UVM-SystemC
Standardization Status
and Latest Developments

Trevor Wieman, SystemC CCI WG Chair

Slides by Michael Meredith, Cadence Design Systems
Outline

- Why UVM-SystemC?
- UVM layered architecture
- UVM-SystemC LRM + proof-of-concept library
- Status Accellera standardization
- Next steps
- You can help!
Why UVM-SystemC?

• Current ESL focus is on system design not on system verification – step-up is required!
• Testbenches and tests for system design are often ad-hoc, unstructured and incomplete
• No reuse of system verification components and environment – missing link to RTL world
• UVM-SystemVerilog lacks native C/C++ integration to support SW-driven verification use cases
• Reuse testbenches and tests in HW prototyping, equipment and Hardware-in-the-loop (HiL) setup
• The top-level contains the test(s), the DUT and interfaces
• DUT interfaces are stored in a configuration database, used by the UVCs to connect to the DUT
• The testbench contains the UVCs, register model, scoreboard and sequencer to execute the stimuli and check the results
• (Almost) everything is configurable
Standardization Status

• Draft 1.0 standard released early 2016
  – Language Reference Manual as public review
  – Alpha release of Proof-of-concept implementation

• Current standardization focus
  – Register Abstraction Layer
  – Constrained Randomization (using CRAVE)
  – Reporting infrastructure
  – Test and regression environment
# Register Abstraction Layer

<table>
<thead>
<tr>
<th>Register Abstraction Layer</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register model containing registers, fields, blocks, etc.</td>
<td>testing</td>
</tr>
<tr>
<td>Register callbacks</td>
<td>testing</td>
</tr>
<tr>
<td>Register adapter, register sequences and transaction items</td>
<td>testing</td>
</tr>
<tr>
<td>Register frontdoor access</td>
<td>testing</td>
</tr>
<tr>
<td>Build-in register test sequencers</td>
<td>development</td>
</tr>
<tr>
<td>Register backdoor access (hdl_path)</td>
<td>testing</td>
</tr>
<tr>
<td>Memory and memory allocation manager</td>
<td>development</td>
</tr>
<tr>
<td>Virtual registers and fields</td>
<td>development</td>
</tr>
<tr>
<td>Randomization of register content</td>
<td>development</td>
</tr>
</tbody>
</table>
class reg_a : public uvm_reg {

public:
    uvm_reg_field* F1;
    uvm_reg_field* F2;

UVM_OBJECT_UTILS(reg_a);

reg_a( uvm_object_name name = "a" ) : uvm_reg(name, 32, UVM_NO_COVERAGE) {}} 

virtual void build() 
{
    F1 = uvm_reg_field::type_id::create("F1");
    F1->configure(this, 8, 0, "RW", false, 0x0, true, false, true);
    F2 = uvm_reg_field::type_id::create("F2");
    F2->configure(this, 8, 16, "RO", false, 0x0, true, false, true);
}
Constrained Randomization

• UVM-SystemC will introduce constrained randomization by using a library called CRAVE:

  • Constrained Random Verification Environment
    – Powerful & extensible constrained random stimuli generator
    – Leverage latest constraint solving technologies
    – C++11 syntax for constraint definition

• More information
  – www.systemc-verification.org/crave
Randomization Example

class ubus_transfer : public uvm_randomized_sequence_item {
public:
  crv_variable<ubus_rw_enum> read_write;
  crv_variable<sc_lv<16> > addr;
  crv_variable<unsigned> size;
  crv_vector<sc_lv<8> > data;
  crv_vector<sc_lv<4> > wait_state;
...

crv_constraint c_read_write { inside( read_write(),
    std::set<ubus_rw_enum> { 
    ubus_rw_enum::READ, ubus_rw_enum::WRITE }));

crv_constraint c_size { inside( size(),
    std::set<int> { 1, 2, 4, 8 } )};

crv_constraint c_data_wait_size {
  data().size() == size(),
  wait_state().size() == size() };
...

CRAVE variables and vectors
Constraint definitions

Work-in-progress: API might change!

© Accellera Systems Initiative
Reporting Infrastructure

• Reporting infrastructure upgraded to UVM 1.2 capabilities
  – Reporting mechanism is now object oriented using the new UVM report message object
  – All std::cout have been replaced by UVM_INFO messages, enabling visibility (verbosity) control
• For the user, no big difference in usage of the API
• Note: UVM report message object extension macros (e.g. UVM_MESSAGE_ADD_INT) not yet available
Test and Regression Environment

• Simple examples available in PoC to demonstrate basic functionality (component instantiation, factory, configuration, reporting, phasing, sequences, etc.)
• Unit tests developed as part of a regression suite – closely following the UVM-SystemVerilog tests
• More integrated examples under development (e.g. ubus and codec example)
• Adding new unit tests for register model testing
Next Steps

• Update LRM
  – Register abstraction classes
  – Reporting classes

• Constrained randomization API and functional coverage

• Introduction of UVM-SystemC in Accellera Multi-language verification standardization

• Align configuration concept with SystemC CCIWG
You Can Help!

• Lots of things to do to grow UVM-SystemC
• How you can contribute
  – Join Accellera and participate in the SystemC Verification Working Group
  – Development of unit tests, examples and applications
  – Contribute to proof-of-concept development: robustness, quality, maturity
Thank You!