VHDL 1076.1 — Analog and Mixed-Signal Extensions to VHDL

Ernst Christen
Analogy, Inc.
9205 S.W. Gemini Drive
Beaverton, OR 97008
christen@analogy.com

Kenneth Bakalar
Compass Design Automation
5457 Twin Knolls Road
Columbia, MD 20845
kenb@compass-da.com

Abstract

This presentation provides an overview of the 1076.1 effort to extend the well established VHDL language to support the description and simulation of continuous and mixed continuous/discrete systems. It begins with a brief history of the effort. That is followed by an overview of the foundations: the design objectives, the base VHDL 1076 language, and the applicable mathematical theory. The body of the presentation describes the elements of the extended language. Each language element is described in the context of the 1076.1 language architecture and illustrated by a brief example. The presentation ends with selected examples illustrating the use of the language for analog and mixed-signal applications.

1. Introduction

IEEE VHDL 1076-1993 (The VHSIC Hardware Description Language) is designed for the description and simulation of digital systems, supporting modeling at various levels of abstractions. The standard, first approved in 1987 and later revised and reaffirmed in 1993, has gained wide acceptance in the last few years.

The effort to extend VHDL to support continuous and mixed continuous/discrete systems started in 1989 with the aim to submit the relevant requirements for the 1993 VHDL revision. However, the complexity of the topic required the formation of a separate working group under the auspices of the Design Automation Standards Committee of the IEEE under PAR 1076.1. Language design first started in 1993, with funding from JESSI and Rome Laboratory. It is now nearing completion, and a ballottable Language Reference Manual is expected to be available in September 1996. The extended language has informally been called VHDL-A, although we now prefer VHDL 1076.1.

2. Foundations

The VHDL 1076.1 language satisfies a set of requirements that have been documented in the 1076.1 Design Objective Document (IEEE PAR 1076.1 1995). It builds on two major foundations, the VHDL 1076 language and the theory of differential/algebraic equations (DAEs).

2.1. Language Design Objectives

The design objectives [1] have been compiled from requirements submitted by interested parties to the working group. They require the VHDL 1076.1 language to be a superset of VHDL 1076-1993 [2], supporting the hierarchical description and the simulation of continuous and mixed continuous/discrete systems with conservative and non-conservative semantics. The language is required to support modeling in electrical and non-electrical disciplines at different abstraction levels. The systems to be described are lumped systems, and the solution of the equations describing the system may include discontinuities. Interactions between the discrete part of a model and its continuous part are to be supported in a flexible and efficient manner.

2.2. VHDL 1076-1993

Being a superset of the VHDL 1076-1993 language, the 1076.1 language design is dependent on the existing 1076 language in many ways. This has both advantages and drawbacks. The VHDL 1076 language [2] provides a firm pragmatic, semantic and syntactic foundation, including structural and functional decomposition, separate compilation, a powerful sequential notation and a type system. On the other hand, it presents many constraints on the design of the 1076.1 language that can be violated only at great peril and with good reason: its scope and visibility rules, the requirement for declaration before use, the structure of names, the discrete simulation cycle, among others. The language design challenge was to judiciously build on this foundation, by re-using as much of the existing language as possible, and to extend the language only where necessary.

2.3. Theory of Differential/Algebraic Equations

The continuous aspects of the behavior of the lumped systems targeted by VHDL 1076.1 can be described by systems of ordinary differential and algebraic equations (DAEs) with time as the independent variable. These equations have the form

$$\underline{F}(\underline{x}, \mathrm{d}\underline{x}/\mathrm{d}t, t) = 0 \qquad (1)$$

where $\underline{F}$ is a vector of expressions, $\underline{x}$ is the vector of unknowns in the equations, $\mathrm{d}\underline{x}/\mathrm{d}t$ is the vector of derivatives of the unknowns with respect to time, and t is time. There is no alternative systematization with equivalent power and scope. Most such systems of equations have no analytic solution, so in practice the solution must be approximated using numerical techniques.

DAEs have been studied extensively by numerical mathematicians over the past 20 years [3]. Although there are open issues, the theory is sufficiently mature to rely on, particularly for the numerical solution of the DAEs.

It follows that VHDL 1076.1 must provide a notation for DAEs, but it does not follow that the language definition must specify a technique for their solution. To the contrary, the 1076.1 language can remain neutral with regard to the selection of numerical methods. It is sufficient for the definition of the language to describe what system of equations (at each time) is described by the text of a model. The solution algorithm itself is referred to as the “analog solver”. Only the results that the analog solver must achieve, and not its algorithm, are characterized in the language definition.

3. The 1076.1 language

In the design of the VHDL 1076.1 language we have used the existing VHDL language - its syntax, semantics, stylistic quirks, unstated principles and definitional style - in satisfying as many of the design requirements as possible, adding only what is essential and still missing. Thorough exploitation of the powerful VHDL engine in all of its aspects has led to some surprising and innovative design choices that set VHDL 1076.1 apart from other continuous system simulation languages.

3.1. Quantities

The unknowns in the collection of DAEs implied by the text of a model are analytic functions of time; that is, they are piecewise continuous with a finite number of discontinuities. At any analog solution point (ASP) the analog solver simultaneously solves for the values of all unknowns. None of the existing VHDL objects get their values in a comparable fashion. For this reason, VHDL 1076.1 introduces a new class of value-bearing objects, the *quantity*, to stand for the unknowns in the DAEs.

Quantities must have scalar subelements of a floating point type, to approximate the real numbers of the underlying formalism. A quantity object can appear in an expression or anywhere else a value of the type is allowed. In the remainder of this section, we describe the characteristics of scalar quantities. The characteristics of a composite quantity are simply the aggregation of the characteristics of its scalar subelements. The behavior of each scalar subelement is independent of the others.

Quantities can be declared anywhere a signal can be declared, except in a package. The following statement declares two quantities `q1` and `q2` of type real:

```
QUANTITY q1, q2: real;
```

A quantity can also be declared as an interface element in a port list. A quantity interface element is called a quantity port, by analogy with signal ports. Interface quantities support signal flow modeling. They have a mode, similar to the mode of an interface signal. For instance, here is the entity declaration of a signal flow model of a two-input adder with two interface quantities of mode IN and one interface quantity of mode OUT:

```
ENTITY adder IS
   PORT (QUANTITY in1, in2: IN real;
         QUANTITY sum: OUT real);
END ENTITY adder;
```

When this model is instantiated the semantics of the language constrain each interface quantity to have the same value as the quantity to which it is connected.

In addition to explicit quantities, the following quantities are implicitly defined by the language.

- The derivative of a quantity Q with respect to time: Q’Dot.
- The integral over time of a quantity Q, from time zero to the current time: Q’Integ.
- The value of a quantity Q at a fixed interval T in the past (ideal delay): Q’Delayed(T).

3.2. Conservative Systems

Systems with conservation semantics - for example, electrical systems obeying Kirchhoff’s laws - merit separate treatment because they are so commonly encountered. Special purpose syntax and semantics can provide a simplified notation that reduces the risk of errors and thereby improves productivity. In VHDL 1076.1, equations describing the conservative aspects of such a system need not be explicitly notated by the modeler. Only the so-called constitutive equations remain the modeler's responsibility.

The description of conservative systems uses a graph-based conceptual model. Take, for example, an electrical network. Here, the vertices of the graph represent equipotential nodes in the circuit, and the edges represent branches of the circuit through which current flows. In the language these ideas are expressed by *branch quantities*, which are the unknowns in the equations describing conservative systems. There are two kinds of branch quantities. *Across quantities* represent effort like effects such as voltage, temperature, or pressure. *Through quantities* represent flow like effects such as current, heat flow rate, or fluid flow rate. The constitutive equations of conservative systems are then expressed by relating the across and through quantities of one or several branches. For example, a resistor has a single branch, and its constitutive equation (Ohm's law) relates the voltage across (the across quantity) and the current through the resistor (the through quantity).

A branch quantity is declared with reference to two *terminals*. A terminal is declared to be of some simple *nature*, or of a composite nature with scalar subelements that are of a simple nature. Each simple nature represents a distinct physical discipline - electrical, thermal, hydraulic, and so on. As an example, the following statements declare two subtypes `voltage` and `current`, a simple nature `electrical`, two terminals of that nature, and an across and two through quantities between the two terminals.

```
SUBTYPE voltage IS real;
SUBTYPE current IS real;

NATURE electrical IS
      voltage ACROSS current THROUGH;

TERMINAL t1, t2: electrical;

QUANTITY v ACROSS i1, i2 THROUGH t1 TO t2;
```

The type of a branch quantity is derived from the nature of its terminals. It may be a composite type. In the example the across quantity `v`, which represents the potential difference between the two terminals, is of type `voltage`, and the type of the two through quantities i1 and i2, which represent two parallel branches, is `current`. The two terminals of a quantity must have elements of the same simple nature and must agree in other specified ways. They are termed the plus terminal and minus terminal, and the direction of the branch is plus to minus - in an electrical system, the direction of positive current flow.

A terminal may be declared anywhere a signal declaration is allowed. In particular, a terminal, like a signal, can be a port of an entity. For instance, the following statement declares the terminal ports of a diode:

```
PORT (TERMINAL anode, cathode: electrical);
```

When a model with this port list is instantiated the connection of terminal ports constructs *nodes* in the hierarchical description at which Kirchhoff's Current Law (or its equivalent in other disciplines) holds. The corresponding conservation equations are automatically extracted from the graph created by the declared branch quantities and terminals and the association of terminals into nodes.

The declaration of a simple nature creates a *reference terminal* (a "ground") which is shared by all terminals with elements of that simple nature. The reference terminal of a terminal T of nature N is designated N'Reference. The declaration of T itself creates two quantities:

- The *reference quantity* T'Reference is an across quantity with T and N'Reference as its plus and minus terminals.
- The *contribution quantity* T'Contribution is a through quantity whose value is equal to the sum of all through quantities incident to T (with appropriate sign).

If T is composite, so are T'Reference and T'Contribution, and the rules apply to each scalar subelement of T.

3.3. Simultaneous Statements

VHDL 1076.1 supplements the sequential and concurrent statements of VHDL 1076 with a new class of language statements, the *simultaneous statements*, for notating differential and algebraic equations. Simultaneous statements contain ordinary VHDL expressions that can be evaluated in the ordinary way. However, the interpretation placed on the resulting value and the effect on the value-bearing objects of the model is novel.

Simultaneous statements can appear anywhere a concurrent signal assignment is allowed. The basic form is the simple simultaneous statement, with the following syntax:

[label:] expression == expression;

For instance, the constitutive equation of a resistor could be written as `i == v/r`; where `i` and `v` are a through and an across quantity, respectively, representing the current through and the voltage across the resistor, and `r` is the resistance value. The expressions may have composite values, in which case there must be a matching subelement on the left for each subelement on the right. The expressions may refer to signals, quantities, constants, literals and functions. When the analog solver has properly established the values of each quantity, the matching subelements of the expressions will be (approximately) equal.

The language defines three additional forms of simultaneous statements.

- The *simultaneous procedural statement.* This form is merely a way to rewrite the function body f in the simultaneous statement f(q, x) == q “in line” (q is an aggregate of quantities and x is an arbitrary collection of other objects).
- The *simultaneous case* and *if statement*s allow the description of piecewise defined behavior. Each contains an arbitrary list of simultaneous statements in its statement parts, including nested simultaneous case and if statements. Only the simultaneous statements selected by the case expressions and chosen by the conditional expressions are considered by the analog solver.

The following example of an ideal switch brings together many of the new language concepts:

```
ENTITY ideal_switch IS
  GENERIC (closed: boolean := true);
  PORT (TERMINAL t1, t2: electrical);
                        -- terminal ports
END ENTITY ideal_switch;

ARCHITECTURE one OF ideal_switch IS
  QUANTITY v ACROSS i THROUGH t1 TO t2;
                        -- branch quantities
BEGIN
  IF closed USE  -- simultaneous if statement
     v == 0.0;   -- simple simult. statement
  ELSE
     i == 0.0;
  END USE;
END ARCHITECTURE one;
```

The equations explicitly denoted by simultaneous statements and the implicit equations that are a consequence of the conservation laws etc. are mapped to a single underlying formalism - the *characteristic expression*. A characteristic expression corresponds to one expression in $\underline{F}(\underline{x}, d\underline{x}/dt, t)$. Each simple simultaneous statement has a collection of characteristic expressions, one for each scalar subelement of the expression.

The analog solver determines the value of each quantity such that the values of all characteristic expressions are close to zero and thus solves the DAEs of the model. To make this possible, each quantity and each characteristic expression has an associated error tolerance. A quantity gets its tolerance from its subtype. If possible, the tolerance of a characteristic expression is determined from the tolerances of the quantities involved, otherwise it must be specified by the user. The detailed semantics and the syntax for the specification of tolerances are incomplete at the time of this writing,

3.4. Time and the Simulation Cycle

Synchronization between the analog solver and the VHDL kernel process requires a common formulation for simulation time that encompasses the requirements for both continuous and discrete simulation. This demand is met by creating a new definitional type named *Universal_Time*, by analogy with Universal_Integer and Universal_Real. Universal_Time must have sufficient precision to represent each value of the physical type Time exactly. The representation is required to equal or exceed in precision the members of the floating point class of types. The simulation cycle is recast using values of Universal_Time for the kernel variables Tc, which represents the current simulation time, and Tn, which represents the next time that a driver will become active or a process will resume. Function NOW is redefined to return the value of the current simulation time (a value of the type Universal_Time) converted with truncation to the nearest value of physical type Time. It is overloaded with another function NOW that returns the value of the current simulation time truncated to the nearest value of type Real.

The VHDL simulation cycle has been augmented to include the execution of the analog solver. The analog solver executes in each simulation cycle just before the current simulation time advances. The solver establishes a sequence of solutions to the DAEs (ASPs) at suitable intervals between the current digital time and the time of the next event. The definitions guarantee that the value of a quantity is always correct when a digital process reads the quantity, and that the value of a digital signal appearing in a simultaneous statement is always correct whenever the corresponding expression is evaluated.

3.5. A/D and D/A Interaction

If any of a set of specified quantities passes designated amplitude *thresholds* before the sequence of ASPs is extended all the way to the time of the next discrete event, the analog solver will terminate prematurely. For any scalar quantity Q the boolean signal Q’Above(level) is TRUE if Q > level and FALSE if Q < level. An event occurs on Q’Above(level) when the sign of the expression Q - level changes. Threshold crossing can be used for A/D conversions, as shown in the following example of a comparator.

```
ENTITY comparator IS
  GENERIC (level: real := 2.5); -- threshold
  PORT (TERMINAL a: electrical; -- elec.input
        SIGNAL s: OUT bit);     -- bit output
END ENTITY comparator;

ARCHITECTURE simple OF comparator IS
  QUANTITY v ACROSS a;-- across quant. to gnd
BEGIN
  s <= ‘1’ WHEN v’above(level)   -- v > level
        ELSE ‘0’;                -- v < level
END ARCHITECTURE simple;
```

A digital process that is sensitive to such a signal executes at the exact time of the threshold crossing, which may have no exact representation in physical type Time.

If a discontinuity occurs in the solution of the DAEs the analog solver must be notified. The *break* statement serves that purpose; it must be executed to generate a kind of pseudo-event at the time of each discontinuity in a model. A discontinuity will occur if, for example, a quantity is equated to a signal in a simple simultaneous statement and an event occurs on that signal. There is no known algorithm that can reliably and efficiently detect and successfully correct for discontinuities without explicit notification of the time of occurrence. Investigations are under way to determine whether an automatic notification can be provided for signal induced discontinuities. The break statement includes a provision for specifying new initial conditions for selected quantities, to be applied after the discontinuity. The following example demonstrates the use of a break statement in a D/A converter.

```
ENTITY dac IS
  GENERIC (vhigh: real := 5.0); -- a for s='1'
  PORT (SIGNAL s: IN bit;    -- bit-valued input
        TERMINAL a: electrical);-- output
END ENTITY dac;

ARCHITECTURE simple OF dac IS
  QUANTITY v ACROSS i THROUGH a;-- to gnd
BEGIN
  IF s='0' USE
     v == 0.0;      -- low output
  ELSE
     v == vhigh;    -- high output
  END USE;
  BREAK ON s;       -- announce discontinuity
END ARCHITECTURE simple;
```

3.6. Initialization

The solution of a system of DAEs in any continuous interval depends only on the values of the unknowns at the beginning of the interval [3]. These values must themselves be a solution of the DAEs, possibly augmented by other equations derived from the original ones [4]. The continuous model must be initialized using a suitable algorithm before a simulation begins and re-initialized at each discontinuity.

In the general case, there are many different initial conditions that satisfy the DAEs because during initialization there are more unknowns than equations (both $\underline{x}$ and $\mathrm{d}\underline{x}/\mathrm{d}t$ are unknowns during initialization). The selection of a particular member of the set is dependent on ancillary information, either built-in conventions or user selected.

The 1076.1 language specifies an initialization algorithm that works reasonably well for low index DAE systems, but allows an implementation to provide other initialization mechanisms. In the absence of user specified initial conditions the system of equations (1) is augmented by

$$\mathrm{d}\underline{x}/\mathrm{d}t = 0 \qquad (2)$$

i.e. the default is to find the quiescent state of the system. In electrical systems this solution is called the DC operating point. Initial conditions are specified with the break statement described earlier, for instance:

```
BREAK q1 => expression1, q2 => expression2;
```

The effect of specifying initial conditions is that suitable equations from the set (2) are replaced by the initial conditions. For the example they are the equations $\mathrm{d}q1/\mathrm{d}t = 0$ and $\mathrm{d}q2/\mathrm{d}t = 0$. There are a number of unresolved issues, because the system of equations formed from (1) and (2) sometimes has no solution, and often the solution is not unique.

For mixed continuous/discrete systems the result of the initialization is defined to have the following characteristics:

- The values of all quantities satisfy the system of equations described by (1) and (2), with initial conditions taken into consideration as described above.
- There are no pending A/D and D/A interactions at time 0.0.

We repeat that an implementation is allowed to provide alternative initialization schemes. Re-initialization after a discontinuity is defined similarly to initialization at time 0, except that (1) is augmented with equations that keep selected quantities at their t^- values.

3.7. Miscellaneous

VHDL 1076.1 contains a variety of other facilities that we briefly summarize here.

Number of Equations. A necessary condition for the solvability of the system of equations (1) is that there be as many equations as unknowns. The definition of the 1076.1 language includes rules enforceable for each model that guarantee that this condition is satisfied.

Frequency Domain. A designer is often interested in the behavior of a continuous system in the frequency domain. The 1076.1 language will include definitions that support such simulations in a portable way, based on the small-signal model derived from the equations (1). The semantics of the corresponding language elements and their syntax is incomplete at the time of this writing.

Mathematical Functions. Such functions are not part of the 1076.1 language proper, but are available through the 1076.2 standardization effort as a VHDL package.

4. Examples

We will demonstrate the use of the VHDL 1076.1 language with two examples. The first example is a diode with self heating. The diode consists of two electrical branches between its anode and cathode, and a thermal branch between its junction and thermal ground.

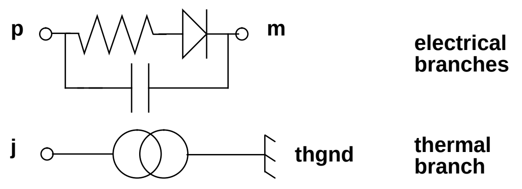

Figure 1. Diode with self heating

For the electrical branches `voltage` and `current` are the across and through types, as described in section 3.2. For the thermal branch the corresponding types are `temperature` and `pwr`. The VHDL 1076.1 implementation of the model is:

```
USE disciplines.electrical_system.all;
USE disciplines.thermal_system.all;
USE ieee.math_real.all;
ENTITY diode_th IS
  GENERIC (is0: real := 1.0e-14;
         n, area: real := 1.0;
         tau, cj0, phi, rd: real := 0.0);
  PORT (TERMINAL p, m: electrical;
        TERMINAL j: thermal);   -- junction
END ENTITY diode_th;

ARCHITECTURE one OF diode_th IS
  QUANTITY v ACROSS id, ic THROUGH p TO m;
  QUANTITY q: charge;     -- junction charge
  QUANTITY vt: voltage;   -- thermal voltage
  QUANTITY temp ACROSS power THROUGH thgnd TO j;
  CONSTANT boltzmann: real := 1.3806226e-23;
  CONSTANT electron_charge: real:= 1.602191e-19;
BEGIN
  id == area*is0*(exp((v-rd*id)/(n*vt)) - 1.0);
  q  == tau*id - 2.0*cj0*sqrt(phi*(phi-v));
  ic == q'dot;
  vt == temp * boltzmann / electron_charge;
  power == v*id;
END ARCHITECTURE one;
```

In the architecture we declare the electrical and thermal branches, two free quantities and two physical constants. The first simultaneous statement defines the current in the resistive branch; it depends on the thermal voltage `vt`. The second and third statements define the current in the capacitive branch. `vt` depends on the temperature at the junction, as shown in the fourth statement, and finally the power dissipated in the diode is the value of the through source that forms the thermal branch.

The second example is a silicon-controlled rectifier. It turns on when it is forward biased and the control voltage is larger than the on voltage. It turns off when the current is smaller than the holding current and the control voltage is smaller than the on voltage. These conditions are expressed by the value of signal `off`. Current is flowing through the SCR when it is on and the voltage across the SCR is larger than `vdrop`. This condition is encoded in signal `zero_current`, whose value controls which of the two simple simultaneous statements is selected. The break statement announces the discontinuity that occurs when `zero_current` changes its value.

```
ENTITY scr IS
  GENERIC (vdrop: voltage := 0.7; -- On vlt. drop
        von: voltage := 0.7;  -- Turn on voltage
        ihold: current := 0.0;-- Holding current
        ron: resistance := 0.1e-9); -- On res.
  PORT (TERMINAL an, cath, gate: electrical);
END ENTITY scr;

ARCHITECTURE ideal OF scr IS
  QUANTITY vscr ACROSS iscr THROUGH an TO cath;
  QUANTITY vcontrol ACROSS gate TO cath;
  SIGNAL off, zero_current: boolean := true;
BEGIN
  off <= true WHEN NOT (vcontrol'Above(von) OR
                       iscr'Above(ihold)) ELSE
         false WHEN vcontrol'Above(von) AND
                    vscr'Above(0.0);
  zero_current <= off OR NOT vscr'Above(vdrop);

  IF zero_current USE
     iscr == 0.0;
  ELSE
     iscr == (vscr - vdrop) / ron;
  END USE;
  BREAK ON zero_current;
END ARCHITECTURE ideal;
```

5. Summary

We have given an overview of the VHDL 1076.1 language scheduled to become an IEEE standard in 1997. We believe that the language is flexible enough to address many if not all of the requirements imposed by continuous and mixed continuous/discrete simulation problems.

References

[1] *VHDL 1076.1 Design Objective Document*. IEEE 1076.1 Working Group, 1995.

[2] *IEEE Standard VHDL Language Reference Manual*. ANSI/IEEE Std 1076-1993.

[3] Brenan, K.E., S.L.Campbell, and L.R.Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. North-Holland, 1989.

[4] Pantelides, C.C. "The Consistent Initialization of Differential-Algebraic Systems". *SIAM J.Sci.Stat.Comput., 9*(2): 213-231, March 1988.