Section 5 Signals

Issue 96 is about supporting attributes for disciplines --Martin

5.1 Analog signals

Analog signals are distinguished from digital signals in that an *analog signal* has a discipline with a continuous domain. Disciplines, nets, nodes, and branches are described in Section 3 and ports are described in Section 7.

This section describes analog branch assignments, signal access mechanisms, and operators in Verilog-AMS HDL.

5.1.1 Access functions

Flows and potentials on nets, ports, and branches are accessed using *access functions*. The name of the access function is taken from the discipline of the net, port, or branch associated with the signal.

Examples:

Example 1

For example, consider a named electrical branch *b* where *electrical* is a discipline with *V* as the access function for the potential and *I* as the access function for the flow. The potential (voltage) is accessed via `V(b)` and the flow (current) is accessed via `I(b)`.

- There can be any number of named branches between any two signals.
- Unnamed branches are accessed in a similar manner, except the access functions are applied to net names or port names rather than branch names.

Example 2

If *n1* and *n2* are electrical nets or ports, then `V(n1, n2)` creates an unnamed branch from *n1* to *n2* (if it does not already exist) and then accesses the branch potential (or the potential difference between *n1* to *n2),* and `V(n1)` does the same from *n1* to the global reference node (*ground*).

- In other words, accessing the potential from a net or port to a net or port defines an unnamed branch. Accessing the potential on a single net or port defines an unnamed branch from that net or port to the global reference node (*ground*). There can only be one unnamed branch between any two nets.

- An analogous access method is used for flows.

Example 3

`I(n1, n2)` creates an unnamed branch from *n1* to *n2* (if it does not already exist) and then accesses the branch flow, and `I(n1)` does the same from *n1* to the global reference node (*ground*).

- Thus, accessing the flow from a net or port to a net or port defines an unnamed branch. Accessing the potential on a single net or port defines an unnamed branch from that net or port to the global reference node (*ground*).
- It is also possible to access the flow passing through a port into a module. The name of the access function is derived from the flow nature of the discipline of the port. In this case, `(<>)` is used to delimit the port name rather than `()`.

Example 4

`I(<p1>)` is used to access the current flow into the module through the electrical port `p1`. This capability is discussed further in 5.1.4.

5.1.2 Probes and sources

An analog component can be represented using a network of probes and controlled sources. The Verilog-AMS HDL uses the concept of *probes* and *sources* as a means of unambiguously representing a network. The mapping between these representations are defined in following subsections.

5.1.2.1 Probes

If no value is specified for either the potential or the flow, the branch is a *probe*. If the flow of the branch is used in an expression anywhere in the module, the branch is a *flow probe*, otherwise the branch is a *potential probe*. Using both the potential and the flow of a probe branch is illegal. The models for probe branches are shown in Figure 5-1.

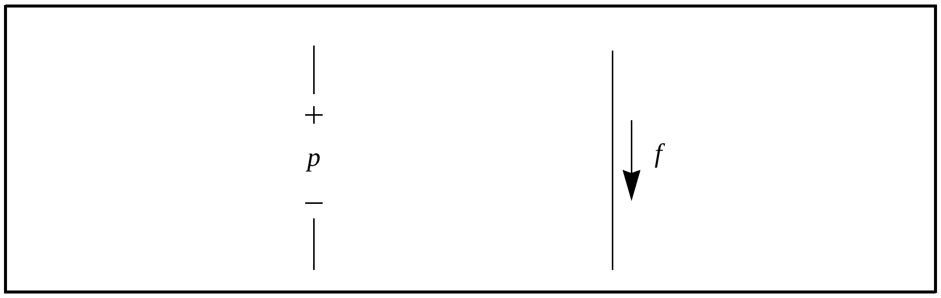


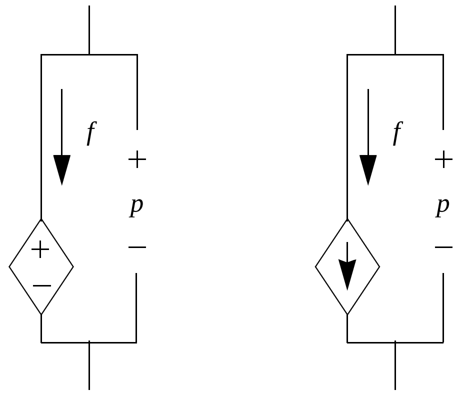
Figure 5-1 Equivalent circuit models for probe branches

The branch potential of a flow probe is zero (`0`). The branch flow of a potential probe is zero (`0`).

5.1.2.2 Sources

A branch, either named or unnamed, is a *source branch* if either the potential or the flow of that branch is assigned a value by a contribution statement (see 5.3) anywhere in the module. It is a *potential source* if the branch potential is specified and is a *flow source* if the branch flow is specified. A branch cannot simultaneously be both a potential and a flow source, although it can switch between them (a *switch branch*).

Both the potential and the flow of a source branch are accessible in expressions anywhere in the module. The models for potential and flow sources are shown in Figure 5-2.



f is a probe which measures the flow through the branch and *p* is a probe which measures the potential across the branch.

Figure 5-2 Equivalent circuit models for source branches

5.1.3 Examples

The following examples demonstrate how to formulate models and the correspondence between the behavioral description and the equivalent probe/source model.

5.1.3.1 The four controlled sources

The following example is used with each of the four behavioral statements listed below. Each statement creates a unique controlled source when inserted into this example.

```
module control_source (p, n, ps, ns);
electrical p, n, ps, ns;
parameter A=1;
branch (ps,ns) in;
branch (p,n) out;
  analog begin
// add behavioral statement here
  end
endmodule
```

The model for a voltage controlled voltage source is

```
V(out) <+ A * V(in);
```

The model for a voltage controlled current source is

```
I(out) <+ A * V(in);
```

The model for a current controlled voltage source is

```
V(out) <+ A * I(in);
```

The model for a current controlled current source is

```
I(out) <+ A * I(in);
```

5.1.3.2 Resistor and conductor

Figure 5-3 shows the model for a linear conductor.

```
module my_conductor(p,n);
    parameter real G=1;
    electrical p,n;
    branch (p,n) cond;

    analog begin
        I(cond) <+ G * V(cond);
    end
endmodule
```

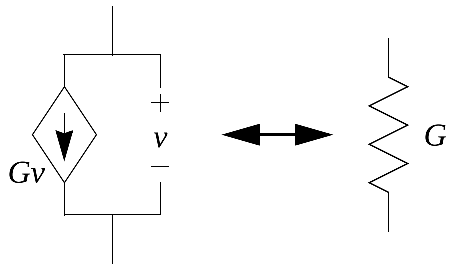


Figure 5-3 Linear conductor model

The assignment to `I(cond)` makes `cond` a current source branch and `V(cond)` simply accesses the potential probe built into the current source branch.

Figure 5-4 shows the model for a linear resistor.

```
module my_resistor(p,n);
    parameter real R=1;
    electrical p,n;
    branch (p,n) res;

    analog begin
        V(res) <+ R * I(res);
    end
endmodule
```

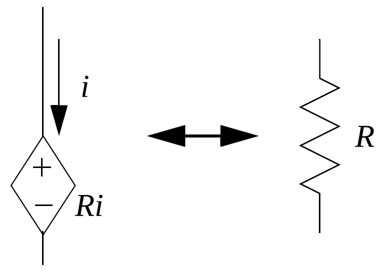


Figure 5-4 Linear resistor model

The assignment to `V(res)` makes `res` a potential source branch and `I(res)` simply accesses the optional flow probe built into the potential source branch.

5.1.3.3 RLC circuits

A series RLC circuit is formulated by summing the voltage across its three components,

$$v(t) \;=\; Ri(t) + L\frac{d}{dt}i(t) + \frac{1}{C}\int_{-\infty}^{t} i(\tau)d\tau$$

which can be defined as

```
V(p, n) <+ R*I(p, n) + L*ddt(I(p, n)) + idt(I(p, n))/C;
```

A parallel RLC circuit is formulated by summing the currents through its three components,

$$i(t) \;=\; \frac{v(t)}{R} + C\frac{d}{dt}v(t) + \frac{1}{L}\int_{-\infty}^{t} v(\tau)d\tau$$

which can be defined as

```
I(p, n) <+ V(p, n)/R + C*ddt(V(p, n)) + idt(V(p, n))/L;
```

5.1.3.4 Simple implicit diode

Verilog-AMS HDL allows components to be described with implicit equations.

Example:

In the following example, which is a simple diode with a series resistor, the model is implicit because the diode current `I(a,c)` appears on both sides of the contribution operator. The current of the diode branch is specified, making it a flow source branch. In addition, both the voltage and current of diode branch is used in the behavioral description.

```
I(a, c) <+ is*(limexp((V(a, c) – rs*I(a, c))/$vt) – 1);
```

5.1.4 Port branches

The port access function accesses the flow into a port of a module. The name of the access function is derived from the flow nature of the discipline of the port. However `(<>)` is used to delimit the port name, e.g., `I(<a>)` accesses the current through module port *a*.

Examples:

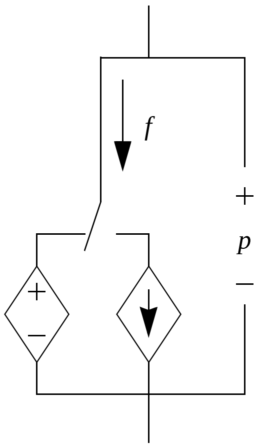
Consider rewriting the *junction diode* so the total diode current is monitored and a message is issued if it exceeds a given value.

```
module diode (a, c);
electrical a, c;
branch (a, c) i_diode, junc_cap;
parameter real is = 1e-14, tf = 0, cjo = 0, imax = 1, phi = 0.7 ;

analog begin
   I(i_diode) <+ is*(limexp(V(i_diode)/$vt) – 1);
   I(junc_cap) <+
      ddt(tf*I(i_diode) - 2*cjo*sqrt(phi*(phi*V(junc_cap))));

   if (I(<a>) > imax)
      $strobe( "Warning: diode is melting!" );
end
endmodule
```

The expression `V(<a>)` is invalid for ports and nets, where `V` is a potential access function. The port branch `I(<a>)` can not be used on the left side of a contribution operator `<+`.

5.1.5 Switch branches

Source branches have the ability to switch between being potential and flow sources. To switch a branch to being a potential source, assign to its potential. To switch a branch to being a flow source, assign to its flow. This type of branch is useful when modeling ideal switches and mechanical stops. The full circuit model for a switched branch is shown in Figure 5-5.



Position of the switch depends on whether a potential or flow is assigned to the branch.

Figure 5-5 Circuit model for a switched source branch

Examples:

An ideal relay (a controlled switch) can be implemented as

```
module relay (p, n, ps, ns);
electrical p, n, ps, ns;
parameter vth=0.5;
integer closed;
analog begin
   closed = (V(ps,ns) >vth ? 1 : 0);

   if (closed)
      V(p,n) <+ 0;
   else
      I(p,n) <+ 0;
end
endmodule
```

A discontinuity of order zero (`0`) is assumed to occur when the branch switches and so it is not necessary to use the **$discontinuity** function with switch branches.

5.1.6 Unassigned sources

If a value is not assigned to a branch, the branch flow is set to zero (`0`).

Examples:

```
if (closed)
   V(p,n) <+ 0;
```

is equivalent to

```
if (closed)
   V(p,n) <+ 0;
else
   I(p,n) <+ 0;
```

5.2 Signal access for vector branches

Verilog-AMS HDL allows ports, nets, and branches to be arranged as vectors, however, the access functions can only be applied to scalars or individual elements of a vector. The scalar element of a vector is selected with an index, e.g., `V(in[1])` accesses the voltage `in[1]`.

The index must be a genvar expression which is an expression consisting of literals and/ or genvar variables. Genvar variables can only be assigned to as the iteration index of for loops; they allow signal access within looping constructs.

Examples:

The following examples illustrate applications of access functions to elements of a an analog signal vector or bus. In the `N-bit DAC` example, the analog vector `in` is accessed within an `analog for` loop containing the genvar variable `i`. In the following `fixed-width DAC` example, literal values are used to access elements of the bus directly.

```
//
// N-bit DAC example.
//

module dac(out, in, clk);
parameter integer width = 8 from [2:24];
   parameter real fullscale = 1.0, vth = 2.5, td = 1n, tt = 1n;
   output out;
   input [0:width-1] in;
   input clk;
   electrical out;
   electrical [0:width-1] in;
   electrical clk;

   real aout;
   genvar i;

   analog begin
      @(cross(V(clk) - vth, +1)) begin
         aout = 0;
         for (i = width - 1; i >= 0; i = i - 1) begin
            if (V(in[i]) > vth) begin
```

```
                aout = aout + fullscale/pow(2, width - i);
            end
            end
        end
        V(out) <+ transition(aout, td, tt);
    end
endmodule

//
// 8-bit fixed-width DAC example.
//

module dac8(out, in, clk);
    parameter real fullscale = 1.0, vth = 2.5, td = 1n, tt = 1n;
    output out;
    input [0:7] in;
    input clk;
    electrical out;
    electrical [0:7] in;
    electrical clk;

    real aout;

    analog begin
        @(cross(V(clk) - 2.5, +1)) begin
            aout = 0;
            aout = aout + ((V(in[7]) > vth) ? fullscale/2.0 : 0.0);
            aout = aout + ((V(in[6]) > vth) ? fullscale/4.0 : 0.0);
            aout = aout + ((V(in[5]) > vth) ? fullscale/8.0 : 0.0);
            aout = aout + ((V(in[4]) > vth) ? fullscale/16.0 : 0.0);
            aout = aout + ((V(in[3]) > vth) ? fullscale/32.0 : 0.0);
            aout = aout + ((V(in[2]) > vth) ? fullscale/64.0 : 0.0);
            aout = aout + ((V(in[1]) > vth) ? fullscale/128.0 : 0.0);
            aout = aout + ((V(in[0]) > vth) ? fullscale/256.0 : 0.0);
        end

        V(out) <+ transition(aout, td, tt);
    end

endmodule
```

The syntax for analog signal access is shown in Syntax 5-1.

```
access_function_reference ::=
      bvalue
    | pvalue

bvalue ::=
    access_identifier ( analog_signal_list )

analog_signal_list ::=
      branch_identifier
    | array_branch_identifier [ genvar_expression ]
    | net_or_port_scalar_expression
    | net_or_port_scalar_expression , net_or_port_scalar_expression

net_or_port_scalar_expression ::=
      net_or_port_identifier
    | array_net_or_port_identifier [ genvar_expression ]
    | vector_net_or_port_identifier [ genvar_expression ]

pvalue ::=
    flow_access_identifier (< port_scalar_expression >)

port_scalar_expression ::=
      port_identifier
|     array_port_identifier [ genvar_expression ]
|     vector_port_identifier [ genvar_expression ]
```

Syntax 5-1—Syntax for scalar selection of vector signals

5.2.1 Accessing net and branch signals

Signals on nets and branches can be accessed only by the access functions of the discipline associated with them. The name of the net or the branch shall be specified as the argument to the access function.

Examples:

```
electrical out, in ;// as defined in 3.4.2.1
parameter real gm = 1 ;

analog
   I(out) <+ gm*V(in) ;

electrical p, n;
branch (p,n) res;
parameter real R = 50;

analog
   V(res) <+ R*I(res);
```

5.2.2 Accessing attributes

Attributes are attached to the nature of a potential or flow and can also be attached to the disciplines. Therefore, the attributes for a net or a branch can be accessed by using the hierarchical referencing operator (`.`) to the potential or flow for the net or branch.

Examples:

```
electrical a, b, n1, n2;
branch (n1, n2) cap ;
parameter real c= 1p;

analog begin
   I(a,b) <+ c*ddt(V(a,b), a.potential.abstol);
   I(cap) <+ c*ddt(V(cap), n1.potential.abstol) ;
end
```

The syntax for referencing access attributes is shown in Syntax 5-2.

```
attribute_reference ::=
    net_identifier . pot_or_flow . attribute_identifier

    | net_identifier.attribute_identifier
```

Syntax 5-2—Syntax for referencing attributes of a net

5.3 Contribution statements

Verilog-AMS HDL uses the *branch contribution operator* <+ to describe analog behavior. This operator is only valid within an *analog block*. Branch contribution statements are statements which use the branch contribution operators to describe behavior in terms of a mathematical mapping of input signals to output signals.

5.3.1 Branch contribution statements

In general, a branch contribution statement consists of two parts, a left-hand side and a right-hand side, separated by a branch contribution operator. The right-hand side can be any expression which evaluates to or can be promoted to a real value. The left-hand side specifies the source branch signal where the right-hand side shall be assigned. It shall consist of a signal access function applied to a branch.

Analog behaviors can be described using:

```
V(n1, n2) <+ expression ;
```

or

```
I(n1, n2) <+ expression ;
```

where `(n1,n2)` represents an unnamed source branch and `V(n1,n2)` refers to the potential on the branch, while `I(n1,n2)` refers to the flow through the branch. The expression can be linear, nonlinear, or dynamic in nature. The left-hand side can not use a port access function.

Examples:

The following modules model a resistor and a capacitor.

```
module resistor(p, n);
   electrical p, n;
   parameter real r = 0;

   analog
      V(p,n) <+ r*I(p, n);

endmodule

module capacitor(p, n);
   electrical p, n;
   parameter real c = 0;

analog
      I(p,n) <+ c*ddt(V(p, n));

endmodule
```

5.3.1.1 Relations

Branch contribution statements implicitly define source branch relations. The branch is directed from the first net of the access function to the second net. If the second net is not specified, the global reference node (*ground*) is used as the reference net.

A branch relation is a path of the flow between two nets in a module. Each net has two quantities associated with it—the potential of the net and the flow out of the net. In electrical circuits, the potential of a net is its voltage, whereas the flow out of the net is its current. Similarly, each branch has two quantities associated with it—the potential across the branch and the flow through the branch.

Examples:

The following module models a simple single-ended amplifier.

```
module amp(out, in);

   input in;
   output out;
   electrical out, in;
parameter real Gain = 1;

   analog
      V(out) <+ Gain*V(in);

endmodule
```

5.3.1.2 Evaluation

A statement is evaluated as follows for source branch contributions:

1. The simulator evaluates the right-hand side.
2. The simulator adds the value of the right-hand side to any previously retained value of the branch for later assignment to the branch. If there are no previously retained values, the value of the right-hand side itself is retained.
3. At the end of the simulation cycle, the simulator assigns the retained value to the source branch.

Parasitics are added to the amplifier shown in 5.3.1.1 by simply adding additional contribution statements to model the input admittance and output impedance.

Examples:

```
module amp(out, in);
   input in;
   output out;
   electrical out, in;
   parameter real Gain = 1, Rin = 1, Cin = 1, Rout = 1, Lout = 1;

   analog begin
      // gain of amplifier
      V(out) <+ Gain*V(in);

      // model input admittance
      I(in) <+ V(in)/Rin;
      I(in) <+ Cin*ddt(V(in));

      // model output impedance
      V(out) <+ Rout*I(out);
      V(out) <+ Lout*ddt(I(out));
   end

endmodule
```

5.3.1.3 Value retention

Contributing a flow to a branch which already has a value retained for the potential results in the potential being discarded and the branch being converted to a flow source. Conversely, contributing a potential to a branch which already has a value retained for the flow results in the flow being discarded and the branch being converted into a potential source. This is used to model switches. It is illegal to contribute to an external switch branch from within an analog block.

Examples:

```
module switch(p, n, cp, cn);
electrical p, n, cp, cn;
   parameter real thresh = 0;
```

```
    analog  begin
       // stop to resolve threshold crossings
       @(cross(V(cp,cn) - thresh, 0));

       if (V(cp,cn) > thresh)
          V(p,n) <+ 0;
       else
          I(p,n) <+ 0;
          end

endmodule
```

The syntax for source contribution statement is shown in Syntax 5-3.

```
analog_branch_contribution ::=
    bvalue <+ analog_expression ;
```

Syntax 5-3—Syntax for branch contribution

5.3.2 Indirect branch assignments

Verilog-AMS HDL allows descriptions which implicitly specify a branch voltage or current in fixed-point form. The branch voltage or current is assigned a value by an expression which uses the branch voltage or current. This occurred in the simple implicit diode model in 5.1.3.4, where `I(a,c)` appeared on both sides of the contribution operator.

Examples:

Consider the model for an ideal *opamp*. In this model, the output is driven to the voltage which results in the input voltage being zero (`0`). The constitutive equation is

```
V(in) == 0
```

which can be formulated as

```
V(out) <+ V(out) + V(in);
```

This statement defines the output of the *opamp* to be a controlled voltage source by assigning to `V(out)` and defines the input to be high impedance by only probing the input voltage. The desired behavior results because the description is formulated in such a way it reduces to `V(in) = 0`. This approach does not result in the right tolerances being applied to the equation if `out` and `in` have different disciplines.

Verilog-AMS HDL includes a special syntax to use in this situation. The above branch contribution can be rewritten using an *indirect branch assignment*:

```
V(out): V(in) == 0;
```

which reads “find `V(out)` so `V(in) == 0`”.

This indicates `out` is driven with a voltage source and the source voltage needs to satisfy the given equation. Any branches referenced in the equation are only probed and not driven. In particular, `V(in)` acts as a voltage probe.

Examples:

A complete description of an ideal *opamp* is:

```
module opamp(out, pin, nin);
   electrical out, pin, nin;

   analog
      V(out):V(pin,nin) == 0;

endmodule
```

Syntax 5-4 shows the syntax for an indirect assignment statement.

```
analog_indirect_branch_assignment ::=
    bvalue : nexpr == analog_expression ;

nexpr ::=
     bvalue
    | pvalue
    | ddt ( bvalue | pvalue )
    | idt ( bvalue | pvalue )
```

Syntax 5-4—Syntax for indirect branch assignment

5.3.2.1 Multiple indirect assignments

For multiple indirect assignments statements, the targets frequently can be paired with any equation.

Examples:

The following ordinary differential equation,

$$\frac{dx}{dt} = f(x, y, z)$$

$$\frac{dy}{dt} = g(x, y, z)$$

$$\frac{dz}{dt} = h(x, y, z)$$

can be written as

```
V(x): ddt(V(x)) == f(V(x), V(y), V(z));
V(y): ddt(V(y)) == g(V(x), V(y), V(z));
V(z): ddt(V(z)) == h(V(x), V(y), V(z));
```

or

```
V(y): ddt(V(x)) == f(V(x), V(y), V(z));
V(z): ddt(V(y)) == g(V(x), V(y), V(z));
V(x): ddt(V(z)) == h(V(x), V(y), V(z));
```

or

```
V(z): ddt(V(x)) == f(V(x), V(y), V(z));
V(x): ddt(V(y)) == g(V(x), V(y), V(z));
V(y): ddt(V(z)) == h(V(x), V(y), V(z));
```

without affecting the results.

5.3.2.2 Indirect assignment and contribution

Indirect assignment is incompatible with contribution. Once a value is indirectly assigned to a branch, it cannot be contributed to using the branch contribution operator `<+`.

It is illegal to indirectly assign to an external branch or contribute to an external branch which has an indirect branch assignment.