SystemC Verification with Model*Sim*

Introduction

Model*Sim* was the first to put the award winning Single Kernel Simulator (SKS) technology in the hands of engineers, enabling transparent mixing of Verilog and VHDL in one design, with a common intuitive graphical interface for development and debug at any level, regardless of the language.

Model*Sim* 5.8 was the first release to enhance the SKS technology by providing native support for SystemC. As far as you are concerned, SystemC is just another design language. You are now able to simulate and debug your SystemC designs basically the same way you have been simulating and debugging your Verilog and VHDL designs. There are just a few new SystemC-specific commands and compiler/simulator options but the methodology remains the same. In addition to being easy to use, native implementation means no more PLI/FLI.

This document introduces SystemC verification with Model*Sim*, starting with setting up the simulation environment, and followed by a discussion of the use model. Information on using SystemC across Verilog and VHDL boundaries is also presented. A sample design is used to illustrate the current capabilities in Model*Sim*.

Environment Setup

Similar to Verilog and VHDL, SystemC is a licensed feature. You need the *systemc_c license* feature in your Model*Sim* license file to simulate SystemC designs. Please contact your Mentor Graphics sales representatives if you currently do not have such feature.

SystemC is part of the Model*Sim* standard installation. You may download the latest Model*Sim* release at **http://www.model.com/products/release.asp**. Note that SystemC support is only available in Model*Sim* release 5.8 and later. Please refer to the Start Here for Model*Sim* document for installation information. Table 1 illustrates the supported operating systems for SystemC and the corresponding required versions of C++ compiler.

Platforms	C++ Compiler Versions
SunOS 5.6 or Later	gcc 3.2[1]
Red Hat Linux 7.2 or Enterprise Linux 2.1 or Later	gcc 3.2.3[2]
HP-UX 11.0 or Later	aCC 3.45 (with Associated Patches)[3]
Windows NT, 2000, XP[4]	MinGW gcc 3.2.3

Table 1: Supported Operating Systems and Compilers for SystemC

[1] *When you install Model*Sim *5.8 or later, you must also install the* modeltech-gcc-3.2-sunos5<6/7/8/9>.tar.gz file *which is available at the same location from where you download Model*Sim*. It contains the version of gcc already configured to work with Model*Sim*. This version of gcc is what Model*Sim *calls automatically to compile and link SystemC code. We strongly recommend against the use of other gcc versions.*

[2] *On Red Hat Linux, you must also install the* modeltech-gcc-3.2.3-rhe21.tar.gz file *which is available at the same location from where you download Model*Sim*. It contains the version of gcc already configured to work with Model*Sim*. This version of gcc is what Model*Sim *calls automatically to compile and link SystemC code. We strongly recommend against the use of other gcc versions.*

[3] *On HP-UX, Model*Sim *automatically uses aCC from /opt/aCC. If aCC is installed somewhere else in your environment, you may specify the full path to aCC using the "CppPath" variable in the* modelsim.ini file.

[4] *Model*Sim *6.0 is the first release that supports SystemC on Windows. The required C++ compiler is MinGW gcc 3.2.3 which is automatically installed when you install ModelSim.*

Note that SystemC is supported on the 32-bit mode only. Please contact your System Administrator if you do not have the required operating systems or compiler versions installed.

Use Model

Model*Sim* provides a unified kernel for SystemC, Verilog, and VHDL. Figure 1 illustrates the use model.

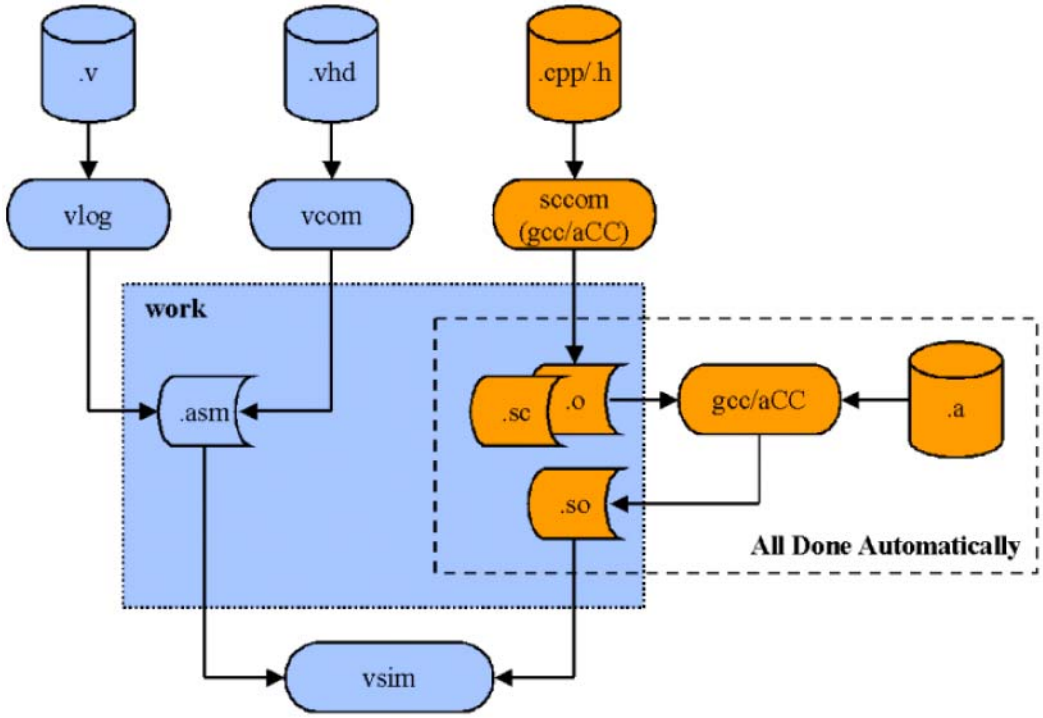

Figure 1: SystemC Use Model

vlog and *vcom* are the Verilog and VHDL compilers, respectively. The compilers compile the Verilog or VHDL source code into machine-independent executables (*.asm*), which are stored in the work library. When you load your design in Model*Sim*, as soon as an instantiation statement is encountered the work library is searched for a matching design unit. Once it is found it is pulled into elaboration. This methodology remains practically unchanged for SystemC

designs. Instead of using *vlog* and *vcom*, *sccom* is the compiler for SystemC source code. *sccom* calls the C compiler (*gcc* on Linux, SunOS, Windows, and *aCC* on HP-UX) to compile your SystemC source code and generates object files (*.o*). It also generates a *.sc* file to store debug information. *sccom* is then used again to link the object files with other compiled libraries, if any, to create a shared object (*.so*) in the current work library. During design elaboration, when Model*Sim* encounters a *.sc* file instead of a *.asm* file, it knows immediately that it is dealing with a SystemC component. The shared object is then pulled into elaboration. Note that the process of collecting the object files and linking them (shown in the dotted box) is handled by *sccom*. Such capability relieves you from having to manage the linking process so that you can focus on the design.

Note that in the process of creating the SystemC source code, *sc_module* is supported instead of *sc_main*. Please refer to the Model*Sim* User's Manual for examples on converting *sc_main* blocks to *sc_module* blocks.

Another advantage of Model*Sim*'s SystemC implementation over other solutions is the true mixed-language capability. Figure 2 illustrates that you can have any combinations of SystemC, Verilog, and VHDL components in a design.

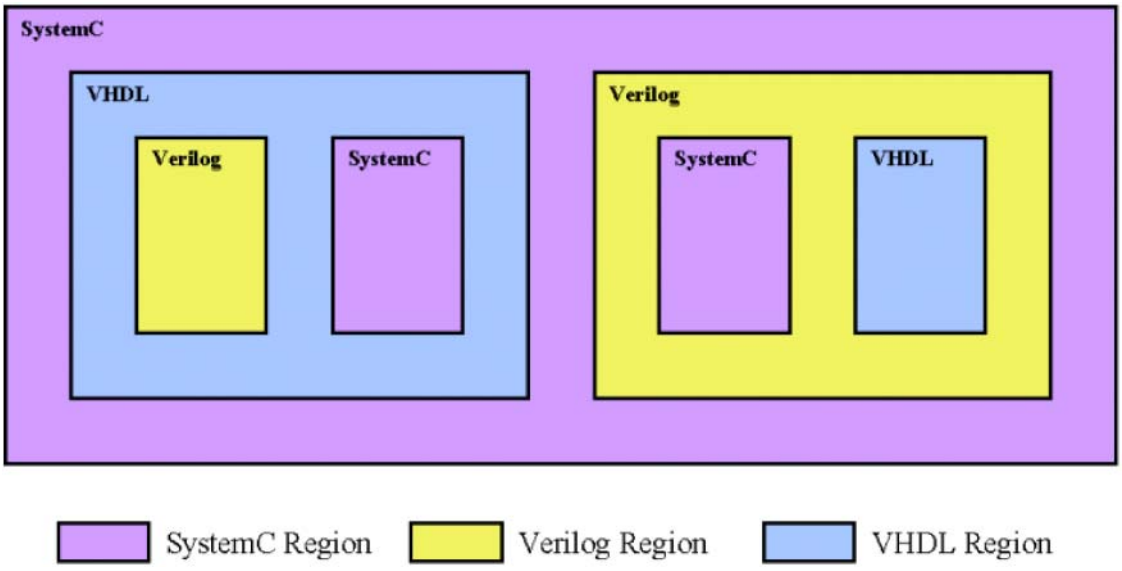

Figure 2: Mixed-Language Environment

There are six scenarios of mixed-language instantiation:

1. SystemC-over-Verilog
2. SystemC-over-VHDL
3. Verilog-over-SystemC
4. VHDL-over-SystemC
5. Verilog-over-VHDL
6. VHDL-over-Verilog

Scenarios 5 and 6 involving Verilog and VHDL are well understood so will not be further discussed in this document. In **scenarios 1 and 2**, what is needed is a stub SystemC module that stands in for each Verilog or VHDL design unit instantiated under SystemC. This is similar to the concept of component declaration in VHDL. As you may know, once a Verilog module is compiled into a library, you can use the *vgencomp* command to generate its equivalent VHDL component declaration to standard output. Similarly, there is a new command called *scgenmod* which generates the stub SystemC modules for you. For example:

```
scgenmod hdl_module
```

writes the stub SystemC module for the HDL (Verilog or VHDL) module *hdl_module* to the standard output. Copy these stub SystemC modules as header (*.h*) files to a directory (<Include/Source Dir> in Figure 3) and include them in the SystemC parent module for later linking. In the example presented in the next section, you will see that the stub SystemC module is nothing more than a wrapper with the HDL design unit declared as an *sc_foreign_module*. Please refer to the *Model*Sim *Command Reference* for complete usage and option information for *scgenmod*. Note that you can always create the stub SystemC modules by hand which is required when custom port types are involved. The utility *scgenmod* is just provided as a convenience.

For **scenario 3**, all you have to do is instantiate the SystemC modules as if they were regular Verilog modules in the Verilog source code.

For **scenario 4**, the command *vgencomp* is enhanced to create VHDL component declarations for SystemC modules. What you need to do is copy the generated component declarations to the VHDL source code. Please refer to the Model*Sim* Command Reference for complete usage and option information for *vgencomp*.

Figure 3 summarizes the typical steps to handle a mixed-language design.

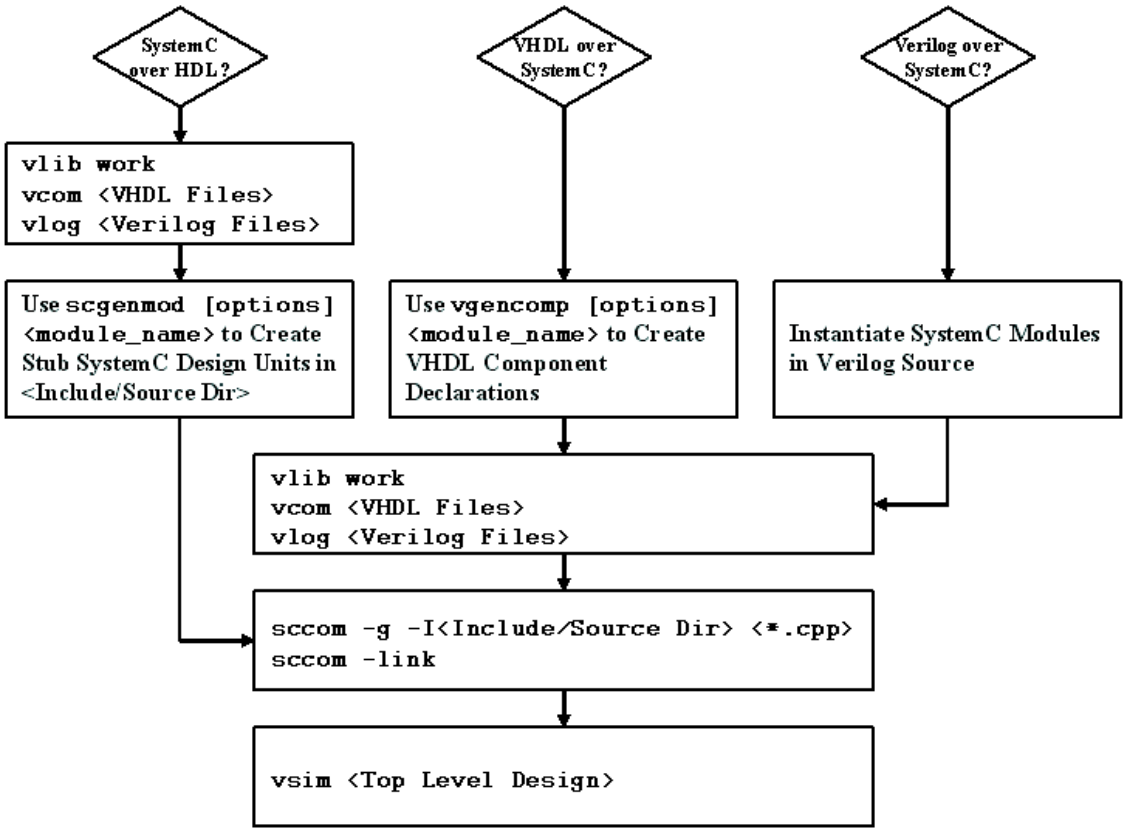

Figure 3: Mixed-Language Design Flow

sccom accepts any normal C++ compiler options with the exception of the *–o* and *–c* options. By default, *sccom* compiles without debugging information. You must specify the *–g* argument, as shown in Figure 3, to compile for debugging.

Note that the *–link* option of *sccom* is used to instruct Model*Sim* to link all the generated object files (*.o*) to create the shared object (*.so*). You do not have to specify individual files to link.

Beginning in Model*Sim* 6.0, you can specify a library for *sccom* into which your SystemC code is compiled. If you compile your SystemC design into a different library than the default work library, you have to specify two options to *sccom* during the link process (*sccom -link*):

-lib: Specifies the library that contains the compiled object files. Multiple *-lib* options can be specified if the design is compiled into multiple libraries.

-work: Specifies the library where the final shared object (*.so*) will reside.

For example:

```
vlib final_work
vlib sc_work
sccom -g -I/usr/systemc/lib -work sc_work top.cpp
sccom -link -work final_work -lib sc_work
vcom -work final_work file1.vhd
vlog -work final_work file2.v
```

When you simulate the design in Model*Sim* 6.0, *vsim* has a new option called *-sclib* to specify the library where the SystemC shared object (*.so*) is created. This option is only needed if the shared object is in a library other than the default work library. For example:

```
vsim -sclib final_work top
```

Model*Sim*'s SystemC implementation is based on the SystemC specification version 2.0.1, which includes support for the SystemC Verification (SCV) Library 1.0. SCV is included in the standard Model*Sim* installation. If your design uses SCV library functions, you will have to compile and link the SystemC source files with the *–scv* option. For example:

```
sccom -scv -g -I/usr/systemc/lib top.cpp
sccom -scv -link
```

Please refer to the Model*Sim* Command Reference for detailed description of all *sccom* options.

Crossing Language Boundaries

Model*Sim* provides a true mixed-language verification environment. Various cross-language capabilities are described below.

Probing and Forcing HDL Signals from SystemC

Beginning in Model*Sim* 6.0, you can probe and force Verilog or VHDL signals and ports (except Verilog registers and integers) from SystemC. For example, in order to have a SystemC signal (including *sc_signal, sc_buffer, sc_signal_resolved, sc_signal_rv*) force an HDL signal, you can do the following in your SystemC code:

```
sc_signal<T> force_sig;
force_sig.control_foreign_signal("<hdl_signal_path>");
```

Whenever the SystemC signal *force_sig* changes value, the HDL signal it is controlling will be updated with the value of *force_sig*. The forced value on the HDL signal remains until there is a subsequent driver transaction.

Similarly, in order to probe an HDL signal from SystemC, you can do the following in your SystemC code:

```
sc_signal<T> probe_sig;
probe_sig.observe_foreign_signal("<hdl_signal_path>");
```

Whenever the HDL signal changes value, the SystemC signal *probe_sig* will be updated with the value of the HDL signal. The value on the SystemC signal remains until there is a subsequent driver transaction.

The argument ("<*hdl_signal_path*>") to the *control_foreign_signal* and *observe_foreign_signal* member functions is the full hierarchical path to the HDL signal or port being probed or forced.

Please refer to the tables for Verilog data type mapping and VHDL data type mapping in the Model*Sim* User's Manual for the list of types supported at the mixed language boundary. If it is a supported boundary type, it is supported for hierarchical references.

Once the *sc_signal* probes or forces an HDL signal, the effect stays throughout the entire simulation. Subsequent calls to *control_foreign_signal* or *observe_foreign_signal* are not allowed.

The primitive channel *sc_signal* is used in the above example. The same also works for *sc_signal_resolved*, *sc_signal_rv*, and *sc_buffer*.

Passing Parameters/Generics between HDLs and SystemC

Also new in Model*Sim* 6.0 is the ability to pass parameters and generics between HDLs and SystemC.

SystemC modules can receive parameters and generics from HDL parents. The following methods can be used in SystemC modules which have immediate HDL parents:

```
int        sc_get_int_param(const char* param_name);
double     sc_get_real_param(const char* param_name);
sc_string sc_get_string_param(const char* param_name, char format_char = 'a');
```

The argument *param_name* is the name of the parameter of type integer, real, or string and *format_char* is used for retrieving string value in binary, octal, decimal, hexadecimal, or ASCII format. Note that *format_char* is only useful for getting parameter value from Verilog parent. It can be 'b' or 'B' for binary, 'o' or 'O' for octal, 'd' or 'D' for decimal, 'h' or 'H' for hexadecimal, 'a' or 'A' for the default ASCII format.

In order to pass generics from VHDL to SystemC, you need to provide a stub entity for the SystemC module instantiated in the VHDL parent. The file name of the stub entity must be _<*module_name*>*.vhd* and it must be copied to the <*current work library*>*/_sc* directory after *sccom -link* and before the VHDL parent is compiled. For examples of HDLs passing parameters and generics to SystemC, please refer to the Model*Sim* User's Manual.

SystemC modules can also pass parameters to HDL instances through the arguments of *sc_foreign_module*:

```
sc_foreign_module(sc_module_name nm, const char*
hdl_name, int num_generics, const char** generic_list);
```

The third argument is the number of generics. The fourth argument is an array of string of size *num_generics*. Each array element is a string of the format "*param_name=param_value*". Note that if *param_value* is a string, you need to enclose it with double quotes.

As mentioned before, *scgenmod* can be used to generate stub SystemC modules that stand in for each HDL design unit instantiated under SystemC. *scgenmod* will create extra arguments for passing parameters if parameters exist in the modules. For examples of SystemC passing parameters to HDLs, please refer to the Model*Sim* User's Manual.

Language Templates

The language templates that you are familiar with for Verilog and VHDL have been enhanced in Model*Sim* 6.0 to support SystemC as well. The templates are architected to help you easily create SystemC designs and test benches. They are a collection of wizards, menus, and dialogs that produce code for new designs, language constructs, logic blocks, etc.

To use the templates, select **File > New > Source > SystemC** in the menu bar in the Main window to create a new file. Once the file is open, select **View > Source > Show Language Templates**. This displays a pane that shows the available templates. Double-click an item on the list to begin creating code. Some of the items bring up wizards while others insert code into your SystemC file. The templates work for Verilog and VHDL as well. You can easily create a mixed-language design using this feature. Please refer to the Model*Sim* User's Manual for detailed description of the language templates.

Debug Capabilities

The integrated debug features that apply to Verilog and VHDL designs in Model*Sim* will apply to SystemC designs in phases. Several enhancements have been made in Model*Sim* 6.0. For example:

- Primitive channels such as *sc_buffer, sc_fifo, sc_mutex,* and *sc_semaphore* are now supported for debug in addition to *sc_signal*.
- Member variables inside an *SC_MODULE* are now supported for debug.
- Aggregates of signals and ports are now supported for debug. Aggregates may be arrays, classes, or structures where all members are signals or ports.
- As discussed earlier, test bench development is made easier with the probing and forcing capability of HDL signals from SystemC modules.
- Windows is now a supported platform for SystemC. MinGW gdb 6.0 is part of the standard installation to enable debugging on Windows.

Let us consider a primitive channel *sc_fifo*. In Model*Sim*, the values contained in *sc_fifo* appear in a definite order. The top-most or left-most value is always the next to be read from the FIFO. The command *examine* can be used to examine a single element or multiple elements of the FIFO and display the current writing to and reading from an *sc_fifo<long>* which has ten elements. Note that if you try to examine an empty element explicitly by itself (*examine {fifo(7)}*) as shown in the example below, ModelSim will display "-*Unused*-".

```
# 100: writing 0
# Executing 'examine fifo' yields
# {{0}}
# 200: writing 1
# Executing 'examine {fifo(0 to 1)}' yields
# {{0 1}}
# 300: writing 2
# 400: writing 3
# 500: writing 4
# 600: writing 5
# 700: writing 6
# Executing 'examine {fifo(0 to 7)}' yields
# {{0 1 2 3 4 5 6}}
# Executing 'examine {fifo(7)}' yields
# -Unused-
# 800: writing 7
# 900: writing 8
# 1000: writing 9
# 1100: Available: 10
# 1100: reading 0
# 1200: Available: 9
# 1200: reading 1
# {{2 3 4 5 6 7 8 9}}
# 1300: Available: 8
# 1300: reading 2
# 1400: Available: 7
# 1400: reading 3
# 1500: Available: 6
# 1500: reading 4
# 1600: Available: 5
# 1600: reading 5
# 1700: Available: 4
# 1700: reading 6
# {{7 8 9}}
```

Figure 4: Sample sc_fifo Outputs

You may also view the contents of the FIFO in the Wave window:

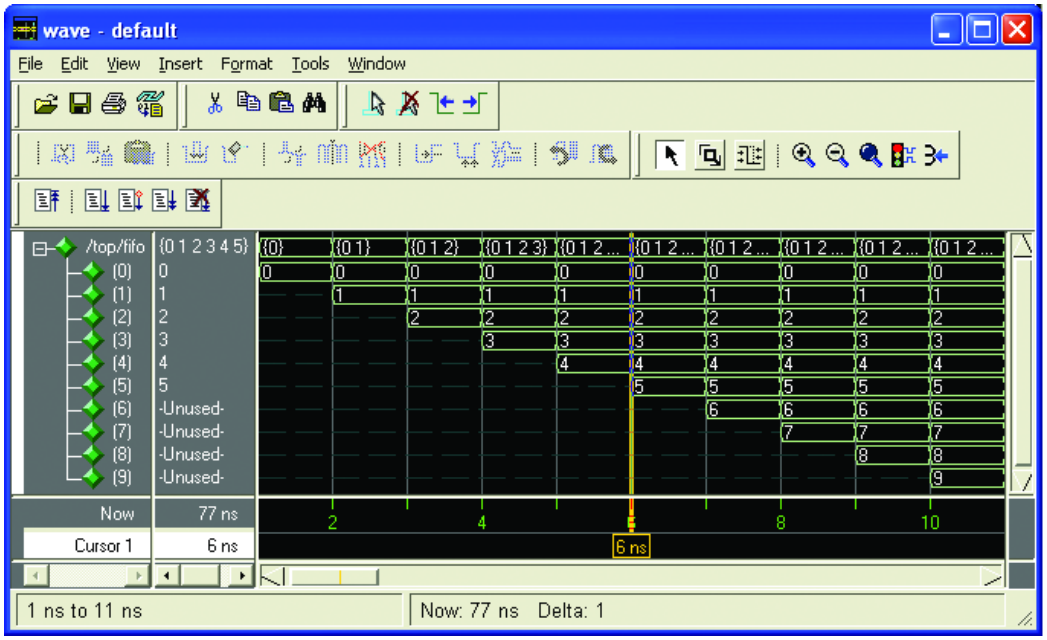

Figure 5: Sample sc_fifo Waveforms

The following example illustrates some of the other debug techniques for SystemC in Model*Sim*. You may have already downloaded the source code of the example along with this application note. Otherwise, you may first register and then download technical information at **http://echo.model.com/model/appnotes/techpubs_reg.asp**.

The design is a ring buffer. Data comes into the buffer at a constant rate and then it is sent back out in frames at a higher rate. The design is implemented in SystemC, VHDL, and Verilog. Figure 6 shows the design hierarchy.

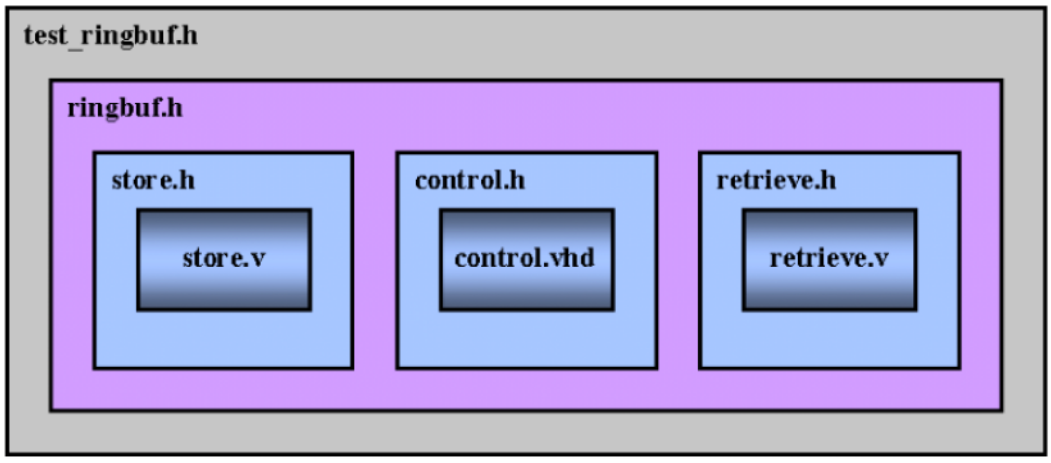

Figure 6: Sample Design Hierarchy

The test bench (*test_ringbuf.h*) and the top level chip (*ringbuf.h*) are implemented in SystemC. There are lower level modules implemented in Verilog and VHDL (*store.v*, *control.vhd*, *retrieve.v*). In other words, we have a SystemC-over-HDL configuration. The *scgenmod* command is used to generate the stub SystemC modules (*store.h*, *control.h*, *retrieve.h*). Examining the stub SystemC modules reveals that they are simply wrappers with the corresponding HDL design unit declared as *sc_foreign_module*.

The blocks store, control, and retrieve are instantiated in the top level chip (*ringbuf.h*). Note that the stub SystemC modules (*store.h*, *control.h*, *retrieve.h*) are included in *ringbuf.h*.

The test bench (*test_ringbuf.h*) includes and instantiates *ringbuf.h*. It also declares, registers, and implements various processes such as *clock_generator*, *reset_generator*, *generate_data*, etc. The *test_ringbuf.cpp* file is merely a wrapper for *test_ringbuf.h* so that it can be compiled by *sccom*.

The commands to compile, link, and load the design are as follows:

```
vlib work
sccom -g -I<path to stub SC modules> test_ringbuf.cpp
sccom -link
vcom -93 control.vhd
vlog store.v retrieve.v
vsim test_ringbuf -do run.do
```

The integrated C Debugger in Model*Sim* is a very useful tool for debugging SystemC designs. The required version of *gdb* is automatically installed during installation and will be picked up by Model*Sim* when the C Debugger is used. If you have specific reasons to use your own version of *gdb*, you may use the command *cdbg_set_debugger*, as shown in the *run.do* macro file, to specify the location of *gdb*. However, we strongly recommend against the use of customized *gdb*. The command *cdbg_balloon_on* enables the source balloon so that if you hover the mouse cursor over a variable in the Source window the current simulation value of that variable is displayed. Please refer to the Model*Sim* User's Manual for more information about the C Debugger.

Executing the commands mentioned above compiles, links, and loads the design. In the Model*Sim* GUI. The Workspace window displays the design hierarchy. In addition to the light blue and dark blue icons, which represent Verilog and VHDL objects, respectively, SystemC objects are represented by green icons. Expand the *ring_INST* instance by clicking on the + icon to see the instantiated Verilog and VHDL blocks. The circular icons that you see when you expand, for example, the *block1* instance indicate design processes. Selecting a process, *outstrobe_gen* for example, will open the Source window with an arrow pointing to the corresponding code. The same process is also highlighted in the Active Processes window which displays all processes which are ready to be executed. If the process that you select contains local (internal) variables, they will be shown in the Locals (formerly the Variables) window.

When you transverse the design hierarchy in the Workspace window, the design objects (constants, nets, registers, signals, variables, generics, and parameters) in the selected region are synchronously displayed in the Objects (formerly the Signals) window. You can always right-click on an item in the Workspace window and select View Declaration or View Instantiation to locate the selected item in the source code.

You can easily add waveforms to the Wave window for analysis by simply dragging an entire design region from the Workspace window or individual signals from the Objects window to the Wave window. You can do the same to add signals to the List window to observe simulation deltas. For post-simulation analysis purposes, SystemC waveforms can be exported in VCD as well as WLF formats.

You may control simulation runs by using the tool bars in the Main window and Wave window. You can advance simulation time by pressing the Run button in conjunction with specifying the Run Length. Simulation may be interrupted at any time by using the Break button. Break-points are supported for SystemC code. If you place a break-point on an executable line of SystemC code, you are taken to the *CDBG>* prompt when simulation stops on that line. Model*Sim* recognizes that you are debugging SystemC code automatically. Note that you can use the *Step* and *Step Over* functions to single-step your SystemC code the same way you do with Verilog and VHDL code.

To examine SystemC signal and variable values at the current simulation time, you may hover the mouse cursor over the signal or variable of interest in the Source window to see its value in the pop-up balloon. There is a new Monitor window in Model*Sim* 6.0. You can drag and drop any SystemC primitive channels and ports to it to observe their values throughout the entire simulation.

If you prefer to control simulation on the command line, the commands that you are familiar with, such as *bp*, *bd*, *examine*, *step*, *run*, *change*, *add*, and *delete* continue to apply to SystemC designs in the Transcript window. Please refer to the Model*Sim* Command Reference for detailed information about these commands.

Beginning in Model*Sim* 6.0, you can use *verror <error number>* to obtain more information on SystemC errors that you may encounter.

The Dataflow window allows you to explore the physical connectivity of your design; to trace events that propagate through the design; and to identify the cause of unexpected outputs. Currently such capabilities apply to Verilog and VHDL designs only. The support for SystemC designs in the Dataflow window will be available in a future release.

To learn about other debug features in Model*Sim*, please consult the Introduction to Model*Sim* 6.0 Debug GUI application note and the Model*Sim* User's Manual.

Summary

Model*Sim* provides a unified kernel for SystemC, Verilog, and VHDL. Our native implementation treats SystemC as just another design language. The simulation methodology and debug capabilities that designers have grown to praise continue to apply to SystemC designs. If you are familiar with the Model*Sim* methodology, simulating and debugging SystemC designs require no additional training. The same flow and features that apply to Verilog and VHDL designs apply to SystemC as well. For those who are new to Model*Sim*, you will be delighted by our powerful and easy-to-use verification environment. Our solutions allow you to evaluate and adopt SystemC in your methodology very efficiently. Furthermore, our integrated environment can also help you significantly shorten your verification cycle. For detailed technical information, please refer to the Model*Sim* User's Manual or contact our Support Center at **support@model.com**.

For more information, call us or visit: www.model.com

Corporate Headquarters
Mentor Graphics Corporation
8005 S.W. Boeckman Road
Wilsonville, Oregon 97070 USA
Phone: 503-685-7000
North American Support Center
Phone: 800-547-4303
Fax: 800-684-1795

Silicon Valley
Mentor Graphics Corporation
1001 Ridder Park Drive
San Jose, California 95131 USA
Phone: 408-436-1500
Fax: 408-436-1501

Europe
Mentor Graphics
Deutschland GmbH
Arnulfstrasse 201
80634 Munich
Germany
Phone: +49.89.57096.0
Fax: +49.89.57096.400

Pacific Rim
Mentor Graphics Taiwan
Room 1603, 16F,
International Trade Building
No. 333, Section 1, Keelung Road
Taipei, Taiwan, ROC
Phone: 886-2-27576020
Fax: 886-2-27576027

Japan
Mentor Graphics Japan Co., Ltd.
Gotenyama Hills
7-35, Kita-Shinagawa 4-chome
Shinagawa-Ku, Tokyo 140
Japan
Phone: 81-3-5488-3030
Fax: 81-3-5488-3031

C&A 7-04 TECH6460-w