Accellera Systems Initiative
SystemC Standards Update

Bishnupriya Bhattacharya
DVCon India, September 10, 2015
Presentation Overview

• Accellera Overview
  – Membership list
  – How to join a WG
  – Global SystemC events

• Number of IEEE-1666 standard downloads

• Accellera SystemC Working Group updates
  – Language & Transaction-Level Modeling
  – Configuration, Control & Inspection
  – Synthesis
  – Analog/Mixed-Signal
  – Verification

• Proposed Working Group information
  – Transaction Level Protocols
All Members Can Join SystemC WGs!

<table>
<thead>
<tr>
<th>Corporate Members</th>
<th>Associate Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD</td>
<td>AGNISYS</td>
</tr>
<tr>
<td>ARM</td>
<td>ALDEC</td>
</tr>
<tr>
<td>cadence®</td>
<td>ALTERA</td>
</tr>
<tr>
<td>ERISSON</td>
<td>ANALOG DEVICES</td>
</tr>
<tr>
<td>freescale®</td>
<td>SPYGLASS</td>
</tr>
<tr>
<td>intel®</td>
<td>berkeley design automation, Inc.</td>
</tr>
<tr>
<td>Mentor Graphics®</td>
<td>BOEING</td>
</tr>
<tr>
<td>NXP</td>
<td>BROADCOM</td>
</tr>
<tr>
<td>QUALCOMM</td>
<td>dialog</td>
</tr>
<tr>
<td>SONICS</td>
<td>DOULOS</td>
</tr>
<tr>
<td>STI</td>
<td>Fraunhofer IIS</td>
</tr>
<tr>
<td>Texas Instruments</td>
<td>IBM</td>
</tr>
<tr>
<td>life.augmented</td>
<td>Imagination</td>
</tr>
<tr>
<td>SYNOPSYS</td>
<td>magillem</td>
</tr>
<tr>
<td></td>
<td>NEC</td>
</tr>
<tr>
<td></td>
<td>NVIDIA</td>
</tr>
<tr>
<td></td>
<td>OFFIS</td>
</tr>
<tr>
<td></td>
<td>PARADIGM WORKS</td>
</tr>
<tr>
<td></td>
<td>Semifore</td>
</tr>
<tr>
<td></td>
<td>UPMC</td>
</tr>
<tr>
<td></td>
<td>XILINX</td>
</tr>
</tbody>
</table>
Join A Working Group And Contribute!

SystemC Synthesis Working Group (SWG)

Charter

This group is responsible for the definition of a synthesizable subset of SystemC.

Chair: Andres Takach, Mentor Graphics
Vice-Chair: Michael Meredith, Forte Design Systems

Background

In August 2009, this group released the Synthesis Subset Draft 1.3 standard synthesis subset draft for public review. The draft features several technical updates. Supported language constructs are now established, and a chapter on processes, clocks, and resets has been added. The draft also includes a discussion on abstraction levels that puts the concepts of the synthesizable subset in the context of the abstraction levels defined for TLM.

Public review of the draft is now closed. The draft is available for download here.

Join this Working Group

If you are an employee of a member company and would like to join this working group (requires login) and click Join Group. WG participation requires right of entry by the group chair.
SystemC Community

- Online at http://accellera.org/community/systemc
- Community forums, upload area for contributions, SystemC news
Global SystemC Presence 2014+

- DVCon USA  March in Silicon Valley
- DAC  June in San Francisco
- **DVCon India**  September in Bangalore
- **DVCon Europe**  October in Munich
- SystemC Japan  June 19, 2015
- Accellera Day Taiwan  1st half of 2015
IEEE 1666 SystemC Downloads


Sponsored by the
Design Automation Standards Committee


© Accellera Systems Initiative
Slide 7
March 2, 2015
SystemC Overview

SystemC Core Language
- Structural Elements
  - Modules
  - Ports
  - Exports
  - Interfaces
  - Channels
- Predefined Channels
  - Signal, clock, FIFO, mutex, semaphore
- Utilities
  - Report handling, tracing
- Event-driven Simulation
  - Events, processes
- Data Types
  - 4-valued logic type
  - 4-valued logic vectors
  - Bit vectors
  - Finite-precision integers
  - Limited-precision integers
  - Fixed-point types

Programming Language C++
ISO/IEC Std. 14882-2003

© Accellera Systems Initiative  Slide 8  March 2, 2015
SystemC Language & TLM WG

- **Charter:** Responsible for the definition and development of the SystemC core language, the foundation on which all other SystemC libraries and functionality are built.

- **Current status**
  - Maintenance release version 2.3.1 of the proof-of-concept simulator in April 2014 (bug fixes, experimental features)

- **Plans for 2014/2015**
  - Discuss new concepts affecting simulation performance
  - Collect, address, refine proposals and errata towards IEEE 1666-201x
SystemC 2.3.1 Maintenance Release

• Release of 2.3.1 in April 2014
  – Bug fixes for known issues wrt. IEEE 1666-2011
  – Some feature additions beyond IEEE 1666-2011 (may require explicit configuration during library build)
  – Code cleanups, deprecation of non-standard
Roadmap for IEEE 1666-201x

• Next IEEE 1666 update later this decade
  – Several errata and proposals already addressed in 2.3.1
  – Formal standardization will be moved to IEEE when sufficient input is available

• LWG/TLMWG are currently collecting proposals
  – Report your favorite missing feature/extension/annoyance
  – Non-Accellera members can use the community forums

• Parallelization of SystemC could be significant driver
  – More contributors needed!
SystemC Synthesis WG

• Charter: To define the SystemC synthesis subset to allow synthesis of digital hardware from high-level specifications.

• Current status
  – Releasing draft standard for 3-month public review
  – www.accellera.org/apps/org/workgroup/swg

• Plans for 2015
  – Process feedback from review in Q2 2015
  – Release standard in Q3 2015
  – Start work on new topics for the second version of the standard
Configuration, Control & Inspection WG

WG is defining these

Goal: Standardizing interfaces between models and tools
SystemC Analog/Mixed-Signal WG

• **Charter:** The SystemC AMS Working Group is responsible for the standardization of the SystemC AMS extensions, defining and developing the language, methodology and class libraries for analog, mixed-signal and RF modeling in SystemC

• **Current status**
  – Released the SystemC AMS 2.0 standard in March 2013

• **Plans 2014/2015**
  – Publish User’s Guide update based on SystemC AMS 2.0
  – IEEE P1666.1 SystemC AMS Working Group started – Accellera contributed SystemC AMS standard to IEEE-SA
SystemC Verification WG

- **Charter:** The Verification Working Group (VWG) is responsible for defining verification extensions to the SystemC language standard, and to enrich the SystemC reference implementation by offering an add-on libraries (SystemC Verification (SCV) library, etc.) to ease the deployment of a verification methodology based on SystemC.

- **Current Status**
  - Released version 2.0 of SystemC Verification library (SCV) in April 2014

- **Plans for 2014/2015**
  - Integrate the UVM verification methodology in SystemC
  - Standardization of coverage APIs (coverage groups, bins, etc.)
  - Further explorations of needs regarding SystemC/TLM
UVM SystemC

• New standard under discussion in VWG
• Materializes the UVM methodology natively in SystemC
• Open source proof-of-concept implementation and LRM have been donated to Accellera
• Language Reference Manual under review/discussion right now
  – Please join us if you are interested!
Transaction Level Protocols PWG

- Proposed Charter
  Provide a blueprint for the creation of SystemC Transaction Level Protocols, provide interoperable interfaces, develop best practices for SystemC interfaces and provide a process to ensure the timely delivery of those interfaces.

- Proposed Scope
  - The working group will be narrowly focused on interfaces implemented in SystemC
  - We will not cover the existing TLM-2.0 ‘generic bus protocol’
  - The scope will be to define best practices and provide a procedure to ensure that interfaces are written to that standard
Transaction Level Protocols PWG – cont’d

• Proposed Goals
  – Establish a well-known location for SystemC Transaction Level Protocol (TLP) interfaces
  – Define and ensure TLP blueprint best practices as new TLP interfaces are standardized
  – Eliminate duplication of development effort and the necessity for adapters between divergent implementations of protocols
  – Establish timely process to incorporate contributions and feedback from wider open source community
  – Have the TLP blueprint used as the standard for interfaces developed outside the Accellera TLP working group (proprietary interfaces)

More info: http://www.accellera.org/activities/proposed_working_groups
Advancing Standards Together

• Share your experiences
  – Visit www.accellera.org and register to post on community forums at forums.accellera.org

• Show your support
  – Record your adoption of standards

• Become an Accellera member
  – Join working groups
Introduction to the Universal Verification Methodology in SystemC

Andy Goodrich
Martin Barnasconi
DVCon March 2, 2015
Outline

• Introduction and motivation
  • UVM ... what is it?
  • Why UVM in SystemC/C++?
• UVM-SystemC development history
• UVM-SystemC principles
• Contribution to Accellera
• UVM-SystemC library overview
  -- Features and examples
• Review and release plan
• Summary and outlook
UVM ... what is it?

• Universal Verification Methodology facilitates the creation of modular, scalable, configurable and reusable test benches
  – Based on verification components with standardized interfaces

• Class library which provides a set of built-in features dedicated to simulation-based verification
  – Utilities for execution (phasing), component overriding (factory), configuration, comparing, scoreboard, reporting, etc.

• Environment supporting migration from directed testing towards Coverage Driven Verification (CDV)
  – Introducing automated stimulus generation, independent result checking and coverage collection
Why UVM in SystemC/C++

- No structured nor unified verification methodology available for ESL design
- UVM in SystemVerilog primarily targeting block/IP level (RTL) verification, not system-level
- Porting UVM to SystemC/C++ enables
  - Creation of more advanced system-level test benches
  - Software-driven (C/C++ based) verification
  - Reuse of verification components between system-level and block-level verification
- Target to make UVM truly universal, and not tied to a particular language

*UVM-SystemC = UVM implemented in SystemC/C++
UVM-SystemC development history

• UVM-SystemC development started in the European Framework 7 project ‘VERDI’ (Verification for Heterogeneous Reliable Design and Integration)
  – Main objective: Development of a unified system-level verification methodology for heterogeneous systems
  – Partners in the VERDI consortium: Fraunhofer, NXP, Infineon, Continental, Magillem, UPMC

• Special thanks to the members of the VERDI consortium to grant the contribution of the LRM and proof-of-concept implementation to Accellera

• More information: http://verdi-fp7.eu/
UVM-SystemC principles

- UVM Standard compliant (currently UVM 1.1)
  - Identical class definitions and method signatures (where feasible)
  - Some minor changes due to clash with C++ reserved keywords
- Comply with SystemC standard w.r.t execution semantics and TLM communication mechanism
- Library based on ISO/IEC 14882:2003 (C++03)
  - Compatible with EDA vendor solutions and flows
  - Limited use of add-on libraries; keep dependencies low
- Maintain a well documented API
  - Defined in a UVM-SystemC Language Reference Manual (LRM)
Contribution to Accellera

• Objective: Seek further industry support and standardization of UVM in SystemC/C++
• UVM-SystemC contribution to Accellera Verification WG
  – UVM-SystemC Language Reference Manual (LRM)
  – UVM-SystemC Proof-of-Concept implementation
  – Both LRM and PoC are released under Apache 2.0 license
• Next steps: Align with SCV and Multi-Language requirements
Industry Interest

- The following companies showed (serious) interest in UVM-SystemC (and some are already using it...)

- AMD
- AGNISYS
- BOSCH
- BROADCOM
- cadence
- Fraunhofer
- freescale
- Intel
- magilem
- NXP
- ST
- UPMC
- AMIQ Consulting
- Circuit Sutra Technologies
- Continental
- Infineon

© Accellera Systems Initiative  Slide 28  March 2, 2015
# UVM-SystemC overview

<table>
<thead>
<tr>
<th>UVM-SystemC functionality</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test bench creation with component classes: agent, sequencer,</td>
<td>✓</td>
</tr>
<tr>
<td>driver, monitor, scoreboard, etc.</td>
<td></td>
</tr>
<tr>
<td>Test creation with test, (virtual) sequences, etc.</td>
<td>✓</td>
</tr>
<tr>
<td>Configuration and factory mechanism</td>
<td>✓</td>
</tr>
<tr>
<td>Phasing and objections</td>
<td>✓</td>
</tr>
<tr>
<td>Policies to print, compare, pack, unpack, etc.</td>
<td>✓</td>
</tr>
<tr>
<td>Messaging and reporting</td>
<td>✓</td>
</tr>
<tr>
<td>Register abstraction layer and callbacks</td>
<td>development</td>
</tr>
<tr>
<td>Coverage groups</td>
<td>development</td>
</tr>
<tr>
<td>Constrained randomization</td>
<td>SCV or CRAVE</td>
</tr>
</tbody>
</table>
UVM layered architecture

Test cases

Spec

Verification environment (test bench)

Sequences

Verification component

Sequencer

Functional coverage

Scoreboard

Driver

Device under test

Monitor

Command

Scenario

Functional

Signal

Test

Device under test

© Accellera Systems Initiative

March 2, 2015
UVM agent

- Component responsible for driving and monitoring the DUT
  - Typically contains three components: Sequencer, Driver and Monitor

- Can contain analysis functionality for basic coverage and checking

- Possible configurations
  - Active agent: sequencer and driver are enabled
  - Passive agent: only monitors signals (sequencer and driver are disabled)

- C++ base class: \texttt{uvm_agent}
class vip_agent : public uvm_agent
{
    public:
        vip_sequencer<vip_trans>* sequencer;
        vip_driver<vip_trans>* driver;
        vip_monitor* monitor;

    UVM_COMPONENT_UTILS(vip_agent)

    vip_agent( uvm_name name )
        : uvm_agent( name ), sequencer(0), driver(0), monitor(0) {}

    virtual void build_phase( uvm_phase& phase )
    {
        uvm_agent::build_phase(phase);

        if ( get_is_active() == UVM_ACTIVE )
        {
            sequencer = vip_sequencer<vip_trans>::type_id::create("sequencer", this);
            assert(sequencer);
            driver = vip_driver<vip_trans>::type_id::create("driver", this);
            assert(driver);
        }

        monitor = vip_monitor::type_id::create("monitor", this);
        assert(monitor);
    }

    UVM-SystemC agent (1)

    Dedicated base class to distinguish agents from other component types

    Registers the object in the factory

    Children are instantiated in the build phase

    Essential call to base class to access properties of the agent

    Call to the factory which creates and instantiates this component dynamically

    NOTE: UVM-SystemC API under review – subject to change
virtual void connect_phase( uvm_phase& phase )
{
    if ( get_is_active() == UVM_ACTIVE )
    {
        // connect sequencer to driver
        driver->seq_item_port.connect(sequencer->seq_item_export);
    }
}

NOTE: UVM-SystemC API under review – subject to change

• UVM introduces the member function connect for the binding, but the SystemC member function bind or operator() can be used instead
A UVM verification component (UVC) is an environment which consists of one or more cooperating agents.

- UVCs or agents may set or get configuration parameters.
- An independent test sequence is processed by the driver via a sequencer.
- Each verification component is connected to the DUT using a dedicated interface.
- C++ base class: `uvm_env`
In this example, the UVM verification component (UVC) contains only one agent. In practice, more agents are likely to be instantiated.
UVM sequences

• Sequences are part of the test scenario and define streams of transactions
• The properties (or attributes) of a transaction are captured in a sequence item
• Sequences are not part of the test bench hierarchy, but are mapped onto one or more sequencers
• Sequences can be layered, hierarchical or virtual, and may contain multiple sequences or sequence items
• Sequences and transactions can be configured via the factory
• C++ base classes: uvm_sequence and uvm_sequence_item
UVM-SystemC sequence item

```cpp
class vip_trans : public uvm_sequence_item
{
  public:
    int addr;
    int data;
    bus_op_t op;

  UVM_OBJECT_UTILS(vip_trans);

  vip_trans( const std::string& name = " vip_trans" )
    : addr(0x0), data(0x0), op(BUS_READ) {}

  virtual void do_print( uvm_printer& printer ) const { ... }
  virtual void do_pack( uvm_packer& packer ) const { ... }
  virtual void do_unpack( uvm_packer& packer ) { ... }
  virtual void do_copy( const uvm_object& rhs ) { ... }
  virtual bool do_compare( const uvm_object& rhs ) const { ... }
};
```

Transaction defined as sequence item

User-defined data items (randomization can be done using SCV or CRAVE)

A sequence item should implement all elementary member functions to print, pack, unpack, copy and compare the data items (there are no field macros in UVM-SystemC)

NOTE: UVM-SystemC API under review – subject to change
template <typename REQ = uvm_sequence_item, typename RSP = REQ>
class sequence : public uvm_sequence<REQ,RSP>
{
public:
    sequence( const std::string& name ) : uvm_sequence<REQ,RSP>( name ) {}

    UVM_OBJECTPARAM_UTILS(sequence<REQ,RSP>);

    virtual void pre_body() {
        if ( starting_phase != NULL )
            starting_phase->raise_objection(this);
    }

    virtual void body() {
        REQ* req;
        RSP* rsp;
        ...
        start_item(req);
        // req->randomize();
        finish_item(req);
        get_response(rsp);
    }

    virtual void post_body() {
        if ( starting_phase != NULL )
            starting_phase->drop_objection(this);
    }
};

A sequence contains a request and (optional) response, both defined as sequence item

Randomization should use SCV or CRAVE library

Optional: get response

Factory registration supports template classes

Raise objection if there is no parent sequence

NOTE: UVM-SystemC API under review – subject to change
A test bench is the environment which instantiates and configures the UVCs, scoreboard, and (optional) virtual sequencer.

The test bench connects:
- Agent sequencer(s) in each UVC with the virtual sequencer (if defined)
- Monitor analysis port(s) in each UVC with the scoreboard subscriber(s)
- Note: The driver and monitor in each agent connect to the DUT using the interface stored in the configuration database

C++ base class: `uvm_env`
class testbench : public uvm_env
{
    public:
        vip_uvc*        uvc1;
        vip_uvc*        uvc2;
        virt_sequencer* virtual_sequencer;
        scoreboard*     scoreboard1;

    UVM_COMPONENT_UTILS(testbench);

    testbench( uvm_name name )
    : uvm_env( name ), uvc1(0), uvc2(0),
        virtual_sequencer(0), scoreboard1(0) {}

    virtual void build_phase( uvm_phase& phase )
    {
        uvm_env::build_phase(phase);

        uvc1 = vip_uvc::type_id::create("uvc1", this);
        assert(uvc1);
        uvc2 = vip_uvc::type_id::create("uvc2", this);
        assert(uvc2);

        uvm_config_db<int>::set(this, "uvc1.*", "is_active", UVM_ACTIVE);
        uvm_config_db<int>::set(this, "uvc2.*", "is_active", UVM_PASSIVE);
        ...

    }
}
... virtual_sequencer = virt_sequencer::type_id::create(  "virtual_sequencer", this);
assert(virtual_sequencer);

scoreboard1 = 
    scoreboard::type_id::create("scoreboard1", this);
assert(scoreboard1);
}

virtual void connect_phase( uvm_phase& phase )
{
    virtual_sequencer->vip_seqr = uvc1->agent->sequencer;

    uvc1->agent->monitor->item_collected_port.connect(  
        scoreboard1->xmt_listener_imp);

    uvc2->agent->monitor->item_collected_port.connect(  
        scoreboard1->rcv_listener_imp);
}

};
UVM test

• Each UVM test is defined as a dedicated C++ test class, which instantiates the test bench and defines the test sequence(s)
• Reuse of tests and topologies is possible by deriving tests from a test base class
• The UVM configuration and factory concept can be used to configure or override UVM components, sequences or sequence items
• C++ base class: `uvm_test`
class test : public uvm_test
{
    public:
    testbench* tb;
    bool test_pass;

test( uvm_name name ) : uvm_test( name ),
        tb(0), test_pass(true) {}

UVM_COMPONENT_UTILS(test);

virtual void build_phase( uvm_phase& phase )
{
    uvm_test::build_phase(phase);
    tb = testbench::type_id::create("tb", this);
    assert(tb);

    uvm_config_db<uvm_object_wrapper*>::set( this,
        tb.uvc1.agent.sequencer.run_phase", "default_sequence",
        vip_sequence<vip_trans>::type_id::get());

    set_type_override_by_type( vip_driver<vip_trans>::get_type(),
                                new_driver<vip_trans>::get_type() );

...

NOTE: UVM-SystemC API under review – subject to change
UVM-SystemC test (2)

```c++
virtual void run_phase( uvm_phase& phase )
{
    UVM_INFO( get_name(),
        "** UVM TEST STARTED **", UVM_NONE );
}

virtual void extract_phase( uvm_phase& phase )
{
    if ( tb->scoreboard1.error )
        test_pass = false;
}

virtual void report_phase( uvm_phase& phase )
{
    if ( test_pass )
        UVM_INFO( get_name(), "** UVM TEST PASSED **", UVM_NONE );
    else
        UVM_ERROR( get_name(), "** UVM TEST FAILED **" );
}
```

NOTE: UVM-SystemC API under review – subject to change
The main program (top-level)

- The top-level (e.g. `sc_main`) contains the test(s) and the DUT
- The interface to which the DUT is connected is stored in the configuration database, so it can be used by the UVCs to connect to the DUT
- The test to be executed is either defined by the test class instantiation or by the argument of the member function `run_test`
int sc_main(int, char* [])
{
    dut* my_dut = new dut("my_dut");
    vip_if* vif_uvc1 = new vip_if;
    vip_if* vif_uvc2 = new vip_if;
    uvm_config_db<vip_if*>::set(0, ".*uvc1.*", "vif", vif_uvc1);
    uvm_config_db<vip_if*>::set(0, ".*uvc2.*", "vif", vif_uvc2);
    my_dut->in(vif_uvc1->sig_a);
    my_dut->out(vif_uvc2->sig_a);
    run_test("test");
    return 0;
}

NOTE: UVM-SystemC API under review – subject to change
UVM-SystemC review & release plan

  - Improvements and enhancements being incorporated in LRM and PoC
  - Code review and development will start in parallel to LRM review

- Release schedule is under discussion in VWG
  - Public draft of UVM-SystemC LRM and proof-of-concept implementation release planned for later this year

- Please join the SystemC VWG if you are interested to contribute to the review and code development!
Summary and outlook

• Universal Verification Methodology in SystemC/C++ contributed to Accellera
  – Compliant with UVM standard (currently UVM 1.1)
  – Both LRM and PoC contribution under Apache 2.0 license
  – LRM and PoC release targeted later this year, timing under discussion in SystemC VWG

• Ongoing developments and discussions
  – Make UVM-SystemC compatible with UVM 1.2 standard
  – Extend UVM-SystemC with constrained randomization and functional coverage features
  – Application of UVM-SystemC as part of a multi-language verification framework