Cadence wreal technology contribution to Accellera Verilog-AMS Standard

Date: 12/15/2009
- Author: Martin O'Leary
- Contributors: Abhijeet Kolpekwar

Note: chapter numbers are those of the LRM2.3.1

1.0 Changes to "3.7 Real Net Declarations"

Section 3.7 will be changed to the following - existing is in green and additions in blue;

The **wreal**, or real net data type, represents a real-valued physical connection between structural entities. A wreal net shall not store its value. A **wreal** net can be used for real-valued nets which are driven by a single driver, such as a continuous assignment. If no driver is connected to a wreal net, its value shall be zero (0.0). Unlike other digital nets which have an initial value of 'z', wreal nets shall have an initial value of zero.

wreal nets can only be connected to compatible interconnect and other wreal or real expressions. They cannot be connected to any other wires, although connection to explicitly declared 64-bit wires can be done via

system tasks $realtobits and $bitstoreal. Compatible interconnect are nets of type **wire**, **tri**, and **wreal** where the IEEE std 1364-2005 Verilog HDL net resolution is extended for **wreal**. When the two nets connected
by a port are of net type **wreal** and **wire**/**tri**, the resulting single net will be assigned as **wreal**. Connection to other net types will result in an error.
Syntax 3-8 shows the syntax for declaring digital nets.

net_declaration ::= *// from A.2.1.3*
...
| **wreal** [discipline_identifier] [range] list_of_net_identifiers **;**
| **wreal** [discipline_identifier] [range] list_of_net_decl_assignments **;**

Syntax 3-8óSyntax for declaring digital nets

Examples:

```
module foo(in, out);
input in;
output out;
wreal in;
electrical out;
analog begin
V(out) <+ in;
end
endmodule

module top();
real stim;
electrical load;
wreal wrstim;
assign wrstim = stim;
foo f1(wrstim, load);
always begin
#1 stim = stim + 0.1;

end
```

Wreal nets can have disciplines - from a discipline resolution point of view, wreals are a type of wire. Their discipline can be assigned using the same mechanisms that wires are assigned disciplines;

- The discipline can be specified directly. For example;

  wreal w;

  ddiscrete w;

- The discipline and domain resolution algorithms will work on wreals in the same manner as on regular wires. A consequence of this is that if a wreal net is given a continuous discipline, it will become a net whose value is computed by the analog solver.
- Using the ‘default_discipline directive with the wreal qualifier

  ‘default_discipline wreal ddiscrete

Multiple wreal drivers are allowed to be connected to the same wreal signal. In this case the signal resolution will be done by the resolution function associated with the disciplines of the connected wreals nets. Only one of the net’s discipline needs to have a resolution function defined. If more than one net’s discipline has a resolution function defined, then it must be the same resolution function. The predefined resolution functions are given in the following table;

resolution func	**description**
DEFAULT	Single active driver only, support for Z state
4STATE	Similar to verilog 4-State resolution for digital nets
SUM	Resolves to a summation of all the driver values
AVG	Resolves to the average of all the driver values
MIN	Resolves to the least value of all the driver values
MAX	Resolves to the largest value of all the driver values

The resolution function of a discipline is set by setting realresolve attribute on a discipline. For example;

```
discipline  wrealDiscipline;
     domain discrete;

      realresolve sum;

enddiscipline
```

There is also a compiler directive, ‘default_realresolution which will set the real resolution function globally. The setting of the real resolution function by the discipline attribute mechanism will take priority however.

The following table is an example of how the resolved value (R) of two wreal drivers (D1, D2) is calculated differently depending of the resolution functions used and the values of D1 and D2.

TABLE 1. Examples of resolved value(R) using different drive values (D1, D2) and different resolution functions

D1	D2	R (Default)	R (4STATE)	R (SUM)	R (AVG)	R (MIN)	R (MAX)
x	x	x	x	x	x	x	x
x	z	x	x	x	x	x	x
x	1.1	x	x	x	x	x	x
z	z	z	z	z	z	z	z
z	1.1	1.1	1.1	1.1	1.1	1.1	1.1
2.2	1.1	x	x	3.3	1.65	1.1	2.2
1.1	1.1	x	1.1	2.2	1.1	1.1	1.1

wreal nets with different disciplines and multiple wreal drivers can be connected together must have consistent resolution function. This means that if the disciplines have t

To model X and Z states for wreals and reals, there are two predefined macros, ‘wrealZState and ‘wrealXState - they represent high impedance state and unknown state respectively. Here is an example of their usage;

```
module buf1(in, en, out);
   input in, en;  output out;
   wreal in, out;
   real outval;
   always @(in,en)  begin
      if (!en)  outval=`wrealZState;
      else if (in === `wrealXState)  outval=0;
      else outval=in;
   end
   assign out=outval;

endmodule;
```

When a wreal net is directly connected to an continuous net, an connect module will be inserted to covert the signal value between the wreal net and the continuous net. Further details are given in section **7.8.6.**

2.0 Changes to “5.10.3 Monitored Events”

Add a new section called “5.10.3.4 absdelta function” with the following content;

According to criteria you set, the simulator can generate an absdelta event when an analog signal changes more than a specified amount, a capability that is typically used to dis-

cretize analog signals. Use the `absdelta` function to specify when the simulator generates an absdelta event.

This function is only allowed in an initial or always block of a Verilog-AMS module. Its definition is as follows;

absdelta_function ::=
absdelta (*expr*, *delta* [, *time_tol* [, *expr_tol*]] **)**

expr is an analog signal expression. *delta* is a real expression specifying an amount of change in the value of *expr*. The simulator generates an event when the *expr* value changes more than *delta* plus or minus *expr_tol*, relative to the *expr* value at the previous event time. *time_tol* is a real expression specifying a time increment after the previous time point. When the current time is within *time_tol* of the previous event time, no event is generated. If *time_tol* is not specified, the default value is the time precision of the digital timescale applied to the module. A specified *time_tol* that is smaller than the time precision is ignored and the time precision is used instead. *expr_tol* is a real expression, which is the largest difference in *expr* that you consider negligible. If you do not specify `expr_tol`, the simulator uses the absolute voltage tolerance (*vabstol*) of the analog solver.

The `absdelta` function generates events for the following times and conditions.

- At time zero.
- At the time when the analog solver finds a stable solution during initialization.
- When the *expr* value changes more than *delta* plus or minus *expr_tol*, relative to the previous absdelta event (but not when the current time is within *time_tol* of the previous absdelta event).
- When *expr* changes direction (but not when the amount of the change is less than *expr_tol*.

The following module describes an event-driven electrical to wreal conversion where the `absdelta` function is used to determine when the electrical input signal is converted to a wreal output signal.

```
‘include "disciplines.vams"

‘timescale 1ns / 100ps


module electrical_to_wreal (e_in, r_out);

input e_in;
```

```
output r_out;

electrical e_in;

wreal r_out;

parameter real vdelta=0.1 from (0:inf); // voltage delta

parameter real ttol=1n from (0:1m]; // time tolerance

parameter real vtol=0.01 from (0:inf); // voltage tolerance


real sampled;

assign r_out = sampled;


always @(absdelta(V(e_in), vdelta, ttol, vtol))

   sampled = V(e_in);


endmodule
```

3.0 Changes to “6.5.3 Real values ports”

The last sentence is the follow paragraph should be struck out;

Verilog-AMS HDL supports ports which are declared to be real-valued and have a discrete-time discipline. This is done using the net type **`wreal`** (defined in 3.7). ~~There can be a maximum of one driver of a real-valued~~

.....

4.0 Changes to “7.8 Automatic insertion of connect modules”

Add a section called “7.8.6 connect module selection and insertion at wreal nets” with the following content;

You can connect `wreal` nets to `electrical` nets, with either one on top, provided that appropriate connect modules are available to translate between signal types.

For example, you can specify the following connection (`wreal` connected to `electrical`) when you have appropriate connect rules and modules available.

```
module top;
wreal w;
child c1(w); // wreal connected to electrical
endmodule

module child(e);
input e;
electrical e;
endmodule
```

When the `wreal` and `electrical` nets are connected, the port can be `input`, `output`, or `inout`, Verilog-AMS HDL uses the signal type (`wreal` or `electrical`) to identify appropriate connectrules. For example, if `myrule` is defined as

```
connectrules myrule;
   connect a2d input electrical output ddiscrete;
   connect wreal_a2d input electrical output ddiscrete;
endconnectrules;
```

The domains and directions are insufficient to distinguish between the `a2d` rule and the `wreal_a2d` rule. If you add the information that the discrete domain signal is actually `wreal`, the Verilog-AMS can select the connect module that has a `wreal` port of the appropriate direction.

Connect modules can be create which have appropriate port types and directions for use in connecting `wreal` nets to `continuous nets`. These can be combined in a single set of connect rules. For example,

```
connectrules ConnRules;
  connect L2E;
  connect E2L;
  connect E2R;
  connect R2E;
endconnectules
```

5.0 Changes to “10.1 Overview”

Add the following to the list of compiler directives;

- ‘wrealZState
- ‘wrealXState

6.0 Changes to “10.5 Predefined Macros”

Add the following;

Verilog-AMS HDL also supports ‘wrealZState and ‘wrealXState for use in wreal modeling. See section 3.7 for more details.

7.0 Changes to “Annex B List of Keywords”

Add ‘absdelta’ as a keyword.

8.0 Changes to “Annex G Change History”

This section should be updated to reflect the wreal changes and additions outlined in this document.