SystemVerilog for FPGA/ASIC Design

SystemVerilog as a first language for FPGA or ASIC design

Course Description:
SystemVerilog for FPGA/ASIC Design prepares the engineer for practical project readiness for FPGA or ASIC design, including RTL synthesis, block-level test benches, and FPGA design flows. Delegates targeting FPGAs will take away a flexible project infra-structure which includes a set of scripts, example designs, modules and constraint files to use, adapt and extend on their own projects. While the emphasis is on the practical SystemVerilog-to-hardware flow for FPGA devices, this training course also provides the essential foundation needed by ASIC and FPGA designers wishing to go on to use the advanced features of SystemVerilog for functional verification.
SystemVerilog for FPGA/ASIC Design is suitable for delegates who are learning SystemVerilog as their first hardware description language. For teams who are already skilled in Verilog or VHDL, this training course can be offered in a shortened form for on-site delivery. For verification teams who are looking to use the class-based features of SystemVerilog for constrained random functional verification, Logtel provides Modular SystemVerilog for in-house training options.
Because Logtel 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.

Level: Standard Level

Training Duration: 4 days

Who should attend?

- Digital hardware design engineers who wish to learn how to use SystemVerilog for FPGA or ASIC hardware design at the register-transfer level (RTL) and for block-level verification.

Prerequisites:
Delegates should have a good working knowledge of digital hardware design, or have attended Essential Digital Design Techniques (or equivalent). No previous SystemVerilog or Verilog knowledge is required.

What will you learn? The course is structured into four distinct sections.

- The SystemVerilog language concepts and constructs essential for FPGA and ASIC design
- How to write SystemVerilog for effective RTL synthesis
- How to target SystemVerilog code to an FPGA device architecture
- How to write simple and efficient SystemVerilog test benches
- The tool flow from SystemVerilog through simulation, synthesis and FPGA place-and-route
- How to write high quality SystemVerilog code that reflects best practice in the industry
- How to write re-usable, parameterisable SystemVerilog code by exploiting parameters
- How to run gate-level simulations

Training materials
Doulos class 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. The materials include:

- Fully indexed class notes creating a complete reference manual
- Workbook full of practical examples and solutions to help you apply your knowledge
- Tool tour guides (to support the tools and technologies used on the course)

Structure and Content

1. Introduction
- The scope and application of SystemVerilog Design and tool flow
- FPGAs
- Introduction to synthesis and synchronous design
- SystemVerilog resources

2. Modules
- Modules Ports
- Continuous assignments
- Comments
- Names
- Design hierarchy
- Module instantiation
- Port connection shorthand
- Test benches
- Simple procedures

3. Numbers and formatting
- Numbers
- 2-valued & 4-valued logic
- Vectors
- Bit and part select
- System calls
- Output formatting
- Time units
- Always blocks
- Ending simulation

Continued …

SystemVerilog for FPGA/ASIC Design

SystemVerilog as a first language for FPGA or ASIC design

Course Content:

... Continued

4. Procedures - Event controls and If
- initial & always
- Event controls
- if
- begin & end
- Combinational logic
- always_comb
- Incomplete assignment and latches
- Unknowns and don't cares

5. Procedures - Case and Loop
- case
- casez & casex
- unique & priority
- for
- repeat
- while
- forever
- break
- continue
- Integer variables

6. Clocks and Flip-flops
- Synthesis of flip-flops and latches
- Avoiding race hazards
- Blocking & non-blocking assignments
- Dealing with the scheduler and with clock skew always_ff
- Synchronous & asynchronous resets
- Clock enables
- Coding templates for synthesis
- Flip-flop inference

7. Operators and Names
- Bitwise, logical, reduction, and equality operators
- Concatenation
- Replication
- Conditional operator
- Hierarchical names

8. Memories
- Array types
- RAM coding style
- Memory inference versus instantiation
- $readmemb

9. Finite State Machines
- State machines architecture
- Coding styles for state machines
- enum
- State encoding
- Unreachable states and input hazards

10. Types and Packages
- Integer types
- struct
- packed
- typedef
- package
- using
- Scope resolution

11. Synthesis of Arithmetic and Counters
- Arithmetic operators and their synthesis
- signed & unsigned values
- Resource sharing
- Exploiting FPGA resources

12. File Organization and Parameterized Modules
- Compilation units
- Compiler directives
- include
- Macros
- Conditional compilation
- parameter
- localparam
- Parameter overriding
- Parameterized modules
- generate

13. Tasks and Functions
- task
- function
- Argument passing
- return
- local declarations
- automatic
- Synthesis of tasks & functions

14. File I/O
- Opening & closing files
- Reading & writing files
- The $display family
- Using $fscanf

15. Interfaces and Clocking Blocks
- interface
- Interface ports
- modport
- clocking
- Clockvars
- Clocking drives
- Avoiding scheduler problems

16. Gate-Level Simulation
- Libraries
- SDF back-annotation
- The DPI
- force & release
- Gate-level language constructs
- Tristates & bus resolution