Intelligent Coverage Driven, modern verification for VHDL based designs in native VHDL with OSVVM

Vijay Mukund Srivastav\textsuperscript{1}, Anupam Maurya\textsuperscript{2}, Prabhat Kumar\textsuperscript{3}, Juhi\textsuperscript{4}, VerifLabs\textsuperscript{1,2}, VerifWorks\textsuperscript{3}, Vecima Networks\textsuperscript{4}
 Agenda

• VHDL usage in the industry
• Modern DV challenges
• Functional Coverage
• Constrained Random generation
• Assertion Based Verification
• Looking ahead: UVM-VHDL?
VHDL usage in the industry

- VHDL is preferred HDL for:
  - FPGA
  - Some ASICs
  - Defence labs
  - Aerospace etc.

- Modern FPGAs are multi-million gate equivalent

- So verifying them in simulation is critical
Recent FPGA survey

What do you typically use to debug & verify your FPGA design?

- Manual RTL code reviews
- Simulation
- Code coverage
- Equivalence checking
- Assertion-based verification
- UVM
- In-lab testing
- Emulation

Table: % of res. using FPGA for ASIC prototyping vs. % of res. using FPGA as target technology

Source: EXOSTIV LABS

© Accellera Systems Initiative
Modern design-verification

• DV (Design-Verification) as a field has grown by leaps and bounds
  – Directed Testing $\rightarrow$ Constrained Random Verification (CRV)
  – Waveform check $\rightarrow$ Assertion Based Verification (ABV)
  – Manual test list $\rightarrow$ Coverage Driven Verification (CDV)

• Common technologies – applicable to all HDL based designs
  • SystemVerilog/UVM – popular for Verilog/SV RTL
  • VHDL $\rightarrow$ Equally capable language as SystemVerilog
  • UVM $\rightarrow$ OSVVM is an option for VHDL designs
What is OSVVM?

- Open Source VHDL Verification Methodology.
- Implemented in VHDL-2008 or VHDL-2002
- Works with regular VHDL simulators.
- Packages are FREE.
- Packages + Methodology for:
  - Functional Coverage (FC)
  - Constrained Random (CR)
  - Intelligent Coverage - Test generation using FC holes
- Mixes well with other approaches (directed, algorithmic, file, random)
- Works in any VHDL test bench.
- Readable by All (in particular RTL engineers).
OSVVM in a nut-shell

- AlertLogPkg → Messaging utilities (a la UVM – Log)
- CoveragePkg → API for Functional Coverage definitions
- RandomPkg → Constrained Random generation
- VerifWorks is adding a UVM-layer on top of OSVVM
  - uvm_info* is now available!
  - Will be donated if there is enough industry interest
- Assertions in native VHDL (erstwhile PSL) and/or SVA
AXI4-Lite Memory controller

- AXI4-Lite protocol
- RTL in VHDL
- VIP implemented using OSVVM
- VerifWorks’s AXI4-Lite Assertion-IP is plugged-in
  - PSL version
  - Native in VHDL 2008
What is Functional Coverage?

• A process in which user specifies the goal explicitly – “what to look for”
• OSVVM provides a package with APIs to capture the “goal”
• User has to identify “what to cover” and also “code them” using the new APIs – Coverage Model
• A tool then monitors the simulation run, annotates the “coverage model” on-the-fly
FCOV Process

Identify the coverage plan (based on DUT Spec, test plan)

For each plan-item, identify
• “when” to cover
• “what” to cover

Code the coverage model using OSVVM API

Run simulation, collect coverage

Analyze the coverage results

Refine cov model

Add more tests
OSVVM Coverage Package API

```pascal
procedure AddBins (AtLeast : integer ; CovBin : CovBinType );

procedure AddCross(
    AtLeast     : integer ;
    Bin1, Bin2 : CovBinType ;
    Bin3, Bin4, Bin5, Bin6, Bin7, Bin8, Bin9, Bin10, Bin11, Bin12, Bin13,
    Bin14, Bin15, Bin16, Bin17, Bin18, Bin19, Bin20 : CovBinType := NULL_BIN
);

function GenBin(AtLeast : integer ; Min, Max, NumBin : integer ) return CovBinType;

function GenBin(Min, Max : integer) return CovBinType ;

function GenBin(A : integer) return CovBinType ;
```
AXI4-Lite Memory Controller FCOV model

```vhdl
architecture f cov of vw_axi4_lite_fcov is

-- Constants
constant AXI_OKAY : std_logic_vector(1 downto 0) := "00";
constant AXI_DECERR : std_logic_vector(1 downto 0) := "11";

-- Coverage objects
shared variable waddr_cov : CovPType;
shared variable raddr_cov : CovPType;
shared variable rd_wr_kind_cov : CovPType;
shared variable addr_cross_rw_cov : CovPType;

begin

  def_fcov_model_p : process
  begin
    waddr_cov.AddBins(GenBin(Min => RW_ST, Max => RW_END, NumBin => 10));
    waddr_cov.AddBins(GenBin(Min => RO_ST, Max => RO_END, NumBin => 10));
    waddr_cov.AddBins(GenBin(Min => WO_ST, Max => WO_END, NumBin => 10));
    waddr_cov.AddBins(GenBin(Min => RSRV_ST, Max => RSRV_END, NumBin => 10));
    raddr_cov.AddBins(GenBin(Min => RW_ST, Max => RW_END, NumBin => 10));
    raddr_cov.AddBins(GenBin(Min => RO_ST, Max => RO_END, NumBin => 10));
    raddr_cov.AddBins(GenBin(Min => WO_ST, Max => WO_END, NumBin => 10));
    raddr_cov.AddBins(GenBin(Min => RSRV_ST, Max => RSRV_END, NumBin => 10));
    rd_wr_kind_cov.AddBins(GenBin(Min => 0, Max => 1));
    wait;
  end process def_fcov_model_p;
```

© Accellera Systems Initiative
Cross coverage

```vhdl
-- create coverage model
ACov.AddCross( GenBin(0, 7), GenBin(0, 7) );  -- Model

while not ACov.IsCovered loop  -- Done?
    Src1 := RV.RandInt(0, 7);  -- Uniform Randomization
    Src2 := RV.RandInt(0, 7);
    DoAluOp(TRec, Src1, Src2);  -- Transaction
    ACov.ICover( (Src1, Src2) );  -- Accumulate
end loop;

ACov.WriteBin;  -- Report
```
Random vs. Directed testing

• Directed tests exercise a specific scenario
  – You direct the test
  – You explicitly orchestrate the interactions
  – What if you miss something?
• Injecting randomness exposes corner cases
• Multiple adoption strategies
• Specify variables to be randomized
• Tool generates random values
Constraint Random generation

- Pure random is hardly useful – takes too long to do something useful
- Need constraints to create interesting scenarios
- Procedural constraints are flexible and powerful
  - Simple flow
  - Most engineers understand procedural programming
  - Easy to debug
  - Native in VHDL language
- Use OSVVM.RandomPkg.all;
Using set membership in constraints

• Restrict the values inside a defined “set”

• Applications:
  – CSR – Configuration & Status Registers – values, modes etc.
  – Test modes
  – instructions etc.

\[
v_{\text{pkt\_len}} := \text{rng\_v.RandUnsigned} (\text{64, 128, 256, 1023})
\]

-- Randomly select a value within a set of values
impure function RandUnsigned (A : integer_vector ; Size : natural) return Unsigned;
Constrained Random Gen – OSVVM style

```plaintext
-- Generate a value in range 0 to 255
DataInt := RV.RandInt(0, 255);

...  

-- Generate a value in range 1 to 9 except exclude values 2,4,6,8
DataInt := RV.RandInt(1, 9, (2,4,6,8));

...

-- Generate a value in set 1,3,5,7,9
DataInt := RV.RandInt( (1,3,7,9) ); -- note two sets of parens required
...

-- Generate a value in set 1,3,5,7,9 except exclude values 3,7
DataInt := RV.RandInt((1,3,7,9), (3,7));
```

Source: OSVVM User Guide
CRV-CDV dilemma

- $\Omega$ - “very large” sample set.
- $\pi$ - probability distribution over $\Omega$.

**Goal:** Sample points $x \in \Omega$ at random from distribution $\pi$. 

© Accellera Systems Initiative 2015
Intelligent Coverage driven Random Generation

- Use FCOV model as a guide to random generation
- Works well for input, stimulus coverage space
- Randomly select holes in Functional Coverage Model

```verilog
-- Reset the DUT
--

uvm_info("Resetting the DUT");
axi_reset;
rng_v.InitSeed(rng_v'instance_name); // Initialize Seed. Typically done once.

for i in 1 to 40 loop
    -- The below gives regular RANDOM, O (N log(N))
    v_addr := rng_v.RandUnsigned(0, 40, 32);
    -- Using intelligent random we achieve in N iterations
    v_addr_int := waddr_cov.RandCovPoint;
    v_addr := to unsigned(v_addr_int, 32);
    v_wdata := rng_v.RandUnsigned(0, 511, 32);
    bus_write(v_addr, v_wdata, v_bus_error);
    bus_read(v_addr, v_rdata, v_bus_error);
    if (v_rdata /= v_wdata) then
        uvm_info("Error:RD Data:" & integer" & " & integer" & ";WR Data: " & integer" & " & " & "v_addr: " & integer" & ");
    end if;
end loop; // i
end_of_sim <= '1';
wait for 10 ns;
finish;
wait;
end process;
```

Cov.RandCovPoint - API
Intelligent CDV

Use FCOV model as a guide to random generation

Randomly select holes in Functional Coverage Model

Achieves in “n” iterations
Assertion Based Verification in VHDL

- Temporal assertions capability from IEEE 1850-PSL ➔ VHDL 2008
- All the modern verification technologies are now available to VHDL users natively
  - No additional cost (of a mixed language simulator for instance).
UVM-VHDL via OSVVM?

• UVM layer on top of OSVVM packages?
• OSVVM provides good packages - like SV’s
  – Constrained Random features
  – Coverage model etc.
• PSL provides ABV features in VHDL (a la SVA)
• UVM is popular in SV world, can we add UVM-VHDL?
UVM-VHDL layer around OSVVM

- VerifWorks has developed basic UVM package
- Covers messaging
- Run time verbosity control (a la +uvm_verbosity)

<table>
<thead>
<tr>
<th>OSVVM</th>
<th>UVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALWAYS</td>
<td>UVM_NONE</td>
</tr>
<tr>
<td>PASSED</td>
<td>UVM_LOW</td>
</tr>
<tr>
<td>FINAL</td>
<td>UVM_MEDIUM</td>
</tr>
<tr>
<td>INFO</td>
<td>UVM_HIGH</td>
</tr>
<tr>
<td>DEBUG</td>
<td>UVM_FULL</td>
</tr>
</tbody>
</table>
Questions ?