SUMMARY OF OSVVM

Base Components of OSVVM

1. Verification Framework

- Test sequencers (TestCtrl) organise processes into test cases
- Each test case is created as a different architecture of TestCtrl.
- Verification components (VCs) use interface signalling protocols.
- Structured similarly to the RTL design and have explicit component connections.

2. Transaction Interfaces and APIs

- Transaction interfaces implemented as VHDL records with "inout" ports
- Transaction APIs implemented as VHDL procedures
- Model Independent Transactions (MIT) form standard interfaces and APIs for related protocols:
    - Address Bus MIT for memory interfaces (e.g. AXI4, Avalon).
    - Stream MIT for stream interfaces (AxiStream, UART, etc.).
- With MIT Verification Componet, Test Case Component development become faster.

3. Test Sequencer (TestCtrl)

- Controls test execution flow
- Contains multiple concurrent processes (one per interface typically)
- Each process handles transactions for its respective interface
- Architecture allows combining directed tests, constrained random tests, scoreboards, and functional coverage

4. Verification Utility Library

- **RandomPkg**: Support for constrained random testing
- **CoveragePkg**: Functional coverage definition and collection
- **ScoreboardPkg**: Self-checking test mechanisms
- **AlertLogPkg**: Error reporting and message filtering
- **MemoryPkg**: Memory modeling and simulation
- **ResolutionPkg**: Resolution functions for interfaces

- **TranscriptPkg**: Transcript file handling
- **TbUtilPkg**: Basic testbench utilities

OSVVM Testing

Features of a Test Case

- All tests in a file.
- Includes control process that start and stop the test.
- Write explicit sequences using transaction API calls
- Test case = Send, Get, and Check transactions + affirmations (checks)
- Affirmations signal pass/fail status and track error counts

Protocol and Parameter Checks

- Built-in AlertLogPkg for protocol violation detection
- Configurable alert levels: FAILURE, ERROR, WARNING
- Control mechanisms for alerts: StopCount, PrintCount, Enable/Disable

Constrained Random Testing

- RandomPkg supports various randomization methods:
    - Range-based: RandInt(Min, Max)
    - Set-based: RandInt((1,2,3,5,7,11))
    - Exclusion sets: RandInt(0, 15, Exclude => (5,11))
    - Weighted distribution: DistInt((70, 10, 10, 5, 5))

Scoreboards

- Self-checking mechanism for data transformation validation
- Implemented as specialized FIFOs with checking capabilities
- Uses package generics to support different data types
- Handles out-of-order execution and dropped values

Functional Coverage

- Tracks test plan completeness
- Two types supported:

    - Item coverage (point coverage): tracks values within a single object
    - Cross coverage: tracks relationships between independent objects
- Simplifies coverage definition, collection, and reporting

Reporting Features

1. Build Summary Report (HTML)

- Summary of the entire build process
- Overview of each test suite and test case
- Links to detailed test reports

2. Test Case Reports (HTML)

- Alert reports showing errors and warnings
- Functional coverage reports
- Scoreboard reports
- Links to simulation transcripts

3. HTML'ized Simulation Transcript

- Errors highlighted in red
- Collapsible sections for easy navigation
- Makes scanning through thousands of lines efficient

4. Requirements Report

- Shows requirements coverage status

Scripting System

- Tool-independent scripts that work across multiple simulators
- Library management, compilation, and simulation commands
- Support for GHDL, NVC, Aldec, Siemens, Synopsys VCS, Cadence Xcelium
- Include mechanism for script modularity

SUMMARY OF UVVM

1. Introduction to UVVM

- UVVM is an open-source VHDL verification methodology designed to improve verification efficiency and design quality.
- Provides a structured infrastructure and architecture for verification.
- Runs on any VHDL-2008 compliant simulator.

2. The Verification Challenge and UVVM's Solution

According to the 2022 Wilson Research Group Functional Verification Study:

- Nearly 50% of project time is consumed by verification activities
- Half of verification time is spent debugging issues

UVVM directly targets these challenges through structured testing methodologies that enhance both quality and efficiency. UVVM provides solutions by

- Structured architecture for improved organization and readability
- Enhanced debugging capabilities through detailed logging and tracing
- Modular components that promote reusability across projects
- Simplified test creation with powerful abstraction mechanisms

3. Features and Capabilities

Utility Functions

Functions for stability checking, clock generation, randomization, message logging, etc.

- Built-in procedures like:
- check_stable(), await_stable()
- clock_generator(), gen_pulse()
- random(), normalize_and_check()
- set_log_file_name(), set_alert_file_name()

Bus Functional Models (BFMs)

- Provides free BFMs for protocols like UART, AXI4-lite, SPI, I2C, Avalon, GPIO, GMII, RGMII and utility interfaces like Clock Generator and Error Injector.
- BFMs simplify data communication between testbenches and DUT (Device Under Test).

Each BFM provides standard procedures for communicating with specific interfaces, greatly simplifying testbench development .However, BFMs have limitations - primarily that a process can only execute one BFM at a time.

VHDL Verification Components (VVCs)

VVCs extends BFMs functionalities

- Architecture: Each VVC consists of:
    - Interpreter: Analyzes incoming commands and determines appropriate actions
    - Command Queue: Stores and manages pending operations
    - Executor: Fetches commands from the queue and executes them via BFMs
- Advanced Features:
    - Delay insertion between commands (critical for testing cycle-related corner cases)
    - Command queuing for handling multiple operations
    - Completion detection for complex sequences
    - Activity registration for monitoring interface status
    - Multicast and broadcast capabilities for simultaneous control
    - Transaction information for detailed logging
    - Local sequencers for executing predefined command sequences
- Extensibility:
    - Easy addition of protocol checkers (bit-rate, frame-rate, gap checkers)
    - Simple integration of local sequencers for higher-level commands
    - Support for split transactions and out-of-order execution

4. Advanced UVVM Features

ESA Extension

- Scoreboards
- Monitors
- Randomization Control: For structured test generation

- Error Injection: Both brute force and protocol-aware approaches
- Property Checkers
- Transaction Info
- Watchdog: Both simple and activity-based implementations
- Hierarchical VVCs: For complex protocol stacks
- Specification Coverage: For requirement-to-test traceability

Advanced Scoreboard-Based Testbenches

- Generic Data Type Support: Works with any data type
- Queue Management: Insert, delete, fetch operations
- Counting:
    - Entered, pending, matched, mismatched,dropped, deleted, initial garbage
- Configuration Options:
    - Allow lossy
    - Allow out-of-order execution
    - Ignore initial mismatches

Specification Coverage

UVVM provides a comprehensive approach to specification coverage:

1. Requirement Specification: Document all requirements
2. Coverage Reporting: Report coverage from test sequencer or other testbench components
3. Summary Generation: Generate comprehensive reports including:
    - Coverage percentage per requirement
    - Test cases covering each requirement
    - Requirements covered by each test case
    - Accumulated coverage across multiple test cases

Enhanced Randomization and Functional Coverage (2020-2022)

- **Enhanced Randomization**:
    - Advanced randomization with a simple syntax
    - Example: addr <= my_addr.rand_range_weight((0,18,4),(19,31,1));
- **Optimized Randomization**:
    - Randomization without replacement
    - Weighted according to target distribution AND previous events

    - Minimizes the number of randomizations needed for a given target
- **Functional Coverage**:
    - Comprehensive bin definitions
    - Automatic tracking of coverage metrics
    - Multiple report formats for different analysis needs