

# ***SystemVerilog Fundamentals***

by

Recognized Verilog & SystemVerilog Guru, Cliff Cummings of Paradigm Works, Inc.

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

## **Course Summary**

### **Course Description**

SystemVerilog Fundamentals is a 2-day, fast-paced, intensive course that introduces new SystemVerilog features for design, simulation, and synthesis. Efficient and proven coding styles are combined with frequent exercises and insightful labs to demonstrate the capabilities of new SystemVerilog features. You will discover that SystemVerilog capabilities are fully backward compatible with Verilog-2001 designs.

The course objective is to introduce engineers to world-class SystemVerilog language capabilities using award-winning materials developed by renowned Verilog & SystemVerilog Guru, Cliff Cummings.

Students will learn, use, and understand the following:

- SystemVerilog language fundamentals
  - includes new SystemVerilog data types and capabilities
  - includes new SystemVerilog RTL and abstraction capabilities
  - includes use of dynamic types and arrays for behavioral modeling
  - includes 7 different SystemVerilog FSM coding styles
  - optionally - includes inclusion of C-models using the new SystemVerilog DPI
  - includes using SystemVerilog Assertions (SVA) for design and verification
  - Students will have a good understanding of SystemVerilog RTL design features.

This SystemVerilog training was developed and is frequently updated by the renowned SystemVerilog guru and IEEE SystemVerilog committee member, Cliff Cummings, who has presented numerous SystemVerilog seminars and training classes worldwide, including the 2003-2004 SystemVerilog NOW! Seminars and 2010 ModelSim SystemVerilog Assertion-Based Verification Seminars.

**Duration**            2 days

**Breakdown**        70% Lecture, 30% Lab

**Level**                Intermediate to Advanced

- Prerequisites**     **This is an advanced SystemVerilog training class that assumes engineers already have a good working knowledge of Verilog.** Engineers with no prior HDL training or experience will struggle in this class.
- Online Details**     The course delivery can be in-person or virtual (virtual courses are convenient for both U.S. and non-U.S. engineers).

## Course Syllabus

### **Half Day #1**

#### **(1) 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

#### **(1a) 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 - and why you should avoid it.
  - \$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
  - LABS: Multiple SystemVerilog types, typedefs, type-casting, and logic labs

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

- The new *always\_type* blocks show design intent and help ensure the construction of proper hardware designs. The *always\_type* blocks are discussed in detail in this section. Enhancements to tasks and functions make them more useful and easier to use.
  - New SystemVerilog operators
  - Enhanced loops & jumping statements
  - 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)
  - SystemVerilog enhancements to tasks & functions
  - ``timescale` directive
  - SystemVerilog *timeunit* & *timeprecision*
  - LABS: simple SystemVerilog combinational and sequential logic labs

## Half Day #2

### (2a) Unique & Priority - *full\_case* & *parallel\_case*. *Timeunit* & *timeprecision*

- This section details how unique and priority are the new SystemVerilog replacements for the dangerous "Evil Twins," *full\_case* *parallel\_case*. *Timeunit* & *timeprecision* enhancements are also described.
  - *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)
  - ``timescale` directive
  - SystemVerilog *timeunit* & *timeprecision*

### (3) Structs, Unions, Packed & Unpacked Arrays

- Packed & unpacked arrays, unions, and structs allow greater abstraction and more concise coding. The new dynamic array types are used more in verification and have been moved to the UVM training course.
  - Structs & assignment patterns

- Packed & unpacked arrays
- Array indexing
- Structs & packed structs
- Unions & packed unions

### **(3a) DPI - Direct Programming Interface - SystemVerilog's C-Language Interface**

- 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

### **(4) 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 - auto-schematic generation
  - Block-level testbenches with implicit ports
  - Advantages & disadvantages
  - \*LABS: implicit port instantiation labs

## **Half Day #3**

### **(5) 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 (*slides included but not lectured*)
- Mixed RTL & Gate simulations (*slides included but not lectured*)

## (6) SystemVerilog FSM Design Techniques

- Seven 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
  - Three always blocks - recommended
  - Two always blocks – recommended (combinational outputs)
  - One always block - avoid this
  - Four always blocks – recommended (better synthesis results)
  - Onehot case(1'b1) - recommended
  - Onehot parameters - avoid this
  - Output encoded - recommended
  - Coding & synthesis efficiency
  - SystemVerilog FSM enhancements
  - Advanced enumerated types
  - LABS: SystemVerilog FSM design labs

## Half Day #4

### (7) 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 introduced after the UVM training class covers virtual classes and virtual methods.
  - 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
- LABS: interface and interface-testbench lab

## **(8) 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%. Other recommendations to reduce SVA coding errors are included in this section.
  - 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