

Advanced SystemVerilog Design & Synthesis

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

There are two versions of this course. If engineers already know SystemVerilog, they should choose the 2-day Sunburst Design - Expert SystemVerilog Design & Synthesis course.

This course combines the content of the 2-day SystemVerilog Fundamentals training, the 2-day Expert SystemVerilog Design & Synthesis course, and the 1-day CDC & FIFO Design course into a single continuous flow over four days.

The course objective is to give engineers world-class SystemVerilog language, expert design & synthesis training using award-winning materials developed by renowned Verilog & SystemVerilog Guru, Cliff Cummings.

Duration	4 days
Breakdown	60% Lecture, 40% Lab
Level	2 days Fundamental and 2 days Expert
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 <u>will struggle</u> 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

(Included in SystemVerilog Fundamentals Training)

(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.

- 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.
 - Strings
 - Packages & :: (package scope operator)
 - SystemVerilog package strategies
 - Static & dynamic type-casting
 - Random number generation: \$random -vs- \$urandom -vs- \$urandom_range
 - LABS: Multiple SystemVerilog types, typedefs, type-casting, and logic labs

(2a) SystemVerilog Operators, Loops, Jumps. 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

(2b) 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

(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) 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

Half Day #4

(7) 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

Half Day #5

(8) Latches & Priority Encoders. Unique & Priority - full_case & parallel_case

- Introduction to Verilog synthesis design flows. Detailed description of two synthesis problem areas: latches and priority encoders. Detailed description of the synthesis directives "full_case" and "parallel_case", and why they should generally be avoided. This section also details how unique and priority are the new SystemVerilog replacements for the dangerous "Evil Twins," full_case and parallel_case.
 - always_latch
 - always blocks & sensitivity lists
 - Generating latches
 - Generating priority encoders
 - Latch & priority encoder guidelines
 - 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)
 - One example using case modifiers (+ 2 more reference examples)
 - LABS: simple SystemVerilog combinational and sequential logic labs

(9) Combinational Logic I & II

- RTL coding styles for combinational logic, including problems and inefficiencies that arise from poor coding styles. Numerous combinational labs identify potential problem areas associated with common combinational coding styles. Verilog-2001 (V2K1) enhancements are discussed, including reasons to avoid over-usage of generate statements. An example of poor usage is I/O pad instantiation (use the more concise and better-supported Array of Instances). More combinational labs highlight many potential problem areas associated with common combinational coding styles.
 - always_comb
 - Continuous assignments
 - Always blocks
 - V2K1 @* and comma-separated sensitivity lists

- Instantiated library elements
- Instantiated primitives
- Synthesizable and non-synthesizable Verilog constructs
- Bitwise -vs- logical operators
- Tasks & functions
- Tri-state drivers
- Bi-directional buses
- Instantiating ASIC/FPGA library primitives
- Guidelines
- LABS: Combinational logic labs I & II

Half Day #6

(10) Sequential Logic

- This section covers coding styles for sequential logic. Efficient design inference using adders and other large resources is also detailed. It also discusses the advantages and disadvantages of instantiation.
 - always_ff
 - Edge-sensitive sensitivity list
 - Basic asynchronous & synchronous resets
 - Additional flip-flop coding styles
 - Simulation/synthesis differences
 - Simulation efficiency
 - Register banks
 - Memories
 - Instantiating Blocks
 - Resource sharing
 - LABS: Sequential logic labs

(11) Synchronous & Asynchronous Reset Design

- Detailed material for selection and usage of synchronous and asynchronous reset design taken from actual design experiences.
 - Synchronous vs. asynchronous resets
 - Reset removal metastability
 - Asynchronous reset synchronizer circuitry
 - Reset distribution trees and techniques
 - Multiple clock domains reset synchronization

(12) **Generate Statements, Arrays of Instances, Verilog-2001 Features**

- This section describes useful Verilog-2001 enhancements, including generate statements, and compares generate statements to existing and powerful arrays of instances.
 - Generate for-loops
 - genvars
 - Arrays of Instances (AOI)
 - Comparing generates to AOI
 - Generate if-else
 - Generate case
 - Multidimensional wire-arrays
 - Nested generate statements
 - Signed arithmetic
 - Problems and gotchas related to signed arithmetic

Half Day #7

(13) **Synthesis Labs Review**

- An in-depth review of all the coding styles used in the combinational & sequential labs and the synthesized results. Draw conclusions about which coding styles yield the most efficient logic implementations.
 - Review of previous synthesis lab results

(14) **SystemVerilog FSM Design Techniques (Included in SystemVerilog Fundamentals Training)**

- 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 #8

(Included in Expert Clock Domain Crossing (CDC) & FIFO Design Techniques Course)

(15) Multi-clock Clock Domain Crossing (CDC) Design Techniques using SystemVerilog

- Very advanced design techniques from Cliff's award-winning presentations on the efficient implementation of multi-clock CDC 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).
 1. Metastability & synchronizers - synchronizing 1-bit signals
 2. Passing multiple control signals - synchronizing multi-bit signals or buses
 - a) Consolidation
 - b) Controlled- multicycle path formulations (MCP)
 - c) FIFO synchronizer
 - d) Gray codes & Gray code counters
 3. Design Partitioning - design & synthesis techniques
 - a) Naming Conventions
 - b) Synthesis scripting & timing analysis issues
 4. Simulation issues
 - a) X-propagation issues
 - b) Synopsys command for SDF files
 - c) Multi-SDF files
 - d) ASIC/FPGA vendor cells and models
 - e) Simulation model to expose synchronization problems
 5. LAB: MCP-controlled synchronization

(16) Multi-clock FIFO Design Techniques using SystemVerilog

- Very advanced design techniques from Cliff's award-winning presentations on the efficient implementation of 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).
 1. Multi-clock FIFO design - a large section on FIFO design and FIFO issues
 2. LAB: 2-clock FIFO