

U1 - SystemVerilog

SystemVerilog for RTL Design and Functional Verification

Objectives

- Understand the role of SystemVerilog in modern digital design and verification environments
- Differentiate between simulation and synthesis, and apply appropriate coding practices
- Master SystemVerilog data types, including integral, composite, and dynamic structures
- Develop efficient RTL code using procedural blocks and proper assignment techniques
- Apply control flow, parallel processing, and casting for robust hardware modeling
- Design scalable systems using modules, interfaces, packages, and hierarchical constructs
- Understand simulation scheduling semantics and avoid race conditions in designs
- Apply object-oriented programming concepts for verification using classes and inheritance
- Create constrained-random test scenarios using randomization and constraints
- Measure and improve design quality using functional and code coverage techniques
- Use assertions to validate design behavior and detect errors early in simulation
- Develop structured test benches using interfaces and basic verification components
- Understand the fundamentals of UVM and its role in modern verification methodologies

Course environment

- Theoretical course
 - PDF course material (in English)
 - The trainer to answer trainees questions during the training and provide technical and pedagogical assistance
- Practical activities
 - Practical work with ModelSim QuestaSim and Vivado Simulator.
 - Practical activities represent from 40% to 50% of course duration
 - Example code, labs and solutions

Prerequisite

- Basic knowledge of digital design concepts (combinational and sequential logic)
- Familiarity with Verilog or VHDL (RTL coding fundamentals)
- Understanding of simulation concepts and waveform analysis
- Basic programming knowledge (C/C++ or similar is a plus)
- Familiarity with FPGA or ASIC design flow is recommended but not mandatory

Environnement du cours

- Cours théorique
 - Support imprimé et PDF (en anglais).
 - Assistance du formateur tout au long de la formation.
- Activités pratiques (40-50% de la durée)
 - Exemples de code, exercices et solutions.
 - Un PC (un par binôme au-delà de 6 stagiaires) avec carte cible si nécessaire.
- Machine virtuelle préconfigurée téléchargeable pour refaire les TP après le cours.
- Chaque session débute par un point avec les stagiaires.

Audience visée

- Tout ingénieur ou technicien en systèmes embarqués possédant les prérequis ci-dessus.

Plan du cours

First Day

Introduction

- Need for systemverilog
- Evolution from verilog
- Design vs verification roles
- Simulation vs synthesis basics

Data Types

- Integral types
- Composite types
 - Enumerations
 - Structures
 - Unions
- Arrays
 - Static arrays
 - Dynamic arrays
 - Associative arrays
 - Queues
- Other types
 - String
 - Event
 - Real / shortreal
 - Chandle
- Synchronization objects
 - Semaphores
 - Mailboxes

Exercise : Declare and manipulate:

- A packed vs unpacked array
- A struct representing a packet (addr, data, valid)
- A queue and perform push/pop operations

Exercise : Use an associative array to count occurrences of values

Exercise : Simple mailbox communication between two processes

Procedural & Process Blocks

- Initial & Final Blocks
- Always Blocks
- Procedural Assignments
 - Blocking vs Non-blocking
- Tasks & Functions
 - User Defined
 - Built-in
- Control Flow
 - Loops
 - Conditional statements
- Parallelism
 - fork...join
 - fork...join_any
 - fork...join_none

- disable fork
- Casting
 - Static casting
 - Dynamic casting

Exercise : Implement a counter module using:

- always_ff and non-blocking assignments

Exercise : Compare blocking vs non-blocking behavior using simulation

Exercise : Write a task to generate a pulse signal

Exercise : Use fork...join to run parallel stimulus threads

Second Day

Design Hierarchy & RTL Constructs

- Data Holders
 - Ports
 - Parameters
 - Localparams
 - Internal variables
- Design Units
 - Module
 - Interface
 - Program Block
- Interfaces
 - Modports
 - Clocking blocks
- Generate Constructs
 - For-generate
 - If-generate
- Elaboration-Time Concepts
 - Constant functions
 - Parameter calculations
 - \$clog2 and other elaboration functions
- Access & Scope
 - Hierarchical references
 - Scope resolution
- Packages
 - Import/export
 - Reusability
- Compilation Units

Exercise : Create an interface to connect DUT and testbench

Exercise : Use generate to instantiate multiple modules

Exercise : Organize code using a package

Scheduling Semantics

- Event regions
- Active / Reactive regions
- Race conditions
- Determinism

Exercise : Write a small testbench showing:

- Race condition between blocking/non-blocking assignments

Exercise : Observe waveform differences in:

- Active vs NBA regions

Exercise : Fix the race condition using proper coding style

Third Day

Classes

- Basics
 - Class syntax
 - Objects
- Class Properties
 - Initialization
 - Constructors
- Inheritance
- Polymorphism
- Casting rules
- Advanced Concepts
 - Parameterized classes
 - Static properties/methods
 - Forward declaration
 - External methods

Exercise : Create a transaction class (addr, data, control)

Exercise : Implement:

- Constructor
- Inheritance

Exercise : Demonstrate polymorphism using virtual methods

Randomization

- Overview
- Random variables
- Constraints
 - Inline constraints
 - Constraints blocks
 - Constraints modes
- Randomization methods
- System functions
- Stimulus Generation Techniques

Exercise : Create a random transaction with:

- Constraints on address range
- Distribution (dist) constraint

Exercise : Use randomize() and display results

Exercise : Add pre_randomize() and post_randomize() hooks

Coverage

- Overview
- Code Coverage
- Functional Coverage
 - Covergroups
 - Coverpoints
 - Bins
 - Cross coverage
 - Ignore bins / illegal bins
 - Coverage options
 - Sampling methods

Exercise : Create:

- Coverpoints
- Bins (including illegal bins)

Exercise : Add cross coverage

Exercise : Run simulation and analyze coverage results

Fourth Day

Assertions

- Immediate Assertions
- Concurrent Assertions
- Temporal Operators
- Sequences & Properties
- Assertion uses in verification

Exercise : Write assertions for:

- Valid signal must follow enable
- FIFO never overflows/underflows

Interfaces & Testbench Architecture

- Interface-based design
- Virtual interfaces
- Driver/Monitor concepts
- Basic testbench structure

Exercise : Build a simple testbench:

- Driver & DUT & Monitor

Exercise : Use an interface + virtual interface

Exercise : Send transactions from driver to DUT

Exercise : Capture outputs using monitor

Introduction to UVM

- Why UVM
- Basic components
- Sequence / Driver / Monitor
- Components
- Factory mechanism