Goal Driven Stimulus Solution
Get yourself out of the redundancy trap

Rohit Bansal
Samsung Semiconductor India R&D,
Bangalore
Outline

• Introduction
• Conventional approach drawbacks
• Proposed Stimulus Solution
• Method implementation with examples
• Results
• Limitations
• Benefits
• Future scope
• References
Introduction

• Functional coverage - One of the key metrics in design verification

• Conventional coverage closure approach
  – Code constrained random test cases --> Run regression --> Review coverage results --> Add test cases for coverage holes --> Run newly added test cases and merge coverage results

![Graph showing coverage improvement over time with slow improvement rate after ~80% and directed closure]

© Accellera Systems Initiative
Conventional Approach Drawbacks

• Redundancies
  – Resource utilization like Load Sharing Facility (LSF), tool licenses, disk usage, etc.
  – Coverage analysis effort
  – Directed test case addition
• Possibility of late design bugs
• Closure time increases exponentially with coverage goals
• Reusability and Scalability
  – Randomization sensitive to changes in stimulus generation code
• All of the above add up with design size → more $$$
• Need faster and cost efficient process to coverage closure
Proposed Stimulus Solution

- Test case coding using smart constraint modelling
  - Uses feedback from earlier randomization calls to guide constraint solver
  - Targets both individual and cross coverage metrics
  - Simple and easy to adopt

```
Already Randomized DB A[$], B[$], C[$]

Query DB

If (Success) Update DB

Randomize(a,b,c) ! (a in A[i] && b in B[i] && c in C[i])

Get total bins

Repeat coverage bins time

Coverage handle

Coverage goal met

Randomization failure could mean over constraints

Not meeting the goal means too few constraints
```

© Accellera Systems Initiative
Method Implementation

• Following macros are defined to implement proposed method

  – uvm_optimization_utils macro

    `uvm_optimization_utils (var_name, var_type, var_hier)`

    1. Define constraint - !var_hier inside q_var_name
    2. Define write method which does file write operations to create database queue and write generated values to database queue

    Code enabled after first run

    DB Queue – var_name_db.sv
    q_var_name [$];

  – ADD_ALREADY_RANDOMIZED macro

    `ADD_ALREADY_RANDOMIZED (var_name, var_val)`

    • Calls the write method defined by above macro for var_name and adds var_val to database

    DB Queue – var_name_db.sv
    q_var_name [$] = {1, 2, ...};

  – CLEAR_PARAMS_QUEUE_COV macro

    `CLEAR_PARAMS_QUEUE_COV (var_name, exp_cov_bins)`

    • If the DB queue size for var_name is equal to expected coverage, the queue is cleared
Individual variable coverage example

```verilog
//-----Test-----
class my_test extends uvm_test;
    `uvm_component_utils(my_test)
    rand `traffic_seq a1;
    rand config_seq b1;
    `uvm_optimization_utils(length.int,a1.length)
    `uvm_optimization_utils(Size.int,a1.Size)
    `uvm_optimization_utils(config_a, int,b1.config_a)

covergroup cg:
    c1 : coverpoint a1.length { bins a[] = {{1:20}}; }
    c2 : coverpoint a1.Size { bins a[] = {{1:10}}; }
    c3 : coverpoint b1.config_a { bins a[] = {{1:5}}; }
    c1xc2xc3 : cross c1,c2,c3;
endgroup

function new (string name = "my_test", uvm_component parent = null);
    super.new(name, parent);
    cg = new;
endfunction : new

function post_randomize();
    `ADD_ALREADY_RANDOMIZED(length,a1.length)
    `ADD_ALREADY_RANDOMIZED(Size,a1.Size)
    `ADD_ALREADY_RANDOMIZED(config_a,b1.config_a)
endfunction

function pre_randomize();
    int act_c1,act_c2,act_c3;
    int total_c1,total_c2,total_c3;
    cg.c1.get_coverage(act_c1,total_c1,c1);    // CLEAR_PARAM_QUEUE_COV(length,total_c1)
    cg.c2.get_coverage(act_c2,total_c2,c2);    // CLEAR_PARAM_QUEUE_COV(Size,total_c2)
    cg.c3.get_coverage(act_c3,total_c3,c3);    // CLEAR_PARAM_QUEUE_COV(config_a,total_c3)
endfunction

function void build_phase (uvm_phase phase);
    super.build_phase(phase);
endfunction : build_phase

task run_phase(uvm_phase phase);
    a1 = `traffic_seq::type_id::create("a1");
    b1 = config_seq::type_id::create("b1");
    phase.raise_object(this);
    this.randomize();
    cg.sample();
    phase.drop_object(this);
endtask : run_phase
endclass : my_test
```
Cross Coverage Example

- Flexibility to add multiple variables in cross constraint
- Note: Use total cross bins for database clear

class my_test1 extends uvm_test;
`uvm_component_utils(my_test1)

`REGISTER_OPTIMIZE_VAR(length, int)
`REGISTER_OPTIMIZE_VAR(Size, int)
`REGISTER_OPTIMIZE_VAR(config_a, int)

`ADD_OPTIMIZE_CROSS_CONSTRAINT_BEGIN(length_Size, length, a1.length)
  `ADD_OPTIMIZE_CROSS_CONSTRAINT_FIELD(Size, a1.Size)
  `ADD_OPTIMIZE_CROSS_CONSTRAINT_FIELD(config_a, bl.config_a)
`ADD_OPTIMIZE_CROSS_CONSTRAINT_END

`REGISTER_OPTIMIZATION_RANGES_BEGIN(length, int)
  `ADD_OPTIMIZATION_RANGE(2, 5)
  `ADD_OPTIMIZATION_RANGE(6, 9)
`REGISTER_OPTIMIZATION_RANGES_END

`REGISTER_OPTIMIZATION_RANGES_BEGIN(Size, int)
  `ADD_OPTIMIZATION_RANGE(2, 5)
  `ADD_OPTIMIZATION_RANGE(6, 15)
  `ADD_OPTIMIZATION_RANGE(16, 19)
`REGISTER_OPTIMIZATION_RANGES_END

function new (string name = "my_test1", uvm_component parent = null);
  super.new(name, parent);
endfunction
endclass
Cross Coverage Example

• uvm_optimization_utils macro split into two

REGISTER_OPTIMIZE_VAR (var_name, var_type)
-> Define write method which does file write operations to create database queue and write generated values to database queue

ADD_OPTIMIZE CROSS CONSTRAINT BEGIN
(constraint_name, var_name, var_hier)
  ➔ constraint constraint_name {!(var_hier inside q_var_name}
ADD_OPTIMIZE CROSS CONSTRAINT FIELD
(var_name, var_hier)
  ➔ && var_hier inside q_var_name
ADD_OPTIMIZE CROSS CONSTRAINT END
  ➔ );}
Ranged Cover Bins Handling

• Database implemented as queue of queues
• Helps avoid generation in same range
  – Additional ranges registration modelling effort

REGISTER_OPTIMIZATION_RANGES_BEGIN (var_name, var_type)
⇒ var_type q_range_var_name[$][$] = {
ADD_OPTIMIZATION_RANGE (start_val, end_val)
⇒ {start_val, end_val},
REGISTER_OPTIMIZATION_RANGES_END
⇒ {}};
Ranged Cover Bins Handling

- Macro modified for new queue data type

```plaintext
write (var_type val)
  -> Query_range_db
  (val, ref var_type
   range[$])
  -> Update DB

If (val inside some
range), return range;
Else return {val};
```

```plaintext
uvm_optimization_utils (var_name, var_type,
  var_hier)
  1. Define constraint
     foreach(q_var_name[i]) (!var_hier inside
      q_var_name[i])
  2. Define write method for writing the queue
to a file
  3. Define method to query range database

DB Queue -
var_name_db.sv
q_var_name [$][$] =
{"1},
{2,5}, ...;
```
Regression flow

• In order to make better utilization of LSF resources
  – Run pre-run script on local machine
  – Submit run script to LSF

Pre run script waits for any variable’s queue size in database to reach value as passed by argument

Pre: script.pl 1
Run first iteration of Test1
Run second iteration of Test1
Pre: script.pl 2
Run second iteration of Test1
Pre
And so on, till regression count
Run first iteration of Test2

Time
Results

<table>
<thead>
<tr>
<th>Discussed Example</th>
<th>Regression count</th>
<th>Parallel runs</th>
<th>Coverage Attained</th>
<th>Simulator Time</th>
<th>Regression Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional</td>
<td>5000</td>
<td>100</td>
<td>90%</td>
<td>3h</td>
<td>3h</td>
</tr>
<tr>
<td>Proposed</td>
<td>1000</td>
<td>100</td>
<td>100%</td>
<td>40min</td>
<td>2h</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interconnect DV (~100 masters)</th>
<th>Transaction count</th>
<th>Regression time</th>
<th>Coverage Attained</th>
<th>Man days</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional</td>
<td>3000</td>
<td>25h</td>
<td>85%</td>
<td>Exp: ~10</td>
</tr>
<tr>
<td>Proposed</td>
<td>320</td>
<td>2.5h</td>
<td>100%</td>
<td>Act: 2</td>
</tr>
</tbody>
</table>

Figure: Basic Example
- Man hours in days
- Machine Utilization in hours
- Conventional approach: 3, 2, 1, 0
- Discussed approach: 1, 0, 1, 2
- Decrease: 3x

Figure: Interconnect DV
- Man hours in days
- Machine Utilization in hours
- Conventional approach: 20, 15, 10, 5, 0
- Discussed approach: 5, 5, 10
- Decrease: 5x, 10x
Limitations

• Bigger designs with complex constraints and huge number of variables
  – Increased constraint solving time and modelling effort

• Transition coverage handling

• Test bench with redundant randomization calls
  – Result in valid scenarios scoped out of constraint solver space

• Limitations due to proposed synchronization scheme
  – Randomization happening late in simulation difficult to handle
  – More regression run time for smaller designs
Key Benefits

• Easy to adopt
  – Uses existing SV language features to implement
  – No dependency on third party tools or models

• Faster automated coverage closure and hence significant cost savings
  – Reduced coverage analysis effort, redundant resource utilization

• Can be used to target complete or partial coverage goals

• Leave time to explore more corner cases

• No test ranking needed

• Constraints and coverage can be scaled as per design
  – No extra coverage closure effort
Future scope

• Deploy and check for more complex cases
• Automate the modelling process
• Reduce synchronization overhead
• Add and test support for transition coverage
References


Thank you!

rohit.bl@samsung.com