Vivado Design Suite Tutorial:

*In Depth Simulation*

UG937 (v 2012.3) November 16, 2012
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 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>10/16/2012</td>
<td>2012.3</td>
<td>Initial release.</td>
</tr>
<tr>
<td>11/16/2012</td>
<td>2012.3</td>
<td>Corrected link to zip file.</td>
</tr>
</tbody>
</table>
# Table of Contents

Revision History ........................................................................................................................................................... 2  
Vivado Simulator Overview ........................................................................................................................................ 5  

## Introduction 

Introduction .................................................................................................................................................................. 5  
Tutorial Flow ................................................................................................................................................................. 5  
Software Requirements ............................................................................................................................................ 5  
Vivado Simulator Description .................................................................................................................................. 6  
Tutorial Design Description ..................................................................................................................................... 6  
Locating Tutorial Design Files ................................................................................................................................ 7  

## Running the Simulator in Vivado IDE 

Introduction .................................................................................................................................................................. 9  
Step 1: Creating a New Vivado Project .............................................................................................................. 9  
Step 2: Setup and Run Behavioral Simulation .............................................................................................. 16  
Next Steps .................................................................................................................................................................. 19  

## Using the Simulator GUI and Debugging the Design 

Introduction ............................................................................................................................................................... 20  
Step 1: Examining and Debugging the Design ............................................................................................ 20  
Step 2: Using the Analog Wave Viewer ........................................................................................................... 22  
Step 3: Understanding Design Bug ................................................................................................................... 24  
Step 4: Changing Signal and Wave Window Properties ............................................................................. 29  
Step 5: Setting Waveform General Options .................................................................................................. 30  
Step 6: Saving the Wave Window Configuration ........................................................................................ 31  
Step 7: Re-Simulating the Design ..................................................................................................................... 31  
Step 8: Using Markers ............................................................................................................................................ 32  
Step 9: Using Cursors ............................................................................................................................................. 33  
Step 10: Measuring Time ...................................................................................................................................... 34  
Step 11: Using Multiple Waveform Configurations .................................................................................... 35  
Step 12: Debugging the Design .......................................................................................................................... 36  
Step 13: Fixing Design Bugs ................................................................................................................................ 40
Summary..................................................................................................................................................................... 41
Running Simulation in Batch Mode......................................................................................................................... 42

Introduction............................................................................................................................................................... 42
Step 1: Preparing the Simulation......................................................................................................................... 42
Step 2: Building the Simulation Snapshot......................................................................................................... 45
Step 3: Manually Simulating the Design........................................................................................................... 46
Result............................................................................................................................................................................ 47
Conclusion.................................................................................................................................................................. 47
Chapter 1

Vivado Simulator Overview

Introduction

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

The tutorial lets you run the Vivado simulator in a Windows environment. To run certain steps successfully in another operating system, some modifications might be required.

Tutorial Flow

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

The tutorial files contain sources in Hardware Description Language (HDL), and, thus demonstrate how the Vivado IDE converts these sources to HDL source files that the Vivado simulator can then compile.

In this flow, you launch Vivado simulator using one of the simulation processes available in the Vivado IDE. In project mode, using the Vivado IDE GUI, you use the Vivado IDE to create a project and implement the design in a Xilinx 7 series FPGA.

Additionally, Chapter 4 shows you how to do the same simulation steps in a non-project mode, where you simulate your design by creating your own Vivado simulator project files and running the HDL linker and simulation executable in command-line or batch file mode.

Software Requirements

To use this tutorial, you must have installed the Vivado Design Suite 2012.2 or above.

For more information about installing Xilinx software, see the Xilinx Design Tools: Installation and Licensing Guide, (UG631).

Before starting the tutorial, ensure that the Vivado software is operational, and that you have downloaded the tutorial design data.
Vivado Simulator Description

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` (VHDL) and `xvlog` (Verilog) parsers for VHDL and Verilog files, respectively, that then store the parsed dump into a HDL library on disk.
- `xelab` (HDL elaborator and linker) command. For a given top-level unit, it 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` (Simulation command.) This executable loads up a simulation snapshot to effect a batch mode simulation, a GUI, or Tcl-based interactive simulation environment.
- Vivado Integrated Design Environment (IDE) simulator GUI.

Note: More information about the Vivado simulator is available in the Vivado Logic Simulation User Guide (UG900).

Tutorial Design Description

The tutorial design uses a set of RTL design sources consisting of Verilog and VHDL. The design contains the following blocks:

- 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 `sinegen.vhd`, and then MUXes the sine select results to the LED output (`sinegen_demo.vhd`).
- A simple testbench to simulate the sine wave generator design (`testbench.v`), 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

For more details on the functionality of this design, see the comments included in the design source files.
Locating Tutorial Design Files

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

Block Diagram

Figure 1 shows a block diagram of the design.

Figure 1: Design Block Diagram

Locating Tutorial Design Files

1. Download the ug937.zip file from the Xilinx website:
   http://www.xilinx.com/support/documentation/dt_vivado2012-3_tutorials.htm

2. Extract the zip file contents into any write-accessible location.

   This tutorial uses the unzipped Vivado_Simulator_Tutorial data.

   You modify the tutorial sample design data while performing this tutorial. A new copy of the original data is required each time you run the 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 a <code>run_xsim.csh</code> file, and the <code>testbench.v</code> 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 <code>readme.txt</code> is a readme file about the tutorial design.</td>
</tr>
</tbody>
</table>
Running the Simulator in Vivado IDE

Introduction

The Vivado™ Integrated Design Environment (IDE) provides an integrated flow with the Vivado simulator that lets you launch simulation directly from within the Vivado IDE. The Vivado IDE generates simulation commands automatically, and run in the background when simulating a design using the Flow Manager.

Step 1: Creating a New Vivado Project

1. Open the Vivado software:
   
   - On Windows, select either the Vivado Desktop icon, or Start > Programs > Xilinx Design Tools 14.3 > Vivado 2012.3 > Vivado.
   
   - On Linux, change the directory to /Vivado_Simulator_Tutorial, and type, vivado.

   The Vivado Getting Started page opens, as shown in Figure 2.

   ![Figure 2: Vivado Getting started page](image)
2. In the Vivado Getting started page, click **Create New Project**.

3. In the **New project** dialog box, click **Next**, then enter the project name (for example, **XSim_Tutorial**).

4. Enter the project location (you can use the browse feature to set the location), and click **Next**.

5. Select Project Type > RTL Project, and click Next.

6. In the Add Source wizard, click **Add Directories** and add the tutorial design /source directory and the /sim directory, and set the **Target Language** to **Verilog**.

7. Click **Next** twice to bypass adding existing IP and constraints.

8. In the Default Part, select Boards > Kintex-7 KC705 Evaluation Board, and click Next.

9. Review the project summary page, and click **Finish**.

Vivado creates a project called **XSim_Tutorial** and displays the Vivado IDE (**Figure 3**).

10. Ensure that **sim_1** shows as the simulation set. You also have the option to create other simulation sets.

    Vivado recreates the hierarchy view and displays the files (**Figure 4**).
Add IP from the IP Catalog to the Project

1. In the Vivado Project Manager, select the IP Catalog button. The IP Catalog opens in a separate window (Figure 5).

2. In the search field, type DDS Compiler. The IP Catalog window highlights the available DDS Compilers.

3. Double-click the DDS Compiler 5.0 AXI4-Stream IP. The Customize IP wizard opens.

Create sine_high IP

1. In Customize IP window, select the Configuration view.
2. In the IP Symbol tab on the left, ensure that Show Disabled Ports is unchecked.
3. In the Component Name field, type sine_high.
4. From the **Configuration Options** pull-down, select **SIN COS LUT only**.

5. Set **Noise Shaping** to **None**.

6. Set the **Hardware Parameters** to **Phase Width** as 16 and **Output Width** as 20.
   Observe that the IP Symbol reflects the configuration (Figure 6).

7. In Customize IP window, select the tab for the Implementation view.

8. Set the **Output Selection** to **Sine** (Figure 7).

![Figure 6: sine_high IP Configuration](image)

![Figure 7: sine_high IP Implementation](image)

10. In **Control Signals**, check **ARESETn** (active-Low) as shown in **Figure 8**.

![Figure 8: Control Signals: ARESETn](image)

11. Select the Summary1 view, review the configurations, and click **OK** (**Figure 9**).

![Figure 9: IP Summary](image)
Create sine_mid IP:

1. Double-click **DDS Compiler 5.0 AXI4-Stream IP**.
2. In Customize IP window, select the Configuration view **Component Name** field, type **sine_mid**.
3. From the **Configuration Options** pull-down, select **SIN COS LUT only**.
4. Set **Noise Shaping** to **None**.
5. In the **Hardware Parameters** set **Phase Width** to 8 and **Output Width** to 18.
6. In Customize IP window, select the tab for the Implementation view.
7. Set the **Output Selection** to **Sine**
8. In Customize IP window, select the Detailed Implementation view.
9. In **Control Signals**, check **ARESETn** (active-Low).
10. Select the Summary1 view, review the configurations, and click **OK**.

Create sine_low IP

1. Double-click **DDS Compiler 5.0 AXI4-Stream IP**.
2. In Customize IP window, select the Configuration view **Component Name** field, type **sine_low**.
3. From the **Configuration Options** pull-down option, select **SIN COS LUT only**.
4. Set **Noise Shaping** to **None**.
5. In the **Hardware Parameters** set **Phase Width** to 6 and **Output Width** to 16.
6. Select the Implementation view and set the **Output Selection** as **Sine**.
8. In **Control Signals**, check **ARESETn** (active-Low).
9. Select the Summary1 view, review the configurations, and click **OK**.
Generate Simulation Sources for IPs

1. Select the IP Sources view and select all three IP configurations.
2. Right-click and select **Generate Output Product** from the context menu (Figure 10).

![Figure 10: Enable Generate Output Products](image)

3. In the Manage Outputs window, set the **Actions** as follows:
   - In the **Instantiation Template, Synthesis, and Testbench**, set the **Action** to **Do Not Generate**.
   - In **Simulation**, set the **Action** to **Generate**, and click **OK** (Figure 11).

![Figure 11: IP Simulation library generation](image)
4. Check that the IP library files are generated with proper library mapping:

- In the Sources Window, go to Libraries.
- In the Design Sources directory, expand the IP hierarchy tree.
- Observe that different libraries exist for the IPs (Figure 12).

![Figure 12: IP Library Hierarchy](image)

---

**Step 2: Setup and Run Behavioral Simulation**

After you have created a Vivado project for the tutorial design, you set up and launch a Vivado behavioral simulation.

**Set Behavioral Simulation Properties**

Set the behavioral simulation properties in Vivado:

1. In the Flow Navigator, click **Simulation Settings**.
2. In the Compilation view, ensure that the debug level is set to the default of **typical**.
3. Change to the Simulation view, and observe:
   - The simulation set name is `sim_1`.
   - The Vivado simulator infers the top-level testbench name as `testbench` automatically (Figure 13).

![Figure 13: Simulation Settings: Compilation](image)

4. In the Simulation view, observe that the Simulation Run Time is set to a default of 1000ns, then click OK (Figure 14).
Step 2: Setup and Run Behavioral Simulation

Launch Behavioral Simulation

You can now launch Vivado simulator to perform a behavioral simulation of the tutorial design.

1. In the Flow Navigator, click **Run Simulation > Run Behavioral Simulation**, shown in Figure 15.

Post-Synthesis and Post-Implementation simulations are available after synthesis and implementation are run, respectively. Those simulations are outside the scope of this tutorial.

The Vivado IDE Simulator GUI (Figure 16) opens after successfully parsing and compiling the design.
By default, the top-level scope variables load in the waveform configuration view. 

**Note:** The `testbench.v` file opens at the point where simulation ends.

**Next Steps**

In Chapter 3, “Using the Vivado Simulator GUI and Debugging the Design,” you can learn more about the GUI features and tools for analyzing and debugging HDL designs.
Using the Simulator GUI and Debugging the Design

Introduction

The Vivado™ simulator GUI contains the waveform configuration window, and Object and Scope Windows. It provides a set of debugging capabilities to quickly examine, debug, and fix the design failure.

See the Vivado Simulator User Guide (UG900) for more information about the GUI components.

In this chapter, you examine design bug, enable debug capabilities, use different debug features to root cause the bug and modify changes in the code, then re-compile and re-launch the simulation.

Step 1: Examining and Debugging the Design

In this section, you look at the functional behavior of the tutorial design, which includes:

- 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
Open an Existing Project

If you completed Chapter 2, a Vivado project file (XSim_Tutorial.xpr) is available in the project area.

1. Double-click the Vivado project file. The Vivado GUI opens and loads the existing project.
2. From the Flow Navigator, click Run Simulation > Run Behavioral Simulation (Figure 17)

![Figure 17: Run Simulation](image)

The Vivado simulator starts compiling your design and loads the simulation snapshot.

Add Signals

Before running simulation for a specified time, you must add signals to the wave window to observe the signal status.

By default, the Vivado simulator adds available simulation objects from the testbench to the wave window. In the case of this tutorial, the following testbench signals load into the waveform window:

- Differential clock signals: sys_clk_p and sys_clk_n - This is a 200 MHz clock generated by the testbench 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 to display the results.

The design focus is to generate sine waves with different frequencies.

Monitor Signals

To observe the correct behavior, monitor a few of the design signals:

11. In the Scopes window, expand the testbench hierarchy.
12. Click the dut instance to list all the signals and constants in the Objects window.
13. From the Objects window, select signals sineSel and sine and add them into wave configuration window using one of the following methods:
14. Drag and drop the selected signals into waveform configuration window.
15. Right-click a signal, and from the context menu select **Add to Wave Window** (Figure 18). You can select multiple signals by holding down the **CTRL** key during selection.

**Figure 18: Add signals to Wave Window**

---

**Step 2: Using the Analog Wave Viewer**

The signal to monitor is an analog signal, and you can view it better in Analog wave mode.

To set the Waveform Style:
1. Right-click and select the `sine[19:0]` signal.
2. From context menu, select **Waveform Style > Analog** (Figure 19).

**Figure 19: Enable Analog Waveform Style**

3. Set the **Radix** for the analog signal as **Signed Decimal** (Figure 20)

**Figure 20: Set Radix to Signed Decimal**
Enable Signal Dumping for Debugging

For debugging, you might require many more signals and scopes to trace failures. However, after you run simulation, you can only see the value changes on the signals that you added into wave window.

You can also enable value change dump for design signals without adding them into the wave window.

To enable signal and scope dumping, enter the following command in the Tcl console, as shown in Figure 21:

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

This enables signal dumping for the scopes `/testbench/dut/*` and `/testbench/dut/u_sinegen/*`. 
Step 3: Understanding Design Bug

Run the simulation by clicking the Run All button.

1. Observe the sine signal output in the waveform.
2. Wave Window can be undocked from Main window layout to view it as standalone.
3. Click the undock icon in the left top corner of the waveform configuration window.
4. To display the full time spectrum in the waveform configuration window, click the Zoom Full View button.
5. You can use the horizontal and vertical scroll bars to view the full waveform configuration.
6. Notice that the low frequency sine output is incorrect (Figure 22).

![Figure 22: Design Bug – Wave View](image)

7. In the above wave view, when `sineSel` is 00, which indicates that `ii` is a low frequency sine selection. During this time, the sine output is not correct (not a proper sine wave).
8. Review the Tcl Console for messages from the testbench (Figure 23).

![Figure 23: Design Bug TCL Console Log](image)
Add Signal Groups

Next, you add signals from other design units to better analyze the functionality of this design. When you add additional signals to the waveform configuration window, the size of the wave window is not large enough to display all signals in the same view. Reviewing all signals would require the use of the vertical scroll bar repeatedly, making the review process tedious.

To enable better viewing, you can collect signals into a Group. With a group, you can collectively show or hide signals of similar purpose.

To group signals in the waveform configuration window:

1. In the waveform configuration window, click 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).
2. With the signals selected, right-click and select New Group (Figure 24).

![Figure 24: Create Signal Group]

3. Type a name for the new group. For this example, name the group TB Signals (Figure 25).

![Figure 25: Signal Group Naming]

Vivado Simulator creates a collapsed group in the waveform configuration window.
4. To expand the group, click once to the left of the group name.

![Figure 24: Create Signal Group]
Figure 26 shows the expanded group view.

![Figure 26: Signal Group View](image)

5. Create another signal group called **DUT Signals** to group signals `sine` and `sine_sel`:
   
   If your signal groups do not match, you can do the following to fix them:
   
   - If you included an unrelated signal, cut and paste it into the main list.
   - If you created the group but missed a signal in the main list, drag and drop the signal from Objects Window into the group.
   - You can start over by ungrouping; right-click and select Ungroup.

### Add Dividers

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

To add dividers to the waveform configuration window:

1. Right-click anywhere in the window, select **New Divider**, then enter a name.

2. Add two dividers named:
   
   - TESTBENCH
   - SineGen

3. Click and drag the **TESTBENCH** divider to the top of the list.

4. Move the other dividers to the bottom of the list.

   **Note:** You can change divider names at any time by highlighting the divider name and right-clicking and selecting the **Rename** option; then entering a new name.
The waveform configuration window should look like the one shown in Figure 27.

![Figure 27: Add Dividers](image)

**Add Signals from Submodules**

Now, add lower-level module signals from the instantiated `sine_gen_demo` module (dut) and `sinegen` module (`u_sinegen`) to study the interactions between these submodules and the testbench test signals. The easiest way to add signal to a group is to filter and then select the signals.

To add the necessary signals:

1. In the Scopes window select the Instances view and expand the hierarchy under `testbench`.

2. **Click the`dut` instance of entity `sine_gen_demo`.**

   Simulation objects associated with the currently highlighted design unit display in the Objects window.

   By default, all types of simulation objects (such as variables, and constants) display in the Objects window.
Figure 28 shows the object types.

You can filter by the simulation object types.

Use the Objects window toolbar to filter by inputs, outputs, bidirectional, internal, constants, and variables.

1. Toggle the object types by clicking the corresponding button.
2. In the Scopes window, select the U_SINEGEN design unit, select input and output ports and internal signals in the Object window toolbar.
3. In the Objects window, select all the displayed objects by type, then drag and drop them into the waveform configuration window, under the sineGen divider.

   **Note:** You can also add these signals to the waveform configuration window using the `add_wave` Tcl command in the Tcl console.

   For example, in the Tcl Console, type:

   ```
   add_wave add_wave /testbench/dut/U_SINEGEN
   ```

4. From the Scope window drag and drop the **U_SINGEN** signal into the waveform configuration window.
5. Create groups for the signals you added. Define the groups as Inputs, Outputs, and Internal for each set of signals (*Figure 29*).
Step 4: Changing Signal and Wave Window Properties

Next, you change the properties of some of the signals shown in the waveform configuration window to better visualize the simulation.

Change the Signal Name Format

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.

To change the signal name format for the `sine[19:0]` and `sel[1:0]` bus signals:

1. In the waveform configuration window, press Ctrl+click to select the `sine[19:0]` and `sel[1:0]` signals listed in the **SineGen** group.
2. Bring up the context menu by right-clicking, and change the name format from **Short** to **Long**.

Change the Signal Radix Format

Some signals interpret better if seen in hexadecimal format instead of binary format: for example, the signals `s_axis_phase_tdata_high`, `s_axis_phase_tdata_mid` and `s_axis_phase_tdata_low`.

Change the radix options for these signals as follows:

1. In the waveform configuration window, select the signals.
2. Right-click and select **Radix > Hexadecimal**.
Step 5: Setting Waveform General Options

Next, customize the look and feel of the complete waveform configuration window and reconfigure some of the default settings:

1. Observe the contents of the Options window by clicking the General Options button. The Waveform Options dialog box opens (Figure 30).

2. In the Default Radix settings pull-down, change the Radix from the Binary default to Hexidecimal.

By default, the area within the waveform is shaded. You can enable or disable the shading by selecting the Draw Waveform Shadow checkbox.

3. You can view the signal list along with line number. To enable or disable the signal list numbering, check or uncheck the Show Signal Indices checkbox.

You can configure the default coloring by signal type to customize the look and feel of waveform view. You can:

- In the Waveform Options dialog box, select the Colors view.
- Set colors for each component by clicking the Color column and choosing colors from the pull-down option.
Step 6: Saving the Wave Window Configuration

You can save the current waveform configuration so it is available for use in future Vivado simulation design sessions. By default, the Vivado simulator saves the current waveform configuration setting as `tb_top_behave1.cfg`.

To save the waveform configuration to a different name:

1. In waveform toolbar, click the **Save Wave Configuration** button.
2. Save the current Wave configuration to the filename `tutorial_1.wcfg`.
   The Vivado simulator saves the waveform configuration for future use.
   **Note**: You can load the saved waveform window configuration using
   **File > Open Waveform Configuration**.

Step 7: Re-Simulating the Design

You are ready to simulate the design again with the updated waveform configuration.

1. Click the **Run All** button.
   **Note**: You can also re-run the simulation by typing `run all` in the Tcl console.
   The simulation runs for about 705ns.
2. After the simulation is complete, click the **Zoom to Full View** button.
   The waveform configuration should look similar to **Figure 31**:
Step 8: Using Markers

The FSM block used in the design top module generates three different sine wave select signals for specific sine outputs from the SineGen block. You can view these different wave selections better using Markers.

To use markers for each sine selection:

1. In the **Name** column of the waveform, 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 window you prefer while running or changing current simulation time, and the Waveform window adjusts accordingly.

Similarly, search each `sineSel` signal changes and adds markers for each mode selection. The waveform window should look similar to **Figure 32**.

---

**Figure 32: Using Markers to Identify SineWave Selection**
Step 9: Using Cursers

As previously observed, the low frequency signals are incorrect in the waveform when the sinSel signal is 00. You can use the main yellow cursor to navigate to different time period or value changes.

In the next steps, you use this cursor to zoom in the wave window when the sinSel changes to 00 where the incorrect behavior initiates. You also use the cursor to measure the period of low frequency wave control.

Use the Zoom Feature

In the waveform configuration window, zoom in to the start of sinSel to review the status of final output signal sine[19:0].

To use a cursor for zooming in on a specific area:

1. Place the cursor on the area by clicking and dragging the main (yellow) cursor close to the marker that represents the start of sinSel 00 (marker at time 102.5ns).

   Because 00 is a default FSM output, move the cursor first to the posedge of clk after the reset is asserted (at time 102.5ns).

   Note: You can also click the Previous Marker and Next Marker toolbar buttons to move the main cursor from marker to marker.

2. Zoom in using the Zoom In button

   The waveform window zooms in around the area specified by the cursor.

3. Repeat Step 2 until you can see the rising and falling edges of the signal clearly at the cursor (Figure 33).

   ![Figure 33: Cursor Zoom on signals](image)

---

**In Depth Simulation**

**UG937 (v 2012.3) November 16, 2012**

www.xilinx.com
Step 10: Measuring Time

You can use your mouse cursor to measure time between two endpoints. You use this feature to confirm the `sineSel` control signal time frame and the corresponding `output_sine[19:0]` during this time frame.

To measure time using cursors:

1. Use the **Snap to Transition** toggle button to snap the cursor to transition edges.
2. Click and hold in an area at time **100.5ns** (main yellow cursor)
3. While holding the button, move the mouse over to the next `sineSel` value change (marker at **3522.5ns**).

A floating Ruler opens at the bottom of waveform configuration window and the time between the two defined endpoints shows as a time delta (Figure 34).

The cursors show that a measured time difference of **3,422ns** is the time that FSM chose low frequency output.

**Note:** You can use the **Floating Ruler** button (available from the waveform window toolbar) to display a hovering ruler over the waveform configuration. This feature is available when performing a time measurement using cursors between two endpoints. The zero (0ps) on the ruler is placed at the first time endpoint. This feature is useful when making multiple time measurements with respect to the first endpoint.
Step 11: Using Multiple Waveform Configurations

Depending on the resolution of the screen, a single waveform configuration window might not display all the signals of interest at the same time. You can open multiple waveform configuration windows, each with their own set of signals and signal properties.

To open a new waveform configuration window:

1. In the Vivado simulator main menu, select File>New Waveform Configuration.

2. A blank waveform configuration opens with default Untitled 1 configuration name.

3. In the main menu, select File >Save Waveform Configuration as to save it in a different name tutorial2.wcfg.

   You can move dividers, groups, and simulation objects to the new waveform configuration.

To move signal groups associated with sineGen unit to a new waveform configuration window:

4. Select signal groups by pressing and holding the Ctrl key, and highlighting signal groups (inputs, outputs, and internals).

5. Right-click on the selection and select Cut Or use shortcut Ctrl-x on the selection

6. Click the new waveform configuration window.

7. Right-click in the waveform configuration window Name column, and select Paste.

   Alternatively, use the shortcut Ctrl-V to paste.
Step 12: Debugging the Design

You examined the design using markers, cursors, and multiple waveform configurations. Now use Vivado simulator debugging features, such as using breakpoints and stepping through source code, to debug the design and address the incorrect output results.

View Source Code

First, look at the tutorial design testbench to learn how each design input is generated.

Do one of the following to open the source code for the tutorial design testbench (testbench.v):

1. In Vivado Simulator main window, select **File > Open File**, and open the file of choice.
2. In the Scopes window, right-click the design unit described by the source file, then select **Go to Source Code**.
3. In the Objects window, right-click on any of the simulation objects declared in the chosen source file, then select **Go to Source Code**.
4. In the Sources view, double-click a source file.

The source file opens using integrated test editor (**Figure 35**).

---

**Figure 35: Integrated Text Editor**
Use Breakpoints and Line Stepping

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 of source code where the breakpoint was set, so you can compare the Wave window results with a particular event in the 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.

You next use both of these debugging features to learn how to chose one of the Sine wave output and pull that file out of the design to debug the failing scenario.

Setting Breakpoints

The $\text{sine[19:0]}$ output is driven from $\text{sineGen}$ VHDL block. The inputs $\text{clk}$, $\text{reset}$, and $\text{sel}$ are correct as expected. Start your debugging with this block.

1. Open the $\text{sinegen}$ source file using one of text editor methods. Start your observation after the reset asserts.

2. Set the breakpoint at line 137 in $\text{sinegen.vhd}$:

   Vivado simulator marks the executable lines using symbol $\bullet$ at the text editor pane along with line numbers. Note that the breakpoint can be set only on the executable lines.

3. Go to line 137 and click $\bullet$

   Observe that the symbol changes to $\checkmark$ to indicate that the breakpoint is set on this line (Figure 36).

   ![Figure 36: Setting a Breakpoint](image)

   Setting a breakpoint causes the simulator to stop every time the counter is incremented by one.

   You can enable or disable particular breakpoint by click again on $\checkmark$. The red mark goes away and the symbol revert to $\bullet$. Each selection toggles between $\bullet$ and $\checkmark$. 


4. To delete all breakpoints in the file, right-click on one of the breakpoints, and select **Delete All Breakpoints** *(Figure 37).*

![Figure 37: Delete All Breakpoints](image)

**Re-Run the Simulation with the Breakpoint Enabled**

Next, rerun the simulation with the breakpoint enabled.

1. Debugging with breakpoints and stepping features work best when you are able to review the console output and the waveform window at the same time.
2. Use the Vivado float feature , or resize the windows to best accommodate the windows so they can be viewed at the same time.
3. Click the **Restart** button to restart the simulation from time 0.
4. Run the simulation by clicking the **Run All** button.

The simulation runs near the start of first counting and stops at line 137.

The focus within the file changes to the text editor where it shows mark indicator and the **Toggle Column Selection** color changes to yellow *(Figure 38).*

![Figure 38: Breakpoint Indicator](image)

Additionally, a message displays in the Tcl console to indicate that the simulator has stopped, including the line of source code last executed by the simulator.
Stepping Through Source Code

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

To step through the simulation:

1. in the `sinegen.vhd` file, add another breakpoint on line 144.
2. As you can observe, this is the line getting executed when the `sel` is 00, which is assigning low frequency wave to output.
3. Click the Step button.

   **Note:** You can also step through the simulation by typing `step` command at the Tcl prompt.
4. Use this process to step through the design, paying attention to each of these signal events:
   - Signal `sine_l` is assigned as low frequency sine wave to the output.
5. Observe in the waveform window that even after 155ns simulation time, `sine_l` is still uninitialized.
6. Trace source of input for `sine_1` and signal `sine_l`. This signal is not driven by any source neither it is connected to low frequency DDS compiler output.
   - Low frequency DDS compiler `sine_low` (U_SL), results output
   - `m_axis_data_tdata` is connected to internal signal
   - `m_axis_data_tdata_sine_low`.

   7. Observe this signal in the waveform configuration window;
8. Enable the **Analog Style** wave mode to see the results much easier way. The signal results are also uninitialized.

   9. In the `sinegen.vhd` file, trace the `m_axis_data_tdata_sine_low` signal.

      At line 111 in the `sinegen.vhd` file, the `m_axis_data_tdata_sine_low` signal is assigned with `sine_1`. Also, this signal is connected to output port of the `sine_low` DDS compiler.

      The interconnect signal `m_axis_data_tdata_sine_low` is driven incorrectly.
Step 13: Fixing Design Bugs

By using breakpoints and line stepping, you have determined that the interconnect low frequency signals are assigned incorrectly.

In the next steps, revise the design code to connect and drive the output $\text{sine[19:0]}$ properly.

1. In the text editor, go to the $\text{sinegen.vhd}$ source code file.

2. Go to line 111, and revise the wrong assignment to internal $\text{sine_l}$:

3. Assign the output of the $\text{sine_low}$ DDS compiler to internal signal $\text{sine_l}$ so that the low frequency values are propagated correctly to the output port when $\text{sel}$ is $00$ (Figure 39).

   
   ![Figure 39: Line 111: m_axis_data_tdata_sine_low](image)

4. Save the file

Verify the Bug Fix

Now that you have corrected the design bug, you re-compile the source code and build new simulation snapshot.

1. Select one of the breakpoints and right-click and select Delete All Breakpoints.

2. Click the Re-launch button.

The Vivado simulator recompiles the source file and re-creates the simulation snapshot.

Now you are ready to simulate with updated design files.

3. Click the Run All button to re-run the simulation.

4. Observe the $\text{sine[19:0]}$, the final sine wave output signal in the waveform configuration; the low frequency sine wave is as expected in Analog Style (Figure 40).
Summary

This completes this chapter. Up to this point in the tutorial, you have run the Vivado simulator using the Project Mode flow in the Vivado IDE, where you:

- Created a project
- Added source files to the project
- Created IP from the IP Catalog and generated simulation libraries
- Added a simulation-only files (`testbench.v`)
- Set simulation properties and launched behavioral simulation
- Reviewed the Vivado Simulator GUI
- Debugged the design bug using different debug features
- Verified a bug fix

In Vivado simulator batch mode, described in Chapter 4, “Running Vivado Simulator in Batch Mode”, you can do all the same steps using the Vivado Tcl command mode.
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.

Chapter 1, Software Requirements and Locating Tutorial Design Files describe the software requirements and location of tutorial design.

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.

**Note:** 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.
**Method 2: 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.

1. Parse individual or multiple Verilog files using the `xvlog` command with the following syntax format:
   ```
xvlog [options] <verilog_file | list_of_files>
   ```

2. 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).*

   **Note:** While you can specify one or more Verilog source files for each individual `xvlog` command, you can specify *only one* VHDL source per `xvhdl` command.

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

3. 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
   ```

4. Save and close the file.
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 -incremental -prj <project file> -s <simulation snapshot><library.top_unit>
```

Where:

- `incremental`: Instructs `xelab` to compile only the files that have changed since the last compile
- `-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 correctly produce the 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:
   a. Use incremental compilation by specifying the `-incremental` switch.
   b. Reference the `simulate_xsim.prj` as the project file.
   c. Specify `run_sineGen` as the simulation snapshot.
   d. 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 xvlog and xvhdl commands have parsed 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 Step 2: Building the Simulation Snapshot.

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`

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

4. Save and close the file.

5. Using the Vivado simulator command prompt, navigate to and run the `simulate_xsim.bat` file.

---

**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 executables and switch options