System Verilog for VHDL Users

Fast-track Verilog for VHDL Users

Course Description

Verilog for VHDL Users is an intensive 2-day course, converting knowledge of VHDL to practical Verilog skills.
Contrasting Verilog and VHDL, this course demonstrates similarities and highlights differences between two hardware description languages and their associated design flows.
The syllabus covers the Verilog language, coding for register transfer level (RTL) synthesis, developing test fixtures, and using Verilog tools.
Labs comprise about 50% of class time, and are based around carefully designed exercises to reinforce and challenge the extent of learning.
Because Doulos is independent, delegates can usually use their choice of design tools during the workshops. Workshops are based around carefully designed exercises to reinforce and challenge the extent of learning, and comprise approximately 50% of class time.

Who should attend?

- Engineers proficient in VHDL who need migrate to Verilog or evaluate SystemVerilog
- Engineers who are proficient in VHDL but need to become competent in the application of, and interaction with, the Verilog HDL as well.

What will you learn?

- The differences and similarities between VHDL and Verilog
- How to use the Verilog language for hardware design and logic synthesis
- How to write thorough Verilog text fixtures to verify your designs
- How to avoid common mistakes when coding Verilog for synthesis

Prerequisites:

- Delegates must have a good working knowledge of VHDL and digital hardware design
- No previous knowledge of Verilog is required

Software Tools

Standard Verilog 1364 complaint simulator

Course materials

Course materials are renowned for being the most comprehensive and user friendly available. Their style, content and coverage are unique in the HDL training world and have made them sought after resources in their own right. Course fees include:

- Fully indexed course notes providing a concise Verilog reference
- Workbook full of practical examples to help delegates apply their knowledge

Course Outline

Introduction

- What is Verilog?
- Brief history and current status
- The PLI
- Scope of Verilog
- Design flow
- Verilog-2001
- SystemVerilog
- Verilog books and Internet resources

Differences between VHDL and Verilog

- "Philosophy"
- Red Tape
- Strong typing
- Determinisim
- Data abstraction
- Structure vs behaviour – Nets vs registers
- Language structure – architecture, packages, configurations, files
- Identifiers
- Output ports
- Implicit wires
- Arrays
- Aggregates
- Signedness
- Operators
- Signal vs variables/nets
- Process vs initial/always
- If, case, loop differences
- File i/o
- Hierarchical names

Verilog Basics

- Modules & ports
- Continuous assignments
- Comments
- Names
- Nets and strengths
- Design hierarchy
- Module instances
- Primitive instances
- Text fixtures
- $monitor
- Initial blocks
- Logic values
- Vectors
- Registers
- Numbers
- Output formatting
- Timescales
- Always blocks
- $stop and $finish
- Using nets and variables correctly

Cont.

Fast-track Verilog for VHDL Users

Cont.

Combinational Logic
- Event control
- If statements
- Begin-endw Incomplete assignment and latches
- Unknown and don't care
- Conditional operator
- Tristates
- Case, casez and casex statements
- Full_case and parellel_case directives
- For, repeat, while and forever loops
- Integers
- Self-disabling blocks
- Combinational logic synthesis

Sequential Logic
- Synthesising flip-flops & latches
- Avoiding simulation race hazards
- Nonblocking assignments
- Asynchronous & synchronous resets
- Clock enables
- Synthesizable always templates
- Designing state machines
- State machine architectures
- Verilog code-based FSM strategy
- State encoding
- Unreachable states & safe design practices
- One-hot machines

Other features of Verilog
- Verilog operators
- Part selects
- Concatenation & replication
- Shift registers
- Conditional compilation
- Parameterisation and generate
- Hierarchical names
- Arithmetic operators and their synthesis
- Signed and unsigned values
- Memory arrays
- RAM modelling and synthesis
- $readmemb and $readmemh

Tasks and Functions
- Understanding tasks
- Task arguments
- Task synchronization
- Tasks and synthesis
- Functions

Test Fixtures
- File I/O – Writing to files; File access using MCDs; Reading from files
- Automated design verification using Verilog
- Force and release
- Gate-level simulation
- Back annotation using SDF
- "Traditional" Verilog libraries
- Configuration and libraries
- Command-line options
- Behavioural modelling

Behavioural Verilog
- Algorithmic coding
- Synchronization using waits & event control
- Concurrent-disabling of always blocks
- Named events
- Fork & join
- High-level modelling using tasks, Implicit FSMs and concurrent-disabling
- Understanding intra-assignment controls
- Overcoming clock skew
- Blocking and nonblocking assignments
- Continuous procedural assignment

Gate Level Verilog
- Structural Verilog
- Using built-in primitives
- Net types & drive strengths
- UDPs Gate, net & path delays
- Specify blocks
- Smart paths
- Pulse rejection
- Cell library modelling

SystemVerilog
- Background
- Who is SystemVerilog for?
- Current status of SystemVerilog
- RTL enhancements
- Interfaces
- Assertions
- Testbenches
- C interface

SystemVerilog - Designing & Verification

Course Description

This comprehensive course is a thorough introduction to SystemVerilog constructs for design and Verification. This class addresses writing RTL code using the new constructs available in SystemVerilog, writing testbenches to verify your design under test (DUT) utilizing the new constructs available in SystemVerilog. New data types, structs, unions, arrays, procedural blocks, and re-usable tasks, functions, and packages, Object-oriented modeling, randomization, code coverage, assertions, and the Direct Programming Interface (DPI) are all covered. The information gained can be applied to any digital design verification approach. This course combines insightful lectures with practical lab exercises to reinforce key concepts.
In this two-day course, you will gain valuable hands-on experience. Incoming students with a Verilog background will finish this course empowered with the ability to more efficiently develop RTL designs and the ability to more efficiently verify designs.

Level: FPGA 1

Who Should Attend?

FPGA designers and logic designers
Hardware and verification engineers

Software Tools:

- Vivado® Design or System Edition 2014.1
- Questa Sim Prime Simulator 10.2c

Hardware:

- Architecture: N/A*
- Demo board: Kintex®-7 FPGA KC705 board*

Skills Gained: After completing this training, you will be able to:

- Describe the features and benefits of using SystemVerilog for RTL design
- Identify and define the new data types supported in SystemVerilog
- Utilize an enumerated data type for coding a finite state machine (FSM)
- Explain how to utilize structures, unions, and arrays
- Describe the new procedural blocks and analyze the affected synthesis results
- Define the enhancements and ability to reuse tasks, functions, and packages
- Identify how to simplify module definitions and instantiations using interfaces
- Examine how to efficiently code in SystemVerilog for FPGA design simulation and synthesis
- Target and optimize Xilinx FPGAs by using SystemVerilog
- Synthesize and analyze SystemVerilog designs with the Vivado Design Suite
- Download a complete SystemVerilog design to an evaluation board
- Describe the advantages and enhancements to SystemVerilog to support verification
- Analyze and use the improvements to tasks and functions
- Discuss and use the various new verification building blocks available in SystemVerilog
- Examine object-oriented programming and apply that to verification
- Explain the various methods for creating random data
- Create and utilize random data for generating stimulus to a DUT
- Identify how SystemVerilog enhances functional coverage for simulation verification
- Utilize assertions to quickly identify correct behavior in simulation
- Identify how the direct programming interface can be used with C/C++ in a verification environment

Course Outline

Days 1-2

1. Introduction to SystemVerilog
2. Data Types
3. Lab 1: SystemVerilog Data Types
4. Structures, Unions, and Arrays
5. Additional Operators in System Verilog
6. Procedural Statements and Flow Control
7. Lab 2: always_ff and always_comb Procedural Blocks
8. Functions, Tasks, and Packages
9. Interfaces
10. Lab 3: Interfaces and Design Download
11. Packages
12. Targeting Xilinx FPGAs

…Cont

Lab Description

Lab 1: System Verilog Data Types – Utilize enumerated data types to build a finite state machine and perform synthesis to analyze the results.
Lab 2: always_ff and always_comb Procedural Blocks – Learn to use the new procedural blocks always_comb, always_ff, and always_latch to produce the synthesized results intended.
Lab 3: Interfaces and Design Download – Use an interface to simplify the module inputs and outputs. Download and verify the design in-circuit.

Cont…

Days 3-4

1. Introduction to SystemVerilog for Verification
2. Data Types
3. Tasks and Functions
4. Lab 1: Implementing Tasks and Functions
5. SystemVerilog Verification Building Blocks
6. Lab 2: Connecting the Testbench to the DUT
7. Object-Oriented Modeling
8. Lab 3: Object-Oriented Modeling
9. Randomization
10. Lab 4: Randomization
11. Coverage
12. Lab 5: Coverage
13. Assertions
14. Lab 6: Assertions
15. Direct Programming Interface

Lab Description

Lab 1: Implementing Tasks and Functions – Use a task and function to provide input data for a DUT and perform simulation.
Lab 2: Connecting the Testbench to the DUT – Utilize new SystemVerilog verification building blocks to connect the input data to the DUT.
Lab 3: Object-Oriented Modeling – Use object-oriented programming concepts to create a class for enhancing the verification of the DUT.
Lab 4: Randomization – Create random data as input into the DUT to fully validate the design.
Lab 5: Coverage – Create and use a coverage group to validate the code coverage for the DUT. Make adjustments and again validate the coverage.
Lab 6: Assertions – Create an assertion to validate all possible conditions are verified for the DUT.