VHDL's OSVVM, The Death of SystemVerilog?

by

Jim Lewis

SynthWorks VHDL Training

Jim@SynthWorks.com

SynthWorks

SynthWorks

VHDL's OSVVM, the Death of SystemVerilog?

This material is derived from SynthWorks' class, VHDL Testbenches and Verification

This material is updated from time to time and the latest copy of this is available at http://www.SynthWorks.com/papers

Contact Information
Jim Lewis, President
SynthWorks Design Inc
11898 SW 128th Avenue
Tigard, Oregon 97223
503-590-4787
jim@SynthWorks.com

www.SynthWorks.com

VHDL's OSVVM, the Death of SystemVerilog?

- OSVVM is a step ahead of SystemVerilog and UVM

Topics
- What is OSVVM, …?
- Constrained Random (CR) Methodology is 2X More Work
- Writing Functional Coverage is Easy
- Constrained Random is 5X or More Slower
- Intelligent Coverage
- OSVVM is More Capable
- Randomization in OSVVM
- OSVVM Loves Any Testbench
- Transactions are more than Structure
- Objections to VHDL
- OSVVM Summary

What is OS-VVM?

- Open Source VHDL Verification Methodology

- Packages + Methodology for:
  - Functional Coverage (FC)
  - Constrained Random (CR)
  - Intelligent Coverage - Test generation using FC holes

- Leading edge verification for your VHDL team
  - Mixes well with other approaches (directed, algorithmic, file, random)
  - Works in any VHDL testbench
  - Readable by All (in particlar RTL engineers)

- Low cost solution to leading edge verification
  - Works with regular VHDL simulators
  - Packages are FREE

What is Functional Coverage?

- Code that observes execution of your test plan
  - Tracks requirements, features, and boundary conditions
  - Model interface and design requirements
  - Required for randomized tests.
- Point Coverage (aka Item Coverage)
  - Track relationships within a single object
  - Bins of values, such as transfer sizes:
    - 1, 2, 3, 4-127, 128-252, 253, 254, 255
- Cross Coverage
  - Track relationships between multiple objects
  - Has the each pair of registers been used with the ALU?
- Test Done =
  - 100 % Functional Coverage + 100 % Code Coverage

What is Constrained Random?

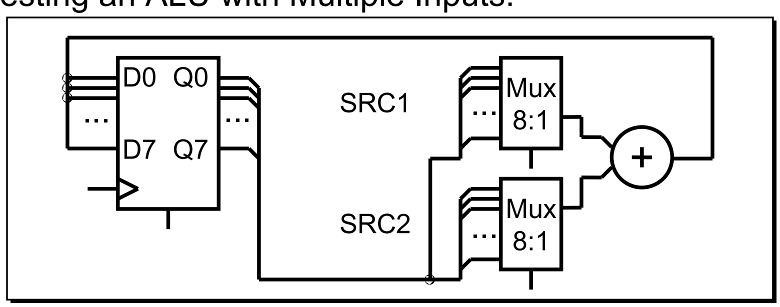
- Constrained Random (CR)
  - Models input values, transactions, and/or sequences using constraints
  - Constraints can be equations (SV) or code (VHDL)
  - SystemVerilog uses a solver to balance the randomization

CR is 2X More Work than OSVVM

- Constrained Random (CR) Methodology
  - Write a randomization constraint model
  - Write a functional coverage model
  - Generate stimulus by randomizing using randomization constraints
- OSVVM Intelligent Coverage Methodology
  - Write a functional coverage model
  - Generate stimulus by randomizing across holes in the FC model
- Result:
  - CR: 2 models: Randomization Constraints + FC
  - OSVVM: Only need a FC Model. Less Work (2X?)

Writing Functional Coverage

- Testing an ALU with Multiple Inputs:



- Need to test every register in SRC1 with every register in SRC2

SRC2

		R0	R1	R2	R3	R4	R5	R6	R7
	R0								
	R1								
S	R2								
R	R3								
C	R4								
1	R5								
	R6								
	R7								

Writing Functional Coverage

```
architecture Test3 of tb is
  shared variable ACov : CovPType ;
begin
 CollectCov : process
   variable RV : RandomPType ; -- randomization object
   variable Src1, Src2 : integer ;
 begin
   ACov.AddCross( GenBin(0,7), GenBin(0,7) );

   while not ACov.IsCovered loop

    Src1 := RV.RandInt(0, 7) ;
    Src2 := RV.RandInt(0, 7) ;

    ACov.ICover( ( Src1, Src2 ) ) ;

    DoAluOp(TRec, Src1, Src2) ;
  end loop ;
  ACov.WriteBin ;
  EndStatus(. . . ) ;
end process ;
```

Coverage Object

Create Cross Coverage Bins

Covered = Done

Uniform Randomization

Collect Coverage

Do Transaction

Functional Coverage with OSVVM is as concise as language syntax.

Constrained Random is 5X or More Slower

- Constrained random (CR) is at best a uniform randomization
  - 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.
  - 315 is approximately 5X too many iterations (64 test cases)
  - The "log N" factor significantly slows down constrained random tests.

SRC1 \ SRC2	R0	R1	R2	R3	R4	R5	R6	R7
R0	6	6	9	1	4	6	6	5
R1	3	4	3	6	9	5	5	4
R2	4	1	5	3	2	3	4	6
R3	5	5	6	3	3	4	4	6
R4	4	5	5	10	9	10	7	7
R5	4	6	3	6	3	5	3	8
R6	3	6	3	4	7	1	4	6
R7	7	3	4	6	6	5	4	5

- "From Volume to Velocity" shows CR tests that are 10X to 100X too slow

Intelligent Coverage

- Randomly select holes in Functional Coverage Model
  - "Coverage driven randomization" - but term is misused by others

- Goal: Generate N Unique Test Cases in N Randomizations
  - Same goal of Intelligent Testbenches

SRC2

SRC1	R0	R1	R2	R3	R4	R5	R6	R7
R0	1	1	1	1	1	1	1	1
R1	1	1	1	1	1	1	1	1
R2	1	1	1	1	1	1	1	1
R3	1	1	1	1	1	1	1	1
R4	1	1	1	1	1	1	1	1
R5	1	1	1	1	1	1	1	1
R6	1	1	1	1	1	1	1	1
R7	1	1	1	1	1	1	1	1

Intelligent Coverage

Same test using Intelligent Coverage

```
architecture Test3 of tb is
  shared variable ACov : CovPType ;
begin
 CollectCov : process
   variable Src1, Src2 : integer ;
 begin
  ACov.AddCross(GenBin(0,7), GenBin(0,7) );
  while not ACov.IsCovered loop

    (Src1, Src2) := ACov.RandCovPoint ;

    ACov.ICover( ( Src1, Src2 ) ) ;
    DoAluOp(TRec, Src1, Src2) ;
  end loop ;
  ACov.WriteBin ; -- Report Coverage
  EndStatus(. . . ) ;
end process ;
```

Intelligent Coverage Randomization

Runs 64 iterations @ 5X faster

Refinement of Intelligent Coverage

- Refinement can be as simple or complex as needed
- Use either directed, algorithmic, file-based or randomization methods.

```
while not ACov.IsCovered loop

  (Reg1, Reg2) := ACov.RandCovPoint ;

  if Reg1 /= Reg2 then
    DoAluOp(TRec, Reg1, Reg2) ;
    ACov.ICover( (Reg1, Reg2) ) ;

  else
    -- Do previous and following diagional
    DoAluOp(TRec, (Reg1-1) mod 8, (Reg2-1) mod 8) ;
    DoAluOp(TRec,  Reg1, Reg2 ) ;
    DoAluOp(TRec, (Reg1+1) mod 8, (Reg2+1) mod 8) ;

    -- Can either record all or select items
    ACov.ICover( (Reg1, Reg2) ) ;
  end if ;

end loop ;
```


OSVVM is More Capable

- Functional Coverage is a data structure
  - Incremental additions supported
  - Captured sequentially - use any code (if, loops, …)
- Each bin can have a different coverage goal
  - Goal = Number of times of value must occur to be covered
  - Coverage goals are also used as randomization weights
- Different coverage goal for each Src1 crossed with any Src2

```
--                Goal   Src1         Src2
ACov.AddCross(     1,    GenBin(0),   GenBin(0,7) ) ;
ACov.AddCross(     2,    GenBin(1),   GenBin(0,7) ) ;
ACov.AddCross(     3,    GenBin(2),   GenBin(0,7) ) ;
ACov.AddCross(     4,    GenBin(3),   GenBin(0,7) ) ;
ACov.AddCross(     5,    GenBin(4),   GenBin(0,7) ) ;
ACov.AddCross(     6,    GenBin(5),   GenBin(0,7) ) ;
ACov.AddCross(     7,    GenBin(6),   GenBin(0,7) ) ;
ACov.AddCross(     8,    GenBin(7),   GenBin(0,7) ) ;
```

Randomization in OSVVM

- Implemented in RandomPkg
- Randomize a value in an inclusive range, 0 to 15, except 5 & 11

```
Data1 := RV.RandInt(Min => 0, Max => 15) ;
Data2 := RV.RandInt(0, 15, (5,11) );  -- except 5 & 11
```

- Randomize a value within the set (1, 2, 3, 5, 7, 11), except 5 & 11

```
Data3 := RV.RandInt( (1,2,3,5,7,11) );
Data4 := RV.RandInt( (1,2,3,5,7,11), (5,11) );
```

- Weighted Randomization: Value + Weight

```
. . .                -- ((val1, wt1), (val2, wt2), ...)
Data5 := RV.DistValInt( ((1,7), (3,2), (5, 1)) );
```

- Weighted Randomization: Weight, Value = 0 .. N-1

```
Data6 := RV.DistInt ( (7, 2, 1) ) ;
```


OSVVM Supports Randomization

- OSVVM uses code patterns to create constraints
  - Example: Weighted selection of paths (test sequences)

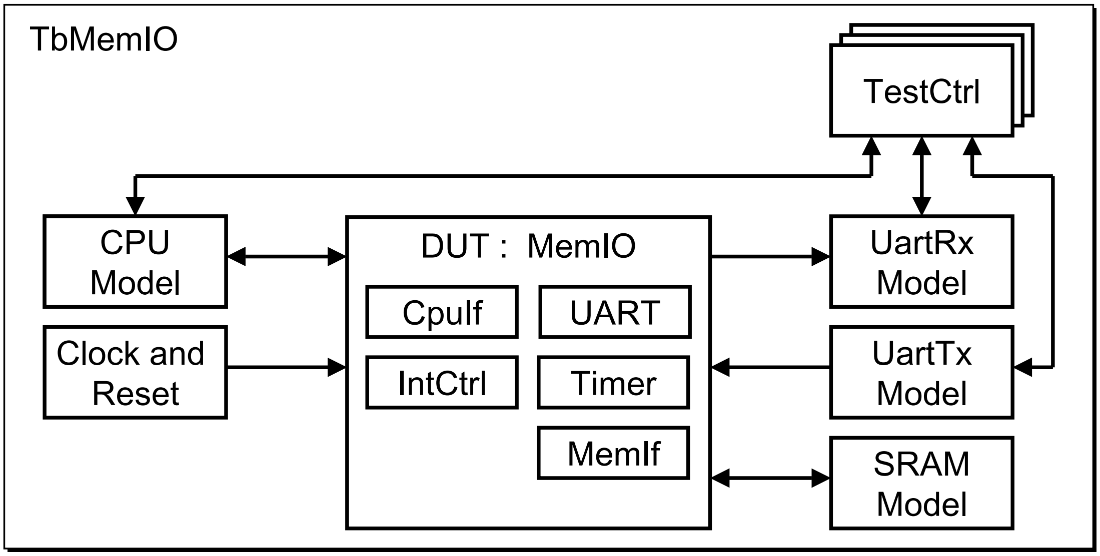
```
StimGen : while TestActive loop     -- Repeat until done
  case RV.DistInt( (7, 2, 1) ) is
    when 0 =>  -- Normal Handling   -- 70%
      . . .
    when 1 =>  -- Error Case 1      -- 20%
      . . .
    when 2 =>  -- Error Case 2      -- 10%
      . . .
```

  - See the RandomPkg Users Guide for more examples and seed setting

- Code patterns can create a constrained random test environment, however,
  - OSVVM uses Intelligent Coverage as the primary randomization
  - Code patterns are used primarily as refinement.

OSVVM Loves Any Testbench

- We prefer transaction based testbenches



- Similar structure to other verification languages
  - Uses regular entities & architectures. Records on interfaces.
  - No OO required.

Transactions are more than Structure

- Structure: Encapsulate Interface Functionality in a Model

```
-- CPU Write
nAds <= '0' after tpd, '1' after tperiod + tpd ;
Addr <= ADDR0 after tpd ;
Data <= X"A5A5" after tperiod + tpd ;
Wr_nRd <= '0' after tpd ;
wait until nRdy = '0' and rising_edge(Clk) ;
```

- Abstract Initiation: Use a procedure to initiate a transaction

```
. . .
CpuWrite(CpuRec, ADDR0, X"A5A5");
CpuRead (CpuRec, ADDR0, DataO);
. . .
```

- Result: Test is more readable
  - Testbench is not the exclusive domain of verification engineers
  - Simplifies writing tests.

Objections to VHDL

- No Solver
  - Intelligent coverage more effective than the best solver
- No Fork & Join
  - Fork & Join are for sequential programming - writing threads.
  - Concurrent programming uses handshaking (like hardware)
    - More effective at bundling of a models intent
- No OO
  - Functional Coverage and Randomization needs data structures not OO
- No Factory Class
  - Factory classes allow swapping of implementations in OO
  - Architectures give the same capability for concurrent programming

OSVVM Summary

- Intelligent Coverage = Simple, Powerful Methodology
  - Define Functional Coverage
  - Randomize across coverage holes
  - Refine with directed, algorithmic, file-based or CR methods
- Faster
  - Test construction: Focus on FC (2X faster)
  - Simulations: No redundant stimulus (5X faster) and No solver
- OSVVM
  - Goes beyond other verification languages (SV and 'e')
  - Readable by All (Verification and RTL engineers)
  - Works in any VHDL environment – in part or whole
  - Is Free – Open Source
- Downloads: http://www.synthworks.com/blog/osvvm
- SystemVerilog? Why bother!

SynthWorks VHDL Training

Comprehensive VHDL Introduction 4 Days
http://www.synthworks.com/comprehensive_vhdl_introduction.htm
A design and verification engineer's introduction to VHDL syntax, RTL coding, and testbenches. Students get VHDL hardware experience with our FPGA based lab board.

VHDL Testbenches and Verification 5 days - OS-VVM bootcamp
http://www.synthworks.com/vhdl_testbench_verification.htm
Learn the latest VHDL verification techniques including transaction-based testing, bus functional modeling, self-checking, data structures (linked-lists, scoreboards, memories), directed, algorithmic, constrained random and coverage driven random testing, and functional coverage.

VHDL Coding for Synthesis 4 Days
http://www.synthworks.com/vhdl_rtl_synthesis.htm
Learn VHDL RTL (FPGA and ASIC) coding styles, methodologies, design techniques, problem solving techniques, and advanced language constructs to produce better, faster, and smaller logic.

SynthWorks offers on-site, public venue, and on-line classes. See:
http://www.synthworks.com/public_vhdl_courses.htm

Going Further / References

- Jim's Blog: www.synthworks.com/blog
- OSVVM Website: www.osvvm.org
- Coverage Package Users Guide and Random Package Users Guide
- "From Volume to Velocity" by Walden Rhines of Mentor Graphics, Keynote speech for DVCon 2011.
  - See http://www.mentor.com/company/industry_keynotes/