Third-Party Simulation Tool Usage Guidelines and Tips

Application Note

FPGA-AN-02084-1.2

August 2026

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Inclusive Language

This document was created consistent with Lattice Semiconductor’s inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice’s inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Figures

Tables

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
DUT	Device Under Test
GSR	Global Set/Reset
GUI	Graphical User Interface
IP	Intellectual Property
OEM	Original Equipment Manufacturer
PLL	Phase-Locked Loop
PUR	Power-Up Reset
RTL	Register Transfer Level
TCL	Tool Command Language

1. Introduction

Project simulation is one of the most important steps in the FPGA project development flow. However, one of the first steps is to decide on what simulation tool to use. All of Lattice's flagship software tools – the Lattice Radiant™ software, Lattice Diamond™ software, and Lattice Propel™ Builder software–come packaged with their own original equipment manufacturer (OEM) simulators. However, these OEM simulators have some feature and performance limitations that may not be ideal for everyone.

As an alternative to the included OEM simulators, Lattice also supports several different types of third-party simulation tools, which you can use to simulate your own Lattice FPGA projects separately from any OEM simulation tool. These supported third-party non-OEM simulation tools include Mentor Graphics® ModelSim, Siemens® Questa® SIM , Synopsys® VCS, and Cadence® Xcelium™.

In general, project simulation requires some register transfer level (RTL) that reflects the design a user wants to implement, as well as a testbench which is meant to model realistic stimulus for the device under test (DUT). However, depending on the intellectual property (IP) and primitives used in the DUT, there is some additional overhead that must be considered when using third-party simulation tools.

- Purely RTL designs—If a design is purely RTL and only consists of Verilog or VHDL code, then out-of-the-box project simulation should not require much additional consideration.
- Designs that use IP or primitives—However, it is common for user designs to use IP or primitives to model some on-device FPGA function. Since these IP or primitives are not purely RTL, a simulation library is required so simulation tools can resolve these references and understand how to model their behavior during simulation.

For OEM simulators, Lattice includes precompiled simulation libraries. However, for third-party simulation tools, you are required to compile the relevant Lattice simulation libraries ahead of time before simulating your designs.

Each Lattice FPGA device family has its own simulation library, whose source files come packaged with every Radiant software and Diamond software installation.

- Diamond software—Both Verilog and VHDL libraries are supported.
- Radiant software—Only Verilog-based simulation libraries are supported.

The source files for each device's simulation library can be found within the following directory:

```
<Lattice tool installation directory>/cae_library/simulation/verilog/<device>
```

The *<Lattice tool installation directory>* is the base directory where either the Radiant or Diamond software is installed to, and the *<device>* refers to the target device. Every Lattice tool release comes with precompiled versions of these libraries that are compatible with the respective tool's included OEM simulator so no additional work is required if you are using an OEM simulation tool from Lattice. However, for third-party simulators, it is recommended that you compile these libraries rather than reference their source files directly to improve simulation runtime and enable easier debugging of errors later down the line.

Note: All of Lattice's simulation libraries are Verilog-based. Because of this, a mixed-language simulation license is required for your target simulation tool of choice if you want to simulate a VHDL or mixed-language design with a third-party tool.

1.1. Compiling Lattice Simulation Libraries

Depending on your non-OEM simulation tool of choice, you have the option to compile your required Lattice simulation libraries before simulating your design with a third-party simulation tool ahead of time. This is true for Mentor Graphics ModelSim, Siemens QuestaSim, Cadence Xcelium, and Synopsys VCS simulators.

Note: Be aware of version dependencies between different versions of your simulation tool. For example, precompiled simulation libraries included with Lattice's OEM version of ModelSim are not directly compatible with other versions of ModelSim that you may have licensed through a third party. For this reason, it is recommended that you recompile your device libraries of choice in advance if you plan to update your simulation tool version.

There are two ways to compile Lattice device simulation libraries:

- Using precompiled libraries with cmpl_libs (recommended).
- Manually referencing device libraries (without cmpl_libs).

1.1.1. Precompiled Libraries Using cmpl_libs (Recommended)

To assist in the process of compiling Lattice FPGA simulation libraries for use with third-party simulators, Lattice provides a *cmpl_libs* TCL script that you can use to semi-automatically compile all the relevant simulation files required to simulate a Lattice FPGA project. This script can be invoked directly from the Radiant or Diamond software's integrated TCL consoles or directly from a command-line environment. The purpose of this script is to simplify the process of creating simulation libraries for Lattice's various supported third-party simulation tools so you do not need to create these libraries manually, or compile all the RTL from a device's simulation library each time you run a simulation for your project.

With that said, the syntax for the *cmpl_libs* method of compiling libraries for use with third-party simulation tools requires you to specify only up to four settings depending on your simulation tool of choice and target device.

The following lists the syntax for *cmpl_libs* for Diamond and Radiant software*:*

Diamond Software cmpl_libs Options

```
> cmpl_libs -sim_path <target simulator base directory> \
-sim_vendor <target simulator> \
-device <target device> \
-lang <select which HDL to compile simulation libraries> \
-target_path <location to generate compiled libraries>
```

Radiant Software cmpl_libs Options

```
> cmpl_libs -sim_path <target simulator base directory> \
-sim_vendor <target simulator> \
-device <target device> \
-64 <compile simulation libraries with 64-bit support> \
-target_path <location to generate compiled libraries>
```

The following table lists the values and descriptions for each cmpl_libs option.

Table 1.1. Values and Descriptions for Each cmpl_libs Option

Setting	Value(s)	Description
-sim_path (Mandatory)	Directory	Used to specify the base installation directory of your third-party simulation tool. For example, if Synopsys VCS is your simulator of choice, point to the directory containing its binary executable.
-sim_vendor	Radiant software: • `mentor` (default for Radiant 2024.2 and earlier) • `siemens` (default for Radiant 2025.1 and later) • `cadence` • `synopsys` • `aldec` Diamond software: `mentor` (default)	Used to specify the target simulation tool. The following lists the default simulator setting by software version: • Radiant software version 2024.2 and earlier—`mentor`. • Radiant software version 2025.1 and later—`siemens`. • Diamond software (all versions) —`mentor`. Using the default simulator setting compiles the specified simulation libraries for use with either QuestaSim or ModelSim simulator depending on the binary specified with `-sim_path`. Although this setting is optional, you must set this correctly if you are using a non-Mentor simulator, as the commands required to compile a simulation library vary by tool. Note that supported third-party simulators differ depending on whether you are using the Radiant or Diamond software. For the Diamond software, only Mentor simulators are supported as third-party simulation tools.
-device	**Radiant software:**	Used to specify the target devices for which simulation libraries will

Setting	Value(s)	Description
	iCE40UP, lifcl, lfcpnx, lfd2nx, lfd2nx1, lfd2nx2, lfmxo4, lfmxo4d, lfmxo5, lfmxo5t, lfmxo5t1, lav_ate_es, lav_atg_es, lav_atx_es, lav_ate_b, ln2_mh, ln2_ct, lav_ate, lav_atg, lav_atx, ut24c, ut24cp, all (default) **Diamond software:** sc, scm, ec, xp, ecp, machxo, ecp2, ecp2m, xp2, ecp3, machxo2, machxo3l, machxo3d, machxo3lfp, lptm, lptm2, ecp5u, ecp5um, lfmnx, lifmd, lifmdf, all (default)	be created. If this option is not specified, simulation libraries will be created for all devices supported by the Lattice tool. This is useful when you do not plan to use multiple device families and want to reduce compilation time with cmpl_libs. Note that the available devices depend on which Lattice software you are using to simulate your design. For example, if you are using the cmpl_libs script from the Radiant software, only devices supported within the Radiant software can be specified for this option.
-target_path	Directory	Specifies where to generate the simulation libraries that are compiled and created with cmpl_libs. By default, if `-target_path` is not specified, the current directory that the script is invoked from will be selected. Depending on the `-device` option selected, cmpl_libs generates additional subdirectories within the target directory corresponding to the required Lattice simulation libraries. For example, if the selected device is ice40up, the directories `ice40up`, `uaplatform`, and `pmi` are generated in the specified target path. Alternatively, if `all` is selected instead of a specific device, additional directories corresponding to the other supported devices are generated, such as lifcl, lfcpnx, lfd2nx, and so on.
-64 (Only in Radiant 2025.1 onwards)	*This is a flag (no value)	Used to compile simulation libraries with 64-bit support.
- lang (Only in Diamond software)	Verilog, VHDL, all (default)	Used to select the hardware description language (HDL) on which your compiled simulation libraries will be based.

The following table lists the full device names for each cmpl_lib *-device* option.

Table 1.2. Device Family Names for cmpl_libs -device Setting

Target Tool	Cmpl_lib Device Setting	Device Family Name
Radiant software	ice40up	iCE40 UltraPlus™
	lifcl	CrossLink™-NX
	lfcpnx ut24cp	CertusPro™-NX
	lfd2nx lfd2nx1 lfd2nx2 ut24c	Certus™-NX
	ln2_mh ln2_ct	Lattice Nexus™ 2
	lfmxo4 lfmxo4d	MachXO4™
	lfmxo5 lfmxo5t lfmxo5t1	MachXO5™-NX
	lav_ate_es lav_atg_es	Avant™

Target Tool	Cmpl_lib Device Setting	Device Family Name
	lav_atx_es lav_ate_b lav_ate lav_atg lav_atx	
Diamond software	xp	LatticeXP™
	xp2	LatticeXP2™
	sc scm	LatticeSC™/M
	ec ecp	LatticeEC™/P
	ecp2 ecp2m	LatticeECP2™/M
	ecp3	LatticeECP3™
	ecp5u ecp5um	LatticeECP5™
	lptm lptm2	Lattice Platform Manager
	machxo2	MachXO2™
	machxo3l	MachXO3L™
	machxo3d	MachXO3D™
	machxo3lfp	MachXO3LF™
	lfmnx	Mach™-NX
	lifmd	CrossLink™
	lifmdf	CrossLinkPlus™

The following lists the methods to compile simulation libraries with the cmpl_libs script:

- Using the integrated TCL console on the Radiant or Diamond software.
- Using the standalone TCL console from the Radiant or Diamond software directory.
- Using the command line.

1.1.1.1. Method 1: Using the Integrated TCL Console

Access the integrated TCL console located at the bottom page of the Radiant and Diamond software GUI. If the TCL console cannot be located from your perspective, enable it through **View > Show Views > TCL console.**

One thing to keep in mind when using this method is the default location compiled libraries are generated to.

- If a project is not open—the compilation directory defaults to the *<Lattice tool installation directory>/bin/nt64* directory.
- If a project is open—the compilation directory defaults to the directory of the active project unless the `-target_path` option is specified.

Note: For this reason, Lattice recommends using the `-target_path` option to avoid any potential issues or confusion.

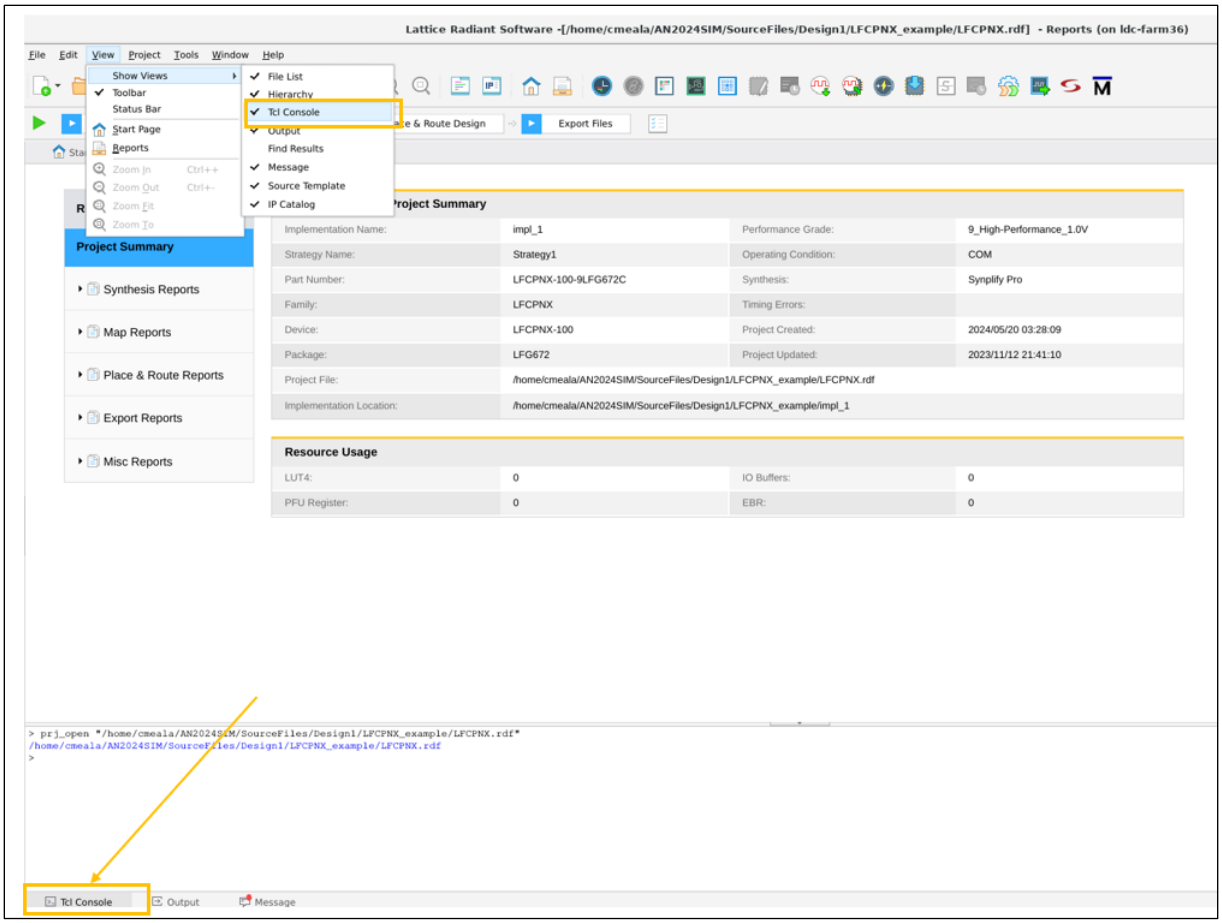

Figure 1.1. Radiant Software Integrated TCL Console

The following are a few examples for compiling simulation libraries using the cmpl_libs script:

1. cmpl_libs: this allows you to verify the usage or syntax used for the *cmpl_libs* command.
2. cmpl_libs *-sim_path* /tools/dist/cadence/XCELIUM/XCELIUM20.09.004/Linux/tools.lnx86/bin/ *-sim_vendor* cadence -device lfd2nx *-target_path* /home/cmeala/AN2024SIM/radiant_models/lfd2nx_model/:
   a. `-sim_path` – has been targeted to the 3rd party tool's binary executable.
   b. *`-target_path`* – is targeted to a folder named lfd2nx_model. This folder is where your logical libraries are located once library compilation is successful.
3. The libraries are successfully generated with the following message:
   Libraries located in <target_path> is updated.

Figure 1.2. TCL Command Examples for Compiling Simulation Libraries

1.1.1.2. Method 2: Using the Standalone TCL Console

Aside from the integrated TCL console described in Method 1, both the Diamond and Radiant software also support a standalone TCL console, which can also be used to compile your simulation libraries. Ultimately this method works the same as Method 1, with the main difference being how you interact with the Radiant or Diamond software's TCL consoles. In this flow, as its name implies, you use a standalone TCL console rather than the one integrated within the Radiant or Diamond software. After you have opened the TCL console, the following few steps to compile the simulation libraries are exactly the same.

To open the standalone TCL console for the Radiant software, execute the pnmainc or radiantc application in the following directory:

- For Windows operating system—These files are located at *<Radiant installation directory>/bin/nt64*.
- For Linux operating system—You must use the radiantc file that is located *at <Radiant installation directory>/bin/lin64*.

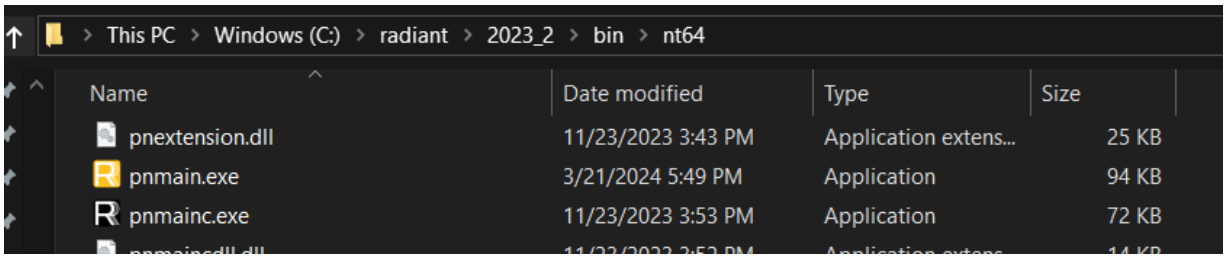

Figure 1.3. Folder Location of pnmainc

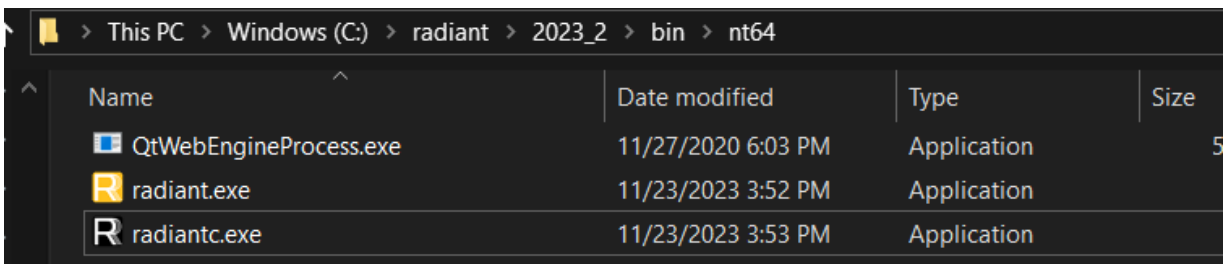

Figure 1.4. Folder Location of the Radiant Software Application

To open the standalone TCL console for the Diamond software, execute the pnmainc application in the following directory:

- For Windows operating system—The file is located at *<Diamond installation directory>/bin/nt64*.
- For Linux operating system—You must use the Diamond file located at *<Diamond installation directory>/bin/lin64*.

Note: You may use the same commands from the example in the Method 1: Using the Integrated TCL Console section to compile the simulation libraries.

Figure 1.5. TCL Command Examples for Compiling Simulation Libraries Using the radiantc Application

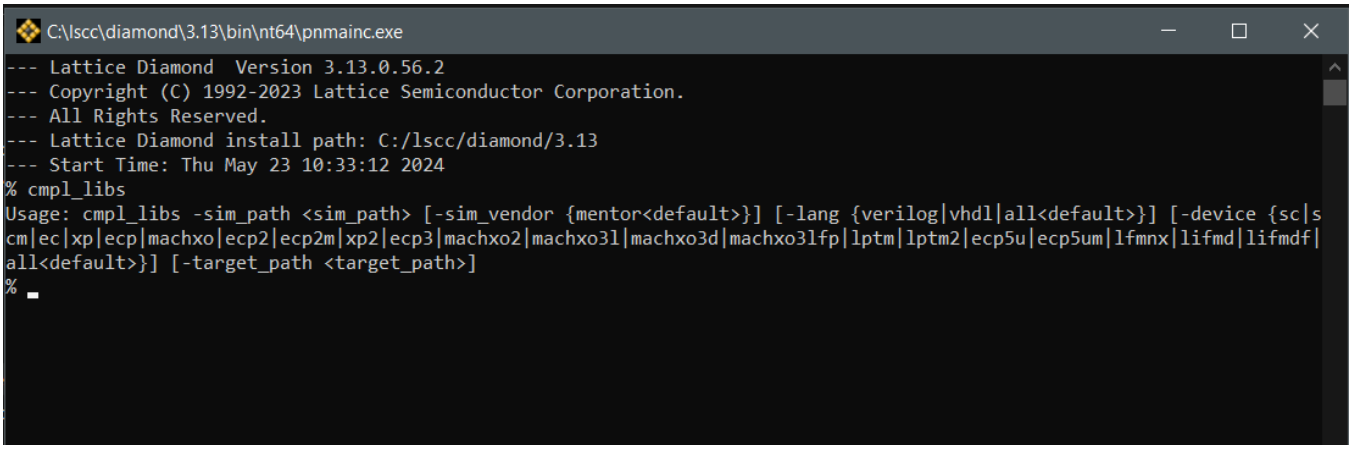

Figure 1.6. TCL Command Examples for Compiling Simulation Libraries Using the pnmainc Application

1.1.1.3. Method 3: Using Command Line

To compile simulation libraries using the command line, you must set the following few environment variables so your command prompt will be able to recognize Lattice-specific commands. The following lists all the environment variables that you must set:

- For Windows operating systems execute the following commands:

```
# PATH
> set PATH=<SW_Installation_Directory>\bin\nt64\;<SW_Installation_Directory>\ispfpga\bin\nt64

# FOUNDRY
> set FOUNDRY=<SW_Installation_Directory>\ispfpga
```

- For Linux operating system execute the following commands:

```
# FOUNDRY (CSH)
> setenv FOUNDRY <SW_Installation_Directory>/ispfpga

# FOUNDRY (BASH)
> export FOUNDRY=<SW_Installation_Directory>/ispfpga

# bindir (CSH)
> setenv bindir <SW_Installation_Directory>/bin/lin64

# bindir (BASH)
> export bindir=<SW_Installation_Directory>/bin/lin64
```

For more information, refer to the *Setting Up the Batch Environment* section in the Scripting Lattice FPGA Build Flow Application Note (FPGA-AN-02073).

The following figure shows an example of setting up the environment variables on a Windows platform.

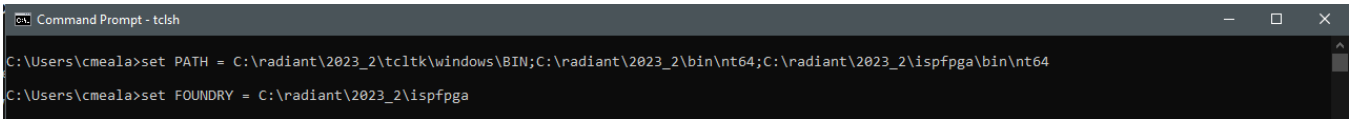

Figure 1.7. An Example of Setting Up the Environment Variables on a Windows Platform

The following figure shows an example of setting up the environment variables on a Linux platform.

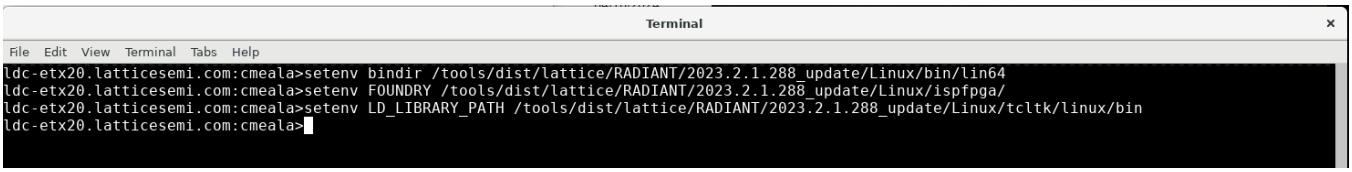

Figure 1.8. An Example of Setting Up the Environment Variables on a Linux Platform

Note: Ensure that TCL version 8.5 or later is installed on your machine. If TCL is not installed, or if an older version is installed, you must add the TCL bin directory to the PATH variable (for Windows), or to the LD_LIBRARY_PATH variable (for Linux), as shown in Figure 1.7 and Figure 1.8.

The following lists the generic syntax for setting the variables:

- For Windows operating system: *<SW_Installation_Directory>\tcltk\windows\bin.*
- For Linux operating system: *<SW_Installation_Directory>/tcltk/linux/bin.*

After all variables are set, invoke the TCL script using the `tclsh` command from the command line to begin using *cmpl_libs*. To verify that the `tclsh` command is running correctly on your machine, run the info `tclversion` command.

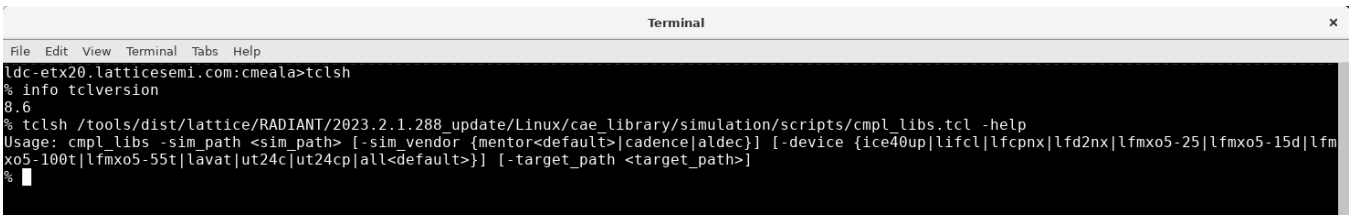

Figure 1.9. Running tclsh from the Command Line or Terminal

In comparison to Method 1 and Method 2, running the *cmpl_libs* script in Method 3 requires you to define the directory of the script file.

The following figure shows an example of libraries that are compiled correctly.

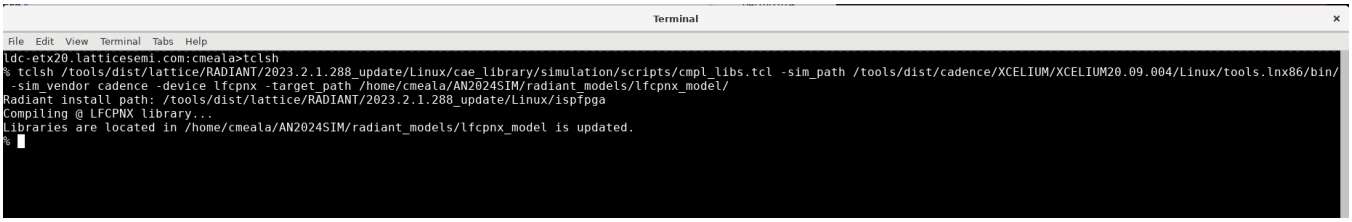

Figure 1.10. Compiling cmpl_libs with tclsh

The following figure shows an example of the content of the compiled libraries.

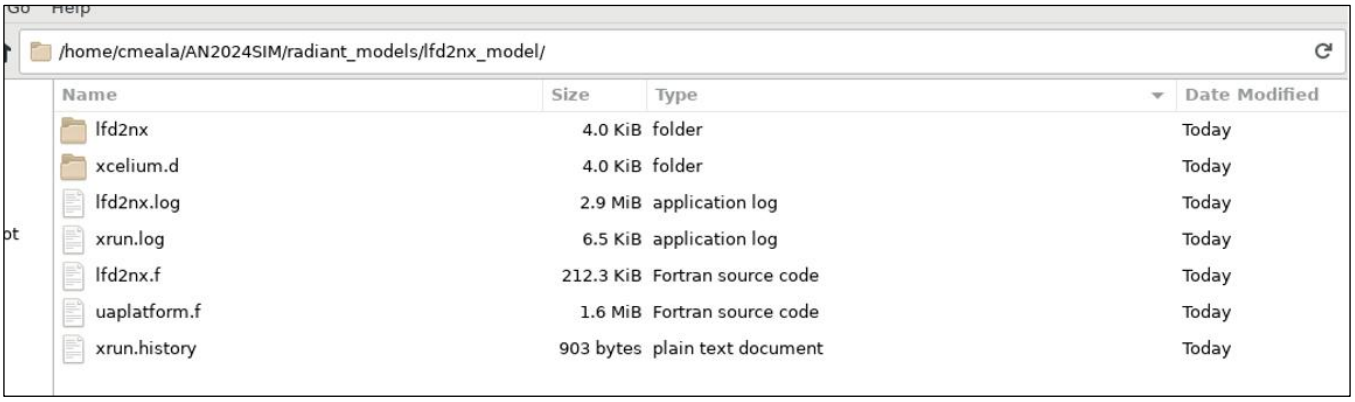

Figure 1.11. Example of cmpl_libs Library Outputs

1.1.1.4. Using Precompiled Simulation Libraries with Third-Party Simulators

After using the *cmpl_libs* script to generate the required Lattice simulation libraries for use with the selected third-party simulation tool, the next steps in the project simulation flow depend on the simulator you plan to use.

When the goal is to run a design with a third-party simulator, the FOUNDRY environment variable defined in the Method 3: Using Command Line section must remain set so that Lattice-specific IP and primitives are visible to the simulator. Additional simulator-specific environment variables may also be required so that the shell can locate the simulator executables and license server. These can be added to the user's shell start-up file or exported from the same command line from which cmpl_libs is invoked.

For non-OEM QuestaSim

```
> setenv QUESTASIM_HOME <path_to_questasim_executables> && setenv PATH $QUESTASIM_HOME/bin:$PATH
> setenv SALT_LICENSE_SERVER <port>@<server>
```

For Cadence Xcelium

```
> setenv XCELIUM_HOME <path_to_xcelium_executables> && setenv PATH $XCELIUM_HOME/bin:$PATH
> setenv LM_LICENSE_FILE <port>@<server>
```

For Synopsys VCS

```
> setenv VCS_HOME <path_to_vcs_executables> && setenv PATH $VCS_HOME/bin:$PATH
> setenv LM_LICENSE_FILE <port>@<server>
```

Note: FOUNDRY is mandatory for any flow that references Lattice libraries directly (see the Method 3: Using Command Line section) and for IP generation through ipgen from `radiantc`. `QUESTASIM_HOME` / `XCELIUM_HOME` / `VCS_HOME`, and

SALT_LICENSE_SERVER / LM_LICENSE_FILE are only required if they have not already been preconfigured by the system administrator at login.

After cmpl_libs has been used to compile the Lattice simulation libraries (through any of Methods 1, 2, or 3 described previously), the precompiled libraries can be used by a third-party simulator without recompilation. The exact mechanism differs between the third-party simulators.

For Non-OEM QuestaSim: Using Library Mappings in the Active modelsim.ini

When you simulate your design with a non-OEM version of the QuestaSim or ModelSim simulator, take note of the version dependencies of the simulator you are using. For example, libraries that are compiled for the OEM version of the ModelSim simulator, which is included in both the Radiant and Diamond software, are not directly compatible with a non-OEM version of the ModelSim simulator, even if it is the same release number. With this in mind, the first step when working with either the ModelSim or QuestaSim simulator is to compile the Lattice FPGA simulation libraries for your simulation tool version by pointing to the correct installation directory using the *cmpl_libs* script's *-sim_path* option.

```
> cmpl_libs -sim_path <questasim_installation_path>/questa_sim/<version>/questasim/bin \
-sim_vendor siemens \
-device <device_arch> \
-64 \
-target_path <user_defined_path>/questasim
```

```
[Library]
lifcl = C:/Files/lattice_libs/lifcl
pmi = C:/Files/lattice_libs/pmi_work
```

Figure 1.12. Library Mappings in an Active ModelSim.ini file for a Third-Party Version of the QuestaSim Software

The next step in the project simulation flow is to update your library mappings for the active simulation tool you are using. To do this, enter some additional lines under the *[Library]* tag of the *modelsim.ini* for your active simulation tool of choice. The purpose of these library mappings is to map a logical library (for example, lifcl) to the physical directory containing the compiled libraries.

After this modification has been completed, you can proceed to use any of these simulation libraries as any other kind of simulation library in the ModelSim or QuestaSim simulator. Based on the example in Figure 1.12, the correct usage in a scripted flow would be to call the ModelSim or QuestaSim command with the following options *-L lifcl -L uaplatform -L pmi* so that the ModelSim or QuestaSim simulator knows which simulation libraries to use when simulating a design.

```
> vsim -64 \
-gui \
-L <logical library name> \
-voptargs=+acc \
work.<top_tb_module> \
-l <log_file> \
-do "add wave *; run -all"
```

The following table lists the description for each setting used above in the vsim command:

Table 1.3. Non-OEM QuestaSim's Options and Descriptions Using Precompiled Libraries Method

Options	Descriptions
-64	Enables 64-bit compilation; required for Avant and newer Lattice device families.
-gui	Launches QuestaSim in graphical mode, opening the waveform viewer and IDE rather than running in batch or console-only mode.
-L	Links the logical library.
-voptargs=+acc	Passes the +acc argument to the optimizer, preserving signal accessibility for all objects during simulation; required for waveform probing and runtime visibility into the design hierarchy.

Options	Descriptions
work.<top_tb_module>	Specifies the top-level module to simulate. This top-level module must be compiled into work before using this option.
-l	Writes simulator output to the specified log file.
-do	Specifies a macro command string or .do script file to execute automatically upon simulator startup, allowing simulation steps to be scripted without manual input.
run -all	A simulator command passed via -do that starts the simulation and runs it until a `$finish` or `$stop` statement is encountered in the testbench, or indefinitely if neither statement is present.

For Cadence Xcelium: Using the -reflib Option

The `-reflib` option in the `xrun` command specifies a read-only reference library containing precompiled design units such as modules and IP blocks. This avoids recompilation of the Lattice device library on every run and is especially valuable for large or collaborative projects. To use it, first run cmpl_libs to produce the precompiled library, then point `xrun` at that target_path location:

```
> cmpl_libs -sim_path <xcelium_installation_path>/XCELIUM/<version>/Linux/tools.lnx86/bin \
-sim_vendor cadence \
-device <device_arch> \
-64 \
-target_path <user_defined_path>/xcelium

# Then in the xrun invocation:
> xrun -64bit \
-sv \
-gui \
-access +rwc \
-timescale 1ns/1ps \
-libext .v,.sv \
-l <log_file> \
-reflib <user_defined_path>/xcelium/<device_arch> \
<design_files> \
<testbench_files> \
-top <top_module>
```

The following table lists the description for each setting used above in the `xrun` command:

Table 1.4. Xcelium's Options and Descriptions Using Precompiled Libraries Method

Options	Descriptions
-64bit	Enables 64-bit compilation; required for Avant and newer Lattice device families.
-sv	Enables System Verilog support.
-gui	Launches SimVision automatically for waveform analysis.
-access +rwc	Enables read, write, and connection debug access for waveform probing.
-timescale 1ns/1ps	Sets the simulation time resolution.
-libext .v,.sv	Specifies the file extensions to search for under -y / library directories.
-l	Writes simulator output to the specified log file.
-reflib	Links the compiled device simulation library generated from cmpl_libs.
-top	Specifies the top-level module to simulate.

For Synopsys VCS: Using synopsys_sim.setup

When cmpl_libs is invoked with `-sim_vendor` synopsys, in addition to the precompiled libraries it also produces a *synopsys_sim.setup* file. This file is the central configuration resource for VCS simulations, mapping logical library names to

the precompiled library locations. VCS automatically detects *synopsys_sim.setup* in the working directory, so once the file is copied into the simulation directory, no library path needs to be specified on the vcs command line.

```
> cmpl_libs -sim_path <vcs_installation_path>/synopsys/VCS/<version>/bin \
-sim_vendor synopsys \
-device <device_arch> \
-64 \
-target_path <user_defined_path>/vcs

# Copy the generated setup file into the working simulation directory:
> cp <user_defined_path>/vcs/synopsys_sim.setup <working_simulation_dir>

# Now invoke vcs from <working_simulation_dir>; library paths are picked up automatically:
> vcs -full64 \
-sverilog \
-debug_access+all \
-timescale=1ns/1ps \
+libext+.v+.sv \
-o <user_defined_path>/simv \
-l <user_defined_path>/output.log \
<design_files> \
<testbench_files> \
-top <top_module>
```

The following table lists the description for each setting used above in the vcs command:

Table 1.5. VCS's Options and Descriptions Using Precompiled Libraries Method

Options	Descriptions
-full64	Enables 64-bit compilation; required for Avant and newer Lattice device families.
-sverilog	Enables System Verilog support.
-debug_access+all	Enables read, write, and connection debug access for waveform probing.
-timescale=1ns/1ps	Sets the simulation time resolution.
+libext+.v+.sv	Specifies the file extensions to search for under -y / library directories (add *.vh*, *.vhl*, *.svh* as needed)
-o	Names the output simulation executable.
-l	Writes simulator output to the specified log file.
-top	Specifies the top-level module to simulate.

Note: Starting with Radiant 2025.1, the -64 option should always be specified and is required for Avant and other newer Lattice device families. The cmpl_libs invocation and the simulator command must agree on the bit-width flag. If the simulator is invoked without the 64-bit option, cmpl_libs must also be run without it; otherwise, both invocations must include the option.

1.1.2. Manual Library Referencing Method

When cmpl_libs is not used—for example, when cmpl_libs support for a particular device-simulator combination has not yet been published, or when reusing existing on-disk libraries is preferred to save space—the Lattice simulation library can be referenced manually using +incdir / -incdir and -f command-line options on every simulator invocation. This bypasses the need to edit the *modelsim.ini* or use the *synopsys_sim.setup* and -reflib mechanisms entirely.

Trade-off: This method is simpler to set up but slower than the precompiled flow because the library is compiled on every run.

Manual referencing requires the FOUNDRY environment variable (introduced in the Method 3: Using Command Line section) to point to the Lattice Radiant software installation so that the simulator can resolve Lattice-specific IP and primitives.

The libraries to be referenced reside under the cae_library and ip/pmi directories of the Radiant software installation. The general structure for the simulator command line is shown below for each tool.

For Non-OEM QuestaSim

```
# Compile device simulation library, RTL and Testbench into work
> vlog -64 \
-sv \
-timescale 1ns/1ps \
+incdir+<radiant_installation_path>/cae_library/simulation/verilog/<device_arch> \
+incdir+<radiant_installation_path>/cae_library/simulation/verilog/<uaplatform | applatform> \
+incdir+<radiant_installation_path>/ip/pmi \
-f <radiant_installation_path>/cae_library/simulation/verilog/<device_arch>/<device_arch>.f \
-f <radiant_installation_path>/cae_library/simulation/verilog/<uaplatform | applatform>/<uaplatform |
applatform>.f \
-f <radiant_installation_path>/ip/pmi/pmi.f \
+incdir+<tb_include_path> \
<design_files> \
<testbench_files>

# Run simulation
> vsim -64 \
-gui \
-voptargs=+acc \
work.<top_module> \
-l <log_file> \
-do "add wave *; run -all"
```

The following table lists the description for each setting used above in the `vlog` and `vsim` command:

Table 1.6. Non-OEM QuestaSim's Options and Descriptions Using Manual Library Referencing Method

Options	Descriptions
-64	Enables 64-bit compilation; required for Avant and newer Lattice device families.
-sv	Enables System Verilog support.
-timescale 1ns/1ps	Sets the simulation time resolution.
+incdir	Specifies where to find the header files referenced by include statements.
-f	Points to the .f of the device simulation libraries and instructs the simulator to read the list of source files from it, ensuring all required library files are compiled in the correct order.
-gui	Launches QuestaSim in graphical mode, opening the waveform viewer and IDE rather than running in batch or console-only mode.
-voptargs=+acc	Passes the +acc argument to the optimizer, preserving signal accessibility for all objects during simulation; required for waveform probing and runtime visibility into the design hierarchy.
work.<top_module>	Specifies the top-level module to simulate.
-l	Writes simulator output to the specified log file.
-do	Specifies a macro command string or .do script file to execute automatically upon simulator startup, allowing simulation steps to be scripted without manual input.
run -all	A simulator command passed via -do that starts simulation and runs it until a `$finish` or `$stop` statement is encountered in the testbench, or indefinitely if neither statement is present.

Note: For manual library referencing, use both `+incdir` to include the device library folder and `-f` to point to the source file lists of the device library. `uaplatform` applies to Nexus devices; `applatform` applies to Avant and Mach-NX devices. Select whichever matches the target architecture.

For Cadence Xcelium

A typical xrun invocation combines tool options, library references, design and testbench source files, and the simulation top module:

```
> xrun -64bit \
-gui \
-ALLOWREDEFINITION \
-sv \
-access +rwc \
-timescale 1ns/1ps \
-libext .v,.sv \
-l <log_file> \
-incdir <radiant_installation_path>/cae_library/simulation/verilog/<device_arch> \
-incdir <radiant_installation_path>/cae_library/simulation/verilog/<uaplatform | applatform> \
-incdir <radiant_installation_path>/ip/pmi \
-f <radiant_installation_path>/cae_library/simulation/verilog/<device_arch>/<device_arch>.f \
-f <radiant_installation_path>/cae_library/simulation/verilog/<uaplatform | applatform>/<uaplatform |
applatform>.f \
-f <radiant_installation_path>/ip/pmi/pmi.f \
<design_files> \
<testbench_files> \
-top <top_module>
```

The following table lists the description for each setting used above in the xrun command:

Table 1.7. Xcelium's Options and Descriptions Using Manual Library Referencing Method

Options	Descriptions
-64bit	Enables 64-bit compilation; required for Avant and newer Lattice device families.
-gui	Launches SimVision automatically for waveform analysis.
-ALLOWREDEFINITION	Permits recompilation and redefinition of duplicate modules, packages, and constructs that would otherwise cause errors.
-sv	Enables System Verilog support.
-access +rwc	Enables read, write, and connection debug access for waveform probing.
-timescale 1ns/1ps	Sets the simulation time resolution.
-libext .v,.sv	Specifies the file extensions to search for under -y / library directories.
-incdir	Specifies where to find the header files referenced by include statements.
-f	Points to the .f file of the device simulation libraries and instructs the simulator to read the list of source files from it, ensuring all required library files are compiled in the correct order.
-l	Writes simulator output to the specified log file.
-top	Specifies the top-level module to simulate.

Note: For manual library referencing, use both -incdir to include the device library folder and -f to point to the source file lists of the device library.

For Synopsys VCS

A typical VCS invocation combines tool options, library references, design and testbench source files, and the simulation top module:

```
> vcs -full64 \
-sverilog \
-debug_access+all \
-timescale=1ns/1ps \
+libext+.v+.sv \
```

```
+incdir+<radiant_installation_path>/cae_library/simulation/verilog/<device_arch> \
+incdir+<radiant_installation_path>/cae_library/simulation/verilog/<uaplatform | applatform> \
+incdir+<radiant_installation_path>/ip/pmi \
-f <radiant_installation_path>/cae_library/simulation/verilog/<device_arch>/<device_arch>.f \
-f <radiant_installation_path>/cae_library/simulation/verilog/<uaplatform | applatform>/<uaplatform | applatform>.f \
-f <radiant_installation_path>/ip/pmi/pmi.f \
-o <user_defined_path>/simv \
-l <log_file> \
<design_files> \
<testbench_files> \
-top <top_module>
```

The following table lists the description for each setting used above in the `vcs` command:

Table 1.8. VCS's Options and Descriptions Using Manual Library Referencing Method

Options	Descriptions
-full64	Enables 64-bit compilation; required for Avant and newer Lattice device families.
-sverilog	Enables System Verilog support.
-debug_access+all	Enables read, write, and connection debug access for waveform probing.
-timescale=1ns/1ps	Sets the simulation time resolution.
+libext+.v+.sv	Specifies the file extensions to search for under -y / library directories. (add *.vh, .vhl, .svh* as needed)
+incdir	Specifies where to find the header files referenced by include statements.
-f	Points to the *.f* file of the device simulation libraries and instructs the simulator to read the list of source files from it, ensuring all required library files are compiled in the correct order.
-o	Names the output simulation executable.
-l	Writes simulator output to the specified log file.
-top	Specifies the top-level module to simulate.

Note: For manual library referencing, use both `-incdir` to include the device library folder and -f to point to the source file lists of the device library. `vlogan` can be used as a separate compile step for incremental flows; `vcs` then performs elaboration and produces `simv`. On VCS, the `-error=noMPD` option demotes a missing port declaration error to a warning when working with legacy code; the Xcelium equivalent is `-nowarn` UIMPTD.

Because the library is recompiled on every run, this flow is appreciably slower than the precompiled flow described in the Precompiled Libraries Using cmpl_libs (Recommended) section, particularly for large IP such as PCIe®. Prefer the precompiled flow whenever cmpl_libs supports the target device-simulator combination.

2. Lattice FPGA Simulation Work Examples with Third-Party Tools

This section describes some work examples using the third-party simulation tools.

2.1. QuestaSim Work Examples

2.1.1. Example 1: PLL Foundation IP QuestaSim Simulation Using Precompiled Libraries (Avant-G70)

This flow demonstrates the precompiled-library approach for an Avant-G70 PLL Foundation IP QuestaSim simulation.

1. Set the environment variables (FOUNDRY is mandatory; QUESTASIM_HOME and SALT_LICENSE_SERVER are optional) as described in the Method 3: Using Command Line section.
2. Launch the Radiant Command-Line Interface to generate the IP:

```
# Launch Radiant command-line interface
> radiantc

# Compile device simulation library for LAV-AT (G70)
% cmpl_libs -sim_path <questasim_installation_path>/questa_sim/2025.3/questasim/bin -sim_vendor
siemens -device lav_atg -64 -target_path <user_defined_path>/compile_library/questasim

# Generate PLL IP
% ipgen -o <user_defined_path>/pll_avantIP -ip <radiant_installation_path>/ip/avant/pll_avant -name
pll_avant0 -a LAV-AT -p LAV-AT-G70 -t LFG676 -sp 1 -op COM
```

3. Inspect the cmpl_libs log files generated in step 2 and confirm there are no compilation errors.
4. Add the compiled library path to the [Library] section of the active *modelsim.ini* file.

```
17 [Library]
18 std = $QUESTASIM_DIR/../std
19 ieee = $QUESTASIM_DIR/../ieee
20 vital2000 = $QUESTASIM_DIR/../vital2000
21 lav_atg = /lsc/scratch/sdc/jlee5/test_questa_cmpl_libs/compile_library/questasim/lav_atg
```

Figure 2.1. Modifying [Library] Section in the Active modelsim.ini

Note: Refer to the Managing Libraries and Settings section in the QuestaSim Lattice Edition Usage Guidelines and Tips Application Notes (FPGA-AN-02053) for details on locating the active *modelsim.ini* file.

5. Launch QuestaSim now (optional) or proceed with terminal commands and launch the GUI only in step 7 through the -GUI option in the vsim command.

```
# Launch QuestaSim GUI
> vsim &
```

6. In QuestaSim, navigate to **File > Change Directory** and select the directory to run the simulation.
7. Execute the following commands in the QuestaSim transcript window (or terminal):

```
# Create a new work library in the current directory
> vlib work

# Map the logical library name ‘work’ to physical directory ‘work’
> vmap work work

# Compile RTL and testbench files (by default, it will be compiled to work)
> vlog -64 \
-sv \
-timescale 1ns/1ps \
+incdir+<user_defined_path>/pll_avantIP/testbench \
<user_defined_path>/pll_avantIP/rtl/pll_avant0.v \
```

```
<user_defined_path>/pll_avantIP/testbench/tb_top.v

# Run simulation with compiled libraries, -L links to the logical library. Update the logical library
(lav_atg) in the active modelsim.ini to point to the device simulation library compiled with cmpl_libs
> vsim -64 \
-L work \
-L lav_atg \
-voptargs=+acc \
work.tb_top \
-l <user_defined_path>/vsim.log \
-do "add wave *;run -all"
```

2.1.2. Example 2: PLL Foundation IP QuestaSim Simulation via Manual Library Referencing (MachXO5)

When cmpl_libs is unavailable or not desired, the manual library-referencing flow described in the Manual Library Referencing Method section can be used. This example targets a MachXO5 PLL Foundation IP for QuestaSim simulation.

1. Set the environment variables (FOUNDRY is mandatory; QUESTASIM_HOME and SALT_LICENSE_SERVER are optional) as described in the Method 3: Using Command Line section.
2. Launch the Radiant Command-Line Interface to compile the device simulation library and generate the IP:

```
# Launch Radiant command-line interface
> radiantc

# Generate PLL IP
% ipgen -o <user_defined_path>/pllIP -ip <radiant_installation_path>/ip/lifcl/pll -name pll0 -a LFMXO5
-p LFMXO5-25 -t BBG256 -sp 9_High-Performance_1.0V -op COM
```

3. Launch QuestaSim now (optional) or proceed with terminal commands and launch the GUI only in step 7 through the -GUI option in the vsim command.

```
# Launch QuestaSim GUI
> vsim &
```

4. In QuestaSim, navigate to **File > Change Directory** and select the directory to run the simulation in the transcript window:

```
# Create a new work library in the current directory
> vlib work

# Map the logical library name ‘work’ to physical directory ‘work’
> vmap work work

# Compile VHDL device library file
> vcom -64 \
-work work \
<radiant_installation_path>/rtf/cae_library/synthesis/vhdl/lfmxo5.vhd

# Compile device library, RTL and testbench files
> vlog -64 \
-sv \
-mfcu \
-timescale 1ns/1ps \
-work work \
+incdir+<radiant_installation_path>/cae_library/simulation/verilog/lfmxo5 \
+incdir+<radiant_installation_path>/cae_library/simulation/verilog/uaplatform \
+incdir+<radiant_installation_path>/ip/pmi \
-f <radiant_installation_path>/cae_library/simulation/verilog/lfmxo5/lfmxo5.f \
-f <radiant_installation_path>/cae_library/simulation/verilog/uaplatform/uaplatform.f \
-f <radiant_installation_path>/ip/pmi/pmi.f \
```

```
+incdir+<user_defined_path>/pllIP/testbench \
<user_defined_path>/pllIP/rtl/pll0.v \
<user_defined_path>/pllIP/testbench/tb_top.v

# Launch simulation
> vsim -64 \
-L work \
-voptargs=+acc \
work.tb_top \
-l <user_defined_path>/vsim.log \
-do "add wave *;run -all"
```

2.2. Xcelium Work Examples

2.2.1. Example 1: PLL Foundation IP Xcelium Simulation Using Precompiled Libraries (Avant-G70)

This flow demonstrates the precompiled-library approach for an Avant-G70 PLL Foundation IP Xcelium simulation.

1. Set the environment variables (FOUNDRY is mandatory; XCELIUM_HOME and LM_LICENSE_FILE are optional) as described in the Method 3: Using Command Line section.
2. Launch the Radiant Command-Line Interface to compile the device simulation library and generate the IP:

```
# Launch Radiant command-line interface
> radiantc

# Compile device simulation library for LAV-AT (G70)
% cmpl_libs -sim_path <cadence_installation_path>/XCELIUM/XCELIUM23.03.003/Linux/tools.lnx86/bin -
sim_vendor cadence -device lav_atg -64 -target_path <user_defined_path>/compile_library/xcelium

# Generate PLL IP
% ipgen -o <user_defined_path>/pll_avantIP -ip <radiant_installation_path>/ip/avant/pll_avant -name
pll_avant0 -a LAV-AT -p LAV-AT-G70 -t LFG676 -sp 1 -op COM
```

3. Inspect the cmpl_libs log files generated in step 2 and confirm there are no compilation errors.
4. Run the xrun command referencing the precompiled library through -reflib:

```
# Run Xcelium simulation; -reflib links to the precompiled device library compiled in Step 2
> xrun -64bit \
-sv \
-gui \
-access +rwc \
-timescale 1ns/1ps \
-libext .v,.sv \
-l <user_defined_path>/pll0_xrun_reflib.log \
-reflib <user_defined_path>/compile_library/xcelium/lav_atg \
-incdir <user_defined_path>/pll_avantIP/testbench \
<user_defined_path>/pll_avantIP/rtl/pll_avant0.v \
<user_defined_path>/pll_avantIP/testbench/tb_top.v \
-top tb_top
```

5. Verify the simulation executable is generated and that SimVision opens automatically with -gui for waveform analysis.

2.2.2. Example 2: PLL Foundation IP Xcelium Simulation via Manual Library Referencing (MachXO5)

When cmpl_libs is unavailable or not desired, the manual library-referencing flow described in the Manual Library Referencing Method section can be used. This example targets a MachXO5 PLL Foundation IP for Xcelium simulation.

1. Set the environment variables (FOUNDRY is mandatory; XCELIUM_HOME and LM_LICENSE_FILE are optional) as described in the Method 3: Using Command Line section.
2. Launch the Radiant Command-Line Interface and generate the IP:
```
# Launch Radiant command-line interface
> radiantc

# Generate PLL IP
% ipgen -o <user_defined_path>/pllIP -ip <radiant_installation_path>/ip/lifcl/pll -name pll0 -a LFMXO5
-p LFMXO5-25 -t BBG256 -sp 9_High-Performance_1.0V -op COM
```
3. Run the xrun command. Use -incdir and -f options to reference the cae_library and pmi directories directly.
```
# Run Xcelium simulation
> xrun -64bit \
-gui \
-ALLOWREDEFINITION \
-sv \
-access +rwc \
-timescale 1ns/1ps \
-libext .v,.sv \
-l <user_defined_path>/pll0.log \
-incdir <radiant_installation_path>/cae_library/simulation/verilog/lfmxo5 \
-incdir <radiant_installation_path>/cae_library/simulation/verilog/uaplatform \
-incdir <radiant_installation_path>/ip/pmi \
-f <radiant_installation_path>/cae_library/simulation/verilog/lfmxo5/lfmxo5.f \
-f <radiant_installation_path>/cae_library/simulation/verilog/uaplatform/uaplatform.f \
-f <radiant_installation_path>/ip/pmi/pmi.f \
-incdir <user_defined_path>/pllIP/testbench/ \
<user_defined_path>/pllIP/rtl/pll0.v \
<user_defined_path>/pllIP/testbench/tb_top.v \
-top tb_top
```
4. Verify the simulation executable is generated and that SimVision opens with -gui for waveform analysis.

2.3. VCS Work Examples

2.3.1. Example 1: PLL Foundation IP VCS Simulation Using Precompiled Libraries (Avant-G70)

This flow demonstrates the precompiled-library approach for an Avant-G70 PLL Foundation IP VCS simulation.

1. Set the environment variables (FOUNDRY is mandatory; VCS_HOME and LM_LICENSE_FILE are optional) as described in the Method 3: Using Command Line section.
2. Launch the Radiant Command-Line Interface to compile the device simulation library and generate the IP:
```
# Launch Radiant command-line interface
> radiantc

# Compile device simulation library for LAV-AT (G70)
% cmpl_libs -sim_path <synopsys_installation_path>/synopsys/VCS/W-2024.09-1/bin -sim_vendor synopsys -
device lav_atg -64 -target_path <user_defined_path>/vcs

# Generate PLL IP
% ipgen -o <user_defined_path>/pll_avantIP -ip <radiant_installation_path>/ip/avant/pll_avant -name
pll_avant0 -a LAV-AT -p LAV-AT-G70 -t LFG676 -sp 1 -op COM
```
3. Inspect the cmpl_libs log files generated in step 2 and confirm there are no compilation errors.

4. Copy *synopsys_sim.setup* into the working simulation directory (see the For Synopsys VCS: Using synopsys_sim.setup section).
5. Run vlogan to compile the design files, then vcs to elaborate and produce the simv executable, and launch DVE/Verdi for waveform analysis:

```
# Compile RTL and testbench files
vlogan -full64 \
-sverilog \
-timescale=1ns/1ps \
+libext+.v+.sv \
-l <user_defined_path>/pll_vcs_cmpl_lib.log \
-work work \
+incdir+<user_defined_path>/pll_avantIP/testbench \
<user_defined_path>/pll_avantIP/rtl/pll_avant0.v \
<user_defined_path>/pll_avantIP/testbench/tb_top.v

# Elaborate and link; -o specifies the simulation executable name
> vcs -full64 \
-sverilog \
-debug_access+all \
-timescale=1ns/1ps \
-o <user_defined_path>/pll_vcs_simv \
-l <user_defined_path>/pll_vcs_sim.log \
-top tb_top

# Launch DVE/Verdi for waveform analysis
> <user_defined_path>/pll_vcs_simv -gui
```

2.3.2. Example 2: PLL Foundation IP VCS Simulation via Manual Library Referencing (MachXO5)

When cmpl_libs is unavailable or not desired, the manual library-referencing flow described in the Manual Library Referencing Method section can be used. This example targets a MachXO5 PLL Foundation IP and bypasses *synopsys_sim.setup* entirely.

1. Set the environment variables (FOUNDRY is mandatory; VCS_HOME and LM_LICENSE_FILE are optional) as described in the Method 3: Using Command Line section.
2. Launch the Radiant Command-Line Interface and generate the IP:

```
# Launch Radiant command-line interface
> radiantc
# Generate PLL IP
% ipgen -o <user_defined_path>/pllIP -ip <radiant_installation_path>/ip/lifcl/pll -name pll0 -a LFMXO5
-p LFMXO5-25 -t BBG256 -sp 9_High-Performance_1.0V -op COM
```

3. Run vcs. Use +incdir and -f directives to reference the cae_library and pmi directories directly, and launch DVE/Verdi for waveform analysis.

```
# Combines compilation and elaboration into a single command
> vcs -full64 \
-sverilog \
-debug_access+all \
-timescale=1ns/1ps \
+libext+.v+.sv \
+incdir+<radiant_installation_path>/cae_library/simulation/verilog/lfmxo5 \
+incdir+<radiant_installation_path>/cae_library/simulation/verilog/uaplatform \
+incdir+<radiant_installation_path>/ip/pmi \
-f <radiant_installation_path>/cae_library/simulation/verilog/lfmxo5/lfmxo5.f \
-f <radiant_installation_path>/cae_library/simulation/verilog/uaplatform/uaplatform.f \
-f <radiant_installation_path>/ip/pmi/pmi.f \
-o <user_defined_path>/pll_simv \
```

```
-l <user_defined_path>/pll0.log \
+incdir+<user_defined_path>/pllIP/testbench/ \
<user_defined_path>/pllIP/rtl/pll0.v \
<user_defined_path>/pllIP/testbench/tb_top.v \
-top tb_top

# Launch DVE/Verdi for waveform analysis
> <user_defined_path>/pll_simv -gui
```

3. Common Pitfalls

This section describes the common pitfalls that you might encounter when simulating your designs with third-party simulation software.

3.1. Pitfall Example 1: Simulating a Protected Component

You may encounter the following error message or some other reference to a protected component during simulation:

*Hierarchical component lookup failed for '{*Name Protected*}' at '{*Name Protected*}'*

The following describes the error message and how to avoid or resolve the error:

- Certain IP or primitives require the instantiation of global set/reset (GSR) and power-up reset (PUR) to function correctly.
- Some of the RTL for these primitives within the associated Lattice simulation library are encrypted, which prevents you from debugging the error further.
- Ensure that both the GSR and PUR are instantiated in the top-level testbench for the project being simulated. For more information, refer to the GSR and PUR sections of the Lattice Radiant Software Help and Lattice Diamond Software Help.

3.2. Pitfall Example 2: Incorrect Compilation of Simulation Libraries

You may encounter the following error messages when there is an incorrect compilation of simulation libraries or incorrect library references during simulation:

- *Design instance not found*
- *Could not resolve reference to…*
- *Unresolved modules…*
- *Hierarchical component lookup failed at …*
- *… of design unit … is unresolved in …*

The following describes the error message and how to avoid or resolve the error:

- If *cmpl_libs* was used to compile the simulation libraries for the target third-party simulation tool of choice, check the *<device name>.log* file that was generated for each compiled library being used. A log file is generated for every device library that was compiled. Review this log to ensure that all device libraries were created and compiled as expected.
- Simulation library contents vary from device to device. If you compiled multiple libraries using *cmpl_libs*, ensure that you are pointing to the correct ones when invoking your simulation tool of choice. In general, you will need to point to *pmi*, *uaplatform*, as well as the device library corresponding to the device of choice for your project (for example, *lfcpnx* for Lattice CertusPro™-NX device family).
- Ensure that all required device libraries are linked during simulation (for example, *lfcpnx*, *pmi*, *uaplatform*). Alternatively, if you are using non-precompiled libraries, ensure that you are using an option which points to the source files for these simulation libraries so your simulation tool of choice can resolve these references.
- Depending on the IP used in a project, there may be some additional simulation-specific files that must be included for simulation. Double-check the IP user guide for the IP you are simulating in your project to ensure that no files are omitted when invoking your simulation tool of choice.

If none of the steps above work, try pointing to the physical RTL, which is included with each base Lattice tool installation using options such as *-incdir* or *-y*. *-incdir* and *-y* are options that are supported by some simulation tools, which can be used to resolve missing module references by pointing directly to files or directories containing those missing references. For example, *-incdir /home/lattice/lscc/radiant/2023.1/cae_library/simulation/lfmxo5*.

3.3. Pitfall Example 3: Incorrect Timescale Settings

You may encounter the following error messages or other similar warnings when there is an incorrect definition or setting of the timescale, which may lead to functional failure during simulation:

- *Unnamed generate block….*
- *Timescale is not defined…*

To avoid or resolve this error, ensure that the correct timescale is set in all the associated testbench files in your simulation environment.

References

- Scripting Lattice FPGA Build Flow Application Note (FPGA-AN-02073)
- Lattice Radiant Software web page
- Lattice Radiant Software Help
- Lattice Insights web page for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.2, August 2026

Section	Change Summary
All	Performed minor editorial and formatting edits.
Abbreviations in This Document	Added the following abbreviations to this section: • PLL • TCL • GUI

Introduction	• Updated this section with minor wording corrections and reformatted the content into bullet lists for improved readability.
	• Updated QuestaSim product reference to Siemens® QuestaSim.
	• Restructured the Compiling Lattice Simulation Libraries section from three console or command-line methods into two approaches.
	• Updated Table 1.1. Values and Descriptions for Each cmpl_libs Option.
	• Updated Table 1.2. Device Family Names for cmpl_libs –device Setting with new MachXO4, Lattice Nexus 2, and expanded Avant device entries.
	• Precompiled Libraries Using cmpl_libs (Recommended)Added the following tables:
	  • Table 1.3. Non-OEM QuestaSim's Options and Descriptions Using Precompiled Libraries Method.
	  • Table 1.4. Xcelium's Options and Descriptions Using Precompiled Libraries Method.
	  • Table 1.5. VCS's Options and Descriptions Using Precompiled Libraries Method.
	  • Table 1.6. Non-OEM QuestaSim's Options and Descriptions Using Manual Library Referencing Method.
	  • Table 1.7. Xcelium's Options and Descriptions Using Manual Library Referencing Method.
	  • Table 1.8. VCS's Options and Descriptions Using Manual Library Referencing Method.
	• Added the Manual Library Referencing Method section.
	• Updated the generic syntax for setting the variables for Linux in the Method 3: Using Command Line section.
	• Updated the title for Figure 1.12. Library Mappings in an Active ModelSim.ini file for a Third-Party Version of the QuestaSim Software.

Lattice FPGA Simulation Work Examples with Third-Party Tools	• Renamed this section title from *Lattice FPGA Simulation with Third-Party Tools* to *Lattice FPGA Simulation Work Examples with Third-Party Tools*.
	• Renamed and expanded the following subsections with detailed simulation workflow guidance for each tool:
	  • QuestaSim Work Examples section.
	  • Xcelium Work Examples section.
	  • VCS Work Examples section.

Common Pitfalls	Updated the Pitfall Example 2: Incorrect Compilation of Simulation Libraries section.

Revision 1.1, June 2024

Section	Change Summary

Introduction	• Updated the Introduction section.
	• Updated the Compiling Lattice Simulation Libraries section.
	• Added the following subsections in the Compiling Lattice Simulation Libraries section:
	  • Method 1: Using the Integrated TCL Console Method 1: Using the Integrated TCL Console section.
	  • Method 2: Using the Standalone TCL Console section.
	  • Method 3: Using Command Line section.

Section	Change Summary
Lattice FPGA Simulation with Third-Party Tools	• Updated the Cadence Xcelium Simulator section. • Updated the following sentence in the Synopsys VCS Simulator section: *From that point onward, you can proceed as normal to simulate your Lattice FPGA project.*
References	Added a new reference—Scripting Lattice FPGA Build Flow Application Note (FPGA-AN-02073).

Revision 1.0, March 2024

Section	Change Summary
All	Initial release.

www.latticesemi.com