Constraints are our best friends when it comes to random generation. They allow us to limit the range of possible scenarios/values as much as we like, from the level of "everything is allowed" down to the level of such heavy constraints that really imply a very directed scenario. In most cases what we want to is to have a set of default constraints that give us a nice distribution of scenarios and values. We will then need to override the default set in some of our test cases by applying additional constraints, or reshaping the distribution or even removing previous constraints. Here's a cool way to do this in SystemVerilog:

First, let's see how you can specify custom distribution in System Verilog:

{code}
```
class frame;
  ...
  rand bit valid;
  constraint user_constraint;
  constraint user_prob {
    valid dist { 1:= 90, 0:= 10}; // default distribution
    // valid dist { 1:=1000000, 0: 1} // "soft" constraint
  }
...
endclass
```
{/code}

This means that the frame will be valid 90% of the times. This very much resembles the soft constraints mechanism in ***e***. Make it a million to 1 ratio, and you''ll end up with a soft constraint "de facto". If you still get an invalid frame after that, go buy a lottery ticket because it''s probably your lucky day.

Now what if we wanted to override this default distribution in a specific test case? For example, we want to write a test case for invalid frames. No problemo. SystemVerilog kind of mimics AOP behavior in allowing you to have placeholders for constraints that you can later on define in your test case. No complaints here... Check this out:

{code}

```
program test;
  ...
  // we"ve already placed a placeholder for this
  frame::user_constraint {
    valid == 0;
  }
  ...
endprogram
```

{/code}

Voila, if you did this right you"ll get only invalid frames.

Now for the tricky part. What if you wanted to completely remove the default constraints becuase you want, for example, a 50/50 chance of valid frames in your test case? There are 3 ways to do that.

1. Change the default constraints in the environment. Hm.. Not so good. In fact - a VERY BAD idea.

2. Use SystemVerilog"s built-in mechanism for setting constraints on and off (look up constraint_mode() in SystemVerilog LRM). Possible, but not really efficient. You can only change constraints mode in a procedural way (i.e. from a function). If you want to eliminate constraints in a declarative manner (and trust me - you want) then check out number 3.

3. Add an auxiliary boolean (bit) variable that controls the default constraints. Keep it at 1 ("TRUE") using a "soft" constraint and override it in your program, if needed, to provide alternative constraints. Here"s the full example:

{code}

```
program test;

class frame;
  rand bit valid;
  rand logic[7:0] data[];
```

```
  // adding a boolean to enable/disable the default constraints
  rand bit keep_default;
  // by default (soft), default constraints are applied
  constraint keep_default_prob {
    keep_default dist {1:=1000000, 0:= 1}; // soft constraint
  }
  // here are the default constraints again, this time controlled by an auxiliary boolean
  constraint user_constraint;
  constraint default_constraints {
    keep_default -> valid dist { 1:= 90, 0:= 10};
    keep_default -> data.size() < 1518;
  }
 endclass

 frame fr;
 initial begin
  fr = new();
  fr.randomize();
  $display($psprintf("valid: %-b",fr.valid));
  $display($psprintf("data size: %-d",fr.data.size()));
 end

 constraint frame::user_constraint {
  // this will eliminate the default constraints
  keep_default == 0;
  // we can now apply any constraints we need for this test case
  valid dist { 1:= 50, 0:= 50};
  data.size() == 200;
 }
 endprogram
```

{/code}

Comments, questions or requests are welcome! Email us at mail@thinkverification.com. We love your feedback.