An Assertion Based Approach to Implement VHDL Functional Coverage

Michael Wazlowski, Susan Eickhoff, Michael Debole – IBM Corp.
Tagbo Ekwueme-Okoli – Cadence Design Systems
Agenda

• Introduction
• Background
• Method
• Results
• Q & A
Introduction

• Methodologies like SV/UVM have become the de facto standard for Coverage Driven Verification

• In a VHDL mixed cycle and event simulation environment the costs of conversion are prohibitive

• This presentation describes a process which leverages UCIS to manage functional coverage in this environment
Introduction

- Chips have become more complex
- With the increases in complexity, enhancements to functional verification methodologies have become necessary
- Coverage Driven Verification and Verification Planning and Management are two of these enhancements
Introduction

• Teams are often forced to use ad-hoc methods to manage increasing amounts of data.
Unified Coverage Interoperability Standard (UCIS)

- Ultimate goal: Universally recognizable and accessible coverage data
- Several benefits flow from this goal
- Two transfer methods

**Coverage producers**
- Simulation
- Static Checks
- Formal Verification
- Emulation

**Coverage consumers**
- Report generation
- RTL Annotation
- Test plan update

Ultimate goal: Universally recognizable and accessible coverage data

Several benefits flow from this goal

Two transfer methods
BugSpray

- IBM Created VHDL coverage extension
- Designers define interesting events to track

```vhdl
[count; event.event_name_0 ; clk] : (comment)
<= signal_a AND NOT signal_b ;
```
Method

Report parsed and XML file generated

XML read and coverage DB generated
Method

- Each report line -> One coverage point
- Hierarchy built from information

• Variable classes identified and stored
• Variable counts identified and stored

• Hierarchy tree completed
Method

XML file read
• XML checked for validity vs schema

Coverage DB generated
• Coverage model built
• Coverage data generated
XML vs API

- **XML**
  - Primarily for data transformation
- **API**
  - Primarily for coverage database access
XML

- Not performance optimized
- But impact can be managed
- Meaning of coverage data can be inferred
API

• Performance optimized
• Meaning of coverage data must be defined
• Requires changes to source code
XML vs API

• The XML-based interchange format was used
  – Ease of implementation/use
  – Ease of Interpretation
  – Portability
Highly generalized model of coverage may be stated as:

```event if (condition) counter++
```

- Variable value change
- Statement execution
- Covergroup sample
- Sample value
- Sequence of signals
- SV bins
- Assertion status
- Any count!
Mapping Coverage Data

• Initially mapped BugSpray coverage to assertions
  – Existence of PSL to BugSpray conversion tool
  – No concept of crosses
• Final decision was to map to the UCIS covergroup construct
  – Allowed the mapping of comments and variable classes
Mapping Coverage Data

• For explanation let’s use the SV covergroup
  – Its constructs closely match the UCIS general model

```
bit [9:0] v_a;
covergroup cg @(posedge clk);
coverpoint v_a;
endgroup

[count; event.event_name_0 ; clk ] : (comment) <= signal_a AND NOT signal_b ;
```
• Team can now leverage automation to free up verification resources
Special Thanks

• IBM – Jeff Plate
• Cadence – Guarav Singh, John Brennan
Q&A

• Any Questions?