



Comprehensive SystemVerilog

5 jours



Présentation

“SystemVerilog (IEEE 1800™) est un nouveau langage basé sur le langage de description de matériel Verilog . Les extensions de System Verilog améliore le langage Verilog dans un certain nombre de secteurs, fournissant des améliorations de productivité pour les concepteurs RTL, les ingénieurs de vérification et tous ceux qui sont impliqués dans l’architecture et la conception système.

Comprehensive System Verilog est un programme de formation complet et intégré qui répond à la demande des ingénieurs de vérification et de tous ceux qui veulent évaluer SystemVerilog pour les applications de conception et de vérification. Ce cours est conçu pour mettre en valeur et faciliter l’utilisation de toutes les possibilités de SystemVerilog pour la conception et la vérification. Cela répond aux besoins des ingénieurs de vérification qui veulent utiliser le potentiel de la vérification basée sur les classes et les techniques orientées objet utilisant SystemVerilog aussi bien que du code RTL, des Assertions et des test benches. Les ingénieurs de conception qui n’ont pas l’intention d’utiliser la vérification basée sur les classes devraient s’orienter vers une version plus courte : SystemVerilog for Design Group qui partage le même contenu : les 3 premiers jours du cours Comprehensive SystemVerilog .

Doulos étant une société indépendante, les participants peuvent utiliser les outils de conception de leur choix durant les applications pratiques qui occupent 50 % du temps de la formation. Ces applications sont présentées sous forme d’exercices soigneusement préparés, afin de faciliter l’acquisition des connaissances.

Objectifs pédagogiques

- Utiliser les possibilités de vérification basées sur les classes pour le développement des test benches.
- Apprendre SystemVerilog pour la conception RTL.
- Mettre en place une méthode de vérification pilotée par la couverture basée sur SystemVerilog.
- Transfert de langage de vérification et de génération de Test bench précédemment utilisé vers la vérification basée sur les classes de SystemVerilog.
- Evaluer les possibilités de SystemVerilog pour la conception et la vérification.
- Analyser comment les équipes de développement peuvent améliorer leur productivité grâce à l’utilisation de SystemVerilog dans la conception et la vérification.

Qu’apprendrez vous ?

Le cours est structuré en différents secteurs distincts :

- **Fundamentals of SystemVerilog for Design** apprend à utiliser SystemVerilog pour la conception de RTL synthétisable, et aborde l’utilisation du langage pour la vérification.
- **SystemVerilog.Assertions** enseigne les principes de la conception et de la vérification basée sur le langage d’assertions, les caractéristiques du langage d’assertion de SystemVerilog pour créer ses propres assertions, et comment packager les bibliothèques de vérificateur.
- **Module-based SystemVerilog Verification** montre comment utiliser SystemVerilog pour construire des test benches bâtis sur une architecture de test bench à base du langage Verilog
- **Class-based System Verilog Verification** décrit comment écrire des test benches orientés objet utilisant les possibilités d’automatisation de SystemVerilog qui supporte une méthodologie de génération aléatoire contrainte

Pilotée par la couverture. Ces possibilités permettent d’écrire des test benches à un niveau d’abstraction plus élevé et d’être plus productif qu’avec un langage de description hardware standard.

Ce cours comprend un commentaire objectif et à jour sur les 3 approches de méthodologies de vérification qui ont été publiées et enseigne les parties du langage qui sont supportées.

Connaissances requises

Une connaissance pratique du langage Verilog est nécessaire.

Pour les participants sans connaissance ou expérience HDL, avoir participé au cours Comprehensive Verilog (ou équivalent) est une nécessité.

Pour les participants sans connaissance de Verilog mais avec une expérience d'utilisation de VHDL, nous proposons un cours rapide Fast-Track Verilog pour les utilisateurs VHDL qui permet aux participants d'acquérir les fondations nécessaires pour suivre le cours SystemVerilog.

Pour les cours sur site, on peut combiner les différents modules de System Verilog pour fournir un cours répondant aux besoins spécifiques.

Supports de cours

Les manuels de cours Doulos sont réputés pour être les plus détaillés et les plus faciles d'utilisation. Leur style, leur contenu et leur exhaustivité sont uniques dans le monde de la formation HDL. Ils sont souvent utilisés comme référence après avoir suivi les cours de formation. Sont compris dans la formation :

- Les notes de cours indexées constituant un manuel de référence Verilog concis.
 - Le cahier d'applications rempli d'exemples et d'applications pratiques pour vous aider à mettre en œuvre vos connaissances.
 - Le « Doulos Golden Reference Guide », aide-mémoire SystemVerilog complet et pratique (syntaxe, sémantique et astuces »).
-

Structure et contenu

Fundamentals of SystemVerilog for Design (Day 1 and Day 2 morning)

The SystemVerilog data type system

enum ♦ typedef ♦ struct ♦ union ♦ packed/unpacked ♦ packages and \$unit ♦ using arrays in SystemVerilog ♦ array and structure literals, assignment patterns

Nets and variables

Key changes in Verilog-2005 and SystemVerilog ♦ continuous assignments to variables ♦ modified driver and connection rules ♦ data types on ports and nets

Modules and processes

Port connection shorthand ♦ type parameters ♦ synthesis idioms for processes ♦ miscellaneous improvements to the language

Design applications of interfaces

The interface construct ♦ interfaces to encapsulate communication ♦ modports ♦ synthesis of interfaces and modports ♦ imported functions for design

SystemVerilog Assertions (Day 2 afternoon)

Introduction to Assertions

Assertions, properties, sequences ♦ clocking and sampling ♦ property implication ♦ uses of assertions ♦ simulation of assertions ♦ formal tools

Assertion methodology

Methodology consequences of assertion-based design and verification ♦ assertion and assumption ♦ benefits of assertions to the designer ♦ protocol checkers

A brief introduction to SVA syntax

Writing simple assertions of your own ♦ sequences and the ## operator ♦ repetition and time ranges ♦ sequence fusion ♦ overview of temporal operators ♦ local variables and actions in assertions

Packaging assertions

Assertions in interfaces and modules ♦ the *bind* construct ♦ deploying verification IP, particularly assertion-based IP

Module-based SystemVerilog Verification (Day 3)

Verification for Design groups

Bus functional models ♦ testbench architecture in classic Verilog ♦ stimulus and response timing
Using SystemVerilog to construct module-level testbenches
Clocking and program blocks ♦ testbench applications of interfaces ♦ building libraries of stimulus patterns (sequences) ♦ writing test cases to control the testbench

Dynamic data types

Strings ♦ queues ♦ dynamic arrays ♦ associative arrays ♦ queue and array methods ♦ foreach loop

Testbench automation

Brief introduction to testbench automation concepts ♦ randomisation, checking and coverage ♦ the need for constraints ♦ randomisation of stimulus data using `std::randomize` and traditional Verilog distribution functions

Class-based SystemVerilog Verification (Days 4 and 5)

Introducing classes

SystemVerilog's class syntax ♦ describing stimulus data and a stimulus generator ♦ randomization of class members (without constraints) ♦ objects and references ♦ constructors and `new` ♦ shallow copy using `new` ♦ writing a custom copy method

Hooking classes to the DUT

Dynamically-constructed test environment vs. statically-elaborated DUT and test harness ♦ using virtual interface and class-based BFMs ♦ the role of clocking and program blocks ♦ appropriate structure for DUT, clock generators and other structural elements ♦ constructing and launching the test environment using `program+initial` ♦ simple class-based testbench architecture

Advanced object-oriented features

Using inheritance to extend data classes ♦ type parameterization of classes ♦ using virtual methods and polymorphism to create generic testbench infrastructure such as channels, FIFOs, scoreboards ♦ deriving custom classes from generic base classes in a library such as AVM or VNM

Constrained randomisation

Full details of the SystemVerilog randomization machinery, including inline and declarative constraints ♦ dynamic control of constraints ♦ using constraints as checkers ♦ procedural randomization: `randcase`, `randsequence` ♦ control fields in classes for flexible constraints ♦ constraints and inheritance

Modelling techniques

Process control in SystemVerilog: new forms of `fork..join` ♦ spawning long-running threads with `fork..join_none` ♦ other modeling support: built-in classes for linked-list, semaphore, mailbox ♦ writing checkers and behavioural reference models

Class-based testbench architecture

Layered architectures – moving verification to higher levels of abstraction ♦ relation between layering and self-contained verification IP ♦ approaches taken by vendor-advocated methodologies (AVM, DTV, VNM) ♦ moving transactions around a testbench ♦ unifying ideas: transaction ports, transaction FIFO, design pattern for publisher/subscriber, callbacks for snooping and error injection ♦ base classes for transaction, transactor, environment ♦ automating the activity of testbench components

Coverage and planning

Coverage-driven TBA methodology ♦ coverage planning as the first step in verification process ♦ SystemVerilog coverage constructs in details ♦ analyzing and interpreting coverage data