Agilent Technologies

Advanced Design System 2002

HDL Cosimulation

February 2002

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
395 Page Mill Road
Palo Alto, CA 94304 U.S.A.

Acknowledgments

Model*Sim* HDL simulator is a trademark of Model Technology Incorporated

Verilog is a registered trademark of Cadence Design Systems

NC-SIM is a trademark of Cadence Design Systems

Contents

Chapter 1: HDL Cosimulation

Advanced Design System's HDL Cosimulation feature allows you to simulate components represented in a hardware description language (HDL) in the same schematic along with other Advanced Design System components. This integrated capability gives you complete design flexibility, and complements other Advanced Design System modules, including DSP Synthesis and 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 various design teams. By being able to cosimulate 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. Both VHDL and Verilog HDL are supported. Agilent Ptolemy provides the signal processing simulation, while either the Model*Sim*™ 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.

Basic Requirements

HDL Cosimulation is an optional feature of Advanced Design System. To run it, you need to have the following:

- Agilent Ptolemy simulator
- One of the following:
  - Model*Sim*/PLUS™ EE V5.2e or higher
  - Verilog® XL Version 3.11 or higher
  - NC-SIM Version 3.2 or higher

Basic Design Flow

The following graph shows the basic HDL Cosimulation design flow:

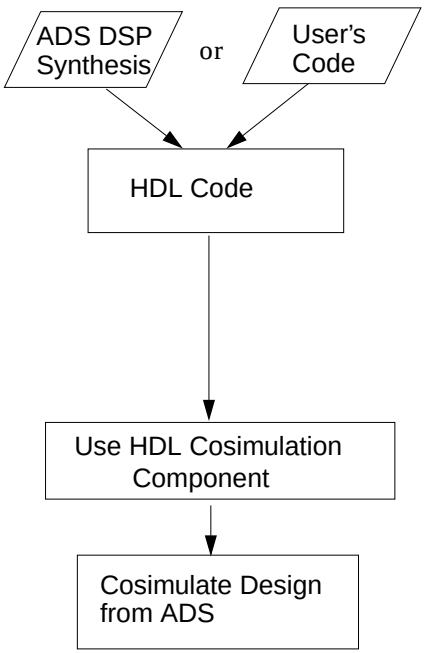

Figure 1-1. HDL Cosimulation Design Flow

Theory of Operation

With the HDL Cosimulation feature, Agilent 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. This design may be created using Advanced Design System's DSP Synthesis, independently by you, or by employing existing intellectual property. 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 Agilent 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 the ADS and the HDL simulator. ADS sends data into the HDL simulator and gets data back without any knowledge of the HDL timing.

HDL Cosimulation does not use the Agilent Ptolemy Timed Synchronous Dataflow (TSDF) domain. Since the HDL cosimulation component acts as an Agilent Ptolemy numeric component, any timed data from other Agilent Ptolemy components (such as a GSM modulator) will be converted to numeric at the HDL cosimulation component's input.

The HDL cosimulation component is a numeric component. But since 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 1-14) 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 Agilent 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

The number of data types supported in Agilent Ptolemy is smaller than the number of data types supported by VHDL. To support the large set of types that you can have in VHDL, a mapping strategy is followed. Table 1-1 shows the types supported. Agilent Ptolemy supports all of the types from the IEEE standard VHDL library.

Note Only bit-vector types are supported in the current release.

Table 1-1. Mapping Between the VHDL and Agilent Ptolemy Data Types

VHDL	Ptolemy
Bit	fix
Bit_vector	fix
Signed	fix
Std_logic	fix
Std_logic_vector	fix

Table 1-1. Mapping Between the VHDL and Agilent Ptolemy Data Types

VHDL	Ptolemy
Std_ulogic	fix
Std_ulogic_vector	fix
Unsigned	fix
Ux01	fix
Ux01z	fix
X01	fix
X01z	fix

Record and multi-dimensional array types in VHDL, are not supported. Verilog module ports have types that can only be either bit or bit vector type and they are mapped onto the Agilent Ptolemy fix type ports.

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 and then the signal value is 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 have to 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 port require precision for data conversion. The precision for inputs and outputs is specified in the parameter named “Inputs” on page 1-10 and for outputs “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.

The arithmetic type by default 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 nine-state std_logic types are mapped to 0 and the value h is mapped to 1.

ADS Generated Combinational and Sequential Logic

The Advanced Design System DSP Synthesis Library contains a mixture of components, some clocked and some not. The *Clock* and *Set* pins on the clocked components, such as a latch or a register, can be left unconnected. When these pins are left unconnected, an automatic signal is sent to those pins.

When the ADS DSP Synthesis tool is used to generated HDL code, the resulting model may have *Clock* and *Set* pins. The HdlCosim component has pins named *Clock* and *Set*, which 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. 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 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.

Since this feature is provided to accommodate Advanced Design System-generated HDL code, the same rules apply if your HDL code has pins named Clock and Set.

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 have to be delayed until after the positive edge of the clock or 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 1-11.

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 "ADS Generated Combinational and Sequential Logic" on page 1-6, above). But 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. Then set CmdArgs = “-do test.do” (refer to “CmdArgs” on page 1-13) on the HdlCosim component block.

   This will stop the simulation after 11000 ns.

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 shows up 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 can be found in the *HDL Blocks* palette or library (*Insert > Component > Component Library > HDL Blocks*). The component for cosimulation with ModelSim EE is called HdlCosim, the VerilogXL cosimulation component is called VxlCosim, and the NC-SIM component is called NCCosim.

ModelSim SE/EE needs 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, wrap the VHDL code with a Verilog module. Refer to the NC-SIM manual for information on using *ncverilog* to simulate VHDL code made part of Verilog code.

The components have one multi input and one multi output fixed-data type port. They also have two more input pins called *Clock* and *Set*, both single bit. They are processed or only read 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 earlier in the section "ADS Generated Combinational and Sequential Logic" on page 1-6.

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

The components are useful to cosimulate only with HDL components having either bit or bitvector type ports, or port types that get mapped to a fixed data type port as shown in Table 1-1. You cannot cosimulate, for example, VHDL components with ports of integer or real data types. This is not a major limitation, since most of the practical HDL models will always have bit or bitvector type ports.

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

HdlSrcFile

This can be specified in either of the following two ways:

- For ADS-generated HDL, use the generated compilation script: Upon HDL generation, the compilation script can be found under your project's synthesis/vhdl or synthesis/verilog sub-directory, typically called as compile_vhdl or compile_verilog, respectively (if HDL code was generated on WIN32 platform, then the file will have a .bat extension). By default, if HdlSrcFile="", cosimulation will look for a compile_verilog file under the

---

current project's synthesis/verilog directory. (See iir_lp_hdlcosim.dsn in the example project iir_filter_prj.) You can also explicitly specify the compilation script to use, as in the examples iir_lp_adsvhdl.dsn and iir_lp_adsvlog.dsn.

- For user HDL code that is not generated using ADS, the HDL must be specified as shown in the example design iir_lp_userhdl.dsn. 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 SE/EE cosimulation (HdlCosim) and have not compiled the code, you must specify any other HDL files on which 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). 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 the compiled code is saved.

  In the case of VerilogXL cosimulation (VxlCosim) and NC-SIM cosimulation (NCCosim), all the 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.

Inputs

This 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.

If the HDL model is generated using ADS DSP Synthesis tool, it may have pins named *Clock* and *Set*. These pins must be specified in the Inputs parameter for correct results.

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

This parameter lets you delay 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 "ADS Generated Combinational and Sequential Logic" on page 1-6. The delay specified for the Clock signal is ignored if the Clock signal is not connected and an automatic clock is being used.

If the *Clock* and *Set* are left 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 "ADS Generated Combinational and Sequential Logic" on page 1-6. 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

This parameter specifies the precision and arithmetic type to be used for a particular input.

In the case of the ADS HdlCosim component, 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

This parameter lists the names of the output ports of the HDL model. The names should be in the same order and in the same case as they appear in the HDL wrapper code, which is generated after performing the model building step.

This list is used when making the output connections between the Agilent 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

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

In the case of the ADS HdlCosim component, 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

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

HdlLibrary

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

In the case of ModelSim SE/EE cosimulation (HdlCosim), this file specifies the library from which the compiled HDL module or entity must be loaded. This parameter can control the compilation in either of the following ways:

- If the code needs to be compiled, HdlLibrary must be empty. This will compile the code under *work* library. For any following re-simulation of the same design, the HDL code need not be recompiled. To turn off compilation, you can 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 has to 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 “=” as explained earlier.

HdlSimulatorGUI

This parameter determines the user interface mode of the HDL simulator. If the HdlSimulatorGUI is On, HDL simulator is started up with its graphical user interface on. You may view the progress of the simulation, graph signals, and even 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 up the HDL simulator, run the simulation, and close the simulator down at the end of simulation, without user interaction.

CmdArgs

This parameter allows you to specify any special simulator command invocation arguments required for simulation of the HDL model. This is useful in the case of VHDL cosimulation using the HdlCosim component when generics are to be initialized.

For example, when using ModelSim, a run.do file can be created and put in the */data* subdirectory of the project as follows:

```
run -all
quit -f
```

Then the syntax of the CmdArgs would be:

```
CmdArgs = “-do run.do”
```

In the case of NC-SIM cosimulation, you can use “-R” to avoid the recompilation of HDL code.

IterationTime

IterationTime is the time that the HDL simulation is run during each firing of the HDL cosimulation component. If the integer value provided to it 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) and then send the data to ADS. This does not check to see if there are any events left 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.

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 left to process. 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 is the execution, because the event queue has to 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.

The value can never be specified as 0. If it were 0, the simulation will stop with a range error flagged.

When a negative iteration time is specified for a Verilog module, a VHDL wrapper is used to instantiate the Verilog module. Negative iteration time will not work with VerilogXL cosimulation.

TimeUnit

This parameter tells the HDL simulator the HDL simulation time resolution unit to be used. Possible settings are *fs, ps, ns, us, ms* and *sec*.

Index