Overloading Modules, Paramsets, and alternative methods of implementing Spice .model cards in Verilog-AMS

1. Introduction

This document will compare three proposed methods of obtaining functionality of SPICE .model cards and some important extensions to that functionality. The functionality is best described in the document, "paramset: A Verilog-A/MS Implementation of SPICE .model Statements," linked on the page `http://www.eda.org/verilog-ams/htmlpages/compact.html`

The three methods are as follows:
1. Paramsets: this is a completely new construct in Verilog-AMS. The proposed syntax can be found in Section 7.3 of the Verilog-AMS LRM 2.2 draft e, which is also linked at the above site. The basic concepts are: Paramsets are collections of parameter values, specifically intended for compact transistor models, which have hundreds of parameters that are specific to a type of device (eg, an NMOS transistor) and shared amongst many (hundreds or thousands) instances that may have different instance-specific parameters (such as length and width). Paramsets contain no behavioral code but are associated with a module that contains the behavioral code. They can be overloaded, and there is an algorithm for picking one for a particular instance.
2. Overloading modules: this consists mainly of allowing several modules to have the same identifier, specifying a resolution method for choosing between them, and some convenience features.
3. Using generate and configurations: this method uses only existing constructs in 1364-2001 Verilog (and SystemVerilog -- though not Verilog-AMS).

Verilog-AMS already provides methods for instantiating SPICE primitives in Verilog netlists, but does not specify the resolution mechanism for primitives that require model cards. The resolution is assumed to involve some sort of SPICE library file. The compact modeling extensions should provide methods to write this library in the Verilog language.

2. Motivating Example

In a typical SPICE netlist, one has instances of transistors for which the designer specifies a model name, a length, and a width. The SPICE library contains the definition of the model, a .model statement, which can contain dozens of parameter values for a BSIM3 transistor model. The SPICE netlist would have this line:

```
     m1 d g s b nmos l=0.18u w=10u
```

and the SPICE library would have several lines of the following form:

```
     .model nmos (level=3 uo=600 tox=1e-7 vto=0.5
     + lmin=0.18u lmax=0.25u wmin=1u wmax=3u
     + cgbo=1e-15 cgdo=3e-15 cgso=3e-15)
```

How should this be translated to a Verilog-AMS netlist? The designer should not be responsible for setting all the values of the process parameters, so the instance line should be a simple translation of the instance line of the SPICE netlist:

```
    nmos #(.l(0.18u), .w(10u)) m1(.d(d), .g(g), .s(s), .b(b));
```

The identifier `nmos` will be resolved to something in an associated library file. This library file would look different under the three potential methods described above.

2.1 Example Using Paramsets

Using paramsets, the library file would have statements like the following:

```
    paramset nmos mos_level_3;
      parameter real l = 0.18u from [0.18u:0.25u);
      parameter real w = 1u from [1u:3u];
      .uo = 600; .tox=1e-7; .vto=0.5;
      .cgbo=1e-15; .cgdo=3e-15; .cgso=3e-15;
    endparamset
```

where `mos_level_3` refers to a module that contains the behavioral equations for a MOSFET. The ports specified on the instance are connected to the ports of `mos_level_3`. No hierarchy is introduced, and the output variables of the module are directly available from the instance. (Output variables are variables marked with a description attribute so that the simulator will print their values for an operating point calculation; Gm of a transistor is a common example of a variable computed by the module that is useful to the designer.) One paramset exists for each model statement in the SPICE library, and the algorithm for choosing a paramset from all those with the same name includes a rule that the parameters (l and w) must fit the specified ranges.

2.2 Example Using Overloaded Modules

Using overloaded modules, the library file would have statements like this:

```
    module nmos interface const mos_level_3;
      parameter real l = 0.18u from [0.18u:0.25u);
      parameter real w = 1u from [1u:3u];
      mos_level_3 #(.*) M(.*);
      defparam M.uo = 600, M.tox=1e-7, M.vto=0.5,
        M.cgbo=1e-15, M.cgdo=3e-15, M.cgso=3e-15;
    endmodule
```

The specification `interface` is used to denote that the module gets its parameters, ports, and output variables from the module `mos_level_3;` the specification `const` denotes that the parameters of `mos_level_3` cannot be overridden by instances of `nmos`, except for the parameters `l` and `w` that are explicitly mentioned in the definition of `nmos`. The syntax .* is already used in SystemVerilog for connecting ports of a module to nets of the instantiating module with the same name (for the nmos' ports, .* means .d(d), .g(g), .s(s), .b(b)). The syntax is applied here to pass the parameters as well (.* here means .l(l), .w(w)). One module exists for each model statement in the SPICE library, and the algorithm for choosing a module is the same as for paramsets.

2.3 Example Using Existing Syntax

Using existing syntax, the library file would have one long module definition with a generate statement:

```
      module nmos (inout d, inout g, inout s, inout b);
        parameter real l = 0.18u;
        parameter real w = 1u;
        generate
          if (l>=0.18u && l<=0.25u && w>=1u && w<=3u) begin
            mos_level_3 #(.*) M(.*);
            defparam M.uo = 600, M.tox=1e-7,M.vto=0.5,
              M.cgbo=1e-15, M.cgdo=3e-15, M.cgso=3e-15;
          end else if (...) begin
            ...
          end
        endgenerate
      endmodule
```

This module will be called the master module in the remainder of this document. The output variables must be declared explicitly. The algorithm for choosing is explicit in the **if** statement of the **generate** block. Both the overloaded module approach and the generate approach make use of the **defparam** statement, which SystemVerilog proposes to deprecate. The alternative is to specify the parameters on the instantiation

```
  mos_level_3 #(.uo(600), .tox(1e-7), .vto(0.5), .cgbo(1e-15),
    .cgdo(3e-15), .cgso(3e-15), .*) M(.*);
```

3. Corner Models

In addition to bins, another common feature in SPICE libraries is corner models: several of the parameters of devices are varied from their nominal or typical values, corresponding to variations in the manufacturing process. Usually, two sets of variations are considered: one that makes digital gates run faster than nominal, and one that makes them run slower. The circuit is checked for both conditions to make sure it still performs correctly (no timing violations, etc.).

This feature can be implemented quite efficiently using paramsets because of the ability to nest paramsets. Suppose for a particular process, the fast/slow/typical corners were achieved by only varying the threshold voltage (vto).

```
      paramset nmos_base mos_level_3;
        parameter real l = 0.18u from [0.18u:0.25u);
        parameter real w = 1u from [1u:3u];
        parameter real vto = 0.5;
        .uo = 600; .tox=1e-7; .vto=vto;
        .cgbo=1e-15; .cgdo=3e-15; .cgso=3e-15;
      endparamset
      paramset nmos nmos_base; // typical nmos
        parameter real l = 0.18u from [0.18u:0.25u);
        parameter real w = 1u from [1u:3u];
        localparam string corner = design.corner from
            {"tt","ts","tf"};
        .l=l; .w=w; .vto=0.5;
      endparamset
```

```
paramset nmos nmos_base; // fast nmos
  parameter real l = 0.18u from [0.18u:0.25u);
  parameter real w = 1u from [1u:3u];
  localparam string corner = design.corner from
      {"ft", "fs", "ff"};
  .l=l; .w=w; .vto=0.4;
endparamset
paramset nmos nmos_base; // slow nmos
  parameter real l = 0.18u from [0.18u:0.25u);
  parameter real w = 1u from [1u:3u];
  localparam string corner = design.corner from
      {"st", "ss", "sf"};
  .l=l; .w=w; .vto=0.6;
endparamset
```

Here, we assume that the corner is set by a parameter in a top-level module (in SPICE , the corner is set by the `.lib` command). The two characters specify whether the nmos and pmos devices, respectively, are slow, typical, or fast ("sf" means slow nmos, fast pmos). This approach is particularly efficient because the values of parameters that do not vary across corners are stored only once, in the base paramset.

The approach could be replicated using overloaded modules; however, it would require adding a second layer of hierarchy.

Implementation of this feature using existing SystemVerilog syntax could be done with a more complicated generate structure, but would more likely be handled with configurations, which are already present in 1364-2001 Verilog; one could bind the top-level design to use a specific library.

4. Drawbacks of the Generate Approach

There are several drawbacks to the generate approach. The two main problems are the inefficiency related to storing of instance-specific parameters and the inflexibility of the master module.

4.1 Memory Inefficiency of the Generate Approach

Consider the case of two sets of models, one for use before a layout has been generated, and one for use after. The pre-layout models would have only two parameters: length and width, and the simulator would store these values for each instance. The post-layout models, however, could have many additional parameters: area and perimeter of the drain and source regions (AD, PD, AS, PS) and the diffusion size for hydrostatic stress (SA, SB in the BSIM4 model). Thus, a post-layout model could contain an additional six parameters that the simulator would need to store for each instance.

(Slightly more explanation is required here: the pre-layout models will in fact have non-zero values for AD, PD, AS, PS, and these values will be computed using the device (gate) width. Since these "model parameters" depend on an instance parameter, one might think that the simulator must store values for them for each instance even in the pre-layout case. However, since computing them requires very few operations relative to the number of operations in the BSIM4 model code, it might actually be more efficient to recompute the values rather than storing the extra four

double-precision numbers for each instance in a million-transistor design. The stress parameters are not dependent on the gate length and width, so pre-layout models never need to store values for them per instance.)
If one uses the generate approach, all eight parameters would need to be parameters of the master module, and the simulator would need to remember values for each of them for each instance. For a pre-layout simulation, this would result in a quadrupling of memory requirements. The paramset approach would map the pre-layout netlist instances to the pre-layout models (by means of the paramset resolution algorithm that specifies selection of the paramset with the fewest un-overridden parameters).

4.2 Inflexibility of the Generate Approach

Now consider the case where a designer wants to generate a specific model to be used in special cases, for which he will specify an extra parameter. Using paramsets, he creates the following:

```
    paramset nmos mos_level_3;
      parameter real l = 0.18u from [0.18u:0.25u);
      parameter real w = 1u from [1u:3u];
      parameter integer special = 0 from [1:1];
      .uo = 600; .tox=1e-7; .vto=vto;
      .cgbo=1e-15; .cgdo=2e-15; .cgso=4e-15;
    endparamset
```

Since the default value for the parameter special is zero, this paramset will not be chosen unless the designer specifies an override of 1. (In this case, the overlap capacitances have been modified to decrease the drain capacitance.) The designer can place this paramset in his design, and he does not require write access to the library files, which are typically stored in a read-only location (and may possibly be encrypted). To obtain this special model using generates, one would need to edit the master module, possibly by copying it to the user's directory (which is hazardous: if the library files are updated on the system, the user will not get the update automatically).

4.3 No Existing Syntax for Output Variable Propagation

Output variables, such as Gm and Cgs, are very important for design. The compact modeling extensions provide for a way to declare output variables in a Verilog-AMS module. However, they will be declared for the behavioral module, and there is no existing syntax to propagate these declarations to the master module. Using existing syntax, all these output variables would have to be redeclared for the master module, and some code would be necessary to copy the value from tbe behavioral module to the master module, such that instances of the master module would have the values accessible.

5. Drawbacks of Overloading Modules

There are a few clear but minor drawbacks to using module overloading.

Current Verilog LRMs specify that modules must have unique names; however, it seems unlikely that anyone is counting on an error being generated in order that the simulation be correct. (It is possible that this error prevents designers from specifying too many include files.)

The extra level of hierarchy is unfortunate. (The corner module example requires two extra levels). Although one simulator is reported to in-line macromodules, this behavior is not standard, and we should not require a change to existing modules or simulators.

One confusion that would exist, using the `interface const` syntax, is that `const` is taken to refer to the parameters (a user may not override those parameters of a module that were obtained from the interface module), but not the ports (a user may specify connections to the ports on the instance line in a manner which looks quite similar to the manner of overriding parameters).

Despite the minor nature of the drawbacks to overloading modules, the SystemVerilog committee members who responded to the e-mail poll on the topic did not seem particularly enthusiastic about the idea. One respondent indicated that SystemVerilog was unlikely to want overloaded modules for any purpose outside compact modeling.

6. Advantages to Paramsets

There are several advantages to the paramset method of implementing .model cards. The main advantage is that the paramset is very close conceptually to the .model card in Spice. This means that most of the optimizations performed in SPICE-like simulators to minimize memory requirements of designs where transistor models share hundreds of parameters would carry over directly. This should dramatically accellerate the acceptance of Verilog-AMS as a compact modeling language. The similarity to existing .model cards would also ease the transition on the part of the model extraction community.

The other advantages of paramsets derive from the fact that they are a new construct. They can be defined not to introduce hierarchy. They can be defined to automatically propagate output variables from the underlying module (rather than requiring extra syntax). The existence of the new construct will only affect users who use compact models in their designs, and will also only require changes to simulators that intend to simulate analog (or mixed-signal) modules. The module statement (and the uniqueness of module identifiers) is unaffected for most designs.