SystemVerilog Essentials Simulation & Synthesis

Course Description

This course provides all necessary theoretical and practical know-how to design programmable logic devices using SystemVerilog standard language.

The course combines 50% theory with 50% practical work in every meeting. The practical labs cover all the theory and also include practical digital design.

SystemVerilog is a significant new enhancement to Verilog and includes major extensions into abstract design, testbench, formal, and C-based APIs.

SystemVerilog also defines new layers in the Verilog simulation strata. These extensions provide significant new capabilities to the designer, verification engineer and architect, allowing better teamwork and co-ordination between different project members.

The course goes into great depth, and touches upon every aspect of the standard with directly connected to the topics needed in the industry today.

The course emphasizes the difference between testing code and synthesizable code.

Course Duration

4 days

Goals

1. Become familiar with SystemVerilog language
2. Use SystemVerilog declaration spaces
3. Use SytsemVerilog User-defined and Enumerated types
4. Use SystemVerilog arrays, structures and unions
5. Become familiar with SystemVerilog procedural blocks, tasks and functions
6. Use SystemVerilog new operators and loop statements
7. Model Finite State Machines with SystemVerilog
8. Design hierarchy with SystemVerilog
9. Become familiar with SystemVerilog interfaces

Intended Users

Hardware or software engineers who would like to design with SystemVerilog

System engineers who would like to enhance their professional skills

Prerequisites

- Background in digital logic
- Verilog language

Course Material

1. Simulator: Modelsim
2. Synthesizer and Place & Route: Quartus Prime
3. Course book (including labs)

Table of Contents

Day #1

- Static and automatic variables
  - Static and automatic variable initialization
  - Synthesis guidelines for automatic variables
  - Guidelines for using static and automatic variables
- Deterministic variable initialization
  - Initialization determinism
  - Initializing sequential logic asynchronous inputs
- Type casting
  - Static casting
  - Dynamic casting
  - Synthesis guidelines
- Constants

- **SystemVerilog User-Defined & Enumerated Types**
  - User-defined types
    - Local typedef definitions
    - Shared typedef definitions
    - Naming convention for user-defined types
  - Enumerated types
    - Enumerated type label sequences
    - Enumerated type label scope
    - Enumerated type values
    - Base type of enumerated types
    - Typed and anonymous enumerations
    - Strong typing on enumerated type opertaions
    - Casting expressions to enumerated types
    - Special system task and methods for enumerated types
    - Printing enumerated types

- **SystemVerilog Arrays, Structures, and Unions**
  - Structures
    - Structures declarations
    - Assigning values to structures
    - Packed and unpacked structures
    - Passing structures through ports
    - Passing structures as arguments to tasks and functions
    - Synthesis guidelines
  - Unions
    - Unpacked unions

        - Tagged unions
        - Packed unions
        - Synthesis guidelines
        - Using structures and unions
    - Arrays
        - Unpacked arrays
        - Packed arrays
        - Using packed & unpacked arrays
        - Initializing arrays at declaration
        - Assigning values to arrays
        - Copying arrays
        - Copying arrays and structures using bit-stream casting
        - Arrays of arrays
        - Using user-defined types with arrays
        - Passing arrays through ports and to tasks and functions
        - Arrays of structures and unions
        - Arrays in structures and unions
        - Synthesis guidelines
    - The foreach array looping construct
    - Array querying system functions
    - The $bits "sizeof" system function
    - Dynamic arrays, associative arrays, sparse arrays and strings

Day #2

- **SystemVerilog Procedural Blocks**
    - Verilog general purpose always procedural block
    - SystemVerilog specialized procedural blocks
        - Combinational logic procedural blocks
        - Latched logic procedural blocks
        - Sequential logic procedural blocks
        - Synthesis guidelines

- **SystemVerilog Tasks & Functions**
    - Implicit task and function statement grouping
    - Returning function values

- Returning before end of tasks and functions
- Void functions
- Passing task/function arguments by name
- Enhanced function formal arguments
- Functions with no formal arguments
- Default formal argument direction and type
- Default formal argument values
- Arrays, structures and unions as formal arguments
- Passing argument values by reference instead of copy
- Named task and function ends
- Empty tasks and functions

- **SystemVerilog Procedural Statement**
  - New operators
    - Increment and decrement operators
    - Assignment operators
    - Equality operators with don't care wildcards
    - Set membership operator (inside)
  - Operand enhancements
    - Operations on 2-state and 4-state types
    - Type casting
    - Size casting
    - Sign casting
  - Enhanced for loops
    - Local variables within for loop declarations
    - Multiple for loop assignments
    - Hierarchically referencing variables declared in for loops
    - Synthesis guidelines
  - Bottom testing do…while loop
  - The foreach array looping construct
  - New jump statements
    - The continue statement
    - The break statement
    - The return statement
    - Synthesis guidelines
  - Enhanced block names
  - Statement labels
  - Enhanced case statements
    - Unique case decisions

        - Priority case statements
        - Unique and priority versus parallel_case and full_case
    - Enhanced if…else decisions
        - Unique if…else decisions
        - Priority if decisions

Day #3

- **SystemVerilog Finite State Machines Modeling**
    - Representing state encoding with enumerated types
    - Reversed case statements with enumerated types
    - Enumerated types and unique case statements
    - Specifying unused state values
    - Assigning state values to enumerated type variables
    - Performing operations on enumerated type variables
    - Using 2-state types in FSM models
    - Synthesis guidelines

- **SystemVerilog Design Hierarchy**
    - Module prototypes
        - Prototype and actual definition
        - Avoiding port declaration redundancy
    - Named ending statements
        - Named module ends
        - Named code block ends
    - Nested module declarations
        - Nested module name visibility
        - Instantiating nested modules
        - Nested module name search rules
    - Simplified netlists of module instances
        - Implicit .name port connections
        - Implicit .* port connection
    - Net aliasing
        - Alias rules
        - Implicit net declarations
        - Using aliases with .name and .*

- Passing values through module ports
  - All types can be passed through ports
  - Module port restrictions in SystemVerilog
- Reference ports
  - Reference ports as shared variables
  - Synthesis guidelines
- Enhanced port declarations
  - Verilog-1995 port declarations
  - Verilog-2001 port declarations
  - SystemVerilog port declarations
- Parameterized types

- **SystemVerilog Interfaces**
  - Interface concept
    - Disadvantages of Verilog's module ports
    - Advantages of SystemVerilog interfaces
    - SystemVerilog interface contents
    - Differences between modules and interfaces
  - Interface declarations
    - Source code declaration order
    - Global and local interface definitions
  - Using interfaces as module ports
    - Explicitly named interface ports
    - Generic interface ports
    - Synthesis guidelines
  - Instantiating and connecting interfaces
  - Referencing signals within an interface
  - Interface modports
  - Using tasks and functions in interfaces
    - Interface methods
    - Importing interface methods
    - Synthesis guidelines for interface methods
    - Exporting tasks and functions
  - Using procedural blocks in interfaces
  - Reconfigurable interfaces
  - Verification with interfaces

Day #4

- **Behavioral & Transaction Level Modeling**
  - Behavioral modeling
  - What is a transaction?
  - Transaction level modeling in SystemVerilog
  - Transaction level models via interfaces
  - Bus arbitration
  - Transactors, adapters, and bus functional models
  - More complex transactions

- **Complete Design Modeled with SystemVerilog**
  - Building a project that encapsulates most of the course contents