# 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 that use the same approach

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

## 1 Clone the OSVVM-Libraries directory

Lets start by doing. The library OSVVM-Libraries contains all of the OSVVM libraries as submodules. Download the entire OSVVM model library using git clone with the `--recursive` flag:

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

## 2 The Script Files

- StartUp.tcl
  - StartUp script for ActiveHDL, GHDL, Mentor, RivieraPro, and VSimSA (ActiveHDL)
  - Detects the simulator running and calls the VendorScript_vendor-name.tcl. Also calls OsvvmProjectScripts.tcl and OsvvmScriptDefaults.tcl
- StartVCS.tcl
  - StartUp script for Synopsys VCS. Does what StartUp.tcl does except is specific to VCS
- StartXcelium.tcl
  - StartUp script for Cadence Xcelium. Does what StartUp.tcl does except is specific to Xcelium
- StartXSIM.tcl
  - StartUp script for Xilinx XSIM. Does what StartUp.tcl does except is specific to Xsim
  - Note, XSIM is currently a alpha level, experimental release.
- VendorScript_tool-name.tcl
  - TCL procedures that do simulator specific actions.

  - "tool-name" = one of (ActiveHDL, GHDL, Mentor, RivieraPro, VSimSA, VCS, Xcelium, Xsim)
  - VSimSA is the one associated with ActiveHDL.
  - Called by StartUp.tcl
- OsvvmProjectScripts.tcl
  - TCL procedures that do common simulator and project build tasks.
  - Called by StartUp.tcl
- OsvvmScriptDefaults.tcl
  - Default settings for the OSVVM Script environment.
  - Called by StartUp.tcl
- LocalScriptDefaults.tcl
  - User default settings for the OSVVM Script environment.
  - Not in OSVVM repository so it will not be replaced on OSVVM updates
  - If it exists, called by StartUp.tcl

## 3 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.

## 4 Initialization

### 4.1 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).

### 4.2 Aldec ActiveHDL

Before doing this in ActiveHDL and VSimSA (ActiveHDL's command window) instead do:

```
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.

## 4.3 GHDL

I currently run GHDL using MSYS2 64 bit under windows. The scripts must run under tcl (tclsh). As a result, to start the OSVVM scripting environment, in a shell window do:

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

To simplify this, I put the `source .../StartUp.tcl` in my `.tclshrc` file and as a result I do not have to do the source command. I have added a short cut that includes `C:\tools\msys64\mingw64.exe winpty rlwrap tclsh`. I added the short cut to my start menu. With these two, one click and you are running in the OSVVM tcl execution environment.

Alternately, if you are not running in windows, create the `.tclshrc` as above and then in your `.bashrc` create the alias `alias gsim='winpty rlwrap tclsh'` to simplify starting tclsh. From there, at the command line type gsim and you are running ghdl in the OSVVM environment.

## 4.4 Synopsys VCS

Synopsys scripts are beta level quality. VCS runs under Unix/Linux. The scripts must run under tcl (tclsh). As a result, to start the OSVVM scripting environment, in a shell window do:

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

To simplify this, I put the `source .../StartVCS.tcl` in my `.tclshrc` file and as a result I do not have to do the source command.

## 4.5 Cadence Xcelium

Cadence Xcelium scripts are alpha level quality. Xcelium runs under Unix/Linux. The scripts must run under tcl (tclsh). As a result, to start the OSVVM scripting environment, in a shell window do:

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

To simplify this, I put the `source .../StartXcelium.tcl` in my `.tclshrc` file and as a result I do not have to do the source command.

### 4.6 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 script shown below:

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

If someone from XILINX is interested, the internal OSVVM utility library testbenches can be provided under an NDA.

## 5 Project Files

A project file is a TCL script that allows the specification of basic tasks to run a simulation:

- library - Make this library the active library. Create it if it does not exist.
- analyze - Compile the design into the active library.
- Simulate - Simulate the design using the active library.
- RunTest - compile and simulate in one step
- include – include another project script
- build – include + start a new log file for this task
- TestSuite - identifies the TestSuite that is active
- TestCase - identify the TestCase that is active

The above tasks are TCL procedures. Hence, a project file is actually a TCL file, and when necessary, TCL can be used, however, the intent is to keep it simple. The naming of the project file is of the form <Name>.pro.

### 5.1 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
simulate TbAxi4_Shared1
```

In OSVVM scripting, calling library activates the library. Hence, there is no need to specify the library in analyze and simulate. This is consistent with VHDL’s sense of the "working library".

## 5.2 Adding Scripts to Simulate

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.

As a result, when simulate runs, it will also include the following files in order, if they exist:

- OsvvmLibraries/Scripts/"ToolVendor".tcl
- OsvvmLibraries/Scripts/"simulator".tcl
- "sim-run-dir"/"ToolVendor".tcl
- "sim-run-dir"/"simulator".tcl
- "sim-run-dir"/"LibraryUnit".tcl
- "sim-run-dir"/"LibraryUnit"_"simulator".tcl
- "sim-run-dir"/wave.do

ToolVendor is either {Aldec, Siemens, Cadence, Synopsys}. Simulator is one of {QuestaSim, ModelSim, RivieraPRO, ActiveHDL, VCS, Xcelium}. LibraryUnit is the name of the design being simulated. "Sim run dir" is the directory from which you run the simulator.

Currently GHDL does not run any extra scripts since it is a batch simulator.

## 5.3 Including Scripts

We build our designs hierarchically. Therefore our scripts need to be build hierarchically.

In TCL, source (or with EDA tools do) is used to run lower level scripts. For an EDA tool, there are two directories of interest, the directory the tool is running in and the script directory. EDA tool settings are directory dependent, so the script cannot change directories. This forces the script to manage the location of the files referenced in the script and makes scripting awkward.

To address this situation, OSVVM adds include. Include manages both the run directory and the script directory. All calls to the OSVVM script API that reference a file, reference the file relative to the current script directory – rather than the run directory. Hence, a script only needs to reference files relative to its location, making path references simple. It also simplifies making directory structure changes to a project.

The script, OsvvmLibraries.pro shown below, is the starting point for compiling all of OSVVM. It simply calls the scripts from the OSVVM Utility library, Verification Component Common library, UART verification component library, and AXI4 verification component libraries using include.

```
include ./osvvm/osvvm.pro
include ./Common/Common.pro
include ./UART/UART.pro
include ./AXI4/AXI4.pro
```

## 5.4 Building the OSVVM Libraries

Build is a layer on top of include (it calls include) that creates a logging point. By default, OSVVM creates collects all tool output for a build into an html based log file in ./logs/<tool_name>-<version>/<build>.html.

In addition, when a test is started with build, run with simulate, and includes a call to "EndOfTestReports" at the end of the VHDL testbench, a build report with the formats YAML, HTML, and JUnit XML reporting. In addition, a detailed test report (in YAML and HTML) will be created that lists out all AlertLogIDs and their state (PASSED, FAILED, Errors, ...) as well as reports of coverage models (if the test has any).

In general, build is called from the simulator API (when we run something) and include is called from scripts.

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

```
build <path_to_OsvvmLibraries>/OsvvmLibraries.pro
```

## 5.5 Running OSVVM Libraries

To run the full OSVVM verification component regression suite use the build shown below.

```
build <path_to_OsvvmLibraries>/RunAllTests.pro
```

Everything in OSVVM is composed hierarchically. So if you want to run a regression on a particular verification component, you simply run its build. These are shown below.

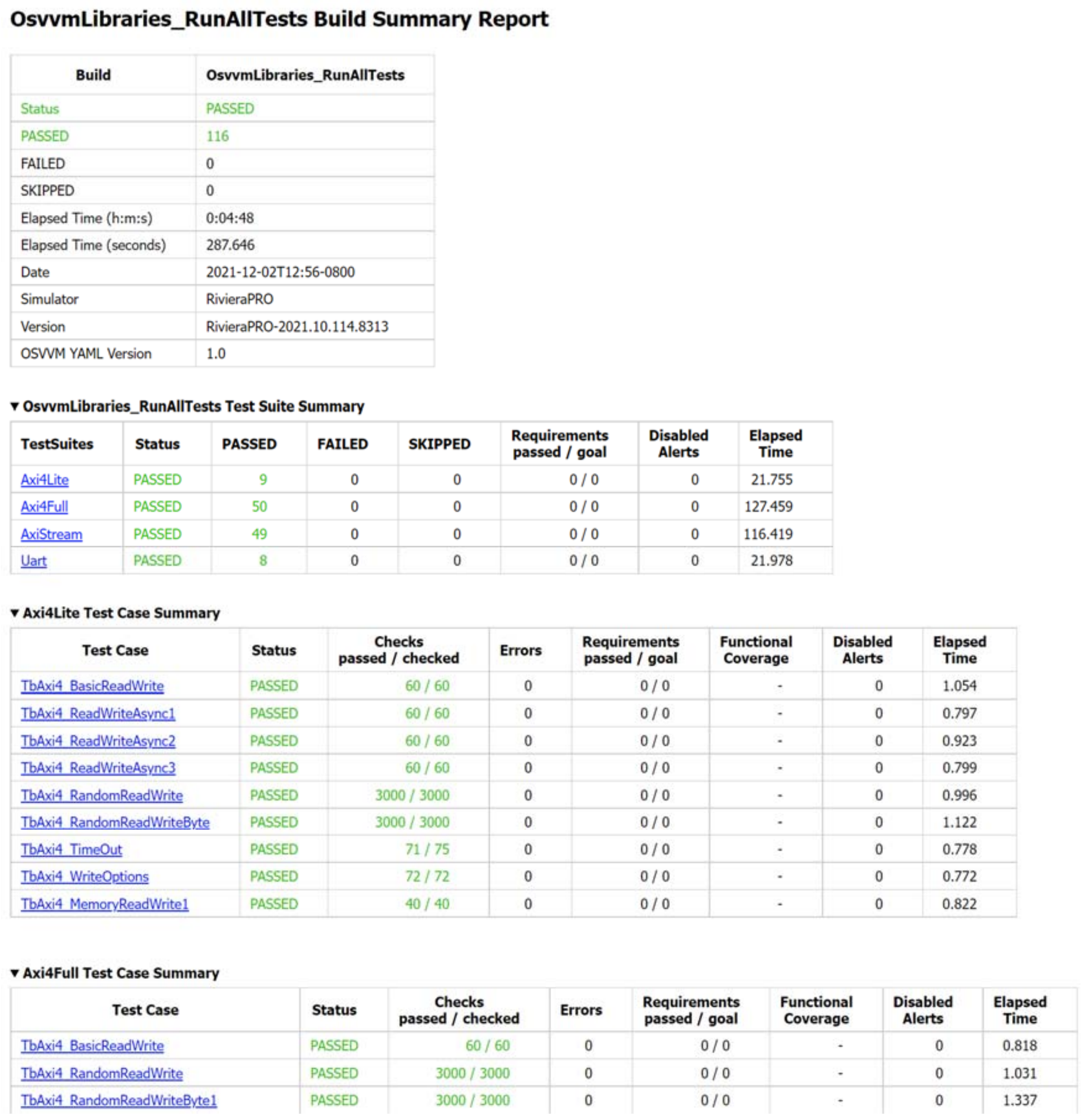
```
build ../AXI4/Axi4/RunAllTests.pro
build ../AXI4/Axi4Lite/RunAllTests.pro
build ../AXI4/AxiStream/RunAllTests.pro
build ../UART/RunAllTests.pro
```

## 5.6 Build Summary Report

The build summary report is a summary of all tests run during that build in YAML, HTML, and JUnit XML. This report allows us to confirm that all tests finished successfully.

The best way to see the reports is run one of the OSVVM regressions. Run build OsvvmLibraries/RunAllTests.pro, then open the file OsvvmLibraries_RunAllTests.html.

The following is an excerpt of OsvvmLibraries_RunAllTests.html. Failures are shown in red (none here). Passing tests are shown in green.



**OsvvmLibraries_RunAllTests Build Summary Report**

| Build | OsvvmLibraries_RunAllTests |
|---|---|
| Status | PASSED |
| PASSED | 116 |
| FAILED | 0 |
| SKIPPED | 0 |
| Elapsed Time (h:m:s) | 0:04:48 |
| Elapsed Time (seconds) | 287.646 |
| Date | 2021-12-02T12:56-0800 |
| Simulator | RivieraPRO |
| Version | RivieraPRO-2021.10.114.8313 |
| OSVVM YAML Version | 1.0 |

▾ **OsvvmLibraries_RunAllTests Test Suite Summary**

| TestSuites | Status | PASSED | FAILED | SKIPPED | Requirements passed / goal | Disabled Alerts | Elapsed Time |
|---|---|---|---|---|---|---|---|
| Axi4Lite | PASSED | 9 | 0 | 0 | 0 / 0 | 0 | 21.755 |
| Axi4Full | PASSED | 50 | 0 | 0 | 0 / 0 | 0 | 127.459 |
| AxiStream | PASSED | 49 | 0 | 0 | 0 / 0 | 0 | 116.419 |
| Uart | PASSED | 8 | 0 | 0 | 0 / 0 | 0 | 21.978 |

▾ **Axi4Lite Test Case Summary**

| Test Case | Status | Checks passed / checked | Errors | Requirements passed / goal | Functional Coverage | Disabled Alerts | Elapsed Time |
|---|---|---|---|---|---|---|---|
| TbAxi4_BasicReadWrite | PASSED | 60 / 60 | 0 | 0 / 0 | - | 0 | 1.054 |
| TbAxi4_ReadWriteAsync1 | PASSED | 60 / 60 | 0 | 0 / 0 | - | 0 | 0.797 |
| TbAxi4_ReadWriteAsync2 | PASSED | 60 / 60 | 0 | 0 / 0 | - | 0 | 0.923 |
| TbAxi4_ReadWriteAsync3 | PASSED | 60 / 60 | 0 | 0 / 0 | - | 0 | 0.799 |
| TbAxi4_RandomReadWrite | PASSED | 3000 / 3000 | 0 | 0 / 0 | - | 0 | 0.996 |
| TbAxi4_RandomReadWriteByte | PASSED | 3000 / 3000 | 0 | 0 / 0 | - | 0 | 1.122 |
| TbAxi4_TimeOut | PASSED | 71 / 75 | 0 | 0 / 0 | - | 0 | 0.778 |
| TbAxi4_WriteOptions | PASSED | 72 / 72 | 0 | 0 / 0 | - | 0 | 0.772 |
| TbAxi4_MemoryReadWrite1 | PASSED | 40 / 40 | 0 | 0 / 0 | - | 0 | 0.822 |

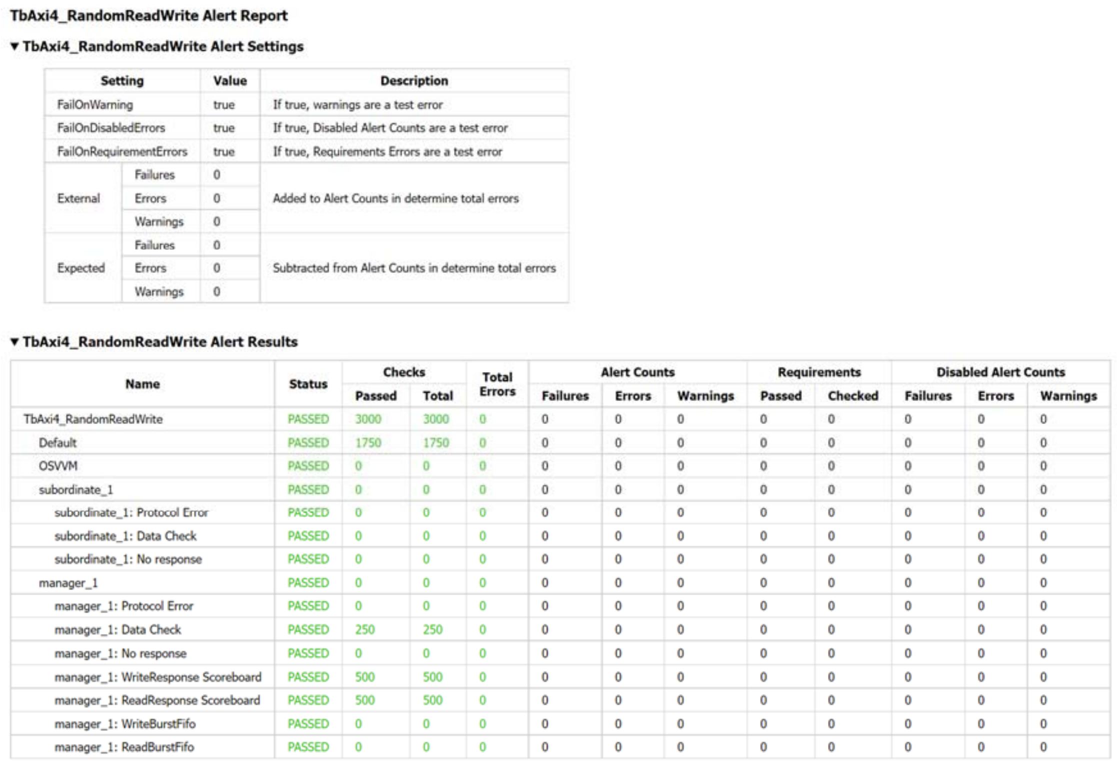
▾ **Axi4Full Test Case Summary**

| Test Case | Status | Checks passed / checked | Errors | Requirements passed / goal | Functional Coverage | Disabled Alerts | Elapsed Time |
|---|---|---|---|---|---|---|---|
| TbAxi4_BasicReadWrite | PASSED | 60 / 60 | 0 | 0 / 0 | - | 0 | 0.818 |
| TbAxi4_RandomReadWrite | PASSED | 3000 / 3000 | 0 | 0 / 0 | - | 0 | 1.031 |
| TbAxi4_RandomReadWriteByte1 | PASSED | 3000 / 3000 | 0 | 0 / 0 | - | 0 | 1.337 |

## 5.7 Detailed Test Report

The detailed test report is a detailed report of alerts and coverage models. The best way to see the reports is run one of the OSVVM regression suites. After running one of the regressions, open one of the HTML files in the directory ./reports (default directory).

The first half of the report is a summary of the alerts encountered for each AlertLogID during the test. This is shown in the following figure.



**TbAxi4_RandomReadWrite Alert Report**

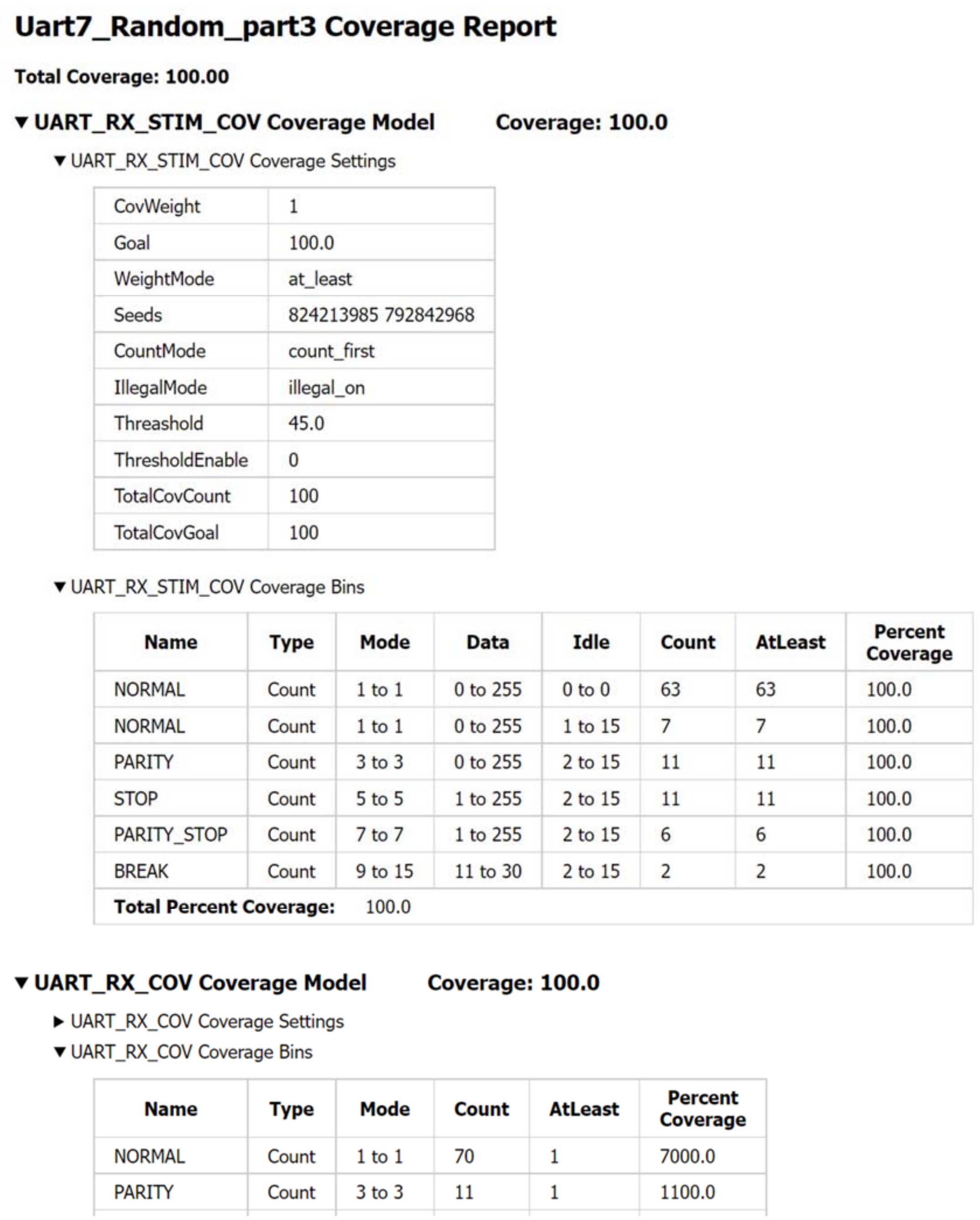
▾ **TbAxi4_RandomReadWrite Alert Settings**

| 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 | |
| Expected | Failures | 0 | Subtracted from Alert Counts in determine total errors |
| | Errors | 0 | |
| | Warnings | 0 | |

▾ **TbAxi4_RandomReadWrite Alert Results**

| Name | Status | Checks | | Total Errors | Alert Counts | | | Requirements | | Disabled Alert Counts | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Passed | Total | | Failures | Errors | Warnings | Passed | Checked | Failures | Errors | Warnings |
| TbAxi4_RandomReadWrite | PASSED | 3000 | 3000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Default | PASSED | 1750 | 1750 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OSVVM | PASSED | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| subordinate_1 | PASSED | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| subordinate_1: Protocol Error | PASSED | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| subordinate_1: Data Check | PASSED | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| subordinate_1: No response | PASSED | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| manager_1 | PASSED | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| manager_1: Protocol Error | PASSED | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| manager_1: Data Check | PASSED | 250 | 250 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| manager_1: No response | PASSED | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| manager_1: WriteResponse Scoreboard | PASSED | 500 | 500 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| manager_1: ReadResponse Scoreboard | PASSED | 500 | 500 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| manager_1: WriteBurstFifo | PASSED | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| manager_1: ReadBurstFifo | PASSED | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The second half of the report is the coverage report for each coverage model defined in the test environment. Note that items with a triangle in front of them can be closed for more compact viewing. The coverage report below is for test TbCov_CovDb_2 in the OSVVM utility library regression suite. Note that coverage model "Test 3" is closed for more compact viewing.



## 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 |
| Threashold | 45.0 |
| ThresholdEnable | 0 |
| TotalCovCount | 100 |
| TotalCovGoal | 100 |

▼ UART_RX_STIM_COV Coverage Bins

| Name | Type | Mode | Data | Idle | Count | AtLeast | Percent Coverage |
|---|---|---|---|---|---|---|---|
| NORMAL | Count | 1 to 1 | 0 to 255 | 0 to 0 | 63 | 63 | 100.0 |
| NORMAL | Count | 1 to 1 | 0 to 255 | 1 to 15 | 7 | 7 | 100.0 |
| PARITY | Count | 3 to 3 | 0 to 255 | 2 to 15 | 11 | 11 | 100.0 |
| STOP | Count | 5 to 5 | 1 to 255 | 2 to 15 | 11 | 11 | 100.0 |
| PARITY_STOP | Count | 7 to 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 to 1 | 70 | 1 | 7000.0 |
| PARITY | Count | 3 to 3 | 11 | 1 | 1100.0 |

## 5.8 VHDL Aspects of Generating Reports

To generate reports, you need to have the following OSVVM test code in your VHDL testbench. 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
  SetAlertLogName("TestName") ;
  . . .
  -- Do some Checks
  AffirmIfEqual(Data, X"A025", "Check Data") ;
  . . .
  -- Generate Reports (replaces call to ReportAlerts)
  EndOfTestReports ;
  std.env.stop(GetAlertCount) ;
end process TestProc ;
```

## 5.9 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 SetAlertLogName as otherwise, a NAME_MISMATCH failure will be generated.

## 5.10 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 this 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 SetAlertLogName (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.

## 5.11 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 declartion 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.

## 5.12 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, then 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.

## 6.1 library <library> [<path>]

Make the library the active library. If the library does not exist, create it and create a mapping to it. Libraries are created in the path specified by LIB_BASE_DIR in Scripts/StartUp.tcl.

## 6.2 LinkLibrary <library> [<path>]

Create a mapping to a library that was already created.

## 6.3 LinkLibraryDirectory [LibraryDirectory]

Map all of the libraries in the specified library directory. If one is not specified, one is looked for in the current directory.

## 6.4 SetLibraryDirectory [LibraryDirectory]

Set the directory in which the libraries will be created. The libraries will be created in this directory in a directory named, "VHDL_LIBS/<tool version>/".

## 6.5 GetLibraryDirectory

Get the Library Directory.

## 6.6 analyze <file>

Compile the file. A path name specified is relative to the location of the current <file>.pro directory location. Library is the one specified in the previous library command.

## 6.7 simulate <design-unit>

Start a simulation on the design unit. Library is the one specified in the previous library command.

## 6.8 RunTest <file> [<name>]

RunTest combines TestCase, analyze, and simulate. RunTest optionally takes two parameters. With two parameters, the first is the file name to analyze, the second is the design unit name to use for simulation and TestCase. With one parameter, the first parameter is the file name. The second parameter is the base name of the file name - any path and file extension are removed.

## 6.9 include [<path>/]<name>

Include accepts an argument "name" that is either a file or a directory. If it is a file and its extension is .pro, .tcl, or .do, it will be sourced.

If "name" is a directory, then files whose name is "name" and whose extension is .pro, .tcl, or .do, it will be sourced.

All paths and names specifed are relative are relative to the current directory from which the script is running.

Extensions of the form ".files" or ".dirs is handled in a manner described in "Deprecated Descriptor Files".

## 6.10 build [<path>/]<name>

Re-initializes the working directory to the script directory, opens a transcript file, and calls include. A path name specified is relative to the location of the current <file>.pro directory location.

### 6.11 SetVHDLVersion

Set VHDL analyze version. Valid values = (2008, 2019, 1993, 2002). OSVVM libraries require 2008 or newer.

### 6.12 GetVHDLVersion

Return the current VHDL Version.

### 6.13 SetSimulatorResolution

Set Simulator Resolution. Any value supported by the simulator is ok.

### 6.14 GetSimulatorResolution

Return the current Simulator Resolution.

### 6.15 TestCase <name>

Set the test case name.

### 6.16 SkipTest <string>

Adds a place holder in the results file for a test case that does not run in a particular tool. The <string> value specifies the reason the test case was skipped in this tool.

### 6.17 TestSuite <name>

Set the test suite name.

### 6.18 SetTranscriptType

Select the Transcript file to be either html or log. The default is html.

### 6.19 GetTranscriptType

Get the Transcript file type (either html or log).

### 6.20 SetCoverageAnalyzeEnable ""

Value true selects use coverage options during analyze. Value "" sets the enable to the specified by SetCoverageEnable.

### 6.21 GetCoverageAnalyzeEnable

Returns the current enable setting.

### 6.22 SetCoverageAnalyzeOptions "value"

Set the coverage options.

## 6.23 GetCoverageAnalyzeOptions

Set the coverage options for analyze to the string value.

## 6.24 SetCoverageSimulateEnable ""

Value true selects use coverage options during simulation. Value "" sets the enable to the specified by SetCoverageEnable.

## 6.25 GetCoverageSimulateEnable

Returns the current enable setting.

## 6.26 SetCoverageSimulateOptions "value"

Set the coverage options for simulate to the string value.

## 6.27 GetCoverageSimulateOptions

Get the coverage options to the string value.

## 6.28 SetCoverageEnable "value"

Set CoverageEnable to value. If no value is specified, then the value is "true". The default value is "true"

## 6.29 GetCoverageEnable

Get the CoverageEnable value.

## 6.30 SetVhdlAnalyzeOptions "value"

Set the VHDL options for analyze to the string value.

## 6.31 GetVhdlAnalyzeOptions

Get the VHDL options for analyze.

## 6.32 SetVerilogAnalyzeOptions "value"

Set the Verilog options for analyze to the string value.

## 6.33 GetVerilogAnalyzeOptions

Get the Verilog options for analyze.

## 6.34 SetExtendedAnalyzeOptions "value"

Set extended (additional) options for analyze to the string value.

### 6.35 GetExtendedAnalyzeOptions

Get extended (additional) options for analyze.

### 6.36 SetExtendedSimulateOptions "value"

Set extended (additional) options for simulate to the string value.

### 6.37 GetExtendedSimulateOptions

Get extended (additional) options for simulate.

### 6.38 LinkCurrentLibraries

EDA tools are centric to the current directory If you change directory, the EDA tool will loose all of its library mapping information. If immediately after a change directory, a LinkCurrentLibraries is called, then all existing libraries will be mapped in the current directory.

### 6.39 RemoveAllLibraries

Delete all of the working libraries.

### 6.40 Undocumented Items

Anything undocumented is experimental and may change or be removed in a future revision.

## 7 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>".

## 8 Release History

For the release history see, CHANGELOG.md

## 9 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.

## 10 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:** Documentation for the OSVVM libraries can be found here

## 11 Copyright and License