EDA Academy
Verification Course Outline

(updated in April 2025)

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

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

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

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

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

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

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: srandom()

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

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

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

SystemVerilog Language - Coverage (Member Free)

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

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

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

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

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

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

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

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)

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

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-Base Verification
4.6. After Assertion-Base 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-Base Verification
4.6. After Assertion-Base 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

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

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-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 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

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

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

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