A Functional Coverage Prototype for SystemC-based Verification of Chipset Designs

Robert Siegmund, Ulrich Hensel, Andreas Herrholz, Isa Volt
AMD Dresden Design Center
Outline

• Chipset Verification using SystemC
• SystemC Coverage API Prototype
  – Cover items and buckets
  – Toggle coverage
  – Cross coverage
  – Coverage reports
• Coverage Hole Analysis
• Conclusions and Summary
Chipset Verification using SystemC

SystemC 2.0

Constrained Random Generation

???

Functional Coverage

Check

DUV

BFM

Monitor

Trans

Host Bus

Device Bus

Trans

C++

SystemC Verification Lib

G

Trans

BFM

Monitor

G

Trans

SystemC 2.0

26.02.2004

Functional Coverage Prototype for SystemC-based Chipset Verification
Concept of Functional Coverage

Bucket fill grades are a measure for functional coverage.
Coverage API for SystemC

Definition of coverage items and buckets

```c++
// create/switch to active coverage database
sr_cover::set_database("bus_trans_coverage");

// define coverage items
sr_cover::create_item(trans.get_addr(), "addr");

// specify buckets for each coverage item
// create bucket [0x1000,0x2fff] with hit goal 5
sr_cover::get_item("addr").bucketize(sc_uint<64>(0x1000), sc_uint<64>(0x2fff), "ac97_space", 5);

// OPTIONAL:special OTHERS bucket
sr_cover::get_item("addr").bucketize(SR_OTHERS);
```

Goal: 5 hits required
Coverage API for SystemC

Coverage API supports definition of `illegal` and `ignore` ranges

```cpp
// specify ignorable ranges for addr
sr_cover::get_item("addr").ignore(
    sr_cover::get_item("addr")() > sc_uint<64>(0x20) &&
    sr_cover::get_item("addr")() < sc_uint<64>(0x3F));

// specify illegal ranges for addr
sr_cover::get_item("addr").illegal(
    sr_cover::get_item("addr")() == sc_uint<64>(0x80));
```
Coverage API for SystemC

Instrumentation of testbench code for coverage analysis

```c
void systemc_process() {

    trans* t; // Bus transaction
    ...

    if(cover_condition) {

        // cover value of fields in transaction
        sr_cover::get_item("addr").update(t->get_addr());
        sr_cover::get_item("cmd").update(t->get_cmd());
        ...
    }
}
```
Toggle Coverage API

Goal: Have all pins/selected signals toggled at least once?

```cpp
sc_in<bool> clk;
sc_out<sc_logic> dout;
sc_signal<bool> data;

// include pins 'clk','dout' in toggle coverage
sr_toggle_coverage::add_pin(clk,"clk");
sr_toggle_coverage::add_pin(dout,"dout");

// specify transitions to 'X' illegal for data
sr_toggle_coverage::illegal_trans("data",sc_logic('0'),sc_logic('X'));

// ignore 1->0 transitions on clk
sr_toggle_coverage::ignore_trans("clk",1,0);
```

Ports and signals of type bool and sc_logic supported.
Cross Coverage API

Cross coverage = cross product of cover items at times t

Application: Covering verification corner cases

```cpp
// cross items "addr", "cmd"
// the cross product item is named "addr_X_cmd"
sr_cover::cross("addr_X_cmd", "addr", "cmd");
```

```
addr.update(0x3455)  
X

cmd.update(write)  
```

```
addr 0x1000..0x2fff  0x3000..0x4fff  0x5000..0x6fff  0x7000..0xffff
```

```
read write
```

```
addr X cmd
```

// cross items "addr", "cmd"
Post Run Analysis of Coverage Holes

Domain of covered type

- User defined item buckets
- minus Ignore expression
- minus Buckets with hits after simulation

= Coverage hole!

Coverage holes are computed automatically using a BDD-based analysis algorithm
### Coverage Reports

** rqst_type **

<table>
<thead>
<tr>
<th>grade</th>
<th>goal</th>
<th>samples</th>
<th>%total</th>
<th>rqst_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
<td>7.41</td>
<td>rqst_trans::msr_write</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>44</td>
<td>54.3</td>
<td>rqst_trans::msr_read</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4.94</td>
<td>rqst_trans::io_read</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4.94</td>
<td>rqst_trans::io_write</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4.94</td>
<td>rqst_trans::non_coh_read</td>
</tr>
</tbody>
</table>

Coverage holes:  
- rqst_trans::coh_read_be
- rqst_trans::coh_write
- rqst_trans::non_coh_write

Result of coverage hole analysis
Conclusion and Summary

- We presented a prototype to measure and analyze functional coverage in SystemC-based verification.

- Functional coverage prototype has been used in SystemC-based verification of an AMD chipset design.

- Using the proposed coverage API, coverage analysis was easy to integrate in in-house SystemC verification environment.

- Coverage API is fully SystemC 2.0 compliant (implemented as add-on library to SystemC).

- Automatic coverage hole analysis provided us a tool to
  - measure the quality of stimulus
  - check if corner cases have been thoroughly exercised.
Trademark Attribution

AMD, the AMD Arrow Logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this presentation are for identification purposes only and may be trademarks of their respective companies.