Intelligent Coverage

- Key Benefits
  - Works in any VHDL testbench
  - Wraps well with other test approaches (directed, algorithmic, file)
  - Recommended to be used with transaction based testbenches
  - Readable by all (in particular RTL engineers)

- Low cost solution to leading edge verification
- Packages are FREE
- Works with VHDL simulators that support minimal VHDL2008

What is Functional Coverage?

- Code that observes execution of your test plan
- Tracks requirements, features, and boundary conditions
- Item Coverage (aka Point Coverage)
  - Track relationships within a single object
  - Bins of transfer sizes: 1, 2, 3, 4-128, 128-252, 252-254, 254
- Cross Coverage
  - Track relationships between multiple objects
  - Has each part of the registers been used with the ALU?
- Why not just Code Coverage?
  - Does not correlate independent items: Bins & Register Pairs
- Key Benefits
  - Required for randomization
  - Recommended for any complex design
  - Test Done = 100% Functional Coverage + 100% Code Coverage

Refinement of Intelligent Coverage

- Goal: Generates N Unique Test Cases in N Randomizations
- Benefit: generates each test case only once time = Intelligent Testbench

Intelligent Coverage Methodology

- While not Allocated
  - High, Reg(1) = IntelligentHandout; 
  - if Reg(1) = Alloc then
  - IntelligentHandout(Reg(1), Reg(2));
  - Alloc(Reg(1));
- else
  - -- in previous and following diagonal
  - IntelligentHandout(Reg(1), Reg(2));
  - Reg(2) = IntelligentHandout(Reg(1), Reg(2));

Constrained Random is Too Slow!

- Constrained random (CR) tests do uniform randomization (VHDL & SV)
- Uniform distributions repeat before generating all cases
- In general, to generate N cases, it takes O(N log N) randomizations
- The uniform randomization in ALU test requires 315 test iterations.
- (If there are approximately 50% at any iteration (96 test cases)
- The "log N" factor significantly slows down constrained random tests.

Intelligent Coverage

- Goal: Generates N Unique Test Cases in N Randomizations
- Benefit: generates each test case only once time = Intelligent Testbench

Intelligent Coverage

- Use either directed, algorithmic, file-based or randomization methods
- Randomize a value in an initiative range, 0 to 15, except S & 11
- Randomize a value within the set {1, 2, 3, 5, 7, 11}, except S & 11
- Randomize Weighted Value, Value = 0-11
- Randomize Weighted Value: Value = (1, 2, 3, 5, 7, 11)

Random Walk across Functional coverage holes
- Randomization only selects bins in the Functional Coverage

Coverage Closure

- Closure = Cover all legal bins in the coverage model
- Intelligent Coverage
- White FC
- Only selects bins that are not covered in the FC
- Coverage depends on running test long enough
- Tests partitioned based on what coverage we want in this test
- Constrained Random
- White CR, White FC.
- All CR controls are open-loop – No FC information is available during sim
- After simulation, merge FC of all tests
- Analyze tests and prune out those that are not increasing FC
- Tests partitioned based on modified controls, constraint sets, and seeds
- Intelligent Coverage is less work than Constrained Random

OSVVM: Advanced Verification for VHDL

- OSVVM consists of packages + methodology for:
  - Constrained Random (CR)
  - Functional Coverage (FC)
  - Intelligent Coverage – Random test generation using FC holes

- Key Benefits
  - Works in any VHDL testbench
  - Wraps well with other test approaches (directed, algorithmic, file)
  - Recommended to be used with transaction based testbenches
  - Readable by all (in particular RTL engineers)

- Low cost solution to leading edge verification
- Packages are FREE
- Works with VHDL simulators that support minimal VHDL2008

Basic Randomization

- Randomize a value in an initiative range, 0 to 15, except S & 11
- Randomize a value within the set {1, 2, 3, 5, 7, 11}, except S & 11
- Randomize Weighted Value, Value = 0-11
- Randomize Weighted Value: Value = (1, 2, 3, 5, 7, 11)

Basic Randomization

- Code patterns create constraints for CR tests
- Randomize values, transactions, and sequences of transactions
- Example: Weighted selection of test sequences (CR)

Writing Functional Coverage

- Testing an ALU with Multiple Inputs:
  - Generates the same test using Intelligent Coverage

Randomization

- Can be used in the testbench
  - Use either directed, algorithmic, file-based or randomization methods

- Random Walk across functional coverage holes
  - Randomization only selects bins in the Functional Coverage

Closure depends on running test long enough.
- Only selects bins that are not covered in the FC

Additional Pieces of VHDL Verification

- TLM = Abstract Inheritance + Transaction Models (entity/architecture)

OSVVM Summary

- Simple, Powerful, Concise Methodology = Intelligent Coverage (IT)
- Define Functional Coverage
- Randomize across coverage holes
- Refine with directed, algorithmic, CR, or IT based methods
- Better than SystemVerilog / V Capabilities
  - Functional Coverage – Incremental, Conditional
  - Intelligent Testbenches built in
  - FC, CR, and IT that can be refined with code
  - Extensible, just add to the packages
  - Works in any VHDL environment – including TLM
  - Supports mixed approaches (directed, algorithmic, file, CR, IT)

© 2014 SynthWorks Design Inc. Jim@Synthworks.com OSVVM was developed by SynthWorks. This poster is derived from SynthWorks’ class, VHDL Testbenches and Verification.