Simulation User Guide (UG072)

All Achronix Devices

Achronix
Data Acceleration

Copyrights, Trademarks and Disclaimers

Notice of Disclaimer

Achronix Semiconductor Corporation

2903 Bunker Hill Lane
Santa Clara, CA 95054
USA

Website: www.achronix.com
E-mail : info@achronix.com

Table of Contents

Chapter 1 : Overview

Simulation Software Tool Flow

The Achronix tool suite includes synthesis and place-and-route software that maps RTL designs (VHDL or Verilog) into Achronix devices. In addition to synthesis and place-and-route functions, the Achronix software tools flow also supports simulation at several flow steps (RTL, synthesized netlist, and post place-and-route netlist), as shown in the figure below.

Functional simulation can be done at the following stages:

- Functional RTL level (referred to as RTL simulation)
- Gate-level, post-synthesis netlist (referred to as gate-level simulation)
- Gate-level, post-place-and-route netlist (referred to as post-route simulation)
- Full-chip bitstream simulation with gate-level netlist

The following diagram shows the stages of simulation in the context of the Achronix software tool flow.

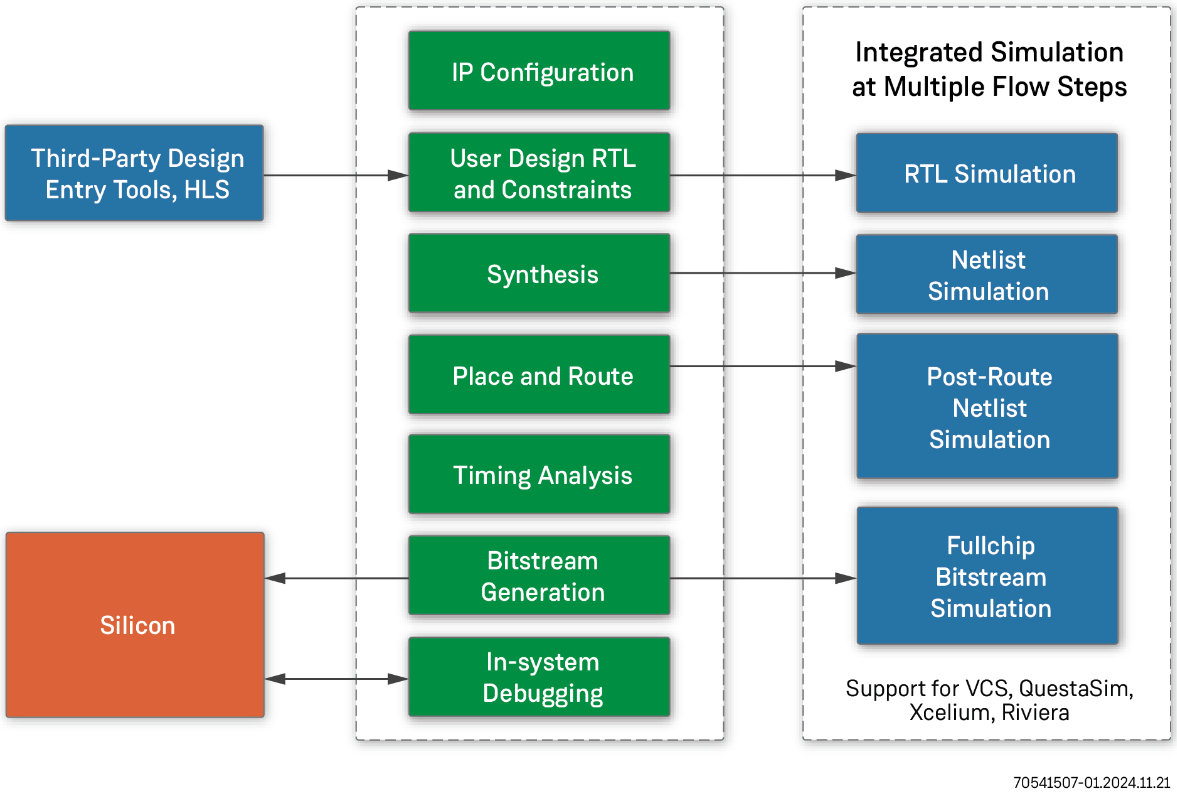

IP Configuration
Integrated Simulation at Multiple Flow Steps
Third-Party Design Entry Tools, HLS
User Design RTL and Constraints
RTL Simulation
Synthesis
Netlist Simulation
Place and Route
Post-Route Netlist Simulation
Timing Analysis
Silicon
Bitstream Generation
Fullchip Bitstream Simulation
In-system Debugging
Support for VCS, QuestaSim, Xcelium, Riviera
70541507-01.2024.11.21

Figure 1 • Stages of Simulation Flow

Simulation Libraries

This guide covers simulation for all Achronix devices. The text in this user guide contains references to `<DEVICE>`. The user should replace this with the target device name for your project, for example `AC7t1500`. The base ACE install package does not contain any simulation libraries by default. The simulation libraries are device-specific and are installed from within the device overlay installation packages.

Table 1 • Achronix Simulation Libraries

Directory			Description
<ace_install_dir>			Directory path to where ACE is installed.
	/libraries		Root directory for simulation and other libraries provided by Achronix. This is the top-level library include directory (+incdir+) for Achronix device libraries.
		/device_models	Contains the technology-specific top-level Achronix device library include files for simulation and synthesis.

The top-level device-specific library include file required for simulation is `<ace_install_dir>/libraries/device_models/<DEVICE>_simmodels.sv`. This top-level include file includes all of the library files for the given target device.

Including Memory Initialization Files

If a design uses memories and includes memory initialization files, the designer needs to consider carefully where to place the files when running simulation or synthesis. Achronix recommends using relative paths when referencing memory initialization files. Relative paths allows for changes in the location of the project without having to change the memory file reference in the RTL design files. However, when using relative paths, the designer must ensure that the path to a memory initialization file is relative to:

- The simulation directory when simulating.
- The ACE directory when generating a bitstream.

If these two relative paths are different, for example at different levels in the project hierarchy, the designer can use compiler directives to choose the correct path for the particular situation. An example is shown below.

Memory Initialization Path Example

```
`ifdef SIMULATION
    .mem_init_file (../../path_from_sim_dir/mem_filename)   // use this path when
simulating
`else
    .mem_init_file (../../path_from_ace_dir/mem_filename)   // use this path for
implementation
`endif
```

The above use of the `SIMULATION` compiler directive can also be a good way to design in special debug features that are only for simulation and not intended to be synthesized. Compiler directives can also be a good way to speed up certain sections of logic for simulation if desired.

Chapter 2 : Simulation from within ACE

As of ACE 10.0, Achronix provides built-in support for configuring the simulation environment via ACE project options and for running simulations via ACE flow steps. Using the built-in ACE simulation flow gives the end user a streamlined user experience, and a way to manage all aspects of the project (IP configuration, synthesis, simulation, place and route) from a single ACE project file.

Example Design

Although it is a simple example the quickstart tutorial design can be used to quickly demonstrate the ACE simulation flow. See ACE Quickstart Tutorial in the ***ACE Users Guide*** (UG070)[1] for details. The ACE simulation flow is capable of supporting nearly any simulation setup for a wide variety of designs and testbench structures across the supported list of simulator tools.

Configuring the Simulation Tool Environment

ACE simulation flow steps currently support installations of Siemens QuestaSim, Aldec Riviera, Cadence Xcelium, or Synopsys VCS to run the simulations. Therefore, the simulator tool must be installed in a directory accessible from the host on which ACE is running. Each simulator requires an environment variable to be set appropriately, as shown in the following table.

Table 2 • Environment Variables Needed for Simulation

Simulation Tool	Environment Variable	Linux Path	Windows Path
Siemens QuestaSim	ACX_QUESTASIM_TOOL_PATH	Set to the Linux executable QuestaSim launcher script path. A default launcher script example can be found in `<ace_install>/examples/simulation_scripts/questasim_launcher`.	Set to the directory path containing the `vlog.exe`, `vcom.exe`, and `vsim.exe` files (i.e., `D:\questa_base64_2023.3\win64`).

1 https://www.achronix.com/documentation/ace-user-guide-ug070

Simulation Tool	Environment Variable	Linux Path	Windows Path
Aldec Riviera	ACX_RIVIERA_TOOL_PATH	Set to the Linux executable Riviera launcher script path. A default launcher script example can be found in `<ace_install>/examples/simulation_scripts/riviera_launcher`. Optionally, for viewing waveforms, either: • add the path to `/bin` sub-directory for Riviera installation directory to `PATH` variable and `ALDEC_LICENSE_FILE` in the shell environment (and not the launcher script), or; • In local launcher script, use `<riviera_install_path>/bin/vsim` instead of `<riviera_install_path>/runvsimsa`.	Set to the file path of the Riviera `runvsimsa.bat` file (i.e., `D:\Aldec\Riviera-PRO-2020.10-x64\runvsimsa.bat`. Optionally, for viewing waveforms, either: • Add the path to `/bin` sub-directory for Riviera installation directory to `PATH` variable in the local environment, or; • Set to the file path of the `<riviera_install_path>/bin/vsim` instead of `<riviera_install_path>/runvsimsa.bat`.
Synopsys VCS	ACX_VCS_TOOL_PATH	Set to the Linux executable VCS launcher script path. A default launcher script example can be found in `<ace_install>/examples/simulation_scripts/vcs_launcher`.	(Not Supported)
Cadence Xcelium	ACX_XCELIUM_TOOL_PATH	Set to the Linux executable Xcelium launcher script path. A default launcher script example can be found in `<ace_install>/examples/simulation_scripts/xcelium_launcher`	(Not Supported)

Configuring the Project Source Files

To run the simulation from ACE, end user design source RTL files and simulation testbench RTL files must be added to the ACE project.

Project Source File Ordering

The order in which source RTL files are added to an ACE project is the order in which the files will be compiled by the simulator. Therefore, if the chosen simulator requires that modules are defined before they are instantiated, add the lowest-level module definition files first, then the top-level module file last. Once files are added to the ACE project, they may be re-order via drag-and-drop in the Projects View in the ACE GUI, or by calling the `move_project_source_file` TCL command.

First, create or restore an ACE project file in the current ACE session. See the ACE Quickstart Tutorial for instructions on how to create the ACE project for the Quickstart example.

In the Projects view, click the project to select it and activate its implementation. Follow these steps to add the design source files for synthesis, simulation, place and route:

1. Click the ( ) **Add Source Files** toolbar button and select **Add RTL Files**.

2. In the Add RTL Files dialog, browse to the directory where the end-user design RTL is located and select all of the files.
3. Click the **Open** button to add the RTL files to the project.

> **Note**
>
> When adding RTL files to an ACE project, adding the top-level ACE library simulation include file for the target device (`<ace_install>/libraries/device_models/<device>_simmodels.sv` (page 4)) is not needed. ACE will automatically locate and add this file to the simulation.

4. Click the ( ) **Add Source Files** toolbar button and select **Add Simulation Testbench Files**.
5. In the "Add Simulation Testbench Files" dialog, browse to the directory where the simulation testbench RTL files are located and select all of the files. For waveform-specific files (`*.do` or `*.tcl`), select simulator-specific file only.

> **Notes**
>
> - `*.f` files may also be added as simulation testbench source files in ACE. When using the default simulation flow, ACE will automatically add these files to the simulator compile commands with the "`-f`" option. ACE will pre-parse the file to determine of the file should be included in VHDL compile, Verilog compile, or both.
> - Additional files maybe added, such as memory initialization files, test vector files, or any other data files that the user design RTL or testbench RTL need to load during the simulation compile or run. When using the default simulation flow, ACE will automatically copy these files into the simulation run directory (`<ace_project_output_dir>/<impl>/sim/<sim_step>/<tool>/`) so they can be loaded using the filename or relative path of "`./<filename>`".

6. Click the **Open** button to add the simulation testbench files to the project.
7. Click the ( ) **Save Project** toolbar button to save the changes back to the `*.acxprj` ACE project file.

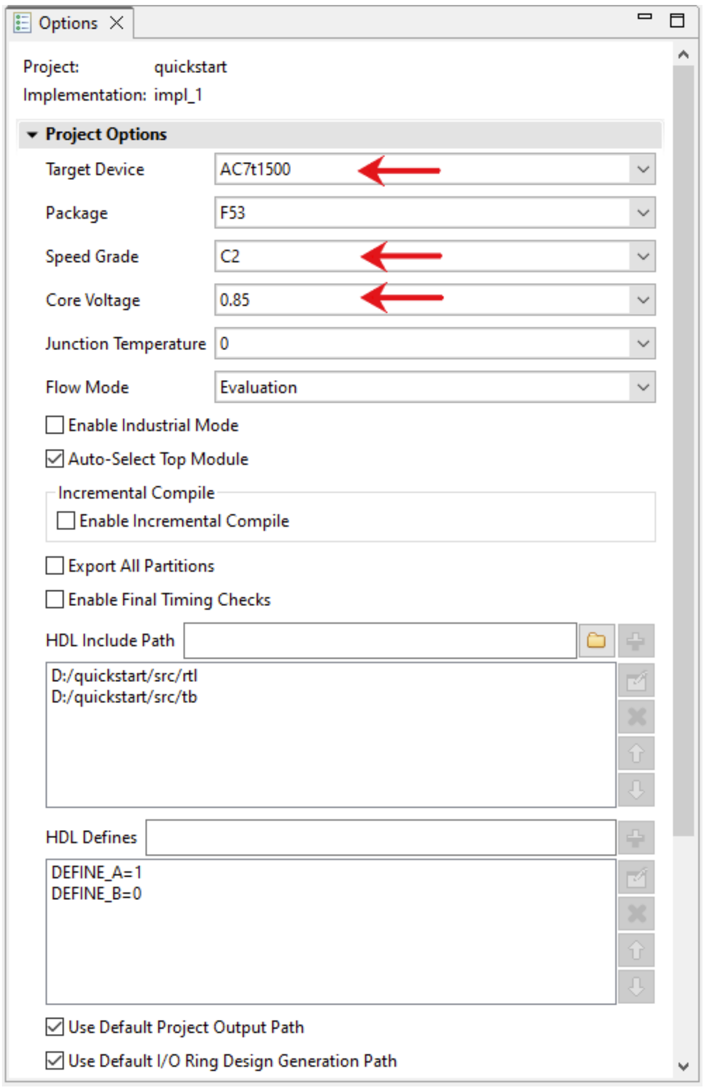
The source files are now configured and saved to the ACE project. All of the source files are now visible in the Projects View under the **Source** tree for the project.

Configuring the Simulation Options

The simulation environment and simulator command-line arguments are configured using the Simulation ACE project options. These options can be configured in the Options View in the ACE GUI, or via the `set_project_option` TCL command.

First, configure the top-level project settings. In the Options View, follow these steps to configure the project options:

1. Expand the Project Settings section and select the Target Device, Speed Grade, and Core Voltage for the project.


Options
Project Options
Enable Industrial Mode
Auto-Select Top Module
Incremental Compile
Enable Incremental Compile
Export All Partitions
Enable Final Timing Checks
Use Default Project Output Path
Use Default I/O Ring Design Generation Path

Project: quickstart
Implementation: impl_1

Target Device	AC7t1500
Package	F53
Speed Grade	C2
Core Voltage	0.85
Junction Temperature	0
Flow Mode	Evaluation

HDL Include Path:
D:/quickstart/src/rtl
D:/quickstart/src/tb

HDL Defines:
DEFINE_A=1
DEFINE_B=0

Figure 2 • ACE Options View

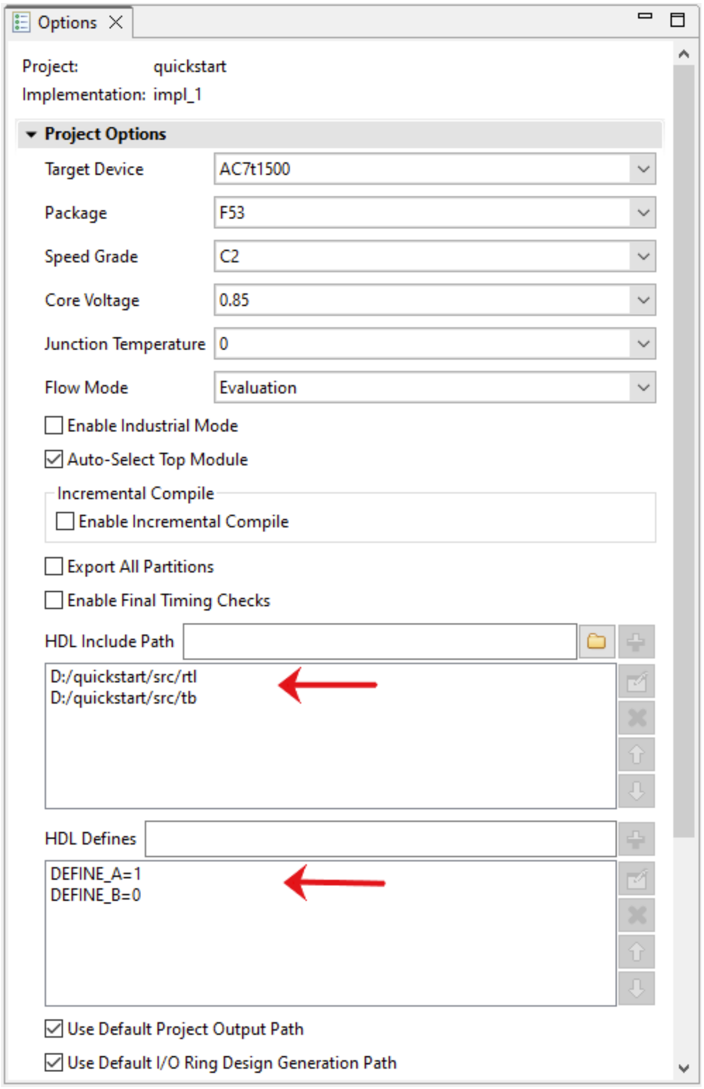
2. In the Project Settings section, scroll down and enter the a semicolon-separated list of directory paths for the HDL Include Path, for example: `<`*`test_dir`*`>/src/rtl;<`*`test_dir`*`>/src/tb`

> **Notes**
> - The HDL Include Path applies to both synthesis and simulation.
> - ACE will automatically add the `<ace_install>/libraries` directory to the HDL Include Path; they do need to be entered in the HDL Include Path project option.

3. If the end-user design or simulation testbench requires any HDL defines, enter them as a space-separated list in the HDL Defines project option, for example: `SIMULATION_DEFINE_A=1 SIMULATION_DEFINE_B=0`

Note

The HDL Defines applies to both synthesis and simulation.



Project: quickstart
Implementation: impl_1

Target Device: AC7t1500
Package: F53
Speed Grade: C2
Core Voltage: 0.85
Junction Temperature: 0
Flow Mode: Evaluation

☐ Enable Industrial Mode
☑ Auto-Select Top Module

Incremental Compile
☐ Enable Incremental Compile

☐ Export All Partitions
☐ Enable Final Timing Checks

HDL Include Path:

```
D:/quickstart/src/rtl
D:/quickstart/src/tb
```

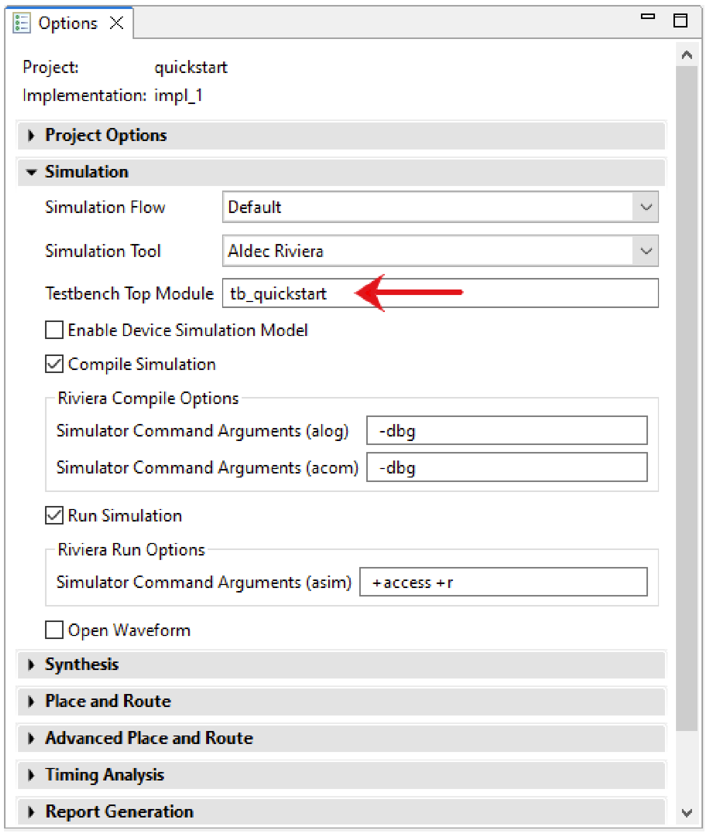
HDL Defines:

```
DEFINE_A=1
DEFINE_B=0
```

☑ Use Default Project Output Path
☑ Use Default I/O Ring Design Generation Path

Figure 3 • Setting HDL Include Path and HDL Defines

4. Scroll down and expand the Simulation section of options and enter the testbench top module name in the Testbench Top Module field.



Project: quickstart
Implementation: impl_1

Simulation

Simulation Flow: Default
Simulation Tool: Aldec Riviera
Testbench Top Module: tb_quickstart
☐ Enable Device Simulation Model
☑ Compile Simulation

Riviera Compile Options
Simulator Command Arguments (alog): -dbg
Simulator Command Arguments (acom): -dbg

☑ Run Simulation

Riviera Run Options
Simulator Command Arguments (asim): +access +r

☐ Open Waveform

Figure 4 • Setting the Testbench Top Module

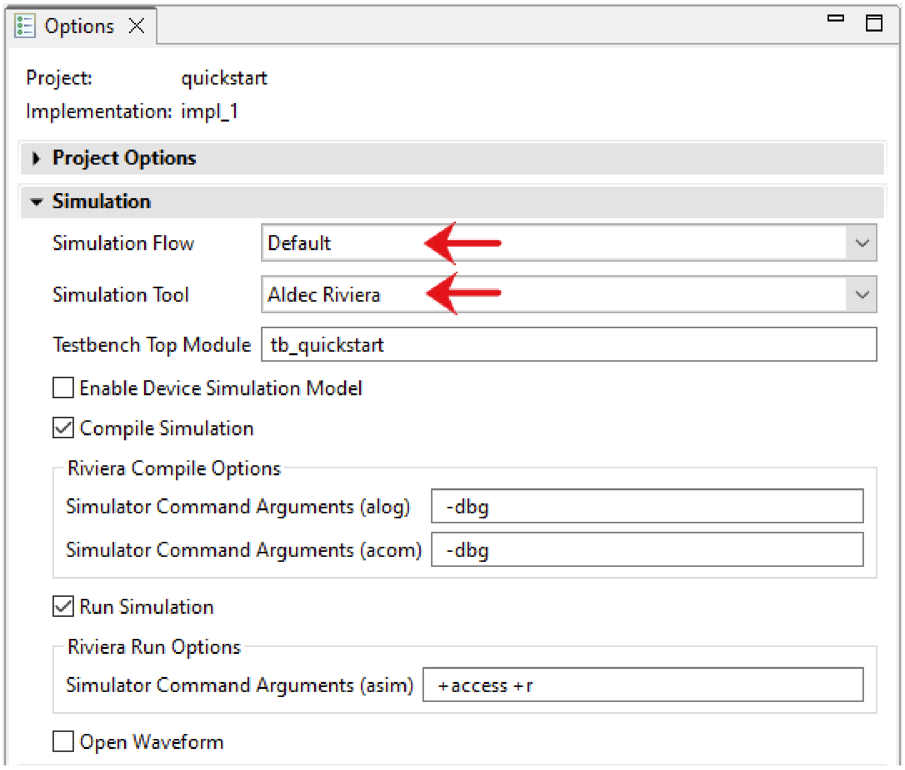
Configuring Simulation

There are two types of simulation flows which can be run via the ACE flow steps:

- Default simulation flow – this flow is the simplest flow for end users. It uses the simulation scripts built into ACE to configure and run the simulation in the background.
- Custom simulation flow – this flow allows the end user to create and use their own custom simulator scripts instead of using the built-in scripts, allowing a custom simulation environment to be connected in to the ACE flow steps.

Default Simulation Flow

To use the Default simulation flow, first set the Simulation Flow project option to Default. Then select the simulator desired.



Project: quickstart
Implementation: impl_1

Simulation Flow: Default
Simulation Tool: Aldec Riviera
Testbench Top Module: tb_quickstart
☐ Enable Device Simulation Model
☑ Compile Simulation
Riviera Compile Options
Simulator Command Arguments (alog): -dbg
Simulator Command Arguments (acom): -dbg
☑ Run Simulation
Riviera Run Options
Simulator Command Arguments (asim): +access +r
☐ Open Waveform

Figure 5 • Setting the Simulator Within ACE

ACE currently supports the following simulators for use with the default simulation flow:

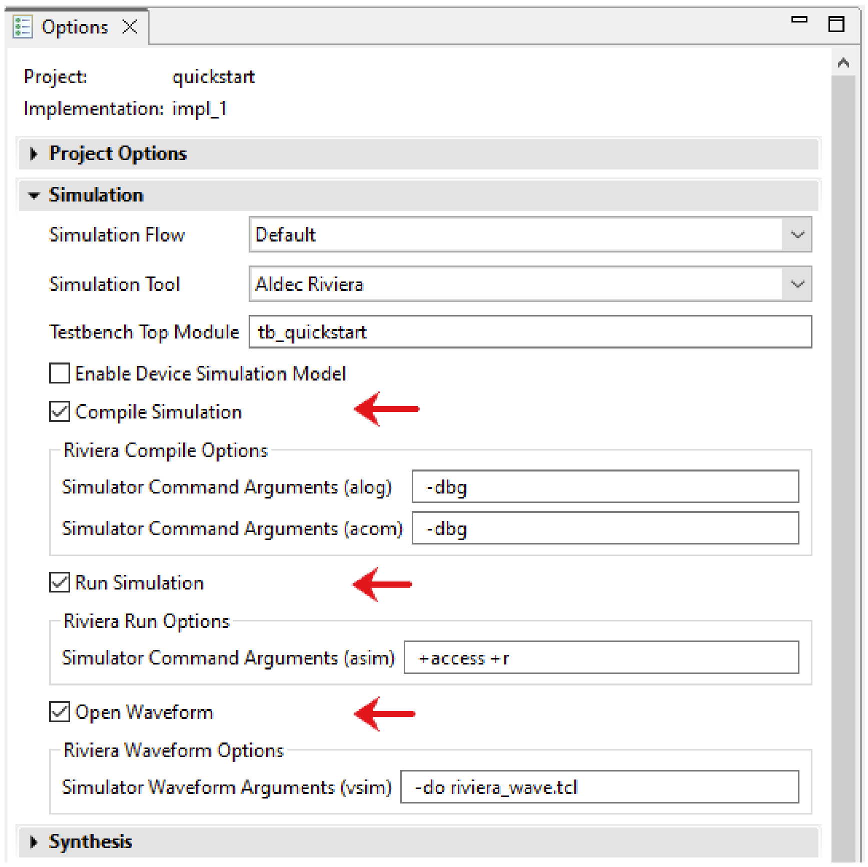
- Siemens QuestaSim
- Aldec Riviera
- Synopsys VCS
- Cadence Xcelium

Each simulator tool has its own simulator-specific set of simulation compile options and simulation run options, which allow command-line arguments to passed to the underlying simulator.

All simulators (except Xcelium) support separate compile and run steps. By default, the ACE project options to **Compile Simulation** and **Run Simulation** are enabled. These settings ensure that the simulation flow step first compiles and changes to the source HDL and testbench, and then runs the simulation. Optionally, the **Run Simulation** option can be disabled to perform a compile-only, or disable the **Compile Simulation** to perform a re-run of the previous compile.

The Cadence Xcelium simulation flow step supports running the **xrun** command for both compile and simulation run.

Optionally, the Open Waveform option can be enabled to open simulator-specific waveform viewer. When enabled, ACE invokes the simulator to load the simulation database that was generated as part of the simulation run. Waveforms are displayed as per the waveform operations specified in waveform (`*.do` or `*.tcl`) file. The simulator arguments for opening the waveform can be set in Simulator Waveform Arguments option.


Options

Project: quickstart
Implementation: impl_1

Project Options

Simulation

Simulation Flow	Default
Simulation Tool	Aldec Riviera
Testbench Top Module	tb_quickstart

☐ Enable Device Simulation Model

☑ Compile Simulation

Riviera Compile Options

Simulator Command Arguments (alog)	-dbg
Simulator Command Arguments (acom)	-dbg

☑ Run Simulation

Riviera Run Options

Simulator Command Arguments (asim)	+access +r

☑ Open Waveform

Riviera Waveform Options

Simulator Waveform Arguments (vsim)	-do riviera_wave.tcl

Synthesis

Figure 6 • Setting Simulation and Compile Options

If targeting a Speedster device which has a device simulation model, a simulation option to **Enable Device Simulation Model** is activated. If this option is enabled, the simulation flow scripts will automatically include and compile the correct device simulation model files for the target device. ACE will also automatically set the `$ACX_USE_DSM` environment variable to 1 before calling the simulator tools.

For all devices, ACE automatically sets the following environment variables prior to calling the simulator, so there is no need to set them manually:

- `$ACE_INSTALL_DIR` - the path to the ACE installation
- `$ACX_DEVICE_INSTALL_DIR` - the path to the device installation area inside the ACE install
- `$DEVICE` - the name of the target device for the project

ACE also automatically adds a define to the simulation to specify the selected simulator:

Table 3 • ACE Simulator Defines

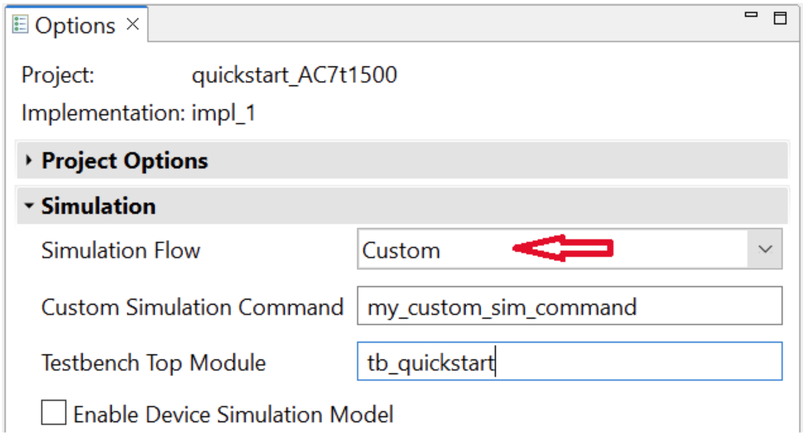
Simulator Tool	Define
Aldec Riviera	+define+RIVIERA
Siemens QuestaSim	+define+QUESTASIM
Synopsys VCS	+define+VCS
Cadence Xcelium	+define+XCELIUM

The project is now ready to run through the flow.

Custom Simulation Flow

Connecting a custom simulation flow into the ACE flow steps requires some ACE Simulation project options to be configured.

1. Set the Simulation Flow option to **Custom**:



Project: quickstart_AC7t1500
Implementation: impl_1

Simulation Flow: Custom
Custom Simulation Command: my_custom_sim_command
Testbench Top Module: tb_quickstart

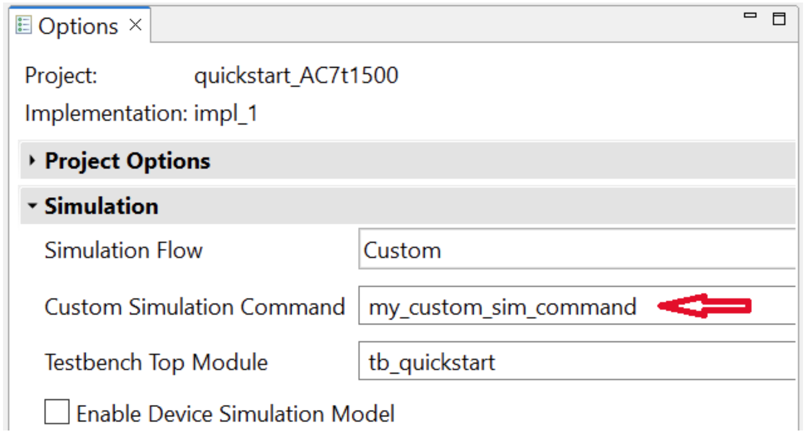
Figure 7 • Selecting the Simulation Flow

2. Next, configure the Custom Simulation Command. The custom simulation command is the TCL command that the ACE flow step will call to compile and run the simulation. Since there are multiple ACE simulation flow steps (as described below), ACE must pass in the simulation flow step ID to the custom simulation command. Therefore, the custom simulation command must be able to take a "`-sim_step <step>`" command-line option, where the values of `<step>` are: `rtl`, `gate`, `routed`, and `final`.

There are two options for configuring the Custom Simulation Command:

- Use a TCL `exec` command to call an external executable script or program.

- Use an `ACE_INIT_SCRIPT` to create a custom TCL proc to call the simulator. See Running ACE in the *ACE Users Guide* (UG070)[2] for more details.



Project: quickstart_AC7t1500
Implementation: impl_1

Simulation Flow: Custom
Custom Simulation Command: my_custom_sim_command
Testbench Top Module: tb_quickstart

Figure 8 • Setting the Customer Simulation Command

If targeting a Speedster device which has a device simulation model, a simulation option to **Enable Device Simulation Model** is activated. If this option is enabled, the generated `<ace_project_output_dir>/<impl>/sim/<simstep>/custom/verilog_filelist.f` will automatically include the correct device simulation model top-level include files for the target device. However, the custom simulation flow needs to account for the `<ace_install>/system/data/<device>/sim/<device>_dsm_incdirs.f` file. ACE also automatically sets the `$ACX_USE_DSM` environment variable to 1 before calling the simulator.

Configuring the Simulation Waveform

To configure simulator-specific waveform viewer as part of "Default Simulation Flow", refer to "Configuring the Project Source Files" and "Configuring the Simulation Options" sections under the chapter, Opening Simulator Waveform Viewer in ACE in the *ACE Users Guide* (UG070)[3] for configuring the Project Source Files and Simulation Options.

2 https://www.achronix.com/documentation/ace-user-guide-ug070
3 https://www.achronix.com/documentation/ace-user-guide-ug070

Table 4 • Simulator-specific Details for Configuring Simulation Waveform

Simulator Tool	Verilog tasks	Simulator GUI Command	Comments
Aldec Riviera	`$asdbDump;`	`vsim -do <input_file>`	• <input_file> can be *.do or *.tcl file • $asdbDump generates dataset.asdb file by default. Refer to Riviera Pro Documentation to provide custom ASDB name. • To dump signal data, ensure +access +r is passed as an argument in **Riviera Run Options** to provide read access.
Siemens QuestaSim	`$wlfdumpvars(0, <tb_top_name>);`	`vsim -gui -voptargs=+acc -view <wlf_filename>.wlf -do <input_file>`	• <input_file> can be *.do or *.tcl file • <wlf_filename>.wlf is passed as -wlf argument in **QuestaSim Run Options**. • To dump signal data, ensure -voptargs=+acc is also passed as an argument in **QuestaSim Run Options** to provide read access.
Synopsys VCS	`$vcdplusfile("sim_output_pluson.vpd"); $vcdpluson(0, <tb_top_name>);`	`dve -full64 -vpd sim_output_pluson.vpd`	• The `vpd` filename is currently fixed to `sim_output_pluson.vpd`
Cadence Xcelium	`$shm_open("<database_name>.shm"); $shm_probe("S");`	`xrun -gui -R -input <input_file>.tcl`	• This generates a `<database_name>.shm` directory which contains the `.trn` file which must be loaded in SimVision GUI during waveform viewing. • Use `-r <snapshot_name>` instead of `-R` to provide a custom snapshot name when running simulation using `xrun`.

Simulator Tool	Verilog tasks	Simulator GUI Command	Comments

Table Notes

- The loading of waveform database is simulator-specific. In QuestaSim (`*.wlf`) and VCS (`*.vpd`), the database file can be passed as part of **Simulator Waveform Arguments** option. In Riviera (`*.asdb`) and Xcelium (`*.trn`), the database file is loaded using a TCL command and is specified in the waveform (`*.do` or `*.tcl`) file. Since the waveform (`*.do` or `*.tcl`) file is added to the ACE project as part of simulation testbench files, it gets copied in the simulation output directory when Simulation flow step is run. Hence, the waveform (`*.do` or `*.tcl`) file is relative to simulation output directory when it is passed as input to **Simulator Waveform Arguments** option.
- If ACX_RIVIERA_TOOL_PATH is set to `vsim` under the `bin/` subdirectory of the Riviera-PRO installation directory, ensure that under Riviera Run Options, use "-c +access+r" to run `vsim` in batch mode.

Running the Simulation Flow Steps

ACE provides the following optional simulation flow steps:

Table 5 • Optional Simulation Flow Steps

Name	ID
RTL Simulation	
– Run RTL Simulation	`run_simulation_rtl`
Synthesis	
– Run Gate-level Netlist Simulation	`run_simulation_gate`
Place and Route	
– Generate Post-Route Simulation Netlist	`write_netlist_routed`
– Run Post-Route Netlist Simulation	`run_simulation_routed`
Design Completion	
– Generate Final Simulation Netlist	`write_netlist_final`
– Run Final Netlist Simulation	`run_simulation_final`

Name	ID

Table Notes

- All flow step IDs can be executed at the ACE GUI Tcl console (see Tcl Console View in the *ACE Users Guide* (UG070[4]) or as part of the user Tcl script that can be invoked when running ACE in batch mode. The following Tcl command allows executing the various flow steps IDs listed:

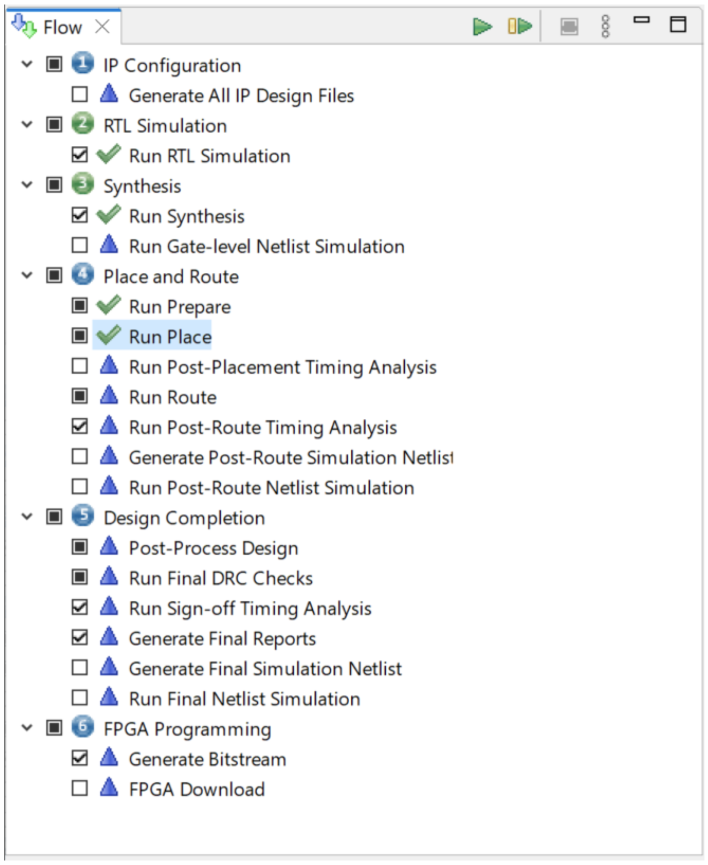
```
                    run [-step <string>] [-stop_at_step <string>] [-
resume] [-ic <string>]
```

- Because advanced users are allowed to create their own flow steps (create_flow_step), this list may be a subset of the flow steps available to users. To see a complete list of current flow step IDs, use the Tcl command get_flow_steps.

To run a given simulation flow step:

1. Enable the flow step by checking the checkbox next to the given flow step in the Flow View in the ACE GUI, or call the `enable_flow_step <step_id>` TCL command.
2. Double-click the flow step in the Flow View in the ACE GUI to run it, or call the `run -step <step_id>` TCL command.

4 https://www.achronix.com/documentation/ace-user-guide-ug070



- IP Configuration
  - ☐ Generate All IP Design Files
- RTL Simulation
  - ☑ Run RTL Simulation
- Synthesis
  - ☑ Run Synthesis
  - ☐ Run Gate-level Netlist Simulation
- Place and Route
  - ■ Run Prepare
  - ■ Run Place
  - ☐ Run Post-Placement Timing Analysis
  - ■ Run Route
  - ☑ Run Post-Route Timing Analysis
  - ☐ Generate Post-Route Simulation Netlist
  - ☐ Run Post-Route Netlist Simulation
- Design Completion
  - ■ Post-Process Design
  - ■ Run Final DRC Checks
  - ☑ Run Sign-off Timing Analysis
  - ☑ Generate Final Reports
  - ☐ Generate Final Simulation Netlist
  - ☐ Run Final Netlist Simulation
- FPGA Programming
  - ☑ Generate Bitstream
  - ☐ FPGA Download

Figure 9 • Selecting Flow Steps to be Run

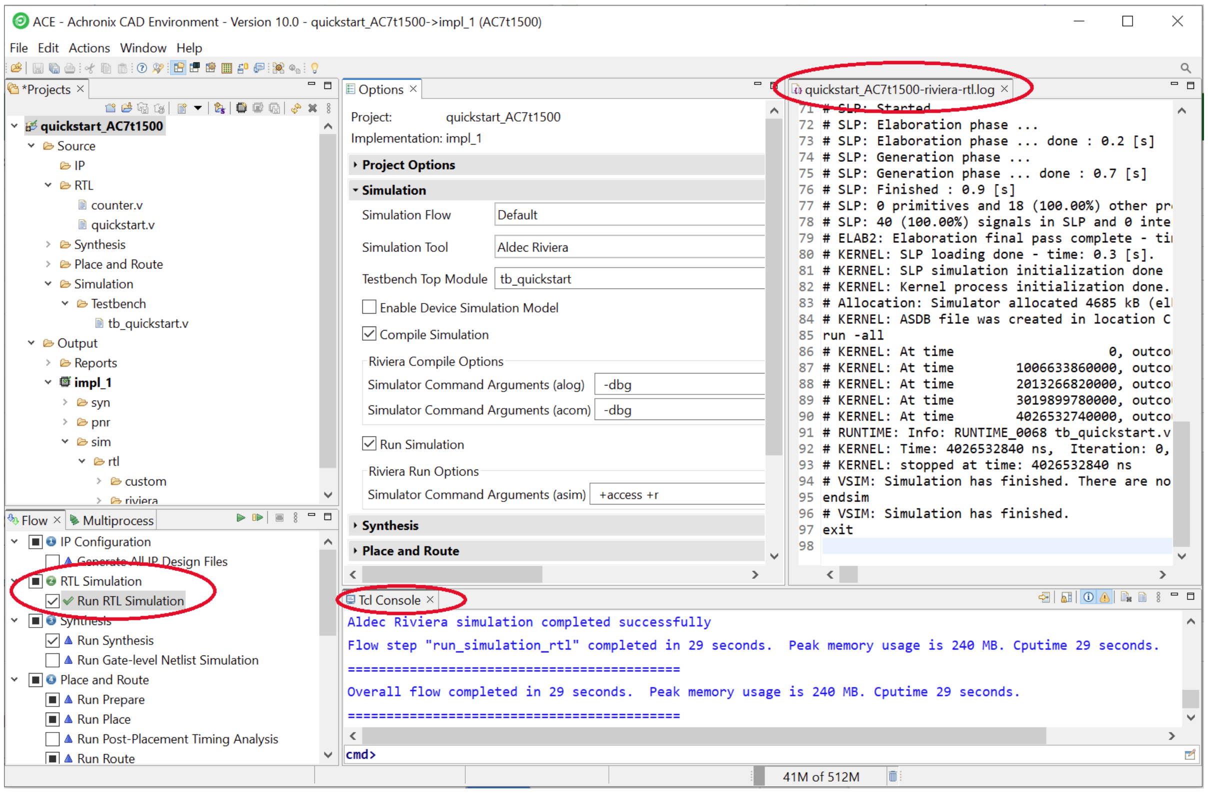
Prior to running the simulation, ACE will change working directories ("cd") into the simulation output directory: `<ace_project_output_dir>/<impl>/sim/<sim_step>/<tool>/`. ACE will then change back to the original ACE working directory after the simulation run completes. Make sure any relative file paths in the simulation testbench and user design RTL are relative to this directory.

Notes

- `*.f` files may also be added as simulation testbench source files in ACE. When using the default simulation flow, ACE will automatically add these files to the simulator compile commands with the "`-f`" option. ACE will pre-parse the file to determine of the file should be included in VHDL compile, Verilog compile, or both.
- Additional files maybe added, such as memory initialization files, test vector files, or any other data files that the user design RTL or testbench RTL need to load during the simulation compile or run. When using the default simulation flow, ACE will automatically copy these files into the simulation run directory (`<ace_project_output_dir>/<impl>/sim/<sim_step>/<tool>/`) so they can be loaded using the filename or relative path of "`./<filename>`".

Viewing the Simulation Outputs

Once a simulation flow step has been run, the simulator log file will automatically be displayed in the ACE GUI Editor Area, and the ACE TCL Console will display any messages about the simulation run.



Project: quickstart_AC7t1500
Implementation: impl_1

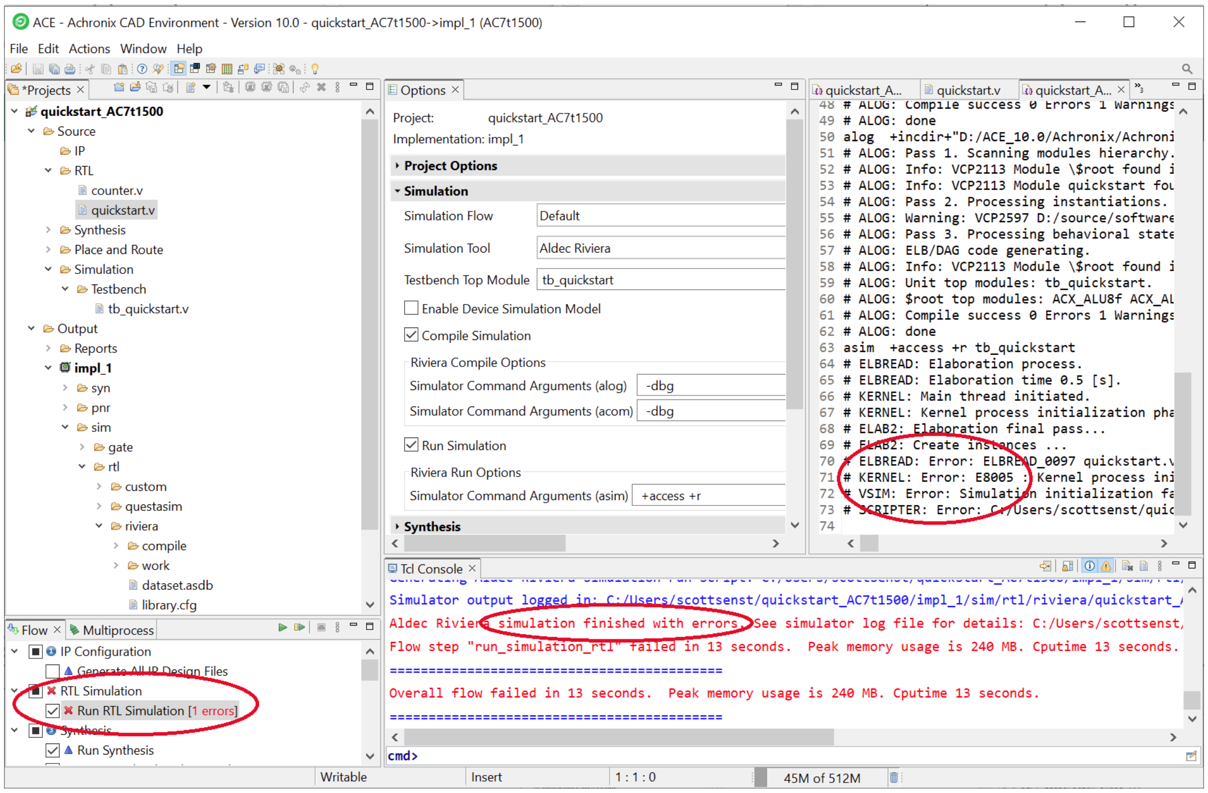
- Simulation Flow: Default
- Simulation Tool: Aldec Riviera
- Testbench Top Module: tb_quickstart
- ☐ Enable Device Simulation Model
- ☑ Compile Simulation
- Riviera Compile Options
  - Simulator Command Arguments (alog): -dbg
  - Simulator Command Arguments (acom): -dbg
- ☑ Run Simulation
- Riviera Run Options
  - Simulator Command Arguments (asim): +access +r

```
# SLP: Started
# SLP: Elaboration phase ...
# SLP: Elaboration phase ... done : 0.2 [s]
# SLP: Generation phase ...
# SLP: Generation phase ... done : 0.7 [s]
# SLP: Finished : 0.9 [s]
# SLP: 0 primitives and 18 (100.00%) other pr
# SLP: 40 (100.00%) signals in SLP and 0 inte
# ELAB2: Elaboration final pass complete - ti
# KERNEL: SLP loading done - time: 0.3 [s].
# KERNEL: SLP simulation initialization done
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 4685 kB (el
# KERNEL: ASDB file was created in location C
run -all
# KERNEL: At time                   0, outco
# KERNEL: At time        1006633860000, outco
# KERNEL: At time        2013266820000, outco
# KERNEL: At time        3019899780000, outco
# KERNEL: At time        4026532740000, outco
# RUNTIME: Info: RUNTIME_0068 tb_quickstart.v
# KERNEL: Time: 4026532840 ns,  Iteration: 0,
# KERNEL: stopped at time: 4026532840 ns
# VSIM: Simulation has finished. There are no
endsim
# VSIM: Simulation has finished.
exit
```

```
Aldec Riviera simulation completed successfully
Flow step "run_simulation_rtl" completed in 29 seconds.  Peak memory usage is 240 MB. Cputime 29 seconds.
===========================================
Overall flow completed in 29 seconds.  Peak memory usage is 240 MB. Cputime 29 seconds.
===========================================
cmd>
```

Figure 10 • Log File and Message Display

If there are any errors in the compile of the simulation, or if the simulator tool returns a non-zero return code, the ACE flow step will error out and stop the flow and ACE will open the simulation log file to the line that shows the error message.



Project: quickstart_AC7t1500
Implementation: impl_1

Project Options

Simulation

Simulation Flow	Default
Simulation Tool	Aldec Riviera
Testbench Top Module	tb_quickstart

☐ Enable Device Simulation Model
☑ Compile Simulation

Riviera Compile Options

Simulator Command Arguments (alog)	-dbg
Simulator Command Arguments (acom)	-dbg

☑ Run Simulation

Riviera Run Options

Simulator Command Arguments (asim)	+access +r

Synthesis

```
# ALOG: Compile success 0 Errors 1 Warnings
# ALOG: done
alog  +incdir+"D:/ACE_10.0/Achronix/Achroni
# ALOG: Pass 1. Scanning modules hierarchy.
# ALOG: Info: VCP2113 Module \$root found i
# ALOG: Info: VCP2113 Module quickstart fou
# ALOG: Pass 2. Processing instantiations.
# ALOG: Warning: VCP2597 D:/source/software
# ALOG: Pass 3. Processing behavioral state
# ALOG: ELB/DAG code generating.
# ALOG: Info: VCP2113 Module \$root found i
# ALOG: Unit top modules: tb_quickstart.
# ALOG: $root top modules: ACX_ALU8f ACX_AL
# ALOG: Compile success 0 Errors 1 Warnings
# ALOG: done
asim  +access +r tb_quickstart
# ELBREAD: Elaboration process.
# ELBREAD: Elaboration time 0.5 [s].
# KERNEL: Main thread initiated.
# KERNEL: Kernel process initialization pha
# ELAB2: Elaboration final pass...
# ELAB2: Create instances ...
# ELBREAD: Error: ELBREAD_0097 quickstart.v
# KERNEL: Error: E8005 : Kernel process ini
# VSIM: Error: Simulation initialization fa
# SCRIPTER: Error: C:/Users/scottsenst/quic
```

```
Simulator output logged in: C:/Users/scottsenst/quickstart_AC7t1500/impl_1/sim/rtl/riviera/quickstart_
Aldec Riviera simulation finished with errors. See simulator log file for details: C:/Users/scottsenst,
Flow step "run_simulation_rtl" failed in 13 seconds.  Peak memory usage is 240 MB. Cputime 13 seconds.
============================================
Overall flow failed in 13 seconds.  Peak memory usage is 240 MB. Cputime 13 seconds.
============================================
cmd>
```

Figure 11 • Simulator Error Message Display

However, depending on the design of the simulation testbench and the simulator, the simulation run could return a zero return code (indicating the simulation passed) while the testbench printed display statements to the simulation log file to indicate there was actually a failure. The ACE built-in simulation script automatically parses the simulation log file to look for error messages using the following regular expressions:

Table 6 • Log File Search RegEx

Log Message Regex	Effect
{^Lint-} or {^Warning-}	Remaining message is ignored until the next blank line
{(ERROR[\s:].*)}	ACE flow step reports an error
{(Error:.*)}	ACE flow step reports an error, unless the line also matches {vsim-8604} or {\$setup}
{Test finished with .* errors}	ACE flow step reports an error

Log Message Regex	Effect
Case insensitive {^error}	ACE flow step reports an error, unless the line also matches {Errors: 0}
{Fatal error}	ACE flow step reports an error
{vsim-PLI-3406}	
Case insensitive {simulation failed}	
Case insensitive {test failed}	

If any messages in the simulation log file match these expressions, ACE will treat it as a simulation failure, even though the simulator returned a zero (passing) return code. Please use the regular expressions above as a guideline for developing testbench error messages to ensure ACE will catch them.

Once simulation has run, any simulation output will be captured in `<ace_project_output_dir>/<impl>/sim/<sim_step>/<tool>/`, where:

- `<sim_step>` is one of `rtl, gate, routed,` or `final`
- `<tool>` is one of `riviera, vcs, questasim, xcelium`.

Simulation output files can be browsed and opened from within the ACE GUI Projects View.

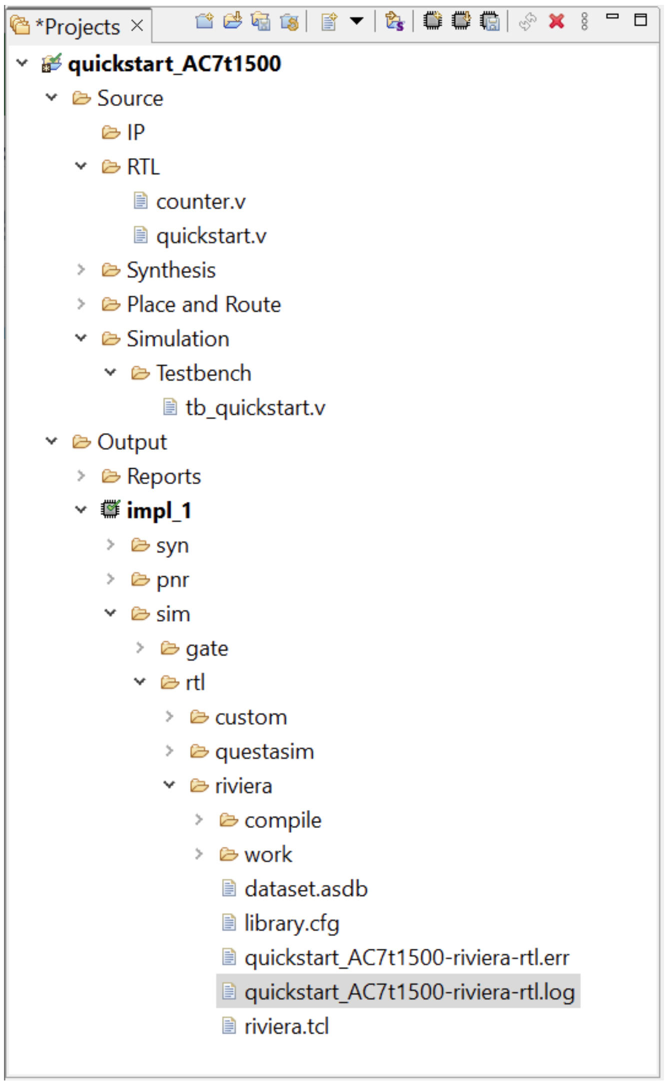


Figure 12 • ACE Projects View

Chapter 3 : Simulation Outside of ACE

All simulation flow steps in ACE are optional. The end user has the freedom to configure their own custom simulation environment and run simulation of their design outside of ACE. The following sections give examples of how to configure custom simulation environments for each simulator outside of ACE.

General Project Setup

The following project directory structure is used in this example:

Table 7 • User Design Project Directory Structure

Directory	Description
<project_dir>	Root directory for the user design project
/src	Source file directory
/rtl	Contains source RTL for the user design
/mem_init_files	Contains memory initialization files for BRAMs or LRAMs
/tb	Contains the simulation testbench for the user design
/syn	Contains the synthesis project area and output
/ace	Contains the ACE project area and output
For Siemens Questasim	
/questasim-rtl	Contains the RTL simulation project area and output
/questasim-gate	Contains the gate-level simulation project area and output
/questasim-final	Contains the post-route (or final) simulation project area and output
For Aldec Riviera	
/riviera-rtl	Contains the RTL simulation project area and output
/riviera-gate	Contains the gate-level simulation project area and output
/riviera-final	Contains the post-route (or final) simulation project area and output

Directory	Description
For Cadence Xcelium	
/xcelium-rtl	Contains the RTL simulation project area and output
/xcelium-gate	Contains the gate-level simulation project area and output
/xcelium-final	Contains the post-route (or final) simulation project area and output
For Synopsys VCS	
/vcs-rtl	Contains the RTL simulation project area and output
/vcs-gate	Contains the gate-level simulation project area and output
/vcs-final	Contains the post-route (or final) simulation project area and output

General RTL Simulation Flow

To perform RTL simulation:

1. First create a `<sim_tool>-rtl` directory under `<project_dir>/src/`.
2. Then change directories (cd) to the new `<project_dir>/src/<sim_tool>-rtl` directory (make it the current working directory) to launch subsequent simulator commands from.
3. Create the simulation project files and add the source files and library paths. This step can be done by creating a `filelist.f` with all the library includes, design files, and compiler directives. For an example of this type of file list, see the `filelist.f` examples in the section, Siemens QuestaSim Simulator Example (page 44) Add the following to the simulator project:
   a. The top-level Achronix technology-specific simulation library include directory path (incdir): `<ace_install_dir>/libraries.`
   b. The top-level Achronix technology-specific simulation library include file, found in `<ace_install_dir>/libraries/device_models/<device>_simmodels.sv.`
   c. The behavioral RTL (Verilog or VHDL) source files for the user design.
   d. The top-level simulation testbench (Verilog or VHDL) files.
4. Run the simulation and observe the output waveform.

General Gate-Level Simulation Flow

To perform gate-level simulation:

1. Create a synthesis project in the `<project_dir>/src/syn/` directory for Synplify Pro to compile and synthesize the source behavioral RTL files to map to Achronix technology. The output of synthesis (Synplify Pro) is a mapped gate-level Verilog netlist in the format that ACE can accept as input for the back-end place-and-route flow. Synplify Pro outputs the synthesized gate-level netlist with the `*.vm` extension.

2. Create a `<sim_tool>-gate` directory under `<project_dir>/src/`.
3. Change directories (cd) to the new `<project_dir>/src/<sim_tool>-gate` directory (make it the current working directory) to launch subsequent simulator commands from.
4. Create the simulation project files and add the source files and library paths. This step can be performed by creating a `filelist.f` with all the library includes, design files, and compiler directives. For an example of this type of file list, see the `filelist.f` examples in the section, Siemens QuestaSim Simulator Example (page 44). Add the following to the simulator project:
    a. The top-level Achronix technology-specific simulation library include directory path (incdir): `<ace_install_dir>/libraries.`
    b. The top-level Achronix technology-specific simulation library include file, found in `<ace_install_dir>/libraries/device_models/<device>_simmodels.sv.`
    c. The synthesized gate-level Verilog netlist file (`*.vm`) for the user design.
    d. The top-level simulation testbench (Verilog or VHDL) files.
5. Run the simulation and observe the output waveform.

General Post-Route Simulation Flow

To perform post-route simulation:

1. Create a project in the `<project_dir>/src/ace/` directory for ACE to place and route the source synthesized gate-level Verilog netlist file (`*.vm` file output by Synplify Pro) for the user design. The user design must be run through the place-and-route flow in ACE, and the **Generate Final Simulation Netlist** flow step must be run to output the post-route gate-level netlist from ACE.
   The post-route gate-level netlist represents the user design logic after all transformations and optimizations made by the flow prior to bitstream generation. ACE outputs the post-route gate-level netlist into the following location: `<project_dir>/src/ace/<active_impl_dir>/pnr/output/<design>_final.v`. This file is encrypted using industry-standard Verilog encryption techniques which are supported by all simulators in the Achronix software tool flow.
2. Create a `<sim_tool>-final` directory under `<project_dir>/src/`.
3. Change directories (`cd`) to the new `<project_dir>/src/<sim_tool>-final` directory (make it the current working directory) to launch subsequent simulator commands from.
4. Create the simulation project files and add the source files and library paths This step can be performed by creating a `filelist.f` with all the library includes, design files, and compiler directives. For an example of this type of file list, see the `filelist.f` examples in the section, Siemens QuestaSim Simulator Example (page 44). Add the following to the simulator project:
    a. The top-level Achronix technology-specific simulation library include directory path (incdir): `<ace_install_dir>/libraries`
    b. The top-level Achronix technology-specific simulation library include file, found in `<ace_install_dir>/libraries/device_models/<device>_simmodels.sv`
    c. The encrypted post-route gate-level Verilog netlist file (`*_final.v`) for the user design.
    d. The top-level simulation testbench (Verilog or VHDL) files
5. Run the simulation and observe the output waveform.

Example Design Description

The example design used in various simulation flows described in this user guide instantiates an 8-bit LFSR that can count both up and down. There is an 8-bit output showing the result of the LFSR counter and an overflow signal.

Note

If simulating a design with BRAM or LRAM and using a memory initialization file, for example, `rom_file.txt`, it has to be present in the mem_init_files directory, relative to where the simulation tool is invoked for all simulators except Aldec Rivera. Otherwise, the simulators will not be able to find the `.txt` file and will error out. For Aldec Riviera, refer to the steps described in Aldec Riviera Simulator Example (page 26) regarding a memory initialization file, `rom_file.txt`.

lfsr_updown_cnt.v

```
`define CNT_WIDTH 8

module lfsr_updown_cnt (
            clk_in     ,     // Clock input
            rst        ,     // Reset input
            en         ,     // Enable input
            down_not_up,     // Up Down input
            cnt        ,     // Count output
            ovrflow          // Overflow output
            );

    input clk_in;
    input rst;
    input en;
    input down_not_up;

    output [`CNT_WIDTH-1 : 0] cnt;
    output                    ovrflow;

    reg [`CNT_WIDTH-1 : 0]    cnt;

    assign ovrflow = (down_not_up) ? (cnt == {{`CNT_WIDTH-1{1'b0}}, 1'b1}) :
                                    (cnt == {1'b1, {`CNT_WIDTH-1{1'b0}}});

    always @(posedge clk_in)
    begin
        if (rst)
            cnt <= {`CNT_WIDTH{1'b0}};
        else if (en) begin
            if (down_not_up) begin
                cnt <= {~(^(cnt & `CNT_WIDTH'b01100011)),cnt[`CNT_WIDTH-1:1]};
            end else begin
```

```
                cnt <= {cnt[`CNT_WIDTH-2:0],~(^(cnt &  `CNT_WIDTH'b10110001))};
            end
        end
    end // always @ (posedge clk_in)

endmodule : lfsr_updown_cnt
```

Aldec Riviera Simulator Example

RTL Simulation in Riviera

Step 1 – Create Simulation Directory

Create a RivieraSim RTL simulation project directory under `<project_dir>/src/`, then change directories (`cd`) to make `<project_dir>/src` as the current working directory to launch Riviera from.

Step 2 – Create a .do File

Create the file `riviera_script.do`. The following commands are used in the `riviera_script.do` file to compile and simulate the design.

```
#Creates a workspace called riviera_rtl_workspace in current directory. It automatically
changes directory to ./riviera_rtl_workspace
workspace.create riviera_rtl_workspace ./

#Creates a design called lfsr_updown_cnt
workspace.design.create lfsr_updown_cnt ./

#Save workspace
workspace.save

#If using a memory initialization file, copy the <mem_file>.txt to riviera_rtl_workspace
directory like this
# cp ../mem_init_files/mem_file.txt .

#Add the design RTL
design.file.add ../rtl/lfsr_updown_cnt.v
design.file.add ../tb/lfsr_updown_cnt_tb.v
design.file.add /<ACE_INSTALL_DIR>/libraries/device_models/<DEVICE>_simmodels.sv

#Compile design
#design.compile test
alog -work lfsr_updown_cnt -incdir {/<ACE_INSTALL_DIR>/libraries/} -msg 5 -dbg -protect
0 -quiet {/<project_dir>/src/rtl/lfsr_updown_cnt.v} {/<project_dir>/src/tb/
lfsr_updown_cnt_tb.v} {/<ace_install_path>/libraries/device_models/
<DEVICE>_simmodels.sv}

#Initialize design
#design.simulation.initialize lfsr_updown_cnt_tb
asim -lib lfsr_updown_cnt -dbg -t 0 -dataset {/<project_dir>/src/results/
lfsr_updown_cnt} -datasetname {sim} lfsr_updown_cnt_tb

#Run
run -all

#Saves workspace
workspace.close -save

quit
```

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Step 3 – Run the Simulation

Run the script `riviera_script.do` to compile and simulate the design using the following command. The results are then copied to the `sim.log` file.

```
/<RIVIERA_INSTALL_DIR>/riviera-pro-2014.06-x86_64/bin/vsimsa -do ./riviera_script.do >
sim.log
```

Once simulation is complete, the `sim.log` displays a message similar to following:

sim.log

```
# ELAB2: Elaboration final pass complete - time: 0.3 [s].
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 7432 kB (elbread=1450 elab2=5844 kernel=137 sdf=0)
# KERNEL: ASDB file was created in location /simulation_example_design/src/results/
lfsr_updown_cnt/dataset.asdb
run -all
# KERNEL: Reset released
# KERNEL:                  87000: rst 0 en 1 updown 0 cnt 11000011 overflow 0
# KERNEL:                 129000: rst 0 en 1 updown 1 cnt 00000010 overflow 0
# KERNEL:                 SIMULATION PASSED!!
# RUNTIME: Info: RUNTIME_0068 lfsr_updown_cnt_tb.v (66): $finish called.
# KERNEL: Time: 169 ns,  Iteration: 1,  Instance: /lfsr_updown_cnt_tb,  Process:
@INITIAL#41_2@.
# KERNEL: stopped at time: 169 ns
# VSIM: Simulation has finished. There are no more test vectors to simulate.
workspace.close -save
quit
```

```
# VSIM: Simulation has finished.
```

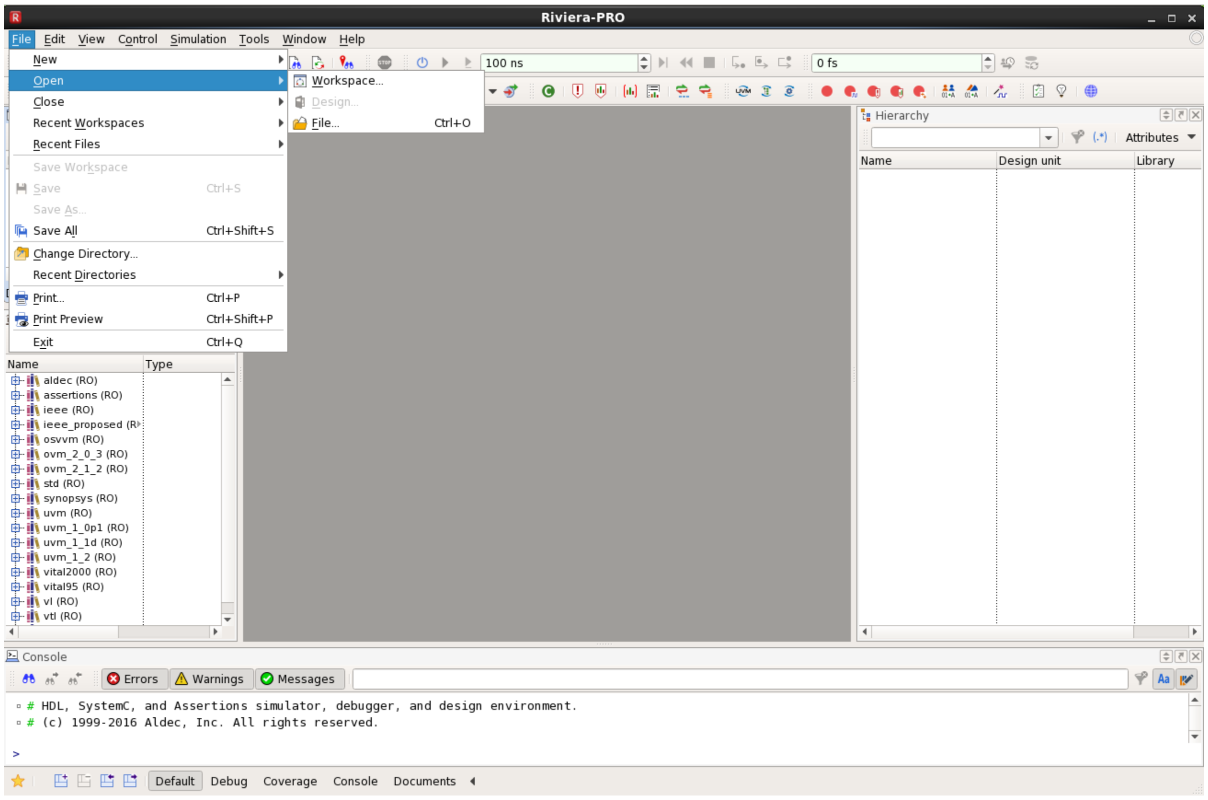
Step 4 – View the Waveform

In Riviera, in order to view the waveform, simulation has to be rerun. Change the directory to `/riviera_rtl_workspace`. Invoke the Riviera GUI from this directory using the following command.

```
/<RIVIERA_INSTALL_DIR>/riviera-pro-2014.06-x86_64/bin/riviera &
```

Step 5 – Open the Workspace

After the Riviera GUI starts up, open the workspace file `riviera_rtl_workspace.rwsp` by selecting **File** → **Open** → **Workspace**.



```
# HDL, SystemC, and Assertions simulator, debugger, and design environment.
# (c) 1999-2016 Aldec, Inc. All rights reserved.
>
```

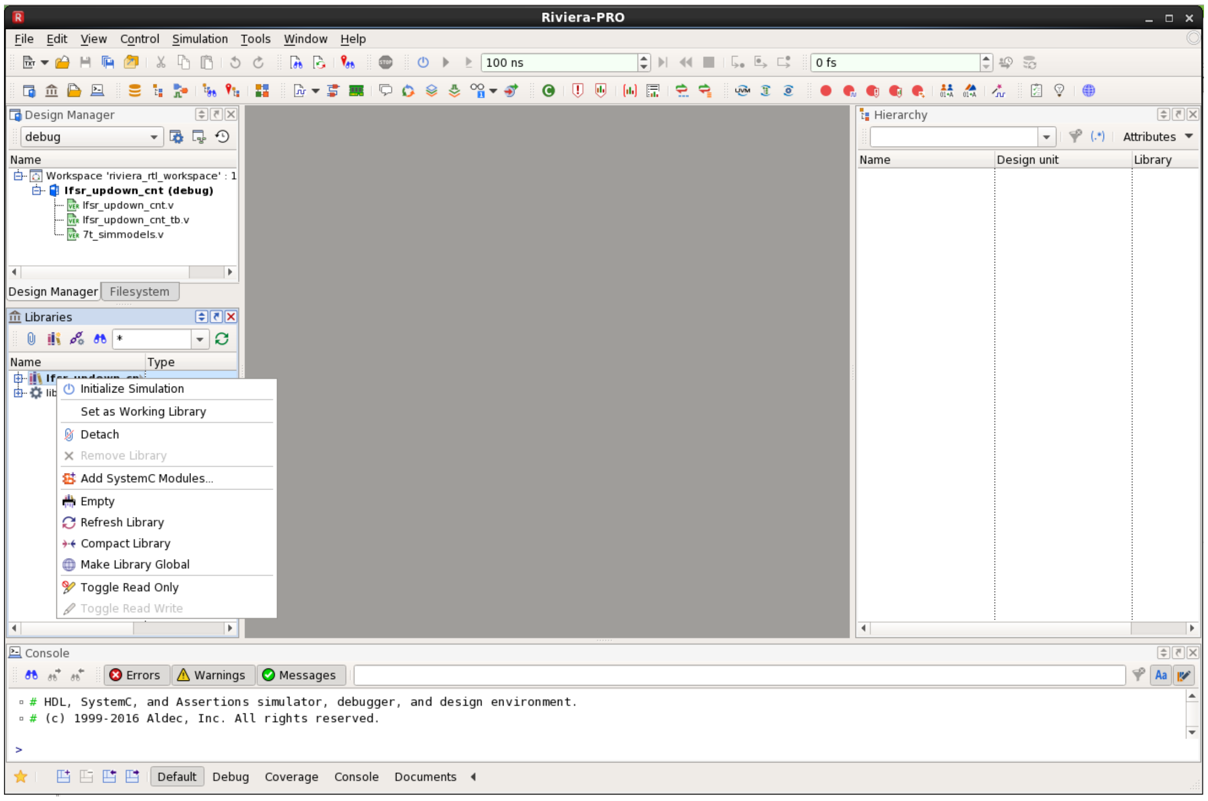
Figure 13 • Riviera Workspace

Note

If using a memory initialization file when simulating BRAM or LRAM, copying the `rom_file.txt` to the workspace design directory is important; otherwise, when re-running simulation for viewing waveform, the `rom_file.txt` will not be found.

Step 6 – Initialize the Simulation

Initialize simulation as shown below by right-clicking on the file `lfsr_updown_cnt_tb.v` in the Libraries pane and selecting **Initialize Simulation**.

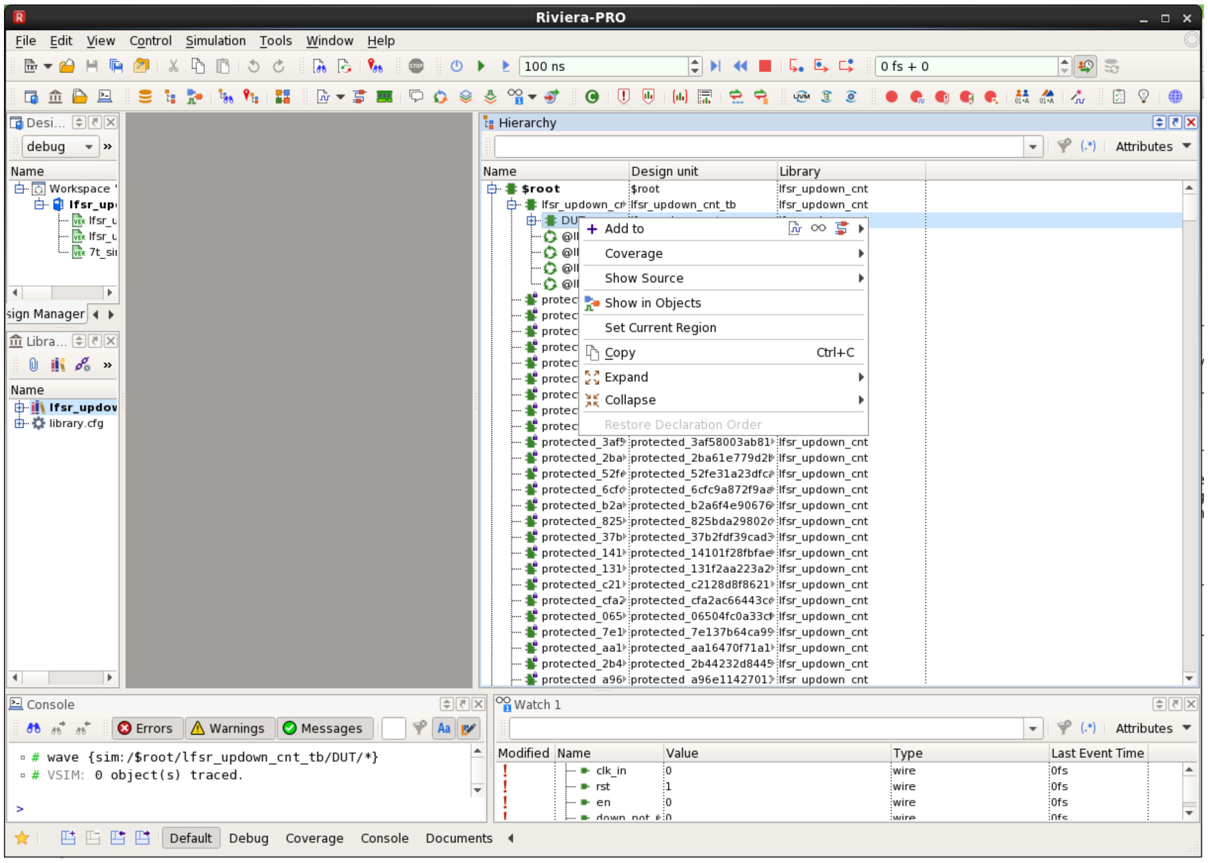


```
# HDL, SystemC, and Assertions simulator, debugger, and design environment.
# (c) 1999-2016 Aldec, Inc. All rights reserved.
>
```

Figure 14 • Initializing Simulation

Step 7 – Add Signals to the Waveform

Add signals to the waveform by right-clicking the DUT in the Hierarchy pane and selecting **Add to → Waveform**.



Name	Design unit	Library
$root	$root	lfsr_updown_cnt
lfsr_updown_cn	lfsr_updown_cnt_tb	lfsr_updown_cnt
protected_3af	protected_3af58003ab81	lfsr_updown_cnt
protected_2ba	protected_2ba61e779d2b	lfsr_updown_cnt
protected_52f	protected_52fe31a23dfc	lfsr_updown_cnt
protected_6cf	protected_6cfc9a872f9aa	lfsr_updown_cnt
protected_b2a	protected_b2a6f4e90676	lfsr_updown_cnt
protected_825	protected_825bda29802	lfsr_updown_cnt
protected_37b	protected_37b2fdf39cad3	lfsr_updown_cnt
protected_141	protected_14101f28fbfae	lfsr_updown_cnt
protected_131	protected_131f2aa223a2	lfsr_updown_cnt
protected_c21	protected_c2128d8f8621	lfsr_updown_cnt
protected_cfa2	protected_cfa2ac66443c	lfsr_updown_cnt
protected_065	protected_06504fc0a33c	lfsr_updown_cnt
protected_7e1	protected_7e137b64ca99	lfsr_updown_cnt
protected_aa1	protected_aa16470f71a1	lfsr_updown_cnt
protected_2b4	protected_2b44232d8445	lfsr_updown_cnt
protected_a96	protected_a96e1142701	lfsr_updown_cnt

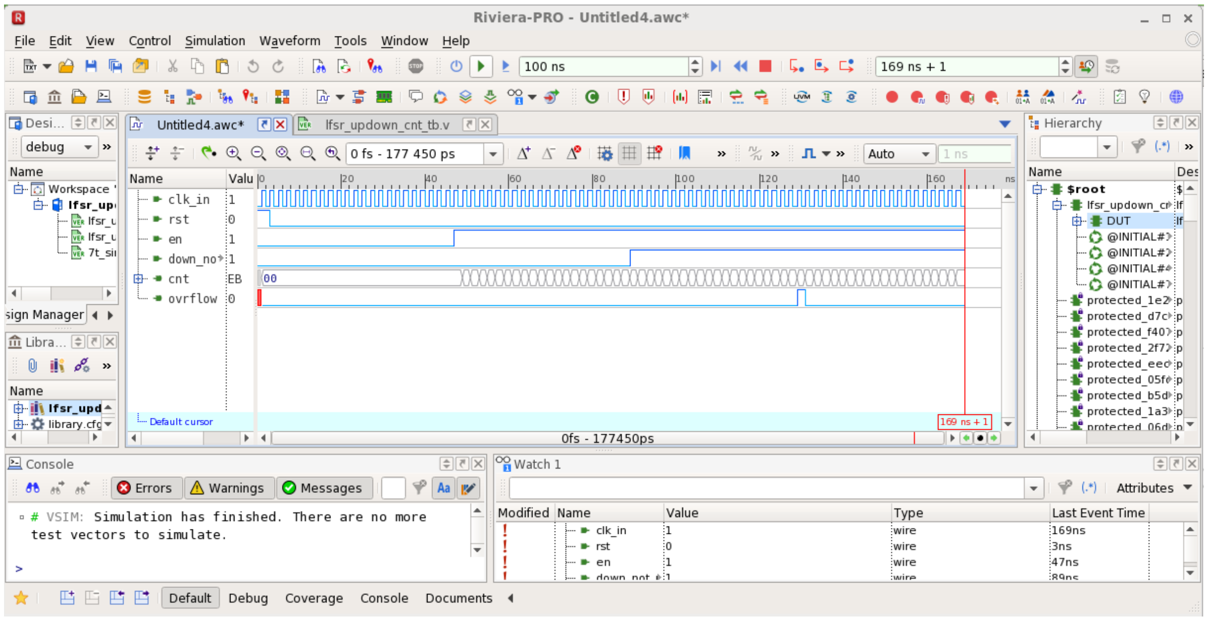
```
# wave {sim:/$root/lfsr_updown_cnt_tb/DUT/*}
# VSIM: 0 object(s) traced.
```

Modified	Name	Value	Type	Last Event Time
!	clk_in	0	wire	0fs
!	rst	1	wire	0fs
!	en	0	wire	0fs

Figure 15 • Adding the Waveform

Step 8 – View the Waveform

Click the **Restart** and **Run -all** button to view the waveform.



```
# VSIM: Simulation has finished. There are no more
test vectors to simulate.
```

Modified	Name	Value	Type	Last Event Time
!	clk_in	1	wire	169ns
!	rst	0	wire	3ns
!	en	1	wire	47ns
!	down_not	1	wire	89ns

Figure 16 • Viewing the Waveform

Gate-Level Simulation in Riviera

For gate-level simulation, a synthesized netlist has to first be generated using Synplify Pro before performing the simulation.

Step 1 – Create the Synthesis Project

Create a new project in Synplify under `<project_dir>/src/syn`, include <ACE_INSTALL_DIR>`/libraries/device_models/<DEVICE>_synplify.sv` followed by the RTL design files and constraint files.

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Step 2 – Synthesize the Design

Synthesize the design using Synplify Pro. Synplify Pro generates a gate-level netlist with `.vm` extension, for the example design, the file `<project_dir>/src/syn/rev_acx/lfsr_updown_cnt.vm` is generated.

Step 3 – Create a Workspace

To run the gate-level simulation, create a workspace and run the script as described in RTL Simulation in Riviera (page 26) except that the gate-level simulation uses the mapped netlist

```
#Creates a workspace called riviera_gate_workspace in current directory. It
automatically changes directory to ./riviera_gate_workspace
workspace.create riviera_gate_workspace ./
```

```
#Creates a design called lfsr_updown_cnt
workspace.design.create lfsr_updown_cnt ./

workspace.save

#If using a memory initialization file, copy the <mem_file>.txt to riviera_rtl_workspace
directory like this
# cp ../mem_init_files/mem_file.txt .

#Add the gate netlist
design.file.add ../syn/rev_2/lfsr_updown_cnt.vm
design.file.add ../tb/lfsr_updown_cnt_tb.v
design.file.add /<ACE_INSTALL_DIR>/libraries/device_models/<DEVICE>_simmodels.sv

#Compile design
#design.compile test
alog -work lfsr_updown_cnt -incdir {/<ACE_INSTALL_DIR>/libraries/} -msg 5 -dbg -protect
0 -quiet {/<project_dir>/src/rtl/lfsr_updown_cnt.v} {/<project_dir>/src/tb/
lfsr_updown_cnt_tb.v} {/<ACE_INSTALL_DIR>/libraries/device_models/<DEVICE>_simmodels.sv}

#Initialize design
#design.simulation.initialize lfsr_updown_cnt_tb
asim -lib lfsr_updown_cnt -dbg -t 0 -dataset {/<project_dir>/src/results/
lfsr_updown_cnt} -datasetname {sim} lfsr_updown_cnt_tb

#Run
run -all

#Saves workspace
workspace.close -save

quit
```

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Step 4 – Run the Simulation

Follow Step 3 of RTL Simulation in Riviera (page 26) to run simulation.

Step 5 – View the Results

Follow Steps 4 to the end of RTL Simulation in Riviera (page 26) to view the waveform.

Post-Route Simulation in Riviera

For post-route simulation, the synthesized gate-level netlist must first be run through place and route using ACE.

Step 1 – Create the ACE Project

Create a new project in ACE under `<project_dir>/src/ace`. Add the gate-level netlist `lfsr_updown_cnt.vm` generated by Synplify Pro plus the constraint files.

Step 2 – Run Place and Route

Run the place and route flow, including the 'Generate Final Simulation Netlist' step to obtain a post-route netlist. In this example, the netlist is generated under `<project_dir>/src/ace/impl_1/output/lfsr_updown_cnt_final.v`.

Step 3 – Create the Workspace

Run the post-route simulation by creating a workspace called final and run the script as described in the RTL Simulation in Riviera (page 26) section except that the post-route simulation uses the final netlist.

```
#Creates a workspace called riviera_final_workspace in current directory. It
automatically changes directory to ./riviera_final_workspace
workspace.create riviera_final_workspace ./

#Creates a design called lfsr_updown_cnt
workspace.design.create lfsr_updown_cnt ./

workspace.save

#Copy the in/exp files
cp ../tb/testvectors.in .
cp ../tb/testvectors.exp .

#If using a memory initialization file, copy the <mem_file>.txt to riviera_rtl_workspace
directory like this
# cp ../mem_init_files/mem_file.txt .

#Add the final netlist
design.file.add ../ace/impl_1/output/lfsr_updown_cnt_final.v
design.file.add ../tb/lfsr_updown_cnt_tb.v
design.file.add /<ACE_INSTALL_DIR>/libraries/device_models/<DEVICE>_simmodels.sv

#Compile design
#design.compile test
alog -work lfsr_updown_cnt -incdir {/<ACE_INSTALL_DIR>/libraries/} -msg 5 -dbg -protect
0 -quiet {/<project_dir>/src/rtl/lfsr_updown_cnt.v} {/<project_dir>/src/tb/
lfsr_updown_cnt_tb.v} {/<ace_install_path>/libraries/device_models/
<DEVICE>_simmodels.sv}

#Initialize design
#design.simulation.initialize lfsr_updown_cnt_tb
asim -lib lfsr_updown_cnt -dbg -t 0 -dataset {/<project_dir>/src/results/
lfsr_updown_cnt} -datasetname {sim} lfsr_updown_cnt_tb
```

```
#Run
run -all

#Saves workspace
workspace.close -save

quit
```

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Step 4 – Run the Simulation

Follow Step 3 of RTL Simulation in Riviera (page 26) to run simulation.

Step 5 – View the Results

Follow Steps 4 to the end of RTL Simulation in Riviera (page 26) to view the waveform.

Cadence Xcelium Simulator Example

RTL Simulation in Xcelium

In order to run the RTL simulation using the Cadence Xcelium tool, follow the steps below:

Caution!

The user must set the `XCELIUM define in order for the Achronix libraries to compile correctly.

Step 1 – Invoke the Xcelium Tool

Invoke the Cadence tool by specifying the path where the tool is installed using the command **xrun** and include the libraries path, path to `<DEVICE>_simmodels.sv`, top-level RTL file, and testbench as shown below:

```
xrun -access +rwc +incdir+<ACE_INSTALL_DIR>/libraries/ /<ACE_INSTALL_DIR>/libraries/
device_models/<DEVICE>_simmodels.sv <project_dir>/src/rtl/lfsr_updown_cnt.v
<project_dir>/src/tb/lfsr_updown_cnt_tb.v -gui -sv
```

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Table 8 • Xcelium Command Options

Options	Required	Description
-access +rwc	Yes	Turn on read, write and/or connectivity access
-gui	No	Invoke the GUI
-sv	Yes	Supports System Verilog constructs
+define+CMEM_READBACK	No	This option is to be used only when reading back configuration memory (CMEM) contents during WGL simulations. **Note** Simulating CMEM readback function takes on the order of hours to complete with the Xcelium simulator.

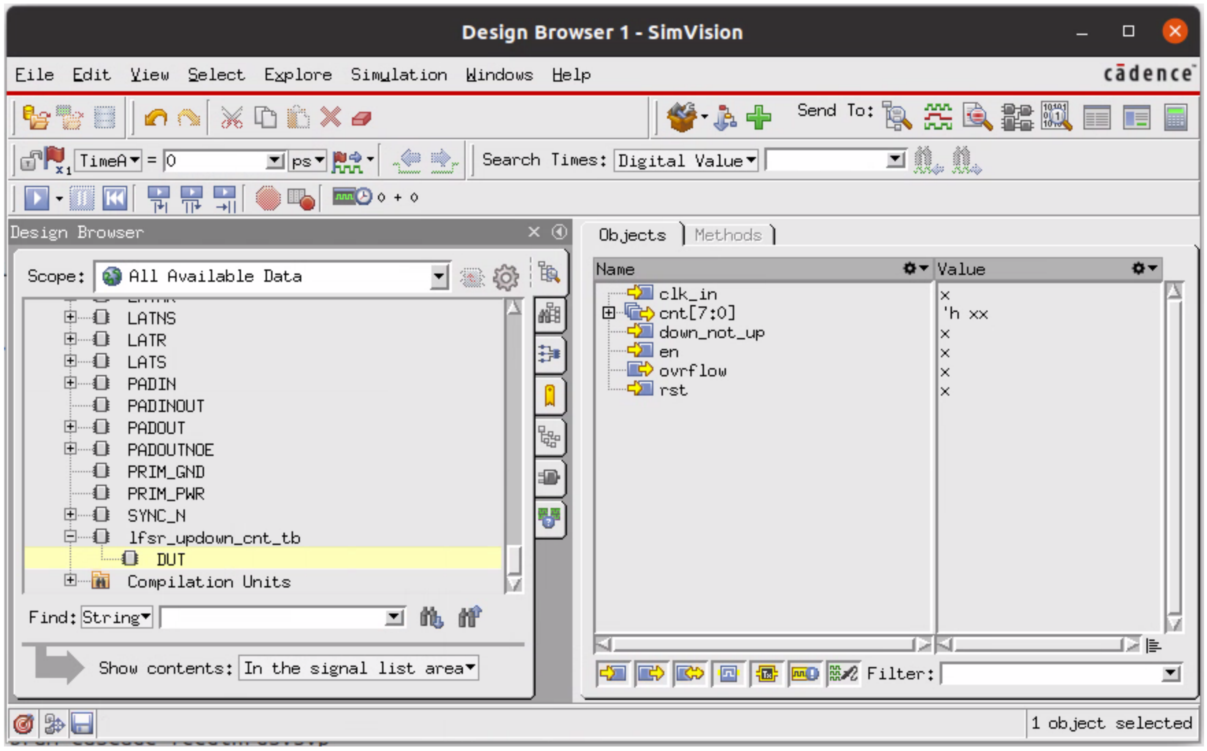
When the compilation and elaboration of the design completes successfully, the following message is displayed at the end of `xrun.log` created in the same directory where the tool is run. Also the graphical debug environment, the SimVision GUI, is launched as shown in Step 2.

```
    Building instance specific data structures.
    Loading native compiled code:     .................... Done
    Design hierarchy summary:
                          Instances  Unique
        Modules:                837     258
        UDPs:                     0       5
        Timing outputs:           4       1
        Registers:             3671     548
        Scalar wires:          8208       -
        Expanded wires:         695      36
        Vectored wires:        1084       -
        Named events:            40       4
        Always blocks:         2821     286
        Initial blocks:         517     111
        Cont. assignments:     2988    1421
        Pseudo assignments:     631     380
        Compilation units:        1       1
        Simulation timescale:   1ps
    Writing initial simulation snapshot:
Loading snapshot worklib.bram_outputs:vp .................... Done
SVSEED default: 1
```

```
ncsim: *W,DSEM2009: This SystemVerilog design is simulated as per IEEE 1800-2009
SystemVerilog simulation semantics. Use -disable_sem2009 option for turning off SV 2009
simulation semantics.
```

Step 2 – Add Signals to the Waveform

From the Design Browser pane in the SimVision main window, scroll down to and select the testbench, `lfsr_updown_cnt_tb`.



Name	Value
clk_in	x
cnt[7:0]	'h xx
down_not_up	x
en	x
ovrflow	x
rst	x

Figure 17 • SimVision Main Window

From the Objects view, select the DUT and all of the required signals. Right-click, select **Send to Waveform Window** to add the signals.

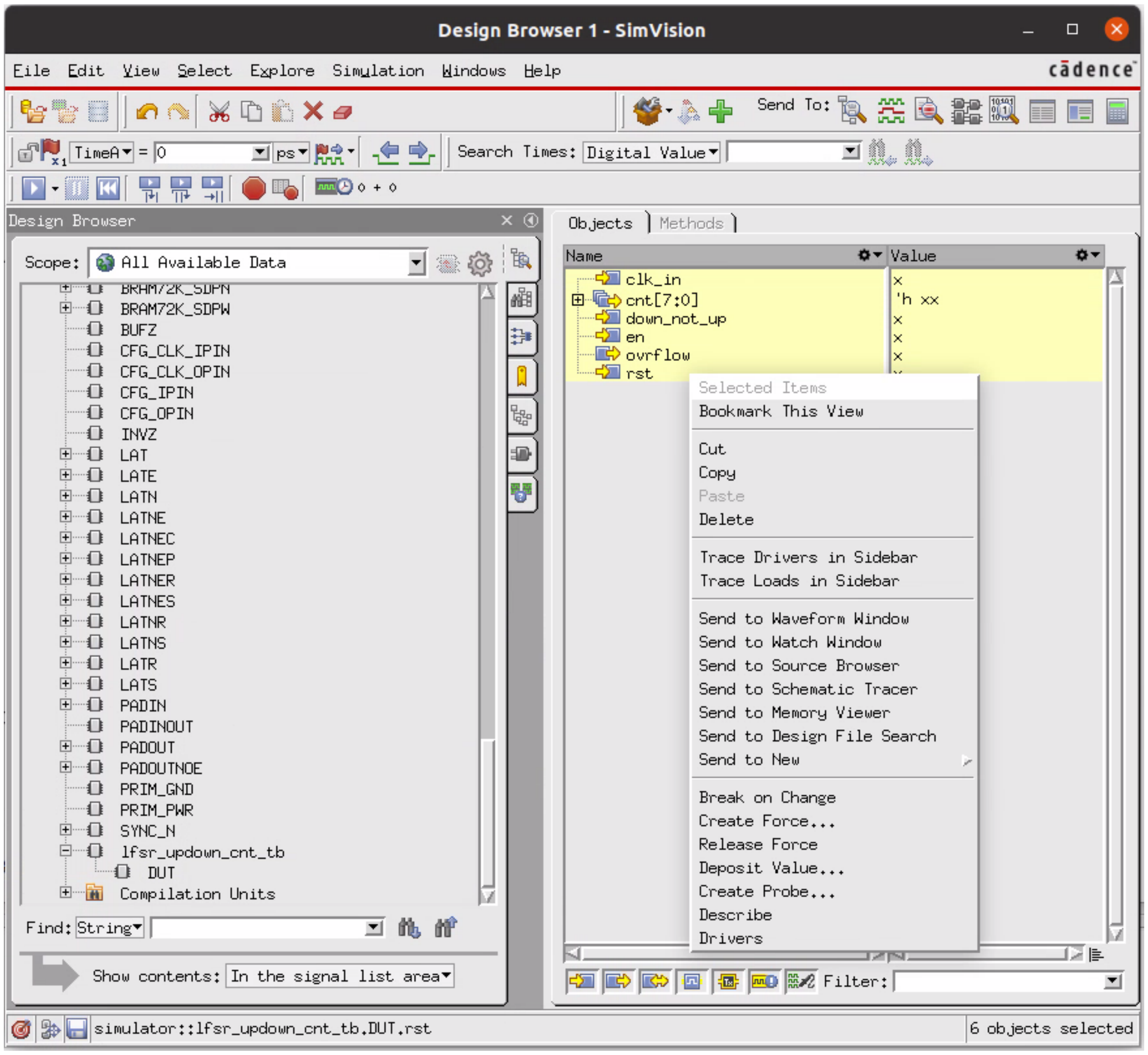


Figure 18 • Adding Signals to the Waveform

The Waveform window will open as shown below.

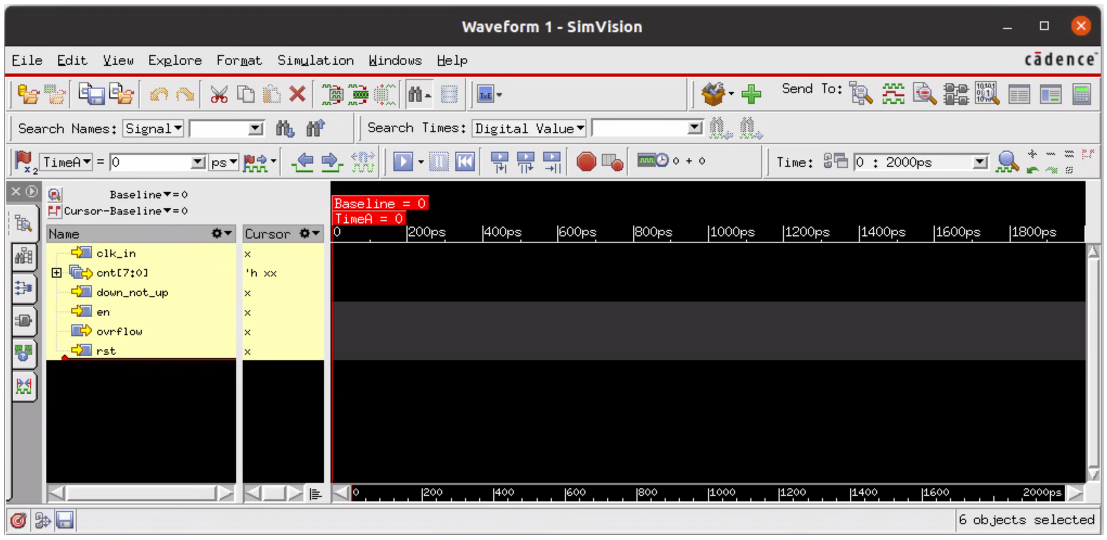


Figure 19 • Xcelium Waveform Window

Step 3 – Run the Simulation

Run simulation by selecting **Simulation** → **Run**.

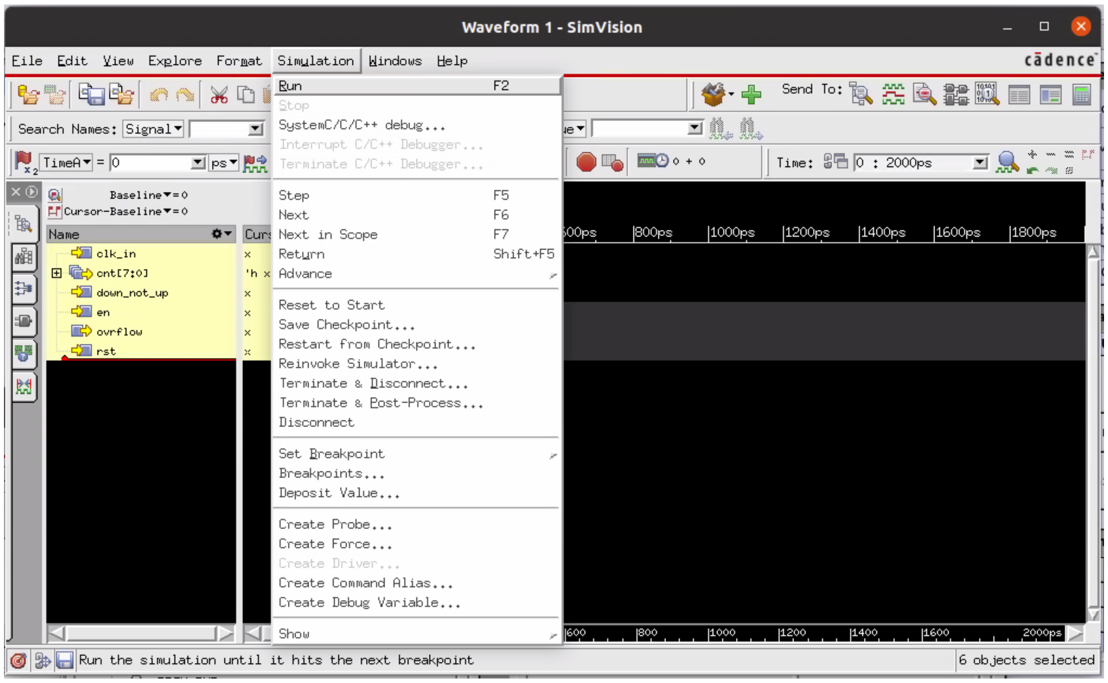


Figure 20 • Running Simulation

Step 4 – View the Waveform

Select the View tab and Zoom Full X. The waveform will be displayed as shown below.

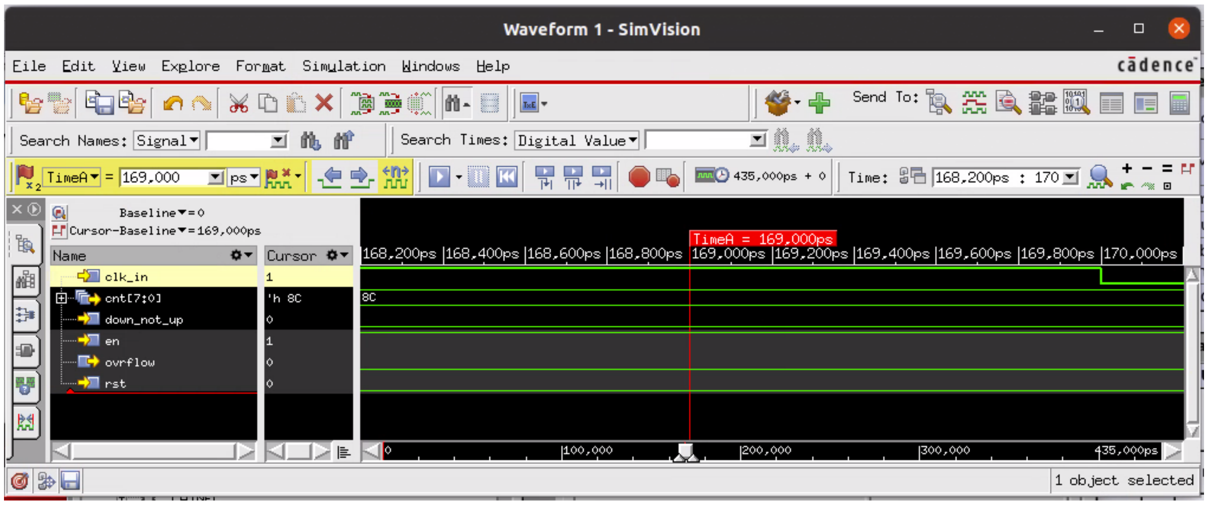
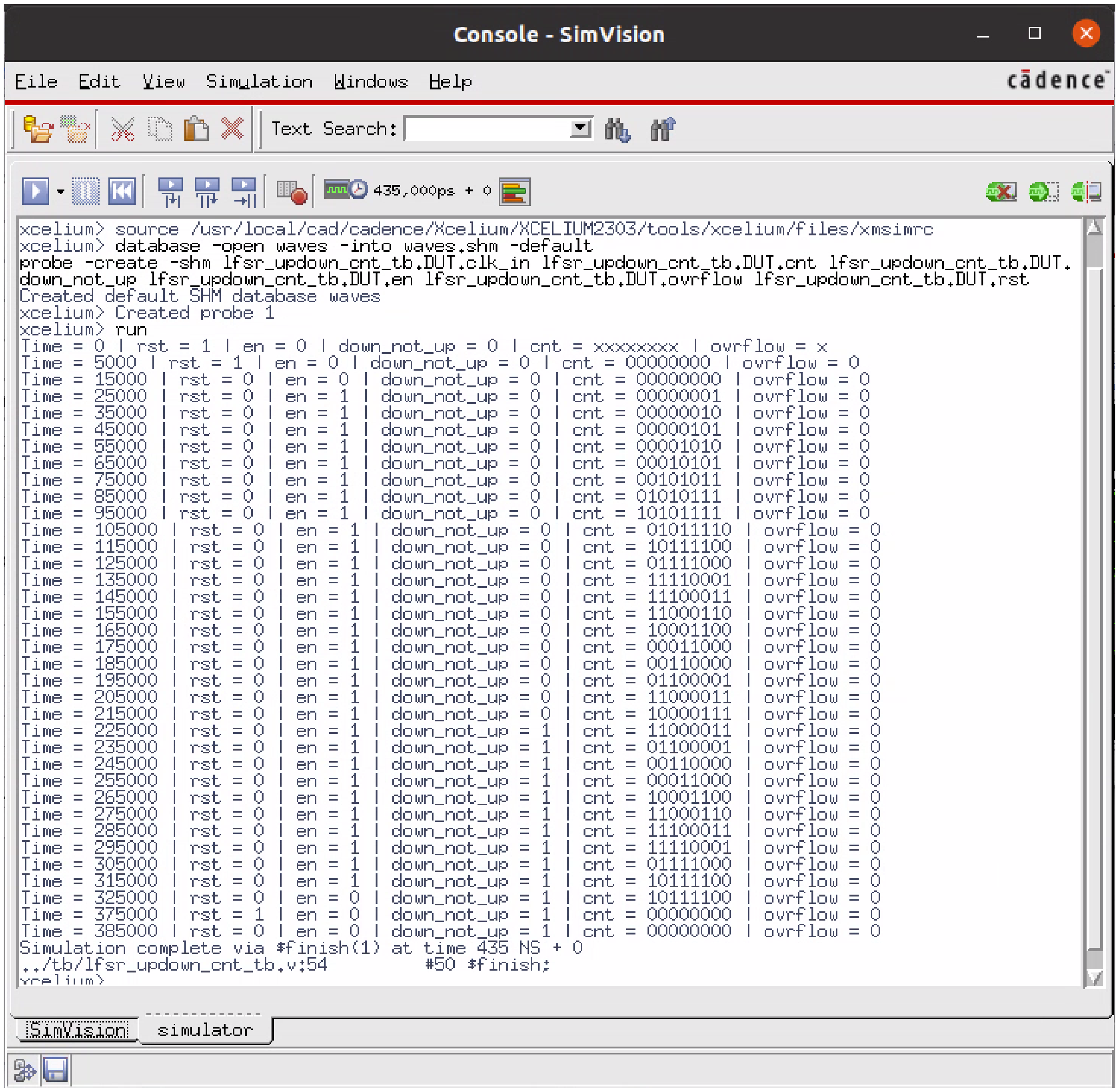


Figure 21 • SimVision Waveform Window

Step 5 – View Console Messages

The console window displays messages from the testbench. It indicates that the test passed successfully and the simulation completion time.



```
xcelium> source /usr/local/cad/cadence/Xcelium/XCELIUM2303/tools/xcelium/files/xmsimrc
xcelium> database -open waves -into waves.shm -default
probe -create -shm lfsr_updown_cnt_tb.DUT.clk_in lfsr_updown_cnt_tb.DUT.cnt lfsr_updown_cnt_tb.DUT.
down_not_up lfsr_updown_cnt_tb.DUT.en lfsr_updown_cnt_tb.DUT.ovrflow lfsr_updown_cnt_tb.DUT.rst
Created default SHM database waves
xcelium> Created probe 1
xcelium> run
Time = 0 | rst = 1 | en = 0 | down_not_up = 0 | cnt = xxxxxxxx | ovrflow = x
Time = 5000 | rst = 1 | en = 0 | down_not_up = 0 | cnt = 00000000 | ovrflow = 0
Time = 15000 | rst = 0 | en = 0 | down_not_up = 0 | cnt = 00000000 | ovrflow = 0
Time = 25000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 00000001 | ovrflow = 0
Time = 35000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 00000010 | ovrflow = 0
Time = 45000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 00000101 | ovrflow = 0
Time = 55000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 00001010 | ovrflow = 0
Time = 65000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 00010101 | ovrflow = 0
Time = 75000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 00101011 | ovrflow = 0
Time = 85000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 01010111 | ovrflow = 0
Time = 95000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 10101111 | ovrflow = 0
Time = 105000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 01011110 | ovrflow = 0
Time = 115000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 10111100 | ovrflow = 0
Time = 125000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 01111000 | ovrflow = 0
Time = 135000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 11110001 | ovrflow = 0
Time = 145000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 11100011 | ovrflow = 0
Time = 155000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 11000110 | ovrflow = 0
Time = 165000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 10001100 | ovrflow = 0
Time = 175000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 00011000 | ovrflow = 0
Time = 185000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 00110000 | ovrflow = 0
Time = 195000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 01100001 | ovrflow = 0
Time = 205000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 11000011 | ovrflow = 0
Time = 215000 | rst = 0 | en = 1 | down_not_up = 0 | cnt = 10000111 | ovrflow = 0
Time = 225000 | rst = 0 | en = 1 | down_not_up = 1 | cnt = 11000011 | ovrflow = 0
Time = 235000 | rst = 0 | en = 1 | down_not_up = 1 | cnt = 01100001 | ovrflow = 0
Time = 245000 | rst = 0 | en = 1 | down_not_up = 1 | cnt = 00110000 | ovrflow = 0
Time = 255000 | rst = 0 | en = 1 | down_not_up = 1 | cnt = 00011000 | ovrflow = 0
Time = 265000 | rst = 0 | en = 1 | down_not_up = 1 | cnt = 10001100 | ovrflow = 0
Time = 275000 | rst = 0 | en = 1 | down_not_up = 1 | cnt = 11000110 | ovrflow = 0
Time = 285000 | rst = 0 | en = 1 | down_not_up = 1 | cnt = 11100011 | ovrflow = 0
Time = 295000 | rst = 0 | en = 1 | down_not_up = 1 | cnt = 11110001 | ovrflow = 0
Time = 305000 | rst = 0 | en = 1 | down_not_up = 1 | cnt = 01111000 | ovrflow = 0
Time = 315000 | rst = 0 | en = 1 | down_not_up = 1 | cnt = 10111100 | ovrflow = 0
Time = 325000 | rst = 0 | en = 0 | down_not_up = 1 | cnt = 10111100 | ovrflow = 0
Time = 375000 | rst = 1 | en = 0 | down_not_up = 1 | cnt = 00000000 | ovrflow = 0
Time = 385000 | rst = 0 | en = 0 | down_not_up = 1 | cnt = 00000000 | ovrflow = 0
Simulation complete via $finish(1) at time 435 NS + 0
../tb/lfsr_updown_cnt_tb.v:54          #50 $finish;
xcelium>
```

Figure 22 • SimVision Console Window

Gate-Level Simulation in Xcelium

For gate-level simulation, a synthesized netlist has to first be generated using Synplify Pro before performing the simulation.

Step 1 – Create the Synthesis Project

Create a new project in Synplify under `<project_dir>/src/syn`, include <ACE_INSTALL_DIR>`/libraries/device_models/<DEVICE>_synplify.sv` followed by the RTL design files and constraint files.

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Step 2 – Synthesize the Design

Synthesize the design using Synplify Pro. Synplify Pro generates a gate-level netlist with `.vm` extension, for the example design, the file `<project_dir>/src/syn/rev_acx/lfsr_updown_cnt.vm` is generated.

Rename the gate-level netlist extension from `.vm` to `.v` so that the Xcelium simulator understands that it is a Verilog file. Otherwise, the simulator will error out. The example netlist is renamed to `lfsr_updown_cnt_gate.v`.

Step 3 – Run Simulation

To run the gate-level simulation, use the same command as described in the **RTL Simulation in Xcelium** (page 34) section, except that the gate-level simulation uses the mapped netlist `lfsr_updown_cnt_gate.v` file instead of source RTL files.

Note

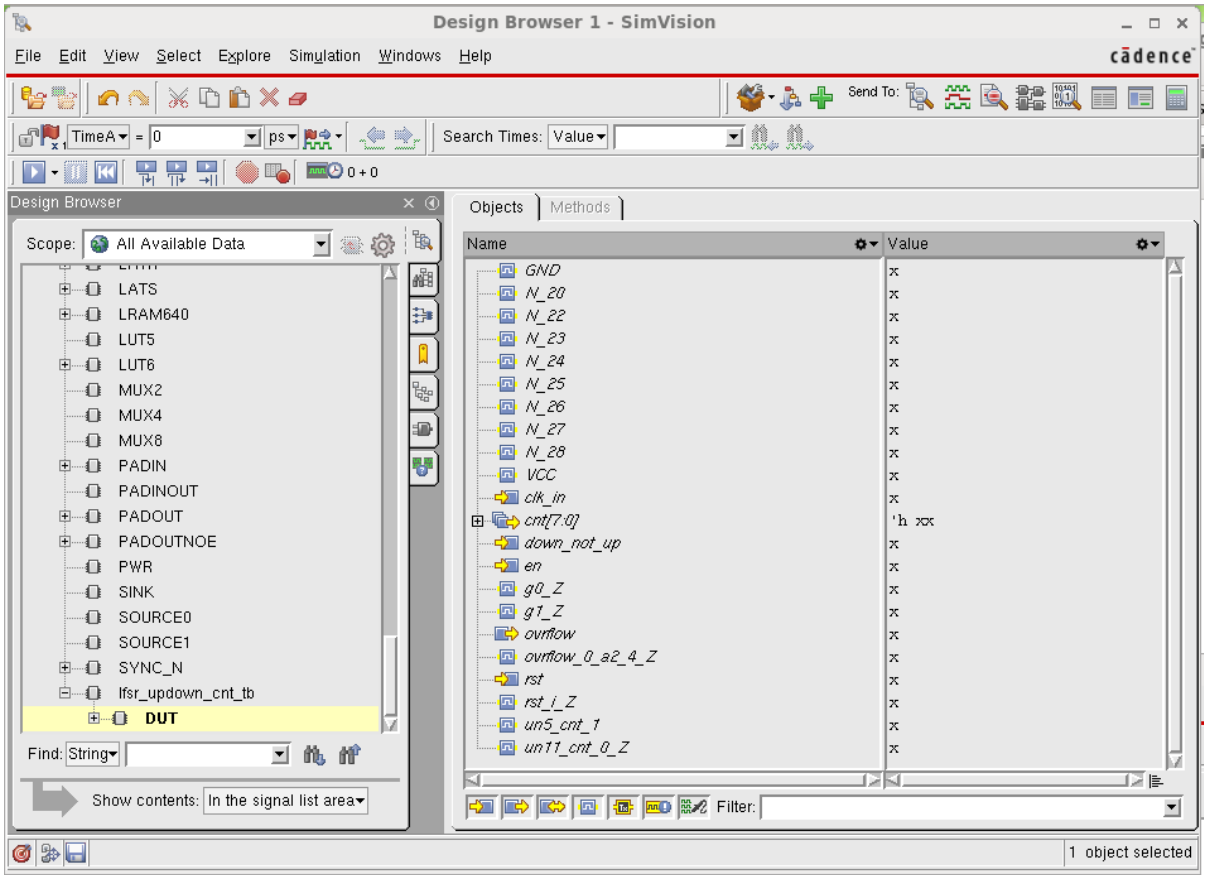
If ACE-driven synthesis is used, the netlist file will be under the `<project_output_dir>/syn/rev_acx/` directory.

```
xrun -access +rwc +incdir+<ACE_INSTALL_DIR>/libraries/ <ACE_INSTALL_DIR>/libraries/
device_models/<DEVICE>_simmodels.sv <project_dir>/src/syn/rev_acx/lfsr_updown_cnt_gate.v
<project_dir>/src/tb/lfsr_updown_cnt_tb.v -gui -sv
```

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Step 4 – View Simulation Results

Follow the same steps described in **RTL Simulation in Xcelium** (page 34) (Steps 2 to 5) for loading the waveform and viewing the results. When the DUT is selected, the gate-level netlist signals appear as shown below.


Design Browser 1 - SimVision
File Edit View Select Explore Simulation Windows Help
cādence
Send To:
TimeA = 0 ps
Search Times: Value
0 + 0
Design Browser
Objects
Methods
Find: String
Show contents: In the signal list area
Filter:
1 object selected

Scope: All Available Data

- LATS
- LRAM640
- LUT5
- LUT6
- MUX2
- MUX4
- MUX8
- PADIN
- PADINOUT
- PADOUT
- PADOUTNOE
- PWR
- SINK
- SOURCE0
- SOURCE1
- SYNC_N
- lfsr_updown_cnt_tb
  - DUT

Name	Value
GND	x
N_20	x
N_22	x
N_23	x
N_24	x
N_25	x
N_26	x
N_27	x
N_28	x
VCC	x
clk_in	x
cnt[7:0]	'h xx
down_not_up	x
en	x
g0_Z	x
g1_Z	x
ovrflow	x
ovrflow_0_a2_4_Z	x
rst	x
rst_i_Z	x
un5_cnt_1	x
un11_cnt_0_Z	x

Figure 23 • Adding Signals to the Waveform

Post-Route Simulation in Xcelium

For post-route simulation, the synthesized gate-level netlist must first be run through place and route using ACE.

Step 1 – Create the ACE Project

Create a new project in ACE under `<project_dir>/src/ace`. Add the gate-level netlist `lfsr_updown_cnt.vm` generated by Synplify Pro plus the constraint files.

Step 2 – Run Place and Route

Run the place and route flow, including the 'Generate Final Simulation Netlist' step to obtain a post-route netlist. In this example, the netlist is generated under `<project_dir>/src/ace/impl_1/output/lfsr_updown_cnt_final.v`.

Step 3 – Run Simulation

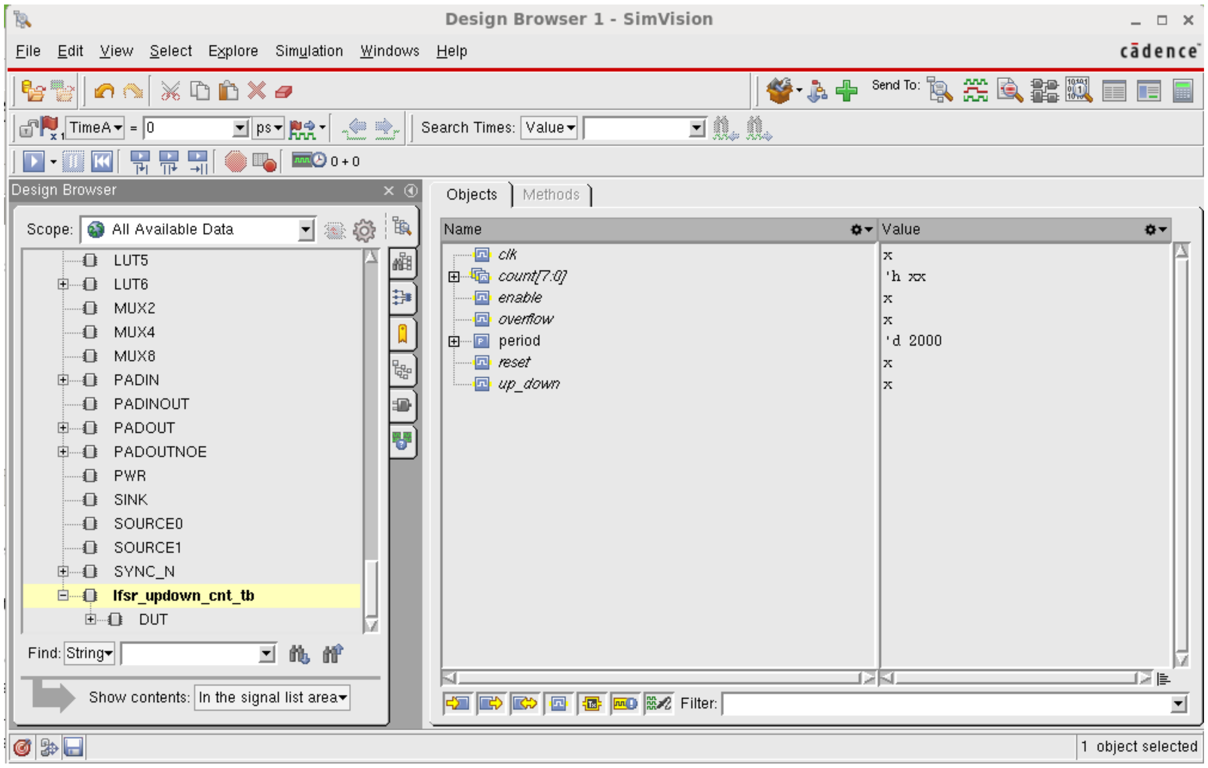
Run the post-route simulation using the same command as was used in RTL and gate-level simulation.

```
xrun -access +rwc +incdir+<ACE_INSTALL_DIR>/libraries/ <ACE_INSTALL_DIR>/libraries/
device_models/<DEVICE>_simmodels.sv <project_dir>/src/ace/impl_1/output/
lfsr_updown_cnt_final.v <project_dir>/src/tb/lfsr_updown_cnt_tb.v -gui -sv
```

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Step 4 – View Simulation Results

In post-route simulation, when DUT is selected, no signals are displayed since the post-route netlist file is encrypted. Instead, select the signals under lfsr_updown_cnt_tb and follow the same steps described in RTL Simulation in Xcelium (page 34) (Steps 2 to 5) for loading the waveform and viewing the results.


Design Browser 1 - SimVision
File Edit View Select Explore Simulation Windows Help
cādence
Send To:
TimeA = 0 ps
Search Times: Value
0 + 0
Design Browser
Objects Methods
Find: String
Show contents: In the signal list area
Filter:
1 object selected

Scope: All Available Data

- LUT5
- LUT6
- MUX2
- MUX4
- MUX8
- PADIN
- PADINOUT
- PADOUT
- PADOUTNOE
- PWR
- SINK
- SOURCE0
- SOURCE1
- SYNC_N
- **lfsr_updown_cnt_tb**
  - DUT

Name	Value
clk	x
count[7:0]	'h xx
enable	x
overflow	x
period	'd 2000
reset	x
up_down	x

Figure 24 • Adding Signals to the Waveform

Siemens QuestaSim Simulator Example

RTL Simulation in QuestaSim

Step 1 - Create the Project

Create the QuestaSim RTL simulation project directory under `<project_dir>/src/questasim-rtl`, then change directories (`cd`) to make it the current working directory to launch the simulator from.

Step 2 - Initialize the Work Library

Initialize the simulator work library using the following QuestaSim command:

```
vlib <project_dir>/src/questasim-rtl/work
```

Step 3 - Create the File List

Create a `filelist.f` file to configure the project defines, include directories, and source files. If using VHDL, the file should appear similar to the following example:

Warning!

VHDL simulation is only supported at the RTL simulation level. Achronix does not provide the VHDL wrapper libraries for the Verilog library primitives. Behavioral VHDL is recommended. For Achronix IP, such as BRAM or DSP blocks, the ACE GUI provides IP configuration tools which can generate a VHDL wrapper on top of the configured Verilog primitive wrapper.

Example vhdl_filelist.f

```
+incdir+<project_dir>/src/tb
+incdir+<project_dir>/src/rtl
+incdir+<ACE_INSTALL_DIR>/libraries
+incdir+<ace_ext_dir>/libraries
<ace_install>/libraries/device_models/<DEVICE>_simmodels.sv
<project_dir>/src/rtl/lfsr_updown_cnt.vhd
<project_dir>/src/tb/lfsr_updown_cnt_tb.vhd
```

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

If using Verilog, the `filelist.f` file should appear similar to the following example:

Example verilog_filelist.f

```
+incdir+<project_dir>/src/tb
+incdir+<project_dir>/src/rtl
+incdir+<ACE_INSTALL_DIR>/libraries
+incdir+<ace_ext_dir>/libraries
<ace_install>/libraries/device_models/<DEVICE>_simmodels.sv
<project_dir>/src/rtl/lfsr_updown_cnt.v
<project_dir>/src/tb/lfsr_updown_cnt_tb.v
+libext+.v
```

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Step 4 - Compile the Design

Before the design can be simulated, it must be compiled. If using VHDL, use the following command to compile the design:

```
vcom -work <project_dir>/src/questasim-rtl/work -f <project_dir>/src/questasim-rtl/
vhdl_filelist.f
```

If using Verilog, use the following command to compile the design:

```
vlog -sv -work <project_dir>/src/questasim-rtl/work -mfcu -f <project_dir>/src/
questasim-rtl/verilog_filelist.f
```

Table 9 • Important Command-Line Options

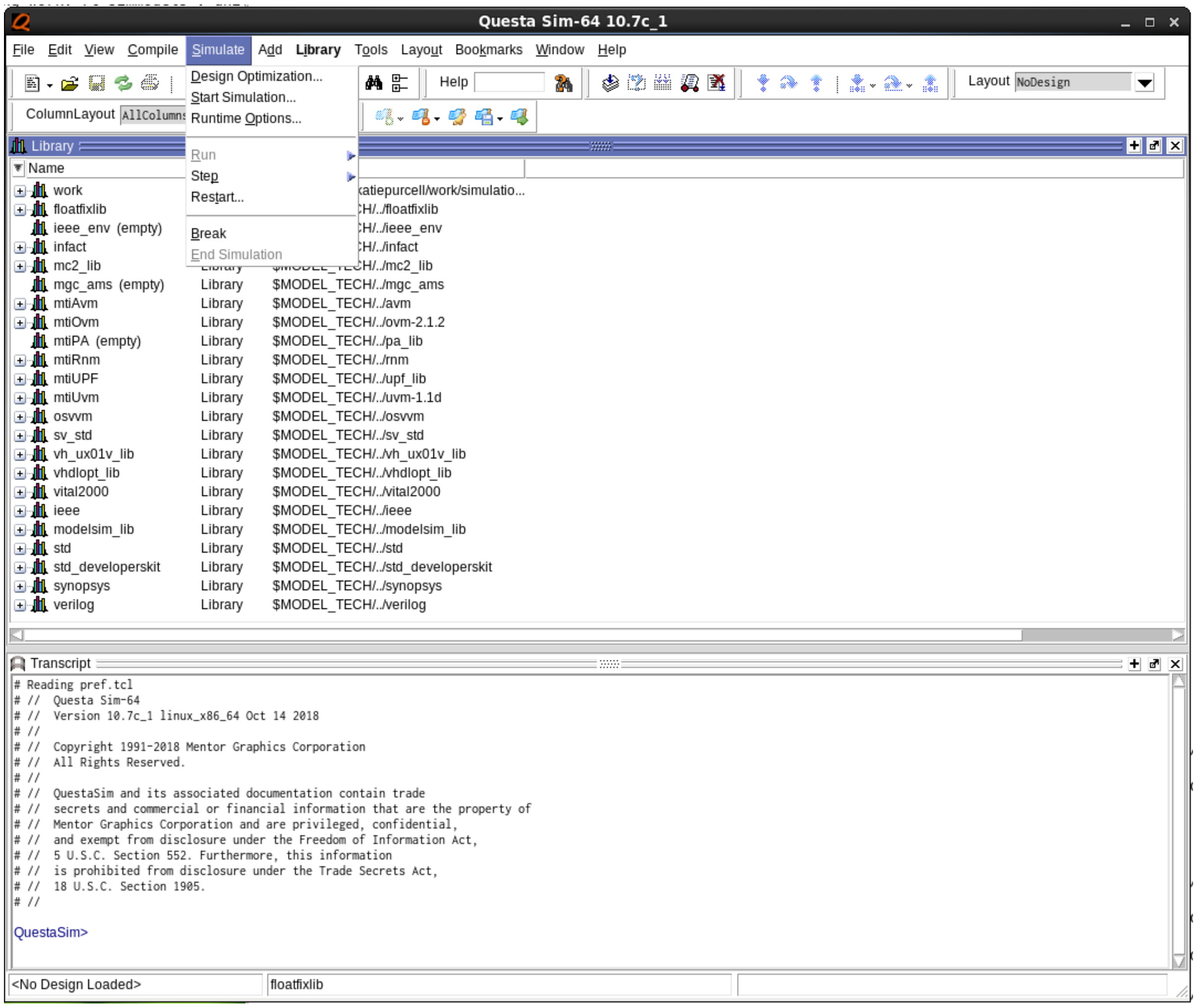
Option	Description
-mfcu	Multi-file compilation unit, all files in command line make up a compilation unit.
-sv	Enable SystemVerilog features and keywords

Step 5 – Prepare the Simulation Run

Open the simulator and load the compiled design using the following command:

```
vsim -lib <project_dir>/src/questasim-rtl/work -gui
```

In the simulator GUI, select **Simulate** → **Start Simulation...** to prepare the simulation:



Name		
work		katiepurcell/work/simulatio...
floatfixlib		H/../floatfixlib
ieee_env (empty)		H/../ieee_env
infact		H/../infact
mc2_lib	Library	$MODEL_TECH/../mc2_lib
mgc_ams (empty)	Library	$MODEL_TECH/../mgc_ams
mtiAvm	Library	$MODEL_TECH/../avm
mtiOvm	Library	$MODEL_TECH/../ovm-2.1.2
mtiPA (empty)	Library	$MODEL_TECH/../pa_lib
mtiRnm	Library	$MODEL_TECH/../rnm
mtiUPF	Library	$MODEL_TECH/../upf_lib
mtiUvm	Library	$MODEL_TECH/../uvm-1.1d
osvvm	Library	$MODEL_TECH/../osvvm
sv_std	Library	$MODEL_TECH/../sv_std
vh_ux01v_lib	Library	$MODEL_TECH/../vh_ux01v_lib
vhdlopt_lib	Library	$MODEL_TECH/../vhdlopt_lib
vital2000	Library	$MODEL_TECH/../vital2000
ieee	Library	$MODEL_TECH/../ieee
modelsim_lib	Library	$MODEL_TECH/../modelsim_lib
std	Library	$MODEL_TECH/../std
std_developerskit	Library	$MODEL_TECH/../std_developerskit
synopsys	Library	$MODEL_TECH/../synopsys
verilog	Library	$MODEL_TECH/../verilog

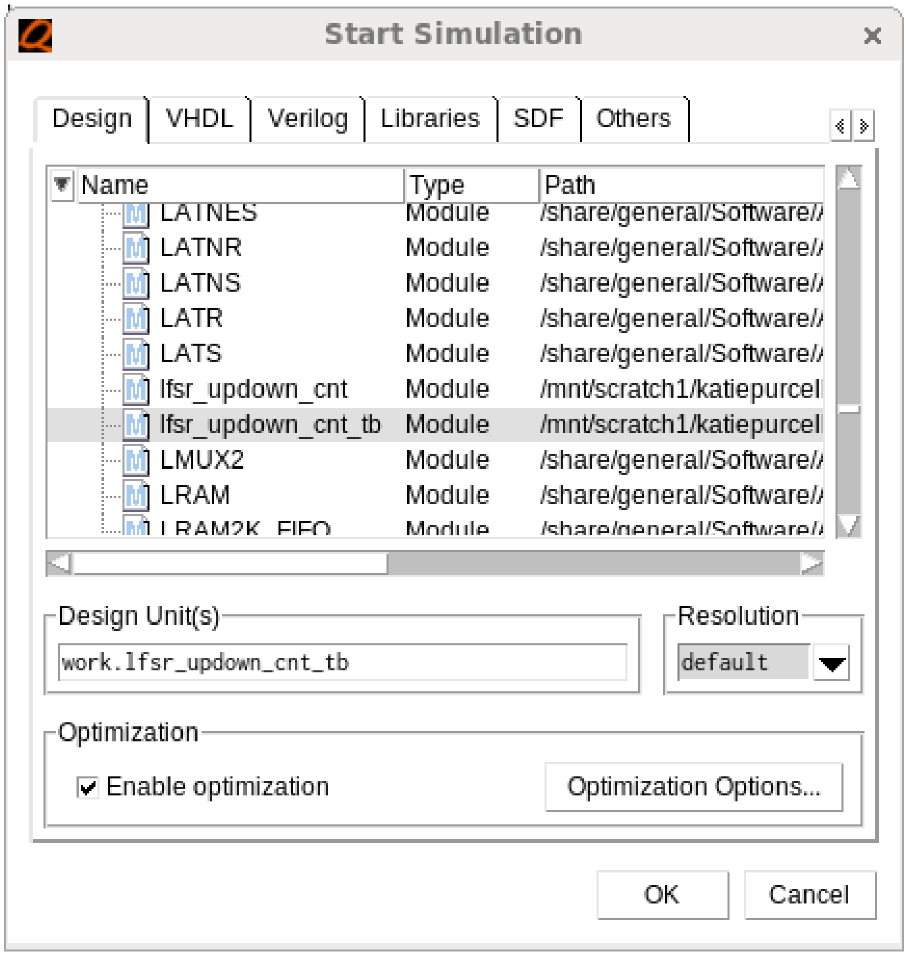
```
# Reading pref.tcl
# //  Questa Sim-64
# //  Version 10.7c_1 linux_x86_64 Oct 14 2018
# //
# //  Copyright 1991-2018 Mentor Graphics Corporation
# //  All Rights Reserved.
# //
# //  QuestaSim and its associated documentation contain trade
# //  secrets and commercial or financial information that are the property of
# //  Mentor Graphics Corporation and are privileged, confidential,
# //  and exempt from disclosure under the Freedom of Information Act,
# //  5 U.S.C. Section 552. Furthermore, this information
# //  is prohibited from disclosure under the Trade Secrets Act,
# //  18 U.S.C. Section 1905.
# //

QuestaSim>
```

<No Design Loaded>   floatfixlib

Figure 25 • QuestaSim GUI

Step 6 – Set up the Waveform

In the Start Simulation dialog, select the `lfsr_updown_cnt_tb.v` testbench file to run. Click on **Optimization Options...** and choose the option for "Apply full visibility to all modules(full debug mode)" and click **OK**:



Name	Type	Path
LATNES	Module	/share/general/Software/
LATNR	Module	/share/general/Software/
LATNS	Module	/share/general/Software/
LATR	Module	/share/general/Software/
LATS	Module	/share/general/Software/
lfsr_updown_cnt	Module	/mnt/scratch1/katiepurcel
lfsr_updown_cnt_tb	Module	/mnt/scratch1/katiepurcel
LMUX2	Module	/share/general/Software/
LRAM	Module	/share/general/Software/
LRAM2K_FIFO	Module	/share/general/Software/

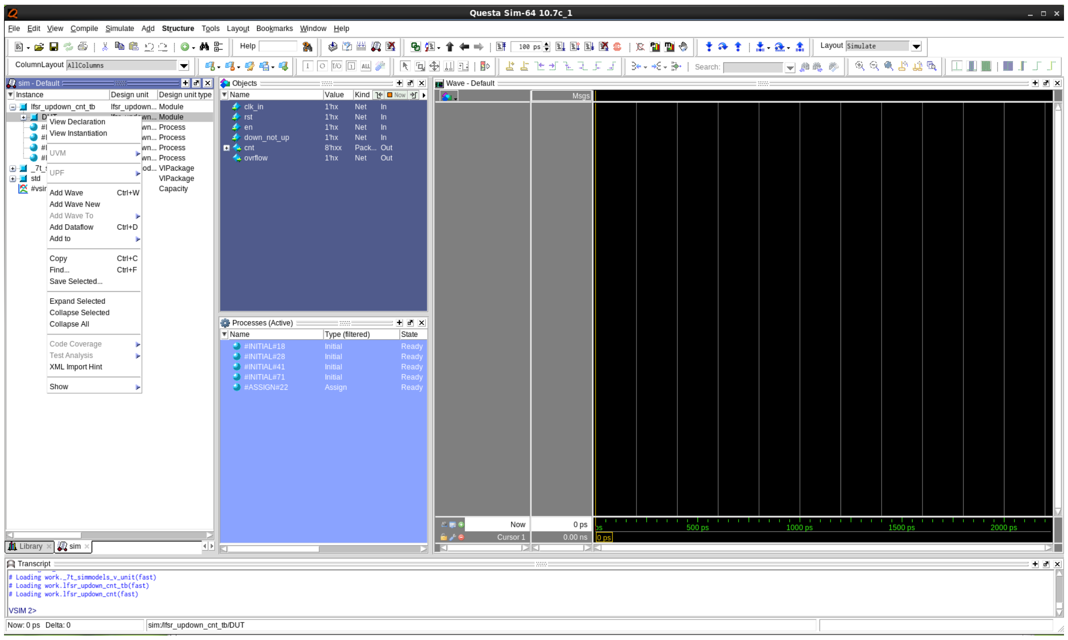
Design Unit(s): work.lfsr_updown_cnt_tb

Resolution: default

[✓] Enable optimization

Figure 26 • Start Simulation Dialog Box

Select the signals to monitor and add them to the waveform by selecting DUT, selecting the desired signals, and then right-clicking and selecting **Add Wave**.



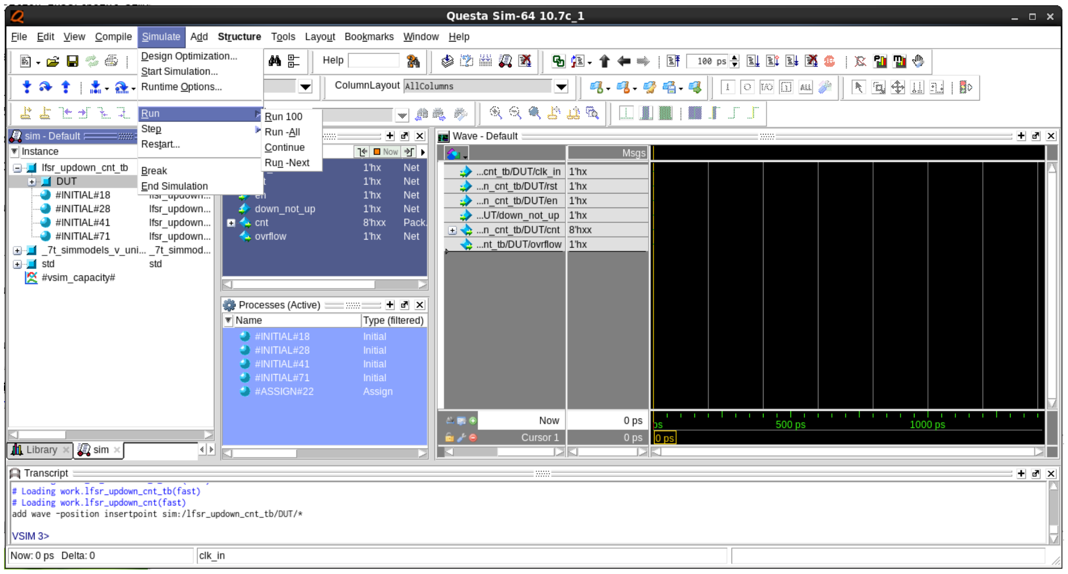
```
# Loading work._7t_simmodels_v_unit(fast)
# Loading work.lfsr_updown_cnt_tb(fast)
# Loading work.lfsr_updown_cnt(fast)

VSIM 2>
```

Now: 0 ps Delta: 0 sim:/lfsr_updown_cnt_tb/DUT

Figure 27 • Selecting Signals to Observe

Step 7 – Run the Simulation

From the simulator GUI, select **Simulate** → **Run** → **Run -All** to start the simulation:



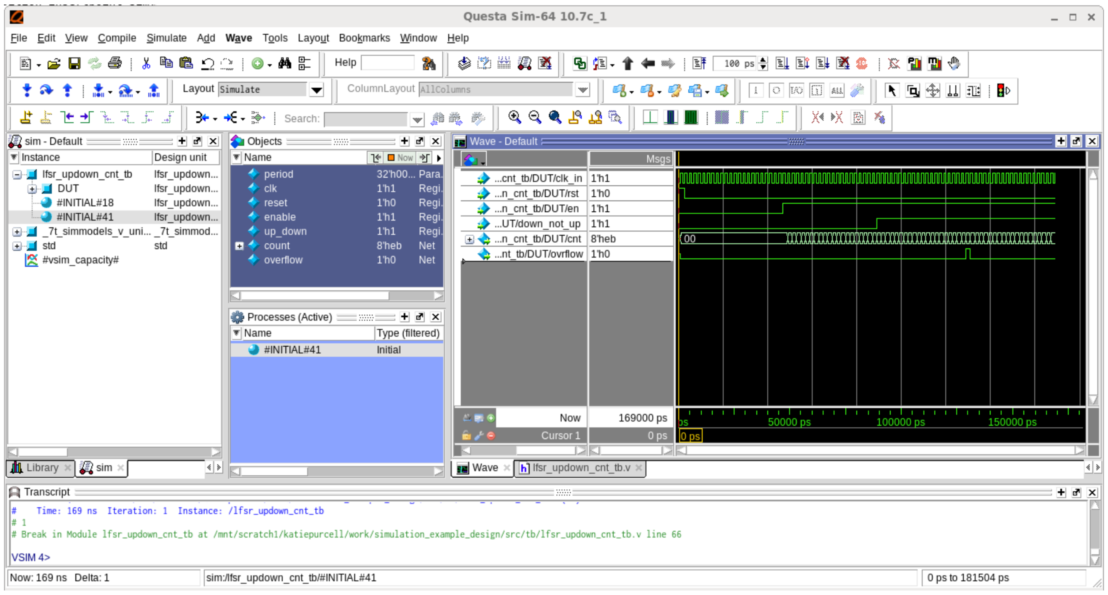
```
# Loading work.lfsr_updown_cnt_tb(fast)
# Loading work.lfsr_updown_cnt(fast)
add wave -position insertpoint sim:/lfsr_updown_cnt_tb/DUT/*

VSIM 3>
```

Now: 0 ps Delta: 0 clk_in

Figure 28 • Starting the Simulation Run

Step 8 – View the Waveform

Simulation results are written to the waveform viewer for review:



Instance	Design unit
lfsr_updown_cnt_tb	lfsr_updown...
DUT	lfsr_updown...
#INITIAL#18	lfsr_updown...
#INITIAL#41	lfsr_updown...
_7t_simmodels_v_uni...	_7t_simmod...
std	std
#vsim_capacity#	

Name	Now	
period	32'h00...	Para.
clk	1'h1	Regi.
reset	1'h0	Regi.
enable	1'h1	Regi.
up_down	1'h1	Regi.
count	8'heb	Net
overflow	1'h0	Net

Name	Type (filtered)
#INITIAL#41	Initial

	Msgs
...cnt_tb/DUT/clk_in	1'h1
...n_cnt_tb/DUT/rst	1'h0
...n_cnt_tb/DUT/en	1'h1
...UT/down_not_up	1'h1
...n_cnt_tb/DUT/cnt	8'heb
...nt_tb/DUT/ovrflow	1'h0
Now	169000 ps
Cursor 1	0 ps

```
#    Time: 169 ns  Iteration: 1  Instance: /lfsr_updown_cnt_tb
# 1
# Break in Module lfsr_updown_cnt_tb at /mnt/scratch1/katiepurcell/work/simulation_example_design/src/tb/lfsr_updown_cnt_tb.v line 66

VSIM 4>
```

Now: 169 ns Delta: 1 | sim:/lfsr_updown_cnt_tb/#INITIAL#41 | 0 ps to 181504 ps

Figure 29 • QuestaSim Waveform Viewer

Gate-Level Simulation in QuestaSim

For gate-level simulation, a synthesized netlist has to first be generated using Synplify Pro before performing the simulation.

Step 1 – Create the Synthesis Project

Create a new project in Synplify under `<project_dir>/src/syn`, include <ACE_INSTALL_DIR>`/libraries/device_models/<DEVICE>_synplify.sv` followed by the RTL design files and constraint files.

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Step 2 – Synthesize the Design

Synthesize the design using Synplify Pro. Synplify Pro generates a gate-level netlist with `.vm` extension, for the example design, the file `<project_dir>/src/syn/rev_acx/lfsr_updown_cnt.vm` is generated.

Step 3 – Set up the Simulation Project

Create a QuestaSim gate-level simulation project directory under `<project_dir>/src/questasim-gate`, then enter this directory to make it the current working directory to launch the simulator from.

Step 4 – Initialize the Work Library

Initialize the simulator work library using the following QuestaSim command:

```
vlib <project_dir>/src/questasim-gate/work
```

Step 5 – Create the File List

Create a `filelist.f` file to configure the project defines, include directories, and source files:

Example verilog_filelist.f

```
+incdir+<project_dir>/src/tb
+incdir+<project_dir>/src/rtl
+incdir+<ACE_INSTALL_DIR>/libraries
+incdir+<ace_ext_dir>/libraries
<ace_install>/libraries/device_models/<DEVICE>_simmodels.sv
<project_dir>/src/syn/rev_acx/lfsr_updown_cnt.vm
<project_dir>/src/tb/lfsr_updown_cnt_tb.v
+libext+.v
```

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Step 6 – Compile the Design

Compile the design for simulation using the following command:

```
vlog -sv -work <project_dir>/src/questasim-gate/work -mfcu -f <project_dir>/src/
questasim-gate/verilog_filelist.f
```

Step 7 – Prepare the Simulation Run

Open the simulator and load the compiled design using the following command:

```
vsim -lib <project_dir>/src/questasim-gate/work -gui
```

Complete the process by following Steps 6 to the end of RTL Simulation in QuestaSim (page 44) above.

Post-Route Simulation in QuestaSim

For post-route simulation, the synthesized gate-level netlist must first be run through place and route using ACE.

Step 1 – Create the ACE Project

Create a new project in ACE under `<project_dir>/src/ace`. Add the gate-level netlist `lfsr_updown_cnt.vm` generated by Synplify Pro plus the constraint files.

Step 2 – Run Place and Route

Run the place and route flow, including the 'Generate Final Simulation Netlist' step to obtain a post-route netlist. In this example, the netlist is generated under `<project_dir>/src/ace/impl_1/output/lfsr_updown_cnt_final.v`.

Step 3 – Set up the Simulation Project

Create the QuestaSim post-route simulation project directory under `<project_dir>/src/questasim-final`, then enter this directory to make it the current working directory to launch the simulator from.

Step 4 – Initialize the Work Library

Initialize the simulator work library using the following QuestaSim command:

```
vlib <project_dir>/src/questasim-final/work
```

Step 5 – Create the File List

Create a `filelist.f` file to configure the project defines, include directories, and source files:

Example verilog_filelist.f

```
+incdir+<project_dir>/src/tb
+incdir+<project_dir>/src/rtl
+incdir+<ACE_INSTALL_DIR>/libraries
+incdir+<ace_ext_dir>/libraries
<ace_install>/libraries/device_models/<DEVICE>_simmodels.sv
<project_dir>/src/ace/impl_1/output/lfsr_updown_cnt_final.v
<project_dir>/src/tb/lfsr_updown_cnt_tb.v
+libext+.v
```

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Step 6 – Compile the Design

Compile the design for simulation using the following command:

```
vlog -sv -work <project_dir>/src/questasim-gate/work -mfcu -f <project_dir>/src/
questasim-final/verilog_filelist.f
```

Step 7 – Prepare the Simulation Run

Open the simulator and load the compiled design using the following command:

```
vsim -lib <project_dir>/src/questasim-final/work -gui
```

Complete the process by following Steps 6 to the end of RTL Simulation in QuestaSim (page 44) above.

Note

In post-route simulations, the user design netlist output from ACE is encrypted. Therefore, only signals from the testbench are observable. The post-route simulation results should functionally match exactly with the gate-level simulation results. However, if an issue is seen in post-route simulation, run a gate-level simulation to debug the issue.

Synopsys VCS Simulator Example

RTL Simulation in VCS

In order to run the RTL simulation using the VCS tool, the following steps have to be followed:

Step 1 – Run the VCS Simulator

Run the simulator using the **vcs** command, including the libraries path, path to `<DEVICE>_simmodels.v`, top-level RTL file and testbench as shown below:

```
vcs +vcs+lic+wait -lca -debug_pp +incdir+<ACE_INSTALL_DIR>/libraries/ <ACE_INSTALL_DIR>/
libraries/device_models/<DEVICE>_simmodels.sv <project_dir>/src/rtl/lfsr_updown_cnt.v
<project_dir>/src/tb/lfsr_updown_cnt_tb.v -sverilog -R -l vcs.log
```

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Table 10 • VCS Command Options

Options	Required	Description
-lca	Yes	IEEE encryption flow in VCS requires this option.
-debug_pp	No	Creates a VPD file (when used with the VCS system task $vcdpluson) and enables DVE for post-processing a design. Using-debug-pp can save compilation time by eliminating the overhead of compiling with -debug and -debug_all.
-sverilog	Yes	Supports SystemVerilog features.

Options	Required	Description
-R	No	Run the executable file immediately after VCS links together the executable file. This option is required so that .vpd file is generated which is used for analyzing waveform.
-l	No	Log file name can be specified with this option where VCS records compilation messages. Runtime messages are also included in the log file if -R option is used along with -l.
+vcs+lic+wait	No	Tells VCS to wait for network license if none is available.

Step 2 – Start the Simulation GUI

After successful completion of compilation and simulation above, a GUI called Discovery Verification Environment (DVE) can be used to post-process a design. In this example `lfsr_updown_cnt_tb_sim.vpd` is generated after completion of Step 1. DVE can be opened using the following command:

```
dve &
```

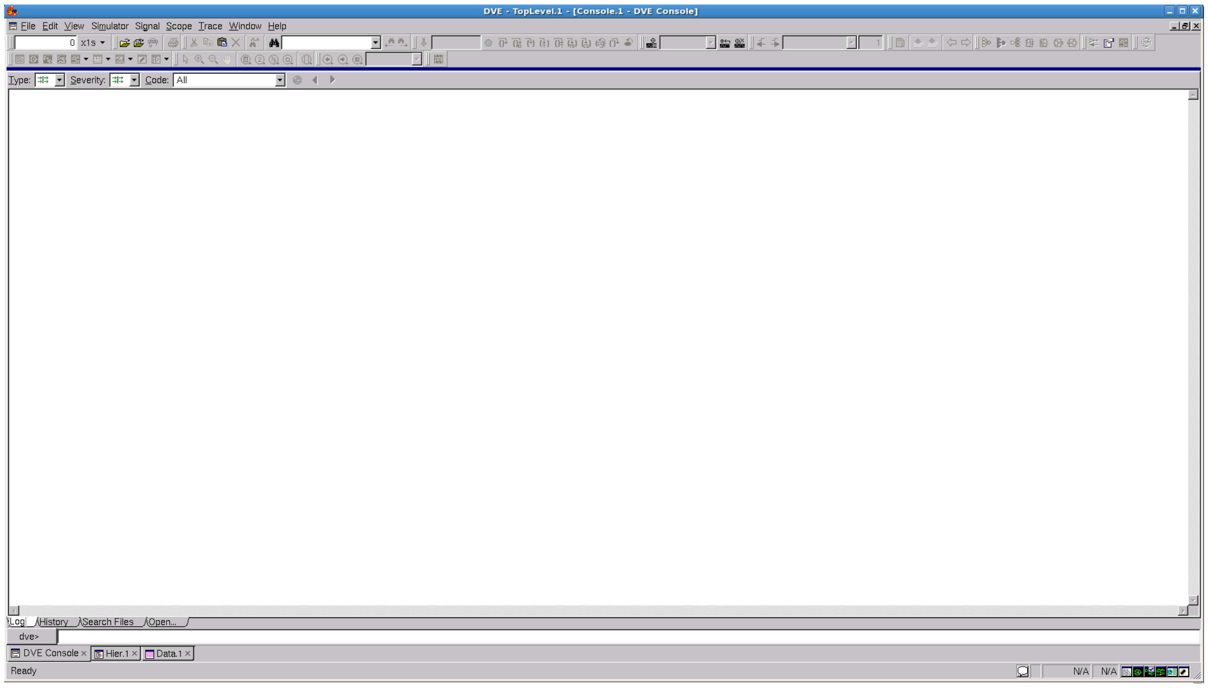
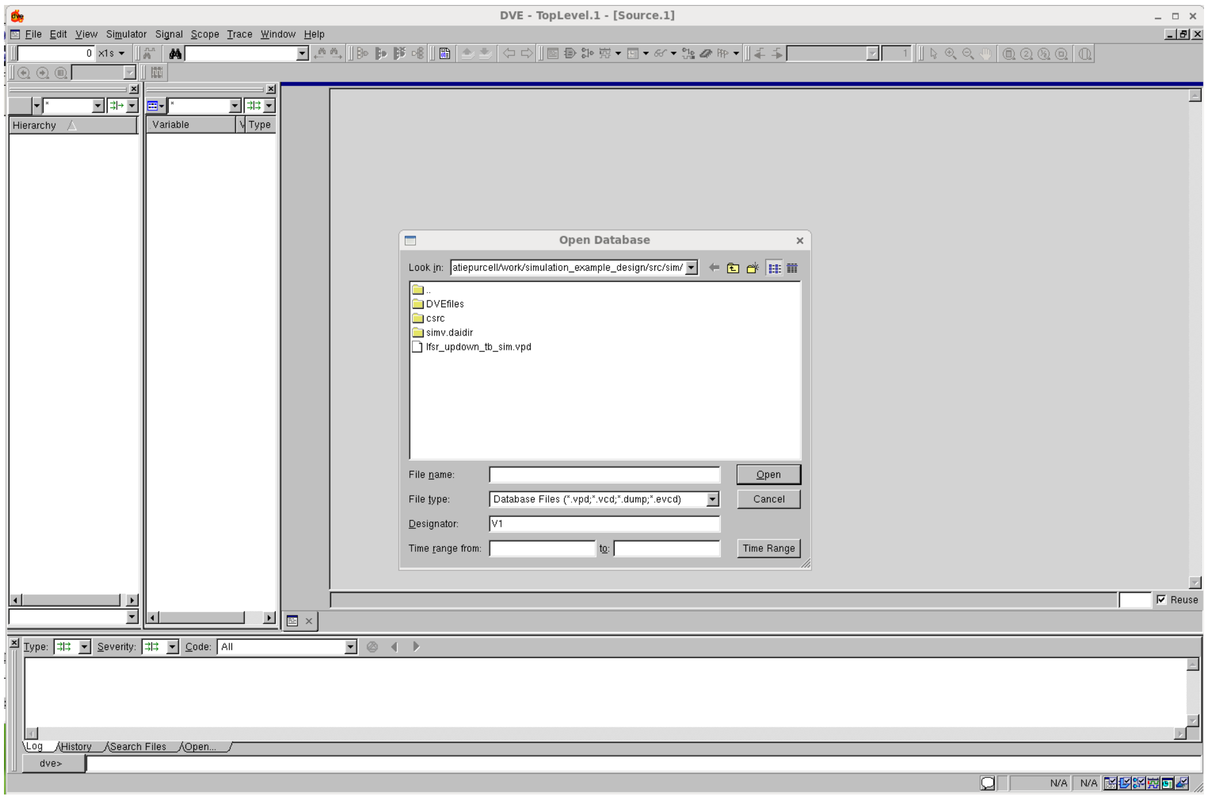

DVE - TopLevel.1 - [Console.1 - DVE Console]
File Edit View Simulator Signal Scope Trace Window Help
Type: Severity: Code: All
Log History Search Files Open...
dve>
DVE Console Hier.1 Data.1
Ready
N/A N/A

Figure 30 • DVE Main Window

Step 3 – Open the Simulation Database

Open the database file
`lfsr_updown_cnt_tb_sim.vpd` file:



Look in: atiepurcell/work/simulation_example_design/src/sim/

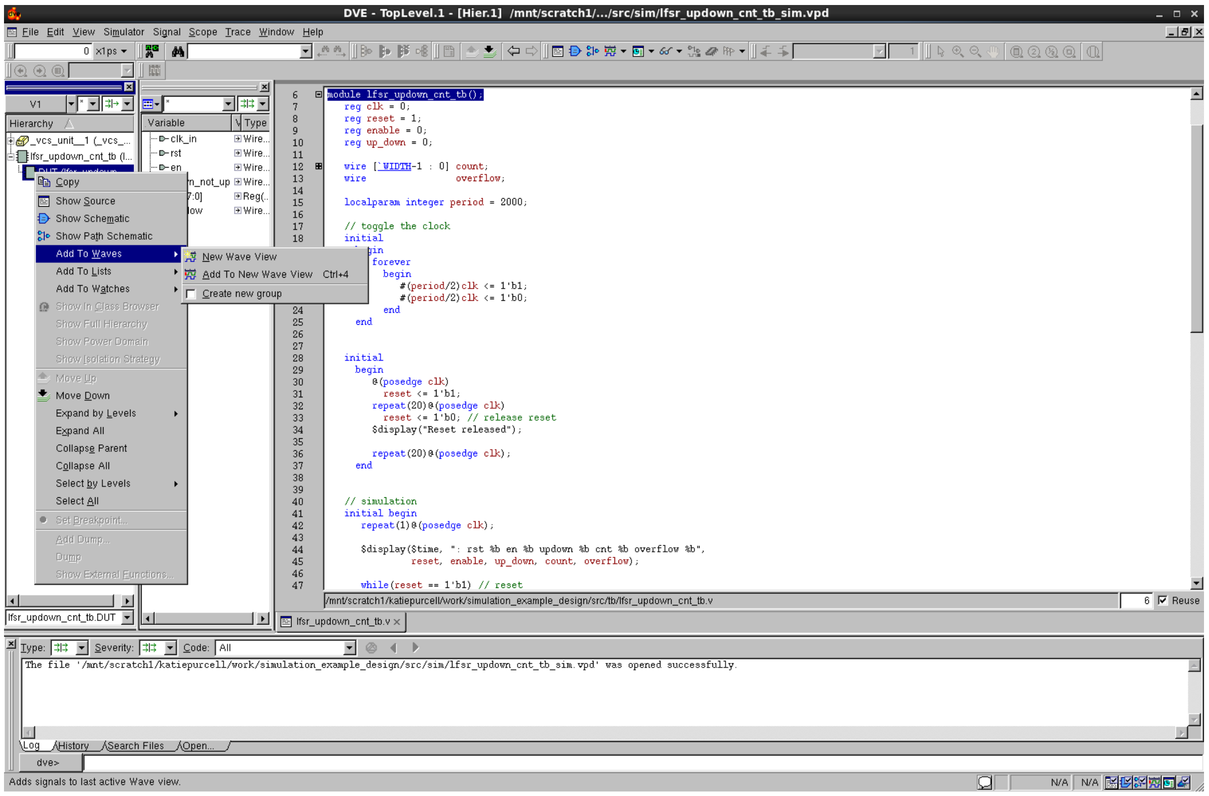
..
DVEfiles
csrc
simv.daidir
lfsr_updown_tb_sim.vpd

File name:
File type: Database Files (*.vpd;*.vcd;*.dump;*.evcd)
Designator: V1
Time range from: to:
Code: All

Figure 31 • Opening the Simulation Database

Step 4 – Add Signals to the Waveform

Select the required signals from the DUT and add them to the waveform by right-clicking and selecting **Add to Waves → New Wave View**:



```
module lfsr_updown_cnt_tb();
   reg clk = 0;
   reg reset = 1;
   reg enable = 0;
   reg up_down = 0;

   wire [`WIDTH-1 : 0] count;
   wire                overflow;

   localparam integer period = 2000;

   // toggle the clock
   initial
     gin
       forever
         begin
            #(period/2)clk <= 1'b1;
            #(period/2)clk <= 1'b0;
         end
     end


   initial
     begin
       @(posedge clk)
         reset <= 1'b1;
       repeat(20)@(posedge clk)
         reset <= 1'b0; // release reset
       $display("Reset released");

       repeat(20)@(posedge clk);
     end


   // simulation
   initial begin
      repeat(1)@(posedge clk);

      $display($time, ": rst %b en %b updown %b cnt %b overflow %b",
               reset, enable, up_down, count, overflow);

      while(reset == 1'b1) // reset
```

```
/mnt/scratch1/katiepurcell/work/simulation_example_design/src/tb/lfsr_updown_cnt_tb.v
```

```
The file '/mnt/scratch1/katiepurcell/work/simulation_example_design/src/sim/lfsr_updown_cnt_tb_sim.vpd' was opened successfully.
```

```
Adds signals to last active Wave view.
```

Figure 32 • Adding Signals to the Waveform

Step 5 – View the Simulation Results

Waveforms can be viewed and analyzed as shown below:

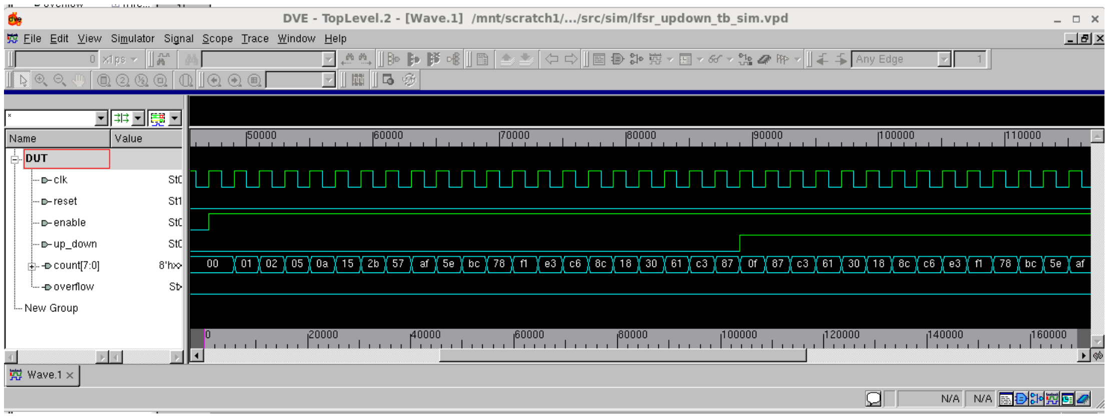

DVE - TopLevel.2 - [Wave.1] /mnt/scratch1/.../src/sim/lfsr_updown_tb_sim.vpd
File Edit View Simulator Signal Scope Trace Window Help
Any Edge
Name
Value
DUT
clk St0
reset St1
enable St0
up_down St0
count[7:0] 8'hxx
overflow St
New Group
50000
60000
70000
80000
90000
100000
110000
00 01 02 05 0a 15 2b 57 af 5e bc 78 f1 e3 c6 8c 18 30 61 c3 87 0f 87 c3 61 30 18 8c c6 e3 f1 78 bc 5e af
0
20000
40000
60000
80000
100000
120000
140000
160000
Wave.1
N/A
N/A

Figure 33 • Viewing the Waveforms

Gate-Level Simulation in VCS

For gate-level simulation, a synthesized netlist has to first be generated using Synplify Pro before performing the simulation.

Step 1 – Create the Synthesis Project

Create a new project in Synplify under `<project_dir>/src/syn`, include <ACE_INSTALL_DIR>`/libraries/device_models/<DEVICE>_synplify.sv` followed by the RTL design files and constraint files.

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Step 2 – Synthesize the Design

Synthesize the design using Synplify Pro. Synplify Pro generates a gate-level netlist with `.vm` extension, for the example design, the file `<project_dir>/src/syn/rev_acx/lfsr_updown_cnt.vm` is generated.

Step 3 – Run the VCS Simulator

To run the gate-level simulation, use the same command as described in **RTL Simulation in VCS** (page 52) above except that the gate-level simulation uses the mapped netlist `lfsr_updown_cnt.vm` instead of source RTL files.

```
vcs +vcs+lic+wait -lca -debug_pp +incdir+<ACE_INSTALL_DIR>/libraries/ <ACE_INSTALL_DIR>/
libraries/device_models/<DEVICE>_simmodels.sv <project_dir>/src/syn/rev_2/
lfsr_updown_cnt.vm <project_dir>/src/tb/lfsr_updown_cnt_tb.v -sverilog -R -l vcs.log
```

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Complete the process by following Steps 2 to the end of **RTL Simulation in VCS** (page 52).

Post-Route Simulation in VCS

For post-route simulation, the synthesized gate-level netlist must first be run through place and route using ACE.

Step 1 – Create the ACE Project

Create a new project in ACE under `<project_dir>/src/ace`. Add the gate-level netlist `lfsr_updown_cnt.vm` generated by Synplify Pro plus the constraint files.

Step 2 – Run Place and Route

Run the place and route flow, including the 'Generate Final Simulation Netlist' step to obtain a post-route netlist. In this example, the netlist is generated under `<project_dir>/src/ace/impl_1/output/lfsr_updown_cnt_final.v`.

Step 3 – Run the VCS Simulator

Run the post-route simulation using the same command as used in RTL and gate-level simulation using the final netlist:

```
vcs +vcs+lic+wait -lca -debug_pp +incdir+<ACE_INSTALL_DIR>/libraries/ <ACE_INSTALL_DIR>/
libraries/device_models/<DEVICE>_simmodels.sv <project_dir>/src/ace/impl_1/output/
lfsr_updown_cnt_final.v <project_dir>/src/tb/lfsr_updown_cnt_tb.v -sverilog -R -l
vcs.log
```

Where `<DEVICE>` represents the name of the target device, such as `AC7t1500`.

Complete the process by following Steps 2 to the end of RTL Simulation in VCS (page 52).

Note

In post-route simulations, the user design netlist output from ACE is encrypted. Therefore, only signals from the testbench are observable. The post-route simulation results should functionally match exactly with the gate-level simulation results. However, if an issue is seen in post-route simulation, run a gate-level simulation to debug the issue.

Chapter 4 : DSM Simulation Package

Device Simulation Model

Many designs require a simulation overlay named the device simulation model (DSM). This package combines the full RTL of the 2D network on chip (NoC) with bus functional models (BFMs) of the interface subsystems that surround the NoC and FPGA fabric. This combination of true RTL for the 2D NoC and models for the interface subsystems allows developing designs within a fast responsive simulation environment, while achieving cycle-accurate interfaces from the NoC, and representative cycle responses from the hard interface subsystems. This simulation environment allows a designer to iterate rapidly to develop and debug their design.

Description

The DSM provides full RTL code for the NoC, combined with BFMs of the surrounding interface subsystems. The structure is wrapped within a SystemVerilog module named per device (i.e., `ac7t1500`). Instantiate one instance of this module within the top-level testbench.

In addition, the DSM provides binding macros such that binding between elements of a design and the same elements within the device is possible. For example, the design might instantiate a 2D NoC access point (NAP). It is then necessary to bind this NAP instance to the NAP in the correct location within the 2D NoC by using the `` `ACX_BIND_NAP_RESPONDER``, `` `ACX_BIND_NAP_INITIATOR``, `` `ACX_BIND_NAP_HORIZONTAL``, `` `ACX_BIND_NAP_VERTICAL`` or `` `ACX_BIND_NAP_ETHERNET`` macro, whichever is appropriate for the design.

Similarly, it is necessary to bind between the ports on the design and the direct-connection interface (DCI) for the interface subsystem. Each DCI within the device is connected to a SystemVerilog interface. This interface can then be directly accessed from the top-level testbench, and signals assigned between the SystemVerilog interface and the ports on the design.

Selecting the Required DSM

DSM Utility Package

There is a DSM package for each device, with each DSM representing the specific features of that device. It is therefore necessary to select the correct DSM within a simulation testbench. Selection of the correct DSM is achieved by including the appropriate DSM utility package. The package then creates macros and functions to access the appropriate DSM. The utility package defines the macro `ACX_DEVICE_NAME`, which is then used to instantiate and refer to the DSM. The following DSM utility packages are available.

Table 11 • DSM Utility Packages

Devices	DSM Utility Package	ACX_DEVICE_NAME
AC7t1500, AC7t1500ES0	`ac7t1500_utils.svh`	ac7t1500

Devices	DSM Utility Package	ACX_DEVICE_NAME
AC7t1400, AC7t1400ES0	`ac7t1400_utils.svh`	ac7t1400
AC7t800, AC7t800ES0	`ac7t800_utils.svh`	ac7t800

Device-Specific Simulation Files

To allow for reusable code, the Achronix simulation flow creates a macro for each device, of the form `ACX_DEVICE_<full device name>`. The appropriate macro is present in simulation (and synthesis) when the appropriate ACE library file is included in the project. These ACE library files are located within the `<ACE_INSTALL_DIR>/libraries/device_models/<full device name>_simmodels.sv` file. The following table lists the available `simmodels.sv` files, and the device specific macro that each creates.

Table 12 • Simulation Model Files and Defines

Device	Simulation Model File	ACX_DEVICE Macro
AC7t1500	`AC7t1500_simmodels.sv`	ACX_DEVICE_AC7t1500
AC7t1500ES0	`AC7t1500ES0_simmodels.sv`	ACX_DEVICE_AC7t1500ES0
AC7t1400	`AC7t1400_simmodels.sv`	ACX_DEVICE_AC7t1400
AC7t1400ES0	`AC7t1400ES0_simmodels.sv`	ACX_DEVICE_AC7t1400ES0
AC7t800	`AC7t800_simmodels.sv`	ACX_DEVICE_AC7t800
AC7t800ES0	`AC7t800ES0_simmodels.sv`	ACX_DEVICE_AC7t800ES0

Instantiate DSM Utility Package

Using the device specific macros, it is possible to create a general DSM instantiation that can be used for multiple devices. In the following example, the ACX_DEVICE_xxxx macro is used to select the appropriate DSM utility package. The macros subsequently created by the package are then used to select the appropriate DSM.

```
    // Include the appropriate DSM utility file which defines the appropriate macros
    // If an unsupported device is selected, then compilation will fail
`ifdef ACX_DEVICE_AC7t1500ES0
    `include "ac7t1500_utils.svh"
`elsif ACX_DEVICE_AC7t1500
    `include "ac7t1500_utils.svh"
`elsif ACX_DEVICE_AC7t800ES0
    `include "ac7t800_utils.svh"
```

```
`endif

    // Instantiate the DSM
    // ACX_DEVICE_NAME is defined in the DSM utility file for the selected device
    // Connect the chip_ready signal
    `ACX_DEVICE_NAME `ACX_DEVICE_NAME (
        .FCU_CONFIG_USER_MODE   (chip_ready),
    );
```

Version Control

The DSM is version controlled. Within a release, new functions might be added and older functions might be deprecated or replaced. The release is indicated both in the package name (`ACE_<major>.<minor>.<patch>_DSM_sim_<update>.zip/tgz`) and in the `readme` file placed in the root directory of the package.

To ensure that the correct version of the DSM is used, a task must be included within the design testbench to confirm the version compatibility. This function should be instantiated as follows:

```
    // The ACX_DEVICE_NAME macro is defined for each DSM within its appropriate utility
package
    initial begin
        // Ensure correct version of DSM is being used
        // This design requires 10.1 as a minimum
        `ACX_DEVICE_NAME.require_version(10, 1, 0, 0);
    end
```

require_version( ) Task

The require_version task has four arguments. In order:

1. Major Version – Matches the major version of the release
2. Minor Version – Matches the minor version of the release
3. Patch – Matches the patch version of the release (optional)
4. Update – Matches the update number of the release (optional)

If either patch or update is not specified, then these arguments should be set to 0. For example, for the 10.1 release, the arguments would be set as 10,1,0,0.

Note

The values can be expressed either as numbers (0-9) or as strings ("0"-"9") or as letters ("a/A", "b/B"), with the letters "a" and "b" representing alpha or beta releases. When deciding on the priority of a release, a number represents a more recent release than a letter; therefore, 8.3.alpha (defined as 8,3,"a",0) precedes the full 8.3 release (designated as 8,3,0,0).

Example Design

An example structure of a user testbench, instantiating both the DSM and the user design under test is shown in the following diagram (see figure 34). This example shows the macros required for the responder NAPs, and the DCIs for two instances of the GDDR6 subsystem. For other forms of NAPs, or for other DCI types, such as DDR, consult the Bind Macros (page 64) and DSM Direct-Connect Interfaces (page 65) tables.

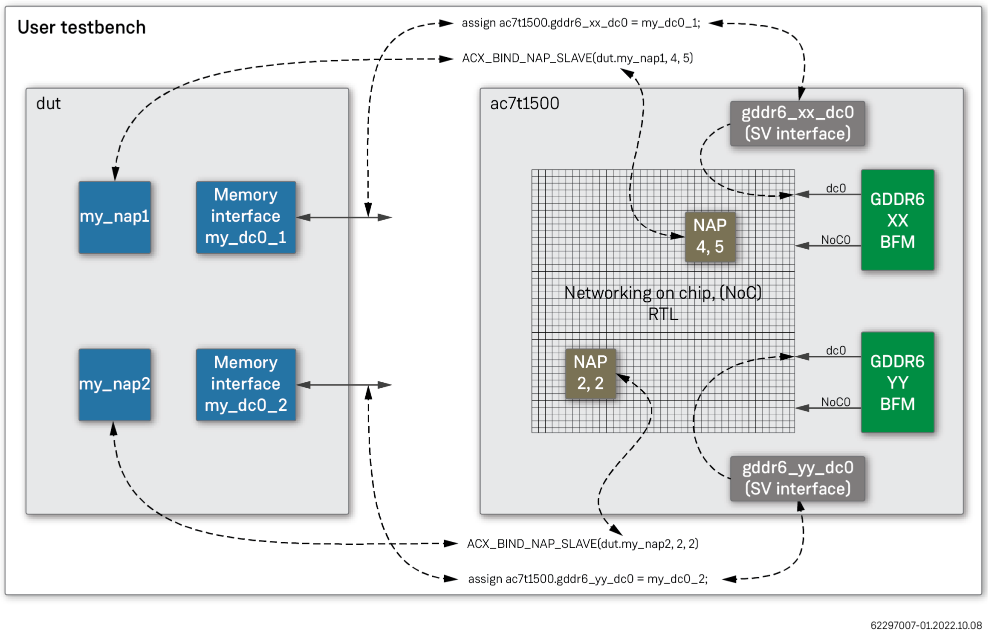

User testbench
assign ac7t1500.gddr6_xx_dc0 = my_dc0_1;
ACX_BIND_NAP_SLAVE(dut.my_nap1, 4, 5)
dut
my_nap1
Memory interface my_dc0_1
my_nap2
Memory interface my_dc0_2
ac7t1500
gddr6_xx_dc0 (SV interface)
dc0
NoC0
GDDR6 XX BFM
NAP 4, 5
Networking on chip, (NoC) RTL
NAP 2, 2
dc0
NoC0
GDDR6 YY BFM
gddr6_yy_dc0 (SV interface)
ACX_BIND_NAP_SLAVE(dut.my_nap2, 2, 2)
assign ac7t1500.gddr6_yy_dc0 = my_dc0_2;
62297007-01.2022.10.08

Figure 34 • Example Simulation Structure

In the previous example, there are two NAPs, `my_nap1` and `my_nap2`. In addition, there are two direct-connect interfaces, `my_dc0_1` and `my_dc0_2`. In the top-level, testbench bindings are made between the NAPs in the design and the NAPs within the device using the ACX_BIND_NAP_RESPONDER macro:

- This macro supports inserting the coordinates of the NAP within the 2D NoC in order that the simulation is aligned with physical placement of the NAP on silicon.
- The DCIs are ports on the user design. These ports are assigned to the appropriate signals within the device direct-connect SystemVerilog interface.

The Verilog code to instantiate the example, based on using the Speedster7t AC7t1500 FPGA, follows.

```
    // ---------------------------------------------
    // Instantiate the DSM
    // ---------------------------------------------
    // Connect the chip ready port
    // Note : All DSM ports are defined, so can be directly connected if required
    `ACX_DEVICE_NAME `ACX_DEVICE_NAME( .FCU_CONFIG_USER_MODE (chip_ready ) );
```

```
// Set the verbosity options on the messages
// Use the inbuilt set_verbosity() task.
initial begin
    `ACX_DEVICE_NAME.set_verbosity(2);
end

// -----------------------------------------------
// Bind NAPs
// -----------------------------------------------
// Bind my_nap1 to location 4,5
`ACX_BIND_NAP_AXI_RESPONDER(dut.my_nap1,4,5);
// Bind my_nap2 to location 2,2
`ACX_BIND_NAP_AXI_RESPONDER(dut.my_nap2,2,2);

// -----------------------------------------------
// Connect to DC interfaces
// -----------------------------------------------
// Create signals to attach to direct-connect interface
logic                         my_dc0_1_clk;
logic                         my_dc0_1_awvalid;
logic                         my_dc0_1_awaddr;
logic                         my_dc0_1_awready;
.....
logic                         my_dc0_2_clk;
logic                         my_dc0_2_awvalid;
logic                         my_dc0_2_awaddr;
logic                         my_dc0_2_awready;
.....

// Connect signals to gddr6_xx_dc0 interface within ac7t1500 device
// Inputs to device
assign `ACX_DEVICE_NAME.gddr6_xx_dc0.awvalid  = my_dc0_1_awvalid;
assign `ACX_DEVICE_NAME.gddr6_xx_dc0.awaddr   = my_dc0_1_awaddr;
....
// Outputs from device
assign my_dc0_1_awready = `ACX_DEVICE_NAME.gddr6_xx_dc0.awready;
....

// Connect signals to gddr6_xx_dc0 interface within ac7t1500 device
// Inputs to device
assign `ACX_DEVICE_NAME.gddr6_yy_dc0.awvalid  = my_dc0_2_awvalid;
assign `ACX_DEVICE_NAME.gddr6_yy_dc0.awaddr   = my_dc0_2_awaddr;
....
// Outputs from device
assign my_dc0_2_awready = `ACX_DEVICE_NAME.gddr6_yy_dc0.awready;
....

// -----------------------------------------------
// Remember to connect the clock!
```

```
    // ---------------------------------------------
    assign my_dc0_1_clk = `ACX_DEVICE_NAME.gddr6_xx_dc0.clk;
    assign my_dc0_2_clk = `ACX_DEVICE_NAME.gddr6_yy_dc0.clk;
```

Note

When using bind macros, the column and row coordinates of the target NAP can be specified. To ensure consistency between simulation and silicon, add matching placement constraints to the ACE placement `.pdc` file, for example:

In simulation

```
`ACX_BIND_NAP_AXI_RESPONDER(dut.my_nap1,4,5);
```

In place and route

```
set_placement -fixed {i:my_nap} {s:x_core.NOC[4][5].logic.noc.nap_s}
```

set_verbosity( ) Task

Alongside specifying the required simulation package version and instantiating the device, the verbosity of the messages that are output from the device simulation model can be controlled. These levels are controlled by the `set_verbosity` task. Refer to the previous code sample for an example showing how to call this function.

The verbosity levels are defined in the following table.

Table 13 • Verbosity Levels

Verbosity Level	Description
0	Print no messages.
1	Print messages from initiator and responder interfaces only.
2	Print messages from level 1 and from each NoC data transfer.
3	Print messages from level 2, port bindings and NoC performance statistics.

Chip Status Output

From initial simulation start, the device operates similarly to its silicon equivalent with an initialization period when the device is in reset. In hardware this occurs during configuration as the bitstream is loaded. After this initialization period, the device asserts the `FCU_CONFIG_USER_MODE` signal to indicate that it has entered user mode, whereby the design starts to operate.

It is suggested that the top-level testbench monitor `FCU_CONFIG_USER_MODE` and delay drive stimulus into the device until this signal is asserted (shown in the previous example by use of a testbench `chip_ready` signal).

Bind Macros

The following bind statements are available.

Table 14 • Bind Macros

Macro	Arguments [1]	Description
`ACX_BIND_NAP_HORIZONTAL`	`user_nap_instance`, `noc_colunm`, `noc_row`	To bind a horizontal streaming NAP, instance `ACX_NAP_HORIZONTAL`.
`ACX_BIND_NAP_VERTICAL`	`user_nap_instance`, `noc_colunm`, `noc_row`	To bind a vertical streaming NAP, instance `ACX_NAP_VERTICAL`.
`ACX_BIND_NAP_AXI_INITIATOR` [2]	`user_nap_instance`, `noc_colunm`, `noc_row`	To bind an AXI initiator NAP, instance `ACX_NAP_AXI_INITIATOR`.
`ACX_BIND_NAP_AXI_RESPONDER` [2]	`user_nap_instance`, `noc_colunm`, `noc_row`	To bind an AXI responder NAP, instance `ACX_NAP_AXI_RESPONDER`.
`ACX_BIND_NAP_ETHERNET`	`user_nap_instance`, `noc_colunm`, `noc_row`	To bind an Ethernet NAP instance, `ACX_NAP_ETHERNET`.

Table Notes

1. `user_nap_instance` is relative to the testbench, not to the top of the simulation. Normally `user_nap_instance` would be of the form `DUT.<hierarchical_path_to_nap>`.
2. For the Speedster7t AC7t800 FPGA, these macros are `ACX_BIND_NAP_AXI_INITIATOR` and `ACX_BIND_NAP_AXI_RESPONDER`.

Direct-Connect Interfaces

Within the device, the non-NAP connections between the high-speed interface subsystems (such as GDDR, DDR, Ethernet and SerDes) and the fabric are known as direct-connect interfaces (DCIs). These are comprised of:

- Additional data ports in the case of the memory interfaces (AXI)
- Dedicated data interfaces for SerDes (direct mode)
- Status and control for Ethernet

For full details of each of the subsystem DCI ports, refer to the appropriate interface subsystem user guide.

Connecting from the user design to the DCI ports involves one of two methods:

- Connecting directly using the interfaces built into the DSM
- Using an ACE-generated port binding file

Note

The Speedster7t AC7t800 FPGA only incorporates DCIs for the SerDes direct mode. All other data flows between interface subsystems and the fabric are made using the NAP and 2D NoC.

Suggested Flows

In general, the direct connection to the DSM ports is used at the commencement of a project, when an ACE project might not yet have been developed. The decision can be made later in the process to use the ACE bindings file. Both methods achieve the same objective —connecting the DUT I/O ports to the appropriate locations within the DSM.

- Direct connect method – makes use of SystemVerilog interfaces. Therefore, it is possible to add additional features such as protocol checking and performance measurements into these interfaces.
- ACE port binding method – assists with confirming consistency of the DUT ports as presented to ACE (from both the netlist and the ACE generated IP files). This flow can be used to help debug any port naming mismatches prior to committing to place and route.

The two methods are detailed as follows.

DSM DC Interfaces

The DSM has a SystemVerilog interface for each DCI port. The available interfaces are listed in the following table.

Table 15 • DSM Direct-Connect Interfaces

Subsystem	Interface Name	Physical Location [1]	GDDR6 Channel	SystemVerilog Interface Type	Data Width	Address Width
GDDR6	`gddr6_1_dc0`	West 1	0	`t_ACX_AXI4`	512	33
GDDR6	`gddr6_1_dc1`	West 1	1	`t_ACX_AXI4`	512	33
GDDR6	`gddr6_2_dc0`	West 2	0	`t_ACX_AXI4`	512	33
GDDR6	`gddr6_2_dc1`	West 2	1	`t_ACX_AXI4`	512	33
GDDR6	`gddr6_5_dc0`	East 1	0	`t_ACX_AXI4`	512	33
GDDR6	`gddr6_5_dc1`	East 1	1	`t_ACX_AXI4`	512	33
GDDR6	`gddr6_6_dc0`	East 2	0	`t_ACX_AXI4`	512	33
GDDR6	`gddr6_6_dc1`	East 2	1	`t_ACX_AXI4`	512	33
DDR4	`ddr4_dc0`	South	–	`t_ACX_AXI4`	512	40
Ethernet	`ethernet_0_dc`	North West	–	`t_ACX_ETHERNET_DCI`	–	–
Ethernet	`ethernet_1_dc`	North East	–	`t_ACX_ETHERNET_DCI`	–	–

Subsystem	Interface Name	Physical Location [1]	GDDR6 Channel	SystemVerilog Interface Type	Data Width	Address Width
Serdes	`serdes_eth0_q0_dc`	North West	–	`t_ACX_SERDES_DCI`	128	–
Serdes	`serdes_eth0_q1_dc`	North West	–	`t_ACX_SERDES_DCI`	128	–
Serdes	`serdes_eth1_q0_dc`	North East[2]	–	`t_ACX_SERDES_DCI`	128	–
Serdes	`serdes_eth1_q1_dc`	North East[2]	–	`t_ACX_SERDES_DCI`	128	–

Table Notes

1. Physical orientation west-to-east is with regards to viewing the die in the floorplan view within ACE. The die is actually rotated about its vertical axis when packaged. Therefore, an interface shown on the floorplan, and listed in this table, as being on the west is physically on the east side of the device when located on the PCB. The north-to-south orientation is not affected and matches with this table, the ACE view, and the device on board.
2. Present on the Speedster7t AC7t800 DSM.

Note

Not all interfaces are available in all devices. Please consult the appropriate device datasheet to understand which interfaces are present in the selected device.

Direct Connect to DSM Interfaces

To connect to any of these interfaces, create a signal in the testbench, and connect it as a port on the DUT. Also, connect the signal to the DSM, using the DSM instance name, the interface name from the DSM Direct-Connect Interfaces (page 65) table, and the element name.

The following example shows how to connect the `awready` and `awvalid` signals for a GDDR AXI interface.

```
// Declare the signals in the testbench
// Note : In order to switch between port binding file and direct connect easily, the
signal
//        names must match the DUT IO port names.
logic   dut_awready;
logic   dut_awvalid;

// Connect to the DSM GDDR_1, DC port 0.
// awready is an output from the DSM, and an input to the DUT
assign dut_awready = `ACX_DEVICE_NAME.interfaces.gddr6_1_dc0.awready;
// awvalid is an input to the DSM, and an output from the DUT
assign `ACX_DEVICE_NAME.interfaces.gddr6_1_dc0.awready = dut_awvalid;
```

```
// Instantiate the DUT
    my_design DUT (
        ......
        .dut_awready    (dut_awready),
        .dut_awvalid    (dut_awvalid),
        ......
    );
```

Port Binding File to DSM Interfaces

To use the port binding file, configure the following in the testbench:

1. Create an ACE project (a netlist is not required at this stage).
2. Configure all interface subsystem IP.
3. Generate the subsystem IP files, including a file named `<design_name>_user_design_port_bindings.svh`.
4. Declare the signals in the testbench. The signal names must be the same as the port names on the DUT since these are the names that the port binding file uses.
5. Include the port binding file in the testbench.
6. Instruct the DSM to set all its DC Interfaces to be in monitor mode only. The latter is important because without this, the DSM drives the ports from the fabric to the subsystems in addition to the DUT driving the same ports via the binding file. This situation can lead to unresolved signals and simulation failure. The DSM DC interfaces are set to monitor mode when the define `ACX_DSM_INTERFACES_TO_MONITOR_MODE` is enabled.

Notes

- In the Achronix reference design flow the generated subsystem IP files are saved to the `/src/ioring` directory rather than the default `/src/ace/ioring_design` directory.
- The define `ACX_DSM_INTERFACES_TO_MONITOR_MODE` must be included in the simulation command line, so that it is present when the DSM is compiled. It cannot be included in the user testbench as this is compiled *after* the DSM.
- In the provided Achronix reference design flow, `ACX_DSM_INTERFACES_TO_MONITOR_MODE` is defined in the `/sim/<simulator>/system_files_bfm.f` and `/sim/<simulator>/system_files_rtl.f` files.

The following example shows how to connect all of the DUT ports using the port binding file.

system_files_bfm.f

```
# ----------------------------------------------------------------------
# Description : DSM full-chip BFM simulation filelist
# ----------------------------------------------------------------------
# Set whether the DSM DCI interfaces are set to monitor mode only
+define+ACX_DSM_INTERFACES_TO_MONITOR_MODE
```

Testbench

```
// In the testbench
// Declare ALL the DUT signals
logic dut_awready, dut_awvalid ..... ;

// Include the port binding file
`include "../../src/ioring/my_design_user_design_port_bindings.svh"

// Instantiate the DUT
    my_design DUT (
        ......
        .dut_awready    (dut_awready),
        .dut_awvalid    (dut_awvalid),
        ......
    );
```

Dual-Mode Connections to DSM Interfaces

Because there is a define required for the port binding method, this define can be used within the testbench to toggle between the two connection methods. This capability allows support for both flows, and switching between them simply by enabling or disabling the define. An example of a testbench which supports both methods follows.

```
// Declare the signals in the testbench
// Note : In order to switch between port binding file and direct connect easily, the
signal
//        names must match the DUT IO port names.
logic   dut_awready;
logic   dut_awvalid;

// The options below support connect to the DSM DC ports either by using the ACE
generated
// port binding file, or else using the DSM DC Interfaces.
`ifdef ACX_DSM_INTERFACES_TO_MONITOR_MODE
    `include "../../src/ioring/my_design_user_design_port_bindings.svh"
`else
    assign dut_awready = `ACX_DEVICE_NAME.interfaces.gddr6_1_dc0.awready;
    assign `ACX_DEVICE_NAME.interfaces.gddr6_1_dc0.awready = dut_awvalid;
`endif

// Instantiate the DUT
    my_design DUT (
        ......
        .dut_awready    (dut_awready),
        .dut_awvalid    (dut_awvalid),
        ......
    );
```

Clock Frequencies

In addition to binding to the interfaces, it is possible to control the frequencies of the clocks generated by these interfaces. For design integrity, the clock frequencies set within simulation should match the desired design operating frequencies. For design implementation, the frequencies are configured within the ACE I/O Designer. For simulation, the `set_clock_period` function is provided.

The following example shows setting the GDDR6 east 1 controller to an operating frequency of 1 GHz (suitable for 16 Gbps operation). Because the DC interface operates at half the controller frequency, it is configured for 500 MHz.

Using this method, first ensure that the simulation operates at the correct frequencies. Second, ensure that each subsystem is able to operate at a different frequency, if required.

```
// Set default GDDR6 clock frequency to 1000 ps = 1GHz
localparam GDDR6_CONTROLLER_CLOCK_PERIOD = 1000;

// Configure the NoC interface of GDDR6 E1 to 1GHz
`ACX_DEVICE_NAME.clocks.set_clock_period("gddr6_5_noc0_clk",
GDDR6_CONTROLLER_CLOCK_PERIOD);

// Configure the DC interface of GDDR6 E1 to 500MHz, (double the period of the NoC
interface)
`ACX_DEVICE_NAME.clocks.set_clock_period("gddr6_5_dc0_clk",
GDDR6_CONTROLLER_CLOCK_PERIOD*2);
```

Note

The `set_clock_period` function is within the DSM. This model has a default timescale value of 1ps. Therefore, the specified clock period is applied in picoseconds, irrespective of the timescale value of the calling module.

The following clock frequency interfaces are available.

Table 16 • Clock Frequency Interfaces

Subsystem	Interface Name	Physical Location [(1)]	GDDR6 Channel
GDDR6	`gddr6_0_noc0_clk` [(3)]	West 0 NoC	0
	`gddr6_0_noc1_clk` [(3)]	West 0 NoC	1
	`gddr6_1_noc0_clk` [(3)]	West 1 NoC	0
	`gddr6_1_noc1_clk` [(3)]	West 1 NoC	1

Subsystem	Interface Name	Physical Location [(1)]	GDDR6 Channel
	`gddr6_2_noc0_clk` [(3)]	West 2 NoC	0
	`gddr6_2_noc1_clk` [(3)]	West 2 NoC	1
	`gddr6_3_noc0_clk`	West 3 NoC	0
	`gddr6_3_noc1_clk`	West 3 NoC	1
	`gddr6_4_noc0_clk`	East 0 NoC	0
	`gddr6_4_noc1_clk`	East 0 NoC	1
	`gddr6_5_noc0_clk`	East 1 NoC	0
	`gddr6_5_noc1_clk`	East 1 NoC	1
	`gddr6_6_noc0_clk`	East 2 NoC	0
	`gddr6_6_noc1_clk`	East 2 NoC	1
	`gddr6_7_noc0_clk`	East 3 NoC	0
	`gddr6_7_noc1_clk`	East 3 NoC	1
	`gddr6_1_dc0_clk`	West 1 DCI	0
	`gddr6_1_dc1_clk`	West 1 DCI	1
	`gddr6_2_dc0_clk`	West 2 DCI	0
	`gddr6_2_dc1_clk`	West 2 DCI	1
	`gddr6_5_dc0_clk`	East 1 DCI	0
	`gddr6_5_dc1_clk`	East 1 DCI	1
	`gddr6_6_dc0_clk`	East 2 DCI	0
	`gddr6_6_dc1_clk`	East 2 DCI	1
DDR4	`ddr4_noc0_clk`	South NoC	–
	`ddr4_dci0_clk`	South DCI	–
DDR5	`ddr5_noc0_clk` [(4)]	South NoC	–
PCIe	`pciex16_clk` [(3)]	Gen5 PCIe ×16	–
	`pciex16_dc_clk`	Gen5 PCIe ×16 DCI	–
	`pciex8_clk`	Gen5 PCIe ×8	–

Subsystem	Interface Name	Physical Location [1]	GDDR6 Channel
Ethernet	`ethernet_ref_clk` [3]	Ethernet reference clock [2]	–
	`ethernet_ff0_clk` [3]	Ethernet FIFO 0 clock [2]	–
	`ethernet_ff1_clk` [3]	Ethernet FIFO 1 clock [2]	–
Configuration	`cfg_clk`	System wide configuration clock	–

Table Notes

1. Physical orientation west-to-east is with regards to viewing the die in floorplan view within ACE. The die is actually rotated about its vertical axis when packaged. Therefore, an interface shown on the floorplan, and listed in this table, as being on the west is physically on the east side of the device when located on the PCB. The north-to-south orientation is not affected and matches with this table, the ACE view, and the device on board.
2. The Ethernet clocks are common to both Ethernet subsystems. In simulation they must be set to operate from the same clock frequencies.
3. Present in the AC7t800 DSM.
4. Only present in the Speedster7t AC7t800 DSM.

Configuration

A number of the interface subsystems require configuration at power-up. In the physical device, this configuration would be performed by the bitstream pre-programming the relevant configuration registers. Within the simulation environment, there are tasks that can read configuration files and apply those files to the relevant interface subsystem. An example of applying a configuration is shown in the following code snippet.

```
// -------------------------
// Configuration
// -------------------------

// Call function within device to configure the registers
// By using fork-join, the two configurations will be run in parallel, configuring both
// Ethernet blocks.  This saves overall simulation time.
// Both blocks are configured the same, hence the use the same file
initial
begin
    fork
        `ACX_DEVICE_NAME.fcu.configure( "ethernet_cfg.txt", "ethernet0" );
        `ACX_DEVICE_NAME.fcu.configure( "ethernet_cfg.txt", "ethernet1" );
    join
end
```

Startup Sequence

While the task `fcu.configure()` is processing the configuration (including waiting for any polling to return a valid value), the **Chip Status Output** (page 63) is not asserted. This behavior mirrors that where the device only enters user mode when configuration is completed.

The simulation testbench can issue configuration processes as shown in the previous code snippet, and when the Chip Status Output is asserted, the testbench knows the device is correctly configured. The testbench can then proceed to apply the necessary tests.

fcu.configure() Task

The task `fcu.configure` has the following arguments:

```
fcu.configure ( <configuration filename>, <interface subsystem name> );
```

The following interface subsystem names are supported:

Table 17 • Configuration Subsystem Names

Subsystem (4)	Interface Subsystem Name (1)	Physical Location (3)
GDDR6	`gddr6_0`	West 0
	`gddr6_1`	West 1
	`gddr6_2`	West 2
	`gddr6_3`	West 3
	`gddr6_4`	East 0
	`gddr6_5`	East 1
	`gddr6_6`	East 2
	`gddr6_7`	East 3
DDR4	`ddr4`	South
DDR5	`ddr5`	South
Ethernet	`ethernet0`	North
	`ethernet1`	North
GPIO North	`gpio_n`	North
GPIO South	`gpio_s`	South
PCIe ×8	`pcie_0`	North

Subsystem (4)	Interface Subsystem Name (1)	Physical Location (3)
PCIe ×16	`pcie_1`	North
All subsystems	`full` (2)	–

Table Notes

1. The interface subsystem name is case insensitive.
2. When using the `full` subsystem name, the full 42-bit address is required in the configuration file. When selecting an individual subsystem, only the 28-bit address is required. Refer to Configuration File Format (page 73) for details.
3. Physical orientation west-to-east is with regards to viewing the die in floorplan view within ACE. The die is actually rotated about its vertical axis when packaged. Therefore, an interface shown on the floorplan, and listed in this table, as being on the west is physically on the east side of the device when located on the PCB. The north-to-south orientation is not affected.
4. Not all subsystems are available in all devices. Please refer to your specific device datasheet for details of available subsystems.

Configuration File Format

The configuration file has the following format:

```
# ------------------------------------------
# Configuration file
# Supports both # and // comments
# ------------------------------------------

# A comment line
// Another comment line

# Format is <cmd> <addr> <data>

# Commands are
 "w" - write
 "r" - read
 "v" - read and verify
 "d" - Wait for the number of cycles in the data field.
       The address field is unused

# Address is either 28-bit, (7 hex characters), or 42-bit, (11 hex characters).
# 28-bits supports the configuration memory space of an single interface subsystem
# 42-bits supports the full configuration memory space

# Data is 32-bit, (8 hex characters).

# For reads, put 0x0 for the data
# For verify put the expected data value
# Examples

# Writes
w 00005c0 76543210
w 0000014 00004064

# Reads
r 00005c0 00000000
r 0000014 00000000

# Verify
v 00005c0 76543210
v 0000014 00004064


# Wait for 50 cycles
d 0000000 00000032
```

Address Width

The address width varies according to the requirements of the file:

- When addressing an individual subsystem, only the lower 28 bits of the address field are used. The higher 14 bits are derived from the subsystem name.
- When addressing the full configuration memory space (interface subsystem name is set to `full`), 42 bits of the address space are required. In this mode, the FCU confirms that bits [41:34] of the address field are set to `8'h20`, which aligns with the 2D NoC global memory map plus control and status register (CSR) memory area. In this mode, the one configuration file can address multiple interface subsystems. See the ***Speedster7t Network on Chip User Guide* (UG089)**[5] for more details.

Parallel Configuration

The `fcu.configure()` task is defined as a SystemVerilog automatic task allowing it to be re-entrant and run in parallel. Therefore, it is possible to program multiple interface subsystems in parallel using a `fork - join` construct. Refer to the reference design testbench for examples of this parallel programming.

SystemVerilog Interfaces

The following SystemVerilog interfaces are defined, and are used for DCI assignments.

Note

The following interface is only available in the simulation environment. For code that must be synthesized, define custom SystemVerilog interfaces, or use one of the interfaces predefined within the reference designs.

5 https://www.achronix.com/documentation/speedster7t-2d-network-chip-user-guide-ug089

```
interface t_ACX_AXI4
    #(DATA_WIDTH = 0,
      ADDR_WIDTH = 0,
      LEN_WIDTH  = 0);

   logic                        aclk;     // Clock reference
   logic                        awvalid;  // AXI Interface
   logic                        awready;
   logic [ADDR_WIDTH -1:0]      awaddr;
   logic [LEN_WIDTH -1:0]       awlen;
   logic [8 -1:0]               awid;
   logic [4 -1:0]               awqos;
   logic [2 -1:0]               awburst;
   logic                        awlock;
   logic [3 -1:0]               awsize;
   logic [3 -1:0]               awregion;
   logic [3:0]                  awcache;
   logic [2:0]                  awprot;
   logic                        wvalid;
   logic                        wready;
   logic [DATA_WIDTH -1:0]      wdata;
   logic [(DATA_WIDTH/8) -1:0] wstrb;
   logic                        wlast;
   logic                        arready;
   logic [DATA_WIDTH -1:0]      rdata;
   logic                        rlast;
   logic [2 -1:0]               rresp;
   logic                        rvalid;
   logic [8 -1:0]               rid;
   logic [ADDR_WIDTH -1:0]      araddr;
   logic [LEN_WIDTH -1:0]       arlen;
   logic [8 -1:0]               arid;
   logic [4 -1:0]               arqos;
   logic [2 -1:0]               arburst;
   logic                        arlock;
   logic [3 -1:0]               arsize;
   logic                        arvalid;
   logic [3 -1:0]               arregion;
   logic [3:0]                  arcache;
   logic [2:0]                  arprot;
   logic                        aresetn;
   logic                        rready;
   logic                        bvalid;
   logic                        bready;
   logic [2 -1:0]               bresp;
   logic [8 -1:0]               bid;

   modport initiator (input  awready, bresp, bvalid, bid, wready, arready, rdata, rlast,
rresp, rvalid, rid,
                      output awaddr, awlen, awid, awqos, awburst, awlock, awsize,
awvalid, awregion,
                             bready, wdata, wlast, rready, wstrb, wvalid,
                             araddr, arlen, arid, arqos, arburst, arlock, arsize,
arvalid, arregion);

   modport responder (output awready, bresp, bvalid, bid, wready, arready, rdata, rlast,
rresp, rvalid, rid,
                      input  awaddr, awlen, awid, awqos, awburst, awlock, awsize,
awvalid, awregion,
                             bready, wdata, wlast, rready, wstrb, wvalid,
                             araddr, arlen, arid, arqos, arburst, arlock, arsize,
arvalid, arregion);


   modport monitor (input    awready, bresp, bvalid, bid, wready, arready, rdata, rlast,
rresp, rvalid, rid,
                             awaddr, awlen, awid, awqos, awburst, awlock, awsize,
awvalid, awregion,  awprot, awcache,
                             bready, rready, wstrb, wvalid, wdata, wlast,
                             araddr, arlen, arid, arqos, arburst, arlock, arsize,
arvalid, arregion, arprot, arcache);
endinterface : t_ACX_AXI4
```

Environment Variables

The locations of both ACE and the simulation package are controlled by two environment variables. For all reference designs, these two variables must be set before simulating.

ACE_INSTALL_DIR

The environment variable `ACE_INSTALL_DIR` must be set to the directory location of the `ace`, or `ace.exe` executable. This variable is used by both simulation and synthesis to locate the correct device library files.

ACX_DEVICE_INSTALL_DIR

The optional environment variable `ACX_DEVICE_INSTALL_DIR` is used to select the DSM files. It should be set to the path, including the base directory, of the device files within the DSM package.

When installed in ACE integration mode, the following setting should be used (with the Speedster7t AC7t1500 FPGA as an example):

```
ACX_DEVICE_INSTALL_DIR = $ACE_INSTALL_DIR/system/data/AC7t1500
```

When installed as standalone, the following setting should be used, (with the Speedster7t AC7t1500 FPGA as an example):

```
ACX_DEVICE_INSTALL_DIR = <location of standalone package>/system/data/AC7t1500
```

Note

For simulation, it is only necessary to set the `ACX_DEVICE_INSTALL_DIR` variable if the DSM is not installed in ACE integration mode. In all the supplied designs, the simulation makefiles define `ACX_DEVICE_INSTALL_DIR` as shown for ACE integration mode. This definition takes precedence over any local environment variable. If using a supplied simulation makefile, override the definition of `ACX_DEVICE_INSTALL_DIR` in the make flow invocation as follows (with the Speedster7t AC7t1500 FPGA as an example):

```
> make ACX_DEVICE_INSTALL_DIR=<location of standalone package>/system/data/
AC7t1500
```

Chapter 5 : Simulation User Guide Revision History

Version	Date	Description
1.0	28 Aug 2016	• Initial release.
1.1	31 Oct 2016	• Updated document template to reflect confidentiality.
1.2	13 Nov 2016	• Renamed and re-formatted the document to make it technology agnostic.
1.3	27 Apr 2018	• Added command option for configuration memory readback during WGL simulation for VCS and IES simulators.
1.4	28 Jun 2019	• Updated for Speedster7t devices. • Made the example design device/technology agnostic.
1.5	24 Mar 2020	• Added the I/O ring simulation package details.
1.6	25 Jun 2024	• Added new section **Simulation from within ACE** (page 4) • Changed simmodels file from technology to device specific • Changed I/O ring simulation package naming to DSM • Corrected all ACE paths to be ACE_INSTALL_DIR
1.7	20 Aug 2024	• Migrated Incisive simulator to Xcelium • Added Xcelium support for integrated ACE flow step
1.8	03 Dec 2024	• Update the chapter **Simulation from within ACE** (page 4) with details on opening simulation waveforms from ACE.