

VHDL-AMS

VHDL-AMS - Analog & Mixed Signal extensions

IEEE Standard 1076.1 (1999)

Superset of VHDL - IEEE Standard 1076-1993

Can be used to model electrical or mechanical systems

Mathematical Foundation

Equations describing continuous parts are differential-algebraic equations (DAEs)

DAE theory has been developed in the last 15 years

Theory covers properties and the numerical solution of DAEs of the form

$F(x, dx/dt, t) = 0$ where x is the vector of unknowns and F is a vector of expressions

Reasons for development

Need for only one simulator as they are expensive

Support for modeling level above Spice

The growth of mixed signal systems

1

VHDL-AMS

Pure VHDL Model of Differentiator

```
entity diff is
  generic (r, c: real);
  port (vi: in real; vo: out real);
end entity diff;
```

Unidirectional signals.
Does not support Kirchoff's law.

architecture simple of diff is

```
begin
  process (vi) is
    variable tnow, tlast: real;
  begin
    tnow := real(now/ns)*1.0e-9;
    vo <= -R*C(vi - vi'last_value)/(tnow - tlast);
    tlast := tnow;
  end process;
end architecture simple;
```

Connects time to real and control
time step. The problem is event
driven nature.

Write own formulae.

2

Quantities and Equations

- Two New Objects for VHDL
 - Terminal
 - Either interface object or local object
 - Terminal associations create analog netlists

terminal t1,t2:electrical;

- Two New Objects for VHDL

- Quantities

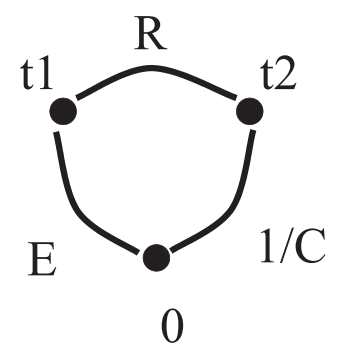
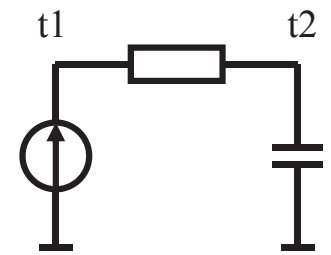
- Either interface object or local object
- Quantity associations create *signal flow* models

quantity a,b,c: voltage;

```
quantity_declaration ::= free_quantity_declaration
                       | branch_quantity_declaration
                       | source_quantity_declaration
```

```
free_quantity_declaration ::=
```

```
  quantity identifier_list : subtype_indication [ := expression ] ;
```



3

Implicit Quantities

- Q'Dot The time derivative of quantity Q
- Q'Integ The integral of Q from 0 to current time
- Q'Delayed(t) The quantity Q at time NOW-t
- S'Ramp[(tr[,tf])] A scalar quantity of the same type as signal S, that follows S with specified rise and fall times
- S'Slew[(max_rising_slope [,max_falling_slope])] Similar to S'Ramp, but with maximum slopes
- Q'Slew[(max_rising_slope,[max_falling_slope])] A scalar quantity that follows Q but with maximum slopes
- Q'ZOH(T,[initial_delay]) Zero-order hold with specified sampling interval and first sampling time
- Q'Ltf(num,den) Laplace transfer function of Q (scalar)
- Q'Ztf(num,den,T [, initial_delay]) Z-domain transfer function of Q (Scalar) with specified sampling interval

4

Branch Quantities

```
branch_quantity_declaration ::= quantity [across_aspect] [through_aspect] terminal_aspect;
across_aspect ::= identifier_list [tolerance_aspect] [:= expression] across
through_aspect ::= identifier_list [tolerance_aspect] [:= expression] through
terminal_aspect ::= plus_terminal_name [to minus_terminal_name];
```

- Defines a named *flow* path or a named *effort* difference; for example current and voltage
- Declared with a plus terminal and minus terminal
quantity v across j,i through t1 to t2;
- Plus terminal and minus terminal must have the same simple nature
- Minus terminals default to “ground”
- A branch quantity is composite if one of the terminals is composite
- Implicit quantity T’Reference is an across quantity directed from T to “ground”
- Implicit quantity T’Contribution is the signed sum of through quantities incident to T

5

Terminals and Natures

- Terminals belong to a nature
Terminal_declaration ::= **terminal** identifier_list : subnature_indication ;
- Two terminals may enter into a terminal association
port map (anode => t1, cathode => t2);
- A locally declared terminal or an unassociated formal terminal is the *root terminal* of a node

```
nature_declaration ::= nature identifier is nature_definition
nature_definition ::= scalar_nature_definition | composite_nature_definition
scalar_nature_definition ::= type_mark across type_mark through
subnature_declaration ::= subnature identifier is subnature_indication
```

- Terminals may be associated only with terminals of like nature
- Across aspect represents effort-like effects -- voltage, temperature, pressure
- Through aspect represents flow-like effects -- current, heat flow rate, fluid flow rate
- N’reference is a terminal - the “ground” for all terminals with nature N

6

Example: Package for electrical systems

```

package electrical_system is
    subtype voltage is real;
    subtype current is real;
    subtype charge is real;
    subtype flux is real;

    nature electrical is voltage across current through;
    nature electrical_vector is array(natural range <>) of electrical;

    alias ground is electrical'reference;
end package electrical_system;

```

7

Source Quantities

```

source_quantity_declaration ::=
    quantity identifier_list : subtype_indication source_aspect;

source_aspect ::=
    spectrum magnitude_simple_expression, phase_simple_expression
    | noise magnitude_simple_expression

function FREQUENCY return real;

```

- Source Quantities specify small-signal stimulus
 - Spectral source quantity for AC simulation
 - Noise source quantity for small-signal noise simulation
- Magnitude and phase expressions may depend on quantities and FREQUENCY

8

Implicit DAEs

- Each Across quantity is the difference between reference quantities of its terminals
- The algebraic sum of through quantities at a root terminal is zero
- The reference quantities of each pair of associated terminals are equal
- Each pair of associated terminals are equal
- Each implicit quantity is constrained to its appropriate value

9

Simultaneous Statements

- Simultaneous Statements express explicit DAEs


```

simultaneous_statement ::=
    simple_simultaneous_statement
    | simultaneous_if_statement
    | simultaneous_case_statement
    | simultaneous_procedural_statement
    | simultaneous_null_statement
      
```
- The semantics of if, case and procedural are derived from the semantics of the simple simultaneous statement
- The Simple Simultaneous Statement
 - Simultaneous statement has a collection of characteristic expressions


```

simple_simultaneous_expression ::=
    [label:] simple_expression ==
    simple_expression [tolerance_aspect];
          
```

Simultaneous Statements

- **Scalar expressions:**
 - The statement has a single characteristic expression - the difference of the statement expressions
- **Composite expressions:**
 - There must be a matching scalar subelement of the right-hand expression for each scalar subelement of the left-hand expression
 - The characteristic expressions are the differences of the matching scalar subelements of the statement expressions

- **The Simultaneous Conditional Statement**

- Selects one of the enclosed sequences of simultaneous statements

simultaneous_if_statement ::=

```
[if_label:]  if condition use                simultaneous_statement_part
              [ elsif condition use          simultaneous_statement_part]
              [ else                            simultaneous_statement_part]
              end use [if_label];
```

11

Simultaneous Statement

- **The Simultaneous Case Statement**

- Selects one of a number of alternative simultaneous statement lists

simultaneous_case_statement ::=

```
[case_label:] case expression use
                simultaneous_alternatives
                end use [case_label];
```

simultaneous_alternative ::=

```
when choices => simultaneous_statement_part
```

- **The Simultaneous Procedural Statement**

- Allows the formulation of equation as in-line sequential code

simultaneous_procedural_statement ::=

```
[procedural_label:] procedural [is]
                    procedural_declarative_part
                    begin
                    sequential_statements
                    end procedural [procedural_label];
```

12

Simultaneous Statement

- Semantics of Simultaneous Procedural Statement

- Defined by rewrite to the form:

$$FP(Ta'(Q1,..Qm),X1,..Xn) == Ta'(Q1,..Qm)$$

- The Qs are quantities, the Xs are quantities, signals, constants or literals
- FP contains local variable declarations corresponding with, and initialized to, the values of Q1..Qm
- The members of Q1..Qm are just those variables that are “written” by sequential statements
- FP returns the aggregate of the values of those variables

13

Examples of Simultaneous Statements

- A linear resistor

```
use electrical_system.all;
```

```
entity resistor is
```

```
    generic (resistance: real);
```

```
    port (terminal n1, n2: electrical);
```

```
end entity resistor;
```

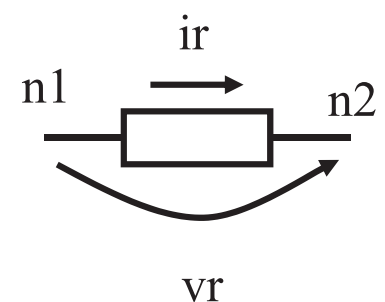
```
architecture signal_flow of resistor is
```

```
    quantity vr across ir through n1 to n2;
```

```
begin
```

```
    ir == vr / resistance;
```

```
end architecture signal_flow;
```



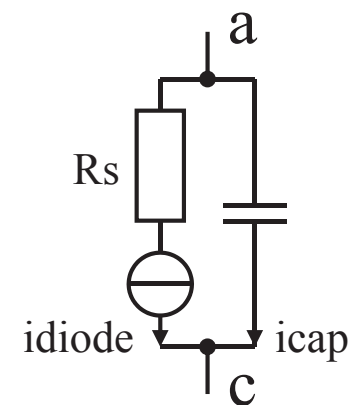
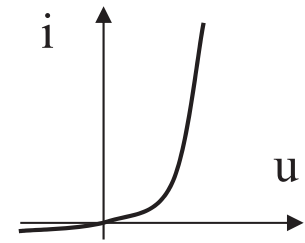
14

Examples of Simultaneous Statements

- A parameterized diode

```

use electrical_system.all, ieee.math_real.all;
entity diode is
    generic (Iss, n, Vt, tau, Rs, Cj0, Vj: real);
    port (terminal a, c: electrical);
end entity diode;
architecture level0 of diode is
    quantity vdiode across idiode, icap through a to c;
    quantity q: charge;
    quantity nsf: real noise sqrt(idiode/frequency);
begin
    idiode == iss * (exp((vdiode-Rs*idiode)/(n*Vt))-1)+nsf;
    q == tau * idiode - 2.0*Cj0*sqrt(Vj**2-Vj*vdiode);
    icap == q'dot;
end architecture level0;
  
```



15

Examples of Simultaneous Statements

- A sinusoid voltage source

```

use electrical_systems.all, ieee.math_real.all;
entity vsource is
    generic (magnitude, freq: real;
             phase: real := 0.0);
    port (terminal a, c: electrical);
end entity vsource;

architecture sine of vsource is
    quantity v across i through a to c;
    quantity ac:real spectrum magnitude/sqrt(2.0),phase;
begin
    v == magnitude * sin(2.0*math_pi*freq*NOW) + ac;
end architecture sine;
  
```

16

Tolerances

- Each quantity and simultaneous statement belongs to a user-definable tolerance group, which can be specified for subtypes, subnatures, quantities and simultaneous statements

```
subtype_indication ::=
```

```
    [resolution_function_name] type_mark [constraint] [tolerance_aspect]
```

```
tolerance_aspect ::= tolerance string_expression
```

```
subnature_indication ::= nature_mark [index_constraint]
```

```
    [tolerance string_expression across string_expression through
```

- The tolerance group of a quantity is specified by its subtype
 - The “closest” tolerance aspect found when tracing subtype (or subnature) indications back to the base type
 - The tolerance group of type real is indicated by “”
- The tolerance group of a simple simultaneous statement is
 - The tolerance group of the quantity if the statement is of the form


```
quantity_name == simple_expression;
simple_expression == quantity_name;
```
 - Can be specified explicitly

17

Tolerance Example

- In package `electrical_system`:


```
subtype voltage is real tolerance “low_voltage”;
subtype current is real tolerance “low_current”;
```
- In a design entity:


```
architecture two of resistor is
    quantity vr across it through n1 to n2;
    -- tolerance group of vr and ir defined by their subtype
    quantity power:real;
    -- default tolerance for power is empty string
begin
    ir == vr/resistance; -- default tolerance group from ir
    power == vr * ir tolerance “other”;
end architecture two;
```

18

Time

- New Time for Mixed-Mode Simulation
 - The internal simulation time is redefined to be of a new definitional type `Universal_Time`
 - Conversion from `Time` or `Real` to `Universal_Time` is exact
 - Conversion from `Universal_Time` to `Time` and `Real`
 - Have identical slopes and intercepts
 - Are linear within the accuracy of the representation of `Real` and the resolution limit of `Time`
 - Always round down to the nearest `Real` or `Time` value
- Overloaded function `NOW`:


```
impure function NOW return Real;
```
- Overloaded `S'Last_event` to return type `Real`

19

Time

- Allow `Real` expression in timeout clause of a wait statement:


```
wait for 0.5;
```

this is equivalent to

```
quantity q: Real; signal s: Real;
q == now-s'Ramp;
process begin
    s<=now;
    wait for 0 ns;
    wait on q'above(0.5);
    ...
end process;
```

20

Implicit Quantities

- S'Ramp[(tr[,tf])] A scalar quantity of the same type as signal S, that follows S with specified rise and fall times
- S'Slew[(max_rising_slope [,max_falling_slope])] Similar to S'Ramp, but with maximum slopes
- Q'Above(E)
 - Implicit Boolean Signal
 - TRUE when Q is above the threshold E and FALSE when Q is below the threshold
 - Q must be a scalar quantity, E must be an expression of the same type as Q
 - The transition between the two states creates an event on the signal
 - The value may be read in any non-static expression
 - The event may be used to trigger process execution

21

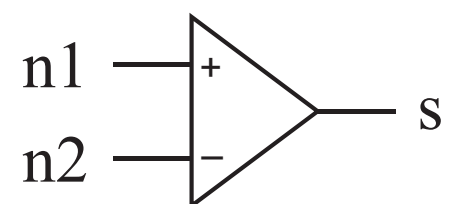
Example using Q'Above(E)

```

entity comparator is
    port (terminal n1, n2: electrical;
          signal s:out bit);
end entity comparator;

architecture ideal of comparator is
    quantity v across n1 to n2;
begin
    with v'Above(0.0) select
        s <=      '1' when true,
                '0' when false;
end architecture ideal;

```



22

Simulation Cycle

- How does the simulation cycle work?
 - Analog simulation cycle must deteriorate into digital one in the limits
 - Analog simulation cycle based on Universal “Real” time
- Analog simulation cycle
 - a) Execute Analog Solver
 - b) Set $T_c = T_n$, terminate if $T_n \leq \text{Time}'\text{High}$ or no active drivers
 - c) Update active explicit signals
 - d) If DOMAIN'Event, switch to time domain equations
 - f) Resume processes
 - g) Execute resumed nonpostponed processes
 - h) Determine T_n
 - i) If DOMAIN = INITIALIZATION_DOMAIN and $T_n > 0$ reset T_n to 0 and set the driver of DOMAIN to
 - TIME_DOMAIN if a time domain simulation follows
 - FREQUENCY_DOMAIN if a frequency domain simulation follows
 - j) Execute resumed postponed processes if $T_n \neq T_c$

23

Discontinuous Models

- An abstract model may display discontinuities in its quantities as its DAEs change with time
- Any of the following *may*, but *does not always*, cause a discontinuity when used in a simultaneous statement:
 - An event on a signal
 - A conditional test on quantities
 - A function call
- NO implicit mechanism can be designed to efficiently and automatically detect every discontinuity without introducing phantoms
- An active break signal cues the analog solver to “process” the discontinuity
- The values of the ASP are the initial values for the next continuous interval

24

Synchronizing Analog to Digital

- A break statement announces a discontinuity in some quantity or its derivative at the instance of execution.
- Tells analog solver to reinitialize for next continuous interval.
- Both sequential and concurrent forms.
- New initial conditions may be specified at the same time.
- A model that causes a discontinuity at some time T and does not execute a break statement at T is erroneous.

```

with din select reff <=
    rof when 'Z';
    ron when others;
break on reff;

```

25

Synchronizing Digital to Analog

```

case s is
when 1 =>
    dout := '1';
    wait on vin'above(vlow);
when 2 =>
    dout := '0';
    wait on vin'above(vlow), vin'above(vhi);

```

- Q'above(E) is an attribute of Q.
- Q must be a scalar quantity. The result is a Boolean signal.
- An event occurs at the instance of threshold crossing.

26

Example: Single-pole double-throw switch

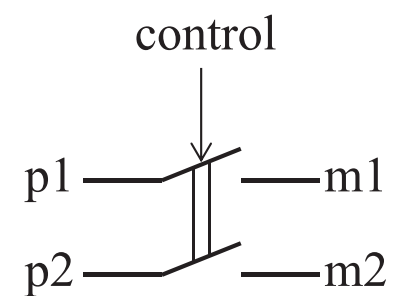
```

entity double_throw is
    port(    signal control:IN bit;
            terminal p1,m1,p2,m2:electrical);
end entity double_throw;

architecture ideal of double_throw is
    quantity v1 across i1 through p1 to m1;
    quantity v2 across i2 through p2 to m2;

begin
    if control = '0' use
        i1 == 0.0; i2 == 0.0;
    else
        v1 == 0.0; v2 == 0.0;
    end use;
    break on control;    -- concurrent break statement
end architecture ideal;

```



27

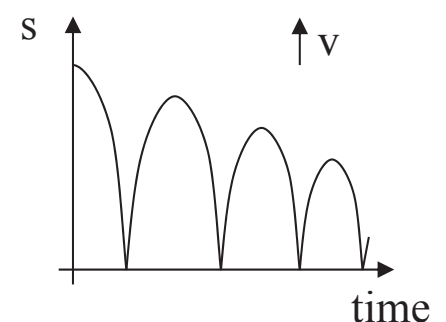
Example: Bouncing Ball

```

entity bouncer is end entity bouncer;
architecture ball of bouncer is
    quantity v:velocity;           -- m/sec
    quantity s:displacement;       -- m
    constant g: real := 9.81;      -- m/sec**2
    constant air_res : real := 0.001; -- 1/m

begin
    s'Dot == v;
    if v>0.0 use
        v'Dot == -g - v**2*air_res;
    else
        v'Dot == -g + v**2*air_res;
    end use;
    break v => -v when not s'Above(0.0);    -- Announce discontinuity, reset
                                              -- velocity value
    break v => 0.0, s => 10.0;             -- Specify initial conditions
end architecture ball;

```



28

Frequency Domain Simulation

- Small signal-model defined as first term of Taylor expansion of $\underline{F}(\underline{x})$ about quiescent point
- AC Simulation
 - Find quiescent point
 - Set simulation frequency
 - Replace base set of CEs with CEs defined by small-signal model
 - Augment small-signal model with frequency domain augmentation set
 - Solve resulting (linear) equations
- Noise Simulation
 - Find quiescent point
 - Set simulation frequency
 - Replace base set of CEs with CEs defined by small-signal model
 - Augment small-signal model with noise augmentation set
 - Create a noise variable corresponding to each quantity
 - For each noise source quantity NQ
 - Replace corresponding CE by NQ - magnitude
 - Solve resulting (linear) equations
 - Add to each noise variable the square of the magnitude of the corresponding quantity
 - Restore CE
 - Set each quantity to square root of corresponding noise variable

29

Intentionally left out

- Special definitions for mixed netlists
 - A designer cannot simply “connect” a quantity port with a terminal or vice-versa, nor a quantity port with a signal
 - Simultaneous statements defining the intended connection must be explicitly specified, for example

```

terminal t:electrical;
quantity v across i through t;                               -- branch to ground
quantity q: voltage;
component foo is
    port(quantity iq:out voltage);                             -- quantity “drives” terminal
end component foo;

c1: foo port map( iq => q);
v == q;                                                         -- ideal connection

```

30