(Enhanced)
UVM Adopter class

Formation de 4 jours

Présentation

La Méthodologie Universelle de Vérification (**UVM**) est construite autour du langage **SystemVerilog** IEEE 1800™. Elle est issue de l'initiative *Accellera Systems Initiative* (ASI) comme une **convergence** des différentes méthodologies propriétaires proposées dans le passé, et elle est désormais supportée et endossée par tous les vendeurs et simulateurs SystemVerilog majeurs du marché (Synopsys, Mentor, Cadence, Aldec...).

Le code source d'UVM et sa documentation sont disponibles librement sous licence open-source Apache.
UVM exploite la partie « Orientée Objet » ou « *Class-based* » du langage SystemVerilog pour offrir un environnement complet (*framework*) de vérification et de déploiement de composants de vérification réutilisables.
UVM intègre le support de stimuli par génération aléatoire contrainte, la génération structurée de séquences, et la modélisation transactionnelle (TLM). Les Bancs de test UVM supportent la couverture fonctionnelle et les assertions. La structure ouverte, le support de l'automatisation, et les interfaces transactionnels standard d'UVM en font l'outil idéal pour construire des environnement de vérification fonctionnelle depuis des simples tests unitaires jusqu'aux plus complexes bancs pilotés dynamiquement par la couverture (*cover-driven*). Enfin, la prise en compte des classes « Register Layer » d'UVM justifie l'extension de la formation « Enhanced » à 4 jours. Ces classes fournissent un mécanisme standard pour actualiser et monitorer tous les registres du système depuis le banc de test, et sont désormais un atout significatif dans la plupart des systèmes.

Les Instructeurs ALSE sont aussi et surtout des Experts en Conception qui savent partager leur savoir-faire avec passion et sont particulièrement appréciés des participants.

Les exercices pratiques, qui occupent environ 50 % du temps de la formation, sont également des éléments clés du succès de ces formations, ils sont indispensables pour un réel apprentissage.
Ils assurent également un contrôle continu de l'acquisition des compétences.

Pour qui ?

NB : cette formation est l'une des plus complexes de notre portfolio et exige de maîtriser préalablement des connaissances importantes (voir plus bas).

Elle est destinée :

- Aux ingénieurs de Vérification qui doivent concevoir ou même seulement utiliser des environnements de vérification basés sur UVM.
- Aux ingénieurs de Conception qui veulent tirer parti de SystemVerilog et d'UVM pour construire des bancs de test puissants pour leurs composants complexes.

Objectifs pédagogiques

NB: Une connaissance préalable très approfondie de SystemVerilog et des Classes SV est indispensable.

- Découvrir les principes de la Librairie Méthodologiques de Vérification UVM pour construire des environnement de Vérification Fonctionnelle.
- La structure standard des composants et environnements UVM
- Comment utiliser le Kit UVM (classes, macros, documentation & exemples) pour construire son propre environnement de vérification.
- Bien utiliser UVM pour configurer, générer les stimuli, reporter et diagnostiquer.
- Savoir construire des composants et environnements UVM complets, puissants et réutilisables.

Qu'apprendrez vous ?

- Les principes de la Vérification Fonctionnelle efficace en SystemVerilog
- La structure standard des composants et environnements UVM
- Comment utiliser le Kit UVM (classes, macros, documentation & exemples) en construisant ses propres environnements de vérification
- Faire bon usage des caractéristiques d'UVM pour la configuration, la génération de stimuli, les diagnostics et le reporting
- Comment construire des composants et des environnements de vérification UVM complets, puissants, versatiles et ré-utilisables

Connaissances pré-requises

Une connaissance préalable complète du langage **SystemVerilog for Design et** de sa partie objet (**Class-based SystemVerilog Verification**) est **indispensable**.
Ces connaissances peuvent (doivent) être acquises par les formations préalables pré-citées.

Supports de cours

Les manuels de cours Doulos sont réputés pour être détaillés, précis et faciles d'utilisation. Leur style, leur contenu et leur exhaustivité sont uniques dans le monde de la formation HDL. Ils sont souvent utilisés ensuite comme référence.

Sont compris dans la formation :

- Le Classeur du cours théorique, avec Index.
  Il constitue un Manuel de Référence concis.
- Le Cahier des Exercices pratiques, qui permettent de mettre en œuvre les connaissances théoriques.
- Le « Doulos Golden Reference Guide » :
  Aide-mémoire UVM très exhaustif et pratique (syntaxe, sémantique, trucs et astuces). Il est destiné à devenir pendant longtemps le compagnon de votre clavier...

(Enhanced) UVM Adopter class
Programme (4 jours)

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

Getting Started with UVM

• Test bench structure • uvm_env and uvm_test • Transaction classes and methods
• Field automation macros • Generating a randomized sequence • Objections • Driver class
• Linking to the DUT • Virtual Interfaces • Config database • Running a test
Practical Exercise: A simple test bench

Monitors and Reporting

• Creating a monitor • Report methods and macros • Report actions and verbosity
• Configuring the UVM report handler • The UVM report catcher • TLM ports, exports, and binding
• Analysis ports • uvm_subscriber • uvm_tlm_analysis_fifo
Practical Exercise: 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 uvm_scoreboard class
• UVM built-in comparators • Field macros and their flags
• Overriding the print, compare and pack methods • Redirecting reports • Log files
Practical Exercise: 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 • TLM imp macros • Coverage on internal states of DUT
Practical Exercise: Creating a coverage collector

Random Stimulus Generation

• Constrained random stimulus • Packing UVM class fields • The sequencer-driver interface
• Controlling the constraint solver • Overriding generated sequence items • Sequence body task
• The uvm_do sequence macro family
Practical Exercise: Constraints and random stimulus

Configuring the Testbench

• Using component names to represent hierarchy • Locating and identifying component instances by name
• Using the UVM factory • Overriding factory defaults • Using the factory with parameterized components
• The resource database and configuration database • Multiple configuration tables
• Debugging configurations • Configuration objects • UVM command-line processor
• UVM command-line arguments
Practical Exercise: Test bench configuration and overriding the factory

Agent Architecture

• "Agent" architecture and its relationship with other verification methodologies
• Standard agent architecture • uvm_agent class • Connecting and configuring agent
• UVM objection mechanism • Objections in sequences • uvm_heartbeat • Enabling heartbeat from test
• OVM legacy methods
Practical Exercise: Configuring agents

Sequences

• The uvm_sequence class • Sequence phases • Sequence callbacks
• Manual and automatic starting of sequences • Nested sequences
• Accessing sequencer members from sequence • Sequence control knobs • Transaction recording
• Reports within sequences • Virtual sequences and sequencers
Practical Exercise: Creating nested and virtual sequences

Layered and Interactive Sequences

• High and low-level sequences • Layered sequence using inheritance • Layered sequencers with TLM ports • Starting high and low-level sequences • Layering with translation sequence • Sequence-driver interaction
• Pipelined responses in driver and sequence • Responses with layered sequencers
• Synchronizing multiple sequencer stacks • try_next_item

Advanced Sequences

• Overriding sequences • Interleaved sequences • Sequence priority and arbitration
• Sequencer arbitration queue • Built-in and user-defined arbitration algorithms • Irrelevant sequences
• Locking and grabbing control of sequences • Push sequencer and driver • UVM sequence library
Practical Exercise: Overriding sequences, grabbing sequences, and using sequence priority

UVM 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
• Driver response • Integrating registers into the environment • Register sequences • Built-in register test sequences
Practical Exercise: Create a simple register model

Advanced Register Topics

• Integration and indirect register access • Front door sequences • Register predictor • Back door access
• Using HDL paths • Memory access through the address map • Register coverage models
• Register behavior • Register callbacks • Register aliasing
Practical Exercise: Integrate a register model into a system-level testbench

Optional Topics

Appendix - Synchronization

• uvm_event • uvm_event_pool • Event callbacks • uvm_barrier

Appendix - Callbacks

• Using callbacks as an alternative to the factory to customize behavior