



World Class Verilog & SystemVerilog Training

Sunburst Design - SystemVerilog OVM/UVM Verification Training

by Recognized Verilog & SystemVerilog Guru, Cliff Cummings of Sunburst Design, Inc.

Cliff Cummings is the only Verilog & SystemVerilog Trainer who helped develop every IEEE & Accellera Verilog, Verilog Synthesis and SystemVerilog Standard.

3 Days

70% Lecture, 30% Lab

Advanced Level

UVM is the unified future of SystemVerilog Verification

The good news is that the *Universal Verification Methodology (UVM)* is largely the same thing as the *Open Verification Methodology (OVM)* with a different first letter and a few enhancements including capabilities donated from VMM. This course teaches OVM & UVM noting the minor changes that differentiate the two methodologies.

Course Objective

Make verification engineers knowledgeable, proficient and productive at both OVM (version 2.1.1) or UVM using training materials developed by renowned Verilog & SystemVerilog Guru, Cliff Cummings.

Upon completion of this course, students will understand:

- SystemVerilog-verification language features
 - includes SystemVerilog classes & methods
 - includes SystemVerilog virtual classes & virtual methods
 - includes SystemVerilog interfaces and virtual interfaces
 - includes SystemVerilog constrained random testing
 - includes SystemVerilog functional coverage
 - includes SystemVerilog stimulus driving and verification sampling strategies
- OVM/UVM-verification language capabilities
 - includes OVM/UVM fundamentals and running tests
 - includes OVM/UVM base classes and reporting
 - includes OVM/UVM creating and properly starting tests
 - includes OVM/UVM testbench components and their usage
 - includes OVM/UVM transaction objects and their usage

For more information, contact:

Cliff Cummings - cliffc@sunburst-design.com - Sunburst Design, Inc. - 503-641-8446

Course Overview

Sunburst Design - SystemVerilog OVM/UVM Verification Training is a 3-day, fast-paced intensive course that focuses advanced verification features using SystemVerilog and the OVM/UVM base class libraries.

Why is OVM/UVM hard to learn?

Many engineers believe they can learn OVM/UVM by picking up and reading a book and the OVM or UVM User Guide. They quickly discover this is exceptionally difficult to do. To learn why it is so hard to learn OVM/UVM from existing materials, see the Appendix notes at the end of this syllabus.

Good OVM/UVM training should address each of the issues that make OVM/UVM materials difficult to understand (as described in the Appendix notes).

Target Audience

Sunburst Design - SystemVerilog OVM/UVM Verification Training is intended for design & verification engineers who require an introduction to IEEE SystemVerilog-2005 capabilities.

Prerequisites (mandatory)

This is a very advanced SystemVerilog design class that assumes engineers already have a good working knowledge of both Verilog and SystemVerilog. Engineers with no prior HDL training or experience will struggle in this class.

Classroom Details

Training is generally conducted at customer facilities and is sometimes offered as an open-enrollment training class. For maximum effectiveness, it is recommended to have one workstation or PC for every two students, with your preferred SystemVerilog simulator licenses (we often can help provide the simulator and temporary training licenses).

Please contact Cliff Cummings to customize the training materials to meet the needs of your engineering team.

For more information, contact:

Cliff Cummings - cliffc@sunburst-design.com - Sunburst Design, Inc. - 503-641-8446

Course Syllabus

Day One

OVM/UVM Resources & Introduction

Section Objective: Share OVM/UVM resources - There are conflicting guidelines from multiple resources regarding OVM/UVM methodologies. When one understands why there are differences, it is easier to learn from the divergent resources. This section explains the rationale behind the differing resources.

- OVM/UVM resources
- OVM/UVM introduction
- OVM/UVM conflicting recommendations - why?

Classes & Class Variables

Section Objective: Learn class basics - OVM and UVM are class libraries used to construct powerful verification environments. Class fundamentals are described in this section.

- SystemVerilog class basics
- Traditional Object Oriented (OO) programming -vs- SystemVerilog Classes
- Class definition & declaration
- Class members (data) & methods (tasks & functions)
- Class handles & using class handles
- Built-in class object constructor - new()
- super & this keywords
- Assigning object handles
- User-defined constructors
- Class extension & inheritance
- Class extension - adding properties & methods
- Class extension - overriding base class methods
- Assigning class handles
- Assigning extended handles to base handles
- Casting base handles to extended handles (technique used in OVM/UVM)
- Chaining new() constructors - illegal new() constructors
- Overriding class methods
- Extending class methods
- Extern methods
- local & protected keywords

For more information, contact:

Cliff Cummings - cliffc@sunburst-design.com - Sunburst Design, Inc. - 503-641-8446

OVM/UVM Overview – First Pass

Section Objective: Learn fundamentals of OVM/UVM testbench development and execution. This section briefly introduces important OVM/UVM fundamentals followed by a lab to help students become familiar with OVM/UVM concepts. Students will not fully understand all of the concepts in this section, but it is important that students do the lab to build a foundation for later learning. Each of the concepts in this section will be taught with greater detail in later sections. Engineers will learn more quickly after they have experienced the lab techniques at least once before tackling advanced OVM/UVM concepts.

- OVM/UVM transactions (data)
- Components (testbench components)
- Display command
- Top module DUT, interface, interface wrapper
- Testbench classes: environment, sequencer, driver, monitor, virtual interface
- Test classes
- Running tests using +OVM_TESTNAME command line switch
- First OVM/UVM Lab

Virtual Classes, Virtual Methods and Virtual Interfaces

Section Objective: Learn fundamentals of virtual classes/methods/interfaces - Virtual classes enable the creation of a set of base classes that provide a template for advanced verification environments. OVM/UVM is a base class library made up of mostly virtual classes that the user extends to create a reusable testbench environment. Virtual methods allow run-time base-method replacement that is a vital part of the OVM/UVM strategy (polymorphism).

- Introduction to Virtual - three types of "virtual"
- Virtual/abstract classes
- Legal & illegal virtual class usage
- Virtual class methods & restrictions
- Virtual Methods and rules
- Virtual -vs- non-virtual method override rules
- Why use virtual methods?
- Polymorphism using virtual methods
- Pure virtual methods (SystemVerilog-2009 update - used by OVM/UVM)
- Pure constraints (SystemVerilog-2009 update)
- Passing type parameters

For more information, contact:

Cliff Cummings - cliffc@sunburst-design.com - Sunburst Design, Inc. - 503-641-8446

Random & Constrained Random Class Variables

Section Objective: Learn about class variable randomization and setting constraints on that randomization - OVM and UVM use classes and constrained random variables for the construction of constrained random testing environments. Randomization and constraint fundamentals are described in this section.

- Directed -vs- random testing
- rand & randc class variables
- randomize() method - Randomizing class variables
- pre_randomize()/post_randomize() methods
- randomize ... with
- rand_mode()
- Randomization constraints
- Simple constraints
- Constraints blocks
- Important constraint rules
- Constraint distribution & set membership - dist & inside
- Constraint distribution operators
- External constraints & usage rules

OVM/UVM Base Classes & Reporting (standard OVM/UVM print/display commands)

Section Objective: Learn about OVM/UVM base classes and basic display and reporting commands.

- OVM/UVM Base Classes
- Introduction to OVM/UVM core base classes, `include files and macros
- Block diagram of DUT-testbench structure
- OVM/UVM verification components
- OVM/UVM components and objects
- OVM/UVM transactions (passing OVM/UVM data & methods - dynamic class objects)
- OVM/UVM factory basics
- Reporting methods & arguments
- How to set OVM/UVM-reporting configurations
- Reporting - file I/O

For more information, contact:

Cliff Cummings - cliffc@sunburst-design.com - Sunburst Design, Inc. - 503-641-8446

Day Two

OVM/UVM Transaction Base Classes

Section Objective: Learn to use and manipulate OVM/UVM transactions - This section answers important questions related to transactions including the basic question, why do transactions have to be classes?

- Why classes -vs- structs?
- Dynamic transaction classes
- `OVM/UVM_object_utils macro
- OVM/UVM_sequence_item -vs- OVM/UVM_transaction
- Field macros
- Randomizable data members
- Randomizable knobs
- Randomization constraints
- OVM/UVM_object constructors
- OVM/UVM sequence body task
- start_item(tx) - finish_item(tx)
- randomize() the transaction
- randomize() the transaction with inline constraints
- OVM/UVM sequences of OVM/UVM_sequence_item and OVM/UVM_sequence

Top Module & DUT

Section Objective: Learn how to connect an OVM/UVM class-based testbench to an actual Design Under Test (DUT) - This section explains the role that interfaces, virtual interfaces and configuration tables play in a testbench environment.

- Top module
- DUT (Design Under Test)
- DUT Interface
- Connecting DUT to DUT interface
- DUT interface handle
- DUT interface wrapper (class-based wrapper)
- Configuration tables
- set/get_config_object (storing the DUT interface handle)
- Virtual interfaces for verification

For more information, contact:

Cliff Cummings - cliffc@sunburst-design.com - Sunburst Design, Inc. - 503-641-8446

OVM/UVM Testbench Driver-Components

Section Objective: Learn to use OVM/UVM drivers, sequencers, agents and environments - Setting up the driver is a critical step. The class-based driver must drive the module-based DUT through a virtual interface that drives a real interface.

- OVM/UVM components to build the testbench structure
- OVM/UVM testbench structure (quasi-static class objects)
- `ovm/uvm_component_utils macros
- OVM/UVM_component constructors
- OVM/UVM components connected through ports & exports
- Testbench driver (get-port configuration)
- Managing the virtual interface - config table - required dynamic casting
- Testbench sequencer (get-export configuration)
- Testbench agent & environment
- User-defined testbench package

OVM/UVM Testbench Monitor, Analysis Ports & Checkers

Section Objective: Learn to use OVM/UVM monitors, analysis ports and checkers - OVM/UVM uses monitors, analysis ports and checkers (and scoreboards) to capture DUT outputs and analyze the outputs using functional coverage for correctness and completeness.

- OVM/UVM testbench components to capture and examine outputs
- OVM/UVM checkers and functional coverage collection through analysis ports
- Testbench monitor in the testbench agent
- Why do we need copy and compare methods?
- Checkers and scoreboards
- Functional coverage collector basics

Fundamentals of Running & Stopping OVM/UVM Tests

Section Objective: Learn proper methods to start and gracefully terminate OVM/UVM tests. This is a poorly documented topic in existing reference materials. Guidelines are presented to help properly stop tests using standard OVM/UVM techniques.

- Including OVM/UVM source files, base classes and macros
- Importing ovm_pkg/uvm_pkg
- run_test() fundamentals
- OVM/UVM phase basics
- Recommended ovm.f/uvm.f and run.f command files (Compilation)
- Selecting tests using +OVM/UVM_TESTNAME command line switch (Simulation)
- Declaring environments in tests
- OVM/UVM_component (test) constructors
- Test - run task & sequencer startup
- global_stop_request
- OVM/UVM_test_done objection

For more information, contact:

Cliff Cummings - cliffc@sunburst-design.com - Sunburst Design, Inc. - 503-641-8446

Day Three

OVM/UVM Factory, Constructors & Transaction Level Modeling (TLM) Basics

Section Objective: Learn the basics of OVM/UVM factories, registration, class construction and introduce the concept of factory overrides. This section will show why factories are important to OVM/UVM testbenches and discuss new() -vs- type_id::create() methods. This section also details how transactions are passed between classes through the use of ports, exports, put-configurations, get-configurations and transport configurations

- OVM/UVM factory basics
- Why is a factory used in OVM/UVM
- What is needed to use the factory
- new() -vs- type_id::create() construction
- Component and data lookup from the factory
- Running without re-compilation
- Tests can make substitutions without changing the testbench source code
- Introduction to factory overrides
- TLM ports & exports
- Why "ports" and "exports"
- TLM put, get and transport configurations
- Transaction-level control flow
- Transaction-level data flow
- Transaction-level transaction type
- Put configurations
- Get configurations
- Transport configurations

Clocking Blocks and Verification Timing

Section Objective: Learn important stimulus and verification timing issues and techniques - SystemVerilog clocking blocks help control timing for advanced OVM/UVM verification environments.

- Testbench stimulus/verification vector timing strategies
- #1step sampling
- Clocking blocks
- Clocking skews
- Default clocking block cycles
- Clocking block scheduling
- OVM/UVM usage of clocking blocks in an interface
- OVM/UVM driver timing using clocking blocks
- OVM/UVM signal sampling using clocking blocks

For more information, contact:

Cliff Cummings - cliffc@sunburst-design.com - Sunburst Design, Inc. - 503-641-8446

Functional Coverage

Section Objective: Learn functional coverage fundamentals - Functional coverage is used to track what has been tested. Functional coverage is used to help answer the question, "are we done testing?"

- Code coverage -vs- functional coverage
- Covergroups & coverpoints
- Auto-bins & user-named bins
- User-named array of bins
- Cross coverage
- Covergroup.sample() method
- Transition bins
- Coverage options & coverage capabilities

Fork-Join Enhancements and Advanced OVM/UVM Sequence Generation

Section Objective: Learn advanced sequence generation techniques - New fork-join capabilities were added to SystemVerilog and they are commonly used by advanced OVM/UVM sequence generation environments.

- New SystemVerilog fork-join processes
- OVM/UVM virtual sequences
- Advanced OVM/UVM sequence generation

Comparing Macros to Methods (time permitting)

Section Objective: Learn about OVM/UVM guideline differences - Many engineers believe they can learn OVM/UVM by picking up and reading a book and the OVM/UVM User Guide. They quickly discover this is exceptionally difficult to do. This section details different guidelines presented by different sources and the ease-of-use versus simulation-efficiency trade-offs.

- Which macros are universally recommended
- Which macros are less simulation efficient
- Macro-ease -vs- method-efficiency
- Field automation macros -vs- Implementing do_copy()/do_compare()
- OVM/UVM sequences and sequencers
- `OVM/UVM_sequencer_utils -vs- `ovm/uvm_component_utils (ease -vs- efficiency)
- start_item()/finish_item() -vs- `OVM/UVM_do & `OVM/UVM_do_with
- sequence.start(sequencer) method -vs- Sequence list and default sequence
- Configurations
- set/get_config_object (recommended)
- set/get_config_int (pros & cons)
- set/get_config_string (pros & cons)

For more information, contact:

Cliff Cummings - cliffc@sunburst-design.com - Sunburst Design, Inc. - 503-641-8446

APPENDIX

Why is OVM/UVM hard to learn?

Many engineers believe they can learn OVM/UVM by picking up and reading a book and the OVM or UVM User Guide. They quickly discover this is exceptionally difficult to do. Why is it so hard to learn OVM/UVM from existing materials?

Through years of experience, Sunburst Design has identified the following reasons why engineers struggle with existing OVM/UVM tutorial materials:

- 1) The OVM User Guide was written by Cadence and teaches Cadence recommended methods, which includes the use of a large number of OVM macros.
- 2) The OVM tutorials on VerificationAcademy.org are shown using Mentor recommended methods, which includes the use of fewer OVM macros and more OVM method calls.
- 3) The OVM Cookbook was written by Mentor employees and is based on an earlier version of OVM (the latest techniques are not shown in the book).
- 4) The above User Guide, tutorials and Cookbook do not acknowledge or explain the alternate methods, so users are left to draw erroneous conclusions that some of the methods shown are flawed, which is not true. Learners need to be taught the pros and cons of the alternate methods so that they understand why there are differences in the various methods presented.
- 5) All the people who have written OVM materials are *really, really* smart software engineers who assume that engineers already understand SystemVerilog syntax and semantics, object oriented programming and polymorphism semantics, and they don't know how to teach these concepts to beginners.
- 6) Many of those who have written OVM materials are software engineers who do not have a strong grasp of good hardware design practices, and it shows in many of the examples.
- 7) The OVM User Guide (chapter 2) and the OVM Cookbook (chapter 3) introduce Transaction Level Modeling (TLM) concepts, including put, get and transport communication, but do a poor job of tying the concepts into the rest of the OVM materials. Engineers often wonder why TLM was introduced in these texts.
- 8) All OVM materials show the driver on the right and the monitor on the left (right to left data-flow inside of the agent). This contradicts known good hardware block diagramming methods (data should flow from left to right in block diagrams) and adds an unnecessary level of confusion to the learning process for those who are familiar with good block diagramming techniques.
- 9) There is a huge shortage of complete simple examples. Most of the publicly available example code is in abbreviated code-snippet form, leaving the new user to guess what is missing. Finding full examples in the materials is rare. One notable example shows OVM used on a large VHDL design, which introduces yet another unknown to the learning process.
- 10) Of course, you must understand classes, class-extension, virtual classes, virtual methods, dynamic casting, polymorphism, randomization, constraints, covergroups, coverpoints, interfaces and virtual interfaces before you can learn OVM. Too many engineers try to learn OVM without a full understanding of these SystemVerilog fundamentals (this is not the fault of OVM authors).
- 11) Classes are applied as stimulus and sampled for verification. Existing materials do not explain why classes are used instead of structs?
- 12) Interfaces, virtual interfaces and their recommended usage-models are somewhat buried in the materials and are poorly explained (most authors assume you understand these concepts without much explanation - they are wrong).
- 13) There are a significant number of typos and mistakes sprinkled throughout the materials and examples. The mistakes leave the learner to try to figure out which coding styles are correct and which have typos.

Sunburst Design OVM/UVM training addresses each of these issues.

For more information, contact:

Cliff Cummings - cliffc@sunburst-design.com - Sunburst Design, Inc. - 503-641-8446