Simulation User Guide (UG072)

All Achronix Devices



Copyrights, Trademarks and Disclaimers

Copyright © 2024 Achronix Semiconductor Corporation. All rights reserved. Achronix, Speedcore, Speedster, and ACE are trademarks of Achronix Semiconductor Corporation in the U.S. and/or other countries All other trademarks are the property of their respective owners. All specifications subject to change without notice.

Notice of Disclaimer

The information given in this document is believed to be accurate and reliable. However, Achronix Semiconductor Corporation does not give any representations or warranties as to the completeness or accuracy of such information and shall have no liability for the use of the information contained herein. Achronix Semiconductor Corporation reserves the right to make changes to this document and the information contained herein at any time and without notice. All Achronix trademarks, registered trademarks, disclaimers and patents are listed at http://www.achronix.com/legal.

Achronix Semiconductor Corporation

2903 Bunker Hill Lane Santa Clara, CA 95054 USA

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

Table of Contents

hapter 1 : Overview	1
Simulation Software Tool Flow	.1
Simulation Libraries	2
Including Memory Initialization Files	2
hapter 2 : Simulation from within ACE	4
Example Design	4
Configuring the Simulation Tool Environment	4
Configuring the Project Source Files	5
Configuring the Simulation Options	6
Configuring Simulation	9
Default Simulation Flow	. 9
Custom Simulation Flow	12
Running the Simulation Flow Steps1	13
Viewing the Simulation Outputs1	.6
hapter 3 : Simulation Outside of ACE2	0
General Project Setup2	20
General RTL Simulation Flow2	21
General Gate-Level Simulation Flow2	21

General Post-Route Simulation Flow	22
Example Design Description	23
Aldec Riviera Simulator Example	24
RTL Simulation in Riviera	24
Step 1 – Create Simulation Directory	24
Step 2 – Create a .do File	24
Step 3 – Run the Simulation	25
Step 4 – View the Waveform	26
Step 5 – Open the Workspace	26
Step 6 – Initialize the Simulation	
Step 7 – Add Signals to the Waveform	
Step 8 – View the Waveform	
Gate-Level Simulation in Riviera	
Step 1 – Create the Synthesis Project	29
Step 2 – Synthesize the Design	29
Step 3 – Create a Workspace	29
Step 4 – Run the Simulation	
Step 5 – View the Results	
Post-Route Simulation in Riviera	31
Step 1 – Create the ACE Project	
Step 2 – Run Place and Route	
Step 3 – Create the Workspace	
Step 4 – Run the Simulation	
Step 5 – View the Results	
Cadence Xcelium Simulator Example	32
RTL Simulation in Xcelium	

Step 2 - Add Signals to the Waveform 34 Step 3 - Run the Simulation 36 Step 4 - View the Waveform 37 Step 5 - View Console Messages 38 Gate-Level Simulation in Xcelium 39 Step 1 - Create the Synthesis Project 40 Step 2 - Synthesize the Design 40 Step 3 - Run Simulation 40 Step 4 - View Simulation Results 40 Post-Route Simulation in Xcelium 41 Step 1 - Create the ACE Project 41 Step 1 - Create the ACE Project 41 Step 1 - Create the ACE Project 41 Step 3 - Run Simulation 42 Step 4 - View Simulation Results 42 Step 3 - Run Simulation Results 42 Step 4 - View Simulation Results 43 RTL Simulation in QuestaSim 43 Step 1 - Create the Project 43 Step 3 - Create the Vieward Simulation Run 44 Step 4 - Compile the Design 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform 45 Step 7 - Run the Simulation Run 44 Step 7 - R		Step 1 – Invoke the Xcelium Tool	
Step 4 - View the Waveform. 37 Step 5 - View Console Messages 38 Gate-Level Simulation in Xcelium 39 Step 1 - Create the Synthesis Project. 40 Step 2 - Synthesize the Design 40 Step 3 - Run Simulation 40 Step 4 - View Simulation Results 40 Post-Route Simulation in Xcelium 41 Step 1 - Create the ACE Project 41 Step 2 - Run Place and Route 41 Step 3 - Run Simulation Results 42 Step 4 - View Simulation Results 43 RTL Simulation in QuestaSim 43 Step 1 - Create the Project 43 Step 2 - Initialize the Work Library 43 Step 3 - Create the File List 43 Step 4 - Compile the Design 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform 45 Step 7 - Run the Simulation 47 Step 8 - View		Step 2 – Add Signals to the Waveform	34
Step 5 - View Console Messages 38 Gate-Level Simulation in Xcelium 39 Step 1 - Create the Synthesis Project 40 Step 2 - Synthesize the Design 40 Step 3 - Run Simulation 40 Step 4 - View Simulation Results 40 Post-Route Simulation in Xcelium 41 Step 2 - Run Place and Route 41 Step 3 - Run Simulation 42 Step 4 - View Simulation Results 42 Step 2 - Run Place and Route 41 Step 3 - Run Simulation 42 Step 4 - View Simulation Results 42 Step 4 - View Simulation Results 42 Step 4 - View Simulator Example 43 RTL Simulation in QuestaSim 43 Step 1 - Create the Project 43 Step 2 - Initialize the Work Library 43 Step 3 - Create the File List 43 Step 4 - Compile the Design 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform 45 Step 7 - Run the Simulation 47 Step 8 - View the Waveform 48 Gate-Level Simulation in QuestaSim </td <td></td> <td>Step 3 – Run the Simulation</td> <td>36</td>		Step 3 – Run the Simulation	36
Gate-Level Simulation in Xcelium 39 Step 1 - Create the Synthesis Project 40 Step 2 - Synthesize the Design 40 Step 3 - Run Simulation 40 Step 4 - View Simulation Results 40 Post-Route Simulation in Xcelium 41 Step 1 - Create the ACE Project 41 Step 2 - Run Place and Route 41 Step 3 - Run Simulation 42 Step 4 - View Simulation Results 42 Step 3 - Run Simulation 42 Step 4 - View Simulation Results 42 Step 4 - View Simulation Results 42 Step 4 - View Simulation Results 42 Step 1 - Create the ACE Project 43 RTL Simulation in QuestaSim 43 Step 1 - Create the Project 43 Step 1 - Create the Project 43 Step 3 - Create the File List 43 Step 3 - Create the Simulation Run 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform 45 Step 7 - Run the Simulation Run 44 Step 8 - View the Waveform 45 Step 8 - View the Waveform <td></td> <td>Step 4 – View the Waveform</td> <td></td>		Step 4 – View the Waveform	
Step 1 - Create the Synthesis Project		Step 5 – View Console Messages	38
Step 2 - Synthesize the Design 40 Step 3 - Run Simulation 40 Step 4 - View Simulation Results 40 Post-Route Simulation in Xcelium 41 Step 1 - Create the ACE Project 41 Step 2 - Run Place and Route 41 Step 3 - Run Simulation 42 Step 4 - View Simulation Results 42 Step 4 - View Simulation Results 42 Step 4 - View Simulation Results 42 Step 5 - Create the Project 43 RTL Simulation in QuestaSim 43 Step 2 - Initialize the Work Library. 43 Step 3 - Create the File List 43 Step 4 - Compile the Design 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform 45 Step 7 - Run the Simulation Run 44 Step 8 - View the Waveform 45 Step 8 - View the Waveform 45 Step 8 - View the Waveform 48 Gate-Level Simulation in QuestaSim 49	Gate	e-Level Simulation in Xcelium	
Step 3 - Run Simulation. 40 Step 4 - View Simulation Results 40 Post-Route Simulation in Xcelium. 41 Step 1 - Create the ACE Project. 41 Step 2 - Run Place and Route. 41 Step 3 - Run Simulation. 42 Step 4 - View Simulation Results 42 Step 4 - View Simulation Results 42 Step 1 - Create the Project. 43 RTL Simulation in QuestaSim 43 Step 2 - Initialize the Work Library. 43 Step 3 - Create the File List. 43 Step 4 - Compile the Design. 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform. 45 Step 7 - Run the Simulation . 47 Step 8 - View the Waveform. 48 Gate-Level Simulation in QuestaSim 49		Step 1 – Create the Synthesis Project	40
Step 4 - View Simulation Results 40 Post-Route Simulation in Xcelium 41 Step 1 - Create the ACE Project 41 Step 2 - Run Place and Route 41 Step 3 - Run Simulation 42 Step 4 - View Simulation Results 42 Siemens QuestaSim Simulator Example 43 RTL Simulation in QuestaSim 43 Step 1 - Create the Project 43 Step 2 - Initialize the Work Library 43 Step 3 - Create the File List 43 Step 4 - Compile the Design 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform 45 Step 7 - Run the Simulation 47 Step 8 - View the Waveform 48 Gate-Level Simulation in QuestaSim 49		Step 2 – Synthesize the Design	40
Post-Route Simulation in Xcelium 41 Step 1 - Create the ACE Project 41 Step 2 - Run Place and Route 41 Step 3 - Run Simulation 42 Step 4 - View Simulation Results 42 Siemens QuestaSim Simulator Example 43 RTL Simulation in QuestaSim 43 Step 1 - Create the Project 43 Step 2 - Initialize the Work Library 43 Step 3 - Create the File List 43 Step 4 - Compile the Design 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform 45 Step 7 - Run the Simulation 47 Step 8 - View the Waveform 48 Gate-Level Simulation in QuestaSim 49		Step 3 – Run Simulation	40
Step 1 - Create the ACE Project 41 Step 2 - Run Place and Route 41 Step 3 - Run Simulation 42 Step 4 - View Simulation Results 42 Siemens QuestaSim Simulator Example 43 RTL Simulation in QuestaSim 43 Step 1 - Create the Project 43 Step 2 - Initialize the Work Library 43 Step 3 - Create the File List 43 Step 4 - Compile the Design 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform 45 Step 7 - Run the Simulation 47 Step 8 - View the Waveform 48 Gate-Level Simulation in QuestaSim 49		Step 4 – View Simulation Results	40
Step 2 - Run Place and Route. 41 Step 3 - Run Simulation. 42 Step 4 - View Simulation Results. 42 Siemens QuestaSim Simulator Example 43 RTL Simulation in QuestaSim 43 Step 1 - Create the Project. 43 Step 2 - Initialize the Work Library. 43 Step 3 - Create the File List. 43 Step 4 - Compile the Design. 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform. 45 Step 7 - Run the Simulation 47 Step 8 - View the Waveform. 48 Gate-Level Simulation in QuestaSim 49	Pos	t-Route Simulation in Xcelium	41
Step 3 - Run Simulation. 42 Step 4 - View Simulation Results 42 Siemens QuestaSim Simulator Example 43 RTL Simulation in QuestaSim 43 Step 1 - Create the Project. 43 Step 2 - Initialize the Work Library. 43 Step 3 - Create the File List. 43 Step 4 - Compile the Design. 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform. 45 Step 8 - View the Waveform. 48 Gate-Level Simulation in QuestaSim 49		Step 1 – Create the ACE Project	
Step 4 - View Simulation Results 42 Siemens QuestaSim Simulator Example 43 RTL Simulation in QuestaSim 43 Step 1 - Create the Project 43 Step 2 - Initialize the Work Library 43 Step 3 - Create the File List 43 Step 4 - Compile the Design 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform 45 Step 7 - Run the Simulation 47 Step 8 - View the Waveform 48 Gate-Level Simulation in QuestaSim 49		Step 2 – Run Place and Route	
Siemens QuestaSim Simulator Example 43 RTL Simulation in QuestaSim 43 Step 1 - Create the Project 43 Step 2 - Initialize the Work Library 43 Step 3 - Create the File List 43 Step 4 - Compile the Design 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform 45 Step 7 - Run the Simulation 47 Step 8 - View the Waveform 48 Gate-Level Simulation in QuestaSim 49		Step 3 – Run Simulation	
RTL Simulation in QuestaSim 43 Step 1 - Create the Project 43 Step 2 - Initialize the Work Library 43 Step 3 - Create the File List 43 Step 4 - Compile the Design 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform 45 Step 7 - Run the Simulation 47 Step 8 - View the Waveform 48 Gate-Level Simulation in QuestaSim 49		Step 4 – View Simulation Results	42
Step 1 - Create the Project	Siemer	ns QuestaSim Simulator Example	43
Step 2 - Initialize the Work Library	RTL	_ Simulation in QuestaSim	
Step 3 - Create the File List 43 Step 4 - Compile the Design 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform 45 Step 7 - Run the Simulation 47 Step 8 - View the Waveform 48 Gate-Level Simulation in QuestaSim 49		Step 1 - Create the Project	43
Step 4 - Compile the Design 44 Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform 45 Step 7 - Run the Simulation 47 Step 8 - View the Waveform 48 Gate-Level Simulation in QuestaSim 49		Step 2 - Initialize the Work Library	43
Step 5 - Prepare the Simulation Run 44 Step 6 - Set up the Waveform 45 Step 7 - Run the Simulation 47 Step 8 - View the Waveform 48 Gate-Level Simulation in QuestaSim 49		Step 3 - Create the File List	43
Step 6 – Set up the Waveform		Step 4 - Compile the Design	
Step 7 - Run the Simulation		Step 5 – Prepare the Simulation Run	
Step 8 – View the Waveform		Step 6 – Set up the Waveform	45
Gate-Level Simulation in QuestaSim		Step 7 – Run the Simulation	47
		Step 8 – View the Waveform	48
Step 1 – Create the Synthesis Project49	Gate	e-Level Simulation in QuestaSim	
		Step 1 – Create the Synthesis Project	49

	Step 2 – Synthesize the Design	
	Step 3 – Set up the Simulation Project	
	Step 4 – Initialize the Work Library	
	Step 5 – Create the File List	50
	Step 6 – Compile the Design	
	Step 7 – Prepare the Simulation Run	50
Pos	st-Route Simulation in QuestaSim	50
	Step 1 – Create the ACE Project	
	Step 2 – Run Place and Route	
	Step 3 – Set up the Simulation Project	51
	Step 4 – Initialize the Work Library	
	Step 5 – Create the File List	
	Step 6 – Compile the Design	
	Step 7 – Prepare the Simulation Run	52
Synop	sys VCS Simulator Example	52
RT	L Simulation in VCS	
	Step 1 – Run the VCS Simulator	
	Step 2 – Start the Simulation GUI	53
	Step 3 – Open the Simulation Database	
	Step 4 – Add Signals to the Waveform	
	Step 5 – View the Simulation Results	55
Ga	te-Level Simulation in VCS	56
	Step 1 – Create the Synthesis Project	56
	Step 2 – Synthesize the Design	56
	Step 3 – Run the VCS Simulator	56
Pos	st-Route Simulation in VCS	57
	Step 1 – Create the ACE Project	

Step 2 – Run Place and Route	
Step 3 – Run the VCS Simulator	
Chapter 4 : DSM Simulation Package	58
Device Simulation Model	
Description	
Selecting the Required DSM	58
DSM Utility Package	
Device-Specific Simulation Files	59
Instantiate DSM Utility Package	59
Version Control	60
require_version() Task	
Example Design	61
set_verbosity() Task	63
Chip Status Output	63
Bind Macros	64
Direct-Connect Interfaces	
Suggested Flows	65
DSM DC Interfaces	65
Direct Connect to DSM Interfaces	
Port Binding File to DSM Interfaces	
Dual-Mode Connections to DSM Interfaces	
Clock Frequencies	69
Configuration	71
Startup Sequence	
fcu.configure() Task	
Configuration File Format	

Address Width	74
Parallel Configuration	74
SystemVerilog Interfaces	74
Environment Variables	76
ACE_INSTALL_DIR	76
ACX_DEVICE_INSTALL_DIR	76
Chapter 5 : Simulation User Guide Revision History	78

Chapter 1: Overview

UG072

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.



Figure 1 • 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		ory	Description
<ace_install_dir></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 toplevel 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 or The Quickstart Design 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.

Simulation Tool	Environment Variable	Linux Path	Windows Path
Siemens QuestaSim	ACX_QUESTASIM_TOOL_PAT H	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.</ace_install>	Set to the directory path containing the vlog.exe, vcom.exe, and vsim.exe files (i.e., D:\questa_base64_ 2023.3\win64).
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.</ace_install>	Set to the file path of the Riviera runvsimsa.bat file (i.e., D:\Aldec\Riviera- PRO-2020.10- x64\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.</ace_install>	(Not Supported)

Table 2 · Environment Variables Needed for Simulation

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

Simulation Tool	Environment Variable	Linux Path	Windows Path
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</ace_install>	(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.

(i) 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.

```
(i) 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 (pag e 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.

(i) 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 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.

7

E Options ×		-	
	art_AC7t1500		^
 Project Options 			
Target Device	AC7t1500	\sim	
Package	F53	\sim	
Speed Grade	C2 <	\sim	
Core Voltage	0.85	\sim	
Junction Temperature	0	\sim	
Flow Mode	Evaluation	\sim	
Auto-Select Top M	odule		
Incremental Compile	I Compile		
Export All Partitions			
Enable Final Timing	Checks		
HDL Include Path	D:/ACE/doc/examples/quickstart/AC7t1500,	/src	
HDL Defines	DEFINE_A=1 DEFINE_B=0		
✓ Use Default Project	✓ Use Default Project Output Path		
✓ Use Default I/O Rin	g Design Generation Path		~
<		>	

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

(i) 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

(i) Note

The HDL Defines applies to both synthesis and simulation.

8

E Options ×	=	. ם	
Project: quickst	art_AC7t1500	^	
Implementation: impl_1			
 Project Options 			
Target Device	AC7t1500 ~		
Package	F53 ×		
Speed Grade	C2 ~		
Core Voltage	0.85 ~		
Junction Temperature	0 ~		
Flow Mode	Evaluation 🗸		
Auto-Select Top Mo	odule		
Incremental Compile			
Enable Incremental Compile			
Export All Partitions	Export All Partitions		
Enable Final Timing	Checks		
HDL Include Path	D:/ACE/doc/examples/quicksta		
HDL Defines	DEFINE_A=1 DEFINE_B=0		
Use Default Project	✓ Use Default Project Output Path		
Use Default I/O Rin	g 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.

E Options ×		-	
Project: quicksta	rt_AC7t1500		^
Implementation: impl_1			
 Project Options 			
 Simulation 			
Simulation Flow	Default		
Simulation Tool	Aldec Riviera ~		
Testbench Top Module	tb_quickstart		
Enable Device Simul	ation Model		
Compile Simulation			
Riviera Compile Optio	ns		
Simulator Command Arguments (alog) -dbg			
Simulator Command A	Arguments (acom) -dbg		
Run Simulation			
Riviera Run Options			
Simulator Command	Arguments (asim) +access +r		
 Synthesis 			
 Place and Route 			
Advanced Place and Place	Route		
Timing Analysis			~
<		>	

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.

E Options ×		•	
Project: quickstar Implementation: impl_1	rt_AC7t1500		^
 Project Options 			
 Simulation 			
Simulation Flow	Default	\sim	
Simulation Tool	Aldec Riviera	\sim	
Testbench Top Module	tb_quickstart		
Enable Device Simula	ation Model		
Compile Simulation			
Riviera Compile Option	าร		
Simulator Command A	rguments (alog)dbg		
Simulator Command A	arguments (acom) -dbg		
Run Simulation			
Riviera Run Options			
Simulator Command A	arguments (asim) +access +r		

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.

		-	Π
E Options ×			_
, ,	rt_AC7t1500		^
Implementation: impl_1			
Project Options			
 Simulation 			
Simulation Flow	Default	~	
Simulation Tool	Aldec Riviera	~	
Testbench Top Module	tb_quickstart		
Enable Device Simul	ation Model		
Compile Simulation			
Riviera Compile Optio	ns		
Simulator Command A	Arguments (alog)dbg		
Simulator Command A	Arguments (acom)dbg		
Run Simulation	\$		
Riviera Run Options			
Simulator Command A	Arguments (asim) +access +r		
 Synthesis 			
Place and Route			
Advanced Place and F	Route		
Timing Analysis			
<		>	

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:

Simulator Tool	Define
Aldec Riviera	+define+RIVIERA
Siemens QuestaSim	+define+QUESTASIM

Simulator Tool	Define
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:

E Options ×			
Project: Implementat	quickstart_AC7t1 ion: impl_1	500	
Project Op	otions		
 Simulation 	ı		
Simulation	Flow	Custom	~~
Custom Sir	mulation Command	my_custom_sim_command	
Testbench ⁻	Top Module	tb_quickstart	
Enable I	Device Simulation Mo	odel	

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 simulatitor. See Running ACE in the ACE Users Guide (UG070)² for more details.

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

E Options ×			
Project: Implementati	quickstart_AC7t1 on: impl_1	500	
Project Op	tions		
 Simulation 			
Simulation	Flow	Custom	
Custom Sin	nulation Command	my_custom_sim_command	
Testbench T	op Module	tb_quickstart	
Enable D	Device Simulation Mo	odel	

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.

Running the Simulation Flow Steps

ACE provides the following optional simulation flow steps:

Table 3 · 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

13

Name	ID
 Run Post-Route Netlist Simulation 	run_simulation_routed
Design Completion	
 Generate Final Simulation Netlist 	write_netlist_final
 Run Final Netlist Simulation 	run_simulation_final

Table Notes

All flow step IDs can be executed at the ACE GUI Tcl console (see Tcl Console View in the ACE Users Guide (UG070³) 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:

```
resume] [-ic <string>]
```

```
run [-step <string>] [-stop_at_step <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.

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



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.

ACE - Achronix CAD Environment - Version 10.0 - qui	ickstart_AC7t1500->impl_1 (AC7t1500)	- 🗆 X
File Edit Actions Window Help		
🧀 🔛 🐚 🗁 🛷 🐚 👘 🕐 📯 📴 📅 🖼 📰 🖓 🗫 🎇 🐅 🖓		٩
[™]	Options ×	Quickstart_AC7t1500-riviera-rtl.log ×
🖆 😂 🗟 🕼 📑 🔻 🗞 🛱 🛱 🍪 🔗 🗶	8 Project: quickstart AC7t1500	A /I # SLP: Stanted
Y 2 quickstart_AC7t1500	Implementation: impl 1	72 # SLP: Elaboration phase
Y 🗁 Source		73 # SLP: Elaboration phase done : 0.2 [s] 74 # SLP: Generation phase
🗁 IP	Project Options	75 # SLP: Generation phase done : 0.7 [s]
✓ ⇔ RTL	✓ Simulation	76 # SLP: Finished : 0.9 [s]
Counter.v	Simulation Flow Default	77 # SLP: 0 primitives and 18 (100.00%) other pr 78 # SLP: 40 (100.00%) signals in SLP and 0 inte
🗎 quickstart.v		78 # SLP: 40 (100.00%) signals in SLP and 0 inte 79 # ELAB2: Elaboration final pass complete - ti
	Simulation Tool Aldec Riviera	80 # KERNEL: SLP loading done - time: 0.3 [s].
 Place and Route Simulation 	Testbench Top Module tb guickstart	81 # KERNEL: SLP simulation initialization done
 Simulation Estbench 		82 # KERNEL: Kernel process initialization done. 83 # Allocation: Simulator allocated 4685 kB (ellocated 4685 kB)
tb quickstart.v	Enable Device Simulation Model	84 # KERNEL: ASDB file was created in location C
 ✓ ➢ Output 	Compile Simulation	85 run -all
> > Reports	Riviera Compile Options	86 # KERNEL: At time 0, outco 87 # KERNEL: At time 1006633860000, outco
∽ ∉ impl 1	Simulator Command Arguments (alog) -dbg	87 # KERNEL: At time 1006653860000, outco
> 🗁 syn	Simulator Command Arguments (acom) -dbg	89 # KERNEL: At time 3019899780000, outco
> 🗁 pnr	Simulator Command Arguments (acom) -dbg	90 # KERNEL: At time 4026532740000, outco
🖌 🗁 sim	Run Simulation	91 # RUNTIME: Info: RUNTIME_0068 tb_quickstart.v 92 # KERNEL: Time: 4026532840 ns, Iteration: 0,
∽ 🗁 rtl	Riviera Run Options	93 # KERNEL: stopped at time: 4026532840 ns
> 🗁 custom		94 # VSIM: Simulation has finished. There are no
> 🖻 riviera	Simulator Command Arguments (asim) +access +r	95 endsim 96 # VSIM: Simulation has finished.
♦ Flow × Nultiprocess	□ → Synthesis	96 # VSIM: Simulation has finished. 97 exit
 IP Configuration 	Place and Route	98
Generate All IP Design Files		> < > >
RTL Simulation		
✓ ✓ Run RTL Simulation	Tcl Console ×	
✓ ■ Synthesis	Aldec Riviera simulation completed successfully	^
✓ ▲ Run Synthesis ▲ Run Gate-level Netlist Simulation	Flow step "run_simulation_rtl" completed in 29 second	nds. Peak memory usage is 240 MB. Cputime 29 seconds.
 Run Gate-level Netlist Simulation Place and Route 		
A Run Prepare	Overall flow completed in 29 seconds. Peak memory w	usage is 240 MB. Cputime 29 seconds.
A Run Place		~
▲ Run Post-Placement Timing Analysis	<	>
A Run Route	✓ cmd>	e di seconda di s
		41M of 512M

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.

· · · · · · · · · · · · · · · · · · ·					
ACE - Achronix CAD Environment - Version 10.0 - quickstart_	AC7t1500->impl_1 (AC7t1	500)			- 🗆 🗙
File Edit Actions Window Help					
🧀 🔛 🕼 🗁 🛷 🐚 🏚 🕢 🐙 📴 🖷 🏙 💤 💭 🛠 🖦 🖓					Q
🍋 *Projects × 🔰 🐸 🖓 🕼 🗑 🐨 🗞 🖗 🕮 🖏 🔗 🗰 🗄 🗖	1 E Options ×				uickstart_A × 🔭 🗖 🗖
v # quickstart_AC7t1500	Project: quick	start AC7t1500	^	48 # ALOG: Compile success 0 H 49 # ALOG: done	Errors 1 Warnings
Y 🗁 Source	Implementation: impl			50 alog +incdir+"D:/ACE_10.0,	/Achronix/Achroni
l₽	Project Options			51 # ALOG: Pass 1. Scanning mo	
✓ l≥ RTL				52 # ALOG: Info: VCP2113 Modu 53 # ALOG: Info: VCP2113 Modu	
Counter.v	 Simulation 			54 # ALOG: Pass 2. Processing	
B quickstart.v > ≥ Synthesis	Simulation Flow	Default		55 # ALOG: Warning: VCP2597 D	
 Synthesis Place and Route 	Simulation Tool	Aldec Riviera		56 # ALOG: Pass 3. Processing 57 # ALOG: ELB/DAG code genera	
 Simulation 				58 # ALOG: Info: VCP2113 Modul	
✓ ▷ Testbench	Testbench Top Modu	ile tb_quickstart		59 # ALOG: Unit top modules: 1	
tb_quickstart.v	Enable Device Sin	nulation Model		60 # ALOG: \$root top modules: 61 # ALOG: Compile success 0 8	
Y 🗁 Output	Compile Simulati	on		62 # ALOG: done	Ŭ
> 🗁 Reports	-Riviera Compile Op	tions		63 asim +access +r tb_quickst 64 # ELBREAD: Elaboration proc	
~ ∅ impl_1				65 # ELBREAD: Elaboration proc	
> 🖻 syn	Simulator Comman			66 # KERNEL: Main thread init:	iated.
> 🗁 pnr 🗸 🗁 sim	Simulator Comman	d Arguments (acom)dbg		67 # KERNEL: Kernel process in	
> 🦢 sim	Run Simulation			68 # ELAB2: Elaboration final 69 # ELAB2: Create instances	
v 🖻 gate				70 FELBREAD: Error: ELBREAD	0097 quickstart.v
> @ custom	Riviera Run Options			71 # KERNEL: Error: E8005 : Ke	
> 🗁 guestasim	Simulator Comman	d Arguments (asim) +access +r		72 # VSIM: Error: Simulation : 73 # SCRIPTER: Error: CorUsers	
✓	→ Synthesis		~	74	y scoresense, quit
> 😕 compile	<		>	<	>
> 🗁 work	Tcl Console ×			ę	3 🔠 🕕 🔺 🖦 🗎 🕴 🗖 🗖
dataset.asdb	-			0,000000000, quieno cai c_,10,000	
library.cfg				art_AC7t1500/impl_1/sim/rtl/ri	
🎭 Flow 🗵 🐌 Multiprocess 💿 🕨 🖿 🗉				lator log file for details: C:	
IP Configuration	Flow step "run_si	mulation_rtl" failed in 13	seconds. P	eak memory usage is 240 MB. Cp	utime 13 seconds.
Generate All IP Design Files					
RTL Simulation	Overall flow fail	led in 13 seconds. Peak mem	ory usage i	s 240 MB. Cputime 13 seconds.	
✓ ¥ Run RTL Simulation [1 errors]					~
✓ ■ Synthesis	<				>
🗹 🛦 Run Synthesis	/ cmd>		_		Z.
Writable	Insert	1:1:0	45M d	of 512M	

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 4 · 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}	
{vsim-PLI-3406}	ACE flow stop reports on error
Case insensitive {simulation failed}	ACE flow step reports an error
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 5 · User Design Project Directory Structure

Directory	Description	
<project_dir></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 43) 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:

 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-androute 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 43). 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:

 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 43). 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.

(i) 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 24) 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
                     ,
                       , // Enable input
            en
            down_not_up, // Up Down input
                           // Count output
            cnt
                     ,
                           // Overflow output
            ovrflow
            );
    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}};</pre>
        else if (en) begin
            if (down_not_up) begin
                cnt <= {~(^(cnt & `CNT_WIDTH'b01100011)),cnt[`CNT_WIDTH-1:1]};</pre>
            end else begin
```

```
cnt <= {cnt[`CNT_WIDTH-2:0],~(^(cnt & `CNT_WIDTH'b10110001))};
end
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 $\langle DEVICE \rangle$ 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:

6		
# KERNEL: Kernel pro # Allocation: Simula # KERNEL: ASDB file	n final pass complete - time: 0.3 [s]. ocess initialization done. otor allocated 7432 kB (elbread=1450 elab2=5844 kernel=137 sdf=0) was created in location /simulation_example_design/src/results/	
lfsr_updown_cnt/data	ISET. ASOD	
run -all		
# KERNEL: Reset rele		
# 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!!	
<pre># RUNTIME: Info: RUN</pre>	<pre>ITIME_0068 lfsr_updown_cnt_tb.v (66): \$finish called.</pre>	
# 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 -sav	/e	
quit		

sim.log

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 \rightarrow Open \rightarrow Workspace.



Figure 13 • Riviera Workspace

(i) 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**.



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** \rightarrow **Waveform**.



Figure 15 • Adding the Waveform

Step 8 - View the Waveform

Click the **Restart** and **Run -all** button to view the waveform.
R	Riviera-	PRO - Untitled4.awc*		_ D X
<u>File Edit View Control Simulation</u>	n W <u>a</u> veform <u>T</u> ools <u>W</u> indow <u>H</u> elp			
🖻 🔻 🧀 💾 🐚 🙋 i 🐰 🖒 📋	5 C 🚡 🔂 🎭 🌚 🕛 🕨 🛓 1	00 ns 🗘 🗎 📢 📕	5. 5 169 ns + 1	4
📑 🏛 🕒 😐 🍍 🐂 🐪	۹، 😫 🗟 🔫 🗊 🎫 🖓 🖒 😣 🖓	🛷 🕘 🕔 🕛 🔲 🚍 🚔	🐵 I 2 🗕 🖨 有 有 有	🗄 🚓 📝 🖄 🖗 🌐
🖬 Desi 🗣 💌 📴 Untitled4.awc	🛚 💽 💽 Ifsr_updown_cnt_tb.v		•	t Hierarchy 🗘 🔍
debug 🔻 » 🕴 🛟 🕀		Δ Δ' 🐺 🌐 🏦 🛄 » % з	Auto 👻 🛙 🛪	▼ Ŷ (.*) »
<pre> workspace for up, for up</pre>				Name Determine D
	× «	0fs - 177450ps	169 ns + 1 ▼ ▶ ● ● ●	
🖻 Console	¢7× 📽 🖓 Wa	tch 1		×3¢
🁌 🚓 🚯 Errors 🛕 Warn	ings 📀 Messages 🛛 😤 🗛 📂			👻 🌳 (.*) Attributes 💌
• # VSIM: Simulation has fin: test vectors to simulate.	shed. There are no more	ed Name Value clk_in 1 rst 0 1 1 	Type wire wire wire	Last Event Time 169ns 3ns 47ns 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 Symplify under <project_dir>/src/syn, include <ACE_INSTALL_DIR>/libraries/ device_models/<DEVICE>_symplify.sv followed by the RTL design files and constraint files.

Where $\langle DEVICE \rangle$ 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 24) 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 24) to run simulation.

Step 5 - View the Results

Follow Steps 4 to the end of RTL Simulation in Riviera (page 24) 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 24) 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 24) to run simulation.

Step 5 - View the Results

Follow Steps 4 to the end of RTL Simulation in Riviera (page 24) 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:

```
<project_dir>/src/tb/lfsr_updown_cnt_tb.v -gui -sv
```

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

Table 6 · 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
		This option is to be used only when reading back configuration memory (CMEM) contents during WGL simulations.
+define+CMEM_READBACK	No	(i) 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 Loading native compiled co				Done
Design hierarchy summary:				
Ins	tances	Unique		
Modules:	837	258		
UDPs:	Θ	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	n snaps	shot:		
Loading snapshot worklib.bram SVSEED default: 1	_output	cs:vp	•••••	Done

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.



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.

		v	Vaveform 1	- SimVision					(×
Eile Edit Yiew Explore	For <u>m</u> at Sim <u>u</u> lat	ion Windows Help.							cāden	c e`
🎥 🐮 💼 🎥 🖉 🔥 🛛	₩ 🗅 🛍 🗙 💭) 🕅 🗑 🏢	TxE -		🗳 - 🖶	Send To:	🗟 🏯 🛛	. 📬 🕅		
Search Names: Signal V	三 他 的	Search Times:	Digital Val	ue 🔻 🗌	I M. M.					
IN TimeA▼ = 0	ps - p: = =	🕨 - 🕄 💽 - 🛄 📧		🔴 🖦 🛙	•••• • ••	Time: 🖁 🗖	0 : 2000p	s 🔳	🔜 🏄 – =	 *
× ⊕ Baseline▼=0		Baseline = 0								
Name	⊙▼ Cursor ⊙▼	TimeA = 0 0 200ps	400ps	600ps 8	00ps 1000ps	1200ps	1400ps	1600ps	1800ps	
Image: Second	x 'h xx									A
down_not_up	×									
en 🔁	×									
ovrflow	×									
💔 🔽 rst	×									
1										
K-C										
										-
			. 400	1600 Is	1000	1200	1400	1600	2000ps 刘	2
Ø 🏽 🗖								6 obje	ects select	ed

Figure 19 • Xcelium Waveform Window

Step 3 – Run the Simulation

Run simulation by selecting **Simulation** \rightarrow **Run**.

			SimVision				_ 0	
		verorm 1 -	SIMVISION					•
Eile Edit Yiew Explore Format	Simulation Windows Help						c ā d e	nce
Search Names: Signal V	Run Stop SystemC/C/C++ debug Interrupt C/C++ Debugger Terminate C/C++ Debugger	F2	ie -	** - ♣ ■ <u>∭</u> , <u>∭</u> ,		0 : 2000ps		
▲ Baseline ▼=0 ■ ■ <	Step Next Next in Scope Return Advance Reset to Start Save Checkpoint Restart from Checkpoint Reinvoke Simulator Terminate & Disconnect Terminate & Dost-Process Disconnect Set Breakpoint Breakpoints	F5 F6 F7 Shift+F5 //		0pş 1000ps	1200ps	1400ps	1600ps 1800ps	
	Deposit Value Create Probe Create Force Create Driver Create Command Alias Create Debug Variable Show		<u>1600 . 180</u>	o , , 1000, , ,	1200	1400, , , 1	600, , , , 2000ps	N
🎯 🎥 🔚 Run the simulation until	it hits the next breakpoint						6 objects sele	cted

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.

r Waveform 7	I - SimVision – 🗆 🙁
Eile Edit Yiew Explore Format Simulation Windows Help	cādence"
🏪 🐏 🖉 🐟 😹 🗅 🛍 🗙 🏬 🗰 📶 🔤 🔲	🗳 - 🛖 🛛 Send To: 🕵 👯 🗟 🎆 📰 📰
Search Names: Signal 💌 🦍 🍿 Search Times: Digital Va	ilue V
💘 TimeA▼ = 169,000 💌 ps▼ 👯 ▼ 👍 💁 🏫 🚺 🗈 ▼ 🔟 🗹 📮 🐺] 🔴 📭 📷 435,000ps + 0 Time: 음급 168,200ps : 170 I 🔍 👷 🗧 🎫
Baseline▼=0 Image: Cursor-Baseline▼=169,000ps Name Image: Cursor Imag	TimeA = 169,000ps 168,800ps 169,000ps 169,200ps 169,400ps 169,600ps 169,800ps 170,000ps
	00
	1 object selected

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.

	lelp	cādenc
🤄 🔭 💥 🗋 🛍 🗶 🛛 Text Search: 🦷	■ 6, 11	
▶ • 🛄 🔣 📳 💾 🏪 🏹 🖬 🔤 🖬 🐼 435,000	ps + 0 📄	X 0
celium> database -open waves -into wave robe -create -shm lfsr_updown_cnt_tb.DU own_not_up lfsr_updown_cnt_tb.DUT.en lf reated default SHM database waves celium> Created probe 1 celium> run ime = 0 rst = 1 en = 0 down_nt_u ime = 15000 rst = 1 en = 0 down_r ime = 25000 rst = 0 en = 1 down_r ime = 35000 rst = 0 en = 1 down_r ime = 45000 rst = 0 en = 1 down_r ime = 65000 rst = 0 en = 1 down_r ime = 65000 rst = 0 en = 1 down_r ime = 65000 rst = 0 en = 1 down_r ime = 75000 rst = 0 en = 1 down_r ime = 85000 rst = 0 en = 1 down_r ime = 105000 rst = 0 en = 1 down_r ime = 105000 rst = 0 en = 1 down_r ime = 105000 rst = 0 en = 1 down_r ime = 115000 rst = 0 en = 1 down_r ime = 125000 rst = 0 en = 1 down_r ime = 15000 rst = 0 en = 1 down_r ime = 15000 rst = 0 en = 1 down_r ime = 165000 rst = 0 en = 1 down_r ime = 165000 rst = 0 en = 1 down_r ime = 165000 rst = 0 en = 1 down_r ime = 125000 rst = 0 en = 1 down_r ime = 125000 rst = 0 en = 1 down_r ime = 225000 rst = 0 en = 1 down_r ime = 225000 rst = 0 en = 1 down_r ime = 225000 rst = 0 en = 1 down_r ime = 225000 rst = 0 en = 1 down_r ime = 225000 rst = 0 en = 1 down_r ime = 225000 rst = 0 en = 1 down_r ime = 235000 rst = 0 en = 1 down_r ime = 235000 rst = 0 en = 1 down_r ime = 305000 rst = 0 en = 1 down_r ime = 325000 rst = 0 en = 1 down_r ime = 325000 rst = 0 en = 1 down_r ime = 325000 rst = 0 en = 1 down_r ime = 325000 rst = 0 en = 1 down_r ime = 325000 rst = 0 en = 1 down_r ime = 325000 rst = 0 en = 1 down_r ime = 325000 rst = 0 en = 1 down_r ime = 325000 rst = 0 en = 0 down_r ime = 325000 rst = 0 en = 0 down_r ime = 335000 rst = 0 en = 0 down_r ime = 335000 rst = 0 en = 0 down_r ime = 335000 rst = 0 en = 0 down_r ime = 375000 rst = 0 en = 0 down_r ime = 375000	JT.clk_in lfsr_updown_cnt_tb_DUT.cnt lfsr_up fsr_updown_cnt_tb_DUT.ovrflow lfsr_updown_cr bat_up = 0 cnt = 00000000 ovrflow = 0 hat_up = 0 cnt = 00000000 ovrflow = 0 hat_up = 0 cnt = 00000010 ovrflow = 0 hat_up = 0 cnt = 00000101 ovrflow = 0 hat_up = 0 cnt = 00001010 ovrflow = 0 hat_up = 0 cnt = 00010101 ovrflow = 0 hat_up = 0 cnt = 00101011 ovrflow = 0 hat_up = 0 cnt = 01010111 ovrflow = 0 hat_up = 0 cnt = 01010111 ovrflow = 0 hat_up = 0 cnt = 01011110 ovrflow = 0 hat_up = 0 cnt = 10101111 ovrflow = 0 hat_up = 0 cnt = 11110001 ovrflow = 0 hat_up = 0 cnt = 11110001 ovrflow = 0 hat_up = 0 cnt = 11110001 ovrflow = 0 hat_up = 0 cnt = 10001100 ovrflow = 0 hat_up = 0 cnt = 10000111 ovrflow = 0 hat_up = 0 cnt = 10000111 ovrflow = 0 hat_up = 0 cnt = 10000110 ovrflow = 0 hat_up = 1 cnt = 10100001 ovrflow = 0 hat_up = 1 cnt = 10100001 ovrflow = 0 hat_up = 1 cnt = 10100001 ovrflow = 0 hat_up = 1 cnt = 10100000 ovrflow = 0 hat_up = 1 cnt = 10111000 ovrflow = 0 hat_up = 1 cnt = 10111000 ovrflow = 0 hat_up = 1 cnt = 10111000 ovrflow = 0 hat_up = 1 cnt = 100000000 ovrflow = 0 hat_up = 1 cnt = 000000000 ovrflow = 0 hat_up = 1 c	alaying and the DUT

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.

UG072

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 $\langle DEVICE \rangle$ 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 32) section, except that the gate-level simulation uses the mapped netlist lfsr_updown_cnt_gate.v file instead of source RTL files.

(i) Note

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

```
<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 32) (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.

_			
B. De	esign Browser 1 - SimVision		- • ×
$\underline{F}ile \underline{E}dit \underline{V}iew \underline{S}elect \underline{E}\underline{x}plore Sim\underline{u}lation \underline{W}indows$	<u>H</u> elp		cādence
🏪 📷 🖉 🖎 🖾 🛍 🗶 🥒) 🗳 - 🚴 +	Send To: 🕵 🎇 🗟 🗱 📗	
⊡ № TimeA • = 0 • ps • № • [earch Times: Value 🕶 🗾 🏨 🏨		
▶ -			
Design Browser 🛛 🗙 🕥	Objects Methods		
Scope: 🚳 All Available Data 🔄 🛞 🔯	Name	o- Value	Ö.
E LATS	GND 0 N 20	x	L L
B		x x	
		x	
		x	
	N_25	x	
	<u> </u>	x	
	N_27	x	
		x	
		x	
PADINOUT		x	
PADOUT	□ □ ··· ·· ·· ·· ·· ·· ·· ·· ·· ·· ·· ··	'h xx	
PADOUTNOE	down_not_up	x	
- O PWR		x	
SINK		x	
SOURCE0	$= \frac{1}{2} g_{1}Z$	x	
SOURCE1	\sim	x	
⊕ O SYNC_N		x	
⊡— 🔲 Ifsr_updown_cnt_tb	$\neg \neg $	x	
🗄 🗐 DUT 🚽	\square $un5_cnt_1$	x	
Find: String	un11_cnt_0_Z	x	-
	4		⊫
Show contents: In the signal list area	🔁 🗈 🖾 🚾 📾 🕬 Filter:		
Ø 🌫 🗖		1 ok	ject selecte

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.

UG072

Step 3 - Run Simulation

UG072

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 32) (Steps 2 to 5) for loading the waveform and viewing the results.



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

```
<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 7 • 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:

-			
4		Questa Sim-64 10.7c_1	_ 0
le <u>E</u> dit <u>V</u> iew <u>C</u> omp	ile <u>S</u> imulate A <u>d</u> d L <u>i</u> brar	rry T <u>o</u> ols Layo <u>u</u> t Boo <u>k</u> marks <u>W</u> indow <u>H</u> elp	
🖹 - 🚅 🔛 🛸 🍈	Design Optimization	👫 🗄 🛛 Help 💦 🦓 🤣 🎇 🎇 🎇 🕈 🔹 🛊 👔 🍰 🗸 August NoDesign	-
ColumnLayout AllColu	<u>Start Simulation</u>	······································	
	Runtime Options		
Library 🚞	Run		
Name	Step		
work	Restart	katiepurcell/work/simulatio	
floatfixlib		H//floatfixlib	
ieee_env (empty)	<u>B</u> reak	;H//inee_env ;H//infact	
mc2 lib	End Simulation	lib	
mgc ams (empty)		. TECH//mgc ams	
mtiAvm		_TECH//avm	
ntiOvm	Library \$MODEL		
👖 mtiPA (empty)		_TECH//pa_lib	
ntiRnm		_TECH//rnm	
mtiUPF		_TECH/./upf_lib	
ntiUvm Nosvvm		_TECH/./uvm-1.1d	
sv std		_TECH/./losvvm . TECH/./lsv std	
vh_ux01v_lib		. TECH/sv_su	
vhdlopt lib		. TECH/hdlopt lib	
vital2000			
ieee	Library \$MODEL_	TECH//ieee	
modelsim_lib	Library \$MODEL	_TECH//modelsim_lib	
std		_TECH//std	
std_developerskit		_TECH//std_developerskit	
synopsys		_TECH/./synopsys	
verilog	Library \$MODEL_	_TECH//verilog	
Transcript			+
eading pref.tcl			
/ Questa Sim-64			
	inux_x86_64 Oct 14 2018		
	18 Mentor Graphics Corpor	ration	
// Copyright 1991-20 // All Rights Reserv	ed.		
// Copyright 1991-20 // All Rights Reserv //		n contain trada	
// Copyright 1991-20 // All Rights Reserv // // QuestaSim and its	associated documentation	n contain trade mation that are the property of	
<pre>// Copyright 1991-20 // All Rights Reserv // // QuestaSim and its // secrets and comme // Mentor Graphics C</pre>	associated documentation rcial or financial inform corporation and are privil	mation that are the property of leged, confidential,	
<pre>// Copyright 1991-20 // All Rights Reserv // QuestaSim and its // secrets and comme // Mentor Graphics C // and exempt from d</pre>	associated documentation rcial or financial inform corporation and are privil lisclosure under the Freed	mation that are the property of leged, confidential, dom of Information Act,	
<pre>// Copyright 1991-20 // All Rights Reserv // // QuestaSim and its // secrets and comme // Mentor Graphics C // and exempt from d // 5 U.S.C. Section // is prohibited from</pre>	associated documentation rcial or financial inform corporation and are privil lisclosure under the Freed 552. Furthermore, this in m disclosure under the Tr	mation that are the property of leged, confidential, dom of Information Act, nformation	
<pre>// Copyright 1991-20 // All Rights Reserv // QuestaSim and its // secrets and comme // Mentor Graphics C // and exempt from d // 5 U.S.C. Section // is prohibited frc // 18 U.S.C. Section</pre>	associated documentation rcial or financial inform corporation and are privil lisclosure under the Freed 552. Furthermore, this in m disclosure under the Tr	mation that are the property of leged, confidential, dom of Information Act, nformation	
<pre>// Copyright 1991-20 // All Rights Reserv // QuestaSim and its // secrets and comme // Mentor Graphics C // and exempt from d // 5 U.S.C. Section // is prohibited frc // 18 U.S.C. Section</pre>	associated documentation rcial or financial inform corporation and are privil lisclosure under the Freed 552. Furthermore, this in m disclosure under the Tr	mation that are the property of leged, confidential, dom of Information Act, nformation	
<pre>// All Rights Reserv // QuestaSim and its // secrets and comme // Mentor Graphics C // and exempt from d // 5 U.S.C. Section</pre>	associated documentation rcial or financial inform corporation and are privil lisclosure under the Freed 552. Furthermore, this in m disclosure under the Tr	mation that are the property of leged, confidential, dom of Information Act, nformation	
<pre>// Copyright 1991-20 // All Rights Reserv // QuestaSim and its // secrets and comme // Mentor Graphics C // and exempt from d // 5 U.S.C. Section // is prohibited from // 18 U.S.C. Section //</pre>	associated documentation rcial or financial inform corporation and are privil lisclosure under the Freed 552. Furthermore, this in m disclosure under the Tr	mation that are the property of leged, confidential, dom of Information Act, nformation	

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**:

Design VHDL Verilog	Libraries	SDF Others	
Name LATNES LATNR LATNS LATR LATS Ifsr_updown_cnt Ifsr_updown_cnt_tb LMUX2 LRAM LRAM I RAM2K_FIFO	Type Module Module Module Module Module Module Module Module	Path /share/general/Softwa /share/general/Softwa /share/general/Softwa /share/general/Softwa /share/general/Softwa /mnt/scratch1/katiepur /mnt/scratch1/katiepur /share/general/Softwa /share/general/Softwa /share/general/Softwa	re// re// re// re// cel cel re// re//
Design Unit(s) work.lfsr_updown_cnt_tb Optimization Enable optimization		Optimization Option	

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**.



Figure 27 • Selecting Signals to Observe

Step 7 – Run the Simulation

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



Figure 28 • Starting the Simulation Run

Step 8 - View the Waveform

Simulation results are written to the waveform viewer for review:

Questa Sim-64 10.7c_1 Ele Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help Image: Section Control of Co	_ = ×
□ □ □ □ □ □ □ □ □ □	
🕇 🏕 🛊 🏦 - 🏤 - 🏦 Layout Simulate 👻 ColumnLayout AllColumns 💌 🤻 - 🖏 - 🧐 - 🥵 - 🥵 1 💿 🗊 💷 🌽 🛄 🕮 🖿	
🛃 上 土 王 王 王 王 王 🐎 + ★ + 孙 Search: 📃 🖉 勇 義 恭 🔍 🍳 🍳 🤽 🎝 🛄 🔲 🔳 📲 🎬 『 J 丁 💥 👯 📓 🦄	
Image: Sime of the second s	+ - ×
 → period 32100 Para. → period 32100 Para. → cik 1h1 Regi. → cik 1h1 Regi. → intritaL#13 lfs_updown → fiNITIAL#11 lfs_updown → finitial → n_cnt_tb/DUT/cik_in intial 	
Now 169000 ps 50000 ps 100000 ps 150000 ps Image: Second	
	= + ♂ ×
<pre># Time: 169 ns Iteration: 1 Instance: /lfsr_updown_cnt_tb #1 #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></pre>	
Now: 169 ns Delta: 1 sim:/ifsr_updown_cnt_tb/#INITIAL#41 0 ps to 181504 ps	M

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 Symplify under <project_dir>/src/syn, include <ACE_INSTALL_DIR>/libraries/ device_models/<DEVICE>_symplify.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 43) 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 43) above.

(i) 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 8 • VCS Command Options

Options	Required	Description						
-Ica	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.
-1	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:



Figure 30 • DVE Main Window

Step 3 – Open the Simulation Database

Open the database file

lfsr_updown_cnt_tb_sim.vpd file:

<u>6</u>	DVE - TopLevel.1 - [Source.1]	_ 0 ×
		_ 8 ×
0 ×1s ▼ # #	∅∅∅∅∪ጶጶ∅[▼	
Image: Second secon	Open Database x Look in: steepurcel/Work/simulation_example_design/src/sim/ ** ** ** ## ## ** ## ## Uversion of the steepurcel/Work/simulation_example_design/src/sim/ ** ** ** ## ## ** ## ## Uversion of the steepurcel/Work/simulation_example_design/src/sim/ ** ** ** ## ## ** ** ## Uversion of the steepurcel/Work/simulation_example_design/src/sim/ ** ** ** ## ** ** ## Uversion of the steepurcel/Work/simulation_example_design/src/sim/ ** ** ** ## ** ** ## Uversion of the steepurcel/Work/simulation_example_design/src/sim/ ** ** ** ## ** ** ## Uversion of the steepurcel/Work/simulation_example_design/src/sim/ ** ** ** ## ** ** ## Uversion of the steepurcel/Work/simulation_example_design/src/sim/ ** ** ** ## ** ** ## Uversion of the steepurcel/Work/simulation_example_design/src/sim/ ** ** ** ** ## ** ** ## Uversion of the steepurcel/Work/simulation_example_design/src/sim/ ** ** ** ** ## ** ** ## If the game Open Cancel Designator: V1 ** ** ** ** ** ** ** ** ** ** ** ** **	
		V
		Reuse
Ippe: III ▼ Seventy: III ▼ Code: All		E
		×
Log AHistory ASearch Files AOpen		
dve>		영영방문교

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** \rightarrow **New Wave View**:



Figure 32 • Adding Signals to the Waveform

Step 5 - View the Simulation Results

Waveforms can be viewed and analyzed as shown below:

<u> </u>	e overnow	S 11191	-11																												
٠							VE - TO		vel.2	- [W	ave.1	.] /m	nnt/so	ratc	h1/	/src/s	im/lfs	r_up	down	_tb_s	im.v	bd								_ □	×
🗱 E	ile <u>E</u> dit <u>V</u> iew		-	<u>S</u> cope	Trace	e <u>₩</u> ir																								_ 6	키 ×
	0 :	x1ps 🔻 🗍 🞢					7	, en e	ħ., ∐ B	• 🕨	F 4	8 🛛 🖻	1 🖆	*			⇒ >>			56° - 1	1: <i>a</i>	FFP 🔻	€	🖡 An	y Edge		V	1			
	€, ⊖, ⊕ [@) 2 & 0		00			7	liii	🛛 🖬	Ì																					
н	-	리다 🕶 🧱	•																												
Nam	ne	Value			500	00			6000	0	1.		1700	00			, 800				9000		1.		10000	00	1		110000)	<u> </u>
ē-	DUT																														
	∎clk	5	StC				$\Box \Box$						ЛЛ				ЛГ					\Box		ЛЛ						ЛП	
	⊳ -reset	5	St1																												
	- D- enable	5	StC																												
		5	StC																	[
		8'r	100	00 X	01 X	02)5 Oa	15	2b !	57 <mark>/</mark> a	f <mark> </mark> 5e	bc	78 X	fi (e3 (ci	6 <mark> </mark> 8c	18	30 6	1 c3	87	Of X	37 (c3	61	<mark>)</mark> 30	18 8	с (сб	i e3	fi (78 🕻 bc	5e X af	
		:	st																												
L	New Group																														I
·																															
				0			, 2000	0 		. 40	000	1		60000			80000	_		100)00 		120	0000	L	¹ *	40000			60000	
4		1)	• (\$\$
X	Wave.1 \times																														
																								9	26		N/A	N/A	B	• 📅 💽 🖌	
				_																											

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).

(i) 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 9 · 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 10 • 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:

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.



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
// -----
```

61

```
UG072
```

```
// 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;
                           my_dc0_1_awready;
logic
. . . . .
                           my_dc0_2_clk;
logic
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;
```

(i) 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 11 • 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 12 • 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

(i) 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.

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_D CI	_	-

Table 13 · DSM Direct-Connect Interfaces

Subsystem	Interface Name	Physical Location ⁽¹⁾	GDDR6 Channel	SystemVerilog Interface Type	Data Width	Address Width
Ethernet	ethernet_1_dc	North East	_	t_ACX_ETHERNET_D CI	-	-
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.

(i) 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_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.
- 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.

(i) 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
```

```
£ _____
```

UG072

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);
```

(i) 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 14 · Clock Frequency Interfaces

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

Subsystem	Interface Name	Physical Location ⁽¹⁾	GDDR6 Channel
	gddr6_1_noc1_clk ⁽³⁾	West 1 NoC	1
	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
0004	ddr4_noc0_clk	South NoC	_
DDR4	ddr4_dci0_clk	South DCI	_
DDR5	ddr5_noc0_clk ⁽⁴⁾	South NoC	_
201	pciex16_clk ⁽³⁾	Gen5 PCle ×16	-
PCIe	pciex16_dc_clk	Gen5 PCIe ×16 DCI	_

Subsystem	Interface Name	Physical Location ⁽¹⁾	GDDR6 Channel
	pciex8_clk	Gen5 PCle ×8	-
	ethernet_ref_clk ⁽³⁾	Ethernet reference clock ⁽²⁾	-
Ethernet	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 15 · Configuration Subsystem Names

Subsystem ⁽⁴⁾	Interface Subsystem Name ⁽¹⁾	Physical Location ⁽³⁾
	gddr6_0	West 0
	gddr6_1	West 1
	gddr6_2	West 2
GDDR6	gddr6_3	West 3
GDDRo	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
Ethernet	ethernet1	North
GPIO North	gpio_n	North
GPIO South	gpio_s	South
PCle ×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 000000000
r 0000014 00000000
# Verify
v 00005c0 76543210
v 0000014 00004064
# Wait for 50 cycles
d 0000000 0000032
```

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)⁴ 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.

⁴ 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,
```

1.7

```
_____
```

UG072

```
output awaddr, awlen, awid, awqos, awburst, awlock, awsize,
awvalid, awregion,
                             bready, wdata, wlast, rready, wstrb, wvalid,
                             araddr, arlen, arid, argos, 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, awgos, 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

(i) 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