Goal Driven Stimulus Solution

Get yourself out of the redundancy trap

Rohit Bansal

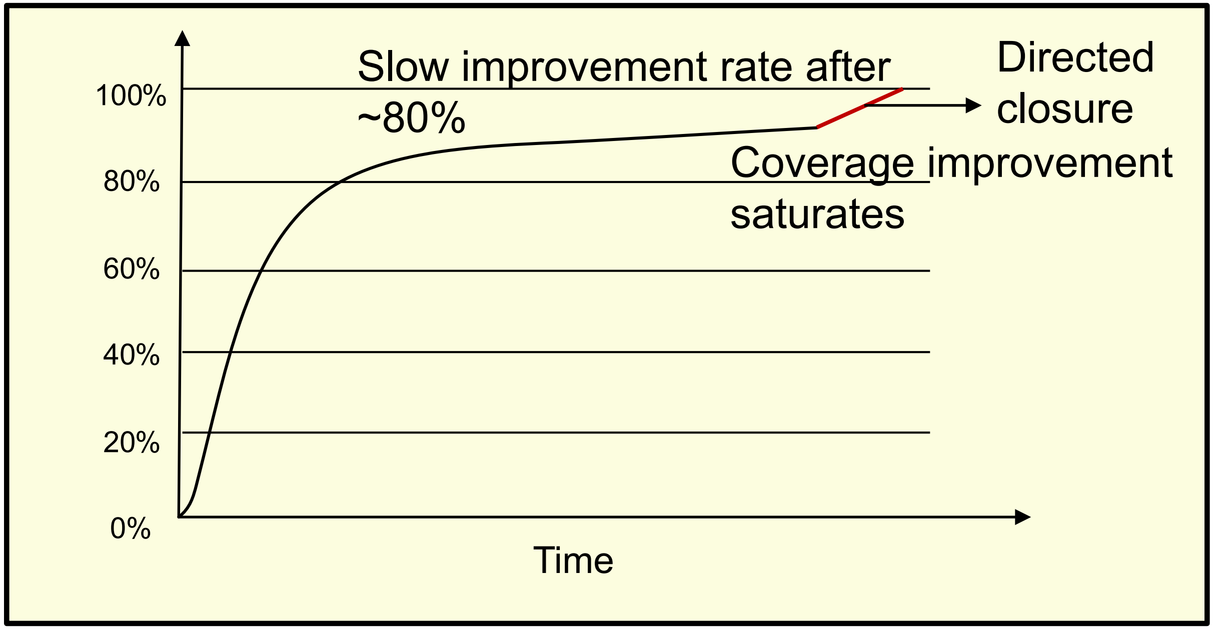
Samsung Semiconductor India R&D, Bangalore

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

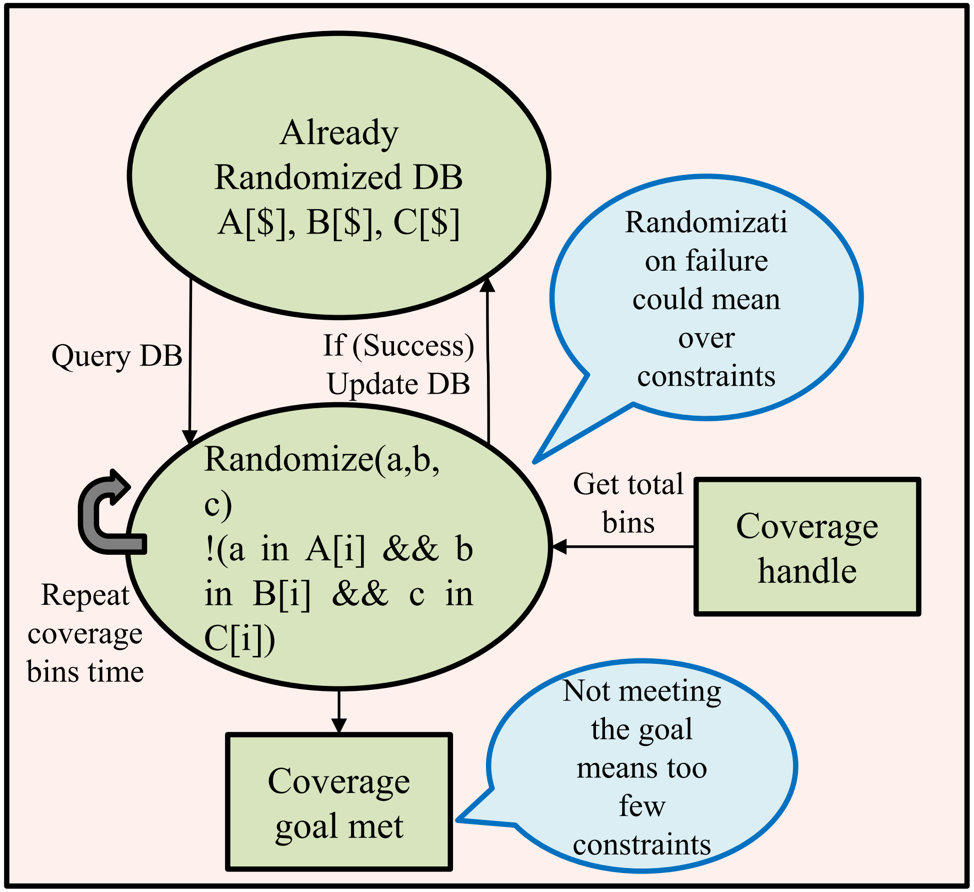


Conventional Approach Drawbacks

- 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



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 (1)

→ DB Queue – var_name_db.sv q_var_name [$]; (2)

  - 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

```
//-----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_cov_c1,act_cov_c2,act_cov_c3;
   int total_cov_c1,total_cov_c2,total_cov_c3;
   cg.c1.get_coverage(act_cov_c1,total_cov_c1);
   cg.c2.get_coverage(act_cov_c2,total_cov_c2);
   cg.c3.get_coverage(act_cov_c3,total_cov_c3);
   `CLEAR_PARAM_QUEUE_COV(length,total_cov_c1)
   `CLEAR_PARAM_QUEUE_COV(Size,total_cov_c2)
   `CLEAR_PARAM_QUEUE_COV(config_a,total_cov_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_objection(this);
   this.randomize();
   cg.sample();
   phase.drop_objection(this);
 endtask : run_phase
endclass : my_test
```

```
class traffic_seq extends uvm_sequence;
  ...
  constraint length_c {length inside {[1:10]};};
  constraint Size_c {Size inside {[1:20]};};
  ...
endclass
```

```
class config_seq extends uvm_sequence;
  ...
  constraint config_a_c {config_a inside {[1:5]};};
  ...
endclass
```

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,b1.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 : new
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

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

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

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

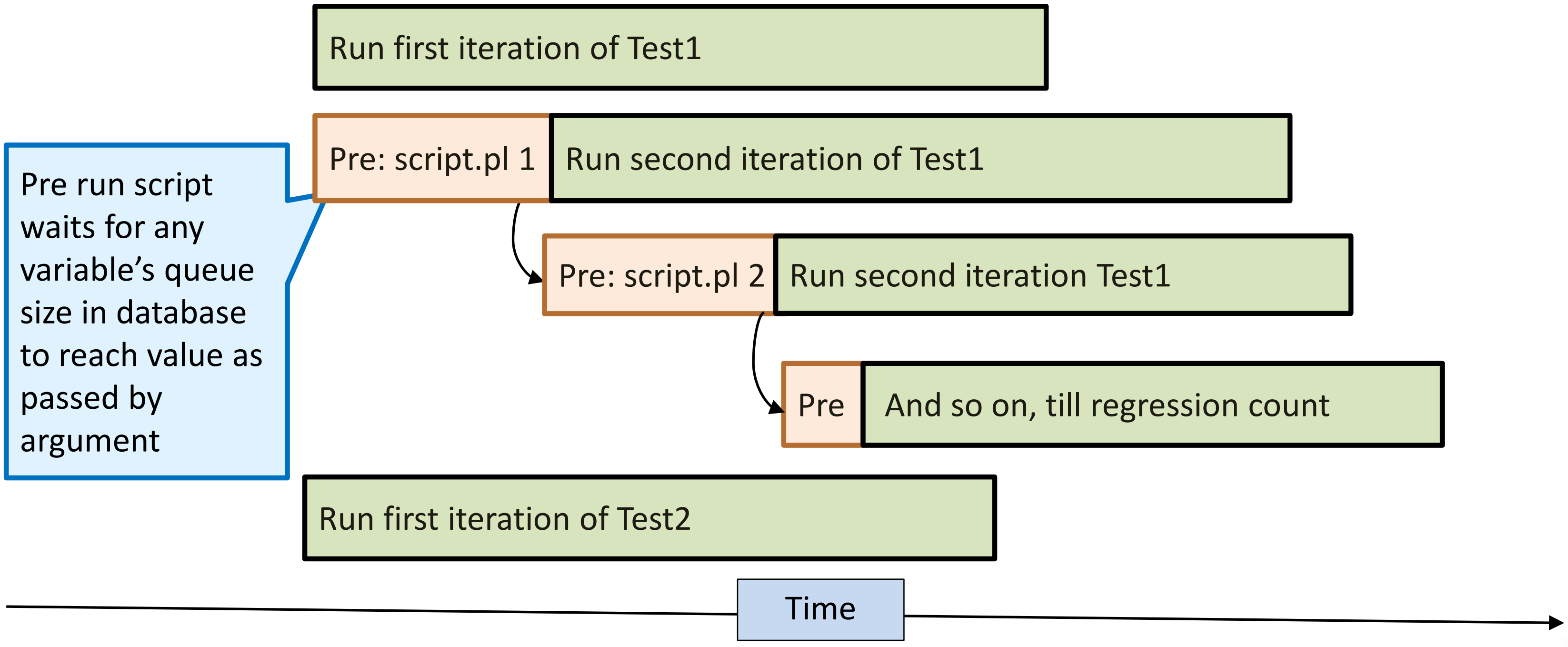
ADD_ALREADY_RANDOMIZED

⇩

DB Queue - var_name_db.sv
q_var_name [$][$] = {{1}, {2,5}, …};

Regression flow

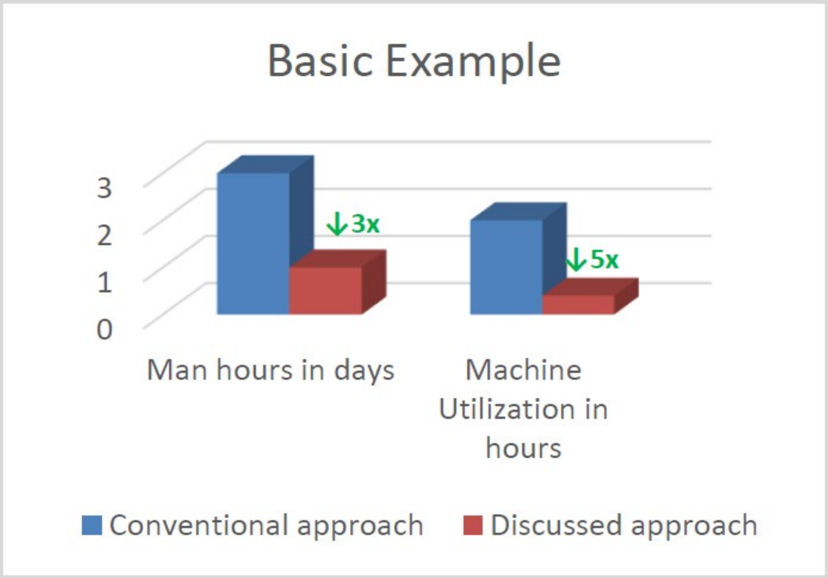
- In order to make better utilization of LSF resources
  - Run pre-run script on local machine
  - Submit run script to LSF

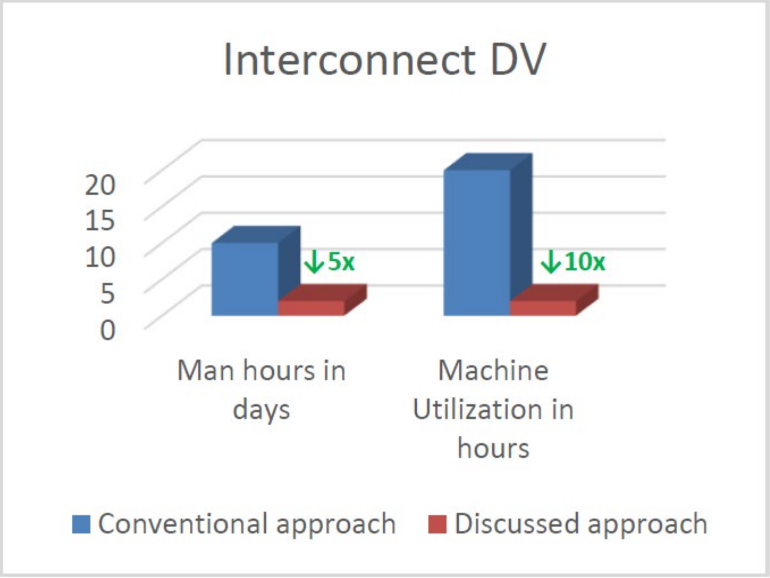


Results

Discussed Example	Regression count	Parallel runs	Coverage Attained	Simulator Time	Regression Time
Conventional	5000	100	90%	3h	3h
Proposed	1000	100	100%	40min	2h

Interconnect DV (~100 masters)	Transaction count	Regression time	Coverage Attained	Man days
Conventional	3000	25h	85%	Exp: ~10
Proposed	320	2.5h	100%	Act: 2





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

[1] "IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language", *IEEE standard 1800-2012*, IEEE-SA Standards Board, New York, 2012.

[2] Chris Spear, *System Verilog for Verification: A Guide to Learning theTestbench Language Features*, New York: Springer, 2012.

[3] Eldon Nelson, "Improving Constrained Random Testing by Achieving Simulation Verification Goals through Objective Functions, Rewinding and Dynamic Seed Manipulation", [Online document], Available at HTTPS: https://dvcon.org/sites/dvcon.org/files/files/2017/07_1.pdf

[4] Aditya Sharma, T. Nagasundaram, "Coverage Closure – Is it a "Game of Dice" or "Top 10 Tests" or "Automated Closure"? ", [Online document], Available at HTTPS: https://dvconindia.org/sites/dvconindia.org/files/archive/2015/proceedings/111_Coverage_Closure.pdf

[5] Cadence, "Xcelium version 18.03" [Online]. Available at HTTP:http://www.cadence.com

[6] Synopsys, "VCS version VN-2017.12" [Online]. Available at HTTP:http://www.synopsys.com

Thank you!

rohit.bl@samsung.com