Boosting SystemC-based Testbenches with Modern C++ and Coverage-Driven Generation

Hoang M. Le, Rolf Drechsler
University of Bremen, Germany
Agenda

• Motivation
• State of the Art (for SystemC)
• Modern C++11 API incl. Coverage
• Coverage-driven Generation
• Wrap-up
Motivation

• SystemC
  – IEEE 1666-2011
  – C++ modeling @ multiple levels of abstraction
  – Open-source reference simulator

• Verification of SystemC models
  – Constrained Random Verification Methodology (e.g. UVM for SystemC)
  – Randomization (CRAVE or SCV)
  – Coverage?
  – Automated Coverage Closure?

![Diagram of SystemC Testbench](image)

© Accellera Systems Initiative
State of the Art (for SystemC)

• Randomization (CRAVE 2.0)
  – SystemVerilog-inspired syntax
  – Random objects
  – Random variables
  – Hard/soft/distribution constraints
  – Efficient constraint solving
    • SMT backends
    • Constraint partitioning

  – Still somewhat cumbersome syntax
State of the Art (for SystemC)

• Coverage (SC VX or SVM)
  – SystemVerilog-inspired syntax
  – Covergroups
  – Coverpoints
  – Coverbins
  – Not expressive enough for complex coverage conditions
  – No automated coverage closure
Our proposed solution

• Using C++11 for more compact constraint syntax

• New coverage API connected to the expression layer of CRAVE 2.0
  – For better expressiveness (same as of the constraint layer)
  – Allow automated coverage closure / coverage-driven generation (coverage expression handled as a special type of constraint in the constraint solving process)
Modern C++11 API incl. Coverage

• Constraint layer
• Automated creation of named object hierarchy
• C++11 allows in-place initialization

```cpp
struct irqmp_regs : public crv_sequence_item {
    crv_variable<unsigned> level_reg { "level_reg" };    // Constraint for level
    crv_variable<unsigned> force_reg { "force_reg" };   // Constraint for force

    crv_constraint level_reg_cstr { "level_reg_cstr",
        level_reg() < (1 << 16),
        (level_reg() & 1) == 0
    };

    crv_constraint force_reg_cstr { "force_reg_cstr",
        (force_reg() & 0xFFFF0001) == 0
    };

    irqmp_regs(crv_object_name) {}  // In-place initialization
};
```
Modern C++11 API incl. Coverage

- Coverage layer
- Example (verification of a SystemC TLM interrupt controller)

```cpp
struct my_covergroup : public crv_covergroup {
    crv_variable<signed> lr { "lr" };
    crv_variable<signed> fr { "fr" };

    crv_coverpoint fwd_lvl_1 { "fwd_lvl_1" };
    crv_coverpoint fwd_lvl_0 { "fwd_lvl_0" };

    expression forced_lvl_1() { return make_expression(fr() & lr()); }
    expression forced_lvl_0() { return make_expression(fr() & ~lr()); }

    my_covergroup(crv_object_name) {
        for (int k = 1; k < 16; k++) {
            fwd_lvl_1.bins(((forced_lvl_1() >> k) & 1) == 1) && (forced_lvl_1() < (2 << k));
            fwd_lvl_0.bins(((forced_lvl_0() >> k) & 1) == 1) && (forced_lvl_0() < (2 << k)) && (forced_lvl_1() == 0);
        }
    }
};
```
Coverage-driven Generation

- Coverage sampling
- Results without CdG

Coverpoint fwd_lvl_1

<table>
<thead>
<tr>
<th>Hits</th>
<th>1000 stimuli</th>
<th>10000 stimuli</th>
<th>20000 stimuli</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>54</td>
<td>207</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>31</td>
<td>54</td>
</tr>
<tr>
<td>3</td>
<td>53</td>
<td>3</td>
<td>68</td>
</tr>
<tr>
<td>4</td>
<td>81</td>
<td>81</td>
<td>78</td>
</tr>
<tr>
<td>5</td>
<td>113</td>
<td>113</td>
<td>172</td>
</tr>
<tr>
<td>6</td>
<td>172</td>
<td>172</td>
<td>297</td>
</tr>
<tr>
<td>7</td>
<td>277</td>
<td>277</td>
<td>377</td>
</tr>
<tr>
<td>8</td>
<td>377</td>
<td>377</td>
<td>508</td>
</tr>
<tr>
<td>9</td>
<td>508</td>
<td>508</td>
<td>758</td>
</tr>
<tr>
<td>10</td>
<td>758</td>
<td>758</td>
<td>823</td>
</tr>
<tr>
<td>11</td>
<td>823</td>
<td>823</td>
<td>1223</td>
</tr>
<tr>
<td>12</td>
<td>1223</td>
<td>1223</td>
<td>1587</td>
</tr>
<tr>
<td>13</td>
<td>1587</td>
<td>1587</td>
<td>2201</td>
</tr>
<tr>
<td>14</td>
<td>2201</td>
<td>2201</td>
<td>2844</td>
</tr>
<tr>
<td>15</td>
<td>2844</td>
<td>2844</td>
<td>3443</td>
</tr>
</tbody>
</table>

Coverpoint fwd_lvl_0

<table>
<thead>
<tr>
<th>Hits</th>
<th>1000 stimuli</th>
<th>10000 stimuli</th>
<th>20000 stimuli</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>21</td>
<td>21</td>
<td>51</td>
</tr>
</tbody>
</table>

Forwarded Interrupt
Coverage-driven Generation

• Manual coverage closure
  – Add more (distribution) constraints?
  – Add more directed tests?
  – Both highly non-trivial

• “Push-button” when constraint and coverage are connected!
  – Convert coverage expressions to constraint expressions
  – In each generation pass
    • Pick one unsatisfied “coverage” constraint (i.e. uncovered bin), add to the constraint set, then apply constraint solving
    • If successful, marked the “coverage” constraint as satisfied, otherwise pick another unsatisfied and repeat
Wrap-up

• CRAVE
  – Open-source constrained random stimuli generator for SystemC/C++
  – Powerful constraint solving technologies
  – Frequently added new features, recently: modern C++11 API, coverage layer & coverage-driven generation

• What’s next?
  – Graph-based specification / Portable Stimulus
  – UVM-SystemC Integration
Acknowledgement

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project Effektiv under contract no. 01IS13022E and by the German Research Foundation (DFG) within the Reinhart Koselleck project DR 287/23-1.
Thanks for your time!
Questions?