Simulation Tutorial

Acknowledgement

This module is derived from a Xilinx lab on Xilinx website.

Goal

- Use Vivado simulator to perform behavioral, functional, or timing simulations
- Be able to use a C program in SDK to interact with the processor.
- Use some software debugging tools in an embedded processor environment.

Requirements

- Xilinx Vivado software
- Enough disk space for the project files

Source Files

For the purpose of this tutorial, all source files are provided

1. Create a Project

1. Choose **RTL Project** in Project Type box

2. Click **Add Directories** in Add Source dialog box and add the following

    - <Extract_Dir>/sources

    - <Extract_Dir>/sim

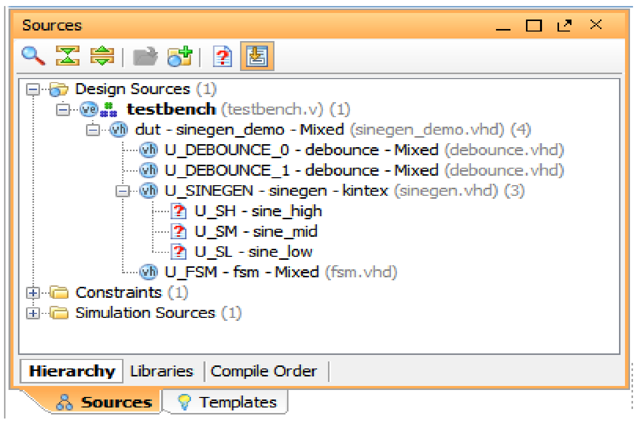
3. Set the Target Language to **Verilog** and Simulator Language to **Mixed**

4. Choose Nexys 4 board

5. Finish creating the projects

2. Adding IP from the IP Catalog

1. Click the '+' character in the Sources window to expand the folers as shown below



```
Design Sources (1)
  testbench (testbench.v) (1)
    dut - sinegen_demo - Mixed (sinegen_demo.vhd) (4)
      U_DEBOUNCE_0 - debounce - Mixed (debounce.vhd)
      U_DEBOUNCE_1 - debounce - Mixed (debounce.vhd)
      U_SINEGEN - sinegen - kintex (sinegen.vhd) (3)
        U_SH - sine_high
        U_SM - sine_mid
        U_SL - sine_low
      U_FSM - fsm - Mixed (fsm.vhd)
Constraints (1)
Simulation Sources (1)
```

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.

You will now add the sine_high, sine_mid, and sine_low modules to the project from the

Xilinx IP Catalog.

Adding Sine High

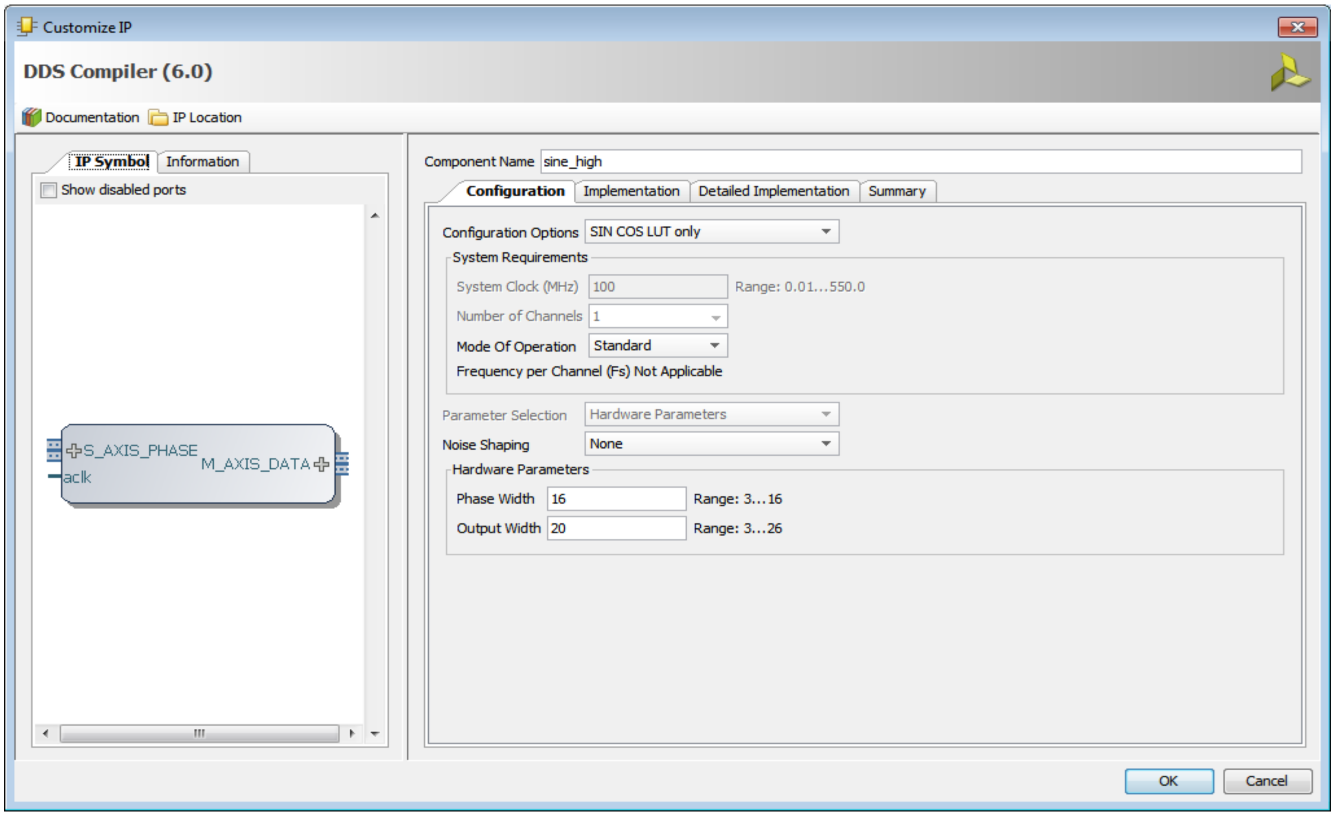
1. In the Flow Navigator, select the IP Catalog button.

2. In the search field of the IP Catalog, type DDS.

The Vivado IDE highlights the DDS Compilers in the IP catalog.

3. Under any category, double-click the DDS Compiler.

The Customize IP wizard opens.



Component Name sine_high

Configuration Options SIN COS LUT only

System Requirements

System Clock (MHz) 100 Range: 0.01...550.0

Number of Channels 1

Mode Of Operation Standard

Frequency per Channel (Fs) Not Applicable

Parameter Selection Hardware Parameters

Noise Shaping None

Hardware Parameters

Phase Width 16 Range: 3...16

Output Width 20 Range: 3...26

4. In the IP Symbol on the left, ensure that Show Disabled Ports is unchecked.

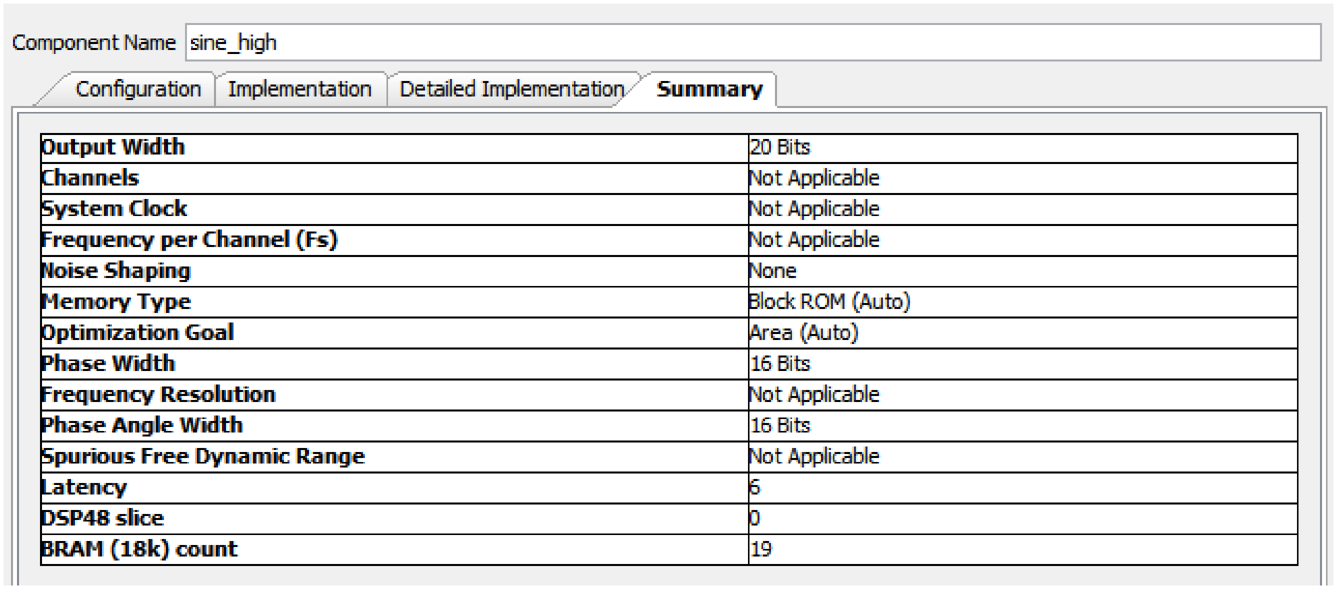
5. Specify the following on the Configuration tab:

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

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

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

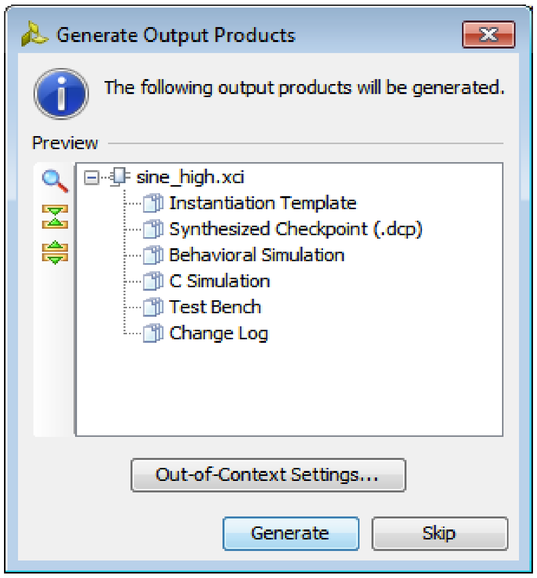
8. Select the Summary tab, review the settings and click OK.



Component Name: sine_high

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

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 below.



The following output products will be generated.

Preview

```
sine_high.xci
    Instantiation Template
    Synthesized Checkpoint (.dcp)
    Behavioral Simulation
    C Simulation
    Test Bench
    Change Log
```

9. ClickGenerate to generate the default output products for sine_high.

Adding Sine Mid

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

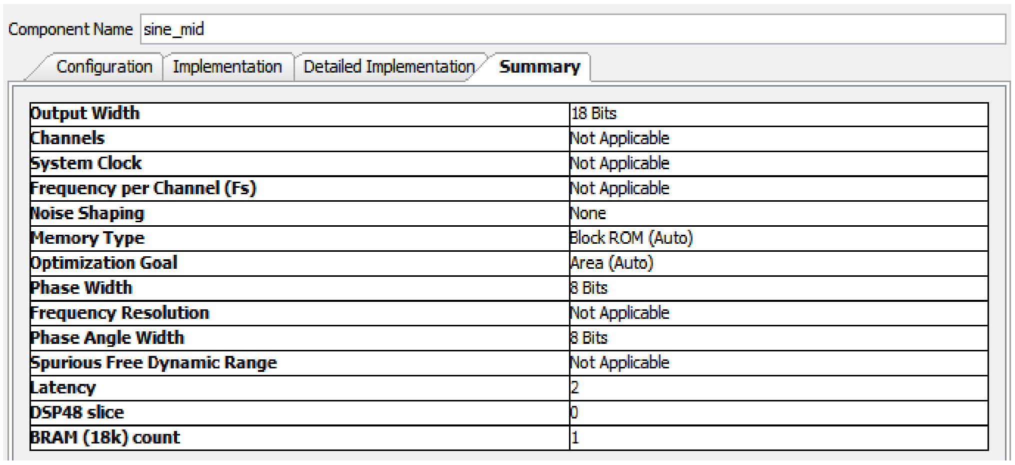
2. Specify the following on the Configuration tab:

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

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

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

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



Component Name: sine_mid

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

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.

6. Click Generate to generate the default output products for sine_mid.

Adding Sine Low

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

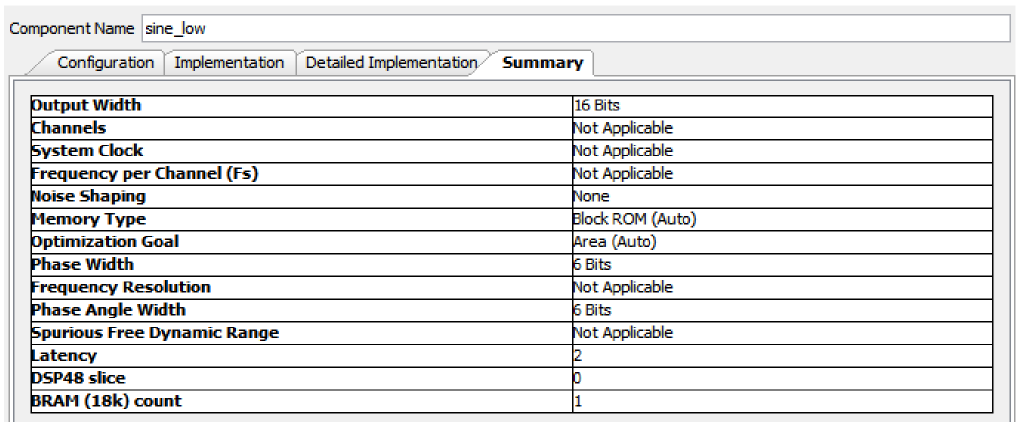
2. Specify the following on the Configuration tab:

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

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

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

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



Component Name: sine_low

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

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.

6. Click Generate to generate the default output products for sine_low.

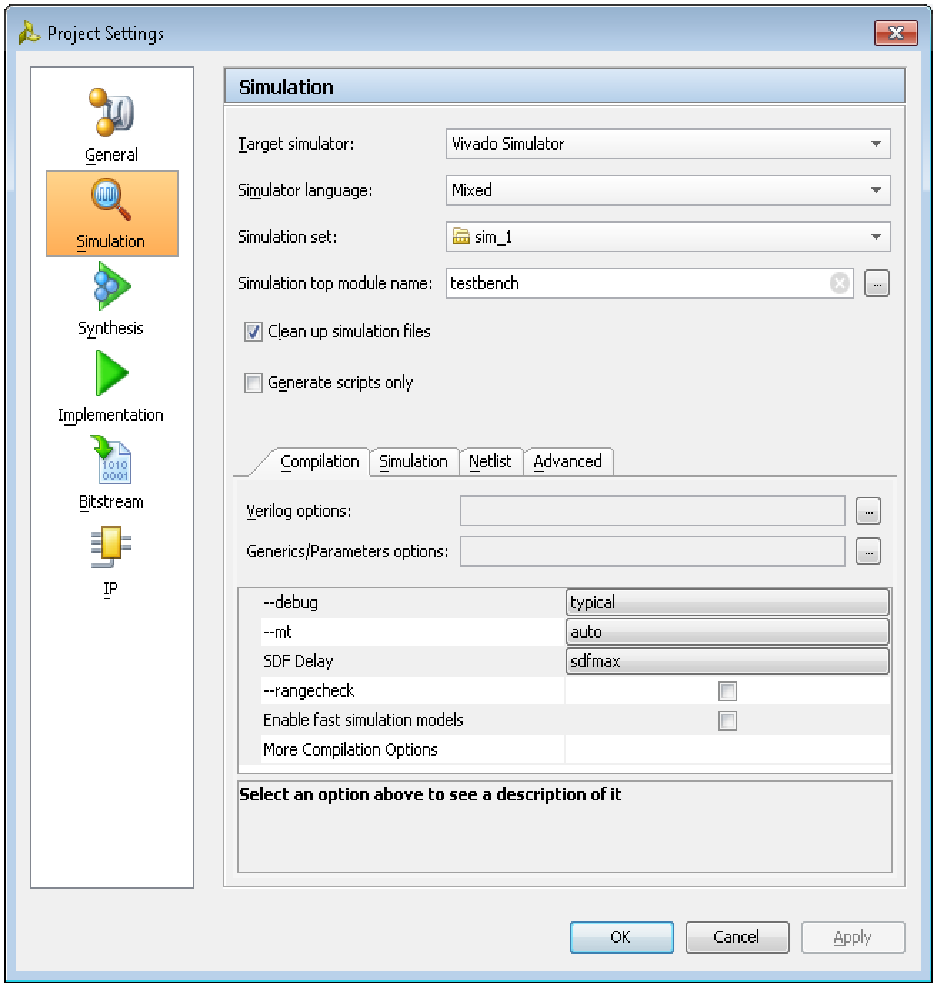
3: Running Behavioral Simulation

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

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

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

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



Target simulator: Vivado Simulator
Simulator language: Mixed
Simulation set: sim_1
Simulation top module name: testbench
[✓] Clean up simulation files
[ ] Generate scripts only

Verilog options:
Generics/Parameters options:

--debug	typical
--mt	auto
SDF Delay	sdfmax
--rangecheck	☐
Enable fast simulation models	☐
More Compilation Options	

Select an option above to see a description of it

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

4. Click OK.

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

5. In the Flow Navigator, click Run Simulation >Run Behavioral Simulation.

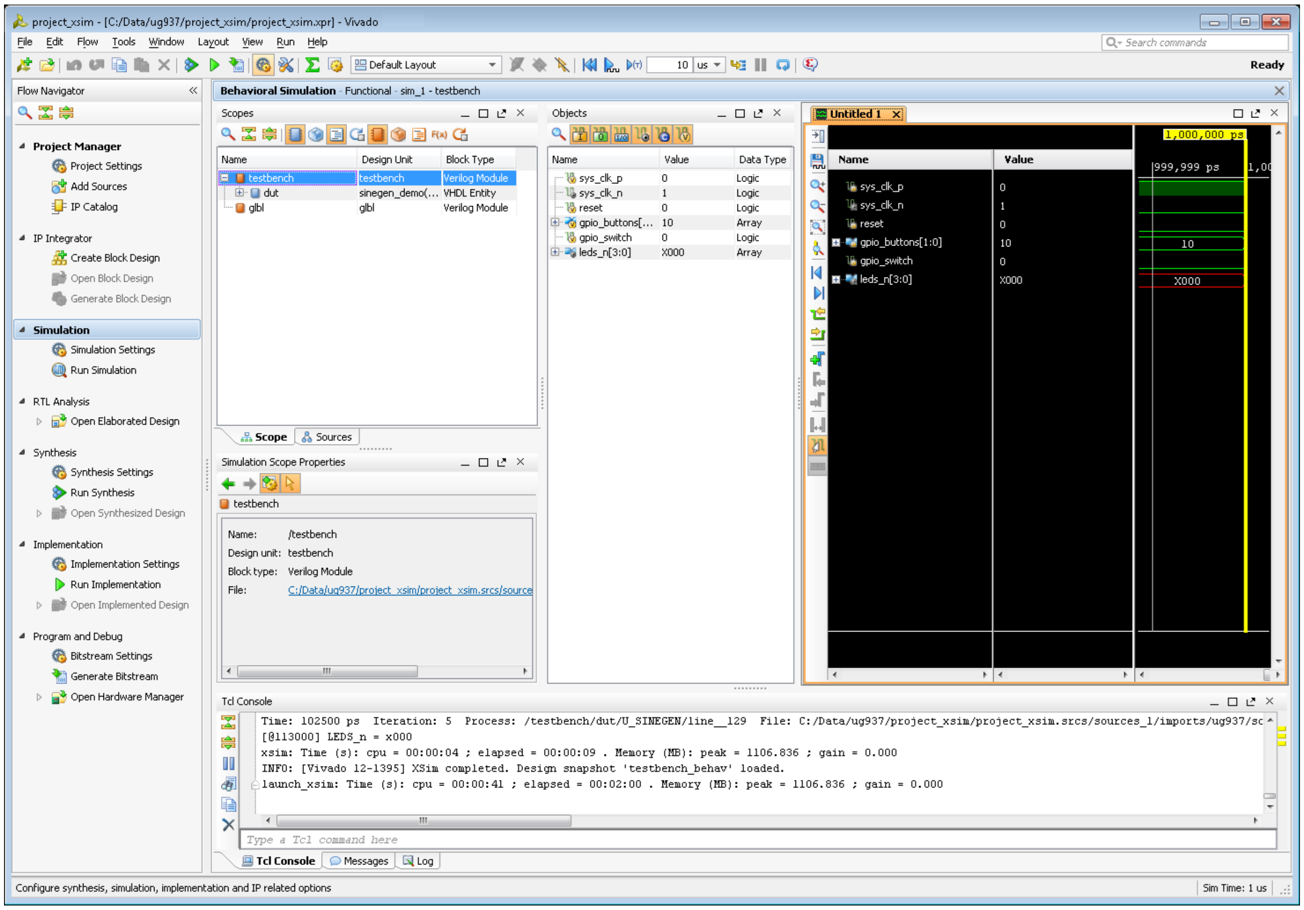
Functional and timing simulations are available post-synthesis and post-implementation.

Those simulations are outside the scope of this tutorial.

When you launch the Run Behavioral Simulation command, the Vivado tool runs xelab in

the background to elaborate and compile the design into a simulation snapshot, which the Vivado simulator can run. When that process is complete, the Vivado tool launches xsim to run the simulation.

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



Scopes:

Name	Design Unit	Block Type
testbench	testbench	Verilog Module
dut	sinegen_demo(…	VHDL Entity
glbl	glbl	Verilog Module

Objects:

Name	Value	Data Type
sys_clk_p	0	Logic
sys_clk_n	1	Logic
reset	0	Logic
gpio_buttons[…	10	Array
gpio_switch	0	Logic
leds_n[3:0]	X000	Array

Simulation Scope Properties: testbench

Name: /testbench
Design unit: testbench
Block type: Verilog Module
File: C:/Data/ug937/project_xsim/project_xsim.srcs/source

Waveform (Untitled 1): 1,000,000 ps; 999,999 ps; 1,00

Name	Value
sys_clk_p	0
sys_clk_n	1
reset	0
gpio_buttons[1:0]	10
gpio_switch	0
leds_n[3:0]	X000

Tcl Console:

```
Time: 102500 ps  Iteration: 5  Process: /testbench/dut/U_SINEGEN/line__129  File: C:/Data/ug937/project_xsim/project_xsim.srcs/sources_1/imports/ug937/sc
[@113000] LEDS_n = x000
xsim: Time (s): cpu = 00:00:04 ; elapsed = 00:00:09 . Memory (MB): peak = 1106.836 ; gain = 0.000
INFO: [Vivado 12-1395] XSim completed. Design snapshot 'testbench_behav' loaded.
launch_xsim: Time (s): cpu = 00:00:41 ; elapsed = 00:02:00 . Memory (MB): peak = 1106.836 ; gain = 0.000
```

Configure synthesis, simulation, implementation and IP related options

Sim Time: 1 us

Debugging the Project

1: Opening the Project

1. click File > Open Recent Project and select the project_xsim project you saved in Lab #1.

2. After the project has opened, from the Flow Navigator click Run Simulation > Run Behavioral Simulation.

The Vivado simulator compiles your design and loads the simulation snapshot.

2: Displaying Signal Waveforms

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

• Running and restarting the simulation to review the design functionality, using signals in the Waveform window, and messages from the testbench shown in the Tcl console.

• Adding signals from the testbench and other design units to the Waveform window so you can monitor their status.

• Adding groups and dividers to better identify signals in the Waveform window.

• Changing signal and wave properties to better interpret and review the signals in the Waveform window.

• Using markers and cursors to highlight key events in the simulation and to perform zoom and time measurement features.

• Using multiple waveform configurations.

Add and Monitor Signals

The focus of the tutorial design is to generate sine waves with different frequencies. To observe the function of the circuit, you will monitor a few signals from the design. Before running simulation for a specified time, you can add signals to the wave window to observe the signals as they transition to different states over the course of the simulation. By default, the Vivado simulator adds simulation objects from the testbench to 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 place holder bus to display the results of the simulation. You will add some new signals to this list to monitor those signals as well.

1. In the Scopes window, click the ‘+’ sign to expand the testbench.

An HDL scope, or scope, is defined by a declarative region in the HDL code such as a module, function, task, process, or begin-end blocks in Verilog. VHDL scopes include entity/architecture definitions, block, function, procedure, and process blocks.

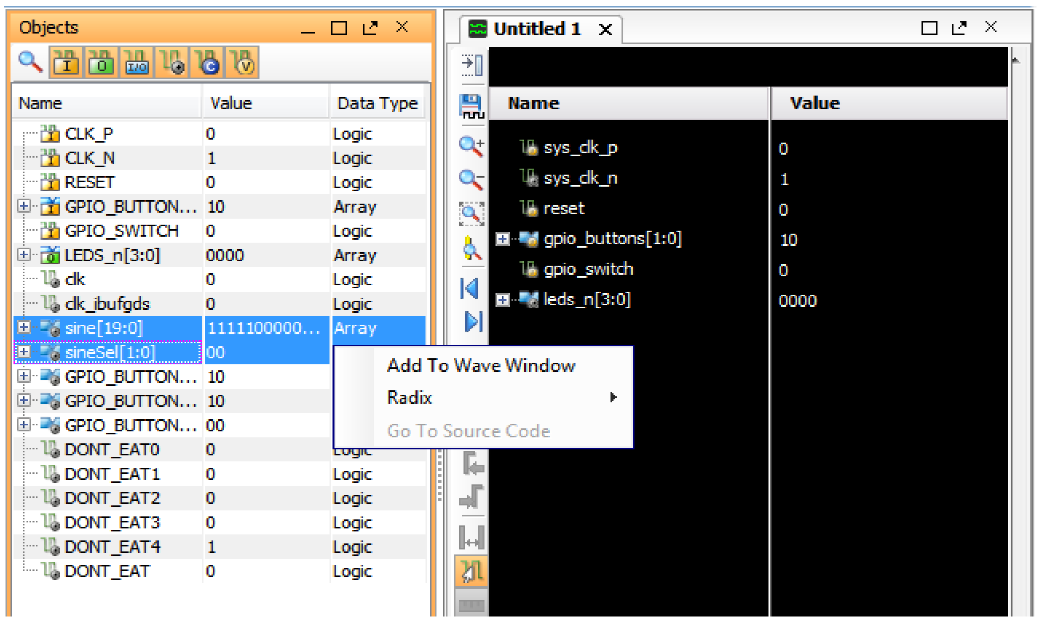
2. In the Scopes window, click to select the dut object. The current scope of the simulation changes from the whole testbench to the selected object. The Objects window updates with all the signals and constants of the selected scope.

3. From the Objects window, select signals sineSel[1:0] and sine[19:0] and add them into Wave Configuration window using one of the following methods:

• Drag and drop the selected signals into the Waveform window.

• Right-click on the signal to open the popup menu, and select Add to Wave Window.

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



Name	Value	Data Type
CLK_P	0	Logic
CLK_N	1	Logic
RESET	0	Logic
GPIO_BUTTON...	10	Array
GPIO_SWITCH	0	Logic
LEDS_n[3:0]	0000	Array
clk	0	Logic
clk_ibufgds	0	Logic
sine[19:0]	1111100000...	Array
sineSel[1:0]	00	
GPIO_BUTTON...	10	
GPIO_BUTTON...	10	
GPIO_BUTTON...	00	
DONT_EAT0	0	Logic
DONT_EAT1	0	Logic
DONT_EAT2	0	Logic
DONT_EAT3	0	Logic
DONT_EAT4	1	Logic
DONT_EAT	0	Logic

Name	Value
sys_clk_p	0
sys_clk_n	1
reset	0
gpio_buttons[1:0]	10
gpio_switch	0
leds_n[3:0]	0000

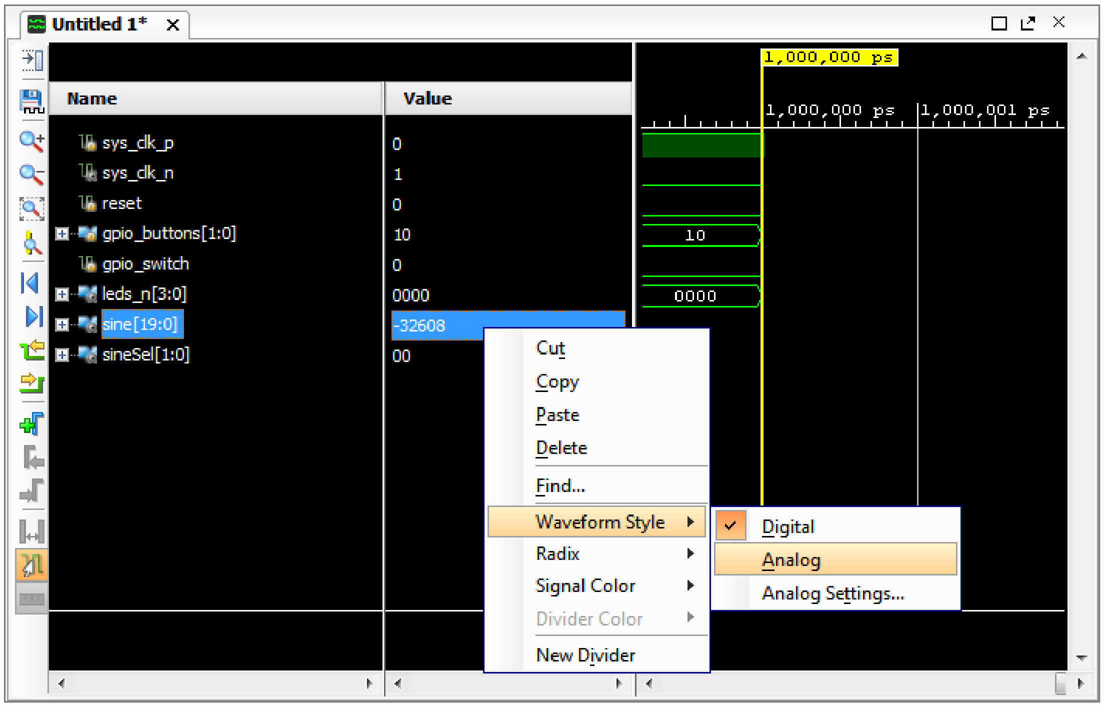
Step 3: Using the Analog Wave Viewer

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

1. In the Waveform window, select the sine[19:0] signal.

2. Right click to open the popup menu, and select Waveform Style > Analog.

3. Right click to open the popup menu again, and set Radix > Signed Decimal.



Name	Value
sys_clk_p	0
sys_clk_n	1
reset	0
gpio_buttons[1:0]	10
gpio_switch	0
leds_n[3:0]	0000
sine[19:0]	-32608
sineSel[1:0]	00

Logging Waveforms for Debugging

The Waveform window lets you review the state of multiple signals as the simulation runs. However, due to its limited size, the number of signals you can effectively monitor in the Waveform window is limited. To identify design failures during debugging, you might need to trace more signals and objects than can be practically displayed in the Waveform window. You can log the waveforms for signals that are not displayed in the Waveform window, by writing them to the simulation waveform database (WDB). After simulation, you can review the transitions on all signals captured in the waveform database file.

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

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

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

The log_wave command writes the specified signals to a waveform database, which is written to the simulation folder of the current project:

<project_name>/<project_name>.sim/sim_1/behav

4: Working with the Waveform Window

Now that you have configured the simulator to display and to log signals of interest in the waveform database, you are ready to run the simulator again.

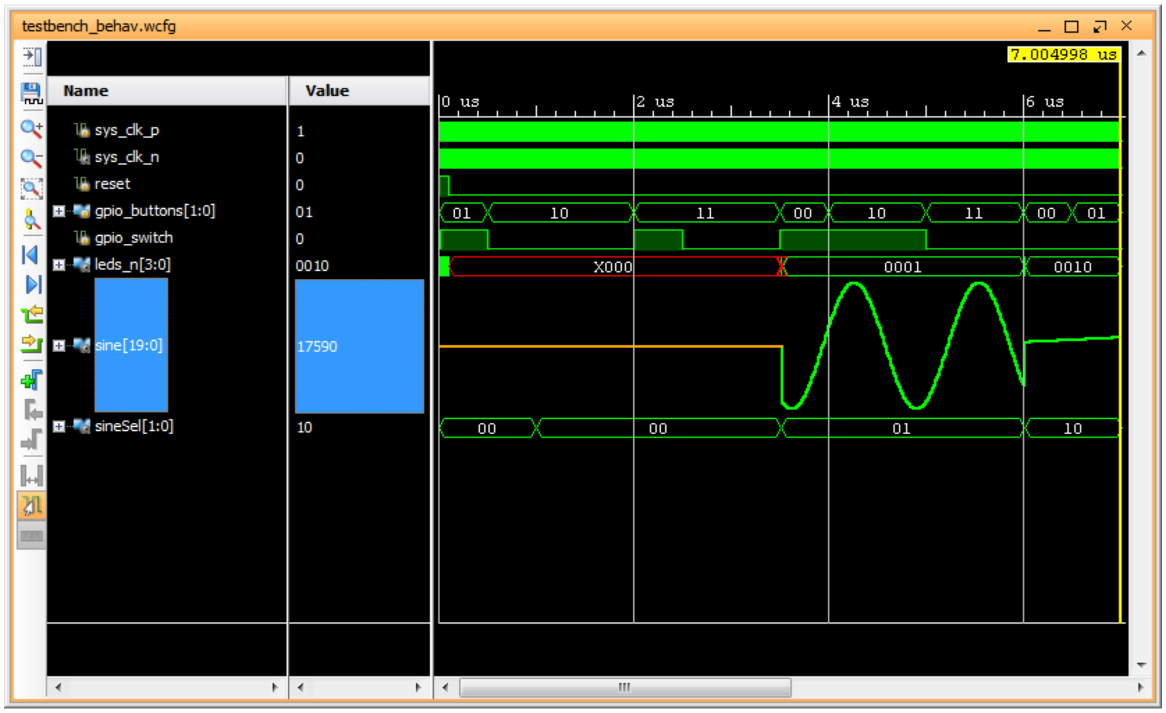
1. Run the simulation by clicking the Run All button.

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

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

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 are zoomed into the waveform, you can use the horizontal and vertical scroll bars to pan down the full waveform.



When the value of sineSel is 00, 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 testbench.

Grouping Signals

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

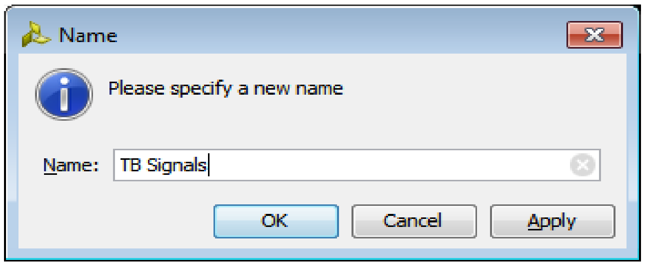
You can group related signals together to make viewing them easier. With a group, you can display or hide associated signals to make the Waveform window less cluttered, and easier to understand.

1. In the Waveform window, 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.

2. With the signals selected right-click to open the popup menu and select New Group. The Name dialog box opens to let you specify the name of the signal group to create.

3. Type TB Signals as the name of this signal group, and click OK.



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

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

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

You can ungroup all signals, thereby eliminating the group. Select a group, right-click to open the popup menu, and select Ungroup. To better visualize which signals belong to which design units, add dividers to separate the signals by design unit.

Adding Dividers

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

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

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

2. Add two dividers named:

- Testbench
- SineGen

3. Click and drag the Testbench divider above the TB Signals group.

4. Move the SineGen divider above the DUT Signals group.

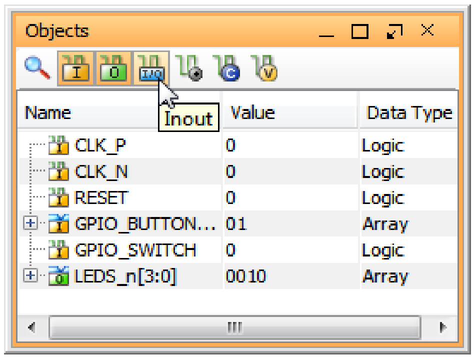
Adding Signals from Sub-modules

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

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

1. In the Scopes window, select and expand the testbench, then select 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. The figure below 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.



Name	Value	Data Type
CLK_P	0	Logic
CLK_N	0	Logic
RESET	0	Logic
GPIO_BUTTON...	01	Array
GPIO_SWITCH	0	Logic
LEDS_n[3:0]	0010	Array

2. Use the Objects window toolbar to enable and disable the different object types.

The types of objects that can be filtered in the Objects window include Input, Output, Inout ports, Internal Signals, Constants, and Variables.

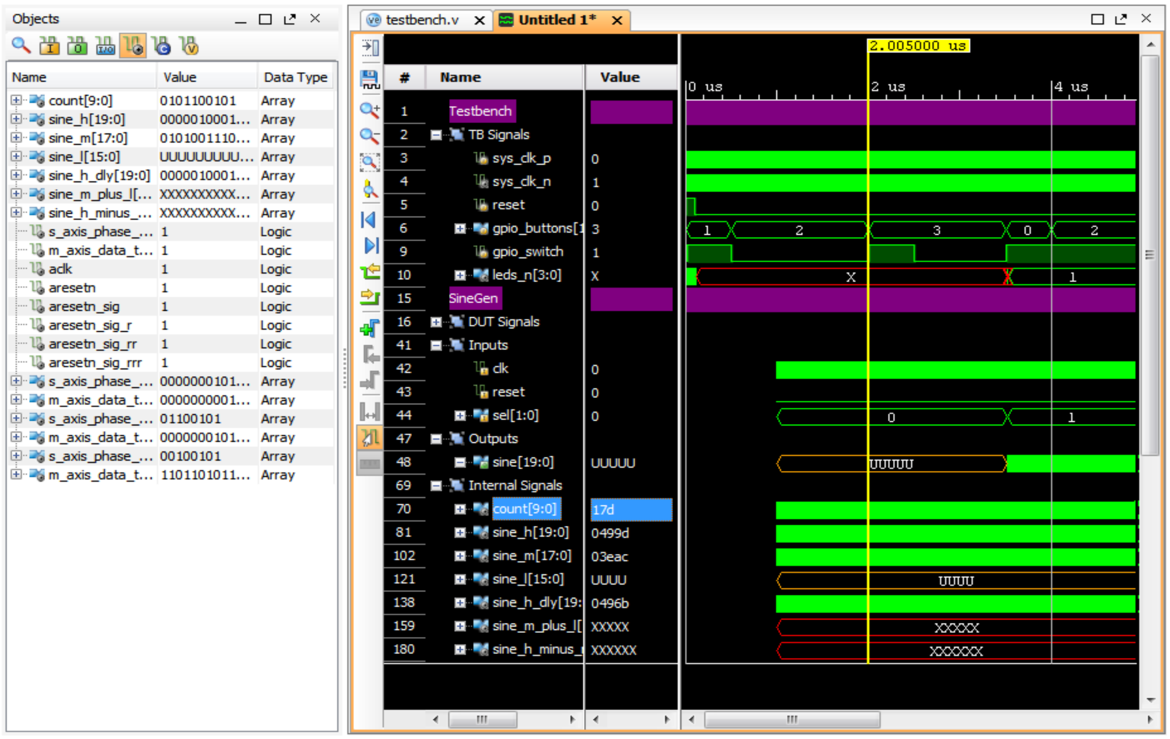
3. In the Scopes window, select the U_SINEGEN design unit.

4. In the Waveform window, right-click beneath the SineGen divider, and use the New Group command to create three new groups called Inputs, Outputs, and Internal Signals.

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

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

7. Repeat step 5 and 6 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.



Name	Value	Data Type
count[9:0]	0101100101	Array
sine_h[19:0]	0000010001...	Array
sine_m[17:0]	0101001110...	Array
sine_l[15:0]	UUUUUUUUU...	Array
sine_h_dly[19:0]	0000010001...	Array
sine_m_plus_l[...	XXXXXXXXXX...	Array
sine_h_minus_...	XXXXXXXXXX...	Array
s_axis_phase_...	1	Logic
m_axis_data_t...	1	Logic
aclk	1	Logic
aresetn	1	Logic
aresetn_sig	1	Logic
aresetn_sig_r	1	Logic
aresetn_sig_rr	1	Logic
aresetn_sig_rrr	1	Logic
s_axis_phase_...	0000000101...	Array
m_axis_data_t...	0000000001...	Array
s_axis_phase_...	01100101	Array
m_axis_data_t...	0000000101...	Array
s_axis_phase_...	00100101	Array
m_axis_data_t...	1101101011...	Array

#	Name	Value
1	Testbench	
2	TB Signals	
3	sys_clk_p	0
4	sys_clk_n	1
5	reset	0
6	gpio_buttons[1	3
9	gpio_switch	1
10	leds_n[3:0]	X
15	SineGen	
16	DUT Signals	
41	Inputs	
42	clk	0
43	reset	0
44	sel[1:0]	0
47	Outputs	
48	sine[19:0]	UUUUU
69	Internal Signals	
70	count[9:0]	17d
81	sine_h[19:0]	0499d
102	sine_m[17:0]	03eac
121	sine_l[15:0]	UUUU
138	sine_h_dly[19:	0496b
159	sine_m_plus_l[	XXXXX
180	sine_h_minus_	XXXXXX

Step 5: Changing Signal Properties

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

Viewing Hierarchical Signal Names

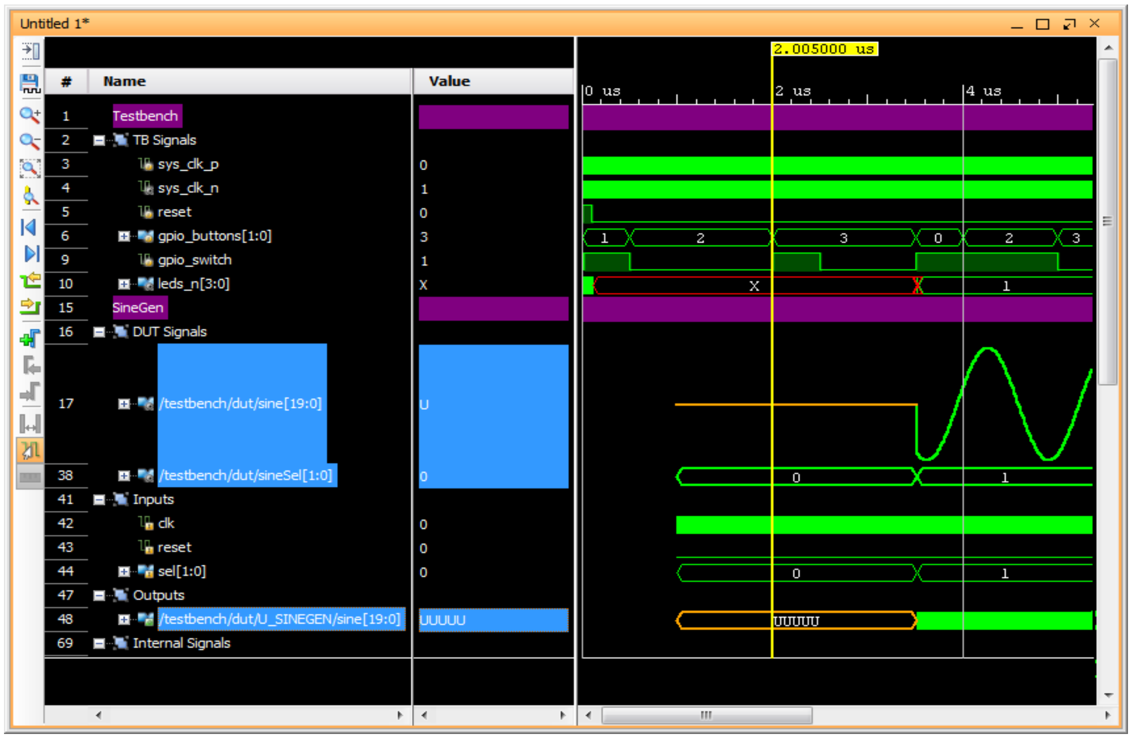
By default, the Vivado simulator adds signals to the waveform configuration using a short name with the hierarchy reference removed. For some signals, it is important to know to which module they belong.

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

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

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

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


Untitled 1*

#	Name	Value
1	Testbench	
2	TB Signals	
3	sys_clk_p	0
4	sys_clk_n	1
5	reset	0
6	gpio_buttons[1:0]	3
9	gpio_switch	1
10	leds_n[3:0]	X
15	SineGen	
16	DUT Signals	
17	/testbench/dut/sine[19:0]	U
38	/testbench/dut/sineSel[1:0]	0
41	Inputs	
42	clk	0
43	reset	0
44	sel[1:0]	0
47	Outputs	
48	/testbench/dut/U_SINEGEN/sine[19:0]	UUUUU
69	Internal Signals	

Viewing Signal Values

You can understand some signal values more clearly, if they display in a different radix format than the default, for instance, hexadecimal values instead of binary values. The default radix is binary, unless you override the radix for a specific object. Supported radix values are default, binary, hexadecimal, octal, ASCII, signed and unsigned decimal, and real.

1. In the Waveform window select the following signals:

s_axis_phase_tdata_sine_high, s_axis_phase_tdata_sine_mid and

s_axis_phase_tdata_sine_low.

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

The values on these signals now display using the specified radix.

Step 6: Saving the Waveform Configuration

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

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

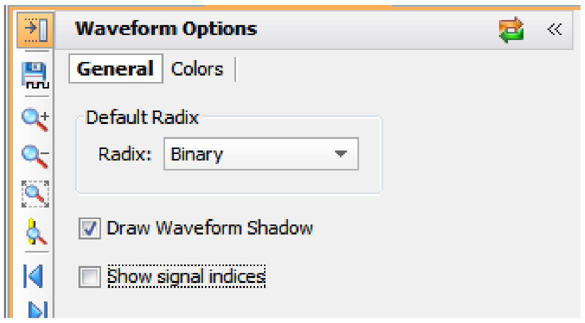
2. Ensure the Default Radix is set to Binary. This defines the default number format for all signals in the Waveform window. The radix can also be set for individual objects in the Waveform window to override the default.

3. Select the Draw Waveform Shadow to enable or disable the shading under the signal waveform.

By default, a waveform is shaded under the high transitions to make it easier to recognize the transitions and states in the Waveform window.

You can also enable or disable signal indices, so that each signal or group of signals is identified with an index number in the Waveform window.

4. Check or uncheck the Show Signal Indices checkbox to enable or disable the signal list numbering.



Default Radix

Radix: Binary

[✓] Draw Waveform Shadow

[ ] Show signal indices

5. In the Waveform Options dialog box, select the Colors view.

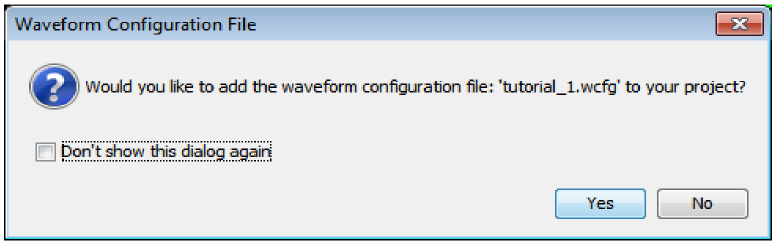
Examine the Waveform Color Options dialog box. You can configure the coloring for elements of the Waveform window to customize the look and feel. You can specify custom colors to display waveforms of certain values, so you can quickly identify signals in an unknown state, or an uninitialized state. With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

6. In the Waveform window sidebar menu, select the Save Wave Configuration button.

7. Save the Wave Configuration into the project folder with the filename tutorial_1.wcfg.

When saving the waveform configuration file, you are prompted to add this file to your current project, as seen below.



Waveform Configuration File

Would you like to add the waveform configuration file: 'tutorial_1.wcfg' to your project?

Don't show this dialog again

Yes No

8. Click Yes, the file is added to the project simulation fileset, sim_1, for archive purposes.

7: Re-Simulating the Design

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

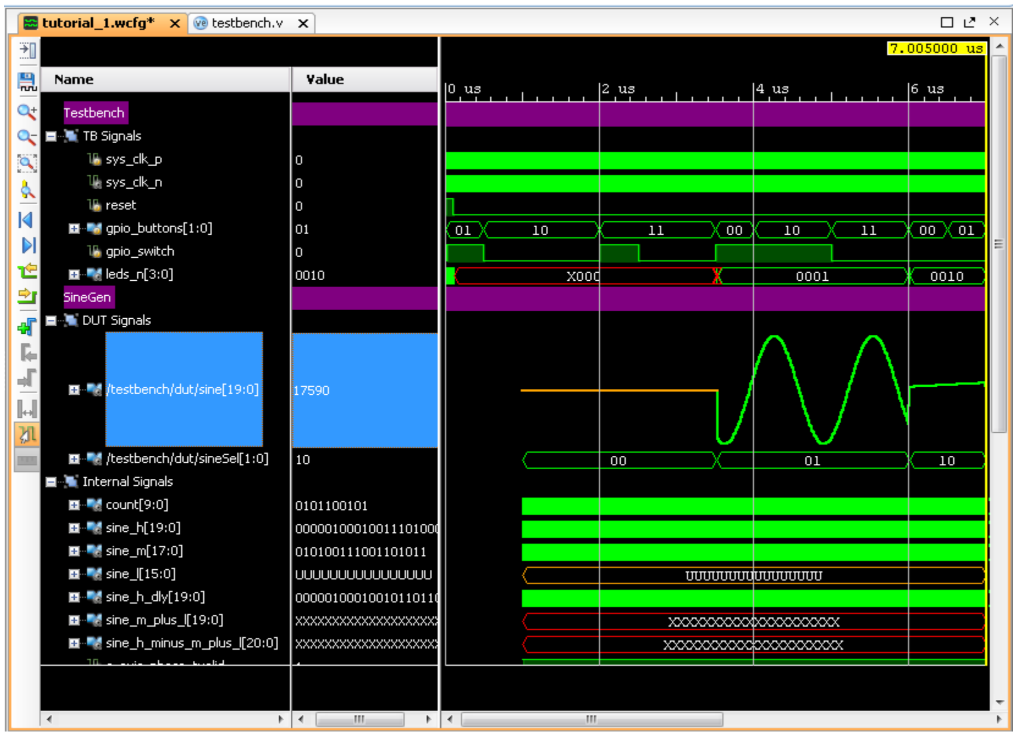
1. Click the Restart button to reset the circuit to its initial state.

2. Click the Run All button.

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

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

The following shows the current simulation results.



tutorial_1.wcfg* testbench.v

Name	Value
Testbench	
TB Signals	
sys_clk_p	0
sys_clk_n	0
reset	0
gpio_buttons[1:0]	01
gpio_switch	0
leds_n[3:0]	0010
SineGen	
DUT Signals	
/testbench/dut/sine[19:0]	17590
/testbench/dut/sineSel[1:0]	10
Internal Signals	
count[9:0]	0101100101
sine_h[19:0]	00000100010011101000
sine_m[17:0]	010100111001101011
sine_l[15:0]	UUUUUUUUUUUUUUUU
sine_h_dly[19:0]	0000010001001011011
sine_m_plus_l[19:0]	XXXXXXXXXXXXXXXXXXXX
sine_h_minus_m_plus_l[20:0]	XXXXXXXXXXXXXXXXXXXX

8: 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.

1. In the Waveform window select the /testbench/dut/sineSel[1:0] signal.

2. In the waveform sidebar menu, click the Go to Time 0 button.

The current marker moves to the start of the simulation run.

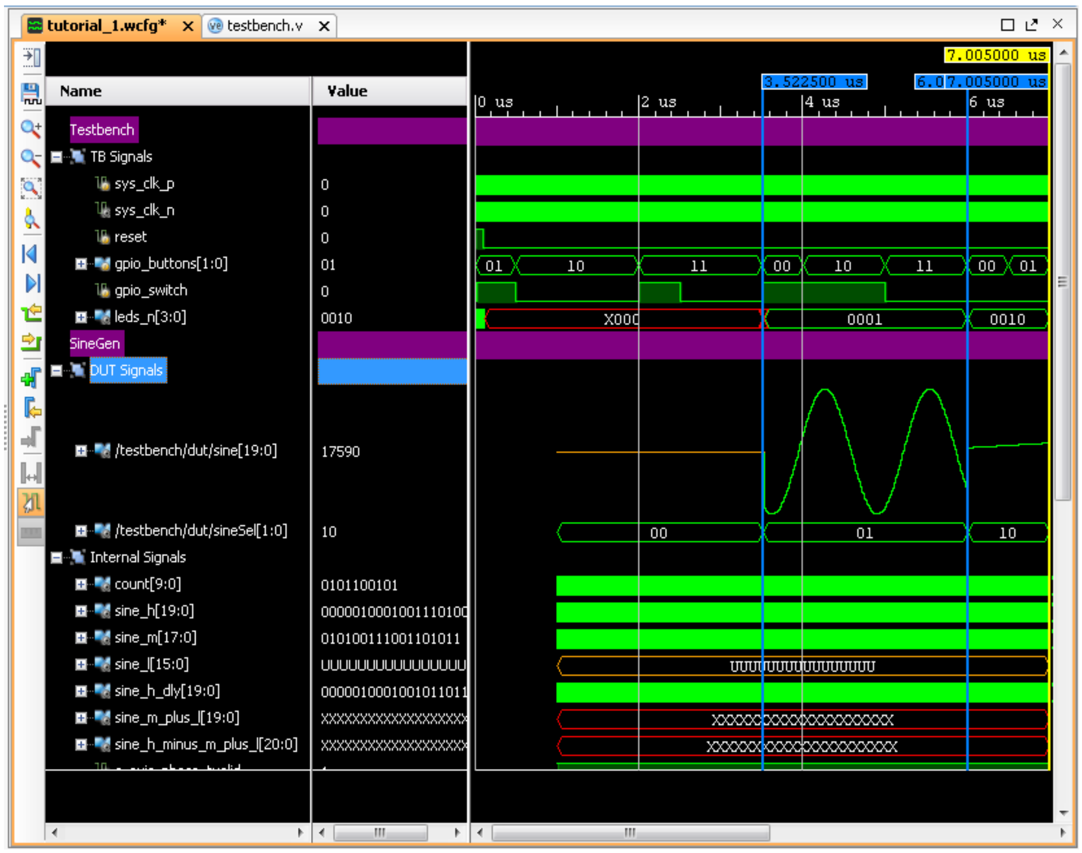
3. Enable the Snap to Transition button to snap the cursor to transition edges.

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 1 microsecond.

5. Click the Next Transition button again and the marker moves to 3.5225 microseconds.

6. Click the Add Marker button



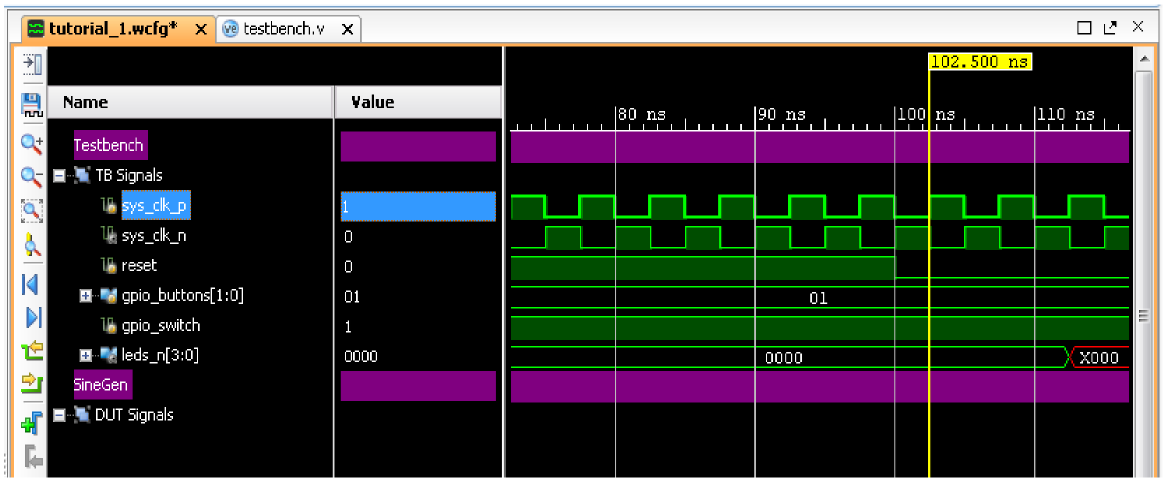
Name	Value
Testbench	
TB Signals	
sys_clk_p	0
sys_clk_n	0
reset	0
gpio_buttons[1:0]	01
gpio_switch	0
leds_n[3:0]	0010
SineGen	
DUT Signals	
/testbench/dut/sine[19:0]	17590
/testbench/dut/sineSel[1:0]	10
Internal Signals	
count[9:0]	0101100101
sine_h[19:0]	0000010001001110100
sine_m[17:0]	010100111001101011
sine_l[15:0]	UUUUUUUUUUUUUUUU
sine_h_dly[19:0]	0000010001001011011
sine_m_plus_l[19:0]	XXXXXXXXXXXXXXXXXXX
sine_h_minus_m_plus_l[20:0]	XXXXXXXXXXXXXXXXXXX

7. 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 the figure. As previously observed, the low frequency signals are incorrect when the sinSel signal value is 00. 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 00, to review the status of the output signal, sine[19:0], and identify where the incorrect behavior initiates. You will also use the cursor to measure the period of low frequency wave control.

8. In the Waveform window, click and drag the cursor from time 0 to zoom into the beginning of the simulation run.

9. 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 below.



Name	Value
Testbench	
TB Signals	
sys_clk_p	1
sys_clk_n	0
reset	0
gpio_buttons[1:0]	01
gpio_switch	1
leds_n[3:0]	0000
SineGen	
DUT Signals	

You can also zoom in to view the waveform more closely by repeatedly clicking the Zoom In button until you achieve the zoom needed to see the details of the signal. The Waveform window zooms in or out around the area centered on the cursor.

10. 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.

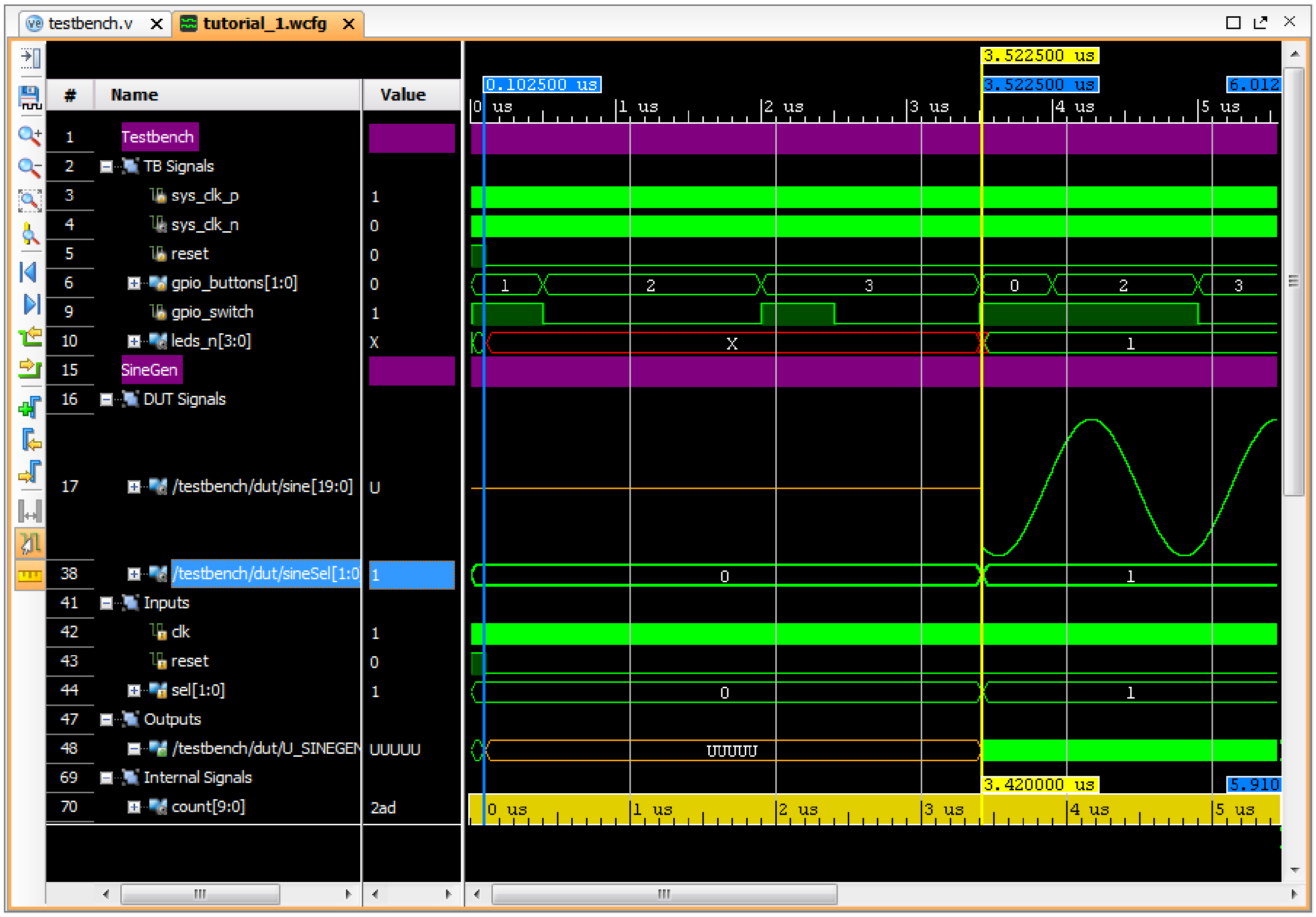
11. Because 00 is the initial or default FSM output, move the cursor to the first posedge of sys_clk_p after reset is asserted low, at time 102.5 ns. You can use the Waveform window to measure time between two points on the timeline.

12. Place a marker at the time of interest, 102.5 ns, by clicking the Add Marker button.

13. 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.



#	Name	Value
1	Testbench	
2	TB Signals	
3	sys_clk_p	1
4	sys_clk_n	0
5	reset	0
6	gpio_buttons[1:0]	0
9	gpio_switch	1
10	leds_n[3:0]	X
15	SineGen	
16	DUT Signals	
17	/testbench/dut/sine[19:0]	U
38	/testbench/dut/sineSel[1:0	1
41	Inputs	
42	clk	1
43	reset	0
44	sel[1:0]	1
47	Outputs	
48	/testbench/dut/U_SINEGEN	UUUUU
69	Internal Signals	
70	count[9:0]	2ad

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.

14. 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 the figure, the ruler measures a time period of 3.420 ns as the period that FSM selected the low frequency output.

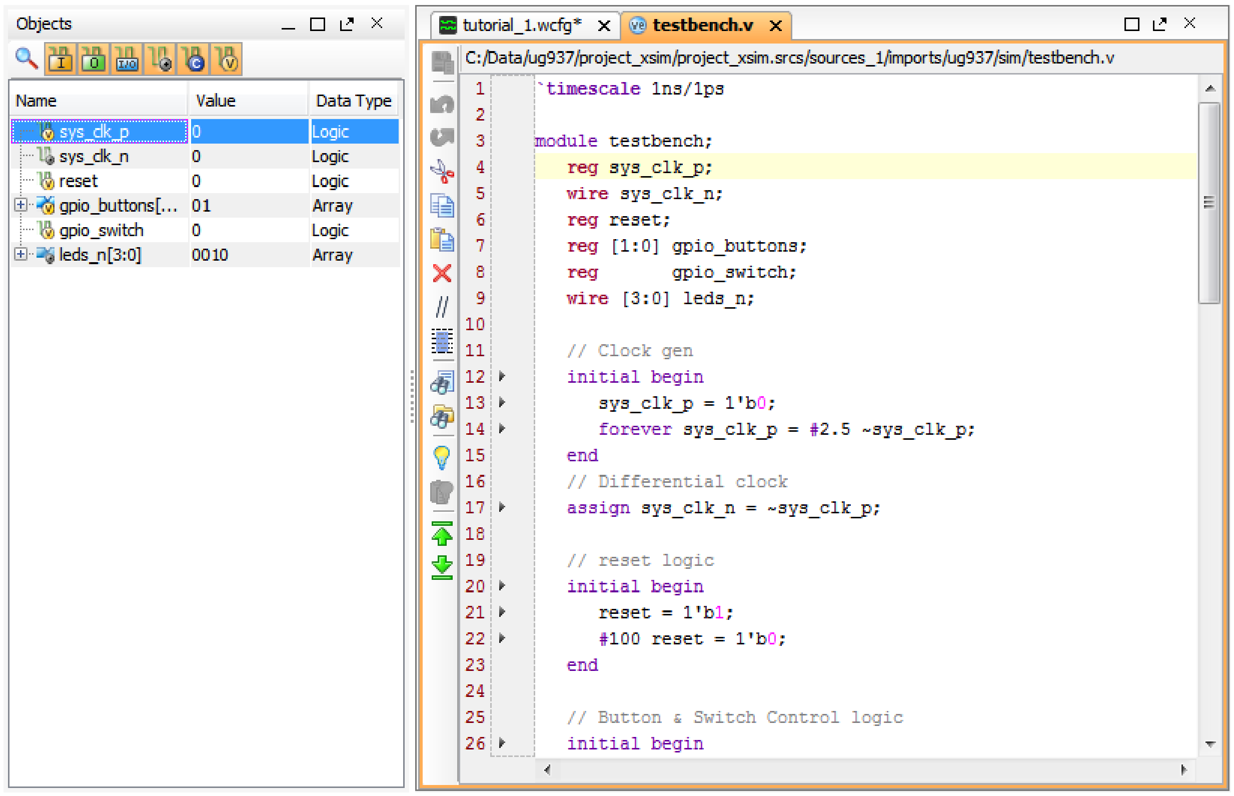
9: Debugging with Breakpoints

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

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

1. Open the testbench.v file by double-clicking on the file in the Sources window, if it is not already open.

The source file opens in the Vivado IDE Text Editor.



Name	Value	Data Type
sys_clk_p	0	Logic
sys_clk_n	0	Logic
reset	0	Logic
gpio_buttons[...	01	Array
gpio_switch	0	Logic
leds_n[3:0]	0010	Array

C:/Data/ug937/project_xsim/project_xsim.srcs/sources_1/imports/ug937/sim/testbench.v

```
`timescale 1ns/1ps

module testbench;
   reg sys_clk_p;
   wire sys_clk_n;
   reg reset;
   reg [1:0] gpio_buttons;
   reg       gpio_switch;
   wire [3:0] leds_n;

   // Clock gen
   initial begin
      sys_clk_p = 1'b0;
      forever sys_clk_p = #2.5 ~sys_clk_p;
   end
   // Differential clock
   assign sys_clk_n = ~sys_clk_p;

   // reset logic
   initial begin
      reset = 1'b1;
      #100 reset = 1'b0;
   end

   // Button & Switch Control logic
   initial begin
```

Note: You can also use File > Open File from the main menu, or Open File from the popup menu in the Sources window.

You can also select an appropriate design object in the Scopes window or Objects window, right-click and select Go to Source Code.

Using Breakpoints

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

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

Select the U_SINEGEN 1. scope in the Scopes window, to list the objects of that scope in the Objects window.

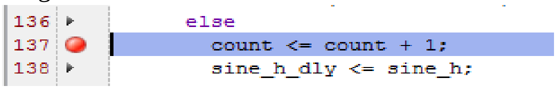
2. In the Objects window, right-click on sine[19:0], and use Go to Source Code to open the sinegen.vhd source file in the Text Editor.

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

3. Scroll to line 137 in the file. Add a breakpoint at line 137 in sinegen.vhd. Note that the breakpoint can be set only on the executable lines. Vivado simulator marks the executable lines using an arrowhead symbol on the left hand margin in the Text Editor, along with the line numbers. Setting a breakpoint causes the simulator to stop at that point, every time the simulator processes that code, or every time the counter is incremented by one.

4. Click the arrowhead in the left margin, to set a breakpoint. Observe that the symbol changes to a to indicate that a breakpoint is set on this line.

You can remove a breakpoint by clicking on the red dot to revert it to an arrowhead.



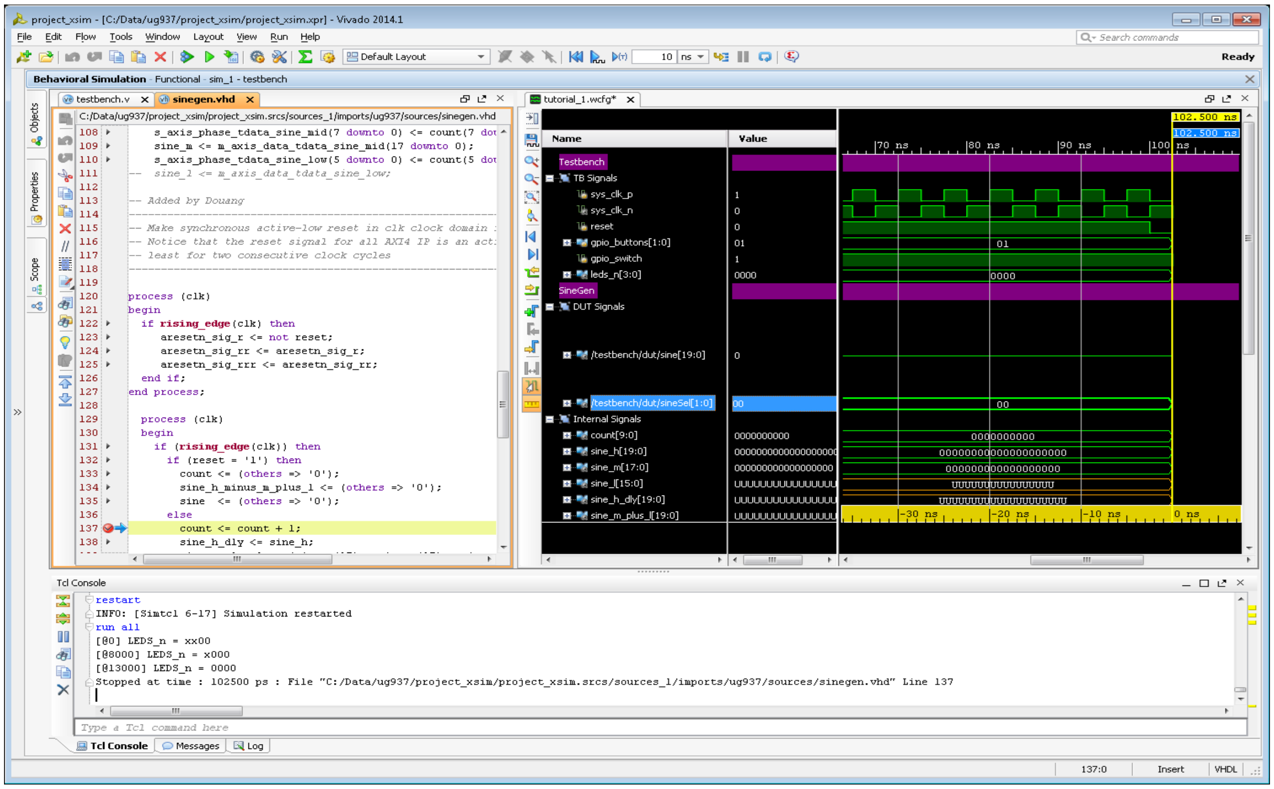
```
136 ▸     else
137 ●       count <= count + 1;
138 ▸       sine_h_dly <= sine_h;
```

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

Debugging in the Vivado simulator, with breakpoints and line stepping, works best when

you can view the Tcl Console, the Waveform window, and the HDL source file at the same time.

5. Resize the windows, and use the window Float command, or the New Vertical Group command, to arrange the various windows so that you can see them all.



```
108      s_axis_phase_tdata_sine_mid(7 downto 0) <= count(7 dow
109      sine_m <= m_axis_data_tdata_sine_mid(17 downto 0);
110      s_axis_phase_tdata_sine_low(5 downto 0) <= count(5 dow
111  --  sine_l <= m_axis_data_tdata_sine_low;
112
113  -- Added by Douang
114  ------------------------------------------------------------
115  -- Make synchronous active-low reset in clk clock domain
116  -- Notice that the reset signal for all AXI4 IP is an act
117  -- least for two consecutive clock cycles
118  ------------------------------------------------------------
119
120  process (clk)
121  begin
122    if rising_edge(clk) then
123       aresetn_sig_r <= not reset;
124       aresetn_sig_rr <= aresetn_sig_r;
125       aresetn_sig_rrr <= aresetn_sig_rr;
126    end if;
127  end process;
128
129    process (clk)
130    begin
131      if (rising_edge(clk)) then
132        if (reset = '1') then
133          count <= (others => '0');
134          sine_h_minus_m_plus_l <= (others => '0');
135          sine  <= (others => '0');
136        else
137          count <= count + 1;
138          sine_h_dly <= sine_h;
```

```
restart
INFO: [Simtcl 6-17] Simulation restarted
run all
[@0] LEDS_n = xx00
[@8000] LEDS_n = x000
[@13000] LEDS_n = 0000
Stopped at time : 102500 ps : File "C:/Data/ug937/project_xsim/project_xsim.srcs/sources_1/imports/ug937/sources/sinegen.vhd" Line 137
```

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

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

The simulation runs to time 102.5 ns, 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 as picoseconds, indicating the line of source code last executed by the simulator.

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

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

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

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

10. 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.

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

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

Stepping Through Source Code

Another useful Vivado simulator debug tool is the Line Stepping feature. With line stepping, you can run the simulator one-simulation unit (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.

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.

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

2. Continue to Step through the design, until the code returns to line 144 of sinegen.vhd. You have stepped through one complete cycle of the circuit. Notice in the Waveform window that while sel is 00, signal sine_l is assigned as a low frequency sine wave to the output sine. Also notice that sine_l remains uninitialized.

3. For debug purposes, initialize the value of sine_l by entering the following add_force command in the Tcl console:

```
add_force /testbench/dut/U_SINEGEN/sine_l 0110011011001010
```

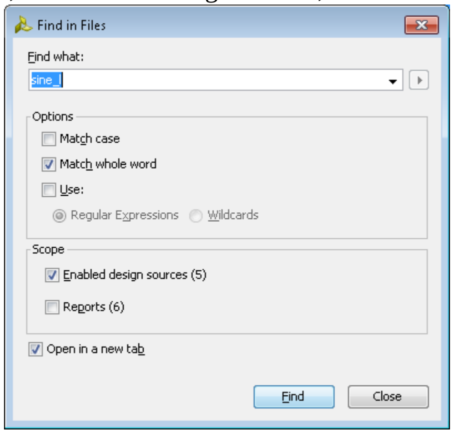
This command will force the value of sine_l into a specific known condition, and can provide a repeating set of values to exercise the signal more vigorously if needed.

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

In the Waveform window notice that the value of sine_l[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 00. Trace the sine_l signal in the HDL source files, and identify the input for sine_l.

5. In the Text Editor, use the Find in files button to search for sine_l.

6. Select the Match whole word checkbox, and Enabled design sources, and click OK.



Find what: sine_l

Options
- [ ] Match case
- [✓] Match whole word
- [ ] Use:
  - [● Regular Expressions] Wildcards

Scope
- [✓] Enabled design sources (5)
- [ ] Reports (6)

[✓] Open in a new tab

The Find in Files results display at the bottom of the Vivado IDE, with all occurrences of sine_l found in sinegen.vhd.

7. Expand the Find in Files results to view the results in the sinegen.vhd file.

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

8. Uncomment line 111 in the sinegen.vhd file, and click the Save File button.

9. In the Tcl Console, remove the force on sine_l: remove_forces -all

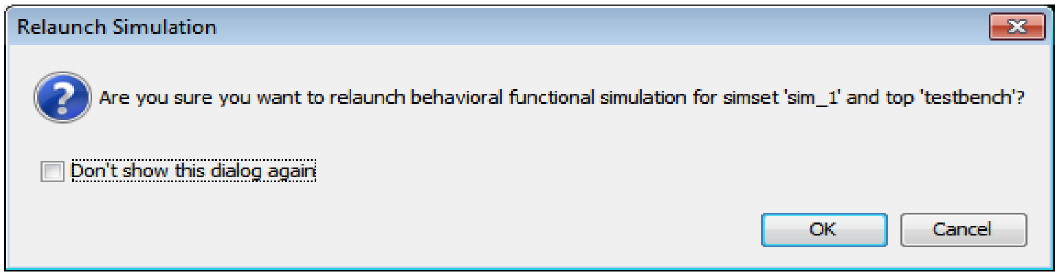
10: Re-launch Simulation

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

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

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

2. Click the Re-launch button on the main toolbar menu. The Vivado IDE prompts you to confirm re-launching the simulator.


Relaunch Simulation

Are you sure you want to relaunch behavioral functional simulation for simset 'sim_1' and top 'testbench'?

Don't show this dialog again

3. Click OK to continue.

Note: 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.

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

Observe the sine[19:0], the final sine wave output signal in the waveform configuration.

The low frequency sine wave looks as expected.

The Tcl console results are:

```
[@3518000] LEDS_n = 0100

[@3523000] LEDS_n = 0001
```

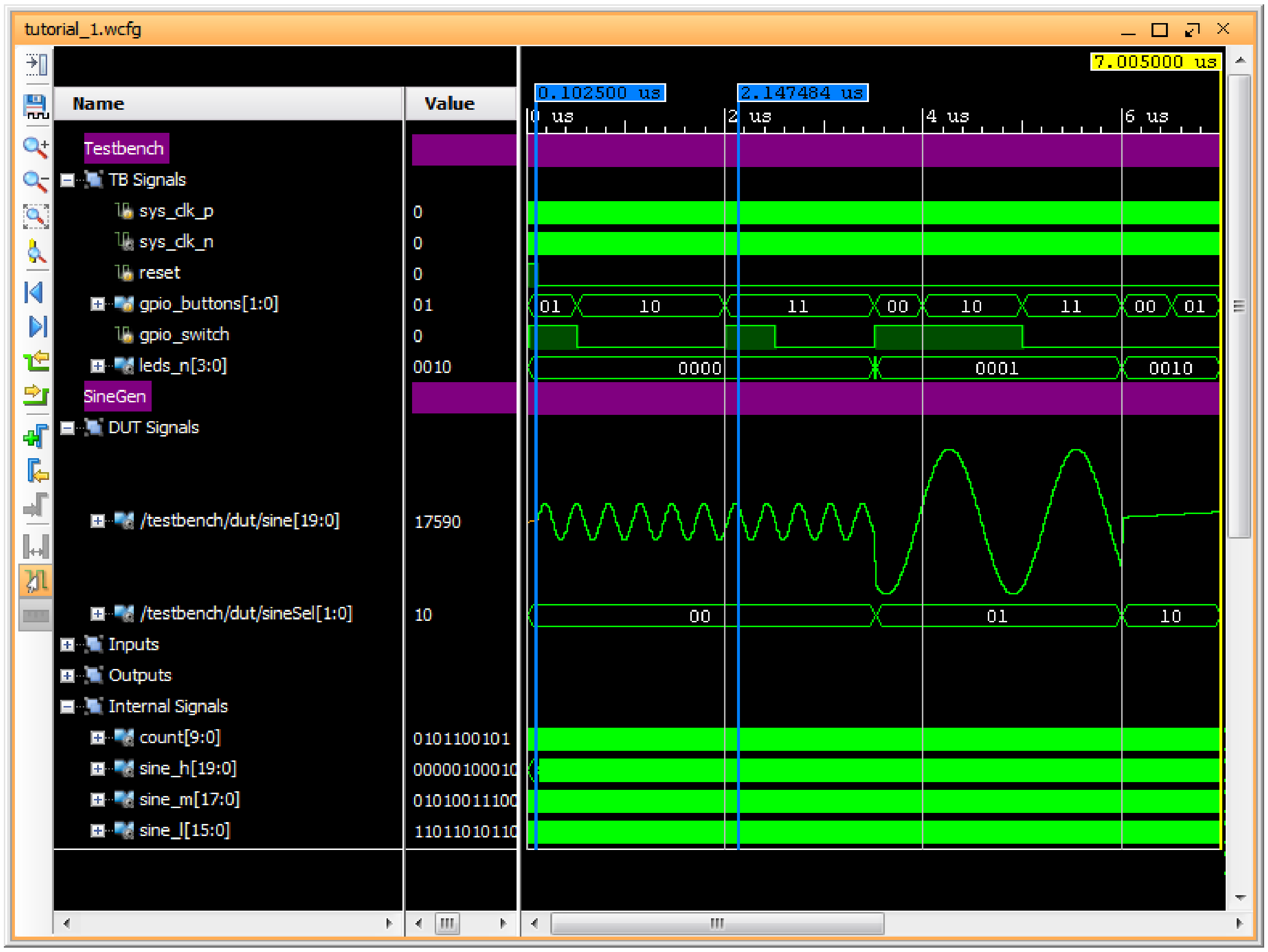
```
[@3523000] LEDS_n = 0001

[@6008000] LEDS_n = 0101

[@6013000] LEDS_n = 0010

[@6013000] LEDS_n = 0010

$finish called at time : 7005 ns : File "ug937/sim/testbench.v" Line 63
```


tutorial_1.wcfg

Name	Value
Testbench	
TB Signals	
sys_clk_p	0
sys_clk_n	0
reset	0
gpio_buttons[1:0]	01
gpio_switch	0
leds_n[3:0]	0010
SineGen	
DUT Signals	
/testbench/dut/sine[19:0]	17590
/testbench/dut/sineSel[1:0]	10
Inputs	
Outputs	
Internal Signals	
count[9:0]	0101100101
sine_h[19:0]	00000100010
sine_m[17:0]	01010011100
sine_l[15:0]	11011010110