Experiences of implementing UVM in System C based verification
TVS - Global Leaders in Test and Verification

Continuous geographical expansion...

Number of Employees by quarter

<table>
<thead>
<tr>
<th>Calendar Year</th>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q3-13</td>
<td>50</td>
</tr>
<tr>
<td>Q4-13</td>
<td>100</td>
</tr>
<tr>
<td>Q1-14</td>
<td>150</td>
</tr>
<tr>
<td>Q2-14</td>
<td>200</td>
</tr>
<tr>
<td>Q3-14</td>
<td>150</td>
</tr>
<tr>
<td>Q4-14</td>
<td>200</td>
</tr>
<tr>
<td>Q1-15</td>
<td>150</td>
</tr>
<tr>
<td>Q2-15</td>
<td>200</td>
</tr>
<tr>
<td>Q3-15</td>
<td>150</td>
</tr>
<tr>
<td>Q4-15</td>
<td>200</td>
</tr>
<tr>
<td>Q1-16 (Est.)</td>
<td>150</td>
</tr>
<tr>
<td>Q2-16 (Est.)</td>
<td>200</td>
</tr>
<tr>
<td>Q3-16 (Est.)</td>
<td>150</td>
</tr>
<tr>
<td>Q4-16 (Est.)</td>
<td>200</td>
</tr>
</tbody>
</table>

USA - 2014
UK - 2008
Germany - 2011
France - 2012
India - 2011
China
South Korea
Singapore - 2014
Index

✓ Background and challenges
✓ The TVS approach
✓ Advantages
✓ Case study: Client
✓ Planned updates & Availability
Background

- TVS has developed a SystemC testbench that is UVM compliant with a TLM 2.0 interface

- Primary objectives
  - **Reusability** - across projects using SystemC designs. It is also possible to reuse this on other HDL’s.
  - **Advanced verification support** – ability to deploy advanced verification techniques
  - **IP quality** – demonstrable improved IP quality via metrics and bug discovery rates
Challenges in Development

- SystemC did not have any inbuilt class libraries equivalent to the UVM class library when the development was started.
- SystemC inbuilt constraint based random verification features were very limited and not robust.
- Replication of Coverage driven verification features supported by UVM is a challenge in SystemC.
- Due to budget constraints, proprietary tools from EDA vendors could not be used initially.
The TVS Development Approach

- Developing the C++ equivalent of UVM – TVS C++ Class library (TVM)
- Enabling constrained random verification through CRAVE
- Achieving Coverage through TVS Functional Coverage Library & asureSIGN
TVM Library Structure
TVS implemented a library of base methodology classes and functions in C++ enabling easy conversion of test bench and tests from TVM-UVM or UVM->TVM

The TVM library mimicked UVM phases such as compile, build, run, check etc

The library includes base classes for Constraint driven randomization using CRAVE

Functional coverage is enabled through a TVM Functional Coverage Library

Code Coverage is enabled through gcov & lcov tools
Sample Implementation

Test case

DUT

Agent 1
- Driver
- Sequencer
- Monitor
  - Func Coverage

Agent 2
- Driver
- Sequencer
- Monitor
  - Func Coverage

Checker
- Scoreboard

Virtual Sequence
- Seq1
- Seq2
Enabling Constrained Random Verification

- Enabled through use of external randomization library called CRAVE
- The CRAVE library allows the users to set constraints in a manner similar to UVM & System Verilog
- CRAVE allows inline constraints, which can be formulated and changed incrementally at run-time

CRAVE is freely available under the MIT license
CRAVE is available via public repositories at GitHub.
Achieving Coverage

- TVS developed a Functional Coverage Library (FCL) to enable coverage driven verification
- Coverage is achieved through TVM functional coverage library and asureSIGN*
- Can be used either with TVM or individually on C++ based environments
- Supports report generation
- Support for different coverage types

*asureSign is TVS’ in-house requirements driven management and verification tool
For more information, please visit [http://www.testandverification.com/solutions/requirements/](http://www.testandverification.com/solutions/requirements/)
Advantages of TVS approach

- TVM supports many UVM features
  - Constructs for agents, monitors, drivers, sequencers and scoreboard
  - Factory registration and overriding
  - Randomization control using constraints
  - Ability to raise and drop objections
  - config_db mechanism
  - Dynamic casting using DCAST similar to System Verilog's $cast
  - Support for time out mechanism
  - Reporting severity similar to UVM
  - Uses TLM ports (interface method class)
Advantages of TVS approach

- TVM supports System Verilog features
  - Verbosity control of debug messages
  - Fine process control (like fork-join, tracking process using process handle) through System-C.
  - Interface class
## UVM Features implemented in TVM

<table>
<thead>
<tr>
<th>Item details</th>
<th>TVM + FCL</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test bench creation with component classes: agent, sequencer, driver, monitor, scoreboard, etc.</td>
<td>Yes</td>
<td>Virtual sequences with p_sequencer present in TVM but need to make them more UVM compliant</td>
</tr>
<tr>
<td>Test creation with test, (virtual) sequences, etc.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Configuration and factory mechanism</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Phasing and objections</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Policies to print</td>
<td>Yes</td>
<td>CPP allows usage of CPP methods to do compare. But need to write own detailed method for deep compare similar to UVM for seq_items etc.</td>
</tr>
<tr>
<td>Policies to Compare</td>
<td>Not Present</td>
<td></td>
</tr>
<tr>
<td>Policies to pack, unpack, etc.</td>
<td>Not present</td>
<td></td>
</tr>
<tr>
<td>Messaging and reporting</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Register abstraction layer</td>
<td>Not present</td>
<td></td>
</tr>
<tr>
<td>Callbacks</td>
<td>SCV callbacks</td>
<td>Can use SCV/ CPP callbacks</td>
</tr>
<tr>
<td>Coverage Groups</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Constrained randomization</td>
<td>SCV or CRAVE</td>
<td>Crave is stronger, default SCV randomization not as good</td>
</tr>
</tbody>
</table>
Initial Deployment

The above methodology has been applied to block verification with the following initial results:

- Re-usable TVM agents were developed for the proprietary protocols based internal busses. These were re-used on subsequent IP blocks.
- High rates of both functional and coverage were achieved on all blocks.
- 100% statement coverage on all blocks.
- The structured feature extraction process was converted to a functional coverage model. TVS were then able to generate tests to hit 100% of the plan.
- Throughout the coverage closure process progress was tracked via asureSIGN and reports generated.
- 24 bugs were discovered.
Portability

- Ported the whole environment and all the tests to Cadence SystemC simulator
  - Some minor changes required to fix crashes
    - For example - Conflict between global and local static variables
    - Cadence has a list of all of the issues and coding guidelines
  - Environment now ported and works
  - Need to follow Cadence coding guidelines when developing tests

- Tests now running on RTL under the Cadence simulator
  - Some timeouts needed fixing
Future updates planned

- Call-backs are not implemented. For now, as it is pure C++ library, C++ call-backs can be used (Courtesy: tedfelix.com/software/C++-callbacks.html).
- TVS has a Virtual Sequencer with p_sequencer based mechanism but it needs further improvement to make it more compatible with methods in UVM.
- There is currently no UVM-RAL model equivalent.
- `uvm_event`, `uvm_event_pool` and message passing using these functions need to be added.
- The functional coverage library support needs to be extended.
Bubble Sort “Proof of Concept” for SW Testing

• Program Specification
  – Input lists of integers, floats, ascii, etc.
  – Reject lists of mixed types
  – Convert unsorted lists to sorted lists

• Can we test the program with constrained input generation?
  – Generate valid and invalid inputs
  – Direct generation towards corner cases
  – Check outputs for correctness
    • Without re-writing an identical checker program
  – Measure what we have tested
Results of Bubble Sort “Proof of Concept”

Lists of
- Integers
- Floats
- Ascii
- etc

List Generator

Checkers
- Check output list is ordered
- Output list contents == input list contents

Software Under Test

Coverage Metrics
- Empty List
- Reverse ordered
- Error cases (mix integers, floats, ascii)
- Etc.

Constrain towards
- Empty lists
- Equal values
- Reverse ordering

Lists
Results for bubble sort
Application of Advanced HW verification to complex System SW Testing

- Robotic vacuum cleaner?
- Mars rover?
- Drone landing on ship in rough seas?

**Automotive**
- Automated parking?
- Lane keeping assistance?
- Driverless cars

**Do we continue to perform directed testing?**
- Hardware adopted constrained random verification
- And ensure requirements tracing for safety standards compliance with asureSIGN
Availability

- The SystemC-UVM infrastructure developed by TVS is available for other engineers to use.
- Compiles to binary and can execute multiple versions in parallel.
- The libraries can be obtained through the TVS website:
  
  http://www.testandverification.com
Thank You