# Simulating Designs for Lattice FPGA Devices

Lattice Semiconductor Corporation
5555 NE Moore Court
Hillsboro, OR 97124
(503) 268-8000

May 2007

## Copyright



## Trademarks



## Disclaimers

latest version of the relevant information to establish, before ordering, that the information being relied upon is current.

## Type Conventions Used in This Document

| Convention | Meaning or Use |
|---|---|
| **Bold** | Items in the user interface that you select or click. Text that you type into the user interface. |
| <*Italic*> | Variables in commands, code syntax, and path names. |
| **Ctrl+L** | Press the two keys at the same time. |
| `Courier` | Code examples. Messages, reports, and prompts from the software. |
| `...` | Omitted material in a line of code. |
| `.`<br>`.`<br>`.` | Omitted lines in code and report examples. |
| `[ ]` | Optional items in syntax descriptions. In bus specifications, the brackets are required. |
| `( )` | Grouped items in syntax descriptions. |
| `{ }` | Repeatable items in syntax descriptions. |
| `\|` | A choice between items in syntax descriptions. |

# Contents

# Simulating Designs for Lattice FPGA Devices

This document explains how to use Synopsys® VCS®, Cadence® NC-Verilog®, Cadence NC-VHDL®, and Aldec Riviera Pro®, and Aldec Active-HDL® software to simulate designs that target Lattice Semiconductor FPGAs. It shows you how to use these simulators to perform functional register-transfer-level (RTL) simulation, post-map simulation, and place-and-route gate-level simulation with and without timing simulation.

**Note**

Lattice Semiconductor does not supply the Synopsys VCS, Cadence NC-Verilog, Cadence NC-VHDL, Aldec Riviera Pro, or Aldec Active-HDL simulators. You must obtain them independently.

## Version Compatibility

Lattice Semiconductor ispLEVER software is compatible with the following versions of Synopsys VCS, Cadence NC-Verilog, Cadence NC-VHDL, and Aldec.

### Synopsys VCS

Lattice Semiconductor ispLEVER software version 7.0 is compatible with Synopsys VCS version 2006.06 on the Solaris and Linux operating systems.

### Cadence NC-Verilog

Lattice Semiconductor ispLEVER software version 7.0 is compatible with Cadence NC-Verilog version 5.83 on the Solaris, Linux, and Windows operating systems.

### Cadence NC-VHDL

Lattice Semiconductor ispLEVER software version 7.0 is compatible with Cadence NC-VHDL version 5.83 on the Solaris, Linux, and Windows operating systems.

### Aldec Riviera Pro

Lattice Semiconductor ispLEVER software version 7.0 is compatible with Aldec Riviera (Pro) version 2007.06 on the Solaris, Linux, and Windows operating systems.

### Aldec Active-HDL

Lattice Semiconductor ispLEVER software version 7.0 is compatible with Aldec Active-HDL version 7.2 SP2 on Windows only.

## Verilog Library Files

The Verilog functional and behavioral simulation library files are installed with the ispLEVER software in the locations shown in Table 1.

**Table 1: Verilog Functional and Behavioral Simulation Library File Locations**

| FPGA Family | Library Location |
|---|---|
| LatticeEC | <*path_to_ispLEVER_installation*>/cae_library/simulation/verilog/ec |
| LatticeECP-DSP | <*path_to_ispLEVER_installation*>/cae_library/simulation/verilog/ecp |
| LatticeECP2-DSP | <*path_to_ispLEVER_installation*>/cae_library/simulation/verilog/ecp2 |
| LatticeXP | <*path_to_ispLEVER_installation*>/cae_library/simulation/verilog/xp |
| LatticeXP2 | <*path_to_ispLEVER_installation*>/cae_library/simulation/verilog/xp2 |
| LatticeSC | <*path_to_ispLEVER_installation*>/cae_library/simulation/verilog/sc |
| MachXO | <*path_to_ispLEVER_installation*>/cae_library/simulation/verilog/machxo |

# VHDL Library Files

The VHDL functional and behavioral simulation library files are installed with the ispLEVER software in the locations shown in Table 2.

**Table 2: VHDL Functional and Behavioral Simulation Library File Locations**

| FPGA Family | Library Location |
|---|---|
| LatticeEC | <*path_to_ispLEVER_installation*>/cae_library/simulation/vhdl/ec/src |
| LatticeECP-DSP | <*path_to_ispLEVER_installation*>/cae_library/simulation/vhdl/ecp/src |
| LatticeECP2-DSP | <*path_to_ispLEVER_installation*>/cae_library/simulation/vhdl/ecp2/src |
| LatticeXP | <*path_to_ispLEVER_installation*>/cae_library/simulation/vhdl/xp/src |
| LatticeXP2 | <*path_to_ispLEVER_installation*>/cae_library/simulation/vhdl/xp2/src |
| LatticeSC | <*path_to_ispLEVER_installation*>/cae_library/simulation/vhdl/sc/src |
| MachXO | <*path_to_ispLEVER_installation*>/cae_library/simulation/vhdl/machxo/src |

# Performing Simulation with Synopsys VCS

This section explains how to perform simulation with Synopsys VCS.

## Functional RTL Simulation

Use the following `vcs` command to perform a functional RTL simulation with one of the libraries shown in Table 1:

```
vcs -RI <design_name>.v <test_bench>.v
-y <Lattice_verilog_library_flocation> +libext+.v +v2k
```

## Post-Map and Place-and-Route Gate-Level Simulation

Use the following `vcs` command to perform post-map and place-and-route gate-level simulation:

```
vcs -RI <design_name>.v <test_bench>.vo
-y <Lattice_verilog_library_location> +libext+.v +v2k
```

## Post-Map and Place-and-Route Gate-Level Simulation with Timing

Use the following procedure to perform post-map and place-and-route gate-level simulation with timing:

1. Add the `sdf_annotate` line at the top-level test bench:

```
$sdf_annotate ("<design_name>.sdf", <instance_name>,
,<sdf_log>, "MAXIMUM");
```

2. Use the following `vcs` command to perform the simulation:

```
vcs -RI <design_name>.v <test_bench>.vo
-y <Lattice_verilog_library_location> +libext+.v +v2k
```

# Performing Simulation with Cadence NC-Verilog

This section explains how to perform simulation with Cadence NC-Verilog.

## Setting Lattice Semiconductor Libraries

Before simulating Lattice Semiconductor FPGA designs in the Cadence NC-Verilog simulator, you must perform the following steps to specify the Lattice Semiconductor simulation libraries.

### Creating Library Definition Files

Before using the NC-Verilog simulator to simulate your design project, you must first create two library definition files named hdl.var and cds.lib in your project folder. The hdl.var and cds.lib files define which libraries are accessible and where they are located. The hdl.var file contains statements that map logical library names to design library names, and the cds.lib file contains statements that map design library names to their physical directory paths.

For example, your local hdl.var file can include the following:

```
DEFINE EC ec_vlog
DEFINE ECP ecp_vlog
DEFINE XP xp_vlog
DEFINE MachXO machxo_vlog
DEFINE SC sc_vlog
DEFINE ECP2 ecp2_vlog
DEFINE XP2 xp2_vlog
```

Your local cds.lib file can include the following:

```
DEFINE ec_vlog <compile_dir>/ec_vlog
DEFINE ecp_vlog <compile_dir>/ecp_vlog
DEFINE xp_vlog <compile_dir>/xp_vlog
DEFINE machxo_vlog <compile_dir>/machxo_vlog
DEFINE sc_vlog <compile_dir>/sc_vlog
DEFINE ecp2_vlog <compile_dir>/ecp2_vlog
DEFINE xp2_vlog <compile_dir>/xp2_vlog
```

**Note**

The <*compile_dir*> variable specifies the location of the compiled files.

Cadence provides a utility called NCLaunch to set up the necessary initialization files and to compile the Verilog source libraries. NCLaunch is available as part of the 2.1 and later releases. Otherwise, setting up the initialization files and compiling the Verilog source libraries is a manual process.

You can create the hdl.var and cds.lib files with any text editor. You must map the physical locations to the logical names before proceeding to the next step. On the UNIX and Linux platforms, you can create these names by using the `mkdir` command as follows:

```
mkdir -p <compile_dir>/ec_vlog
mkdir -p <compile_dir>/ecp_vlog
mkdir -p <compile_dir>/xp_vlog
mkdir -p <compile_dir>/machxo_vlog
mkdir -p <compile_dir>/sc_vlog
mkdir -p <compile_dir>/ecp2_vlog
mkdir -p <compile_dir>/xp2_vlog
```

If you want the logical library names to be available for all designs, use `INCLUDE` or `SOFTINCLUDE` in the location of your master hdl.var and cds.lib files.

For example, you can add the following line to your master cds.lib file:

```
INCLUDE $path/cds.lib
```

**Note**

The *path* variable specifies the location of your own cds.lib file.

Or, you can directly include the library definition in the master hdl.var and cds.lib files.

Now the master hdl.var and cds.lib files include the Lattice Semiconductor library definitions. The next time that you want to simulate Lattice Semiconductor designs in the NC-Verilog simulator, you do not need to create you own hdl.var and cds.lib files again.

## Parsing and Analyzing Lattice Semiconductor Simulation Libraries

After creating your own hdl.var and cds.lib files, you must parse and analyze the Lattice Semiconductor simulation libraries by using the NC-Verilog simulator. These libraries must be parsed and analyzed only once.

To parse and analyze Lattice Semiconductor Verilog simulation libraries, run the following commands in the NC-Verilog simulator:

- LatticeEC Verilog:

```
ncvlog -messages -work ec_vlog
$isplever_install_path/cae_library/simulation/verilog/ec/*.v
```

- LatticeECP Verilog:

```
ncvlog -messages -work ecp_vlog
$isplever_install_path/cae_library/simulation/verilog/ecp/
*.v
```

- LatticeECP2 Verilog:

```
ncvlog -messages -work ecp2_vlog
$isplever_install_path/cae_library/simulation/verilog/ecp2/
*.v
```

- LatticeXP Verilog:

```
ncvlog -messages -work xp_vlog
$isplever_install_path/cae_library/simulation/verilog/xp/*.v
```

- LatticeXP2 Verilog:

```
ncvlog -messages -work xp2_vlog
$isplever_install_path/cae_library/simulation/verilog/xp2/
*.v
```

- MachXO Verilog:

```
ncvlog -messages -work machxo_vlog
$isplever_install_path/cae_library/simulation/verilog/
machxo/*.v
```

- LatticeSC Verilog:

```
ncvlog -messages -work sc_vlog
$isplever_install_path/cae_library/simulation/verilog/sc/*.v
```

## Functional RTL Simulation

Use the following commands to perform a functional RTL simulation with one of the libraries shown in Table 1:

```
rm –rf work
mkdir work
ncvlog -work work <design_name>.v <test_bench>.v
ncelab -lib_binding –access +rwc work.<test_bench>
ncsim -GUI work.<test_bench>
```

**Note**

If you are using NC-Sim version 5.4 or older, remove the `-lib_binding` option from the `ncelab` command.

## Post-Map and Place-and-Route Gate-Level Simulation

Use the following commands to perform post-map and place-and-route gate-level simulation:

```
rm –rf work
mkdir work
```

```
ncvlog -work work <design_name>.v <test_bench>.v
ncelab -lib_binding –access +rwc work.<test_bench>
ncsim -GUI work.<test_bench>
```

## Post-Map and Place-and-Route Gate-Level Simulation with Timing

Use the following procedure to perform post-map and place-and-route gate-level simulation with timing:

1. Add the `sdf_annotate` line at the top-level test bench:

   ```
   $sdf_annotate ("<design_name>.sdf", <design_name>,
   ,<sdf_log>, "MAXIMUM");
   ```

2. Use the following commands to perform the simulation:

   ```
   rm –rf work
   mkdir work
   ncvlog -work work <design_name>.v <test_bench>.v
   ncelab -lib_binding –access +rwc work.<test_bench>
   ncsim -GUI work.<test_bench>
   ```

# Performing Simulation with Cadence NC-VHDL

This section explains how to perform simulation with Cadence NC-VHDL.

## Setting Lattice Semiconductor Libraries

Before simulating Lattice Semiconductor FPGA designs in the Cadence NC-VHDL simulator, you must perform the following steps to set the Lattice Semiconductor simulation libraries.

### Creating Library Definition Files

Before using the Cadence NC-VHDL simulator to simulate your design project, you must first create two library definition files named hdl.var and cds.lib in your project folder. The hdl.var and cds.lib files define which libraries are accessible and where they are located. The hdl.var file contains statements that map logical library names to design library names, and the cds.lib file contains statements that map design library names to their physical directory paths.

For example, your local hdl.var file can include the following:

```
DEFINE EC ec_vhdl
DEFINE ECP ecp_vhdl
DEFINE XP xp_vhdl
DEFINE MachXO machxo_vhdl
DEFINE SC sc_vhdl
DEFINE ECP2 ecp2_vhdl
DEFINE XP2 xp2_vhdl
```

Your local cds.lib file can include the following:

```
DEFINE ec_vhdl <compile_dir>/ec_vhdl
DEFINE ecp_vhdl <compile_dir>/ecp_vhdl
DEFINE xp_vhdl <compile_dir>/xp_vhdl
DEFINE machxo_vhdl <compile_dir>/machxo_vhdl
DEFINE sc_vhdl <compile_dir>/sc_vhdl
DEFINE ecp2_vhdl <compile_dir>/ecp2_vhdl
DEFINE xp2_vhdl <compile_dir>/xp2_vhdl
```

**Note**

The <*compile*_dir> variable specifies the location of the compiled files.

Cadence provides a utility called NCLaunch to set up the necessary initialization files and to compile the VHDL source libraries. NCLaunch is available as part of the 2.1 and later releases. Otherwise, setting up the initialization files and compiling the VHDL source libraries is a manual process.

You can create the hdl.var and cds.lib files with any text editor. You must map the physical locations to the logical names before you proceed to the next step. On the UNIX and Linux platforms, you can create these names by using the `mkdir` command as follows:

```
mkdir -p <compile_dir>/ec_vhdl
mkdir -p <compile_dir>/ecp_vhdl
mkdir -p <compile_dir>/xp_vhdl
mkdir -p <compile_dir>/machxo_vhdl
mkdir -p <compile_dir>/sc_vhdl
mkdir -p <compile_dir>/ecp2_vhdl
mkdir -p <compile_dir>/xp2_vhdl
```

If you want the logical library names to be available for all designs, use `INCLUDE` or `SOFTINCLUDE` in the location of your master hdl.var and cds.lib files.

For example, you can add the following line to your master cds.lib file:

```
INCLUDE $path/cds.lib
```

**Note**

The *path* variable specifies the location of your own cds.lib file.

Or, you can directly include the library definition in the master hdl.var and cds.lib files.

Now the master hdl.var and cds.lib files include the Lattice Semiconductor library definitions. The next time that you want to simulate Lattice Semiconductor designs in the NC-VHDL simulator, you do not need to create you own hdl.var and cds.lib files again.

In addition to defining Lattice Semiconductor VHDL libraries, you must map the vital2000 logical library to the IEEE library location by adding the following line to your cds.lib file:

```
DEFINE vital2000 <NC_VHDL_Libraries_Folder>/IEEE
```

**Note**

The <*NC_VHDL_libraries_folder*> variable specifies the folder containing the NC-VHDL libraries.

## Parsing and Analyzing Lattice Semiconductor Simulation Libraries

After creating your own hdl.var and cds.lib files, you must parse and analyze the Lattice Semiconductor simulation libraries by using the NC-VHDL simulator. These libraries must be parsed and analyzed only once.

To parse and analyze Lattice Semiconductor VHDL simulation libraries, run the following commands in NC-VHDL simulator.

- LatticeEC VHDL:

  ```
  ncvhdl -messages -work ec_vhdl -smartorder
  $isplever_install_path/cae_library/simulation/vhdl/ec/src/
  *.vhd
  ```

- LatticeECP VHDL:

  ```
  ncvhdl -messages -work ecp_vhdl -smartorder
  $isplever_install_path/cae_library/simulation/vhdl/ecp/src/
  *.vhd
  ```

- LatticeECP2 VHDL:

  ```
  ncvhdl -messages -work ecp2_vhdl -smartorder
  $isplever_install_path/cae_library/simulation/vhdl/ecp2/src/
  *.vhd
  ```

- LatticeXP VHDL:

  ```
  ncvhdl -messages -work xp_vhdl -smartorder
  $isplever_install_path/cae_library/simulation/vhdl/xp/src/
  *.vhd
  ```

- LatticeXP2 VHDL:

  ```
  ncvhdl -messages -work xp2_vhdl -smartorder
  $isplever_install_path/cae_library/simulation/vhdl/xp2/src/
  *.vhd
  ```

- MachXO VHDL:

  ```
  ncvhdl -messages -work machxo_vhdl -smartorder
  ```

```
$isplever_install_path/cae_library/simulation/vhdl/machxo/
src/*.vhd
```

- LatticeSC VHDL:

```
ncvhdl -messages -work sc_vhdl -smartorder
$isplever_install_path/cae_library/simulation/vhdl/sc/src/
*.vhd
```

## Functional RTL Simulation

Use the following commands to perform a functional RTL simulation with one of the libraries shown in Table 2:

```
rm –rf work
mkdir work
ncvhdl -work work <design_name>.vhd <test_bench>.vhd
ncelab -lib_binding –access +rwc work.<test_bench>
ncsim –GUI work.<test_bench>
```

**Note**

If you are using NC-Sim version 5.4 or older, remove the `-lib_binding` option from the `ncelab` command.

## Post-Map and Place-and-Route Gate-Level Simulation

Use the following commands to perform post-map and place-and-route gate-level simulation:

```
rm –rf work
mkdir work
ncvhdl -work work <design_name>.vho <test_bench>.vhd
ncelab -lib_binding –access +rwc work.<test_bench>
ncsim –GUI work.<test_bench>
```

## Post-Map and Place-and-Route Gate-Level Simulation with Timing

Use the following procedure to perform post-map and place-and-route gate-level simulation with timing:

1. Compile the SDF file:

   ```
   ncsdfc <design_name>.sdf
   ```

   This command generates the compiled SDF file, <*design_name*>.sdf.X.

2. Create an <*sdf_cmd*> file with the following contents:

   ```
   COMPILED_SDF_FILE = “<design_name>.sdf.X”, SCOPE =
   :<instance_name>,
   LOG_FILE = “<sdf_log>”, MTM_CONTROL = "MAXIMUM";
   ```

3. Use the following commands to perform the simulation:

   ```
   rm –rf work
   mkdir work
   ncvhdl -work work <design_name>.vho <test_bench>.vhd
   ```

```
ncelab -lib_binding –access +rwc –sdf_cmd_file <sdf_cmd>
work.<test_bench>
ncsim –GUI work.<test_bench>
```

# Performing Simulation with Aldec Riviera Pro

This section explains how to perform simulation with the Aldec Riviera Pro simulator.

## Creating or Updating Lattice Semiconductor's FPGA Vendor Libraries

Create VHDL or Verilog libraries by following the procedures in this section.

### Creating VHDL Libraries

If you do not have Lattice Semiconductor's FPGA VHDL libraries as pre-compiled vendor libraries, the following steps are required to create them:

1. Create Lattice Semiconductor's VPGA VHDL libraries:

   ```
   # create new VHDL libraries
   alib sc
   alib ec
   alib xp
   alib ecp
   alib machxo
   alib ecp2
   alib xp2
   ```

   **Note**

   The VHDL library names are strict.

2. For each of the six VHDL libraries, compile the source files by using the `acom` command:

   ```
   acom -O2 -work <library_name> -f
   <vhdl_lib_compile_order_file>
   ```

   where:

   - *<vhdl_lib_compile_order_file>* is the file containing the proper compilation order for the *<library_name>* VHDL library. You must create this file.
   - The `-O2` optimization switch eliminates unnecessary warnings.

   If you want to place the debugging information in the library, use the `-dbg` switch. Debugging may slow down simulation.

   The compilation order for the SC library is as follows:

   ```
   <sc_vhdl_src_folder>/ORCACOMP.vhd
   <sc_vhdl_src_folder>/ORCA_SEQ.vhd
   <sc_vhdl_src_folder>/ORCA_IO.vhd
   <sc_vhdl_src_folder>/ORCA_MEM.vhd
   <sc_vhdl_src_folder>/ORCA_CMB.vhd
   ```

```
<sc_vhdl_src_folder>/ORCA_CNT.vhd
<sc_vhdl_src_folder>/ORCA_MIS.vhd
```

The compilation order for the EC library is as follows:

```
<ec_vhdl_src_folder>/ORCACOMP.vhd
<ec_vhdl_src_folder>/ORCA_SEQ.vhd
<ec_vhdl_src_folder>/ORCA_IO.vhd
<ec_vhdl_src_folder>/ORCA_CMB.vhd
<ec_vhdl_src_folder>/ORCA_CNT.vhd
<ec_vhdl_src_folder>/ORCA_MEM.vhd
<ec_vhdl_src_folder>/ORCA_MISC.vhd
<ec_vhdl_src_folder>/ORCA_LUT.vhd
```

The compilation order for the XP library is as follows:

```
<xp_vhdl_src_folder>/ORCACOMP.vhd
<xp_vhdl_src_folder>/ORCA_SEQ.vhd
<xp_vhdl_src_folder>/ORCA_IO.vhd
<xp_vhdl_src_folder>/ORCA_CMB.vhd
<xp_vhdl_src_folder>/ORCA_CNT.vhd
<xp_vhdl_src_folder>/ORCA_MEM.vhd
<xp_vhdl_src_folder>/ORCA_MISC.vhd
<xp_vhdl_src_folder>/ORCA_LUT.vhd
```

The compilation order for the ECP library is as follows:

```
<ecp_vhdl_src_folder>/ORCACOMP.vhd
<ecp_vhdl_src_folder>/ORCA_SEQ.vhd
<ecp_vhdl_src_folder>/ORCA_IO.vhd
<ecp_vhdl_src_folder>/ORCA_CMB.vhd
<ecp_vhdl_src_folder>/ORCA_CNT.vhd
<ecp_vhdl_src_folder>/ORCA_MEM.vhd
<ecp_vhdl_src_folder>/ORCA_MISC.vhd
<ecp_vhdl_src_folder>/ORCA_LUT.vhd
<ecp_vhdl_src_folder>/ORCA_MULT.vhd
```

The compilation order for the MachXO library is as follows:

```
<machxo_vhdl_src_folder>/MACHXOCOMP.vhd
<machxo_vhdl_src_folder>/MACHXO_SEQ.vhd
<machxo_vhdl_src_folder>/MACHXO_IO.vhd
<machxo_vhdl_src_folder>/MACHXO_CMB.vhd
<machxo_vhdl_src_folder>/MACHXO_CNT.vhd
<machxo_vhdl_src_folder>/MACHXO_LUT.vhd
<machxo_vhdl_src_folder>/MACHXO_MEM.vhd
<machxo_vhdl_src_folder>/MACHXO_MISC.vhd
```

The compilation order for the ECP2 library is as follows:

```
<ecp2_vhdl_src_folder>/ECP2COMP.vhd
<ecp2_vhdl_src_folder>/ECP2_SEQ.vhd
<ecp2_vhdl_src_folder>/ECP2_IO.vhd
<ecp2_vhdl_src_folder>/ECP2_CMB.vhd
<ecp2_vhdl_src_folder>/ECP2_CNT.vhd
<ecp2_vhdl_src_folder>/ECP2_MEM.vhd
<ecp2_vhdl_src_folder>/ECP2_MISC.vhd
<ecp2_vhdl_src_folder>/ECP2_LUT.vhd
<ecp2_vhdl_src_folder>/ECP2_MULT.vhd
```

The compilation order for the XP2 library is as follows:

```
<xp2_vhdl_src_folder>/XP2COMP.vhd
<xp2_vhdl_src_folder>/XP2_SEQ.vhd
```

```
<xp2_vhdl_src_folder>/XP2_IO.vhd
<xp2_vhdl_src_folder>/XP2_CMB.vhd
<xp2_vhdl_src_folder>/XP2_CNT.vhd
<xp2_vhdl_src_folder>/XP2_MEM.vhd
<xp2_vhdl_src_folder>/XP2_MISC.vhd
<xp2_vhdl_src_folder>/XP2_LUT.vhd
<xp2_vhdl_src_folder>/XP2_MULT.vhd
```

where each of the folders called <*lib_name_vhdl_src_folder*> is a folder containing VHDL source files for the appropriate Lattice Semiconductor library.

3. Set the status of the newly created library as read-only to prevent accidental overwriting:

```
setlibrarymode -ro <library_name>
```

## Creating Verilog Libraries

If you do not have Lattice Semiconductor's FPGA Verilog libraries as pre-compiled vendor libraries, the following steps are required to create them:

1. Create Lattice Semiconductor's VPGA Verilog libraries:

```
# create new Verilog libraries
alib ovi_sc
alib ovi_ec
alib ovi_xp
alib ovi_ecp
alib ovi_machxo
alib ovi_ecp2
alib ovi_xp2
```

**Note**

The Verilog library names are not strict, but they conform to Aldec's naming convention for Verilog vendor libraries.

2. For each of the six Verilog libraries, compile the source files by using the `alog` command:

```
alog -quiet -work <library_name> <verilog_lib_src_folder>/
*.v
```

where <*verilog_lib_src_folder*> is the folder containing the Verilog source files for the <*library_name*> library.

If you want to place the debugging information in the library, use the `-dbg` switch. Debugging may slow down simulation.

**Note**

Ignore the warning messages.

3. Set the status of the newly created library as read-only to prevent accidental overwriting:

```
setlibrarymode -ro <library_name>
```

### Updating Lattice Semiconductor's Vendor Libraries

If you obtained the updated source code of Lattice Semiconductor's FPGA libraries, you can update the VHDL libraries, Verilog libraries, or both by using the following procedure (for each one of the VHDL or Verilog libraries):

1. Using the `cd` command, navigate to the directory where your vendor library resides. The vendor libraries are usually located in the <*install_dir*>/vlib folder when pre-compiled by Aldec.
2. Set the library mode to read-write by using the `setlibrarymode` command:

   ```
   setlibrarymode -rw <library_name>
   ```

3. Compile the source code into the library. Use the `acom` command for VHDL source files and the `alog` command for Verilog source files.

   ```
   acom -O2 -work <library_name> -f
   <vhdl_lib_compile_order_file>
   ```

   or

   ```
   alog -quiet -work <library_name> <verilog_lib_src_folder>/
   *.v
   ```

4. Set the library to the read-only mode to prevent accidental overwriting:

   ```
   setlibrarymode -ro <library_name>
   ```

   **Note**

   You can enter the commands either in command-line mode or in the graphical console. Alternatively, you can write a macro that will prepare and compile the libraries, as follows:

   1. Include in the macro the appropriate commands for the libraries that you want to create or update.
   2. Run the runvsimsa script (UNIX, Linux) or the runvsimsa.bat batch file (Windows) command with the macro as a command-line argument.

## Performing Verilog Simulation

Use the procedures described in this section to perform a Verilog simulation.

### Functional RTL Simulation

Use the following commands to perform a functional RTL simulation with one of the libraries shown in Table 1:

```
alib work
alog -v2k -work work <design_name>.v <test_bench>.v
asim work.<test_bench>
```

**Note**

If you have a pre-existing work library, you can clear its contents by using the following command:

```
adel -lib work -all
```

### Post-Map and Place-and-Route Gate-Level Simulation

Use the following commands to perform post-map and place-and-route gate-level simulation:

```
alib work
alog -quiet -v2k -work work -l <library_name> <design_name>.vo
<test_bench>.v
asim -O5 +access +rw work.<test_bench>
```

where <*library_name*> is the FPGA Verilog library to be searched for the low-level units.

### Post-Map and Place-and-Route Gate-Level Simulation with Timing

Use the following procedure to perform post-map and place-and-route gate-level simulation with timing:

1. Add the `sdf_annotate` line at the top-level test bench:

   ```
   $sdf_annotate ("<design_name>.sdf", <instance_name>,
   ,<sdf_log>, "MAXIMUM");
   ```

2. Use the following commands to perform the simulation:

   ```
   alib work
   alog -quiet -v2k -work work -l <library_name>
   <design_name>.vo <test_bench>.v
   asim -O5 +access +rw work.<test_bench>
   ```

## Performing VHDL Simulation

Use the procedures described in this section to perform a VHDL simulation.

### Functional RTL Simulation

Use the following commands to perform a functional RTL simulation with one of the libraries shown in Table 2:

```
alib work
acom -work work <design_name>.vhd <test_bench>.vhd
asim -t ps work.<test_bench>
```

#### Note

If you have a pre-existing work library, you can clear its contents by using the following command:

```
adel -lib work -all
```

#### Note

Before you do any VHDL post-map simulation (with or without timing), you must first map the vital2000 logical library to the IEEE library location by using the following:

```
amap vital2000 <install_dir>/vlib/ieee/ieee.lib
```

where <*install_dir*> is the Aldec's Riviera installation folder.

### Post-Map and Place-and-Route Gate-Level Simulation

Use the following commands to perform post-map and place-and-route gate-level simulation:

```
alib work
acom -02 -work work <design_name>.vho <test_bench>.vhd
asim -t ps -noglitchmsg work.<test_bench>
```

### Post-Map and Place-and-Route Gate-Level Simulation with Timing

Use the following commands to perform post-map and place-and-route gate-level simulation with timing:

```
alib work
acom -02 -work work <design_name>.vho <test_bench>.vhd
asim -t ps -noglitchmsg -sdfmax
/<instance_name>="<design_name>.sdf" work.<test_bench>
```

#### Note

The “/” character is Aldec’s default hierarchy separator.

## Aldec Active-HDL

You can use the same procedure as in “Performing Simulation with Aldec Riviera Pro” on page 11 for creating or updating Lattice Semiconductor’s FPGA (vendor) libraries and simulating Verilog or VHDL designs in Active-HDL with a few modifications. Refer to the *Active-HDL On-line Documentation* for more details.

An alternative and easy way recommended for updating the Lattice Semiconductor FPGA libraries with Active-HDL is to obtain the pre-compiled Lattice Semiconductor FPGA libraries directly from the following directory:

`<ispLEVER_installation_folder>/active-hdl/lattice/Vlib`

You can either copy these libraries, along with the Library.cfg file, to your Active-HDL Vlib folder, overwriting any old libraries, or you can modify the Active-HDL Vlib/Library.cfg file to map to the Lattice Semiconductor FPGA libraries installed with ispLEVER (you can instead do the mapping with the `amap` command).

#### Note

Before you do any VHDL post-map simulation (with or without timing), first verify that the vital2000 logical library is mapped to the vital2000 library location (use `alist` to view the list of available libraries). This is the default mapping when you install Active-HDL, but if it is missing, for whatever reason, you can map the vital2000 library by using the following command:

```
amap vital2000 <install_dir>/vlib/vital2000/vital2000.lib
```

where `<install_dir>` is the Aldec’s Active-HDL installation folder.