2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Systematic Constraint Relaxation (SCR): Hunting for Over-Constrained Stimulus

Debarshi Chatterjee, Spandan Kachhadia, Ismet Bayraktaroglu, Siddhanth Dhodhi

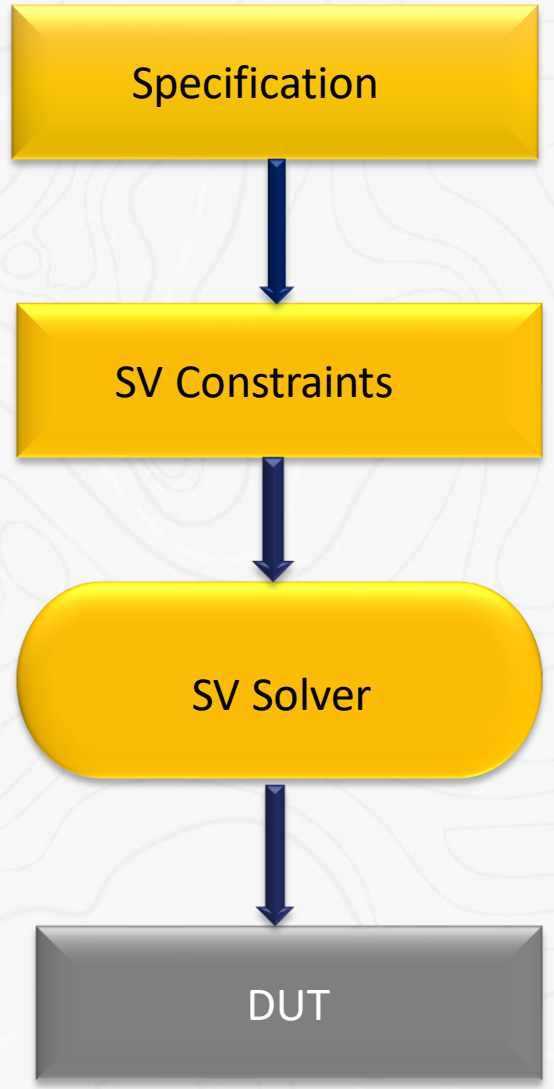
Nvidia Corporation

NVIDIA

accellera SYSTEMS INITIATIVE

Introduction

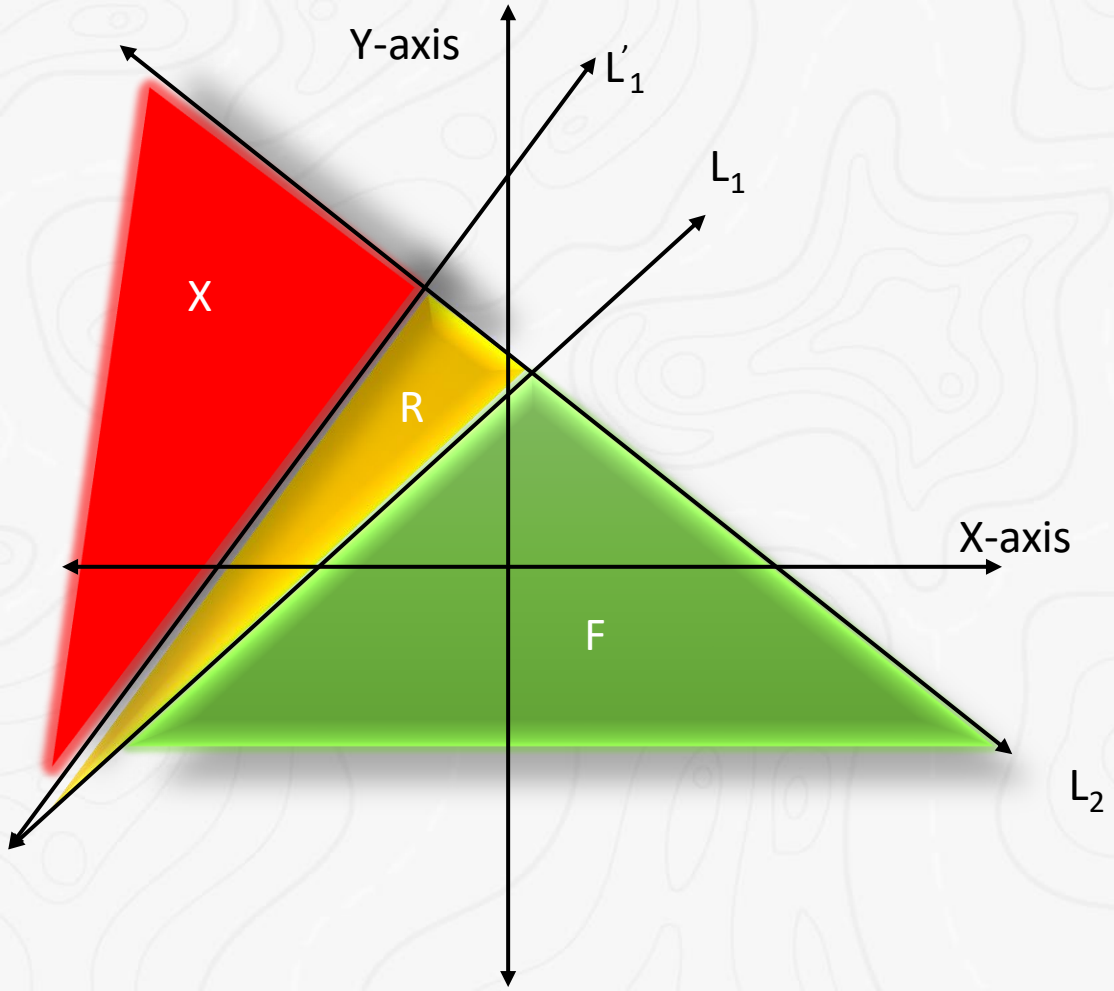
- System Verilog Constraint Solver
  - Facilitates development of random tests
  - Constraints carve out feasible space – stimulus that DUT can support
- Possible Root Causes of Overconstraints
  - Temporary Constraints added to mask out known design/checker issues
  - Incorrect Understanding of specification
  - Change in specifications not reflected in constraints
- Why over-constraints are bad?!
  - Coverage Loss and possible bug escapes
  - Can silently degrade the quality of stimulus



Prior Work

- No prior published work on automatically identifying over-constraints
- Standard Approach in Industry : Coverage Analysis
- Why Coverage Analysis is not sufficient?
  - Common mode misunderstanding in writing illegal bins
  - Does not directly point towards an over-constraint
  - Human in loop – debug effort

Simple Analysis



(Summary) Relax L_i and generate P: $P \notin F$ and S==PASS => $P \in R$ => L_i is over-constrained

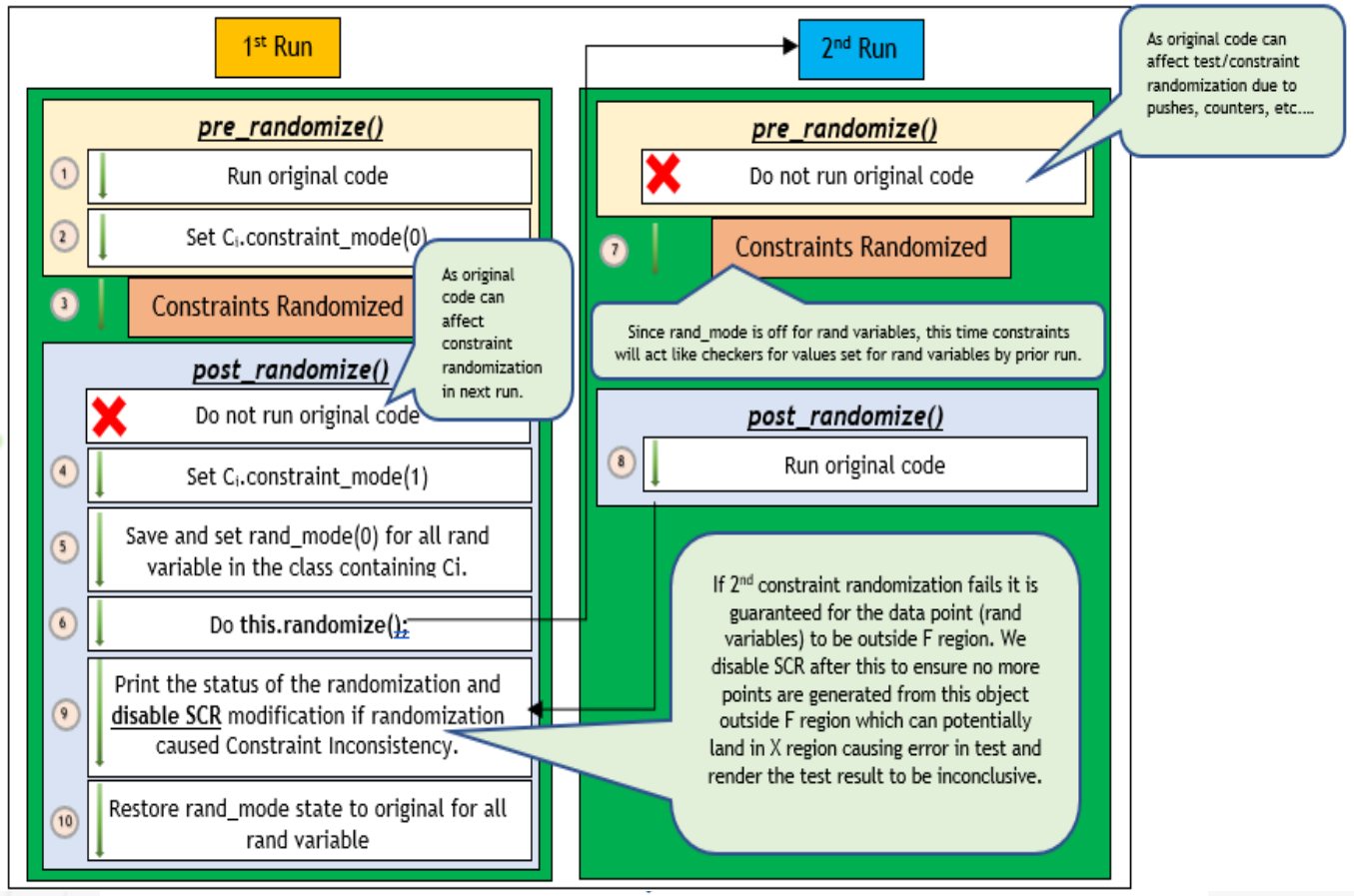
SCR - Flowchart

Search for all files containing constraints

For each file, script 1)Breaks bigger constraints blocks into single constraint per block 2)Insert custom pre-randomize and post-randomize

Rebuild and Run Tests

In each test, randomly select a constraint block C_i to be disabled



Identify SCR region for each test using the table below:

Observation		Inference		
Randomization status from step 9 (P ∉ F)	Status of test (S)	Points in F region	Points in R region	Points in X region
No	PASS	At least 1	0	0
No	FAIL	At least 1	0	0
Yes	PASS	?	At least 1	0
Yes	FAIL	?	?	?

Selected constraint condition is Over constrained.

SCR – Constraint Splitting Demo

```
1  class SCR_demo;
2    rand bit cmd_type;
3    rand bit [7:0] length;
4    rand bit [31:0] payload[$];
5    rand bit inc_payload;
6
7  //  constraint var_randomization {
8  //    if (cmd_type == 0) {
9  //      length < 64;
10 //    } else {
11 //      length < 32;
12 //    }
13 //    payload.size == length;
14 //    foreach (payload[i]) {
15 //      (inc_payload == 1 && i!=0) -> (payload[i] > payload[i-1]);
16 //    }
17 //                                  };
18
19   constraint var_randomization_0 {
20     if (cmd_type == 0) {
21       length < 64;
22     }
23   }
24   constraint var_randomization_1 {
25     if ( !( (cmd_type == 0)) ) {
26       length < 32;
27     }
28   }
29   constraint var_randomization_2 {
30     payload.size == length;
31   }
32   constraint var_randomization_3 {
33     foreach (payload[i]) {
34       (inc_payload == 1 && i!=0) -> (payload[i] > payload[i-1]);
35     }
36   }
37 endclass
```

Original constraints commented by SCR script

Modified split constraints inserted by SCR script

Results

- Ran 300 tests on Integration TB after script modified the constraints
- 22 cases of over-constraints were identified by the SCR
- 19/22 cases did not cause coverage loss
- 3/22 cases were over-constraints causing coverage loss

Example of over-constraint not leading to coverage loss

ID	Fields					
ID1	F1	F2		F3	F4	F5
ID2	F1	F6	F7		F8	F9	F5

```
Constraint c1{
  F3 inside [x, y, z];
}
```

```
Constraint c1{
  ID==ID1 -> F3 inside [x, y, z];
}
```

Conclusion

- Shows potential to quickly and automatically identify over-constraints
- Can identify redundant constraints leading to loss of simulation performance
- Limitations:
  - It is not guaranteed to find all over-constraint
  - Cannot find duplicated over-constraints