



Agilent Technologies

HDL Cosimulation

August 2005

Notice

The information contained in this document is subject to change without notice.

Agilent Technologies makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Agilent Technologies shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty

A copy of the specific warranty terms that apply to this software product is available upon request from your Agilent Technologies representative.

Restricted Rights Legend

Use, duplication or disclosure by the U. S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 for DoD agencies, and subparagraphs (c) (1) and (c) (2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 for other agencies.

© Agilent Technologies, Inc. 1983-2005
395 Page Mill Road, Palo Alto, CA 94304 U.S.A.

Acknowledgments

Mentor Graphics is a trademark of Mentor Graphics Corporation in the U.S. and other countries.

Microsoft[®], Windows[®], MS Windows[®], Windows NT[®], and MS-DOS[®] are U.S. registered trademarks of Microsoft Corporation.

Pentium[®] is a U.S. registered trademark of Intel Corporation.

PostScript[®] and Acrobat[®] are trademarks of Adobe Systems Incorporated.

UNIX[®] is a registered trademark of the Open Group.

Java[™] is a U.S. trademark of Sun Microsystems, Inc.

SystemC[®] is a registered trademark of Open SystemC Initiative, Inc. in the United States and other countries and is used with permission.

Contents

1 HDL Cosimulation

Introduction	1-1
Basic Requirements	1-2
Theory of Operation	1-2
Supported HDL Data Types.....	1-3
Bidirectional HDL Ports	1-4
Precision for Bit-Vector Type Ports	1-5
Automatic Clock and Set Signals.....	1-5
Time-Specified Signals in User HDL Code.....	1-6
HDL Simulator Licenses	1-7
HDL Cosimulation Components and Their Parameters.....	1-8
HdlSrcFile	1-9
Inputs.....	1-10
InputPhases	1-10
InputPrecisions	1-10
Outputs	1-11
OutputPrecisions	1-11
HdlModelName.....	1-11
HdlLibrary	1-12
HdlSimulatorGUI.....	1-12
CmdArgs	1-13
IterationTime.....	1-13
TimeUnit	1-14

Index

Chapter 1: HDL Cosimulation

Introduction

With the ADS HDL cosimulation feature you can simulate components represented in a hardware description language (HDL) in the same schematic with other ADS components. This integrated capability provides complete design flexibility, and complements other ADS modules, including Digital Filter Designer.

The ability to design all portions of a communications product in one integrated environment can eliminate design errors resulting from disconnects among design teams. By cosimulating with HDL designs, you can easily incorporate your existing HDL intellectual property into new designs.

With HDL cosimulation, you can test hardware defined in HDL with a DSP algorithm, or use an algorithm written in HDL within an existing ADS design. VHDL and Verilog HDL are supported. ADS Ptolemy provides the signal processing simulation, while the *ModelSim*TM HDL simulator from Model Technology Incorporated, Verilog[®] XL from Cadence[®] Design Systems, or NC-SIM from Cadence[®] Design Systems simulates the HDL code. This cosimulation capability in one design environment makes it easy to test HDL components along with complex ADS system designs and see the effect on the entire system.

Important When using an *HdlCosim* component on the Sun5.10 platform, the *ModelSim* simulator will error out because the *ModelSim* simulator is not supported on Sun5.10.

When using a *VxlCosim* component with the Cadence Verilog-XL version 5.00p001, HDL cosimulation will fail. This failure is because Verilog-XL crashes and dumps core for certain types of Verilog code.

Basic Requirements

HDL Cosimulation is an optional feature of ADS. To run it, you must have:

- ADS Ptolemy simulator
- One of the following HDL simulators:
 - ModelSim/PLUS™
 - Verilog® XL
 - NC-SIM

Note For the latest versions of HDL simulators supported on ADS, see *Before You Begin > Check the System Requirements* in [Windows Installation](#) or [UNIX and Linux Installation](#).

Theory of Operation

With the HDL Cosimulation feature, ADS Ptolemy has been configured to cosimulate with either the ModelSim, VerilogXL, or NC-SIM HDL simulator. In this use model, you first create the HDL design. The design must be compiled and it is recommended to test the simulation with ModelSim, VerilogXL, or NC-SIM before cosimulation.

If the code is not compiled, you can use ADS to compile the code before cosimulation. Cosimulation requires information regarding the VHDL entity or Verilog module that you want to cosimulate with. This is used to generate HDL wrappers that incorporate user code and C-interface code to create an inter-process communication (IPC) link between ADS and the HDL simulator.

The cosimulation can be run in graphical user interface mode to monitor the HDL simulation. It can also be run in the background processing mode.

HDL Cosimulation uses the ADS Ptolemy Synchronous Dataflow (SDF) domain, in which numeric signals are consumed and produced by the HDL cosimulation component. There is no timing information communicated between ADS and the HDL simulator. ADS sends data into the HDL simulator and receives data without any knowledge of the HDL timing.

HDL Cosimulation does not use the ADS Ptolemy Timed Synchronous Dataflow (TSDF) domain. Since the HDL cosimulation component acts as an ADS Ptolemy

numeric component, any timed data from other ADS Ptolemy components will be converted to numeric at the HDL cosimulation component's input.

The HDL cosimulation component is a numeric component. Because the HDL simulation is time driven, it is initiated at every fixed interval for each firing of the HDL cosimulation component in ADS. The time scale used by the HDL simulator is independent of the ADS simulation.

Each time the HDL cosimulation component is fired, the HDL simulator receives input values from other ADS components and uses them to perform the HDL simulation. Once the HDL simulator is finished with its processing, it passes the simulation results back to ADS. These passed values are then the inputs for other ADS components, and thus the simulation cycle continues. This cycle repeats as many times as the scheduler requires. Each time the HDL cosimulation component is fired, the HDL simulation duration is determined by the value of the `IterationTime` parameter (see [“IterationTime” on page 1-13](#)) in the HDL cosimulation component. You must determine how long the HDL simulator should run before its outputs are sent back to the HDL component. This timing information should not be confused with the timing used in other ADS Ptolemy timed components.

From the HDL simulator engine's point of view, the ADS input interface is viewed as forcing values onto the ports. At the output interface of the HDL cosimulation component, the results are converted back into ADS format and sent to the connecting ADS component.

You can specify the HDL simulation to run until the HDL simulator has no more events to process by specifying a negative iteration time. Using this method, the outputs are guaranteed to be stable since there are no more events left in the simulator that might change them. This method is less efficient than the fixed positive iteration time method, as the HDL simulator must be monitored to determine when all events have been processed. Also, it will not work for certain HDL models where some designs never run out of events, such as those with internal clock signals. When a negative iteration time is specified for a Verilog module, a VHDL wrapper is used to instantiate the Verilog module.

Supported HDL Data Types

HDL cosimulation currently supports various bit and bit-vector type HDL ports; they are all mapped to the Ptolemy fixed data type port.

In the case of Verilog HDL, which supports only bit and bit-vector type ports, HDL cosimulation will support any type of Verilog port. In the case of VHDL, which has a

large set of data types, HDL cosimulation will only support the ports that are of bit and bit-vector types described in the IEEE `std_logic_1164` library.

Bidirectional HDL Ports

For a bidirectional VHDL port, two ports are created on the cosimulation model. One port is an *input* port named `<VHDL portname>In`, while the other port is an *output* port named `<VHDL portname>Out`. The input data on the inout type port is applied by ADS for the first half of the HDL iteration time; the signal value is then changed to a tri-state condition. You can drive the output data on an inout type port only during the second half of the HDL iteration time, when the value has been changed to a tri-state condition by ADS. You must set the inout port to a tri-state condition during the first half of the HDL iteration time period, so that ADS can drive the new input data value on the inout port.

Bidirectional ports are not supported in Verilog cosimulation.

Precision for Bit-Vector Type Ports

Bit-vector type HDL ports that get mapped to fixed type ports require precision for data conversion. The precision for inputs and outputs is specified in the parameter named “Inputs” on page 1-10 and “Outputs” on page 1-11.

The precision for a port is specified as two integers separated with a dot (for example 2.14). The first part is the number of bits used for representing the integral part and the second is the number of bits used for representing the fractional part of the value on the port.

By default, the arithmetic type is two’s complement. To specify an unsigned arithmetic type, append *u* to the precision specification. For example, an unsigned 2.14 can be specified as *2.14u*.

To repeat a particular precision specification you can use square bracket notation. For example, 2.14u 2.14u 2.14u 1.2 3.4 3.4 can also be represented as 2.14u[3] 1.2 3.4[2].

The least significant bit (LSB) of the fixed data will always be assigned to the lowest indexed element, and the most significant bit (MSB) will always be assigned to the highest indexed element of the HDL vector port. Since the fixed data bit has only two possible values (0 and 1) the values x, u, z, -, w, and l for 9-state std_logic types are mapped to 0 and the value h is mapped to 1.

Automatic Clock and Set Signals

HDL models that have pins named *Clock* and *Set* are treated differently. The HdlCosim component has pins named *Clock* and *Set* that are modeled as optional pins, meaning you can leave them unconnected. Clock and Set must be part of your Inputs specification, so that they will be processed as mentioned in the section “Inputs” on page 1-10.

If you connect any signal to *Clock* and *Set* pins, that signal will be passed to the HDL port. If you leave the pins unconnected, the default Clock and Set signals will be driven on the Clock and Set HDL ports.

The default clock is of a 50% duty cycle and has a period equal to the HDL iteration time. The positive clock edge occurs at $\text{IterationTime}/2$. The default value for the Set pin during the first iteration is a logic low at $1/4$ times the HDL iteration time, a logic high at $3/4$ times the HDL iteration time, and a logic high for iterations after that.

The timing of application of inputs for the HDL code generated for a mixed logic is crucial. For example consider a multiplexer (non-clocked, or in general any combinational logic) followed by a latch (clocked, or any sequential logic). The input multiplexer would be triggered the moment input is applied and would produce results with zero delay. If the following component is a clocked component (for example, sequential logic like a latch), then it will be triggered during the same iteration cycle at the positive edge of the clock. So, in the above example, the multiplexer and the latch will be triggered in the same clock cycle. In the corresponding fixed-point design, the multiplexer followed by a latch (*Clock* and *Set* unconnected) would fire the multiplexer in one cycle and the latch in the next, producing a delay of one cycle. The HDL cosimulation results will appear one cycle earlier when compared to the equivalent ADS component simulation results.

To match the results, the inputs to the HDL cosimulation block must be delayed until after the positive edge of the clock or $\text{IterationTime}/2$. The inputs will be applied to multiplexer or combinational logic after the positive clock edge. The latch will latch this result only in the next firing of the HDL cosimulation block or the next positive clock edge (the automatic clock has one positive clock edge per firing).

The delay for each input ports is specified in the parameter “[InputPhases](#)” on [page 1-10](#).

Time-Specified Signals in User HDL Code

When HDL code has internal clocks or time-specified signals (for example, wait statements in VHDL code) the HDL cosimulation may keep running until all the events in the user HDL code are processed. The number of events generated in user HDL code can be infinite (for example, when you have an internal clock).

You can avoid using an internal clock and use the ADS *Clock* instead (refer to the previous section “[Automatic Clock and Set Signals](#)” on [page 1-5](#), above). If this is not possible, then infinite event processing can be avoided if you know how long the HDL simulation needs to run to complete the cosimulation, with all of the ADS iterations. Different simulators have different mechanisms to break a simulation after a certain simulation time. Here is an example using ModelSim:

1. Use the ModelSim simulator to create a file called *test.do* under your project's data directory.

For example, *test.do* may look like this:

```
run 11000
quit -f
```

2. Set `CmdArgs = "-do test.do"` (refer to [“CmdArgs” on page 1-13](#)) on the `HdlCosim` component block.

This will stop the simulation after 11000 nsec.

The total run time can be calculated as equal to:

The number of ADS iterations (depends on the DF controller setup and the different sinks used in the design) multiplied by the `IterationTime` specified on the `HdlCosim` block.

Alternatively, you can also open the ModelSim UI mode and use multiple `run 100` commands to see how long it takes before the *ADS has completed its simulation* message appears in the ModelSim UI. This time can then be used to create the `test.do` file.

Do not use the `run -all` command, which will process all the events in the HDL simulation.

HDL Simulator Licenses

The HDL simulator can only be invoked for one HDL primary design unit, like an entity or a module. So for each HDL cosimulation model, a different HDL simulator is invoked, which uses an independent license. Currently, there is no way to save/restore the state from one entity to another. However, you can collect all the adjacent HDL cosimulation models into one top-level HDL entity and generate one HDL cosimulation model for that entity.

The ADS HDL Cosimulation feature has an independent license. Only one license is required for a design that has more than one HDL cosimulation model.

HDL Cosimulation Components and Their Parameters

HDL cosimulation components are available in the *HDL Blocks* palette or library (*Insert > Component > Component Library > HDL Blocks*). The ModelSim cosimulation component is `HdlCosim`; the VerilogXL cosimulation component is `VxlCosim`; and, the NC-SIM cosimulation component is `NCCosim`.

Some HDL simulators require compiled HDL code before simulation. If the user code has not been compiled, HDL cosimulation can compile the user code before cosimulation or use existing compiled HDL code depending on the `HdlSrcFile` and `HdlLibrary` settings. The process is further simplified for ADS-generated Verilog code.

Note NC-SIM cosimulation uses *ncverilog* for compilation and simulation. For VHDL cosimulation, the VHDL code will automatically be wrapped with a Verilog module. Refer to the NC-SIM manual for information on using *ncverilog* to simulate VHDL code made part of Verilog code.

Also refer to example design *iir_lp_ncvhdl.dsn*; access this design from the ADS Main window > *File > Example Project > SDFHdlCosim > iir_filter_prj*.

The components have one multi-input and one multi-output fixed-data type ports. They also have single bit input pins *Clock* and *Set*. The *Clock* and *Set* inputs are processed only if they are part of the “[Inputs](#)” on page 1-10 parameter specification; otherwise they are ignored. If they are part of the `Inputs` parameter, the behavior is the same as explained previously in the section “[Automatic Clock and Set Signals](#)” on page 1-5.

Note *Clock* and *Set* must not be part of the inputs that are being connected to the multi-input port.

HDL cosimulation models have parameters that enable you to control cosimulation with the HDL simulator. The following sections describe the parameters that require user input.

HdlSrcFile

HdlSrcFile can be specified as follows:

- For user HDL code, the HDL source code file must be specified as shown in example design *iir_lp_userhdl.dsn* (ADS Main window > *File* > *Example Project* > *SDFHdlCosim* > *iir_filter_prj*). Here, the HdlSrcFile must be the file that contains the VHDL entity or Verilog module information that you want to cosimulate with.

When you use ModelSim cosimulation (HdlCosim) and have not compiled the code, you must specify any other HDL files that the first file in HdlSrcFile depends on. The HDL files can be specified after the first file using space as the separator (see *iir_lp_userhdl.dsn* in the example project: ADS Main window > *File* > *Example Project* > *SDFHdlCosim* > *iir_filter_prj*). The files will be compiled at the beginning of the simulation in the reverse order of the specification. A *work* library is created automatically under project's data directory where all compiled code is saved.

In the case of VerilogXL cosimulation (VxlCosim) and NC-SIM cosimulation (NCCosim), all files required for simulation must be specified as mentioned above. In the case of NC-SIM cosimulation (NCCosim), you can use pre-compiled libraries as described in the NC-Verilog Reference manual.

- For user HDL code that contains a long list of dependency files to be compiled, you can create another file that contains the whole list of files. You must specify that file as the value for the *HdlSrcFile* parameter using < as the first character followed by the file name. See the specification of *HdlSrcFile*="<*iir_lp_flat.txt*" in the example design file *iir_lp_confvhdl.dsn* (ADS Main window > *File* > *Example Project* > *SDFHdlCosim* > *iir_filter_prj*).

For user-VHDL code containing a configuration specification in a file different from the file containing the entity specification, *and* if you want ADS to compile the user code prior to cosimulation, then the file specification order is important. The order is important because ADS compiles the files in reverse order of specification, and the configuration specification file must be compiled *after* the entity specification file. Therefore, the configuration specification file must be specified prior to the entity specification file. For such a specification to identify the entity specification file, the user-VHDL entity specification file must be preceded with a + character. See example design *iir_lp_confvhdl.dsn* (ADS Main window > *File* > *Example Project* > *SDFHdlCosim* > *iir_filter_prj*).

Inputs

The `Inputs` parameter lists the names of the input ports of the HDL model. All the HDL input port names that need to be updated from ADS must be specified.

This list is used to make the input connections between the ADS ports and the HDL ports. The `BusMerge` component must be used on the input port when there is more than one input to be connected. The first (last) input port in the list is connected to the bottom (top) most input port on the `BusMerge`, and so on. The `BusMerge` must have a number of input ports equal to the number of strings specified in the `Inputs` string array, except if the model has pins named *Clock* and *Set*. The `BusMerge` component must not have pins corresponding to *Clock* and *Set*, instead they must be connected to the *Clock* and *Set* ports of the ADS `HdlCosim` component. To use automatic *Clock* and *Set* signals, leave the *Clock* and *Set* ports unconnected or hanging.

InputPhases

The `InputPhases` parameter delays the application of the input to the HDL model. It is an array of integers. The time unit is the same as specified by the `TimeUnit` parameter, described later. The `InputPhases` parameter specifies the delay for the application inputs during an iteration, as explained in the section [“Automatic Clock and Set Signals” on page 1-5](#). The delay specified for the *Clock* signal is ignored if the *Clock* signal is not connected and an automatic clock is being used.

If *Clock* and *Set* are unconnected and the `InputPhases` parameter is not specified, the inputs are automatically delayed for 3/4 times the `IterationTime` specified during each cycle for the reasons mentioned in earlier section [“Automatic Clock and Set Signals” on page 1-5](#). If any phase delay values are specified by the user, those values will be used.

In the case of the ADS `HdlCosim` component, the order of the delay specification must be the same as the order of the input names specified in the `Inputs` parameter.

InputPrecisions

The `InputPrecisions` parameter specifies the precision and arithmetic type to be used for a particular input.

The order of the precision specification must be the same as the order of the input names specified in the `Inputs` parameter.

Precision and arithmetic type specification was explained earlier in the section [“Precision for Bit-Vector Type Ports” on page 1-5](#).

Outputs

The Outputs parameter lists the names of the output ports of the HDL model. All HDL output port names that need to be read into ADS must be specified.

This list is used when making the output connections between the ADS Ptolemy ports and the HDL ports. The BusSplit component is used on the output port. The first (last) output port in the list is connected to the bottom (top) most output port on the BusSplit, and so on. The BusSplit component must have a number of output ports equal to the number of strings specified in the Outputs string array.

Note In the case of an inout VHDL port, specify the port name in the Inputs as well as the Outputs parameter.

OutputPrecisions

The OutputPrecisions parameter specifies the precision and arithmetic type to be used for a particular output.

The order of the precision specification must be the same as the order of the output names specified in the Outputs parameter, see [“Outputs” on page 1-11](#).

Precision and arithmetic type specification was explained earlier in the section [“Precision for Bit-Vector Type Ports” on page 1-5](#).

HdlModelName

HdlModelName is the name of the HDL entity or module to cosimulate with.

For a Verilog module, specify the module name to cosimulate with. For a VHDL entity you can specify this parameter in the following ways:

- To select an entity that has only one architecture, the syntax is

```
HdlModelName="<entity>"
```

- To select an entity along with a particular architecture when more than one is available, the syntax is

```
HdlModelName=" <entity>.<architecture>"
```

- To select an entity along with its configuration specification, the syntax is

```
HdlModelName=" <entity>+<configuration>"
```

See example design *iir_lp_confvhdl.dsn* (ADS Main window > *File* > *Example Project* > *SDFHdlCosim* > *iir_filter_prj*).

HdlLibrary

The `HdlLibrary` parameter does not exist for VerilogXL cosimulation (`VxlCosim`) and NC-SIM cosimulation (`NCCosim`).

In the case of ModelSim cosimulation (`HdlCosim`), this parameter specifies the library from which the compiled HDL module or entity must be loaded. This parameter can control the compilation as follows:

- If the code needs to be compiled, `HdlLibrary` must be empty. This will compile the code under *work* library. For any subsequent re-simulation of the same design, the HDL code need not be recompiled. To turn off compilation, specify `HdlLibrary=work`.
- If you have already compiled the code in another library (for example, *hdllib*) then only the file that has the entity or module specification needs to be specified for `HdlSrcFile`, and `HdlLibrary` should be set to *hdllib*.

Before starting ADS, the `MODELSIM` environment variable must be set to a *modelsim.ini* file that has the mapping information for *hdllib*.

If `MODELSIM` is not set, you can specify the mapping for the library using `=`, for example `HdlLibrary="hdllib=/user/xyz/hdllib"`. You can specify more than one library by separating them using spaces, and can specify mappings for any of the libraries using `=`.

HdlSimulatorGUI

The `HdlSimulatorGUI` parameter determines the user interface mode of the HDL simulator. If the `HdlSimulatorGUI` is On, the HDL simulator is started with its graphical user interface on. You can view the progress of the simulation, graph signals, and edit values while the simulation is running.

The ModelSim command `Restart` is not supported during cosimulation. To restart HDL cosimulation, quit and restart the ADS simulation.

Note If the `HdlSimulatorGUI` is On and `IterationTime` is negative, use `run -all` in `ModelSim` to perform cosimulation. The other `run` commands will only increment the HDL simulation time and will not cosimulate.

If the `HdlSimulatorGUI` is Off, the simulator is run in the background. ADS will start the HDL simulator, run the simulation, and close the simulator at the end of simulation without user interaction.

CmdArgs

The `CmdArgs` parameter specifies special simulator command invocation arguments required for simulation of the HDL model.

IterationTime

`IterationTime` is the time that the HDL simulation is run during each firing of the HDL cosimulation component. If the integer value provided is positive, the HDL simulator will simulate for the specified number of time units (where the time units are specified by the parameter “`TimeUnit`” on page 1-14) then send data to ADS. This does not check to see if there are any events still to be processed in the simulator. This feature is useful if you are running a model whose output data is to be sampled periodically at a predetermined time.

Note The value can never be specified as 0 because the simulation will stop with a range error flagged.

Negative iteration time is valid only for `ModelSim` VHDL cosimulation. If the value is negative, the HDL simulator is run until all the events are processed. The magnitude of the value specifies the minimum amount of time to run before checking to see if there are any events still to be processed. The output data is read after the event queue becomes empty. This facility can slow down the simulation due to the overhead of monitoring the simulation event queue. The lower the magnitude, the slower the execution because the event queue must be polled more often. This facility is useful when the time the model takes to provide stable/correct data output varies. This will not work for certain models that never run out of events, such as those with internal clock signals.

When a negative iteration time is specified for a Verilog module to be cosimulated using ModelSim, a VHDL wrapper is used to instantiate the Verilog module.

Note Negative iteration time will not work with Cadence NCsim or VerilogXL cosimulation.

TimeUnit

The TimeUnit parameter specifies the HDL simulation time resolution unit: *fs*, *ps*, *ns*, *us*, *ms* or *sec*.

For VHDL simulation using ModelSim, TimeUnit will control the VHDL simulation time resolution.

For Verilog simulation with any supported HDL simulator, TimeUnit will add timescale directives to the top-level cosimulation wrapper. Users can have timescale directives for different modules in their code. If any user module does not have a timescale specified, TimeUnit will be used to generate a default timescale. The smallest of the different timescale specifications will control the Verilog simulation time resolution.

Index

C

CmdArgs, 1-13

H

hdl cosimulation, 1-1

HdlLibrary, 1-12

HdlModelName, 1-11

HdlSimulatorGUI, 1-12

HdlSrcFile, 1-9

I

InputPhases, 1-10

InputPrecisions, 1-10

Inputs, 1-10

IterationTime, 1-13

O

OutputPrecisions, 1-11

Outputs, 1-11

T

TimeUnit, 1-14

