8.3 Behavioral interaction

Verilog-AMS HDL supports several types of block statements for describing behavior, such as `analog` blocks, `initial` blocks, and `always` blocks. Typically, non-analog behavior is described in `initial` and `always` blocks, assignment statements, or assign declarations. There can be any number of `initial` and `always` blocks in a particular Verilog-AMS HDL module. However there can only be one `analog` block in that module.

Nets and variables in the continuous domain are termed *continuous nets* and *continuous variables* respectively. Likewise nets, regs and variables in the discrete domain are termed *discrete nets, discrete regs*, and *discrete variables*. In Verilog-AMS HDL, the nets and variables of one domain can be referenced in the other's context. This is the means for passing information between two different domains (continuous and discrete). Read operations of nets and variables in both domains are allow from both contexts. Write operations of nets and variables are only allowed from the context of their domain.

Verilog-AMS HDL provides ways to:

- access discrete primaries (e.g., nets, regs, or variables) from a continuous context
- access continuous primaries (e.g., flows, potentials, or variables) from a discrete context
- detect discrete events in a continuous context
- detect continuous events in a discrete context

The specific time when an event from one domain is detected in the other domain is subject to the synchronization algorithm described in 8.3.6 and Section 9. This algorithm also determines when changes in nets and variables of one domain are accessible in the other domain.

8.3.1 Accessing discrete nets and variables from a continuous context

Discrete nets and variables can be accessed from a continuous context. However, because the data types which are supported in continuous contexts are more restricted than those supported in discrete contexts, certain discrete types can not be accessed in a continuous context.

Table 8-1 lists how the various discrete net/variable types can be accessed from a continuous context.

Table 8-1—Discrete net/reg/variable access from a continuous context

Discrete net/reg/ variable type	Examples	Equivalent continuous variable type	Access to this discrete net/reg/variable type from a continuous context
real	**real** `r;` **real** `rm[0:8];`	`real`	Discrete reals are accessed in the continuous context as real numbers.
integer	**integer** `i;` **integer** `im[0:4];`	`integer`	Discrete integers are accessed in continuous context as integer numbers.
bit	**reg** `r1;` **wire** `w1;` **reg** `[0:9] r[0:7];` **reg** `r[0:66];` **reg** `[0:34] rb;`	`integer`	Discrete bit and bit groupings (buses and part selects) are accessed in the continuous context as integer numbers. The sign bit (`bit 31`) of the integer is always set to zero (`0`). The lowest bit of the bit grouping is mapped to the `0th bit` of the integer. The next bit of the bus is mapped to the `1st bit` of the integer and so on. If the bus width is less than 31 bits, the higher bits of the integer are set to zero (`0`). Access of discrete bit groupings with greater than 31 bits is illegal.

The syntax for a Verilog-AMS HDL primary is defined in Syntax 8-1.

primary ::=
  number
  | identifier
  | identifier **[** expression **]**
  | identifier **[** *msb*_constant_expression **:** *lsb*_constant_expression **]**
  | concatenation
  | analog_function_call
  | string
  | access_function

Syntax 8-1—Syntax for primary

Examples:

The following example accesses the discrete primary `in` from a continuous context.

```
module onebit_dac (in, out);
input in;
inout out;
wire in;
electrical out;

analog
   if (in == 0)
      V(out) <+ 0.0;
   else
      V(out) <+ 3.0;

endmodule
```

Issue 24,58: Section 8.3.2 cleanup. The current LRM is ambigous in accessing X & Z bits in the analog context. (Sri)

8.3.2 Accessing X and Z bits of a discrete net in a continuous context

Discrete nets can contain bits which are set to x (*unknown*) or z (*high impedance*). Verilog-AMS HDL supports ~~comparisons which take account of x and z bits in the continuous context. The specific features are:~~ accessing of 4-state logic values within the analog context. The x and z states must be translated to equivalent analog real or integer values before being used within the analog context. The language supports the following specific features which provide a mechanism to perform this conversion.

- the case equality operator (===)
- the case inequality operator (!==)
- the **case**, **casex**, and **casez** statements
- binary, octal and hexadecimal numeric constants which can contain x and z as digits.

The case equality and case inequality operators have the same precedence as the equality operator.

Example:

```
module a2d(dnet, anet);
input dnet;
wire dnet;
logic dnet;
output anet;
electrical out;

analog begin
```

```
        case (dnet)
            1'b1:var = 5;
            1'bx:var = var;// hold value
            1'b0:var = 0;
            1'bz:var = 2.5; // high impedence - float value
        endcase
            V(anet) <+ var;

    end
    endmodule
```

Note: case statement may be replaced with if-else-if statement using the case equality operators to perform the 4-state logic value comparisons.

Accessing digital net and digital binary constant operands are supported within analog context expressions. It is an error these operands returns 'x' or 'z' bits values when solved. It will be an error if the value of the digital variable being accessed in the analog context goes either to 'x' or 'z'.

Example:

```
module converter(dnet,anet);
reg dnet;
electrical anet;
integer var1;
real var2;

initial begin
    dnet = 1'b1;
    #50 dnet = 1'bz;
    $finish;
end

analog begin
    var1 = 1'bx;// error
    var2 = 1'bz;// error
    var1 = 1 + dnet;// error after #50

    if (dnet == 1'bx)// error
        $display("Error to access x bit in continous context");

    V(anet) <+ 1'bz;// error
    V(anet) <+ 1'bz;// error after #50
end
endmodule
```

~~All operators, functions, and statements are allowed in continuous contexts, except for case-equality, case-inequality, case, casex, and casez, which shall report an error if the expressions they operate on contain x or z bits.~~

The syntax for the features which support `x` and `z` comparisons in a continuous context is defined in 2.5 and 6.5. Support for `x` and `z` is limited in the analog blocks as defined above.

Note: Consult *IEEE 1364-1995 Verilog HDL* for a description of the semantics of these operators.

8.3.3 Accessing continuous nets and variables from a discrete context

All continuous nets can be probed from a discrete context using access functions. All probes which are legal in a continuous context of a module are also legal in the discrete context of a module. Therefore for Verilog-AMS HDL, the definition of *IEEE 1364-1995 Verilog HDL*'s *primary* is shown in Syntax 8-2.

```
digital_primary ::=
        digital_number
      | identifier
      | identifier [ digital_expression ]
      | identifier [ digital_msb_constant_expression : digital_lsb_constant_expression ]
      | digital_concatenation
      | digital_multiple_concatenation
      | digital_function_call
      | ( digital_mintypmax_expression )
      | access_function_reference
```

Syntax 8-2—Syntax for digital_primary

Examples:

The following example accesses the continuous net `V(in)` from the discrete context is.

```
module sampler (in, clk, out);
inout in;
input clk;
output out;
electrical in;
wire clk;
reg out;

always @(posedge clk)
   out = V(in);

endmodule
```

Continuous variables can be accessed for reading from any discrete context in the same module where these variables are declared. Because the discrete domain can fully represent all continuous types, a continuous variable is fully visible when it is read in a discrete context.

8.3.4 Detecting discrete events in a continuous context

Discrete events can be detected in a Verilog-AMS HDL continuous context. The arguments to discrete events in continuous contexts are in the discrete context. A discrete event in a continuous context is non-blocking like the other event types allowed in continuous contexts. The syntax for events in a continuous context is shown in Syntax 8-3.

```
event_control_statement ::=
    event_control statement_or_null

event_control ::=
     @ event_identifier
    | @ ( event_expression )

event_expression ::=
     global_event
    | event_function
    | digital_expression
    | event_identifier
    | posedge digital_expression
    | negedge digital_expression
    | event_expression or event_expression
```

Syntax 8-3—Syntax for event control statement

Examples:

The following example shows a discrete event being detected in an analog block.

```
module sampler3 (in, clk1, clk2, out);
input in, clk1, clk2;
output out;
wire clk1;
electrical in, clk2, out;

analog begin
   @(posedge clk1 or cross(V(clk2), 1))
      vout = V(in);
   V(out) <+ vout;
end

endmodule
```

8.3.5 Detecting continuous events in a discrete context

In Verilog-AMS HDL, monitored continuous events can be detected in a discrete context. The arguments to these events are in the continuous context. A continuous event in a discrete context is blocking like other discrete events. For Verilog-AMS HDL, the definition of *IEEE 1364-1995 Verilog HDL*'s *event_expression* is shown in Syntax 8-4.

```
digital_event_expression ::=
        digital_expression
      | event_identifier
      | posedge digital_expression
      | negedge digital_expression
      | event_function
      | digital_event_expression or digital_event_expression
```

Syntax 8-4—Syntax for digital event expression

Examples:

The following example detects a continuous event in an always block.

```
module sampler2 (in, clk, out);
input in, clk;
output out;
wire in;
reg out;
electrical clk;

always @(cross(V(clk) - 2.5, 1))
   out = in;

endmodule
```

8.3.6 Concurrency

Verilog-AMS HDL provides synchronization between the continuous and discrete domains. Simulation in the discrete domain proceeds in integer multiples of the digital tick. This is the smallest value of the second argument of the **`timescale** directive (see section 16.7 in *IEEE 1364-1995 Verilog HDL*). Thus, values calculated in the digital domain shall be constant between digital ticks and can only change at digital ticks.

Simulation in the continuous domain appears to proceed continuously. Thus, there is no time granularity below which continuous values can be guaranteed to be constant.

The rest of this section describes synchronization semantics for each of the four types of mixed-signal behavioral interaction. Any synchronization method can be employed, provided the semantics preserved. A typical synchronization algorithm is described in 9.2.

8.3.6.1 Analog event appearing in a digital event control

In this case, an analog event, such as `cross` or `timer`, appears in an `@()` statement in the digital context.

Examples:

```
always begin
   @(cross(V(x) - 5.5,1))
   n = 1;
end
```

When it is determined the event has occurred in the analog domain, the statements under the event control shall be scheduled in the digital domain at the largest digital time tick smaller than or equal to the time of the analog event. This event shall not be schedule in the digital domain earlier than the current digital event (see 9.2.3).

8.3.6.2 Digital event appearing in an analog event control

Examples:

```
analog begin
   @(posedge n)
   r = 3.14
end
```

In this case, a digital event, such as `posedge` or `negedge`, appears in an `@()` statement in the analog context.

When it is determined the event has occurred in the digital domain, the statements under the event control shall be executed in the analog domain at the time corresponding to a real promotion of the digital time (e.g., `27 ns` to `27.0e-9`).

8.3.6.3 Analog primary appearing in a digital expression

In this case, an analog primary (variable, potential, or flow) whose value is calculated in the continuous domain appears in a expression which is in the digital context; thus the analog primary is evaluated in the digital domain.

The expression shall be evaluated using the analog value calculated for the time corresponding to a real promotion of the digital time at which the expression is evaluated.

8.3.6.4 Digital primary appearing in an analog expression

In this case, a digital primary (reg, wire, integer, etc.) whose value is calculated in the discrete domain appears in an expression which is in the analog context; thus the analog primary is evaluated in the continuous domain.

The expression shall be evaluated using the digital value calculated for the greatest digital time tick which is less than or equal to the analog time when the expression is evaluated.