Simulation User Guide (UG072)

All Achronix Devices

Achronix®
Data Acceleration

Copyrights, Trademarks and Disclaimers

Achronix Semiconductor Corporation

2903 Bunker Hill Lane
Santa Clara, CA 95054
USA

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

Table of Contents

Chapter - 1: 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-Routed 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:

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

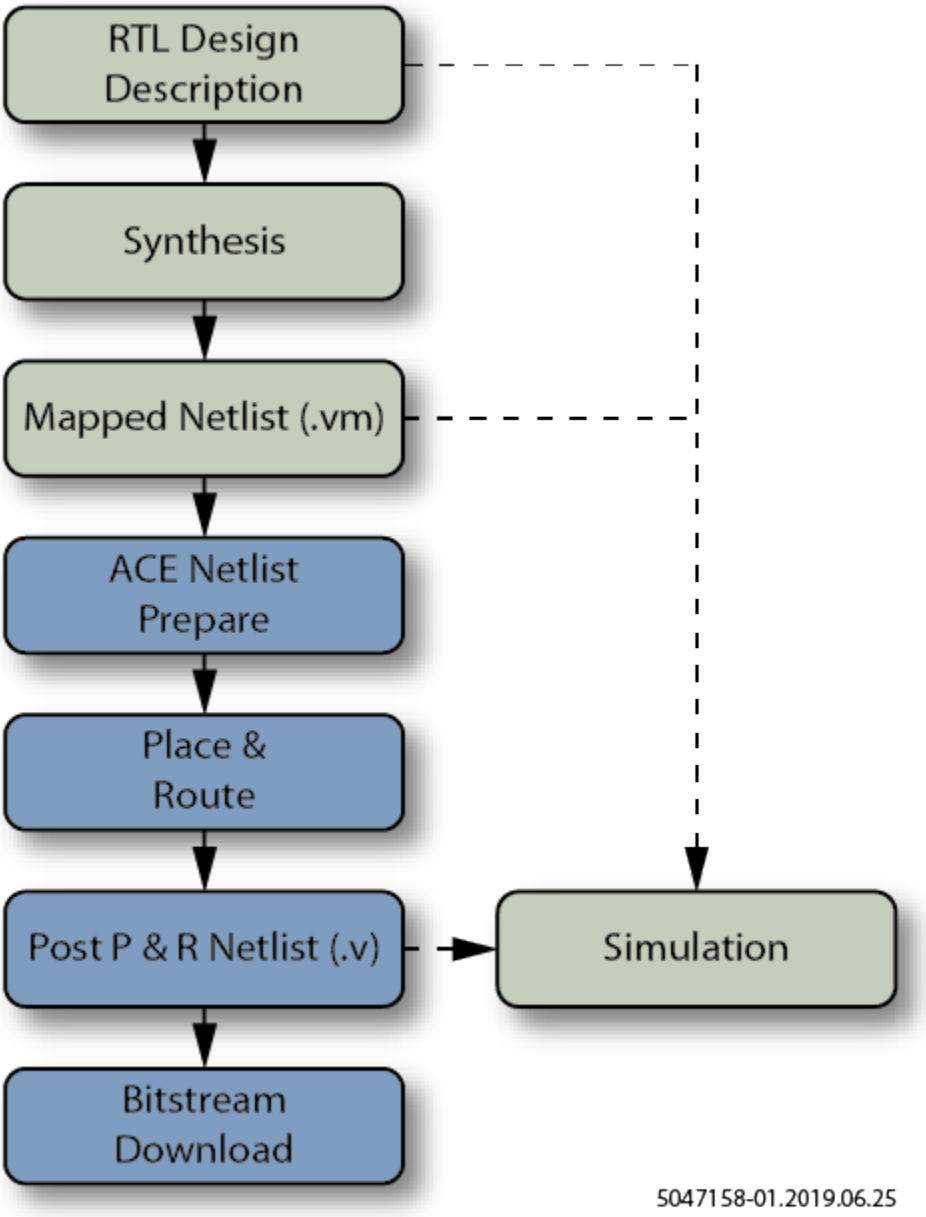

Figure 1: ***Simulation Flow***

Chapter - 2: Simulation Libraries

This guide covers simulation for all Achronix devices. It is incumbent on the user to select the correct technology library. The text in this user guide contains references to `<technology>`. The user should simply replace this with the abbreviated version of the technology name as specified in the table below, e.g., `22i`.

Table 1: ***Achronix Simulation Libraries***

Technology	Abbreviation	Simulation Model File Name	Device Families
Speedster22i	`22i`	`22i_simmodels.v`	Speedster22i FPGAs
Speedster16t	`16t`	`16t_simmodels.v`	Speedcore eFPGAs
Speedster7t	`7t`	`7t_simmodels.v`	Speedster7t FPGAs

Chapter - 3: General Project Setup

User Design Project Directory Structure

The following project directory structure is used in this example:

Directory	Description
<project_dir>	root directory for the user design project
/src	/src
/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 Mentor 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
For Cadence Incisive	
/incisive-rtl	Contains the RTL simulation project area and output
/incisive-gate	Contains the gate-level simulation project area and output
/incisive-final	Contains the post-route (or final) simulation project area and output

Directory	Description
For Synopsis 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

ACE Installation Library Directory Structure

Files used in the ACE installation are located under the ACE install directory as follows:

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

ACE Extension I/O Ring Library Directory Structure (Speedcore Only)

If the user design instantiates ASIC I/O ring logic outside the Speedcore instance, files from the ACE extensions directory must be included from the file structure below:

Directory	Description
<ace_ext_dir>	Directory path to the ACE extensions directory being used (should match the $ACE_EXT_DIR environment variable setting).
/libraries	Root directory for simulation and other libraries for the ASIC I/O ring provided by the ASIC integrator. This is the top-level library include directory (+incdir+) for ASIC I/O ring libraries.
/sim	Contains the technology-specific top-level ASIC I/O ring library include file for simulation.

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 - 4: 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, Mentor Questa Sim Simulator Example (see page 35) 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/<technology>_simmodels.v`
    c. (Speedcore only) Optionally, the top-level ASIC I/O ring technology-specific simulation library include file, found in `<ace_ext_dir>/libraries/sim/<ioring_library_file>.`
    d. The behavioral RTL (Verilog or VHDL) source files for the user design
    e. The top-level simulation testbench (Verilog or VHDL) files
4. Run the simulation and observe the output waveform.

Chapter - 5: General Gate-Level Simulation Flow

To perform gate-level simulation:

1. Create a synthesis project in the `<project_dir>/src/syn/` directory for Synplify Pro to compile and synthesize the source behavioral RTL files to map to Achronix technology. The output of the synthesis tool (Synplify Pro) is a mapped gate-level Verilog netlist in the format that the ACE tool accepts as input for the back-end place-and-route flow. Synplify Pro outputs the synthesized gate-level netlist with the `*.vm` extension.
2. Then create a `<sim_tool>-gate` directory under `<project_dir>/src/`.
3. Then 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 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, Mentor Questa Sim Simulator Example (see page 35). 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/<technology>_simmodels.v`
    c. (Speedcore only) Optionally, the top-level ASIC I/O ring technology-specific simulation library include file, found in `<ace_ext_dir>/libraries/sim/<ioring_library_file>.v`
    d. The synthesized gate-level Verilog netlist file (`*.vm`) for the user design
    e. The top-level simulation testbench (Verilog or VHDL) files
5. Run the simulation and observe the output waveform

Chapter - 6: General Post-Route Simulation Flow

To perform post-route simulation:

1. Create a project in the `<project_dir>/src/ace/` directory for ACE to place and route the source synthesized gate-level Verilog netlist file (`*.vm` file output by Synplify Pro) for the user design. The user design must be run through the place-and-route flow in the ACE tool, 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 tools flow prior to bitstream generation. ACE outputs the post-route gate-level netlist into the following location: `<project_dir>/src/ace/<active_impl_dir>/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. Then 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 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, Mentor Questa Sim Simulator Example (see page 35). 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/<technology>_simmodels.v`
    c. (Speedcore only) Optionally, the top-level ASIC I/O ring technology-specific simulation library include file, found in `<ace_ext_dir>/libraries/sim/<ioring_library_file>.v`
    d. The encrypted post-route gate-level Verilog netlist file (`*_final.v`) for the user design.
    e. The top-level simulation testbench (Verilog or VHDL) files
5. Run the simulation and observe the output waveform.

Chapter - 7: 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.

lfsr_updown_cnt.v

```
`define CNT_WIDTH 8

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

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

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

       reg [`CNT_WIDTH-1 : 0]    cnt;

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

    always @(posedge clk_in)
    begin
        if (rst)
              cnt <= {`CNT_WIDTH{1'b0}};
        else if (en) begin
               if (down_not_up) begin
                  cnt <= {~(^(cnt & `CNT_WIDTH'b01100011)),cnt[`CNT_WIDTH-1:1]};
               end else begin
                  cnt <= {cnt[`CNT_WIDTH-2:0],~(^(cnt &  `CNT_WIDTH'b10110001))};
               end
        end
      end // always @ (posedge clk_in)

endmodule : lfsr_updown
```

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 Simulator Example regarding a memory initialization file, `rom_file.txt`.

Chapter - 8: 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 `<technology>_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/<technology>_simmodels.v <project_dir>/src/rtl/lfsr_updown_cnt.v <project_dir>/src
/tb/lfsr_updown_cnt_tb.v -sverilog -R -l vcs.log
```

Where `<technology>` is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

Table 2: *Command Options*

Options	Required	Description
-lca	Yes	IEEE encryption flow in VCS requires this option
-debug_pp	No	Creates a VPD file (when used with the VCS system task $vcdpluson) and enables DVE for post-processing a design. Using-debug-pp can save compilation time by eliminating the overhead of compiling with -debug and -debug_all.
-sverilog	Yes	Supports System Verilog features.
-R	No	Run the executable file immediately after VCS links together the executable file. This option is required so that .vpd file is generated which is used for analyzing waveform.
-l	No	Log file name can be specified with this option where VCS records compilation messages. Runtime messages are also included in the log file if -R option is used along with -l.
+vcs+lic+wait	No	Tells VCS to wait for network license if none is available.
+define+CMEM_READBACK	No	This option is to be used only when reading back configuration memory (CMEM) contents during WGL simulations. To reduce simulation time for CMEM readback, use the command line option: `-Xalex=0x10000000`

Step 2 – Start the Simulation GUI

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

```
dve &
```

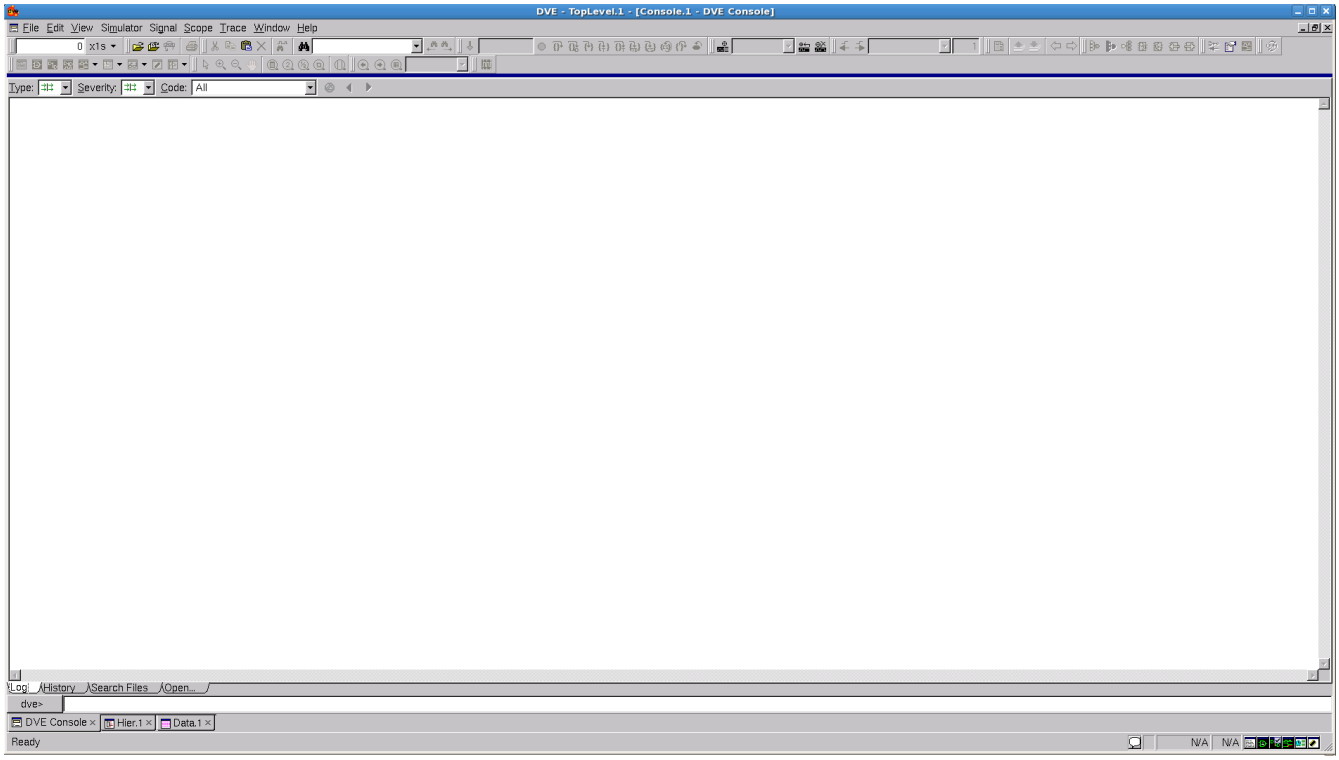

Figure 2: *DVE Main Window*

Step 3 – Open the Simulation Database

Open the database file `lfsr_updown_cnt_tb_sim.vpd` file:

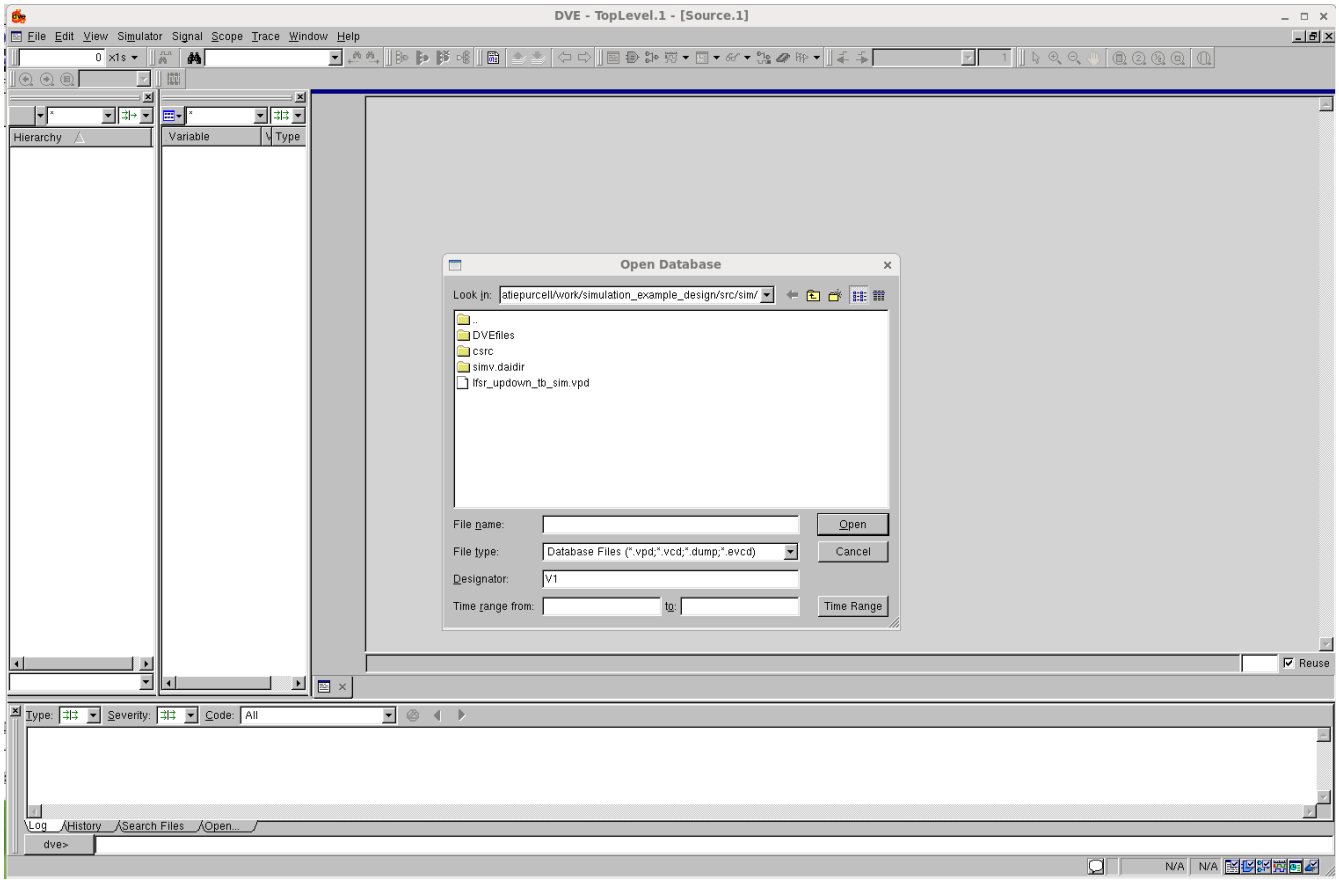

Figure 3: ***Opening the Simulation Database***

Step 4 – Reset the Layout

Click on the Hier tab at the bottom left of the window and reset layout by selecting **Window → Reset Layout**:

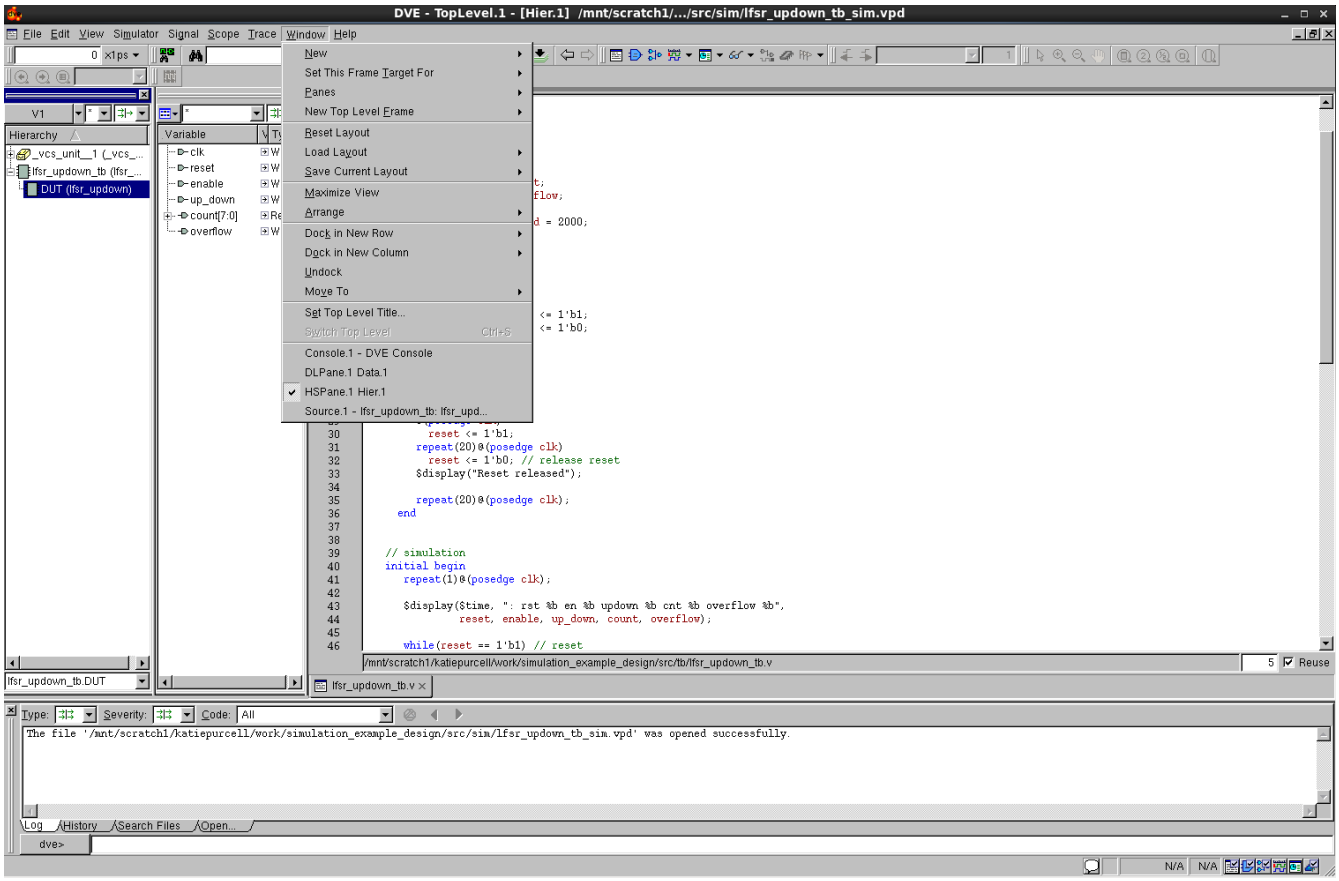

Figure 4: ***Resetting the Layout***

Step 5 – Add Signals to the Waveform

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

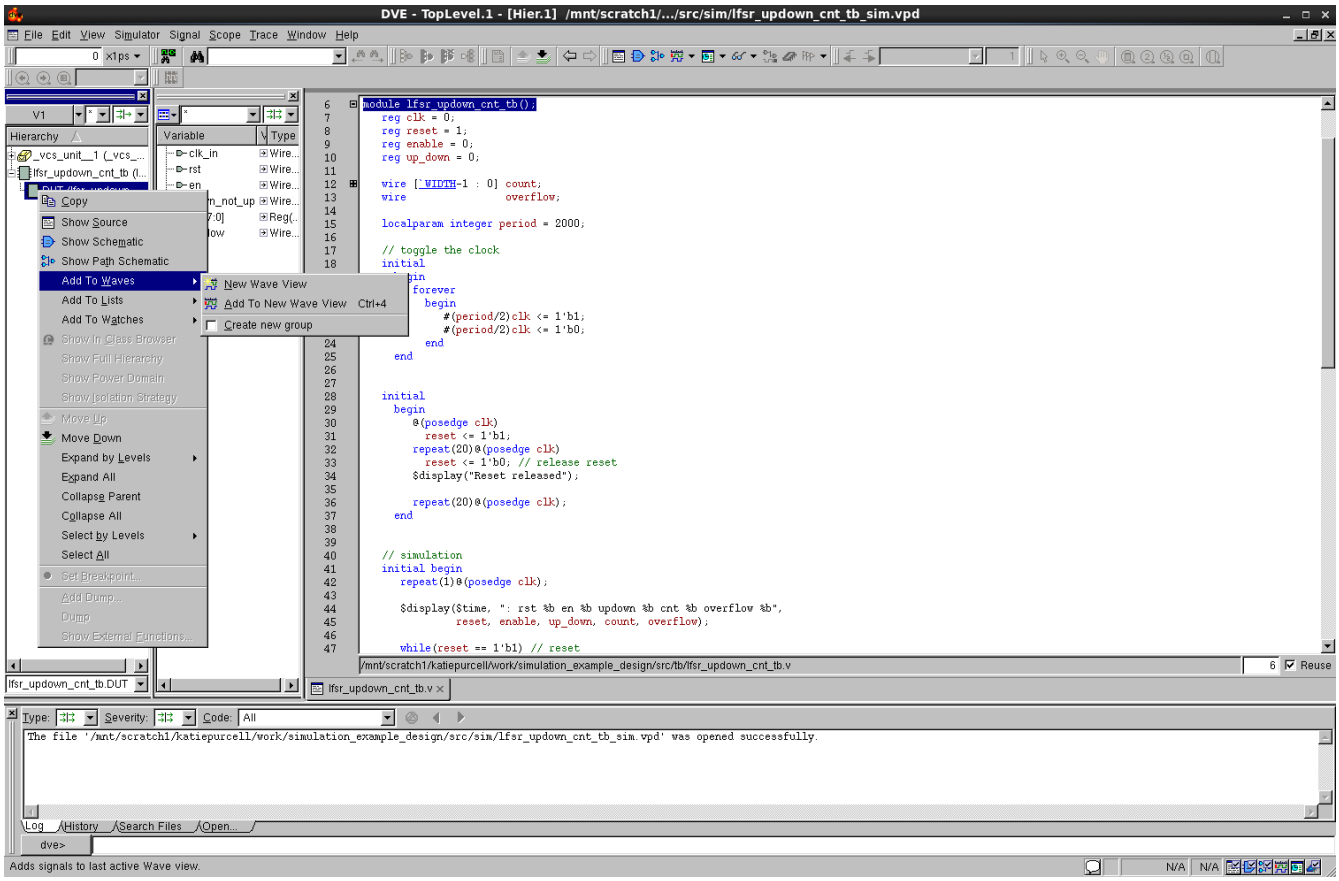

Figure 5: ***Adding Signals to the Waveform***

Step 6 – View the Simulation Results

Waveforms can be viewed and analyzed as shown below:

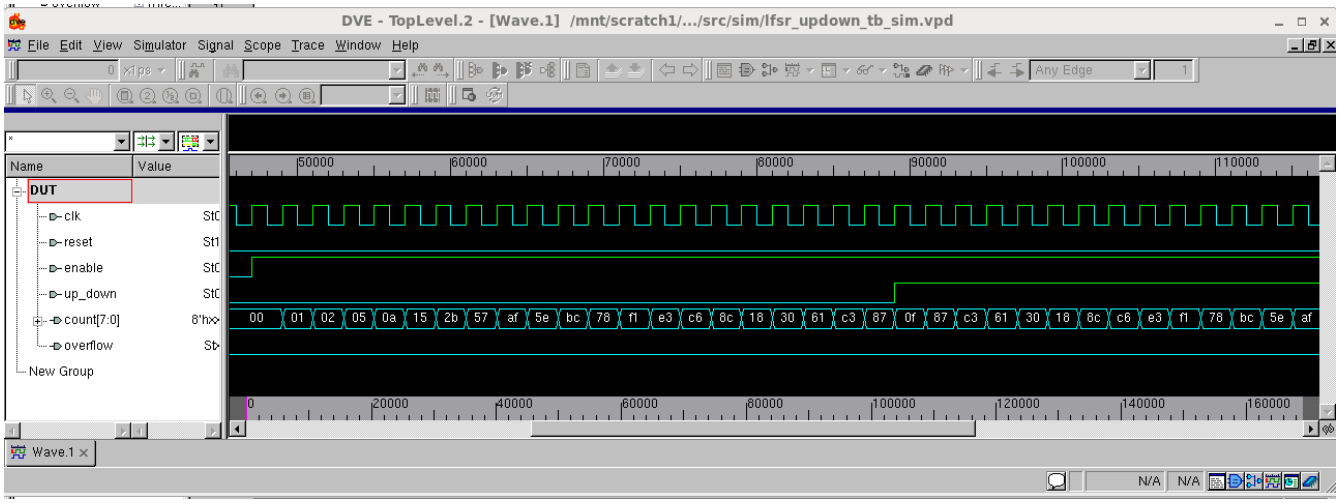

Figure 6: ***Viewing the Waveforms***

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/<technology>_synplify.v followed by the RTL design files and constraint files.

Where <technology> is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

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 (see page 19) 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/<technology>_simmodels.v <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 <technology> is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

Complete the process by following Steps 2 to the end of RTL Simulation in VCS (see page 19).

Post-Route Simulation in VCS

For post-route simulation, the synthesized gate-level neltist 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 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 would be 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/<technology>_simmodels.v <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 `<technology>` is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

Complete the process by following Steps 2 to the end of RTL Simulation in VCS (see page 19).

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 - 9: Cadence Incisive Simulator Example

RTL Simulation in Incisive

> **Note**
>
> Cadence Incisive support for Speedster7t devices will be available soon.

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

Step 1 – Invoke the Incisive Tool

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

```
irun -access +rwc +incdir+<ace_install_dir>/libraries/ /<ace_install_dir>/libraries/device_models
/<technology>_simmodels.v <project_dir>/src/rtl/lfsr_updown_cnt.v <project_dir>/src/tb
/lfsr_updown_cnt_tb.v -gui -sv
```

Where `<technology>` is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

Table 3: *Command Options*

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

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

```
    Building instance specific data structures.
    Loading native compiled code:     ..................... Done
    Design hierarchy summary:
                          Instances  Unique
        Modules:                837     258
        UDPs:                     0       5
        Timing outputs:           4       1
        Registers:             3671     548
        Scalar wires:          8208       -
        Expanded wires:         695      36
        Vectored wires:        1084       -
        Named events:            40       4
        Always blocks:         2821     286
        Initial blocks:         517     111
        Cont. assignments:     2988    1421
        Pseudo assignments:     631     380
        Compilation units:        1       1
        Simulation timescale:   1ps
    Writing initial simulation snapshot:
Loading snapshot worklib.bram_outputs:vp ..................... Done
SVSEED default: 1
ncsim: *W,DSEM2009: This SystemVerilog design is simulated as per IEEE 1800-2009 SystemVerilog
simulation semantics. Use -disable_sem2009 option for turning off SV 2009 simulation semantics.
```

Step 2 – Add Signals to the Waveform

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

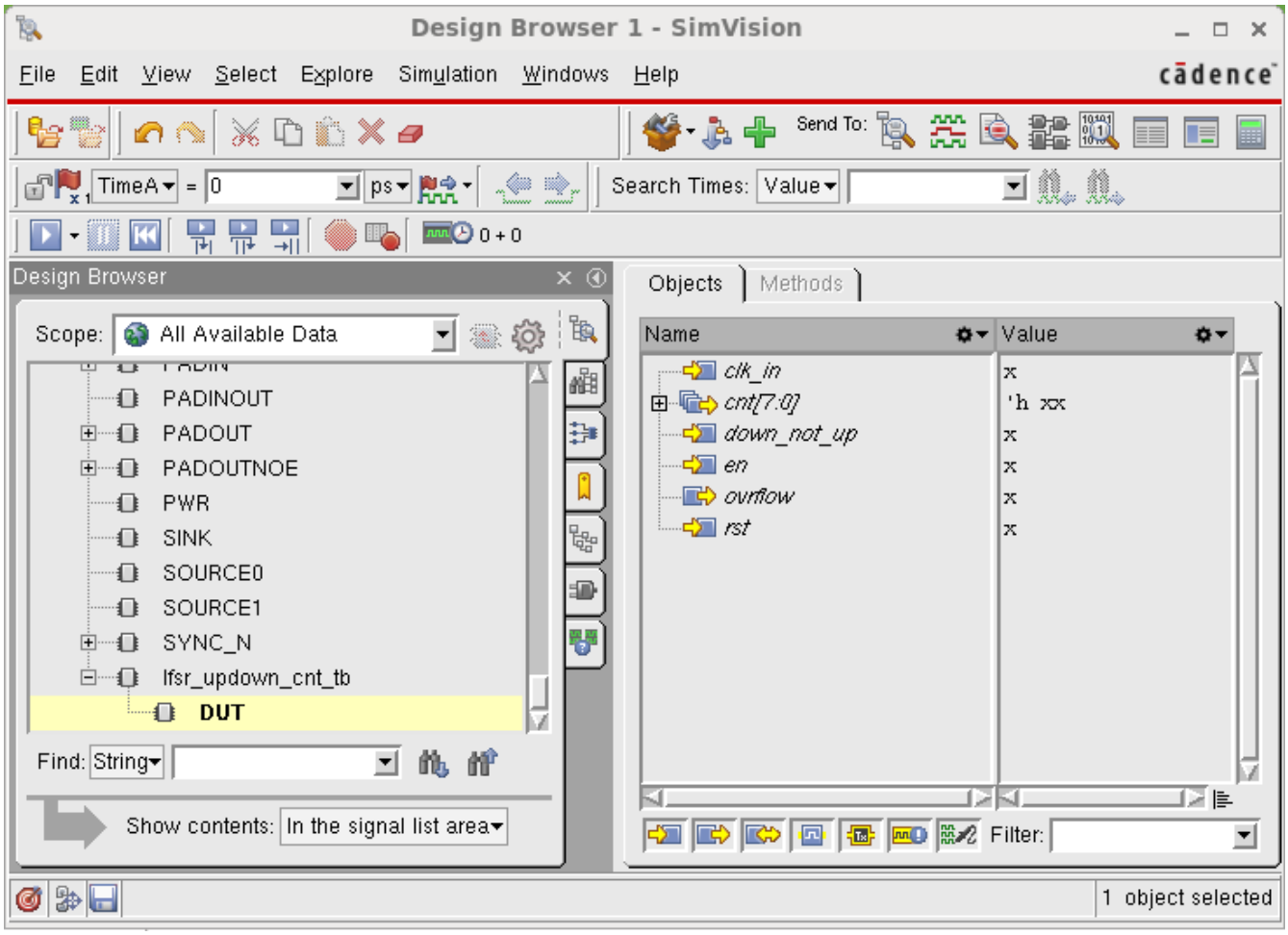

Figure 7: *SimVision Main Window*

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

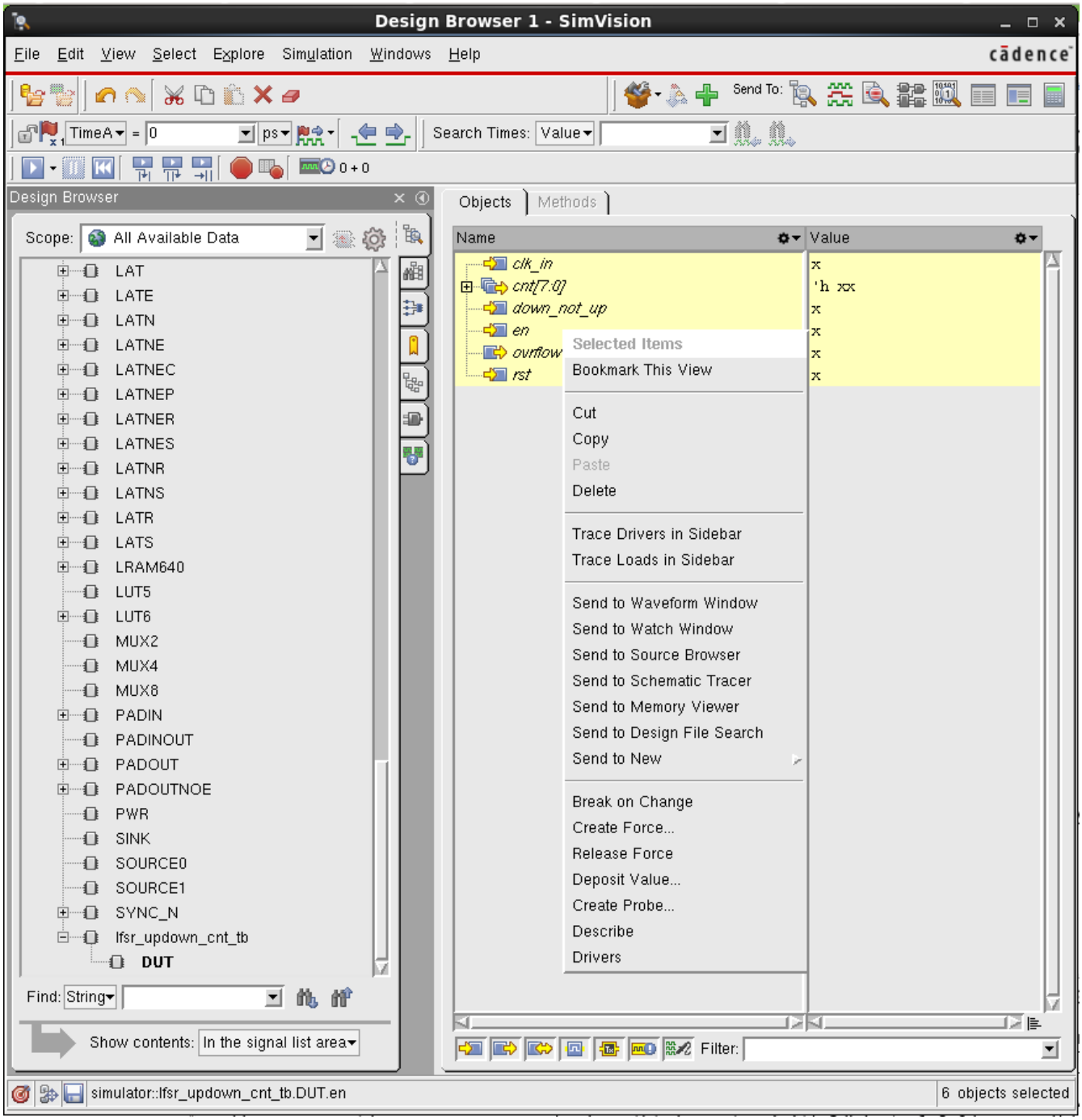

Figure 8: ***Adding Signals to the Waveform***

The Waveform window will open as shown below.

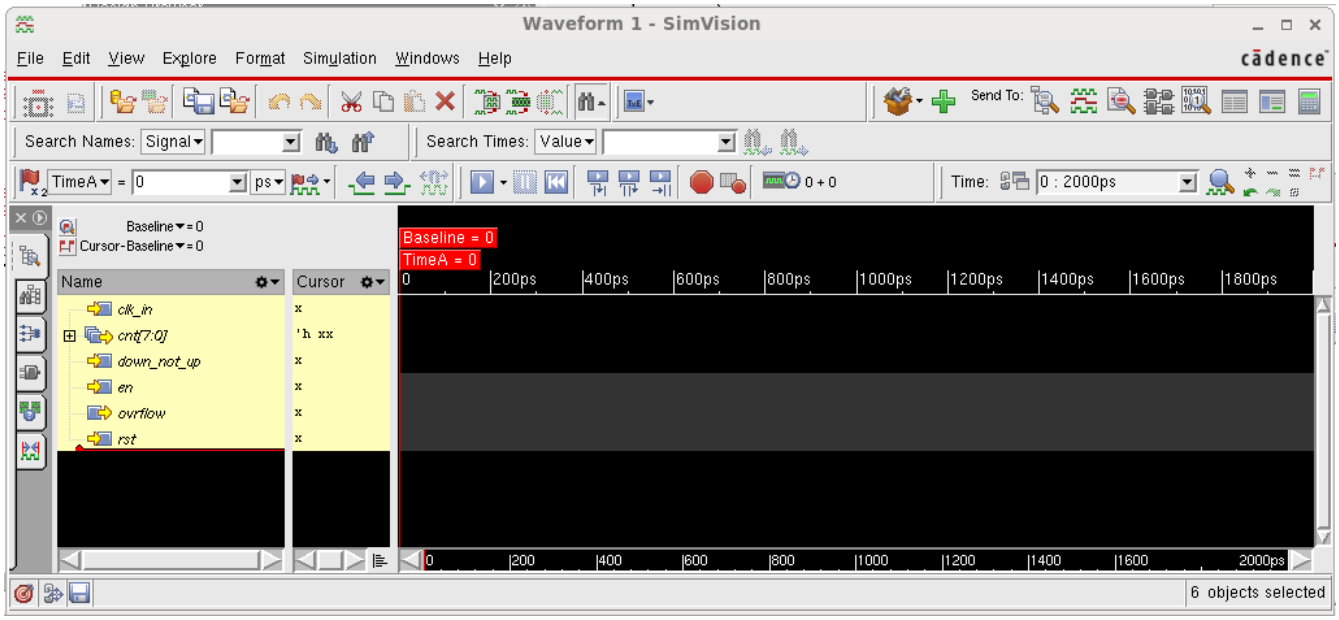

Figure 9: ***Incisive Waveform Window***

Step 3 – Run the Simulation

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

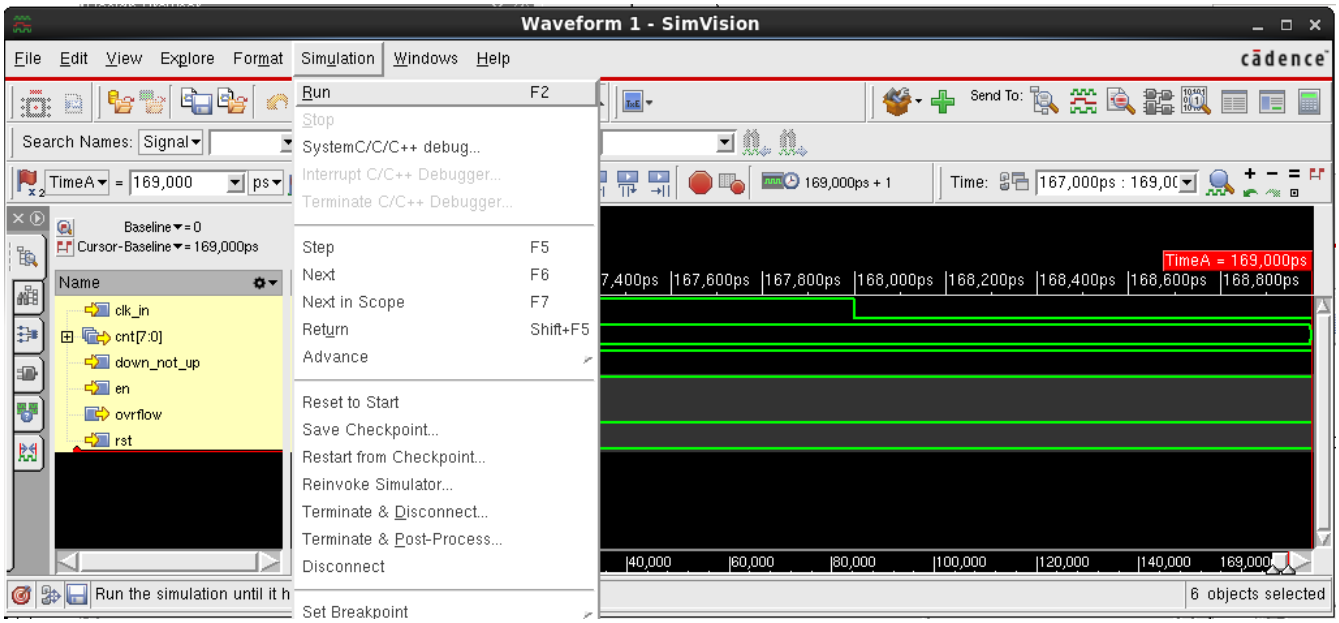

Figure 10: ***Running Simulation***

Step 4 – View the Waveform

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

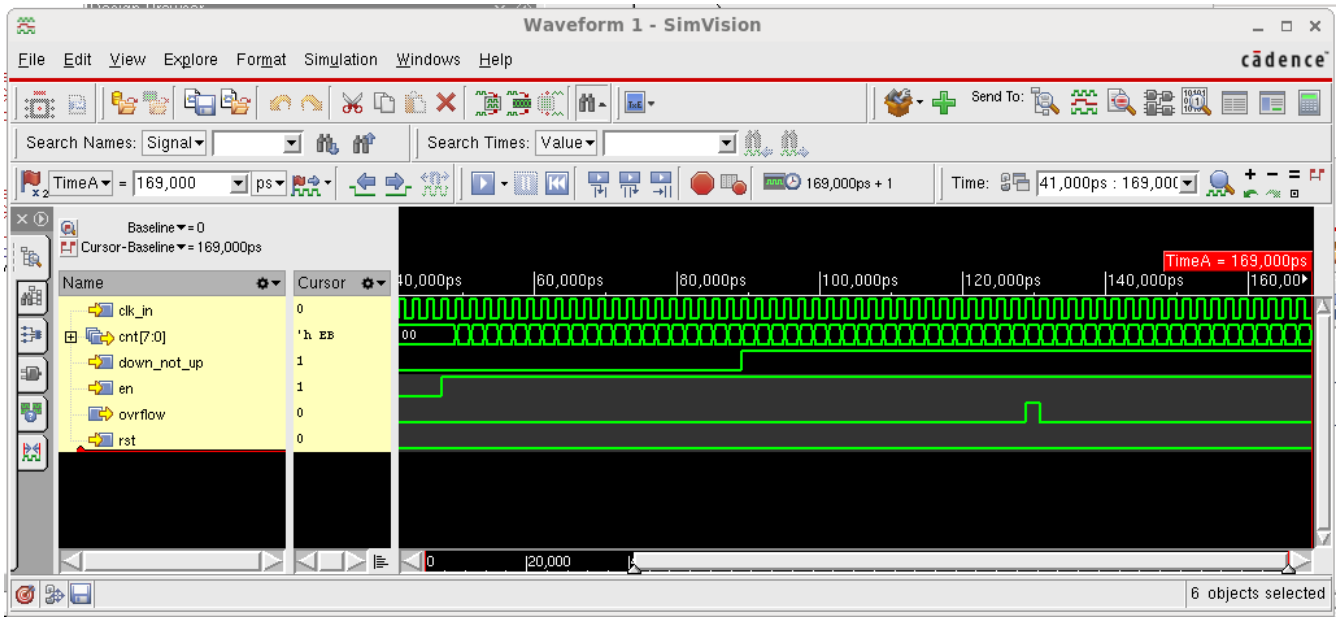

Figure 11: ***SimVision Waveform Window***

Step 5 – View Console Messages

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

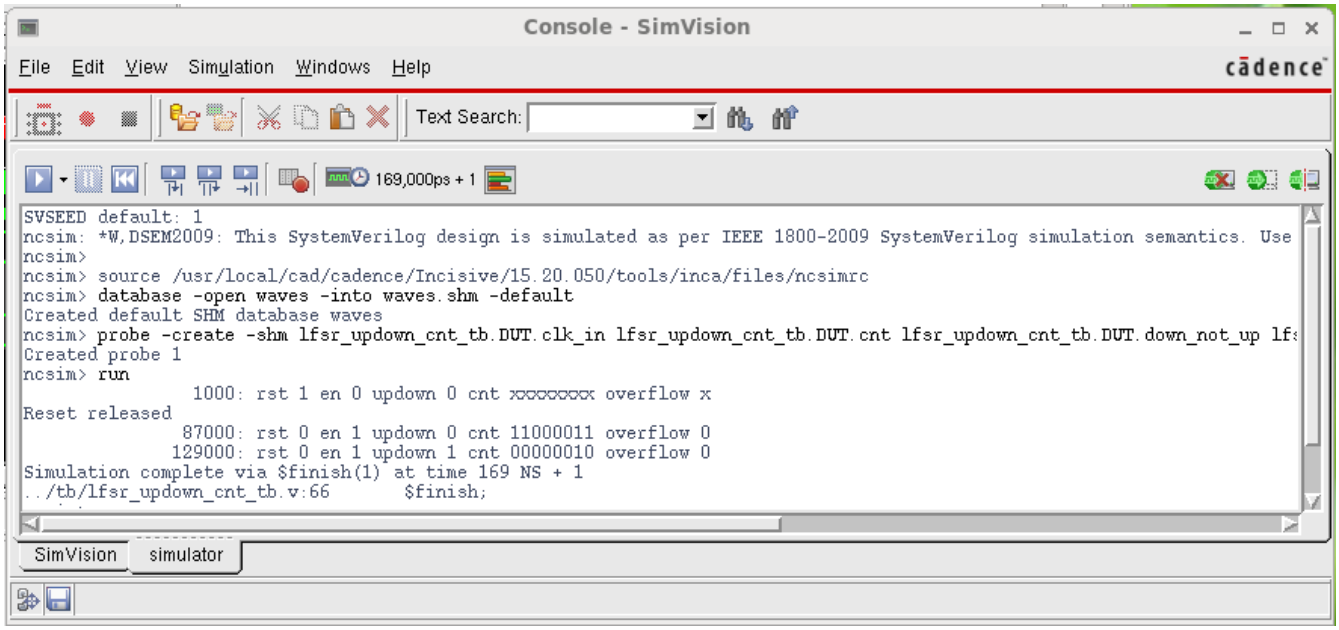

Figure 12: ***SimVision Console Window***

Gate-Level Simulation in Incisive

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/<technology>_synplify.v` followed by the RTL design files and constraint files.

Where `<technology>` is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

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 cadence simulator understands that it is a Verilog file. Otherwise, the cadence simulator tool 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 Incisive (see page 26) section, except that the gate-level simulation uses the mapped netlist `lfsr_updown_cnt_gate.v` file instead of source RTL files.

```
irun -access +rwc +incdir+<ace_install_dir>/libraries/ <ace_install_dir>/libraries/device_models
/<technology>_simmodels.v <project_dir>/src/syn/rev_acx/lfsr_updown_cnt_gate.v <project_dir>/src
/tb/lfsr_updown_cnt_tb.v -gui -sv
```

Where `<technology>` is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

Step 4 – View Simulation Results

Follow the same steps described in RTL Simulation in Incisive (see page 26) (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.

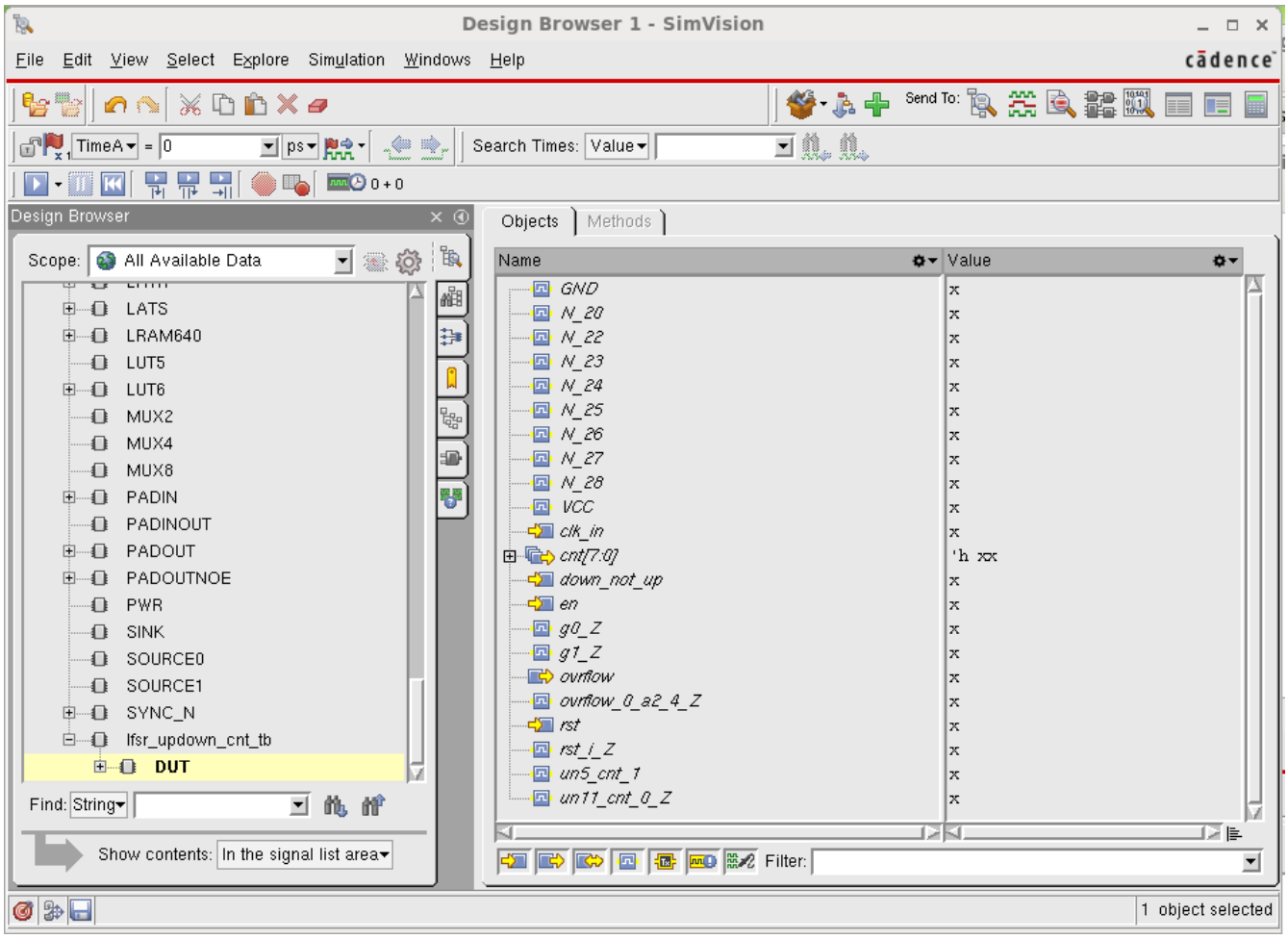

Figure 13: ***Adding Signals to the Waveform***

Post-Route Simulation in Incisive

For post-route simulation, the synthesized gate-level neltist 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 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 would be generated under `<project_dir>/src/ace/impl_1/output/lfsr_updown_cnt_final.v`.

Step 3 – Run Simulation

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

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

Where `<technology>` is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

Step 4 – View Simulation Results

In post-route simulation, when DUT is selected, no signals are displayed since the post-route netlsit file is encrypted. Instead, select the signals under lfsr_updown_cnt_tb and follow the same steps described in RTL Simulation in Incisive (see page 26) (Steps 2 to 5) for loading the waveform and viewing the results.

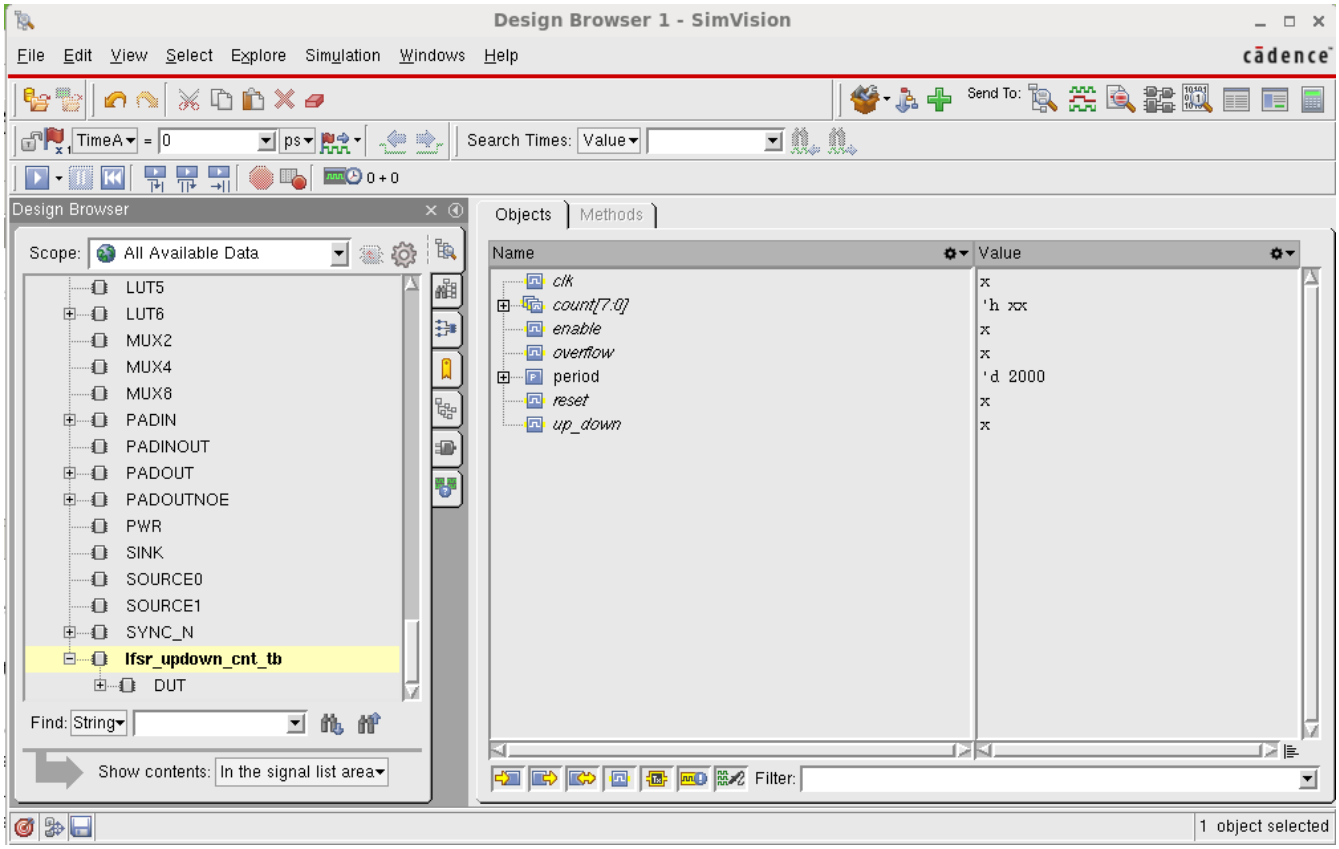

Figure 14: ***Adding Signals to the Waveform***

Chapter - 10: Mentor Questa Sim Simulator Example

RTL Simulation in Questa Sim

Step 1 - Create the Project

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

Step 2 - Initialize the Work Library

Initialize the simulator work library using the following Questa Sim command:

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

Step 3 - Create the File List

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

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

Example vhdl_filelist.f

```
+incdir+<project_dir>/src/tb
+incdir+<project_dir>/src/rtl
+incdir+<ace_install>/libraries
+incdir+<ace_ext_dir>/libraries
<ace_install>/libraries/device_models/<technology>_simmodels.v
<project_dir>/src/rtl/lfsr_updown_cnt.vhd
<project_dir>/src/tb/lfsr_updown_cnt_tb.vhd
```

Where `<technology>` is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

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

Where <technology> is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

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 4: *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:

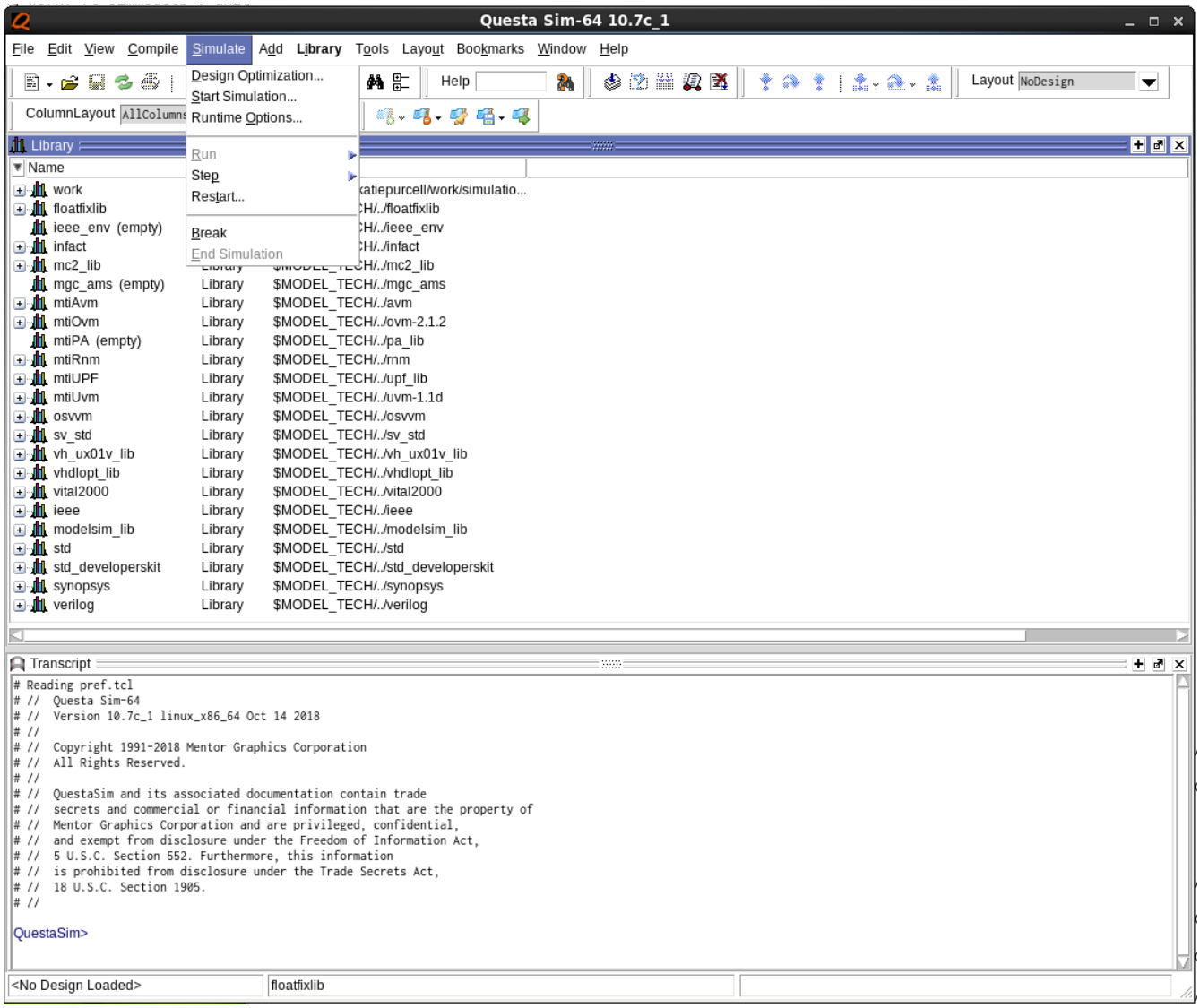

Figure 15: *Questa Sim GUI*

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**:

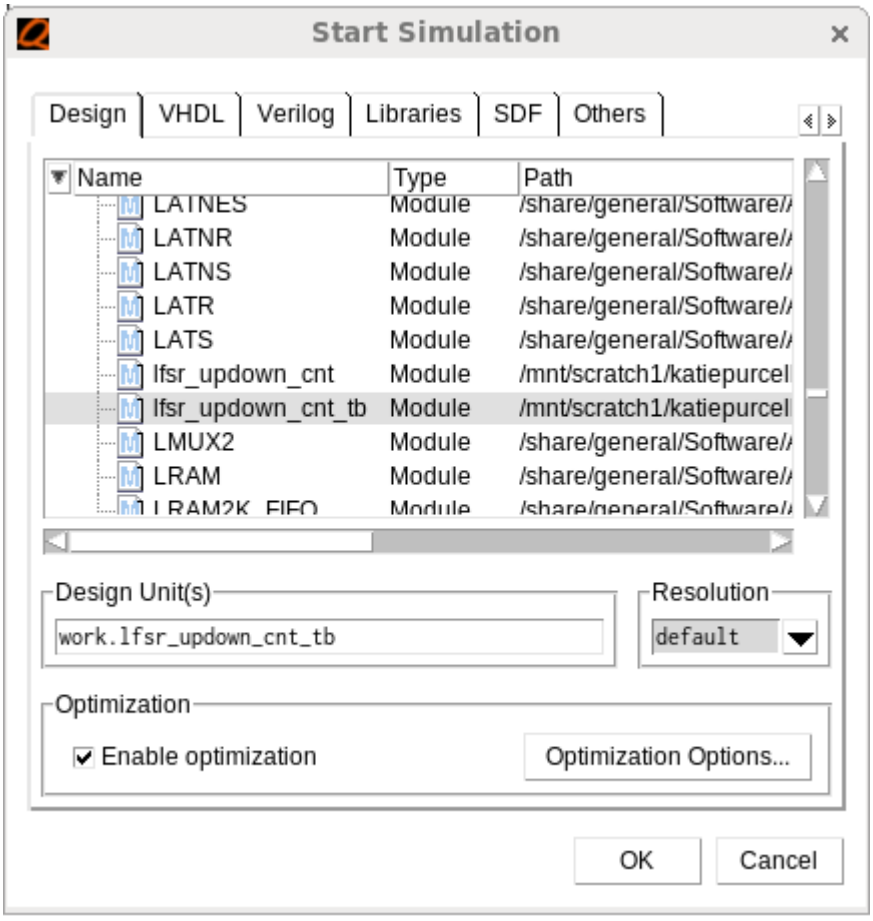

Figure 16: ***Start Simulation Dialog Box***

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**.

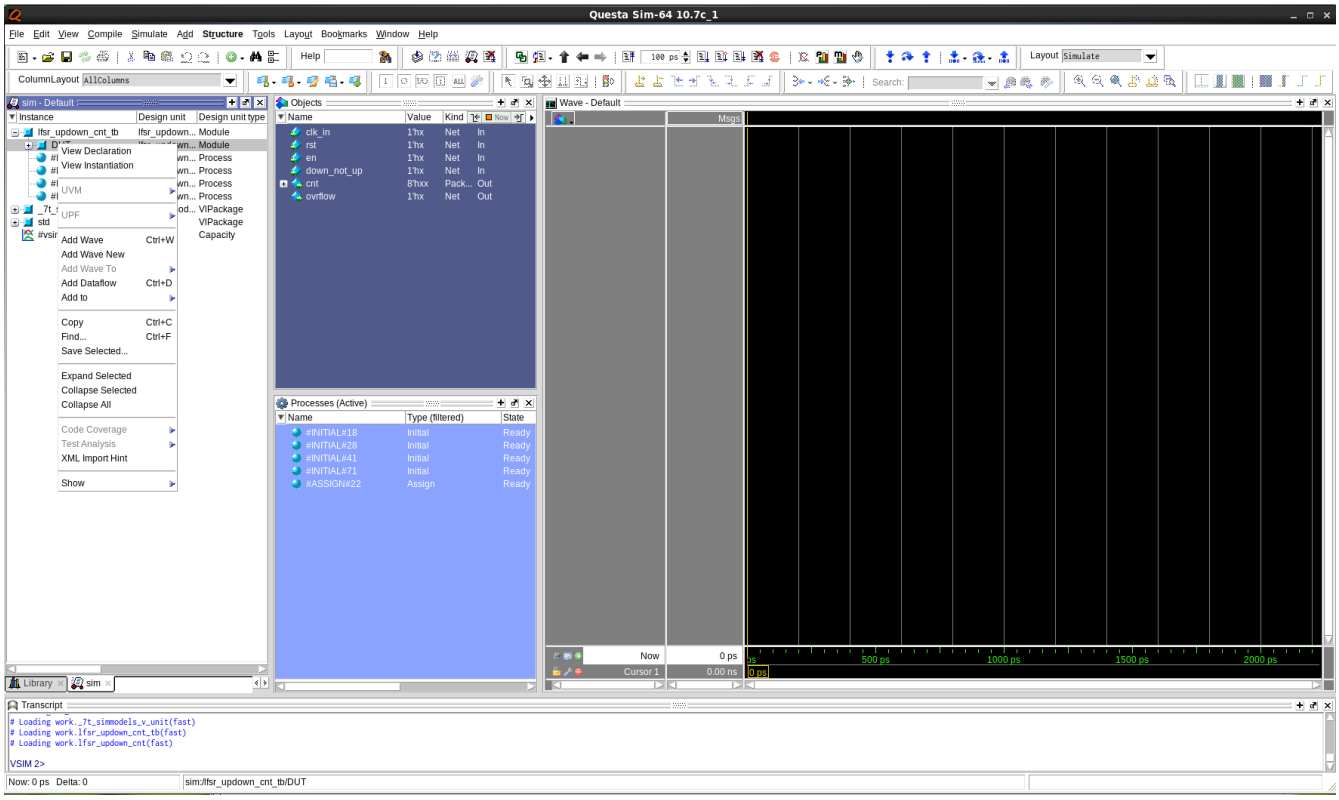

Figure 17: ***Selecting Signals to Observe***

Step 7 – Run the Simulation

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

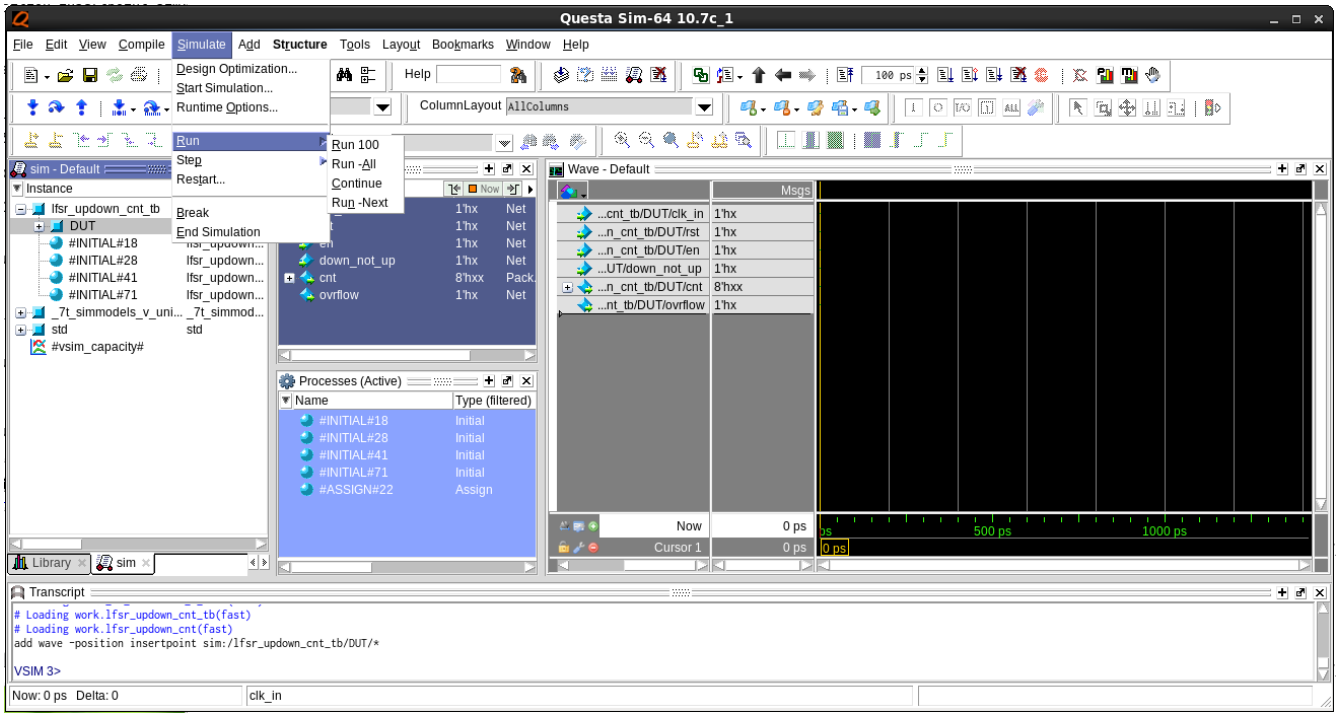

Figure 18: *Starting the Simulation Run*

Step 8 – View the Waveform

Simulation results are written to the waveform viewer for review:

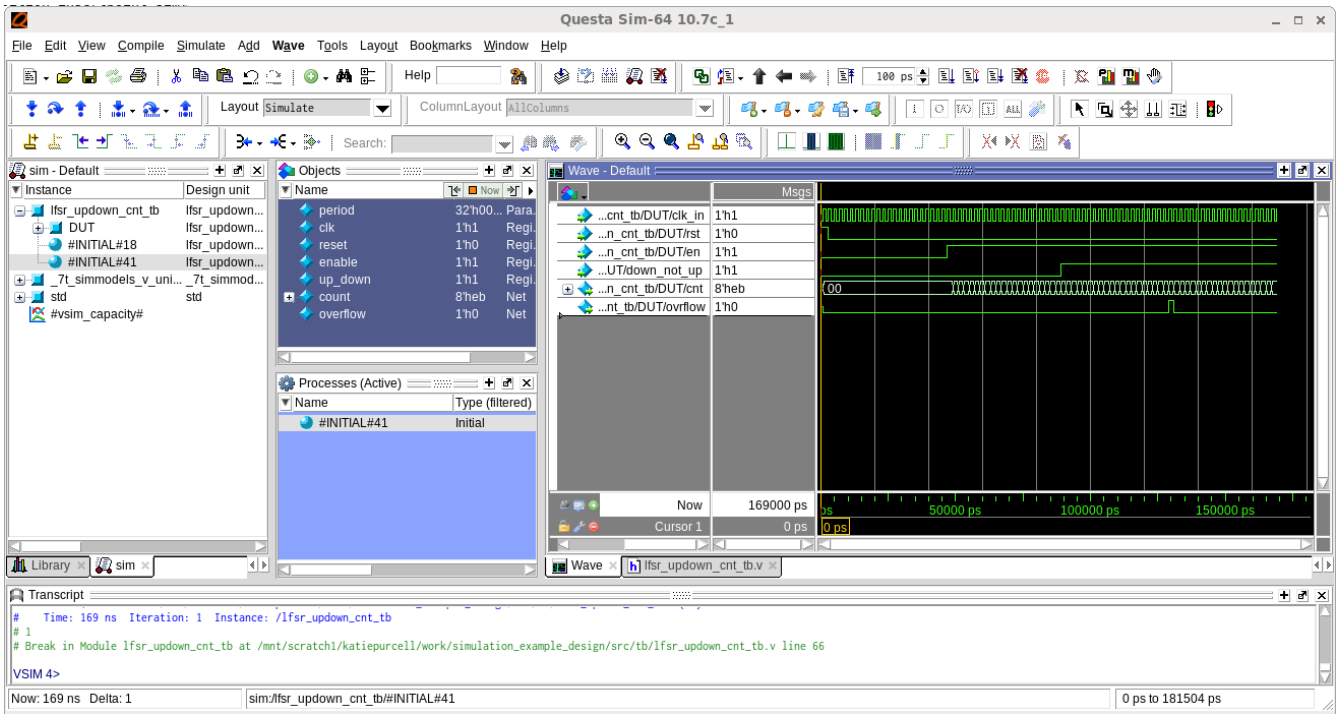

Figure 19: *Questa Sim Waveform Viewer*

Gate-Level Simulation in Questa Sim

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/<technology>_synplify.v` followed by the RTL design files and constraint files.

Where `<technology>` is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

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

Step 4 – Initialize the Work Library

Initialize the simulator work library using the following Questa Sim 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>/libraries
+incdir+<ace_ext_dir>/libraries
<ace_install>/libraries/device_models/<technology>_simmodels.v
<project_dir>/src/syn/rev_acx/lfsr_updown_cnt.vm
<project_dir>/src/tb/lfsr_updown_cnt_tb.v
+libext+.v
```

Where `<technology>` is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

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 Questa Sim (see page 35) above.

Post-Route Simulation in Questa Sim

For post-route simulation, the synthesized gate-level neltist 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 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 would be generated under `<project_dir>/src/ace/impl_1/output/lfsr_updown_cnt_final.v`.

Step 3 – Set up the Simulation Project

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

Step 4 – Initialize the Work Library

Initialize the simulator work library using the following Questa Sim 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>/libraries
+incdir+<ace_ext_dir>/libraries
<ace_install>/libraries/device_models/<technology>_simmodels.v
<project_dir>/src/ace/impl_1/output/lfsr_updown_cnt_final.v
<project_dir>/src/tb/lfsr_updown_cnt_tb.v
+libext+.v
```

Where `<technology>` is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

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 Questa Sim (see page 35) above.

Note

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

Chapter - 11: 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_path>/libraries/device_models/<technology>_simmodels.v

#Compile design
#design.compile test
alog -work lfsr_updown_cnt -incdir {/<ace_install_path>/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/<technology>_simmodels.v}

#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 <technology> is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

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_path>/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 this:

```
sim.log

# ELAB2: Elaboration final pass complete - time: 0.3 [s].
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 7432 kB (elbread=1450 elab2=5844 kernel=137 sdf=0)
# KERNEL: ASDB file was created in location /mnt/scratch1/katiepurcell/work
/simulation_example_design/src/results/lfsr_updown_cnt/dataset.asdb
run -all
# KERNEL: Reset released
# KERNEL:                87000: rst 0 en 1 updown 0 cnt 11000011 overflow 0
# KERNEL:               129000: rst 0 en 1 updown 1 cnt 00000010 overflow 0
# KERNEL:               SIMULATION PASSED!!
# RUNTIME: Info: RUNTIME_0068 lfsr_updown_cnt_tb.v (66): $finish called.
# KERNEL: Time: 169 ns,  Iteration: 1,  Instance: /lfsr_updown_cnt_tb,  Process: @INITIAL#41_2@.
# KERNEL: stopped at time: 169 ns
# VSIM: Simulation has finished. There are no more test vectors to simulate.
workspace.close -save
quit
# VSIM: Simulation has finished.
```

Step 4 – View the Waveform

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

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

Step 5 – Open the Workspace

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

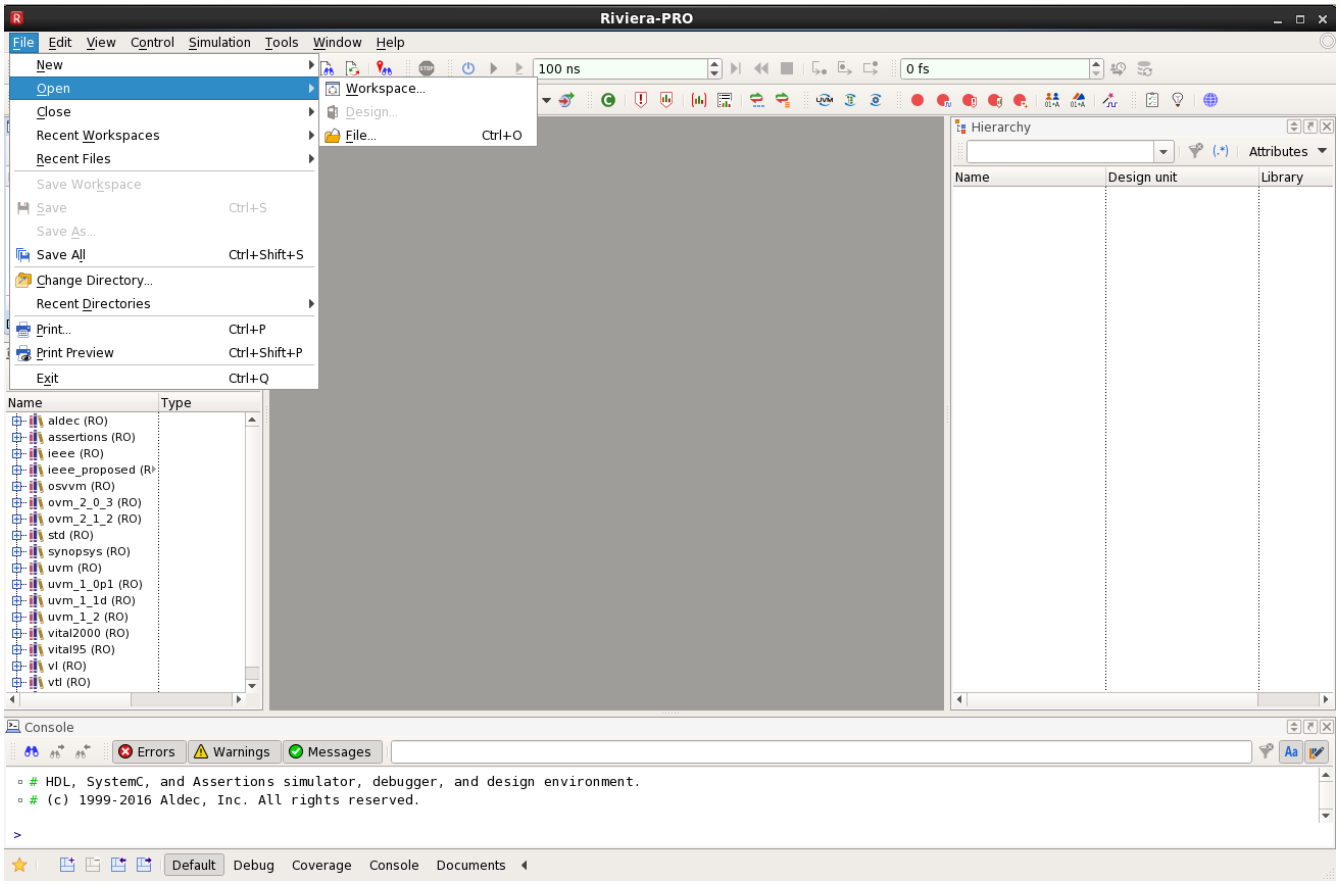

Figure 20: *Riviera Workspace*

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

Step 6 – Initialize the Simulation

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

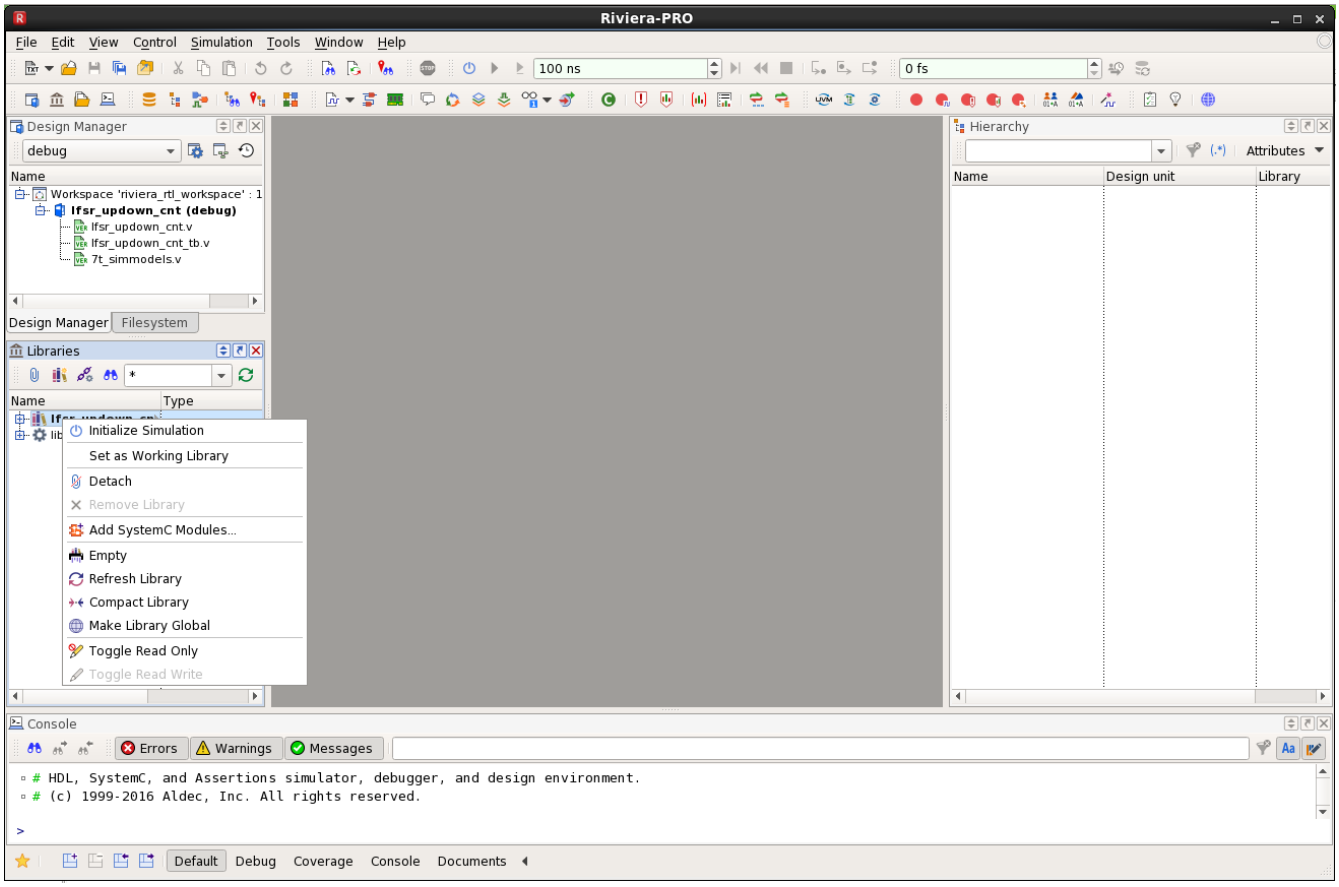

Figure 21: *Initializing Simulation*

Step 7 – Add Signals to the Waveform

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

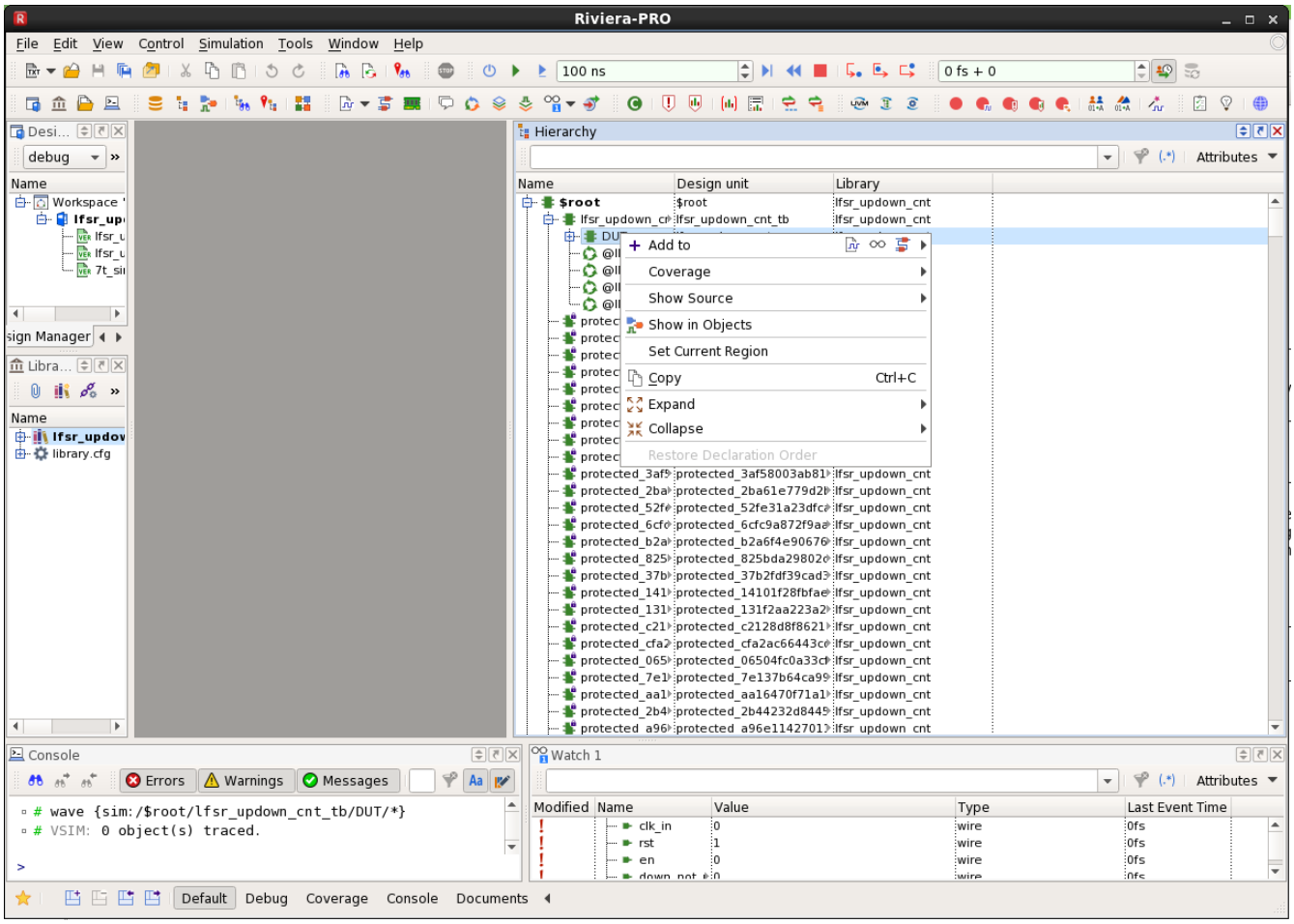

Figure 22: *Adding the Waveform*

Step 8 – View the Waveform

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

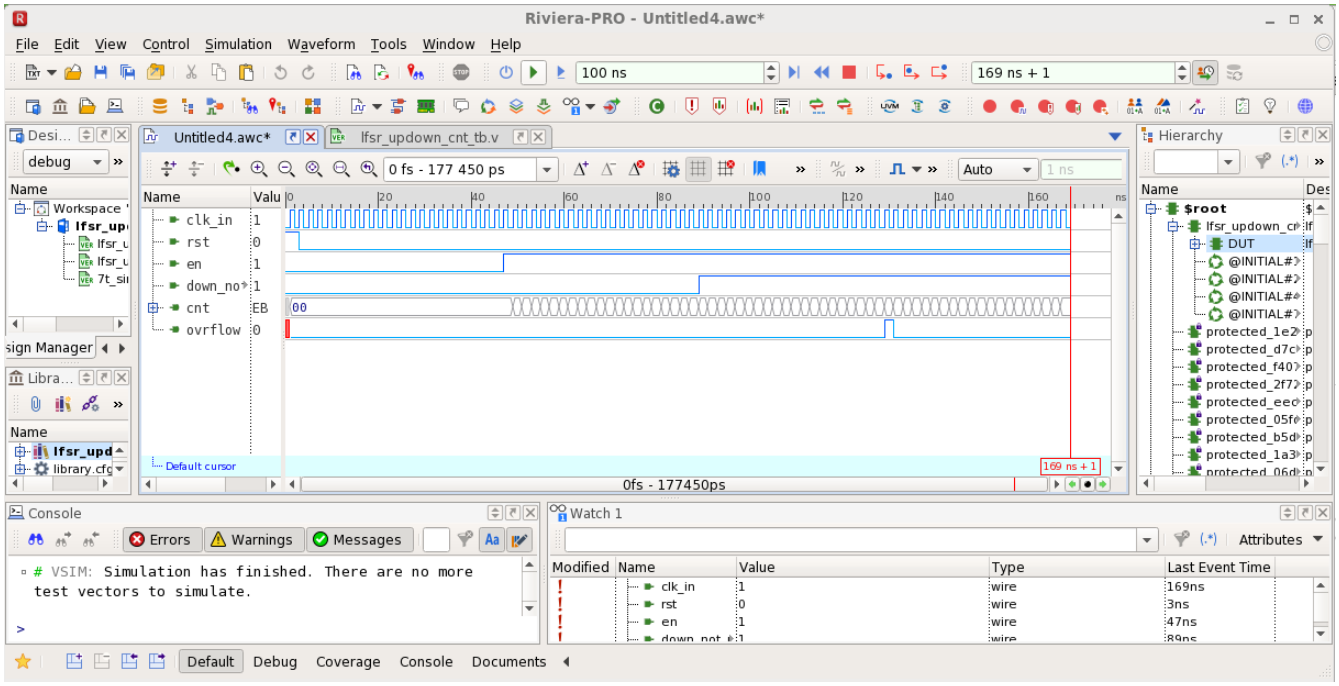

Figure 23: *Viewing the Waveform*

Gate-Level Simulation in Riviera

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

Step 1 – Create the Synthesis Project

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

Where `<technology>` is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

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 (see page 44) 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_path>/libraries/device_models/<technology>_simmodels.v

#Compile design
#design.compile test
alog -work lfsr_updown_cnt -incdir {/<ace_install_path>/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/<technology>_simmodels.v}

#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 <technology> is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

Step 4 – Run the Simulation

Follow Step 3 of RTL Simulation in Riviera (see page 44) to run simulation.

Step 5 – View the Results

Follow Steps 4 to the end of RTL Simulation in Riviera (see page 44) to view the waveform.

Post Route Simulation in Riviera

For post-route simulation, the synthesized gate-level neltist 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 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 would be 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 (see page 44) 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_path>/libraries/device_models/<technology>_simmodels.v

#Compile design
#design.compile test
alog -work lfsr_updown_cnt -incdir {/<ace_install_path>/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/<technology>_simmodels.v}

#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 `<technology>` is one of the abbreviated options from the table in the Simulation Libraries (see page 10) section.

Step 4 – Run the Simulation

Follow Step 3 of RTL Simulation in Riviera (see page 44) to run simulation.

Step 5 – View the Results

Follow Steps 4 to the end of RTL Simulation in Riviera (see page 44) to view the waveform.

Chapter - 12: I/O Ring Simulation Package

I/O Ring Simulation Package

Many designs require a simulation overlay named I/O Ring Simulation Package. This package combines the full RTL of the 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 NoC and models for the interface subsystems allows users to develop their 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 I/O ring simulation structure provides full RTL code for the NoC and BFM models of the surrounding interface subsystems. This structure is wrapped within a SystemVerilog module named per device, i.e., ac7t1500. The user needs to instantiate one instance of this module within their top-level testbench.

In addition, the simulation package provides binding macros such that the user can bind between elements of their design and the same elements within the device. For example, the design may instantiate a NoC access point (NAP). It is then necessary to bind this NAP instance to the NAP in the correct location within the NoC by using the `ACX_BIND_NAP_SLAVE, `ACX_BIND_NAP_MASTER, `ACX_BIND_NAP_HORIZONTAL, or `ACX_BIND_NAP_VERTICAL 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.

Example Design

An example structure of the I/O ring simulation, combined within a testbench, and with a design under test is shown in the diagram (see page 54) below. This example shows the macros required for the slave NAPs, and the DCIs for two instances of the GDDR subsystem. For other forms of NAPs, or for other DCI types, such as DDR, consult the Bind Macros (see page 56) and Direct-Connection Interfaces (see page 56) tables.

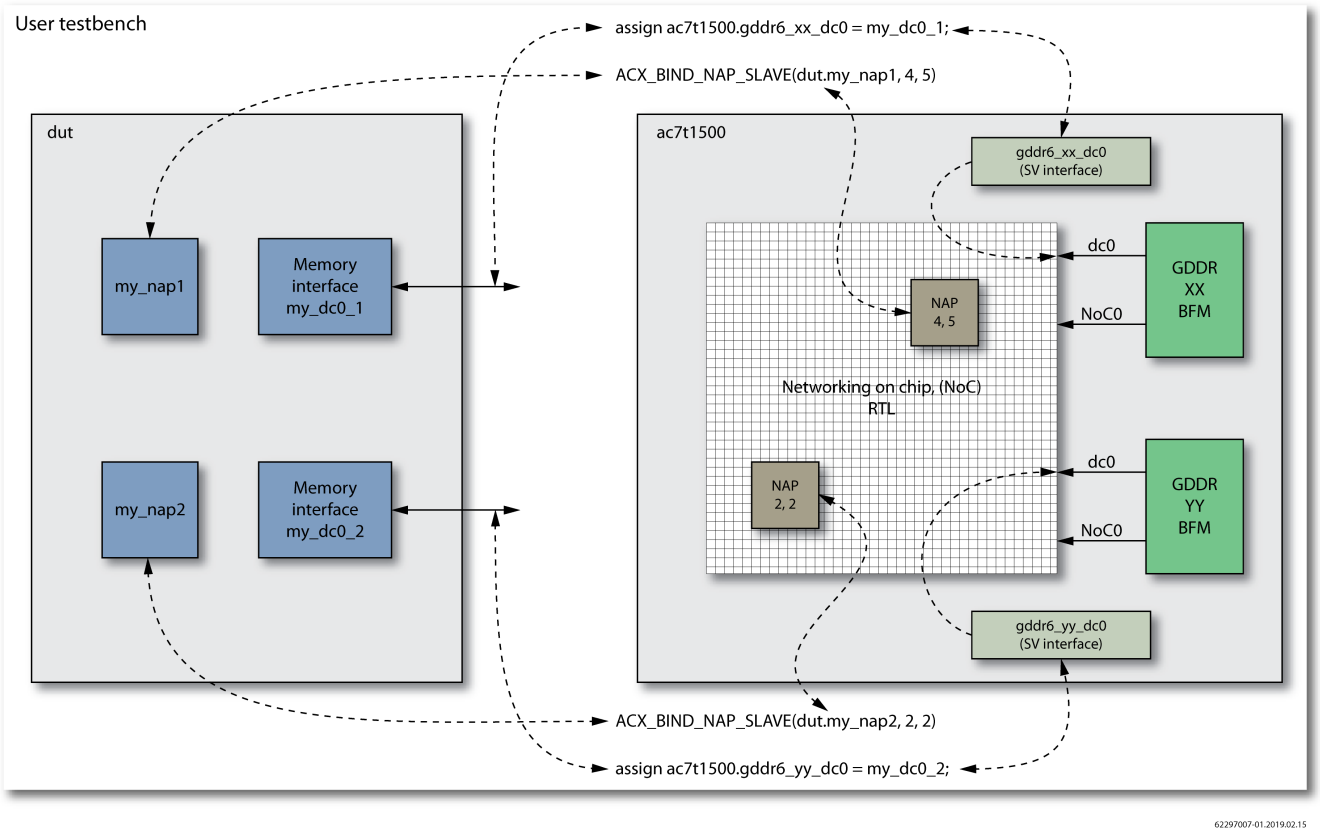

Figure 24: ***Example I/O Ring Simulation Structure***

In the example above, 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_SLAVE macro. This macro supports inserting the coordinates of the NAP within the 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 then assigned to the appropriate signals within the device direct-connect SystemVerilog interface.

The Verilog code to instantiate the above is shown below. This example is based on using the ac7t1500 device

```
// ---------------------------------------------
// Instantiate Speedster7t1500
// ---------------------------------------------
// Connect all ports to same named signals
ac7t1500 ac7t1500( .* );

// Set the chip_ready signal
assign chip_ready = ac7t1500.FCU_CONFIG_USER_MODE;

// Set the verbosity options on the messages
initial begin
    ac7t1500.verbosity = 3;
end

// ---------------------------------------------
// Bind NAPs
```

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

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


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


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


// -----------------------------------------------
// Remember to connect the clock!
// -----------------------------------------------
assign my_dc0_1_clk = ac7t1500.gddr6_xx_dc0.clk;
assign my_dc0_2_clk = ac7t1500.gddr6_yy_dc0.clk;
```

Note

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

In simulation

```
`ACX_BIND_NAP_AXI_SLAVE(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}
```

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 only starts to drive stimulus into the device once this signal is asserted (shown in the example above by use of a testbench `chip_ready` signal).

Bind Macros

The following bind statements are available.

Table 5: *Bind Macros*

Macro	Arguments	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_MASTER	user_nap_instance, noc_colunm, noc_row	To bind an AXI master NAP, instance ACX_NAP_AXI_MASTER.
ACX_BIND_NAP_AXI_SLAVE	user_nap_instance, noc_colunm, noc_row	To bind an AXI slave NAP, instance ACX_NAP_AXI_SLAVE.

Direct-Connect Interfaces

The following direct-connect interfaces are available.

Table 6: *Direct-Connect interfaces*

Subsystem	Interface Name	Physical Location	GDDR Channel	SystemVerilog Interface Type	Data Width	Address Width
GDDR	gddr6_e1_dc0	East 1	0	t_ACX_AXI4	512	33
GDDR	gddr6_e1_dc1	East 1	1	t_ACX_AXI4	512	33
GDDR	gddr6_e2_dc0	East 2	0	t_ACX_AXI4	512	33
GDDR	gddr6_e2_dc1	East 2	1	t_ACX_AXI4	512	33
GDDR	gddr6_w1_dc0	West 1	0	t_ACX_AXI4	512	33
GDDR	gddr6_w1_dc1	West 1	1	t_ACX_AXI4	512	33
GDDR	gddr6_w2_dc0	West 2	0	t_ACX_AXI4	512	33
GDDR	gddr6_e1_dc0	East 1	0	t_ACX_AXI4	512	33
DDR	ddr4_dc0	South	NA	t_ACX_AXI4	512	40

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.

SystemVerilog interfaces

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

Note

The interface below is only available in the simulation environment. For code that must be synthesized, users need to define their own SystemVerilog interfaces, or use one of the interfaces predefined within the reference designs.

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

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

   modport slave  (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);

endinterface : t_ACX_AXI4
```

Installation

There is a simulation package per device, available for both Linux and Windows. The packages are named `<device>_IO_BFM_Sim_<version>.tgz` for Linux and `<device>_IO_BFM_Sim_<version>.zip` for Windows. The packages are available on the Achronix self-service FTP site at secure.achronix.com, located in the `/Achronix/ACE/Speedster7t` folder. As each device package is independent, it is possible to either download only the device the user is targeting, or all device packages.

Any package is only required to be installed once — it is common for all designs targeting the selected device.

ACE Integration

The recommended installation method is to merge the contents of the package into the current ACE installation. The package contains a root directory `/system`. The contents of this folder should be merged with the selected ACE installation `/system` folder.

> **Warning!**
>
> The contents of the simulation package consists of files that are not present in the base ACE installation. These files should not replace or overwrite any existing files. However, if the user has already downloaded an earlier version of the simulation package, then they should select "overwrite" to ensure the latest version of the simulation files are written to the ACE installation.

Standalone

In certain instances it may not be possible for a user to modify their existing ACE installation. In these cases it is possible to install the package separately and to simulate using files from both this simulation package and the existing simulation files within ACE.

To install as standalone, simply uncompress the package to a suitable location.

> **Note**
>
> All reference designs are configured for the simulation package to be integrated within ACE. If the standalone method is selected, the user must edit the necessary environment variables in the reference design makefiles.

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 environment variable ACX_DEVICE_INSTALL_DIR is used to select the device I/O ring simulation files. It should be set to the base directory of the device files within the I/O ring simulation package. For example if the ac7t1500 device is selected, then the device base directory is yuma-alpha-rev0.

When the installation is done as ACE integration mode, then the following setting should be used:

```
ACX_DEVICE_INSTALL_DIR = $ACE_INSTALL_DIR/system/data/yuma-alpha-rev0
```

When the installation is done as standalone, the the following setting should be used:

```
ACX_DEVICE_INSTALL_DIR = <location of standalone package>/system/data/yuma-alpha-rev0
```

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.