2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Fully Automated Functional Coverage Closure

Manohar Kodi, Nvidia Graphics India Pvt Ltd.
Sagar Sudam Patil, Nvidia Graphics India Pvt Ltd.
Ranjith Nair, Nvidia Graphics India Pvt Ltd.

NVIDIA®

accellera SYSTEMS INITIATIVE

2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

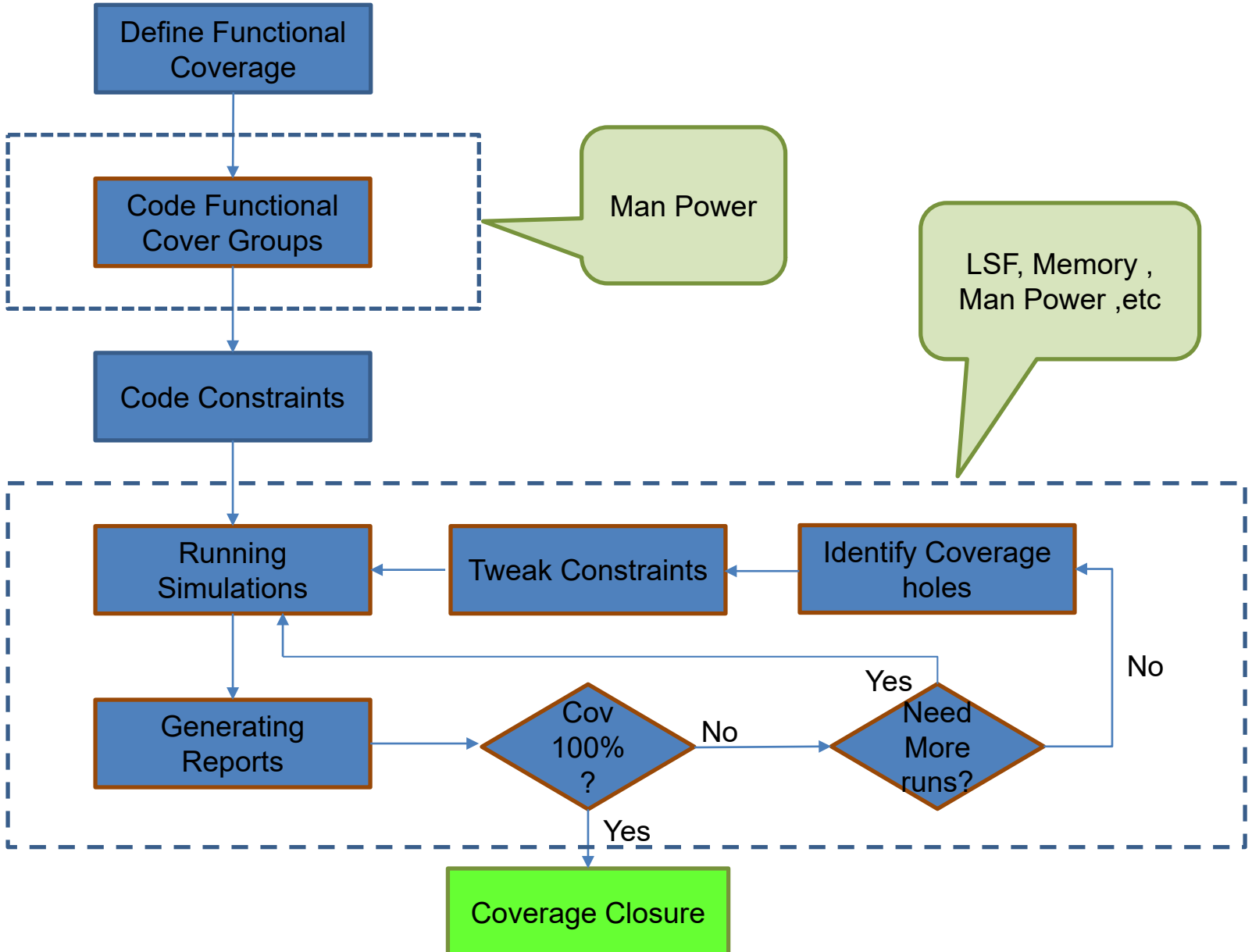
- Functional Coverage
- Methodology Introduction
- Details of the Methodology
- Scripts
- Results

accellera SYSTEMS INITIATIVE

Functional Coverage

- Functional coverage is the main metric for measuring the stimulus quality in metric driven verification.
- Verification engineers use it to sign off complex design verification features.

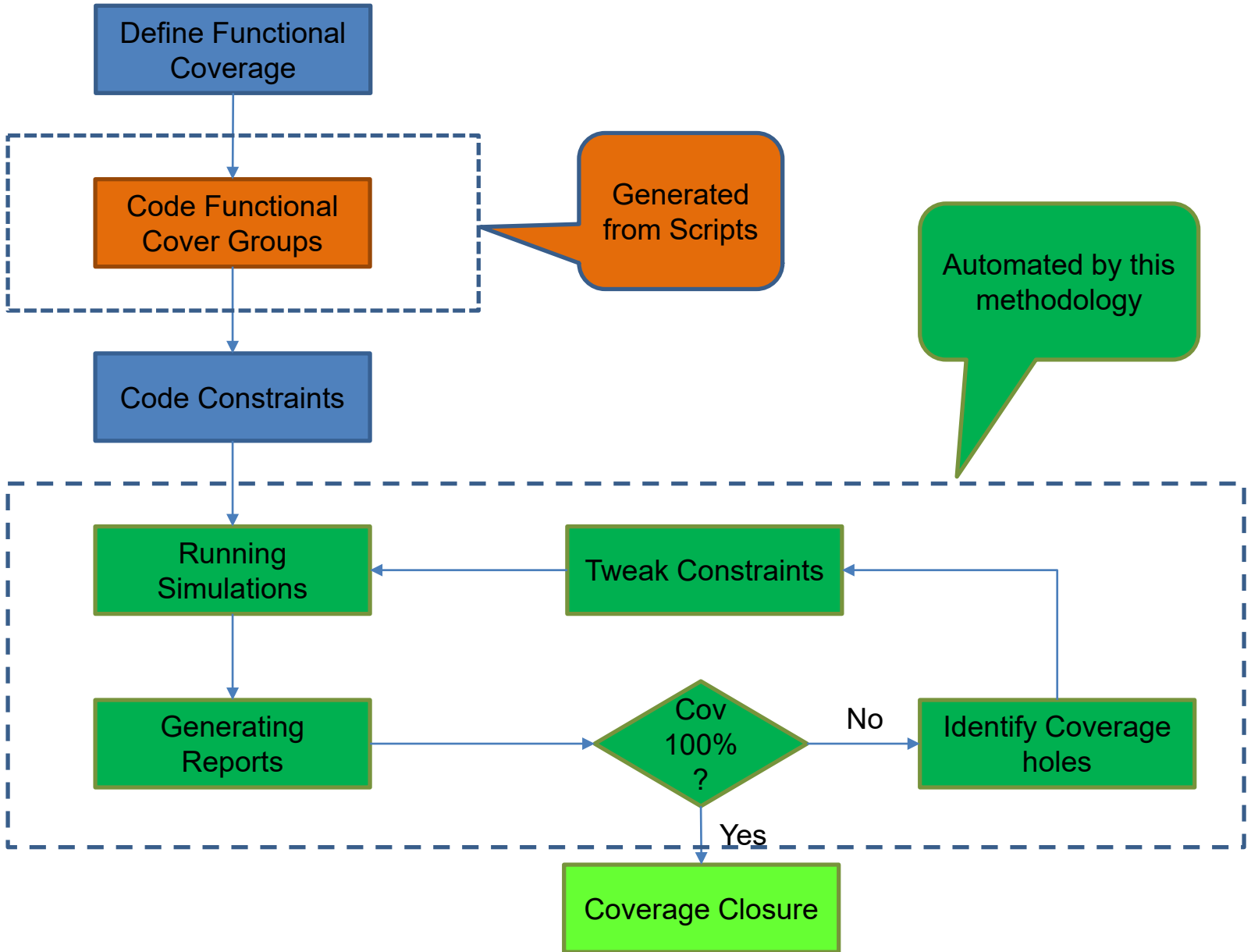
Functional Coverage As We Know



Traditional Functional Coverage

- Significant amount of time and effort is spent only to reach the total coverage which may still not be 100%.
- Need to analyze the coverage and override the constraints, rerun the tests and merge again.
- Lot of resources like LSF(Load Sharing Facility), memory, man power etc.

Automated Functional Coverage



Introduction to Methodology

- The main contributions of the method is a way to reduce the number of LSF and remove manual effort required to hit the 100% functional coverage.
- Automatic functional coverage closure using a systematic way of writing functional coverage points.
- Leveraging the system functions provided by SV LRM to tweak the constraints based on the coverage DB.

Traditional way of coding cover group

```
class sample_coverage_monitor extends uvm_monitor;
  bit var_1;
  bit [1:0] var_2;
  bit [2:0] var_3;
covergroup TraditionalSampleCovGrp;
    option.per_instance = 1;
    cov_var_1  : coverpoint var_1 {bins     b0 = {0}; bins     b1 = {1};}
    cov_var_2  : coverpoint var_2 {bins     b0 = {0}; bins     b1 = {1}; bins    b2 = {2}; bins     b3 = {3}; }
    cov_var_3  : coverpoint var_3 {bins     b0 = {0}; bins     b1 = {1}; bins     b4 = {4}; bins     b5 = {5};
                                   bins     b6 = {6};}
    cross_cov_var_1_cov_var_2 : cross cov_var_1,cov_var_2;
  endgroup
endclass
```

Systematic way of coding cover group

```
class sample_coverage_monitor extends uvm_monitor;
  bit var_1;
  bit [1:0] var_2;
  covergroup SampleCovGrp;
   option.per_instance = 1;
   cov_0_var_1 : coverpoint var_1 {bins b = {0};}  cov_1_var_1 : coverpoint var_1 {bins b = {1};}
   cov_0_var_2 : coverpoint var_2 {bins b = {0};}  cov_1_var_2 : coverpoint var_2 {bins b = {1};}
   cov_2_var_2  : coverpoint var_2 {bins b = {2};}
   cross_cov_0_var_1_cov_0_var_2 : cross cov_0_var_1,cov_0_var_2;
   cross_cov_0_var_1_cov_1_var_2 : cross cov_0_var_1,cov_1_var_2;
   cross_cov_0_var_1_cov_2_var_2 : cross cov_0_var_1,cov_2_var_2;
endgroup
endclass
```

Coverage System Tasks

- $load_coverage_db (name) — Load from the given filename the cumulative coverage information for all coverage group types.
- $get_coverage ( ) — Returns as a real number in the range 0 to 100 the overall coverage of all coverage group types. This number is computed as described above.
- get_coverage() — Calculates type coverage number (0...100) for each cover group or cover point or cross.

Random Test Example

```
int array_cov_value[1][$];
if(env.cov_10g.SampleCovGrp.cross_cov_0_var_1_cov_0_var_2.get_coverage() == 100) begin //{
  var12_val[2] = 0; var12_val[1:0] = 0;
  idx_ar = array_cov_value[0].find_first_index(x) with (x == var12_val);
  array_cov_value[0].delete(idx_ar[0]);
  array_num_point[0] = array_num_point[0] - 1;
  cfg_1.var_1 = 0;
  cfg_1.var_2 = 0;
end //}
if(array_cov_value[0].size() != 0) begin //{
    randomize(var_1_2) with {var_1_2 inside {array_cov_value[0]}; };
    cfg_1.var_1 = var_1_2[2];
    cfg_1.var_2 = var_1_2[1:0];
    end //}
```

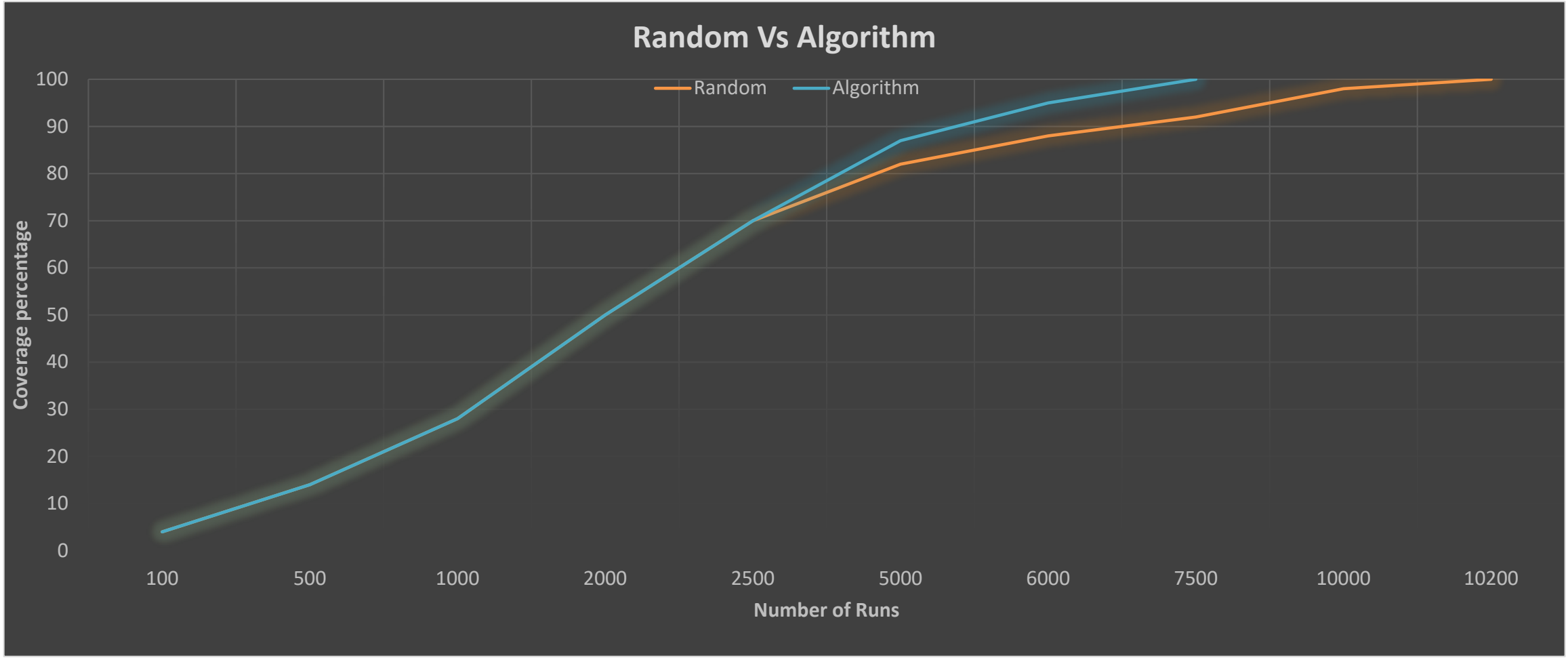
Scripts

- Generate functional cover group and logic inside the random test from the xls cover points.
- Script to run regression and break running the regression after achieving 100% functional coverage.

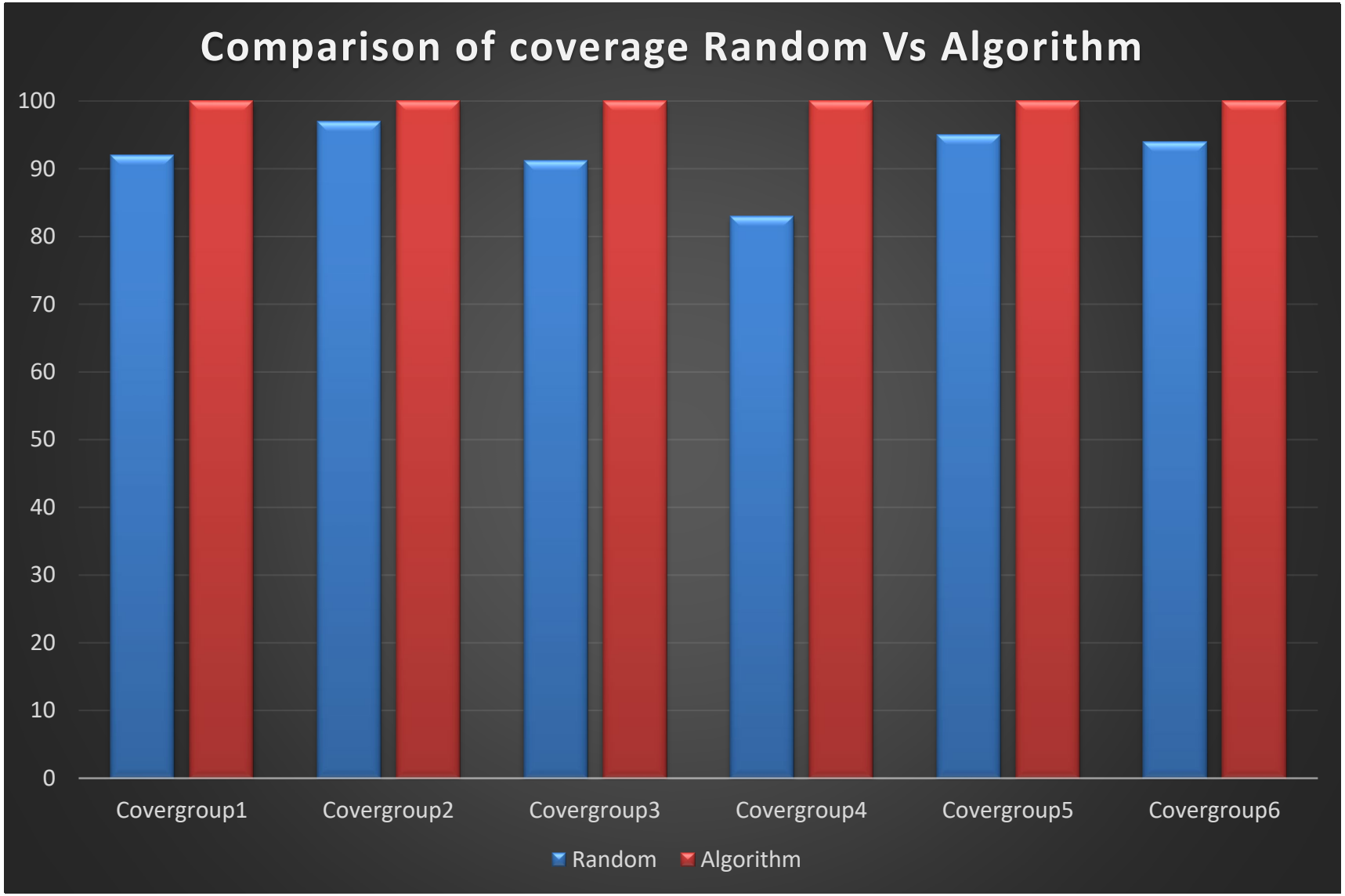
Table with cover group and runs for coverage closure

Cover Group Name	Number of Bins	Number of Runs for 100% using algorithm
CoverGroup1	7867	1600
CoverGroup2	10410	1600
CoverGroup3	260	200
CoverGroup4	860	700
CoverGroup5	13902	6000
CoverGroup6	8017	7500

Results


Random Vs Algorithm
Random
Algorithm
Coverage percentage
100
90
80
70
60
50
40
30
20
10
0
100
500
1000
2000
2500
5000
6000
7500
10000
10200
Number of Runs

Results


Comparison of coverage Random Vs Algorithm
100
90
80
70
60
50
40
30
20
10
0
Covergroup1
Covergroup2
Covergroup3
Covergroup4
Covergroup5
Covergroup6
Random
Algorithm

Conclusion

- Adopting a fully automated functional coverage closure method can substantially reduce the number of tests in regression.
- Practically speaking, the traditional coverage closure takes several days as it involves lot of manual intervention, report generation, changing constraints and rerunning regressions, But in this fully automated functional coverage closure approach we will get 100% coverage in one or two days based on the complexity of the module, LSF and Queue availability and with out manual intervention.

2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Q & A

accellera SYSTEMS INITIATIVE