

# EDA Academy

## Verification Course Outline

(updated in April 2025)

<b>Verilog Coding Series.....</b>	<b>2</b>
Verilog Coding - Fundamentals (Public FREE).....	2
Verilog Coding - Synthesis (Member Free).....	6
Verilog Coding - Verification (Member Free).....	10
Verilog Coding - Design (Member Free).....	15
<b>SystemVerilog Language Series.....</b>	<b>19</b>
SystemVerilog Language - Design (Member Free).....	19
SystemVerilog Language - Verification (Member Free).....	23
SystemVerilog Language - Advanced (Member Free).....	27
SystemVerilog Language - Testbench (Member Free).....	31
SystemVerilog Language - Coverage (Member Free).....	36
SystemVerilog Language - Assertion (Member Free).....	40
<b>UVM Series.....</b>	<b>45</b>
Universal Verification Methodology - SystemVerilog (Member Free).....	45
Universal Verification Methodology - Fundamentals (Member Free).....	48
Universal Verification Methodology - Advanced (Member Free).....	52
Universal Verification Methodology - Register Verification I (Member Free).....	56
Universal Verification Methodology - Register Verification II (Member Free).....	60
<b>Formal Verification Series.....</b>	<b>63</b>
Introduction to Formal Verification.....	63
Formal Verification: SVA Coding.....	66
Formal Verification: PSL Coding.....	69
Formal Verification Full Course List.....	71
Resource: Formal Verification of Properties in Hardware Design (PDF).....	73
<b>ABV Series.....</b>	<b>79</b>
Introduction to Assertion Based Verification - SVA (Member Free).....	79
Introduction to Assertion Based Verification - PSL (Member Free).....	81
<b>SVA/PSL Coding Series.....</b>	<b>83</b>
SystemVerilog Assertion (SVA) - Fundamentals (Member Free).....	83
SystemVerilog Assertion (SVA) - Formal (Member Free).....	86
SystemVerilog Assertion (SVA) - Advanced (Member Free).....	90
Property Specification Language (PSL) - Fundamentals (Member Free).....	93
Property Specification Language (PSL) - Formal (Member Free).....	96
Property Specification Language (PSL) - Advanced (Member Free).....	100



# Verilog Coding Series

## Verilog Coding - Fundamentals (Public FREE)

### 1. Logic Design Fundamentals

- 1.1. Basic Logic Gates
- 1.2. Full Adder Example
- 1.3. Simplifying Logic Functions with Boolean Algebra
- 1.4. Simplifying Logic Functions Using Karnaugh Maps
- 1.5. Designing with NAND and NOR Gates
- 1.6. Hazards in Combinational Circuits
- 1.7. Types of Flip-Flops
- 1.8. Mealy Sequential Circuit Design
- 1.9. Design of a Moore Sequential Circuit
- 1.10. State Equivalence and Optimization in Sequential Circuits
- 1.11. Key Timing Considerations in Sequential Circuits
- 1.12. Tristate Buffers and Their Role in Digital Circuits

### 2. Introduction to Verilog

- 2.1. Key Concepts in Digital Design
- 2.2. Hardware Description Languages (HDLs)
- 2.3. Levels of Abstraction in HDL Design
- 2.4. Benefits of Using HDL
- 2.5. Roles in HDL-Based Digital System Design
- 2.6. Challenges in Adopting HDL
- 2.7. Key Features of Verilog Language
- 2.8. Basics of Verilog Module Declaration
- 2.9. Verilog Module Hierarchy and Instantiation
- 2.10. Port Connection Syntax in Verilog
- 2.11. Connecting Ports in Module Instances
- 2.12. Using Procedural Constructs in Verilog
- 2.13. Synchronizing Module Behaviors
- 2.14. Naming Rules, Case Sensitivity, and Commenting in Verilog
- 2.15. Simulating HDL Designs: Compilation, Libraries, and Elaboration

### 3. Verilog Data Types

- 3.1. Verilog Value Set and Simulation Behavior
- 3.2. Verilog Data Types Overview
- 3.3. Verilog Net and Register Rules
- 3.4. Verilog Port Type Rules



- 3.5. Declaring Vectors
- 3.6. Assigning Between Different Vector Widths
- 3.7. Specifying Literal Values
- 3.8. Automatic Extension of Unsigned Literals
- 3.9. Variable Vector Selection, Declaring Nets
- 3.10. Handling Undeclared Identifiers
- 3.11. Net Declaration Assignment
- 3.12. Resolving Net Conflicts
- 3.13. Types of Variables
- 3.14. Integer and reg Assignments
- 3.15. Arrays in Verilog
- 3.16. Declaring Module Parameters
- 3.17. Local Parameters and Parameter Passing

#### 4. Applying Verilog Operators

- 4.1. Bit-Wise Operators
- 4.2. Unary Reduction Operators
- 4.3. Logical Operators
- 4.4. Arithmetic Operators
- 4.5. Enhanced Signed Arithmetic
- 4.6. Shift Operators
- 4.7. Relational Operators
- 4.8. Equality Operators
- 4.9. Conditional Operator
- 4.10. Concatenation Operator
- 4.11. Replication Operator
- 4.12. Verilog Operator Precedence

#### 5. Making Procedural Statements

- 5.1. Describing Module Behavior
- 5.2. Synchronizing Module Behaviors
- 5.3. Interactions Between Procedural Blocks
- 5.4. Procedural Assignments
- 5.5. Conditional Statements
- 5.6. Case Statements
- 5.7. Casex Statements
- 5.8. Casez Statements
- 5.9. while Loop Statements
- 5.10. for Loop Statements
- 5.11. epeat Loop Statements
- 5.12. forever Loop Statements



## 6. Assignment Types in Verilog

- 6.1. Understanding Blocking Assignments
- 6.2. Race Conditions in Blocking Assignments
- 6.3. Impact of Blocking Assignment Order
- 6.4. Understanding Nonblocking Assignments
- 6.5. Nonblocking Assignments in Sequential Procedures
- 6.6. Using Temporary Variables in Sequential Logic
- 6.7. Managing Multiple Assignments in Procedures
- 6.8. Understanding Continuous Assignments
- 6.9. Multiple Continuous Assignments
- 6.10. Understanding Procedural Assignments
- 6.11. Multiple Procedural Assignments
- 6.12. Avoiding Combinational Feedback Loops
- 6.13. Generate Statements in Verilog
- 6.14. Generate Statement - Conditional If
- 6.15. Generate Statement - Conditional Case
- 6.16. Generate Statement - For Loop

## 7. Simulation Cycle

- 7.1. Procedural Blocks and Event Control
- 7.2. Blocking Procedural Assignment
- 7.3. Nonblocking Procedural Assignment
- 7.4. Simulation Cycle: 1/6
- 7.5. Simulation Cycle: 2/6
- 7.6. Simulation Cycle: 3/6
- 7.7. Simulation Cycle: 4/6
- 7.8. Simulation Cycle: 5/6
- 7.9. Simulation Cycle: 6/6
- 7.10. Simulation Cycle Summary
- 7.11. Verilog Timing Controls
- 7.12. Event Controls
- 7.13. Level-Sensitive Event Controls
- 7.14. Delay Control
- 7.15. Timescale Directive

## 8. Functions and Tasks

- 8.1. Verilog Subroutines Overview
- 8.2. Function Declaration
- 8.3. Calling Functions



- 8.4. Constant Functions
- 8.5. Task Declaration
- 8.6. Calling Tasks
- 8.7. Disabling Tasks
- 8.8. Issues in Functions and Tasks
- 8.9. Automatic Tasks
- 8.10. Argument Passing in Verilog
- 8.11. Subroutine Side Effects
- 8.12. Accessing Module Variables via Subroutines

## 9. Verilog Compiler Directives

- 9.1. Using the `define Directive
- 9.2. Using the `ifdef Directive
- 9.3. Using the `include Directive
- 9.4. Using the `timescale Directive
- 9.5. Using the `begin\_keywords, `end\_keywords Directive
- 9.6. Using the `pragma Directives
- 9.7. Using the `default\_nettype Directives
- 9.8. Key Verilog Compiler Directives



## Verilog Coding - Synthesis (Member Free)

### 1. Logic Design Fundamentals

- 1.1. Basic Logic Gates
- 1.2. Full Adder Example
- 1.3. Simplifying Logic Functions with Boolean Algebra
- 1.4. Simplifying Logic Functions Using Karnaugh Maps
- 1.5. Designing with NAND and NOR Gates
- 1.6. Hazards in Combinational Circuits
- 1.7. Types of Flip-Flops
- 1.8. Mealy Sequential Circuit Design
- 1.9. Design of a Moore Sequential Circuit
- 1.10. State Equivalence and Optimization in Sequential Circuits
- 1.11. Key Timing Considerations in Sequential Circuits
- 1.12. Tristate Buffers and Their Role in Digital Circuits

### 2. Introduction to Verilog

- 2.1. Key Concepts in Digital Design
- 2.2. Hardware Description Languages (HDLs)
- 2.3. Levels of Abstraction in HDL Design
- 2.4. Benefits of Using HDL
- 2.5. Roles in HDL-Based Digital System Design
- 2.6. Challenges in Adopting HDL
- 2.7. Key Features of Verilog Language
- 2.8. Basics of Verilog Module Declaration
- 2.9. Verilog Module Hierarchy and Instantiation
- 2.10. Port Connection Syntax in Verilog
- 2.11. Connecting Ports in Module Instances
- 2.12. Using Procedural Constructs in Verilog
- 2.13. Synchronizing Module Behaviors
- 2.14. Naming Rules, Case Sensitivity, and Commenting in Verilog
- 2.15. Simulating HDL Designs: Compilation, Libraries, and Elaboration

### 3. Logic Synthesis with Verilog

- 3.1. Understanding Logic Synthesis
- 3.2. Benefits of Logic Synthesis
- 3.3. Using Synthesis Tools
- 3.4. Key Steps in Synthesis
- 3.5. Netlist Simulation Methods
- 3.6. Literal Logic Inference in Synthesis



- 3.7. Impact of Coding Style on Synthesis
- 3.8. Synthesis Challenges and Limitations
- 3.9. Technology-Specific Optimization
- 3.10. FPGA-Specific Synthesis Challenges
- 3.11. Synthesizable Verilog Constructs
- 3.12. Verilog Coding Style

## 4. Designing RTL Logic for Synthesis

- 4.1. Modeling Combinational Logic
- 4.2. Incomplete Event List
- 4.3. Complete Event List
- 4.4. Incomplete Assignments
- 4.5. Complete Assignments
- 4.6. Continuous Assignments
- 4.7. Modeling Combinational Logic Summary
- 4.8. Modeling Sequential Logic
- 4.9. Normal Behavior
- 4.10. Reset Behavior
- 4.11. Incomplete Assignments
- 4.12. Blocking vs. Nonblocking Assignments
- 4.13. Temporary vs. Persistent Variables
- 4.14. Modeling Sequential Logic Summary
- 4.15. Modeling Latch Logic
- 4.16. Modeling Three-State Logic
- 4.17. Using Synthesis Attributes
- 4.18. Pragma "full\_case" in Synthesis
- 4.19. Pragma "parallel\_case" in Synthesis
- 4.20. Pragma "implementation" in Synthesis
- 4.21. Pragma "sync\_set\_reset" or "async\_set\_reset" in Synthesis
- 4.22. Synthesis Attributes
- 4.23. Unsupported and Ignored Verilog Constructs

## 5. FSM Design for Synthesis

- 5.1. Finite State Machine (FSM) Overview
- 5.2. Defining FSM States
- 5.3. Read-Write Synchronizer FSM
- 5.4. One-Block FSM Coding
- 5.5. Two-Block FSM Coding
- 5.6. Three-Block FSM Coding
- 5.7. Three-Block FSM: Combinational Outputs
- 5.8. Optimizing Register Count



- 5.9. Optimizing Power and Noise: Gray Encoding
- 5.10. Optimizing Performance: One-Hot Encoding
- 5.11. State Bit Indexing in One-Hot Encoding

## 6. Preventing RTL and Netlist Mismatches

- 6.1. Preventing Indeterminate Behavior
- 6.2. Careful Use of Synthesis Attributes
- 6.3. Careful Use of Conditional Compilation
- 6.4. Ensure Complete Sensitivity List
- 6.5. Handling Temporary Variables Properly
- 6.6. Align RTL and Synthesis Models
- 6.7. Handling Unknown 'x' Values
- 6.8. casex vs. casez in Synthesis
- 6.9. Variable Declaration in Synthesis
- 6.10. Delay Controls in Synthesis

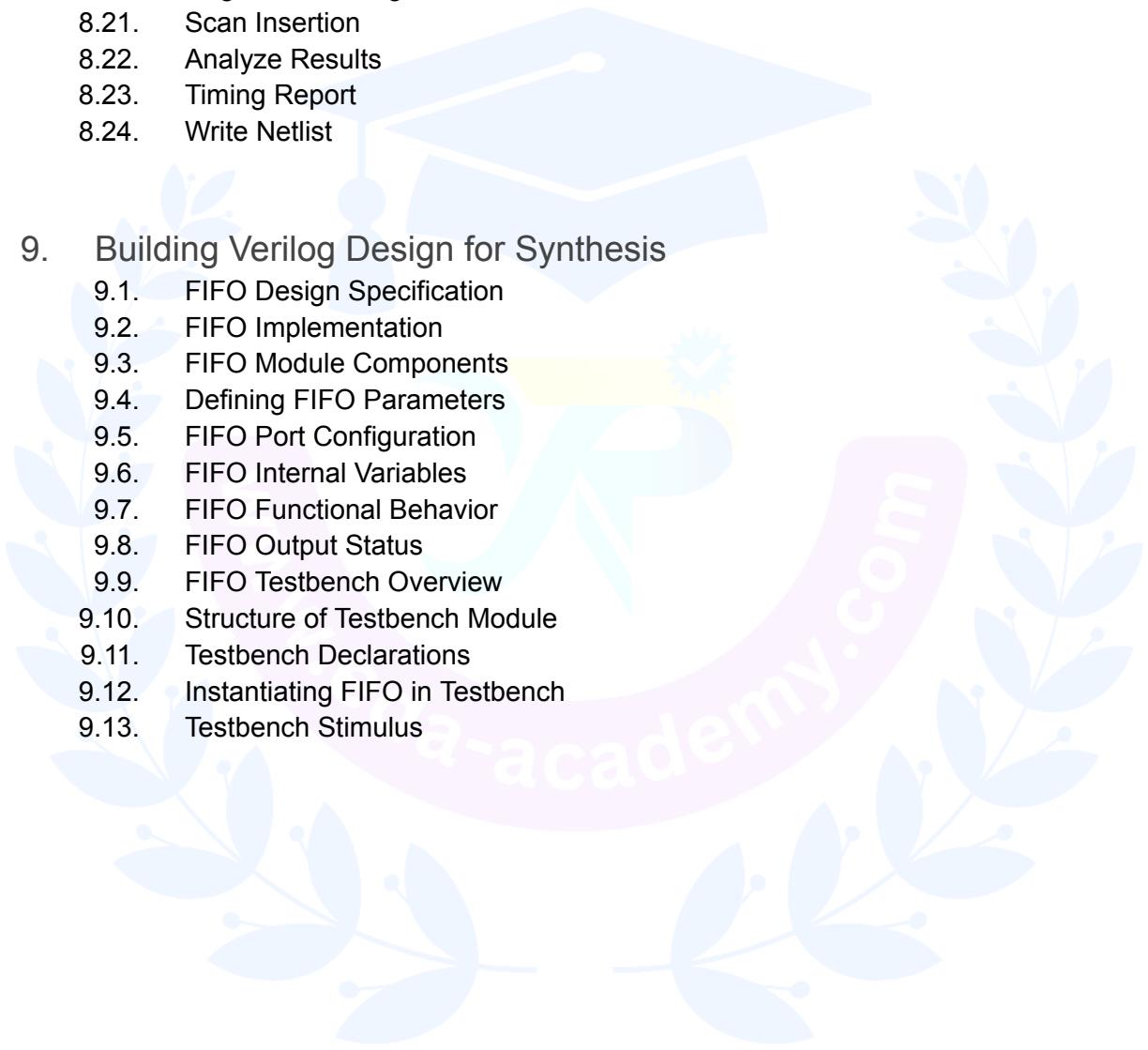
## 7. Managing the RTL Coding Process

- 7.1. Project Naming Conventions
- 7.2. Partitioning for Synthesis
- 7.3. Register Block Outputs
- 7.4. Group Combinational Logic
- 7.5. Keep Resources Together
- 7.6. Separate Auxiliary and Core Logic
- 7.7. Separate Blocks Needing Different Synthesis Techniques
- 7.8. Partitioning for Design Reuse
- 7.9. Coding RTL for Synthesis

## 8. Managing the Logic Synthesis Process

- 8.1. Goals of Logic Synthesis
- 8.2. Logic Synthesis Process Overview
- 8.3. Logic Synthesis Inputs and Outputs
- 8.4. Logic Synthesis Basic Flow
- 8.5. Reading HDL Source
- 8.6. Elaboration Process
- 8.7. Applying Constraints
- 8.8. Map to Generic Cells and Optimize
- 8.9. Pruning Unloaded Logic
- 8.10. Optimizing Arithmetic Expressions
- 8.11. Sharing Common Sub-Expressions
- 8.12. Balance Resource Sharing and Duplication



- 
- 8.13. Apply Carry-Save Adder (CSA) Transformations
  - 8.14. ChipWare Library and Synthesis
  - 8.15. Tradeoffs
  - 8.16. Manual Macrocell Instantiation
  - 8.17. Map to Technology Cells and Optimize
  - 8.18. Technology-Dependent Optimization
  - 8.19. Boundary Optimizations
  - 8.20. Register Retiming
  - 8.21. Scan Insertion
  - 8.22. Analyze Results
  - 8.23. Timing Report
  - 8.24. Write Netlist

## 9. Building Verilog Design for Synthesis

- 9.1. FIFO Design Specification
- 9.2. FIFO Implementation
- 9.3. FIFO Module Components
- 9.4. Defining FIFO Parameters
- 9.5. FIFO Port Configuration
- 9.6. FIFO Internal Variables
- 9.7. FIFO Functional Behavior
- 9.8. FIFO Output Status
- 9.9. FIFO Testbench Overview
- 9.10. Structure of Testbench Module
- 9.11. Testbench Declarations
- 9.12. Instantiating FIFO in Testbench
- 9.13. Testbench Stimulus



## Verilog Coding - Verification (Member Free)

### 1. Logic Design Fundamentals

- 1.1. Basic Logic Gates
- 1.2. Full Adder Example
- 1.3. Simplifying Logic Functions with Boolean Algebra
- 1.4. Simplifying Logic Functions Using Karnaugh Maps
- 1.5. Designing with NAND and NOR Gates
- 1.6. Hazards in Combinational Circuits
- 1.7. Types of Flip-Flops
- 1.8. Mealy Sequential Circuit Design
- 1.9. Design of a Moore Sequential Circuit
- 1.10. State Equivalence and Optimization in Sequential Circuits
- 1.11. Key Timing Considerations in Sequential Circuits
- 1.12. Tristate Buffers and Their Role in Digital Circuits

### 2. Introduction to Verilog

- 2.1. Key Concepts in Digital Design
- 2.2. Hardware Description Languages (HDLs)
- 2.3. Levels of Abstraction in HDL Design
- 2.4. Benefits of Using HDL
- 2.5. Roles in HDL-Based Digital System Design
- 2.6. Challenges in Adopting HDL
- 2.7. Key Features of Verilog Language
- 2.8. Basics of Verilog Module Declaration
- 2.9. Verilog Module Hierarchy and Instantiation
- 2.10. Port Connection Syntax in Verilog
- 2.11. Connecting Ports in Module Instances
- 2.12. Using Procedural Constructs in Verilog
- 2.13. Synchronizing Module Behaviors
- 2.14. Naming Rules, Case Sensitivity, and Commenting in Verilog
- 2.15. Simulating HDL Designs: Compilation, Libraries, and Elaboration

### 3. Verification Techniques

- 3.1. Key Points of Simulation in Design Verification
- 3.2. Evolution of RTL Simulation Techniques
- 3.3. Accelerators and Emulators in RTL Design
- 3.4. Debugging and Performance Profiling
- 3.5. Maximizing Regression Efficiency
- 3.6. Hardware Debugging and Simulation Performance



- 3.7. Simulation Sequencing in Verilog
- 3.8. Simulation Evaluation Methods
- 3.9. RTL Simulation Optimization Techniques I
- 3.10. RTL Simulation Optimization Techniques II
- 3.11. Register Initialization Methods
- 3.12. Simulating Tri-state Buses
- 3.13. Assertion Monitors in Two-State Simulation
- 3.14. Two-State Simulation Benefits

#### 4. Verification Strategies

- 4.1. Specification and Design Decomposition
- 4.2. RTL Implementation and Synthesis
- 4.3. Directed Testing
- 4.4. Random Testing
- 4.5. Transaction Analyzer
- 4.6. Chip Initialization Verification
- 4.7. Synthesizable Testbench Approach
- 4.8. Orthogonal Verification Principle
- 4.9. Ad-hoc and Programming Code Metrics
- 4.10. State Machine and User Defined Metrics
- 4.11. Fault Coverage and Regression Analysis
- 4.12. Event-Based Monitoring
- 4.13. Assertion Checkers

#### 5. Verilog Constructs for Verification

- 5.1. Data Types – real, time, and realtime
- 5.2. Case Equality Operators
- 5.3. Procedural Continuous Assignments – force and release
- 5.4. Loop Statements – while
- 5.5. Loop Statements – forever and repeat
- 5.6. Named Events – event and ->
- 5.7. Level-Sensitive Event Control – wait
- 5.8. Parallel Blocks: fork and join
- 5.9. Advanced Concepts of Parallel Blocks
- 5.10. initial and always Constructs
- 5.11. Disabling Tasks and Named Blocks – disable

#### 6. Design Behavior and RTL Modeling

- 6.1. Levels of Abstraction in Verilog
- 6.2. Comparing Behavioral and RTL Modeling



- 6.3. Bus Interface Controller Model
- 6.4. Bus Interface Controller Implementation
- 6.5. Combinational and Sequential Logic
- 6.6. Bus Interface Controller RTL
- 6.7. Testbench Types
- 6.8. Testbenches with File I/O

## 7. Using System Tasks and Functions

- 7.1. Displaying Messages – \$display and \$write
- 7.2. Formatting Text Output
- 7.3. Format Specifications
- 7.4. Escape Sequences in Format Strings
- 7.5. Displaying Messages – \$strobe
- 7.6. Getting Simulation Time – \$time, \$stime, \$realtime
- 7.7. Displaying Messages – \$monitor
- 7.8. Formatting the Time Display – \$timeformat
- 7.9. Opening Files – \$fopen
- 7.10. Writing to Files – \$fdisplay, \$fmonitor, \$fstrobe and \$fwrite
- 7.11. File Input – \$readmemb and \$readmemh
- 7.12. File Input Data Format
- 7.13. Enhanced C-Style File I/O I
- 7.14. Enhanced C-Style File I/O II
- 7.15. Simulation Control – \$finish and \$stop
- 7.16. Passing Real Numbers Through Ports
- 7.17. Getting Command-Line Values
- 7.18. Dumping Value-Change Data (VCD)
- 7.19. VCD Signal Selection – \$dumpvars
- 7.20. Dumping Extended Value-Change Data (EVCD)
- 7.21. Checkpointing the Simulation – \$save and \$restart

## 8. Generating Stimulus for Testing

- 8.1. Simulation Inputs and Outputs
- 8.2. Simulation Process
- 8.3. Organizing the Testbench
- 8.4. Using Hierarchical Names
- 8.5. Generating Clocks
- 8.6. Generating Stimulus
- 8.7. Incremental Testing Approach
- 8.8. Generating In-Line Stimulus
- 8.9. Generating Stimulus Using Loops
- 8.10. Generating Stimulus Using Tasks



- 8.11. Generating Random Stimulus
- 8.12. Common Random Distributions
- 8.13. Testing Boundary Conditions
- 8.14. Worst-Case Test Example – FIFO Model
- 8.15. Worst-Case Test Example – FIFO Test Tasks
- 8.16. Worst-Case Test Example – FIFO Test Sequence
- 8.17. Testing Protocol Interactions
- 8.18. Sweep Test Example – Task Definitions
- 8.19. Sweep Test Example – Test Sequence
- 8.20. Capturing and Replaying Vectors
- 8.21. Resolving Vector Capture and Replay Issues
- 8.22. Capturing Vectors to Files
- 8.23. Applying Vectors from Files
- 8.24. Vector Capture and Replay Example – Model
- 8.25. Vector Capture and Replay Example – Testbench
- 8.26. Vector Capture and Replay Example – Capture

## 9. Testbench Development and Application

- 9.1. Design Verification Challenge
- 9.2. Design Verification Plan
- 9.3. Design Verification Goal
- 9.4. System-Level Test and Configurability
- 9.5. System-Level Test and Communication
- 9.6. Completion Criteria
- 9.7. Coverage Metrics
- 9.8. Code Coverage Example – Arbiter Model and Test
- 9.9. Code Coverage Example – Block Coverage Report
- 9.10. Managing Design Space
- 9.11. Testing Design Regression
- 9.12. Self-Checking Test
- 9.13. Self-Checking Test – Example I
- 9.14. Self-Checking Test – Example II
- 9.15. Test Configuration Using Source Code Constructs
- 9.16. Test Configuration Using Source Code Constructs – Example
- 9.17. Test Configuration Using Run-Time Scripts
- 9.18. Test Configuration Using Run-Time Scripts – Example I
- 9.19. Test Configuration Using Run-Time Scripts – Example II
- 9.20. Test Configuration Using Microcode
- 9.21. Test Configuration Using Microcode – Example
- 9.22. Test Configuration Using the PLI
- 9.23. Test Configuration Using the PLI – Example
- 9.24. Script-Driven Testbench Overview
- 9.25. Script-Driven Testbench in Verilog-1995



9.26. Script-Driven Testbench in Verilog-2001



## Verilog Coding - Design (Member Free)

### 1. Logic Design Fundamentals

- 1.1. Basic Logic Gates
- 1.2. Full Adder Example
- 1.3. Simplifying Logic Functions with Boolean Algebra
- 1.4. Simplifying Logic Functions Using Karnaugh Maps
- 1.5. Designing with NAND and NOR Gates
- 1.6. Hazards in Combinational Circuits
- 1.7. Types of Flip-Flops
- 1.8. Mealy Sequential Circuit Design
- 1.9. Design of a Moore Sequential Circuit
- 1.10. State Equivalence and Optimization in Sequential Circuits
- 1.11. Key Timing Considerations in Sequential Circuits
- 1.12. Tristate Buffers and Their Role in Digital Circuits

### 2. Introduction to Verilog

- 2.1. Key Concepts in Digital Design
- 2.2. Hardware Description Languages (HDLs)
- 2.3. Levels of Abstraction in HDL Design
- 2.4. Benefits of Using HDL
- 2.5. Roles in HDL-Based Digital System Design
- 2.6. Challenges in Adopting HDL
- 2.7. Key Features of Verilog Language
- 2.8. Basics of Verilog Module Declaration
- 2.9. Verilog Module Hierarchy and Instantiation
- 2.10. Port Connection Syntax in Verilog
- 2.11. Connecting Ports in Module Instances
- 2.12. Using Procedural Constructs in Verilog
- 2.13. Synchronizing Module Behaviors
- 2.14. Naming Rules, Case Sensitivity, and Commenting in Verilog
- 2.15. Simulating HDL Designs: Compilation, Libraries, and Elaboration

### 3. Combinational Logic Design

- 3.1. Basic Gates Using Assign Statements
- 3.2. Majority Logic and Concatenation
- 3.3. Shift Operations
- 3.4. Multiplexer I
- 3.5. Multiplexer II
- 3.6. Demultiplexer I
- 3.7. Demultiplexer II



- 3.8. Decoder I
- 3.9. Decoder II
- 3.10. Decoder III
- 3.11. Encoder I
- 3.12. Encoder II
- 3.13. Comparator I
- 3.14. Comparator II

## 4. Sequential Logic Design

- 4.1. Metastability in Bistable Circuits
- 4.2. Basic NOR Latch Operation
- 4.3. SR Latch
- 4.4. D Latch I
- 4.5. D Latch II
- 4.6. D Flip-Flop I
- 4.7. D Flip-Flop II
- 4.8. D Flip-Flop III
- 4.9. Register I
- 4.10. Register II
- 4.11. Counter I
- 4.12. Counter II
- 4.13. Parallel to Serial Converter
- 4.14. Pattern Sequence Detector

## 5. Arithmetic Logic Design

- 5.1. Arithmetic Circuit Design in Verilog
- 5.2. Positional Number Representation
- 5.3. Binary Addition of Unsigned Numbers
- 5.4. Signed Numbers and Arithmetic Operations
- 5.5. Adder I
- 5.6. Adder II
- 5.7. Adder III
- 5.8. Adder IV
- 5.9. Adder V
- 5.10. Subtractor I
- 5.11. Subtractor II
- 5.12. Multiplier I
- 5.13. Multiplier II
- 5.14. Multiplier III
- 5.15. Divider I
- 5.16. Divider II



- 5.17. Fixed-Point Numbers
- 5.18. Floating-Point Numbers
- 5.19. Binary-Coded-Decimal Representation

## 6. FSM Logic Design

- 6.1. Finite-State Machines (FSM) Overview
- 6.2. State Machine Design
- 6.3. Components of FSM Logic Synthesis
- 6.4. FSM Design Process
- 6.5. FSM Coding - Modeling the States
- 6.6. FSM Coding - State Memory Block
- 6.7. FSM Coding - Next State Logic Block
- 6.8. FSM Coding - Output Logic Block
- 6.9. FSM Example - Serial Bit Sequence Detector I
- 6.10. FSM Example - Serial Bit Sequence Detector II
- 6.11. FSM Example - Vending Machine Controller I
- 6.12. FSM Example - Vending Machine Controller II
- 6.13. FSM Example - 2-Bit Binary Up/Down Counter I
- 6.14. FSM Example - 2-Bit Binary Up/Down Counter II

## 7. Synchronous Logic Design

- 7.1. Synchronous Sequential Machines
- 7.2. Synthesis Procedure
- 7.3. Equivalent States
- 7.4. Moore Machines Overview
- 7.5. Moore Machines Example 1
- 7.6. Moore Machines Example 2
- 7.7. Moore Machines Example 3
- 7.8. Moore Machines Example 4
- 7.9. Moore Machines Example 5
- 7.10. Moore Machines Example 6
- 7.11. Mealy Machines Overview
- 7.12. Mealy Machines Example 1
- 7.13. Mealy Machines Example 2
- 7.14. Mealy Machines Example 3
- 7.15. Mealy Machines Example 4
- 7.16. Mealy Machines Example 5
- 7.17. Synchronous Registers Overview
- 7.18. Synchronous Registers Example 1
- 7.19. Synchronous Registers Example 2
- 7.20. Synchronous Registers Example 3



- 7.21. Synchronous Registers Example 4
- 7.22. Synchronous Counters Overview
- 7.23. Synchronous Counters Example 1
- 7.24. Synchronous Counters Example 2
- 7.25. Synchronous Counters Example 3

## 8. Asynchronous Logic Design

- 8.1. Asynchronous Sequential Machines
- 8.2. Synthesis of Asynchronous Machines
- 8.3. Understanding Hazards in Circuits
- 8.4. Oscillations in Asynchronous Machines
- 8.5. Races in Sequential Machines
- 8.6. Asynchronous Design Example 1 I
- 8.7. Asynchronous Design Example 1 II
- 8.8. Asynchronous Design Example 1 III
- 8.9. Asynchronous Design Example 2 I
- 8.10. Asynchronous Design Example 2 II
- 8.11. Asynchronous Design Example 3 I
- 8.12. Asynchronous Design Example 3 II
- 8.13. Asynchronous Design Example 4 I
- 8.14. Asynchronous Design Example 4 II

## 9. Digital System Design

- 9.1. Bus Structure I
- 9.2. Bus Structure II
- 9.3. Bus Structure III
- 9.4. Simple Processor I
- 9.5. Simple Processor II
- 9.6. Bit Counter I
- 9.7. Bit Counter II
- 9.8. Bit Counter III
- 9.9. Bit Counter IV
- 9.10. Shift-and-Add Multiplier I
- 9.11. Shift-and-Add Multiplier II
- 9.12. Shift-and-Add Multiplier III
- 9.13. Shift-and-Add Multiplier IV
- 9.14. Enhanced Divider I
- 9.15. Enhanced Divider II
- 9.16. Enhanced Divider III
- 9.17. Enhanced Divider IV



# SystemVerilog Language Series

## SystemVerilog Language - Design (Member Free)

### 1. Introduction to SystemVerilog

- 1.1. Evolution of Verilog and SystemVerilog
- 1.2. Original Verilog
- 1.3. Verilog vs. VHDL Development
- 1.4. Verilog-95 and Verilog-2001
- 1.5. SystemVerilog extensions to Verilog
- 1.6. SystemVerilog Replaces Verilog
- 1.7. Simplified and Synthesizable RTL Design
- 1.8. Data Type Enhancements in SystemVerilog
- 1.9. Key Building Blocks of SystemVerilog Verification
- 1.10. Key Features of SystemVerilog Verification
- 1.11. Simplifying Design with Interfaces
- 1.12. Randomization in SystemVerilog
- 1.13. Functional Coverage in SystemVerilog
- 1.14. Assertions in SystemVerilog
- 1.15. Direct Programming Interface in SystemVerilog

### 2. SystemVerilog Data Types

- 2.1. SystemVerilog Logic vs Reg
- 2.2. Verilog Data Type Rules
- 2.3. Verilog Data Type Example
- 2.4. SystemVerilog Data Type Rules
- 2.5. SystemVerilog Data Type Example
- 2.6. Variable Assignment Restrictions
- 2.7. Multiple Procedural Assignments
- 2.8. 2-State Data Types
- 2.9. 2-State vs 4-State Logic
- 2.10. Unsized Literals
- 2.11. Time Literals
- 2.12. Using Time Units in Assignments
- 2.13. timeunit and timeprecision Declarations

### 3. Procedures and Procedural Statements

- 3.1. Naming and Labeling Code Blocks
- 3.2. Declaring Variables in Unnamed Blocks
- 3.3. Enhanced for Loop in SystemVerilog



- 3.4. do ... while Loop
- 3.5. foreach Loop
- 3.6. break and continue Statements
- 3.7. Verilog case Statement
- 3.8. SystemVerilog priority case
- 3.9. SystemVerilog unique case
- 3.10. SystemVerilog priority if and unique if
- 3.11. Using iff in Event Control
- 3.12. Procedural Block Types in SystemVerilog
- 3.13. Combinational Blocks with always\_comb
- 3.14. always\_comb vs. always @\*
- 3.15. Latch Blocks with always\_latch
- 3.16. Register Blocks with always\_ff

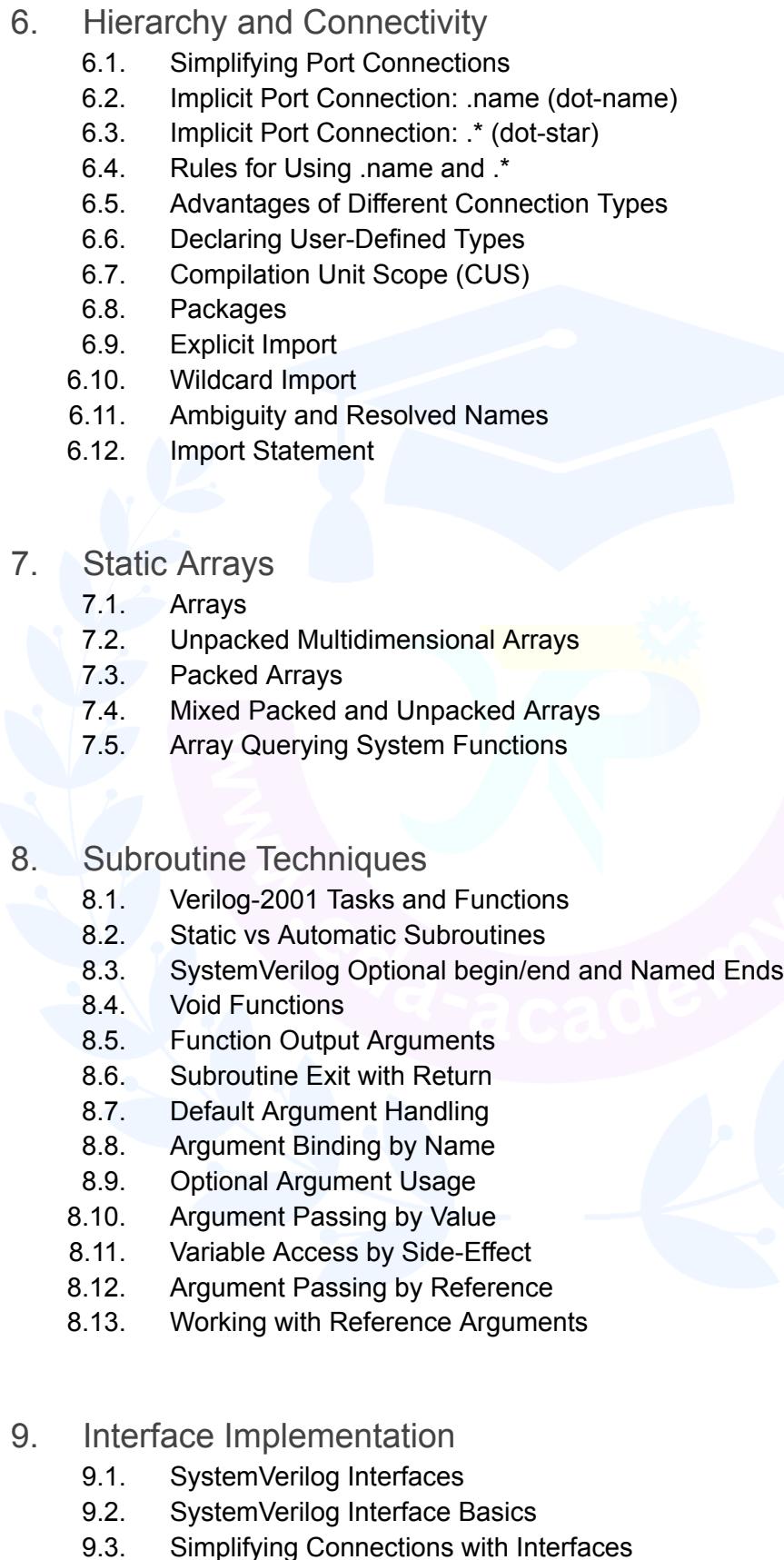
#### 4. Applying SystemVerilog Operators

- 4.1. Assignment Operators in SystemVerilog
- 4.2. Using Assignment Operators
- 4.3. Pre- and Post-Increment/Decrement Operators
- 4.4. Assignment Patterns
- 4.5. Wildcard Equivalence Operators
- 4.6. Set Membership Operator
- 4.7. Using inside Operator with case
- 4.8. Operator Precedence and Associativity
- 4.9. Array Assignment Patterns

#### 5. Custom Data Types

- 5.1. Primitive and User-Defined Types
- 5.2. Defining Types with typedef
- 5.3. Enumerated Types
- 5.4. Enumerated Type Values
- 5.5. Explicit Enumerate Value Encoding
- 5.6. Specifying Enumerate Base Type
- 5.7. Enumerated Type Value Name Sequences
- 5.8. Enumerated Type Access Methods
- 5.9. Enumerated Type Initialization
- 5.10. Structures
- 5.11. Keyed Pattern Assignments
- 5.12. Packed Structures
- 5.13. New Type Declaration Regions



- 
- 6. Hierarchy and Connectivity**
    - 6.1. Simplifying Port Connections
    - 6.2. Implicit Port Connection: .name (dot-name)
    - 6.3. Implicit Port Connection: .\* (dot-star)
    - 6.4. Rules for Using .name and .\*
    - 6.5. Advantages of Different Connection Types
    - 6.6. Declaring User-Defined Types
    - 6.7. Compilation Unit Scope (CUS)
    - 6.8. Packages
    - 6.9. Explicit Import
    - 6.10. Wildcard Import
    - 6.11. Ambiguity and Resolved Names
    - 6.12. Import Statement
  - 7. Static Arrays**
    - 7.1. Arrays
    - 7.2. Unpacked Multidimensional Arrays
    - 7.3. Packed Arrays
    - 7.4. Mixed Packed and Unpacked Arrays
    - 7.5. Array Querying System Functions
  - 8. Subroutine Techniques**
    - 8.1. Verilog-2001 Tasks and Functions
    - 8.2. Static vs Automatic Subroutines
    - 8.3. SystemVerilog Optional begin/end and Named Ends
    - 8.4. Void Functions
    - 8.5. Function Output Arguments
    - 8.6. Subroutine Exit with Return
    - 8.7. Default Argument Handling
    - 8.8. Argument Binding by Name
    - 8.9. Optional Argument Usage
    - 8.10. Argument Passing by Value
    - 8.11. Variable Access by Side-Effect
    - 8.12. Argument Passing by Reference
    - 8.13. Working with Reference Arguments
  - 9. Interface Implementation**
    - 9.1. SystemVerilog Interfaces
    - 9.2. SystemVerilog Interface Basics
    - 9.3. Simplifying Connections with Interfaces



- 9.4. Challenges of Bus Connections Without Interfaces
- 9.5. Bus Connections Using Interfaces
- 9.6. Interface Port Mapping Rules
- 9.7. Accessing Interface Ports
- 9.8. Accessing Interface Instances
- 9.9. Define Interface Ports
- 9.10. Interface Port Applications
- 9.11. Interface Parameterization
- 9.12. Modports in Interfaces
- 9.13. Modport Selection in Module Declaration
- 9.14. Modport Selection in Module Instantiation
- 9.15. Shared Interface Tasks
- 9.16. Interface Methods
- 9.17. Modport Interface Methods
- 9.18. Using Generic Interfaces



# SystemVerilog Language - Verification (Member Free)

1. Introduction to SystemVerilog
  - 1.1. Evolution of Verilog and SystemVerilog
  - 1.2. Original Verilog
  - 1.3. Verilog vs. VHDL Development
  - 1.4. Verilog-95 and Verilog-2001
  - 1.5. SystemVerilog extensions to Verilog
  - 1.6. SystemVerilog Replaces Verilog
  - 1.7. Simplified and Synthesizable RTL Design
  - 1.8. Data Type Enhancements in SystemVerilog
  - 1.9. Key Building Blocks of SystemVerilog Verification
  - 1.10. Key Features of SystemVerilog Verification
  - 1.11. Simplifying Design with Interfaces
  - 1.12. Randomization in SystemVerilog
  - 1.13. Functional Coverage in SystemVerilog
  - 1.14. Assertions in SystemVerilog
  - 1.15. Direct Programming Interface in SystemVerilog
2. Basic Verification Features
  - 2.1. SystemVerilog Strings Overview
  - 2.2. String Operators
  - 2.3. String Methods I
  - 2.4. String Methods II
  - 2.5. Immediate Assertions
  - 2.6. Action Blocks in Assertions
  - 2.7. Assertion Severity Levels
  - 2.8. Immediate Assertion for Simple Protocol Check
  - 2.9. Immediate vs Concurrent Assertions
  - 2.10. Enhancements to Fork-Join Behavior
  - 2.11. Process Control: disable fork, wait fork
  - 2.12. Fine-Grain Process Control
  - 2.13. Useful Design Features in Verification
  - 2.14. SystemVerilog Unions
  - 2.15. Using Structures with Unions
3. Verification Blocks
  - 3.1. Verilog Event Scheduling
  - 3.2. SystemVerilog Event Scheduler
  - 3.3. SystemVerilog Program Block
  - 3.4. SystemVerilog Final Blocks



- 3.5. Clocking Block
- 3.6. Clocking Block Syntax
- 3.7. Clocking Block Output Drive
- 3.8. Clocking Block Input Sampling
- 3.9. Input and Output Skew in Clocking Blocks
- 3.10. Example: Clocking Block with Skews
- 3.11. Example: Output Skew Synchronization
- 3.12. Clocking Block Options and Defaults
- 3.13. Cycle Delay in Clocking Blocks
- 3.14. Using Hierarchical Expressions

#### 4. Stimulus Generation with Randomization

- 4.1. CPU Coverage Test Case
- 4.2. CPU Test Stimulus Setup
- 4.3. Looped CPU Instruction Stimulus
- 4.4. Limitations of Structured Stimulus
- 4.5. Randomization and Constraints
- 4.6. Pseudo-Random Number Generators
- 4.7. Unsigned Random Number Functions
- 4.8. Using randomize() for Scope Variables
- 4.9. Ensuring Random Stability
- 4.10. Setting the Random Seed
- 4.11. Managing Thread Seeding
- 4.12. Using Constraint Blocks
- 4.13. Conditional Constraints
- 4.14. Random Weighted Case: randcase
- 4.15. Random Sequence Generation: randsequence
- 4.16. Random Production Weights
- 4.17. Production Statements: case, if-else, repeat
- 4.18. Aborting Productions: break, return
- 4.19. Passing Values in Productions

#### 5. Class Fundamentals in SystemVerilog

- 5.1. Class Overview in SystemVerilog
- 5.2. Variables of the Class Type
- 5.3. Class Handles and Memory Safety
- 5.4. Class Properties and Methods
- 5.5. External Method Declaration
- 5.6. Class Constructor
- 5.7. Example: Class Definition and Instantiation
- 5.8. Static Properties



- 5.9. Static Methods
- 5.10. Current Object Handle: this
- 5.11. Class Properties as Class Instances
- 5.12. Aggregation vs. Inheritance
- 5.13. Example: Simple Inheritance
- 5.14. Example: Inheritance and Constructors
- 5.15. Multi-Layer Inheritance
- 5.16. Data Encapsulation in Classes
- 5.17. Class Parameters

## 6. Polymorphism and Virtuality

- 6.1. Polymorphism in SystemVerilog
- 6.2. Copy a Sub-Class Instance to a Parent Handle
- 6.3. Copy a Parent Instance to a Sub-Class Handle
- 6.4. \$cast
- 6.5. Advantages of Polymorphism
- 6.6. Accessing Class Members with Polymorphism
- 6.7. Virtual Method Resolution
- 6.8. Class Method Resolution
- 6.9. Virtual Classes and Pure Virtual Methods

## 7. Class-Based Stimulus Generation

- 7.1. Random Class Properties
- 7.2. Randomizing Class Objects: randomize(), pre\_randomize() and post\_randomize()
- 7.3. Randomization in Aggregate Classes
- 7.4. In-Line Random Variable Control
- 7.5. Controlling Randomization: rand\_mode()
- 7.6. Example: rand\_mode()
- 7.7. Constraint Blocks
- 7.8. Constraint Block Inheritance
- 7.9. Constraint Expressions: Set Membership
- 7.10. Constraint Expressions: Weighted Distributions
- 7.11. Constraint Expressions: Conditional Constraints
- 7.12. Constraint Expressions: Iterative Constraints
- 7.13. Controlling Constraints: constraint\_mode()
- 7.14. Example: Application of constraint\_mode()
- 7.15. Randomization Procedure
- 7.16. Randomization Ordering and Solution Probability
- 7.17. Setting the Random Seed: srand()



## 8. Functional Coverage with Covergroups

- 8.1. Structural and Functional Coverage
- 8.2. Types of Functional Coverage
- 8.3. Data-Oriented Functional Coverage
- 8.4. Defining a Coverage Model
- 8.5. Automatic Coverage Bins
- 8.6. Defining Explicit Bins
- 8.7. Explicit Scalar and Vector Bins
- 8.8. Example: Explicit Scalar and Vector Bins
- 8.9. Counting Bins in a Covergroup
- 8.10. Cross Coverage
- 8.11. Automatic Cross Bins Example
- 8.12. Explicit Cross Bins and Selections
- 8.13. Ignored and Illegal Cross Bins
- 8.14. Defining Easier Cover Cross Bins
- 8.15. Class-Based Coverage Model
- 8.16. Tracking Transitions in Coverage
- 8.17. Specify Coverage Options
- 8.18. Reference: Type-Specific type\_option Fields
- 8.19. Reference: Instance-Specific option Fields
- 8.20. Reference: Covergroup Methods

## 9. Queues and Dynamic and Associative Arrays (QDA)

- 9.1. Dynamic Arrays
- 9.2. Dynamic Array Example
- 9.3. Associative Arrays
- 9.4. Associative Array Methods
- 9.5. Example: Associative Array Lookup Table
- 9.6. Example: Associative Array-More Efficient Lookup Table
- 9.7. Queues
- 9.8. Queue Methods
- 9.9. Example: Queue Methods
- 9.10. Example: Queue Indexing
- 9.11. Array Manipulation Methods
- 9.12. Reference: Array Locator Methods
- 9.13. Reference: Array Ordering Methods
- 9.14. Reference: Array Reduction Methods



## SystemVerilog Language - Advanced (Member Free)

### 1. SystemVerilog Miscellaneous Features

- 1.1. Final Blocks: final
- 1.2. Understanding Unions
- 1.3. Union Example
- 1.4. Fine-Grain Process Control
- 1.5. Instance Tree Root: \$root
- 1.6. Module References in SystemVerilog
- 1.7. Nested Modules
- 1.8. Unsigned Random Numbers
- 1.9. Random Sequence Generation: randsequence
- 1.10. Random Production Weights
- 1.11. Production Statements: case, if-else, repeat
- 1.12. Aborting Productions: break, return
- 1.13. Passing Values between Productions

### 2. SystemVerilog 2012 Features

- 2.1. SystemVerilog Single Inheritance
- 2.2. SystemVerilog Interface Classes
- 2.3. Interface Classes
- 2.4. Rules for Interface Classes
- 2.5. Interface Class - Type Access
- 2.6. Interface Class Inheritance and Implementation I
- 2.7. Interface Class Inheritance and Implementation II
- 2.8. Interface Class Inheritance and Implementation III
- 2.9. Hard Constraints
- 2.10. SystemVerilog 2012 Soft Constraints
- 2.11. Soft Constraint Priority Rules
- 2.12. Soft Constraints and the Constraint Solver
- 2.13. Uniqueness Constraints
- 2.14. Uniqueness Constraints Limitations

### 3. Transaction Based Verification

- 3.1. What is a Transaction
- 3.2. Class-Based Testbenches
- 3.3. Transaction Based Verification Structure
- 3.4. Transactor Structure
- 3.5. Simple Transactor with Interfaces
- 3.6. The Testbench Drives the Transactor
- 3.7. Defining Views with Modports



- 3.8. Using Clocking Blocks in Interfaces
- 3.9. Core Verification Component (VC) Architecture
- 3.10. Virtual Interface Connections
- 3.11. Virtual Interface Module Example
- 3.12. Virtual Interface Class Example
- 3.13. Virtual Interface Limitations
- 3.14. Benefits of Transaction Based Verification (TBV)

## 4. Functional Coverage Modeling

- 4.1. Metric-Driven Verification Review
- 4.2. Metric-Driven Verification Flow
- 4.3. Benefits of Metric-Driven Verification
- 4.4. Key Coverage Considerations
- 4.5. Coverage Options Overview
- 4.6. Metric-Driven Verification Goals
- 4.7. Integrating Coverage Techniques
- 4.8. Explicit Coverage in SystemVerilog
- 4.9. Coverage Placement Strategies
- 4.10. Interface Monitor Coverage Declaration
- 4.11. Interface Monitor Coverage Trigger
- 4.12. Module Monitor Coverage
- 4.13. When to Cover
- 4.14. Comparing With and Without MDV

## 5. Interprocess Synchronization

- 5.1. Blocking Event Trigger
- 5.2. Nonblocking Event Trigger
- 5.3. Persistent Event Trigger: triggered
- 5.4. Event Sequencing: wait\_order
- 5.5. Mailboxes
- 5.6. Mailbox Methods
- 5.7. Process Synchronization with a Mailbox
- 5.8. Typeless Mailbox
- 5.9. Mailbox Parameters
- 5.10. Semaphores
- 5.11. Semaphore Methods
- 5.12. Semaphore-Based Process Synchronization
- 5.13. Event Variables
- 5.14. Merging Events
- 5.15. Reclaiming Events



- 
- 6. Direct Programming Interface (DPI)**
    - 6.1. The Verilog PLI
    - 6.2. The SystemVerilog DPI
    - 6.3. DPI Characteristics
    - 6.4. SystemVerilog to C Data Type Mapping I
    - 6.5. SystemVerilog to C Data Type Mapping II
    - 6.6. Importing C Functions and Tasks
    - 6.7. Ensuring Data Type Compatibility
    - 6.8. Import Linkage Name
    - 6.9. Task and Function Import in SystemVerilog
    - 6.10. Context Tasks and Functions
    - 6.11. Pure Functions and Simulation Optimization
    - 6.12. Exporting Tasks and Functions
    - 6.13. Export Linkage Name
    - 6.14. Aspects of Task and Function Export
    - 6.15. Imported and Exported Functions Example
    - 6.16. Compilation Options for Linked C Code
    - 6.17. DPI Advantages and Limitations
    - 6.18. Disable Handshake Protocol
    - 6.19. Disable Handshake for Functions
    - 6.20. Disable Handshake for Tasks
  - 7. Assertion Based Verification (ABV)**
    - 7.1. Specifying Properties
    - 7.2. What is an Assertion
    - 7.3. Defining Assertion
    - 7.4. Assertions Monitor Design Properties
    - 7.5. What are Assertions used for
    - 7.6. What aren't Assertions used for
    - 7.7. Why to Use Assertions
    - 7.8. Who to Write Assertions
    - 7.9. Where to Use Assertions
    - 7.10. Issues with Assertions I
    - 7.11. Issues with Assertions II
  - 8. Introduction to SystemVerilog Assertions (SVA)**
    - 8.1. Concurrent Assertions
    - 8.2. Structure of Concurrent Assertions
    - 8.3. Defining a Simple Property
    - 8.4. Naming and Asserting Property
    - 8.5. Clocking and Evaluating Properties



- 8.6. Assertion and Property Evaluation
- 8.7. Example of Assertion Evaluation
- 8.8. Assertion Placement in SystemVerilog
- 8.9. Sequences
- 8.10. Same Cycle Sequence Implication:  $|-\>$
- 8.11. Next Cycle Sequence Implication:  $|=>$
- 8.12. Sequence Property Analysis
- 8.13. Disabling Properties
- 8.14. Assertion Status
- 8.15. Cycle Delay Repetition
- 8.16. Consecutive Sequence Repetition
- 8.17. Consecutive Repetition with Ranges

## 9. Static Verification

- 9.1. Formal Verification Methods
- 9.2. Static vs. Dynamic Assertion-Based Verification
- 9.3. Property Checking Overview
- 9.4. Property Checking and Simulation Synergy
- 9.5. Using Property Checking
- 9.6. Verifying Property Behaviors
- 9.7. Using Input Constraints
- 9.8. Applying Constraints: Scan Mode
- 9.9. Benefits of Property Checking
- 9.10. Case Study with Property Checking
- 9.11. Benefits of Adding Assertions



# SystemVerilog Language - Testbench (Member Free)

## 1. Introduction to SystemVerilog

- 1.1. Evolution of Verilog and SystemVerilog
- 1.2. Original Verilog
- 1.3. Verilog vs. VHDL Development
- 1.4. Verilog-95 and Verilog-2001
- 1.5. SystemVerilog extensions to Verilog
- 1.6. SystemVerilog Replaces Verilog
- 1.7. Simplified and Synthesizable RTL Design
- 1.8. Data Type Enhancements in SystemVerilog
- 1.9. Key Building Blocks of SystemVerilog Verification
- 1.10. Key Features of SystemVerilog Verification
- 1.11. Simplifying Design with Interfaces
- 1.12. Randomization in SystemVerilog
- 1.13. Functional Coverage in SystemVerilog
- 1.14. Assertions in SystemVerilog
- 1.15. Direct Programming Interface in SystemVerilog

## 2. Data Types, Literals, and Packaging

- 2.1. Logic Type as a Replacement for Verilog reg
- 2.2. Data Types in SystemVerilog
- 2.3. 2-State and 4-State Value Objects
- 2.4. User-Defined Types with typedef
- 2.5. Enumeration Data Types
- 2.6. SystemVerilog Structures
- 2.7. Packed Structures
- 2.8. SystemVerilog Strings
- 2.9. String Operators
- 2.10. SystemVerilog Time Literals
- 2.11. Using Time Units in Assignments
- 2.12. SystemVerilog Packages
- 2.13. Explicit and Wildcard Imports
- 2.14. Ambiguity and Resolved Names
- 2.15. Import Placement
- 2.16. Randomizing Scope Variables
- 2.17. Defining Constraint Blocks
- 2.18. Random Weighted Case: randcase

## 3. Tasks and Functions

- 3.1. Verilog-2001 Tasks and Functions



- 3.2. Static vs Automatic Subroutines
- 3.3. SystemVerilog Optional begin/end and Named Ends
- 3.4. Void Functions
- 3.5. Function Output Arguments
- 3.6. Subroutine Exit with Return
- 3.7. Default Argument Handling
- 3.8. Argument Binding by Name
- 3.9. Optional Argument Usage
- 3.10. Argument Passing by Value
- 3.11. Variable Access by Side-Effect
- 3.12. Argument Passing by Reference
- 3.13. Working with Reference Arguments

#### 4. Interfaces

- 4.1. SystemVerilog Interfaces
- 4.2. SystemVerilog Interface Basics
- 4.3. Simplifying Connections with Interfaces
- 4.4. Challenges of Bus Connections Without Interfaces
- 4.5. Bus Connections Using Interfaces
- 4.6. Interface Port Mapping Rules
- 4.7. Accessing Interface Ports
- 4.8. Accessing Interface Instances
- 4.9. Define Interface Ports
- 4.10. Interface Port Applications
- 4.11. Interface Parameterization
- 4.12. Modports in Interfaces
- 4.13. Modport Selection in Module Declaration
- 4.14. Modport Selection in Module Instantiation
- 4.15. Shared Interface Tasks
- 4.16. Interface Methods
- 4.17. Modport Interface Methods
- 4.18. Using Generic Interfaces

#### 5. Queues, Dynamic Arrays, and Associative Arrays

- 5.1. Packed vs. Unpacked Arrays
- 5.2. Understanding Unpacked Arrays
- 5.3. Packed Array Fundamentals
- 5.4. Dynamic Arrays
- 5.5. Dynamic Array Example
- 5.6. Associative Arrays
- 5.7. Associative Array Methods



- 5.8. Associative Array Lookup Table I
- 5.9. Associative Array Lookup Table II
- 5.10. Queues
- 5.11. Queue Methods
- 5.12. Queue Methods Example
- 5.13. Array Manipulation Methods
- 5.14. Array Locator Methods
- 5.15. Comparison of Array Operations

## 6. Classes

- 6.1. Class Overview in SystemVerilog
- 6.2. Variables of the Class Type
- 6.3. Class Handles and Memory Safety
- 6.4. Class Properties and Methods
- 6.5. External Method Declaration
- 6.6. Class Constructor
- 6.7. Example: Class Definition and Instantiation
- 6.8. Static Properties
- 6.9. Static Methods
- 6.10. Current Object Handle: this
- 6.11. Class Properties as Class Instances
- 6.12. Aggregation vs. Inheritance
- 6.13. Example: Simple Inheritance
- 6.14. Example: Inheritance and Constructors
- 6.15. Multi-Layer Inheritance
- 6.16. Data Encapsulation in Classes
- 6.17. Class Parameters

## 7. Class-Based Random Stimulus

- 7.1. Random Class Properties
- 7.2. Randomizing Class Objects: randomize(), pre\_randomize() and post\_randomize()
- 7.3. Randomization in Aggregate Classes
- 7.4. In-Line Random Variable Control
- 7.5. Controlling Randomization: rand\_mode()
- 7.6. Example: rand\_mode()
- 7.7. Constraint Blocks
- 7.8. Constraint Block Inheritance
- 7.9. Constraint Expressions: Set Membership
- 7.10. Constraint Expressions: Weighted Distributions
- 7.11. Constraint Expressions: Conditional Constraints



- 7.12. Constraint Expressions: Iterative Constraints
- 7.13. Controlling Constraints: constraint\_mode()
- 7.14. Example: Application of constraint\_mode()
- 7.15. Randomization Procedure
- 7.16. Randomization Ordering and Solution Probability
- 7.17. Setting the Random Seed: random()

## 8. Covergroup Coverage

- 8.1. Structural and Functional Coverage
- 8.2. Types of Functional Coverage
- 8.3. Data-Oriented Functional Coverage
- 8.4. Defining a Coverage Model
- 8.5. Automatic Coverage Bins
- 8.6. Defining Explicit Bins
- 8.7. Explicit Scalar and Vector Bins
- 8.8. Example: Explicit Scalar and Vector Bins
- 8.9. Counting Bins in a Covergroup
- 8.10. Cross Coverage
- 8.11. Automatic Cross Bins Example
- 8.12. Explicit Cross Bins and Selections
- 8.13. Ignored and Illegal Cross Bins
- 8.14. Defining Easier Cover Cross Bins
- 8.15. Class-Based Coverage Model
- 8.16. Tracking Transitions in Coverage
- 8.17. Specify Coverage Options
- 8.18. Reference: Type-Specific type\_option Fields
- 8.19. Reference: Instance-Specific option Fields
- 8.20. Reference: Covergroup Methods

## 9. Interprocess Synchronization

- 9.1. Blocking Event Trigger
- 9.2. Nonblocking Event Trigger
- 9.3. Persistent Event Trigger: triggered
- 9.4. Event Sequencing: wait\_order
- 9.5. Mailboxes
- 9.6. Mailbox Methods
- 9.7. Process Synchronization with a Mailbox
- 9.8. Typeless Mailbox
- 9.9. Mailbox Parameters
- 9.10. Semaphores
- 9.11. Semaphore Methods



- 9.12. Semaphore-Based Process Synchronization
- 9.13. Event Variables
- 9.14. Merging Events
- 9.15. Reclaiming Events



# SystemVerilog Language - Coverage (Member Free)

## 1. Introduction to Coverage

- 1.1. Coverage Goals
- 1.2. Types of Coverage
- 1.3. Code Coverage Essentials
- 1.4. Functional Coverage for Data Signals
- 1.5. Functional Coverage for Control Signals
- 1.6. FSM (Finite State Machine) Coverage
- 1.7. Transaction Coverage
- 1.8. The Coverage Process
- 1.9. Step 1: Instrument the Design
- 1.10. Step 2: Collect the Coverage Data
- 1.11. Step 3: Reduce the Coverage Data
- 1.12. Coverage Collection & Reduction Overview
- 1.13. Step 4: Analyze the Coverage Data
- 1.14. Comprehensive Coverage Methods

## 2. Code Coverage

- 2.1. Block Coverage Basics
- 2.2. Code Block Definition
- 2.3. Rules for Code Blocks
- 2.4. How Many Code Blocks
- 2.5. Understanding Code Branches
- 2.6. Understanding Expression Coverage
- 2.7. Understanding Toggle Coverage
- 2.8. Default Toggle Coverage Constructs
- 2.9. Understanding FSM Coverage
- 2.10. Guidelines for FSM Extraction
- 2.11. Accepted FSM Coding Guidelines
- 2.12. One-Block FSM Coding Example
- 2.13. Two-Block FSM Coding Example
- 2.14. One-Hot FSM Coding Example

## 3. Data-Oriented Function Coverage

- 3.1. Data-Oriented Functional Coverage Overview
- 3.2. Defining a Covergroup
- 3.3. Instantiating a Covergroup
- 3.4. Defining Coverpoints
- 3.5. Automatically Created Coverpoint Bins
- 3.6. Defining Coverpoint Value Bins



- 3.7. Example Defining Coverpoint Value Bins
- 3.8. Defining Coverpoint Transition Bins
- 3.9. Example Defining Coverpoint Transition Bins
- 3.10. Defining Coverpoint Crosses
- 3.11. Automatically Created Cross Bins
- 3.12. Defining Cross Bins
- 3.13. Example Defining Cross Bins
- 3.14. Using Covergroups in Classes
- 3.15. Example Using Covergroups in Classes

## 4. Control-Oriented Function Coverage

- 4.1. Control-Oriented Functional Coverage Overview
- 4.2. Key Verification Terms and Definitions
- 4.3. Directing Functional Verification Tools
- 4.4. Specifying Action Blocks in Assertions
- 4.5. Composing Boolean Expressions
- 4.6. Boolean Functions
- 4.7. Composing Sequence Expressions
- 4.8. Clocking Sequences
- 4.9. Sequence Fusion and Concatenation
- 4.10. Defining Cycle Delay Ranges
- 4.11. Defining Sequence Operations
- 4.12. Defining Sequence Repetition: Consecutive
- 4.13. Defining Sequence Repetition: Non-Consecutive
- 4.14. Defining Sequence Repetition: Go-To
- 4.15. Declaring and Instantiating Sequences
- 4.16. Using Local Variables in Sequences
- 4.17. Overview of SVA Sequence Operators
- 4.18. Example SVA Sequences
- 4.19. Composing Property Expressions
- 4.20. Weak and Strong Properties
- 4.21. Declaring and Instantiating Properties
- 4.22. Overview of SVA Property Operators
- 4.23. Example SVA Properties

## 5. Functional Coverage Modeling

- 5.1. Metric-Driven Verification Review
- 5.2. Metric-Driven Verification Flow
- 5.3. Benefits of Metric-Driven Verification
- 5.4. Key Coverage Considerations
- 5.5. Coverage Options Overview



- 5.6. Metric-Driven Verification Goals
- 5.7. Integrating Coverage Techniques
- 5.8. Explicit Coverage in SystemVerilog
- 5.9. Coverage Placement Strategies
- 5.10. Interface Monitor Coverage Declaration
- 5.11. Interface Monitor Coverage Trigger
- 5.12. Module Monitor Coverage
- 5.13. When to Cover
- 5.14. Comparing With and Without MDV

## 6. Verification Metrics

- 6.1. Key Verification Metrics
- 6.2. Understanding Verification Granularity
- 6.3. Optimizing Manual Verification Effort
- 6.4. Enhancing Verification Effectiveness
- 6.5. Achieving Verification Completeness
- 6.6. Verification Environment Reusability
- 6.7. Simulation Result Reusability
- 6.8. Coverage-Driven Verification Overview
- 6.9. Transaction-Driven Verification
- 6.10. Constrained Random Generation
- 6.11. Automatic Result Checking
- 6.12. Coverage Collection
- 6.13. Directed Tests
- 6.14. Stages of Coverage-Driven Verification

## 7. Metric Driven Verification

- 7.1. Metric Driven Verification (MDV) Overview
- 7.2. MDV: A Closed-Loop Process
- 7.3. MDV: Enhancing Productivity
- 7.4. MDV Productivity Benefits
- 7.5. Bug Detection Analysis: Test-Driven vs. Metric-Driven
- 7.6. Tool-Reported Coverage vs. Plan-Based Coverage
- 7.7. Limitations of Directed Testing
- 7.8. Benefits of Coverage-Driven Verification
- 7.9. Importance of Planning in CDV
- 7.10. MDV Across Verification Platforms
- 7.11. Metric-Driven Verification with Verification Manager
- 7.12. Key Advantages of Metric-Driven Verification



## 8. Assertion-Based Verification (ABV) Methodology

- 8.1. Traditional Verification
- 8.2. Traditional Verification Disadvantages
- 8.3. Assertion-Based Solution
- 8.4. Advantages of Assertion-Based Verification
- 8.5. Before Assertion-Base Verification
- 8.6. After Assertion-Base Verification
- 8.7. Verification Testbench based on Assertion
- 8.8. Property specification in Assertion-Based Verification
- 8.9. Assertion in Simulation Testbench
- 8.10. Assertion in Formal Verification Testbench
- 8.11. Assertion-Based Verification Methodology

## 9. Assertion-Based Verification (ABV) in Verification Tools

- 9.1. Coverage Driven Verification
- 9.2. Who Writes Assertions and Why
- 9.3. Introduction to Assertion-Based Verification
- 9.4. Assertion-Based Verification Flow
- 9.5. Assertion-Based Verification (ABV) in Formal
- 9.6. Assertion-Based Verification (ABV) in Simulation
- 9.7. Assertion-Based Verification (ABV) in Emulation/Acceleration
- 9.8. Assertion-Based Verification (ABV) in Functional Coverage
- 9.9. Dynamic & Formal Verification
- 9.10. Formal Verification Technology Factors
- 9.11. Simulator Overhead for Dynamic ABV
- 9.12. ABV in Plan to Closure Methodology



# SystemVerilog Language - Assertion (Member Free)

## 1. Assertion-Based Verification

- 1.1. Specifying Properties
- 1.2. What is an Assertion
- 1.3. Defining Assertion
- 1.4. Assertions Monitor Design Properties
- 1.5. What are Assertions used for
- 1.6. What aren't Assertions used for
- 1.7. Why to Use Assertions
- 1.8. Who to Write Assertions
- 1.9. Where to Use Assertions
- 1.10. Issues with Assertions I
- 1.11. Issues with Assertions II

## 2. Immediate and Concurrent Assertions

- 2.1. Terminology for SystemVerilog Assertion I
- 2.2. Terminology for SystemVerilog Assertion II
- 2.3. SVA Verification Directives
- 2.4. Why use SystemVerilog Assertions (SVA)
- 2.5. SystemVerilog Assertions Overview
- 2.6. Immediate Assertions
- 2.7. Limitations of Immediate Assertions
- 2.8. Concurrent Assertions
- 2.9. Building blocks of SVA
- 2.10. Steps to Create an SVA checker
- 2.11. SVA Assertion Structure
- 2.12. SVA Property Placement

## 3. Simple Boolean Assertions

- 3.1. Defining Design Behavior
- 3.2. How to Name and Assert Properties
- 3.3. Property Clocking in SVA
- 3.4. Understanding Counter Intuitive Clock Behaviour
- 3.5. Clocked Property Evaluation
- 3.6. Counter Intuitive Clocks
- 3.7. Using DUT Clock Edges
- 3.8. Default Clock Usage
- 3.9. Placing Assertions
- 3.10. Same Cycle Implication
- 3.11. Next Cycle Implication



- 3.12. FSM Verification with SVA
- 3.13. FSM Assertion Checks
- 3.14. Understanding Assertion Overlapping
- 3.15. Edge-Triggered Functions
- 3.16. \$past Function
- 3.17. \$stable Function
- 3.18. \$countones() and \$isunknown() Functions
- 3.19. \$onehot() and \$onehot0() Functions

## 4. Sequences

- 4.1. Sequence Operators and Features
- 4.2. Understanding Sequence Examples in SVA
- 4.3. Sequence Implication
- 4.4. Conditional and Unconditional Properties
- 4.5. Never Properties in SVA Assertions
- 4.6. Sequence Property Analysis
- 4.7. Edge-triggered Sequences with \$rose and \$fell
- 4.8. Handling Assertions with Disable Properties
- 4.9. Using Default Disable
- 4.10. Synchronous Abort Operators
- 4.11. Assertion Status
- 4.12. Cycle Delay Repetition
- 4.13. Cycle Delay Repetition Ranges
- 4.14. Consecutive Repetition
- 4.15. Consecutive Repetition with Ranges
- 4.16. Consecutive Repetition: Special Ranges
- 4.17. Non-Consecutive Repetition
- 4.18. Go-To Repetition
- 4.19. Non-Consecutive and Go-To Ranges
- 4.20. Non-Consecutive Repetition Range Example
- 4.21. Go-To Repetition Range Example
- 4.22. Repetition Shorthand
- 4.23. Property Abstraction
- 4.24. Challenges in Assertion Specification
- 4.25. Issues with Under-Specifying Assertions

## 5. Sequence Composition

- 5.1. Introduction to Named Sequences
- 5.2. Sequence Clocking
- 5.3. Sequence Composition Operators
- 5.4. Sequence fusion Operator



- 5.5. Sequence or Operator
- 5.6. Sequence and Operator
- 5.7. Sequence intersect Operator
- 5.8. Sequence Operator Examples
- 5.9. first\_match Operator
- 5.10. first\_match Operator: Removing Undesired Failures
- 5.11. Sequence throughout Operator
- 5.12. Sequence within Operator

## 6. Advanced SVA Features

- 6.1. Assertion Evaluation Process
- 6.2. Detecting Sequence Endpoints
- 6.3. Triggered Sequence Method
- 6.4. Sequence Endpoints in SVA
- 6.5. Sequence Arguments
- 6.6. Property Arguments
- 6.7. Action Blocks in Assertions
- 6.8. Local Variable
- 6.9. Local Variable Example
- 6.10. Handling Overlapping Properties
- 6.11. Challenges with Local Variables as Arguments
- 6.12. Data Integrity Verification with Local Variables
- 6.13. SVA Data Integrity Assumptions
- 6.14. Using Multi-clocked Sequences
- 6.15. Using Multi-clocked Properties
- 6.16. Procedural Block Assertions
- 6.17. Complex Property Example
- 6.18. Simplifying Complex Clock Expressions
- 6.19. Property Clocking Tips

## 7. Coding Guidelines

- 7.1. What Does Inefficient SVA Mean
- 7.2. Inefficient SVA in Simulation: Example
- 7.3. Can I Use the Same Properties in Simulation and Formal
- 7.4. What are the Symptoms of Inefficient SVA in Formal
- 7.5. Considerations for Efficient SVA
- 7.6. Simplifying Property – Example 1
- 7.7. Simplifying Property – Example 2
- 7.8. Simplifying Property – Example 3
- 7.9. Simplifying Property – Example 4
- 7.10. Express In Native Language



- 7.11. Keep Assertions Simple
- 7.12. Recommended Not to Be Used in Formal
- 7.13. Recommended SVA Coding Styles I
- 7.14. Recommended SVA Coding Styles II
- 7.15. Recommended SVA property modelling I
- 7.16. Recommended SVA property modelling II

## 8. Functional Coverage

- 8.1. Assessing Test Data Effectiveness
- 8.2. Understanding Coverage Metrics
- 8.3. Defining Functional Coverage
- 8.4. Using Cover Directive
- 8.5. Simulation vs. Formal Coverage
- 8.6. Cover Statement
- 8.7. Bus Protocol Example 1
- 8.8. Bus Protocol Example 2
- 8.9. Debugging Assertions with Coverage
- 8.10. Detecting Enabling Conditions
- 8.11. Cover Groups in SystemVerilog
- 8.12. Understanding Cover Properties
- 8.13. Understanding Cover Sequences
- 8.14. SVA LRM: 2012/2017 Changes
- 8.15. Backward Compatibility Issue with cover Verification Directive
- 8.16. Infinite Coverage in SVA

## 9. Practical SVA Application

- 9.1. Assertions Ensure Correct Behavior of Design
- 9.2. Case 1: AMBA AHB Slave Interface
- 9.3. Simplified Single Transfer
- 9.4. Wait States
- 9.5. Incomplete Transfers
- 9.6. AMBA AHB Incomplete Transfer Assertion
- 9.7. Four Beat Incrementing Burst Example
- 9.8. Step 1: Identify Atomic Behaviors
- 9.9. Step 2: Define Sequences for Atomic Behaviors
- 9.10. Step 3: Build Compound Sequences
- 9.11. Step 4: Write assert Directives
- 9.12. Step 5: Write cover Directives
- 9.13. Case 2: Asynchronous FIFO Design
- 9.14. FIFO Black-Box Property Checks
- 9.15. Asynchronous FIFO Implementation



9.16. FIFO White-Box Property Checks

## 10. Static Verification

- 10.1. Formal Verification Methods
- 10.2. Static vs. Dynamic Assertion-Based Verification
- 10.3. Property Checking Overview
- 10.4. Property Checking and Simulation Synergy
- 10.5. Using Property Checking
- 10.6. Verifying Property Behaviors
- 10.7. Using Input Constraints
- 10.8. Applying Constraints: Scan Mode
- 10.9. Benefits of Property Checking
- 10.10. Case Study with Property Checking
- 10.11. Benefits of Adding Assertions



# UVM Series

## Universal Verification Methodology - SystemVerilog (Member Free)

### 1. Basic Classes

- 1.1. Key Aspects of Simple Classes
- 1.2. Safe and Managed Class Handles
- 1.3. Managing Variable Data Length
- 1.4. Randomized Frame Length
- 1.5. In-Line Constraints
- 1.6. Managing Constraints in Classes
- 1.7. Understanding Class Inheritance
- 1.8. Inheritance and Constructors
- 1.9. Using this and super Handles
- 1.10. Constraints in Subclasses
- 1.11. Constraints with Multiple Subclasses
- 1.12. Aggregate Classes

### 2. Static Properties and Methods

- 2.1. Understanding Static Properties
- 2.2. Static Property Characteristics
- 2.3. Understanding Static Methods
- 2.4. Static Properties Use Case
- 2.5. Static Properties and Methods

### 3. Virtuality and Polymorphism

- 3.1. Polymorphism in SystemVerilog
- 3.2. Copying Subclass Instance to Parent Handle
- 3.3. Copying Parent Instance to Subclass Handle
- 3.4. \$cast: Handling Subclass Instances
- 3.5. Polymorphism Example
- 3.6. Class Member Access in Polymorphism
- 3.7. Virtual Methods
- 3.8. Resolving Class Method Calls
- 3.9. Virtual Classes and Pure Virtual Methods
- 3.10. Parameterized Classes



## **4. Common Class Operations**

- 4.1. Reference Copy
- 4.2. Clone
- 4.3. Shallow and Deep Clone
- 4.4. Deep Cloning
- 4.5. Printing Method
- 4.6. Print Policy Control
- 4.7. Polymorphism and Encapsulation
- 4.8. Constructing a Print Array
- 4.9. Building and Using the Print Method
- 4.10. Aggregate Classes
- 4.11. Managing Instance Names
- 4.12. Using Instance Names

## **5. Components**

- 5.1. Architecture for Driving Data into DUT
- 5.2. Verification Component Hierarchy
- 5.3. Instance Names and Parent Pointers
- 5.4. Base Class Implementation for Components
- 5.5. Simple Component Connections

## **6. DUT Connection**

- 6.1. SystemVerilog Interface Basics
- 6.2. Driving Data into DUT from Module
- 6.3. Driving Data into DUT from Class Via Interface Instance
- 6.4. Reuse Issues in Class Connection
- 6.5. Virtual Interface
- 6.6. Interface Methods

## **7. Building a Verification Component**

- 7.1. Verification Component Functionality
- 7.2. Sequencer
- 7.3. Driver
- 7.4. Monitor
- 7.5. Agent
- 7.6. Verification Component Essentials
- 7.7. Instantiating Verification Component



## 8. Class-Based Testbench

- 8.1. Transaction-Based Verification
- 8.2. Simple Testbench: PDS Port
- 8.3. Classes as Component Structures
- 8.4. Parent Handle
- 8.5. Using Parent Handles for Pathnames
- 8.6. Sequencer Stimulus Generator
- 8.7. Simple Driver
- 8.8. Simple Monitor
- 8.9. PDS Verification Component
- 8.10. VC Configuration
- 8.11. Simple VC Instantiation
- 8.12. Sequencer Policy
- 8.13. Setting Sequencer Policy
- 8.14. Testbench Wrapper
- 8.15. Test Class Functionality
- 8.16. Simplified Component Class Diagram
- 8.17. Features of the Class Library

## 9. Dynamic Testbench

- 9.1. Dynamic Testbench Behavior
- 9.2. Policy vs. Factory for Test Selection
- 9.3. Factory Design for Object Creation
- 9.4. Using Factories in SystemVerilog
- 9.5. Factory Overrides
- 9.6. Component Configuration with Factories
- 9.7. Constructor Configuration Challenges
- 9.8. Construction of Class Properties
- 9.9. Build Method (Leaf Components)
- 9.10. Build Method (Hierarchical Components)
- 9.11. Builder Configurations
- 9.12. Factory vs. Builder in UVM



# Universal Verification Methodology - Fundamentals (Member Free)

## 1. Introduction to UVM Methodology

- 1.1. The Need for Methodology
- 1.2. Advantages of Verification Methodology I
- 1.3. Advantages of Verification Methodology II
- 1.4. What Is UVM
- 1.5. Metric-Driven Verification Review
- 1.6. Hardware Acceleration
- 1.7. Signal-Based vs Transaction-Based Acceleration
- 1.8. Creating Reusable Environments
- 1.9. Creating Reusable Components
- 1.10. Self-Checking and Coverage
- 1.11. Enhancing Interface Flexibility
- 1.12. Moving to System Level
- 1.13. Monitors, Checking and Coverage
- 1.14. Active and Passive Modes
- 1.15. Interface UVC Architecture
- 1.16. Common Interface
- 1.17. Vision for Multi-Language UVM
- 1.18. What Is a UVC
- 1.19. Interface and Module UVCs
- 1.20. UVM Library Layers
- 1.21. Simplified UVM Class Hierarchy
- 1.22. Using the UVM Class Library
- 1.23. Locating UVM Library Files
- 1.24. Key Advantages of UVM

## 2. DUT and Project Overview

- 2.1. DUT: YAPP Router
- 2.2. Packet Router DUT Specification
- 2.3. YAPP Input Port Protocol
- 2.4. Packet Router DUT Registers Specification
- 2.5. HBUS Protocol
- 2.6. How Will We Verify the YAPP

## 3. Data Modeling

- 3.1. Data Modeling: Requirements
- 3.2. Data Item Example: YAPP Packet



- 3.3. Design for Test Writing Using Knobs
- 3.4. Basic Constraints for Legal Packets
- 3.5. Layering Constraints for Testing
- 3.6. Example: Layering Constraints
- 3.7. Class Methods
- 3.8. UVM Inheritance Tree for Data Items
- 3.9. Data Modeling with UVM
- 3.10. UVM Field Macros
- 3.11. Field Macros for Array Types
- 3.12. Flag Arguments for Field Macros
- 3.13. Field Macro Usage Examples
- 3.14. Copy and Clone Automation Methods
- 3.15. Compare Automation Method
- 3.16. Print Automation Method
- 3.17. Transaction Recording in UVM
- 3.18. Creating and Using a UVM Package
- 3.19. Compiling UVM Code

#### 4. Simulation Phases

- 4.1. Simulation Phases for Components
- 4.2. Simulation Phases and Coordination
- 4.3. UVM Simulation Phase Names and Descriptions
- 4.4. Run-Time Sub-Phases
- 4.5. Guidelines for Run-Time Sub-Phases
- 4.6. Build Phase
- 4.7. Connect Phase
- 4.8. Activating Phases
- 4.9. Key Points on Phase Methods

#### 5. Building Interface UVC

- 5.1. Router Verification Environment: Front end
- 5.2. Modeling Topology with UVM
- 5.3. Structural Components Inheritance Tree
- 5.4. Component Macros and Constructor
- 5.5. Interface UVC Hierarchy
- 5.6. Sequencer-Driver TLM Connection
- 5.7. Sequencer-Driver Operation
- 5.8. Driver, Sequencer, Sequence and Default Settings
- 5.9. Creating the Agent: Declarations
- 5.10. Creating the Agent: Build and Connect
- 5.11. Creating the UVC Top-Level



- 5.12. Directory Structure
- 5.13. UVC Directory Structure
- 5.14. UVC Package and Include Files
- 5.15. Top-Level Module for Initial Testing
- 5.16. Messages for Debug and Error Reporting
- 5.17. Message Macros: Info, Verbosity
- 5.18. Full Range of Message Macros
- 5.19. Message Actions

## 6. Test Classes

- 6.1. Router Verification Environment: Test Classes
- 6.2. Requirements for Test Classes
- 6.3. uvm\_test Inheritance Tree
- 6.4. Simple Test Example
- 6.5. Option 1: Select Test in run\_test
- 6.6. Option 2: UVM\_TESTNAME
- 6.7. run\_test and uvm\_test\_top
- 6.8. uvm\_top
- 6.9. Topology Report Example

## 7. Configuration

- 7.1. Configuring Topology
- 7.2. Configuring is\_active Agent Property
- 7.3. Setting Config Properties
- 7.4. Config Property Setting Example
- 7.5. Configuration Mechanism Overview
- 7.6. Selected Configuration Database Methods
- 7.7. Setting YAPP Agent to Passive from Test
- 7.8. Config Setting Rules
- 7.9. Debugging Configuration Settings
- 7.10. set\_config Methods

## 8. Factory and Override Methods

- 8.1. Enhancing Object Behavior in UVM
- 8.2. Constraint Layering
- 8.3. Constraint Layering Problem
- 8.4. Solution: Factory
- 8.5. UVM Factory Overview
- 8.6. Object Factory Use: create
- 8.7. Type Overrides with Factory



- 8.8. Overriding Specific Instances
- 8.9. Type Versus Instance Overrides
- 8.10. Alternative Syntax for Type/Instance Overrides
- 8.11. Override Rules
- 8.12. Notes About the Factory

## 9. UVM Sequences

- 9.1. Understanding UVM Sequences
- 9.2. UVM Sequence Features
- 9.3. Sequences and UVCs Overview
- 9.4. uvm\_sequence Base Class
- 9.5. UVM Sequences Example
- 9.6. The do Operation
- 9.7. Understanding uvm\_do Macro Execution
- 9.8. Additional `uvm\_do Macros
- 9.9. Directed Stimulus Using Macros
- 9.10. Alternative to uvm\_do Macros: Explicit Flow
- 9.11. Sequence Properties
- 9.12. Nesting Sequences
- 9.13. Executing Sequences
- 9.14. Setting Default Sequence in Run Phase
- 9.15. Executing UVC Sequence in Test
- 9.16. Sequence Callback Methods
- 9.17. Ending Simulation
- 9.18. Objection Methods in Sequences
- 9.19. Test Objections
- 9.20. Objection Changes in UVM1.2
- 9.21. Drain Time



## Universal Verification Methodology - Advanced (Member Free)

### 1. DUT Connections

- 1.1. Router Verification Environment: Testbench
- 1.2. Understanding the Testbench
- 1.3. Using a Testbench for Multiple Tests
- 1.4. Using a Library of Testbenches
- 1.5. Using a Testbench Class to Specify Configuration
- 1.6. Review of SystemVerilog Interfaces
- 1.7. Interface Example (Module-Based Verification)
- 1.8. Interface Connection to UVM Driver and Monitor
- 1.9. Virtual Interface Example
- 1.10. UVM and Hardware Top Modules
- 1.11. Setting the Virtual Interface
- 1.12. Getting the Virtual Interface
- 1.13. Convenience Types for Interface Assignment
- 1.14. Handling Virtual Interface Limitations

### 2. Multiple UVCs

- 2.1. Integrating Multiple UVCs
- 2.2. Multi-Agent Interface UVCs
- 2.3. Interface UVC Development
- 2.4. Interface UVC Reuse in System Development
- 2.5. Complex Interface UVC Flexibility
- 2.6. Clock and Reset UVC Features
- 2.7. Clock and Reset UVC for Hardware Acceleration
- 2.8. Integrating Multiple UVCs
- 2.9. Compiling with Multiple UVCs
- 2.10. Configuration Objects
- 2.11. Getting Configuration Objects
- 2.12. Advantages and Disadvantages of Configuration Objects

### 3. Virtual Sequences

- 3.1. Understanding Virtual Sequences
- 3.2. Effective Use of Virtual Sequences
- 3.3. Adding a Router Virtual Sequencer
- 3.4. Router Virtual Sequencer Functions
- 3.5. Procedure for Adding a Virtual Sequencer
- 3.6. Declaring Virtual Sequencers
- 3.7. p\_sequencer Variable Usage
- 3.8. Declaring Virtual Sequences



- 3.9. Virtual Sequence Objection
- 3.10. Instantiating & Connecting Virtual Sequencer
- 3.11. Setting the Virtual Sequence

## 4. Implementing a Scoreboard

- 4.1. Router Verification Environment: Adding a Scoreboard
- 4.2. Scoreboard Architecture
- 4.3. Key Considerations for Scoreboard Design
- 4.4. YAPP Router Scoreboard Implementation
- 4.5. Communication Between Class Components
- 4.6. Communication with UVCs: Analysis Interface
- 4.7. Analysis Interface Implementation
- 4.8. TLM Concepts: Port and Imp
- 4.9. Simple Analysis Interface Example
- 4.10. Simple Analysis Interface Connection
- 4.11. Managing Multiple Analysis Imp Connections
- 4.12. Managing Multiple Analysis Interface Connections
- 4.13. YAPP Scoreboard Architecture
- 4.14. Cloning Analysis Write Data
- 4.15. Simple Scoreboard Comparison
- 4.16. UVM Scoreboard Comparison
- 4.17. Complex Router Module UVC

## 5. Transaction Level Modelling (TLM)

- 5.1. TLM Application Programming Interface (API)
- 5.2. Concepts: Data and Control Flow
- 5.3. Concepts: Blocking and Nonblocking
- 5.4. Overview of Uni-Directional TLM Methods
- 5.5. Selected Connector and Method Options
- 5.6. Understanding TLM Connection Implementation
- 5.7. Understanding Port and Imp
- 5.8. Get Example
- 5.9. Get Connection
- 5.10. Hierarchical Connections and Exports
- 5.11. Uni-Directional TLM Communication Models
- 5.12. Analysis Broadcast
- 5.13. Broadcast Analysis Example
- 5.14. Broadcast Analysis Connection
- 5.15. TLM FIFO Component
- 5.16. TLM FIFO Example
- 5.17. TLM FIFO Connection



- 5.18. TLM FIFO Methods
- 5.19. Characteristics of Analysis FIFO
- 5.20. Analysis FIFO Scoreboard
- 5.21. Analysis FIFO Testbench Connection
- 5.22. YAPP Scoreboard Using Analysis FIFOs
- 5.23. Bi-Directional TLM Transport Connection
- 5.24. TLM2 for UVM

## 6. Functional Coverage Modeling

- 6.1. Metric-Driven Verification Review
- 6.2. Metric-Driven Verification Flow
- 6.3. Benefits of Metric-Driven Verification
- 6.4. Key Coverage Considerations
- 6.5. Coverage Options Overview
- 6.6. Metric-Driven Verification Goals
- 6.7. Integrating Coverage Techniques
- 6.8. Explicit Coverage in SystemVerilog
- 6.9. Coverage Placement Strategies
- 6.10. Interface Monitor Coverage Declaration
- 6.11. Interface Monitor Coverage Trigger
- 6.12. Module Monitor Coverage
- 6.13. When to Cover
- 6.14. Comparing With and Without MDV

## 7. UVM Register Modeling

- 7.1. Verification Scenario – Memory Controller
- 7.2. Verifying the Memory Controller
- 7.3. UVM Register Models
- 7.4. UVM Register API
- 7.5. Integration of Register Sequences and Models
- 7.6. Simplifying Register Model Generation
- 7.7. IP-XACT XML for mode0\_reg
- 7.8. Generated Register Definition for mode0\_reg
- 7.9. Integrating Register Model
- 7.10. Integrate Register Model into Testbench
- 7.11. Selected UVM Built-In Sequences for Registers
- 7.12. Executing a Built-In Register Sequence
- 7.13. Register Operations and User Stimulus
- 7.14. Register API and Access Policies
- 7.15. Basic Operations: Read/Write
- 7.16. Accessing mirrored: predict/get\_mirrored\_value



- 7.17. Backdoor DUT Access: Peek/Poke
- 7.18. Model Access Only: get/set/randomize
- 7.19. Update: DUT to Match Register Model
- 7.20. Mirror: Register Model to Match DUT
- 7.21. Executing Register API Calls in Test

## 8. UVM Sequence Library

- 8.1. What Is a Sequence Library
- 8.2. Basic Sequence Library Declaration
- 8.3. Default Sequence Library Behavior
- 8.4. Selecting a Sequence Library
- 8.5. Executing a Sequence Library
- 8.6. Debugging a Sequence Library
- 8.7. Changing Default Behavior
- 8.8. Customizing a Library Instance
- 8.9. Sequence Library Implementation
- 8.10. User-Defined Selection Mode
- 8.11. Sequence Library Configuration Object
- 8.12. Alternative for Adding Sequence to Library
- 8.13. Sequence Library: Limitations & Applications

## 9. UVM-IEEE Migration Issues

- 9.1. UVM-IEEE Library: Structure and Migration
- 9.2. Deprecated Constructs in UVM-IEEE
- 9.3. Automation Policies in UVM1.x
- 9.4. Automation Policies in UVM-IEEE
- 9.5. uvm\_top Accessor Methods
- 9.6. Debugging methods
- 9.7. Rationalization of Sequence Macros
- 9.8. Missing Sequence Macros in UVM-IEEE: Option 1
- 9.9. Missing Sequence Macros in UVM-IEEE: Option 2



# Universal Verification Methodology - Register Verification I

## (Member Free)

1. Introduction to Register Modeling
  - 1.1. Register Verification and Reference Models
  - 1.2. Overview of Verification Scenario
  - 1.3. Steps to Verify the Design
  - 1.4. UVM Register Modeling Framework
  - 1.5. UVM Register Model Class Hierarchy
  - 1.6. Register API
  - 1.7. Frontdoor and Backdoor Access
  - 1.8. Prediction and Register Updates
  - 1.9. Address Maps in Register Models
  - 1.10. Benefits of Register Layer Methodology
  - 1.11. Register Layer Overview
2. Overview of Project DUT
  - 2.1. DUT: YAPP Router
  - 2.2. Packet Router DUT Specification
  - 2.3. Router Input Port Protocol Overview
  - 2.4. HBUS Protocol Overview
  - 2.5. Router Address Mappings and Block Names
  - 2.6. Router Configuration Registers
  - 2.7. Router Enable Register Bits
  - 2.8. Router Memory Blocks
  - 2.9. Router Verification Environment
  - 2.10. Recorded HBUS Transaction
3. Generating the Register Model
  - 3.1. Understanding the UVM Register Model
  - 3.2. Register Model Generators
  - 3.3. iregGen: XML-Based Register Generator
  - 3.4. Understanding IP-XACT and XML
  - 3.5. IP-XACT Register Definitions
  - 3.6. Using iregGen
  - 3.7. IP-XACT for Simple Register – addr0\_cnt\_reg
  - 3.8. Generated Register Definition for addr0\_cnt\_reg
  - 3.9. Field Configuration in Registers
  - 3.10. IP-XACT for Register with Fields – en\_reg
  - 3.11. Generated Register Definition for en\_reg



- 3.12. IP-XACT: Wrap Registers in an Address Block
- 3.13. Generated Register Block with Address Map
- 3.14. Memory Management in Address Blocks
- 3.15. IP-XACT: Memory Definition
- 3.16. Generated Memory
- 3.17. IP-XACT: Top Level Declarations
- 3.18. Register Model Top Level 1 – Register Block
- 3.19. Register Model Top Level 2 – Memories

## 4. IP-XACT Vendor Extensions

- 4.1. Defining Register Access Policies
- 4.2. UVM Field Access Policies
- 4.3. Extending IP-XACT with Vendor Extensions
- 4.4. Vendor Extension: Constraints and Description
- 4.5. Vendor Extension: type
- 4.6. Vendor Extension: resets
- 4.7. Vendor Extension: hdl\_path
- 4.8. Simple Model Customization

## 5. Creating and Modifying Adapters

- 5.1. Register Adapter Overview
- 5.2. Register Sequence to UVC Adapter
- 5.3. UVC to Register Adapter
- 5.4. Adapter class – uvm\_reg\_adapter
- 5.5. Register Data Structure: uvm\_reg\_bus\_op
- 5.6. HBUS UVC Data Item: hbus\_transaction
- 5.7. HBUS Adapter – constructor and reg2bus
- 5.8. Adapter bus2reg
- 5.9. HBUS UVC Package and Adapter
- 5.10. Extending the Adapter
- 5.11. Declaring and Using the Adapter Extension
- 5.12. Accessing the Adapter Extension

## 6. Simple Model Integration

- 6.1. Simple Model Integration: Assumptions
- 6.2. Register Model Creation and Simple Integration
- 6.3. Configuration Database Overview
- 6.4. Simple Adapter Connection
- 6.5. Managing Multiple Address Maps
- 6.6. Resetting the Register Model



- 6.7. Detecting Reset
- 6.8. Checking Topology for Adapter Connection
- 6.9. Test Execution of a Built-In Register Sequence

## 7. Register Sequences

- 7.1. Register Sequences Require a Model Handle
- 7.2. Retrieve Register Model from Configuration Database
- 7.3. Sequence Callbacks
- 7.4. Base Sequence Callbacks
- 7.5. Register Sequence Using UVM Built-In Sequences
- 7.6. Subset of UVM Built-In Sequences for Registers
- 7.7. Built-In Sequences and Access Policies
- 7.8. Customizing Register Sequences
- 7.9. Configuring Built-In Sequences

## 8. Prediction

- 8.1. Understanding Prediction
- 8.2. Implicit Prediction
- 8.3. Challenges with Implicit Prediction
- 8.4. Explicit Prediction
- 8.5. Predictor Component Functions
- 8.6. Predictor Creation and Configuration
- 8.7. Analysis TLM Components
- 8.8. Predictor Connection
- 8.9. Passive Prediction
- 8.10. Prediction Modes
- 8.11. Manual Prediction from Sequences
- 8.12. Manual Prediction from Scoreboard
- 8.13. Initiating Manual Prediction

## 9. Register and Memory Access Methods

- 9.1. Data Modelling and the Register Model
- 9.2. Mirroring and the Register Model
- 9.3. Choosing the Right Register API Method
- 9.4. Register write() Method Signature
- 9.5. Basic Access: Read/Write – Frontdoor
- 9.6. Basic Access: Read/Write – Backdoor
- 9.7. Examples of Read and Write
- 9.8. Backdoor DUT Access: Peek/Poke
- 9.9. Examples of Peek and Poke



- 9.10. Accessing Mirrored: predict/get\_mirrored\_value
- 9.11. Model Access Only: get/set
- 9.12. Model Access Only: randomize
- 9.13. Update DUT to Match Register Model – Frontdoor
- 9.14. Update DUT to Match Register Model – Backdoor
- 9.15. Mirror Register Model to Match DUT – Frontdoor
- 9.16. Mirror Register Model to Match DUT – Backdoor
- 9.17. Mirror Usage Examples
- 9.18. Disabling check-on-read
- 9.19. Convenience Handles
- 9.20. Memory Access Methods
- 9.21. Memory write() Method
- 9.22. Memory burst\_write() Method



# Universal Verification Methodology - Register Verification II

## (Member Free)

### 1. Advanced Access API and Introspection

- 1.1. Selected Register Model Introspection Methods
- 1.2. Introspection with Register API
- 1.3. Filtering Register and Field Queues
- 1.4. Examples of get, set, update, and randomize
- 1.5. Introspection Methods Overview

### 2. Applying Coverage to Register Model

- 2.1. Defining Coverage with Vendor Extension
- 2.2. Auto-Generated Covergroups for Fields
- 2.3. Covergroup Sampling in UVM
- 2.4. Adding and Modifying Coverage Groups
- 2.5. Coverage API Overview

### 3. Custom Register Modelling

- 3.1. Register Arrays with Vendor Extensions
- 3.2. Register Array in Model
- 3.3. Multi-Dimensional Register Arrays
- 3.4. HDL Path for Register Array
- 3.5. Indirect Registers
- 3.6. Indirect Registers: IP-XACT Code
- 3.7. Indirect Registers: UVM code
- 3.8. Indirect Registers: Register Model Topology
- 3.9. Register Write: Hooks and Callbacks
- 3.10. Customized Access Policy Using post\_predict
- 3.11. Customized Access Policy Callback Registration
- 3.12. Register Dependencies – Enabled Register
- 3.13. Enabled Register Callback Registration
- 3.14. Register Dependencies: Aliased Register
- 3.15. Aliased Register Callbacks
- 3.16. Shared (Twin) Registers
- 3.17. Shared Registers: XML
- 3.18. Shared Registers: UVM

### 4. Using the Register Model in Scoreboards

- 4.1. YAPP Router Scoreboard Overview



- 4.2. YAPP Router Scoreboard Structure
- 4.3. Router Reference Model and Scoreboard
- 4.4. Efficient Use of Register Model
- 4.5. Module UVC and Register Model
- 4.6. Access Register Model in Reference Model
- 4.7. Updating the Reference Model – TLM Declarations
- 4.8. Using Register Model to Query DUT Configuration
- 4.9. Verifying Volatile (Read-Only) Registers
- 4.10. YAPP Address Counter Validation
- 4.11. Setting Mirrored in the Register Model
- 4.12. mirror() Check Results

## 5. Field-Level Access

- 5.1. Register and Field Level Access
- 5.2. Register Example
- 5.3. Byte Lanes
- 5.4. Field Level Access Support
- 5.5. Field Access Overview

## 6. Active Monitoring of DUT Registers

- 6.1. Active Monitoring
- 6.2. User-Defined Backdoor Declaration
- 6.3. Integrating User-Defined Backdoor
- 6.4. Issues with a Hardwired User-Defined Backdoor
- 6.5. User-Defined Backdoor with Interface
- 6.6. Active Monitoring via Interface
- 6.7. User-Defined Backdoor Integration with Interface
- 6.8. Interface Array for Active Monitor Scaling
- 6.9. Register Array for Active Monitor Scaling

## 7. User-Defined Frontdoor

- 7.1. Conventional Frontdoor Operation
- 7.2. Understanding User-Defined Frontdoors
- 7.3. User Frontdoor Operation
- 7.4. Register Data Handling in uvm\_reg\_item
- 7.5. Indirect Register Access Example
- 7.6. User-Defined Frontdoor Declaration
- 7.7. Simple Indirect Register Address
- 7.8. User-Defined Frontdoor Integration



## 8. Running Built-in Register Sequences and Tests

- 8.1. UVM Built-In Sequences for Registers
- 8.2. UVM Built-In Sequences for Memories
- 8.3. UVM Built-In Sequences for Registers and Memories
- 8.4. Built-In Sequences and Access Policies
- 8.5. Setting Skip Attributes via Resource Database
- 8.6. Overview of `uvm_reg_mem_built_in_seq`

## 9. IP-XACT 2014 Features

- 9.1. IP-XACT Name Space
- 9.2. Field-Level Reset in IP-XACT
- 9.3. Defining Multiple Resets
- 9.4. Defining HDL Path in IP-XACT



# Formal Verification Series

## Introduction to Formal Verification

### 1. Formal Verification Overview

- 1.1. What is Formal Verification
- 1.2. Formal Verification – Historical Perspective
- 1.3. Formal Verification Technology is Growing
- 1.4. Why need Formal Verification
- 1.5. Requirements for Formal verification
- 1.6. Formal Verification Challenges
- 1.7. Formal Verification Tools Vendor
- 1.8. Major Vendors Formal Tool Comparison
- 1.9. Formal Capability Levels

### 2. Formal Analysis

- 2.1. Compile a Formal Model
- 2.2. Formal Model Concept
- 2.3. Applying a proof algorithm
- 2.4. Formal Proof Results
- 2.5. Formal Proof Performance
- 2.6. Characteristics of performance
- 2.7. Formal Tool Setup and Control
- 2.8. Formal Debug

### 3. Property Checking

- 3.1. Defining Property Checking
- 3.2. Verification with Property Checking
- 3.3. Property Checking Work Flow
- 3.4. Property Checking Benefits
- 3.5. Property Checking Guidelines
- 3.6. Specifying Properties
- 3.7. Observability and Controllability
- 3.8. Formal Property Checking Framework

### 4. Formal Property Verification (FPV)

- 4.1. Dynamic & Formal Verification
- 4.2. Formal Verification Technology Factors



- 4.3. Formal Property Verification(FPV)
- 4.4. FPV Process Flow
- 4.5. Inputs and Outputs for FPV
- 4.6. Creating FPV Testbench
- 4.7. Where to Use FPV

## 5. Formal Complexity

- 5.1. Formal Complexity Defined
- 5.2. Measure of Complexity
- 5.3. Properties and Complexity
- 5.4. Complexity Analysis
- 5.5. Complexity Reduction Methods I
- 5.6. Complexity Reduction Methods II
- 5.7. Complexity Reduction Methods III
- 5.8. Complexity Reduction Methods IV

## 6. Formal Coverage

- 6.1. Formal Coverage – Controllability and Observability
- 6.2. Formal Coverage – Defined
- 6.3. Formal Coverage – Models
- 6.4. Formal Coverage – Types
- 6.5. Formal Coverage – Metrics
- 6.6. Formal Coverage – Measure
- 6.7. Formal Coverage – Criteria

## 7. Formal Signoff Methodology

- 7.1. Formal Signoff – Achieving
- 7.2. Formal Signoff – Challenges and Rewards
- 7.3. Formal Signoff – ROI and Criteria
- 7.4. Formal Signoff – Tracking
- 7.5. Formal Signoff – Environment
- 7.6. Formal Signoff – Flow
- 7.7. Formal Signoff with Full Prove Flow
- 7.8. Formal Signoff with Coverage Flow

## 8. Formal Verification Applications

- 8.1. Formal Verification Applications
- 8.2. Major Types of Formal Verification Apps



- 8.3. General Design Issues
- 8.4. Safety/Security
- 8.5. Structural Operation
- 8.6. Assertion Creation



# Formal Verification: SVA Coding

1. Property Checking
  - 1.1. Defining Property Checking
  - 1.2. Verification with Property Checking
  - 1.3. Property Checking Work Flow
  - 1.4. Property Checking Benefits
  - 1.5. Property Checking Guidelines
  - 1.6. Property Types
  - 1.7. Assertions
  - 1.8. Covers
  - 1.9. Assumptions
  - 1.10. Specifying Properties
  - 1.11. Observability and Controllability
  - 1.12. Formal Property Checking Framework
2. Formal Property Verification (FPV)
  - 2.1. Dynamic & Formal Verification
  - 2.2. Formal Verification Technology Factors
  - 2.3. Formal Property Verification(FPV)
  - 2.4. FPV Process Flow
  - 2.5. Inputs and Outputs for FPV
  - 2.6. Creating FPV Testbench
  - 2.7. FPV Testbench – Cycle-Based Models
  - 2.8. FPV Testbench – Constraints
  - 2.9. FPV Testbench – End-to-End Checkers
  - 2.10. FPV Testbench – Functional Covers
  - 2.11. FPV Testbench – Formal VIP
  - 2.12. Where to Use FPV
3. Introduction to SystemVerilog Assertions (SVA)
  - 3.1. Terminology for SystemVerilog Assertion I
  - 3.2. Terminology for SystemVerilog Assertion II
  - 3.3. SVA Verification Directives
  - 3.4. Why use SystemVerilog Assertions (SVA)
  - 3.5. SystemVerilog Assertions Overview
  - 3.6. Immediate Assertions
  - 3.7. Limitations of Immediate Assertions
  - 3.8. Concurrent Assertions
  - 3.9. Building blocks of SVA



- 3.10. Steps to Create an SVA checker
- 3.11. SVA Assertion Structure
- 3.12. SVA Property Placement

## 4. SystemVerilog Assertions (SVA) Basic Syntax

- 4.1. SVA Recommended Subset
- 4.2. Boolean Expressions I
- 4.3. Boolean Expressions II
- 4.4. Boolean Expressions III
- 4.5. Sequences I
- 4.6. Sequences II
- 4.7. Sequences III
- 4.8. Sequences IV
- 4.9. Properties I
- 4.10. Properties II
- 4.11. Properties III
- 4.12. Properties IV
- 4.13. Liveness I
- 4.14. Liveness II
- 4.15. Glue Logic I
- 4.16. Glue Logic II

## 5. Coding Guidelines

- 5.1. What Does Inefficient SVA Mean
- 5.2. Inefficient SVA in Simulation: Example
- 5.3. Can I Use the Same Properties in Simulation and Formal
- 5.4. What are the Symptoms of Inefficient SVA in Formal
- 5.5. Considerations for Efficient SVA
- 5.6. Simplifying Property – Example 1
- 5.7. Simplifying Property – Example 2
- 5.8. Simplifying Property – Example 3
- 5.9. Simplifying Property – Example 4
- 5.10. Express In Native Language
- 5.11. Keep Assertions Simple
- 5.12. Recommended Not to Be Used in Formal
- 5.13. Recommended SVA Coding Styles I
- 5.14. Recommended SVA Coding Styles II
- 5.15. Recommended SVA property modelling I
- 5.16. Recommended SVA property modelling II



## 6. Use of Auxiliary HDL Code

- 6.1. What Is Auxiliary Code
- 6.2. Auxiliary Code vs SVA
- 6.3. Auxiliary Code – Example 1 (I)
- 6.4. Auxiliary Code – Example 1 (II)
- 6.5. Auxiliary Code – Example 1 (III)
- 6.6. Auxiliary Code – Example 1 (IV)
- 6.7. Auxiliary Code – Example 2 (I)
- 6.8. Auxiliary Code – Example 2 (II)
- 6.9. Auxiliary Code – Example 2 (III)
- 6.10. Auxiliary Code – Example 3 (I)
- 6.11. Auxiliary Code – Example 3 (II)
- 6.12. Auxiliary Code – Example 3 (III)
- 6.13. Auxiliary Code – Example 4 (I)
- 6.14. Auxiliary Code – Example 4 (II)
- 6.15. Auxiliary Code – Example 5 (I)
- 6.16. Auxiliary Code – Example 5 (II)
- 6.17. Auxiliary Code – Example 5 (III)
- 6.18. Auxiliary Code – Example 6 (I)
- 6.19. Auxiliary Code – Example 6 (II)
- 6.20. Auxiliary Code – Example 6 (III)
- 6.21. Auxiliary Code – Example 7 (I)
- 6.22. Auxiliary Code – Example 7 (II)
- 6.23. Auxiliary Code – Example 7 (III)

## 7. Verification Components (vcomp)

- 7.1. Verification Component – Defined
- 7.2. Verification Component – Example
- 7.3. Verification Component – Characteristics
- 7.4. Techniques for Placing Assertion
- 7.5. Technique Comparison with Methods
- 7.6. Anatomy of a Verification Component
- 7.7. Interface Verification Component Example
- 7.8. Verification Component Design Techniques
- 7.9. Verification Component Parameterization
- 7.10. Coverage Points
- 7.11. Benefits of Coverage Points
- 7.12. Utilizing Formal Verification IP



# Formal Verification: PSL Coding

## 1. Property Checking

- 1.1. Defining Property Checking
- 1.2. Verification with Property Checking
- 1.3. Property Checking Work Flow
- 1.4. Property Checking Benefits
- 1.5. Property Checking Guidelines
- 1.6. Property Types
- 1.7. Assertions
- 1.8. Covers
- 1.9. Assumptions
- 1.10. Specifying Properties
- 1.11. Observability and Controllability
- 1.12. Formal Property Checking Framework

## 2. Formal Property Verification (FPV)

- 2.1. Dynamic & Formal Verification
- 2.2. Formal Verification Technology Factors
- 2.3. Formal Property Verification(FPV)
- 2.4. FPV Process Flow
- 2.5. Inputs and Outputs for FPV
- 2.6. Creating FPV Testbench
- 2.7. FPV Testbench – Cycle-Based Models
- 2.8. FPV Testbench – Constraints
- 2.9. FPV Testbench – End-to-End Checkers
- 2.10. FPV Testbench – Functional Covers
- 2.11. FPV Testbench – Formal VIP
- 2.12. Where to Use FPV

## 3. Introduction to Property Specification Language (PSL)

- 3.1. Terminology for Property Specification Language I
- 3.2. Terminology for Property Specification Language II
- 3.3. Property Specification Language Overview
- 3.4. Layers of PSL
- 3.5. PSL Flavors
- 3.6. PSL Keywords
- 3.7. PSL Layers & Flavors Example
- 3.8. PSL Statement Placement
- 3.9. Comments and Assertions
- 3.10. Verification Units



- 3.11. Always and Never Properties
- 3.12. Links to Formal Verification

## 4. Property Specification Language (PSL) Basic Syntax

- 4.1. PSL Recommended Subset
- 4.2. Boolean Expressions I
- 4.3. Boolean Expressions II
- 4.4. Boolean Expressions III
- 4.5. SERE I
- 4.6. SERE II
- 4.7. SERE III
- 4.8. SERE IV
- 4.9. Properties I
- 4.10. Properties II
- 4.11. Properties III
- 4.12. Properties IV
- 4.13. Liveness I
- 4.14. Liveness II
- 4.15. Glue Logic I
- 4.16. Glue Logic II

## 5. Coding Guidelines

- 5.1. Property Abstraction
- 5.2. Over-Constraining Assertions
- 5.3. Under-constraining Assertions
- 5.4. Fully Specifying Assertions
- 5.5. Overlapping Assertions
- 5.6. Assertions Overlap Example
- 5.7. Avoiding Overlap
- 5.8. Never Failing or Completing Properties
- 5.9. Restrictions on never
- 5.10. Implication: Fulfilling Conditions
- 5.11. Implication: Enabling Conditions
- 5.12. Using Verilog Text Replacement Macros
- 5.13. Verilog Conditional PSL Parsing
- 5.14. Guarded Properties
- 5.15. Naming Restrictions
- 5.16. Verification Components (vcomps)



## 6. Use of Auxiliary HDL Code

- 6.1. What Is Auxiliary Code
- 6.2. Auxiliary Code vs PSL
- 6.3. Auxiliary Code – Example 1 (I)
- 6.4. Auxiliary Code – Example 1 (II)
- 6.5. Auxiliary Code – Example 1 (III)
- 6.6. Auxiliary Code – Example 1 (IV)
- 6.7. Auxiliary Code – Example 2 (I)
- 6.8. Auxiliary Code – Example 2 (II)
- 6.9. Auxiliary Code – Example 2 (III)
- 6.10. Auxiliary Code – Example 3 (I)
- 6.11. Auxiliary Code – Example 3 (II)
- 6.12. Auxiliary Code – Example 3 (III)
- 6.13. Auxiliary Code – Example 4 (I)
- 6.14. Auxiliary Code – Example 4 (II)
- 6.15. Auxiliary Code – Example 5 (I)
- 6.16. Auxiliary Code – Example 5 (II)
- 6.17. Auxiliary Code – Example 5 (III)
- 6.18. Auxiliary Code – Example 6 (I)
- 6.19. Auxiliary Code – Example 6 (II)
- 6.20. Auxiliary Code – Example 6 (III)
- 6.21. Auxiliary Code – Example 7 (I)
- 6.22. Auxiliary Code – Example 7 (II)
- 6.23. Auxiliary Code – Example 7 (III)

## 7. Verification Units

- 7.1. What is a Verification Unit
- 7.2. Syntax Definition
- 7.3. Syntax Example (VHDL)
- 7.4. Syntax Example (Verilog)
- 7.5. Inheritance
- 7.6. Verification Unit Types
- 7.7. VUInt Binding
- 7.8. Default Verification Unit
- 7.9. Modeling Layer
- 7.10. Modeling Layer Example
- 7.11. Inheritance Binding and Modeling Layer
- 7.12. Applying Inheritance
- 7.13. Verification Unit Advantages



## Formal Verification [Full Course List](#)

1. Introduction to Formal Verification ([Available](#))
2. Formal Verification Fundamentals
3. SVA Coding for Formal Verification ([Available](#))
4. PSL Coding for Formal Verification ([Available](#))
5. Property Checking
6. Functional Signoff with Formal
7. Formal Verification Applications
8. Formal Verification: Auto Formal
9. Formal Verification: Formal Apps
10. Formal Verification: Unit Signoff
11. Formal Verification: Block Signoff
12. Formal Verification: System Signoff



## Resource: [Formal Verification of Properties in Hardware Design](#) (PDF)

1. Introduction
  - 1.1. Motivation for Embracing Formal Verification
  - 1.2. Target Audience and Learning Approach
  - 1.3. Chapter Structure and Practical Themes
2. Introduction to Formal Verification
  - 2.1. Design Flow and Verification Challenges
  - 2.2. Introduction to IC Design Complexity
  - 2.3. The Digital Design Process and Synthesis Flow
  - 2.4. The Role and Importance of Verification
  - 2.5. Misconceptions about Verification in Synthesis-Driven Design
  - 2.6. Verification Methods and the Rise of Formal Techniques
  - 2.7. Formal Techniques in Hardware Verification
  - 2.8. Definition and Objectives
  - 2.9. Categories of Formal Verification
  - 2.10. Theorem Proving
  - 2.11. Model Checking and Language Containment
  - 2.12. Equivalence Checking
  - 2.13. Combined Approaches and Design Transformations
  - 2.14. Fundamentals of Formal Analysis
  - 2.15. Concept of Formal Modeling
  - 2.16. Reachability and State Depth
  - 2.17. Proof Algorithm in Formal Verification
  - 2.18. Interpretation of Proof Results
  - 2.19. Factors Affecting Formal Proof Performance
  - 2.20. Internal Characteristics of Formal Engines
  - 2.21. Practical Needs of Formal Verification
  - 2.22. Correctness and Separation from Synthesis Tools
  - 2.23. Automation and User Guidance
  - 2.24. Performance and Predictability
  - 2.25. Integration with Design Environment
  - 2.26. Support for Debug and Error Localization
  - 2.27. Application to RTL and Gate-Level Verification
  - 2.28. Opportunities for Optimization and Design-Specific Strategies
3. Formal Verification Strategies Using Assertions
  - 3.1. Observing Failures and Coverage through Traces



- 3.2. Verification Result Confidence in Formal Analysis
- 3.3. Deterministic Verification Outcomes
- 3.4. Flexible Verification with Incomplete Methods
- 3.5. Bounded Formal Verification
- 3.6. Managing Abstraction and Approximation in Formal Verification
- 3.7. Understanding Approximation
- 3.8. Overapproximation and Its Implications
- 3.9. Underapproximation in Practice
- 3.10. The Problem of Contradictory Assumptions
- 3.11. Pruning and Model Simplification
- 3.12. Applying Formal Methods Throughout RTL Verification
- 3.13. Exhaustive Verification
- 3.14. Lightweight Verification
- 3.15. Early RTL Verification
- 3.16. Defining and Validating Interface Assumptions
- 3.17. The Role of Assumptions in Formal Verification
- 3.18. Challenges in Proving Assumptions Formally
- 3.19. Complementary Simulation-Based Validation
- 3.20. Formal Verification Efficiency
- 3.21. Hybrid Verification Approaches

## 4. Property Checking

- 4.1. Property Checking in Formal Verification
- 4.2. Understanding the Concept
- 4.3. Mathematical Completeness and Exhaustiveness
- 4.4. No Dependence on Testbenches
- 4.5. Formal Modeling and Property Evaluation
- 4.6. Combining with Simulation for Comprehensive Coverage
- 4.7. Enhancing Design Assurance with Property Checking
- 4.8. Flow of Property Checking
- 4.9. Value of Property Checking
- 4.10. Best Practices in Property Definition
- 4.11. Property-Based Verification Concepts
- 4.12. Categories of Properties
- 4.13. Assertions for Design Checking
- 4.14. Coverage Properties for Reachability
- 4.15. Environmental Assumptions
- 4.16. Property Specification and Formal Verification
- 4.17. Structure of Property Descriptions
- 4.18. Observability and Controllability
- 4.19. Flow of Formal Verification



## 5. Formal Property Verification

- 5.1. Understanding Formal Property Checking
- 5.2. Exploring Exhaustive Design Verification through Formal Methods
- 5.3. Defining the Role of Formal Properties
- 5.4. Exhaustive Verification with FPV
- 5.5. The Formal Checking Mechanism
- 5.6. Comprehensive Strategies in Formal and Dynamic Verification
- 5.7. Exploring Dynamic and Formal Approaches
- 5.8. Critical Considerations for Formal Efficiency
- 5.9. Applying Property-Based Verification
- 5.10. Formal Property Verification Flow and Applications
- 5.11. FPV Workflow
- 5.12. Key Inputs and Expected Outputs
- 5.13. Best Use Cases and Application Limits
- 5.14. Constructing a Formal Verification Environment
- 5.15. Cycle-Based Models
- 5.16. Input Constraints
- 5.17. End-to-End Assertions
- 5.18. Functional Coverage Properties
- 5.19. Formal Verification IP (VIP)

## 6. Coverage Driven Formal Property Verification

- 6.1. Exploring Dynamic Coverage Strategies
- 6.2. Enhanced Formal Coverage via Mutation Analysis
- 6.3. Mutation Coverage Principles
- 6.4. Interpreting Mutations
- 6.5. Types of Mutation-Based Coverage
- 6.6. FSM-Level Coverage Strategies
- 6.7. Source-Level Mutation Coverage
- 6.8. Exploring the Gap Between Structural and Behavioral Goals
- 6.9. Functional vs Structural Coverage
- 6.10. Focus of Existing Metrics
- 6.11. Limitations of Structural Metrics
- 6.12. Case Study: Gray Code Counter
- 6.13. The Need for Stronger Properties
- 6.14. Fault-based Specification Completeness Assessment
- 6.15. Limitations of Traditional Mutation Coverage
- 6.16. Specification Verification Without Implementation
- 6.17. Introducing a Fault-Driven Methodology
- 6.18. Coverage Defined by Fault Sensitivity
- 6.19. Clarifying Misuse of Structural Coverage



## 7. Formal Verification Techniques for Arbiter Designs

- 7.1. Arbitration Logic and Fairness
- 7.2. Types of Arbitration Mechanisms
- 7.3. Ensuring Liveness in Arbitration
- 7.4. Expressing and Validating Fairness Properties
- 7.5. Ensuring Grant Fairness
- 7.6. Simple Arbitration with Fairness Constraints
- 7.7. Assertion for Fair Arbitration
- 7.8. Fairness for Other Clients
- 7.9. Scaling Fair Arbitration
- 7.10. Arbiter-Specific Properties
- 7.11. Exclusive Grant Control
- 7.12. Latency Enforcement
- 7.13. Valid Grant Preconditions
- 7.14. Priority-Based Grant Control
- 7.15. Two-Client Priority Rule
- 7.16. Assertion Violation Example
- 7.17. Scaling to Multiple Clients
- 7.18. Automation of Priority Assertions
- 7.19. Dynamic Weighted Arbitration
- 7.20. Credit Mechanism and Arbitration Logic
- 7.21. Enforcing Credit Conditions Through Assertions
- 7.22. Preventing Starvation with Credit Fairness
- 7.23. Adaptive Client Prioritization
- 7.24. Event-Driven Priority Switching
- 7.25. Assertion-Based Enforcement
- 7.26. Structured Behavior Partitioning
- 7.27. Handling Overlapping Requests
- 7.28. Challenges with Traditional Assertions
- 7.29. Using Tags for Accurate Matching
- 7.30. Latency Awareness in Assertion Design
- 7.31. Implications for High-Performance Design
- 7.32. Designing a Reusable Assertion-Based Verification Block for an Arbiter
- 7.33. Interface Structure and Signal Roles
- 7.34. Behavioral Requirements in Natural Language
- 7.35. Creating the Assertion Interface
- 7.36. Adding Analysis Capabilities
- 7.37. Integrating and Reusing Assertion Logic

## 8. Modeling and Verifying Controllers with Formal Properties

- 8.1. Designing and Verifying a Basic Memory Controller
- 8.2. Simplified Memory Interface Logic
- 8.3. Overview of Interface and Signal Definitions



- 8.4. Functional Overview of SDRAM Controller Transactions
- 8.5. Supported Memory Transactions
- 8.6. State Control and Transitions
- 8.7. SDRAM Transaction Flows
- 8.8. Property Description in Natural Language
- 8.9. Assertion Integration in SDRAM Control
- 8.10. Signal Interface and Modport Connection
- 8.11. Communication with Analysis Environment
- 8.12. Error Status Propagation and Reporting
- 8.13. Signal Modeling for Assertion Readability
- 8.14. Naming Convention Consistency
- 8.15. Assertion Design for Controller Behavior
- 8.16. Encapsulate Assertions for Verification Use

## 9. Applying Formal Methods to Interface Verification

- 9.1. Designing Assertions for Serial Communication
- 9.2. Serial Interface Structure
- 9.3. Initiating and Executing Transactions
- 9.4. Master and Slave Roles
- 9.5. Addressing and Command Initiation
- 9.6. Acknowledgment and Data Exchange
- 9.7. Simplified Protocol Model
- 9.8. Defining Interface Behaviors
- 9.9. Assertion Integration with Bus Monitor
- 9.10. Structured Bus Communication Interface
- 9.11. Reporting Mechanism Through Analysis Port
- 9.12. Making Assertions Reusable
- 9.13. Serial Bus Reset Behavior
- 9.14. Detecting Start Signal Errors
- 9.15. Ensuring Order of Bus Events
- 9.16. Validating Transfer Sizes
- 9.17. Wrapping Assertions into Verification Components
- 9.18. Enabling Communication via Analysis Ports
- 9.19. Reusable and Scalable Integration

## 10. Designing and Verifying Datapath with Formal Properties

- 10.1. Efficient Queue Verification
- 10.2. Introduction to Queue Design and Usage
- 10.3. Queue Characteristics and Interface Simplicity
- 10.4. Assertion-Based Verification Approach
- 10.5. Simplified Queue Interface Overview



- 10.6. Queue Operations and Signal Behavior
- 10.7. Basic Enqueue and Dequeue Process
- 10.8. Clearing a Middle Entry
- 10.9. Behavioral Expectations in Natural Language
- 10.10. Assertion Integration in Monitor Design
- 10.11. Defining the Signal Interface
- 10.12. Connecting to Analysis Components
- 10.13. Defining Error Reporting Mechanisms
- 10.14. Assertion-Based Verification of Queue Behavior
- 10.15. Assertion Integration for Monitor Design



# ABV Series

## Introduction to Assertion Based Verification - SVA (Member Free)

### 1. Assertion

- 1.1. Specifying Properties
- 1.2. What is an Assertion
- 1.3. Defining Assertion
- 1.4. Assertions Monitor Design Properties
- 1.5. What are Assertions used for
- 1.6. What aren't Assertions used for
- 1.7. Why to Use Assertions
- 1.8. Who to Write Assertions
- 1.9. Where to Use Assertions
- 1.10. Issues with Assertions I
- 1.11. Issues with Assertions II

### 2. Introduction to SystemVerilog Assertions (SVA)

- 2.1. Terminology for SystemVerilog Assertion I
- 2.2. Terminology for SystemVerilog Assertion II
- 2.3. SVA Verification Directives
- 2.4. Why use SystemVerilog Assertions (SVA)
- 2.5. SystemVerilog Assertions Overview
- 2.6. Immediate Assertions
- 2.7. Limitations of Immediate Assertions
- 2.8. Concurrent Assertions
- 2.9. Building blocks of SVA
- 2.10. Steps to Create an SVA checker
- 2.11. SVA Assertion Structure
- 2.12. SVA Property Placement

### 3. SystemVerilog Assertions (SVA) Basic Syntax

- 3.1. SVA Recommended Subset
- 3.2. Boolean Expressions I
- 3.3. Boolean Expressions II
- 3.4. Boolean Expressions III
- 3.5. Sequences I
- 3.6. Sequences II
- 3.7. Sequences III



- 3.8. Sequences IV
- 3.9. Properties I
- 3.10. Properties II
- 3.11. Properties III
- 3.12. Properties IV
- 3.13. Liveness I
- 3.14. Liveness II
- 3.15. Glue Logic I
- 3.16. Glue Logic II

#### 4. Assertion-Based Verification (ABV) Methodology

- 4.1. Traditional Verification
- 4.2. Traditional Verification Disadvantages
- 4.3. Assertion-Based Solution
- 4.4. Advantages of Assertion-Based Verification
- 4.5. Before Assertion-Based Verification
- 4.6. After Assertion-Based Verification
- 4.7. Verification Testbench based on Assertion
- 4.8. Property specification in Assertion-Based Verification
- 4.9. Assertion in Simulation Testbench
- 4.10. Assertion in Formal Verification Testbench
- 4.11. Assertion-Based Verification Methodology

#### 5. Assertion-Based Verification (ABV) in Verification Tools

- 5.1. Coverage Driven Verification
- 5.2. Who Writes Assertions and Why
- 5.3. Introduction to Assertion-Based Verification
- 5.4. Assertion-Based Verification Flow
- 5.5. Assertion-Based Verification (ABV) in Formal
- 5.6. Assertion-Based Verification (ABV) in Simulation
- 5.7. Assertion-Based Verification (ABV) in Emulation/Acceleration
- 5.8. Assertion-Based Verification (ABV) in Functional Coverage
- 5.9. Dynamic & Formal Verification
- 5.10. Formal Verification Technology Factors
- 5.11. Simulator Overhead for Dynamic ABV
- 5.12. ABV in Plan to Closure Methodology



## Introduction to Assertion Based Verification - PSL (Member Free)

### 1. Assertion

- 1.1. Specifying Properties
- 1.2. What is an Assertion
- 1.3. Defining Assertion
- 1.4. Assertions Monitor Design Properties
- 1.5. What are Assertions used for
- 1.6. What aren't Assertions used for
- 1.7. Why to Use Assertions
- 1.8. Who to Write Assertions
- 1.9. Where to Use Assertions
- 1.10. Issues with Assertions I
- 1.11. Issues with Assertions II

### 2. Introduction to Property Specification Language (PSL)

- 2.1. Terminology for Property Specification Language I
- 2.2. Terminology for Property Specification Language II
- 2.3. Property Specification Language Overview
- 2.4. Layers of PSL
- 2.5. PSL Flavors
- 2.6. PSL Keywords
- 2.7. PSL Layers & Flavors Example
- 2.8. PSL Statement Placement
- 2.9. Comments and Assertions
- 2.10. Verification Units
- 2.11. Always and Never Properties
- 2.12. Links to Formal Verification

### 3. Property Specification Language (PSL) Basic Syntax

- 3.1. PSL Recommended Subset
- 3.2. Boolean Expressions I
- 3.3. Boolean Expressions II
- 3.4. Boolean Expressions III
- 3.5. SERE I
- 3.6. SERE II
- 3.7. SERE III
- 3.8. SERE IV
- 3.9. Properties I
- 3.10. Properties II



- 3.11. Properties III
- 3.12. Properties IV
- 3.13. Liveness I
- 3.14. Liveness II
- 3.15. Glue Logic I
- 3.16. Glue Logic II

## 4. Assertion-Based Verification (ABV) Methodology

- 4.1. Traditional Verification
- 4.2. Traditional Verification Disadvantages
- 4.3. Assertion-Based Solution
- 4.4. Advantages of Assertion-Based Verification
- 4.5. Before Assertion-Based Verification
- 4.6. After Assertion-Based Verification
- 4.7. Verification Testbench based on Assertion
- 4.8. Property specification in Assertion-Based Verification
- 4.9. Assertion in Simulation Testbench
- 4.10. Assertion in Formal Verification Testbench
- 4.11. Assertion-Based Verification Methodology

## 5. Assertion-Based Verification (ABV) in Verification Tools

- 5.1. Coverage Driven Verification
- 5.2. Who Writes Assertions and Why
- 5.3. Introduction to Assertion-Based Verification
- 5.4. Assertion-Based Verification Flow
- 5.5. Assertion-Based Verification (ABV) in Formal
- 5.6. Assertion-Based Verification (ABV) in Simulation
- 5.7. Assertion-Based Verification (ABV) in Emulation/Acceleration
- 5.8. Assertion-Based Verification (ABV) in Functional Coverage
- 5.9. Dynamic & Formal Verification
- 5.10. Formal Verification Technology Factors
- 5.11. Simulator Overhead for Dynamic ABV
- 5.12. ABV in Plan to Closure Methodology



# SVA/PSL Coding Series

## [SystemVerilog Assertion \(SVA\) - Fundamentals](#) (Member Free)

### 1. Assertion

- 1.1. Specifying Properties
- 1.2. What is an Assertion
- 1.3. Defining Assertion
- 1.4. Assertions Monitor Design Properties
- 1.5. What are Assertions used for
- 1.6. What aren't Assertions used for
- 1.7. Why to Use Assertions
- 1.8. Who to Write Assertions
- 1.9. Where to Use Assertions
- 1.10. Issues with Assertions I
- 1.11. Issues with Assertions II

### 2. Introduction to SystemVerilog Assertions (SVA)

- 2.1. Terminology for SystemVerilog Assertion I
- 2.2. Terminology for SystemVerilog Assertion II
- 2.3. SVA Verification Directives
- 2.4. Why use SystemVerilog Assertions (SVA)
- 2.5. SystemVerilog Assertions Overview
- 2.6. Immediate Assertions
- 2.7. Limitations of Immediate Assertions
- 2.8. Concurrent Assertions
- 2.9. Building blocks of SVA
- 2.10. Steps to Create an SVA checker
- 2.11. SVA Assertion Structure
- 2.12. SVA Property Placement

### 3. Fundamentals of Boolean Assertions

- 3.1. Defining Design Behavior
- 3.2. How to Name and Assert Properties
- 3.3. Property Clocking in SVA
- 3.4. Understanding Counter Intuitive Clock Behaviour
- 3.5. Clocked Property Evaluation
- 3.6. Counter Intuitive Clocks
- 3.7. Using DUT Clock Edges
- 3.8. Default Clock Usage



- 3.9. Placing Assertions
- 3.10. Same Cycle Implication
- 3.11. Next Cycle Implication
- 3.12. FSM Verification with SVA
- 3.13. FSM Assertion Checks
- 3.14. Understanding Assertion Overlapping
- 3.15. Edge-Triggered Functions
- 3.16. \$past Function
- 3.17. \$stable Function
- 3.18. \$countones() and \$isunknown() Functions
- 3.19. \$onehot() and \$onehot0() Functions

#### 4. Sequences: Structure and Applications

- 4.1. Sequence Operators and Features
- 4.2. Understanding Sequence Examples in SVA
- 4.3. Sequence Implication
- 4.4. Conditional and Unconditional Properties
- 4.5. Never Properties in SVA Assertions
- 4.6. Sequence Property Analysis
- 4.7. Edge-triggered Sequences with \$rose and \$fell
- 4.8. Handling Assertions with Disable Properties
- 4.9. Using Default Disable
- 4.10. Synchronous Abort Operators
- 4.11. Assertion Status
- 4.12. Cycle Delay Repetition
- 4.13. Cycle Delay Repetition Ranges
- 4.14. Consecutive Repetition
- 4.15. Consecutive Repetition with Ranges
- 4.16. Consecutive Repetition: Special Ranges
- 4.17. Non-Consecutive Repetition
- 4.18. Go-To Repetition
- 4.19. Non-Consecutive and Go-To Ranges
- 4.20. Non-Consecutive Repetition Range Example
- 4.21. Go-To Repetition Range Example
- 4.22. Repetition Shorthand
- 4.23. Property Abstraction
- 4.24. Challenges in Assertion Specification
- 4.25. Issues with Under-Specifying Assertions

#### 5. Utilizing Sequence Operators in Assertions

- 5.1. Introduction to Named Sequences



- 5.2. Sequence Clocking
- 5.3. Sequence Composition Operators
- 5.4. Sequence fusion Operator
- 5.5. Sequence or Operator
- 5.6. Sequence and Operator
- 5.7. Sequence intersect Operator
- 5.8. Sequence Operator Examples
- 5.9. first\_match Operator
- 5.10. first\_match Operator: Removing Undesired Failures
- 5.11. Sequence throughout Operator
- 5.12. Sequence within Operator

## 6. Understanding Coverage in Verification

- 6.1. Assessing Test Data Effectiveness
- 6.2. Understanding Coverage Metrics
- 6.3. Defining Functional Coverage
- 6.4. Using Cover Directive
- 6.5. Simulation vs. Formal Coverage
- 6.6. Cover Statement
- 6.7. Bus Protocol Example 1
- 6.8. Bus Protocol Example 2
- 6.9. Debugging Assertions with Coverage
- 6.10. Detecting Enabling Conditions
- 6.11. Cover Groups in SystemVerilog
- 6.12. Understanding Cover Properties
- 6.13. Understanding Cover Sequences
- 6.14. SVA LRM: 2012/2017 Changes
- 6.15. Backward Compatibility Issue with cover Verification Directive
- 6.16. Infinite Coverage in SVA

## 7. Fairness and Liveness in SVA

- 7.1. Strong vs. Weak Properties
- 7.2. Liveness vs. Safety
- 7.3. Liveness Properties
- 7.4. Default Strength
- 7.5. Standard Incompatibilities
- 7.6. Linear Temporal Logic (LTL) Operators
- 7.7. eventually vs. s\_eventually
- 7.8. Example of LTL Operators



## SystemVerilog Assertion (SVA) - Formal (Member Free)

### 1. Introduction to SystemVerilog Assertions (SVA)

- 1.1. Terminology for SystemVerilog Assertion I
- 1.2. Terminology for SystemVerilog Assertion II
- 1.3. SVA Verification Directives
- 1.4. Why use SystemVerilog Assertions (SVA)
- 1.5. SystemVerilog Assertions Overview
- 1.6. Immediate Assertions
- 1.7. Limitations of Immediate Assertions
- 1.8. Concurrent Assertions
- 1.9. Building blocks of SVA
- 1.10. Steps to Create an SVA checker
- 1.11. SVA Assertion Structure
- 1.12. SVA Property Placement

### 2. Fundamentals of Boolean Assertions

- 2.1. Defining Design Behavior
- 2.2. How to Name and Assert Properties
- 2.3. Property Clocking in SVA
- 2.4. Understanding Counter Intuitive Clock Behaviour
- 2.5. Clocked Property Evaluation
- 2.6. Counter Intuitive Clocks
- 2.7. Using DUT Clock Edges
- 2.8. Default Clock Usage
- 2.9. Placing Assertions
- 2.10. Same Cycle Implication
- 2.11. Next Cycle Implication
- 2.12. FSM Verification with SVA
- 2.13. FSM Assertion Checks
- 2.14. Understanding Assertion Overlapping
- 2.15. Edge-Triggered Functions
- 2.16. \$past Function
- 2.17. \$stable Function
- 2.18. \$countones() and \$isunknown() Functions
- 2.19. \$onehot() and \$onehot0() Functions

### 3. Sequences: Structure and Applications

- 3.1. Sequence Operators and Features
- 3.2. Understanding Sequence Examples in SVA



- 3.3. Sequence Implication
- 3.4. Conditional and Unconditional Properties
- 3.5. Never Properties in SVA Assertions
- 3.6. Sequence Property Analysis
- 3.7. Edge-triggered Sequences with \$rose and \$fell
- 3.8. Handling Assertions with Disable Properties
- 3.9. Using Default Disable
- 3.10. Synchronous Abort Operators
- 3.11. Assertion Status
- 3.12. Cycle Delay Repetition
- 3.13. Cycle Delay Repetition Ranges
- 3.14. Consecutive Repetition
- 3.15. Consecutive Repetition with Ranges
- 3.16. Consecutive Repetition: Special Ranges
- 3.17. Non-Consecutive Repetition
- 3.18. Go-To Repetition
- 3.19. Non-Consecutive and Go-To Ranges
- 3.20. Non-Consecutive Repetition Range Example
- 3.21. Go-To Repetition Range Example
- 3.22. Repetition Shorthand
- 3.23. Property Abstraction
- 3.24. Challenges in Assertion Specification
- 3.25. Issues with Under-Specifying Assertions

## 4. Fairness and Liveness in SVA

- 4.1. Strong vs. Weak Properties
- 4.2. Liveness vs. Safety
- 4.3. Liveness Properties
- 4.4. Default Strength
- 4.5. Standard Incompatibilities
- 4.6. Linear Temporal Logic (LTL) Operators
- 4.7. eventually vs. s\_eventually
- 4.8. Example of LTL Operators

## 5. Coding Guidelines

- 5.1. What Does Inefficient SVA Mean
- 5.2. Inefficient SVA in Simulation: Example
- 5.3. Can I Use the Same Properties in Simulation and Formal
- 5.4. What are the Symptoms of Inefficient SVA in Formal
- 5.5. Considerations for Efficient SVA
- 5.6. Simplifying Property – Example 1



- 5.7. Simplifying Property – Example 2
- 5.8. Simplifying Property – Example 3
- 5.9. Simplifying Property – Example 4
- 5.10. Express In Native Language
- 5.11. Keep Assertions Simple
- 5.12. Recommended Not to Be Used in Formal
- 5.13. Recommended SVA Coding Styles I
- 5.14. Recommended SVA Coding Styles II
- 5.15. Recommended SVA property modelling I
- 5.16. Recommended SVA property modelling II

## 6. Use of Auxiliary HDL Code

- 6.1. What Is Auxiliary Code
- 6.2. Auxiliary Code vs SVA
- 6.3. Auxiliary Code – Example 1 (I)
- 6.4. Auxiliary Code – Example 1 (II)
- 6.5. Auxiliary Code – Example 1 (III)
- 6.6. Auxiliary Code – Example 1 (IV)
- 6.7. Auxiliary Code – Example 2 (I)
- 6.8. Auxiliary Code – Example 2 (II)
- 6.9. Auxiliary Code – Example 2 (III)
- 6.10. Auxiliary Code – Example 3 (I)
- 6.11. Auxiliary Code – Example 3 (II)
- 6.12. Auxiliary Code – Example 3 (III)
- 6.13. Auxiliary Code – Example 4 (I)
- 6.14. Auxiliary Code – Example 4 (II)
- 6.15. Auxiliary Code – Example 5 (I)
- 6.16. Auxiliary Code – Example 5 (II)
- 6.17. Auxiliary Code – Example 5 (III)
- 6.18. Auxiliary Code – Example 6 (I)
- 6.19. Auxiliary Code – Example 6 (II)
- 6.20. Auxiliary Code – Example 6 (III)
- 6.21. Auxiliary Code – Example 7 (I)
- 6.22. Auxiliary Code – Example 7 (II)
- 6.23. Auxiliary Code – Example 7 (III)

## 7. Formal Property Verification (FPV)

- 7.1. Dynamic & Formal Verification
- 7.2. Formal Verification Technology Factors
- 7.3. Formal Property Verification(FPV)
- 7.4. FPV Process Flow



- 7.5. Inputs and Outputs for FPV
- 7.6. Creating FPV Testbench
- 7.7. FPV Testbench – Cycle-Based Models
- 7.8. FPV Testbench – Constraints
- 7.9. FPV Testbench – End-to-End Checkers
- 7.10. FPV Testbench – Functional Covers
- 7.11. FPV Testbench – Formal VIP
- 7.12. Where to Use FPV



## SystemVerilog Assertion (SVA) - Advanced (Member Free)

### 1. Assertion-Based Verification (ABV) Methodology

- 1.1. Traditional Verification
- 1.2. Traditional Verification Disadvantages
- 1.3. Assertion-Based Solution
- 1.4. Advantages of Assertion-Based Verification
- 1.5. Before Assertion-Based Verification
- 1.6. After Assertion-Based Verification
- 1.7. Verification Testbench based on Assertion
- 1.8. Property specification in Assertion-Based Verification
- 1.9. Assertion in Simulation Testbench
- 1.10. Assertion in Formal Verification Testbench
- 1.11. Assertion-Based Verification Methodology

### 2. Assertion-Based Verification (ABV) in Verification Tools

- 2.1. Coverage Driven Verification
- 2.2. Who Writes Assertions and Why
- 2.3. Introduction to Assertion-Based Verification
- 2.4. Assertion-Based Verification Flow
- 2.5. Assertion-Based Verification (ABV) in Formal
- 2.6. Assertion-Based Verification (ABV) in Simulation
- 2.7. Assertion-Based Verification (ABV) in Emulation/Acceleration
- 2.8. Assertion-Based Verification (ABV) in Functional Coverage
- 2.9. Dynamic & Formal Verification
- 2.10. Formal Verification Technology Factors
- 2.11. Simulator Overhead for Dynamic ABV
- 2.12. ABV in Plan to Closure Methodology

### 3. Mastering Advanced SVA Techniques

- 3.1. Assertion Evaluation Process
- 3.2. Detecting Sequence Endpoints
- 3.3. Triggered Sequence Method
- 3.4. Sequence Endpoints in SVA
- 3.5. Sequence Arguments
- 3.6. Property Arguments
- 3.7. Action Blocks in Assertions
- 3.8. Local Variable
- 3.9. Local Variable Example
- 3.10. Handling Overlapping Properties



- 3.11. Challenges with Local Variables as Arguments
- 3.12. Data Integrity Verification with Local Variables
- 3.13. SVA Data Integrity Assumptions
- 3.14. Using Multi-clocked Sequences
- 3.15. Using Multi-clocked Properties
- 3.16. Procedural Block Assertions
- 3.17. Complex Property Example
- 3.18. Simplifying Complex Clock Expressions
- 3.19. Property Clocking Tips

## 4. Advanced Property Constructs in SVA

- 4.1. Property Forming Constructs
- 4.2. Using until Operator
- 4.3. Using until\_with Operator
- 4.4. Using s\_until Operator
- 4.5. Using s\_until\_with Operator
- 4.6. until Operator Pros & Cons
- 4.7. Handling Input Livelocks
- 4.8. Resolving Input Livelocks Using always
- 4.9. Understanding the Followed-By Operator
- 4.10. Meaning of the Followed-By Operator
- 4.11. Liveness Covers
- 4.12. implies Operator
- 4.13. Understanding implies with  $\Rightarrow$
- 4.14. if-else Property Expression Syntax
- 4.15. Practicality of if-else
- 4.16. if-else Example with Sequence on RHS
- 4.17. if-else Example with Sequence on LHS
- 4.18. Conjunction Property Expression Syntax
- 4.19. Practicality of Conjunction
- 4.20. Conjunction Property Example
- 4.21. Disjunction Property Expression Syntax
- 4.22. Practicality of Disjunction
- 4.23. Disjunction Property Example
- 4.24. iff Operator
- 4.25. nexttime Property Expression Syntax
- 4.26. Weak Indexed nexttime Property
- 4.27. Strong Indexed s\_nexttime Property
- 4.28. Practicality of nexttime Property



## 5. Efficient SVA Property Reuse

- 5.1. Sequence Arguments
- 5.2. Property Arguments
- 5.3. Assertion Generation
- 5.4. Verification Component Modules
- 5.5. Assertion Binding
- 5.6. Assertion Binding Example
- 5.7. Conditional Properties
- 5.8. Understanding Properties
- 5.9. Simulation of a Block
- 5.10. Simplifying Formal Verification
- 5.11. Nondeterministic Constants
- 5.12. Handling Formal Complexity

## 6. Verification Components (vcomp)

- 6.1. Verification Component – Defined
- 6.2. Verification Component – Example
- 6.3. Verification Component – Characteristics
- 6.4. Techniques for Placing Assertion
- 6.5. Technique Comparison with Methods
- 6.6. Anatomy of a Verification Component
- 6.7. Interface Verification Component Example
- 6.8. Verification Component Design Techniques
- 6.9. Verification Component Parameterization
- 6.10. Coverage Points
- 6.11. Benefits of Coverage Points
- 6.12. Utilizing Formal Verification IP

## 7. Strategies for Verification Completeness

- 7.1. Functional Verification Scale
- 7.2. Property Set Completeness
- 7.3. Verification Completeness Example
- 7.4. Completeness Metrics
- 7.5. Design Mutation Detection
- 7.6. Correcting Fundamental Logic Errors
- 7.7. Achieving Verification Completeness
- 7.8. Assessing Verification Completeness
- 7.9. Verification Execution Completeness
- 7.10. Automation in Verification



# Property Specification Language (PSL) - Fundamentals (Member Free)

## 1. Assertion

- 1.1. Specifying Properties
- 1.2. What is an Assertion
- 1.3. Defining Assertion
- 1.4. Assertions Monitor Design Properties
- 1.5. What are Assertions used for
- 1.6. What aren't Assertions used for
- 1.7. Why to Use Assertions
- 1.8. Who to Write Assertions
- 1.9. Where to Use Assertions
- 1.10. Issues with Assertions I
- 1.11. Issues with Assertions II

## 2. Introduction to Property Specification Language (PSL)

- 2.1. Terminology for Property Specification Language I
- 2.2. Terminology for Property Specification Language II
- 2.3. Property Specification Language Overview
- 2.4. Layers of PSL
- 2.5. PSL Flavors
- 2.6. PSL Keywords
- 2.7. PSL Layers & Flavors Example
- 2.8. PSL Statement Placement
- 2.9. Comments and Assertions
- 2.10. Verification Units
- 2.11. Always and Never Properties
- 2.12. Links to Formal Verification

## 3. Boolean Expressions

- 3.1. Creating Boolean Properties
- 3.2. Naming and Asserting Properties
- 3.3. Using Labels in Assertions
- 3.4. Issues With Labels
- 3.5. Using report in PSL
- 3.6. VHDL Boolean Expressions
- 3.7. VHDL Flavor Properties
- 3.8. Verilog Flavor Properties
- 3.9. Clocked Properties



- 3.10. Counter Intuitive Clock Behaviour
- 3.11. Clocked vs Unclocked Properties
- 3.12. Default Clock Statement
- 3.13. Conditional Assertion Check

## 4. PSL Foundation Language Basics

- 4.1. Next Cycle Checks
- 4.2. next[N] Repetition
- 4.3. eventually! Operator
- 4.4. Assertion Overlapping
- 4.5. until Operator
- 4.6. before Operator
- 4.7. Inclusive until\_ Operator
- 4.8. Inclusive before\_ Operator
- 4.9. abort Operator
- 4.10. async\_abort and sync\_abort
- 4.11. Bounding Operator Precedence
- 4.12. Termination Operator Precedence
- 4.13. Foundation Language Examples
- 4.14. Cascading Next
- 4.15. Sequences

## 5. Sequential Extended Regular Expressions

- 5.1. Understanding SERE
- 5.2. Conditional SERE Operators
- 5.3. SERE vs. Foundation Language
- 5.4. Conditional vs. Unconditional Properties
- 5.5. Assertion Status
- 5.6. SERE Analysis
- 5.7. SERE Definition
- 5.8. Consecutive SERE Repetition
- 5.9. Consecutive SERE Repetition with Ranges
- 5.10. Go-to and Non-consecutive Ranges
- 5.11. SERE Repetition: Special Ranges
- 5.12. Repetition without SERE
- 5.13. Cycle Repetition: Special Ranges
- 5.14. Non-Consecutive Repetition
- 5.15. Go-to Repetition
- 5.16. Non-Consecutive Repetition Range Example
- 5.17. Go-to Repetition Range Example
- 5.18. Repetition Examples



- 5.19. Abort with SERE
- 5.20. Named Sequences
- 5.21. Using Named Sequences
- 5.22. Named Conditions in PSL

## 6. Sequence Composition Operators

- 6.1. Sequence Composition Operators
- 6.2. Fusion Composition :
- 6.3. Or Composition |
- 6.4. Exclusive and Multiple SERE Or
- 6.5. Non-Length Matching And Composition &
- 6.6. Length Matching And Composition &&
- 6.7. Sequence Composition Examples
- 6.8. SERE vs. Foundation Language

## 7. Coverage in Verification

- 7.1. Assessing Test Data Effectiveness
- 7.2. Understanding Coverage Metrics
- 7.3. Defining Functional Coverage
- 7.4. Using Cover Directive
- 7.5. Named Cover Statements
- 7.6. Reducing Test Vectors
- 7.7. Practical Coverage Approach
- 7.8. Debugging Assertions with Coverage
- 7.9. Detecting Enabling Conditions
- 7.10. Parameterized Operators in Coverage



# Property Specification Language (PSL) - Formal (Member Free)

## 1. Introduction to Property Specification Language (PSL)

- 1.1. Terminology for Property Specification Language I
- 1.2. Terminology for Property Specification Language II
- 1.3. Property Specification Language Overview
- 1.4. Layers of PSL
- 1.5. PSL Flavors
- 1.6. PSL Keywords
- 1.7. PSL Layers & Flavors Example
- 1.8. PSL Statement Placement
- 1.9. Comments and Assertions
- 1.10. Verification Units
- 1.11. Always and Never Properties
- 1.12. Links to Formal Verification

## 2. Boolean Expressions

- 2.1. Creating Boolean Properties
- 2.2. Naming and Asserting Properties
- 2.3. Using Labels in Assertions
- 2.4. Issues With Labels
- 2.5. Using report in PSL
- 2.6. VHDL Boolean Expressions
- 2.7. VHDL Flavor Properties
- 2.8. Verilog Flavor Properties
- 2.9. Clocked Properties
- 2.10. Counter Intuitive Clock Behaviour
- 2.11. Clocked vs Unclocked Properties
- 2.12. Default Clock Statement
- 2.13. Conditional Assertion Check

## 3. PSL Foundation Language Basics

- 3.1. Next Cycle Checks
- 3.2. next[N] Repetition
- 3.3. eventually! Operator
- 3.4. Assertion Overlapping
- 3.5. until Operator
- 3.6. before Operator
- 3.7. Inclusive until\_ Operator
- 3.8. Inclusive before\_ Operator



- 3.9. abort Operator
- 3.10. async\_abort and sync\_abort
- 3.11. Bounding Operator Precedence
- 3.12. Termination Operator Precedence
- 3.13. Foundation Language Examples
- 3.14. Cascading Next
- 3.15. Sequences

## 4. Sequential Extended Regular Expressions

- 4.1. Understanding SERE
- 4.2. Conditional SERE Operators
- 4.3. SERE vs. Foundation Language
- 4.4. Conditional vs. Unconditional Properties
- 4.5. Assertion Status
- 4.6. SERE Analysis
- 4.7. SERE Definition
- 4.8. Consecutive SERE Repetition
- 4.9. Consecutive SERE Repetition with Ranges
- 4.10. Go-to and Non-consecutive Ranges
- 4.11. SERE Repetition: Special Ranges
- 4.12. Repetition without SERE
- 4.13. Cycle Repetition: Special Ranges
- 4.14. Non-Consecutive Repetition
- 4.15. Go-to Repetition
- 4.16. Non-Consecutive Repetition Range Example
- 4.17. Go-to Repetition Range Example
- 4.18. Repetition Examples
- 4.19. Abort with SERE
- 4.20. Named Sequences
- 4.21. Using Named Sequences
- 4.22. Named Conditions in PSL

## 5. Coding Guidelines

- 5.1. Property Abstraction
- 5.2. Over-Constraining Assertions
- 5.3. Under-constraining Assertions
- 5.4. Fully Specifying Assertions
- 5.5. Overlapping Assertions
- 5.6. Assertions Overlap Example
- 5.7. Avoiding Overlap
- 5.8. Never Failing or Completing Properties



- 5.9. Restrictions on never
- 5.10. Implication: Fulfilling Conditions
- 5.11. Implication: Enabling Conditions
- 5.12. Using Verilog Text Replacement Macros
- 5.13. Verilog Conditional PSL Parsing
- 5.14. Guarded Properties
- 5.15. Naming Restrictions
- 5.16. Verification Components (vcomps)

## 6. Use of Auxiliary HDL Code

- 6.1. What Is Auxiliary Code
- 6.2. Auxiliary Code vs PSL
- 6.3. Auxiliary Code – Example 1 (I)
- 6.4. Auxiliary Code – Example 1 (II)
- 6.5. Auxiliary Code – Example 1 (III)
- 6.6. Auxiliary Code – Example 1 (IV)
- 6.7. Auxiliary Code – Example 2 (I)
- 6.8. Auxiliary Code – Example 2 (II)
- 6.9. Auxiliary Code – Example 2 (III)
- 6.10. Auxiliary Code – Example 3 (I)
- 6.11. Auxiliary Code – Example 3 (II)
- 6.12. Auxiliary Code – Example 3 (III)
- 6.13. Auxiliary Code – Example 4 (I)
- 6.14. Auxiliary Code – Example 4 (II)
- 6.15. Auxiliary Code – Example 5 (I)
- 6.16. Auxiliary Code – Example 5 (II)
- 6.17. Auxiliary Code – Example 5 (III)
- 6.18. Auxiliary Code – Example 6 (I)
- 6.19. Auxiliary Code – Example 6 (II)
- 6.20. Auxiliary Code – Example 6 (III)
- 6.21. Auxiliary Code – Example 7 (I)
- 6.22. Auxiliary Code – Example 7 (II)
- 6.23. Auxiliary Code – Example 7 (III)

## 7. Formal Property Verification (FPV)

- 7.1. Dynamic & Formal Verification
- 7.2. Formal Verification Technology Factors
- 7.3. Formal Property Verification(FPV)
- 7.4. FPV Process Flow
- 7.5. Inputs and Outputs for FPV
- 7.6. Creating FPV Testbench



- 7.7. FPV Testbench – Cycle-Based Models
- 7.8. FPV Testbench – Constraints
- 7.9. FPV Testbench – End-to-End Checkers
- 7.10. FPV Testbench – Functional Covers
- 7.11. FPV Testbench – Formal VIP
- 7.12. Where to Use FPV



# Property Specification Language (PSL) - Advanced (Member Free)

## 1. Assertion-Based Verification (ABV) Methodology

- 1.1. Traditional Verification
- 1.2. Traditional Verification Disadvantages
- 1.3. Assertion-Based Solution
- 1.4. Advantages of Assertion-Based Verification
- 1.5. Before Assertion-Base Verification
- 1.6. After Assertion-Base Verification
- 1.7. Verification Testbench based on Assertion
- 1.8. Property specification in Assertion-Based Verification
- 1.9. Assertion in Simulation Testbench
- 1.10. Assertion in Formal Verification Testbench
- 1.11. Assertion-Based Verification Methodology

## 2. Assertion-Based Verification (ABV) in Verification Tools

- 2.1. Coverage Driven Verification
- 2.2. Who Writes Assertions and Why
- 2.3. Introduction to Assertion-Based Verification
- 2.4. Assertion-Based Verification Flow
- 2.5. Assertion-Based Verification (ABV) in Formal
- 2.6. Assertion-Based Verification (ABV) in Simulation
- 2.7. Assertion-Based Verification (ABV) in Emulation/Acceleration
- 2.8. Assertion-Based Verification (ABV) in Functional Coverage
- 2.9. Dynamic & Formal Verification
- 2.10. Formal Verification Technology Factors
- 2.11. Simulator Overhead for Dynamic ABV
- 2.12. ABV in Plan to Closure Methodology

## 3. Mastering Advanced PSL Techniques

- 3.1. Built-in PSL Functions
- 3.2. ended() Function
- 3.3. countones() Function
- 3.4. isunknown() Function
- 3.5. onehot() and onehot0() Functions
- 3.6. stable() Function
- 3.7. stable() over Multiple Cycles
- 3.8. nondet() and nondet\_vector() Functions
- 3.9. Using ended() in SERE



- 3.10. Advanced Use of ended()
- 3.11. Parameterized Sequences
- 3.12. Nested Sequence Example
- 3.13. HDL Type Parameters
- 3.14. Parameterized Properties
- 3.15. Replicated Properties
- 3.16. Using Replication
- 3.17. Property Declaration Using for
- 3.18. Property Declaration: for vs forall
- 3.19. Parameterized Sequence Operators
- 3.20. Understanding Macro Capabilities
- 3.21. Using Macros for Assertion Generation
- 3.22. Comparison: %for, forall, and for
- 3.23. Understanding Local Variables
- 3.24. Utilizing Local Variables in PSL

## 4. Understanding Clocking Properties

- 4.1. Evaluating Properties Timing
- 4.2. Delta Cycle Analysis
- 4.3. Inactive Clock Edge Evaluation
- 4.4. Creating Strobes
- 4.5. Complex Property Example
- 4.6. Simplifying Complex Clock Expressions
- 4.7. rose() vs. rising\_edge (VHDL)
- 4.8. rose() vs. posedge (Verilog)
- 4.9. Using Multi-clocked Sequences
- 4.10. Using Multi-clocked Properties
- 4.11. Counter Intuitive Clock Behaviour
- 4.12. Counter Intuitive Clock Behaviour Example
- 4.13. Unclocked Properties
- 4.14. Issues With Combinational Design Properties
- 4.15. Unclocked Properties with next
- 4.16. Clocking In Formal Verification
- 4.17. Property Clocking Tips

## 5. Verification Units

- 5.1. What is a Verification Unit
- 5.2. Syntax Definition
- 5.3. Syntax Example (VHDL)
- 5.4. Syntax Example (Verilog)
- 5.5. Inheritance



- 5.6. Verification Unit Types
- 5.7. VUInt Binding
- 5.8. Default Verification Unit
- 5.9. Modeling Layer
- 5.10. Modeling Layer Example
- 5.11. Inheritance Binding and Modeling Layer
- 5.12. Applying Inheritance
- 5.13. Verification Unit Advantages

## 6. Practical Property Writing

- 6.1. Assertions Ensure Correct Behavior of Design
- 6.2. Case 1: AMBA AHB Slave Interface
- 6.3. Simplified Single Transfer
- 6.4. Wait States
- 6.5. Incomplete Transfers
- 6.6. AMBA AHB Incomplete Transfer Assertion
- 6.7. Four Beat Incrementing Burst Example
- 6.8. Step 1: Select Alternative Behaviors
- 6.9. Step 2: Define Sequences
- 6.10. Step 3: Build Compound Sequences
- 6.11. Step 4: Define Enabling Condition
- 6.12. Step 5: Check Coverage
- 6.13. Case 2: Asynchronous FIFO Design
- 6.14. FIFO Black-Box Property Checks
- 6.15. Asynchronous FIFO Implementation
- 6.16. FIFO White-Box Property Checks

## 7. Property Set Completeness

- 7.1. Scale of the Functional Verification Problem
- 7.2. Completeness of Property Sets
- 7.3. Verification in Completeness Example
- 7.4. Completeness Metrics
- 7.5. Design Mutation
- 7.6. We Don't Even Have the Fundamentals Correct
- 7.7. Can We Achieve Completeness?
- 7.8. Assessing Completeness
- 7.9. Verification Execution Completeness
- 7.10. What Can Be Automated?

