Problem: How to reuse random constraints?

SystemVerilog constraints may be added to an object via inheritance or with inline constraints specified when the object is randomized (e.g. obj.randomize with ()). But the SV language doesn’t define a way to easily reuse constraints in multiple objects.

Solution: Put constraints in “Policy Classes”

Putting the constraints in a standalone class allows them to be defined once then added into other objects as needed. Policies can be mixed and matched in any combination.

Examples

The two examples below illustrate the concept for a simple address transaction. Two policies constrain the generated addresses to lie within permitted regions and outside prohibited regions.

More details in the paper

See the paper for more applications of this idea:
- policy_list classes encapsulate a list of policies. Lists may be nested to any number of levels.
- Policies with persistent state information e.g. keep track of recently used addresses and use them in constraints for subsequent randomizations.

Conclusions

Randomization policy classes provide a flexible and efficient way to add different types of constraints into an object being randomized. This technique can be used with native SystemVerilog or can be applied to UVM.

SV Example

This example adapts the SV example to UVM. A sequence adds the same policy classes to the same address transaction which has been converted to a UVM sequence item.

UVM Example

This example uses policy classes to add additional reusable constraints to a transaction object.