UVM Adopter Class

Duration: 3 Days

Course Overview:
Course description with information on the course – marketing oriented.
The Universal Verification Methodology (UVM) is a standard functional verification methodology for SystemVerilog, controlled by Accellera, and endorsed and supported by all major SystemVerilog simulator vendors. The source code and documentation are freely available under an open-source Apache license. UVM 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.
UVM has comprehensive support for constrained random stimulus generation, including structured sequence generation, and for transaction-level modeling. UVM testbenches also support functional coverage collection and assertions. UVM exploits the object-oriented programming (or "class-based") features of SystemVerilog. The open structure, extensive automation, and standard transaction-level interfaces of UVM 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 detailed knowledge of building class-based verification environments using SystemVerilog. The course leads delegates through to full verification project readiness by focusing on the in-depth practical application of UVM using commercial verification tools such as Cadence Incisive® Enterprise Simulator, Mentor Graphics Questa™Sim, and Synopsys® VCS®. 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 UVM verification environment for a small example system.

Who should attend?
Verification engineers who wish to deploy complex SystemVerilog verification environments using UVM

Design engineers who wish to make full use of SystemVerilog's verification capabilities for test bench development using UVM

Prerequisites:
A detailed knowledge of how to build a class-based SystemVerilog verification environment is essential. For engineers with no class-based SystemVerilog knowledge or experience the Doulos Comprehensive SystemVerilog or SystemVerilog for Verification Specialists courses provide an appropriate preparation. For onsite 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 are unique in the HDL training world, and have made them sought after resources in their own right. The materials include:
Fully indexed class notes creating a complete reference manual
Lab files comprising the complete SystemVerilog/UVM source files and scripts

---

Course Content:

1. Introduction to UVM
- Course structure
- motivation
- principles of coverage-driven verification
- benefits
- transaction level modelling
- the UVM kit test bench organisation
- UVM class summary
- overview of key UVM features

2. Getting Started with UVM
- Test bench structure
- uvm_env and uvm_test
- field automation macros
- basic reporting
- transaction classes
- generating a randomized sequence
- driver class linking to the DUT
- virtual interfaces
- running a test

Continued …

UVM Adopter Class

Duration: 3 Days

Course Content:

… Continued

- Lab - a simple test bench

3. Monitors and Reporting
- Creating a monitor
- the UVM printer
- reports and actions
- configuring the UVM report handler
- the UVM report catcher
- TLM ports, exports, and binding
- analysis ports uvm_subscriber
- tlm_analysis_fifo
- Lab - Monitor with analysis ports

4. Checkers and Scoreboards
- The role of assertions
- structural versus protocol assertions
- reference models
- monitor operation
- sampling signal values
- scoreboards and the uvm_scoreboard class
- UVM built-in comparators
- field macros and their flags
- overriding the compare method
- redirecting reports
- log files
- Lab - implementing a checker

5. 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

6. Random Stimulus Generation
- Constrained random stimulus
- packing UVM class fields
- the sequencer-driver interface
- controlling the constraint solver
- serial I/O example
- overriding generated sequence items
- the uvm_do sequence macro family
- Lab - constraints and random stimulus

7. Configuring the Testbench
- Using component names to represent hierarchy
- locating and identifying component instances by name
- using the UVM factory registering fields with factory
- overriding factory defaults
- using the factory with parameterized components
- the resource database and configuration database
- virtual interface wrappers
- configuring multiple tests
- configuration with command-line arguments
- stopping a test
- Lab - testbench configuration and overriding the factory

8. Agent Architecture
- "Agent" architecture and its relationship with other verification methodologies
- class monitors and drivers
- standard agent architecture
- uvm_agent
- sequence library and default UVM sequences
- communication between sequencer and driver
- connecting and configuring agent
- end-of-test mechanisms
- objections
- Lab - configuring an agent

9. Sequences
- The uvm_sequence class
- sequence phases
- sequence callbacks
- starting sequences
- nested sequences
- sequence control knobs
- virtual sequences and sequencers
- Lab - creating nested and virtual sequences

10. More on Sequences
- Overriding sequences
- getting response from sequence driver
- interleaved sequences
- sequence priority and arbitration
- grabbing control of sequences
- sequence layering
- Lab - overriding sequences, grabbing sequences, and using sequence priority

Optional Topics

11. Appendix - Callbacks
- Using callbacks as an alternative to the factory to customize behavior

12. Appendix - Register Layer
- Register layer architecture and features
- front door and back door access
- mirroring and updating
- defining register fields, registers, and register blocks
- address maps
- register adapters
- integrating registers into the environment
- register sequences
- built-in register test sequences
- Lab - defining and integrating a register model and creating a register test

Continued …

UVM Adopter Class

Duration: 3 Days

Course Content:

… Continued

13. Appendix - Advanced Register Topics

- Indirect register access
- front door sequences
- register predictor
- back door access
- using HDL paths
- memory access through the address map
- register coverage models
- register model generators
- RALGEN
- RGM

14. Summary