Enriching UVM in SystemC with AMS extensions for randomization and functional coverage

Thilo Vörtler, Thomas Klotz, Karsten Einwich, Yao Li, Zhi Wang, Marie-Minerve Louërat, Jean-Paul Chaput, François Pêcheux, Ramy Iskander, Martin Barnasconi
Outline

• Motivation
• UVM for SystemC and SystemC-AMS
• Randomization
• Coverage
• Summary
Outline

• Motivation
  • UVM for SystemC and SystemC-AMS
  • Randomization
  • Coverage
• Summary
Motivation (1)

• UVM is currently being available for SystemC-AMS

• The UVM methodology relies heavily on the use of randomization of sequences and functional coverage collection

• Randomization and coverage APIs are not defined in UVM but inherited from System Verilog
Goal: Definition of randomization and coverage API for UVM in SystemC/AMS

*continuous distribution functions
Outline

• Motivation
• UVM for SystemC and SystemC-AMS
• Randomization
• Coverage
• Summary
UVM for SystemC-AMS

Requirements for randomization and coverage API:

• Current available SystemC libraries don’t offer a simple API that is familiar to UVM users
• Extensions for dealing with real values needed for AMS verification are needed

→ Proposal for an SCV extension (scvx) for further standardization within Accellera
UVM for SystemC/AMS

- Sequences that drive stimulus to the DUT are randomized.
- Coverage is used in monitors and scoreboards to check the progress of the verification process.
Outline

• Motivation
• UVM for SystemC and SystemC-AMS
• Randomization
• Coverage
• Summary
## Randomization (1)

Supported constructs (CRAVE 1.0 as backend)

<table>
<thead>
<tr>
<th>Functionality</th>
<th>SystemVerilog</th>
<th>UVM-SystemC (scvx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random variable declaration</td>
<td>rand T</td>
<td>scvx_rand&lt;T&gt;</td>
</tr>
<tr>
<td>Enable or disable random variable</td>
<td>rand_mode(...)</td>
<td>rand_mode(...)</td>
</tr>
<tr>
<td>Constraint block declaration</td>
<td>constraint</td>
<td>scvx_constraint</td>
</tr>
<tr>
<td>Enable or disable constraint</td>
<td>constraint_mode(...)</td>
<td>constraint_mode(...)</td>
</tr>
<tr>
<td>Randomization container object</td>
<td>-</td>
<td>scvx_rand_object</td>
</tr>
<tr>
<td>Randomize method</td>
<td>randomize()</td>
<td>randomize()</td>
</tr>
<tr>
<td>Randomize method with inline constraint</td>
<td>randomize() with ...</td>
<td>randomize_with(... )</td>
</tr>
</tbody>
</table>
Randomization (2)

```c++
class simplesum : public scvx::scvx_rand_object
{
public:
  scvx::scvx_rand< int > x, y, z;
  scvx::scvx_constraint c1, c2, c3;
  simplesum( scvx::scvx_name name )
    : x("x"), y("y"), z("z"),
      c1("c1"), c2("c2"), c3("c3")
  {
    c1( z() == x() + y() );
    c2( x() == 5 );
    c3( y() > 0 && y() < 10 );
  }
  void print_result() const
  {
    cout << name() << " : " << z << " == "
    << x << " + " << y << endl;
  }
}; // class simplesum

int sc_main(int, char*[
{
  simplesum s("simplesum");
  bool result = s.randomize();
  if (result) s.print_result();
  else cout << "No solution found." << endl;
  s.c2.constraint_mode( false );
  result = s.randomize_with( s.x() == 10 );
  if (result) s.print_result();
  else cout << "No solution found." << endl;
  s.y.rand_mode( false );
  result = s.randomize();
  if (result) s.print_result();
  else cout << "No solution found." << endl;
  return 0;
}
```
Randomization (3)

• Console output:

$ ./simplesum.exe
constraint c1 registered.
constraint c2 registered.
constraint c3 registered.
simplesum: 13 == 5 + 8
constraint c2 disabled.
in-line constraint ci_0 applied (disabled after use)
simplesum: 13 == 10 + 3
random variable 'y' made inactive (value remains 3).
simplesum: 33556493 == 33556490 + 3
Randomization (4)

- Analog extensions (C++11 based solver)

<table>
<thead>
<tr>
<th>Distribution function</th>
<th>UVM-SystemC (scvx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal distribution</td>
<td>scvx_normal_distribution</td>
</tr>
<tr>
<td>Uniform distribution</td>
<td>scvx_uniform_real_distribution</td>
</tr>
<tr>
<td>Bernoulli distribution</td>
<td>scvx_bernoulli_distribution</td>
</tr>
<tr>
<td>Piece-wise linear probability distribution function</td>
<td>scvx_piecewise_linear_probability_distribution</td>
</tr>
<tr>
<td>Discretized probability distribution function</td>
<td>scvx_discrete_probability_distribution</td>
</tr>
</tbody>
</table>

```cpp
scvx::scvx_rand<real> a;
a.set_distribution( dist_type(dist_params) );
```
Randomization (5)

Uniform distribution example using `scvx_uniform_real_distribution`
Outline

• Motivation
• UVM for SystemC and SystemC-AMS
• Randomization
• Coverage
• Summary
Coverage (1)

- Supported features

<table>
<thead>
<tr>
<th>Functionality</th>
<th>SystemVerilog</th>
<th>UVM-SystemC (scvx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage model</td>
<td>covergroup</td>
<td>scvx_covergroup</td>
</tr>
<tr>
<td>Coverage points</td>
<td>coverpoint</td>
<td>scvx_coverpoint</td>
</tr>
<tr>
<td>Coverage state bins</td>
<td>bins</td>
<td>bins()</td>
</tr>
<tr>
<td>Illegal bins</td>
<td>illegal_bins</td>
<td>illegal_bins()</td>
</tr>
<tr>
<td>Ignore bins</td>
<td>ignore_bins</td>
<td>ignore_bins()</td>
</tr>
<tr>
<td>Coverage options</td>
<td>option</td>
<td>option</td>
</tr>
</tbody>
</table>
class cg: public scvx::scvx_covergroup
{
    public:
    scvx::scvx_coverpoint cp_m;
    scvx::scvx_coverpoint cp_n;
    cg( scvx::scvx_name name, int& m, int& n )
        : cp_m( "cp_m", m ),
        cp_n( "cp_n", n )
    {
        option.auto_bin_max = 16;
        cp_m.bins("bin_a") =
            scvx::list_of(4, 0, 1, 2, 3);
        cp_m.bins("bin_b", scvx::SINGLE_BIN) =
            scvx::list_of(4, 4, 5, 6, 7);
        cp_m.ignore_bins("ignore_bins") = 6;
        cp_n.ignore_bins("ignore_bins") = 13;
    }
};

int sc_main(int, char*[])
{
    int m; // variable to be covered
    int n; // variable to be covered
    int stimuli_m[] =
        { 3, 5, 6, 5, 3, 6, 5, 5, 3, 3 };
    int stimuli_n[] =
        { 13, 1, 6, 3, 16, 12, 8, 3, 13, 3 };  
    cg cg_inst("cg_inst", m, n);
    for ( int i = 0; i < 10; i++ )
    {
        m = stimuli_m[i];
        n = stimuli_n[i];
        cg_inst.sample();
    }
    cg_inst.report();
    return 0;
}
Coverage (3)

$ ./test.exe

Covergroup: cg_inst

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>Expected</th>
<th>Covered</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>cp_m</td>
<td>7</td>
<td>2</td>
<td>28.57</td>
</tr>
<tr>
<td>cp_n</td>
<td>15</td>
<td>5</td>
<td>33.33</td>
</tr>
</tbody>
</table>

TOTAL: 22 7 31.82

coverpoint: cp_m

<table>
<thead>
<tr>
<th>Name</th>
<th>Percent</th>
<th>Hitrate</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin_a[0]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bin_a[1]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bin_a[2]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bin_a[3]</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>bin_b[4]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bin_b[5]</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>bin_b[7]</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Coverage statistics are printed at the end of the simulation.
- When covering real values ranges are used to identify bins.
- Support of Unified Coverage Interoperability Standard (UCIS) under discussion.
Outline

• Motivation
• UVM for SystemC and SystemC-AMS
• Randomization
• Coverage
• Summary
Summary

• Currently UVM for SystemC is discussed as being a standard in SystemC Verification Working Group, however the standard doesn’t describe an coverage and randomization API

• Randomization and coverage API with extensions to real values described. However, constrained randomization for real values is hard!

• API will be submitted to Accellera for standardization
Acknowledgements

The development of the UVM-SystemC methodology and library has been supported by the European Commission as part of the Seventh Framework Programme (FP7) for Research and Technological Development in the project 'VERIFICATION FOR HETEROGENEOUS RELIABLE DESIGN AND INTEGRATION' (VERDI). The research leading to these results has received funding from the European Commission under grand agreement No 287562.
Questions