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 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

- 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.

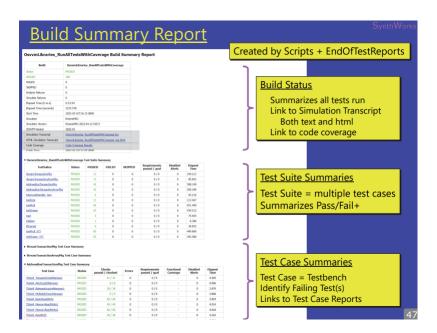
•

- OSVVM scoreboards use, sender (TxProc) change even, receiver (RxProc) process same keep, control easy do.
- OSVVM functional coverage, test plan track, random tests what do show, different coverage types with relations track, and code coverage missing parts complete.

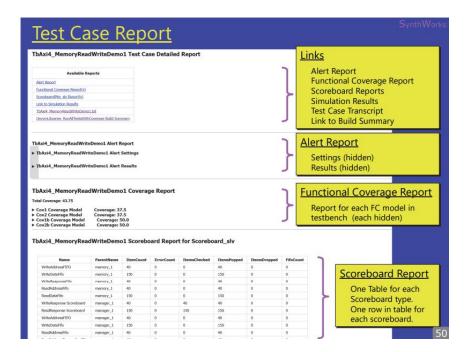
Writing Functional Coverage architecture CR_1 of TestCtrl is Coverage Object signal RxCov : CoverageIdType ; RxProc : process Construct the Data Structure begin RxCov <= NewID("RxCov", TB_ID); Define coverage model wait for 0 ns ; AddBins(RxCov, GenBin(1) -- Normal AddBins(RxCov, GenBin(3) Parity Error AddBins(RxCov, GenBin(5)); -- Stop Error AddBins(RxCov, GenBin(7));
AddBins(RxCov, GenBin(9, 15, 1)); Parity + Stop for I in 1 to 10000 loop Get(RxRec, RxD, RxOp); Check(SB,(RxD, RxOp)); ICover(RxCov, to_integer(RxOp)); end loop ; Functional Coverage with OSVVM is as simple and concise as language syntax

Before OSVVM scripts;

- Issues
 Need help (TCL code) to find the source directory
 Simulator Specific
 Simulator API repeats the same information on many calls
- OSVVM unmatched test reports create, test completion message, summary reports, requirements summaries, test case reports, HTML'ized simulation transcript, JUnit report and more give.
- Build finish when one line mini report create, errors have first see place this.



•



- 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.

Gap checker

- UVVM, simple data communication verify for BFMs 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
- 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) VVC: Easy to extend (2) - Easy to handle split transactions - Easy to add local sequencers - Easy to add checkers/monitors/etc - Easy to handle out of order execution *_VVC *_VVC Interpreter Executor Interpreter Executor Is command for me? - Is command for me? - Fetch from queue Fetch from queue - Is it to be queued? Is it to be queued? Case on what to do Case on what to do Command Command Call relevant BFM(s) If not: Case on what to do Call relevant BFM(s) Queue Queue Case on what to do & Execute transaction & Execute transaction Bit-rate checker Bit-rate checker Queue Frame-rate checker Frame-rate checker

Response-Executor

Gap checker

•

- 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.