

Applications Note – 113

Using ModelSim PE 5.X With Xilinx Alliance Software

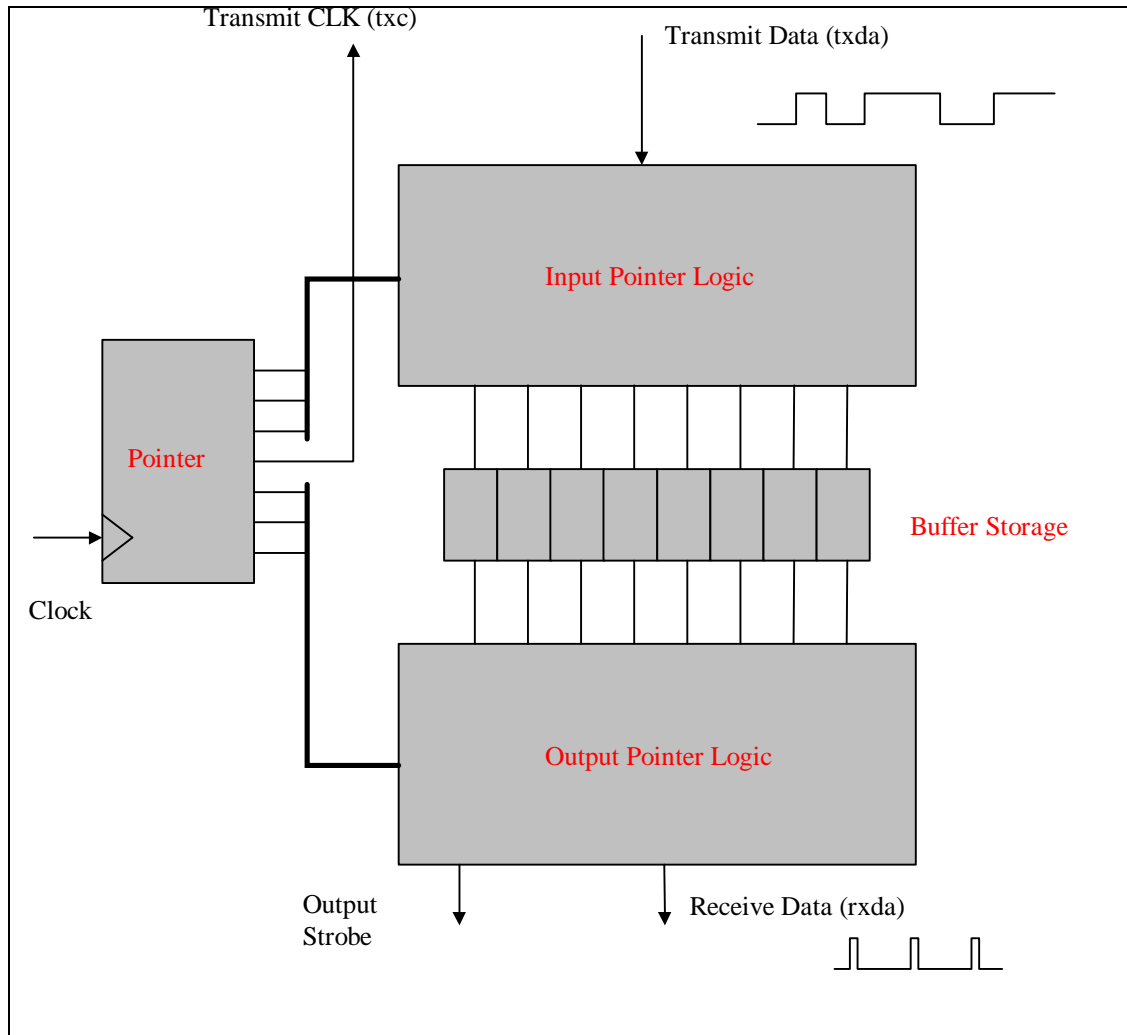
22 January 1999 Revision 1.0

Using ModelSim PE 5.X With Xilinx Alliance Software.....	1
Section 1. ModelSim RTL Flow	2
Section 2. Synthesis and Place and Route.	10
Section 3. Compiling Gate Level Libraries for Simulation	14
Compiling the VHDL gate level libraries with the ModelSim UI.....	14
UNISIM	14
SIMPRIM.....	16
Compiling the VHDL gate level library using the command line	17
UNISIM	17
SIMPRIM.....	18
Compiling the Xilinx Verilog gate level library with the ModelSim UI.....	18
UNISIM	18
SIMPRIM.....	20
Compiling the Xilinx Verilog gate level library using the command line	21
UNISIM	21
SIMPRIM.....	22
Section 4. Compile and Timing Simulation of the VHDL Gate Level Source Files	23
Section 5. Compile and Timing Simulation of the Verilog Gate Level Source Files	27
Section 6. Using LogiBLOX and CoreGen Models	31
LogiBLOX.....	31
CoreGEN	33
For more information	35

ModelSim is a single kernel, dual language simulator. You are able to run either Verilog or VHDL separately or mixed in the same design. You can have Verilog modules instantiated in VHDL architectures or VHDL entities instantiated in Verilog Modules. You can even mix languages at any level of abstraction and with any number of hierarchical levels i.e Verilog module instantiated by a VHDL architecture called from a Verilog module. One simulator, one interface, two languages. Xilinx Alliance software is capable of outputting both Verilog and VHDL netlists, there are no restrictions on which language you choose. ModelSim uses compiled HDL libraries, you must first compile the Xilinx supplied libraries. The Xilinx Alliance ModelSim flow for each language is slightly different and is described in this document. Each language flow will be described with command line and User Interface examples. A language label will be attached to the text detailing the appropriate language. It is possible to skip the sections for the language that you do not use.

Section 1. ModelSim RTL Flow

The design used in this example is very simple and easy to understand. It is a simple ring buffer from a data communications application. Transmit Data is input into the buffer at a constant rate clocked by the transmit clock (txc). The storage location address of the incoming data for transmission, transmit data (txda) is pointed to by a counter. Each location of the buffer is pointed to in turn in a sequential fashion. The output pointer logic is driven by the output of lower order bits of the same counter and an output strobe. This generates a valid output location address for the receive data. This results in the data being output in a burst fashion, a diagram of the design is shown below.



The design has been decomposed into three blocks, these three blocks are connected by a netlist. The design itself is driven by a testbench that provides both stimulus and some self checking routines. The complete set of files are available in both VHDL and Verilog. VHDL files have the .vhd extension and Verilog files have the .v extension. The size of the design can be controlled by two parameters / constants that are set at the top level. counter_size defines the size of the counter necessary to address the buffer and buffer_size defines the length of the storage buffer. The testbench generates data for the ring buffer with a Pseudo random data pattern generator. The Pseudo random data is generated using a 20 bit Linear Feedback Shift Register, the LFSR is set up to produce a 2^{20} Pseudo random patterns. The testbench also includes self checking routines that analyse the data output from the ring buffer and print messages to warn of any differences.

ModelSim can be used in batch mode, command line, or with the User Interface (UI). Batch mode is the typical method when running regression tests. Command line mode is very similar to batch mode in the fact that the UI is not displayed, the only interface is a command line console. The user interface mode can accept both command line, and UI input. The following commands can be used in any mode. Note that the view * (view all windows) and the add wave /* (add all signals at top level to wave form window) will show results only when in UI mode. If you save these commands in a file ("xilinx_rtl.do" is a common macro file naming convention) it can be used in UI mode, command, or batch modes.

VHDL User

```
cd <design directory>
vlib work
vmap work work
vcom control.vhd retrieve.vhd store.vhd Ringrtl.vhd
vcom Testring.vhd config_rtl.vhd
vsim work.test_bench_rtl
view *
add wave /*
run 100000000
```

Verilog User

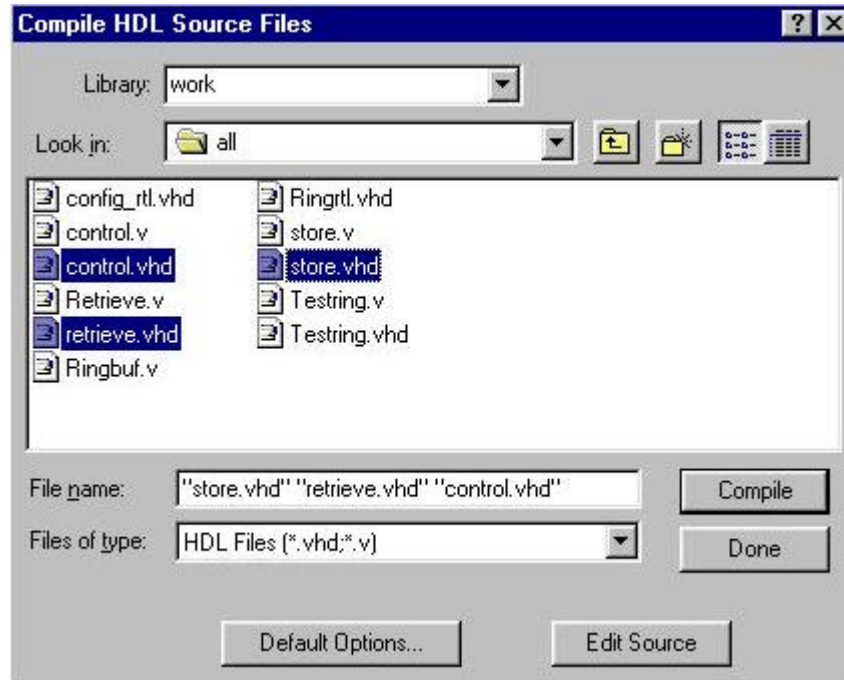
```
cd <design directory>
vlib work
vmap work work
vlog control.v retrieve.v store.v Ringrtl.v
vlog Testring.v
vsim work.test_ringbuf
view *
add wave /*
run 100000000
```

The steps above are explained below and should be carried out using the UI in the following way. Start ModelSim by double clicking the icon or modelsim.exe.

1. Change into the design directory. Select **File** ➤ **Change Directory**, use the file browser to locate the desired directory. This directory becomes the working directory. Any library that is now created will be placed into this directory by default.
2. Create a working library. Select **Library** ➤ **Create A New Library**, ensure that "a new library and logical mapping to it" is selected, enter work in the library box as shown below.



3. Select **OK** to except the entry, this will execute both the vlib and vmap commands shown in the command line script listing above.
4. Compile the design files. Select the **Compile** button. This is the first button on the left hand side and will start the Compile HDL Source Files menu. This menu will automatically select the correct language compiler based upon the source file selected. (vcom for VHDL source files and vlog for Verilog source files).



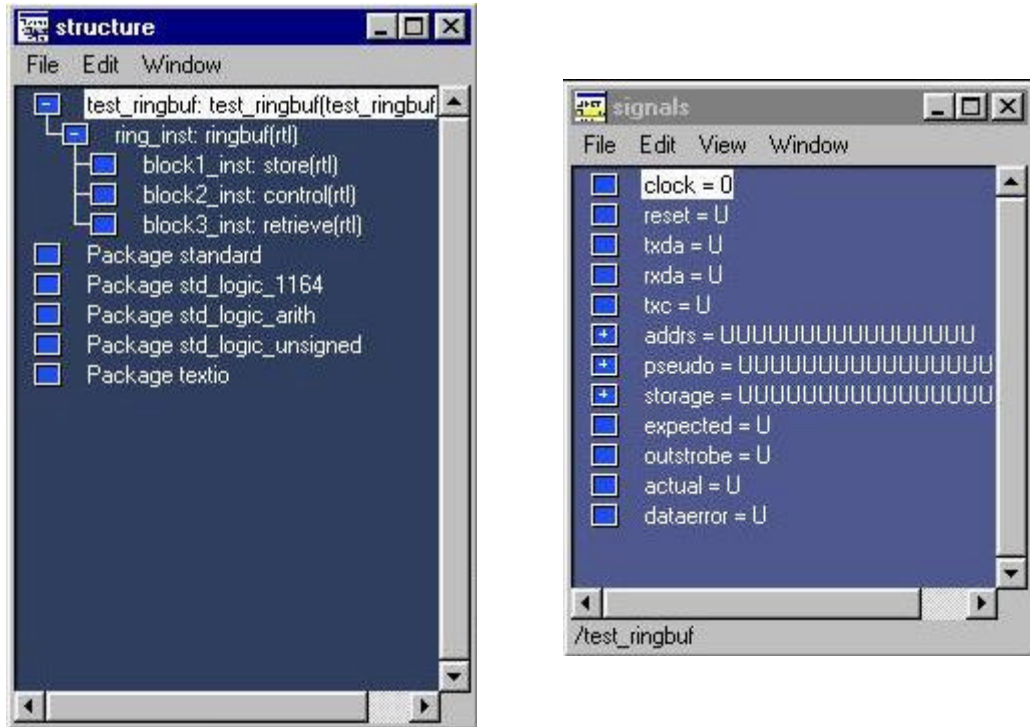
For the VHDL design select the lower level blocks, control.vhd, retrieve.vhd, store.vhd and then select the **Compile** button. (Multiple select is achieved by holding down the control key and selecting with the mouse button). Then compile the following files in the order stated by selecting the file and then selecting the Compile button. Select the top level design block Ringrtl.vhd, the testbench file Testring.vhd and finally the configuration file config_rtl.vhd. It is also possible to compile a single file by simply double clicking with the mouse on the desired file. For the Verilog design select all the .v files and then select the **Compile** button.

5. Loading the simulator. Select **File ▾ Load New Design** or select the Load Design button. This will display the Load Design dialog box, which displays the design units available in each of the libraries. Ensure that the design tab is highlighted. Ensure that the work library is visible in the library section. Each of the units in the work library should be displayed along with a description of the type of unit. The description will be entity, architecture, config or package for VHDL units and module for Verilog units. Each unit type is also highlighted with a different colour. At the beginning of the line with a VHDL entity there in a '+' sign. Toggling this '+' sign with the mouse shows the architectures that have been compiled for the entity. For the VHDL design select the configuration **test_bench_rtl** and then the **load** button. For the Verilog design select the test bench module **test_ringbuf** and then the **load** button. This will load each of the design units needed for the simulation of either the VHDL configuration or the Verilog testbench module. Note that the cursor line in the ModelSim main window changes to VSIM > this indicates that simulation mode in active.
6. Opening debug windows. Select **View ▾ All**, this will open all of the debug windows. To make sure that all of the windows are positioned without any overlap select **Window ▾ Initial Layout**. The complete screen should be filled with the nine windows including the main ModelSim window. Each of these windows are explained in the following text.



VHDL User

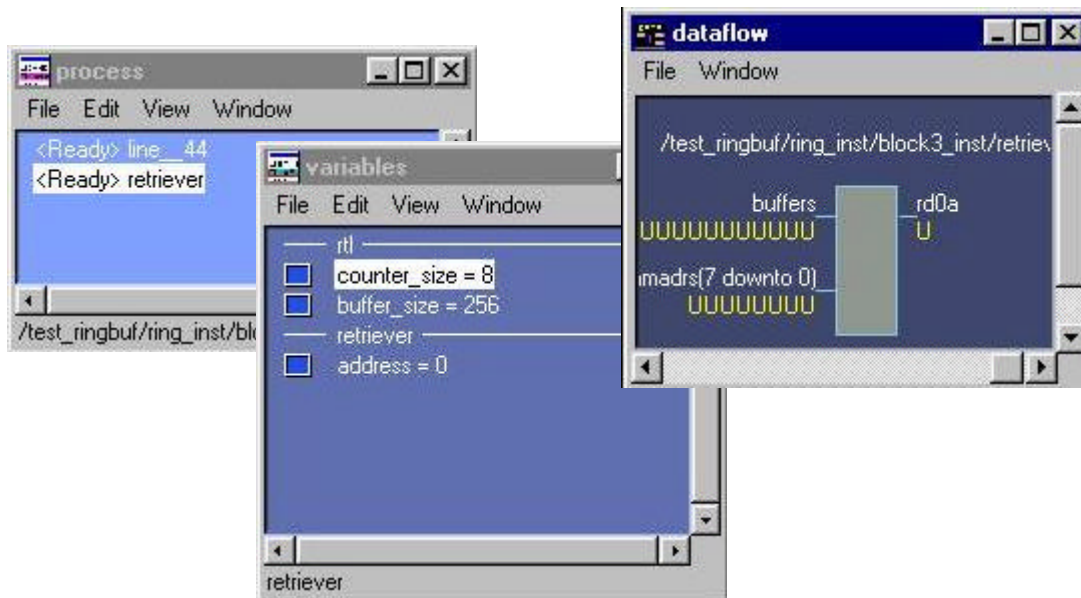
The structure window can be used to explore the design hierarchy, each of the squares represent a VHDL design unit. A plus sign inside the square shows that there are child units and by clicking with the mouse the hierarchy can be expanded and collapsed. The signal window lists the signals at the level of hierarchy pointed to by the structure window. Using these two windows together any signal in the design hierarchy can be added to the wave window.



All of the signals at a particular level of the hierarchy can be added by dragging the square from the structure window across and dropping it into the wave window. To add all of the signals at the top level select the **test_ringbuf** square and while holding the select mouse key down (left hand button) drag it across and drop it into the wave window. The source window can also be used to locate signals and ports and add them to the waveform window. When a level of hierarchy is selected by the Structure window, the associated source file will be displayed in the source window. It is possible to add signals in the architecture or ports in the entity to the wave window by double clicking the desired signal/port and then dragging and dropping it into the wave window.

It is also possible to drop any of the signals or ports into the list window. The List window displays the results of your simulation run in tabular format. The window is divided into two adjustable panes, which allow you to scroll horizontally through the listing on the right, while keeping time and delta visible on the left. Note that constants, generics, parameters, and memories are not viewable in the List or Wave windows.

The last three windows are the Process, Variables and Dataflow windows. The Process window displays a list of processes and indicates the pathname of the instance in which the process is located. There are two views possible in the process window. The first is a list of the active processes, this is the default view. **Select View** **P In Region** in the Process window, the second view is a list of the processes in the region selected by the Structure window.



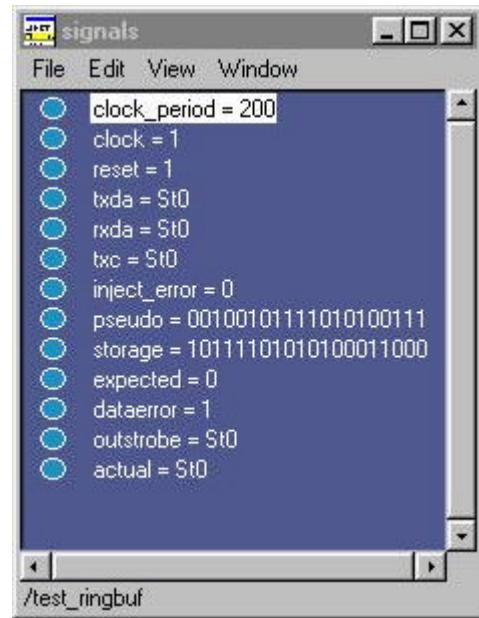
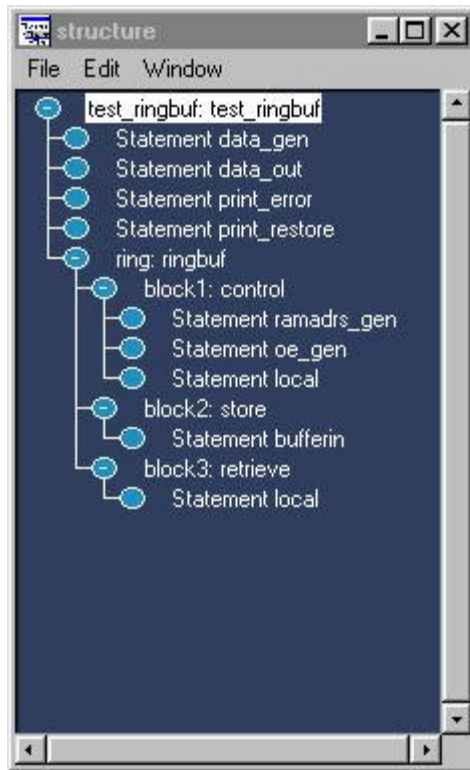
Select **block3_inst** in the Structure window, the process window should show the two processes in this instance. Select the retriever process in the Process window. The process is displayed in the source window, with an arrow pointing to the first executable line in the process. The Variables window displays the variables in the architecture selected by the structure window, in this case the architecture is RTL and the generics counter_size and buffer_size are displayed. It also displays the variable address in the retriever process. The variable address can be dragged and dropped into the list window.

Finally, the Dataflow window allows you to trace VHDL signals through the design. A process is displayed with all the signals read by the process shown as inputs on the left of the window, and all the signals driven by the process on the right. A signal displays in the center of the window with all the processes that drive the signal on the left, and all the processes that read the signal on the right. With the retriever process selected by the Process window, the retriever process will be displayed in the dataview window. Buffers and ramdrs are the inputs to the process displayed on the left hand side of the box and rd0a is the output displayed on the right hand side of the box. Double click on the signal rd0a this will change the view in the Dataview window to show what the signal is driving. In this case it is driving a single process, double click the right hand process name and the Dataflow window will display the new process. The source window also points to the new process, in this case it is a concurrent signal assignment that drives the rxda port.

Verilog User

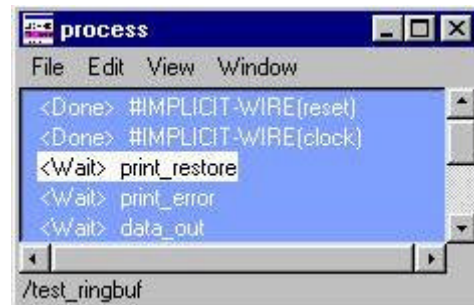
The structure window can be used to explore the design hierarchy, each of the circles represent a Verilog design unit. A plus sign inside the circle shows that there are child units and by clicking with the mouse the hierarchy can be expanded and collapsed. The signal window lists the wires and registers at the level of hierarchy pointed to by the structure window. Using these two windows together any wire or register in the design hierarchy can be added to the wave window.

All of the signals at a particular level of the hierarchy can be added by dragging the circle from the structure window across and dropping it into the wave window. To add all of the signals at the top level select the **test_ringbuf** circle and while holding the select mouse key down (left hand button) drag it across and drop it into the wave window. The source window can also be used to locate wires and registers and add them to the waveform window. When a level of hierarchy is selected by the Structure window, the associated source file will be displayed in the source window. It is possible to add wires and registers in a module to the wave window by double clicking the desired wire/register and then dragging and dropping it into the wave window.



It is also possible to drop any of the wires or registers into the list window. The List window displays the results of your simulation run in tabular format. The window is divided into two adjustable panes, which allow you to scroll horizontally through the listing on the right, while keeping time and delta visible on the left. Note that constants, generics, parameters, and memories are not viewable in the List or Wave windows.

The last three windows are the Process, Variables and Dataflow windows. The Process window displays a list of always blocks and indicates the pathname of the instance in which the block is located. There are two views possible in the process window. The first is a list of the active always blocks, this is the default view. Select **View ▸ In Region** in the Process window, the second view is a list of the always blocks in the region selected by the Structure window.

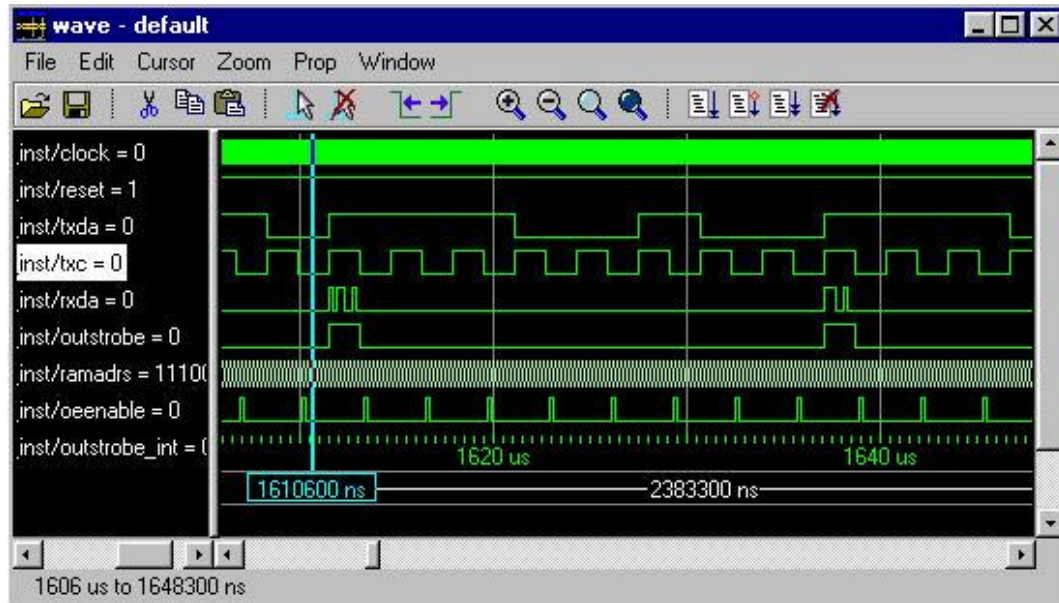


Select **test_ringbuf** in the Structure window, the process window should show all of the always blocks in this instance. Select the **print_restore** always block in the Process window. The always block is displayed in the source window, with an arrow pointing to the first executable line in the always block. The Variables window

displays the variables in the module selected by the structure window. The variables can be dragged and dropped into the list window.

VHDL & Verilog User

- Running the simulation. There are two ways of running the simulation for a specified time using the GUI. The first one is to type run 10000000 for VHDL or run 1000000000 for Verilog at the VSIM prompt in the main window. The other is to enter the time value into the run window in the set of menu icons at the top of the main window, and then hitting the Run button as shown below.



At the end of the run time the wave window will display the results of the simulation. It can be seen from the display that txda is a steady pseudo-random data stream and rxda represents the same data in a burst of eight bits. The number of bits is set by the value of the buffer_size parameter / constant.

- Running the simulation again. If it is necessary to re-run the simulation due to the fact that the design needs to change or it is necessary to single step the design then the simulator can be returned to time zero by using the restart command. Select the toplevel design in the structure window, so that the source is displayed in the source window. Select **Edit** **Read Only** in the source window menu and ensure that the tick is removed. This allows modifications to be made to the source file. Find the instantiation of the ringbuf and modify the Verilog parameters or VHDL constants to increase the size of the buffer. Increase the counter size to 6 and the buffer size to 64.

Verilog Code

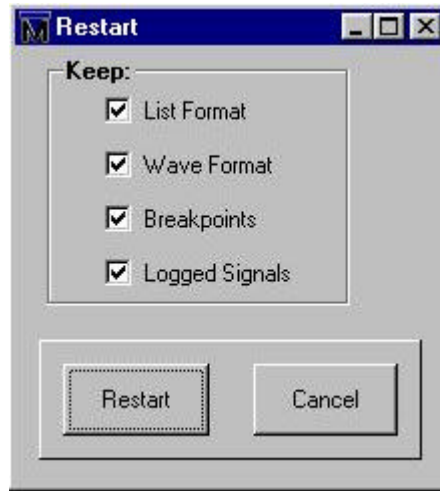
```
module ringbuf ( clock ,reset , txda,
rxda, txc, outstrobe);
// Design Parameters Control Design
parameter counter_size = 6;
parameter buffer_size = 64;
```

VHDL Code

```
outstrobe : OUT std_logic
);
constant counter_size : integer := 6;
constant buffer_size : integer := 64;
END ringbuf;
```

- Select **File** **Save** in the source window menu to save the source file. Select the Compile button. This is the first button on the left hand side and will start the Compile HDL Source Files menu. For the VHDL design compile ringrtl.vhd, followed by testring.vhd and config_rtl.vhd. For the Verilog design compile ringrtl.v.

Select **File ▸ Restart** in the main window menu, this will display the restart menu. The restart command allows the simulation to be restarted with the same list format, wave format, breakpoints and logged signals. Each one of these can be disabled and a new set of signals or breakpoints used. Select the Restart button and then hit the Run button to run for the same period as before. It is possible to toggle through the history of commands by using the up and down arrows in the main window.



- Setting break points. It is possible to add breakpoints to source code at any time during a simulation. Select the top level block in the structure window, this will load the testbench into the source window. On the left hand side of the source window the line numbers are displayed. The simulator can be broken on any line number displayed in green. Line numbers displayed in black are either non-executable or have been optimised away during compilation. Move down to the **generate_data** process or always block and place a breakpoint on the pseudo-random assignment. Breakpoints are set by clicking on the line that you wish to break the simulator, by the line number. This will display a circle next to the selected line, breakpoints can be removed by clicking on the line again. Click on the line of code shown below, this will break the simulator in the pseudo-random data generator.

VHDL User :

The line reads `pseudo <= pseudo(18 DOWNT0 0) & NOT (pseudo(2) XOR pseudo(19));`

Verilog User :

The line reads `pseudo <= {pseudo[18:0],pseudo[2] ^~ pseudo[19]};`

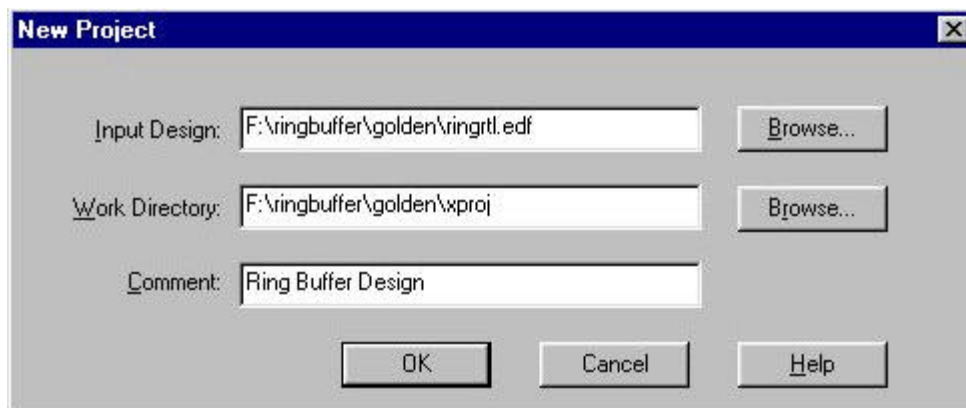
- Select **Run ▸ Run -all** in the main window, simulation will be broken on the line selected. It is now possible to single step through the simulation using the step and step over buttons in either the source or main window. All windows are updated as you step through the simulation.



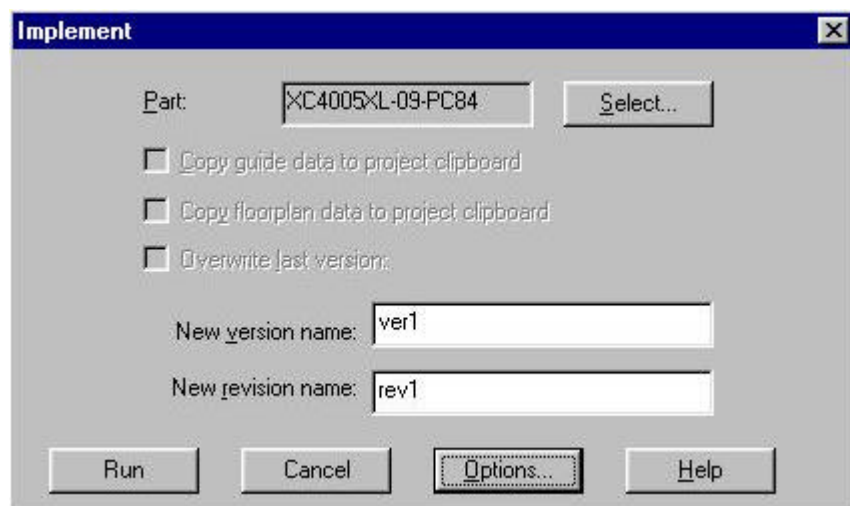
Section 2. Synthesis and Place and Route.

This section details how to run the M1 tool-set and output the files necessary to use in a post layout simulation. As ModelSim can be used with any Synthesis tool the implementation steps have not been included in this document. It is possible to use any of the available Synthesis tools to produce an EDIF file to use as input to the place and route tool. The VHDL and Verilog source files for this Application Note are available on the Model Technology Web site along with an EDIF file to use as input to this section if you do not have access to a synthesis tool. The following section details the options when using the Design Manager GUI. The details on which programs need to be run to implement the design using the command line are shown at the end of this section.

1. Start the Xilinx Design Manager and select **File ▸ New Project**. This will display the New Project dialog box. Select the **Browse** button on the input design line and find the input EDIF file, this will be the output generated from your synthesis tool or the supplied EDIF file called ringrtl.edf. If your synthesis tool has output an ncf file, this file includes constraints passed by the Synthesis tool, then ensure that it is in the same directory as the EDIF netlist. The working directory will be automatically filled out, creating by default an xproj directory in the directory of the input netlist file. This directory will contain all of the working files and report files used for the design during place and route. Select the **OK** button.

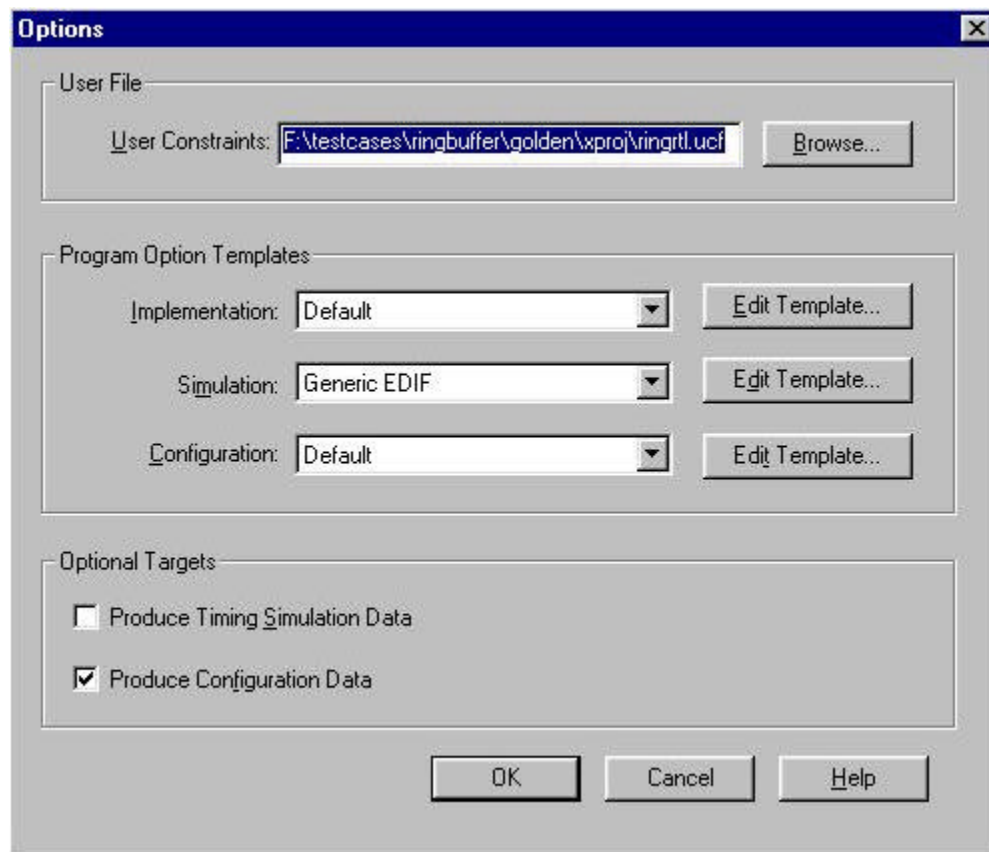


2. Once the EDIF netlist has been parsed, the next step is to implement the design. Select **Design ▸ Implement**, this will display the implement dialog box. Most



synthesis tools will pass on the details of which device and package you are using so the part information will have been automatically filled out. If it has not then select the **Select** button next to the part selection box and locate the device you wish to use. This design example will fit the smallest XC4000 part, as long as the parameters / constants are kept to the size stated in the last section, so select XC4005XL-09-PC84.

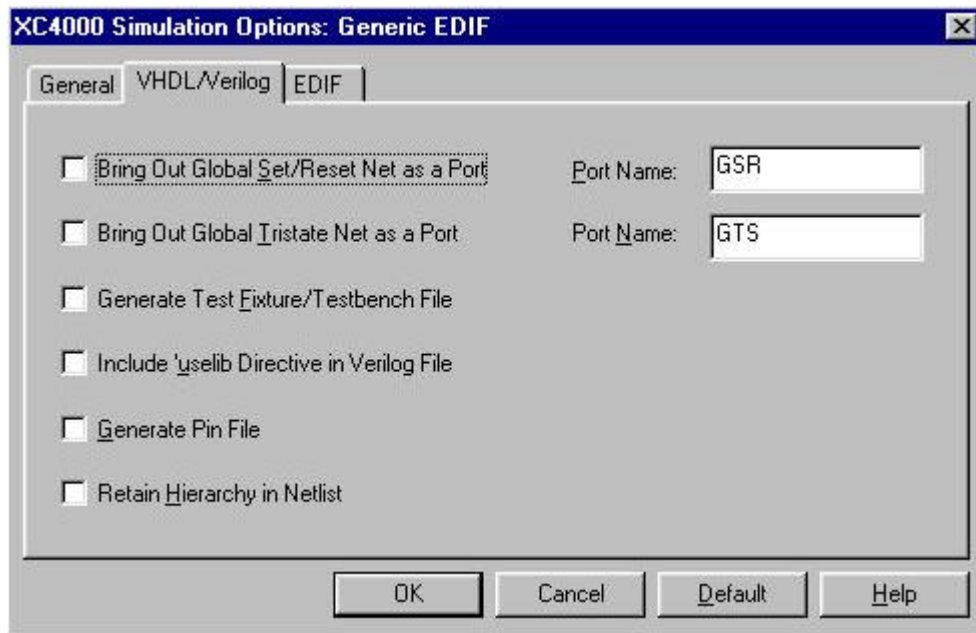
3. To set up the options for this implementation select the **Options** button at the bottom of the dialog box. This displays the options menu. Under the Program Options Templates there is a Simulation section, by default this is set to Generic EDIF.



Select the down pointing triangle on the right side of the Simulation box. This will display a list of output netlist formats. If you require a Verilog netlist select ModelSim Verilog, and if you require a VHDL netlist select ModelSim VHDL. This automatically adjusts the Simulation Template to the settings required for each ModelSim product. Select the Edit Template option next to simulation, this will display the template edit dialog box. There are three tabs on this dialog box, the first is General. Here the Simulation netlist type will have been set to either Verilog or VHDL depending on last selection.

4. There are two other options on the General tab, the first is Correlate Simulation Data input to design. With this option selected the resulting netlist will attempt to mimic the same logic gates and net names as those in the original input EDIF netlist. Deselection of this option will create netlist that contains the same logic gates and net names as those in the optimised implemented netlist. There is a possibility that gates implemented by the Synthesis tool maybe optimised out during any stage of implementation. This means that often some gates disappear or their function gets implemented by other gates. Make sure this option is deselected to save processing time if correlation is not required. If this option is selected, warnings will be reported if there are any mis-matches between the input and implemented netlists. The Simulation Netlist Name can be set to the desired name, by default it is time_sim. Select the VHDL/Verilog tab, by default all of the check boxes are deselected. Each of these options are described below:

Bring Out Global Set/Reset as a Port - If this is de-selected the Global Set/Reset will be implemented in the netlist as a VHDL statement when targetting VHDL. At the top of the netlist file there is a statement that waits for 100nSecs and then removes reset. This time delay can be changed by modifying the output netlist. Selecting this option creates a Global Set/Reset port on the top-level simulation module or entity. This port is connected to all flip-flop and latch primitives in the design. Stimulating this port automatically sets or resets every flip-flop and latch to its initial state, as determined in the design. The Port Name field can be used to change the default port name.



Bring Out Global Tristate as a Port - The option does the same as the last but for the global tri-state pin.

Generate Test Fixture/Testbench File - This option writes out a Verilog test fixture file or a VHDL test bench file depending on the chosen language. The test fixture file has a .tv extension and the test bench file has a .tvhd extension. The file includes an instantiation of the implemented design.

Include `uselib Directive in Verilog File -This is a Verilog option and writes a library path pointing to the SIMPRIM library into the output Verilog (.v) file. The path is written as follows :

```
`uselib dir=$XILINX/verilog/data libext=.vmd
```

where \$XILINX is the location of the Xilinx software. The Xilinx gate level libraries are explain in the following section.

Generate Pin File - This option writes out a netlist signal to physical pin mapping file that can be useful during Simulation debug.

Retain Hierarchy in Netlist -This option writes out a Verilog HDL or VHDL file that retains the hierarchy in the original input design netlist, grouping logic based on the original design hierarchy.

Select the options that are required for the output netlist and Select **OK**. For information, the EDIF tab in the simulation options template is for setting EDIF only options

5. On the Optional Targets section of the options dialog box there is a Produce Timing Simulation Data check box. Check this box to output the netlist configured to the options that have been chosen. Note without this checkbox selected no netlist files will be written. If required then change the options in the implementation and configuration templates. Select **OK** on the options dialog box and then **RUN** on the implement dialog box. This will open the flow engine the design will be translated, mapped, placed and routed, timing generated and finally a configuration bit map will be produced. The time_sim.v or time_sim.vhd netlist file will be placed in the working directory along with the back annotation file, time_sim.sdf. The following is a script that can be used on the PC to implement the device in batch mode. All of the commands needed are shown.

REM Build The NGD Database

```
ngdbuild -p xc4005xl-09-pc84 -uc ringrtl.ucf ringrtl.edf ringrtl.ngd
```

REM Map Logic To Device

```
map -p xc4005xl-09-pc84 -o map.ncd ringrtl.ngd ringrtl.pcf
```

REM Place And Route Design

```
par -w -ol 2 -d 0 map.ncd ringrtl.ncd ringrtl.pcf
```

REM Static Timing

```
trce ringrtl.ncd ringrtl.pcf -e 3 -o ringrtl.twr
```

REM Extract Timing Data

```
ngdanno ringrtl.ncd
```

REM Write Netlist And SDF Data

```
Verilog User : ngd2ver -w ringrtl.nga time_sim.v
```

```
VHDL User : ngd2vhdl -w ringrtl.nga time_sim.vhd
```

REM Create Bitmap File

```
bitgen ringrtl.ncd -l -w -f bitgen.ut
```

Note typing ngd2ver OR ngd2vhdl at the command line will return all the options for these programs. This will allow the options explained in the simulation options to be set on the command line.

Section 3. Compiling Gate Level Libraries for Simulation

This section details where to find the source files for the Xilinx gate level libraries and how to compile them ready for simulation in ModelSim. It covers the four options created by the ability to use either the GUI or command line in ModelSim for either VHDL or Verilog. Xilinx have models to support HDL designs at three different points in the design flow. It is important to understand which libraries are used at each stage of the design flow. Simulation is supported at the RTL level by being able to instantiate UNISIM library components, LogiBLOX modules and CoreGEN models. Gate level post synthesis simulation is supported by the UNISIM library components. Pre-route simulation and post implementation back-annotated timing simulation is supported by the SIMPRIM library. Both the UNIPRIM and SIMPRIM libraries adhere to IEEE-STDs. The VHDL libraries use the VITAL IEEE-STD-1076.4 standard, and the Verilog library uses the IEEE-STD-1364 standard. ModelSim fully accelerates the VITAL_Timing and primitive VITAL libraries.

The UNISIM Library is used for functional simulation only, and contains default unit delays. This library includes all the Xilinx Unified Library components that are inferred by most popular synthesis tools. In addition, the UNISIM Library includes components that are commonly instantiated, such as IOs and memory cells. The cells in the UNISIM library are device dependant.

The SIMPRIM models have the appropriate functionality to allow back-annotation of timing information using an SDF (Standard Delay Format) File. The netlist output by ngd2ver and ngd2vhdl contain instantiations of SIMPRIMs models that can be annotated with the generated SDF file containing the appropriate block and net delay data from the place and route process. The SIMPRIM library is completely device independent, it is purely a method of modeling the timing within the silicon.

The use of LogiBLOX and CoreGEN models is covered in the section titled 'Using LogiBLOX and CoreGen Models'.

If the Xilinx A1 software is loaded on the same machine as ModelSim there will be an environment variable called XILINX. This environmental variable will be set and pointing to the Xilinx A1 tools installation. This variable is used in the next sections to reference where to find the appropriate source files.

NB – The logical library names used for both the VHDL and Verilog sections are the same for each library therefore attempting to follow both language sections consecutively will cause the Model's to be overwritten.

Compiling the VHDL gate level libraries with the ModelSim UI

UNISIM

The UNISIM (UNified SIMulation) libraries are only used for simulating at the RTL level, and pre-NGDBUILD stage of the design flow. The cells contained in the library are device dependant and only include unit delay timing. The VHDL source code for the libraries can be found at : \$XILINX/vhdl/src/unisims. This directory contains the following files.

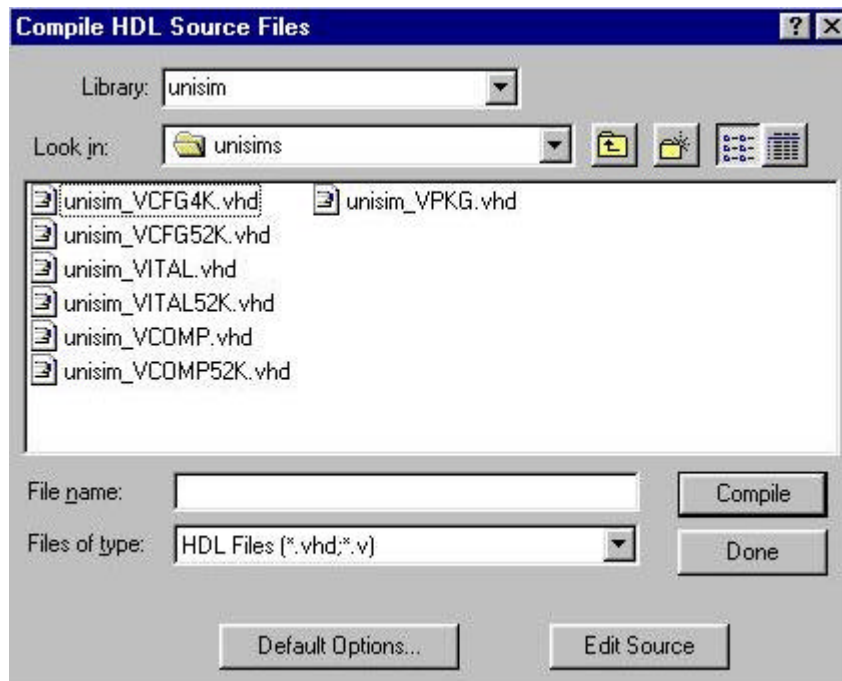
- unisim_VCOMP.vhd (component declaration file)
- unisim_VCOMP52K.vhd (substitution component declaration file for XC5200 designs)
- unisim_VPKG.vhd (package file)
- unisim_VITAL.vhd (model file)
- unisim_VITAL52K.vhd (additional model file for XC5200 designs)
- unisim_VCFG4K.vhd (configuration file for XC4K edge decoders)
- unisim_VCFG52K.vhd (configuration file for XC5200 internal decoders)

As can be seen there are device specific files due to the fact that some models within the difference families have slightly different functionality. For this reason , not all of these files can be compiled into the same library. To be able to use both 4K and 52K family libraries, they have to be compiled into separate directories as a UNISIM library. Then for each design the mapping of the UNISIM logical name would need to be changed to the appropriate directory.

1. Change into the directory in which you wish to store the compiled libraries. Select **File > Change Directory**, use the file browser to locate the desired directory. Any library that is now created will be placed into this directory by default.
2. Create a unisim library. Select **Library > Create A New Library**, ensure that "a new library and logical mapping to it" is selected, enter 'unisim' in the library box as shown below.



3. Select **OK** to except this entry, this will execute both the vlib and vmap commands to create a unisim library and directory with the same name in the working directory.
4. Compile the unisim source files. Select the **Compile** button. This will display the Compile HDL Source Files dialog box. Set the library to unisim using the pull down menu. Use the 'Look in' selection to locate the UNISIM source files. These will be



found in the Xilinx installation in \$XILINX/vhdl/src/unisims.

The order of compilation is important for VHDL, the files and the compilation order for each of the families is shown below. Compile each file in turn by selecting and pressing compile or double clicking on the file.

XC5200 Devices	All Devices Except XC5200
unisim_VCOMP52K.vhd unisim_VPKG.vhd unisim_VITAL.vhd unisim_VITAL52K.vhd unisim_VCFG52K.vhd	unisim_VCOMP.vhd unisim_VPKG.vhd unisim_VITAL.vhd unisim_VCFG4K.vhd

The UNISIM library is now ready to be used for simulation. If both families are required then these two sets of files can be compiled into two separate UNISIM libraries, each located in a different directory. Then use the 'a map to an existing library' option in the create library dialog.

SIMPRIM

The SIMPRIM libraries are used for simulations post-implementation, the design stages include post-NGDBuild, post-MAP and full timing simulation post place and route. The SIMPRIM VHDL libraries are written using VITAL libraries, the packages defined by the standard are fully accelerated by ModelSim. VITAL libraries include some overhead for timing checks and back-annotation of timing data. The SIMPRIM back-annotation library keeps these checks on by default; however, you or your system administrator can turn them off. You must edit and re-compile the SIMPRIM components file after setting the generics. The VHDL source code for the libraries can be found at : \$XILINX/vhdl/src/simprims. This directory contains the following files.

- simprim_Vcomponents.vhd (VITAL Component Package)
- simprim_Vpackage.vhd (VITAL Table Package)
- simprim_VITAL.vhd (Architecture(VITAL) and Configurations)

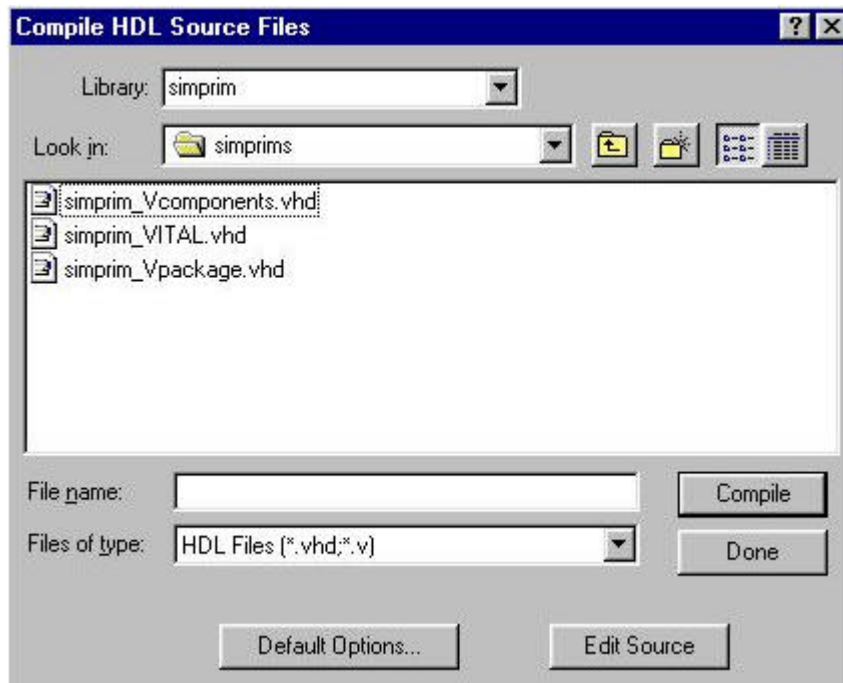
These libraries are completely design independent therefore are used for the timing simulation of any Xilinx device family.

1. Change into the directory in which you wish to store the compiled libraries. Select **File** **Change Directory**, use the file browser to locate the desired directory. Any library that is now created will be placed into this directory by default.
2. Create a simprim library. Select **Library** **Create A New Library**, ensure that "a



new library and logical mapping to it" is selected, enter 'simprim' in the library box as shown below.

3. Select **OK** to except this entry, this will execute both the vlib and vmap commands to create a simprim library and directory with the same name in the working directory.
4. Compile the simprim source files. Select the **Compile** button. This will display the



Compile HDL Source Files dialog box. Set the library to simprim using the pull down menu. Use the 'Look in' selection to locate the SIMPRIM source files. These will be found in the Xilinx installation in \$XILINX/vhdl/src/simprims. The files need to be compiled in the following order Vcomponents, Vpackage and VITAL as shown in the list above. The SIMPRIM library is now ready to be used for simulation.

Compiling the VHDL gate level library using the command line

UNISIM

The following commands need to be executed on the command line to compile each of the UNISIM source files. The cells contained in the library are device dependant and only include unit delay timing. This means that there is a different set of files that need to be compiled to allow simulation of the XC5200 or the XC4000 families. Below are the two scripts for each family.

XC5200 Devices

```
cd <directory storage location>
vlib unisim
vmap unisim unisim
vcom -work unisim {$XILINX/vhdl/src/unisims/unisim_VCOMP52K.vhd }
vcom -work unisim {$XILINX/vhdl/src/unisims/unisim_VPKG.vhd }
vcom -work unisim {$XILINX/vhdl/src/unisims/unisim_VITAL.vhd }
vcom -work unisim {$XILINX/vhdl/src/unisims/unisim_VITAL52K.vhd }
vcom -work unisim {$XILINX/vhdl/src/unisims/unisim_VCFG52K.vhd }
```

```

cd <directory storage location>
vlib unisim
vmap unisim unisim
vcom -work unisim {$XILINX/vhdl/src/unisims/unisim_VCOMP.vhd }
vcom -work unisim {$XILINX/vhdl/src/unisims/unisim_VPKG.vhd }
vcom -work unisim {$XILINX/vhdl/src/unisims/unisim_VITAL.vhd }
vcom -work unisim {$XILINX/vhdl/src/unisims/unisim_VCFG4K.vhd }

```

SIMPRIM

The following commands need to be executed on the command line to compile each of the SIMPRIM source files. The SIMPRIM VHDL libraries are written using VITAL libraries, the packages defined by the standard are fully accelerated by ModelSim. These commands can be saved in a 'do' to run as a script.

```

cd <directory storage location>
vlib simprim
vmap simprim simprim
vcom -work simprim {$XILINX/vhdl/src/simprims/simprim_Vcomponents.vhd}
vcom -work simprim {$XILINX/vhdl/src/simprims /simprim_Vpackage.vhd}
vcom -work simprim {$XILINX/vhdl/src/simprims /simprim_VITAL.vhd}

```

If the SIMPRIM library exists already compiled in a shared area then only the following line needs to be executed, the vmap command maps the logical library to the physical directory.

```
vmap simprim F:/some_directory/vendors/xilinx/simprim
```

Compiling the Xilinx Verilog gate level library with the ModelSim UI

UNISIM

The UNISIM (UNified SIMulation) libraries are only used for simulating at the RTL level, and pre-NGDBUILD stage of the design flow. The cells contained in the library are device dependant and only include unit delay timing. The Verilog source code model files for the libraries can be found at : \$XILINX/verilog/src/uni<technology>, where the technologies are as follows;

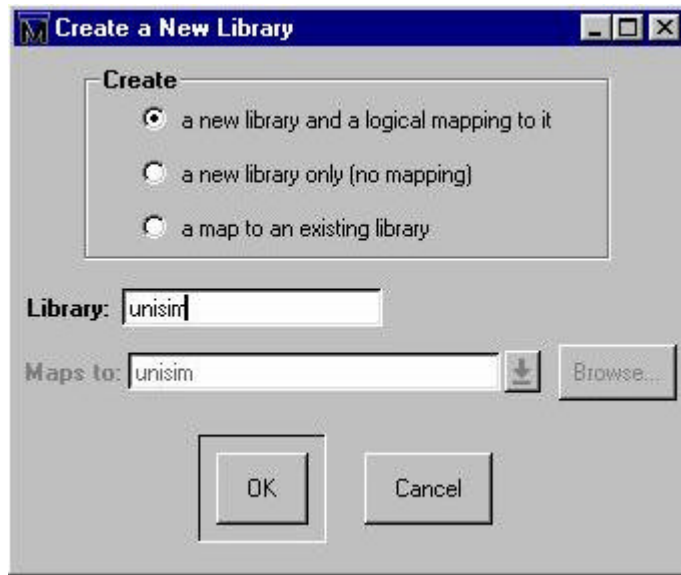
Uni3000 (XC3000 families)	Uni4000e (XC4000E families)
Uni4000x (XC4000X families)	Uni5200 (XC5200 family)
Uni9000 (XC9000 families)	UNISPARTAN (Spartan families)
UNISPARTANXL (Spartan XL families)	UNIVIRTEX (Virtex families)

Each of these directories contain <model>.v files for their associated family. Because there are a few cells with functional differences between Xilinx devices, a separate library is provided for each supported device. For example, decoders contain pull-ups in some devices and not in others. Also the global reset simulation mechanism differs across certain families

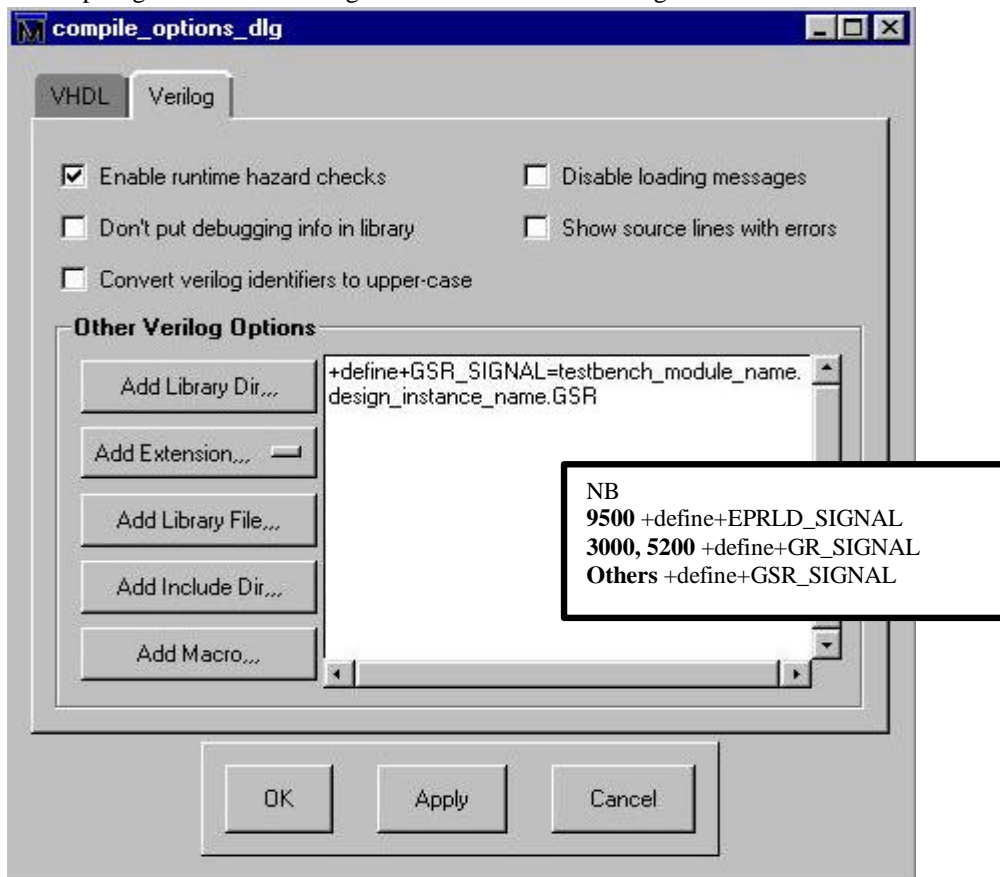
9500 – Global reset is labeled “EPRLD”.
5200 – Global reset is labeled “GR” and is active high.
3000 – Global reset is labeled “GR” and is active low.
4000, 4000X, SPARTAN & VIRTEX – Global reset is labeled “GSR”.

To be able to use more than one device family libraries, they have to be compiled into separate directories as a UNISIM library. Then for each design the mapping of the UNISIM logical name would need to be changed to the appropriate directory.

1. Change into the directory in which you wish to store the compiled libraries. Select **File ▸ Change Directory**, use the file browser to locate the desired directory. Any library that is now created will be placed into this directory by default.
2. Create a unisim library. Select **Library ▸ Create A New Library**, ensure that "a new library and logical mapping to it" is selected, enter 'unisim' in the library box as shown below.



3. Select **OK** to except this entry, this will execute both the vlib and vmap commands to create a unisim library and directory with the same name in the working directory.
4. Compiling the unisim Verilog source files. If the GSR signal needs to be controlled



during simulation then the following set up is required before compiling the library. The library must be compiled with a special switch specific to your design. Select the **Compile** button. This will display the compile dialog box, at the bottom select the **Default Options** button. This will display the compiler options dialog box, select the **Verilog** tab. This will display the dialog box above.

Select the **Add Macro** button and add a **macro name** of GR/EPRLD/GSR_SIGNAL and a **value** of testbench_model_name.design_instance_name.GSR. Where the testbench_model_name is the name of your test bench and the design_instance_name is the name of the instantiation of your design. Note this has to be done on a project by project basis if the control of the GSR is necessary. Select the **OK** button on this dialog box.

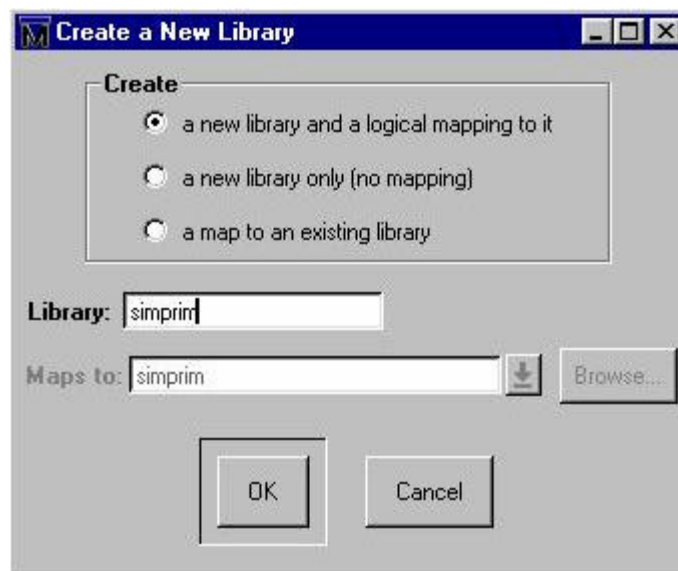
- Using the 'Look in' browser, locate the directory \$XILINX/verilog/src/uni<technology> where technology is the device family that you are using. Select the complete list of .v files, by selecting the first file in the list with the mouse and then selecting the last file in the list with the mouse while holding down the shift key. Select the **Compile** button, this will compile the compile library.

SIMPRIM

The SIMPRIM libraries are used for simulations post-implementation, the design stages include post-NGDBuild, post-MAP and full timing simulation post place and route. They are also used in RTL simulations using models generated by LogiBLOX. The SIMPRIM Verilog libraries are written using Verilog language primitives and UPD's which are fully accelerated by ModelSim. Verilog libraries include the overhead of runtime timing checks. These checks can be disabled at simulation time by using the 'disable timing checks in specify blocks' in the Verilog options menu of the load design dialog box. The Verilog source code can be found at : \$XILINX/verilog/src/simprims. This directory contains the all the .vmd model files for the simprim library.

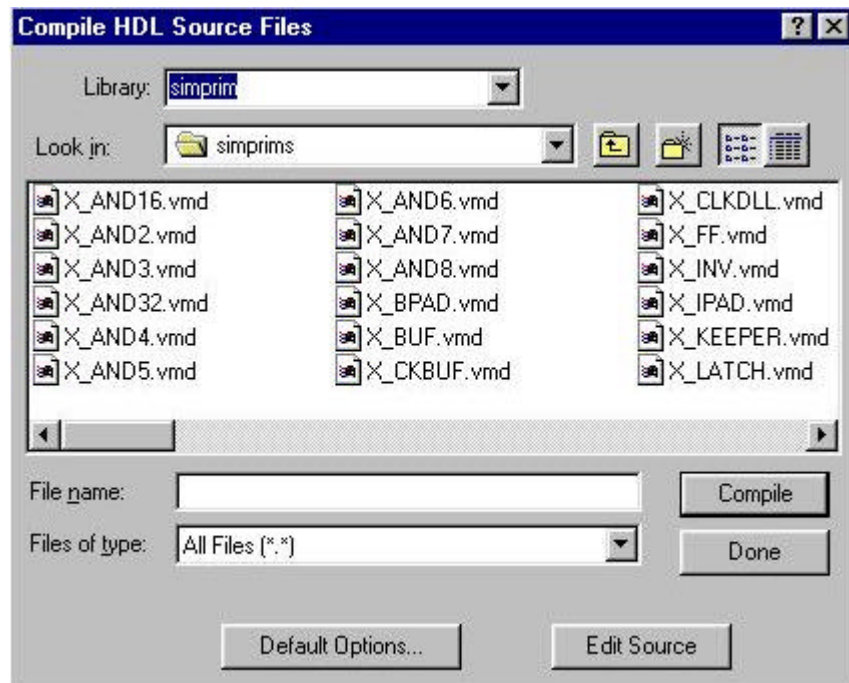
These libraries are completely design independent and therefore are used for the timing simulation of any Xilinx device family.

- Change into the directory in which you wish to store the compiled libraries. Select **File ▾ Change Directory**, use the file browser to locate the desired directory. Any library that is now created will be placed into this directory by default.
- Create a simprim library. Select **Library ▾ Create A New Library**, ensure that "a new library and logical mapping to it" is selected, enter 'simprim' in the library box as shown below.



- Select **OK** to except this entry, this will execute both the vlib and vmap commands to create a simprim library and directory with the same name in the working directory.

4. Compile the simprim source files. Select the **Compile** button. Select the **Compile** button. This will display the Compile HDL Source Files dialog box. Set the library to simprim using the pull down menu. Use the 'Look in' selection to locate the



SIMPRIM source files. These will be found in the Xilinx installation in \$XILINX/verilog/src/simprims.

Change the **files of type** to All files (*.*). Select the complete list of .vmd files, by selecting the first file in the list with the mouse and then selecting the last file in the list with the mouse while holding down the shift key. Select the **Compile** button, this will compile the compile library. The SIMPRIM library is now ready to be used for simulation.

Compiling the Xilinx Verilog gate level library using the command line

UNISIM

The following commands need to be executed on the command line to compile each of the UNISIM source files. The cells contained in the library are device dependant and only include unit delay timing. This means that there is a different set of files that need to be compiled to allow simulation of each of the Xilinx technology families. If the GSR signal needs to be controlled during simulation then it is necessary to compile the library with a macro defined. This is done on a design by design basis due to the fact that the GSR needs to be connected via a hierarchical name. There is an example below with and without the GSR connected. Note that GSR is for 4000, 4000X, SPARTAN and VIRTEX devices, see above for global names for other device families.

With GSR

```
cd <directory storage location>
vlib unisim
vmap unisim unisim
vlog -work unisim \
+define+GSR_SIGNAL=testbench_module_name.design_instance_name.GSR \
$XILINX/verilog/src/technology_name/*.v
```

- **testbench_model_name** = The name of your test bench module.

- ***design_instance_name*** = The instance name of your design.
- ***technology_name*** = The technology used in your design.

Without GSR

```
cd <directory storage location>
vlib unisim
vmap unisim unisim
vlog -work unisim $XILINX/verilog/src/technology_name/*.v
```

- ***technology_name*** = The technology used in your design.

SIMPRIM

The following commands need to be executed on the command line to compile each of the SIMPRIM source files. The SIMPRIM Verilog libraries are written using Verilog language primitives and UPD's which are fully accelerated by ModelSim. These commands can be saved in a 'do' to run as a script.

```
cd <directory storage location>
vlib simprim
vmap simprim simprim
vlog -work simprim {$XILINX/verilog/src/simprims/*.vmd}
```

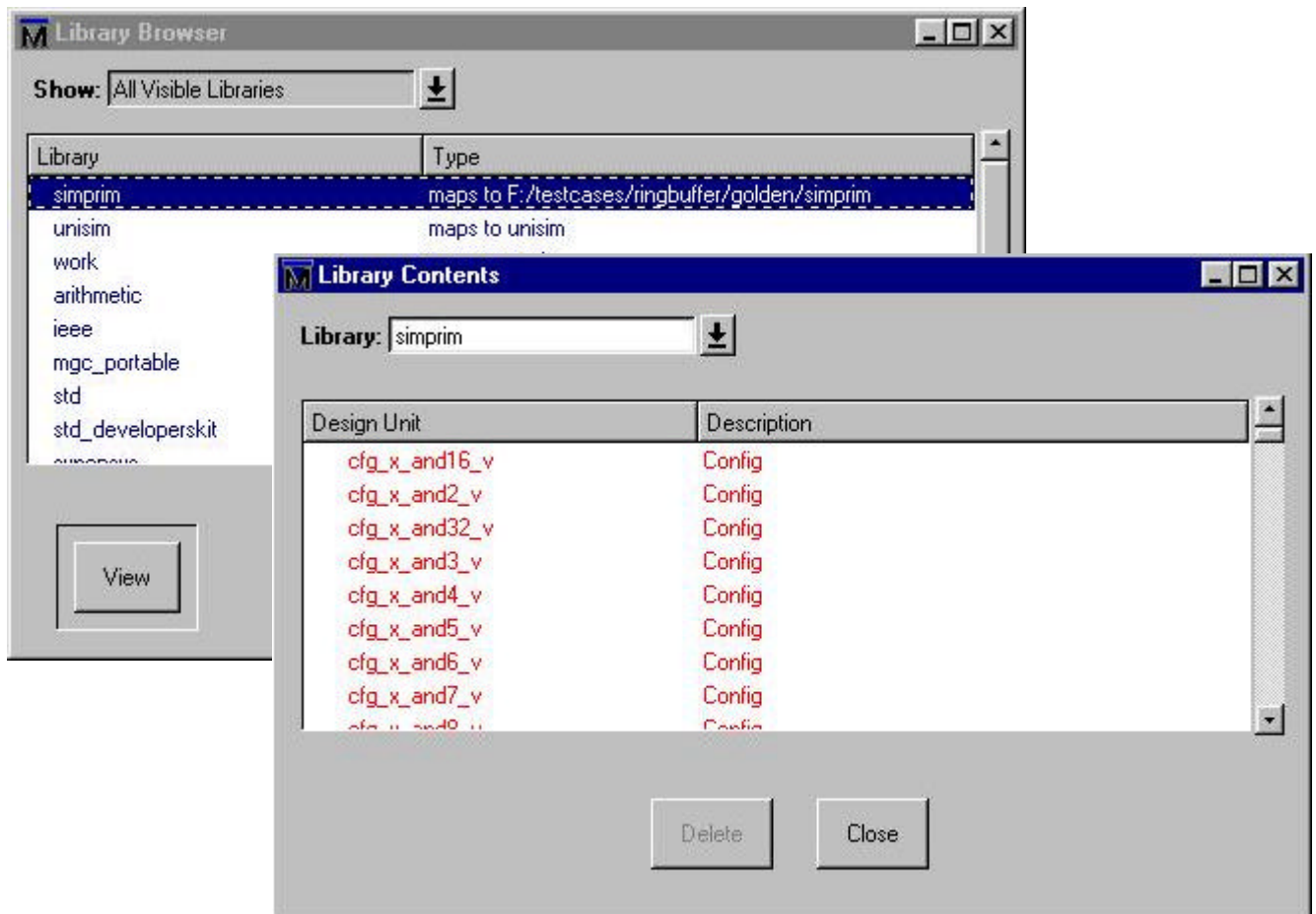
If the SIMPRIM library exists already compiled in a shared area then only the following line needs to be executed, the vmap command maps the logical library to the physical directory.

```
vmap simprim F:/some_directory/vendors/xilinx/simprim
```

Section 4. Compile and Timing Simulation of the VHDL Gate Level Source Files

This section details how to use ModelSim to run a full timing gate level simulation using Xilinx SIMPRIM's. This section includes a step by step guide on how to use the GUI to run the gate level simulations, however it does assume two important steps prior to starting. The first is that a VHDL VITAL structural netlist and SDF file has been exported from the place and route process, see section 2 of this document for full details. Secondly, the SIMPRIM's library source files are compiled into a library named 'simprim', see section 3 of this document for full details. ModelSim fully accelerates the VITAL timing and primitive packages, this means that there are built in routines that implement the behavior of these functions, the VHDL is not evaluated line by line. The VITAL packages are pre-compiled and ready to use when the product is installed. Xilinx use these timing and primitive packages to construct the cells from their technology and therefore benefit from the acceleration. As part of the VITAL standard there is a method set down to allow timing information to be taken from an SDF (Standard Delay Format) file and annotated onto the models. ModelSim reads an SDF file output from the place and route process and annotates the timing values in the file onto the generics of the model. A simulator is said to be VITAL compliant when it accelerates the VITAL packages and allows SDF data to be annotated onto VITAL models.

1. Ensure that the simprim library has been compiled and that there is a correct mapping to the library. For full details of how to compile the library see section 3 of this document. Select **Library ► Browse Libraries...**, select the simprim line and hit the **view button**, this will display the library dialog boxes as below.



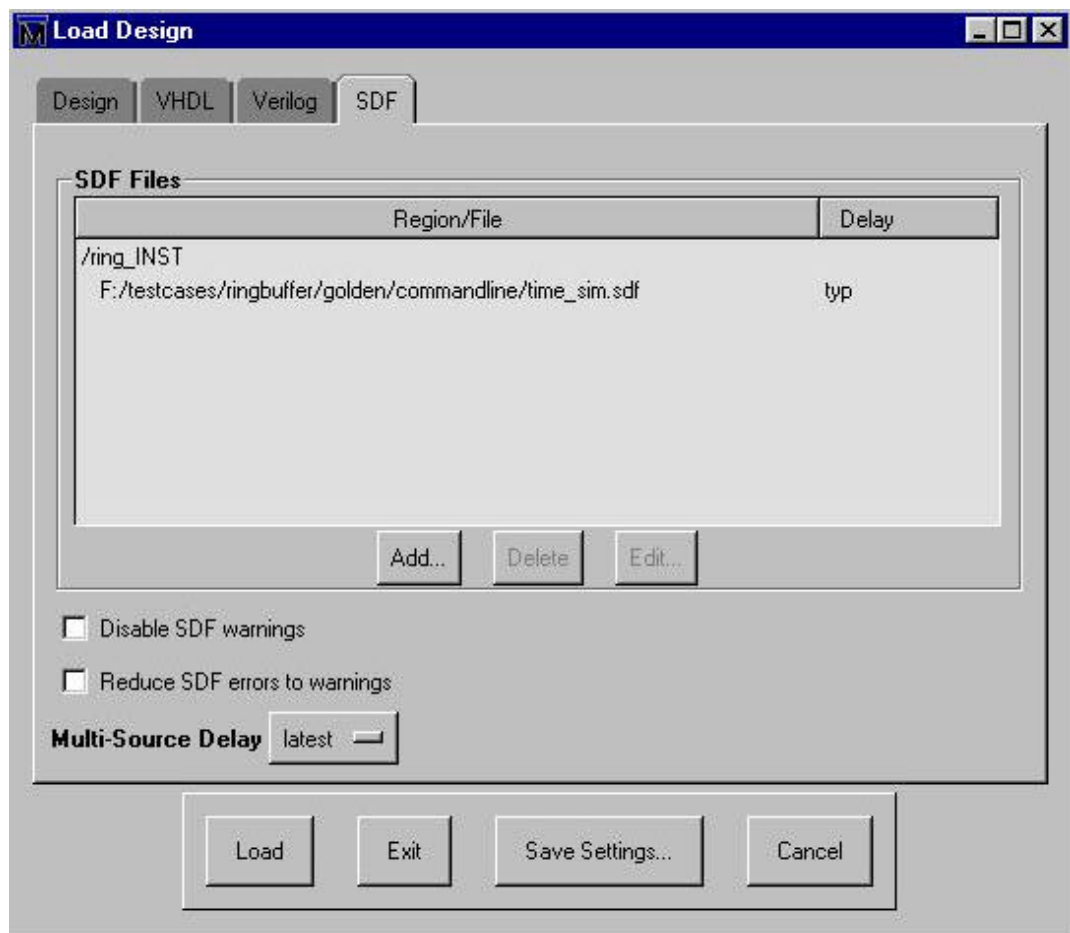
Close both of the dialog boxes after ensuring the correct mapping.

2. Compile the netlist output from M1, if the default name was unchanged then the file name for this file will be time_sim.vhd. The structural netlist can be compiled directly into the work library used for the RTL simulations, however this does have an effect on the set up for running RTL simulations. The reason for this is the fact that the entity that is produced by M1 has the same name as the entity used for RTL simulations. This is a requirement to allow the use of the same testbench without modification. However the library use clauses are not the same in the two entities and the dependency rules of VHDL mean that all files from the entity upwards have to be re-compiled. The best solution to get around this problem is to compile the structural netlist into a newly created library. A new configuration can then be written to bind the new structural design into the existing testbench. Select **Library ▸ Create A New Library**, ensure that "a new library and logical mapping to it" is selected, enter 'gates' in the library box and then select **OK**. Select the **Compile** button. This is the first button on the left hand side and will start the Compile HDL Source Files menu. This menu will automatically select the correct language compiler based upon the source file selected. Locate the time_sim.vhd file using the file browser. Ensure that the library is set to 'gates'. Select the **compile** button.

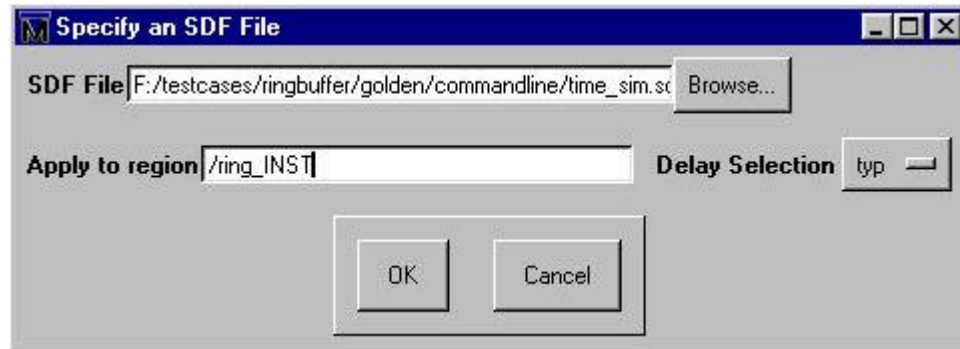
Note : time_sim.vhd includes the structural netlist for the design and two components ROC (Reset On Configuration) and TOC (Tri-state On Configuration) The delay for these actions to occur can be modified by changing the time generic on each of these components, the default is 100ns for ROC and 0ns for TOC.

Change the library in the compile dialog window back to 'work' and compile the config_gate.vhd source file. Cancel the compile dialog box.

3. Loading the simulator. Select **File ▸ Load New Design** or select the Load Design button. This will display the Load Design dialog box, which displays the design units available in each of the libraries. Ensure that the design tab is highlighted. Ensure that the work library is visible in the library section. Each of the units in the work library should be displayed along with a description of the type of unit, select the

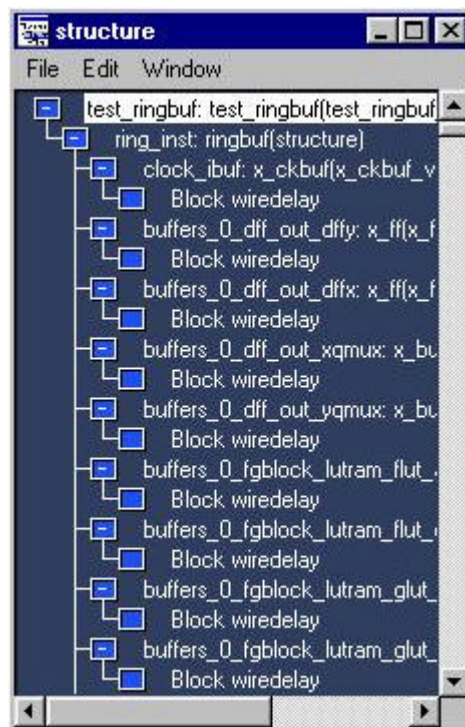


configuration named **test_bench_gate**. Select the **SDF** tab, this will display the SDF options. This menu sets up the annotation of the SDF file, it allows multiple SDF files to be annotated into different regions of a model. Select **Add...**, this will display the specify an SDF file dialog box.



Use the browse button to locate the SDF file that was generated by the place and route process. The default name of this file will be time_sim.sdf. The Apply to region selection needs to be set to the instantiated name of the device, in this case the name of the instantiation is **ring_INST**. Select the **OK** button, and re-select the design tab. Ensure that the **test_bench_gate** is still selected and hit **load** design. It is possible to run the simulation without the SDF file. This is done by not selecting the SDF file, simulations in this case have no net loading information and only have internal gate timing.

4. This will load all the units necessary to run the gate level simulation and will read the SDF file and annotate them onto the generics of the gate level cells. Open both the



structure and wave windows, select **View ▾ Structure** and **View ▾ Wave**. Notice that the structure window shows each of the gates in the design. Select the test bench **test_ringbuf** and drag it over and drop it into the waveform viewer.

5. Run the simulation. There are two ways of running the simulation for a specified time using the GUI. The first one is to type run 10000000 at the VSIM prompt in the main window. The other is to enter the time value into the run window in the set of menu

icons at the top of the main window, and then hitting the Run button. At the end of the run time the wave window will display the results of the simulation. It can be seen from the display that txda is a steady pseudo-random data stream and rxda represents the same data in a burst of bits.

The following is a script file that can be used to run the gate level simulation with SDF back annotation.

```
cd <design directory>  
vlib gates  
vmap gates gates  
vcom -work gates time_sim.vhd  
vcom -work work config_gate.vhd  
vsim -sdftyp /ring_INST=F:/time_sim.sdf -multisource_delay \  
latest work.test_bench_gate  
view *  
add wave /*  
run 1000000
```

To run without the delays annotated from the SDF file then the vsim line above should be replaced with the following line.

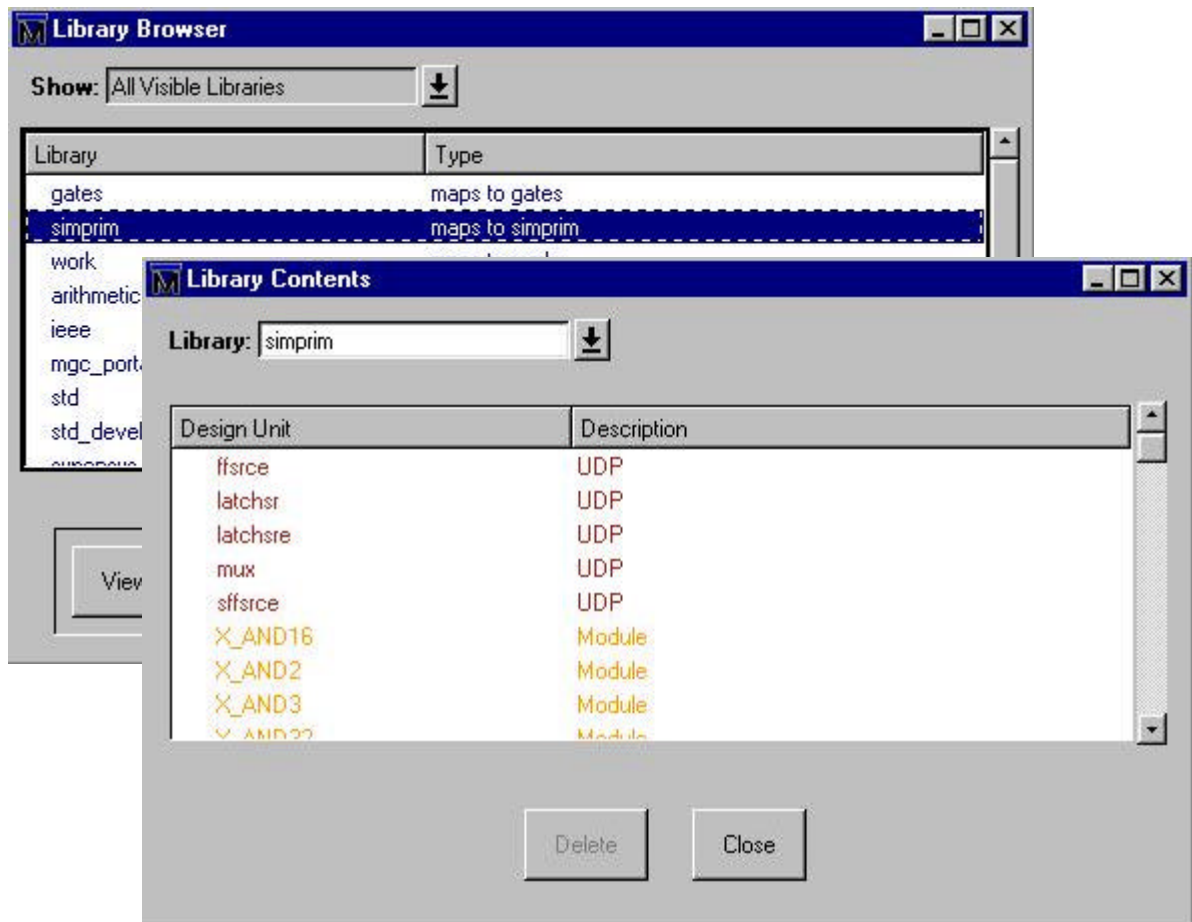
```
vsim latest work.test_bench_gate
```

In this case simulation time will be shorter but the model will not be a complete representation of the place and routed device. This is just a functional type simulation and can be used to verify that the Synthesis tool has correctly implemented the design.

Section 5. Compile and Timing Simulation of the Verilog Gate Level Source Files

This section details how to use ModelSim to run a full timing gate level simulation using Xilinx SIMPRIM's. This section includes a step by step guide on how to use the GUI to run the gate level simulations, however it does assume two important steps prior to starting. The first is that a Verilog structural netlist and SDF file has been exported from the place and route process, see section 2 of this document for full details. Secondly, the SIMPRIM's library source files are compiled into a library named 'simprim', see section 3 of this document for full details. ModelSim fully accelerates the Verilog primitives and UDP's (User Defined Primitives), this means that there are built in routines that implement the behavioral of these functions, the Verilog is not evaluated line by line. Xilinx use these primitives and UDP's to construct the cells from their technology and therefore benefit from the acceleration. As part of the an OVI (Open Verilog International) standard there is a method set down to allow timing information to be taken from an SDF (Standard Delay Format) file and annotated onto the models. ModelSim reads an SDF file output from the place and route process and annotates the timing values in the file onto the parameters (specparam's) of the model.

1. Ensure that the simprim library has been compiled and that there is a correct mapping to the library. For full details of how to compile the library see section 3 of this document. Select **Library ▸ Browse Libraries**, select the simprim line and hit the **view button**, this will display the library dialog boxes.



Close the library contents dialog box after ensuring the correct mapping.

2. Compiling the netlist output from MI, if the default name was unchanged then the file name for this file will be time_sim.v. The module name by default will be the same name as used by the RTL version of the device. This means if the new structural netlist is compiled into the library used for the RTL simulations, the top level of the RTL version will be over-written. This is not too much of a problem if the RTL design is small and quick to compile, but if you wish to switch between RTL and gate level simulations then this can become monotonous. A solution to this is to compile this new netlist into a separate library and to delete the top level design from the work library. The top level RTL version can then be re-compiled into a new library and the two versions can be selected using the `-L` switch in the simulator. The following instructions will detail this method. If it is not important to switch between gate and RTL simulations then the new netlist can be compiled into the work library. The simulator switch `-L` is then only needed to select the simprim library. Select the work library in the library browser and press the **View** button, this will show the contents of the work library. Select the RTL top level module **ringbuf** and then press the **delete** button at the bottom of the dialog box. Answer **yes** to verify the deletion of the module. Close the library browser using the **close** button.
3. Before compiling the netlist it is important to understand how the global reset and tri-state are controlled. Every storage cell in the simprim library has a GSR (Global Set Reset) pin. On the device during power on, the complete device is reset. There is a net within the netlist that allows the designer to connect some stimulus to simulate the action of both the GSR and the GTS (Global Tri-state). In the test bench file there are two initial blocks that produce the stimulus for the GSR and GTS signals, the GSR is shown below.

```
reg gsr, gts;  
initial begin gsr = 1'b0;  
              #100 gsr = 1'b1;  
end
```

To connect the gsr signal from the test bench into the netlist it is necessary to compile the netlist file with the GSR_SIGNAL and/or GTS_SIGNAL macros defined and pointing to the drivers in the testbench. Note that the GSR is normally active high, in this design the reset is active low therefore is inverted. These steps are shown in the following instructions.

4. Create a new library to hold the gate level netlist. Select **Library ▾ Create A New Library**, enter the library name 'gates' and then press **OK**. Select the **Compile** button. This is the first button on the left hand side and will start the Compile HDL Source Files menu. Change the library to the newly created gates library. Select the **default options** button at the bottom of the dialog and then select the verilog tab. Select the add macro button and enter GSR_SIGNAL as the macro name and test_ringbuf.gsr as the value, press **OK**. If the gts signal needs to be added then select add macro again and enter GTS_SIGNAL as the macro name and test_ringbuf.gts as the value, press **OK**. Press the OK button on the options dialog.



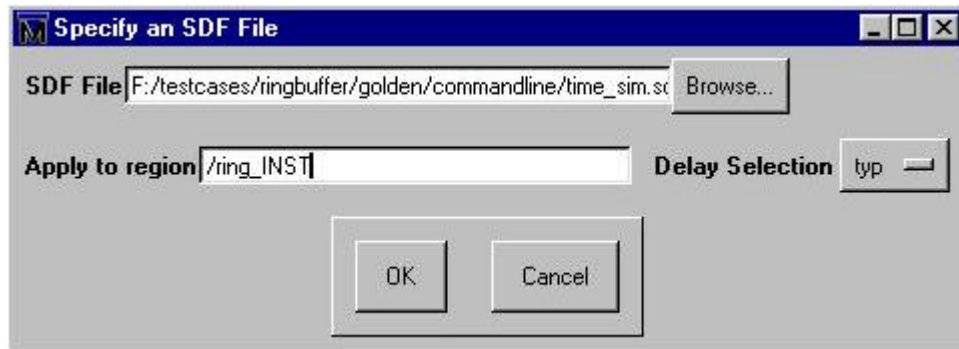
Select the time_sim.v file and press the compile button. This will compile the netlist into the gates library. Cancel the compile dialog box.

5. Loading the simulator. Select **File ▾ Load New Design** or select the Load Design button. This will display the Load Design dialog box, which displays the design units

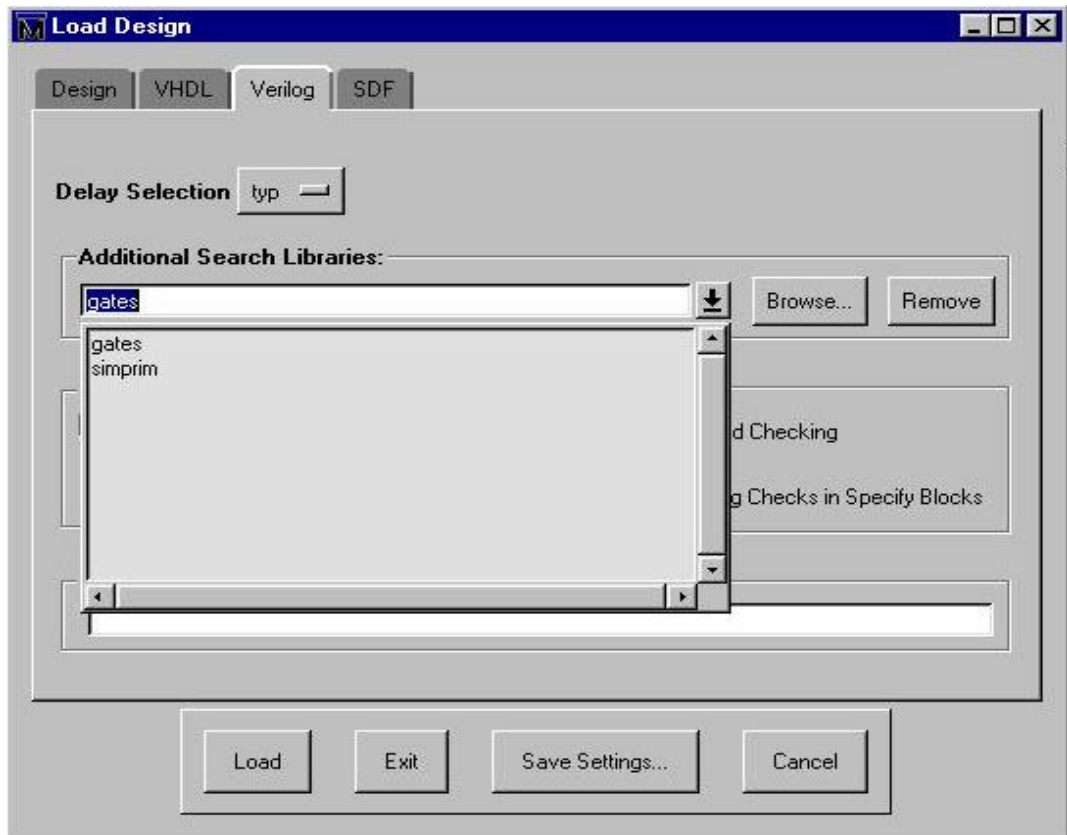
available in each of the libraries. Ensure that the design tab is highlighted. Ensure that the work library is visible in the library section. Each of the units in the work library should be displayed along with a description of the type of unit, **select** the module called **test_ringbuf**. There is an SDF tab on the compile dialog for setting up where the SDF file can be found and which region to apply the file. The netlist that is output by Xilinx has the \$sdf_annotate task included therefore it is not necessary to locate the file from the dialog box. If control is needed over the annotation process, for example you wish to simulate with and without the annotation, then it is necessary to comment out the annotation task from the generated netlist. The line in the time_sim.v file that needs commenting out is as follows.

```
// initial $sdf_annotate("time_sim.sdf");
```

If the SDF file needs to be applied using the dialog box then select the SDF tab and then the **Add** button. Locate the SDF file and apply it to the ring_INST region, which is the instantiation of the device. Note. When using the \$sdf_annotate task the time_sim.sdf file must either be in the working directory or have an absolute path.



Now select the Verilog tab on the compile dialog. It is necessary to detail the search paths for ModelSim to find modules that are not located in the work library.



Enter 'simprim' in the additional search libraries box and press return, then enter 'gates' and press return.

Return to the Design tab and ensure that the **test_ringbuf** module is still selected, and then press **Load**. If the netlist has not been compiled into the gates library then it is not necessary to include the gates library in the search paths.

6. This will load all the units necessary to run the gate level simulation and will read the SDF file and annotate them onto the specprams of the gate level cells. Open both the structure and wave windows, select **View ▾ Structure** and **View ▾ Wave**. Notice that the structure window shows each of the gates in the design. Select the test bench **test_ringbuf** and drag it over and drop it into the waveform viewer.
7. Run the simulation. There are two ways of running the simulation for a specified time using the GUI. The first one is to type run 1000000000 at the VSIM prompt in the main window. The other is to enter the time value into the run window in the set of menu icons at the top of the main window, and then hitting the Run button. At the end of the run time the wave window will display the results of the simulation. It can be seen from the display that txda is a steady pseudo-random data stream and rxda represents the same data in a burst of bits.

The following is a script file that can be used to run the gate level simulation with SDF back annotation.

```
cd <design directory>
vlib gates
vmap gates gates
vlog +define+GSR_SIGNAL=test_ringbuf.gsr \
    +define+GTS_SIGNAL=test_ringbuf.gts \
    -work gates {F:/testcases/ringbuffer/golden/time_sim.v}
vsim -sdftyp /ring_INST=F:/time_sim.sdf -multisource_delay \
latest -L simprim -L gates work.test_bench_gate
view *
add wave /*
run 1000000000
```

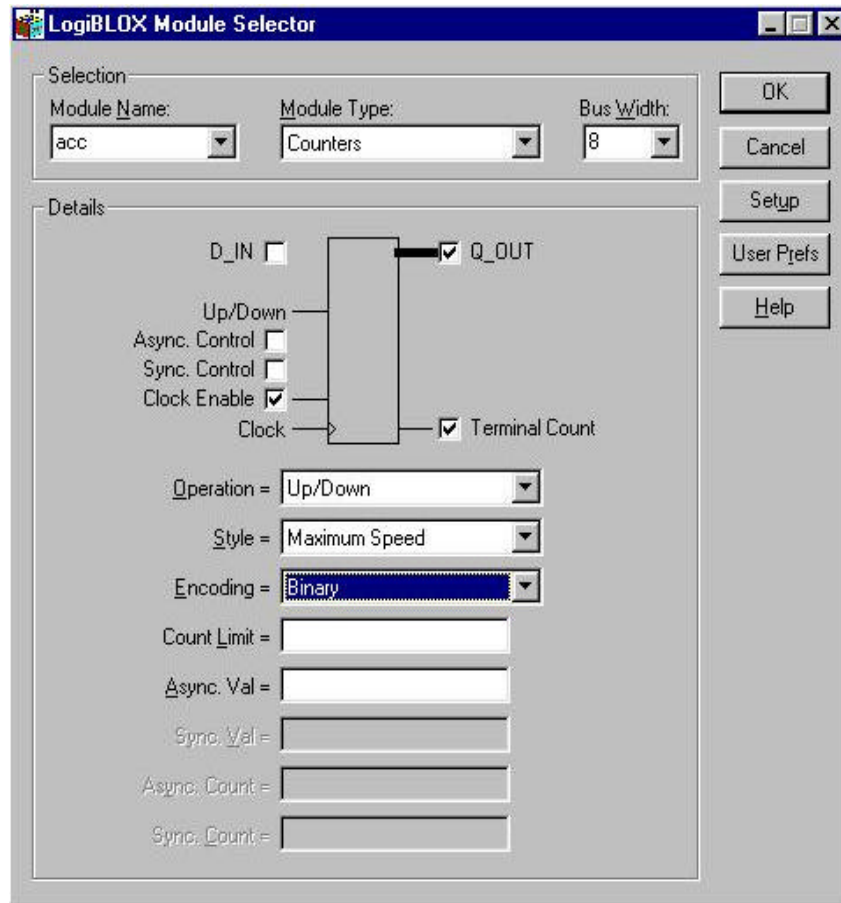
If it is not necessary to use the GSR or GTS then the +define lines can be removed from the vlog command line. If the \$sdf_annotate task is being used within the Xilinx netlist then the -sdftyp and -multisource_delay switches can be removed from the vsim command line.

Note : To run the RTL simulations again the top level RTL module needs to be re-compiled in the work library or into a new library pointed to by the vsim -L switch.

Section 6. Using LogiBLOX and CoreGen Models

LogiBLOX

LogiBLOX is a graphical interactive design tool that you can use to create high-level modules such as counters, shift registers, and multiplexers. LogiBLOX includes both a library of generic modules and a set of tools for customizing them. Using the LogiBLOX graphical user interface (GUI) you can create and process high-level LogiBLOX modules that fit into either a schematic-based design or HDL synthesis-based design.



The solution to simulation is slightly different between the two languages. There is a LogiBLOX library written in VHDL which is used by the behavioral models generated by LogiBLOX. For Verilog a structural netlist is generated using the UNISIM library. The following instructions detail how to set up the simulation environment to simulate using either language.

1. Start the LogiBLOX GUI. If this is the first time it has been started then the setup dialog box will be displayed. If it is not the first time select the **setup** button. Select the vendor tab and set the vendor to 'other' and the bus notation to 'BI'. Select the project directory and set it using the browse button to a working directory. Select the device family tab and select a family, for example XC4000XL. Select the options tab and select behavioral VHDL netlist and structural Verilog netlist in the simulation netlist section. Select the VHDL template and Verilog template in the component declaration section. Finally press the **OK** button to apply the settings.
2. Back in the module selector set the module type to counters. Select the bit width to 8. De-select the tick in the D_IN check box and select the tick in the terminal count check box. Select the **OK** button to confirm all settings and generate the module.

- This will generate four files that can be used for simulation. <module>.vhd is a behavioral VHDL model that uses the VHDL LogiBLOX libraries. <module>.v is a Verilog structural netlist that uses the UNISIM gate level library. <module>.vhi includes a template component declaration and a template component instantiation of the generated module. <module>.vei includes a template module declaration and a template module instantiation of the generated module.

VHDL User

- Change into the directory in which you wish to store the LogiBLOX libraries. Select **File ▾ Change Directory**, use the file browser to locate the desired directory. Create a logiblox library. Select **Library ▾ Create A New Library**, ensure that "a new library and logical mapping to it" is selected, enter 'logiblox' in the library box and select **OK**. Select the **Compile** button. Change the library to the newly created 'logiblox' library. Change directory to \$XILINX/vhdl/src/logiblox and compile the VHDL files in the following order, mvlutil.vhd, mvlarith.vhd and logiblox.vhd. Select the **done** button.
- If no work library exists, select **Library ▾ Create A New Library**, ensure that "a new library and logical mapping to it" is selected, enter 'work' in the library box and select **OK**.
- Select the **Compile** button. Change the library to the newly created 'work' library. If the module name has been kept to the default and the file output from LogiBLOX will be acc.vhd. Compile the acc.vhd file followed by the testacc.vhd file into the work library.
- Loading the simulator. Select **File ▾ Load New Design** or select the Load Design button. This will display the Load Design dialog box, which displays the design units available in each of the libraries. Ensure that the design tab is highlighted. Ensure that the work library is visible in the library section. Select the **test_counter** entity and press the **load** button.
- Select **View ▾ Structure** and **View ▾ Wave**, select the top level block in the structure window and drag and drop it into the wave window.
- Type 10000000 in the Run box in the main window and then hit the run for button as shown below.



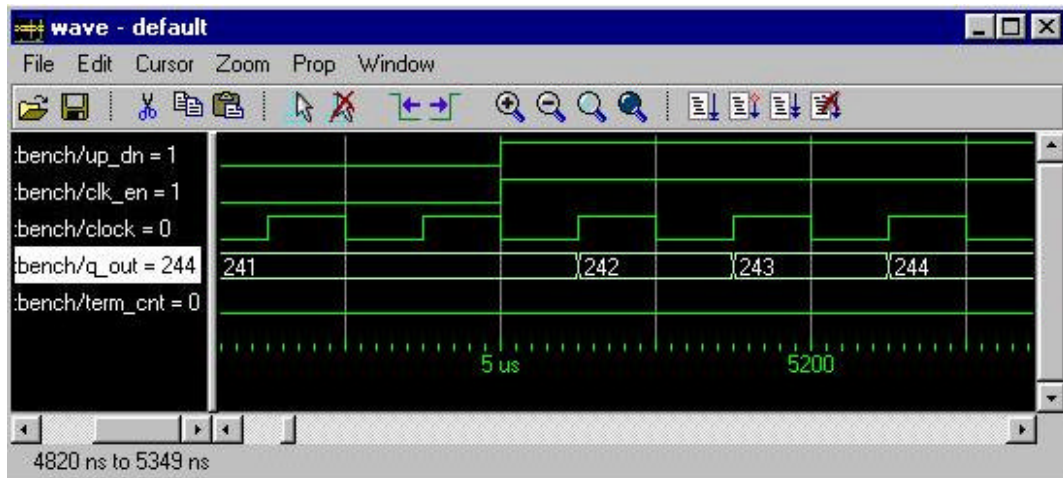
Verilog User

- Ensure that there is a mapping to the UNISIM Verilog library, the UNISIM library will have to be re-compiled to connect to the gsr and gts, see section 3 of this document for details. The GSR_SIGNAL macro should be set to test_counter.gsr and the GTS_SIGNAL macro should be set to test_counter.gts. If no work library exists, select **Library ▾ Create A New Library**, ensure that "a new library and logical mapping to it" is selected, enter 'work' in the library box and select **OK**.
- Select the **Compile** button. Change the library to the newly created 'work' library. If the module name has been kept to the default and the file output from LogiBLOX will be acc.v. Compile the acc.v file followed by the testacc.v file into the work library.
- Loading the simulator. Select **File ▾ Load New Design** or select the Load Design button. This will display the Load Design dialog box, which displays the design units available in each of the libraries. Ensure that the design tab is highlighted. Ensure that the work library is visible in the library section. Select the **test_counter** module and then the Verilog tab. Add uniprim into the additional search library box and then select the **load** button.
- Select **View ▾ Structure** and **View ▾ Wave**, select the top level block in the structure window and drag and drop it into the wave window.

- Type 100000000 in the Run box in the main window and then hit the run for button as shown below.



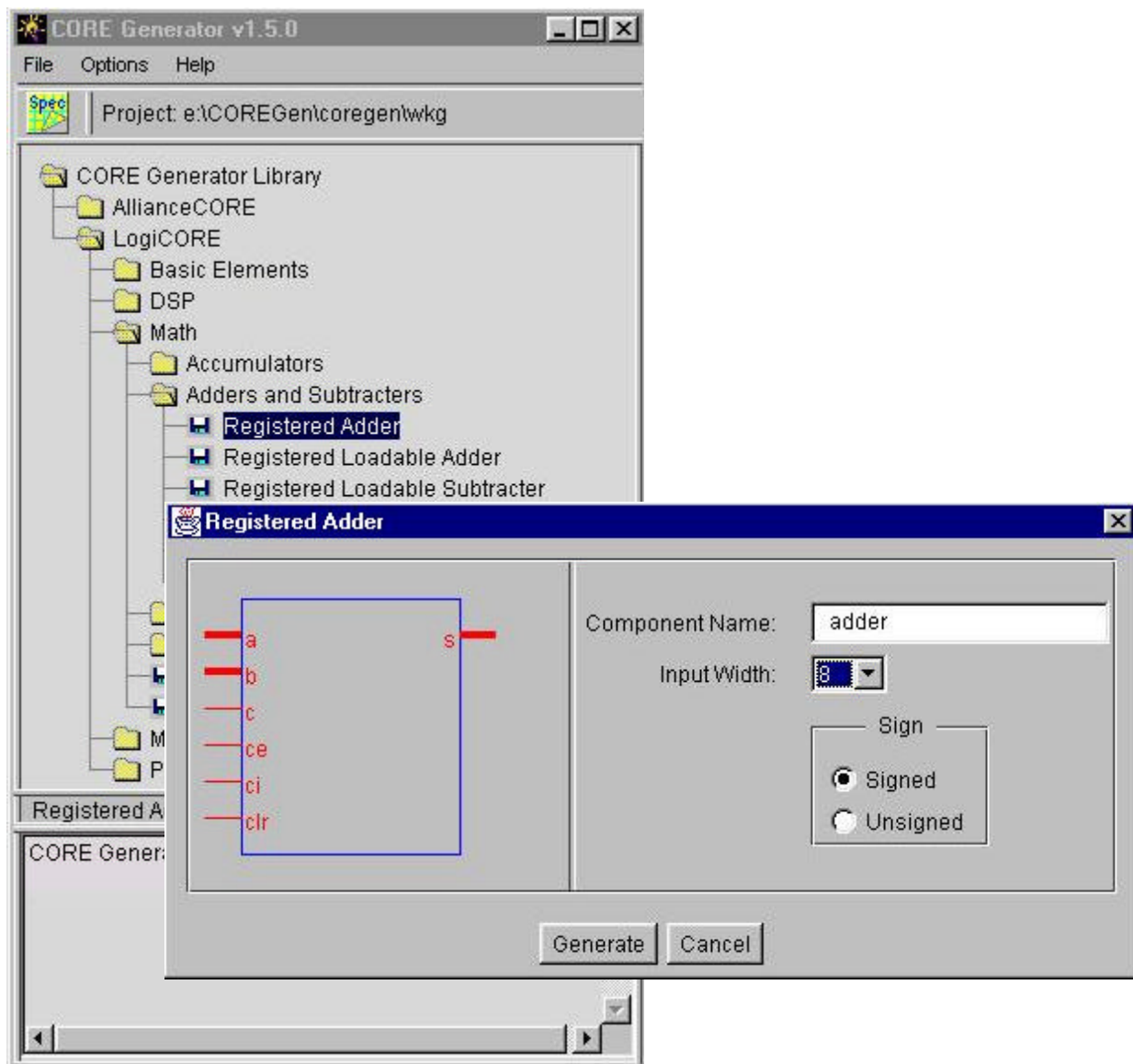
The output from both the VHDL and Verilog simulations should show a simple binary count which will count up and down as shown below.





CoreGEN

The Xilinx CORE Generator System (CoreGEN) provides the designer with a catalog of ready-made functions ranging in complexity from simple arithmetic operators such as adders, accumulators and multipliers, to system-level building blocks including filters, transforms and memories. CoreGEN outputs both VHDL and Verilog behavioral models for simulation. The following section details how to set up the VHDL library and how to simulate using either a VHDL or Verilog model.

- Start the CoreGEN GUI, the product will start and the core generator options will be displayed. In the output products select, VHDL behavioral simulation model, VHDL instantiation template, Verilog behavioral simulation model and Verilog instantiation template options. Select a target technology, leaving it at default is fine for this exercise. Select the **OK** button to apply the selections.
- Back in the CoreGEN GUI there is a browser that allows the selection of a library of modules that can be generated. The core generator library is made up of the AllianceCore and LogiCore libraries. The AllianceCore library includes models from Xilinx's Alliance partners and includes various models for applications such as ATM, USB and standard microprocessors. The LogiCore library includes models for standard building blocks such as DSP Filters, standard math's functions and Xilinx's own PCI functions. Select **Options ▸ System Options** and set the working directory, this is the directory that all files will be written.
- Double click the LogiCORE folder, than open the Math's folder and finally the Adders and Subtracters. This will display a number of models, select the Registered Adder by double clicking. This will open the Registered Adder model builder dialog box. Enter a component name of 'adder' and change the width to 8 bit unsigned, finally hit the **Generate** button.
- This will generate four files that can be used for simulation. <module>.vhd is a behavioral VHDL model that uses the VHDL CoreGEN libraries. <module>.v is a self contained Verilog behavioral model. <module>.vhi includes a template component declaration and a template component instantiation of the generated module. <module>.vei includes a template module declaration and a template module instantiation of the generated module.



VHDL User

5. In ModelSim, change into the directory in which you wish to store the CoreGen utilities library. Select **File ▸ Change Directory**, use the file browser to locate the desired directory. Create a 'xul' library. Select **Library ▸ Create A New Library**, ensure that "a new library and logical mapping to it" is selected, enter 'xul' in the library box and select **OK**. Select the **Compile** button. Change the library to the newly created 'xul' library. Change directory to <installed_directory>\COREGen\coregen\ip\xilinx\xul and compile the ul_utils.vhd file. Select the **done** button. 
6. If no work library exists, select **Library ▸ Create A New Library**, ensure that "a new library and logical mapping to it" is selected, enter 'work' in the library box and select **OK**.
7. Select the **Compile** button. Change the library to the newly created 'work' library. The model file output from CoreGEN is adder.vhd. Compile this model followed by the testadder.vhd test bench file.
8. Loading the simulator. Select **File ▸ Load New Design** or select the Load Design button. This will display the Load Design dialog box, which displays the design units available in each of the libraries. Ensure that the design tab is highlighted. Ensure that the work library is visible in the library section. Select the **test_adder** entity and press the **load** button. 

9. Select **View ▾ Structure** and **View ▾ Wave**, select the top level block in the structure window and drag and drop it into the wave window.
10. Type 20000 in the Run box in the main window and then hit the run for button as shown below.



Verilog User

5. If no work library exists, select **Library ▾ Create A New Library**, ensure that "a new library and logical mapping to it" is selected, enter 'work' in the library box and select **OK**.
6. Select the **Compile** button. Change the library to the newly created 'work' library. Compile the file output by coreGEN, adder.v into the work library. Compile the testadder.v file into the work library.
7. Loading the simulator. Select **File ▾ Load New Design** or select the Load Design button. This will display the Load Design dialog box, which displays the design units available in each of the libraries. Ensure that the design tab is highlighted. Ensure that the work library is visible in the library section. Select the **test_adder** module and then select the **load** button.
8. Select **View ▾ Structure** and **View ▾ Wave**, select the top level block in the structure window and drag and drop it into the wave window.
9. Type 20000000 in the Run box in the main window and then hit the run for button as shown below.



The output from both the VHDL and Verilog simulations should show the addition of the three vector values.

For more information

Consult the other technical notes under the support pages of <http://www.model.com> and your Synthesis and FPGA vendor's documentation for more information on HDL simulation for FPGA design. You can also visit Xilinx's Web page, where there is a technical note on HDL simulation which can be found at www.xilinx.com/techdocs/1923.htm.