1 About OSVVM

OSVVM is an advanced verification methodology that defines a VHDL verification framework, verification utility library, verification component library, scripting API, and co-simulation capability that simplifies your FPGA or ASIC verification project from start to finish. Using these libraries you can create a simple, readable, and powerful testbench that is suitable for either a simple FPGA block (aka a unit level test) or a complex ASIC (a full chip and/or system test).

OSVVM is developed by the same VHDL experts who have helped develop VHDL standards. We have used our expert VHDL skills to create advanced verification capabilities that provide:

- A structured transaction-based verification framework using verification components.
- An accelerated approach for developing verification components that uses the transaction interface and API defined in our transaction support library.
- Improved reuse and reduced project schedules.
- Improved readability and reviewability by the whole team including hardware, software, and system engineers.
- Buzz word verification features including Constrained Random, Functional Coverage, Scoreboards, FIFOs, Memory Models, error logging and reporting, and message filtering that are simple to use and work like built-in language features.
- Co-simulation of C++ models in a hardware simulator environment.
- A scripting API that runs any simulator using the same script. OSVVM scripting supports GHDL, NVC, Aldec Riviera-PRO and ActiveHDL, Siemens Questa and ModelSim, Synopsys VCS, and Cadence Xcelium.
- Unmatched test reporting with HTML based test suite reports, test case reports, and logs that facilitate debug and test artifact collection.
- Support for continuous integration (CI/CD) with JUnit XML test suite reporting.
- A rival to the verification capabilities of SystemVerilog + UVM.

Looking to improve your VHDL verification methodology? OSVVM provides a complete solution for VHDL ASIC or FPGA verification. There is no new language to learn. It is simple, powerful, and concise. Each piece can be used separately. Hence, you can learn and adopt pieces as you need them.

Important benefits of OSVVM:

- **Each piece is independent**
  - Add them to your current VHDL testbench incrementally.
- **Verification framework that is**
  - Simple enough to use on small blocks
  - So simple in fact that we don't need a "Lite" or "Easy" approach
  - It is powerful enough to use on large, complex FPGAs and ASICs

  - Using the same framework architecture for RTL, Core, and System tests facilitates reuse between them
  - Test cases are readable by RTL, verification, software, and system engineers
  - It is simple enough that you just need VHDL engineers and not verification specialists.
  - Our Model Independent Transactions (MIT) define a common set of transactions for Address Bus and Streaming Interfaces
- **Verification utility library that**
  - Simplifies Self-checking, Error handling, and Message Filtering
  - Implements Constrained Random, Functional Coverage, Scoreboards, FIFOs, Memory Models
  - Is simple to use and works like built-in language features
- **Co-Simulation capability that supports**
  - Running C++ models and code on simulated hardware
    - Instruction set simulators or cycle accurate processor models
    - System models with a mix of software models and logic IP
  - Writing of tests in C++
    - Test development environment extended to software engineers
  - Connection to an external program via TCP/IP sockets
    - Where model might not be callable directly from the C++ code
- **Unmatched Test reporting**
  - JUnit XML for use with continuous integration (CI/CD) tools.
  - HTML Build Summary Report for reporting test suite level information
  - HTML Test Case Detailed report for each test case.
  - HTML based Alert, Functional Coverage, and Scoreboard Reports
  - HTML based test transcript/log files
  - Find and debug issues faster
- **Verification component library**
  - Free open source verification components for AXI4 Full, AXI4 Lite, AXI Stream, UART, and DPRAM
  - More in progress
- **One Script to Run Simulators**
  - Same script supports GHDL, Aldec Riviera-PRO and ActiveHDL, Siemens QuestaSim and ModelSim, Synopsys VCS, and Cadence Xcelium
- **It is free open source.**
  - It upgrades an ordinary VHDL license with full featured verification capabilities.

SynthWorks has been using OSVVM for 25+ years in our training classes and consulting work. During that time, we have innovated new capabilities and evolved our existing ones to increase re-use and reduce effort and time spent.

2 Getting Started with OSVVM

The best way to get started with OSVVM is to run the demo examples.

2.1 Download OSVVM

OSVVM is available as either a git repository OsvvmLibraries or a zip file from OSVVM 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
```

2.2 Running the Demos

A great way to get oriented with OSVVM is to run the demos.

If you are using Aldec's Rivera-PRO or Siemen's QuestaSim/ModelSim do the following.

- Step 1: Create a directory named sim that is in the same directory that contains the OsvvmLibraries directory.
- Step 2: Start your simulator and go to the sim directory.
- Step 3: Do the following in your simulator command line:

```
source ../OsvvmLibraries/Scripts/StartUp.tcl
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.

If you are using GHDL, Aldec Active-HDL, Synopsys VCS, or Cadence Xcelium, see the OSVVM Script User Guide for details on running the scripts in these tool. The start up aspect for each of these is slightly different, however, once you get the simulator into OSVVM script mode, running the scripts is the same.

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

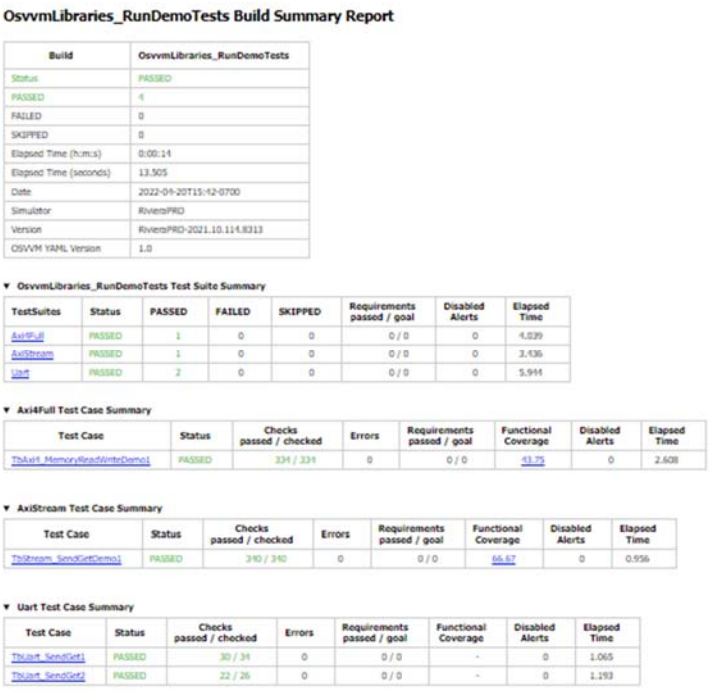
The Build Summary Report allows us to quickly confirm if a build passed or quickly identify which test cases did not PASS.

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, shown below, is in the file OsvvmLibraries_RunDemoTests.html.



OsvvmLibraries_RunDemoTests Build Summary Report

Build	OsvvmLibraries_RunDemoTests
Status	PASSED
PASSED	4
FAILED	0
SKIPPED	0
Elapsed Time (h:m:s)	0:00:14
Elapsed Time (seconds)	13.505
Date	2022-04-20T15:42-0700
Simulator	RivieraPRO
Version	RivieraPRO-2021.10.114.8313
OSVVM YAML Version	1.0

▼ OsvvmLibraries_RunDemoTests Test Suite Summary

TestSuites	Status	PASSED	FAILED	SKIPPED	Requirements passed / goal	Disabled Alerts	Elapsed Time
Axi4Full	PASSED	1	0	0	0 / 0	0	4.039
AxiStream	PASSED	1	0	0	0 / 0	0	3.436
Uart	PASSED	2	0	0	0 / 0	0	5.944

▼ Axi4Full Test Case Summary

Test Case	Status	Checks passed / checked	Errors	Requirements passed / goal	Functional Coverage	Disabled Alerts	Elapsed Time
TbAxi4_MemoryReadWriteDemo1	PASSED	331 / 331	0	0 / 0	43.75	0	2.608

▼ AxiStream Test Case Summary

Test Case	Status	Checks passed / checked	Errors	Requirements passed / goal	Functional Coverage	Disabled Alerts	Elapsed Time
TbStream_SendGetDemo1	PASSED	340 / 340	0	0 / 0	66.67	0	0.956

▼ 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	1.065
TbUart_SendGet2	PASSED	22 / 26	0	0 / 0	-	0	1.193

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_RunDemoTests Build Summary Report

Build	OsvvmLibraries_RunDemoTests
Status	PASSED
PASSED	4
FAILED	0
SKIPPED	0
Elapsed Time (h:m:s)	0:00:14
Elapsed Time (seconds)	13.505
Date	2022-04-20T15:42-0700
Simulator	RivieraPRO
Version	RivieraPRO-2021.10.114.8313
OSVVM YAML Version	1.0

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_RunDemoTests Test Suite Summary**

TestSuites	Status	PASSED	FAILED	SKIPPED	Requirements passed / goal	Disabled Alerts	Elapsed Time
Axi4Full	PASSED	1	0	0	0 / 0	0	4.039
AxiStream	PASSED	1	0	0	0 / 0	0	3.436
Uart	PASSED	2	0	0	0 / 0	0	5.944

Test Suite Summary

3.1.3 Test Case Summary

The remainder of the Build Summary Report is Test Case Summary, shown 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	2.608

▼ AxiStream Test Case Summary

Test Case	Status	Checks passed / checked	Errors	Requirements passed / goal	Functional Coverage	Disabled Alerts	Elapsed Time
TbStream_SendGetDemo1	PASSED	340 / 340	0	0 / 0	66.67	0	0.956

▼ 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	1.065
TbUart_SendGet2	PASSED	22 / 26	0	0 / 0	-	0	1.193

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>.

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 Test Information link table 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.

TbStream_SendGetDemo1 Test Case Detailed Report

Available Reports
Alert Report
Functional Coverage Report(s)
ScoreboardPkg_slv Report(s)
Link to Simulation Results
./results/TbStream_SendGetDemo1.txt

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.

TbStream_SendGetDemo1 Alert Report

▼ **TbStream_SendGetDemo1 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	

▼ **TbStream_SendGetDemo1 Alert Results**

Name	Status	Checks		Total Errors	Alert Counts			Requirements		Disabled Alert Counts		
		Passed	Total		Failures	Errors	Warnings	Passed	Checked	Failures	Errors	Warnings
TbStream_SendGetDemo1	PASSED	340	340	0	0	0	0	0	0	0	0	0
Default	PASSED	67	67	0	0	0	0	0	0	0	0	0
OSVVM	PASSED	0	0	0	0	0	0	0	0	0	0	0
Cov1	PASSED	0	0	0	0	0	0	0	0	0	0	0
Cov2	PASSED	0	0	0	0	0	0	0	0	0	0	0
Cov1a	PASSED	0	0	0	0	0	0	0	0	0	0	0
Cov2a	PASSED	0	0	0	0	0	0	0	0	0	0	0
Cov1b	PASSED	0	0	0	0	0	0	0	0	0	0	0
Cov2b	PASSED	0	0	0	0	0	0	0	0	0	0	0
transmitter_1	PASSED	0	0	0	0	0	0	0	0	0	0	0
No response	PASSED	0	0	0	0	0	0	0	0	0	0	0
TransmitFifo	PASSED	0	0	0	0	0	0	0	0	0	0	0
TxBurstFifo	PASSED	0	0	0	0	0	0	0	0	0	0	0
receiver_1	PASSED	30	30	0	0	0	0	0	0	0	0	0
Data Check	PASSED	32	32	0	0	0	0	0	0	0	0	0
No response	PASSED	0	0	0	0	0	0	0	0	0	0	0
ReceiveFifo	PASSED	0	0	0	0	0	0	0	0	0	0	0
RxBurstFifo	PASSED	211	211	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 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

Functional Coverage Report

3.3.4 Scoreboard Report(s)

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

TbStream_SendGetDemo1 Scoreboard Report

▼ **TransmitFifo Scoreboard**

Name	TransmitFifo
ItemCount	317
ErrorCount	0
ItemsChecked	0
ItemsPopped	317
ItemsDropped	0

▼ **ReceiveFifo Scoreboard**

Name	ReceiveFifo
ItemCount	317
ErrorCount	0
ItemsChecked	0
ItemsPopped	317
ItemsDropped	0

▼ **TxBurstFifo Scoreboard**

Name	TxBurstFifo
ItemCount	253
ErrorCount	0
ItemsChecked	0
ItemsPopped	253
ItemsDropped	0

Scoreboard Report

3.4 Test Case Transcript

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

HTML Simulator Transcript

4 Structured Testbench Framework

Some methodologies (or frameworks) are so complex that you need a script to create initial starting point for writing verification components, test cases, and/or the test harness. SystemVerilog + UVM is certainly like this. There are even several organizations that propose that you use their "Lite" or "Easy" approach.

OSVVM is simple enough to use on small blocks and powerful enough to use on large, complex chips or systems. This allows us to use the same style of framework for RTL, Core, and Chip level verification - which in turn facilitates re-use of verification components and test cases. OSVVM has added the abstractions needed to make our verification component based approach as easy as the "Lite" approach of other methodologies.

SynthWorks has been using this framework for 25+ years in our training classes and consulting work. During that time, we have innovated new capabilities and evolved our existing ones to increase re-use and reduce effort spent.

When we examine OSVVM's framework in detail, we see that it has many similar elements to SystemVerilog + UVM. However, one thing not present is OO language constructs. Instead OSVVM uses ordinary VHDL constructs, such as structural and behavioral code. This makes it readily accessible to both verfication and RTL engineers.

4.1 Going Further

Read the following documents for more information on OSVVM's Structured Testbench Framework.

Document	User Guide
OSVVM's Structured Testbench Framework	OSVVM_structured_testbench_framework.pdf
OSVVM's Verification Component Developer's Guide	OSVVM_verification_component_developers_guide.pdf
OSVVM's Test Writers User Guide	OSVVM_test_writers_user_guide.pdf
OSVVM's Address Bus Model Independent Transactions Users Guide	Address_Bus_Model_Independent_Transactions_user_guide.pdf
OSVVM's Stream Model Independent Transactions Users Guide	Stream_Model_Independent_Transactions_user_guide.pdf

5 VHDL Utility Library

The OSVVM Utility Library <osvvm> implements the advanced verification capabilities found in other verification languages (such as SystemVerilog and UVM) as packages. The list below lists out many of the OSVVM features and the package in which they are implemented.

- Constrained Random test generation (RandomPkg)
- Functional Coverage with hooks for UCIS coverage database integration (CoveragePkg)
- Intelligent Coverage Random test generation (CoveragePkg)
- Utilities for testbench process synchronization generation (TbUtilPkg)
- Utilities for clock and reset generation (TbUtilPkg)
- Transcript files (TranscriptPkg)
- Error logging and reporting - Alerts and Affirmations (AlertLogPkg)
- Message filtering - Logs (AlertLogPkg)
- Scoreboards and FIFOs (data structures for verification) (ScoreboardGenericPkg)
- HTML and JUnit XML test reporting (ReportPkg, AlertLogPkg, CoveragePkg, ScoreboardGenericPkg)

- Memory models (MemoryPkg)
- Transaction-Level Modeling Support (TbUtilPkg, ResolutionPkg)

Through the years, the packages have been updated many times. Now, all of the packages that create data structures (AlertLogPkg, CoveragePkg, ScoreboardGenericPkg, and MemoryPkg) use singleton data structures. Usage of singletons simplifies API to an ordinary call interface - ie: no more shared variables and protected types.

5.1 Going Further

Read the following documents for more information on OSVVM's VHDL Utility Library.

Document	User Guide	Quick Reference
OSVVM Test Writer's User Guide - a general overview to usage	OSVVM_test_writers_user_guide.pdf	None
AlertLogPkg	AlertLogPkg_user_guide.pdf	AlertlogPkg_quickref.pdf
TranscriptPkg	TranscriptPkg_user_guide.pdf	TranscriptPkg_quickref.pdf
RandomPkg	RandomPkg_user_guide.pdf	RandomPkg_quickref.pdf
CoveragePkg	CoveragePkg_user_guide.pdf	CoveragePkg_quickref.pdf
ScoreboardGenericPkg	ScoreboardPkg_user_guide.pdf	Scoreboard_QuickRef.pdf
TbUtilPkg	TbUtilPkg_user_guide.pdf	TbUtilPkg_quickref.pdf
MemoryPkg	MemoryPkg_user_guide.pdf	None
TextUtilPkg	TextUtilPkg_user_guide.pdf	None

6 Scripting Library

The goal of OSVVM's scripting is to have "One Script to Run them All" - where all is any simulator.

Current supported simulators are

- GHDL (Free Open Source simulator),
- Aldec's Active-HDL and Riviera-PRO,
- Siemen's ModelSim and QuestaSim,
- Synopsys' VCS, and
- Cadence's Xcelium.

OSVVM scripts are a TCL based API layer that provides a tool independent means to simulate (and perhaps in the future synthesize) your design. The API uses TCL procedures to create the abstraction layers – which is why they have the extension .pro.

The scripts are executable TCL, so the full power of TCL can be used when needed (such as is in osvvm.pro).

6.1 Going Further

Read the following documents for more information on OSVVM's Scripting.

OSVVM Script User Guide	Script_user_guide.pdf
OSVVM Script library	OSVVM-Scripts

7 OSVVM Verification Component Library

OSVVM's growing verification component library is tabulated below.

Verification Component(s)	User Guide	Repository
Axi4 Full (Manager, Memory, and Subordinate) VCs	Axi4_VC_user_guide.pdf	AXI4
Axi4 Lite (Manager, Memory, and Subordinate) VCs	Axi4_VC_user_guide.pdf	AXI4
AxiStream Transmitter and Receiver VCs	AxiStream_user_guide.pdf	AXI4
UART Transmitter and Receiver VCs	None	UART
DpRam behavioral model and DpRam controller	None	DpRam

Note all of the OSVVM verification components use the model independent transaction interfaces which are defined in OSVVM-Common. It is required to be in the directory OsvvmLibraries/Common.