Laboratory Assignment 3 – Adding IP Sources and Simulating a Design

Introduction

This lab 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. It is based on Xilinx Logic Simulation Tutorial UG937 (v2016.3).

Objectives

After completing this lab, you will be able to:

- Run the simulator in Vivado IDE
- Add simple IPs from the IP Catalog
- Display signal waveforms with analog viewer and debug the design with breakpoints
- Use cursors, markers and measuring time

Procedure

This lab is broken into steps that consist of general overview statements providing information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

Design Description

The design shown in Figure 1 consists of: a sine wave generator that generates high, medium, and low frequency sine waves (sinegen.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 the sine wave selector (debounce.vhd); and a design top module that resets FSM and the sine wave generator, and then multiplexes the sine select results to the LED output (sinegen_demo.vhd).

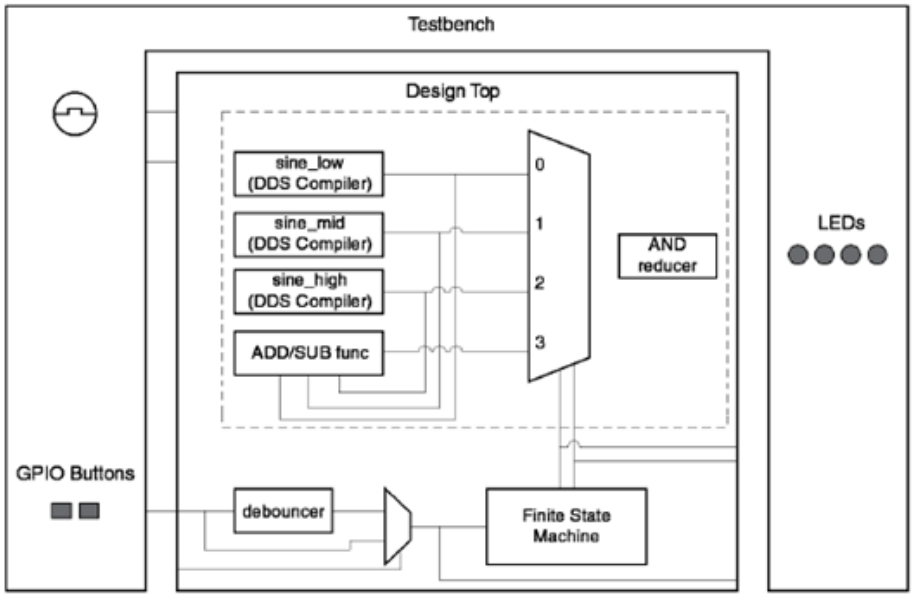
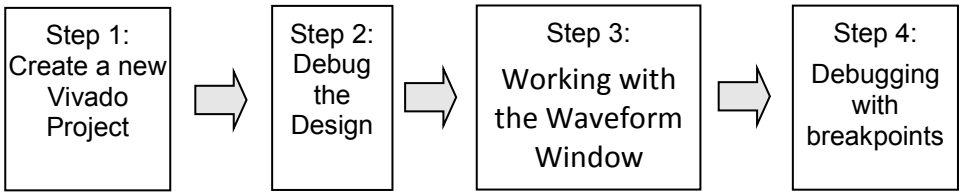


Figure 1. Frequency synthesis Design

General Flow



In the instructions below;
{***sources***} refers to: C:\Users\MEE_CE\sources
{***labs***} refers to : C:\ Users\MEE_CE\labs

Create a new Vivado Project — Step 1

1-1. Launch Vivado and create a project targeting XC7A35TCPG236-1 (Basys3) and using the Verilog HDL.

1-1-1. Start downloading the source files provided (Moodle) and save them in {sources}/lab3

1-1-2. Open Vivado and click **Create New Project** to start the wizard. You will see *Create A New Vivado Project* dialog box. Click **Next**.

1-1-3. Click the Browse button of the *Project location* field of the **New Project** form, browse to **{labs}**, and click **Select**.

1-1-4. Enter **lab3** in the *Project name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.

1-1-5. Select **RTL Project** option in the *Project Type* form, and click **Next**.

1-1-6. In the Add Source dialog box, click Add Directories and add the extracted tutorial design data:

- <Extract_Dir>/sources
- <Extract_Dir>/sim

Note: You can press the Ctrl key to click and select multiple files or directories.

1-1-7. Set the Target Language to **Verilog** to indicate the netlist language for synthesis.

1-1-8. Set the **Simulator Language** to Mixed as seen in Figure 2. Click **Next.**

1-1-9. Click **Next** twice to bypass the *Add Existing IP* and *Add Constraints* dialog boxes.

1-1-10. In the *Default Part* form, using the **Parts** option and various drop-down fields of the **Filter** section, select **XC7A35TCPG236-1** (for the Basys3) part. Using the **Boards** option, you may select the **Basys3**.

1-1-11. Click **Next**.

1-1-12. Click **Finish** to create the Vivado project.

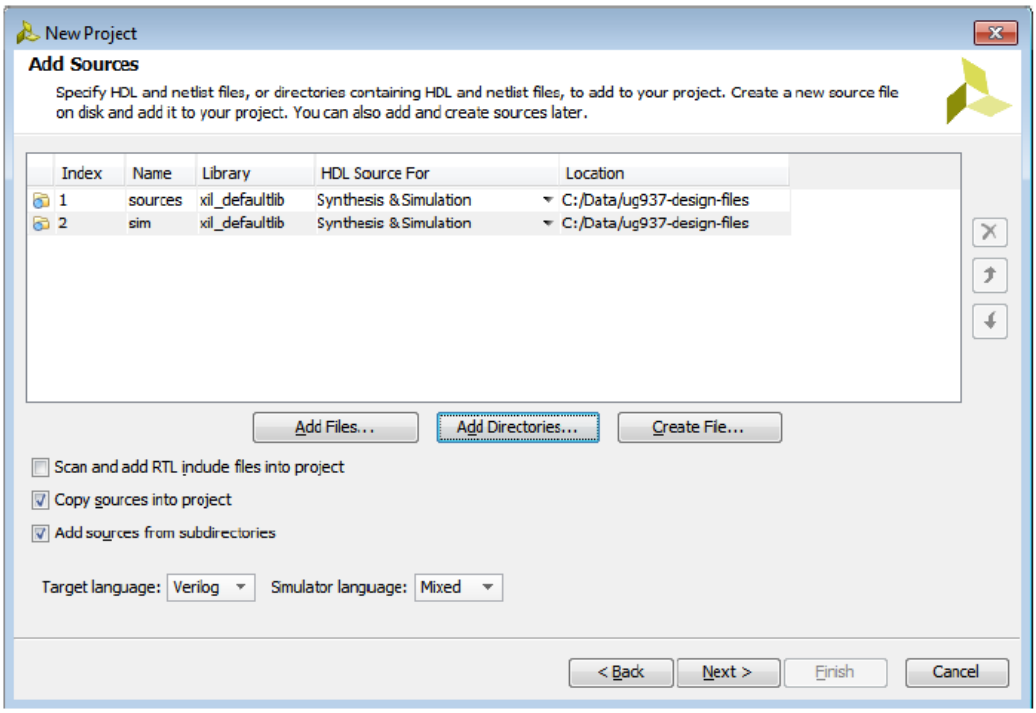


Figure 2. Opening the source file

1-2. Adding IP from the IP Catalog.

1-2-1. The Sources window displays the source files that you have added during project creation. The **Hierarchy** tab displays the hierarchical view of the source files.

1-2-2. Click the '+' character in the Sources window to expand the folders as shown in Figure 3.

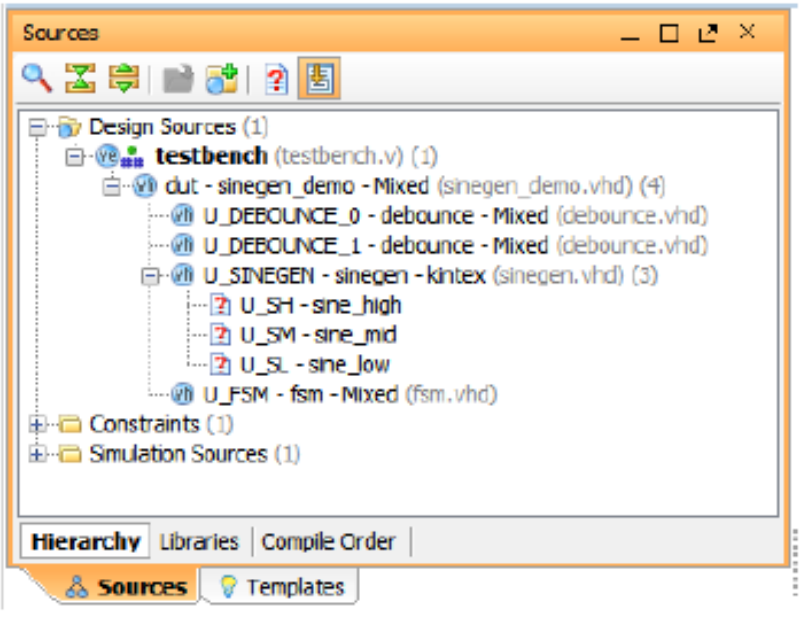


Figure 3. Sources Window

Notice that the Sine wave generator (sinegen.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. Next, you add the sine_high, sine_mid, and sine_low modules to the project from the Xilinx IP Catalog.

1-3. Adding Sine High.

1-3-1. In the Flow Navigator, select the **IP Catalog** button. The IP Catalog opens in the graphical windows area.

1-3-2. In the search field of the IP Catalog, type DDS. The Vivado IDE highlights the DDS Compilers in the IP catalog. Under any category, double-click the **DDS Compiler**. The Customize IP wizard opens (Figure 4).

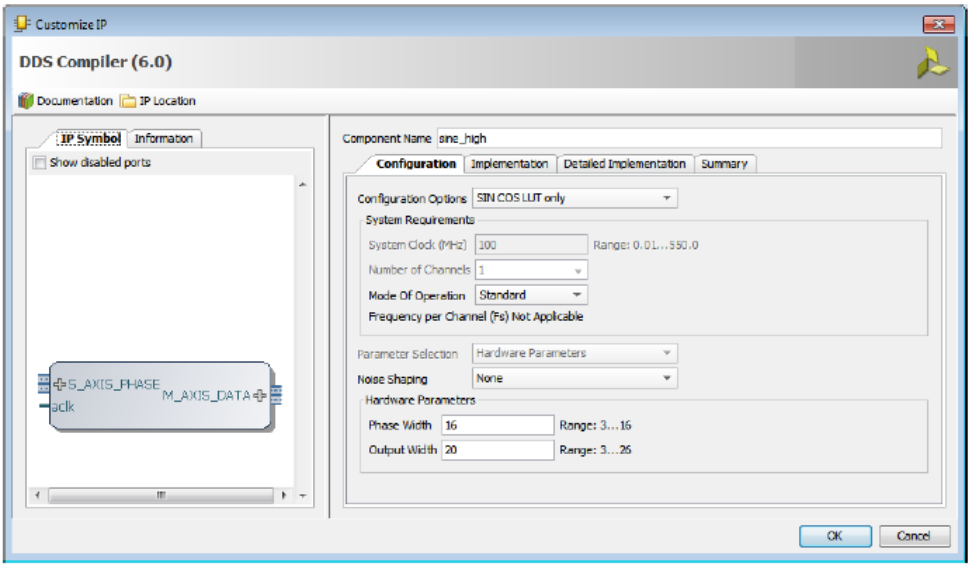


Figure 4. Customize IP - DDS Compiler

1-3-3. In the IP Symbol on the left, ensure that **Show disabled ports** is unchecked.

1-3-4. Specify the following on the Configuration tab:

- **Component Name**: type *sine_high*
- **Configuration Options**: select **SIN COS LUT only**
- **Noise Shaping**: select **None**
- Under **Hardware Parameters**, set **Phase Width** to 16 and **Output Width** to 20

1-3-5. On the **Implementation** tab, set **Output Selection** to **Sine**

1-3-6. On the **Detailed Implementation tab**, set **Control Signals** to **ARESETn** (active-Low)

1-3-7. On the Summary tab, review the settings and click OK (Figure 5).

Component Name sine_high

Configuration | Implementation | Detailed Implementation | **Summary**

Output Width	20 Bits
Channels	Not Applicable
System Clock	Not Applicable
Frequency per Channel (Fs)	Not Applicable
Noise Shaping	None
Memory Type	Block ROM (Auto)
Optimization Goal	Area (Auto)
Phase Width	16 Bits
Frequency Resolution	Not Applicable
Phase Angle Width	16 Bits
Spurious Free Dynamic Range	Not Applicable
Latency	6
DSP48 slice	0
BRAM (18k) count	19

Figure 5. Sine High Summary

When the *sine_high* IP core is added to the design, the output products required to support the IP in the design must be generated. The Generate Output Products dialog box displays, as shown in Figure 6. The output products allow the IP to be synthesized, simulated, and implemented as part of the design.

1-3-8. Click Generate to generate the default output products for *sine_high*.

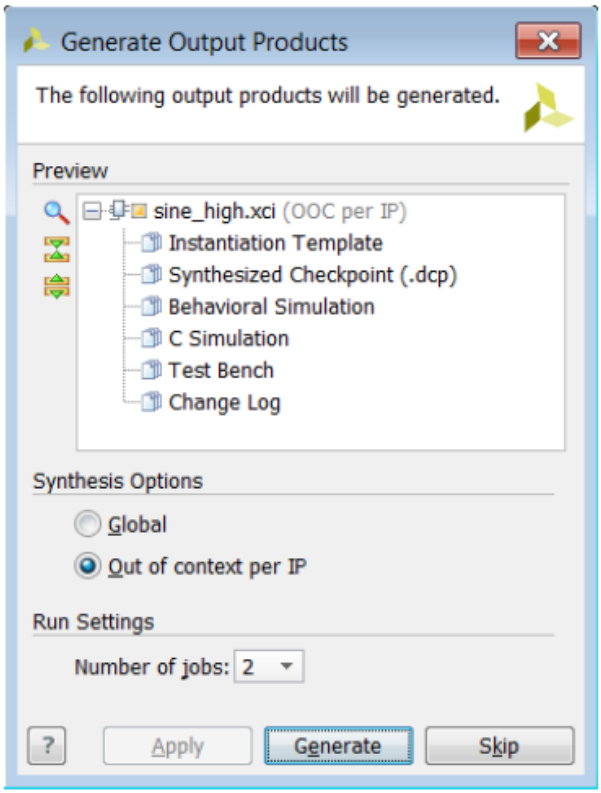


Figure 6. Generate Output Products

1-4. Adding Sine Mid.

1-4-1. In the **IP catalog**, double-click the **DDS Compiler** IP a second time.

1-4-2. Specify the following on the Configuration tab:

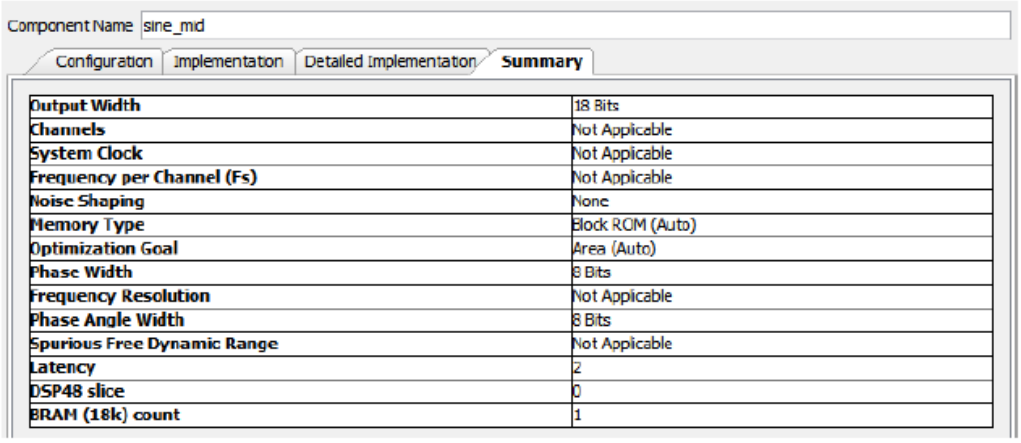
- **Component Name**: type *sine_mid*
- **Configuration Options**: select **SIN COS LUT only**
- **Noise Shaping**: select **None**
- Under **Hardware Parameters**, set the **Phase Width** to 8, and the **Output Width** to 18

1-4-3. On the **Implementation** tab, set the **Output Selection** to **Sine**

1-4-4. On the **Detailed Implementation** tab, set **Control Signals** to **ARESETn** (active-Low)

1-4-5. Select the **Summary tab**, review the settings and click OK (Figure 7). When the *sine_mid* IP core is added to the design, the Generate Output Products dialog box displays to generate the output products required to support the IP in the design.

1-4-6. Click **Generate** to generate the default output products for *sine_mid*.


Component Name: sine_mid

Configuration | Implementation | Detailed Implementation | **Summary**

Output Width	18 Bits
Channels	Not Applicable
System Clock	Not Applicable
Frequency per Channel (Fs)	Not Applicable
Noise Shaping	None
Memory Type	Block ROM (Auto)
Optimization Goal	Area (Auto)
Phase Width	8 Bits
Frequency Resolution	Not Applicable
Phase Angle Width	8 Bits
Spurious Free Dynamic Range	Not Applicable
Latency	2
DSP48 slice	0
BRAM (18k) count	1

Figure 7. Sine Mid Summary

1-5. Adding Sine Low.

1-5-1. In the **IP catalog**, double-click the **DDS Compiler** IP for the third time.

1-5-2. Specify the following on the **Configuration** tab:

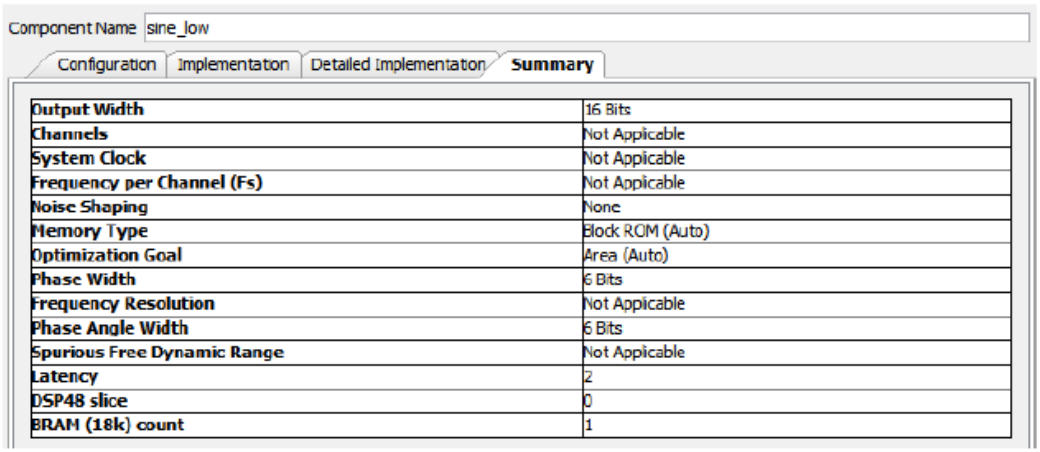
- **Component Name**: type *sine_low*
- **Configuration Options**: select **SIN COS LUT only**
- **Noise Shaping**: select **None**
- Under **Hardware Parameters**, set the **Phase Width** to 6, and the **Output Width** to 16

1-5-3. On the **Implementation** tab, set the **Output Selection** to **Sine**

1-5-4. On the **Detailed Implementation** tab, set **Control Signals** to **ARESETn** (active-Low)

1-5-5. Select the **Summary tab**, review the settings and click OK (Figure 8). When the *sine_low* IP core is added to the design, the Generate Output Products dialog box displays to generate the output products required to support the IP in the design.

1-5-6. Click **Generate** to generate the default output products for *sine_low*.


Component Name: sine_low

Configuration | Implementation | Detailed Implementation | **Summary**

Output Width	16 Bits
Channels	Not Applicable
System Clock	Not Applicable
Frequency per Channel (Fs)	Not Applicable
Noise Shaping	None
Memory Type	Block ROM (Auto)
Optimization Goal	Area (Auto)
Phase Width	6 Bits
Frequency Resolution	Not Applicable
Phase Angle Width	6 Bits
Spurious Free Dynamic Range	Not Applicable
Latency	2
DSP48 slice	0
BRAM (18k) count	1

Figure 8. Sine Low Summary

Debugging the Design — Step 2

2-1. Run Behavioral Simulation.

2-1-1. In the Flow Navigator, click **Simulation Settings**. The following defaults are automatically set:

- **Simulation set**: select **sim_1**
- **Simulation top-module name**: set **testbench**

2-1-2. In the **Simulation ta**b, observe that the **Simulation Run Time** is 1000ns. Click **OK**.

With the simulation settings properly configured, you can launch Vivado simulator to perform a behavioral simulation of the design.

2-1-3. In the Flow Navigator, click **Run Simulation >Run Behavioral Simulation**.

Functional and timing simulations are available post-synthesis and post-implementation. Those simulations take into consideration logic and routing delays but are outside the scope of this tutorial. In the Vivado IDE, the simulator GUI opens after successfully parsing and compiling the design (Figure 9).

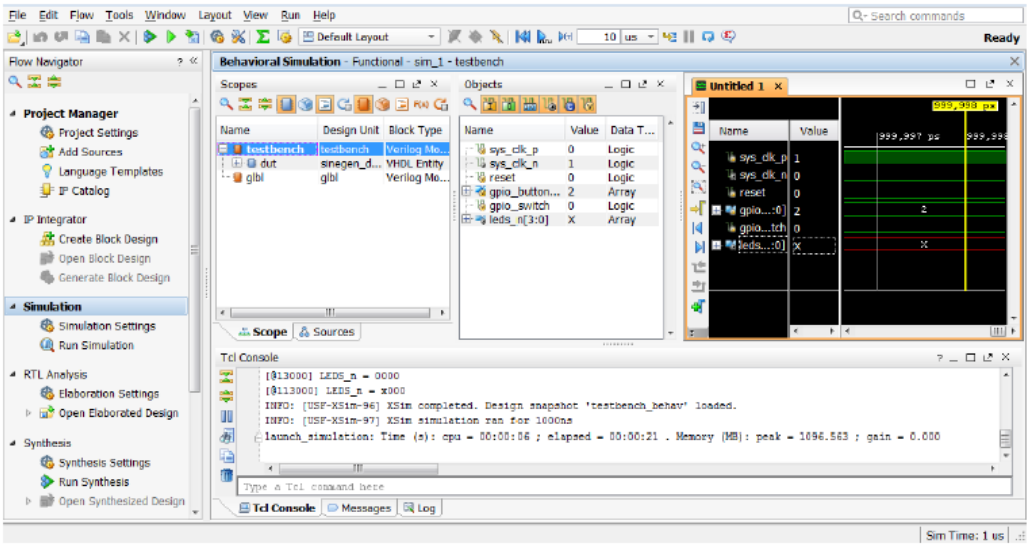


Figure 9. Vivado Simulation GUI

By default, the Vivado simulator adds **only the top-level HDL objects** display in the **Waveform window**. In the case of this tutorial, the following testbench signals load automatically:

- 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.
- Reset signal (reset). Provides control to reset the circuit.
- 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 placeholder bus to display the results of the simulation.

To observe the function of the circuit, you may also add some new signals to this list, as will be shown in the next section of this lab.

2-2. Add and Monitor Signals.

2-2-1. In the **Scopes window**, click the '+' sign to expand the testbench. (It might be expanded by default.).

An HDL scope, or scope, is defined by a declarative region in the HDL code, such as a module, function, task, process, or named blocks in Verilog. VHDL scopes include entity/architecture definitions, blocks, functions, procedures, and processes.

2-2-2. In the Scopes window, click to select the device under test (**dut)** object.

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, as shown in Figure 10.

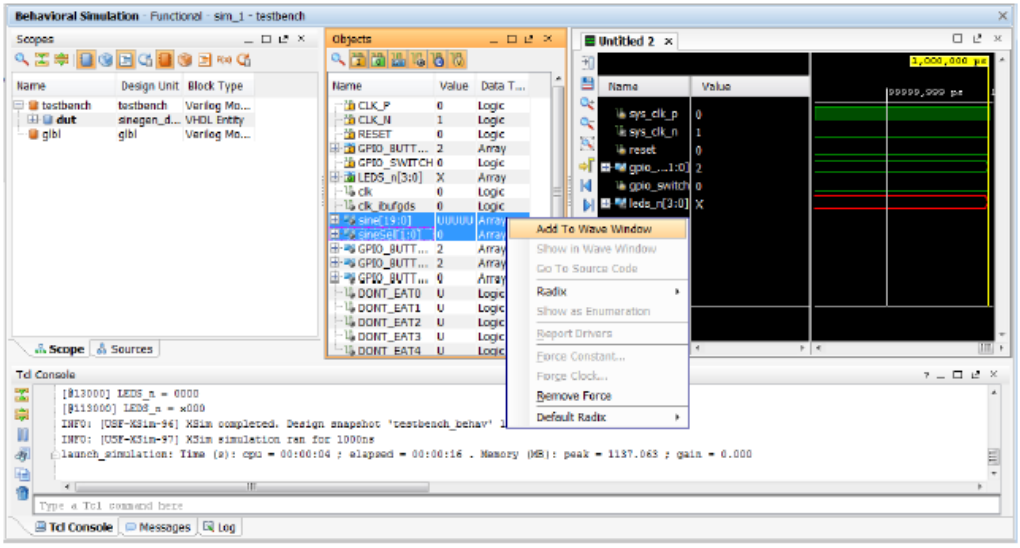


Figure 10. Add signals to Wave Window

2-2-3. From the Objects window, select signals sine[19:0] and sineSel[1:0] 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.

2-3. 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.

2-3-1. In the **Waveform window**, select the sine[19:0] signal.

2-3-2. Right click to open the popup menu, and select **Waveform Style > Analog** (Figure 11).

2-3-3. Right click to open the popup menu again, and select **Radix > Signed Decimal** also shown in Figure 11. Note that this is mandatory because sine wave samples and signed and represented in two's complement. If you do not select the signed radix, the binary information will not be correctly interpreted by the analog wave viewer.

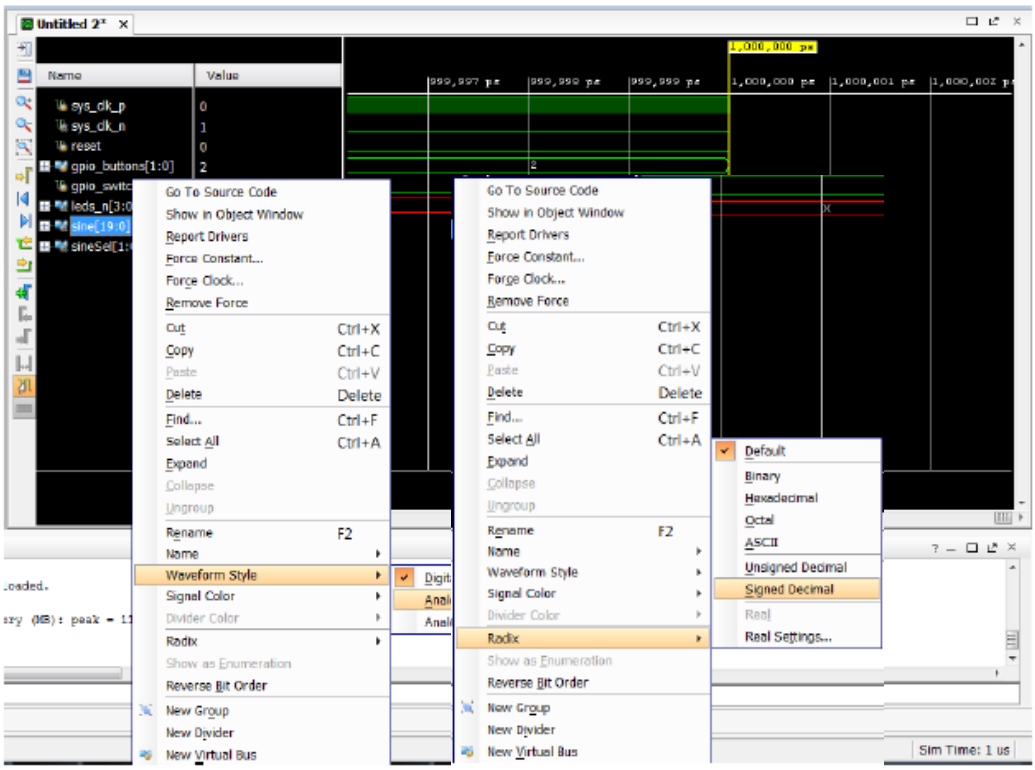


Figure 11. Enable Analog Waveform Style and Radix

Working with the Waveform Window — Step 3

3-1. Now that you have configured the simulator to display and log signals of interest into the waveform database, run the simulator again.

3-1-1. Run the simulation by clicking the **Run All** button. You can also Reset the simulation and Run for a specified time, using the buttons shown in Figure 12.

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

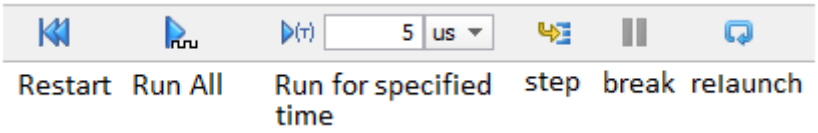


Figure 12. Control Simulation Buttons

3-1-2. Click the **Float** button in the top right corner of the Waveform Configuration window.

3-1-3. Display the whole 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 zoom into the waveform, you can use the horizontal and vertical scroll bars to pan down the full waveform.

As seen in Figure 13, when the value of sineSel is 0, which indicates a low frequency sine selection, the analog *sine[19:0]* output is not a proper sine wave, indicating a problem in the design or the testbench.

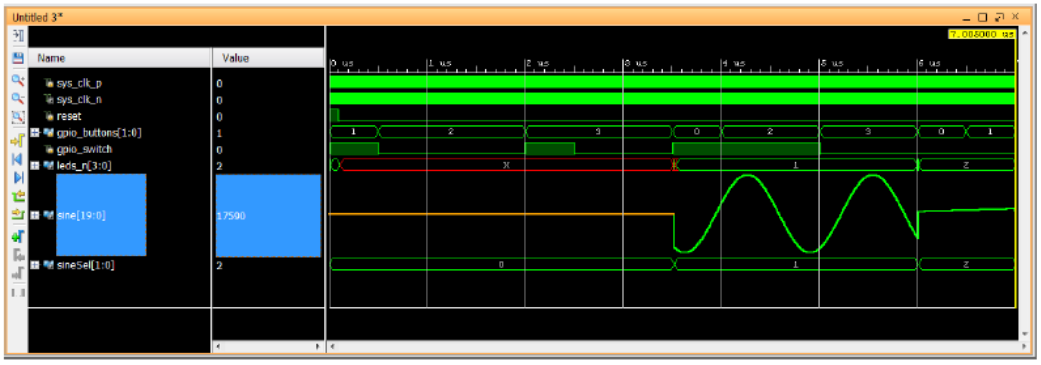


Figure 13. Design Bug - Wave View

3-2. Grouping signals and adding dividers.

Next, you 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.

3-2-1. In the **Waveform window**, select all signals in the testbench unit: sys_clk_p, sys_clk_n, reset, gpio_buttons, gpio_switch, and leds_n.

Note: Press and hold the **Ctrl** key, or **Shift** key, to select multiple signals.

3-2-2. With the signals selected right-click to open the popup menu and select **New Group**. Rename it as **TB Signals**. Vivado simulator creates a collapsed group in the waveform configuration window. To expand the group, click the '+' to the left of the group name

3-2-3. Create another signal group called **DUT Signals** to group signals sine[19:0] and sine_sel[1:0].

You can add 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.

3-2-4. 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.

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

3-2-5. 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.

3-2-6. Add two dividers named: **Testbench** and **SineGen**

3-2-7. Move the **SineGen** divider above the **DUT Signals** group.

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**.

3-3. 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.

3-3-1. In the Scopes window, select and expand the **Testbench**, then select and expand **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 14 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.

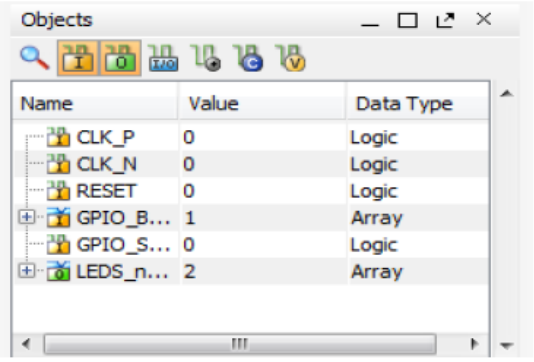


Figure 14. Object Filters

3-3-2. 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**.

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

3-3-4. In the Waveform window, right-click in the empty space below the signal names, and use the New Group command to create three new groups called **Inputs**, **Outputs**, and **Internal Signals**.

TIP: 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.

3-3-5. In the **Objects window**, select the Input filter to display the Input objects.

3-3-6. Select the **Input objects** in the Objects window, and drag and drop them onto the **Input group** you created in the Waveform window.

3-3-7. Repeat the two previous steps above to filter the **Output objects** and drag them onto the **Output group**, and filter the **Internal Signals** and drag them onto the **Internal Signals group**.

3-4. Changing Signal Properties of some of the signals shown in the Waveform window to better visualize the simulation results.

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.

3-4-1. In the Waveform window, hold **Ctrl** and click to select the sine[19:0] and sineSel[1:0] signals listed in the DUT signals group, under the SineGen divider.

3-4-2. Hold **Ctrl**, and click to select the sine[19:0] signals listed in the Outputs group, under the SineGen divider.

3-4-3. Right-click in the Waveform window to open the popup menu, and select the Name > Long command.

The displayed name changes to include the hierarchical path of the signal. You can now see that the sine[19:0] signals under the DUT Signals group refers to different objects in the design hierarchy than the sine[19:0] signals listed under the Outputs group. See Figure 15.

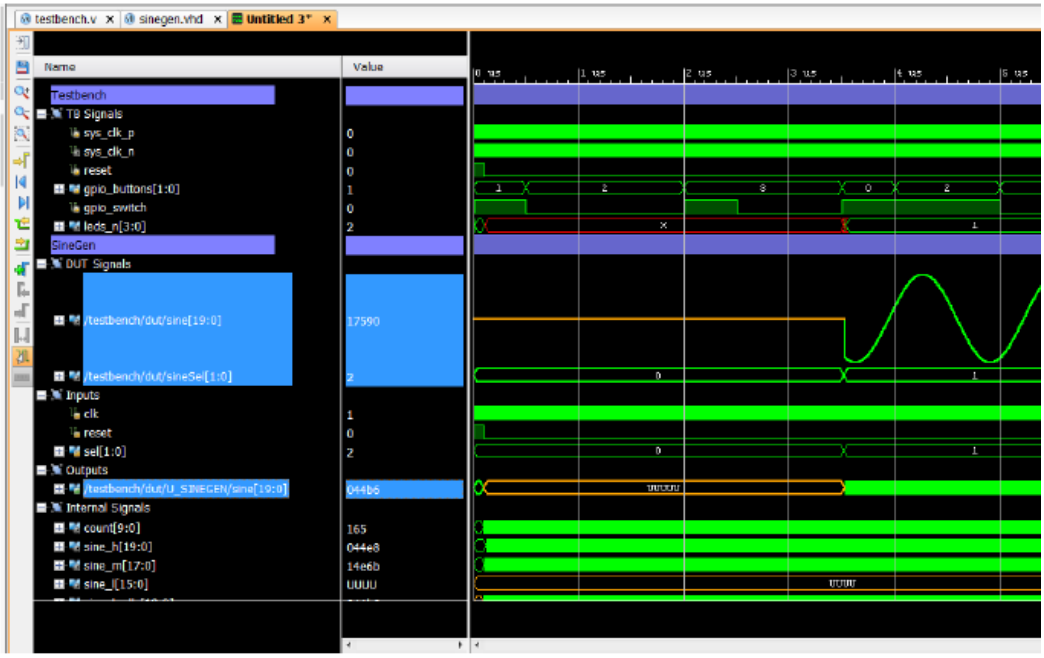


Figure 15. Long Signal Names

You can better understand some signal values if they display in a different radix format than the default, for instance, binary values instead of hexadecimal values. The default radix is Hexadecimal unless you override the radix for a specific object. Supported radix values are Binary, Hexadecimal, Octal, ASCII, Signed and Unsigned decimal. You can set any of the above values as Default using Default Radix option.

3-4-4. In the Waveform window, select the following signals:

- s_axis_phase_tdata_sine_high
- s_axis_phase_tdata_sine_mid
- s_axis_phase_tdata_sine_low

3-4-5. Right-click to open the popup menu, and select **Radix > Binary**. The values on these signals now display using the specified radix.

3-5. Saving the Waveform Configuration and re-simulating the design.

You can customize the look and feel of the Waveform window, and then save the Waveform configuration to reuse in future simulation runs. The Waveform configuration file defines the displayed signals, and the display characteristics of those signals.

3-5-1. In the Waveform window, click the **Options** button on the sidebar menu.  The Waveform Options dialog box opens to the **General** tab.

3-5-2. Ensure the **Default Radix** is set to **Hexadecimal**.

This defines the default number format for all signals in the Waveform window. The radix can also be set for individual objects in the Waveform window to override the default.

3-5-3. Select the **Draw Waveform Shadow**, as shown in Figure 16, 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.

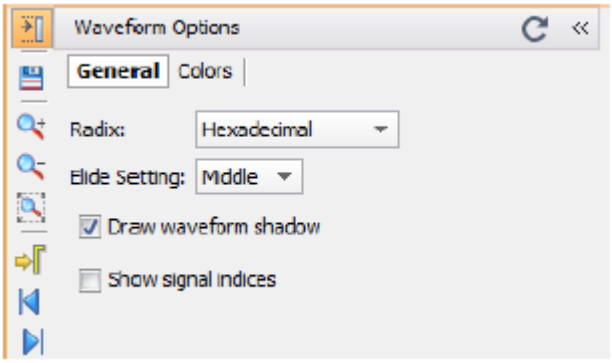


Figure 16. Waveform Options - General View

3-5-4. Check or uncheck the **Show signal indices** check box to enable or disable the signal list numbering.

3-5-5. 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. The Waveform window configures 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.

3-5-6. In the Waveform window sidebar menu, select the **Save Wave Configuration** button. 

3-5-7. Save the Wave Configuration into the project folder with the filename **lab3.wcfg**.

3-5-8. Click **Yes**. The file is added to the project simulation fileset, sim_1, for archive purposes.

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

3-5-9. Click the **Restart** button to reset the circuit to its initial state.

3-5-10. 2. Click the **Run All** button. The simulation runs for about 7005 ns. If you do not restart the simulator prior to executing the Run All command, the simulator runs continuously until interrupted.

3-5-11. After the simulation is complete, click the **Zoom Fit** button to see the whole simulation timeline in the Waveform window. Figure 17 shows the current simulation results.

Figure 17. Simulation Waveform at Time 705 ns

3-6. Using Cursors, Markers, and Measuring Time.

The Finite State Machine (U_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.

3-6-1. In the **Waveform window** select the /testbench/dut/sineSel[1:0] signal, as shown in Figure 18.

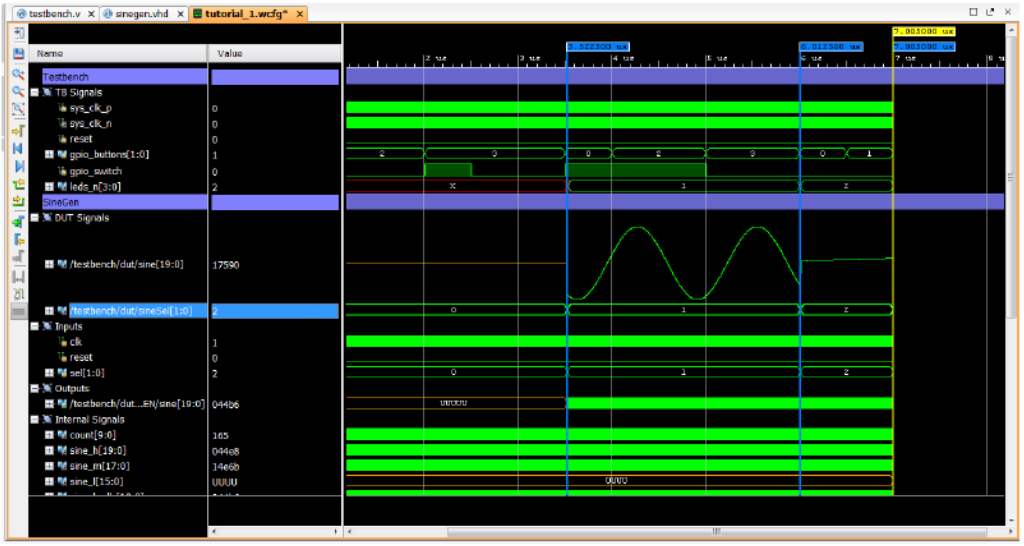


Figure 18. Using Markers

3-6-2. In the waveform sidebar menu, click the **Go to Time 0** button  .The current marker moves to the start of the simulation run.

3-6-3. Enable the **Snap to Transition** button to snap the cursor to transition edges. 

3-6-4. From the waveform sidebar menu, click the **Next Transition** button.  The current marker moves to the first value change of the selected sineSel[1:0] signal, at 3.5225 microseconds.

3-6-5. Click the **Add Marker** button. 

3-6-6. Search for all transitions on the sineSel signal, and add markers at each one.

With markers identifying the transitions on sineSel, the Waveform window should look similar to Figure 18. As previously observed, the low frequency signals are incorrect when the sinSel signal value is 0.

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 sineSel is 0 to review the status of the output signal, sine[19:0], and identify where the incorrect behavior initiates. You also use the cursor to measure the period of low frequency wave control.

TIP: 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.

3-6-7. In the Waveform window, click the **Go to Time 0** option, then click the **Zoom in** button repeatedly to zoom into the beginning of the simulation run.

3-6-8. Continue to zoom in the Waveform window as needed, until you can see the reset signal asserted low, and you can see the waveform of the clock signals, sys_clk_p and sys_clk_n, as seen in Figure 19. The Waveform window zooms in or out around the area centered on the cursor.

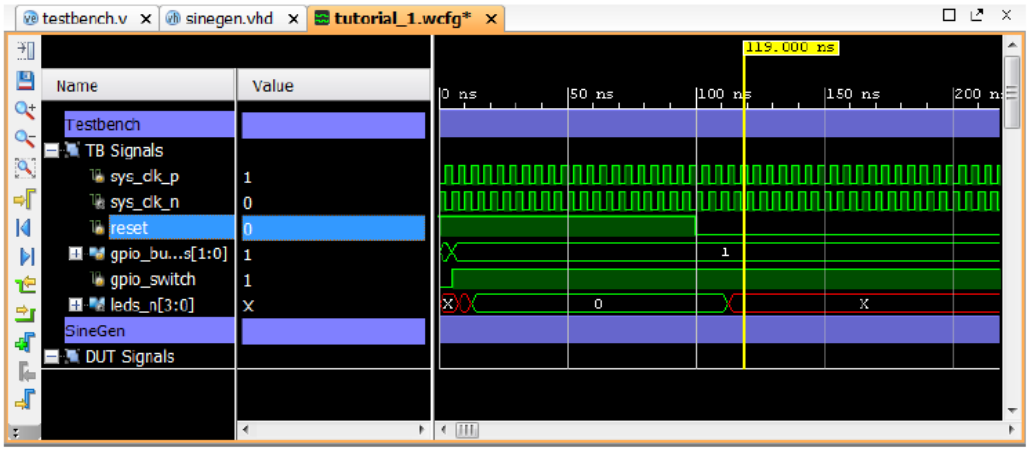


Figure 19. Viewing Reset and Clock Signals

3-6-9. Place the main Waveform window cursor on the area by clicking at a specific time or point in the waveform. You can also click on the main cursor, and drag it to the desired time.

3-6-10. Because 0 is the initial or default FSM output, move the cursor to the first posedge of sys_clk_p after reset is asserted low, at time 102.5 ns, as seen in Figure 20. You can use the Waveform window to measure time between two points on the timeline.

3-6-11. Place a marker at the time of interest, 102.5 ns, by clicking the **Add Marker** button. 

3-6-12. Click to select the marker.

The Floating Ruler button displays a ruler  at the bottom of the Waveform window useful for measuring time between two points. Use the floating ruler to measure the sineSel control signal period, and the corresponding output_sine[19:0] values during this time frame.

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 selected marker. As you move the cursor along the timeline, the ruler measures the time difference between the cursor and the marker.

You can move the cursor along 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.

3-6-13. 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 20, the ruler measures a time period of 3.420 ns as the period that FSM selected the low frequency output.

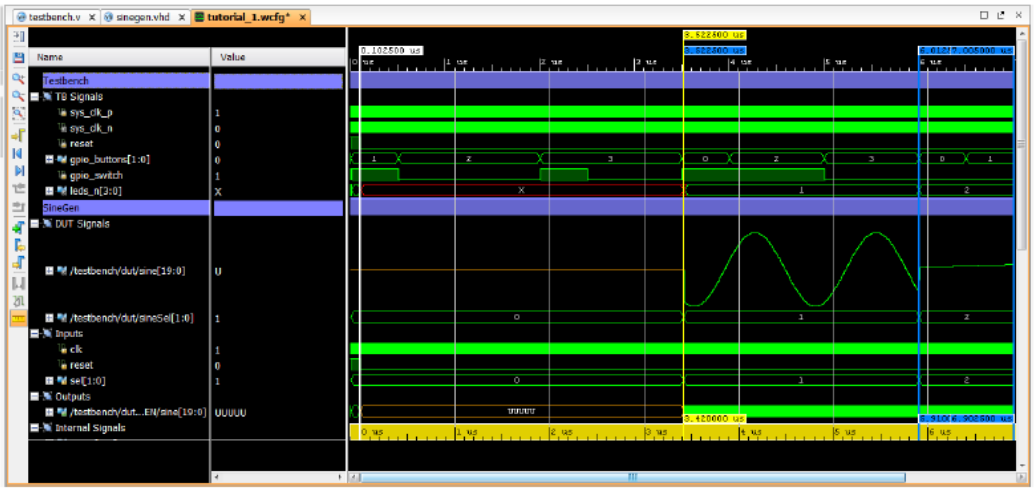


Figure 20. Measuring Time in the Waveform

Debugging with Breakpoints — Step 4

4-1. Read the previously saved checkpoint (checkpoint_1) in order to analyze the results without going through the actual synthesis process.

4-1-1. First, open the tutorial design testbench to learn how the simulator generates each design input.

4-1-2. Open the testbench.v file by double-clicking the file in the Sources window, if it is not already open. The source file opens in the Vivado IDE Text Editor, as shown in Figure 21.

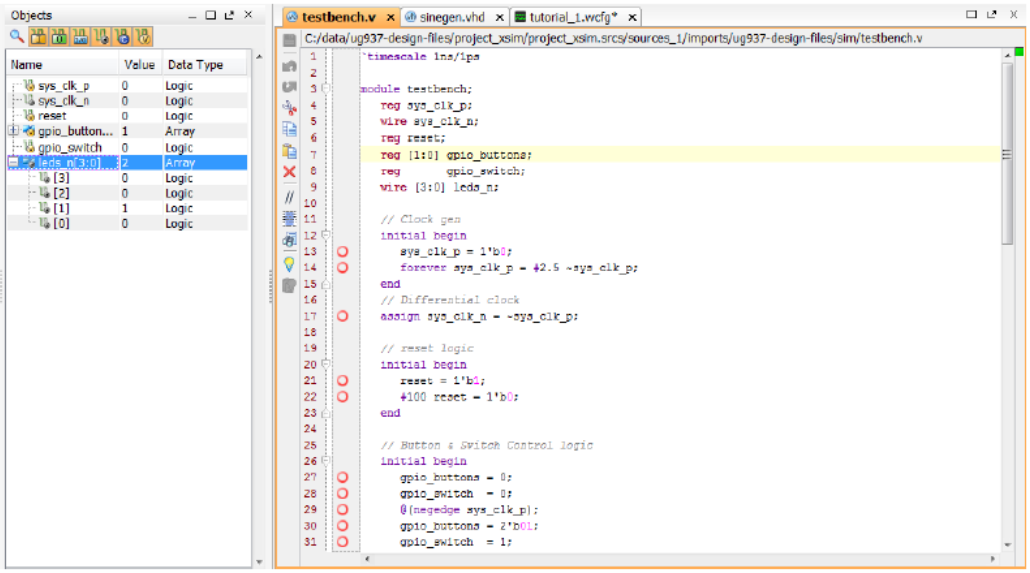


Figure 21. Integrated Text Editor

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 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.

4-1-3. Select the **U_SINEGEN** scope in the Scopes window to list the objects of that scope in the Objects window.

4-1-4. In the Objects window, right-click sine[19:0] and use **Go to Source Code** to open the sinegen.vhd source file in the Text Editor.

TIP: If you do not see the sine[19:0] signal in the Objects window, make sure that the filters at the top of the Objects window are set properly to include Output objects.

4-1-5. Looking through the HDL code, the clk, reset, and sel inputs are correct as expected. Scroll to line 137 in the file and set your first breakpoint after the reset asserts low (at line 137).

Note that the breakpoint can be set only on the executable lines. Vivado simulator marks the executable lines with an empty red circle, on the left hand margin of the Text Editor, beside 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.

4-1-6. Click the red circle in the left margin, to set a breakpoint, as shown in Figure 22. Observe that the empty circle becomes a red dot to indicate that a breakpoint is set on this line. Clicking on the red dot removes the breakpoint and reverts it to the empty circle

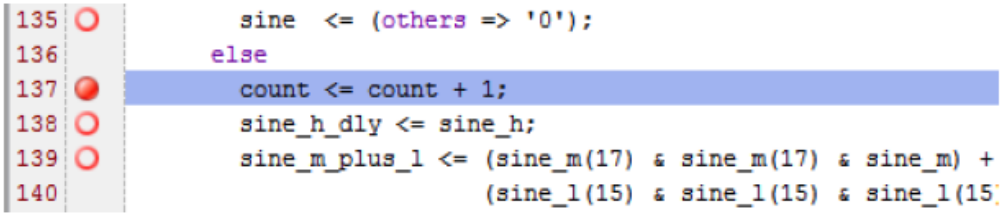


Figure 22. Setting a Breakpoint

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

4-1-7. 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, as shown in Figure 23. So, resize the windows, and use the **window Float** command or the New Vertical Group command to arrange the various windows so that you can see them all.

4-1-8. Click the **Restart** button to restart the simulation from time 0.

4-1-9. Run the simulation by clicking the **Run All** button.

4-1-10. The simulation runs to time 102.5 ns, 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 highlights the line. A message also displays in the Tcl console to indicate that the simulator has stopped at a specific time, displayed in picoseconds, indicating the line of source code last executed by the

simulator.

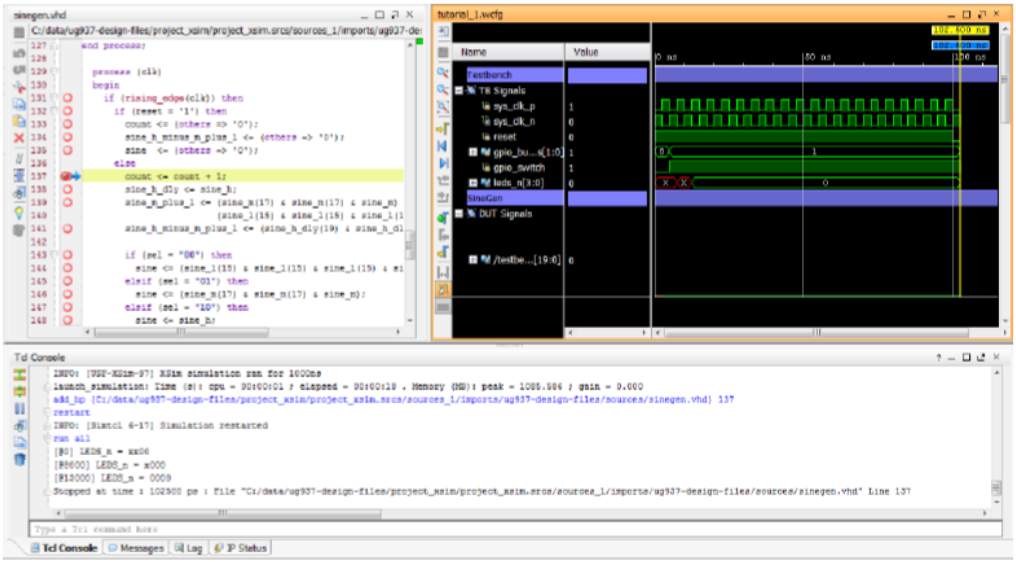


Figure 23. Arrange Windows for Debugging

4-1-11. 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.

4-1-12. 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 0. This code assigns, with bit extension, the low frequency signal, sine_l, to the output, sine.

4-1-13. In the Waveform window, select sine_l[15:0] in the Internal Signals group, and holding Ctrl, select sine[19:0] in the Outputs group. These selected signals are highlighted in the Waveform window, making them easier for you to monitor.

4-1-14. Run the simulation by clicking **the Run All** button. Once again, the simulation stops at the breakpoint, this time at line 144.

4-2. 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 (line, process, task) at a 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.

4-2-1. On the Vivado simulator toolbar menu, 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.

4-2-2. 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 0, signal sine_l is assigned as a low frequency sine wave to the output sine. Also, notice that sine_l remains uninitialized.

4-2-3. For debug purposes, initialize the value of sine_I by entering the following add_force command in the Tcl console:

add_force /testbench/dut/U_SINEGEN/sine_I 0110011011001010

This command forces the value of sine_I into a specific known condition, and can provide a repeating set of values to exercise the signal more vigorously, if needed.

4-2-4. Continue the simulation by clicking the Run All button a few more times.

In the Waveform window, notice that the value of sine_I[15:0] is now set to the value specified by the add_force command, and this value is assigned to the output signal sine[19:0] since the value of sel is still 0.

4-2-5. Trace the sine_I signal in the HDL source files, and identify the input for sine_I.

4-2-6. In the Text Editor, right-click to open the popup menu, and select the Find in files button  to search for sine_I.

4-2-7. Select the Match whole word and Enabled design sources checkboxes, as shown in Figure 24, and click Find. The Find in Files results display at the bottom of the Vivado IDE, with all occurrences of sine_I found in the sinegen.vhd file.

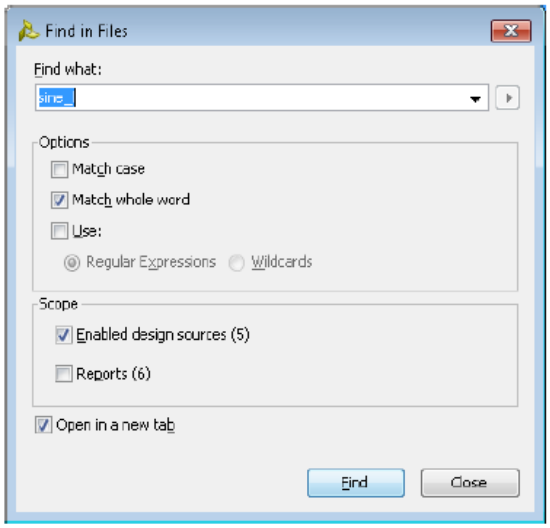


Figure 24. Find in Files

4-2-8. Expand the Find in Files results to view the results in the sinegen.vhd file.

The second result, on line 111, identifies a problem with the design. At line 111 in the sinegen.vhd file, the m_axis_data_tdata_sine_low signal is assigned to sine_I. Since line 111 is commented out, the sine_I signal is not connected to the low frequency DDS compiler output, or any other input.

4-2-9. Uncomment line 111 in the sinegen.vhd file, and click the **Save File** button.

4-2-10. In the Tcl Console, remove the force on sine_I: **remove_forces -all.**

4-3. Relaunch Simulation

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

4-3-1. In sinegen.vhd, select one of the breakpoints, right-click and select **Delete All Breakpoints**.

4-3-2. Click the **Relaunch button** on the main toolbar menu. If prompted to save the Wave Config file, click **yes**.

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. The relaunch button will be active only after one successful run of Vivado Simulator using launch_simulation.

4-3-3. Click the **Run All** button (Figure 25) to run the simulation. Observe the sine[19:0], the analog signal in the waveform configuration. The low frequency sine wave looks as expected.

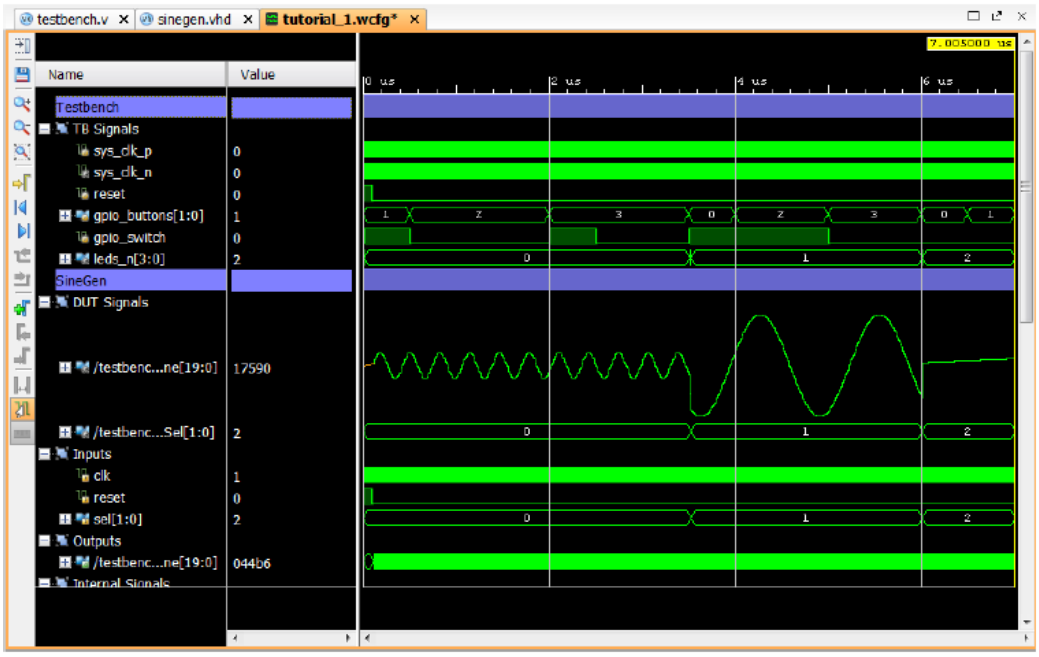


Figure 25. Corrected Low Frequency Output

Conclusion

After reviewing the simulation results, you may close the simulation, and close the project. This completes Lab 3. In this lab you have:

- Run the Vivado simulator using the Project Mode flow in Vivado IDE
- Created a project, added source files, and added IP
- Added a simulation-only file (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