

Comprehensive Verilog-2001 Design & Best Coding Practices

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

Sunburst Design – Comprehensive Verilog-2001 Design & Best Coding Practices is a 4-day, fast-paced, intensive course on the IEEE 1364-2001 Verilog Hardware Description Language and its use for hardware design and verification. This course emphasizes RTL modeling and efficient testbench techniques while teaching Verilog-2001 syntax. Unlike traditional Verilog courses that start with gate-level design and work towards behavioral coding styles, the Verilog-2001 Design course focuses on important behavioral design and synthesis coding styles first, along with Verilog verification techniques, to instill and reinforce good coding habits early and often during the entire 4-day course. This course prepares engineers to take SystemVerilog training.

Upon completion of this course, students will:

- Write efficient Verilog-2001 RTL models
 - have a detailed understanding of the important Verilog-2001 language features
 - understand the necessary constructs for
 - design, synthesis and verification
 - simulation of gate-level netlists
 - know how to write and simulate
 - complex hardware models
 - simple synthesizable models
 - self-checking testbenches
 - know how to run efficient Verilog simulations
 - be immediately productive at modeling and simulating complex Verilog designs

Duration	4 days
Breakdown	50% Lecture, 50% Lab
Level	Beginner to Advanced
Prerequisites	No prior HDL experience required, but knowledge of digital design concepts is strongly recommended.
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

Comparing this 3-day Class to the 4-day Comprehensive Verilog Class

The 3-day class is intended to cover all the same important Verilog RTL concepts as the 4-day class, but removes the lecture and labs on infrequently used behavioral topics, as well as in-depth discussion of gate primitives, switch primitives, User Defined Primitives (UDPs), and specify blocks, which are rarely used by typical Verilog designers. Most Verilog design and verification engineers are probably better served by this more condensed 3-day Verilog Training course.

The 4th-day material (spread throughout the course as appropriate) includes additional material on efficient combinational and sequential RTL coding. Also included is additional material on assembling self-checking Verilog testbenches.

Half Day #1

(1) Introduction to Verilog Modeling

- An introduction and overview of major Verilog-2001 modeling basics.
 - Overview of Verilog Resources
 - Modules
 - Port and net declarations
 - V2K1 ANSI-C style module headers
 - Instantiation with positional and named ports
 - Procedural blocks: initial & always
 - Hierarchy
 - Introduction to synthesis design flows
 - Power-user guidelines (presented in section 1 - detailed in later sections)

(2) Verilog HDL Syntax & Semantics

- Detailed instruction of important Verilog-2001 (V2K1) language syntax.
 - Good formatting = fewer bugs & better documentation
 - Comments
 - V2K1 attributes
 - Identifier names
 - Name scopes
 - Language tokens
 - Numbers and logic values
 - Net & variable types
 - Scalars & vectors
 - Multi-dimensional arrays
 - Port declaration styles
 - Parameters
 - Verilog's 4+ logic values

(3) Design Verification & Running Simulations

- An introduction to writing Verilog testbenches and running Verilog simulations.
 - Strength basics
 - Simulation times, delays, timescales and time formatting
 - Writing Verilog testbenches
 - Repeat-loop & forever loop
 - Important compiler directives
 - Display and formatting commands
 - System tasks for simulation control
 - Using multiple Verilog source files & Verilog command files
 - Running a Verilog simulation
 - Lab: basic testbench development

Half Day #2

(4) Continuous Assignments, Operators, Verification & Running Simulations

- Detailed discussion of continuous assignments with design examples, followed by an overview of Verilog-2001 operators, also with examples.
 - Continuous assignments
 - Procedural continuous assignments (do not use these!)
 - Required net declarations
 - ?: conditional operator

(5) Programming Statements & Timescales

- Detailed discussion of blocking and nonblocking assignments, followed by an overview of Verilog-2001 programming statements with examples. This section concludes with a discussion of Verilog timescales and their impact on simulation efficiency.
 - Verilog operators - arithmetic, bitwise, logical, unary reduction, more
 - Sequential & parallel statement groups
 - Blocking & nonblocking assignments (introduced)
 - Time delays
 - Level-sensitive timing controls
 - Edge-sensitive timing controls
 - Sensitivity lists (V2K1)
 - If-else & case statements
 - For, while, repeat & forever loops
 - Tasks, functions and automatic (V2K1)
 - Rise, fall, min, max delays
 - `timescale & \$timeformat
 - Lab: (optional) `timescale & \$timeformat capability and efficiency

Half Day #3

(6) Combinational Logic Modeling I

- Behavioral & synthesizable coding styles for modeling combinational logic. Includes multiple Verilog-2001 enhancements.
 - Sensitivity lists
 - Continuous assignments
 - Procedural combinational blocks
 - V2K1 comma-separated and @* sensitivity lists
 - Inertial & transport delays
 - Correct methods for adding behavioral timing delays

(7) Combinatorial Logic Modeling II

- Additional coding styles to efficiently do RTL modeling of combinatorial logic.
 - Latches (modeling & avoidance)
 - Priority Encoders
 - Expert RTL coding topics
 - Muxes
 - Lab: combinational model and verify an 8-bit ALU

Half Day #4

(8) Sequential Logic Modeling

- Behavioral & synthesizable coding styles for modeling sequential logic
 - Sensitivity lists
 - Flip-flops and latches
 - Synchronous and asynchronous inputs
 - Where to add timing delays
 - Lab: combinational model and verify an 8-bit ALU
 - Lab: sequential model and verify a pipeline
 - Lab: (optional) extra labs

(9) RAM and ROM Modeling

- Behavioral coding styles for modeling RAMs and ROMs
 - Modeling memories
 - Array declarations
 - System tasks to load memories
 - Preferred coding style for read operations
 - Preferred coding style for write operations
 - Testing bi-directional ports
 - Basic timing constraints
 - Lab: model and verify a RAM

Half Day #5

(10) Blocking vs. Nonblocking Assignments

- Detailed instructions about Verilog blocking and nonblocking assignments. Detailed description of how the Verilog event queue works. Coding guidelines using blocking & nonblocking assignments. After using blocking and nonblocking assignments for two days, we are now ready to give the details for the guidelines already presented.
 - Different types of blocking assignments
 - Different types of nonblocking assignments
 - Explanation of Verilog event scheduling
 - Assignment execution order
 - Pipeline examples
 - Delay line modeling
 - Common misconceptions about nonblocking assignments
 - Blocking & nonblocking assignment guidelines

(11) Structural Netlists

- How to construct a hierarchical Verilog netlist.
 - Design hierarchy
 - Module instantiation
 - Port & net mismatches
 - Parameterized models
 - Redefining parameters
 - V2K1 Multi-dimensional arrays
 - Arrays of Instance
 - Generate statements
 - Lab: model and verify a hierarchical design

Half Day #6

(12) State Machine Design

- Detailed description and guidelines for coding Verilog state machines.
 - Moore, Mealy, binary & onehot state machines
 - State machine coding style guidelines
 - Parameter states vs. `define states (do not use `define)
 - Two always block state machine coding style
 - One always block state machine coding style
 - Three always block, registered output, coding style
 - Lab: EISA bus arbiter state machine design

(13) Verilog Gates and SDF Timing

- Introduction to gate primitives, User Defined Primitives, specify blocks and SDF backannotated timing.
 - Component models
 - Gate primitives
 - Time delays
 - User Defined Primitives (UDPs)
 - Specify blocks
 - Timing checks
 - SDF back annotation - \$sdf_annotate command
 - Optional Lab: SDF Backannotation
 - Optional Lab: UDP (User Defined Primitive)

Half Day #7

(14) Testing & Testbenches

- This section covers various testbench stimulus and verification techniques, including use of Verilog tasks to generate bus functional models. Includes a detailed discussion of hierarchical referencing to probe designs and generate self-checking testbenches. Also details the contents of SIMUTIL, a set of simulation tasks for rapid development of self-checking testbenches.
 - V2K1 named parameter passing and defparam avoidance
 - V2K1 file I/O enhancements
 - Testing approaches
 - Advanced loops, tasks, and functions & V2K1 automatic tasks and functions
 - Bus functional model testing
 - Global variables
 - Hierarchical referencing
 - ASCII "string" display
 - Dumpfiles and waveform viewer strategies
 - Self-checking tests
 - Correct timing and techniques for applying stimulus and verifying results
 - SIMUTIL self-checking testbench tasks
 - Regression test generation

(15) Verilog Wizardry (Introduce & Start Project)

- This is the final project for the course. Students will use all aspects of the Verilog language in an actual design flow. Students will draw upon what they have learned in the previous two and a half days to model, simulate and verify a complete Digital Signal Processor design.
 - Lab: model and verify a synthesizable Digital Signal Processor

Half Day #8

(16) Verilog Wizardry (Complete Lab Intensive Project)

- Lab: Complete model and verify a synthesizable Digital Signal Processor