Versatile UVM Scoreboarding

Jacob Andersen, Peter Jensen, Kevin Steffensen

SyoSil ApS, Copenhagen, Denmark

ABSTRACT

All UVM engineers employ scoreboarding for checking DUT versus reference model behavior, but only few spend their time wisely by employing an existing scoreboard architecture. Main reason is that existing frameworks have inadequately served the user needs, and have failed to accelerate the user efficiency in the debug situation. This work presents a better UVM scoreboard framework, focusing on scalability, architectural separation and connectivity to foreign environments. Our scoreboard architecture has successfully been used in UVM testbenches at various architectural levels, across models (RTL, SystemC) and on physical devices (FPGA/ASICs).
Based on our work, the SystemVerilog/UVM user ecosystem will be able to improve how scoreboards are designed, configured and reused across projects, applications and models/architectural levels.

MOTIVATION – CURRENT LANDSCAPE

Addressing the increasing challenges met when performing functional verification, UVM proposes a firm and productive approach for how to build and reuse verification components, environments and sequences/tests. When it comes to describing how to scoreboard and check the behavior of your design against one or more reference models, the UVM code base as well as the UVM ecosystem offers less help:
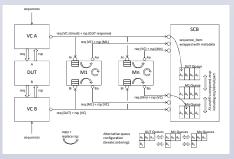
- UVM native scoreboard is empty
- Existing user donations are limited in versatility, employ blocking "expect" function as reference model[1]
- Expect function inhibits use of time consuming reference models (e.g. SystemC)
- Expect function inhibits use of multiple concurrent models

A reusable scoreboard is key for productivity and easy debug. We identify following user needs, naturally being addressed by our scoreboard architecture:

- Fast out of the box, easy to configure
- Consistent re-use
- Scalability (any number of models, queues, producers, compare methods)
- Clean interfaces to self contained models, e.g. SystemC
- Accelerated debug
- Inherently best performance
- Linear and not polynomial queue search complexity
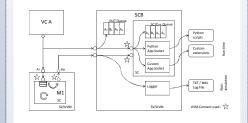- Advanced use : Connect to foreign environments

SCALABILITY & ARCHITECTURAL SEPARATION

Our scoreboard is able to simultaneously interface and compare any number of models: Design models (RTL, gate level), timed/untimed reference models (SystemVerilog, SystemC, C/C++, Python), as well as physical devices like FPGA prototypes/ASICs. As a logical consequence, we assume a clear architectural separation between the models and the scoreboard implementation, the latter containing queues and comparison mechanisms:
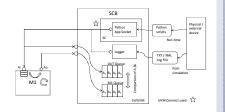

NON-UVM CONNECTIVITY

Besides interfacing to UVM using analysis ports, establishing links to non-UVM/non-SystemVerilog code is essential to keep the scoreboard versatile and reusable, enabling the use of external checkers and debug aiding scripts. For this purpose, the scoreboard framework offers a number of external interfaces:

The UVM Connect library enables seamless TLM1/TLM2 communication between SystemVerilog/UVM and SystemC [2]. We employ the library for implementing most run-time interfaces depicted above. For connecting analysis ports between SystemVerilog/SystemC, we employ uvmc_tlm1 sockets with generic payloads, using the "sv2sc2sv" pattern where SystemVerilog and SystemC both act as producer/consumer.

The scoreboard can be used with transactions streams from external resources, e.g. by obtaining logs from devices running in the lab (silicon, FPGA, emulators). Depending on the log format, we use either the XML Interface or the Python App Socket to retrieve the log transactions as UVM sequence items:

QUEUE ORGANIZATION

For each model (M1 … Mn) attached to the scoreboard, any number of queues can be handled. Each queue contains meta-transactions, wrapping UVM sequence items along with metadata. This allows same queue to contain different transaction types not necessarily comparable with each other. The metadata ensure that only queue elements of same type are compared.

CONFIGURATION AND QUEUE INSERTION

The scoreboard utilizes the UVM configuration database such that it can be reconfigured on the test case level. This allows changing e.g. the number of queues and compare algorithms. For instance, a user extension of cl_syoscb_cfg can be used for this purpose:

```
class cl_scb_myconfig extends cl_scb_uvm_config;
  function new(string name = "cl_scb_myconfig");
    scb_cfg.set_queues({"RTL", "M1"});
    scb_cfg.set_primary_queue("RTL");
    scb_cfg.set_producer("A", {"RTL", "M1"});
    scb_cfg.set_producer("B", {"RTL", "M1"});
  endfunction
endclass
```

Once the scoreboard is properly configured a standard uvm_sequence_item easily can be inserted into the scoreboard without manually managing the meta data. In the below example, the verification environment uses the transaction based API to retrieve the analysis export for connection with the analysis port of the verification component:

```
cl_scb_uvm scb;
…
myvc.ap.connect(scb.get_aexport("RTL", "A"));
```

Alternatively, manually add to queue by implementing an analysis export write method:

```
function void cl_myscb::write_A(A_seq_item item);
  this.add_item("RTL", "A", item);
endfunction
```
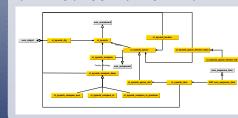
QUEUE COMPARISON METHODS

Pre-packaged comparison methods are available, for instance:

Custom compare methods are easily authored and configured for use on the testcase level. Compare methods are implemented using the built-in iterator/locator mechanism to search, traverse and compare the queues. Queue searching is done by calculating hash keys by MD5'ing byte representations of the sequence items, resulting in a linear rather than a polynomial search complexity.

IMPLEMENTATION

The generic scoreboard architecture is implemented by extending standard UVM base classes. This allows us to use the UVM Factory to specialize a scoreboard implementation, e.g. by changing the comparison algorithm for a specific test.

IMPROVEMENTS

Since first published [4], the underlying implementation of the UVM scoreboard has undergone some changes in order to improve the usability and tool compatibility with the three big simulator vendors. In general three major changes have been done:

- Changes in the class hierarchy
- Changed methods for enforcing APIs
- Minor changes to obtain simulator compatibility, now supporting
  - Synopsys VCS® version J-2014.12
  - Mentor Questa® Advanced Simulator version 10.3c
  - Cadence® Incisive® version 14.10.014

SUCCESS STORIES

Our UVM scoreboard architecture has been across numerous UVM/VMM projects. Typically we see such projects obtaining an approximate 15% code reduction compared to creating the scoreboard from scratch using the empty uvm_scoreboard class. Scoreboard setup, configuration and validation can be done in less than a day, even for complex designs, offering easy ramp-up for engineers new to UVM and the use of scoreboards. Furthermore, experienced engineers easily pick up and extend test benches created using the scoreboard library, as the scoreboard look and feel is same across applications and engineers. Out of the box, engineers benefit from an inherent top performing scoreboard with very good debug capabilities, prepared for hooking up to external interfaces.

CONCLUSIONS

In this work we propose an industry-proven, scalable UVM scoreboard architecture, able to interface to any number of design models across languages, methodologies, abstractions and physical form. Any relationship between data streams can be checked using pre-packaged and custom compare methods, and we make it easy to interface external checker and debug aiding applications. Based on our work, the SystemVerilog/UVM user ecosystem will be able to improve how scoreboards are designed, configured and reused across projects, applications and models/architectural levels.

GENERAL AVAILABILITY

Our UVM Scoreboard architecture has been released for general availability under the Apache 2.0 license, featuring the UVM Scoreboard base classes, examples as well as release notes and documentation.

The package can be obtained from following web resources:

Accellera UVM Forum:
   http://forums.accellera.org/files/file/119-versatile-uvm-scoreboard/
SyoSil homepage:
   http://syosil.com/index.php?pageid=33

Any suggestions for how to improve the base classes and examples are very welcome, including potential bug reports. Please direct such feedback per email to the authors at
   scoreboard@syosil.com

Get the Scoreboard UVM code today !

REFERENCES

[1] Accellera Forum, UVM Resources, http://forums.accellera.org/files/category/3-uvm/
[2] Mentor Graphics, UVM Connect Library, https://verificationacademy.com/topics/verification-methodology/uvm-connect
[3] Accellera Standards, SCE-MI (Standard Co-Emulation Modeling Interface), http://www.accellera.org/downloads/standards/sce-mi
[4] Versatile UVM Scoreboarding, Andersen, Jensen & Steffensen, DVCon Europe 2014
[5] "IEEE Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language." IEEE Std 1800-2012, 2012.