1 OSVVM Overview

1.1 About OSVVM

OSVVM is an advanced verification methodology that defines a verification framework, VHDL utility library, VHDL verification component library, and a scripting library.

OSVVM supports the same capabilities that verification languages such as SystemVerilog + UVM support - including transaction level modeling, verification components, functional coverage, constrained random tests, Intelligent Coverage random tests, data structures (such as scoreboards, FIFOs, and Memories), error logging and reporting, message filtering, and advanced test reporting (HTML and JUnit XML for CI/CD).

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
  - We don't use OO language constructs so RTL engineers can write test environments
  - Our Model Independent Transactions (MIT) define a common set of transactions for Address Bus and Streaming Interfaces
- **Utility Library**
  - Simplifies Self-checking, Error handling, and Message Filtering
  - Constrained Random, Functional Coverage, Scoreboards, FIFOs, Memory Models
  - Capabilities are simple to use and feel like built-in language features
- **Unmatched Test reporting**
  - JUnit XML for use with CI/CD tools.
  - HTML Build Summary Report for reporting test suite level information
  - HTML Test Case Detailed report for each test case.
  - HTML based test 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.

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 are easy to use.

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.

1.2 Getting Started with OSVVM

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

1.2.1 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
```

1.2.2 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.

1.3 OSVVM Reports

If you did not run the demos in the last step, if you go back and do it, you can explore a live version of 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.

Lets take a look at the reports created in the demo.

1.3.1 Explore the Build Summary Report

In your sim directory, open the file OsvvmLibraries_RunDemoTests.html in an internet browser. This is OSVVM's Build Summary Report. See `Figure {number} {name} <BuildSummaryReportFig>`.

OsvvmLibraries_RunDemoTests Build Summary Report

Build	OsvvmLibraries_RunDemoTests
Status	PASSED
PASSED	4
FAILED	0
SKIPPED	0
Elapsed Time (h:m:s)	0:00:13
Elapsed Time (seconds)	12.580
Date	2022-03-29T10:44-0700
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	3.662
AxiStream	PASSED	1	0	0	0 / 0	0	3.411
Uart	PASSED	2	0	0	0 / 0	0	5.478

▾ **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	2.265

▾ **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	0.990

▾ **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	1.007
TbUart_SendGet2	PASSED	22 / 26	0	0 / 0	-	0	0.937

Build Summary Report

To run tests in OSVVM, you call build to initiate a sequence of commands. This is much like source (or do) except OSVVM's build (and include) track where the script is running from so user level scripts do not need to.

The first thing you find in the Build Summary Report is the overall status of the test. See `Figure {number} {name} <BuildSummaryFig>`.

OsvvmLibraries_RunDemoTests Build Summary Report

Build	OsvvmLibraries_RunDemoTests
Status	PASSED
PASSED	4
FAILED	0
SKIPPED	0
Elapsed Time (h:m:s)	0:00:13
Elapsed Time (seconds)	12.571
Date	2022-03-28T09:01-0700
Simulator	RivieraPRO
Version	RivieraPRO-2021.10.114.8313
OSVVM YAML Version	1.0

Build Summary Status

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. `Figure {number} {name} <TestSuiteSummaryFig>` 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	3.576
AxiStream	PASSED	1	0	0	0 / 0	0	3.309
Uart	PASSED	2	0	0	0 / 0	0	5.577

Test Suite Summary

Note that any place in the report there is a triangle preceding text (such as the one above), pressing on the triangle will rotate it and hide the information. Pressing it again will reveal the hidden information.

The final set of tables in the Build Summary Report is the Test Case Summary. See `Figure {number} {name} <TestCaseSummaryFig>`. This table contains pass/fail information for each test case in the test suite.

▼ 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	2.325

▼ 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	0.939

▼ 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	1.028
TbUart_SendGet2	PASSED	22 / 26	0	0 / 0	-	0	1.042

Test Case Summary

1.3.2 Explore the Test Case Detailed Report

In the Test Case Summary for Axi4Full, click on TbAxi4_MemoryReadWriteDemo1. This will load the Test Case Detailed Report.

The first item in the Test Case Detailed Report is an HTML table of the available reports. See `Figure {number} {name} <TestCaseDetailedReportFig>`. The table for TbAxi4_MemoryReadWriteDemo1, shown 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.

TbAxi4_MemoryReadWriteDemo1 Test Case Detailed Report

Available Reports
Alert Report
Functional Coverage Report(s)
ScoreboardPkg_slv Report(s)
Link to Simulation Results
./results/TbAxi4_MemoryReadWriteDemo1.txt

Test Case Detailed Report

The next item in the Test Case Detailed Report is the Alert Report. See `Figure {number} {name} <AlertReportFig>`.

TbAxi4_MemoryReadWriteDemo1 Alert Report

▼ TbAxi4_MemoryReadWriteDemo1 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_MemoryReadWriteDemo1 Alert Results

Name	Status	Checks		Total Errors	Alert Counts			Requirements		Disabled Alert Counts		
		Passed	Total		Failures	Errors	Warnings	Passed	Checked	Failures	Errors	Warnings
TbAxi4_MemoryReadWriteDemo1	PASSED	334	334	0	0	0	0	0	0	0	0	0
Default	PASSED	20	20	0	0	0	0	0	0	0	0	0
OSVVM	PASSED	0	0	0	0	0	0	0	0	0	0	0
:tbaxi4memory:memory_1::memory	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
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
memory_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
Data Check	PASSED	0	0	0	0	0	0	0	0	0	0	0
WriteAddressFIFO	PASSED	0	0	0	0	0	0	0	0	0	0	0
WriteDataFifo	PASSED	0	0	0	0	0	0	0	0	0	0	0
WriteResponseFifo	PASSED	0	0	0	0	0	0	0	0	0	0	0
ReadAddressFifo	PASSED	0	0	0	0	0	0	0	0	0	0	0
ReadDataFifo	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
Protocol Error	PASSED	0	0	0	0	0	0	0	0	0	0	0
Data Check	PASSED	16	16	0	0	0	0	0	0	0	0	0
No response	PASSED	0	0	0	0	0	0	0	0	0	0	0
WriteResponse Scoreboard	PASSED	40	40	0	0	0	0	0	0	0	0	0

Alert 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.

If the test case collected functional coverage, then the next item in the report is the Functional Coverage Report. See `Figure below`.

TbAxi4_MemoryReadWriteDemo1 Coverage Report

Total Coverage: 43.75

▶ **Cov1 Coverage Model** **Coverage: 37.5**
▼ **Cov2 Coverage Model** **Coverage: 37.5**

▶ Cov2 Coverage Settings
▼ Cov2 Coverage Bins

Name	Type	Bin1	Count	AtLeast	Percent Coverage
	COUNT	0 to 0	0	1	0.0
	COUNT	1 to 1	1	1	100.0
	COUNT	2 to 2	1	1	100.0
	COUNT	3 to 3	0	1	0.0
	COUNT	4 to 4	0	1	0.0
	COUNT	5 to 5	1	1	100.0
	COUNT	6 to 6	0	1	0.0
	COUNT	7 to 7	1	1	100.0
	COUNT	32 to 32	1	1	100.0
	COUNT	33 to 33	1	1	100.0
	COUNT	34 to 34	1	1	100.0
	COUNT	35 to 35	0	1	0.0
	COUNT	36 to 36	1	1	100.0
	COUNT	37 to 37	0	1	0.0
	COUNT	38 to 38	1	1	100.0
	COUNT	39 to 39	0	1	0.0
	COUNT	64 to 64	0	1	0.0
	COUNT	65 to 65	1	1	100.0
	COUNT	66 to 66	0	1	0.0
	COUNT	67 to 67	0	1	0.0
	COUNT	68 to 68	0	1	0.0
	COUNT	69 to 69	0	1	0.0
	COUNT	70 to 70	1	1	100.0

Functional Coverage Report

If the test case had scoreboards, then the next item in the report is the Scoreboard Report. See `Figure below`.

TbAxi4_MemoryReadWriteDemo1 Scoreboard Report

▼ WriteAddressFIFO Scoreboard

Name	WriteAddressFIFO
ItemCount	40
ErrorCount	0
ItemsChecked	0
ItemsPopped	40
ItemsDropped	0

▼ WriteDataFifo Scoreboard

Name	WriteDataFifo
ItemCount	150
ErrorCount	0
ItemsChecked	0
ItemsPopped	150
ItemsDropped	0

Scoreboard Report

2 Structured Testbench Framework

2.1 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.

2.1.1 Going Further

Read the following documents for more information on OSVVM's Structured Testbench Framework. You will find these in OsvvmLibraries/Documentation.

Document	User Guide
Framework	OSVVM_structured_testbench_framework.pdf
VC Developer's Guide	OSVVM_verification_component_developers_guide.pdf
Test Writers UG	OSVVM_test_writers_user_guide.pdf
Address Bus MIT UG	Address_Bus_Model_Independent_Transactions_user_guide.pdf
Stream MIT UG	Stream_Model_Independent_Transactions_user_guide.pdf

3 VHDL Utility Library

3.1 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.

3.1.1 Going Further

Read the following documents for more information on OSVVM's VHDL Utility Library. You will find these in OsvvmLibraries/Documentation.

Document	User Guide	Quick Reference
Test Writer's UG	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

4 Scripting Library

4.1 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).

4.1.1 Going Further

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

OSVVM Script User Guide	OsvvmLibraries/Documentation/Script_user_guide.pdf

5 OSVVM Verification Component Library

5.1 OSVVM Verification Component Library

OSVVM's growing verification component library is tabulated below.

Verification Component(s)	User Guide	Source	Repository
Axi4 Full (Manager, Memory, and Subordinate) VCs	Axi4_VC_user_guide.pdf	OsvvmLibraries/AXI4/Axi4	AXI4
Axi4 Lite (Manager, Memory, and Subordinate) VCs	Axi4_VC_user_guide.pdf	OsvvmLibraries/AXI4/Axi4Lite	AXI4
AxiStream Transmitter and Receiver VCs	AxiStream_user_guide.pdf	OsvvmLibraries/AXI4/AxiStream	AXI4
UART Transmitter and Receiver VCs	None	OsvvmLibraries/UART	UART
DpRam behavioral model and DpRam controller	None	OsvvmLibraries/DpRam	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.