-launch Simulation</td>
<td>43</td>
</tr>
<tr>
<td>Summary</td>
<td>44</td>
</tr>
</tbody>
</table>
Vivado Simulator Overview

Introduction

This Xilinx® Vivado™ Design Suite tutorial provides designers with an in-depth introduction to the Vivado simulator.

The Vivado simulator is a Hardware Description Language (HDL) simulator that lets you perform behavioral, functional, and timing simulations for VHDL, Verilog, and mixed-language designs. The Vivado simulator environment includes the following key elements:

- **xvhdl** and **xvlog**: Parsers for VHDL and Verilog files, respectively, that store the parsed files into an HDL library on disk.
- **xelab**: HDL elaborator and linker command. For a given top-level unit, xelab loads up all sub-design units, performs static elaboration, and links the generated executable code with the simulation kernel to create an executable simulation snapshot.
- **xsim**: Vivado simulation command that loads a simulation snapshot to effect a batch mode simulation, a GUI, or Tcl-based interactive simulation environment.
- **Vivado Integrated Design Environment (IDE)**: An interactive design-editing environment that provides the simulator user-interface and common waveform viewer.

Tutorial Description

This tutorial provides a design flow in which you can use Vivado simulator for performing behavioral, functional, or timing simulation from the Vivado Integrated Design Environment (IDE).

**IMPORTANT**: The tutorial files are configured to run Vivado simulator in a Windows environment. To run elements of this tutorial under the Linux operating system, some file modifications may be needed.

You will run the Vivado simulator in both Project Mode, using a design project to manage design sources and the design flow, and in Non-Project mode, managing the design more directly. For more information about Project Mode and Non-Project Mode, refer to the *Vivado Design Suite User Guide: Design Flows Overview* (UG892).
shows a block diagram of the tutorial design.

The tutorial design consists of the following blocks:

- A Sine wave generator that generates high, medium, and low frequency sine waves; plus an amplitude sine wave (sinegen.vhd).
- DDS compilers that generate low, middle, and high frequency waves: (sine_low.vhd, sine_mid.vhd, and sine_high.vhd).
- A Finite State Machine (FSM) to select one of the four sine waves (fsm.vhd).
- A Debouncer that enables switch-selection between the raw and the debounced version of Sine wave selector (debounce.vhd).
- A design top module that resets FSM and the Sine wave generator, and then MUXes the sine select results to the LED output (sinegen_demo.vhd).
• A simple testbench (testbench.v), to simulate the sine wave generator design that:
  • Generates a 200 MHz input clock for the design system clock, sys_clk_p.
  • Generates GPIO button selections.
  • Controls raw and debounced sine wave select.

Note: For more information about testbenches, see Writing Efficient Testbenches (XAPP199).

Software and Hardware Requirements

This tutorial requires the 2013.1 Vivado Design Suite or later. For installation instructions, see Vivado Design Suite User Guide: Release Notes, Installation, and Licensing (UG973).

Vivado Design Suite supported Operating Systems include Redhat 5.6 Linux 64 and 32 bit, and Windows 7, 64 and 32 bit. Xilinx recommends a minimum of 2 GB of RAM when using the Vivado tool.

Locating Tutorial Design Files

1. Download the ug937.zip file from the Xilinx website:
   -or-
   https://secure.xilinx.com/webreg/clickthrough.do?cid=198571
2. Extract the zip file contents into any write-accessible location.

   This tutorial refers to the extracted ug937.zip file contents as <Extract_Dir>.

   Note: You will modify the tutorial design data while working through this tutorial. You should use a new copy of the original data each time you start this tutorial.

   The following table describes the contents of the ug937.zip file.

<table>
<thead>
<tr>
<th>Directories/Files</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/sources</td>
<td>Contains the HDL files necessary for the functional simulation</td>
</tr>
<tr>
<td>/sim</td>
<td>Contains the testbench.v file</td>
</tr>
<tr>
<td>/completed</td>
<td>Contains the completed files, and a Vivado 2012.3 project of the tutorial design, for reference</td>
</tr>
<tr>
<td>/scripts</td>
<td>Contains the scripts that you run during the tutorial</td>
</tr>
<tr>
<td>readme.txt</td>
<td>The readme.txt is a readme file about the tutorial design.</td>
</tr>
</tbody>
</table>
Lab 1: Running the Simulator in Vivado IDE

Step 1: Creating a New Project

The Vivado™ Integrated Design Environment (IDE) lets you launch simulation from within design projects, automatically generating the necessary simulation commands and files.

On Windows, launch the Vivado IDE:

- **Start** > **All Programs** > **Xilinx Design Tools** > **Vivado 2013.1** > **Vivado 2013.1**

  **Note:** As an alternative, click the **Vivado 2013.1** Desktop icon to start the Vivado IDE.

---

1 Your Vivado Design Suite installation may called something different than **Xilinx Design Tools** on the **Start** menu.
Create a new project for managing source files, add IP to the design, and run behavioral simulation.

1. In the Vivado IDE Getting started page, click **Create New Project**.
2. In the **New project** dialog box, click **Next**, and enter a project name (for example, `project_xsim`).

![Create Project](image)

**Figure 3: Create Project**

3. For **Project Location**: browse to the folder containing the extracted tutorial data, `<Extract_Dir>`, and click **Next**.
4. In the **Project Type** dialog box, select **RTL Project**, and click **Next**.
5. In the Add Source dialog box, click **Add Directories** and add the extracted tutorial design data:
   - `<Extract_Dir>/source`
   - `<Extract_Dir>/sim`.
6. Set the Target Language to Verilog.
7. Click Next.
8. Click **Next** twice to bypass the Add Existing IP and Add Constraints dialog boxes.

9. In the Default Part dialog box, specify **Boards**, and select **Kintex-7 KC705 Evaluation Platform**, and click **Next**.
10. **Review** the **Project Summary** dialog box.
11. Click **Finish** to create the project.

Vivado opens the new project in the Vivado IDE, using the default view layout.

![Figure 6: Vivado IDE – Default Layout](image)
Step 2: Adding IP from the IP Catalog

The Sources window displays the source files that were added to the project when it was created. The Hierarchy tab displays the hierarchical view of the source files.

![Sources window](image)

Figure 7: Sources window

Notice, that the Sine wave generator (`singen.vhd`) references cells that are not found in the current design sources. In the Sources window, the missing design sources are marked by the missing source icon.

You will now add the `sine_high`, `sine_mid`, and `sine_low` modules to the project from the Xilinx IP Catalog.

Adding Sine High

1. In the Flow Navigator, select the **IP Catalog** button.
   - The IP Catalog opens in the graphical windows area. For more information on the specifics of the Vivado IDE, refer to the *Vivado Design Suite User Guide: Using the Vivado IDE* ([UG893](#)).
2. In the **search field** of the IP Catalog, type **DDS**.
   - The Vivado IDE highlights the DDS Compilers in the IP catalog.
3. Under any category, double-click the **DDS Compiler AXI4-Stream IP**.
   - The Customize IP wizard opens.
4. In the IP Symbol on the left, ensure that **Show Disabled Ports** is unchecked.

5. Specify the following on the **Configuration** tab **Error! No sequence specified.**:
   - Component Name: **sine_high**
   - Configuration Options: **SIN COS LUT only**
   - Noise Shaping: **None**
   - Under Hardware Parameters, set **Phase Width: 16**, and **Output Width: 20**

6. On the **Implementation** tab, set **Output Selection: Sine**

7. On the **Detailed Implementation** tab, set **Control Signals: ARESETn (active-Low)**

8. Select the **Summary** tab, **review** the settings and click **OK**.
Adding Sine Mid

1. In the IP catalog, double-click the DDS Compiler AXI4-Stream IP a second time.

2. Specify the following on the Configuration tab:
   - Component Name: sine_mid
   - Configuration Options: SIN COS LUT only
   - Noise Shaping: None
   - Under Hardware Parameters, set Phase Width: 8, and Output Width: 18

3. On the Implementation tab, set Output Selection: Sine

4. On the Detailed Implementation tab, set Control Signals: ARESETn (active-Low)

5. Select the Summary tab, review the settings and click OK.

Figure 10: Sine Mid Summary
Adding Sine Low

1. In the IP catalog, double-click the DDS Compiler AXI4-Stream IP, for the third time.
2. Specify the following on the Configuration tab:
   - Component Name: sine_low
   - Configuration Options: SIN COS LUT only
   - Noise Shaping: None
   - Under Hardware Parameters, set Phase Width: 6, and Output Width: 16
3. On the Implementation tab, set Output Selection: Sine
4. On the Detailed Implementation tab, set Control Signals: ARESETn (active-Low)
5. Select the Summary tab, review the settings as seen in Figure 11, and click OK.

![Component Parameters Configuration](image)

**Figure 11: Sine Low Summary**

For more information on working with IP cores and the Xilinx IP Catalog, refer to the Vivado Design Suite User Guide: Design with IP (UG896). You can also work through the Vivado Design Suite Tutorial: Designing with IP (UG939).
Step 3: Generating Simulation Outputs for IP

With three instances of the DDS Compiler IP added to your project as `sine_high`, `sine_mid`, and `sine_low`, you can generate the output products required to synthesize and simulate the design.

1. In the Sources window, select `sine_high`, `sine_mid`, and `sine_low`.
   Right-click to open the popup menu, and select **Generate Output Products**.

   ![Generate Output Products](image)

   **Figure 12: Generate Output Products**

2. In the Manage Outputs dialog box, shown in **Figure 13**, set the **Actions** as follows:
   - **Do Not Generate**: C Simulation, Testbench.
   - **Do Nothing**: Instantiation Template, Synthesis
   - **Generate**: Simulation

3. Click **OK**.
4. In the **Sources** window, go to the **Libraries** tab.

5. In the **Design Sources** folder, expand the **IP** folder. 
   Observe the library structure for the IPs.

**Figure 13: Manage Output Products dialog box**

**Figure 14: IP Library Hierarchy**
Step 4: Running Behavioral Simulation

After you have created a Vivado project for the tutorial design, you set up and launch Vivado simulator to run behavioral simulation. Set the behavioral simulation properties in Vivado:

1. In the Flow Navigator, click Simulation Settings. The following defaults are automatically set:
   - Simulation set: sim_1
   - Simulation top-module name: testbench

2. In the Compilation tab, ensure that the debug level is set to typical, which is the default value.

3. In the Simulation tab, observe that the Simulation Run Time is 1000ns.

4. Click OK.

   With the simulation settings properly configured, you can launch Vivado simulator to perform a behavioral simulation of the design.
5. In the Flow Navigator, click **Run Simulation > Run Behavioral Simulation**

   Functional and timing simulations are available post-synthesis and post-implementation. Those simulations are outside the scope of this tutorial.

   When you launch the Run Behavioral Simulation command, the Vivado tool runs `xelab` in the background to elaborate and compile the design into a simulation snapshot, which the Vivado simulator can run. When that process is complete, the Vivado tool launches `xsim` to run the simulation.

   In the Vivado IDE, the simulator GUI opens after successfully parsing and compiling the design. By default, the top-level HDL objects display in the waveform window.

![Vivado Simulation GUI](image)

*Figure 16: Vivado Simulation GUI*
Conclusion

In this lab, you have created a new Vivado Design Suite project, added HDL design sources, added IP from the Xilinx IP catalog and generated IP outputs needed for simulation, and then run behavioral simulation on the elaborated RTL design.

This concludes Lab #1. You can continue Lab #2 by starting at Step 2: Debugging the Design.

You can close the simulation, project, and the Vivado IDE, to start Lab #2 at a later time.

1. Click File > Close Simulation to close the open simulation.
   Select OK if prompted to confirm closing the simulation.
2. Click File > Close Project to close the open project.
3. Click File > Exit to exit the Vivado tool.
Lab 2: Debugging the Design

The Vivado™ simulator GUI contains the Waveform window, and Object and Scope Windows. It provides a set of debugging capabilities to quickly examine, debug, and fix design problems. See the Vivado Design Suite User Guide: Logic Simulation (UG900) for more information about the GUI components.

In this lab, you will:

- Enable debug capabilities
- Examine a design bug
- Use different debug features to find the root cause of the bug
- Make changes to the code
- Re-compile and re-launch the simulation

---

**Step 1: Opening the Project**

This lab continues from the end of Lab #1 in this tutorial. You must complete Lab #1 prior to beginning Lab #2. If you closed the Vivado IDE, or the tutorial project, or the simulation at the end of Lab #1, you will need to open them again.

Start by loading the Vivado Integrated Design Environment (IDE).

Start > All Programs > Xilinx Design Tools > Vivado 2013.1 > Vivado 2013.1

**Note:** As an alternative, click the Vivado 2013.1 Desktop icon to start the Vivado IDE.

The Vivado IDE opens. Now, open the project from Lab #1, and run behavioral simulation.

1. From the main menu, click **File > Open Recent Project** and select the project you saved in Lab #1.
2. After the project has opened, from the Flow Navigator click **Run Simulation > Run Behavioral Simulation**.
   - The Vivado simulator compiles your design and loads the simulation snapshot.

---

2 Your Vivado Design Suite installation may called something different than Xilinx Design Tools on the Start menu.
Step 2: Displaying Signal Waveforms

In this section, you examine features of the Vivado simulator GUI that help you monitor signals and analyze simulation results, including:

- Running and restarting the simulation to review the design functionality, using signals in the Wave window and messages from the testbench shown in the Tcl console
- Adding signals from the testbench and other design units to the wave configuration window so you can monitor their status
- Adding groups and dividers to better identify signals in the waveform configuration window
- Changing signal and wave properties to better interpret and review the signals in the wave configuration window
- Using markers and cursors to highlight key events in the simulation and to perform zoom and time measurement features
- Using multiple waveform configurations

Add and Monitor Signals

The focus of the tutorial design is to generate sine waves with different frequencies. To observe the function of the circuit, you will monitor a few signals from the design. Before running simulation for a specified time, you can add signals to the wave window to observe the signal as it transitions to different states over the course of the simulation.

By default, the Vivado simulator adds simulation objects from the test bench to the waveform window. In the case of this tutorial, the following test bench signals load automatically:

- Differential clock signals: sys_clk_p and sys_clk_n: This is a 200 MHz clock generated by the test bench, and is the input clock for the complete design.
- GPIO buttons (gpio_buttons[1:0]): Provides control signals to select different frequency sine waves.
- GPIO switch (gpio_switch): Provides a control switch to enable or disable debouncer logic.
- LEDs (leds_n[3:0]): A place holder bus to display the results of the simulation.

You will add some new signals to this list to monitor those signals as well.

1. In the Scopes window, click the ‘+’ sign to expand the testbench.
   
   An HDL scope, or scope, is defined by a declarative region in the HDL code such as a module, function, task, process, or begin-end blocks in Verilog. VHDL scopes include entity/architecture definitions, block, function, procedure, and process blocks.

2. In the Scopes window, click to select the DUT object.
Step 3: Using the Analog Wave Viewer

The current scope of the simulation changes from the whole testbench to the selected object. The Objects window updates with all the **signals and constants** of the selected Scope.

3. From the Objects window, **select** signals `sineSel[1:0]` and `sine[19:0]` and add them into Wave Configuration window using one of the following methods:
   - **Drag and drop** the selected signals into the Waveform window.
   - Right-click on the signal to open the **popup menu**, and select **Add to Wave Window**.

   **Note:** You can select multiple signals by holding down the CTRL key during selection.

![Figure 17: Add signals to Wave Window](image)

---

**Step 3: Using the Analog Wave Viewer**

The `sine[19:0]` signals you are monitoring are analog signals, which you can view better in Analog wave mode. You can choose to display a given signal as Digital or Analog in the Waveform window.

1. In the Waveform window, **select** the `sine[19:0]` signal.
2. Right click to open the popup menu, and select **Waveform Style > Analog**.
3. Right click to open the popup menu again, and set the **Radix** for the analog signal as **Signed Decimal**.
Step 3: Using the Analog Wave Viewer

Logging Waveforms for Debugging

The Waveform window lets you review the state of multiple signals as the simulation runs. However, due to its limited size, the number of signals you can monitor in the Waveform window is also limited. For debugging, you might need to trace many more signals and scopes than can be displayed in the Waveform window to identify design failures. You can also log the waveform for signals without having to add them into the Waveform window. After you run simulation, you can review the transitions on signals that you either displayed in the Waveform window, or captured in a wave log file.

1. **Enable logging** of the waveform for the specified HDL objects by entering the following command in the Tcl console:

   ```tcl
   log_wave [get_objects /testbench/dut/*] [get_objects /testbench/dut/u_sinegen/*]
   ```

   This command enables signal dumping for the specified HDL objects, /testbench/dut/* and /testbench/dut/u_sinegen/*.

---

3 There is no GUI equivalent for this Tcl command. Refer to the Vivado Design Suite Tcl Command Reference Guide ([UG835](https://www.xilinx.com)) for more information on the command.
Step 4: Working with the Waveform Window

Now that you have configured the simulator to display signals of interest in the Waveform window, or capture them in the wave log file, you are ready to run the simulator again.

1. Run the simulation by clicking the Run All button.

   Observe the sine signal output in the waveform. The Wave Window can be undocked from Main window layout to view it as standalone.

2. Click the undock icon in the left top corner of the waveform configuration window.

3. Display the full time spectrum in the Waveform window, by clicking the Zoom Fit button.

   Notice that the low frequency sine output is incorrect. You can view the waveform in detail by zooming into the Waveform window. When you are zoomed into the waveform, you can use the horizontal and vertical scroll bars to pan down the full waveform.

In the above wave view, when sineSel is 00, which indicates a low frequency sine selection, the sine output is not a proper sine wave, and is therefore not correct.
Grouping Signals

Now you will add signals from other design units to better analyze the functionality of the whole design. When you add signals to the Waveform window, the limited size of the window makes it difficult to display all signals at the same time. Reviewing all signals would require the use of the vertical scroll bar, making the review process difficult.

You can group related signals together to make viewing them easier. With a group, you can display or hide associated signals to make the Waveform window less cluttered, and easier to understand.

1. In the Waveform window, press and hold the Ctrl key, and select all signals in the testbench unit: sys_clk_p, sys_clk_n, gpio_buttons, gpio_switch, and leds_n.
2. With the signals selected, right-click to open the popup menu and select New Group. The Name dialog box opens to let you specify the name of the signal group to create.
3. Type TB Signals as the name of this signal group, and click OK.

![Figure 20: Name Signal Group](image)

Vivado simulator creates a collapsed group in the waveform configuration window. To expand the group, click the ‘+’ to the left of the group name.

4. Create another signal group called DUT Signals to group signals sine[19:0] and sine_sel[1:0].

You can add signals to a group, or remove signals from a group as needed. Cut and paste signals from the list of signals in the Waveform window, or drag and drop a signal from one group into another.

You can also drag and drop a signal from the Objects Window into the Waveform window, or into a group.

You can ungroup all signals, thereby eliminating the group. Select a group, right-click to open the popup menu, and select Ungroup.

To better visualize which signals belong to which design units, add dividers to separate the signals by design unit.
Adding Dividers

Dividers let you create visual breaks between signals, or groups of signals, to more easily identify related objects.

1. In the Waveform window, right-click to open the popup menu and select New Divider.

   The Name dialog box opens to let you name the divider you are adding to the Waveform window.

2. Add two dividers named:
   - Testbench
   - SineGen

3. Click and drag the Testbench divider to the top of the list.

4. Move SineGen to the bottom of the list.

   **TIP:** You can change divider names at any time by highlighting the divider name and selecting the Rename command from the popup menu, or change the color with Divider Color.

---

**Figure 21: Add Dividers**
Adding Signals from Sub-modules

You can also add signals from different levels of the design hierarchy to study the interactions between these modules and the testbench. The easiest way to add signals from a sub-module is to filter objects and then select the signals to add to the Waveform view.

You will add signals from the instantiated `sine_gen_demo` module (DUT), and the `sinegen` module (U_SINEGEN).

1. In the Scopes window, **select** and expand the **testbench**, then select the **DUT**.
   
   Simulation objects associated with the currently selected scope display in the Objects window.
   
   By default, all types of simulation objects display in the Objects window. However, you can limit the types of objects displayed by selecting the object filters at the top of the Objects window. **Figure 22** shows the Objects window with the Input and Output port objects enabled, and the other object types are disabled. Move the cursor to hover over a button to see the tooltip for the object type.

   ![Figure 22: Object Filters](image)

2. Use the Objects window toolbar to **enable** and **disable** the different **object types**.
   
   The types of objects that can be filtered in the Objects window include Input, Output, Inout ports, Internal Signals, Constants, and Variables.

3. In the Scopes window, **select** the **U_SINEGEN** design unit.

4. In the Waveform window, right-click beneath the **SineGen divider**, and use the **New Group** command to create three new groups called **Inputs**, **Outputs**, and **Internal Signals**.
   
   **Note:** If you create the group on top of, or containing, any of the current objects in the Waveform window, simply drag and drop the objects to separate them as needed.

5. In the Objects window, select the Input **filter to display** the **Input** objects.

6. **Select** the **Input objects** in the Objects window, and **drag and drop** them onto the Input group you created in the Waveform window.
Step 5: Changing Signal Properties

You can also change the properties of some of the signals shown in the Waveform window to better visualize the simulation results.

Viewing Hierarchical Signal Names

By default, the Vivado simulator adds signals to the waveform configuration using a short name with the hierarchy reference removed. For some signals, it is important to know to which module they belong.

1. In the Waveform window, hold Ctrl and click to select the `sine[19:0]` and `sineSel[1:0]` signals listed in the DUT group, under the SineGen divider.

2. Click to select the `sine[19:0]` signals listed in the Outputs group, under the SineGen divider.

3. Right-click in the Waveform window to open the popup menu, and select the Name > Long command.

Figure 23: Configuring the Wave Window
The displayed name changes to include the hierarchical path of the signal. You can now see by looking that the `sine[19:0]` signals under the DUT and Output groups refer to different objects in the design hierarchy.

**Figure 24: Long Signal Names**

### Viewing Signal Values

You can understand some signal values more clearly, if they display in a different radix format than the default, for instance, hexadecimal values instead of binary values. The default radix is binary, unless you override the radix for a specific object.

Supported radix values are default, dec, bin, oct, hex, unsigned, and ascii.

1. In the Waveform window select the following signals: `s_axis_phase_tdata_high`, `s_axis_phase_tdata_mid` and `s_axis_phase_tdata_low`.

2. Right-click to open the popup menu, and select **Radix > Hexadecimal**.

There are currently no values displayed for the selected signals. However, when you restart the simulation, the values display using the format specified by the radix.

In **Figure 24** you can see that the two highlighted `sine[19:0]` signals are displaying values as a signed decimal and in binary format.
Step 6: Saving the Waveform Configuration

You can customize the look and feel of the Waveform window, and then save the Waveform configuration to reuse in future simulation runs.

1. In the Waveform window, click the Options button on the sidebar menu. The Waveform Options dialog box opens to the General tab.
2. Change the Default Radix from the Binary default to Hexadecimal. This changes the number format used by all signals in the Waveform window, overriding the default for the Vivado simulator. As seen earlier, the radix can also be set for individual objects in the Waveform window.
3. Select the Draw Waveform Shadow to enable or disable the shading under the signal waveform. By default, a waveform is shaded under the high transitions to make it easier to recognize the transitions and states in the Waveform window. You can also enable or disable signal indices, so that each signal or group of signals is identified with an index number in the Waveform window.
4. Check or uncheck the Show Signal Indices checkbox to enable or disable the signal list numbering.

![Waveform Options](image)

Figure 25: Waveform Options – General View

5. In the Waveform Options dialog box, select the Colors view. Examine the Waveform Color Options dialog box. You can configure the coloring for elements of the Waveform window to customize the look and feel. You can specify custom colors to display waveforms of certain values, so you can quickly identify signals in an unknown state, or an uninitialized state.

With the Waveform window configured with your preferences, you can save the current waveform configuration so it is available for use in future Vivado simulation sessions.

By default, the Vivado simulator saves the current waveform configuration setting as testbench_behav.wcfg.
6. In the Waveform window sidebar menu, select the **Save Wave Configuration** button.

7. Save the Wave Configuration into the project folder with the filename `tutorial_1.wcfg`. When saving the waveform configuration file, you are prompted to add this file to your current project. If you click OK, the file is added to the project simulation files set, `sim_1`, for archive purposes.

![](image)

**Figure 26: Add Waveform Configuration**

You can load a previously saved waveform configuration file using the **File > Open Waveform Configuration** command.

**Working with Multiple Waveform Configurations**

You can also have multiple Waveform windows, and waveform configuration files open at one time. This is useful when the number of signals you want to display exceeds the ability to display them in a single window. Depending on the resolution of the screen, a single Waveform window might not display all the signals of interest at the same time. You can open multiple Waveform windows, each with their own set of signals and signal properties.

1. To add a new Waveform window, select **File > New Waveform Configuration**.

   An untitled Waveform window opens with a default name. You can add signals, define groups, add dividers, set properties and colors that are unique to this Waveform window.

   With multiple Waveform windows open, you can copy and paste between them.

2. Select signal groups in the first Waveform window by pressing and holding the **Ctrl** key, and selecting the following groups: **Inputs**, **Outputs**, and **Internal Signals**.

3. Right-click to open the popup menu, and select **Cut**, or use the shortcut **Ctrl-x** on the selected groups to cut them from the current Waveform window.

4. Select the new Waveform window to make it active.

5. Right-click on the selection, and select **Paste**, or use the shortcut **Ctrl-v** to paste the signal groups into the current Waveform window.

6. Select **File > Save Waveform Configuration**, or click the **Save Wave Configuration** button, and save the waveform configuration to a file called `tutorial_2.wcfg`. 
7. Close the new Waveform window.

Step 7: Re-Simulating the Design

With the various signals, signal groups, dividers, and attributes you have added to the Waveform window, you are now ready to simulate the design again.

1. Click the **Restart** button to reset the circuit to its initial state.

2. Click the **Run All** button.

   The simulation runs for about 705ns. If you do not Restart the simulator prior to Run All, the simulator will run continuously until interrupted.

3. After the simulation is complete, click the **Zoom Fit** button to see the whole simulation timeline in the Waveform window.

   Figure 27 shows the current simulation results.

---

**Figure 27: Simulation Waveform at Time 705ns**
Step 8: Using Cursors, Markers, and Measuring Time

The Finite State Machine (FSM) module used in the top-level of the design generates three different sine-wave select signals for specific outputs of the SineGen block. You can identify these different wave selections better using Markers to highlight them.

1. In the Name column of the Waveform window, select the `sineSel[1:0]` signal.
2. From the waveform toolbar, click the **Next Transition** button.
   The current marker moves to next value change.
3. Click the **Add Marker** button.
   **Note:** By default, the waveform window displays the time unit in microseconds. However, you can use whichever measurement you prefer while running or changing current simulation time, and the Waveform window adjusts accordingly.
4. Search for each `sineSel` signal transition, and add markers at each one.
   With markers identifying the transitions on `sineSel`, the Waveform window should look similar to **Figure 26**. As previously observed, the low frequency signals are incorrect when the `sinSel` signal value is 00.

![Figure 28: Using Markers](image_url)
You can also use the main Waveform window cursor to navigate to different simulation times, or locate value changes. In the next steps, you use this cursor to zoom into the Waveform window when the \texttt{sineSel} is 00, to identify where the incorrect behavior initiates. You will also use the cursor to measure the period of low frequency wave control.

5. In the Waveform window, \textbf{zoom in} to the start of the simulation, \textbf{time 0}, to review the status of the output signal, \textit{sine[19:0]}.

6. \textbf{Place} the main Waveform window \textbf{cursor} on the area \textbf{by clicking} on and dragging the cursor to the start of the simulation.

You can also place the cursor by clicking at the specific time in the Waveform window timeline.

7. \textbf{Because} 00 is the initial or default FSM output, \textbf{move} the cursor \textbf{to} the \textbf{first posedge} of \textit{clk} after reset is asserted low, at time 102.5 ns.

\textbf{Note:} You can select a signal and use the \textbf{Previous Marker} and \textbf{Next Marker} buttons to move the cursor between markers.

8. \textbf{Zoom in} to view the waveform more closely by \textbf{repeatedly} clicking the \textbf{Zoom In} button until you achieve the zoom needed \textbf{to see the details} of the signal, as seen in \textbf{Figure 27}.

The waveform window zooms in or out around the area centered on the cursor.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure29.png}
\caption{Cursor Zoom on Signals}
\end{figure}
You can also use the Waveform window main cursor to measure time between two points on the timeline. The Floating Ruler button displays a ruler at the bottom of the Waveform window useful for measuring time between two points.

Use this feature to measure the \texttt{sineSel} control signal period, and the corresponding \texttt{output_sine[19:0]} values during this time frame.

9. Enable the \textbf{Snap to Transition} button to snap the cursor to transition edges.

10. Place a \textbf{marker} at the time of interest, or \textbf{102.5 ns} in this case, and click to select the marker.

\textbf{Note:} If a marker is already placed at that location, click the marker to select it.

When you select the marker, a floating ruler opens at the bottom of the Waveform window, with time 0 on the ruler positioned at the marker. As you move the main cursor along the timeline, the ruler measures the time difference between the cursor and the selected marker.

11. If the ruler does not appear when you select the marker, enable the \textbf{Floating Ruler} button from the Waveform window sidebar menu.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Figure_30.png}
\caption{Measuring Time in the Waveform}
\end{figure}

Scroll down the timeline and place the cursor at the next \texttt{sineSel} transition, which occurs at 3522.5 ns.
You can move the timeline in a number of ways. You can scroll the horizontal scroll bar at the bottom of the Waveform window. You can zoom out, or zoom fit to view more of the time line, reposition the cursor as needed, and then zoom in for greater detail.

12. **Select** sineSel from the list of signals in the Waveform window, and use the **Next Transition** command to move to the specific transition of interest.

As shown in **Figure 28**, the ruler measures a time period of 3,420 ns as the period that FSM selected the low frequency output.

---

**Step 9: Debugging with Breakpoints**

You have examined the design using cursors, markers, and multiple Waveform windows. Now you will use Vivado simulator debugging features, such as breakpoints, and line stepping, to debug the design and identify the cause of the incorrect output.

First, open tutorial design testbench to learn how the simulator generates each design input.

1. **Open** the `testbench.v` file by double-clicking on the file in the Sources window.

The source file opens in the Vivado IDE Text Editor.

![Figure 31: Integrated Text Editor](image)
Note: You can also use File > Open File from the main menu, or Open File from the popup menu in the Sources window. You can also select an appropriate design object in the Scopes window or Objects window, right-click and select Go to Source Code.

Using Breakpoints

A breakpoint is a user-determined stopping point in the source code used for debugging the design. When simulating a design with set breakpoints, simulation of the design stops at each breakpoint to verify the design behavior. After the simulation stops, an indicator shows in the text editor next to the line in the source file where the breakpoint was set, so you can compare the Wave window results with a particular event in the HDL source.

You will use breakpoints to debug the error with the low frequency signal output that you previously observed. The erroneous `sine[19:0]` output is driven from the `sineGen` VHDL block. Start your debugging with this block.

1. Select the **U_SINEGEN** scope selected in the Scopes window, to list the objects from that scope in the Objects window.

2. In the Objects window, right-click on `sine[19:0]`, and use Go to Source to open the `sinegen.vhd` source file in the Text Editor.

   Looking through the HDL code, the `clk`, `reset`, and `sel` inputs are correct as expected. Set your first breakpoint after the `reset` asserts low at line 137.

3. Scroll to line 137 in the file.

   Add a breakpoint at line 137 in `sinegen.vhd`. Note that the breakpoint can be set only on the executable lines. Vivado simulator marks the executable lines using an arrowhead symbol on the left hand margin in the Text Editor, along with the line numbers.

   Setting a breakpoint causes the simulator to stop at that point, every time the simulator processes that code, or every time the counter is incremented by one.

4. Click the arrowhead in the left margin, to set a breakpoint.

   Observe that the arrow symbol changes to a dot to indicate that a breakpoint is set on this line. You can remove a breakpoint by clicking on the red dot to revert it to an arrowhead.

   ![Figure 32: Setting a Breakpoint](image)

To delete all breakpoints in the file, right-click on one of the breakpoints, and select Delete All Breakpoints.

Debugging in the Vivado simulator, with breakpoints and line stepping, works best when you can view the Tcl Console, the Waveform window, and the HDL source file at the same time.
5. **Resize** the windows, and use the window **Float** command if needed, to **arrange** the various **windows** so that you can see them all.

**TIP:** When you have arranged windows to perform a specific task, such as simulation debug in this case, you can save the view layout to reuse it when needed. Use the **Layout > Save Layout As** command from the main menu to save view layouts. See the Vivado Design Suite User Guide: Using the Vivado IDE (UG893) for more information on using view layouts.

6. Click the **Restart** button to restart the simulation from time 0.

7. Run the simulation by clicking the **Run All** button. The simulation runs to time 102.5 ns (displayed below as microseconds), or near the start of first counting, and stops at the breakpoint at line 137. The focus within the Vivado IDE changes to the Text Editor, where it shows the breakpoint indicator and the line highlights in yellow.

![Image of Vivado IDE interface with windows arranged for debugging](image-url)

**Figure 33:** Arrange Windows for Debugging

---

Logic Simulation

UG937 (v 2013.1) March 20, 2013

www.xilinx.com 39
A message also displays in the Tcl console to indicate that the simulator has stopped at a specific time, indicating the line of source code last executed by the simulator.

8. Continue the simulation by clicking the **Run All** button.

The simulation stops again at the breakpoint. Take a moment to examine the values in the Waveform window. Notice that the `sine[19:0]` signals in the Outputs group are uninitialized, as are the `sine_l[15:0]` signals in the Internal signals group.

9. In the Text Editor, **add another breakpoint at line 144** of the `sinegen.vhd` source file.

   This line of code runs when the value of `sel` is 00. This code assigns, with bit extension, the low frequency signal, `sine_l`, to the output, `sine`.


    These selected signals are highlighted in the Waveform window, making them easier for you to monitor.

11. Run the simulation by clicking the **Run All** button.

    Once again the simulation stops at the breakpoint, this time at line 144.

### Stepping Through Source Code

Another useful Vivado simulator debug tool is the **Line Stepping** feature. With line stepping, you can run the simulator one-simulation unit at the time. This is helpful if you are interested in learning how each line of your source code affects the results in simulation.

Step through the source code line-by-line, and examine how the low frequency wave is selected, and whether the DDS compiler output is correct.

1. **Click the Step button**

   The simulation steps forward to the next executable line, in this case in another source file. The `fsm.vdh` file is opened in the Text Editor. You may need to relocate the Text Editor to let you see all the windows as previously arranged.

   **Note:** You can also type the `step` command at the Tcl prompt.

2. **Continue to Step through the design, until the code returns to line 144 of `sinegen.vhd`**

   You have stepped through one complete cycle of the circuit. Notice in the Waveform window that while `sel` is 00, signal `sine_l` is assigned as a low frequency sine wave to the output `sine`. Also notice that `sine_l` remains uninitialized.

   Initialize the value of `sine_l` using the `add_force` command. This command will force the value of `sine_l` into a specific known condition, and can provide a repeating set of values to exercise the signal more vigorously if needed. Refer to the **Vivado Design Suite User Guide: Logic Simulation** (UG900) for more information on using `add_force`.  

---

*Logic Simulation*

**UG937 (v 2013.1) March 20, 2013*
3. Type the following into the Tcl Console: \texttt{add\_force sine\_l 0110011011001010}

4. Continue the simulation by clicking the \textbf{Run All} button.

   In the Waveform window notice that the value of \texttt{sine\_l[15:0]} is now set to the value specified by the \texttt{add\_force} command, and this value is assigned to the output signal \texttt{sine[19:0]}, since the value of \texttt{sel} is still 00.

5. \textbf{Trace} the \texttt{sine\_l} signal in the HDL source files, and \textbf{identify the input} for \texttt{sine\_l}.

6. In the Text Editor, use the \textbf{Find text in multiple files} button to search for \texttt{sine\_l}.

7. Select the \textbf{Match whole word} checkbox, and click \textbf{OK}.

   ![Figure 34: Find in Files](image)

   The Find in Files results display at the bottom of the Vivado IDE, with the different occurrences of \texttt{sine\_l} in the HDL source files. In this case, all occurrences are found in \texttt{sinegen.vhd}.

8. \textbf{Click} on the Find in Files results to view the results in the \texttt{sinegen.vhd} file.

   The second result, on line 111, identifies the problem with the design. At line 111 in the \texttt{sinegen.vhd} file, the \texttt{m\_axis\_data\_tdata\_sine\_low} signal is assigned to \texttt{sine\_l}. Since line 111 is commented out, the \texttt{sine\_l} signal is not connected to the low frequency DDS compiler output, or any other input.

9. \textbf{Uncomment line 111} in the \texttt{sinegen.vhd} file, and click the \textbf{Save File} button.

10. In the Tcl Console, remove the force on \texttt{sine\_l}: \texttt{remove\_forces -all}
Step 10: Re-launch Simulation

By using breakpoints and line stepping, you identified the problem with the low frequency output of the design, and corrected it.

Since you modified the source files associated with the design, you must recompile the HDL source and build new simulation snapshot. Not just restarting the simulation at time 0 in this case, but rebuilding the simulation from scratch.

1. In `sinegen.vhd`, select one of the breakpoints, right-click and select **Delete All Breakpoints**.
2. Click the **Re-launch** button on the main toolbar menu.

   The Vivado IDE prompts you to confirm re-launching the simulator.

   ![Image of Re-launch Simulation dialog]

   **Figure 35: Confirm Relaunch**

3. Click **OK** to continue.

   The Vivado simulator recompiles the source files with `xelab`, and re-creates the simulation snapshot. Now you are ready to simulate with the corrected design files.

4. Click the **Run All** button to run the simulation.

   Observe the `sine[19:0]`, the final sine wave output signal in the waveform configuration. The low frequency sine wave looks as expected.

   The Tcl console results are:

   ```
   [03518000] LEDs_n = 0100
   [03523000] LEDs_n = 0001
   [03523000] LEDs_n = 0001
   [06008000] LEDs_n = 0101
   [06013000] LEDs_n = 0010
   [06013000] LEDs_n = 0010
   $finish called at time : 7005 ns : File "ug937/sim/testbench.v" Line 63
   ```
Figure 36: Corrected Low Frequency Output

Summary

This completes Lab #2. Up to this point in the tutorial, between Lab #1 and Lab #2, you have:

- Run the Vivado simulator using the Project Mode flow in the Vivado IDE.
- Created a project, added source files, and added IP.
- Added a simulation-only files (testbench.v).
- Set simulation properties and launched behavioral simulation.
- Added signals to the Waveform window.
- Configured and saved the Waveform Configuration file.
- Debugged the design bug using breakpoints and line stepping.
- Corrected an error, re-launched simulation, and verified the design.

In Lab #3 you will examine the Vivado simulator batch mode.
Lab 3: Running Simulation in Batch Mode

Introduction

You can use the Vivado™ simulator Non-Project Mode flow to simulate your design without setting up a project in Vivado Integrated Design Environment (IDE).

In this flow, you:

- Prepare the simulation project by manually creating a Vivado simulator project file or use `xvlog` and `xvhdl` parser commands to create a simulation snapshot using `xelab` command.
- Start the Vivado simulator GUI by running the `xsim` command with snapshot generated by `xelab` command.

Step 1: Preparing the Simulation

The Vivado simulator Non-Project Mode flow lets you simulate your design without setting up a project in the Vivado IDE.

In this flow, you compile the HDL files and create a simulation snapshot by either:

- Manually creating a Vivado simulator project file specifying all HDL files to be compiled, and using the `xelab` command to reference the project file and create a simulation snapshot.
- Using the explicit parser commands `xvlog` and `xvhdl` to parse the source files and using `xelab` to create a simulation snapshot from the parsed file set in memory.

Following completion of this step, you can launch the Vivado Simulator GUI by running `xsim` with the `-snapshot` command created by `xelab`.

Method 1: Manually Create Vivado Simulator Project File

The Vivado Simulator Project File specifies each file and its associated library that is to be parsed and compiled for simulation. The best use for this method is when the files are repeatedly simulated such as during development.

The typical syntax for a Vivado simulator project file is as follows:

```
verilog|vhdl <library_name> {<file_name>.v|.vhd}
```
Where:

verilog|vhdl: Specifies whether the source is a Verilog or a VHDL file.

<library_name>: Specifies the library to which to compile the source. If unspecified, the default library for compilation is work.

<file_name>.v|.vhd: Specifies the name of the file to compile.

**IMPORTANT:** While you can specify one or more Verilog source files on a given line, you can specify only one VHDL source on a given line.

In this section, you will build a Vivado Simulator Project File by modifying an existing project file to include the missing source files of the design. The missing Verilog and VHDL files should be compiled using the general guidelines described above.

To build a Vivado simulator project file for the tutorial design:

1. **Browse** to the /scripts folder.

2. **Open** the `simulate_xsim.prj` project file with a text editor.

   The project file is incomplete.

3. **Add** the following **missing HDL sources** and their associated libraries along with the necessary commands to compile them to the project file:

   ```
   work : /sources/sinegen.vhd
   work : /sources/fsm.vhd
   work : /sources/debounce.vhd
   work : /sources/sinegen_demo.vhd
   work : /sim/testbench.v
   ```

4. **Save** and close the file.

   **Note:** You do not need to list the sources based upon their order of dependency. The `xelab` command automatically resolves the order of dependencies and processes the files in the appropriate order.

For reference, a completed version of the tutorial files can be found in the `/completed` folder.

**Method2: Create Parser Commands**

In addition to compiling HDL sources based on a project file, you can compile individual files directly from the command line. Use this method for single simulation runs, or shell script and makefile compilation.

In this method, you parse the Verilog and VHDL file using the `xvlog` and `xvhdl` commands on the command line.
Parse individual or multiple Verilog files using the `xvlog` command with the following syntax format:

```
xvlog [options] <verilog_file | list_of_files>
```

Parse the individual VHDL files using the `xvhdl` command with the following syntax format:

```
xvhdl [options] <VHDL_file>
```

For a complete list of available `xvhdl` command options, see the Vivado Design Suite User Guide: Logic Simulation (UG900).

1. **Open** and edit the `/scripts/parse_standalone.bat` file by adding the following missing HDL sources and their associated libraries along with the necessary commands to compile them.

   ```
   work : /sources/sinegen.vhd
   work : /sources/fsm.vhd
   work : /sources/debounce.vhd
   work : /sources/sinegen_demo.vhd
   work : /sim/testbench.v
   ```

2. **Save** and close the file.

   For reference, a completed version of the tutorial files can be found in the `/completed` folder.

---

**Step 2: Building the Simulation Snapshot**

In this step, the `xelab` command uses the project file created in the previous section to elaborate, compile, and link all the sources for the design. This creates a simulation snapshot that lets you to run the simulation in the Vivado Simulator GUI.

**Use the xelab Command**

The typical `xelab` command syntax is:

```
xelab -prj <project file> -s <simulation snapshot> <library.top_unit>
```

Where:

```markdown
- **-prj**: Specifies a Vivado simulation project file to use for input
- **-s**: Specifies the name of the simulation snapshot output file
- `<library.top_unit>`: Specifies the top design unit
```
Method 1: xelab with Vivado Simulator Project File

In this method, you use the xelab command with the project file completed in the previous step to elaborate, compile, then link all the sources for the design, and create the simulation snapshot.

The provided xelab batch file is incomplete, and you need to modify to produce the correct simulation snapshot using the xelab command guidelines provided above.

1. **Browse** to the /scripts folder.
2. **Open** the xelab_batch.bat file using a text editor.
   This xelab command is incomplete.
3. Using the syntax information provided above, **edit the command line** so it includes the following options:
   - Use incremental compilation by specifying the \(-\)incremental switch.
   - Reference the simulate_xsim.prj as the project file.
   - Specify run_sineGen as the simulation snapshot.
   - Specify work.testbench as the top-level design unit for simulation.
4. **Save** and close the batch file.

Method 2: Use xelab with xvlog and xvhdl Commands

In this method you use the xelab command after the xlog and xvhdl commands have pared the HDL into memory

1. **To open the ISE command prompt**, go to **Start > Programs > Xilinx ISE Design Suite > Accessories** and click the **ISE Design Suite 32bit** command prompt.
2. **Using the ISE command prompt**, navigate to and run the xelab_batch.bat file to run xelab.
   After the xelab command completes compiling source code, elaborating design units, and linking the object code, a simulation snapshot (run_sineGen) is available in the /scripts folder.
3. **Browse** to the /completed folder to see the completed version of the xelab batch file for comparison.
Step 3: Manually Simulating the Design

In this step, you launch the Vivado simulator GUI by running the xsim command with the simulation snapshot that you generated using the xelab command in Lab #2.

After you complete this step, you can use the Vivado simulator GUI to explore the design in more detail.

Run the Simulation Executable

The command syntax when launching the simulation executable is:

```
xsim -gui <snapshot> -view <wave_configuration_file> -wdb <waveform_database_file>
```

Where:
- `-gui`: Launches Vivado simulator in GUI mode
- `<snapshot>`: Specifies the snapshot that you generated with xelab command
- `-view`: Opens the specified waveform configuration file within the simulator GUI
- `-wdb`: Specifies the file name of the simulation database output file

Launch Simulation

1. **Browse** to the `/scripts` folder from the downloaded files.
2. **Open** the `simulate_xsim.bat` file using a text editor. The batch file is intentionally blank.
3. Using the command syntax information, **edit** the **batch file** so it includes the following settings:
   - Simulation snapshot name: `run_sineGen`
   - Launch in GUI mode: `-gui` option
   - Simulation database output name: `simulate_xsim.wdb`
4. Save and close the file.
5. From the Vivado simulator command prompt, **run** the `simulate_xsim.bat` file.

   **Note:** The tutorial files do not provide a waveform configuration file. You created the `tutorial_1.wcfg` file in the previous chapter.
Result

The Vivado Simulator GUI opens and loads the design. The simulator time remains at 0ns until you specify a run time.

You can browse to the /completed folder for a completed version of the simulate_xsim.bat batch file to compare your results.

Conclusion

In this tutorial, you:

- Created a Vivado IDE project.
- Downloaded source files and ran Vivado simulation.
- Examined the simulation customization features.
- Debugged and fixed a known issue within the source files.
- Ran a Vivado simulation in batch mode using the Vivado simulation executable and switch options.