36th Design Automation Conference
New Orleans, June 21-25, 1999

Analog and Mixed-Signal Modeling Using the VHDL-AMS Language

Ernst Christen
Beaverton, OR

Kenneth Bakalar
Rockville, MD

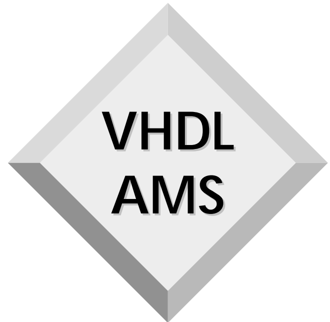
Allen M. Dewey
Durham, NC

Eduard Moser
Stuttgart, Germany

VHDL AMS

Tutorial Organization

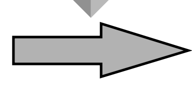
- Part I: Introduction to the VHDL-AMS Language
  - Continuous Time Concepts
  - Mixed Continuous/Discrete Time Concepts
  - Frequency Domain and Noise Modeling
- Part II: VHDL-AMS in Practical Applications
  - VHDL-AMS Modeling Guidelines
  - VHDL-AMS Modeling Techniques
    IC Applications
  - Modeling at Different Levels of Abstraction
    Telecom Applications
  - Modeling Multi-Disciplinary Systems
    Automotive Applications
  - MEMS Modeling Using the VHDL-AMS Language



Part I:

Introduction to the VHDL-AMS Language

Outline



- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

VHDL AMS

What is VHDL-AMS

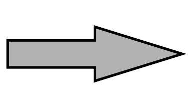
- IEEE Std. 1076-1993:
  - VHDL (VHSIC Hardware Description Language) supports the description and simulation of event-driven systems.
- IEEE Std. 1076.1-1999:
  - Extension to VHDL to support the description and simulation of analog and mixed-signal circuits and systems
- IEEE Std. 1076.1-1999 together with IEEE Std. 1076-1993 is informally known as VHDL-AMS
- VHDL-AMS is a strict superset of IEEE Std. 1076-1993
  - Any model valid in VHDL 1076 is valid in VHDL-AMS and yields the same simulation results

VHDL AMS

Why is VHDL-AMS needed?

- VHDL 1076 is suitable for modeling and simulating discrete systems
- Many of today's designs include at least some continuous characteristics:
  - System design
    - Mixed-signal electrical designs
    - Mixed electrical/non-electrical designs
    - Modeling design environment
  - Analog design
    - Analog behavioral modeling and simulation
  - Digital design
    - Detailed modeling (e.g. submicron effects)
- Designers want a uniform description language

Outline



- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

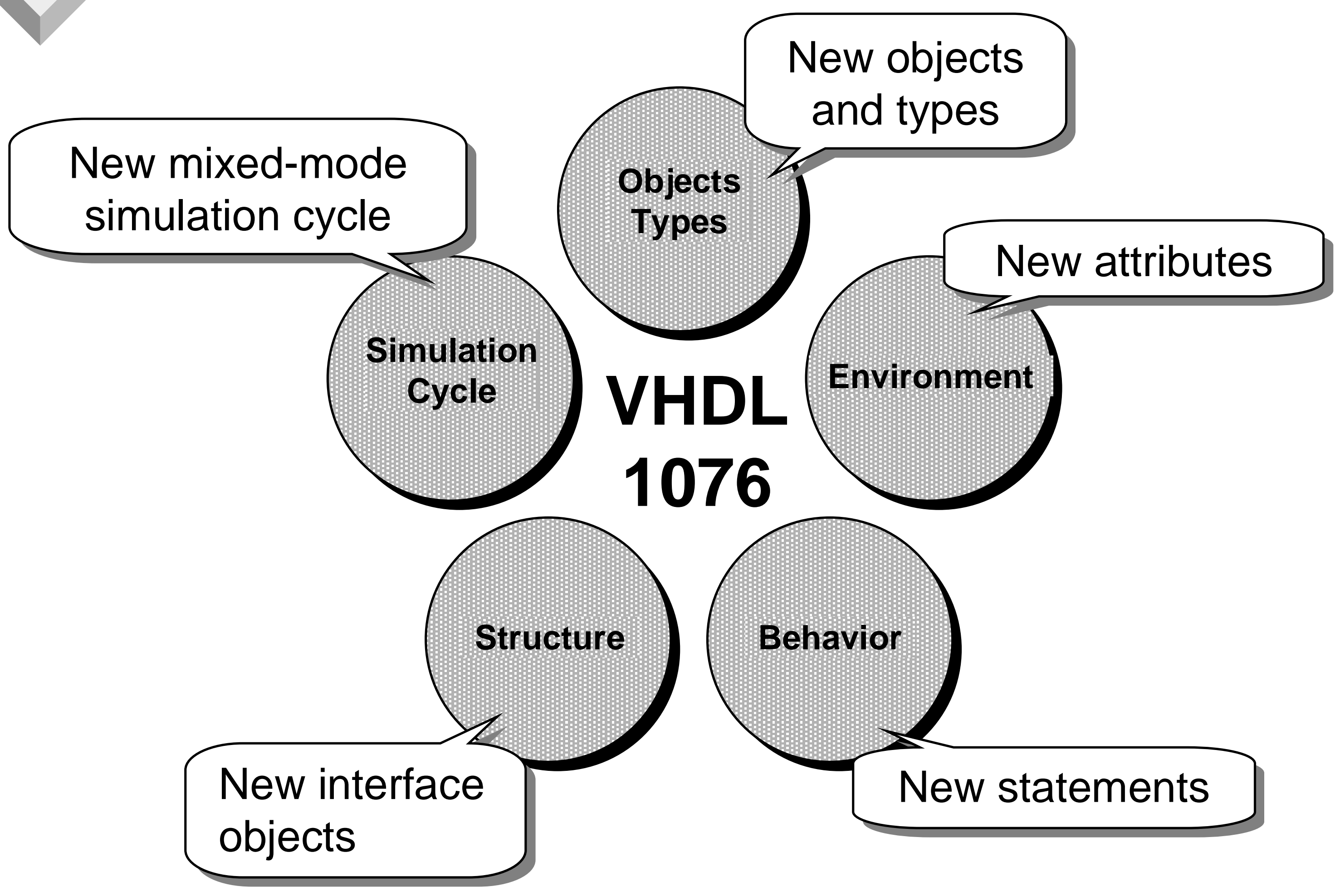
VHDL
AMS

VHDL 1076 Overview

- Entity defines interface of the model of a subsystem or physical device
- Each entity has one or more architectures, each implementing the behavior or structure of the subsystem or physical device
- Packages define collections of re-usable declarations and definitions: types, constants, functions etc.
- Strong type system
- Event-driven behavior described by processes that are sensitive to signals
- Well-defined simulation cycle, based on a canonical engine
- Predefined language environment

VHDL AMS

VHDL-AMS Language Architecture



VHDL-AMS Highlights (1)

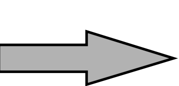
- Superset of VHDL 1076-1993
  - Full VHDL 1076-1993 syntax and semantics is supported
- Adds new simulation model supporting continuous behavior
  - Continuous models based on differential algebraic equations (DAEs)
  - DAEs solved by dedicated simulation kernel: the analog solver
  - Handling of initial conditions, piecewise-defined behavior, and discontinuities
  - Optimization of the set of DAEs being solved and how the analog solver computes its solution are outside the scope of VHDL-AMS

VHDL AMS

VHDL-AMS Highlights (2)

- Extended structural semantics
  - Conservative semantics to model physical systems
    e.g. Kirchhoff's laws for electrical circuits
  - Non-conservative semantics for abstract models
    Signal-flow descriptions
  - Mixed-signal interfaces
    Models can have digital and analog ports
- Mixed-signal semantics
  - Unified model of time for a consistent synchronization of mixed event-driven/continuous behavior
  - Mixed-signal initialization and simulation cycle
  - Mixed-signal descriptions of behavior
- Frequency domain support
  - Small-signal frequency and noise modeling and simulation

Outline

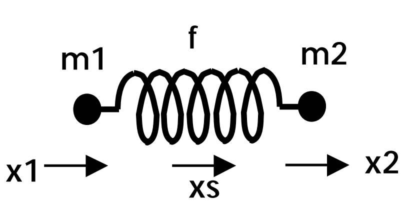


- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

VHDL AMS

Vibration in biatomic molecule

- First approximation: one-dimensional spring model



$$m_1\ddot{x}_1 = -f \bullet (x_1 - x_2)$$
$$m_2\ddot{x}_2 = -f \bullet (x_2 - x_1)$$
$$x_s = (m_1 x_1 + m_2 x_2)/(m_1 + m_2)$$
$$Energy = 0.5 \bullet (m_1\dot{x}_1^2 + m_2\dot{x}_2^2 + f \bullet (x_1 - x_2)^2)$$

```
use work.types.all;
entity Vibration is
end entity Vibration;

architecture H2 of Vibration is  -- hydrogen molecule
  quantity x1, x2, xs: displacement;
  quantity energy: REAL;
  constant m1, m2: REAL := 1.00794*1.6605655e-24;
  constant f: REAL := 496183.3;
begin
  x1'dot'dot == -f*(x1 - x2) / m1;
  x2'dot'dot == -f*(x2 - x1) / m2;
  xs == (m1*x1 + m2*x2)/(m1 + m2);
  energy == 0.5*(m1*x1'dot**2 + m2*x2'dot**2 + f*(x1-x2)**2);
end architecture H2;
```

VHDL AMS

Quantities (1)

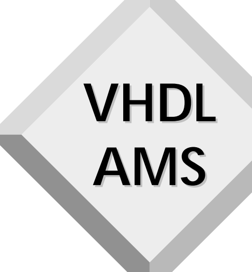
```
architecture H2 of Vibration is
   ...
   quantity x1, x2, xs: displacement;
   quantity energy: REAL;
   ...
begin
   ...
```

- New object in VHDL 1076.1
- Represents an unknown in the set of DAEs implied by the text of a model
- Continuous-time waveform
- Scalar subelements must be of a floating-point type
- Default initial value for scalar subelements is 0.0

Quantities (2)

```
   quantity x1, x2, xs: displacement;
   ...
begin
   x1'dot'dot == -f*(x1 - x2) / m1;
   ...
   energy == 0.5*(m1*x1'dot**2 +
          m2*x2'dot**2 + f*(x1-x2)**2);
```

- For any quantity Q, the attribute name Q'Dot denotes the derivative of Q w.r.t. time
- Q'Dot is itself a quantity



Simultaneous Statements (1)

```
architecture H2 of Vibration is
   ...
begin
   x1'dot'dot == -f*(x1 - x2) / m1;
   x2'dot'dot == -f*(x2 - x1) / m2;
   xs == (m1*x1 + m2*x2)/(m1 + m2);
   energy == 0.5*(m1*x1'dot**2 +
          m2*x2'dot**2 + f*(x1-x2)**2);
end architecture H2;
```

- New class of statements in VHDL 1076.1
- Simple simultaneous statements express relationships between quantities
  - Left-hand side and right-hand side must be expressions with scalar subelements of a floating point type
  - Statement is symmetrical w.r.t. its left-hand and right-hand sides
  - Expressions may involve quantities, constants, literals, signals, and (possibly user-defined) functions
  - At least one quantity must appear in a simultaneous statement

Simultaneous Statements (2)

- Analog solver is responsible for computing the values of the quantities such that the relationships hold (subject to tolerances)
- Simultaneous statements may appear anywhere a concurrent statements may appear
- The order of simultaneous statements does not matter
- Other forms for simultaneous statements:
  - Simultaneous if statement
  - Simultaneous case statement
  - Simultaneous procedural statement

VHDL AMS

Tolerances (1)

- Numerical algorithms used by analog solver can only find an approximation of the exact solution
- Tolerances are used to specify how good the solution must be
- Each quantity and each simultaneous statement belongs to a tolerance group indicated by a string expression
  - All members of a tolerance group have the same tolerance characteristics
- The language does not define how a tool uses tolerance groups
  - For example, abstol and reltol in SPICE-like algorithms

Tolerances (2)

- A quantity gets its tolerance group from its subtype

```
package types is
  subtype displacement is REAL
                         tolerance "default_displacement";
  ...
end package types;
```

```
architecture H2 of Vibration is  -- hydrogen molecule
  quantity x1, x2, xs: displacement;
  quantity energy: REAL;
  ...
```

- Type REAL belongs to an unnamed tolerance group: its tolerance code is ""

VHDL AMS

Tolerances (3)

- A simple simultaneous statement whose LHS or RHS is a quantity gets its tolerance group from the quantity, otherwise the tolerance group must be specified

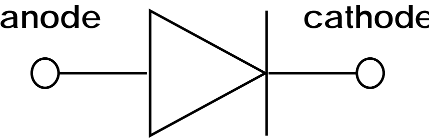
```
x1'dot'dot == -f*(x1 - x2) / m1;
m2 * x2'dot'dot == -f*(x2 - x1) tolerance "displacement";
xs == (m1*x1 + m2*x2)/(m1 + m2);
energy == 0.5*(m1*x1'dot**2 + m2*x2'dot**2 + f*(x1-x2)**2);
```

Outline

- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

Parameterized Diode

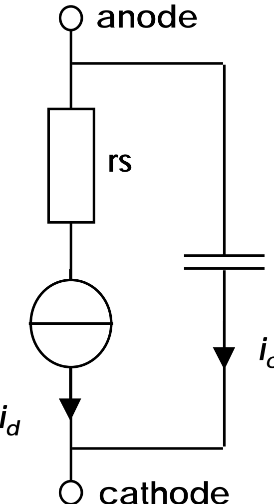
- Example of a conservative model of an electrical component



- Simple large-signal model

$$i_d = \text{is} \cdot \left(e^{(v_d - \text{rs} \cdot i_d)/\text{n} \cdot \text{vt}} - 1\right)$$

$$i_c = \frac{d}{dt}\left(\text{tt} \cdot i_d - 2 \cdot \text{cj0} \cdot \sqrt{\text{vj}^2 - \text{vj} \cdot v_d}\right)$$



VHDL AMS

VHDL-AMS Model of Diode

```
library IEEE, Disciplines;
use Disciplines.electrical_system.all;
use IEEE.math_real.all;
entity Diode is
  generic (iss: REAL := 1.0e-14;
           n, af: REAL := 1.0;
           tt, cj0, vj, rs, kf: REAL := 0.0);
  port (terminal anode, cathode: electrical);
end entity Diode;

architecture Level0 of Diode is
  quantity vd across id, ic through anode to cathode;
  quantity qc: charge;
  constant vt: REAL := 0.0258;    -- thermal voltage
begin
  id == iss * (exp((vd-rs*id)/(n*vt)) - 1.0);
  qc == tt*id - 2.0*cj0 * sqrt(vj**2 - vj*vd);
  ic == qc'dot;
end architecture Level0;
```

VHDL AMS

Terminal

```
library IEEE, Disciplines;
use Disciplines.electrical_system.all;
...
entity Diode is
   ...
   port (terminal anode, cathode: electrical);
end entity Diode;
```

- New object in VHDL 1076.1
- Basic support for structural composition with conservative semantics
- Belongs to a nature
  - Nature **electrical** defined in package **electrical_system**

VHDL AMS

Nature

- Represents a physical discipline or energy domain
  - Electrical and non-electrical disciplines
- Has two aspects related to physical effects
  - Across: effort like effects (voltage, velocity, temperature, etc.)
  - Through: flow like effects (current, force, heat flow rate, etc.)
- A nature defines the types of the across and through quantities incident to a terminal of the nature
- A scalar nature additionally defines the reference terminal for all terminals whose scalar subelements belong to the scalar nature
- A nature can be composite: array or record
  - All scalar subelements must have the same scalar nature
- No predefined natures in VHDL 1076.1

Electrical Systems Environment

```
package electrical_system is
   subtype voltage is REAL tolerance "default_voltage";
   subtype current is REAL tolerance "default_current";
   subtype charge  is REAL tolerance "default_charge";
   nature electrical is
      voltage        across          -- across type
      current        through         -- through type
      electrical_ref reference;      -- reference terminal
   alias ground is electrical_ref;
   nature electrical_vector is
      array(NATURAL range <>) of electrical;
end package electrical_system;
```

- Assume package is compiled into a library `Disciplines`

VHDL AMS

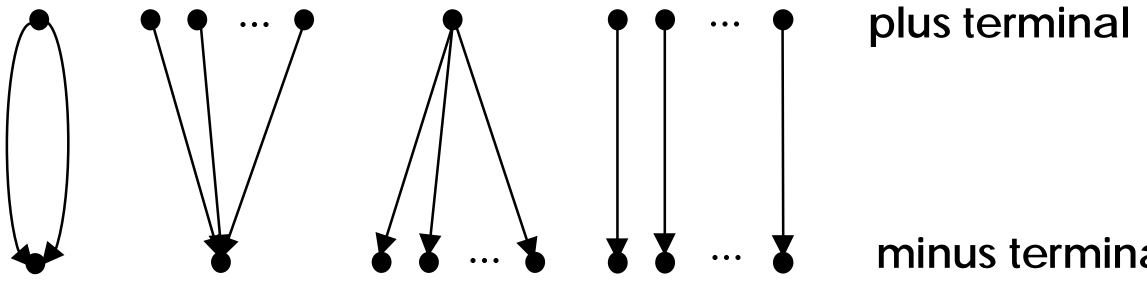
Branch Quantities (1)

```
architecture Level0 of Diode is
   quantity vd across id, ic through anode to cathode;
   ...
begin
   ...
end architecture Level0;
```

- Declared between two terminals
  - Plus terminal and minus terminal
  - Minus terminal defaults to reference terminal of nature
- **vd** is an across quantity: it represents the voltage between terminals **anode** and **cathode**
  - vd = $v_{anode} - v_{cathode}$
- **id** and **ic** are through quantities: they represents the currents in the two parallel branches
  - Both currents flow from terminal **anode** to terminal **cathode**

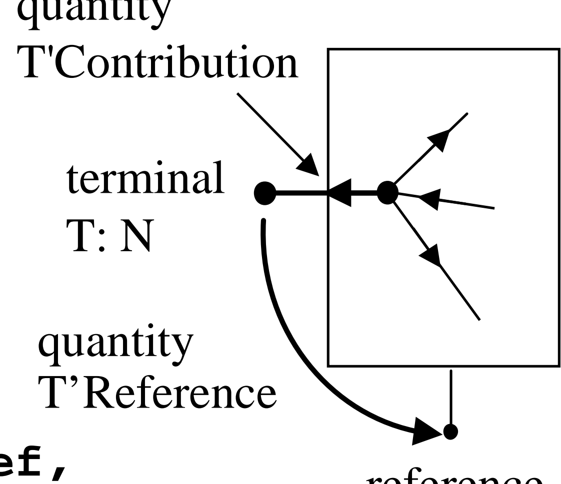
Branch Quantities (2)

- A branch quantity gets its type from the nature of its plus and minus terminals
- The scalar subelements of the plus and minus terminal of a branch quantity must belong to the same scalar nature
- Multiple across quantities declared between the same terminals have the same value
- Multiple through quantities declared between the same terminals define distinct parallel branches



Terminal Attributes

- T′Reference
  - Implicit across quantity with terminal T of nature N as plus terminal and reference terminal of N as minus terminal (e.g. voltage to ground)
- T′Contribution
  - Implicit through quantity
  - Value equals the sum of the values of all through quantities incident to T (with the appropriate sign)



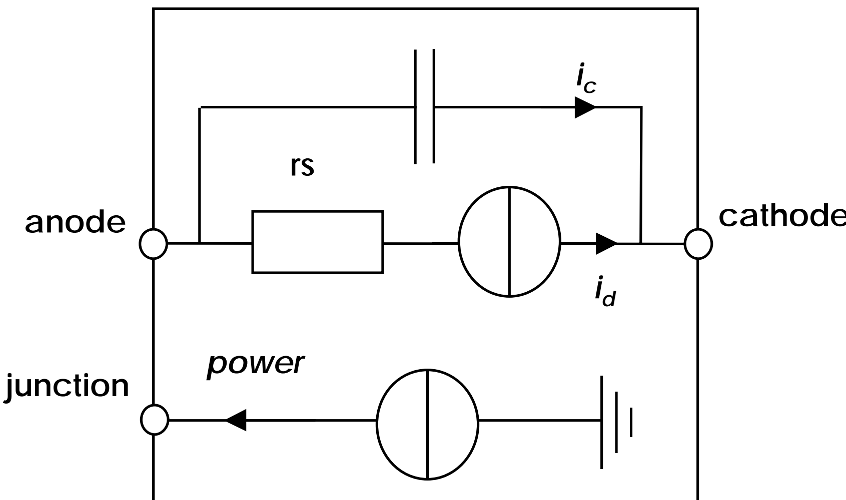
- For the diode model
  - Reference terminal is `electrical_ref`, aliased to `ground`
  - `vd` = `anode'reference` – `cathode'reference`
  - `anode'contribution` = `id` + `ic`
  - `cathode'contribution` = –(`id` + `ic`)

Outline

- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

Diode with Self Heating

- Example of a mixed-technology model:
  Electro-thermal interaction in a diode



- Thermal voltage VT now depends on temperature
- Thermal branch is a through source that represents the power dissipated in the diode

VHDL AMS

Diode with Self Heating VHDL-AMS Model Environment

- Thermal nature

```
package thermal_system is
   subtype temperature is REAL
                 tolerance "default_temperature";
   subtype heatflow is REAL
                 tolerance "default_heatflow";
   nature thermal is
      temperature across
      heatflow    through
      thermal_ref reference;
end package thermal_system;
```

- Physical constants, ambient temperature

```
package environment is
   constant boltzmann    : REAL := 1.381e-23;   -- in J/K
   constant elec_charge  : REAL := 1.602e-19;   -- in C
   constant ambient_temp: REAL := 300.0;        -- in K
   ...
end package environment;
```

VHDL AMS

Diode with Self Heating
VHDL-AMS Entity Declaration

```
library IEEE, Disciplines;
use Disciplines.electrical_system.all;
use Disciplines.thermal_system.all;
use Disciplines.environment.all;
use IEEE.math_real.all;
entity DiodeTh is
  generic (iss: REAL := 1.0e-14;
           n, af: REAL := 1.0;
           tt, cj0, vj, rs, kf: REAL := 0.0);
  port (terminal anode, cathode: electrical;
        terminal junction: thermal);
end entity DiodeTh;
```

VHDL AMS

Diode with Self Heating
VHDL-AMS Architecture Body

```
architecture Level0 of DiodeTh is
  quantity vd across id, ic through anode to cathode;
  quantity temp across power through
                                  thermal_ref to junction;
  quantity qc: charge;
  quantity vt: voltage;        -- thermal voltage
begin
  qc == tt*id - 2.0*cj0 * sqrt(vj**2 - vj*vd);
  ic == qc'dot;
  id == iss * (exp((vd-rs*id)/(n*vt)) - 1.0);
  vt == temp * boltzmann / elec_charge;
  power == vd * id;
end architecture Level0;
```

VHDL AMS

Test Bench for Diode with Self Heating

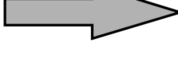

```
library Disciplines;
use Disciplines.electrical_system.all;
use Disciplines.thermal_system.all;
entity TestBench is
end entity TestBench;

architecture DiodeWithHeatSink of TestBench is
   terminal a, b: electrical;
   terminal j, h: thermal;
begin
   v0: entity Vdc generic map (dc => 1.0)
          port map (p => a, m => ground);
   r1: entity Resistor generic map (r => 1.0e3)
          port map (p => a, m => b);
   d1: entity DiodeTh port map (anode => b,
          cathode => ground, junction => j);
   heatres: entity ResistorTh generic map (r => 0.1)
          port map (p => j, m => h);
   heatsink: entity CapacitorTh generic map (c => 0.008)
          port map (p => h, m => thermal_ref);
   rad: entity ResistorTh generic map (r => 10.0)
          port map (p => h, m => thermal_ref);
end architecture DiodeWithHeatSink;
```

Basic Thermal Elements

```
library Disciplines;
use Disciplines.thermal_system.all;
entity ResistorTh is
   generic (r: REAL);
   port (terminal p, m: thermal);
end entity ResistorTh;

architecture Ideal of ResistorTh is
   quantity temp across power through p to m;
begin
   power == temp / r;
end architecture Ideal;
```

```
library Disciplines;
use Disciplines.thermal_system.all;
entity CapacitorTh is
   generic (c: REAL);
   port (terminal p, m: thermal);
end entity CapacitorTh;

architecture Ideal of CapacitorTh is
   quantity temp across power through p to m;
begin
   power == c * temp'dot;
end architecture Ideal;
```

Outline



- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

Piecewise Defined Behavior (1)

```
library IEEE, Disciplines;
use Disciplines.electrical_system.all;
use IEEE.math_real.all;
entity Compressor is
  generic (vmax : REAL := 10.0;
           a    : REAL := 100.0;
           gain : REAL := 1.0 );
  port (terminal ip, im, op, om : electrical);
end entity Compressor;
```

```
architecture A_law of Compressor is
  quantity vin across ip to im;
  quantity vout across iout through op to om;
  constant alog: REAL := 1.0 + log(a);
begin
  if -vmax/a < vin and vin < vmax/a use
    vout ==  gain * a * vin / ( alog * vmax );
  elsif vin > 0.0 use
    vout ==  gain * ( alog + log( vin/vmax) ) / alog;
  else
    vout == -gain * ( alog + log(-vin/vmax) ) / alog;
  end use;
end architecture A_law;
```

Piecewise Defined Behavior (2)

- Simultaneous if statement selects one of the statement parts based on the value of one or more conditions
- Each of the statement parts of a simultaneous if statement can include any of the simultaneous statements
  - Simple simultaneous statement
  - Simultaneous if statement
  - Simultaneous case statement
  - Simultaneous procedural statement
- Watch out for discontinuities in quantities and their derivatives!

Piecewise Defined Behavior (3)

```
library Disciplines;
use Disciplines.electrical_system.all;
entity VoltageLimiter is
   generic (vlim: REAL);       -- open loop gain
   port (terminal ip, im, op, om: electrical);
end entity VoltageLimiter;
```

```
architecture Bad of VoltageLimiter is
   quantity vin across ip to im;
   quantity vout across iout through op to om;
begin
   if vin > vlim use
      vout == vlim;
   elsif vin < -vlim use
      vout == -vlim;
   else
      vout == vin;
   end use;
end architecture Bad;
```

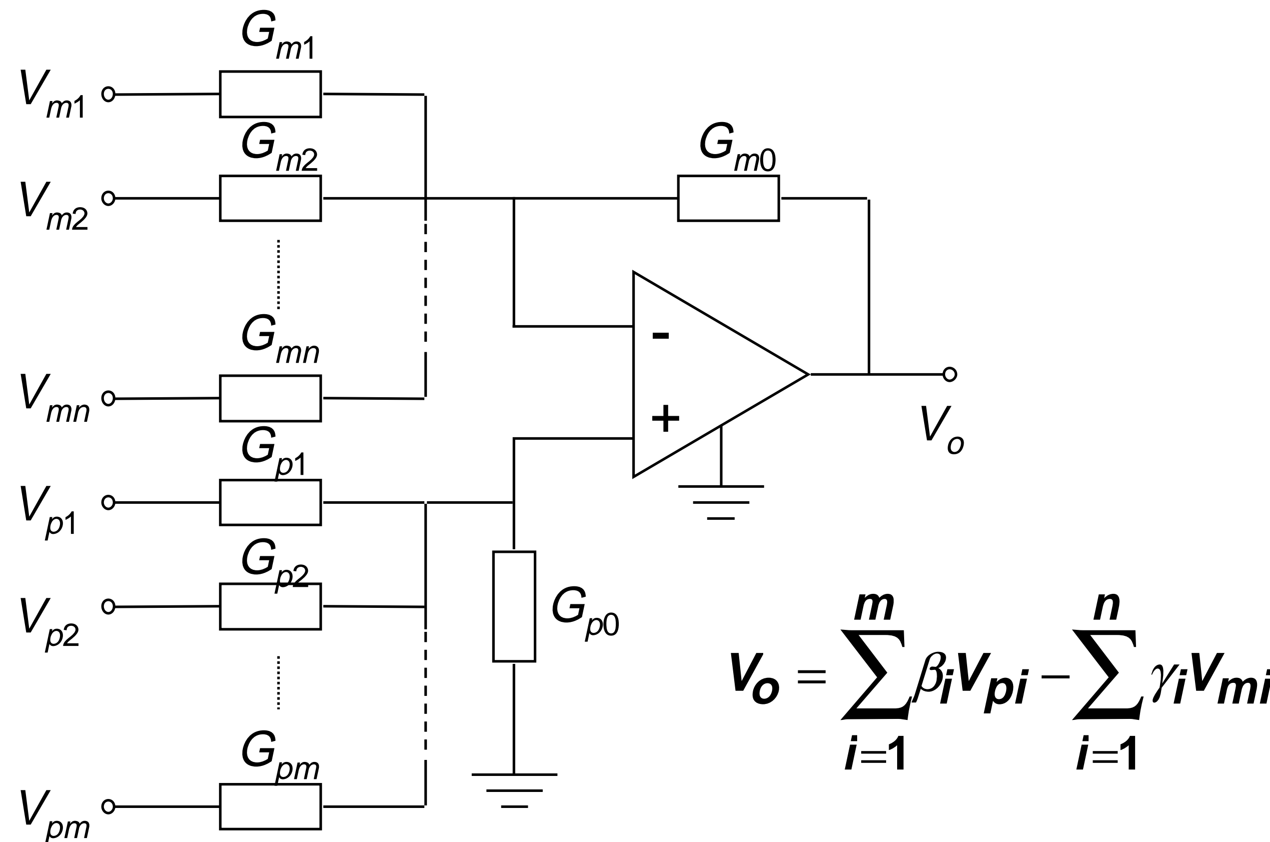
Discontinuity in first derivative

Outline

- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

VHDL AMS

Weighted Summer



$$V_o = \sum_{i=1}^{m} \beta_i V_{pi} - \sum_{i=1}^{n} \gamma_i V_{mi}$$

VHDL AMS

Generic Weighted Summer VHDL-AMS Entity Declaration

```
library Disciplines;
use Disciplines.electrical_system.all;

entity WeightedSummer is
   generic (beta, gamma: REAL_VECTOR);
   port (terminal inp, inm: electrical_vector;
         terminal o: electrical);
end entity WeightedSummer;
```

- New predefined type `real_vector`

```
type REAL_VECTOR is array (NATURAL range <>) of REAL;
```

VHDL AMS

Generic Weighted Summer VHDL-AMS Architecture Body: Declarations

```
architecture Proc of WeightedSummer is
   quantity vp across inp to ground;
   quantity vm across inm to ground;
   quantity vo across io through o to ground;
begin
   ...
end architecture Proc;
```

- Branch quantities **vp** and **vm** are composite because terminals **inp** and **inm** are composite

VHDL AMS

Generic Weighted Summer VHDL-AMS Architecture Body: Statements

- Using a simultaneous procedural statement

```
   ...
begin
  procedural is
    variable bvs, gvs: REAL := 0.0;
  begin
    for i in beta'range loop
      bvs := bvs + beta(i) * vp(i);
    end loop;
    for i in gamma'range loop
      gvs := gvs + gamma(i) * vm(i);
    end loop;
    vo := bvs - gvs;
  end procedural;
end architecture Proc;
```

- Allows writing equations using a sequential language
  - Supports all sequential statements except wait, signal assignment, break

Generic Weighted Summer
VHDL-AMS Architecture Body Revisited

VHDL AMS

- Using a simple simultaneous statement and an overloaded function

```
architecture Simult of WeightedSummer is
  ... -- same declarations as in Proc

  function "*" (a: REAL_VECTOR;
                b: electrical_vector'across)
                              return REAL is
    variable result: REAL := 0.0;
  begin     -- compute dot product
    for i in a'range loop
      result := result + a(i) * b(i);
    end loop;
    return result;
  end function "*";

begin
  vo == beta * vp - gamma * vm;
end architecture Simult;
```

Outline

- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

Signal-Flow Modeling (1)

```
entity AdderIntegrator is
  generic (k1, k2: REAL := 1.0);
  port(quantity in1, in2: in REAL;
       quantity output: out REAL);
end entity AdderIntegrator;

architecture Sfg of AdderIntegrator is
  quantity qint: REAL;
begin
  qint == k1*in1 + k2*in2;
  output == qint'integ;
end architecture Sfg;
```

- Interface quantities have mode `in` or `out`

Signal Flow Modeling (2)

- Modes are used for solvability checks
- Modes are also used to determine correctness of a port association:
  - A quantity port with mode `in` can be associated with any kind of quantity as an actual
  - A quantity port with mode `out` can only be associated with an actual that is:
    - A quantity port with mode `out`
    - A free quantity
    - A branch quantity
  - The same quantity can be associated as an actual with at most one quantity port with mode `out`

VHDL AMS

Signal Flow Test Bench

```
library IEEE;
use IEEE.math_real.all;
architecture Sfg of TestBench is
  quantity in1, in2, s: REAL;
  terminal t1, t2: electrical;
begin
  in1 == exp(-NOW);    -- NOW is current time
  i1: entity Isine
        generic map (ampl => 1.0, freq => 1.0e3)
        port map (p => ground, m => t1);
  r1: entity Resistor generic map (r => 1.0e3)
        port map (p => t1, m => t2);
  c1: entity Capacitor generic map (c => 1.0e-9)
        port map (p => t2, m => ground);
  cv: entity Electrical2Sfg(Across2Sfg)
        port map (p => t2, m => ground, output => in2;
  ai: entity AdderIntegrator port map (in1 => in1,
        in2 => in2, output => s);
end architecture Sfg;
```

t1
t2
e^{-t}
in1
Electrical To SFG
in2
s

Signal Flow Modeling: Conversion

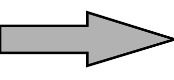
- Terminals and quantities cannot be connected directly, conversion models are needed

```
library Disciplines;
use Disciplines.electrical_system.all;
entity Electrical2Sfg is
  port (terminal p, m: electrical;
        quantity output: out REAL);
end entity Electrical2Sfg;
```

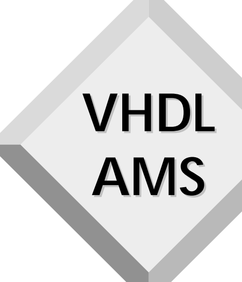
```
architecture Across2Sfg of Electrical2Sfg is
  quantity v across p to m;
begin
  output == v;
end architecture Across2Sfg;
```

```
architecture Through2Sfg of Electrical2Sfg is
  quantity v across i through p to m;
begin
  v == 0.0;
  output == i;
end architecture Through2Sfg;
```

Outline



- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion



Solvability Checks (1)

```
entity Vdc is
   generic (dc: REAL);
   port (terminal p, m: electrical);
end entity Vdc;
```

```
architecture Bad of Vdc is
   quantity v across p to m;
begin
   v == dc;
end architecture Bad;
```

~~OK~~

```
architecture Good of Vdc is
   quantity v across i through p to m;
begin
   v == dc;
end architecture Good;
```

OK

- A necessary condition for solvability is that there be as many equations as unknowns in the model
- In a VHDL-AMS design entity the number of equations must equal the number of through quantities, free quantities and interface quantities with mode `out`
- Each (scalar) simultaneous statement creates one equation
- In the example:
  - One equation is defined in both architectures
  - Only architecture Good has declaration for a through quantity
  - The language defines an implicit equation for each across quantity

Solvability Checks (2)

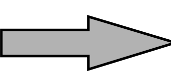
```
entity SfgAmp is
   generic (gain: REAL := REAL'high);
   port (quantity input: in REAL;
         quantity output: out REAL);
end entity SfgAmp;

architecture Ideal of SfgAmp is
begin
   if gain /= REAL'high use
      output == gain * input;
   else
      input == 0.0;    -- infinite gain
   end use;
end architecture Ideal;
```

OK

- There is one interface quantity with mode `out`
- For any value of gain there is one equation

VHDL AMS

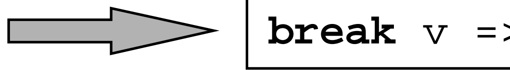
Outline

- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
-  Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

VHDL AMS

Initial Conditions (1)

- An initial condition specifies the value of a quantity at the beginning of a continuous interval
  - Beginning of a time domain simulation
  - After a discontinuity
- Initial conditions are specified with the break statement:



```
break v => 0.0, s => 10.0;
```

  - The initial condition for quantity v is 0.0, for quantity s, 10.0
- Initial conditions replace implicit equations while finding an analog solution point. An initial condition for Q replaces
  - the equation Q'Dot == 0 while finding the quiescent state
  - the equation Q == Q(t-) when re-initializing after discontinuity
- If an initial condition must be specified for a quantity Q whose derivative Q'Dot does not appear in the model, the user must specify which implicit equation to replace

VHDL AMS

Initial Conditions (2)

```
entity Capacitor is
  generic (C: REAL; ic: REAL := REAL'low);
  port (terminal p, m: electrical);
end entity Capacitor;
```

```
architecture One of Capacitor is
  quantity v across i through p to m;
begin
  i == C * v'dot;
  break v => ic when ic /= REAL'low;
end architecture One;
```

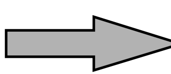
v == ic replaces v'dot == 0

```
architecture Two of Capacitor is
  quantity v across i through p to m;
  quantity q : charge;
begin
  q == c * v;
  i == q'dot;
  break for q use v => ic
                when ic /= REAL'low;
end architecture Two;
```

v == ic replaces q'dot == 0

Outline

- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor



- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

VHDL AMS

Implicit Quantities (1)

- Q'Dot
  - The derivative of quantity Q with respect to time
- Q'Integ
  - The integral of quantity Q over time from zero to current time
- Q'Slew(max_rising_slope, max_falling_slope)
  - Follows Q, but its derivative w.r.t. time is limited by the specified slopes. Default for max_falling_slope is max_rising_slope, default for max_rising_slope is infinity.
- Q'Delayed(T)
  - Quantity Q delayed by T (ideal delay, T >= 0)

VHDL AMS

Implicit Quantities (2)

- Q′Ltf(num, den)
  - Laplace transfer function whose input is Q
- Q′ZOH(T, initial_delay)
  - A sampled version of quantity Q (zero-order hold)
- Q′Ztf(num, den, T, initial_delay)
  - Z-domain transfer function whose input is Q
- S′Ramp(tr, tf)
  - A quantity that follows signal S, but with specified rise and fall times. Default for tf is tr, default for tr is 0.0
- S′Slew(max_rising_slope, max_falling_slope)
  - A quantity that follows signal S, but its derivative w.r.t. time is limited by the specified slopes.
    Default for max_falling_slope is max_rising_slope, default for max_rising_slope is infinity.

Outline

- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

VHDL AMS

Ideal Comparator VHDL-AMS Entity Declaration

```
entity Comparator is
  generic (vthresh: REAL);        -- threshold
  port (terminal ain, ref: electrical;
        signal dout: out BOOLEAN);
end entity Comparator;
```

- Keyword `signal` is optional but indicates intent:
  - Interface *terminals* `ain` and `ref`
  - Interface *signal* `dout`

VHDL AMS

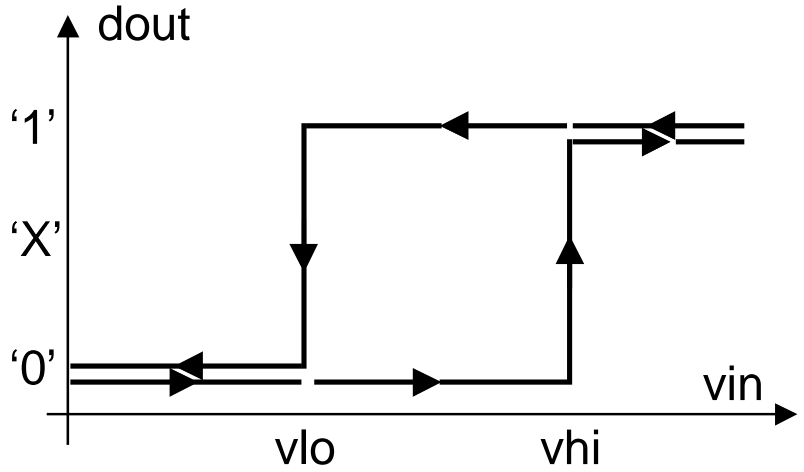
Ideal Comparator
VHDL-AMS Architecture Body

```
architecture Ideal of Comparator is
  quantity vin across ain to ref;
begin
  dout <= vin'above(vthresh);
end architecture Ideal;
```

- Threshold crossing detected with Q'Above(E), a boolean signal that
  - is FALSE when the value of quantity Q is below threshold E
  - is TRUE when the value of quantity Q is above threshold E
- Q must be a scalar quantity, E must be an expression of the same type as Q
- An event occurs on signal Q'Above(E) at the exact time of the threshold crossing
- A process can be sensitive to Q'Above(E), since it is a signal

Comparator with Hysteresis

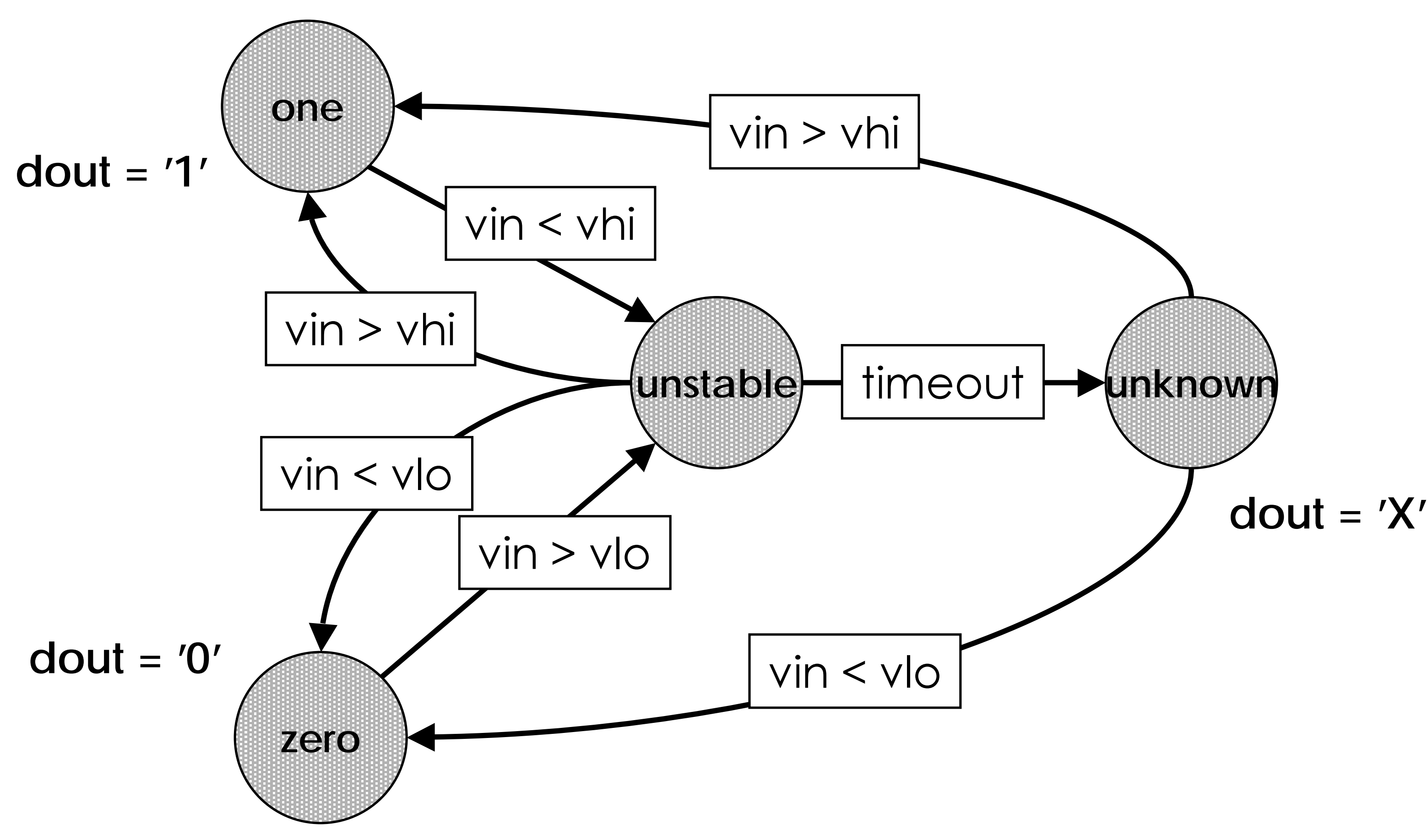
- Conversion of electrical quantity to std_logic signal
- Hysteresis



- dout becomes 'X' if vin stays in transition region for longer than the specified timeout

VHDL AMS

Comparator with Hysteresis State Diagram



VHDL AMS

Comparator with Hysteresis VHDL-AMS Declarations

```
library IEEE, Disciplines;
use IEEE.std_logic_1164.all;
use Disciplines.electrical_system.all;
entity ComparatorHyst is
   generic (vlo, vhi: REAL;   -- thresholds
            timeout: DELAY_LENGTH);
   port (terminal ain, ref: electrical;
         signal dout: out std_logic);
end entity ComparatorHyst;
```

```
architecture Hysteresis of ComparatorHyst is
   type states is (unknown, zero, one, unstable);
   quantity vin across ain to ref;
   function level(vin, vlo, vhi: REAL) return states is
   begin
      if    vin < vlo then return zero;
      elsif vin > vhi then return one;
      else                 return unknown;
      end if;
   end function level;
begin
   ...
```

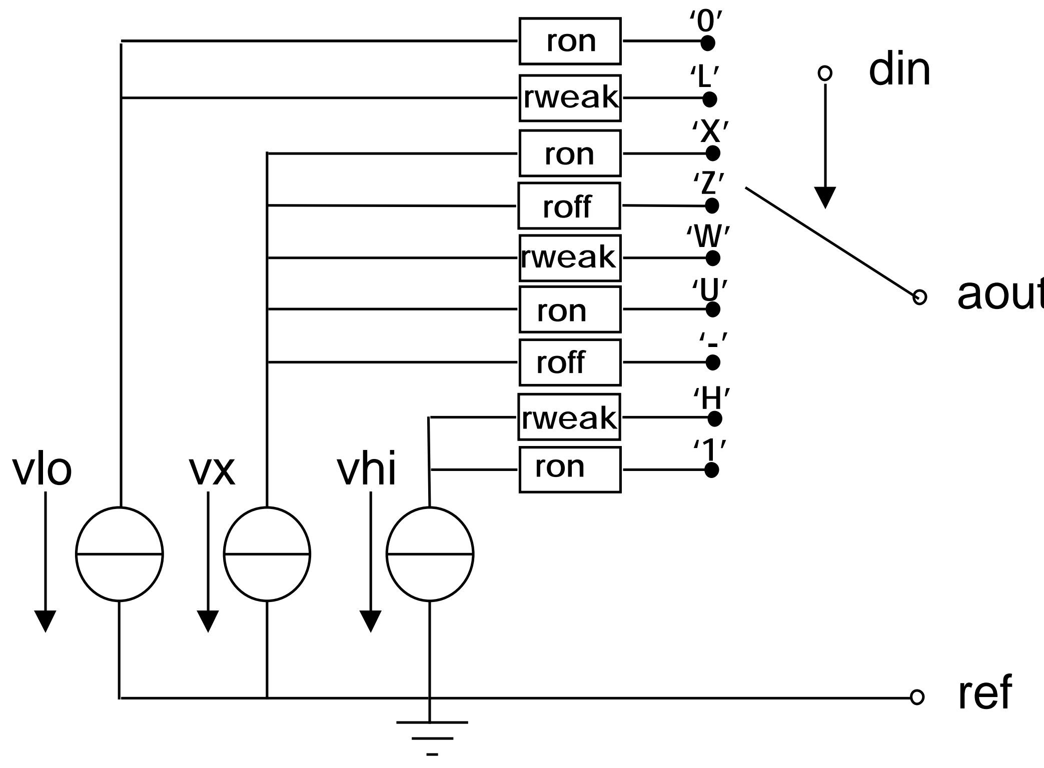
VHDL AMS

Comparator with Hysteresis
VHDL-AMS Process Implementing FSM

```
   ...
   process
      variable state: states := level(vin, vlo, vhi);
   begin
      case state is
         when one =>
            dout <= '1';
            wait on vin'Above(vhi);     -- wait for change
            state := unstable;
         when zero =>
            dout <= '0';
            wait on vin'Above(vlo);     -- wait for change
            state := unstable;
         when unknown =>
            dout <= 'X';
            wait on vin'Above(vhi), vin'Above(vlo);
            state := level(vin, vlo, vhi);
         when unstable =>
            wait on vin'Above(vhi), vin'Above(vlo) for timeout;
            state := level(vin, vlo, vhi);
      end case;
   end process;
end architecture Hysteresis;
```

D/A Converter

- Conversion of a std_logic signal to a voltage:
  - Signal-controlled voltage source with output resistance



VHDL AMS

D/A Converter
VHDL-AMS Entity Declaration

```
library ieee;
use ieee.std_logic_1164_pkg.all;
entity Dac is
   generic (vlo  : REAL := 0.2;     -- output voltage low
            vx   : REAL := 2.5;     -- output voltage unknown
            vhi  : REAL := 4.8;     -- output voltage high
            ron  : REAL := 0.1;     -- output resist. strong states
            rweak: REAL := 1.0e4;   -- output resist. weak states
            roff : REAL := 1.0e9;   -- output resist. high imp.
            tt   : REAL := 1.0e-9);-- transition time
   port (signal din: in std_logic;
         terminal aout, ref: electrical);
end entity Dac;
```

VHDL AMS

D/A Converter
VHDL-AMS Architecture Body

```
architecture Simple of Dac is
   type real_table is array(std_logic) of REAL;
   constant r_table: real_table :=
    (ron, ron, ron, ron, roff, rweak, rweak, rweak, roff);
   constant v_table: real_table :=
    (vx, vx, vlo, vhi, vx, vx, vlo, vhi, vx);
   quantity vout across iout through aout to ref;
   signal reff: REAL; -- effective output resistance
   signal veff: REAL; -- effective output voltage
begin
   reff <= r_table(din);
   veff <= v_table(din);
   vout == veff'ramp(tt) - iout * reff'ramp(tt);
end architecture Simple;
```

- Tables ordered according to type std_logic
- Output voltage and resistance ramp linearly from previous value

Outline

- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

VHDL AMS

VHDL-AMS Model Execution

- A VHDL-AMS model is the result of the elaboration of the design hierarchy
  - Digital part => set of processes + digital simulation kernel
  - Analog part => set of equations + analog solver
- Two phases
  - Determination of quiescient state of the model
    - Includes initialization phase and simulation cycles at time 0 ns
  - Simulation: time domain, small-signal frequency, or noise
    - For time domain simulation, time >= 0 ns
- Reduces to the VHDL 1076 initialization and simulation cycle if the model does not include any quantities
- Only the analog solver is executed after initialization if the model does not include any signals

VHDL-AMS Initialization

NOW := 0 sec

Set initial values for signals, quantities

Set Q'above(E) to Q>E

Run all processes

Find analog solution

Update signals

Run processes

Zero delay

yes

no

Update DOMAIN signal

Quiescent state

Time domain

yes

no

Run DOMAIN sensitive processes

Time domain

Frequency domain

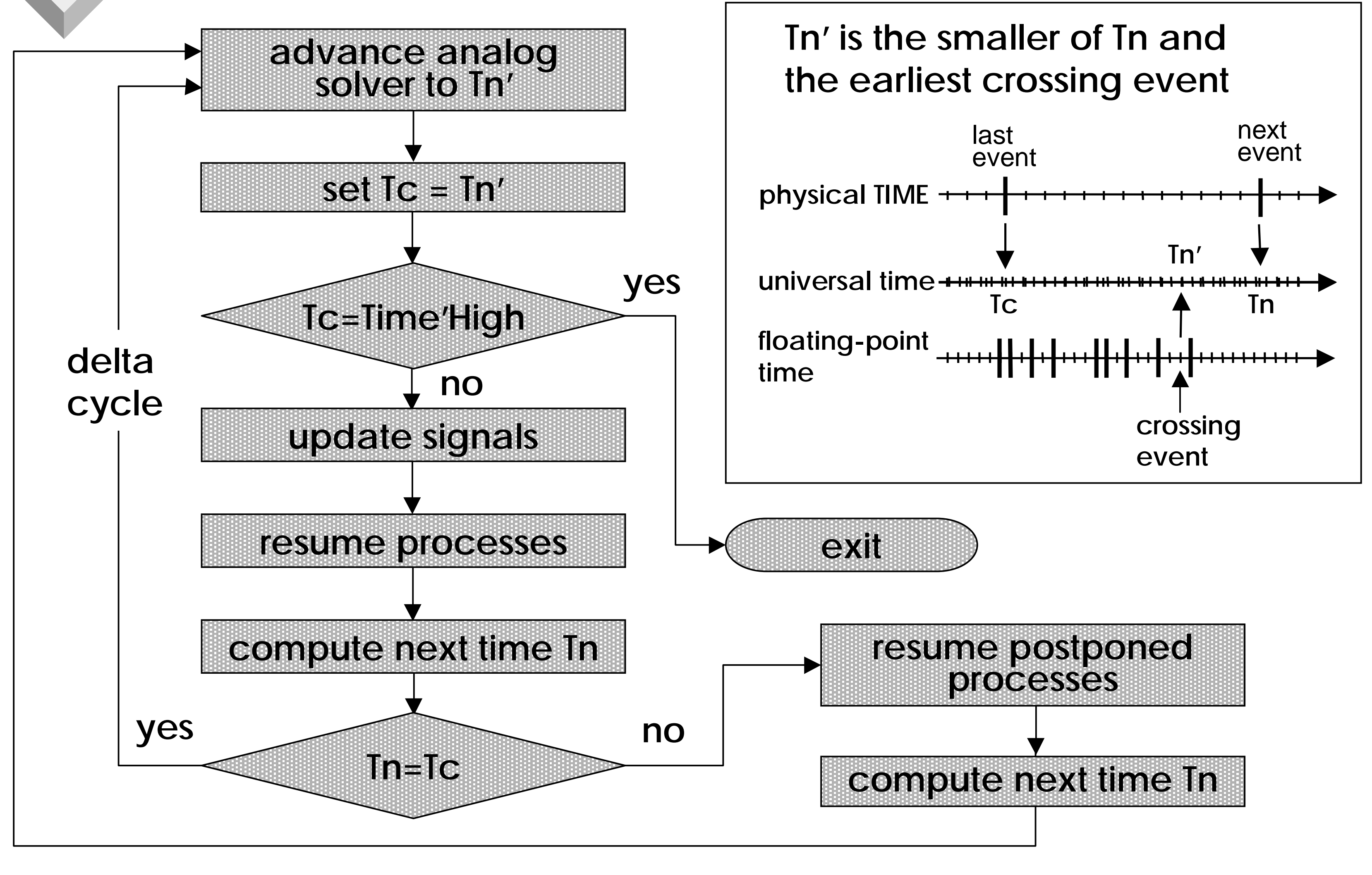
VHDL AMS

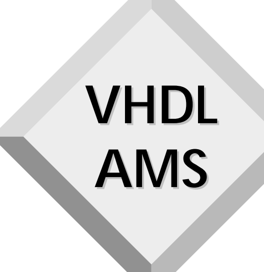
DOMAIN Signal

- New predefined signal DOMAIN of type DOMAIN_TYPE
  - Enumerated type with values QUIESCENT_DOMAIN, TIME_DOMAIN, FREQUENCY_DOMAIN
  - Set to QUIESCENT_DOMAIN during initialization
  - Set to TIME_DOMAIN or FREQUENCY_DOMAIN when the quiescent state has been reached (i.e., when there are no pending events at time 0), depending on whether a time domain or a frequency domain simulation follows
- DOMAIN can be used to write models that exhibit different behavior in different domains

VHDL AMS

VHDL-AMS Simulation Cycle
Time Domain Simulation





Outline

- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution

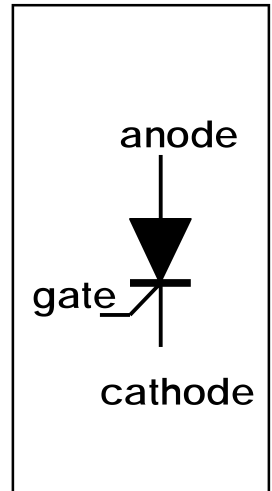
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

VHDL AMS

Silicon Controlled Rectifier VHDL-AMS Entity Declaration

```
library IEEE, Disciplines;
use IEEE.math_real.all;
use Disciplines.electrical_system.all;
entity Scr is
  generic (von   : voltage := 0.7;   -- Turn on voltage
           ihold : current := 0.0;   -- Holding current
           iss   : REAL := 1.0e-12);-- Saturation current
  port (terminal anode, cathode, gate: electrical);
end entity Scr;
```

- SCR turns on if voltage across SCR is positive and control voltage is larger than the on voltage `von`
- SCR turns off if control voltage is below the on voltage `von` and current falls below the holding current `ihold`



VHDL AMS

Silicon Controlled Rectifier VHDL-AMS Architecture Body

```
architecture Ideal of Scr is
   quantity vscr across iscr through anode to cathode;
   quantity vcntl across gate to cathode;
   signal ison: BOOLEAN;
   constant vt: REAL := 0.0258;   -- thermal voltage
begin
   process
      variable off: BOOLEAN := true;
   begin
      ison <= not off;
      case off is
         when true =>
            wait until vcntl'Above(von) and vscr'Above(0.0);
            off := false;
         when false =>
            wait until not (vcntl'Above(von) or iscr'Above(ihold));
            off := true;
      end case;
   end process;

   if ison use
      iscr == iss * (exp(vscr/vt) - 1.0);
   else
      iscr == 0.0;
   end use;
   break on ison;
end architecture Ideal;
```

Break Statement

- New concurrent statement
- Announces a discontinuity in the solution of the DAEs
  - In the example `iscr` changes discontinuously
- Analog solver must re-initialize for next continuous interval
- Break on event may include condition
- New initial conditions may be specified on quantities
- A VHDL-AMS model that causes a discontinuity on a quantity at some time T and does not execute a break statement at T is erroneous
  - Exception: discontinuities caused by using S'Ramp, S'Slew, Q'Slew, Q'ZOH, Q'Ztf

VHDL AMS

Voltage Limiter Revisited

```
architecture Good of VoltageLimiter is
  quantity vin across ip to im;
  quantity vout across iout through op to om;
begin
  if vin'Above(vlim) use
    vout == vlim;
  elsif not vin'Above(-vlim) use
    vout == -vlim;
  else
    vout == vin;
  end use;
  break on vin'Above(vlim), vin'Above(-vlim);
end architecture Good;
```

- Compare with implementation on slide 40
  - Relational expressions have been replaced by names of implicit signals detecting change of state
  - Break statement announces discontinuity in derivative

Bouncing Ball

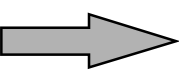
VHDL AMS

```
library Disciplines;
use Disciplines.mechanical_system.all;
entity BounceBall is
end entity BounceBall;

architecture Ideal of BounceBall is
  quantity v: velocity;           -- m/s
  quantity s: displacement;       -- m
  constant G: REAL := 9.81;       -- m/s**2
  constant Air_Res: REAL := 0.1; -- 1/m
begin
  -- Specify initial conditions
  break v => 0.0, s => 10.0;
  -- announce discontinuity and reset velocity value
  break v => -v when not s'above(0.0);
  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;
end architecture Ideal;
```

VHDL AMS

Outline

- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball



- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

VHDL AMS

Time-dependent Modeling
Sinusoid Voltage Source

```
library Disciplines, IEEE;
use Disciplines.electrical_system.all;
use IEEE.math_real.all;
entity Vsine is
   generic (ampl, freq: REAL);
   port (terminal p, m: electrical);
end entity Vsine;
```

```
architecture Sine of Vsine is
   quantity v across i through p to m;
   limit v: electrical'across with 1.0/(20.0*freq);
begin
   v == ampl * sin(math_2_pi*freq*NOW);
end architecture Sine;
```

Time-Dependent Modeling

- Predefined function NOW has been overloaded
  - NOW returning type TIME
  - NOW returning type REAL
- Analog solver computes the solution at certain times only
  - Time steps usually depend on tolerances
- For independent sources and free-running oscillators tolerances may not be sufficient to yield smooth waveform
- Step limit specification forces re-evaluation of listed quantities within interval specified by expression
  - Expression is evaluated after each analog solution point
  - Expression may depend on quantities and signals

VHDL AMS

Outline

- ♦ Introduction
- ♦ Brief Overview of VHDL-AMS
- ♦ Basic Concepts: DAEs
- ♦ Systems with Conservation Semantics: Diode
- ♦ Mixed Technology: Diode with Self Heating
- ♦ Piecewise Defined Behavior: Compressor,Voltage Limiter
- ♦ Procedural Modeling: Weighted Summer
- ♦ Signal-Flow Modeling: Adder-Integrator, Conversions
- ♦ Solvability: Voltage Source, Signal Flow Amplifier
- ♦ Initial Conditions: Capacitor
- ♦ Implicit Quantities
- ♦ Mixed-Signal Modeling: Comparators, D/A Converter
- ♦ VHDL-AMS Model Execution
- ♦ Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- ♦ Time-Dependent Modeling: Sinusoid Voltage Source
- ➡ ♦ Frequency Domain Modeling: Current Source, Filter
- ♦ Noise Modeling: Resistor, Diode
- ♦ Conclusion

VHDL AMS

Frequency Domain Modeling

- Frequency domain simulation based on small-signal model
  - Obtained by linearizing equations about quiescent point
- Spectral source quantities allow a user to specify stimulus in the frequency domain
  - Magnitude and phase
  - Can be frequency dependent
    - Predefined function FREQUENCY can be called in the declaration of source quantities only
  - Value of spectral source quantity is 0.0 except during frequency domain simulation
- Laplace and z-domain transfer functions can be used to describe the behavior of abstract filters
- No support for more general frequency domain modeling because language scope is restricted to lumped systems

VHDL AMS

Current Source With AC Spectrum

```
library Disciplines, IEEE;
use Disciplines.electrical_system.all;
use IEEE.math_real.all;
entity Isine is
   generic (ampl, freq: REAL;
            mag, phase: REAL := 0.0);
   port (terminal p, m: electrical);
end entity Isine;
```

```
architecture Sine of Isine is
   quantity i through p to m;
   quantity ac: REAL spectrum mag, phase;
   limit i: electrical'through with 1.0/(20.0*freq);
begin
   i == ampl * sin(math_2_pi*freq*NOW) + ac;
end architecture Sine;
```

Second Order Lowpass Filter

- Behavior specified by pole frequency, pole Q and gain

```
library Disciplines, IEEE;
use Disciplines.electrical_system.all;
use IEEE.math_real.all;
entity Lowpass2 is
  generic (fp, qp: REAL;
           gain: REAL := 1.0);
  port (terminal input, output, ref: electrical);
end entity Lowpass2;
```

```
architecture One of Lowpass2 is
  quantity vin across input to ref;
  quantity vout across iout through output to ref;
  constant wp : REAL := math_2_pi*fp;
  constant num: REAL_VECTOR := (0 => gain*wp*wp);
  constant den: REAL_VECTOR := (wp*wp, wp/qp, 1.0);
begin
  vout == vin'Ltf(num, den);
end architecture One;
```

VHDL AMS

Outline

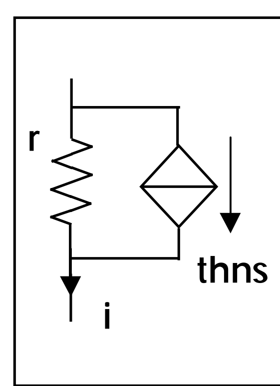
- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

Noise Modeling

- Support for noise modeling in the frequency domain
- Noise source quantities allow a user to specify a noise spectrum
  - Power spectrum
  - Can depend on frequency by calling predefined function FREQUENCY in the definition of the noise spectrum
  - Can depend on operating point by including quantity names in the definition of the noise spectrum
  - Value of noise source quantity is 0.0 except during noise simulation

Resistor Model with Thermal Noise

```
library Disciplines;
use Disciplines.electrical_system.all;
use Disciplines.environment.all;
entity Resistor is
   generic (r: REAL);
   port (terminal p, m: electrical);
end entity Resistor;
```




```
architecture Noisy of Resistor is
   quantity v across i through p to m;
   quantity thns : REAL noise 4.0*ambient_temp*boltzmann/r;
begin
   assert r /= 0.0;
   i == v / r + thns;
end architecture Noisy;
```

- Resistor current is sum of ohmic current and thermal noise current represented by noise source quantity **`thns`**

Diode with Flicker Noise

```
architecture Noisy of Diode is
   quantity vd across id, ic through p to m;
   quantity qc: REAL;
   quantity flns: REAL noise kf * id**af / FREQUENCY;
   constant vt: REAL := 0.0258;   -- thermal voltage
begin
   id == iss * (exp((vd-rs*id)/(n*vt)) - 1.0) + flns;
   qc == tt*id - 2.0*cj0 * sqrt(vj**2 - vj*vd);
   ic == qc'dot;
end architecture Noisy;
```

- Flicker noise current represented by noise source quantity **`flns`** depends on quiescent state diode current and is inversely proportional to the simulation frequency

VHDL AMS

Outline

- Introduction
- Brief Overview of VHDL-AMS
- Basic Concepts: DAEs
- Systems with Conservation Semantics: Diode
- Mixed Technology: Diode with Self Heating
- Piecewise Defined Behavior: Compressor,Voltage Limiter
- Procedural Modeling: Weighted Summer
- Signal-Flow Modeling: Adder-Integrator, Conversions
- Solvability: Voltage Source, Signal Flow Amplifier
- Initial Conditions: Capacitor
- Implicit Quantities
- Mixed-Signal Modeling: Comparators, D/A Converter
- VHDL-AMS Model Execution
- Discontinuities: SCR, Voltage Limiter, Bouncing Ball
- Time-Dependent Modeling: Sinusoid Voltage Source
- Frequency Domain Modeling: Current Source, Filter
- Noise Modeling: Resistor, Diode
- Conclusion

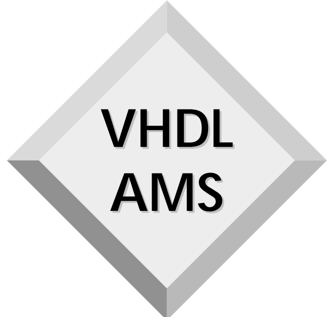
VHDL AMS

Conclusion

- VHDL 1076.1 extends VHDL 1076 to the continuous domain
  - Keeps VHDL 1076 fundamental philosophy
  - Adds support for continuous and mixed continuous/discrete behavior
  - Builds on solid mathematical foundations
- VHDL-AMS is equally applicable to electrical and non-electrical domains
  - Mixed discipline
  - Control systems
- Two separate but related standards
  - IEEE Std. 1076-1993 for digital (event-driven) applications
  - IEEE Std. 1076.1-1999 for digital AND mixed-signal applications

Additional Information

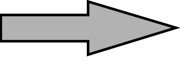
- 1076.1 Working Group
  - Reporting to Design Automation Standards Committee (DASC) of IEEE Computer Society
- Email reflector vhdl-ams@eda.org
  - All requests to owner-vhdl-ams-request@eda.org
- Web site at http://www.eda.org/vhdl-ams/



Part II:

VHDL-AMS
in Practical Applications

Outline



- VHDL-AMS Modeling Guidelines
- VHDL-AMS Modeling Techniques
  - IC Applications
- Modeling at Different Levels of Abstraction
  - Telecom Applications
- Modeling of Multi-Disciplinary Systems
  - Automotive Applications
- MEMS Modeling Using the VHDL-AMS Language

VHDL
AMS

Overview of VHDL-AMS Modeling Guidelines

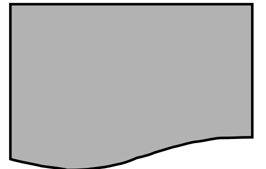
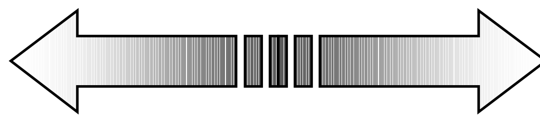
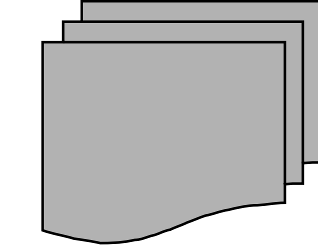
- Package Architecture
- Types and Subtypes
- Natures and Subnatures
- Physical and Mathematical Constants
- Simulation Control

VHDL-AMS Modeling Utilities

- VHDL-AMS modeling requires common set of utilities and styles
  - Avoid needless duplication
  - Promote interoperation
  - Establish coordinated family of dialects for specialized modeling applications
- Initial group effort has concentrated on data modeling
  - Behavioral modeling will follow
- Common Modeling Utilities
  - Packages
  - Types/Subtypes
  - Natures/Subnatures
  - Physical/Mathematical Constants
  - Simulation Control

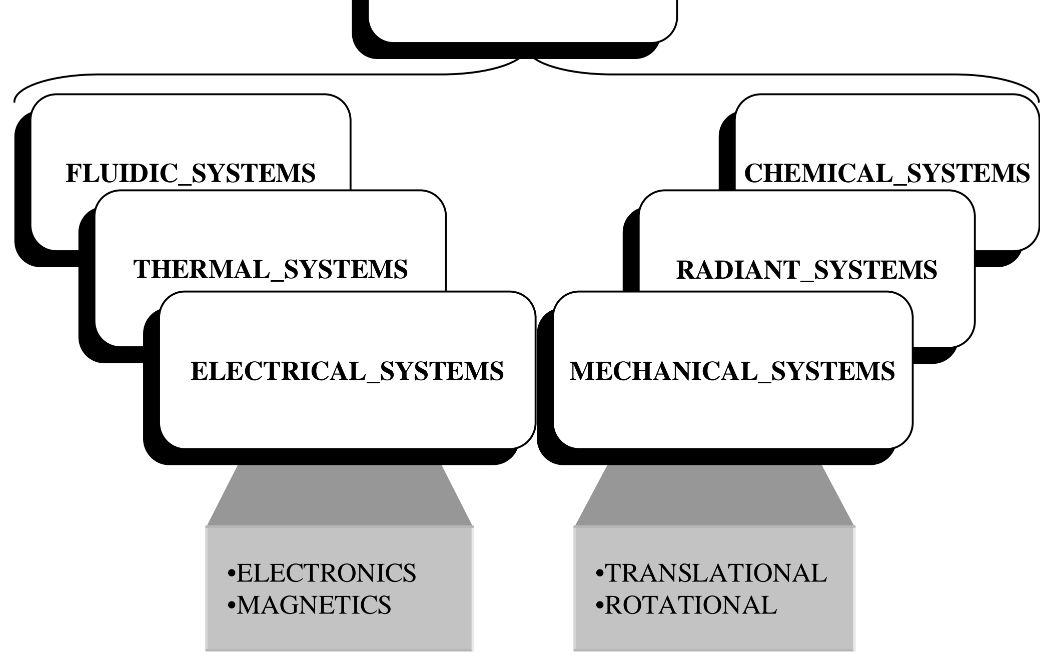
Packages

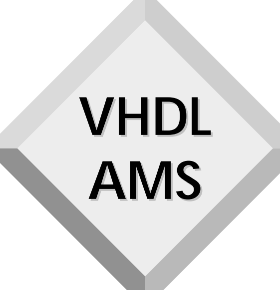
- Domain abstractions implemented as package(s)
- Packages provide convenient set of abstractions and operations to compose models
  - Quickly
  - Consistently
  - Limited domain expertise
- Issue: What level of package aggregation/segmentation?

Package Architecture

- Proposal: Two-tier package hierarchy
  - Single package containing declarations common across energy domains
  - Collection of packages - one package per energy domain
- Segmentation
  - Allows separate communities of interest per discipline
- Aggregation
  - Allows variety of composite system modeling by "mixing-and-matching"





MEMS VHDL-AMS Modeling

```
library IEEE;
use IEEE.MATH_REAL.all;
use IEEE.ELECTRICAL_SYSTEMS.all;
use IEEE.MECHANICAL_SYSTEMS.all;
entity LOUDSPEAKER is
  ....................
port (
  terminal PLUS, MINUS : ELECTRICAL;
  terminal CONE        : TRANSLATIONAL);
end entity LOUDSPEAKER;
```

Add mathematical constants/functions
- MATH_2_PI = 2π

Add electrical domain
- ELECTRICAL
- FLUX_DENSITY

Add mechanical domain
- TRANSLATIONAL
- MASS
- STIFFNESS
- DAMPING

Across/Through Quantities

Energy Domain	Across Quantity	Through Quantity
Electrical	Voltage	Current
Magnetic	MMF	Magnetic Flux
Translational	Displacement	Force
Rotational	Angle	Torque
Fluidic	Pressure	Flow Rate
Thermal	Temperature	Heat Flux

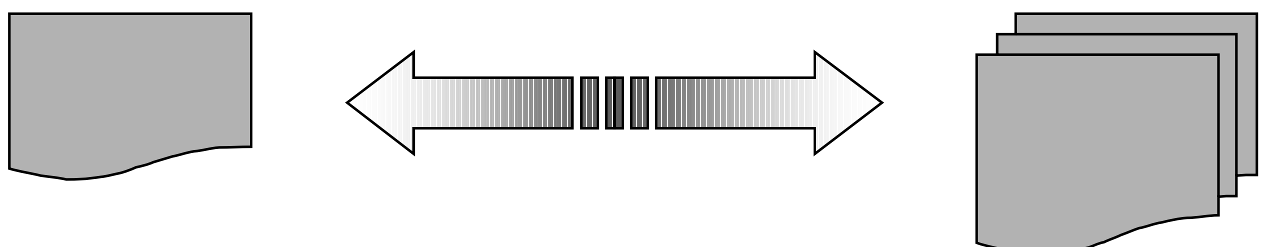
```
nature ELECTRICAL is
    VOLTAGE across
    CURRENT through
    ELECTRICAL_REF reference;

nature MAGNETIC is
    MMF across
    FLUX through
    MAGNETIC_REF reference;

nature TRANSLATIONAL is
    DISPLACEMENT across
    FORCE through
    TRANSLATIONAL_REF reference;

nature ROTATIONAL is
    ANGLE across
    TORQUE through
    ROTATIONAL_ref reference;

nature FLUIDIC is
    PRESSURE across
    FLOW_RATE through
    FLUIDIC_REF reference;

nature THERMAL is
    TEMPERATURE across
    HEAT_FLUX through
    THERMAL_REF reference;
```

VHDL-AMS Types and Subtypes

- Natures define types for across and through branch quantities
  - **VOLTAGE, CURRENT, DISPLACEMENT, VELOCITY, FORCE,** and **ANGLE** - examples of types
- Strong data typing is common aspect of complex software programming and digital system modeling
- Strong data typing (predefined and user defined) is a major aspect of VHDL
- VHDL-AMS limits use of strong data typing - emphasizes floating point types
  - Model analytic continuous functions of time
  - Support practical implementation issues of analog solver

VHDL-AMS Types and Subtypes

- Quantities must be of a floating point type
  - Across and Through types in nature declarations must be of a floating point type
- Issue: What level of floating point types/subtype aggregation and/or segmentation?



- Proposal: Do not impose strong data typing
  - Floating point types are declared as subtypes of single parent type **`REAL`**

VHDL-AMS Types and Subtypes

Sample common declarations

```
-- electrical_systems
subtype VOLTAGE is REAL tolerance "DEFAULT_VOLTAGE";
subtype CURRENT is REAL tolerance "DEFAULT_CURRENT";
subtype CHARGE is REAL tolerance "DEFAULT_CHARGE";

-- mechanical_systems
subtype DISPLACEMENT is REAL tolerance "DEFAULT_DISPLACEMENT";
subtype FORCE is REAL tolerance "DEFAULT_FORCE";
subtype VELOCITY is REAL tolerance "DEFAULT_VELOCITY";
subtype MASS is REAL tolerance "DEFAULT_MASS";
subtype STIFFNESS is REAL tolerance "DEFAULT_STIFFNESS";
subtype DAMPING is REAL tolerance "DEFAULT_DAMPING";

-- fluidic_systems
subtype PRESSURE is REAL tolerance "DEFAULT_PRESSURE";
```

VHDL AMS

Package: ENERGY_SYSTEMS

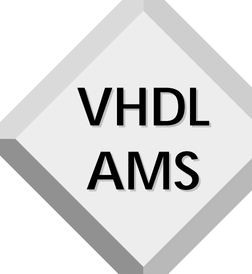
- Physical and mathematical constants often used in modeling coupled-energy systems are defined in package **ENERGY_SYSTEMS**

```
package ENERGY_SYSTEMS is
  -- common scaling factors
  constant  PICO  : REAL := 1.0e-12;
  constant  NANO  : REAL := 1.0e-9;
  constant  MICRO : REAL := 1.0e-6;
  constant  MILLI : REAL := 1.0e-3;
                ●●●
  constant  KILO  : REAL := 1.0e+3;
  constant  MEGA  : REAL := 1.0e+6;
  constant  GIGA  : REAL := 1.0e+9;

  -- permittivity of vacuum <FARADS/METER>
  constant EPS0 : REAL := 8.8542*PICO;

  -- permeability of vacuum <HENRIES/METER>
  constant MU0 : REAL := 4.0e-6 * MATH_PI;

  -- electron charge <COULOMB>
  constant Q : REAL :=  1.60218e-19;

  -- acceleration due to gravity <METERS/SQ_SEC>
  constant GRAV : REAL := 9.81;
end package ENERGY_SYSTEMS;
```

Package: - ELECTRICAL_SYSTEMS

- Defines commonly used quantity types and defining nature for electrical domain

```
library IEEE;
use IEEE.ENERGY_SYSTEMS.all;
package ELECTRICAL_SYSTEMS is
  -- subtype declarations
  subtype VOLTAGE is REAL tolerance "DEFAULT_VOLTAGE";
  subtype CURRENT is REAL tolerance "DEFAULT_CURRENT";
  subtype CHARGE is REAL tolerance "DEFAULT_CHARGE";
  subtype RESISTANCE is REAL tolerance "DEFAULT_RESISTANCE";
  subtype CAPACITANCE is REAL tolerance "DEFAULT_CAPACITANCE";

  -- nature declarations
  nature ELECTRICAL is VOLTAGE across CURRENT through
                                     ELECTRICAL_REF reference;

  -- alias declarations
  alias GROUND is ELECTRICAL_REF;
end package ELECTRICAL_SYSTEMS;
```



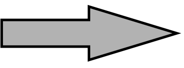
Package: MECHANICAL_SYSTEMS

- Defines commonly used quantity types and defining nature for mechanical domain

```
library IEEE;
use IEEE.ENERGY_SYSTEMS.all;
package MECHANICAL_SYSTEMS is
  -- subtype declarations
  subtype DISPLACEMENT is REAL tolerance "DEFAULT_DISPLACEMENT";
  subtype FORCE is REAL tolerance "DEFAULT_FORCE";
  subtype VELOCITY is REAL tolerance "DEFAULT_VELOCITY";
  subtype ACCELERATION is REAL tolerance "DEFAULT_ACCELERATION";
  subtype MASS is REAL tolerance "DEFAULT_MASS";
  subtype STIFFNESS is REAL tolerance "DEFAULT_STIFFNESS";
  subtype DAMPING is REAL tolerance "DEFAULT_DAMPING";

  -- nature declarations
  nature TRANSLATIONAL is DISPLACEMENT across FORCE through
                                       TRANSLATIONAL_REF reference;

  -- alias declarations
  alias ANCHOR is TRANSLATIONAL_REF;
end package MECHANICAL_SYSTEMS;
```

Outline

- VHDL-AMS Modeling Guidelines



- VHDL-AMS Modeling Techniques
  - IC Applications
- Modeling at Different Levels of Abstraction
  - Telecom Applications
- Modeling of Multi-Disciplinary Systems
  - Automotive Applications
- MEMS Modeling Using the VHDL-AMS Language

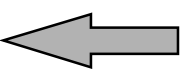
Overview

- Technology and geometry parameters
  - Sharing of technology parameters
- Ambient temperature
  - Propagation through design hierarchy
- Global nets
  - Power distribution
  - Chassis ground

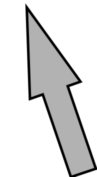
Technology and Geometry Parameters

- Semiconductors on a chip vary in geometry, but share technology parameters
- Want to share technology parameters among different instances of a device
- In SPICE:

```
* technology parameters specified with .MODEL card
.MODEL d d1234  is 1e-13 rs 10
…
d1 34 57 d1234
d2 23 45 d1234 1.5
```



two instances, one with default area, the other with explicit area



same technology parameters for both instances

Sharing of Technology Parameters

- Represent the technology parameters as a generic of a record type
- For the diode we had:

```
generic (iss: REAL := 1.0e-14;
         n, af: REAL := 1.0;
         tt, cj0, vj, rs, kf: REAL := 0.0);
```

- Equivalent record type declaration

```
type DiodeModel is record
   iss, n, rs, tt, cj0, vj, af, kf: REAL;
end record;
```

  - Declared in package **diode_pkg** for re-use

VHDL AMS

Revised Diode Architecture

```
architecture Noisy of Diode2 is
  quantity vd across id, ic through p to m;
  quantity qc: REAL;
  quantity flns: REAL noise model.kf*id**model.af / FREQUENCY;
  constant vt: REAL := boltzmann * ambient_temp / elec_charge;
begin
  id == area * model.iss * (exp((vd-model.rs*id)/(model.n*vt))
                            - 1.0) + flns;
  qc == model.tt*id - 2.0*model.cj0 *
                            sqrt(model.vj**2 - model.vj*vd);
  ic == qc'dot;
end architecture Noisy;
```

Handling of Defaults

- Declare a constructor function for an object of the record type
  - Function parameters are technology parameters with defaults
  - Return value has record type

```
function DiodeModelValue(
       iss: REAL := 1.0e-14;
       n, af: REAL := 1.0;
       tt, cj0, vj, rs, kf: REAL := 0.0) return DiodeModel is
begin
  return DiodeModel'(iss, n, rs, tt, cj0, vj, af, kf);
end function DiodeModelValue;
```

parameters ordered according to record type

Revised Diode Entity

```
library IEEE, Disciplines;
use Disciplines.electrical_system.all;
use Disciplines.environment.all;
use IEEE.math_real.all;
use work.diode_pkg.all;
entity Diode2 is
   generic (model: DiodeModel := DiodeModelValue;
            area: REAL := 1.0);
   port (terminal anode, cathode: electrical);
end entity Diode2;
```

- Generic `model` now has record type with default

VHDL AMS

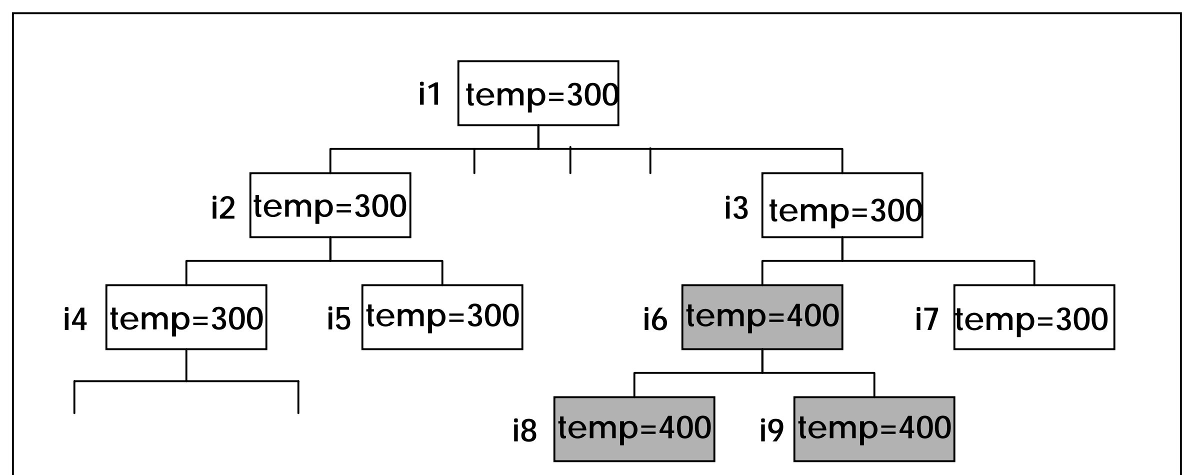
Use in Netlist

```
use work.diode_pkg.all;

...

terminal n23, n34, n45, n57, ... : electrical;

constant d1234: DiodeModel :=
       DiodeModelValue(iss => 1.0e-13, rs => 10.0);

...

d1: entity Diode2 generic map (model => d1234)
            port map (anode => n34, cathode => n57);
d2: entity Diode2 generic map (model => d1234, area => 1.5)
            port map (anode => n23, cathode => n45);

...

d12: entity Diode2 generic map
              (model => DiodeModelValue(iss => 1.0e-15))
            port map (anode => ..., cathode => ...);
```

new functionality

Ambient Temperature: The Problem

- Most physics based models are temperature dependent
- For many applications dependency on ambient temperature is sufficient (no self heating)needed)
- Ambient temperature is the same in all or most instances of a design hierarchy



Ambient Temperature: The Solution

- Each entity has a generic named `temp` of type `REAL`
  - Its initial value is the ambient temperature from package environment
- Ambient temperature for a subtree of the design hierarchy is defined at the root of the subtree
- Temperature is passed through subtree by associating the generic `temp` of an instance as an actual with the formal `temp` of each subinstance

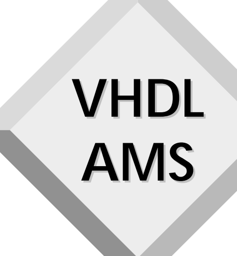
VHDL AMS

Ambient Temperature: Example

```
library Disciplines;
use Disciplines.environment.all;
entity e3 is
   generic (...;
            temp: REAL := ambient_temp);
   port (...);
end entity e3;
```

```
architecture one of e3 is
   component e6 is
      generic (...; temp: REAL); port (...);
   end component e6;
   component e7 is
      generic (...; temp: REAL); port (...);
   end component e7;
begin
   i6: e6 generic map (..., temp => 400.0) port map (...);
   i7: e7 generic map (..., temp => temp) port map (...);
end architecture one;
```

Defines temperature in subtree rooted at i6
Propagates temperature to subtree rooted at i7



Global Nets

- Power distribution and ground nets are typically not drawn in a schematic
  - These nets are global nets
- A global nets can be represented in VHDL-AMS as a terminal declared in a package
- A VHDL-AMS model that makes this terminal visible does not need a port for power or ground
  - VHDL-AMS model corresponds exactly to schematic
- Properties of global net must be defined exactly once, typically in the root instance of a design hierarchy
  - Voltage/impedance for power distribution nets
  - Impedance to electrical reference for ground nets

Global Nets: Example

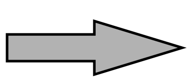
```
library Disciplines;
use Disciplines.electrical_system.all;
package PowerDistribution is
  terminal vcc, vee, chassis: electrical;
end package PowerDistribution;
```

```
library Disciplines;
use Disciplines.electrical_system.all;
use work.PowerDistribution.all;
entity Inverter is
  port (terminal inp, outp: electrical);
end entity Inverter;
```

only functional terminals declared

```
architecture one of Inverter is
  constant qnpn: BjtModel := BjtModelValue(kind => npn);
begin
  r1: entity resistor generic map (r => 1.0e3)
          port map (p => vcc, m => outp);
  t1: entity bjt generic map (model => qnpn)
          port map (collector => outp, base => inp,
                        emitter => chassis);
end architecture one;
```

Global Nets: Test Bench

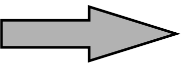


```
library Disciplines;
use Disciplines.electrical_system.all;
use work.PowerDistribution.all;
entity testbench is
end entity testbench;
```

```
architecture example of testbench is
   ...
begin
   -- define properties of global nets
   vvcc: entity Vdc generic map (dc => 5.0)
            port map (p => vcc, m => ground);
   vvee: entity Vdc generic map (dc => 3.0)
            port map (p => vee, m => ground);
   rgnd: entity Resistor generic map (r => 10.0)
            port map (p => chassis, m => ground);
   cgnd: entity Capacitor generic map (c => 30.0e-12)
            port map (p => chassis, m => ground);
   ...
   inv1: entity Inverter port map (inp => ..., outp => ...);
end architecture example;
```

Outline

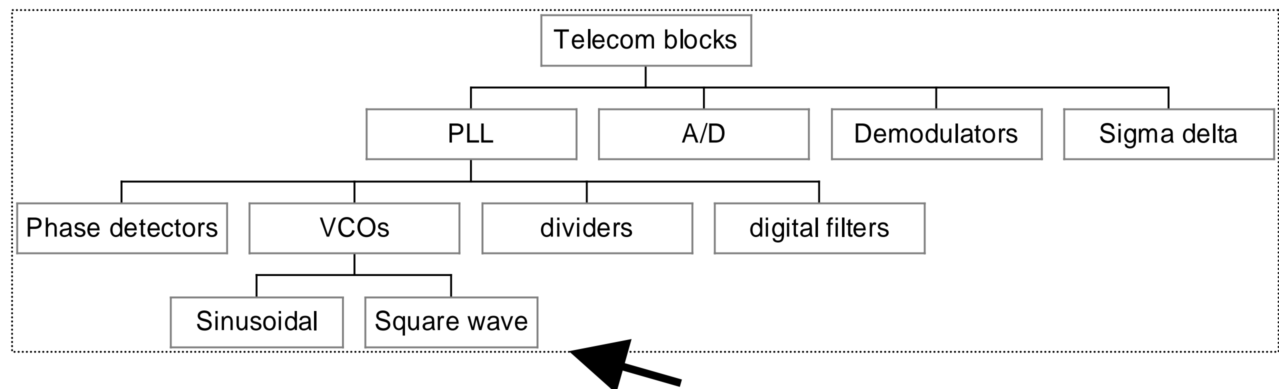
- VHDL-AMS Modeling Guidelines
- VHDL-AMS Modeling Techniques
  - IC Applications



- Modeling at Different Levels of Abstraction
  - Telecom Applications
- Modeling of Multi-Disciplinary Systems
  - Automotive Applications
- MEMS Modeling Using the VHDL-AMS Language

A VCO from a Telecom Library

- Why does the world need analog behavioral libraries?
  - Behavioral simulation reduces design cycle time
  - Creating verified, flexible, reusable models is time consuming
  - "Black box" models are not good enough
- Standard analog HDLs are the enabling technology
- The VCO is designed to be a member of a large, coordinated model set



The VCO in its Native Element

VCO_op

FM Modulator

Phase Detector

Loop Filter

VCO

Output Filter

info

VCO SIN

PLL_in

in1

in2

FQM PD

PD_op1

PD_op2

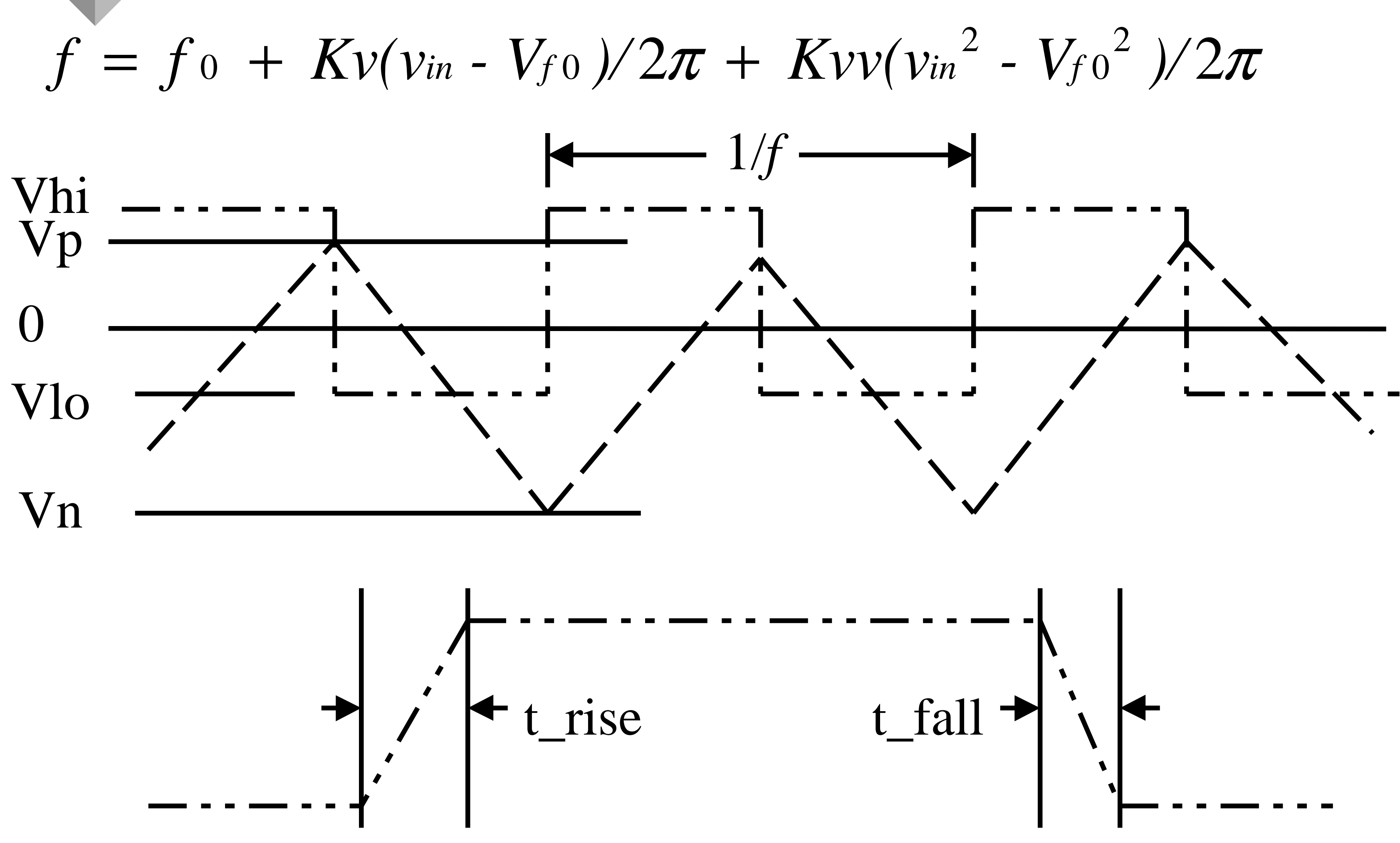
VCO SQR

outp

FM Demodulator using the PLL Model Set

Parameters of the VCO

$$f = f_0 + Kv(v_{in} - V_{f0})/2\pi + Kvv(v_{in}^2 - V_{f0}^2)/2\pi$$



Theory of Operation

$$v_{out2} = \int_0^t A_{tot}\, dt$$ the triangular wave

Where:

$$A_{tot} =$$ the instantaneous slope of the triangular wave

When *vout1* = *Vhi*:

$$k_1(v_{in} - V_{fo}) + k_{11}(v_{in}^2 - V_{fo}^2) + 2 f_o(V_p - V_n)$$

When *vout1* = *Vlo*:

$$-\left[k_1(v_{in} - V_{fo}) + k_{11}(v_{in}^2 - V_{fo}^2) + 2 f_o(V_p - V_n)\right]$$

and:

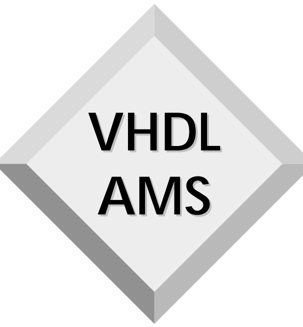
$$k_1 = K_v(V_p - V_n)/\pi, \quad k_{11} = K_{vv}(V_p - V_n)/\pi$$

The Model

```
library IEEE, Disciplines;
use Disciplines.ElectroMagnetic_system.all, IEEE.math_real.all;

entity vco_sqr_tri_1_1 is
  generic (
    Vp          : real := 1.0;          -- High level of triangular waveform
    Vn          : real := -1.0;         -- Low level of triangular wavefor
    Vhi         : real := 5.0;          -- Output high level
    Vlo         : real := 0.0;          -- Output low
    V_f0        : real := 1.0;          -- Input voltage corresponding to f0
    f0          : real := 100.0e3;      -- Output center frequency
    fmin        : real := 10.0e3;       -- Minimum allowable output frequency
    fmax        : real := 200.0e3;      -- Maximum allowable output frequency
    kv          : real := 100.0e3;      -- VCO linear gain  rad/s/V
    kvv         : real := 100.0e1;      -- VCO gain quadratic  rad/s/V^2 (arch. a2 only)
    PHI         : real := 0.0;          -- Initial phase shift of the output
    t_rise      : real := 1.0e-9;       -- Rise time of the output
    t_fall      : real := 1.0e-9;       -- Fall time of the output
    R_ip        : real := 100.0e3;      -- Shunt input resistance
    C_ip        : real := 0.1e-12;      -- Shunt input capacitance (arch. a2 only)
    R_op        : real := 1.0e3;        -- Series output resistance
    C_op        : real := 0.1e-12 );    -- Shunt output capacitance (arch. a2 only)
  port (
    terminal  p1, p2, p3, gnd  : electrical );
   .
   .
   .
```

Check the Generics!

```
    .
    .
    .
begin
  assert Vp>Vn           report "Vp must be > Vn.";
  assert Vhi>Vlo         report "Vhi must be > Vlo.";
  assert f0>0.0          report "f0 must be > zero.";
  assert Kv>0.0          report "Kv must be > zero.";
  assert Kvv>0.0         report "KVV must be > zero.";
  assert fmax>fmin       report "fmax must be > fmin.";
  assert fmax>=f0        report "fmax must be >= f0.";
  assert fmin<=f0        report "fmin must be <= f0.";
  assert PHI<=180.0 and PHI>=-180.0
                         report "PHI must > -180 and < 180 degrees.";
  assert t_rise>1.0e-12 report "t_rise should be > 1 ps." severity warning;
  assert t_fall>1.0e-12 report "t_fall should be > 1 ps." severity warning;
  assert R_ip>0.0        report "R_ip must be > zero.";
  assert R_op>0.0        report "R_op must be > zero.";
  assert C_ip>=0.0       report "C_ip should be >= zero." severity warning;
  assert C_op>=0.0       report "C_op should be >= zero." severity warning;
end entity vco_sqr_tri_1_1;
```



1st Architecture: Constants and Quantities

```
architecture a1 of vco_sqr_tri_1_1 is
  function ifelse (a: boolean; b,c:real) return real is begin
    if a then return b; else return c; end if;
  end;
  constant k1     : real := Kv*(Vp-Vn)/math_pi;                -- contrib to slope for 1 v in
  constant V1     : real := V_f0 + 2.0*math_pi*(fmax-f0)/Kv;  -- Input corresponding to fmax
  constant V2     : real := V_f0 - 2.0*math_pi*(f0-fmin)/Kv;  -- Input corresponding to fmin
  constant v_init2: real := Vp - (abs(PHI)/180.0)*(Vp-Vn);           -- initial value of vout2
  constant v_init1: real := ifelse(phi>=0.0 and phi<180.0, vlo, vhi);-- initial val of vout1
  constant dc     : real := (vhi+vlo)/2.0;                           -- dc of square wave

  quantity vin across iin through p1 to gnd;  -- input branch
  quantity vin_lim: real;                     -- limited version of input
  quantity T_vout1 across i_out1 through p2;  -- true output branch
  quantity T_vout3 across i_out3 through p3;  -- triangel wave output
  quantity vout1: real := v_init1;            -- square wave output
  signal   svout1: real := v_init1;           -- discrete form of vout1
  quantity vout2     : real := v_init2;       -- tri output
  quantity Atot      : real;                  -- integrand of tri generator
begin
   .
   .
   .
```

The Schmitt Trigger

```
    .
    .
    .
begin
 schmitt: process
    variable low: boolean := v_init1 < dc;
  begin
    if low then
      wait until not vout2'above(Vn);
      svout1 <= vhi;
    else
      wait until vout2'above(Vp);
      svout1 <= vlo;
    end if;
    low := not low;
  end process;
    .
    .
    .
```

VHDL AMS

The Equations

```
    .
    .
    .
  -- input load
  iin == vin / R_ip;
  -- input limiter
  If vin'above(V1) then
    Vin_lim == V1;
  elsif vin'above(V2) then
    Vin_lim == vin;
  else
    Vin_lim == V2;
  end if;
  -- limiter induced discontinuitites
  break on vin'above(V1), vin'above(V2);

  vout1  == svout1'ramp(t_rise, t_fall);
  Atot   == sign(vout1-dc)* (k1*(Vin_lim-V_f0) + 2.0*f0*(Vp-Vn));
  vout2  == Atot'integ + v_init2;
  i_out1 == (T_vout1 - vout1) / R_op;
  i_out3 == (T_vout3 - vout2) / R_op;
end architecture a1;
```

VHDL AMS

Adding Second Order Effects

```
library IEEE, Disciplines;
use Disciplines.ElectroMagnetic_system.all, IEEE.math_real.all;

entity vco_sqr_tri_1_1 is
  generic (
    Vp         : real := 1.0;          -- High level of triangular waveform
    Vn         : real := -1.0;         -- Low level of triangular wavefor
    Vhi        : real := 5.0;          -- Output high level
    Vlo        : real := 0.0;          -- Output low
    V_f0       : real := 1.0;          -- Input voltage corresponding to f0
    f0         : real := 100.0e3;      -- Output center frequency
    fmin       : real := 10.0e3;       -- Minimum allowable output frequency
    fmax       : real := 200.0e3;      -- Maximum allowable output frequency
    kv         : real := 100.0e3;      -- VCO linear gain  rad/s/V
    kvv        : real := 100.0e1;      -- VCO gain quadratic  rad/s/V^2 (arch a2 only)
    PHI        : real := 0.0;          -- Initial phase shift of the output
    t_rise     : real := 1.0e-9;       -- Rise time of the output
    t_fall     : real := 1.0e-9;       -- Fall time of the output
    R_ip       : real := 100.0e3;      -- Shunt input resistance
    C_ip       : real := 0.1e-12;      -- Shunt input capacitance (arch a2 only)
    R_op       : real := 1.0e3;        -- Series output resistance
    C_op       : real := 0.1e-12 );    -- Shunt output capacitance (arch a2 only)
  port (
    terminal  p1, p2, p3, gnd  : electrical );
   .
   .
   .
```

Check the Generics!

```
    .
    .
    .
begin
  assert Vp>Vn          report "Vp must be > Vn.";
  assert Vhi>Vlo        report "Vhi must be > Vlo.";
  assert f0>0.0         report "f0 must be > zero.";
  assert Kv>0.0         report "Kv must be > zero.";
  assert Kvv>0.0        report "KVV must be > zero.";
  assert fmax>fmin      report "fmax must be > fmin.";
  assert fmax>=f0       report "fmax must be >= f0.";
  assert fmin<=f0       report "fmin must be <= f0.";
  assert PHI<=180.0 and PHI>=-180.0
                        report "PHI must > -180 and < 180 degrees.";
  assert t_rise>1.0e-12 report "t_rise should be > 1 ps." severity warning;
  assert t_fall>1.0e-12 report "t_fall should be > 1 ps." severity warning;
  assert R_ip>0.0       report "R_ip must be > zero.";
  assert R_op>0.0       report "R_op must be > zero.";
  assert C_ip>=0.0      report "C_ip should be >= zero." severity warning;
  assert C_op>=0.0      report "C_op should be >= zero." severity warning;
end entity vco_sqr_tri_1_1;
```

VHDL AMS

2nd Architecture: Constants

```
architecture a2 of vco_sqr_tri_1_1 is
  function ifelse (a: boolean; b,c:real) return real is begin
    if a then return b; else return c; end if;
  end;
  function f2v (f: real)  return real is
    constant fr: real  := f-f0;
    constant tk: real  := Kv + 2.0*Kvv*V_f0;
    constant kr: real  := 8.0*math_pi*Kvv*fr;
  begin
    if kvv = 0.0 and kv/=0.0 then
      return V_f0 + 2.0*math_pi*fr/Kv;
    elsif tk**2 + kr >= 0.0 then
      return V_f0 + (-tk+SQRT(tk**2+kr))/(2.0*Kvv);
    else
      report "Please check values of Kv, Kvv, fmin and fmax. ";
    end if;
  end function f2v;

  constant k1     : real := Kv*(Vp-Vn)/math_pi;        -- contrib to slope for 1 v in
  constant k11    : real := Kvv*(Vp-Vn)/math_pi;
  constant V1     : real := f2v(fmax);                 -- Input corresponding to fmax
  constant V2     : real := f2v(fmin);                 -- Input corresponding to fmin

   .
   .
```

Quantities

```
    .
    .
    .
  quantity vin across iin through p1 to gnd;  -- input branch
  quantity vin_lim: real;                     -- limited version of input
  quantity T_vout1 across p2 to gnd;
  quantity i_Rout, i_Cout through p2;
  quantity i_Cgnd through gnd;
  quantity T_vout3 across i_out3 through p3;  -- triangle wave output
  quantity vout1: real := v_init1;            -- square wave output
  signal   svout1: real := v_init1;           -- discrete form of vout1
  quantity vout2     : real := v_init2;       -- tri output
  quantity Atot      : real;                  -- integrand of tri generator
begin
    .
    .
    .
```

The Schmitt Trigger is Unaltered

```
    .
    .
    .
begin
 schmitt: process
    variable low: boolean := v_init1 < dc;
  begin
    if low then
      wait until not vout2'above(Vn);
      svout1 <= vhi;
    else
      wait until vout2'above(Vp);
      svout1 <= vlo;
    end if;
    low := not low;
  end process;
    .
    .
    .
```

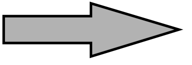
VHDL AMS

The Equations Change

```
  .
  .
  .
-- input load
iin == vin / R_ip;
-- input limiter
If vin'above(V1) then
  Vin_lim == V1;
elsif vin'above(V2) then
  Vin_lim == vin;
else
  Vin_lim == V2;
end if;
-- limiter induced discontinuitites
break on vin'above(V1), vin'above(V2);

vout1  == svout1'ramp(t_rise, t_fall);
Atot   == sign(vout1-dc)* (k1*(Vin_lim-V_f0) + 2.0*f0*(Vp-Vn))
  + k11*(Vin_lim**2 - V_f0**2));
vout2  == Atot'integ + v_init2;
i_Rout == (T_vout1 - vout1) / R_op;
i_Cout == C_op * T_vout1'dot;
i_Cgnd == -i_Cout;
i_out3 == (T_vout3 - vout2) / R_op;
end architecture a2;
```

Outline

- VHDL-AMS Modeling Guidelines
- VHDL-AMS Modeling Techniques
  - IC Applications
- Modeling at Different Levels of Abstraction
  - Telecom Applications



- Modeling of Multi-Disciplinary Systems
  - Automotive Applications
- MEMS Modeling Using the VHDL-AMS Language

VHDL AMS

Overview

- Introduction
- Use of Mixed Domain - Mixed Level Models
- Control Design for Revolving Load
- Plant Model for Component Design
- Approach Towards Unified Modeling, Practical Implications
- Conclusion

Modeling of Multi-Disciplinary Systems

- Mixed domain-modeling
  - Combining different engineering disciplines (e.g., control, mechanics, hydraulics)
- Mixed level-modeling
  - Combining different levels of abstraction (e.g., behavioral model of plant, behavioral model of controler and detailed model of amplifier)

Component Library Using VHDL-AMS

- Documentation
  - VHDL-AMS allows precise behavioral description including discontinuous behavior
- Modeling
  - Single-source
  - Detailed vs. abstract model can be verified
- Simulation
  - All simulators are optimized towards a special-purpose (e.g., electronic, hydraulic, control, real-time)
  - Unique semantic allows conversion into many foreign (non-native) simulators

Use of Mixed-Domain/Mixed-Level (1)

Controler and plant design and validation

Domain simulator

Control simulator

Plant
(detailed)

Plant
(abstract)

Control

Plant
(abstract)

Control

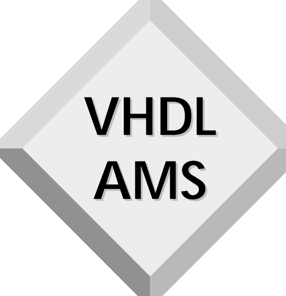
Plant
(abstract)

Control

(real) controler

Real-time simulator

(real) plant

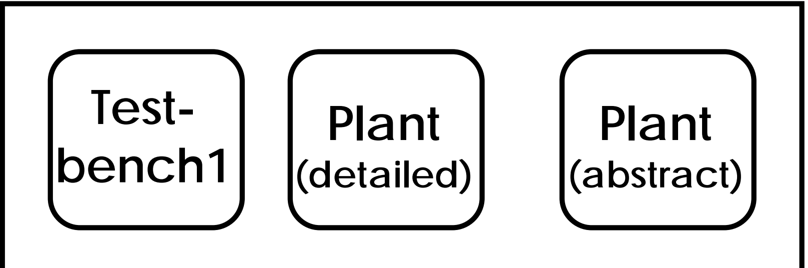


Use of Mixed-Domain/Mixed-Level (2)

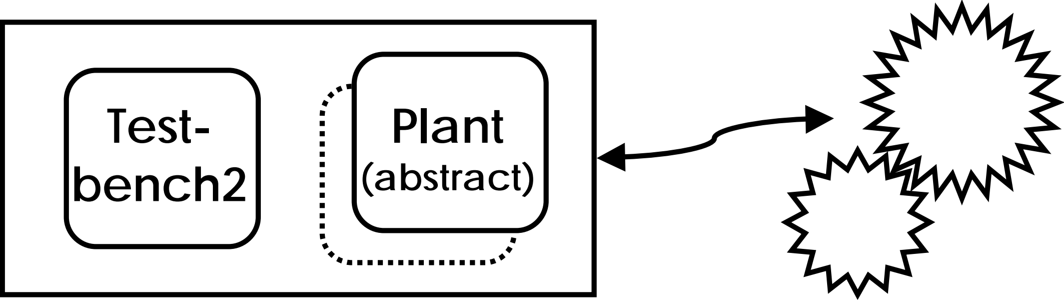
Model validation and parameter extraction

Domain simulator

Validate abstract vs. detailed model



Parameter extraction

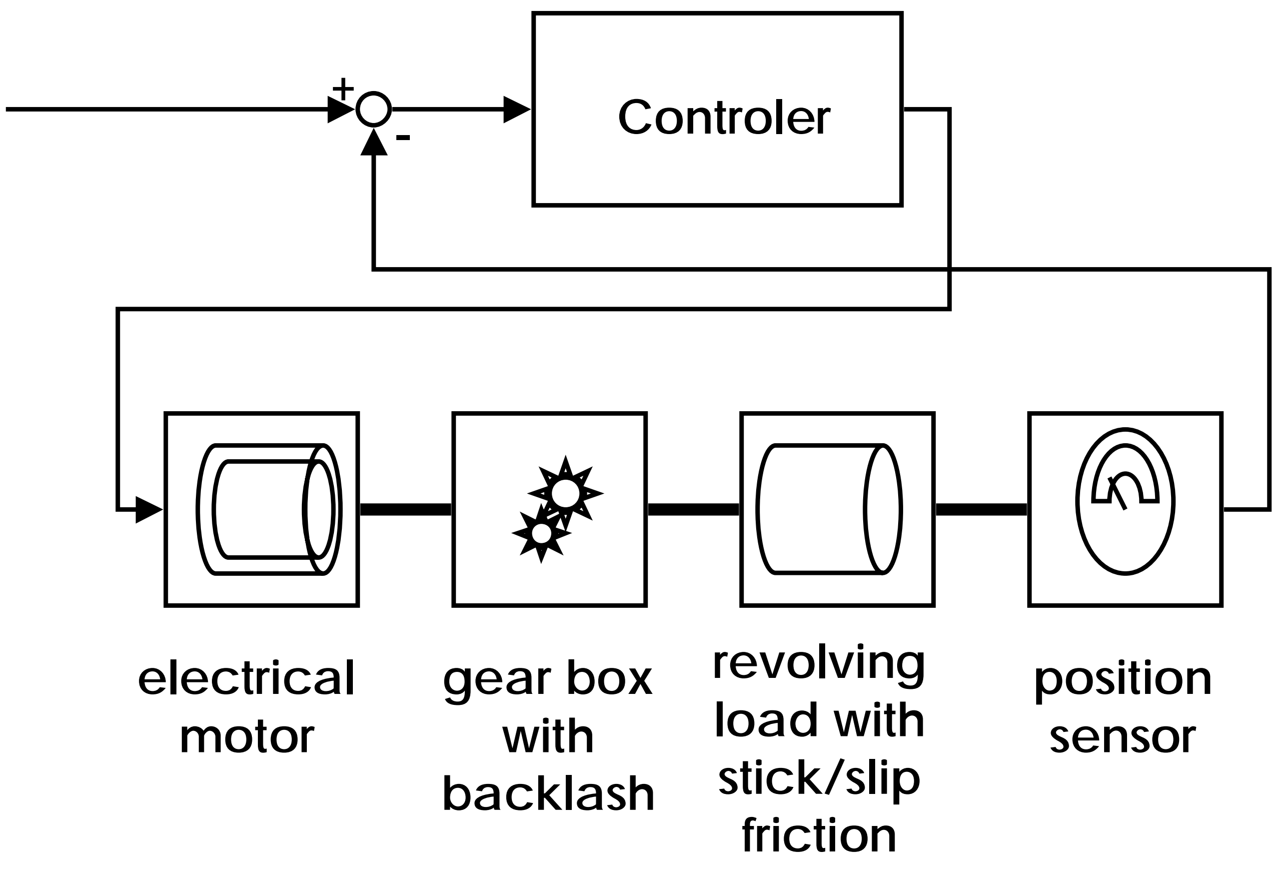


Real-time simulator

VHDL AMS

Controler Design for Revolving Load

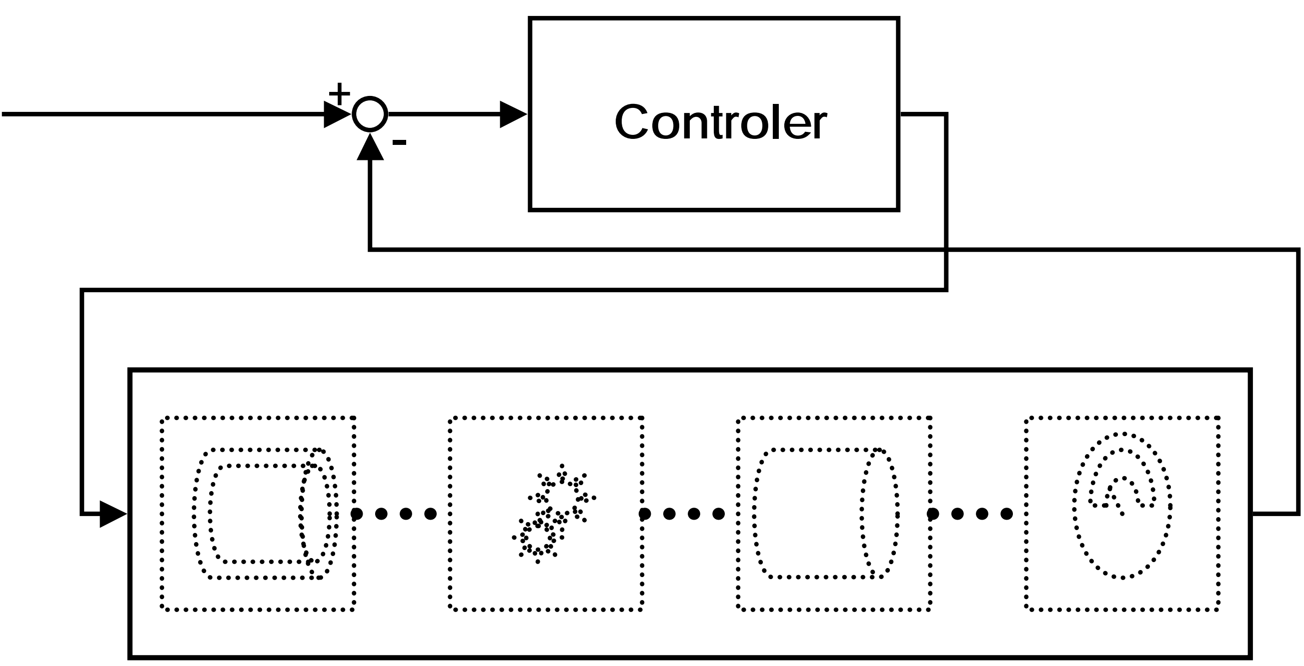
Plant model as vehicle for controler design and verfication (reuse of detailed mechanical component model)



VHDL AMS

Controler Design for Revolving Load

Plant model as vehicle for controler design and verfication (abstract plant model, verified vs. detailed model)

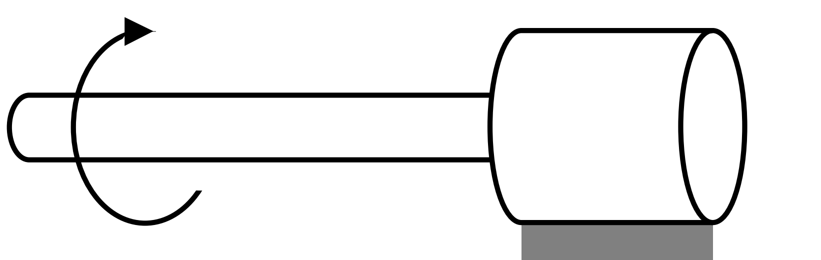


abstract (behavioral) model of revolving load including backlash and stick/slip friction
(VHDL-AMS source code see Appendix)

VHDL AMS

Revolving Load: Stick/Slip-Friction

m:	Momentum
p:	Angular position
w:	Angular velocity
dw:	Angular acceleration



$$p = \int w \, dt$$

IF motion_mode = slip USE

$$w = \int ((m - m_slip \bullet sign(w)) / j) \, dt$$

if motion_mode = stick USE

$$w = 0.0$$

VHDL AMS

Revolving Load: Stick/Slip-Friction

Process which determines the motion mode of the load (toggles betwen stick and slip motion)

```
motion_ctrl: process
begin
   motion_mode <= stick;       -- load sticks
   wait until m_load‘Above(m_stick)  -- wait until abs(m)<m_stick
      or not m_load‘Above(-m_stick);

   motion_mode <= slip;        -- load slips
   wait on w_load‘Above(0.0)  -- wait until w=0 and abs(m)<m_slip
      until not m_load‘Above(m_slip) and m_load‘Above(-m_slip);
end process motion_ctrl;
```

VHDL AMS

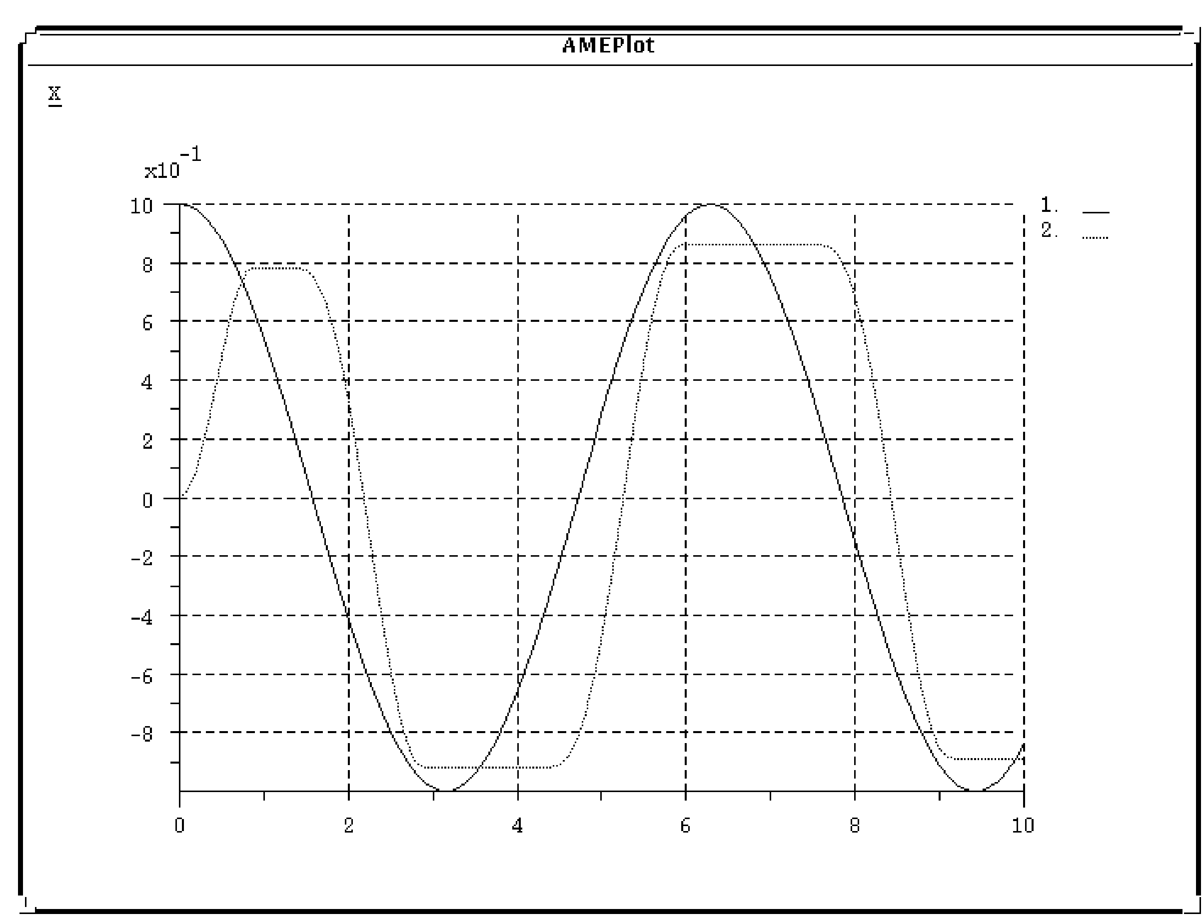
Revolving Load: Stick/Slip-Friction

Equations of motion depending on the motion mode

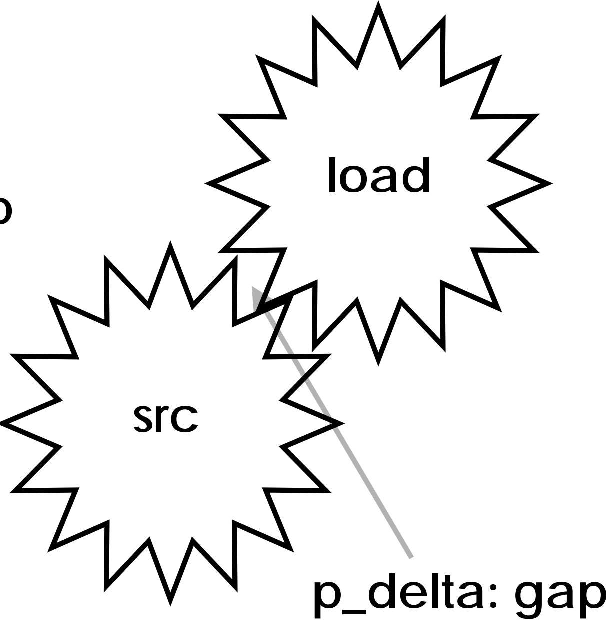
```
p_load'DOT == w_load;          -- p = integral of w

if motion_mode = slip use
   -- w = integral of ((m - sign(w) * m_slip) / j)
   w_load'Dot == calc_internal_momentum(m_load,
                       w_load'Above(0.0), m_slip) / j_load;
else                           -- if motion_mode = stick
   w_load == 0.0;
end use;

break on motion_mode;          -- restart analog kernel
```

Simulation Results of Stick/Slip-Friction

The simulation results feature only the stick/slip friction postion on the load (2.) with sinusoidal momentum (1.)



Revolving Load: Backlash in Gear-Box



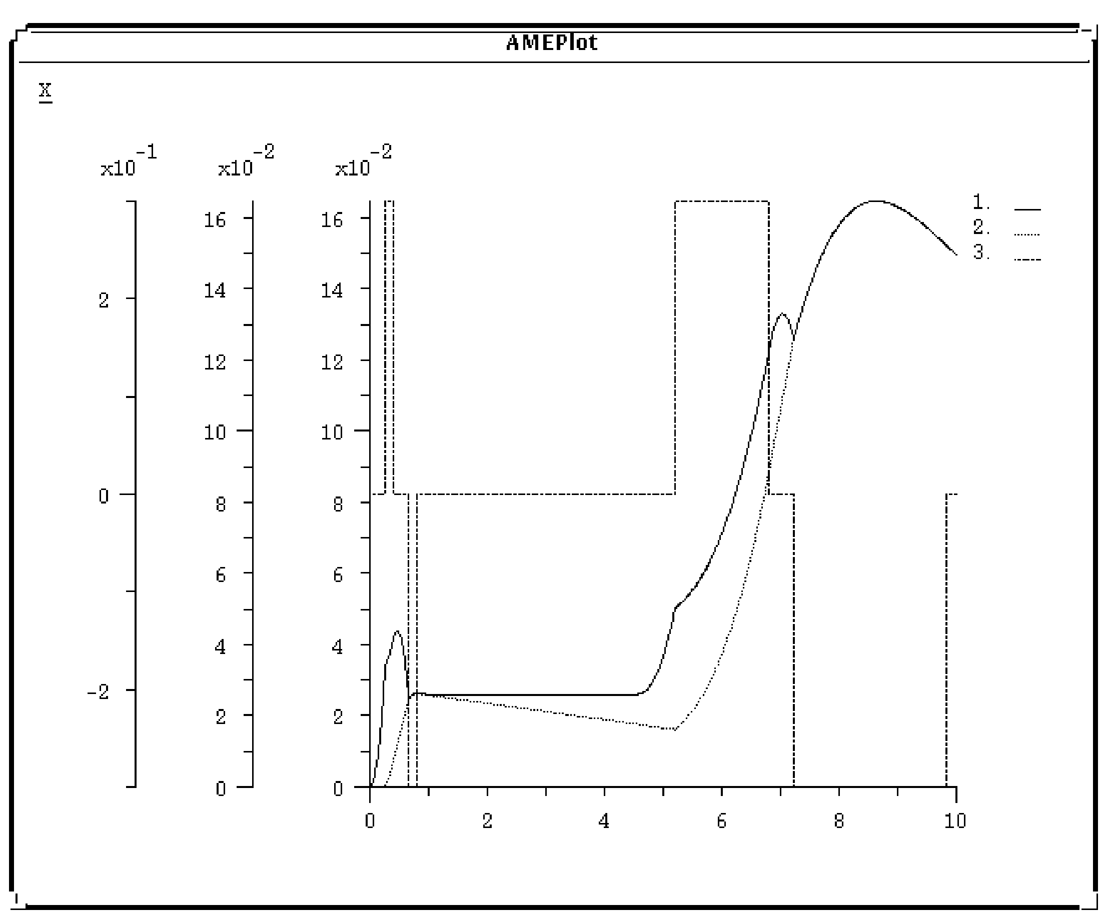
Process features only one side of the gap
(for details see appendix)

```
connection_ctrl: process
begin
   connection_mode <= loose;
   wait until p_src'Above(p_load)
   connection_mode <= coupled;
   break w_src => ... ,
         w_load => ... ;
   wait dw_load'Above(dw_src));
end process connection_ctrl;
```

src: electrical motor

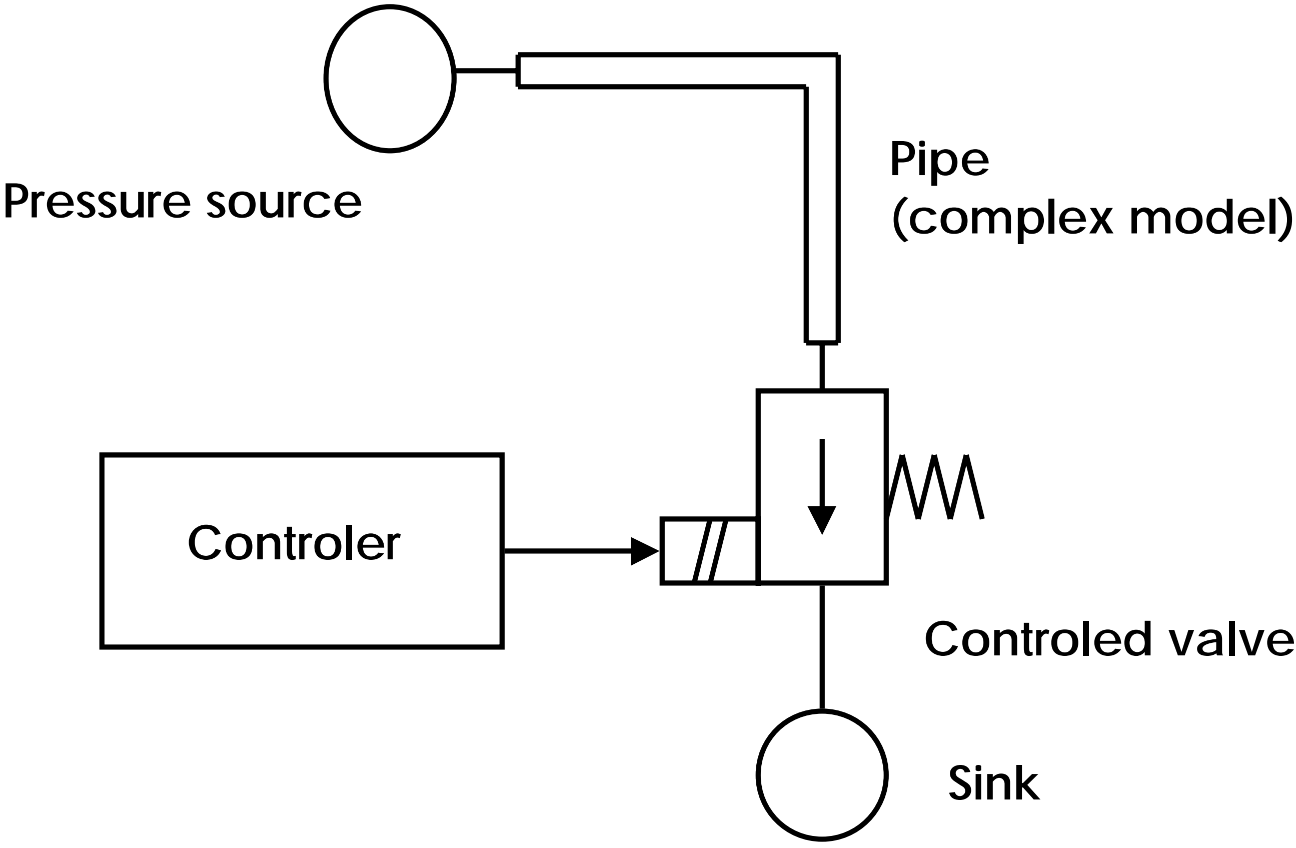
Simulation Results of Backlash

Results feature the position of load (1.), electrical rotor (2.) and status of backlash (3.) [loose = 0, connected otherwise]



Plant Modeling for Component Design

Analysis of plant should reveal destroing effects on valve and pipe due to cavitation (domains: Hydraulics, mechanics and control)



Including Multi Body Systems

- Natural description is topology and geometrical information
  - consisting of joints, masses, links, springs, etc.
- Most simulators never derive a fixed system of differential algebraic equations (DAE) but determine the equations on the fly (during runtime)
- Integration of this models via procedural interface. Simulator vendor distributes library for
  - setting up the system (initialization)
  - derive residual of DAE, or derivative of state variables
  - derive output values (from state variables)

Scenario for Model Exchange

This scenario is supported by European car makers and supplies, partially funded by the EC (BRITE-EURAM-Projects TOOLSYS and ODECOMS)

VHDL AMS

Approach Towards Unified Modeling (1)

- VHDL-AMS is a powerful and large language
- For most mixed-domain modeling only few concepts from (event-driven) VHDL are necessary
- Most continuous simulators have little or no support of the complex event-driven features
- Its usage with other then native VHDL-AMS simulators needs conversion
  - these conversion is simple, if only the basic concepts of event-driven modeling are permitted, and
  - a tiny kernel for these basic features is provided

VHDL AMS

Approach Towards Unified Modeling (2)

- VHDL-AMS subset (proposal)
  - (event-driven) VHDL is pretty much restricted to "synthesis-able" VHDL (e.g. IEEE PAR 1076.6)
    - no history information or delays on signals
    - no memory allocation
    - no global variables
- Further restrictions for real-time simulation
  (and simulators only supporting ordinary differential equations)
  - no conservative communication (no nodes, terminals)
  - only equations of type
    - q_i = f(q_1, ...)
    - q_k'dot = g(q_1, ...)
  - equations of type 1 must be sortable

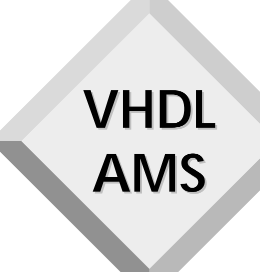
VHDL AMS

Approach Towards Unified Modeling (3)

- Modeling discontinuous behavior needs careful design
  - otherwise portability is difficult to achieve
- Style guide must be developed
  - standard natures, global parameters
  - standard concepts (slip/stick, bouncing ball)
  - never use discontinuous functions (e.g., sign(x)) but

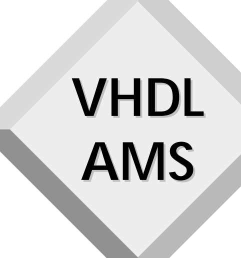
```
if x´above(0.0) use ...
```

  - Coupling elements and defined communication mechanism between domains
  - ...



Conclusion

- Automotive industry is using mixed-domain/mixed-level modeling for design and validation
- The language is an important step towards unified modeling
- Further steps are necessary (modeling techniques)
- VHDL-AMS subset is needed for many applications (e.g., real-time)
- Experiments demonstrate the suitability for many applications



Appendix: VHDL-AMS Source Code of Revolving Load

- Example features an abstract (behavioral) model of the revolving load (see above)
- Inputs: Momentum on armature of motor (m_src)
  Momentum on load (m_load)
- Output: Position of load (p_load)
- Processes: motion control (stick/slip),
  coupling control (loose, coupled ...)
- Equations: Equations of motion

Plant Model: Revolving Load

VHDL-AMS Source Code No. 1

```
ENTITY revolving_load IS
  GENERIC (
    j_src: REAL := 1.0;         -- inertia of electrical motor
    j_load: REAL := 1.0;        -- inertia of load
    m_slip: REAL := 0.01;       -- friction (on load) during slip mode
    m_stick: REAL := 0.1;       -- friction (on load) during stick mode
    delta_p: REAL := 0.034      -- gap between electrical motor and load (angle)
    );
  PORT (
    QUANTITY m_src : IN REAL;   -- momentum generated by electrical motor
    QUANTITY m_load : IN REAL;  -- outer momentum (not from electrical motor)
                                --   on load
    QUANTITY p_load : OUT REAL  -- rotational position of load
    );
END ENTITY revolving_load;
```

VHDL AMS

Plant Model: Revolving Load

VHDL-AMS Source Code No 2

```
ARCHITECTURE behavioral OF revolving_load IS
  QUANTITY p_src:  REAL;  -- rotational position of electrical motor
  QUANTITY w_src:  REAL;  -- rotational velocity of electrical motor
  QUANTITY dw_src: REAL;  -- rotational acceleration of electrical motor
  QUANTITY w_load:  REAL;  -- rotational velocity of load
  QUANTITY dw_load: REAL;  -- rotational acceleration of load (allways
                           --      calculated as if connection_mode = loose)
  -- slip_type defines, if the load sticks or slips
  TYPE slip_type IS ( stick, slip );
  SIGNAL motion_mode: slip_type;          -- handles slip/stick friction status
  -- connection_type defines the status of the connection between the
  --   electrical motor and load due to the backlash:
  --     loose : no connection
  --     coupled_low: connected and p_src = p_load
  --     coupled_high: connected and p_src + delta_p = p_load
  TYPE connection_type IS ( loose, coupled_low, coupled_high );
  SIGNAL connection_mode: connection_type;  -- handles connection status

  -- further quantities are for internal use
  QUANTITY m_load_ext: REAL; -- external momentum on load
  QUANTITY dw_connected: REAL;  -- rotational acceleration connected inertia
                     -- ( allways calculated as if connection_mode = coupled)
```

VHDL AMS

Plant Model: Revolving Load

VHDL-AMS Source Code No 3

```
-- calc_internal_momentum derives internal momentum
--  given the external momentum (m) substracting slip friction m_slip
--  or returning 0.0 if motion_mode = stick
FUNCTION calc_internal_momentum(motion_mode: slip_type; m: REAL;
        direction: BOOLEAN; m_slip: REAL)
  RETURN REAL IS
BEGIN
  IF ( motion_mode = slip ) THEN
    IF direction THEN RETURN m - m_slip;
    ELSE RETURN m + m_slip; END IF;
  ELSE
    RETURN 0.0;
  END IF;
END;
```

Plant Model: Revolving Load

VHDL-AMS Source Code No. 4

```
BEGIN

  -- motion_ctrl determines the motion_mode between slip and stick mode
  -- (toggles between stick and slip motion)
  motion_ctrl: PROCESS
  BEGIN
    motion_mode <= stick;              -- load sticks
                                       -- wait until abs(m) < m_stick
    WAIT UNTIL m_load_ext'ABOVE(m_stick) OR NOT m_load_ext'ABOVE(-m_stick);
    motion_mode <= slip;               -- load slips
    WAIT ON w_load'ABOVE(0.0)          -- wait until w = 0 and abs(m) < m_slip
      UNTIL NOT m_load_ext'ABOVE(m_slip) AND m_load_ext'ABOVE(-m_slip);
  END PROCESS motion_ctrl;

  BREAK ON motion_mode;                -- notice discontinuity
```

VHDL AMS

Plant Model: Revolving Load

VHDL-AMS Source Code No. 5

```
-- connection_ctrl determines the connection_mode due to backlash,
-- toggles between loose and connected (on low or high bound)
connection_ctrl: PROCESS
BEGIN
  connection_mode <= loose;
  WAIT UNTIL p_src'ABOVE(p_load) OR NOT p_src'ABOVE(p_load - delta_p);
  IF p_src'ABOVE(p_load) THEN          -- p_src touches p_load
    connection_mode <= coupled_low;    -- connected on low bound
  ELSE                                 -- p_src touches p_load - delta_p
    connection_mode <= coupled_high;   -- connected on high bound
  END IF;
  -- calculate new common velocities w_src = w_load (reset state variables)
  BREAK w_src => ((j_src * w_src + j_load * w_load) / (j_src + j_load)),
    w_load => ((j_src * w_src + j_load * w_load) / (j_src + j_load));
  -- if acceleration of (individual) objects fullfill condition below
  -- change connection_mode to loose
  WAIT ON dw_load'ABOVE(dw_src), dw_src'ABOVE(dw_load)
    UNTIL (connection_mode = coupled_low AND dw_load'ABOVE(dw_src)) OR
          (connection_mode = coupled_high AND dw_src'ABOVE(dw_load));
  BREAK;                               -- notice discontinuity
END PROCESS connection_ctrl;
```

VHDL AMS

Plant Model: Revolving Load

VHDL-AMS Source Code No. 6

```
- initial setting of position and velocity of electrical motor and load
BREAK
  p_src => 0.0,
  w_src => 0.0,
  p_load => 0.0,
  w_load => 0.0;

-- state equations
p_src'DOT  == w_src;              -- p_src := integral of w
p_load'DOT == w_load;             -- p_load := integral of w

IF connection_mode = loose USE    -- if loose
  w_src'DOT  == dw_src;           -- w_src  := integral of dw_src
  w_load'DOT == dw_load;          -- w_load := integral of dw_load
ELSE                              -- if connected
  w_src'DOT  == dw_connected;     -- w_src  := integral of dw_connected
  w_load'DOT == dw_connected;     -- w_src  := integral of dw_connected
END USE;
```

VHDL AMS

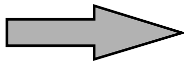
Plant Model: Revolving Load

VHDL-AMS Source Code No. 7

```
  -- internal quantities
  dw_src  ==      m_src / j_load;       -- dw_src (as if not connected)
  -- dw_load (as if not connected, with friction)
  dw_load ==      calc_internal_momentum(motion_mode, m_load,
                    w_load'ABOVE(0.0), m_slip) / j_load;
  -- dw_connected (as if connected, with friction)
  dw_connected == calc_internal_momentum(motion_mode, m_load + m_src,
                    w_src'ABOVE(0.0), m_slip) / (j_src + j_load);

  -- m_load_ext is used for determining motion_mode (motion_ctrl)
  IF connection_mode = loose USE
    m_load_ext == m_load;
  ELSE
    m_load_ext == m_load + m_src;
  END USE;

END ARCHITECTURE behavioral;
```

Outline

- VHDL-AMS Modeling Guidelines
- VHDL-AMS Modeling Techniques
  - IC Applications
- Modeling at Different Levels of Abstraction
  - Telecom Applications
- Modeling of Multi-Disciplinary Systems
  - Automotive Applications



- MEMS Modeling Using the VHDL-AMS Language

VHDL AMS

MEMS Accelerometers

- Open-Loop Accelerometers
  - Acceleration is sensed by measuring displacement of a seismic (proof/shuttle) mass
  - Advantages: low cost and small size
  - Disadvantages: nonlinearity/hysteresis effects and fatigue
- Closed-Loop Accelerometers
  - Acceleration is sensed by measuring force required to maintain position of seismic (proof/shuttle) mass
  - Advantages: reduced transverse sensitivity
  - Disadvantages: more circuitry and higher costs

VHDL AMS

Accelerometer Performance Characteristics

- Dynamic range of acceleration
  - Displacement limits
- Frequency response
  - Mechanical and electrical time constants
- Linearity
  - Parasitics
- Transverse (out-of-plane) sensitivity
  - Seismic mass suspension
- Temperature sensitivity
- Noise floor

Accelerometer Applications

smart bombs
machine health
air bags
active suspension
heads up display
shipping
navigation

bandwidth <Hz>

10^{+4}
10^{+3}
10^{+2}
10^{+1}
1
10^{-1}

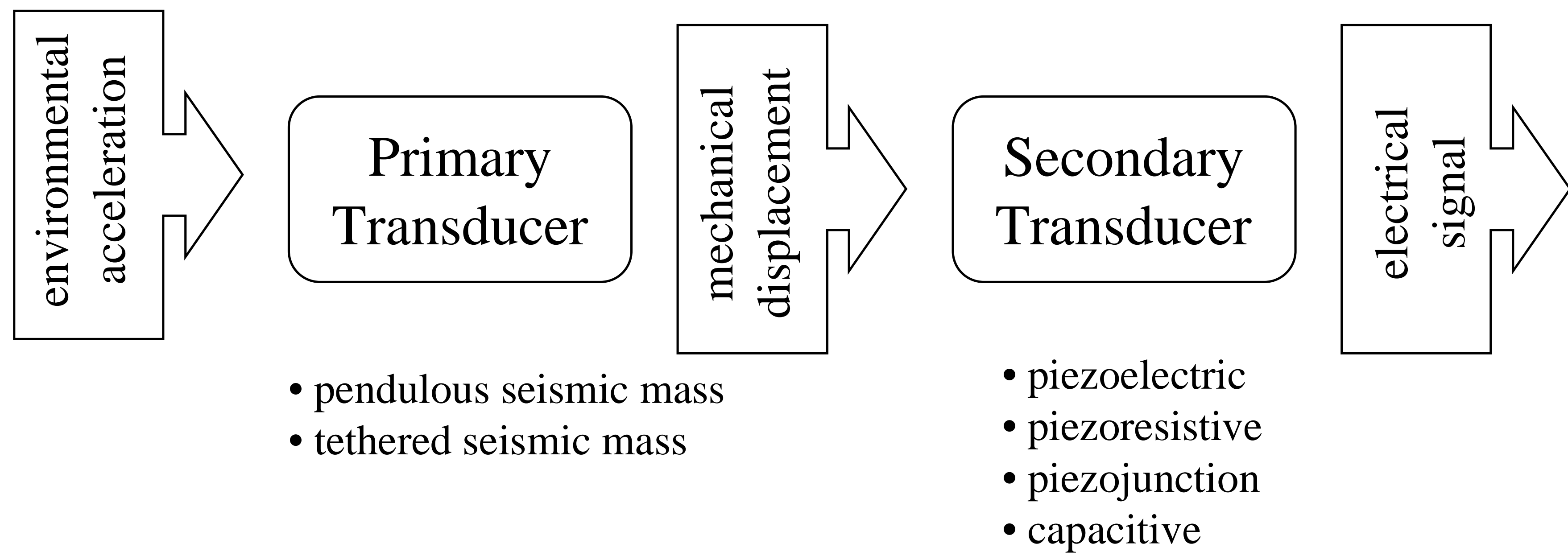
10^{-6} 10^{-4} 10^{-2} 1 10^{+2} 10^{+4}

acceleration <g>

- B. Boser, UCBerkeley

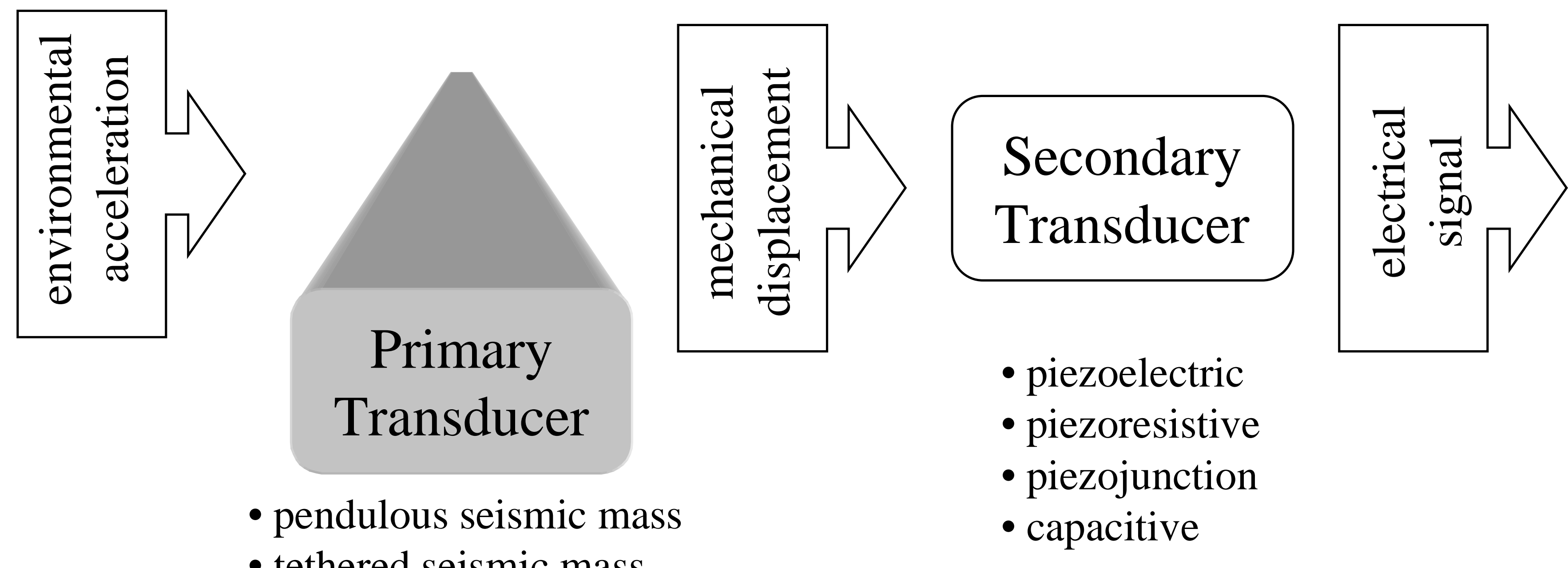
MEMS Accelerometers

- Sensing is typically performed in two steps
  - Transform acceleration to mechanical displacement
  - Transform mechanical displacement to electrical signal

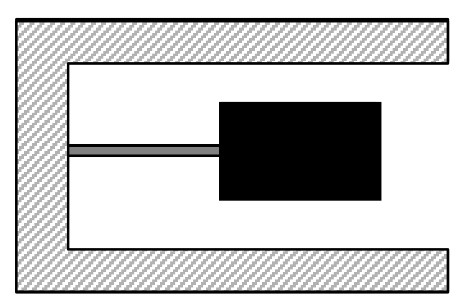


MEMS Accelerometers

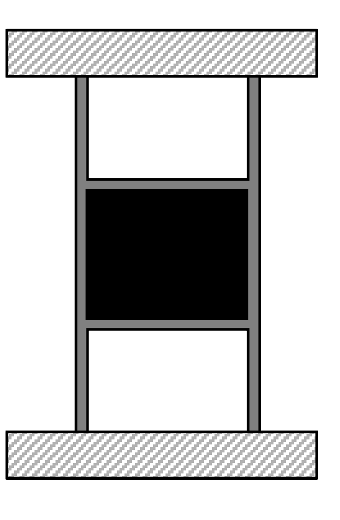
- Sensing is typically performed in two steps
  - Transform acceleration to mechanical displacement
  - Transform mechanical displacement to electrical signal



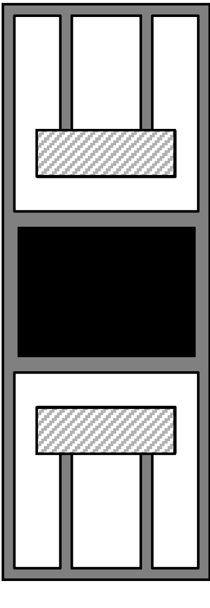
Microflexural Structures



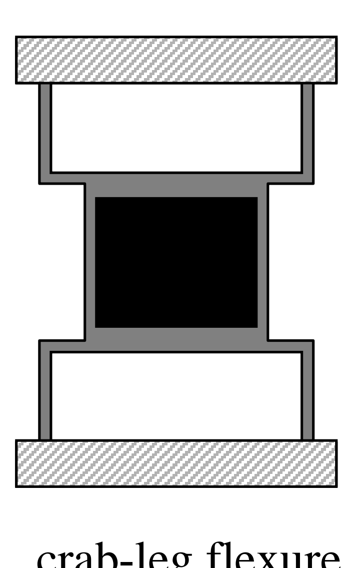
single cantilever



hammock flexure

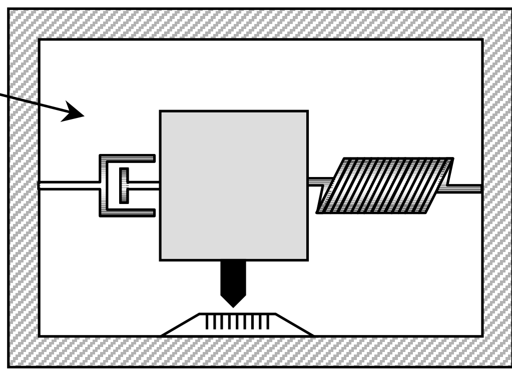


folded flexure



crab-leg flexure

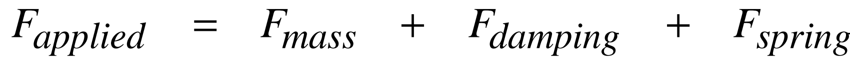
damped harmonic oscillator



2nd order differential equation

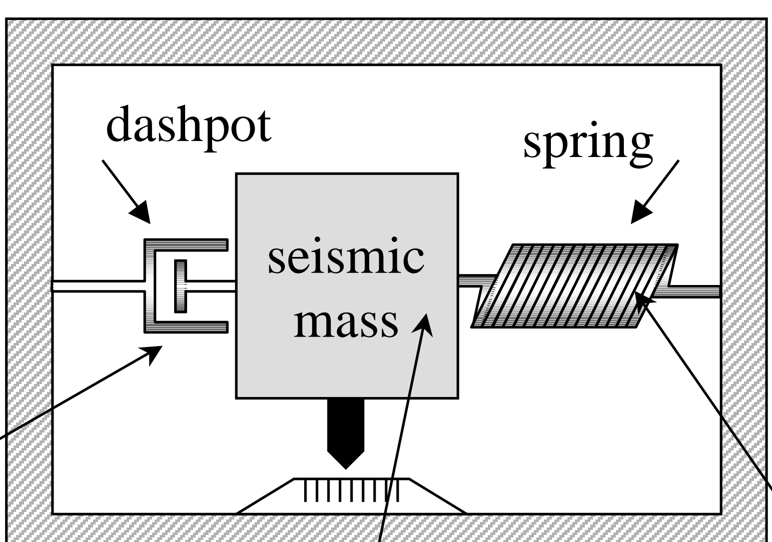
VHDL AMS

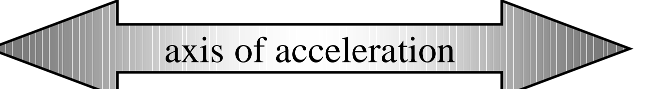
Damped Harmonic Oscillator



$$F_{applied} = F_{mass} + F_{damping} + F_{spring}$$

$$F(t) = M\frac{d^2x}{dt^2} + D\frac{dx}{dt} + Kx$$





Model frictional resistance as damping force proportional to rate of movement (velocity)

Model inertia as force proportional to rate of velocity (acceleration)

Model structural elasticity as spring force proportional to movement

2nd Order Differential Equation

- 2nd order ordinary differential equations (ODEs) naturally arise as mathematical models of physical systems - one per each degree of freedom.

VHDL AMS

Canonical Response

- Interia, dissipative, elasticity characteristics determine transient (time domain) and bandwidth (frequency domain) response.

$$\frac{F(t)}{M} = G(t) = \frac{d^2x}{dt^2} + 2\xi\omega_0\frac{dx}{dt} + \omega_0 x$$

$$\omega_0 = \sqrt{\frac{K}{M}}$$

• natural resonant frequency - oscillation with no damping/ forcing

$$\xi = \frac{D}{2\sqrt{K M}}$$

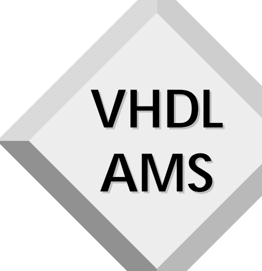
• damping factor - actual damping/critical damping

$$\omega = \omega_0\sqrt{1 - \xi^2}$$

• damped resonant frequency

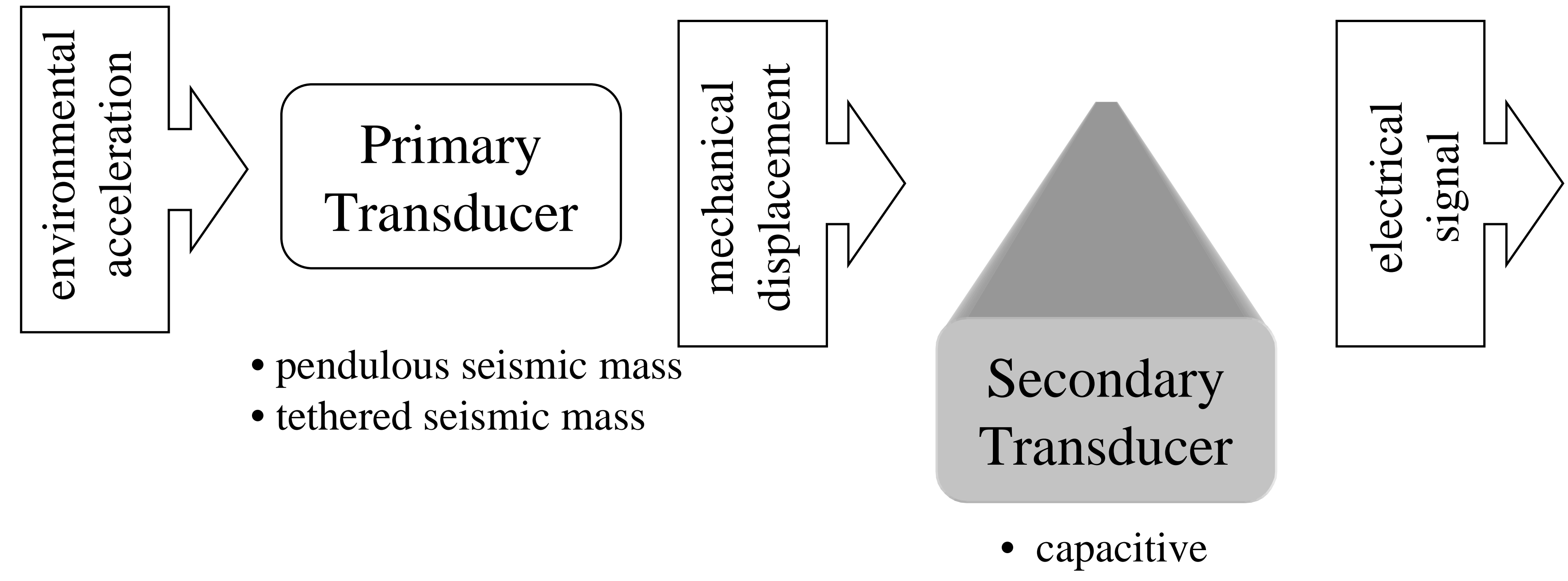
$$\frac{x}{a} = \frac{1}{\omega_0^2}$$

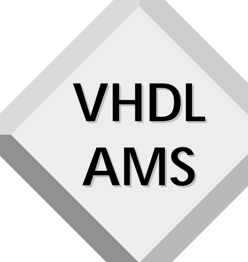
• primary transducer transfer function - displacement/unit acceleration



MEMS Accelerometers

- Sensing is typically performed in two steps
  - Transform acceleration to mechanical displacement
  - Transform mechanical displacement to electrical signal



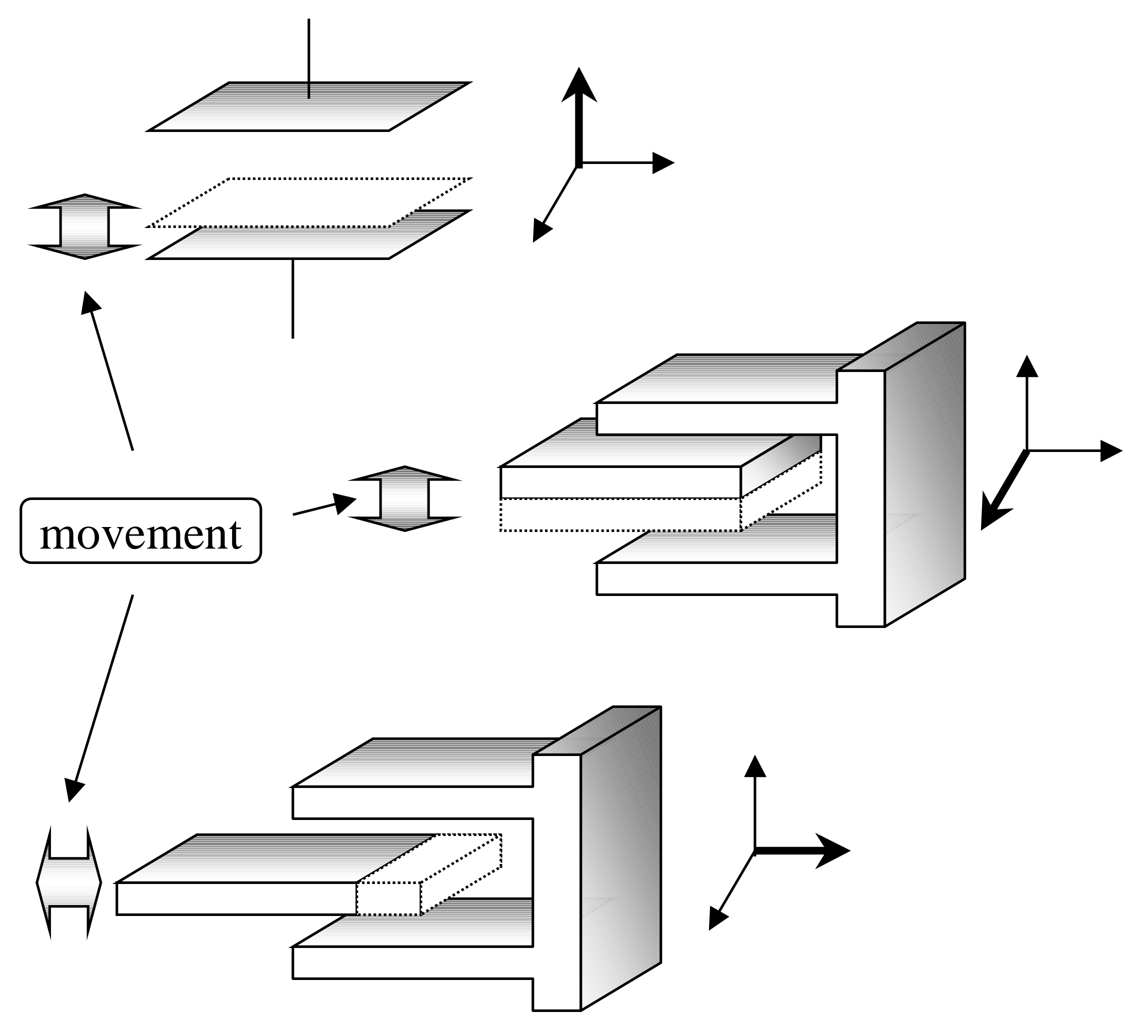


Capacitive Sensing

- Seismic mass forms one plate of a parallel plate capacitance. Movement of mass changes area/gap between parallel plates and, consequently the capacitance.
- Advantages
  - low temperature sensitivity
  - noncontacting transduction
  - insensitivity to magnetic fields
  - operational reliability
- Disadvantages
  - parasitics
  - undesired electrostatic forces

Capacitive Sensing Configurations

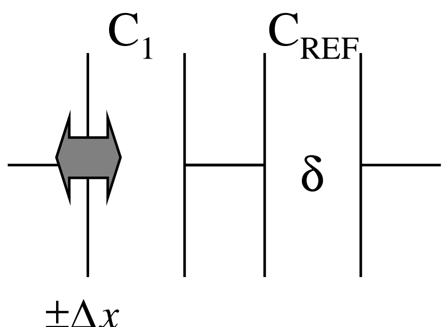
- Parallel plate
- Transverse comb
- Lateral comb

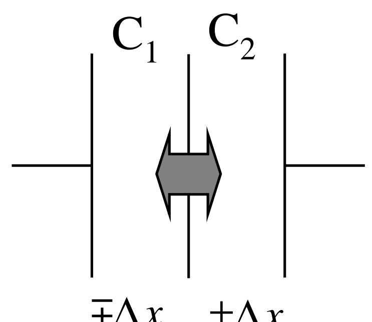


Differential Capacitance Sensing

VHDL AMS

- Easier to detect relative (differential) change rather than absolute change
- Differential configuration preferred over single-ended for linearity.





$$C_1 - C_{REF} \approx \frac{\varepsilon A}{\delta}\left\{\frac{\Delta x}{\delta} + \left(\frac{\Delta x}{\delta}\right)^2\right\}$$

$$C_1 - C_2 \approx 2\frac{\varepsilon A}{\delta}\left\{\frac{\Delta x}{\delta} - \left(\frac{\Delta x}{\delta}\right)^3\right\}$$

Transverse Linear Comb Drive

Capacitance Bridge (Analog Sampling)

- Comb finger capacitances connected in parallel to add to form C_1 and C_2

V_{sample}

$-V_{sample}$

$$C_1 = \frac{\varepsilon A}{\delta - x}$$

$$C_2 = \frac{\varepsilon A}{\delta + x}$$

$$V_{sense} = V_{sample}\left\{\frac{C_1 - C_2}{C_1 + C_2}\right\} = V_{sample}\left(\frac{x}{\delta}\right)$$

modulating factor - x(t) - kHz

carrier frequency - MHz

MEMS Accelerometers

- Remaining processing conditions electrical signal
  - Extracts modulated acceleration information

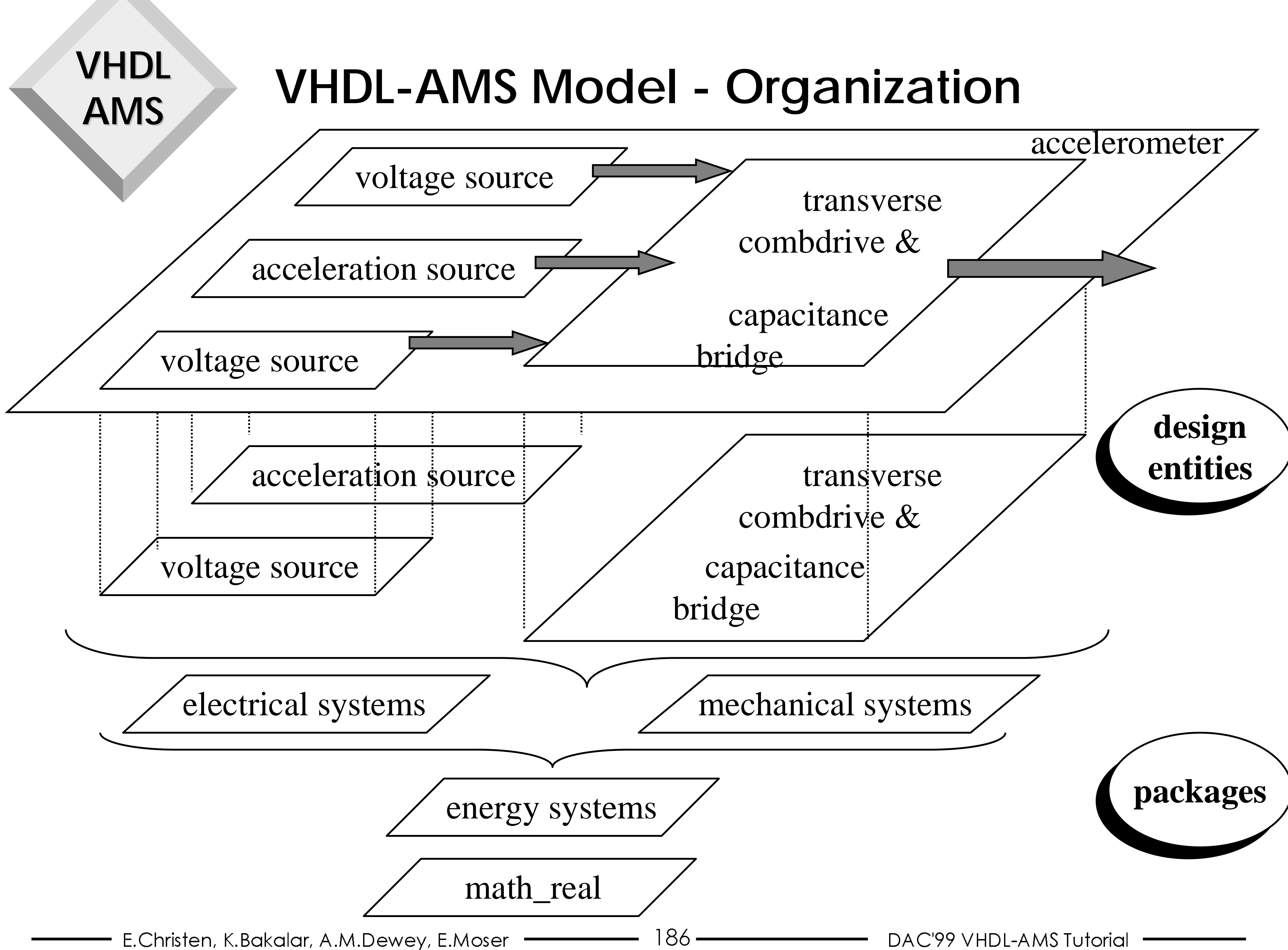
a_{in} → environmental acceleration → Primary Transducer → mechanical displacement → Secondary Transducer → electrical signal

VHDL-AMS

Highpass Filter → modulated signal - no DC → Demodulator → demodulated signal → Lowpass Filter → voltage signal → V_{out}

$$\frac{V_{out}}{a_{in}} \propto \frac{V_{sample}}{\omega_0^2\,\delta}$$

VHDL-AMS Model - Organization



Design Entity: Acceleration Source

- Alternative forcing functions for ambient acceleration

```
library IEEE;
use IEEE.MATH_REAL.all;
use IEEE.MECHANICAL_SYSTEMS.all;
entity ENV_FORCE is
  generic (constant MAG_AC, MAG_DC : FORCE := 0.0;
           constant FREQ : REAL := 0.0);
  port (terminal PT1, PT2 : TRANSLATIONAL);
end entity ENV_FORCE;
```

```
architecture DC of ENV_FORCE is
  quantity FORCE through PT1 to PT2;
begin
  FORCE == MAG_DC;
end architecture DC;
```

```
architecture SINE of ENV_FORCE is
  quantity FORCE through PT1 to PT2;
begin
  FORCE == MAG_AC*sin(MATH_2_PI*FREQ*NOW);
end architecture SINE;
```

VHDL AMS

Design Entity: Voltage Source

- Carrier frequency for analog sampling

```
library IEEE;
use IEEE.MATH_REAL.all;
use IEEE.ELECTRICAL_SYSTEMS.all;
entity VSOURCE is
  generic (constant MAG_AC, MAG_DC : VOLTAGE := 0.0;
           constant FREQ : REAL  := 0.0);
  port (terminal P, M : ELECTRICAL);
end entity VSOURCE;

architecture SINE of VSOURCE is
  quantity V across I through P to M;
begin
  V == MAG_AC*sin(MATH_2_PI*FREQ*NOW) + MAG_DC;
end architecture SINE;
```

Design Entity: Transducer

- Design entity declaration couples energy domains
  - Electrical domain
  - Mechanical domain

```
library IEEE;
use IEEE.ENERGY_SYSTEMS.all
use IEEE.ELECTRICAL_SYSTEMS.all;
use IEEE.MECHANICAL_SYSTEMS.all;
entity COMB_DRIVE is
  generic (M  : MASS      := 0.16*NANO;
           D  : DAMPING   := 4.0e-6;
           K  : STIFFNESS := 2.6455;
           A  : REAL      := 2.0e-6*110.0e-6 ;
           D0 : REAL      := 1.5e-6);

  port(terminal PROOF_MASS, REF : TRANSLATIONAL;
       terminal TOP_EL, MID_EL, BOT_EL : ELECTRICAL);
end entity COMB_DRIVE;
```

mechanical properties

geometric properties

Design Entity: Transducer (Continued)

```
architecture BCR of COMB_DRIVE is
  -- free quantities
  quantity VEL     : VELOCITY;              quantity DTM,DBM : DISPLACEMENT;
  quantity QTM,QBM : CHARGE;                quantity CTM,CBM : CAPCITANCE;

  -- branch quantities
  quantity POS across FORCE through PROOF_MASS to REF;
  quantity VTM across ITM   through TOP_EL     to MID_EL;
  quantity VBM across IBM   through BOT_EL     to MID_EL;
begin
  -- compute displacement of comb drive
  VEL == POS'DOT;
  FORCE == K*POS + D*VEL + M*VEL'DOT;

  DTM == D0 + POS; DBM == D0 - POS;

  -- compute change in capacitance
  CTM == A*EPS0/DTM;  CBM == A*EPS0/DBM;

  -- compute generated current
  QTM == CTM*VTM;     QBM == CBM*VBM;
  ITM == QTM'DOT;     IBM == QBM'DOT;
end architecture BCR;
```

mechanical dynamics

electrical dynamics

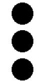
Design Entity: MEMS Accelerometer

```
library IEEE;
use IEEE.ELECTRICAL_SYSTEMS.all;
use IEEE.MECHANICAL_SYSTEMS.all;
entity ACCELEROMETER is
end entity ACCELEROMETER;
```

```
architecture TOP_LEVEL of ACCELEROMETER is
 terminal SMASS : TRANSLATIONAL;
 terminal TOP, MID, BOT : ELECTRICAL;
begin
 F1:entity WORK.ENV_FORCE(SINE)
    generic map (MAG_AC=>0.16e-9*5.0*GRAV, FREQ=>100.0)
    port map (PT1=>SMASS, PT2=>ANCHOR);

 V1:entity WORK.VSOURCE
    generic map (MAG_AC=>300.0e-3, FREQ=>1.0*MEGA, MAG_DC=>0.0)
    port map (P=>TOP, M=>GROUND);

 V2:entity WORK.VSOURCE
    generic map (MAG_AC=>-300.0e-3, FREQ=>1.0*MEGA, MAG_DC=>0.0)
    port map (P=>BOT, M=>GROUND);
              ⋮
```

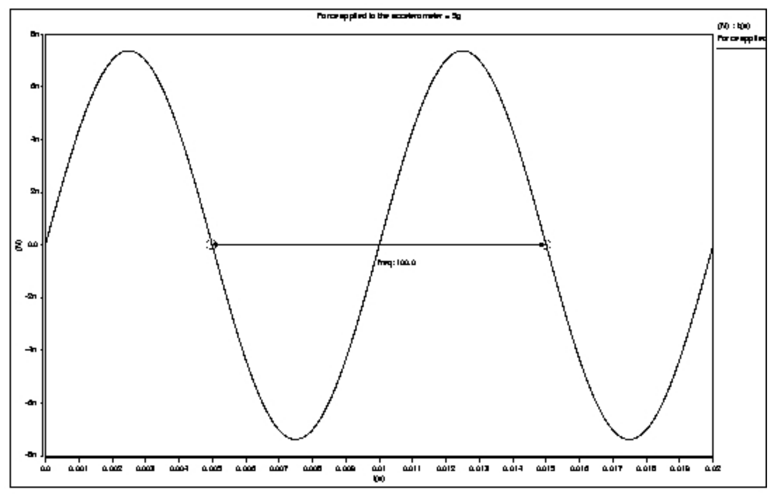
VHDL AMS

Design Entity: MEMS Accelerometer (Continued)

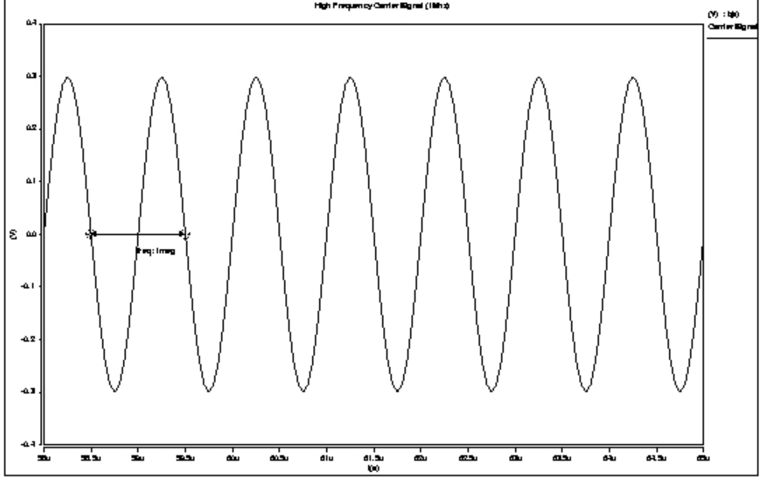


```
A1:entity WORK.COMB_DRIVE
    generic map (M => 0.16e-9,
                 K => 2.6455,
                 D => 4.0e-6,
                 A => 2.0e-6*110.0e-6,
                 D0 => 1.5e-6)

    port map (   PROOF_MASS => SMASS,
                 REF => ANCHOR,
                 TOP_EL => TOP,
                 MID_EL => MID,
                 BOT_EL => BOT);

R1:entity WORK.RESISTOR
   generic map (RNOM => 3.0*MEGA)
   port map (P=>MID, M=>GROUND);

end architecture TOP_LEVEL;
```

Sample Simulation Results



Input Force

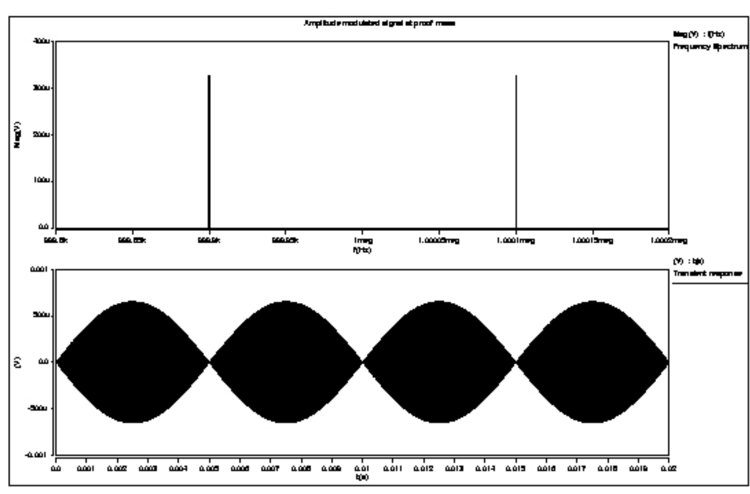


Analog Voltage
Carrier Frequency

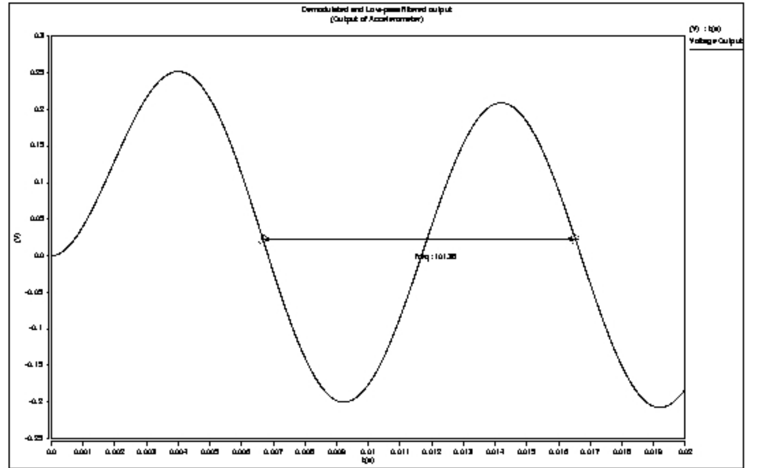
Sample Simulation Results

Seismic Mass Position

Modulated Signal



Demodulated and Lowpass Filtered Signal

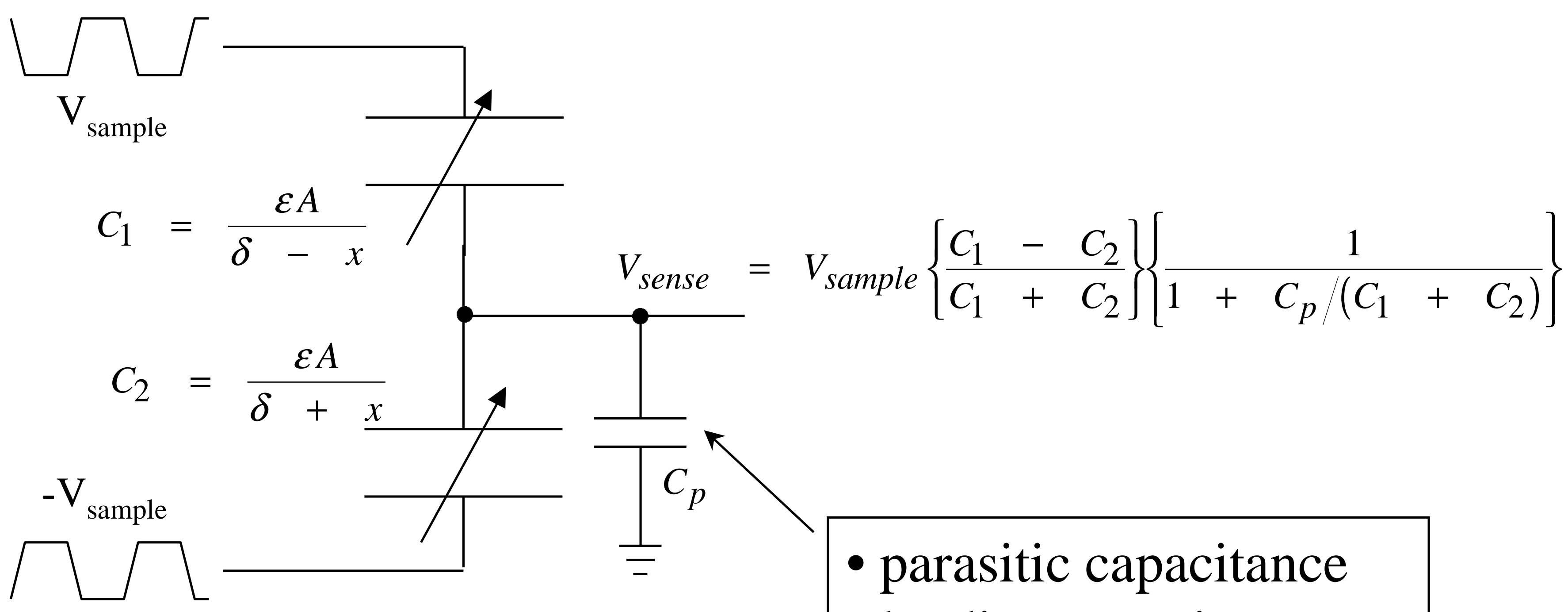


Improved MEMS Accelerometer Model

- Model shows only first-order physics
  - Not adequate for practical design
- Model assumes constant values for mechanical elasticity and damping
  - Parametric values per microflexural structure
  - Interaction of electrostatic forces
- Model assumes basic capacitance models
  - Parasitics
  - Nonparallel plates

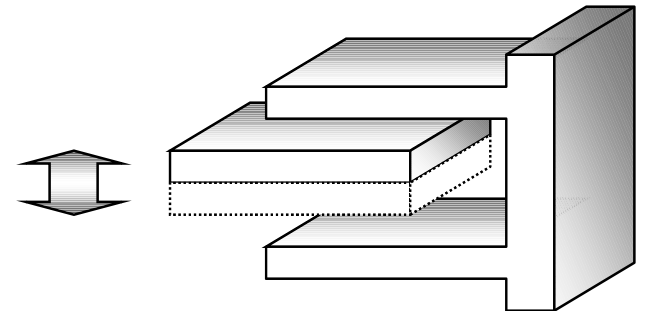
Parasitic Capacitances

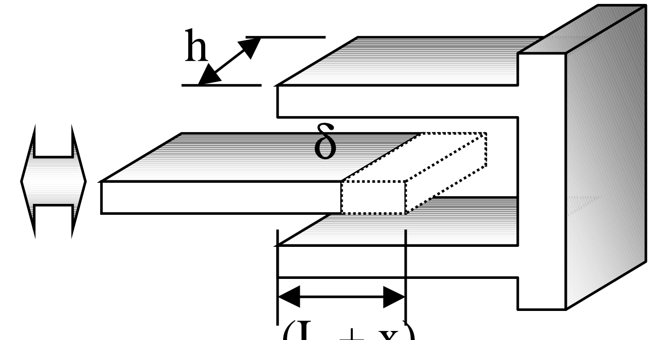
- Stray and fringe field capacitances influence transduction properties
  - Introduces nonlinearity in modulation
  - Attenuates gain



Electrostatics

- Potential energy - $E = \int_Q V\,dQ = \int_Q \frac{Q}{C}\,dQ = \frac{1}{2}\frac{Q^2}{C} = \frac{1}{2}CV^2$
- Electrostatic force - $F = \frac{\partial E}{\partial x} = \frac{\partial}{\partial x}\left(\frac{1}{2}CV^2\right) = \frac{1}{2}V^2\frac{\partial C}{\partial x}$
- Electrostatic spring coefficient - $k_e = \frac{\partial F}{\partial x} = \frac{1}{2}V^2\frac{\partial^2 C}{\partial x^2}$





$$C(x) = \frac{\varepsilon A}{(\delta + x)} = C_0\left(1 + \frac{x}{\delta}\right)^{-1}$$

$$F(x) = \frac{1}{2}V^2\frac{C_0}{\delta}\left(1 + \frac{x}{\delta}\right)^{-2}$$

$$k_e(x) = \frac{\partial F}{\partial x} = V^2\frac{C_0}{\delta^2}\left(1 + \frac{x}{\delta}\right)^{-3}$$

$$C(x) = 2\frac{\varepsilon h}{\delta}(L + x) = C_0\left(1 + \frac{x}{L}\right)$$

$$F = \frac{1}{2}V^2\frac{C_0}{L} = V^2\frac{\varepsilon h}{\delta}$$

$$k_e = \frac{\partial F}{\partial x} = 0$$

VHDL AMS

Electrostatic/Mechanical Elasticity

- Electrostatic force opposes structural restoring force
- Addition of electrostatic force “softens” the transducer
- Elasticity is combination of mechanical and electrical spring constants
  - Frequency-pulling

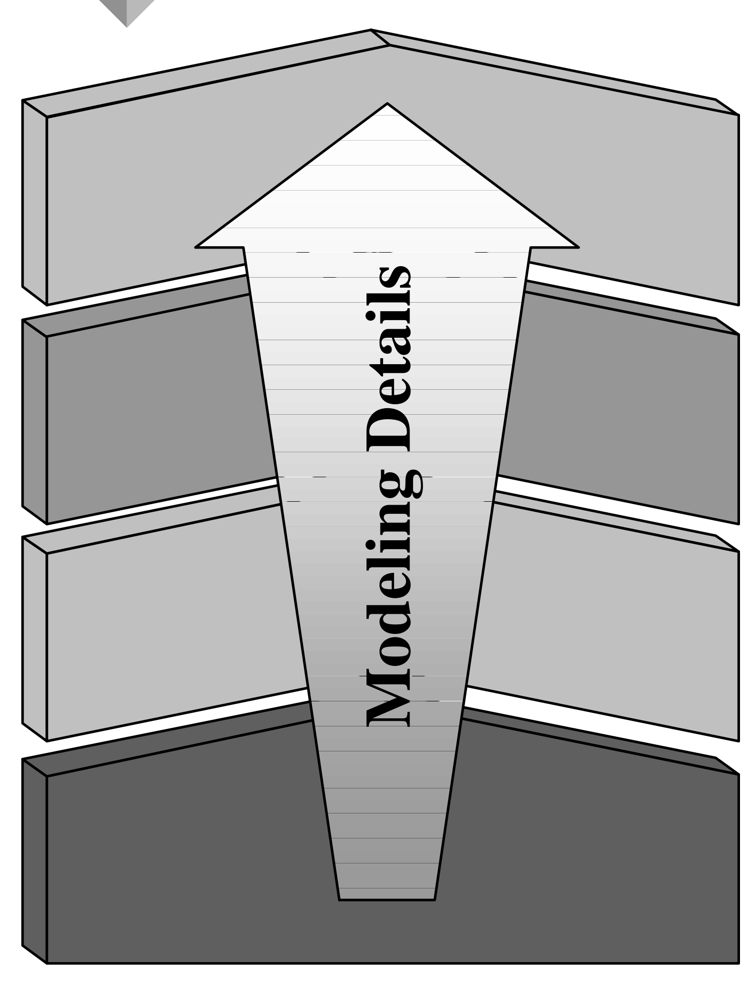
$$K_{electrical} = \frac{\partial}{\partial x}(F_1 - F_2)$$

$$K = K_{electrical} + K_{mechanical}$$

$$\omega_0 = \sqrt{\frac{K_{electrical} + K_{mechanical}}{M}}$$

VHDL AMS

Construct Modeling Levels



- Noise
- Temperature
- High-order modal modeling

- Nonparallel plate capacitances
- Squeeze film damping
- Residual material stresses

- Electrostatic spring constants
- Parametric mechanical elasiticity
- Parasitic capacitances

Basic lumped-element model

- 2nd-order harmonic mechanical oscillator
- Constant coefficients