Versatile UVM Scoreboarding

Jacob Andersen, Peter Jensen, Kevin Steffensen
SyoSil ApS, Copenhagen, Denmark
A Scoreboard (Noun)
Scoreboarding (Verb)

- **Scoreboarding** is a ... method, ... , for dynamically scheduling a pipeline so that the instructions can execute out of order .... In a scoreboard, the data dependencies of every instruction are logged ...

(Source: Wikipedia)
HDL Scoreboards
(UVM, VMM, OVM, AVM, yourVM...)

DUT

REF
Motivation –

Current UVM Scoreboard Landscape

• UVM native scoreboard is empty
• Existing user donations are limited in versatility
  – Employ blocking “expect” function as REF
  – Inhibits use of time consuming REFs (e.g. SystemC)
  – Inhibits use of multiple concurrent models
• A reusable SCB is key for productivity and easy debug
  – We identify same principle structure across designs
• Some test bench stats from 8 SV test benches
  – 10 – 75K lines
  – Scoreboards were 3-25% of the test bench size, in avg 15%
SCB User Needs – What are those?

- 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. SC
- Accellerated debug
- Inherently best performance
  - Wants linear and not polynomial search complexity
- Advanced use: Connect to foreign environments
Scalability & Architectural Separation

• Any number of models
  – One primary model, multiple trailing models
  – Design models: RTL, gate, FPGA/ASIC postsil
  – Timed/untimed REF models: SV, SC, Python, C/C++, ...

• Any number of queues
  – One queue per model, per port, per transaction type, ...
  – Items in queues are tagged with metadata
SCB, Bi-dir Port Info Flow
(e.g. read/write SoC protocol)

VC A

DUT A

VC B

DUT B

sequences

req (VC stimuli) + rsp (DUT response)

req (VC) + rsp (M1)

req (VC) + rsp (Mn)

req (M1) + rsp (VC)

req (Mn) + rsp (VC)

req (DUT) + rsp (VC)

sequences

copy + replace rsp:

Alternative queue configuration (breaks ordering):

DUT Queues

M1 Queues

Mn Queues

sequence_item wrapped with metadata

A's and B's compared, resp. including req/stimuli part

© Accellera Systems Initiative
Metadata tagging of Sequence Items: One Queue per Model

Queue: Model X
Ports A,B

Queue: Model Y
Ports A,B

Comparison only if metadata tag matches
Metadata tagging of Sequence Items: One Queue per Model per Port

Comparison only if metadata tag matches
SCB, Uni-dir Port Info Flow
(e.g. packet switching)

Alternatively add A's to aid debug and comparison rules:
Connectivity

• Logs for post-sim analysis
  – TXT, XML

• Analysis scripts, run-time
  – C/C++/Python

• Non-SV models, run-time
  – C/C++/SC/Py/FPGA/ASIC
Connectivity; external extensions

VC A

DUT Queue
A_1 B_1 A_2 A_3

SCB

SC/C++ Queue
A_1 B_1 A_2 A_3

Python App Socket

Custom App Socket

Logger

TXT / XML Log File

Python scripts

Custom extensions

Run-time

Post-simulation

SV/UVM

M1 SC

SV/UVM

UVM Connect used:
Connectivity; external devices

- SCB
- Python App Socket
- Logger
- DUT Queue
- M1 Queue
- Python scripts
- TXT/XML Log File
- Physical/external device
- Run-time
- Post-simulation
- UVM Connect used:

© Accellera Systems Initiative
Configuring the SCB

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

Using the UVM Factory, the SCB Configuration can be changed on the test case level
Insertion of Elements into Queues

Use analysis export offered by API:

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

Implement analysis export write method, manually add to queue:

```plaintext
function void cl_myscb::write_A(A_seq_item item);
    this.add_item("RTL", "A", item);
endfunction
```
3 Simple Built-in Comparison Methods

In Order

<table>
<thead>
<tr>
<th>A1</th>
<th>B1</th>
<th>A2</th>
<th>A3</th>
</tr>
</thead>
</table>

matches

<table>
<thead>
<tr>
<th>A1</th>
<th>B1</th>
<th>A2</th>
<th>A3</th>
</tr>
</thead>
</table>

In Order by Producer

<table>
<thead>
<tr>
<th>A1</th>
<th>B1</th>
<th>A2</th>
<th>A3</th>
</tr>
</thead>
</table>

matches

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>B1</th>
<th>A3</th>
</tr>
</thead>
</table>

Out of Order:

<table>
<thead>
<tr>
<th>A1</th>
<th>B1</th>
<th>A2</th>
<th>A3</th>
</tr>
</thead>
</table>

matches

<table>
<thead>
<tr>
<th>A3</th>
<th>A2</th>
<th>A1</th>
<th>B1</th>
</tr>
</thead>
</table>

Custom compare methods are easily authored and configured for use on the testcase level.
How to Compare Efficiently?

• Do not call A1.compare(A3) repeatedly
  – Polynomic complexity!

• Our SCB instead
  – On insertion, SCB
    • Calculates hash key by MD5’ing A1.pack_bytes()
    • Inserts A1 into associative array
  – On comparison, SCB
    • Calculates hash key by MD5’ing A3.pack_bytes()
    • Checks whether the key exists in the associative array
  – Linear Complexity!
Extensions of UVM Classes
Debug Capabilities

• Comparison error reports
  – What went wrong, remaining contents of queues
• Full scoreboard debug
  – The full queue trace
  – From time zero, or certain window/file size
• Use extern APIs
  – Analysis scripts in e.g. Python
  – XML analysis tools & transformations
Success Stories

• Used across UVM/VMM projects
• 15% code (15% time) saved
• We do SCB setup/config plus validation in less than a day for even complex designs
• Easy for newbees
• Same look&feel across all SCBs
• Out of the box, inherent
  – Top performance
  – Very good debug capabilities
  – Prepared for external interfaces
We are sharing the SCB

• What’s in the package?
  – UVM SCB source code
  – Docs
  – Examples

• Register to get code, email
  – info@syosil.com
Questions
Comments
Remarks

Thank you!