

SUMMARY of OSVVM

- The OSVVM verification framework has a structure similar to SystemVerilog, including verification components, a test sequencer, and separate test case architectures.
- The implementation of the OSVVM verification framework brings together the DUT, verification components, and test sequencer using structural code similar to RTL.
- In summary, OSVVM implementation combines DUT, verification components, and test sequencer with RTL-like structural code.
- **In OSVVM, the transaction API is implemented as VHDL procedures that handshake with the verification component using the transaction record and retrieve the results.**
- OSVVM's easy approach defines common transaction interfaces and APIs for interfaces that perform similar transactions using model independent transactions (MIT).
- The benefits of OSVVM MIT include accelerating the development of verification components and test cases, supporting co-simulation, simplifying documentation, and providing more information in user guides.
- OSVVM test scenario, send, get and check transactions and affirmations (checks) use for test, affirmations show pass/fail and count errors.
- OSVVM alerts, protocol checks easy do, error messages give, errors count, and different levels (FAILURE, ERROR, WARNING) alerts create.
-

Logs Simplify Debug SynthWorks

- Logs are conditional printing (messaging)

```
Log(TbID, "Sequence 1 Starting", ALWAYS) ;
. . .
Log(TbID, "Test Last Failed Here", DEBUG) ; -- Disabled
```

- Log Levels: ALWAYS (default), DEBUG, INFO, FINAL, PASSED
- Logs only print when enabled

- Controls: Enable/Disable

```
SetLogEnable(DEBUG, FALSE) ; -- Disable Alerts
```

- Log output for above

```
%% 2200 ns Log ALWAYS In TB, Sequence 1 Starting
```

- Message with level DEBUG does not print since it is disabled

24

- OSVVM test finalization, test finish check, timeout check, reports create, affirmations check, and test pass confirm.
- FIFO's directed and constrained random test compare, and realistic stimulus create and similar items wide variety ideal benefits emphasize.
- It is difficult to synchronize the randomization logic of these two processes.

Test Case Report

SynthWorks

TbAx14_MemoryReadWriteDemo1 Test Case Detailed Report

Available Reports

- [Alert Report](#)
- [Functional Coverage Report\(s\)](#)
- [Scoreboard\(s\) Report\(s\)](#)
- [Link to Simulation Results](#)
- [TbAx14_MemoryReadWriteDemo1.net](#)
- [Overview: RunAllTestsWithCoverage Build Summary](#)

TbAx14_MemoryReadWriteDemo1 Alert Report

▶ TbAx14_MemoryReadWriteDemo1 Alert Settings

▶ TbAx14_MemoryReadWriteDemo1 Alert Results

TbAx14_MemoryReadWriteDemo1 Coverage Report

Total Coverage: 43.75

▶ Cov1 Coverage Model

▶ Cov2 Coverage Model

▶ Cov1a Coverage Model

▶ Cov2b Coverage Model

Coverage: 37.5

Coverage: 37.5

Coverage: 50.0

Coverage: 50.0

TbAx14_MemoryReadWriteDemo1 Scoreboard Report for Scoreboard_slv

Name	ParentName	ItemCount	ErrorCount	ItemsChecked	ItemsPopped	ItemsDropped	FailCount
WriteAddressFifo	memory_1	40	0	0	40	0	0
WriteDataFifo	memory_1	150	0	0	150	0	0
WriteResponseFifo	memory_1	40	0	0	40	0	0
ReadAddressFifo	memory_1	40	0	0	40	0	0
ReadDataFifo	memory_1	150	0	0	150	0	0
WriteResponse Scoreboard	manager_1	40	0	40	40	0	0
ReadResponse Scoreboard	manager_1	150	0	150	150	0	0
WriteAddressFifo	manager_1	40	0	0	40	0	0
WriteDataFifo	manager_1	150	0	0	150	0	0
ReadAddressFifo	manager_1	40	0	0	40	0	0

Links

- Alert Report
- Functional Coverage Report
- Scoreboard Reports
- Simulation Results
- Test Case Transcript
- Link to Build Summary

Alert Report

- Settings (hidden)
- Results (hidden)

Functional Coverage Report

Report for each FC model in testbench (each hidden)

Scoreboard Report

One Table for each Scoreboard type.
One row in table for each scoreboard.

- OSVVM HTML'ized simulation transcript, It makes debugging easier by allowing you to review things quickly.
- In many ways, OSVVM offers a more sensible approach to testing.

SUMMARY of UVVM

- UVVM is an open-source VHDL verification methodology.
- UVVM, simple data communication verify for BFM's and utility library use UART and SBI over data send, control and wait show, error situations emphasize, and wide interface support give.
- In UVVM, VVCs make interface verification easy, test scenarios modular and reusable do.
- In UVVM, BFM to VVC transition, test sequencer direct DUT instead VVC with communication make, test scenarios more modular and expandable do, delay insert, command queuing, completion detection like extra features give.
-

VVC: Easy to extend (1)

- Easy to add local sequencers
- Easy to add checkers/monitors/etc

Interpreter

- Is command for me?
- Is it to be queued?
- If not:
Case on what to do

***_VVC**

Command Queue

Executor

- Fetch from queue
- Case on what to do
- Call relevant BFM(s) & Execute transaction

Bit-rate checker

Frame-rate checker

Gap checker

VVC: Easy to extend (2)

- Easy to handle split transactions
- Easy to handle out of order execution

Interpreter

- Is command for me?
- Is it to be queued?
- If not:
Case on what to do

***_VVC**

Command Queue

Executor

- Fetch from queue
- Case on what to do
- Call relevant BFM(s) & Execute transaction

Bit-rate checker

Frame-rate checker

Gap checker

Queue

Response-Executor

- **UVVM VVCs, multiple interfaces simultaneously manage, reusable and expandable, single sequencer from controlled can, cycle corner cases target, split transactions and out of order protocols easy handle, broadcast and multicast use, better overview and maintenance give.**
- UVVM generic scoreboard, VVC based test environment expected and actual data compare, statistics keep, queue manage, generic data types support, configuration records and counting features have.
- UVVM enhanced randomization, readability increase, maintenance easy make, and better coding invest big gain give.
- UVVM functional coverage, bins use by coverage target reach make and transition coverage also support.