VHDL Analog and Mixed-Signal Extensions Through Examples

Alain Vachoux

Swiss Federal Institute of Technology (EPFL)
Dept. of Electrical Engineering
Integrated Systems Center (C3i)
CH-1015 Lausanne, Switzerland
Tel: +41 21 693 69 84
Fax: +41 21 693 46 63
e-mail: vachoux@leg.de.epfl.ch

Abstract – VHDL 1076.1 denotes an effort to enhance the IEEE VHDL 1076 standard with analog and mixed-signal capabilities. At the time of this writing (November 1995), the development of the extensions is nearing completion. An IEEE ballot to adopt the extensions as a new IEEE standard should happen in Q1 97. This tutorial provides an overview of the proposed extensions through a number of simple, but characteristic, examples.

I. Introduction

VHDL is an IEEE standard language for the description and the simulation of digital hardware (IEEE Std 1076-1993)[1][2]. Since its first release in 1987, *aka* VHDL 87, VHDL progressively took a central place in top-down hardware design flows as well as in all major EDA environments available in the market. It even went beyond its original goal as its use has extended to support synthesis, formal verification, and testing.

Since 1987, VHDL went through a first restandardization phase that corrected some language inconsistencies and added new language features. This culminated with a new version of the standard known as VHDL 93 [3]. These modifications have been done to satisfy a number of requirements coming from users and tool developers based on their experience in VHDL 87. Among these requirements, many of them were asking to add analog and mixed-signal capabilities to the language. Due to the complexity of the task, it has been decided to develop such extensions in a separate effort that would not preclude the restandardization of "digital" VHDL. A working group was then formed under the auspices of the IEEE Design Automation Standards Committee with the charter to develop extensions to VHDL 1076 that support the description and the simulation of systems with continuous behavior over amplitude and time (PAR 1076.1)[4][1]. The language design is now nearing completion. A draft 1076.1 Language Reference Manual is available, and a ballotable version is expected by the end of 1996. The extended language has informally been called VHDL-A, at the beginning, and then VHDL-AMS (for "Analog and Mixed-Signal"), although the official name is VHDL 1076.1.

The work within the 1076.1 working group is the result of a collective effort from many individuals from industry and academy. It has been supported by grants from European JESSI and ESPRIT programs, and US Rome Laboratory. In addition, EIAJ (Electronic Industries Association of Japan) is today very active in the validation of the language design by providing a feedback from a user perspective.

II. Language Overview

The VHDL 1076.1 language is designed to meet a set of design objectives that have been documented in the 1076.1 Design Objective Document [5]. The objectives have been formalized from requirements submitted to the working group. They mainly ask the language to be built on two major foundations: the VHDL 1076 language and the theory of differential and algebraic equations.

The scope of the language is the description and the simulation of continuous and mixed continuous/discrete systems at different abstraction levels. This generalization to other disciplines of the support of analog and mixed-signal electrical systems is readily possible since they all use the same paradigms to describe system structure and behavior. The language supports the modeling of lumped systems with conservative and non-conservative (signal-flow) semantics. Continuous behavior is described with differential and algebraic equations (DAEs) whose solution may include discontinuities. Discrete behavior is described with the full capabilities of VHDL 1076. Interactions between the continuous part and the discrete part are supported at both the description and the simulation levels in a flexible and efficient way.

The solution of a continuous 1076.1 model is computed by a conceptual agent referred to as the "analog solver". The VHDL 1076.1 language defines the system of equations that are solved and the results the analog solver has to achieve, but

[1] For more information on the working group activities and achievements, see http://www.vhdl.org/vi/analog/.

ASP-DAC '97
0-89791-851-7/$5.00

it does not define any algorithm to compute the solution. It is assumed, in fair generality, that the solution of continuous models cannot be computed exactly, so a mechanism to define tolerances is included in the language. The language supports time domain simulation and limited frequency domain simulation. The latter is based on a small-signal model derived from the (time domain) equations given in the text of the model. Small-signal frequency domain and time domain noise simulation are also supported.

The simulation of a discrete 1076.1 model strictly follows the existing VHDL 1076 simulation cycle. The simulation of mixed continuous/discrete models is supported by a new mixed-mode simulation cycle that defines the synchronization between the analog solver and the VHDL 1076 kernel process. The analog solver establishes a sequence of solutions to the DAEs referred to as "analog solution points" or ASPs at suitable times between the time of the current discrete event and the time of the next event. A mechanism to limit the maximum timestep size is defined in a way that is independent of any simulation algorithm. Only time domain simulation is currently supported for mixed continuous/discrete models.

The VHDL 1076.1 language is required to be a superset of VHDL 1076-1993. First of all, any legal VHDL 1076-1993 description is accepted and its simulation must produce the same results as if the description was simulated with a VHDL 1076 simulator. A VHDL 1076.1 model may actually reuse all model organization, structuring, and programming facilities in VHDL 1076. It also extends some existing VHDL 1076 constructs and adds a few new language constructs and semantics as we'll see in the next section. Great care has been taken however to keep the VHDL 1076 philosophy (e.g. scope and visibility rules, declaration before use, strong typing, etc.)

III. VHDL 1076.1 Through Examples

This section intends to provide a practical introduction to VHDL 1076.1 through a number of simple, but characteristic, model examples.

A. Conservative Systems

The VHDL 1076.1 language defines special syntax and semantics to support the description and the simulation of conservative systems such as electrical systems satisfying Kirchhoff's laws. In VHDL 1076.1, the equations describing the conservative aspects are implicitly defined (in the VHDL sense), so only the so-called constitutive equations have to appear explicitly in the text of the model.

Let us take the example of a simple large-signal diode model whose constitutive equations are:

$$i_d = \mathrm{IS} \cdot \left(e^{(v_d - \mathrm{RS} \cdot i_d)/(\mathrm{N} \cdot \mathrm{VT})} - 1 \right) \qquad (1)$$

$$i_c = \mathrm{TT} \cdot i_d - 2 \cdot \mathrm{CJ0} \cdot \sqrt{\mathrm{VJ}^2 - \mathrm{VJ} \cdot v_d} \qquad (2)$$

where i_d is the ideal diode current, i_c is the associated capacitor current,v_d is the diode voltage, and the other uppercase variables are SPICE2 diode model parameters [6]. A VHDL 1076.1 description of this diode model may be done as shown in Example 1 (bold characters are VHDL 1076.1 reserved keywords.) This example actually illustrates many of the new capabilities introduced in VHDL 1076.1.

```
entity diode is
   generic (ISS, N, VT, TT, CJ0, VJ, RS: real);
   port (terminal p, m: electrical);
end entity diode;

library IEEE;
use IEEE.math_real.all;
architecture level0 of diode is
   quantity vd across id, ic through p to m;
   quantity qc: real;
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);
end architecture level0;
```

Example 1: *VHDL 1076.1 diode model*

The entity declaration defines the interface of the model. As in VHDL 1076, generic parameters hold static values, i.e. values that are known before the simulation actually begins (Parameter IS is actually denoted as ISS as IS is a VHDL reserved word.) The port declaration that follows defines two interface objects called *terminals* (more formally, *terminal ports*), in this case a plus terminal `p` (the anode) and a minus terminal `m` (the cathode). Terminals are central to the support of conservative semantics. A terminal is a new object in VHDL 1076.1 that does not bear any value. It rather defines the kind of energy that it is flowing through it. Consequently, a terminal does not have a type, but it belongs to a *nature* that represents a physical discipline. Example 1 deals with the electrical discipline, so a nature called `electrical` has been defined somewhere else (possibly in some package).

Electrical circuit theory assumes that two quantities are associated with any electrical terminal, namely a voltage and a current. These quantities are then used in constitutive equations that define the electrical behavior between terminals. They represent the unknowns of the set of DAEs that define the behavior of the complete modeled system. The set of DAEs is built from the constitutive equations, the equations derived from the structural composition of design entities, and the equations derived from the conservative semantics.

The VHDL 1076.1 language does the same, although in a more indirect way. It first introduces a new class of real-valued objects called *quantities* whose values are computed at ASPs by the analog solver as solutions of the set of DAEs. A special kind of quantities, called *branch quantities*, are defined between two terminals, a plus terminal and a minus terminal. An *across branch quantity* represents an effort like effect such as the voltage in electrical systems. A through branch quantity represents a flow like effect such as the current in electrical systems.

A nature in VHDL 1076.1, such as nature `electrical` (Example 2), defines the *types* of the branch quantities that may be associated with any terminal of that nature. As a result, strong type checking does not allow to connect together terminals of different natures.

```
subtype voltage is real;
subtype current is real;
nature electrical is
   voltage across current through;
```

Example 2: *Definition of nature electrical*

The declaration of nature `N` implicitly defines a single *reference terminal* for that nature that is designated by `N'Reference` (e.g., the ground for nature `electrical`).Then, the declaration of a terminal `T` of nature `N` automatically creates two branch quantities. `T'Reference` is an across branch quantity defined between `T` and `N'Reference`. `T'Contribution` is a through branch quantity whose value is equal to the sum of all through branch quantities that are associated with terminal `T` (with appropriate sign). In Example 1, `p'reference` represents the voltage potential of terminal `p` with reference to the ground `electrical'reference`, while `p'contribution` represents the terminal current that flows at terminal `p`. The same definitions apply to terminal `m`, *mutatis mutandis*.

The architecture body in Example 1 describes the diode behavior by expressing the constitutive equations (1) and (2). As physical quantities v_d, i_d, and i_c are defined relatively to the two diode terminals, the VHDL 1076.1 model declares the across branch quantity `vd` and the two through branch quantities `id`, and `ic` between the two interface terminals `p` and `m`. The type of a branch quantity is derived from the nature of its terminals (both terminals have to be of the same nature). The branch quantity declaration also defines that

```
vd = p'reference – m'reference
```

and that `id` and `ic` are currents in two parallel branches flowing from terminal `p` to terminal `m`.

The constitutive equations (1) and (2) are readily expressed using a new class of statements introduced in VHDL 1076.1 and called *simultaneous statements*. They come in addition to the existing sequential and concurrent statements and provide a notation for DAEs. Simultaneous statements may appear anywhere a concurrent statement is allowed. They come in several forms, the basic one being the *simple simultaneous statement*:

```
[label:] expression == expression;
```

where *`expression`* may be any VHDL expression that evaluates to a numeric value. The statement is symmetrical, so the analog solver is responsible to compute the values of all involved quantities at each ASP in such a way that both expressions are approximately equal. How "approximately equal" they are depends on tolerances that may be associated to quantities and to simultaneous statements.

In Example 1, the constitutive equations are expressed with three simple simultaneous statements. A third statement is needed since VHDL 1076.1 does not allow to write the time derivative of an expression. Hence, a second quantity declaration declares a so-called *free quantity* `qc` that is not associated with any terminal. Its role is to act as an intermediate unknown to hold a charge expression. The capacitor current `ic` is then expressed as the time derivative of the charge designated by the implicitly defined quantity `qc'dot`. Note that the three simultaneous statements may be given in any order. They will be gathered before simulation in the set of DAEs to be solved by the analog solver. The conservative semantics that are associated with terminals and branch quantities result with the following additional implicitly defined equations:

```
p'contribution == id + ic
m'contribution == –(id + ic)
```

The mathematical functions `sqrt` and `exp` are taken from the IEEE 1076.2 standard MATH_REAL package.

The application of the VHDL 1076.1 language is not limited to electrical systems only. Any conservative system for which suitable nature and across plus through branch quantities may be defined may be described in VHDL 1076.1 (Table 1). The place is unfortunately missing in this paper to present examples in other disciplines.

Nature	*Across*	*Through*
Mechanical	Velocity	Force
Rotational	Velocity	Torque
Thermal	Temperature	Heat flow rate
Hydraulic	Pressure	Fluid flow rate

Table 1: *Other possible disciplines in VHDL 1076.1*

B. Signal-Flow Modelling

Signal-flow modeling deals with abstract descriptions of continuous systems where conservative semantics are not enforced. Example 3 gives a signal-flow description of an adder-integrator.

```
entity adder_integrator is
   generic (k1, k2: real);
   port (quantity in1, in2: in real;
         quantity si: out real);
end entity adder_integrator;

architecture sfg of adder_integrator is
   quantity qs: real;
begin
   qs == k1*in1 + k2*in2;
   si == qs'integ;
end architecture sfg;
```

Example 3: *Signal-flow model of an adder-integrator*

The port list in the entity declaration defines interface objects, called *quantity ports*, that do not carry any conservative semantics. Quantity ports have a mode that is restricted to be `in` or `out`. The meaning of modes on quantity ports is different than from the one that exists for signal ports. They define legal quantity port associations and they provide a support to perform solvability checks on the description. In the latter case, mode `out` indicates that the architecture provides an equation for this quantity, while mode `in` indicates that this equation is provided in another block.

The architecture expresses the adder-integrator behavior with two simple simultaneous statements. It declares a local free quantity `qs` to hold the expression to be integrated over time, i.e. the weighted sum of the two input quantities `in1` and `in2`. In VHDL 1076.1, the integral over time, from time zero to the current time, of a quantity `q` is an implicitly declared quantity designated by `q'integ`. It is not allowed to express the integral of an expression, so the local quantity `qs` is required to make a reference to `qs'integ`.

Mixed conservative/signal-flow descriptions are possible in VHDL-1076.1. For instance, Example 1 could be augmented with another quantity port of mode `out` called `power`, so the power consumption of the diode is computed with a fourth simultaneous statement of the form:

```
power == vd * (id + ic);
```

C. Piecewise Defined Behaviour

The VHDL 1076.1 language provides a way to describe behavior that may change according to the region of operation the model is working in. Example 4 gives a simple behavior of the gain stage of an OTA. It is assumed that `gmnom` denotes the nominal transconductance, `dvmax` denotes the input saturation voltage, and `imax` denotes the output saturation current. All these values are statically determined before simulation. Also, `vinput` is the input voltage, and `gain_current` is the output current of the gain stage.

```
assert abs(gmnom*dvmax) = abs(imax);
if vinput > dvmax use
   gain_current == imax;
else if vinput < dvmax use
   gain_current == –imax;
else
   gain_current == gmnom * vinput;
end use;
```

Example 4: *OTA gain stage model*

Example 4 shows the use of the new *simultaneous if statement* to indicate which simple simultaneous statement to consider depending on the value of the input voltage. In this case, the analog solver will consider different sets of DAEs to solve that change dynamically during simulation. Each conditional branch may contain any number of simultaneous statements. The concurrent assertion statement is here to avoid any discontinuity in the quantity `gain_current`.

The VHDL 1076.1 language also provides the *simultaneous case statement* for which the selection is performed according to the evaluation of some expression.

D. Mixed-Signal Interactions

The VHDL 1076.1 language allows to write models with mixed-signal interactions. An A/D interaction may occur when a process is made sensitive to a threshold crossing in the values of some quantities. A D/A interaction may occur when the value of some signals are referred to in simple simultaneous statements.

Example 5 illustrates the A/D interaction with the model of a simple comparator.

```
entity comparator is
   generic (level: real := 2.5); -- threshold
   port (terminal ain: electrical;
         signal dout: out bit);
end entity comparator;

architecture simple of comparator is
   quantity vin across ain; -- to ground
begin
   process begin
      wait on vin'above(level);
      if vin'above(level) then -- vin > level
         dout <= '1';
      else                     -- vin < level
         dout <= '0';
      end if;
   end process;
end architecture simple;
```

Example 5: *Simple comparator model*

The entity declaration declares a port list with mixed connection points. Note that the keyword **signal** is now explicitly required.

The architecture declares the across branch quantity vin that senses the voltage between terminal ain and the (implicitly defined) electrical ground. Then, a process is made sensitive to a threshold crossing with a new implicit signal in VHDL 1076.1. For any quantity q, the boolean signal q'above(level) is TRUE if q > level and FALSE if q < level. An event occurs on the signal when the sign of the expression q – level changes.

Note that the process must execute at the exact time of the threshold crossing, which may not be representable with a value of physical type Time. The VHDL 1076.1 language introduces a unified formulation for simulation time that addresses the requirements for both continuous and discrete simulation. A common floating-point time is defined with appropriate conversion functions to and from real-valued and physical-valued times. Predefined function NOW is also overloaded to return the value of the current continuous simulation time.

Example 6 illustrates the D/A interaction with the model of a generic digital to analog converter.

```
entity dac is
   generic (vmax: real := 5.0); -- max. voltage
   port (terminal aout: electrical;
         signal din: in bit_vector);
end entity dac;

architecture simple of dac is
   function bit_to_real (b: bit_vector)
      return real is
      variable wsum: integer := 0;
   begin
      for i in b'range loop
         wsum := 2*wsum + bit'pos(b(i));
      end loop;
      return real(wsum)/2.0**b'length;
   end function bit_to_real;
   quantity vout across iout through aout;
begin
   break on din; -- announce discontinuity
   vout == vmax * bit_to_real(din);
end architecture simple;
```

Example 6: *Generic D/A converter model*

The architecture declares one across branch quantity vout and one through branch quantity iout since the output stage of the model is equivalent to an (ideal) voltage source between terminal aout and the electrical ground.

Since values on signal din will change abruptly, discontinuities are likely to appear in the time waveform denoted by quantity vout. The VHDL 1076.1 language forces the modeler to explicitly indicate that a discontinuity may occur with the new *break statement*. The break statement schedules a transaction on a new implicit signal break that will force the analog solver to reevaluate the state of the continuous part of the model. Example 6 uses the concurrent form of the break statement to force the analog solver to resume each time an event occurs on signal din.

E. Analog Discontinuities

Abstract models of physical systems are likely to exhibit discontinuities in quantities. The VHDL 1076.1 language uses its existing discrete event mechanism to handle discontinuities that may be induced in the model. Example 7 illustrates this case with the simple model of the dynamics of an infinitely rigid ball that falls down from an initial height to an horizontal ground and then bounces when it hits the ground.

```
library VHDL_AMS;
use VHDL_AMS.mechanical.all;
entity bouncer is
end entity bouncer;

architecture simple of bouncer is
   quantity v: velocity := 0.0;
   quantity s: displacement := 10.0;
   constant G: real := 9.81;
   constant Air_Res: real := 0.1;
begin
   s'dot == v;
   if v > 0.0 use
      v'dot == –G – v**2*Air_Res;
   else
      v'dot == –G + v**2*Air_Res;
   end use;
   break v => –v when not s'above(0.0);
end architecture simple;
```

Example 7: *Bouncing ball model*

The entity declaration is reduced to its simplest form since the model describes a closed system. The architecture defines a signal-flow model using free quantities as unknowns. The types velocity and displacement are floating-point types defined in package mechanical (not shown here). Two constitutive equations are needed to describe the system behavior. The appropriate equation for the acceleration is dynamically selected with a simultaneous if statement according to the sign of the velocity. The break statement specifies a new initial condition on quantity v to model an instantaneous change in the ball direction. Note that there is not any discontinuity on v'dot at crossover point v = 0.0 so no break is required by the condition on v in the simultaneous if statement.

F. Initialization

All objects in a VHDL 1076.1 model must be given an initial value before the simulation actually begins, i.e. the value at time 0.0. Constants, variables, and signals get their initial values the same way as in VHDL 1076. Quantities, on the other hand, may get their initial value in a different way. DAE theory requires that all quantities in a model have a consistent value at time 0.0 that is obtained through a special algorithm. The VHDL 1076.1 language provides an initialization algorithm that computes the quiescent state of the system. In electrical systems, this solution is called the DC operating point.

Initial values on quantities are defined in their declaration. In Example 7, the quantity `s` is given an initial value of 10.0. A new rule that only holds for quantities is that their default initial value is always 0.0. Initial values are used as starting values for the computation of the quiescent state.

Initial conditions are given as separate equations derived from a special use of the break statement. Example 8 illustrates the use of a break statement to define an initial condition on a capacitor voltage.

```
entity capacitor is
   generic (C: real; v0: real := 0.0);
   port (terminal p, m: electrical);
end entity capacitor;

architecture bce of capacitor is
   quantity vc across ic through p to m;
begin
   ic == C * v'dot;
   break v => v0 when v0 /= 0.0;
end architecture bce;
```

Example 8: *Initial condition on a capacitor model*

The generic parameter `v0` is used here as a flag to decide whether the initial condition must apply or not. The concurrent break statement is only activated during the initialization phase since it is equivalent to a process having a wait statement without any sensitivity clause. The effect of the break statement is to replace the default implicit equation `v'dot == 0.0` by `v - v0 == 0.0` during the DC operating point computation.

IV. Other VHDL 1076.1 Features

The VHDL 1076.1 language offers other capabilities that are briefly summarized here.

Mixed Netlists. Since VHDL 1076.1 provides three kinds of ports, namely signal ports, terminal ports, and quantity ports, there is a need to define how a structural hierarchy may be built using design entities with arbitrary combinations of ports. VHDL 1076.1 defines rules for the direct association (port map) of terminal and quantity ports, provided that their types match. All other possible direct port associations are not allowed, so some explicit conversion block is required. Examples are A/D and D/A converters for mixed-signal conversion, and transducers for mixed-nature conversion.

Solvability Checks. A necessary condition for the solvability of a system of DAEs is that there are as many equations as unknowns. In VHDL 1076.1, unknowns are represented by quantities, and equations are derived from simultaneous statements and from the structural composition of design entities. Rules are then defined to check the solvability at the block level using existing scoping rules in VHDL.

Frequency Domain. VHDL 1076.1 provides a support for small-signal frequency analysis. Rules are defined to derive a small-signal model from a time domain description. A way to define small-signal stimulus is also provided without having to use explicit complex numbers. A specific small-signal frequency simulation phase is added.

Tolerances. The analog solver needs some information about how accurately the DAEs must be solved. VHDL 1076.1 provides a way to define groups of quantities or simultaneous statements that belong to the same tolerance, but without specifying a tolerance value, however. The model remains neutral to any information that would be implementation dependent.

Timestep Control. VHDL 1076.1 provides a mechanism to limit the maximum size of the timestep used by the analog solver. This size may be dynamically adjusted during simulation whenever needed.

V. Conclusions

This paper provided a tutorial introduction to the VHDL 1076.1 language that provides analog and mixed-signal extensions to VHDL 1076. The language actually encompasses more than that as it is designed to address more general continuous and mixed continuous/discrete problems. VHDL 1076.1 is scheduled to become an IEEE standard in 1997.

References

[1] *VHDL Language Reference Manual*, ANSI/IEEE Standard 1076-1993, SH16840, IEEE Press, 1993.

[2] P. J. Ashenden, *The Designer's Guide to VHDL*, Morgan Kaufmann, 1996.

[3] J.-M. Bergé, A. Fonkoua, S. Maginot, J. Rouillard, *VHDL '92*, Kluwer Academic Publishers, 1993.

[4] E. Christen, K. Bakalar, *VHDL 1076.1 – Analog and Mixed-Signal Extensions to VHDL*, IEEE Proc. EURO-DAC'96 with EURO-VHDL'96, pp. , Sept. 1996.

[5] *VHDL 1076.1 Design Objective Document*, version 2.3, 1076.1 Working Group, 1995.

[6] G. Massobrio, P. Antognetti, *Semiconductor Device Modelling with SPICE*, 2nd ed., McGraw-Hill, 1993.