Proposal for supporting m-factor in Verilog-AMS LRM

Preliminary 0.2

Date: **05/24/04**

1 Introduction

This document outlines a proposal to support the SPICE m-factor construct in Verilog-AMS LRM.

2 References

[1] Device Modeling Committee proposal for mfactor(multiplicity factor)

[2] Cadence Spectre documentation for mfactor

[3] Cadence AMS Documentation on how mfactors are netlisted

[4] Accellera Verilog-AMS LRM - Multiplicity factor on subciruits

3 Introduction to M-factor

M-factor refers to the instance multiplicity scaling factor used in SPICE and supported by most SPICE-like simulators. It is also known to many extraction and LVS tools.

It is used to implicitly create multiple parallel copies of a device without the need to instantiate a whole set of such devices in parallel. It helps simulation performance because leaf-level devices can scale themselves to behave as it they were multiple devices connected in parallel.

The m-factor's value is inherited from subcircuit to subcircuit down an instantiation hierarchy.

3.1 Example (from Spectre, a SPICE-like simulator)

```
I1 vdd gnd one // instantiation of the subckt one

subckt one(a,b)
   I2 a b two m=3
ends

subckt two(a,b)
   I3 a b three m=4
ends

subckt three(a,b)
```

```
      I4 a b resistor m=2
ends
```

According to [2]Cadence Spectre documentation for mfactor, on page 1:

“ Subcircuits always have an implicitly defined parameter `m`. This parameter is passed to all components in the subcircuit and each component is expected to multiply it by its own multiplicity factor. In this way, it is possible to efficiently model several copies of the subcircuit in parallel. It is an error to attempt to explicitly define `m` on a `parameters` line. Also, because `m` is only implicitly defined, it is not available for use in expressions in the subcircuit.“

In many SPICE simulators, ‘m’ is an implicit parameter of every subckt. The effective value of m-factor for every subcircuit is the product of the mfactor value(local contribution) of itself with the effective mfactor value of its parent. In the example above:

The effective value of ‘m’ in I1 is '1'.

The effective value of ‘m’ in I1.I2 is '3' (3*1).

The effective value of ‘m’ in I1.I2.I3 is ‘12’ (4*3).

Finally, the effective value of ‘m’ in the resistor is ‘24’ (2*12).

3.2 Uses of m-factor

It is extensively used in analog design and is handled correctly by schematic tools, including Cadence’s Composer product. Note that the word ‘m’ is specifically used to refer to m-factor in SPICE. Unlike SPICE, in schematics, the m-factor value can be specified as a property with any name like ‘multiplicity factor’ or ‘scaling factor’. The schematic and its netlisting program can be then used to map that property to the appropriate special name ‘m’ in SPICE or other SPICE-like languages.

The m-factor construct is not supported in the Verilog-A/Verilog-AMS “netlist” languages[4]Accellera Verilog-AMS LRM - Multiplicity factor on subciruits, on page 1 and this document describes a couple of proposals and a recommendation to provide that support.

4 Terminology/Definitions

For the purposes of describing m-factor in Verilog-AMS, it can be seen as containing

three components. They are:

1. Implicit mfactor: This refers to the default value of m-factor in every Verilog-AMS module. Its value is 1. It will be used to compute the effective mfactor value for the module in the absence of the explicit mfactor value.

2. Explicit mfactor: This refers to the local mfactor contribution for every Verilog-AMS instance that is specified explicitly as a part of the instantiation statement. When specified, it will be used(instead of the implicit mfactor value) to compute the effective mfactor value for the module.

3. Effective mfactor: This refers to the effective(inherited) mfactor value for every Verilog-AMS instance/module. If the explicit mfactor is specified for an instance, the effective mfactor for that instance/module is the product of its explicit mfactor and the effective mfactor of its parent. In the absence of the explicit mfactor for an instance, its effective mfactor is the product of its implicit mfactor and the effective mfactor of its parent.

5 Requirements of m-factor support in Verilog-AMS

1. The mfactor value needs to be propagated through the hierarchy which can be any combination of analog and digital blocks. Note that existing digital blocks will not have a mfactor but the mfactor value still needs to be propagated through these digital blocks.

2. It is not acceptable to modify any existing digital blocks to support m-factor. Appropriate mfactor values should automatically propagate through these blocks in the hierarchy.

3. At each hierarchical level, the effective mfactor value is the product of the instance's local mfactor value(which can be explicit or implicit) and its parent's effective mfactor value.

4. If the mfactor is not specified for a module(instance), its local contribution value should be considered as 1. In other words, the effective mfactor value for such a module will be the same as its parent module's effective mfactor value.

5. The mfactor value(expression) can be considered like a globally static expression i.e a constant.

6. It should be possible to refer to the effective mfactor value in analog behavioral code. Also, it is neither required nor expected to be used in digital behavioral code. Note that the syntax and semantics for analog behavioral access of mfactor is being defined by the device modelling subcommittee[1]Device Modeling Committee

proposal for mfactor(multiplicity factor), on page 1.

7. The value of mfactor will always be a numeric(i.e an integer or real) and it shall be greater than zero.

8. It will not be possible to set the mfactor value via the **defparam** statement. This is to avoid a cycle when evaluating the RHS parameter expression in the **defparam** statement. For eg, consider the following **defparam** statement in an instance, say top.foo1.foo11:

```
defparam top.m = m;
```

Here, the RHS value is the effective mfactor value in top.foo1.foo11, which already would have been computed using the module top's effective mfactor value. It is not clear whether the defparam statement will change the explicit or effective mfactor value in the LHS. In any case, changing the top module's mfactor will inturn, introduce a change in its value, which is a cycle.

6 Proposals

There are four proposals(with pros and cons) which address the requirements for using mfactor in Verilog-AMS. They specify how the mfactor value can be set explicitly or used implicitly to calculate the effective mfactor for each instance/ module. The suggested semantics(/syntax) to set the mfactor value and propagate it through the hierarchy is expected to be common for both structural and behavioral designs.

The first proposal suggests the use of attributes to propagate the values through the hierarchy using parameter association by name and order. The second and third proposals are variations of a common theme - suggesting a new keyword to specify the mfactor parameter to enable parameter association by order. The fourth proposal suggests the use of a special parameter name(instead of a keyword) for the mfactor parameter.

6.1 Two new attributes to refer to the mfactor parameter

This proposal suggests the use of two new attributes: the passed_mfactor and the inherited_mfactor .

The passed_mfactor attribute is SOLELY for PASSING THE MFACTOR DOWN A HIERARCHY.

The inherited_mfactor attribute will enable the possibility of using parameter association by order when the mfactor is passed down the hierarchy.

6.1.1 Passing the mfactor down the hierarchy

To pass the mfactor down the hierarchy, a parameter will be passed to an instance with the value of mfactor(explicit mfactor) that is desired to be passed. Also, an attribute will be added to the instance called passed_mfactor which will specify which parameter assignment is the mfactor assignment. e.g.;

```
module top;
(* passed_mfactor = “m” *) Foo #(.m(3)) F1;
endmodule

module Foo;
resistor #(.r(1k)) R1(a,b);
endmodule
```

Notes:

1. The attribute is on the instance.

2. The module being instantiated (in this case Foo) does not have to have the parameter (in this case m) declared in its interface - in fact this would be the typical situation (just like SPICE/Spectre). In the absence of this parameter declaration, Foo is considered to have a default implicit mfactor of value ‘1’.

3. If the passed_mfactor specifies a parameter that doesn’t exist on the instantiation line, then, the attribute value will be ignored. In other words, the instance will not have an explicit mfactor setting. Therefore, the implicit mfactor for that instance will be used to calculate its effective mfactor. e.g. in the above example if there is no "m" parameter assignment in the instantiation of F1.

4. Although in the examples in this document, "m" is the parameter name typically used in the passed_mfactor attribute, other parameter names can be used e.g:

```
(* passed_mfactor = “mfactor” *) Foo #(.mfactor(3)) F1;
```

5. The effective mfactor value for every module/subcircuit is calculated as specified in 3.Effective mfactor: This refers to the effective(inherited) mfactor value for every Verilog-AMS instance/module. If the explicit mfactor is specified for an instance, the effective mfactor for that instance/module is the

product of its explicit mfactor and the effective mfactor of its parent. In the absence of the explicit mfactor for an instance, its effective mfactor is the product of its implicit mfactor and the effective mfactor of its parent., on page 3

6. When netlists are generated from schematics, the netlister program[3]Cadence AMS Documentation on how mfactors are netlisted, on page 1 can typically generate the passed_mfactor attribute whenever a parameter of name ‘m’ is encountered in the instantiation statement. The schematics/netlister program can also use a special database property to indicate/identify a mfactor parameter with any name other than ‘m’.

6.1.2 Enable parameter association by order for explicit mfactor

In to specify the explicit mfactor value using the parameter association by order notation, the mfactor needs to be declared in the module. This can be enabled using a new parameter declaration that has an associated attribute called "(* inherited_mfactor *)". The parameter declared will typically be called ’m’ but any legal Verilog identifier name can used.

The default value of the mfactor parameter will always be ‘1’. It will be an error to set it to any other value in the parameter declaration statement. It will also be an error if the identifiers used for mfactor in the passed_mfactor and inherited_mfactor attributes do not match. It will be error to use the declared mfactor parameter in any expression inside the module. Eg:

```
module top;
(* passed_mfactor = “m” *) Foo #(3) F1;
endmodule

module Foo;
(* inherited_mfactor *) parameter real m = 1;

resistor #(.r(1k)) R1(a,b);
endmodule
```

The netlister program can generate the inherited_mfactor attribute whenever a parameter of name ‘m’ is encountered in the cellview being netlisted. The schematics/netlister program can also use a special database property to indicate/identify a mfactor parameter with any name other than ‘m’.

6.1.3 Pros

1. This approach allows mfactor values to be propagated through the hierarchy including those modules(like digital blocks) which do not have the inherited_mfactor declaration.

2. The name of ‘mfactor’ is not restricted to be ‘m’ and can be any identifier. Attributes are used to identify the mfactor parameter for every module/ instance.

3. The schematics and netlister program can either generate these attributes for a parameter of name ‘m’ or can use database properties to identify any parameter name other than ‘m’ as a mfactor and generate attributes for that parameter name.

4. As the mfactor parameter is not allowed in any expression within the module, the presence of the mfactor parameter declaration does not cause any issues to the digital blocks and connectivity of the design.

6.1.4 Cons

1. Attributes do not belong to the language. They are just annotations to the design. So, the attributes mechanism should not be used to add new semantics(meaning) to the language or have any implications on the existing language semantics. In this proposal, the ‘passed_mfactor’ attribute is only used to identify the mfactor parameter in the instance ‘parameter_value_ assignment’ list. It is not modifying any of the semantics of the instance ‘parameter_value_assignment’ list.

   The ‘inherited_mfactor’ attribute is used to identify the mfactor parameter at the time of its declaration. This attribute does modify the semantics of parameter declaration, in that, its default value and override values do not follow the regular semantics for parameter declaration. Instead, its final value is the effective mfactor value as defined earlier in the proposal.

2. The name of the parameter is used in ‘passed_mfactor’ attribute to identify the mfactor parameter(explicit mfactor) in the instantiation statement. This approach will be feasible only when a named parameter override is used in the instantiation statement for instances of modules which do not have a corresponding ‘inherited_mfactor’ declaration.

6.2 New keyword to enable parameter association by order

This proposal continues to recommend the use of 'passed_mfactor' to refer to the mfactor parameter in the instantiation statement. It address the issue with using 'inherited_mfactor' to modify the semantics of a parameter declaration by suggesting a new keyword '**mfactor**' and declaration to identify mfactor parameter. The new declaration can be called the '**mfactor** declaration' similar to the '**ground** declaration' which already exists in Verilog-AMS. For eg:

```
module top;
(* passed_mfactor = "m" *) Foo #(3) F1;
endmodule

module Foo;
parameter real m = 1;
mfactor m;

resistor #(.r(1k)) R1(a,b);
endmodule
```

The default value of the mfactor parameter will always be '1'. It will be an error to set it to any other value in the parameter declaration statement. It will also be an error if the identifiers used for mfactor in the passed_mfactor attribute and the mfactor declaration do not match. It will be error to use the declared mfactor parameter in any expression inside the module.

6.2.1 Pros

1. This approach allows mfactor values to be propagated through the hierarchy including those modules which do not have the inherited_mfactor declaration.

2. The name of 'mfactor' is not restricted to be 'm' and can be any identifier. The attribute 'passed_mfactor' is used to identify the mfactor parameter for every instance. A new keyword '**mfactor**' and declaration '**mfactor** declaration' is used to identify the special mfactor parameter within a module. This new declaration will define the new semantics associated with the mfactor parameter.

3. The schematics and netlister program can either generate the passed_mfactor attribute and **mfactor** keyword for a parameter of name 'm'

or can use database properties to identify any parameter name other than ‘m’ as a mfactor and generate the attribute/keyword for that parameter name.

4. As the mfactor parameter is not allowed in any expression within the module, the presence of the mfactor parameter declaration does not cause any issues to the digital blocks and/or connectivity of the design.

6.2.2 Cons

1. Adding a new keyword to the language can cause compatibility issues with existing designs. On preliminary investigations, looks like the keyword ‘**mfactor**’ is not as prevalent as the keyword ‘**m**’ itself. However, this still needs further investigation.

2. The name of the parameter is used in ‘passed_mfactor’ attribute to identify the mfactor parameter(explicit mfactor) in the instantiation statement. This approach will be feasible only when a named parameter override is used in the instantiation statement for instances of modules which do not have a corresponding ‘**mfactor** declaration’.

6.3 New keyword/Parameter type to enable parameter association by order

This proposal continues to recommend the use of ‘passed_mfactor’ to refer to the mfactor parameter in the instantiation statement. In 6.2New keyword to enable parameter association by order, on page 8, the keyword **mfactor** is used to identify the parameter ‘m’ as a special type of parameter. This proposal is a variation of the keyword usage, in that, it eliminates the new declaration ‘**mfactor** declaration’. Instead, the keyword **mfactor** is used to refer to a new type of parameter similar to the parameter types - **integer** and **real**. For eg:

```
module top;
(* passed_mfactor = “m” *) Foo #(3) F1;
endmodule

module Foo;
parameter mfactor real m = 1;

resistor #(.r(1k)) R1(a,b);
endmodule
```

The default value of the mfactor parameter will always be ‘1’. It will be an error to

set it to any other value in the parameter declaration statement. It will also be an error if the identifiers used for mfactor in the passed_mfactor attribute and **mfactor**-typed parameter declaration do not match. It will be error to use the declared mfactor parameter in any expression inside the module.

6.3.1 Pros

1. This approach allows mfactor values to be propagated through the hierarchy including those modules which do not have the inherited_mfactor declaration.

2. The name of 'mfactor' is not restricted to be 'm' and can be any identifier. The attribute 'passed_mfactor' is used to identify the mfactor parameter for every instance. A new keyword '**mfactor**' is used to identify the special type of mfactor parameter. This new type will define the new semantics associated with the mfactor parameter.

3. The schematics and netlister program can either generate the passed_mfactor attribute and **mfactor** keyword for a parameter of name 'm' or can use database properties to identify any parameter name other than 'm' as a mfactor and generate the attribute/keyword for that parameter name.

4. As the mfactor parameter is not allowed in any expression within the module, the presence of the mfactor parameter declaration does not cause any issues to the digital blocks and/or connectivity of the design

6.3.2 Cons

1. **mfactor** is not really a data type like **integer** or **real**.

2. Adding a new keyword to the language can cause compatibility issues with existing designs. On preliminary investigations, looks like the keyword '**mfactor**' is not as prevalent as the keyword '**m**' itself. However, this still needs further investigation.

3. The name of the parameter is used in 'passed_mfactor' attribute to identify the mfactor parameter(explicit mfactor) in the instantiation statement. This approach will be feasible only when a named parameter override is used in the instantiation statement for instances of modules which do not have a corresponding inherited_mfactor declaration.

6.4 New special parameter in instance parameter list

This proposal is an alternative to the use of 'passed_mfactor' attribute in the

instantiation statement to pass the explicit mfactor value. It instead uses the identifier ‘passed_mfactor’ as a special parameter name which can be used in the instance parameter list. For eg:

```
module top;
Foo #(.passed_mfactor(3)) F1;
endmodule

module Foo;
resistor #(.r(1k)) R1(a,b);
endmodule
```

This proposal can be considered in conjunction with any of the proposals:

6.2New keyword to enable parameter association by order, on page 8,

6.3New keyword/Parameter type to enable parameter association by order, on page 9,

6.4.1 Pros

1. No new keyword or attributes are required to specify the mfactor parameter. It is implicitly identified using the special identifier ‘passed_mfactor’ to specify the explicit mfactor value for that instance.

2. As the mfactor parameter is not allowed in any expression within the module, the presence of the mfactor parameter declaration does not cause any issues to the digital blocks and/or connectivity of the design

6.4.2 Cons

1. Potential for conflicts if the same parameter name is used in an existing design as a regular parameter. This needs further investigation.

2. The mfactor parameter is tied to the name ‘passed_mfactor’ in Verilog-AMS modules. No other name can become a mfactor parameter in Verilog-AMS.

3. Any existing schematics/netlister programs which associate the name ‘m’ to mfactor will have to map this name to ‘passed_mfactor’ when generating Verilog-AMS netlists.

7 Recommendation

The proposal 6.3New keyword/Parameter type to enable parameter association by order, on page 9 is not recommended as mfactor is not really a data type like integer or real.

That brings us to three possibilties:

1. The proposal 6.1Two new attributes to refer to the mfactor parameter, on page 4, uses the attribute mechanism and is already implemented by Cadence. The possible reservation about this proposal could be in the use of the “inherited_mfactor” attribute to modify the semantics of a parameter declaration. It can be perceived as using attributes to modify the language semantics which may be undesirable.

2. The proposal 6.2New keyword to enable parameter association by order, on page 8, addresses the potential issue with using “inherited_mfactor” attribute. So, a combination of this proposal with the “passed_mfactor” attribute in 6.1Two new attributes to refer to the mfactor parameter, on page 4, is the recommendation.

3. The proposal 6.4New special parameter in instance parameter list, on page 10 provides an alternative to the “passed_mfactor” attribute mechanism. In this proposal, the special identifier “passed_mfactor” is not a keyword but has a special meaning which is more of a “magic”. This could still be considered along with the proposal 6.2New keyword to enable parameter association by order, on page 8.