February 28 – March 1, 2012

BRINGING CONTINUOUS DOMAIN INTO SYSTEMVERILOG COVERGROUPS

by

<table>
<thead>
<tr>
<th>Prabal K Bhattacharya, Swapnajit Chakraborti, Donald O’Riordan, Vaibhav Bhutani</th>
<th>Scott Little</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cadence Design Systems</td>
<td>Intel Corporation</td>
</tr>
</tbody>
</table>
Agenda

• Introduction to Functional Coverage
• Challenges posed by Analog and Mixed-Signal Verification
• Proposal for mixed-signal functional coverage
  – Based on SystemVerilog P1800 standard
• Real Valued coverage point
  – Language extension
  – Challenges and open issues
• Putting things together: a Voltage Detector Example
• Conclusion
Functional Coverage Basics

• Measure of “How Much”?
  – Of Design Functionality exercised by Verification Environment

• An expression of design intent, but can be developed independent from the design

• Coverage measurement can be done in the same breath as design verification
  – High Quality tests in less time

• SystemVerilog P1800 standard provides rich support for expressing and measuring such intent

• Well understood, established practice in digital verification world
What about Analog/ Mixed-Signal?

- Think Mixed-Signal SoC
- Complex integration of both discrete domain and continuous domain signals
- Analog verification
  - State space exhaustion problem
    - Operating modes
    - Process variation
    - Corners analysis
  - Verify all this in context of the SoC
- Need to extend traditional metric-driven verification techniques to continuous domain objects

Prabal K Bhattacharya et al., Cadence Design Systems. Scott Little, Intel Corporation
Functional Coverage for Mixed-Signal

• Two primary drivers
  – Analog effects must be captured in the coverage analysis
  – Functional coverage model must be reusable as design configurations change

• Need formal definition of mixed-signal functional coverage

• Choice of language
  – SystemVerilog represents an overwhelming majority

• Choice of data type
  – Floating point for representing continuous domain behavior

• Proposal to extend SystemVerilog coverage point to support real data type
Real Valued Coverage Point

• We propose to extend a SystemVerilog coverage point specification to be
  – An integral or real valued expression
• Addressing the real number continuum challenge
  – Coverpoint of type real must have user-defined bins having
    • A range, or
    • A finite set of real numbers
  – Automatic bin declaration is disallowed

covergroup g1 @(posedge clk);
  c: coverpoint var;
endgroup

Not legal if var is real
Real Valued Coverpoint: Building Blocks

• Range specification

<table>
<thead>
<tr>
<th>Open</th>
<th>Closed</th>
<th>Half Open/Half Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a:b)</td>
<td>[a:b]</td>
<td>[a:b)</td>
</tr>
</tbody>
</table>

Example

<table>
<thead>
<tr>
<th>For any r completely inside the range:</th>
</tr>
</thead>
<tbody>
<tr>
<td>a &lt; r &lt; b</td>
</tr>
<tr>
<td>a ≤ r ≤ b</td>
</tr>
<tr>
<td>a ≤ r &lt; b</td>
</tr>
<tr>
<td>a &lt; r ≤ b</td>
</tr>
</tbody>
</table>

• How to slice a range into bins: range_precision
  - A new option introduced to divide a given range and create automatic bins
  - Instance-specific covergroup option
  - Positive real number
  - Only needed for vector bins
Real Valued Coverpoints: Scalar Bins

- Contains either a singleton value or a range
- For a singleton value, value of the coverpoint must *exactly* match
  - More on this later
- For range specification, value of coverpoint needs to satisfy the range expression
  - Range can be (half-)open or (half-)closed

```
covergroup g1 @(posedge clk);
  bins b = {0, [0.5:0.8], 1.0};
endgroup
```
Real Valued Coverpoints: Vector Bins

- A vector bin with
  - Range $<a:b>$
    - $a$, $b$ are real
    - $<$ and $>$ are left and right bounds
      - Can be open ($($ or $)$) or closed ($[$ or $]$)
  - range_precision option set to $r$

- Will result in bins
  $<a : a+r), [a+r : a+2r), ... [a+m*r : b>$
  $m=n-1$, where $n=$number of bins created
Real Valued Coverpoints: Vector Bins

• Let’s now look at a simple example

```vcd
covergroup g1 @(posedge clk);
  option.range_precision = 0.1;
bins b1[] = {[3.5:3.8]};
endgroup
```

• Applying the rules set forth, following bins will be created

<table>
<thead>
<tr>
<th>Bin Name</th>
<th>Range for scoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1[3.5:3.6)</td>
<td>[3.5:3.6)</td>
</tr>
<tr>
<td>b1[3.6:3.7)</td>
<td>[3.6:3.7)</td>
</tr>
<tr>
<td>b1[3.7:3.8]</td>
<td>[3.7:3.8]</td>
</tr>
</tbody>
</table>
Real Valued Coverpoints: End Point

- Follows SystemVerilog $ syntax
  - Left range: \(-DBL\_MAX\)
  - Right range: \(+DBL\_MAX\)
- Hardware/OS dependent
- Overflow possible for vector bins
  - \([a:b]\) where \(\text{abs}(a-b) > DBL\_MAX\)
  - Implementation may choose to error
- Construction of bin ranges will be done in the same way
  - Fixed size: range will be divided into equal parts
  - Unspecified size: range will be divided using range_precision option
Real Valued Coverpoints: Ignoring Bins

```plaintext
option.range_precision = 0.1;
coverpoint a {
  bins b1[] = {[2.4:2.8]};
  ignore_bins ig = {2.5};
}

option.range_precision = 0.1;
coverpoint b {
  bins b1[] = {[2.4:2.8]};
  ignore_bins ig = {[2.5:2.6]};
}

option.range_precision = 0.1;
coverpoint c {
  bins b1 = {[2.4:2.8]};
  ignore_bins ig = {[2.59:2.63],2.79};
}
```
Real Valued Coverpoints: Duplicate Values Across Bins

```c
option.range_precision = 0.2;
coverpoint a {
    bins b1[] = {[2.4:2.8], [2.5:3.0]};
}
```

- `b1[2.4:2.6]`
- `b1[2.6:2.8]`
- `b1[2.5:2.7]`
- `b1[2.7:2.9]`
- `b1[2.9:3.0]`

Prabal K Bhattacharya et al., Cadence Design Systems. Scott Little, Intel Corporation
Real Valued Coverpoints: Challenges and Open Issues

• Numerical difficulties with floating point numbers
  - Floating point numbers are specified using binary fractions
  - Round-off error expected dependent on available precision
  - Therefore name of a bin may not correspond to the exact value of the coverpoint
  - Round-off error in floating point arithmetic
    • Applying range_precision may yield unexpected results
Real Valued Coverpoints: Fudge Factor

- Idea is to use a tolerant numerical match rather than an exact match
  - Using a fudge factor as a new covergroup option
  - Use well-known floating point comparison algorithm such as `approximatelyEqual` (Knuth)

```c
bool approximatelyEqual(float a, float b, float fudge)
{
    return fabs(a - b) <= ((fabs(a) < fabs(b) ? fabs(b) : fabs(a)) * fudge);
}
```
Illustrating Real Valued Coverpoint

- Voltage detector circuit with SystemVerilog testbench
- Warning and error events when input signal crosses threshold
- Coverage measurement to monitor regions of operation for the input signal
- Constrained random real numbered stimulus
- Voltage detection performed using Verilog-AMS with wreal
Illustrating Real Valued Coverpoint

Driver Voltage Detector

Coverage
module top();

logic clk;
always #1 clk = ~clk;

initial
begin
clk = 0;
#10000 $finish;
end

wire [1:0] lvw_trim;
wire [1:0] lvd_trim;

tbDriver dr1(.*);
voltageDetector voltD(.lowVwarn(lowVwarn),
    .lowVdet (lowVdet ),
    .vDetect (vDetect ),
    .lvw trimmed(lvw trimmed),
    .lvd trimmed(lvd trimmed));

realCov rc1(.*);
endmodule

covergroup lvdLvwCombos @(posedge clk);

//Capture the various ranges for LVW
lvwValues: coverpoint vDetect {
    bins uLow = {$:1.97};
    bins low  = [1.98:1.99];
    bins med  = [2.0:2.69];
    bins high = [2.7:2.99];
    bins uHigh = {[3.0:$]};
}

//Capture the various ranges for LVD
lvdValues: coverpoint vDetect {
    bins uLow = {$:1.87};
    bins low  = [1.88:1.89];
    bins med  = [1.9:2.59];
    bins high = [2.6:2.89];
    bins uHigh = {[2.9:$]};
}

lvwCombos: cross lvw_trim, lvwValues;
lvdCombos: cross lvd_trim, lvdValues;
endgroup
Coverage Results

//Capture the various ranges for LVW
lvwValues: coverpoint vDetect {
    bins uLow = {{$:1.97}};
    bins low = {{1.98:1.99}};
    bins med = {{2.0:2.69}};
    bins high = {{2.7:2.99}};
    bins uHigh = {{3.0:$}};
}
Next Steps

• Future work to support real data type
    • Coverpoint expression having both real and integral variables
    • Transition/Wildcard bins with real valued coverpoints
    • Other items
  - Formal treatment of tolerance or “fudge factor” and naming of vector bins

• Plan for standardization of the proposal and work with the SV-EC committee
Conclusions

• Mixed-signal Verification is getting more complex than ever
• Analog effects need to be accounted for in the verification metrics
• Functional coverage for objects belonging to continuous domain is a key part of extending metric driven verification to mixed-signal
• Proposal considers extending SystemVerilog standard to support real valued coverpoint
• Most SystemVerilog coverage constructs and semantics extend naturally for real data type
  – Vector binning requires introducing a precision factor
• Overview of open challenges and next steps