



**World Class Verilog & SystemVerilog Training**

**Sunburst Design - Advanced SystemVerilog for Design & Verification**

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.*

**4 Days**

**70% Lecture, 30% Lab**

**Advanced Level**

**Course Objective**

Make design and verification engineers productive using SystemVerilog using award winning materials developed by renowned Verilog & SystemVerilog Guru, Cliff Cummings.

Upon completion of this course, students will:

- Write efficient SystemVerilog reference models
  - includes new SystemVerilog data types and capabilities
  - includes new SystemVerilog RTL and abstraction capabilities
- Write complex self-checking testbenches
  - includes the use of new SystemVerilog Hardware Verification Language (HVL) capabilities
  - includes object-oriented stimulus generation using classes
  - includes constrained random stimulus generation
  - includes functional coverage capabilities
  - **\*NEW\*** includes optional OVM verification fundamentals and labs
  - includes proven techniques for generating self-checking tests
  - *may include* C-program interaction using the new SystemVerilog DPI
- Write efficient synthesizable SystemVerilog RTL models
  - includes new SystemVerilog data types and capabilities
  - includes new SystemVerilog RTL and abstraction capabilities
  - includes six different FSM coding styles
  - includes Clock Domain Crossing (CDC) and FIFO design techniques

## Course Overview

*Sunburst Design - Advanced SystemVerilog for Design & Verification* is a 4-day fast-paced intensive course that focuses on new and advanced design and verification features of SystemVerilog.

**\*NEW\* - Enhanced Verification Flow** - Based on seven years of teaching SystemVerilog, Sunburst Design has discovered that it is best to teach object-oriented class-based verification concepts early and often. Day-1 - includes Classes & Randomization (with labs) / Day-2 includes Constrained Random Variables in classes, Functional Coverage, Virtual Classes & Methods (with labs) / Day-3 includes Virtual Interfaces (with labs) and optional OVM fundamentals (with optional labs). When properly taught, these topics are not difficult but because they are new, it takes time and practice doing multiple labs for the concepts to be mastered.

This SystemVerilog training was developed and is frequently updated by the renowned SystemVerilog guru and IEEE SystemVerilog committee member, Cliff Cummings, who has presented at numerous SystemVerilog seminars and training classes world wide, including the 2003-2004 SystemVerilog NOW! Seminars and 2004-2005 ModelSim SystemVerilog Verification Shindigs.

The 1000+ page binder and 190+ page lab guide for this 4-day course covers all of the important SystemVerilog coding styles for design and verification. These materials are constantly being updated with the latest clarifications and corrections passed by the IEEE SystemVerilog committee, of which Cliff is an active participant. Numerous proven usage guidelines are taught and explained.

This fast-paced course teaches the IEEE 1800 advanced SystemVerilog capabilities for both design and verification tasks. Efficient and proven coding styles are combined with frequent exercises and insightful labs to demonstrate the power of the new SystemVerilog features. You will discover that SystemVerilog capabilities are fully backward compatible with Verilog-2001 designs.

The course content can be modified to meet the customized needs of individual design and/or verification teams.

## Target Audience

*Sunburst Design - Advanced SystemVerilog for Design & Verification* is intended for all design & verification engineers who require in-depth knowledge on the IEEE SystemVerilog-2005 standard. ***This course has been updated to include optional OVM fundamentals (with labs) for verification engineers that plan to use OVM.***

## Prerequisites (mandatory)

*This is a very advanced SystemVerilog class that assumes engineers already have a good working knowledge of the Verilog language.*

This course assumes that students have a practical working knowledge of Verilog HDL or have completed Verilog HDL training. Engineers with VHDL synthesis experience and some Verilog exposure will do well in this class. Engineers with no prior HDL training or experience will struggle in this class. Engineers with weak Verilog knowledge or experience should consider adding the 1-day, *Sunburst Design - Accelerated Introduction to Verilog-2001 & Best Known Coding Practices* course to fully prepare for advanced SystemVerilog training.

## The Sunburst Design - Advantage

Who is teaching your "expert" and "advanced" classes? Most companies will not tell you because their instructors might not have much design experience or may never have participated on any of the Verilog or SystemVerilog Standards groups or presented at industry recognized conferences. Go to our web site and read about the Sunburst Design - Instructors - they are simply the best at what they do and they have the experience and qualifications to offer world-class training.

### Sunburst Design Courses:

- *Sunburst Design - Advanced SystemVerilog for Design & Verification* - 4-5 days
  - *Sunburst Design - Advanced SystemVerilog for Design* - 3 days
  - *Sunburst Design - Advanced SystemVerilog for Verification* - 3-4 days
  
- *Sunburst Design - Expert Verilog-2001 for Synthesis & Verification* - 4 days
  - *Sunburst Design - Expert Verilog-2001 & Coding for RTL Design & Synthesis* - 2 days
  - *Sunburst Design - Expert Verilog-2001 Design, RTL Synthesis & Verification Techniques* - 2 days
  
- *Sunburst Design - Comprehensive Verilog-2001 Design & Best Coding Practices* - 4 days
  - *Sunburst Design - Introduction to Verilog-2001 & Best Coding Practices* - 2 days
  - *Sunburst Design - Advanced Verilog-2001 Knowledge & Design Practices* - 2 days
  - *Sunburst Design - Accelerated Introduction to Verilog-2001 & Best Known Coding Practices* - 1 day

**Course Customization?** - Sunburst Design courses can be customized to include **your** company's coding guidelines or to modify the course for a different audience. Sections can be added or deleted from a course to meet you company's needs.

## Course Syllabus

### Day One

#### SystemVerilog Enhancements & Methodology Overview

- Includes a quick review of SystemVerilog resources available to design & verification engineers.

- Verilog & SystemVerilog Keywords
- SystemVerilog Books & Resources
- SystemVerilog Enhancements Strategy & High-Level Methodology

#### Data Types & Typedefs

- Includes data types, enumerated types, compilation units, packages, casting and randomization functions.

- Nets & Variables Fundamentals & Guidelines
- Blocking & Nonblocking Assignment Fundamentals & Guidelines
  
- SystemVerilog data types
- Enhanced literal numbers syntax
- Resolved & Unresolved types
- 4-state & 2-state types
- Typedefs
- Near-Universal types
- SystemVerilog type usage guidelines
- Enumerated types
- Struct data type intro
- Type parameters
- Intro to the SystemVerilog program construct
- \$unit & \$root
- Compilation units & separate compilation
- Packages & :: (package scope operator)
- SystemVerilog package strategies
- Strings
- Static & dynamic type-casting
- Random number generation: \$random -vs- \$urandom -vs- \$urandom\_range
- Simulation command aliases & switch definitions
- LABS: Multiple SystemVerilog types, typedefs, type-casting and logic labs

## **SystemVerilog Operators, Loops, Jumps. Intro to Logic-Specific Processes, Enhanced functions & tasks**

- New always\_type blocks help RTL designers. To help verification engineers understand design constructs, the always\_type blocks are briefly introduced in this section. Enhancements to tasks and functions make them more useful and easier to use.

- New SystemVerilog operators
- Enhanced loops & jumping statements
- always\_comb / always\_latch / always\_ff - (1-slide introduction only)
- always @\* -vs- always\_comb
- SystemVerilog enhancements to tasks & functions
- SystemVerilog priority & unique - modifiers for case- & if-statements
- `timescale directive
- SystemVerilog timeunit & timeprecision

## **Classes, Class Variables & Randomization of Class Variables**

- Object oriented programming using classes and constrained random variables for the construction of powerful verification environments are described in this section.

- SystemVerilog class basics
- Traditional Object Oriented (OO) programming -vs- SystemVerilog Classes
- Class definition & declaration
- Class members & methods
- Class handles
- Class object construction - new()
- super & this keywords
- Assigning object handles
- User-defined constructors
- Class extension & inheritance
- Adding properties & methods to extended classes
- Chaining new() constructors - illegal new() constructors
- Overriding class methods
- Extending class methods
- local & protected keywords
- Constrained Random Variables
- Directed -vs- Random Testing
- rand & randc class data types
- randomize - Randomizing class variables
- randcase
- Built-in randomization methods
- Reference: randsequence & examples
- LABS: classes, methods, class extension, random variables and randomization

## Day Two

### **Nonblocking Assignments, Race Conditions & SystemVerilog Event Scheduling**

- SystemVerilog is fully backward compatible with Verilog-2001 (it is also fully race backward compatible!) This section describes in detail how the new SystemVerilog event scheduling works and how it will reduce race conditions between RTL designs and verification suites.

- Verilog-2001 Event Scheduling
- 8 guidelines for RTL coding & nonblocking assignments
- SystemVerilog enhanced scheduling - includes IEEE 1800 latest updates
- Verilog -vs- SystemVerilog race conditions
- Scheduling of new SystemVerilog commands
- \* Blocking & Nonblocking Assignment Details
- \* Mixed RTL & Gate simulations

### **Implicit .\* and .name Port Instantiation**

- Implicit port connections can reduce top-level ASIC and FPGA coding efforts by more than 70% and simultaneously enforce greater port type checking.

- Verilog-2001 positional & named ports
- SystemVerilog .\* implicit ports
- SystemVerilog .name implicit ports
- Implicit port connection rules & comparisons - includes IEEE 1800 latest updates
- Strong port-type checking
- New debugging techniques - automatic expansion of .\* ports
- Block-level testbenches with implicit ports
- Advantages & disadvantages
- LABS: implicit port instantiation labs
- LABS (optional) : SystemVerilog random numbers

### **Interfaces**

- Interfaces are a powerful new form of abstraction and this section details how they work for design and verification. This section also discusses when and when not to use interfaces. Virtual interfaces are described after the introduction of virtual classes and virtual methods on day three.

- Interface usage overview
- Introduction to generic interfaces
- Interfaces -vs- records
- How interfaces work
- 4 requirements for good interface usage
- Interfaces - legal & illegal usage
- Interface constructs
- Interface modports
- Generic interfaces
- LABS: multiple interface and interface-protocol labs

## **Constrained Random Variables, Functional Coverage and Virtual Classes, Methods and Interfaces**

- Random variables & constrained random testing are important HVL enhancements to SystemVerilog to assist the verification task. Functional coverage enables engineers to verify what has already been tested and to focus additional stimulus generation to meet untested functionality.

- Randomization constraints
- Simple constraints
- Multi-Statement constraints
- Important constraint rules
- Constraint distribution & set membership - dist & inside
- Constraint distribution operators
- External constraints
- Covergroups
- Coverpoints
- Coverpoint bins & labels
- Cross coverage
- Covergroup options
- Coverage capabilities
- 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)
- Pure constraints (SystemVerilog-2009 update)
- Passing type parameters
- Virtual interfaces
- Three testbench requirements to communicate through virtual interfaces
- Connecting a design to a class-based verification suite using virtual interfaces
- LABS: constrained random stimulus and reports

## **OVM - Introduction & Fundamentals (Optional)**

*- Optional verification lecture topic with labs - best to add a 4<sup>th</sup> day to the verification flow for adequate lecture and lab time.*

- The Open Verification Methodology (OVM) is a class-based methodology for constructing and executing high-level verification suites. This section teaches the fundamentals and functionality of OVM

- OVM resources
- Introduction to OVM core base classes, include files and macros
- OVM testbench structure (quasi-static class objects)
- OVM transactions (passing OVM data & methods - dynamic class objects)
- Ports & exports
- Puts & gets
- Transaction-level connection: (1) Control flow, (2) Data flow, (3) Transaction data type
- Put configurations
- Get configurations
- Transport configurations
- OVM Drivers
- OVM Checkers
- OVM Monitors
- OVM Agents
- OVM Standard Phases
- Additional OVM topics
- LAB: first OVM testbench
- LAB: OVM testbench with virtual interface
- LAB: OVM testbench with agents



## Day Three

### **Program Blocks & Clocking & Hardware Verification Language**

- SystemVerilog's new built-in Hardware Verification Language (HVL) capabilities are detailed and how program and clocking blocks facilitate testing is explored.

- Fundamental testbench construction
- Program blocks
- Program block interaction with modules
- Final blocks
- Testbench stimulus/verification vector timing strategies
- Verification event scheduling
- #1step sampling
- Clocking blocks
- Clocking skews
- Default clocking block cycles
- Clocking block scheduling
- New SystemVerilog fork-join processes

### **Structs, Unions, Packed & Unpacked Arrays, Semaphores & Mailboxes**

- Packed & unpacked arrays, unions and structs allow greater abstraction and more concise coding. The new dynamic array types facilitate behavioral modeling and assist in the development of verification environments. Semaphores and mailboxes are sometimes used in advanced verification methodologies such as OVM and VMM

- Data organization
- Structs & assignment patterns
- Packed & unpacked arrays
- Array indexing
- Structs & packed structs
- Unions & packed unions
- Dynamic arrays & methods
- foreach loop
- Associative arrays & methods
- Queues & concatenation operations
- Queue methods
- Semaphores & methods
- Interesting semaphore key usage
- Mail boxes & methods
- Bounded & unbounded mailboxes

## **DPI - Direct Programming Interface - SystemVerilog's C-Language Interface**

*(Optional section - may be omitted to give more time to other topics)*

- The Direct Programming Interface (DPI) can be used to simulate C-code with SystemVerilog code. This section describes how this can be done and how DPI programming differs from PLI programming.

- DPI layers
- function import
- function export
- task export
- Using SystemVerilog simulation timing in a C model
- DPI -vs- PLI example
- No PLI required
- How to compile and simulate C-code with SystemVerilog designs
- SystemVerilog & SystemC
- LAB: SystemVerilog using C-code functions

## **SVA - SystemVerilog Assertions**

- This section details how the SystemVerilog Assertion (SVA) syntax works and how assertions can be used for design and verification. Special macro-techniques are shown to reduce assertion coding effort by up to 80%.

- What is an assertion? / Who should add assertions?
- Assertion benefits - bug detection efficiency
- SystemVerilog assertion types
- SystemVerilog immediate assertions
- SystemVerilog concurrent assertions
- Assert & cover properties & labels
- Properties and assert property
- Overlapping & non-overlapping implications
- Edge testing functions
- Sequences
- Vacuous success
- Property styles
- Reduced assertion coding effort using macros
- Macros with default arguments (SystemVerilog-2009 update)
- Assertion coding style efficiency benchmarks
- SystemVerilog assertion system functions
- Sampled value functions
- Assertion severity tasks
- Assertion and coverage example of an FSM design
- Binding SVA to an existing model
- Bind command details and guidelines
- LABS: SystemVerilog Assertions with synchronous FIFO design

## Day Four - (Not lectured in the 3-Day SystemVerilog for Verification Class)

### Logic Specific Processes, Unique & Priority - full\_case & parallel\_case

- The new always\_type blocks show design intent and help ensure construction of proper hardware designs. The always\_type blocks are discussed in detail in this section. This section also details how unique and priority are new SystemVerilog replacements for the dangerous "Evil Twins," full\_case parallel\_case.

- Logic specific processes (always\_type blocks) document designer intent
- always\_comb
- always\_latch
- always\_ff
- Added design checks using always\_type blocks
- always @\* -vs- always\_comb
- void functions
- always\_comb & void functions
- Combinational sensitivity
- Design encapsulation through void functions
- always\_ff for DDR? (SystemVerilog-2009 enhancement)
- full\_case parallel\_case, "the Evil Twins"
- What is full\_case?
- What is parallel\_case?
- unique & priority case
- unique & priority if
- unique0 (SystemVerilog-2009 enhancement)
- Three examples using case modifiers
- \* LABS: simple SystemVerilog combinational and sequential logic labs
- \* Multiple small synthesis examples

### SystemVerilog FSM Design Techniques

- Six different FSM coding styles, enhanced with new SystemVerilog constructs, are detailed and compared for coding and synthesis efficiency. Multiple FSM designs are benchmarked for coding style efficiency.

- FSM coding goals
- Moore & Mealy
- Binary & Onehot
- ASIC -vs- FPGA FSM design
- Review proven FSM coding styles
- One always block - avoid this
- Two always blocks - recommended
- Three always blocks - recommended
- Onehot case(1'b1) - recommended
- Onehot parameters - avoid this
- Output encoded - recommended

- Coding & synthesis efficiency
- Verilog-2001 FSM enhancements
- SystemVerilog FSM enhancements
- Advanced enumerated types
- LABS: SystemVerilog FSM design labs

### **Multi-clock Clock Domain Crossing (CDC) & FIFO Design Techniques using SystemVerilog**

- Very advanced design techniques from Cliff's award-winning presentations on the efficient implementation of multi-clock CDC & FIFO designs. These materials are not specific to SystemVerilog but solutions are shown using SystemVerilog syntax (advanced techniques that all design engineers should know - the stuff you did not learn in college).

- Metastability
- Multi-clock Clock Domain Crossing (CDC) design & synthesis strategies
- Multi-signal CDC techniques
- MTBF (Mean Time Before Failure)
- Syncing before passing multiple CDC signals
- Multiple CDC signals - consolidation
- Multiple CDC signals - synchronization
- Multiple CDC signals - Multi-Cycle Path (MCP) Formulation
- Synchronizing counters
- Gray codes
- Gray code counters
- CDC Design partitioning
- CDC simulation issues
- CDC gate-level simulation X-avoidance techniques
- Multi-clock FIFO design techniques from Cliff's award-winning presentations

### **Classroom Details**

Training is generally conducted at your facilities. For maximum effectiveness, we recommend having 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).

### **For more information, contact:**

Cliff Cummings - [cliffc@sunburst-design.com](mailto:cliffc@sunburst-design.com) - Sunburst Design, Inc. - 503-641-8446