

# OVM Adopter Class

## Advanced Level – 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 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
- Design engineers who wish to make full use of SystemVerilog's verification capabilities for testbench development using OVM

### 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

For further information contact your local Doulos [Sales Office](#).



# OVM Adopter Class

## Advanced Level – 3 days

essential precursor. For in-house courses, [Modular SystemVerilog](#) precursor training can be tailored to your team's profile. Contact Doulos to discuss options that suit your needs.

### Course materials

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

- Fully indexed course notes creating a complete reference manual
- Lab files comprising the complete SystemVerilog/OVM source files and scripts

### Structure and content

#### Introduction to OVM

Course structure • motivation • principles • benefits • transaction level modelling • overview of AVMM and URM • the OVM kit, libraries and examples • graphical notation • test bench organisation • OVM class summary • naming conventions

#### Verification Methodology

Assertion based verification • functional coverage • structural coverage • constrained random verification • coverage-driven verification • the verification process and verification planning • coverage models • creating reusable verification environments

#### Getting Started with OVM

Test bench structure • ovm\_env and ovm\_test • field automation macros • basic reporting • transaction classes • generating random stimulus • using tlm\_fifo channels • driver class • linking to the DUT • virtual interfaces • running a test • Lab – a simple test bench

#### OVM Communication Mechanisms

Connecting the component hierarchy • ports and exports • establishing the connections • blocking and non-blocking tlm interfaces • implementing an export • ovm\_component and ovm\_threaded\_component • minimal verification environment • simulation phases • Lab – TLM communication

#### Analysis Ports

ovm\_analysis\_port / export • connecting analysis ports and exports • ovm\_subscriber • tlm\_analysis\_fifo • Lab – Analysis ports and components

#### Checkers and Scoreboards

The role of assertions • structural versus protocol assertions • reference models • monitors • 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

For further information contact your local Doulos [Sales Office](#).



# OVM Adopter Class

## Advanced Level – 3 days

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 • constraining ovm\_random\_stimulus • controlling the constraint solver • serial I/O example • overriding generate\_stimulus • 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 • 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 • OVM sequencer • 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 • manual control of sequences • complex sequences • introduction to virtual sequences virtual sequencers • Lab – creating and extending user-defined sequences

### Verification Environment Interactions

Using tlm\_fifo analysis ports • customising the report formatter • report severity and actions • 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

### Classes – OOP Primer/Review

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

## Related courses

- Comprehensive SystemVerilog
- Modular SystemVerilog (in-house delivery only)

For further information contact your local Doulos [Sales Office](#).

