

The OSVVM Simulator Script Library

The OSVVM Simulator Script Library provides a simple way to create and activate libraries, compile designs, and run simulations.

The intent of this scripting approach is to:

- Run the same scripts on any simulator
- Be as easy to read as a compile order list.
- Know the directory the script is in, the script only manages relative paths to itself. No Awkward path management in the scripts.
- Simplify integration of other libraries

This is an evolving approach. So it may change in the future. Input is welcome.

1 Start by Running the Demo

1.1 Download OSVVM Libraries

OSVVM is available as either a git repository [OSVVM Libraries](#) or a zip file from [osvvm.org Downloads Page](#).

On GitHub, all OSVVM libraries are a submodule of the repository OsvvmLibraries. Download all OSVVM libraries using git clone with the “-recursive” flag:

```
$ git clone --recursive https://github.com/osvvm/OsvvmLibraries
```

1.2 Create a Sim directory

Create a simulation directory. Generally I name this "sim" or "sim_vendor-name". Creating a simulation directory means that cleanup before running regressions is just a matter of deleting the sim directory and recreating a new one.

The following assumes you have created a directory named "sim" in the OsvvmLibraries directory.

Alternately, you can run simulations out of the Scripts, but cleanup is a mess as a simulator tends to create numerous temporaries.

1.3 Start the Script environment in the Simulator

Do the actions appropriate for your simulator.

1.3.1 Aldec RivieraPRO, Siemens QuestaSim and ModelSim

Initialize the OSVVM Script environment by doing:

```
source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartUp.tcl
```

Want to avoid doing this every time? In Aldec RivieraPro, set the environment variable, ALDEC_STARTUPTCL to StartUp.tcl (including the path information). Similarly in Mentor QuestaSim/ModelSim, set the environment variable, MODELSIM_TCL to StartUp.tcl (including the path information).

1.3.2 Aldec ActiveHDL

Initialize the OSVVM Script environment by doing:

```
scripterconf -tcl  
do -tcl <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartUp.tcl
```

Want to avoid doing this every time? For ActiveHDL, edit /script/startup.do and add above to it. Similarly for VSimSA, edit /BIN/startup.do and add the above to it. Note, with 2021.02, you no longer need to set the "Start In" directory to the OSVVM Scripts directory.

1.3.3 GHDL in Windows

Initialize the OSVVM Script environment by doing:

```
winpty tclsh  
source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartUp.tcl
```

To simplify this, put `source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartUp.tcl` in the `.tclshrc` file. You can also add a windows short cut that includes `C:\tools\msys64\mingw64.exe winpty tclsh`.

1.3.4 GHDL in Linux

Initialize the OSVVM Script environment by doing:

```
rlwrap tclsh  
source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartUp.tcl
```

To simplify this, put `source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartUp.tcl` in the `.tclshrc` file. In bash, add `alias gsim='rlwrap tclsh'` to your `.bashrc`.

1.3.5 NVC in Windows

Initialize the OSVVM Script environment by doing:

```
winpty tclsh  
source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartNVC.tcl
```

To simplify this, put `source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartNVC.tcl` in the `.tclshrc` file.

1.3.6 NVC in Linux

Initialize the OSVVM Script environment by doing:

```
rlwrap tclsh  
source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartNVC.tcl
```

To simplify this, put `source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartNVC.tcl` in the `.tclshrc` file.

1.3.7 Synopsys VCS

Initialize the OSVVM Script environment by doing:

```
rlwrap tclsh  
source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartVCS.tcl
```

To simplify this, put `source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartVCS.tcl` in the `.tclshrc` file and in bash, add alias `ssim='rlwrap tclsh'` to your `.bashrc`.

1.3.8 Cadence Xcelium

Initialize the OSVVM Script environment by doing:

```
rlwrap tclsh  
source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartXcelium.tcl
```

To simplify this, add the following line to the `.tclshrc` file.

```
source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartXcelium.tcl
```

In bash, add alias `ssim='rlwrap tclsh'` to your `.bashrc`.

1.3.9 Xilinx XSIM

Using OSVVM in Xilinx XSIM is under development. So far, Xilinx seems to be able to compile OSVVM utility library, however, we have not had any of our internal test cases pass.

To run OSVVM scripts in XSIM, start Vivado and then run the StartXSIM using:

```
source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartXSIM.tcl
```

1.4 Run the Demos

Do the following in your simulator command line:

```
build ../OsvvmLibraries
build ../OsvvmLibraries/RunDemoTests.pro
```

These will produce some reports, such as OsvvmLibraries_RunDemoTests.html. We will discuss these in the next section, OSVVM Reports.

2 Writing Scripts by Example

OSVVM Scripts are an API layer that is build on top of TCL. The API layer simplifies the steps of running simulations. For most applications you will not need any TCL, however, it is there if you need more capability.

2.1 Basic Script Commands

- **library <library-name>**
 - Make this library the active library. Create it if it does not exist.
- **analyze <VHDL-file>**
 - Compile (aka analyze) the design into the active library.
- **simulate <test-name>**
 - Simulate (aka elaborate + run) the design using the active library.
- **include <script-name>.pro**
 - Include another project script
- **build <script-name>.pro**
 - Start a script from the simulator. It is include + start a new log file for this script.

Scripts are named in the form <script-name>.pro. The scripts are TCL that is agumented with the OSVVM script API. The script API is created using TCL procedures.

For more details, see Command Summary later in this document.

2.2 Running a Simple Test

At the heart of running a simulation is setting the library, compiling files, and starting the simulation. To do this, we use library, analyze, and simulate.

The following is an excerpt from the scripts used to run OSVVM verification component library regressions.

```
library osvvm_TbAxi4_MultipleMemory
analyze TestCtrl_e.vhd
analyze TbAxi4_MultipleMemory.vhd
analyze TbAxi4_Shared1.vhd
```

```
TestCase TbAxi4_Shared1
simulate TbAxi4_Shared1
```

In OSVVM scripting, calling library activates the library. An analyze or simulate that follows library uses the specified library. This is consistent with VHDL's sense of the "working library".

Note that there are no paths to the files. For OSVVM commands that use paths, the path is always relative to the directory the script is located in unless an absolute path is specified.

The above script is in the file, testbench_MultipleMemory.pro. It can be run by specifying:

```
build ../OsvvmLibraries/AXI4/Axi4/testbench_MultipleMemory/testbench_MultipleMemory.pro
```

If you were to open testbench_MultipleMemory.pro, you would find that RunTest is used instead as it is an abbreviation for the analyze, TestCase and simulate when the names are the same.

2.3 Simulating with Generics

To specify generics, use the OSVVM generic function. Generic is called in the call to simulate as shown below. Note the square brackets are required and tell TCL to call the function to create the arguments for simulate. Calling generic this way allows OSVVM to do set generics using the method required by each simulator.

```
library default
simulate Tb [generic WIDTH 8]
simulate Tb [generic G1 5] [generic G2 7]
```

Release 2022.09 removed the necessity to put quotes around the options specified with simulate.

2.4 Debugging and Logging Signal Values (for later display)

By default, OSVVM scripting focuses on running regressions fast. Adding debugging information, logging signals, and/or displaying waveforms will slow things down. In addition, by default, if one simulation crashes, the scripts will continue and run the next simulation.

To add debugging information to your simulation, call SetDebugMode. If you do not call SetDebugMode, the debug mode is false. If you call SetDebugMode without a true or false value, the default is true.

```
SetDebugMode true
```

To log signals so they can be displayed after the simulation finishes, call SetLogSignals. If you do not call SetLogSignals, the log signals mode is false. If you call SetLogSignals without a true or false value, the default is true.

```
SetLogSignals true
```

Whether analyze or simulate stop on a failure or not is controlled by the internal variables `AnalyzeErrorStopCount` and `SimulateErrorStopCount`. By default, these values are set to 0, which means do not stop. Setting them to a non-zero value, causes either analyze or simulate to stop when the specified number of errors occur. Hence, to stop after one error, set them as follows.

```
set ::osvwm::AnalyzeErrorStopCount 1
set ::osvwm::SimulateErrorStopCount 1
```

To do all of the above in one step, call `SetInteractiveMode`. If you call `SetInteractiveMode` without a true or false value, the default is true. If you do not like the OSVVM default settings, you can add any of these to your `LocalScriptDefaults.tcl`.

Also note that there are scripts that automatically run when you call a simulation (see next section). You can use these scripts to display waveforms.

2.5 Scripts that Run during Simulate if they exist

Often with simulations, we want to add a custom waveform file. This may be for all designs or just one particular design. We may also need specific actions to be done when running on a particular simulator.

When `simulate` (or `RunTest`) is called, it will source the following files in order, if they exist:

- `<ToolVendor>.tcl`
- `<ToolName>.tcl`
- `wave.do`
- `<LibraryUnit>.tcl`
- `<LibraryUnit>_<ToolName>.tcl`
- `<TestCaseName>.tcl`
- `<TestCaseName>_<ToolName>.tcl`

`ToolVendor` is either {Aldec, Siemens, Cadence, Synopsys}. `ToolName` is one of {QuestaSim, ModelSim, RivieraPRO, ActiveHDL, VCS, Xcelium}. `LibraryUnit` is the name specified to simulate. `TestCaseName` is the name specified to `TestCase`.

It will search for these files in the following directories - `OsvvmLibraries/Scripts` - `CurrentSimulationDirectory` - `CurrentWorkingDirectory`

`CurrentSimulationDirectory` is the directory in which the simulator is running. `CurrentWorkingDirectory` is the directory of the script that calls either `RunTest` or `simulate`.

Currently GHDL and NVC do not run any extra scripts since they are batch simulators.

2.6 Adding Other Wave Files

To include wave files with names different from above, use the DoWaves function. DoWaves is called in the call to simulate as shown below. Note the square brackets are required and tell TCL to call the function to create the arguments for simulate.

If the wave1.do file is not in CurrentSimulationDirectory, then it will need path information. In Aldec and Siemens, these are run via the simulator command line (via -do). The method of running them may change in the future (and may use source).

```
library default
simulate Tb [DoWaves wave1.do]
simulate Tb [DoWaves wave1.do wave2.do]
```

2.7 Saving Waveforms with GHDL and NVC

GHDL and NVC run in a batch mode, but can save waveforms to see with a separate viewer (Gtkwave). To save waveforms for GHDL and NVC, call SetSaveWaves. If you do not call SetSaveWaves, the debug mode is false. If you call SetSaveWaves without a true or false value, the default is true.

```
SetSaveWaves true
```

2.8 Including Scripts

We build our designs hierarchically. Therefore our scripts need to be build hierarchically. When one script calls another script, such as OsvvmLibraries.pro does, we use include. The code for OsvvmLibraries.pro is as follows. The if is TCL and is only building the UART, AXI4, and DpRam if their corresponding directories exist.

```
include ./osvvm/osvvm.pro
include ./Common/Common.pro

if {[DirectoryExists UART]} {
    include ./UART/UART.pro
}
if {[DirectoryExists AXI4]} {
    include ./AXI4/AXI4.pro
}
if {[DirectoryExists DpRam]} {
    include ./DpRam/DpRam.pro
}
```

Note the paths specified to include are relative to OsvvmLibraries directory since that is where OsvvmLibraries.pro is located.

2.9 Building the OSVVM Libraries

Build is a layer on top of include (it calls include) that creates a logging point. In general, build is called from the simulator API (when we run something) and include is called from scripts.

By default, OSVVM creates collects all tool output for a build into an html based log file in ./logs/<tool_name>-<version>/<script-name>.html.

To compile all of the OSVVM libraries, use build as shown below.

```
build ../OsvvmLibraries/OsvvmLibraries.pro
```

2.10 Running OSVVM Test Cases

All OSVVM verification components are delivered with their regression test suite. There is also a script, named RunAllTests.pro, that runs all of the tests for that specific VC.

To run the AXI4 Full verification component regression suite, use the build shown below.

```
build ../OsvvmLibraries/AXI4/Axi4/RunAllTests.pro
```

Everything in OSVVM is composed hierarchically. If you want to run all AXI4 (Axi4 Full, Axi4Lite, and AxiStream), use the build shown below.

```
build ../OsvvmLibraries/AXI4/RunAllTests.pro
```

Similarly to run the tests for all VC in OsvvmLibraries use the build shown below.

```
build ../OsvvmLibraries/RunAllTests.pro
```

For most VC and OsvvmLibraries, there is a RunDemoTests.pro that runs a small selection of the VC test cases.

2.11 Do not use TCL's source or EDA tool's do

OSVVM uses include since it helps manage the path of where the script files are located. Include uses TCL's source internally. However, if you use TCL's source (or EDA tool's do) instead, you will not get include's directory management features and your scripts will need to manage the directory paths themselves.

2.12 Do not use TCL's cd

Simulators create files containing library mappings and other information in the simulation directory. If you use `cd` you lose all of this information. OSVVM tracks the simulation directory in the variable `::osvvm::CurrentSimulationDirectory`.

OSVVM tracks the directory in which scripts run as `CurrentWorkingDirectory`. All OSVVM API commands run relative to `CurrentWorkingDirectory`. When you call a script in another directory using `include`, `CurrentWorkingDirectory` is automatically updated to be the directory that contains the script. When `include` finishes it restores `CurrentWorkingDirectory` to be its value before `include` was called.

If while running a script, you need to adjust the `CurrentWorkingDirectory`, use `ChangeWorkingDirectory`. Like `cd`, `ChangeWorkingDirectory` allows either relative or absolute paths.

```
ChangeWorkingDirectory src
analyze Axi4Manager.vhd
```

If you need to determine a path relative to the `CurrentWorkingDirectory`, use `JoinWorkingDirectory`. In the following, the relative path used by `LinkLibraryDirectory` is

```
LinkLibraryDirectory [JoinWorkingDirectory RelativePath]
```

3 OSVVM's Reports

Good reports simplify debug and help find problems quickly. This is important as according to the [2020 Wilson Verification Survey FPGA](#) verification engineers spend 46% of their time debugging.

OSVVM produces the following reports:

- HTML Build Summary Report for human inspection that provides test completion status.
- JUnit XML Build Summary Report for use with continuous integration (CI/CD) tools.
- HTML Test Case Detailed report for each test case with Alert, Functional Coverage, and Scoreboard reports.
- HTML based simulator transcript/log files (simulator output)
- Text based test case transcript file (from `TranscriptOpen`)

The best way to see the reports is to look at the ones from the demo. If you have not already done `build OsvvmLibraries/RunDemoTests.pro`, then do so now.

3.1 HTML Build Summary Report

The Build Summary Report allows us to quickly confirm if a build passed or quickly identify which test cases did not PASS.

The Build Summary Report has three distinct pieces:

- Build Status
- Test Suite Summary
- Test Case Summary

For each Test Suite and Test Case, there is additional information, such as Functional Coverage and Disabled Alert Count.

In the sim directory, the Build Summary Report is in the file OsvvmLibraries_RunDemoTests.html.

OsvvmLibraries_RunDemoTestsWithCoverage Build Summary Report

Build	OsvvmLibraries_RunDemoTestsWithCoverage
Status	PASSED
PASSED	5
FAILED	0
SKIPPED	0
Analyze Failures	0
Simulate Failures	0
Elapsed Time (hours)	0:00:32
Elapsed Time (seconds)	31.825
Date	2022-09-11T14:31:0700
Simulator	RiversPRO
Version	RiversPRO-2022.04.117.8517
OSVVM YAML Version	2022.09
Simulation Transcript	OsvvmLibraries_RunDemoTestsWithCoverage.log
HTML Simulation Transcript	OsvvmLibraries_RunDemoTestsWithCoverage_log.html
Code Coverage	Code Coverage Results

▼ OsvvmLibraries_RunDemoTestsWithCoverage Test Suite Summary

TestSuite	Status	PASSED	FAILED	SKIPPED	Requirements passed / goal	Disabled Alerts	Elapsed Time
Axi4Full	PASSED	2	0	0	0 / 0	0	6.930
Axi4Stream	PASSED	1	0	0	0 / 0	0	4.723
Uart	PASSED	2	0	0	0 / 0	0	6.603

▼ Axi4Full Test Case Summary

Test Case	Status	Checks passed / checked	Errors	Requirements passed / goal	Functional Coverage	Disabled Alerts	Elapsed Time
TbAxi4_MemoryReadWriteDemo1	PASSED	334 / 334	0	0 / 0	51.75	0	2.631
TbAxi4_Axi4Req3_0by	PASSED	36 / 36	0	0 / 0	-	0	1.070

▼ Axi4Stream Test Case Summary

Test Case	Status	Checks passed / checked	Errors	Requirements passed / goal	Functional Coverage	Disabled Alerts	Elapsed Time
TbAxi4Stream_SendGetDemo1	PASSED	340 / 340	0	0 / 0	66.67	0	1.649

▼ Uart Test Case Summary

Test Case	Status	Checks passed / checked	Errors	Requirements passed / goal	Functional Coverage	Disabled Alerts	Elapsed Time
TbUart_SendGet1	PASSED	30 / 31	0	0 / 0	-	0	1.229
TbUart_SendGet2	PASSED	22 / 26	0	0 / 0	-	0	1.116

Build Summary Report

Note that any place in the report there is a triangle preceding text, pressing on the triangle will rotate it and either hide or reveal additional information.

3.1.1 Build Status

The Build Status, shown below, is in a table at the top of the Build Summary Report. If code coverage is run, there will be a link to the results at the bottom of the Build Summary Report.

OsvvmLibraries_RunDemoTestsWithCoverage Build Summary Report

Build	OsvvmLibraries_RunDemoTestsWithCoverage
Status	PASSED
PASSED	5
FAILED	0
SKIPPED	0
Analyze Failures	0
Simulate Failures	0
Elapsed Time (h:m:s)	0:00:32
Elapsed Time (seconds)	31.825
Date	2022-09-11T14:31-0700
Simulator	RivieraPRO
Version	RivieraPRO-2022.04.117.8517
OSVVM YAML Version	2022.09
Simulation Transcript	OsvvmLibraries_RunDemoTestsWithCoverage.log
HTML Simulation Transcript	OsvvmLibraries_RunDemoTestsWithCoverage_log.html
Code Coverage	Code Coverage Results

Build Status

3.1.2 Test Suite Summary

When running tests, test cases are grouped into test suites. A build can include multiple test suites. The next table we see in the Build Summary Report is the Test Suite Summary. The figure below shows that this build includes the test suites Axi4Full, AxiStream, and UART.

▼ OsvvmLibraries_RunDemoTestsWithCoverage Test Suite Summary

TestSuites	Status	PASSED	FAILED	SKIPPED	Requirements passed / goal	Disabled Alerts	Elapsed Time
Axi4Full	PASSED	2	0	0	0 / 0	0	6.930
AxiStream	PASSED	1	0	0	0 / 0	0	4.723
Uart	PASSED	2	0	0	0 / 0	0	6.603

Test Suite Summary

3.1.3 Test Case Summary

The remainder of the Build Summary Report is Test Case Summary, see below. There is a separate Test Case Summary for each test suite in the build.

▼ Axi4Full Test Case Summary

Test Case	Status	Checks passed / checked	Errors	Requirements passed / goal	Functional Coverage	Disabled Alerts	Elapsed Time
TbAxi4_MemoryReadWriteDemo1	PASSED	334 / 334	0	0 / 0	43.75	0	2.631
TbAxi4_AxiXReso3_slv	PASSED	36 / 36	0	0 / 0	-	0	1.070

► AxiStream Test Case Summary

▼ Uart Test Case Summary

Test Case	Status	Checks passed / checked	Errors	Requirements passed / goal	Functional Coverage	Disabled Alerts	Elapsed Time
TbUart_SendGet1	PASSED	30 / 34	0	0 / 0	-	0	1.229
TbUart_SendGet2	PASSED	22 / 26	0	0 / 0	-	0	1.146

Test Case Summary

3.2 JUnit XML Build Summary Report

The JUnit XML Build Summary Report works with continuous integration (CI/CD). The CI/CD tools use this to understand if the test is passing or not. They also have facilities for displaying the report - however, the OSVVM HTML format provides a superset of information.

OSVVM runs regressions on GitHub.

3.3 HTML Test Case Detailed Report

For each test case that is run (simulated), a Test Case Detailed Report is produced that contains consists of the following information:

- Test Information Link Table
- Alert Report
- Functional Coverage Report(s)
- Scoreboard Report(s)
- Link to Test Case Transcript (opened with Transcript Open)
- Link to this test case in HTML based simulator transcript

After running one of the regressions, open one of the HTML files in the directory `./reports/<test-suite-name>`. An example one is shown below.

TbUart_SendGet1 Test Case Detailed Report

Available Reports	
Alert Report	
ScoreboardPlus svs Report(s)	
Link to Simulation Results	
TbUart_SendGet1.txt	
OpenUMLibraries.RunOnMethods Build Summary	

TbUart_SendGet1 Alert Report

▼ TbUart_SendGet1 Alert Settings

Setting	Value	Description	
FailOnWarning	true	If true, warnings are a test error	
FailOnDisabledErrors	true	If true, Disabled Alert Counts are a test error	
FailOnRequirementErrors	true	If true, Requirements Errors are a test error	
External	Failures	0	Added to Alert Counts in determine total errors
	Errors	0	
	Warnings	0	
	Failures	0	
Expected	Errors	-1	Subtracted from Alert Counts in determine total errors
	Warnings	0	

▼ TbUart_SendGet1 Alert Results

Name	Status	Checks		Total Errors	Alert Counts			Requirements		Disabled Alert Counts		
		Passed	Total		Failures	Errors	Warnings	Passed	Checked	Failures	Errors	Warnings
TbUart_SendGet1	PASSED	30	34	0	0	1	0	0	0	0	0	0
Default	PASSED	16	16	0	0	0	0	0	0	0	0	0
OSVVM	PASSED	0	0	0	0	0	0	0	0	0	0	0
UART_SBI	PASSED	0	0	0	0	0	0	0	0	0	0	0
uarttx_1	PASSED	1	1	0	0	0	0	0	0	0	0	0
TransmitFifo	PASSED	0	0	0	0	0	0	0	0	0	0	0
uartrx_1	FAILED	10	14	4	0	4	0	0	0	0	0	0
ReceiveFifo	PASSED	0	0	0	0	0	0	0	0	0	0	0

TbUart_SendGet1 Scoreboard Report

Name	ItemCount	ErrorCount	ItemsChecked	ItemsPopped	ItemsDropped	FifoCount
TransmitFifo	20	0	0	20	0	0
ReceiveFifo	20	0	0	20	0	0

Test Case Detailed Report

Note that any place in the report there is a triangle preceding text, pressing on the triangle will rotate it and either hide or reveal additional information.

3.3.1 Test Information Link Table

The Test Information Link Table is in a table at the top of the Test Case Detailed Report. The figure below has links to the Alert Report (in this file), Functional Coverage Report (in this file), Scoreboard Reports (in this file), a link to simulation results (if the simulation report is in HTML), and a link to any transcript files opened by OSVVM.

TbUart_SendGet1 Test Case Detailed Report

Available Reports
Alert Report
ScoreboardPkg_slv Report(s)
Link to Simulation Results
TbUart_SendGet1.txt
OsvvmLibraries_RunDemoTestsWithCoverage_Build_Summary

Test Information Link Table

3.3.2 Alert Report

The Alert Report, shown below, provides detailed information for each AlertLogID that is used in a test case. Note that in the case of expected errors, the errors still show up as FAILED in the Alert Report and are rectified in the total error count.

TbUart_SendGet1 Alert Report

▼ TbUart_SendGet1 Alert Settings

Setting	Value	Description
FailOnWarning	true	If true, warnings are a test error
FailOnDisabledErrors	true	If true, Disabled Alert Counts are a test error
FailOnRequirementErrors	true	If true, Requirements Errors are a test error
External	Failures	0
	Errors	0
	Warnings	0
Added to Alert Counts in determine total errors		
Expected	Failures	0
	Errors	4
	Warnings	0
Subtracted from Alert Counts in determine total errors		

▼ TbUart_SendGet1 Alert Results

Name	Status	Checks		Total Errors	Alert Counts			Requirements		Disabled Alert Counts		
		Passed	Total		Failures	Errors	Warnings	Passed	Checked	Failures	Errors	Warnings
TbUart_SendGet1	PASSED	30	34	0	0	4	0	0	0	0	0	0
Default	PASSED	16	16	0	0	0	0	0	0	0	0	0
OSVVM	PASSED	0	0	0	0	0	0	0	0	0	0	0
UART_SB1	PASSED	0	0	0	0	0	0	0	0	0	0	0
uarttx_1	PASSED	4	4	0	0	0	0	0	0	0	0	0
TransmitFifo	PASSED	0	0	0	0	0	0	0	0	0	0	0
uarttx_1	FAILED	10	14	4	0	4	0	0	0	0	0	0
ReceiveFifo	PASSED	0	0	0	0	0	0	0	0	0	0	0

Alert Report

3.3.3 Functional Coverage Report(s)

The Test Case Detailed Report contains a Functional Coverage Report, shown below, for each functional coverage model used in the test case. Note this report is not from the demo.

Uart7_Random_part3 Coverage Report

Total Coverage: 100.00

▼ UART_RX_STIM_COV Coverage Model Coverage: 100.0

▼ UART_RX_STIM_COV Coverage Settings

CovWeight	1
Goal	100.0
WeightMode	AT_LEAST
Seeds	824213985, 792842968
CountMode	COUNT_FIRST
IllegalMode	ILLEGAL_ON
Threshold	45.0
ThresholdEnable	FALSE
TotalCovCount	100
TotalCovGoal	100

▼ UART_RX_STIM_COV Coverage Bins

Name	Type	Mode	Data	Idle	Count	AtLeast	Percent Coverage
NORMAL	COUNT	1	0 to 255	0	63	63	100.0
NORMAL	COUNT	1	0 to 255	1 to 15	7	7	100.0
PARITY	COUNT	3	0 to 255	2 to 15	11	11	100.0
STOP	COUNT	5	1 to 255	2 to 15	11	11	100.0
PARITY_STOP	COUNT	7	1 to 255	2 to 15	6	6	100.0
BREAK	COUNT	9 to 15	11 to 30	2 to 15	2	2	100.0
Total Percent Coverage:		100.0					

▼ UART_RX_COV Coverage Model Coverage: 100.0

► UART_RX_COV Coverage Settings

▼ UART_RX_COV Coverage Bins

Name	Type	Mode	Count	AtLeast	Percent Coverage
NORMAL	COUNT	1	70	1	7000.0
PARITY	COUNT	3	11	1	1100.0

Functional Coverage Report

3.3.4 Scoreboard Report(s)

The Test Case Detailed Report contains a Scoreboard Report, shown below. There is a row in the table for each scoreboard model used in the test case.

TbAxi4_MemoryReadWriteDemo1 Scoreboard Report

Name	ItemCount	ErrorCount	ItemsChecked	ItemsPopped	ItemsDropped	FifoCount
WriteAddressFifo	40	0	0	40	0	0
WriteDataFifo	150	0	0	150	0	0
WriteResponseFifo	40	0	0	40	0	0
ReadAddressFifo	40	0	0	40	0	0
ReadDataFifo	150	0	0	150	0	0
WriteResponse Scoreboard	40	0	40	40	0	0
ReadResponse Scoreboard	150	0	150	150	0	0
WriteAddressFifo	40	0	0	40	0	0
WriteDataFifo	150	0	0	150	0	0
ReadAddressFifo	40	0	0	40	0	0
ReadAddressTransactionFifo	40	0	0	40	0	0
ReadDataFifo	150	0	0	150	0	0
WriteBurstFifo	118	0	0	118	0	0
ReadBurstFifo	118	0	108	118	0	0

Scoreboard Report

3.4 Test Case Transcript

OSVVM's transcript utility facilitates collecting all test output to into a single file, as shown below.

```
%% Log ALWAYS in Default, Transmit 32 words at 110 ns
%% Log INFO in transmitter_1, Axi Stream Send. TData: 00000001 TStrb: 1111 TKeep: 1111 TID: 00 TDest: 0 TUser: 0 TLast: 0 Operation# 1 at 110 ns
%% Log INFO in receiver_1, Word Receive. Operation# 1 Data: 00000001 TID: 00 TDest: 0 TUser: 0 TLast: 0 at 120 ns
%% Log PASSED in Default, RxData Received : 00000001 at 120 ns
%% Log INFO in transmitter_1, Axi Stream Send. TData: 00000002 TStrb: 1111 TKeep: 1111 TID: 00 TDest: 0 TUser: 0 TLast: 0 Operation# 2 at 120 ns
%% Log INFO in receiver_1, Word Receive. Operation# 2 Data: 00000002 TID: 00 TDest: 0 TUser: 0 TLast: 0 at 130 ns
%% Log PASSED in Default, RxData Received : 00000002 at 130 ns
%% Log INFO in transmitter_1, Axi Stream Send. TData: 00000003 TStrb: 1111 TKeep: 1111 TID: 00 TDest: 0 TUser: 0 TLast: 0 Operation# 3 at 130 ns
%% Log INFO in receiver_1, Word Receive. Operation# 3 Data: 00000003 TID: 00 TDest: 0 TUser: 0 TLast: 0 at 140 ns
%% Log PASSED in Default, RxData Received : 00000003 at 140 ns
%% Log INFO in transmitter_1, Axi Stream Send. TData: 00000004 TStrb: 1111 TKeep: 1111 TID: 00 TDest: 0 TUser: 0 TLast: 0 Operation# 4 at 140 ns
%% Log INFO in receiver_1, Word Receive. Operation# 4 Data: 00000004 TID: 00 TDest: 0 TUser: 0 TLast: 0 at 150 ns
%% Log PASSED in Default, RxData Received : 00000004 at 150 ns
%% Log INFO in transmitter_1, Axi Stream Send. TData: 00000005 TStrb: 1111 TKeep: 1111 TID: 00 TDest: 0 TUser: 0 TLast: 0 Operation# 5 at 150 ns
%% Log INFO in receiver_1, Word Receive. Operation# 5 Data: 00000005 TID: 00 TDest: 0 TUser: 0 TLast: 0 at 160 ns
%% Log PASSED in Default, RxData Received : 00000005 at 160 ns
%% Log INFO in transmitter_1, Axi Stream Send. TData: 00000006 TStrb: 1111 TKeep: 1111 TID: 00 TDest: 0 TUser: 0 TLast: 0 Operation# 6 at 160 ns
%% Log INFO in receiver_1, Word Receive. Operation# 6 Data: 00000006 TID: 00 TDest: 0 TUser: 0 TLast: 0 at 170 ns
%% Log PASSED in Default, RxData Received : 00000006 at 170 ns
%% Log INFO in transmitter_1, Axi Stream Send. TData: 00000007 TStrb: 1111 TKeep: 1111 TID: 00 TDest: 0 TUser: 0 TLast: 0 Operation# 7 at 170 ns
%% Log INFO in receiver_1, Word Receive. Operation# 7 Data: 00000007 TID: 00 TDest: 0 TUser: 0 TLast: 0 at 180 ns
%% Log PASSED in Default, RxData Received : 00000007 at 180 ns
%% Log INFO in transmitter_1, Axi Stream Send. TData: 00000008 TStrb: 1111 TKeep: 1111 TID: 00 TDest: 0 TUser: 0 TLast: 0 Operation# 8 at 180 ns
%% Log INFO in receiver_1, Word Receive. Operation# 8 Data: 00000008 TID: 00 TDest: 0 TUser: 0 TLast: 0 at 190 ns
%% Log PASSED in Default, RxData Received : 00000008 at 190 ns
%% Log INFO in transmitter_1, Axi Stream Send. TData: 00000009 TStrb: 1111 TKeep: 1111 TID: 00 TDest: 0 TUser: 0 TLast: 0 Operation# 9 at 190 ns
%% Log INFO in receiver_1, Word Receive. Operation# 9 Data: 00000009 TID: 00 TDest: 0 TUser: 0 TLast: 0 at 200 ns
%% Log PASSED in Default, RxData Received : 00000009 at 200 ns
%% Log INFO in transmitter_1, Axi Stream Send. TData: 0000000A TStrb: 1111 TKeep: 1111 TID: 00 TDest: 0 TUser: 0 TLast: 0 Operation# 10 at 200 ns
%% Log INFO in receiver_1, Word Receive. Operation# 10 Data: 0000000A TID: 00 TDest: 0 TUser: 0 TLast: 0 at 210 ns
%% Log PASSED in Default, RxData Received : 0000000A at 210 ns
%% Log INFO in transmitter_1, Axi Stream Send. TData: 0000000B TStrb: 1111 TKeep: 1111 TID: 00 TDest: 0 TUser: 0 TLast: 0 Operation# 11 at 210 ns
%% Log INFO in receiver_1, Word Receive. Operation# 11 Data: 0000000B TID: 00 TDest: 0 TUser: 0 TLast: 0 at 220 ns
%% Log PASSED in Default, RxData Received : 0000000B at 220 ns
%% Log INFO in transmitter_1, Axi Stream Send. TData: 0000000C TStrb: 1111 TKeep: 1111 TID: 00 TDest: 0 TUser: 0 TLast: 0 Operation# 12 at 220 ns
%% Log INFO in receiver_1, Word Receive. Operation# 12 Data: 0000000C TID: 00 TDest: 0 TUser: 0 TLast: 0 at 230 ns
%% Log PASSED in Default, RxData Received : 0000000C at 230 ns
%% Log INFO in transmitter_1, Axi Stream Send. TData: 0000000D TStrb: 1111 TKeep: 1111 TID: 00 TDest: 0 TUser: 0 TLast: 0 Operation# 13 at 230 ns
%% Log INFO in receiver_1, Word Receive. Operation# 13 Data: 0000000D TID: 00 TDest: 0 TUser: 0 TLast: 0 at 240 ns
%% Log PASSED in Default, RxData Received : 0000000D at 240 ns
%% Log INFO in transmitter_1, Axi Stream Send. TData: 0000000E TStrb: 1111 TKeep: 1111 TID: 00 TDest: 0 TUser: 0 TLast: 0 Operation# 14 at 240 ns
%% Log INFO in receiver_1, Word Receive. Operation# 14 Data: 0000000E TID: 00 TDest: 0 TUser: 0 TLast: 0 at 250 ns
%% Log PASSED in Default, RxData Received : 0000000E at 250 ns
```

Test Case Transcript

3.5 HTML Simulator Transcript

Simulator transcript files can be long. The basic OSVVM regression test (OsvvmLibraries/RunAllTests.pro), produces a log file that is 84K lines long. As a plain text file, this is not browsable, however, when converted to an html file it is. OSVVM gives you the option to create either html (default), shown below, or plain text. In the html report, any place there is a triangle preceding text, pressing on the triangle will rotate it and either hide or reveal additional information.

```
▶ build ../../OsvvmLibraries/RunDemoTests.pro
▶ include ../../OsvvmLibraries/RunDemoTests.pro
▶ include ./AXI4/Axi4/RunDemoTests.pro
▶ TestSuite Axi4Full1
▶ include ./testbench
▶ analyze ../TestCases/OsvvmTestCommonPkg.vhd
▶ analyze TestCtrl_e.vhd
▶ analyze TbAxi4.vhd
▶ analyze TbAxi4Memory.vhd
▶ RunTest ../TestCases/TbAxi4_MemoryReadWriteDemo1.vhd
▶ RunTest ../TestCases/TbAxi4_AxiXResp3_slv.vhd
▶ include ./AXI4/AxiStream/RunDemoTests.pro
▶ TestSuite AxiStream
▶ include ./testbench
▶ analyze ../TestCases/OsvvmTestCommonPkg.vhd
▶ analyze TestCtrl_e.vhd
▶ analyze TbStream.vhd
▶ RunTest ../TestCases/TbStream_SendGetDemo1.vhd
▶ include ./UART/RunDemoTests.pro
▶ TestSuite Uart
▶ include ./testbench/TestHarness.pro
▶ analyze OsvvmTestCommonPkg.vhd
▶ analyze TestCtrl_e.vhd
▶ analyze TbUart.vhd
▶ RunTest ../testbench/TbUart_SendGet1.vhd
▶ RunTest ../testbench/TbUart_SendGet2.vhd
Build Start time 16:53:26 PDT Sun Sep 11 2022
Build Finish time 16:53:38, Elapsed time: 0:00:12
Build: OsvvmLibraries_RunDemoTests PASSED, Passed: 5, Failed: 0, Skipped: 0, Analyze Errors: 0, Simulate Errors: 0
```

HTML Simulator Transcript

4 How To Generate Reports

4.1 VHDL Aspects of Generating Reports

To generate reports, you need to have the following in your VHDL testbench:

- Name your test case with `SetTestName("TestName")`.
- Do some self-checking with `AffirmIf`, `AffirmIfEqual`, or `AffirmIfNotDiff`.
- End the test case with `EndOfTestReports`.

These following code snippet shows these in use. More details of this are in [OSVVM Test Writers User Guide](#) in the documentation repository.

```
-- Reference to OSVVM Utility Library
library OSVVM ;
context OSVVM.OsvvmContext ;
. . .
TestProc : process
begin
    -- Name the Test
    SetTestName("TbDut") ;
    . . .
    -- Do some Checks
    AffirmIfEqual(Data, X"A025", "Check Data") ;
    . . .
    -- Generate Reports (replaces call to ReportAlerts)
    EndOfTestReports ;
    std.env.stop(GetAlertCount) ;
end process TestProc ;
```

4.2 Generating Reports and Simple Tests

If we have a simple test, where the design name is `Dut.vhd` and the testbench is `TbDut.vhd`, then we can run it with the following script

```
# File name: Dut.pro
analyze Dut.vhd
analyze TbDut.vhd
simulate TbDut
```

If we run this test with using build Dut.pro, Dut and TbDut will be compiled into the library named default. The simulation TbDut will run and a build summary report will be created with only one test case in it. The test suite will be named Default. The test case will be named TbDut. Be sure to name the test internally to TbDut using SetTestName as otherwise, a NAME_MISMATCH failure will be generated.

4.3 Generating Reports and Running Tests without Configurations

In OSVVM, we use the testbench framework shown in the [OSVVM Test Writers User Guide](#) (see documentation repository). The test harness in the following example is named TbUart. The test sequencer entity is in file TestCtrl_e.vhd. Tests are in architectures of TestCtrl in the files, TestCtrl_SendGet1.vhd, TestCtrl_SendGet2.vhd, and TbtCtrl_Scoreboard1.vhd. The tests are run by calling "simulate TbUart". TestCase is used to specify the test name that is running. This is needed here as otherwise the name TbUart would be used. The test case that is run is the latest one that was analyzed.

```
TestSuite Uart
library    osvvm_TbUart
analyze    TestCtrl_e.vhd
analyze    TbUart.vhd

TestCase   TbUart_SendGet1
analyze    TestCtrl_SendGet1.vhd
simulate   TbUart

TestCase   TbUart_SendGet2
analyze    TestCtrl_SendGet2.vhd
simulate   TbUart

TestCase   TbUart_Scoreboard1
analyze    TestCtrl_Scoreboard1.vhd
simulate   TbUart
```

The above call to TestCase puts the TestCase name into the build test summary YAML file. If the simulation for any reason fails to run, there will be no test status information in the YAML file. As a result, when the build summary report is being created, it will detect this as a test failure.

Another possibility in the above test scenario is that a particular test case fails to analyze. In this case, if the script continues and calls simulate, the previously successfully compiled test will run. In this case, if each test is given a unique name in VHDL using SetTestName (which is also recorded in the YAML file), then the VHDL test name will not match the test case name and a NAME_MISMATCH failure will be generated by the scripts.

4.4 Generating Reports and Running Tests with Configurations

The OSVVM verification component regression suite uses configurations to specify an exact architecture to run in a given test. We give the configuration, the test case, and the file the same name. We also put the configuration declaration at the end of the file containing the test case (try it, you will understand why). When we run a test that uses a configuration, simulate specifies the configuration's design unit name. Hence, we revise the sequence of running one test to be as follows.

```
TestCase  TbUart_SendGet1
analyze   TbUart_SendGet1.vhd
simulate  TbUart_SendGet1
```

When running a large test suite, this gets tedious, so we added a shortcut named RunTest that encapsulates the above three steps into the single step. This changes our original script to the following. If the name in RunTest has a path, the path is only used with analyze.

```
TestSuite Uart
library   osvvm_TbUart
analyze   TestCtrl_e.vhd
analyze   TbUart.vhd

RunTest   TbUart_SendGet1.vhd
RunTest   TbUart_SendGet2.vhd
RunTest   TbUart_Scoreboard1.vhd
```

One advantage of using configurations is that on a clean build (library deleted before starting it), if a test case fails to analyze, then the corresponding configuration will fail to analyze, and the simulation will fail to run. If this happens, it will be detected and recorded as a test failure in the build summary report.

5 Turning on Code Coverage

Code coverage is a metric that tells us if certain parts of our design have been exercised or not. Turning on code coverage with OSVVM is simple. In the following example, we enable coverage options during analysis and simulation separately.

```
# File name: Dut.pro
SetCoverageAnalyzeEnable true
analyze   Dut.vhd
SetCoverageAnalyzeEnable false
SetCoverageSimulateEnable true
analyze   TbDut.vhd
simulate  TbDut
SetCoverageSimulateEnable false
```

Note that CoverageAnalyzeEnable is specifically turned off before compiling the testbench so that the testbench is not included in the coverage metrics.

You can also set specific options by using SetCoverageAnalyzeOptions and SetCoverageSimulateOptions. By default, OSVVM sets these options so that statement, branch, and statemachine coverage is collected.

When coverage is turned on for a build, coverage is collected for each test. If there are multiple test suites in the build, when a test suite completes execution, the coverage for each test in the test suite is merged. When a build completes the coverage from each test suite is merged and an html coverage report is produced.

6 Command Summary

Commands are case sensitive. Single word names are all lower case. Multiple word names are CamelCase.

The following are general API commands.

- **SetLibraryDirectory [LibraryDirectory]**
 - Set the directory in which the libraries will be created to LibraryDirectory.
 - If LibraryDirectory is not specified, use the CurrentSimulationDirectory.
 - By default, libraries are created in <LibraryDirectory>/VHDL_LIBS/<tool version>/.
- **GetLibraryDirectory**
 - Get the Library Directory.
- **library <LibraryName> [<path>]**
 - Make LibraryName found in library directory specified by path the active library.
 - Create the LibraryName if it does not exist.
 - If path is not specified, use the library directory specified by SetLibraryDirectory.
- **LinkLibrary <library> [<path>]**
 - Create a mapping to a library that is in the library directory specified by path.
 - If path is not specified, use the library directory specified by SetLibraryDirectory.
- **LinkLibraryDirectory [LibraryDirectory]**
 - Map all of the libraries in the specified LibraryDirectory. If LibraryDirectory is not specified, use the library directory specified by SetLibraryDirectory.
- **LinkCurrentLibraries**
 - If you use cd, then use LinkCurrentLibraries immediately after to map all current visible libraries to the new CurrentSimulationDirectory.

- **RemoveLibrary LibraryName [<path>]**
 - Remove the named library. Path is only used to find and delete libraries that have not been mapped in OSVVM.
- **RemoveLibraryDirectory [<path>]**
 - Remove the Library specified in path.
 - If path is not specified, the library directory specified by SetLibraryDirectory is used.
- **RemoveAllLibraries**
 - Call RemoveLibraryDirectory on all library directories known to OSVVM.
- **analyze [<path>/]<name> [options]**
 - Analyze (aka compile) the design into the active library.
 - Name must be a file with an extension that is *.vhd or.vhdl* for vhdl, *.v for verilog*, or *.sv* for SystemVerilog.
 - Paths are relative to CurrentWorkingDirectory.
- **simulate <TestName> [options]**
 - Simulate (aka elaborate + run) the design using the active library.
 - TestName is a library unit (entity or configuration)
 - options may be one or more options to the simulator or see generic.
- **generic <name> <value>**
 - Called in the options part of simulate as `simulate tb1 [generic width 5]`
- **DoWaves <wave1.do> ...**
 - Called in the options part of simulate as `simulate tb1 [DoWaves wave1.do wave2.do]`
 - Paths used with wave files are relative to CurrentSimulationDirectory.
- **SetSecondSimulationTopLevel <library>.<TestName>**
 - Sets the name of a second library unit to use during simulation.
 - Called before simulate.
- **TestCase <test-name>**
 - Identify the TestCase that is active.
 - Must match name in the testbench call to SetTestName (aka SetAlertLogName).
- **RunTest [<path>/]<name>**
 - Combines analyze, TestCase, and simulate into one step.
 - TestCase and simulate are set to the base name of file.
 - Paths are relative to CurrentWorkingDirectory.
- **SkipTest <test-name> Reason**
 - Add Skip test to the Build Summary Reports with Reason as part of the report.
- **TestSuite <test-suite-name>**
 - Identify the current TestSuite. If not specified the name is default.

- **include [<path>/]<name>**
 - Include another project script. If name is a file and its extension is .pro, .tcl, or .do, it will be sourced. If name is a directory then any file whose name is name and extension is .pro, .tcl, or .do will be sourced.
 - Paths are relative to CurrentWorkingDirectory.
- **build [<path>/]<name>**
 - Start a script from the simulator. It is include + start a new log file for this script.
 - Paths are relative to CurrentWorkingDirectory.
- **SetTranscriptType [html|log]**
 - Select the Transcript file to be either html or log. The default is html.
- **GetTranscriptType**
 - Get the Transcript file type (either html or log).
- **ChangeWorkingDirectory <RelativePath>**
 - Changes CurrentWorkingDirectory to file join \$CurrentWorkingDirectory \$RelativePath
- **JoinWorkingDirectory <RelativePath>**
 - Returns file join \$CurrentWorkingDirectory \$RelativePath

In all commands that accept a path, relative paths (including no path) is relative to the directory in which the current script is running. With the command name, "[]" indicates a parameter is optional. If shown in a highlighted code example [generic G1 5] then the code must contain the "[]".

The following commands set options for analyze and simulate.

- **SetVHDLVersion [2008 | 2019 | 1993 | 2002]**
 - Set VHDL analyze version. OSVVM libraries require 2008 or newer.
- **GetVHDLVersion**
 - Return the current VHDL Version.
- **SetSimulatorResolution <value>**
 - Set Simulator Resolution. Any value supported by the simulator is ok.
- **GetSimulatorResolution**
 - Return the current Simulator Resolution.
- **SetCoverageAnalyzeEnable [true|false]**
 - To collect coverage for a design, SetCoverageEnable and SetCoverageAnalyzeEnable must be enabled when it is analyzed.
 - If true, enable coverage during analyze,
 - If false, disable coverage during analyze.
 - If not specified, true is the default.
 - Initialized to false (so simulations run faster)
- **GetCoverageAnalyzeEnable**
 - Returns the setting for coverage during analyze.

- **SetCoverageAnalyzeOptions <options>**
 - Use the string specified in options as the coverage options during analyze.
- **GetCoverageAnalyzeOptions**
 - Return the coverage options for analyze.
- **SetCoverageSimulateEnable [true|false]**
 - To collect coverage during a simulation, SetCoverageEnable and SetCoverageSimulateEnable must be enabled the simulation is started.
 - If true, enable coverage during simulate,
 - If false, disable coverage during simulate.
 - If not specified, true is the default.
 - Initialized to false (so simulations run faster)
- **GetCoverageSimulateEnable**
 - Returns the setting for coverage during simulate.
- **SetCoverageSimulateOptions <options>**
 - Use the string specified in options as the coverage options during simulate.
- **GetCoverageSimulateOptions**
 - Return the coverage options for simulate.
- **SetCoverageEnable [true|false]**
 - If true, set coverage enable to true.
 - If false, set coverage enable to false.
 - If not specified, true is the default.
 - Initialized to true.
- **GetCoverageEnable**
 - Get the CoverageEnable value.
- **SetVhdlAnalyzeOptions <options>**
 - Set the VHDL options for analyze to options.
- **GetVhdlAnalyzeOptions**
 - Get the VHDL options for analyze.
- **SetVerilogAnalyzeOptions <options>**
 - Set the Verilog options for analyze to options.
- **GetVerilogAnalyzeOptions**
 - Get the Verilog options for analyze.
- **SetExtendedAnalyzeOptions <options>**
 - Set extended (additional) options for analyze to options.
- **GetExtendedAnalyzeOptions**
 - Get extended (additional) options for analyze.
- **SetExtendedSimulateOptions <options>**
 - Set extended (additional) options for simulate to options.
- **GetExtendedSimulateOptions**
 - Get extended (additional) options for simulate.

- **SetDebugMode [true|false]**
 - If true, add debugging options during analyze and simulate.
 - If false, do not add debugging options during analyze and simulate.
 - If not specified, true is the default.
 - Initialized to false (so simulations run faster)
- **GetDebugMode**
 - Returns the state of DebugMode.
- **SetLogSignals [true|false]**
 - If true, log signals during simulate.
 - If false, do not log signals during simulate.
 - If not specified, true is the default.
 - Initialized to false (so simulations run faster)
- **GetLogSignals**
 - Returns the state of LogSignals.
- **SetInteractiveMode [true|false]**
 - If DebugMode was not set with SetDebugMode, then set it using this value
 - If LogSignals was not set with SetLogSignals, then set it using this value.
 - If true, sets variables AnalyzeErrorStopCount and SimulateErrorStopCount to 1
 - If false, sets variables AnalyzeErrorStopCount and SimulateErrorStopCount to previous value
 - If not specified, true is the default.
 - Initialized to false (so simulations run faster)
- **GetInteractiveMode**
 - Returns the state of InteractiveMode.

The values for a commands options value are typically simulator dependent. To keep a set of scripts simulator independent, be sure to call these at a high level, such as in `LocalScriptDefaults.tcl`.

The following are options currently only for GHDL and NVC.

- **SetExtendedElaborateOptions <options>**
 - Set extended (additional) options for simulate to options.
- **GetExtendedElaborateOptions**
 - Get extended (additional) options for simulate.
- **SetExtendedRunOptions <options>**
 - Set extended (additional) options for simulate to options.
- **GetExtendedRunOptions**
 - Get extended (additional) options for simulate.
- **SetSaveWaves [true|false]**
 - If true, save waveforms during simulate. If not specified, true is the default. Initialized to false (so simulations run faster)

- **GetSaveWaves**
 - Returns the state of LogSignals.

Helper functions - used to minimize the amount of TCL used in PRO scripts - FileExists <name> - if file name exists, return true otherwise false. - DirectoryExists <name> - if directory name exists, return true otherwise false.

Caution any undocumented commands are experimental and may change or be removed in a future revision.

7 Variables

7.1 Variables set by OSVVM Scripts

All osvvm VendorScripts_XXX.tcl set the variables ToolVendor, ToolName, ToolType, and ToolNameVersion. These are useful for personalizing scripts. For example,

```
if {$ToolName eq "GHDL"} {
    # ... do something based on GHDL
}
```

ToolVendor is the name of the vendor. ToolName is the name of the tool. ToolType can be either "simulator" or "synthesis". ToolNameVersion is formatted "<ToolName-version>", where version is specific to a tool and revision.

Note that ToolName was formerly named simulator. The variable simulator is deprecated. Use ToolName instead.

The settings for ToolVendor and ToolName is as defined in the table below.

ToolVendor	ToolName	ToolType	Notes
Aldec	ActiveHDL	simulator	
Aldec	RivieraPRO	simulator	
Aldec	VSimSA	simulator	ActiveHDL command line
Cadence	Xcelium	simulator	
GHDL	GHDL	simulator	
NVC	NVC	simulator	
Siemens	ModelSim	simulator	
Siemens	QuestaSim	simulator	
Synopsys	VCS	simulator	
Xilinx	XSIM	simulator	Still in Debug
Xilinx	Vivado	synthesis	Currently supports analyze

These variables can be used to do tool specific actions in scripts. I use the following in my LocalScriptDefaults.tcl (see next section) file.

```
if {${::osvvm::ToolVendor} eq "Siemens"} {  
    SetExtendedAnalyzeOptions "-quiet"  
    SetExtendedSimulateOptions "-quiet"  
}
```

7.2 Variables used to configure OSVVM

OSVVM sets variables in the file OsvvmDefaultSettings.tcl. Do not change this file. Instead, create a LocalScriptsDefaults.tcl. An easy way to do this is to copy Example_LocalScriptDefaults.tcl to LocalScriptDefaults.tcl. LocalScriptDefaults.tcl is not in the OSVVM release - which allows you to modify it and not have it overwritten when you update your release.

Complete documentation for each variable is in the Example_LocalScriptDefaults.tcl file.

Using LocalScriptDefaults, you can change things such as

- OSVVM created directories with reports, results, and libraries.
- TCL Error signaling
- Generate html transcript
- Generate a single tcl script for everything run

Note that some of the OSVVM commands are can also be set using variables.

8 Script File Summary

- **StartUp.tcl**
 - StartUp script for running ActiveHDL, GHDL, Mentor, RivieraPro, and VSimSA (ActiveHDL)
 - Detects the simulator running and calls StartUpShared.tcl
- **StartGHDL.tcl**
 - Version of StartUp.tcl that is specific to GHDL
- **StartNVC.tcl**
 - Version of StartUp.tcl that is specific to NVC
- **StartVCS.tcl**
 - Version of StartUp.tcl that is specific to VCS
- **StartVCS.tcl**
 - Version of StartUp.tcl that is specific to VCS
- **StartXcelium.tcl**
 - Version of StartUp.tcl that is specific to Xcelium
- **StartXSIM.tcl**
 - Version of StartUp.tcl that is specific to Xsim
 - Note, XSIM is currently a alpha level, experimental release.

- **OsvvmProjectScripts.tcl**
 - TCL procedures that do common simulator and project build tasks.
 - Called by StartUpShared.tcl
- **VendorScript_tool-name.tcl**
 - TCL procedures that do simulator specific actions.
 - "tool-name" = one of (ActiveHDL, GHDL, NVC, Siemens, RivieraPro, VSimSA, VCS, Xcelium, Xsim)
 - VSimSA is the one associated with ActiveHDL.
 - Called by StartUpShared.tcl
- **OsvvmDefaultSettings.tcl**
 - Default variable settings for the OSVVM Script environment.
 - Do not modify this file, instead modify LocalScriptDefaults.tcl
 - Called by StartUpShared.tcl
- **LocalScriptDefaults.tcl**
 - User default settings for the OSVVM Script environment.
 - See previous section for directions on creating this file.
 - If it exists, called by StartUpShared.tcl
- **OsvvmRequiredSettings.tcl**
 - Private settings for OSVVM.
 - Called by StartUpShared.tcl
- **CallbackDefaults.tcl**
 - Callbacks for modifying OSVVM commands and error handling
 - Do not modify this file, instead modify LocalCallbacks.tcl
 - Called by StartUpShared.tcl
- **LocalCallbacks.tcl**
 - User overloading of OSVVM CallbackDefaults.tcl
 - If it exists, called by StartUpShared.tcl
- **LocalCallbacks_tool-name.tcl**
 - Simulator specific user overloading of OSVVM CallbackDefaults.tcl
 - If it exists, called by StartUpShared.tcl

9 Generating Reports when a Simulation or Build Ends in Error

If a simulation crashed and there are no test case reports, they can be created by calling Simulate2Html as follows.

```
Simulate2Html <TestCaseName> <TestSuiteName> <TestCaseFileName>
```

If no generics are set, then `TestCaseFileName` is the same as `TestCaseName`. If generics are set, `TestCaseFileName` is `TestCaseName_GenericName_Value`.

If the build failed, use `Report2Html` to create the build summary report from the YAML file and use `Log2Osvvm` to create the HTML log file from the text base log file.

```
Report2Html <YamlFileName>  
Log2Osvvm <LogFileName>
```

10 Note on Scripts for Siemens

During simulation OSVVM suppresses QuestaSim/ModelSim messages 8683 and 8684. These are warnings about potential issues with port drivers due to QuestaSim/ModelSim using non-VHDL compliant optimizations. The potential issues these warn about do not occur with OSVVM interfaces. As a result, these warnings are suppressed because they consume significant time at the startup of simulations.

You can learn more about these messages by doing “verror 8683” or “verror 8684” from within the tool GUI.

10.1 verror 8683

An output port has no default expression in its declaration and has no drivers. The VHDL LRM-compliant value it propagates to higher-level connected signals may not be what is desired. In particular, this behavior might not correspond to the synthesis view of initialization. The vsim switch “-defaultstdlogicinittoz” or “-forcestdlogicinittoz” may be useful in this situation.

10.2 OSVVM Analysis of Message # 8683

OSVVM interfaces that is used to connect VC to the test sequencer (TestCtrl) use minimum as a resolution function. Driving the default value (type'left) on a signal has no negative impact. Hence, OSVVM disables this warning since it does not apply.

10.3 verror 8684

An output port having no drivers has been combined with a higher-level connected signal. The port will get its initial value from this higher-level connected signal; this is not compliant with the behavior required by the VHDL LRM.

LRM compliant behavior would require the port's initial value come from its declaration, however, since it was combined or collapsed with the port or signal higher in the hierarchy, the initial value came from that port or signal.

LRM compliant behavior can be obtained by preventing the collapsing of these ports with the vsim switch -donotcollapsepartiallydriven. If the port is collapsed to a port or signal with the same initialization (as is often the case of default initializations being applied),

there is no problem and the proper initialization is done and the simulation is LRM compliant.

10.4 OSVVM Analysis of Message # 8684

Older OSVVM VC use records whose elements are `std_logic_vector`. These VC initialize port values to 'Z'. QuestaSim non-VHDL compliant optimizations, such as port collapsing, remove these values. If you are using older OSVVM verification components, you can avoid any impact of this non compliant behavior if you initialize the transaction interface signal in the test harness to all 'Z'.

Hence, OSVVM disables this warning since it does not apply if you use the due care recommended above.

OSVVM recommends that you migrate older interfaces to the newer that uses types and resolution functions defined in `ResolutionPkg` such as `std_logic_max`, `std_logic_vector_max`, or `std_logic_vector_max_c` rather than `std_logic` or `std_logic_vector`. `ResolutionPkg` supports a richer set of types, such as `integer_max`, `real_max`, ...

11 Deprecated Descriptor Files

Include with a file extension of ".dirs" or ".files" is deprecated and is only supported for backward compatibility.

<Name>.dirs is a directory descriptor file that contains a list of directories. Each directory is handled by calling "include <directory>".

<Name>.files is a file descriptor that contains a list of names. Each name is handled by calling "analyze <name>". If the extension of the name is ".vhd" or ".vhdl" the file will be compiled as VHDL source. If the extension of the name is ".v" the file will be compiled as verilog source. If the extension of the name is ".lib", it is handled by calling "library <name>".

12 Release History

For the release history see, [CHANGELOG.md](#)

13 Participating and Project Organization

The OSVVM project welcomes your participation with either issue reports or pull requests. For details on [how to participate](#) see

You can find the project [Authors here](#) and [Contributors here](#).

14 More Information on OSVVM

OSVVM Forums and Blog: <http://www.osvvm.org/>

SynthWorks OSVVM Blog: <http://www.synthworks.com/blog/osvvm/>

Gitter: <https://gitter.im/OSVVM/Lobby>

Documentation: osvvm.github.io

Documentation: Documentation for the OSVVM libraries can be found [here](#)

15 Copyright and License

Copyright (C) 2006-2022 by [SynthWorks Design Inc.](#)

Copyright (C) 2022 by [OSVVM contributors](#)

This file is part of OSVVM.

Licensed under Apache License, Version 2.0 (the "License")
You may not use this file except in compliance with the License.
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.