What is random stability?

Random stability is the resistance of random results to code changes

- **Every code is stable and unstable to some extent (stability is not a binary attribute)**

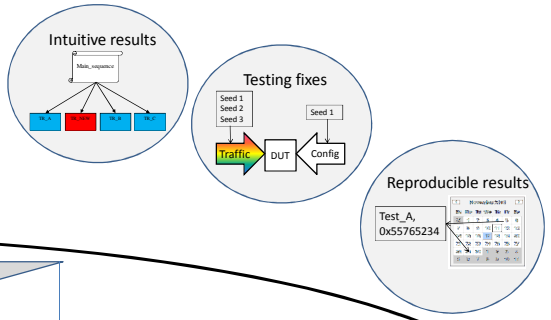
In a random-unstable testbench a wide range of code modifications, in a wide perimeter from actual randomization, could affect random results

In a random-stable testbench only a small set of code modifcations occurring in a very limited area could affect random results

Random Stability – Don't leave it to chance

Author : Avidan Efody
Presenter Rich Edelman

Why is random stability important?



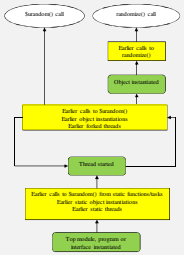
SystemVerilog Random Stability

Results always depend on execution order:

- By default, on absolute execution path
- With manual seeding, on relative execution path

Absolute path dependency means results are sensitive to numerous changes

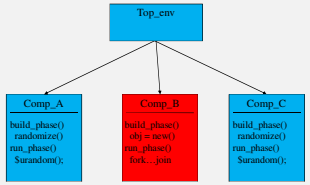
- Random stability implies manual seeding



UVM component stability requirements

Randomization in components

- Shouldn't change due to additional components



UVM component stability implementation

With UVM-1.1a components are manually seeded according to their unique full name

- After they're created
- Before each task or function phase

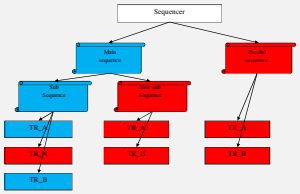
As long as a component doesn't change name

- All calls to randomize() and $urandom() return same results.

UVM sequence stability requirements

Randomization in Sequences

- Shouldn't change due to additional sequences or transactions



UVM sequence stability implementation

UVM-1.1a solution for sequences is similar to that of components

- Sequences and sequence items are seeded according to their full name

Unfortunately:

- UVM-1.1a doesn't force this name to be unique
  - Identically named sequences seeded by an additional counter
  - This makes them sensitive to new instantiations
- UVM-1.1a doesn't reseed body()
  - Therefore $urandom() calls default to dependency on full execution path
- Sequence is reseeded only at start()
  - Which takes place after randomize() according to use model

Improving UVM sequence stability

Enforce unique sequence/transaction names

- Is helpful for debugging/communication
- **Can be easily achieved using simple UVM extensions**

reseed body prior to execution

```
virtual task pre_start();
  process proc = process::self();
  proc.srandom(uvm_create_random_seed("body", get_full_name()));
endtask
```

Reseed the sequence prior to randomization

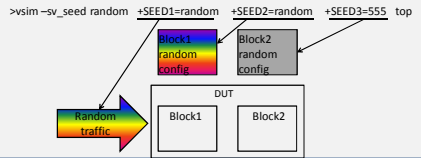
```
my_seq1 = my_seq::type_id.create("my_seq1");
my_seq1.randomize();
```

```
my_seq1 = my_seq::type_id.create("my_seq1");
my_seq1.set_item_context(this,get_sequencer());
my_seq1.randomize();
```

Making the most of random stability

Individual seeding of testbench parts

- Allows users to change random results in a specific part of a testbench, while keeping all other results intact
- Useful for stressing a specific area where a bug was found, or a fix was made



For More info on UVM random stability…

Visit…

www.verificationacademy.com

- Individual seeding package
- Unique sequence/transaction package
- Other examples

Or get a copy of the full paper…