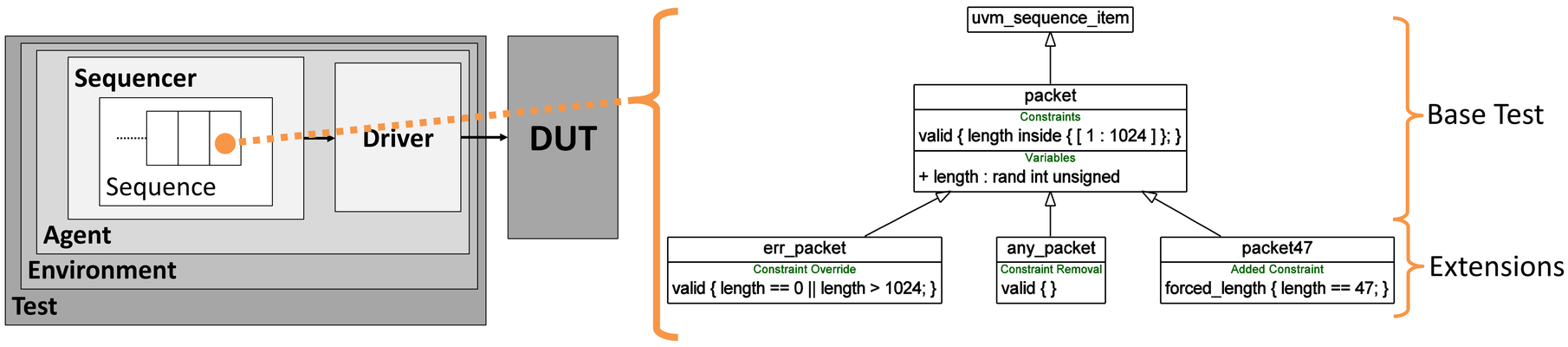
Interchangeable SystemVerilog Random Constraints

Jeremy Ridgeway
LSI Corporation, Inc. San Jose, CA

Abstract

SystemVerilog constraints are declarative. To modify or override a constraint, explicit details about the test bench must be known, the new constraint correctly implemented, and simulation recompiled. This approach is time- and knowledge-expensive. We propose a standard suite of SystemVerilog constraint building blocks that are lazily and dynamically instantiated during simulation. Coupled with a front-end parser and test bench-wide resource manager, random values incur fully interchangeable constraints specified as a string and provided on the command-line or via function call. This results in easily applied and manipulated constraints with no test inheritance or instrumentation requirement, or even recompilation.

Native SystemVerilog Constraints

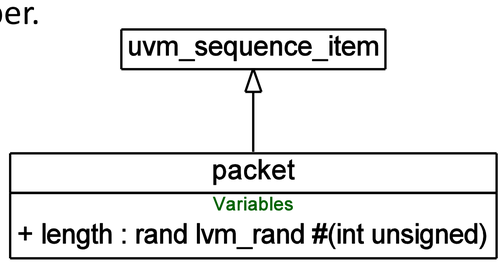


Must extend leaf class only; Need to know constraint block names;
Override in a test through UVM factory

Need an Understanding of Test Bench

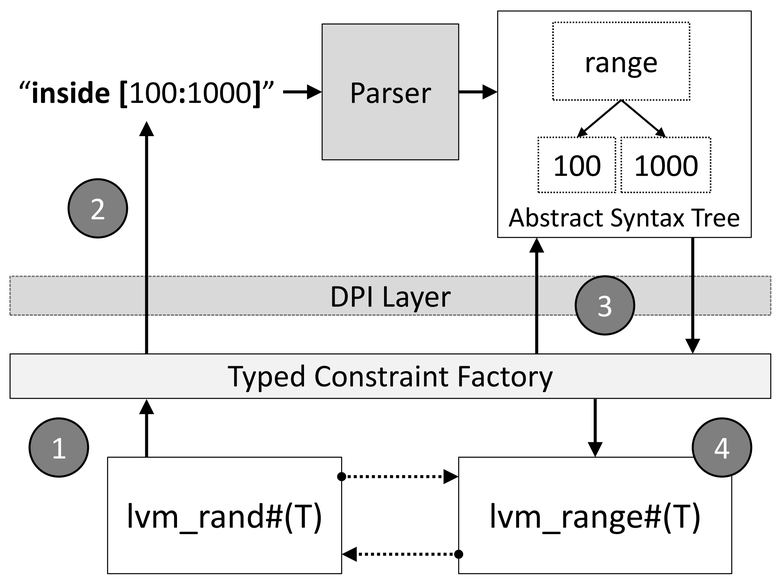
Interchangeable Constraints

LSI Verification Methodology type parameterized random class is instantiated in place of 'rand' member.



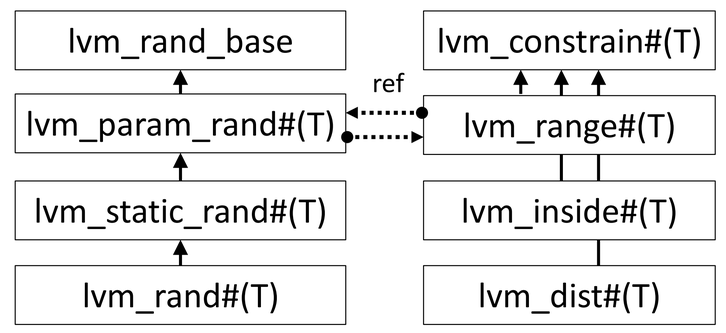
Calling lvm_rand function push() starts the sequence:

1. Call pkt.length.push("inside [100:1000]").
2. New parser instantiated to process string and generate abstract syntax tree (AST).
3. Conversion from AST to SystemVerilog constraint using generalized containers.
4. Well-formed constrained is returned as a reference to lvm_rand instance.



LVM random variables have a unique name either globally or within a UVM scope. Constraint strings may be interchanged via: *API, UVM configuration or resource database, and on the command line*.

SystemVerilog class hierarchy for supported constraints:



The constraint factory resides in lvm_param_rand#(T). The *static* LVM random class has APIs to interchange constraints:

```
class lvm_static_rand#(type T) extends lvm_param_rand#(T);
    rand T value;
    virtual function void push(string n);
    virtual function void pop();
    virtual function T next();
endclass
```

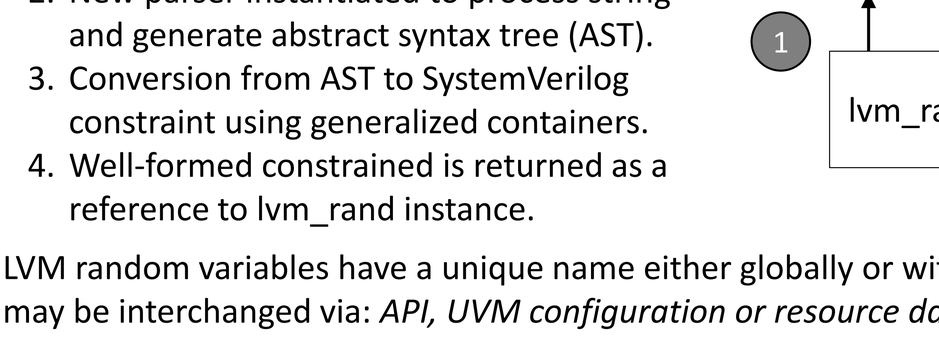
User class, lvm_rand#(T) may *dynamically* interchange constraint. Randomization is over local constraint parameters. E.g. a constant constraint:

```
loc_rand.randomize with { value == local::const_value };
```

Need Only Know Random Value Name

Experimental Results

SystemVerilog Constraints **vs.** **Interchangeable Constraints**

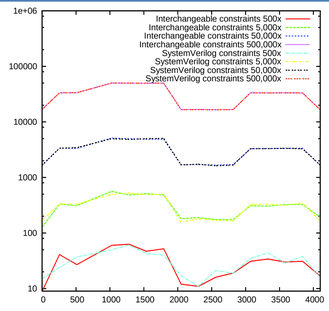


Single test bench with run-time choice between SystemVerilog or Interchangeable constraints. Test 1 constraint:

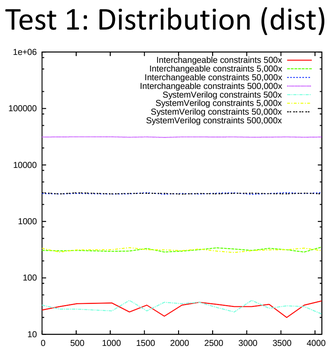
```
dist{1:=1, 256:=2, 512:=2, 1024:=3, 1280:=3, 1536:=3,
     1792:=3, 2048:=1, 2304:=1, 2560:=1, 2816:=1,
     3072:=2, 3328:=2, 3584:=2, 3840:=2, 4096:=1 }
```

Test	LVM Time	LVM Memory	Native Time	Native Memory
t1_500000	51.280 s	40 KB	22.830 s	200 KB
t1_50000	4.550 s	40 KB	2.260 s	200 KB
t1_5000	0.710 s	40 KB	0.240 s	200 KB
t1_500	0.060 s	40 KB	0.010 s	200 KB

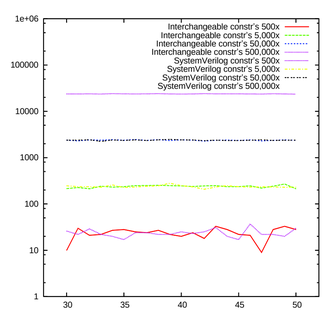
Test 1: Constraint Solver Performance



Test 1: Distribution (dist)



Test 2: Inside set



Test 3: Inside range

Test	LVM Time	LVM Memory	Native Time	Native Memory
t1_500000	1,394.8 s	4,228.1 MB	85.3 s	178.7 MB
t1_50000	130.0 s	455.6 MB	8.9 s	143.3 MB
t1_5000	12.8 s	153.0 MB	1.7 s	139.8 MB
t1_500	2.0 s	130.9 MB	1.0 s	136.4 MB

Test 1: Cumulative Simulation Performance

Our C-to-SystemVerilog number conversion was reliant on UVM library DPI regex. Overall, this cost 94.73% of PLI access time and 48.88% simulation time.

DPI Function	Percentage
dpi_regcomp	**62.47%**
dpi_refgree	**16.97%**
dpi_regexec	**11.24%**
uvm_re_match	4.05%
lvm_constraint_parse_string	2.94%
lvm_constraint_parse_elem_nextval	1.95%
other / unreported	0.38%
Total	100%

Test 1: Time Usage PLI Breakdown
ROOT CAUSE

Conclusions

Our interchangeable approach provides a straightforward way to specify constraints as a string either to a function or on the command line. We showed that the resultant distributions, in the macro, converge with native constraints. However, it does incur a significant overhead that could undermine accessibility. One bottleneck resides in the UVM library regex function that consumed nearly half of all simulation time.
There are advantages in breaking down SystemVerilog constraints that may prove significant:

- Ease of use,
- Random coercion,
- Reduced failed solving,
- Easier for designer to debug.

BNF Grammar Available