Introduction

VHDL-AMS

IEEE std. 1076.1-1999

Based on tutorials

"Introduction to VHDL-AMS", FDL2000
T J Kazmierski, University of Southampton
"Analog and Mixed-Signal Modeling Using the VHDL-AMS Language", DAC1999
Ernst Christen, Analogy Kenneth Bakalar, Mentor Graphics
Allen Dewey, Duke University Eduard Moser, Bosch

Outline

- ❑ VHDL-AMS vs VHDL
- ❑ Simple analog behaviour: Low-pass filter
- ❑ Natures and terminals
- ❑ Nature examples: electrical, thermal
- ❑ Quantities vs signals
- ❑ Implicit quantities
- ❑ Simultaneous statements
- ❑ **break** statement

VHDL-AMS versus VHDL

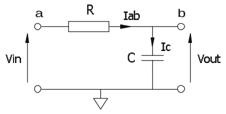
- ❑ IEEE 1076 - 1993
  - ➢ VHDL (VHSIC Hardware Description Language) - description and simulation of event-driven systems
- ❑ IEEE 1076.1 - 1999
  - ➢ Superset of VHDL'93 to include descriptions of systems that are continuous both in time and amplitude
- ❑ IEEE 1076.1 - 1999 is informally known as VHDL-AMS
- ❑ A VHDL-AMS simulator can also simulate VHDL descriptions
  - ➢ Digital part => set of processes + digital simulation kernel
  - ➢ Analog part => set of equations + analog solver

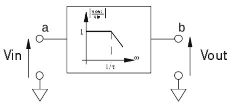
Outline

- ❑ VHDL-AMS vs VHDL
- ❑ **Simple analog behaviour: Low-pass filter**
- ❑ Natures and terminals
- ❑ Nature examples: electrical, thermal
- ❑ Quantities
- ❑ Implicit quantities
- ❑ Simultaneous statements
- ❑ **break** statement

Simple analog behaviour

Low-pass filter





$$\tau \frac{dVout}{dt} + Vout = Vin$$

$$H(s) = \frac{1}{\tau s + 1}$$

$\tau = R \cdot C$

Simple analog behaviour

Low-pass filter time-domain model

$\tau \frac{dVout}{dt} + Vout = Vin$

```
entity LowPass is
   generic ( tau: real := 1.0E-6);
   port ( quantity Vin: voltage;
             quantity Vout: out voltage);
end entity LowPass;
architecture DifferentialEqn of LowPass is
begin
 Vin - Vout == tau*Vout'DOT;
end architecture;
```

Simultaneous statement

Q'DOT is the time differential of quantity Q

Simple analog behaviour

Low-pass filter   s-domain model

```
entity LowPass is
   generic ( tau: real := 1.0E-6);
   port ( quantity Vin: voltage;
             quantity Vout: out voltage);
end entity LowPass;
architecture Transfer of LowPass is
   constant num: real_vector :=(0=>1.0);
   constant den: real_vector := (tau,1.0);
begin
   Vout == Vin'LTF(num,den);
end architecture;
```

transfer function:

$$\frac{1}{\tau s + 1}$$

Q'LTF is the Laplace Transfer Function, an attribute of quantity Q

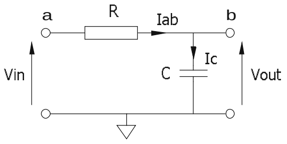
numerator and **den**ominator are static expressions of type real_vector

Quantities and simultaneous statements

- ❑ The VHDL-AMS simulator evaluates quantities such that the constraints specified by the simultaneous statements are satisfied with certain accuracy
- ❑ Analog accuracy is controlled by user-specified or the default tolerances

Simple analog behaviour

Low-pass filter circuit model (I)

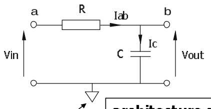


```
entity LowPass2 is
   generic ( R: real := 1.0E3;
              C: real := 1.0E-9);
   port ( terminal a,b: electrical);
end entity LowPass2;
```

nature

Simple analog behaviour

Low-pass filter circuit model (II)



ground

```
architecture circuit of LowPass2 is
  quantity Vin across a to ground;
  quantity Vab across Iab through a to b;
  quantity Vout across  Ic through b to ground;
begin
    Vab ==Iab*R;
    Ic  == C*Vout'DOT;
end architecture circuit;
```

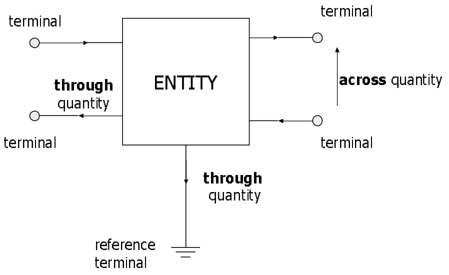
Outline

- ❑ VHDL-AMS vs VHDL
- ❑ Simple analog behaviour: Low-pass filter
- ❑ **Natures and terminals**
- ❑ Nature examples: electrical, thermal
- ❑ Quantities
- ❑ Implicit quantities
- ❑ Simultaneous statements
- ❑ **break** statement

Nature

- ❑ Characterises a physical discipline (electrical, mechanical, magnetic, thermal etc.)
- ❑ Has two aspects to model circuits of physical entities:
  - ➢ **across** aspect: effort, strength (voltage, velocity, field strength, temperature),
  - ➢ **through** aspect: flow through entity (current, force, flux, heat flow)
  - ➢ a nature defines data types for both aspects
- ❑ Defines properties of connectivity points in a network (terminals)

Terminals



Outline

- ❑ VHDL-AMS vs VHDL
- ❑ Simple analog behaviour: Low-pass filter
- ❑ Natures and terminals
- ❑ **Nature examples: electrical, thermal**
- ❑ Quantities
- ❑ Implicit quantities
- ❑ Simultaneous statements
- ❑ **break** statement

Electrical nature

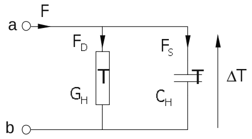
```
package ElectricalDomain is
  subtype voltage is real range –1.0E4 to 1.0E4 tolerance "voltage";
  subtype current is real range –1.0E2 to 1.0E2 tolerance "current";

  nature electrical is voltage across
                       current through
                       ground reference;

  nature electrical_vector is
           array (natural range <>) of electrical;

  alias undefined is real'LOW; -- undefined value used in
                               -- some SPICE-like models
  -- some physical constants
 constant Boltzmann : real :=1.380662e-23; -- Boltzmann constant
 constant ElectronCharge : real :=1.6021892e-19; -- electronic charge

end package ElectricalDomain;
```

Thermal nature

```
package ThermalDomain is

  subtype temperature is real range 0.0 to 1.0E4 tolerance "temperature";
  subtype heat_flow is real tolerance "heat_flow";

 nature thermal is
          temperature  across
          heat_flow through
           Tambient reference; -- thermal 'ground'

end package ThermalDomain;
```

Thermal example

```
entity heater is
  generic (-- heat conductance:
              GH: real := 1.0E-3; --W/K
              -- heat storage:
              CH: real := 0.1); -- Ws/K
  port (terminal a,b: thermal);
end entity core;
```



$$F_D = G_H \cdot \Delta T$$

$$F_S = C_H \cdot \frac{d\Delta T}{dt}$$

$$F = F_D + F_S$$

```
architecture behavior of heater is
  quantity DeltaT across FD, FS through a to b;
begin
   -- heat dissipation:
   FD == GH*DeltaT;
   -- heat storage:
   FS == CH*DeltaT'DOT; -- differential equation
end architecture behavior; -- of heater
```

Outline

- ❑ VHDL-AMS vs VHDL
- ❑ Simple analog behaviour: Low-pass filter
- ❑ Natures and terminals
- ❑ Nature examples: electrical, thermal
- ❑ **Quantities vs signals**
- ❑ Implicit quantities
- ❑ Simultaneous statements
- ❑ **break** statement

Quantities versus signals

- A quantity represents a continuous-time waveform, a signal is a discrete-time waveform
- A quantity is an unknown in the set of simultaneous differential-algebraic equations; a signal is driven by VHDL processes
- A scalar quantity must be of a floating-point type, signals can be of type bit, enumerated, integer, floating-point, ...

Outline

- VHDL-AMS vs VHDL
- Simple analog behaviour: Low-pass filter
- Natures and terminals
- Nature examples: electrical, thermal
- Quantities vs signals
- **Implicit quantities**
- Simultaneous statements
- **break** statement

Implicit quantities (I)

Q'DOT	time derivative of Q
Q'INTEG	time integral of Q from time=**0** to **now**
Q'DELAYED(T)	Q delayed by T (T >= 0)
Q'LTF(NUM,DEN)	Laplace Transfer Function with NUM and DEN as polynomial coefficients

Q'SLEW[(MAX_RISING_SLOPE[,MAX_FALLING_SLOPE])]
quantity that follows quantity Q with max limits on dQ/dt
default max_falling_slope is max_risign_slope
default max_rising_slope is infinite

Implicit quantities (II)

S'RAMP[(TRISE[,TFALL])]
A quantity that follows signal S, but with specified rise and fall times.
Default for TFALL is TRISE.
Default for TRISE is 0.0

S'SLEW[(RISING_SLOPE[,FALLING_SLOPE])]
A quantity Q that follows signal S, but its dQ/dt is limited by the specified slopes.
Default for max_ falling_ slope is max_ rising_ slope.
Default for max_ rising_ slope is infinity.

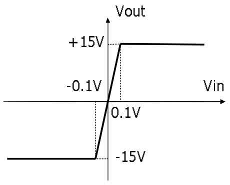
Outline

- VHDL-AMS vs VHDL
- Simple analog behaviour: Low-pass filter
- Natures and terminals
- Nature examples: electrical, thermal
- Quantities
- Implicit quantities
- **Simultaneous statements**
- **break** statement

Some simultaneous statements

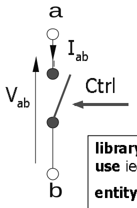
- Simultaneous **if** statement
- Simultaneous **case** statement

Simultaneous **if** statement



```
entity amplifier is
   generic(gain: real := 150.0);
   port(quantity Vin: voltage;
         quantity Vout: out voltage);
end entity amplifier;

architecture DC of amplifier is
begin
   if Vin < -15.0/gain use
         Vout == -15.0;
   elsif Vin >15.0/gain use
         Vout == 15.0;
   else
         Vout == gain*Vin;
   end if;
end architecture DC;
```

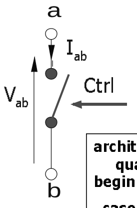
Simultaneous **case** statement



```
std_logic values:
  'U'      -- Uninitialized
  'X'      -- Forcing  Unknown
  '0'      -- Forcing  0
  '1'      -- Forcing  1
  'Z'      -- High Impedance
  'W'      -- Weak     Unknown
   'L'     -- Weak     0
  'H'      -- Weak     1
  '-'      -- Don't care
```

```
library std_logic_1164;
use ieee.std_logic_1164.all;

entity IdealSwitch is
         port( Ctrl: std_logic;
                terminal a,b: electrical);
end entity;
```

Simultaneous **case** statement



```
architecture behaviour of IdealSwitch is
    quantity Vab across Iab through a to b;
begin
  case Ctrl use
    when    '1' | 'H' => Vab == 0; -- switch closed
    when others    => Iab == 0; -- switch open
  end case;

end architecture;
```

Outline

- ❑ VHDL-AMS vs VHDL
- ❑ Simple analog behaviour: Low-pass filter
- ❑ Natures and terminals
- ❑ Nature examples: electrical, thermal
- ❑ Quantities
- ❑ Implicit quantities
- ❑ Simultaneous statements
- ❑ **break statement**

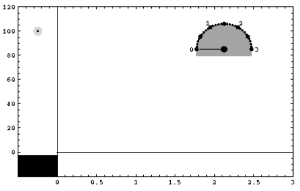
break statement

- ❑ A **break** statement overrides quantity values
  It executes concurrently with the analog model

Problems solved with break statement:

- ❑ Analog solver must re-initialize for next continuous interval
  - ➢ break statement announces a discontinuity in the solution of the differential algebraic equations
- ❑ Setting initial values for quantities
- ❑ A break statement for quantity Q replaces
  - ➢ the equation Q'Dot == 0 while finding the quiescent state
  - ➢ the equation Q == Q(t-) when re-initializing after discontinuity

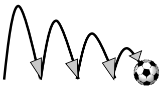
Bouncing ball



Bouncing ball

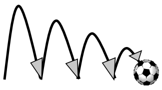
Modelling of waveform discontinuities

```
architecture bouncing of ball is
  quantity v: velocity;
  quantity s: displacement;
  constant G: real  := 9.81; -- G-force
  constant AirResistance: real  := 0.1;
begin
 -- Specify initial conditions
  break v => 0.0, s => 10.0 ;
 -- Introduce discontinuity when ball hits ground and reset velocity value:
  break v => -v when v'ABOVE(0.0) and not s'ABOVE(0.0);
  s'DOT == -v;
  if v > 0.0 use
          v'DOT == G  - AirResistance*v**2; -- falling
    else v'DOT == G + AirResistance*v**2; -- rising
  end if;
end architecture bouncing;
```



ABOVE attribute

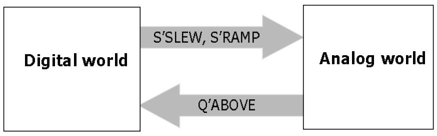
```
break v => -v
         when v'ABOVE(0.0) and not s'ABOVE(0.0);
```



- ❑ Q'ABOVE(Expression) is a signal associated with quantity Q
  - an event occures on the signal Q'ABOVE when Expression is becomes FALSE or TRUE;
- ❑ Processes can be sensitive to Q'ABOVE, since it is a signal

A/D and D/A interfacing summary

- ❑ Q'ABOVE(E) is a signal that announces discrete events in response to quantity variations.
- ❑ S'RAMP and S'SLEW are quantities that announce analog variations in response to discrete events on signals.



"Conclusions" ➔ Further reading

- ❑ http://www.eda.org/vhdl-ams
  - ➢ Home page of the IEEE 1076.1 (VHDL_AMS) Working Group
- ❑ http://www.vhdl-ams.com/
  - ➢ Useful site with links to tools and models
- ❑ http://vhdl.org/vi/analog/ftp-files/documentation/tutdac99.pdf
  - ➢ Tutorial with many examples, appr. 200 pages

search tutdac99 with yahoo