Discrete Analog Modeling in IEEE-1800
A Proposal Framework

Gordon Vreugdenhil, Mentor Graphics Corp.

1 Introduction

The basic intent of a *discrete analog* model is to provide an abstraction of an analog model which can be efficient simulated in an event-driven digital simulator. In particular, this means that a generalized analog equation solver must not be participating in the simulation.

This document describes a general set of requirements and basic foundations for a set of changes to IEEE 1800 (*SystemVerilog*[†]) which address the need for discrete analog modeling in the context of large designs.

The term *behavioral analog* will be used when referring to systems using any continuous domain disciplines as described in Verilog-AMS. Within this document, the term *conservative net* will be used to refer to any net with a continuous time domain discipline, not just the truly conservative domains identified in Verilog-AMS. This distinction is important so that digital nets with continuous drivers are not confused with "continuous nets" in analog terminology. The terms *analog space* and *discrete space* will be used to indicate aspects of a system that require an analog continuous time equation solver versus existing SystemVerilog discrete time digital simulation.

2 Requirements

There are two fundamental requirement areas which this proposal addresses: substitutability of models and features for computation within abstract models. In addition, the proposal incorporates the following assumptions:

- No continuous time equation modeling is permitted (no solver).
- There must be reasonable path for aligning the Accellera Verilog-AMS standard with the functionality in this proposal.
- It is *not* necessary to be strictly compatible with Accellera Verilog-AMS functionality or assumptions as long as there is a reasonable expectation that Accellera Verilog-AMS models could co-simulate with a system implementing the recommended functionality.

The assumptions here are not intended to preclude the possibility of some limited form of "efficient" equation solving in the future but does not direct address the relationships needed for that to be feasible.

[†] Since IEEE 1800-2009 has subsumed IEEE-1364 (*Verilog*) all references in this document will be to SystemVerilog even for semantics originally defined by Verilog.

2.1 Substitutability

2.1.1 Behavioral Analog

When substituting between a behavioral analog model and a discrete analog model, there will in general be requirements for mapping a single conservative net connection in the behavioral analog to a multi-valued set in the discrete analog model. This occurs due to the fact that continuous disciplines carry both a flow and potential in a behavioral analog model. Thus a requirement for discrete analog modeling is to support one of the following:

- A composite net (i.e. a digital net that can simultaneously carry multiple values)
- An automatic mapping from a conservative port to multiple digital ports

Although it is possible to consider the second option, the composite net option more closely matches user models – the "net" is fundamentally representing a physical connection that has multiple properties.

Assuming that a composite net is the preferred solution, the HDL source must then be able to describe a connection at various levels of detail including at least:

- digital logic (single 2/4 state or "strength" mapped values)
- composite digital nets
- analog conservative nets

2.1.2 Spice level models

Interconnect with Spice level models is not yet addressed in this proposal.

2.2 Computation and Resolution

Computation is expressed in fundamentally different manners in behavioral analog versus digital models. In analog behavioral models, computation is expressed declaratively by using equations. The computational effects and resolution effects of loads and drivers on a net are expressed and solved simultaneously. In digital (event driven) simulation, behavior is expressed as computation with a *separate* set of semantic rules which govern the resolution of values on nets.

The impact of the digital rules is that one must carefully determine rules for net value resolution independently and bear in mind the impact of feedback between driver computation and resolution in terms of the event and value propagation cycle.

In SystemVerilog, a net is defined to have *drivers* (1800-2009 Clause 6.6) which contribute values to the net resolution function which determines the value of the final net. The term *topologically connected net* will be used in this document to refer to such nets and includes the impact of port connectivity as well as of *net aliasing* in SystemVerilog (1800-2009 Clause 10.11). In terms of analog behavior, the resolution of all drivers generally corresponds to the equation impact of solving a system of equations for a *node* with respect to the various contributions to the node.

The main difference between the analog concept of simultaneous solutions and the digital concept of computation and resolution is in how a final value for a net is attained. In the analog space, a solver is responsible for finding a solution to the system of equations such that the system has converged with respect to the contributions even when there is feedback between a conservative net and contributions to the net. In the discrete space, if the resolved value of a net changes, the net value simply propagates back into the computations and new driving values are computed.

Due to the relationship between drivers and computation in the discrete space, it is critical to have a well-defined model for the relationship between drivers, resolution, and resolved value propagation in order for users to be able to write stable convergent models across various simulator implementations.

3 Generalized Interconnect Modeling

3.1 Generic interconnect Model

We propose that SV add a new keyword, "`interconnect`"[‡]. An `interconnect` conveys pure connectivity and is not able to describe other semantics. So while an `interconnect` could be used as a port type or possibly as a direct net declaration, no operations other than port connection or SystemVerilog alias statements would be valid on an `interconnect`. The SystemVerilog port connection rules (1800-2009 Clause 23.3.3) would be modified such that any net type would dominate an `interconnect`. The alias statement would be modified such that an `interconnect` would be permitted in an alias with any net type but the resulting net type would be the dominating net type.

The result of these rules is that a model could describe port connections without having to describe the semantics of the connection. Separating the connectivity from any behavior for the connection allows for simple substitution of fundamentally different behavior without ambiguity or having unexpected behavior "leak in" from unintended use.

Example:

```
module top;
   interconnect w;
   child   c(w);
   child2 c2(w);
endmodule

module child(wire w);
endmodule

module child2(supply0 w);
endmodule
```

[‡] Rather than introducing a new keyword, the combination "`abstract wire`" could be used to describe the intent.

In this example, there is a single resulting topological `supply0` net. It would not be legal within module `top` to have either:

```
assign w = 1;
```

or

```
initial $display(w);
```

since no behavior is permitted.

3.2 Relationship to Verilog-AMS

The proposed interconnect semantics are essentially the same as the semantics followed by an empty discipline in Verilog-AMS. Verilog-AMS says the following (Verilog-AMS 2.3.1, Clause 3.6.2.4):

> If it [a net] is referenced in behavioral code, then it must have a net type.

The inverse of that is required here – if a declaration is an `interconnect`, it must not have behavioral code.

Verilog-AMS also has the following:

> In these cases, the net shall be treated as having an empty discipline. If the net is referenced in behavioral code, then it shall be treated as having empty discipline with a domain binding of **`discrete`**, otherwise it shall be treated as having empty discipline with no domain binding.

This additional description allows Verilog-AMS to state rules regarding the overall behavior of **discrete** and **continuous** domains.

The difference in effect is subtle, but we believe that it is important. In Verilog-AMS, if one has a wire with a behavioral reference, one cannot directly determine whether the *intent* of the user was to have interconnect semantics and that the behavioral reference is incorrect. That determination cannot be made until full elaboration and discipline resolution. In our proposal, the local determination can be done directly; interconnect models simply do not allow any behavioral references at all.

It would be possible to directly adopt a subset of the discipline syntax from the Verilog-AMS standard rather than using the proposed syntax. However, given that this proposal is slightly different and also deals with composite net recommendations, we feel that having distinct syntax that maps in a direct manner to Verilog-AMS reduces potential problems for unanticipated definitional issues.

4 Composite Nets and Resolution Functions

4.1 Existing SystemVerilog Support

SystemVerilog already allows unpacked composite (array or struct) nets (1800-2009 Clause 6.7) but with the restriction that all members of an unpacked composite are (recursively) 4-state packed elements. The resolution of such nets is defined on a bit level by appealing to the fundamental built-in resolution definitions for the final net type of the bit.

The basic assumption in this definition is that bit-level connections are the only form of *semantically atomic* connection; all composite connections (packed or unpacked) are assumed to represent independent physical connections in the underlying system. This also implies that composite nets may be associated with complex selects and aggregation operations since the final topological net relationships are determined at the bit level. Although SystemVerilog allows the use of `vectored` as an advisory keyword, a vectored net still follows the same bit-level resolution semantics. The keyword is only a hint to the implementation that it *may* choose to restrict complex port associations, bit and part selects, and strength designations.

The overall bit-level semantic assumption is not valid in a discrete analog model. A particular connection may have multiple values (current and voltage for example) which represent a single physical connection. In addition, independent resolution of the distinct values is not appropriate; the resolution semantics must be described on the "connection" level, not on the level of the individual values. Finally, the resolution semantics depend on the user model for the system and restricting the resolution to predefined methods is not sufficiently flexible.

4.2 Atomic composite nets

We propose to extend the net declaration syntax by allowing the (existing) keyword *primitive* in a net declaration to denote that the net declaration is an atomic net.

```
net_declaration ::=
    net_type [ drive_strength | charge_strength ] [ vectored | scalared ]
    data_type_or_implicit [ delay3 ] list_of_net_decl_assignments ;
|   primitive wire data_type_or_implicit [ delay3 ] list_of_net_decl_assignments ;
```

The keyword *primitive* was chosen in this proposal simply to avoid introducing a new keyword such as *atomic*.

A *primitive* net declaration will relax the semantic restriction for other nets by allowing a wider range of data types. At least unpacked struct types including integral and real types will be permitted; additional types can be considered for inclusion but should not include any dynamic type (class handles, dynamic arrays, queues, and the like). A primitive net shall require an explicit data type in its declaration.

A primitive net shall not permit aggregate port associations or port associations with selects to fields within the primitive net. So a primitive net port may only be associated in the whole with another primitive net port of a matching type.

There are three aspects to expressing resolution for primitive nets: definition of a resolution function, association of a resolution function with a net declaration, and rules for dealing with conflicting resolution functions on a topological net.

4.3 Resolution Functions

A resolution function for primitive nets of type *T* shall:

- be an automatic function
- not assign to variables outside the scope of the function

- not create threads or perform non-blocking assignments
- shall have one formal of type `const ref T[]`
- shall have a return type of `T`

The intent here is that resolution functions are side-effect free – they may observe but may not change state outside the function and the result shall be only a function of observed external state and the formal argument.

No member of the formal argument can be changed by the resolution function; that is guaranteed by the *const* designation in the required formal type.

4.3.1 Resolution function calls from the simulator

The resolution function for a topological net is called by the simulator when the state of any contributor has changed. All resolution calls occur between the *active* and *inactive* scheduling regions in SystemVerilog (1800-2009 Clause 4.4). This aligns with the "1b – Explicit D2A" scheduling region in Verilog-AMS (Verilog-AMS 2.3.1, Clause 8.4.1).

If the value returned by the resolution function is different than the current value on the topological net, the net is activated in the same manner as any other net in SystemVerilog. By default, posedge/negedge sensitivities are not defined on a primitive net; an implementation is free to define default behavior. See the next section for how to define posedge/negedge transitions within a resolution function.

4.3.2 Determining driven components of a primitive net

The following system functions shall be defined:

- $current_value()
  Returns the current value of the primitive net.
- $is_resolution()
  Returns 1'b1 if the current call was initiated by the simulator during a primitive net resolution; returns 1'b0 otherwise.
- $is_active(expression)
  This system function shall take an expression that is a field select to an element of the formal argument of the function. It shall return a one-bit value. The return shall be 1'b1 if the function call has been made by the simulation engine during resolution and if the selected field has been driven. It shall return 1'b0 otherwise.
- $indicate_edge(integer_expression)
  Takes a signed integer value. Allows the resolution function to indicate whether the new value should be considered as a negative edge, no change, a non-edge change, or a positive edge. The input value mapping is as follows:
  - < 0 means negative edge
  - 0 means no change
  - 1 means non-edge change
  - >1 means positive edge

This mapping allows the resolution function to override the simulators default interpretation of a value change. An explicit non-zero value edge designation shall cause the simulator to activate the net even if the primitive net value has not changed. An explicit zero value edge designation shall cause the simulator to avoid activating the primitive net even if the value has changed.

These system functions allow a resolution function to define its behavior in terms of normal pre-defined resolution considerations – when values are considered to have "changed" and what drivers are contributing to the resolved net.

4.4 Associating resolution functions with nets

A net shall specify its resolution function by using a net resolution specification.

```
net_resolution_specification ::=
    wire ps_or_hierarchical_net_identifier use ps_or_hierachical_tf_identifier;
  | wire type ps_type_identifier use ps_or_hierachical_tf_identifier;
```

The `ps_or_hierarchical_net_identifier` shall resolve to a primitive net. The `ps_or_hierarchical_tf_identifier` shall resolve to the name of a legal resolution function for the type of the primitive net. This form allows systems to associate resolution functions separately from the net, which allows for testbench design elements and/or generate conditions to control resolution selection. It shall be illegal for the net identifier or tf identifier to fail to resolve. It shall be legal to have multiple net_resolution_specifications for a given net as long as there are no resolution function conflicts between the specifications.

A default resolution function may be associated with a type. If a topological net has no specified resolution function for any member in the topological net, the default resolution function shall be used. It shall be an error if no resolution function is associated with a primitive topological net; in other words, every topological net of a primitive type shall have at least one net_resolution_specification on a net within the topological network or shall have a default resolution function for the primitive type.

4.5 Resolution function conflicts

A topological net shall have at most one user defined resolution function, however that resolution function may be associated at multiple times with the net. It shall be an error if a multiple nets in a topological net relationship specify functions that are not exactly the same function instance.

4.6 Drivers on primitive nets

A primitive net may be driven only by a continuous assignment. The LHS of the continuous assignment may drive either the entire primitive net value or may drive only a portion of the primitive net. In order to drive the entire primitive net, normal type compatibility rules are applied.

In order to drive only some parts of a primitive net, we propose that a relaxed form of the structure assignment pattern rules be permitted. The relaxation would change the statement in 1800-2009 Clause 10.9.2 from:

> Every member shall be covered by one of these rules.

to the following:

> If the target of a structure assignment patter is not a primitive net then every member shall be covered by one of these rules. If the target is a primitive net then the simulator shall consider the covered fields as being active and the remaining fields as being inactive (see the description of the $is_active system function). Any uncovered field in a primitive net assignment shall not be modified by the assignment.

Example:

```
package mytypes;
        typedef struct { real a, b; } T;
endpackage
module top;
import mytypes::*;
        primitive wire T w;
        assign w = '{ a = 5.0 };
endmodule
```

5 Type Conversion

This proposal does not directly address type conversion, which general parallels the concept of a connect module in Verilog-AMS. There are a number of options here including port-name specific variations of configurations, automatic insertion rules, etc. The main issue to be resolved before this can be developed further is whether it is critical to allow type conversions across primitive net types to consume time or whether they can be described in a pure manner, similar to resolution functions.

5.1 Configurations

There are a number of changes that could be considered for configurations. It would be fairly simple to define rules for type mapping within regions of the design via configurations. The same is the case for net resolution specifications; there the syntax was explicitly chosen to make the relationship more obvious.

More general changes to configurations would include named re-association of ports, similar to a "port map" in VHDL in order to allow more general forms of automatic connectivity management.