



OVM Adopter Class

3 days



Open Verification Methodology (OVM) is a non-proprietary functional verification methodology based on SystemVerilog. The source code and documentation are freely available under an open-source Apache license. OVM offers a complete framework for the creation of sophisticated functional verification environments in SystemVerilog, and encourages the development and deployment of re-usable verification components.

It has comprehensive support for constrained random stimulus generation, including structured sequence generation, and for transaction-level modelling. OVM testbenches also support functional coverage collection and assertions. OVM exploits the object-oriented programming (or "class-based") features of SystemVerilog. The open structure, extensive automation, and standard transaction-level interfaces of OVM make it suitable for building functional verification environments ranging from simple block-level tests to the most complex coverage-driven testbenches.

Delegates for this course must start with a working knowledge of SystemVerilog. The course takes delegates through to full SystemVerilog verification project readiness by focussing on the verification principles and the in-depth practical application of OVM 2.0 using commercial verification tools such as Mentor Graphics Questa™ Sim and Cadence Incisive® Enterprise Simulator.

Workshops comprise approximately 50% of class time, and are based around carefully designed exercises to reinforce and challenge the extent of learning. During the hands-on workshops, delegates will build a complete OVM verification environment for a small example system.

Who should attend?

- Verification engineers who wish to deploy complex SystemVerilog verification environments using OVM 2.0
- Design engineers who wish to make full use of SystemVerilog's verification capabilities for test bench development using OVM 2.0

What will you learn?

- The principles of effective functional verification using SystemVerilog
- The standard structure of OVM components and environments
- How to use the OVM kit (classes, macros, documentation and examples) in constructing your own verification environments
- Making good use of OVM features for configuration, stimulus generation, reporting and diagnostics
- How to build complete, powerful, reusable class-based OVM verification components and environments

Pre-requisites

A basic working knowledge of SystemVerilog is essential. For engineers with no SystemVerilog knowledge or experience the Doulos **Comprehensive SystemVerilog** course or equivalent is an essential precursor. For onsite courses, **Modular SystemVerilog** precursor training can be tailored to your team's profile. Contact Doulos to discuss options that suit your needs

Training materials

Doulos training materials are renowned for being the most comprehensive and user friendly available. Their style, content and coverage is unique in the EDA training world, and has made them sought after resources in their own right. Fees include

- Fully indexed class notes creating a complete reference manual
- Lab files comprising the complete SystemVerilog/OVM source files and scripts
- Doulos OVM Golden Reference Guide

Structure and Content

Introduction to OVM

Course structure • motivation • principles of coverage-driven verification • benefits • transaction level modelling • overview of AVM and URM • the OVM kit • test bench organisation • OVM class summary • overview of key OVM features

Getting started with OVM

Test bench structure • ovm_env and ovm_test • field automation macros • basic reporting • transaction classes • generating a randomized sequence • driver class • linking to the DUT • virtual interfaces • running a test • Lab - a simple test bench

Monitors and Reporting

Creating a monitor • the OVM printer • reports and actions • configuring the OVM report handler • ovm_analysis_port / export • connecting analysis ports and exports • ovm_subscriber • tlm_analysis_fifo • Lab - Monitor with analysis ports

Checkers and Scoreboards

The role of assertions • structural versus protocol assertions • reference models • monitor operation • sampling signal values • scoreboards and the ovm_scoreboard class • OVM built-in comparators • specifying match rules • redirecting reports • log files • Lab - implementing a checker

Functional Coverage

Separating data gathering from coverage analysis • property-based coverage • property variables and actions • covergroup and coverpoint • cross coverage • binning • analysis subscriber • coverage on internal states of DUT • Lab - creating a coverage collector

Random Stimulus Generation

Constrained random stimulus • packing OVM class fields • emulating ROM with instruction driver • creating sequences manually • controlling the constraint solver • serial I/O example • overriding generated sequence items • Lab - constraints and random stimulus

Configuring the Testbench

Using component names to represent hierarchy • locating and identifying component instances by name • using the OVM factory • registering fields with factory • overriding factory defaults • using the factory with parameterized components • setting and getting configuration details • virtual interface wrappers • configuring multiple tests • configuration with command-line arguments • stopping a test • Lab - testbench configuration and overriding the factory

Agent Architecture

"Agent" architecture and its relationship with other verification methodologies • class monitors and drivers • standard agent architecture • ovm_agent • sequence library and default OVM sequences • communication between sequencer and driver • connecting and configuring agent • Lab - building a simple agent with sequencer

Sequences

Sequencer and sequences - the ovm_sequence class • creating custom sequences • sequence macros and the body task • sequence phases • configuring sequences • complex sequences • introduction to virtual sequences virtual sequencers • Lab - creating and extending user-defined sequences

Hierarchical Testbench Components

TLM interfaces and ports • implementing an export • using tlm_fifo analysis ports • coverage-based test controllers • error injection • child process control

Supplementary Subjects

More on Sequences

Using callbacks for sequence interaction • getting response from sequence driver • grabbing control of sequences • concurrent sequence control • multi-layer sequences using inheritance • multi-layer virtual sequences

Class-OOP Primer/Review

Object-Oriented Programming • class • object • method • constructor • extends • inheritance • overriding • virtual method • up-cast • parameterised class

Doulos acknowledges all trademarks and registered trademarks as the property of their respective owners.