UVM Adopter Class

Course Description:
The Universal Verification Methodology (UVM) is a standard functional verification methodology for SystemVerilog, controlled by Accellera Systems Initiative (ASI), 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 modelling. 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. The UVM register classes provide a standard mechanism to automatically set and monitor every register within the device under test
This training course teaches best practice as documented in the Easier UVM Coding Guidelines from Doulos. Many of the coding examples and exercise used in this training course are compliant with the Easier UVM Coding Guidelines. Delegates may take advantage of the Easier UVM Code Generator to become more productive with UVM after attending the course. Easier UVM itself conforms fully to the UVM standard (which is currently at version 1.2).

Training Duration: 4 days

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.

What will you learn? The course is structured into four distinct sections.
- The principles of effective functional verification using UVM
- The standard structure of UVM components and environments
- How to use the UVM base class library in constructing your own verification environments
- Making good use of UVM features for configuration, stimulus generation, reporting and diagnostics
- How to build complete, powerful, reusable class-based UVM verification components and environments
- How to use the UVM register layer

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 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 class notes creating a complete reference manual
- Lab files comprising the complete SystemVerilog/UVM source files and scripts
- a UVM Golden Reference Guide.

Structure and Content

1. Introduction to UVM
- What is UVM?
- Why UVM?
- The UVM Family Tree
- Versions of UVM
- Constrained Random Verification
- Tests versus Testbench
- Configurable Verification Components
- UVM Class Hierarchy
- Simulation Phases
- Customization - The Factory
- Transaction Level Modelling Required / Provided Interfaces
- Push vs Pull Connections
- Layered Sequential Stimulus Virtual Sequences Hierarchical Configuration
- Doulos - Easier UVM Subject Areas for the UVM Adopter

Continued ...

UVM Adopter Class (Cont.)

Course Content:

... Continued

2. Getting Started with UVM
- Simple Testbench Architecture
- Sequence Item Class
- Methods of a Transaction
- Overriding do_compare
- Alternative - Field Macros
- Sequence Class
- Alternatives: Do Macros or Function Calls
- Objections
- Driver
- Driver run_phase Task
- Simple Testbench Architecture
- DUT Interface
- Environment - Constructor
- Build Phase
- Connect Phase
- The Config Database
- Environment - run_phase
- Test Class - build_phase
- Top-Level Module
- uvm_top
- Running a Test
- Exercise - Stimulus Generator and Driver

3. Monitors and Reporting
- Driver versus Monitor
- Monitor Class
- Monitor Run Task
- Reporting
- Report Macros
- Controlling Verbosity
- Setting Actions
- Severity and Actions
- Scope of Actions
- Writing to a Log File
- The Reporting Set Methods
- The Report Catcher
- Registering Report Catchers

4. Transaction-Level Modeling
- TLM Connections
- Blocking put, get, and peek
- Blocking versus Non-Blocking Methods
- TLM Ports, Exports, and Imps
- Child Port Connection
- Child Export Connection
- Implementing an Export
- Monitor with Devolved Tasks
- Analysis Ports
- Monitor with Analysis Port
- Subscriber
- Registering Subscribers
- uvm_tlm_analysis_fifo
- Subscriber with uvm_tlm_analysis_fifo
- Subscriber with FIFO Implementation
- Debugging Connections
- Exercise - Monitors and Subscribers

5. Checkers and Scoreboards
- Example Design
- Separating Test from Test Harness
- Interfaces
- Parameters & Interfaces
- Modports
- Bus Properties
- Adding ISS Reference Model
- Sampling Bus in Interface
- Bus Monitor Class
- Detecting Bus Sequence
- Bus Monitor Run Task
- Verification Environment
- Scoreboard
- Comparator Input Requirements
- Field Macros
- Field Macro Flags
- Overriding do_compare
- Field Macros and Overridden Methods
- Using Policy Objects
- print and sprint
- Flags, Methods and Printers
- Overriding do_pack
- Packing Bits and Bytes
- Redirecting Reports
- Exercise - Checkers

6. Functional Coverage
- How Much Coverage?
- Coverage Collection
- Detecting States in Bus Monitor
- Property-Based Coverage
- Transferring Coverage Data to Monitor
- Controlling Cross Coverage
- Coverage Collector Class
- Subscriber Classes
- Plumbing the Coverage Collector Class
- TLM imp Macros
- Coverage on Internal States of DUT
- Exercise - Coverage collection

7. Random Stimulus Generation
- Random Stimulus
- The Example
- Instruction Class
- Instruction Sequence and Sequencer
- Starting a Sequence
- Instruction Driver
- Sequencer-Driver Interface
- Constraining the Instruction
- The dist Constraint Operator
- Run-Time Phasing Stimulus Phases
- Sequence Body Method
- Sequence Macros
- Using the Sequence Macros
- Exercise - Constrained random

Continued ...

UVM Adopter Class (Cont.)

Course Content:

... Continued

8. Factory and Configuration
- UVM Component Names
- Searching for Component Instances
- Globs and Regular Expressions
- Iterating the Children
- Factory Overrides "By Type"
- Parameterized Objects and Components
- Factory Overrides "By Name"
- Constrained Random Generation
- The Configuration Database
- Verilog / VHDL versus UVM
- Setting Configuration Table Entries
- Getting Configuration Table Entries
- Multiple Configuration Tables
- Debugging Configuration Settings
- Configuration Objects
- Configuring the Component Hierarchy
- Working with Multiple Tests
- UVM Command Line Processor
- Standard Command Line Arguments
- Exercise - Configuration

9. Agent Architecture
- UVM Agent
- Agent Build
- Connecting Agent Components
- Configuring an Agent
- Using the Built-in "is_active" Field
- Overriding get_is_active
- Exercise - Agents

10. Objections
- Objection Mechanism
- Objections from a Component
- Automatic Objections
- Objections from a Sequence
- Raise/Drop Objections Per-Cycle
- Objection Drain Time versus Time-out
- Extending a Phase
- Propagating Objections
- set_propagate_mode
- Stopping a Test - OVM Legacy

11. Sequences
- Class uvm_sequence
- Sequence/Driver Interaction
- Overriding the Callbacks
- Sequence Callbacks
- Starting a Sequence Manually
- Starting from a Parent Sequence
- Starting a Sequence Automatically
- Sequences-of-Sequences
- Sequence Control Knobs
- Nested Sequences
- Finding Sequencer from Sequence
- `uvm_declare_p_sequencer
- Configuration of Control Knobs
- Transaction Recording
- Marking the Start & End of Transactions
- Reports within Sequences
- Virtual Sequences and Sequencers
- Virtual Sequencer
- Virtual Sequence on Sequencer
- Standalone Virtual Sequence
- Test with null Sequencer
- Exercise - Sequences and virtual sequences

12. Layered Sequences and Agents
- Multi-Layer Sequences
- Extended Sequences
- Low-Level Sequence
- High-Level Frame Transaction
- High-Level Sequence
- Layered Sequencers
- Low-Level Sequencer
- High-Level Sequence
- Layering Sequence
- Starting with a Virtual Sequence
- Direct Connection to the Sequencer
- Alternative Layering Sequence
- Request and Response
- The Driver Response
- Pipelined Responses in the Driver
- Pipelined Responses in the Sequence
- Matching Requests and Responses
- Reponses with Layered Sequencers
- Layering Agents
- Multiple Agents / Sequencer Stacks
- Driver calls try_next_item

13. Events and Barriers
- uvm_event
- Event Action
- Passing Data with Event Trigger
- Event Pools
- uvm_event_pool
- UVM Event Callbacks
- Event Callback Class
- Register Callbacks with Component
- uvm_barrier
- Setting Barrier

14. Advanced Sequencer Topics
- Interleaved Sequences
- Sequence Priority
- The Arbitration Queue
- Setting the Arbitration Algorithm
- Arbitration Algorithms and Priority
- User-Defined Arbitration Algorithm
- Irrelevant Sequences
- Virtual Sequences Revisited
- Sequencer Lock
- Lock versus Grab
- Push Sequencer and Driver
- Push Driver Implementation
- The UVM Sequence Library
- Sequence Library = Fancy Sequence
- Controlling Sequence Selection

Continued ...

UVM Adopter Class (Cont.)

Course Content:

... Continued

- Setting Properties with the Config DB
- Exercise - Advanced sequences

15. UVM Register Layer
- UVM Tests and DUT Registers
- UVM Register Layer
- Register Layer Architecture
- UVM Register Layer Features
- Register Layer Organisation
- Accessing Registers By Name
- Integrating the Register Model
- Frontdoor Register Access
- Backdoor Register Access
- Mirroring
- Code Example
- Updating
- Register Model
- An Example - Serial I/O
- Serial I/O Memory-Mapped Registers
- Unit-Level Register Model Integration
- Define Registers and Fields
- Configure the Register
- Create the Register Block
- Build the Registers
- Create the Address Map
- Register Adapter
- Provide a Driver Response
- Using Byte Enables
- Integration into the Environment
- Accessing the Register Model
- Register Test Case
- Random Register Testing
- Built-in Sequences
- Using Built-in Tests
- Exercise - Register Layer

16. Advanced Register Topics
- Indirect Register Access
- System-Level Register Model Integration
- Front Door Sequence
- Front Door Sequence Integration
- Responses Predictor
- Predictor Integration
- Direct Access Through Backdoor
- Adding the HDL Path
- HDL Backdoor Access
- Memory Access within an Address Map
- Adding Memory to an Address Map
- Coverage Models
- Instantiate Coverage Model during Build
- Enable/Disable Coverage
- Coverage in a Register Block
- Sample Method in a Register Block
- Enabling Coverage
- Special Register Behaviors
- Register Access Flow and Callbacks
- Example: Register Aliasing
- Exercise - Frontdoor sequence and predictor

17. Appendix - Callbacks and Heartbeat
- Callbacks
- Callback Base Class
- Insert Callbacks into Component
- Implement Specific Callbacks
- Add the Specific Callbacks
- uvm_heartbeat
- Adding Heartbeat to Driver
- Enabling Heartbeat from Test

18. Appendix - UVM Run-Time Phasing
- Motivation
- Background
- The Common Phases of UVM
- Phase Methods & Objects
- The UVM Run-Time Phases
- Default Synchronization
- Phase Method Synch
- Domains
- Unsynchronized Domains
- Explicit Synchronization
- Synchronized Phases
- User-Defined Phases
- Extended Schedule
- Add after_phase
- Add before_phase
- Add with_phase
- A Schedule from Scratch
- set_domain / define_domain
- Overriding define_domain
- Calling set_domain
- Start and End of each Phase
- phase_started
- phase_ready_to_end
- phase_ended
- Making Sequences Phase-Aware
- Reactive Stimulus
- Phase Jumping
- VIP Creation
- VIP Integration
- User-Defined Phases
- Phase Ordering
- Recommendations

19. Appendix - UVM 1.2
- New Features
- UVM 1.1d-to-1.2 Pitfalls
- Features Deprecated in UVM 1.2
- UVM_NO_DEPRECATED
- UVM_POST_VERSION_1_1