7

A semantic model for VHDL-AMS

Natividad Martínez Madrid, Peter T. Breuer, Carlos Delgado Kloos
Área de Ingeniería Telemática, Universidad Carlos III de Madrid
Butarque 15, E-28911 Leganés, Madrid, Spain.
Tel/Fax: +34(1)624-9953/9430, Email: nmadrid@it.uc3m.es

Abstract

This paper examines the analog extension to VHDL currently being discussed by the VHDL 1076.1 Working Group and proposes an underlying semantic model for it. The current white papers contain much discussion of syntactic and operational details and an abstraction away from that level of detail is presented here. This semantic model for VHDL-AMS extends our generally accepted semantics for VHDL (Breuer et al. 1995, Breuer & Martínez Madrid 1995) in a natural way. It integrates the imperative, discrete-event semantics of VHDL and the declarative, continuous semantics of the AMS extension. The former is based on imperative pre-emptive scheduling primitives, and the latter is based on differential equations for varying real-valued quantities.

Keywords

VHDL-AMS, process algebra, denotational semantics, differential equations

1 INTRODUCTION

The aim of this paper is to set out a semantic model for VHDL-AMS, the analog and mixed-signal extension to the IEEE standard hardware description language VHDL (IEEE 1994). VHDL itself supports only digital hardware and discrete time. VHDL-AMS is being developed by IEEE working group 1076.1.

The VHDL-AMS extension aims to support the description and simulation of continuous and mixed continuous/discrete systems. Extending an existing language has both advantages and disadvantages. One disadvantage is that the AMS extension is constrained to preserve the original syntax and its 'simulation cycle' semantics. One advantage is that building upon VHDL brings technical and exploitation-oriented gains. Only the extension to the language needs designing and the barriers to its acceptance will be low; the expertise in the existing user-base will transfer easily into the new technology.

The VHDL-AMS LRM draft expresses the semantics of the language through a natural language description of the basic simulation cycle. This is satisfactory for a working document that must be accessible to a large forum. But, to

Table 1 Properties associated with the VHDL-AMS name space division

variables	assigned instantaneously	cannot be scheduled
signals	cannot be assigned instantaneously	preemptively scheduled
quantities	can be reinitialized	governed by diff. eqns.
processes	invariant	invariant

be able to perform formal manipulations (property verification, refinement, and so on) on VHDL-AMS, a formal semantics of the language is required. A formal semantics is also required simply to clarify the current language draft and to indicate possible lines of development or problematic areas in the present natural language description.

This paper presents a semantic model for VHDL-AMS which is consistent with the formal semantics that we have developed for VHDL (Breuer et al. 1997, Breuer et al. 1995, Breuer & Martínez Madrid 1995). Minimal changes have been introduced in the existing framework. The chief difficulty lies in the integration of the imperative and discrete-event semantics of VHDL with the declarative, continuous-time semantics of the AMS extension. And a further complication arises from the fact that VHDL contains scheduling constructs. The latter obliged us in Breuer et al. (1995) to consider VHDL state as extending across time from a factual past into a hypothetical future. This 'state' (we call it a *world line*) is modified at future time points by the action of VHDL scheduling imperatives. The AMS extension permits the user to define quantities in terms of connecting sets of differential equations acting at every moment in time and therefore affects the whole of a world line. Our tactic has been to rewrite the semantic equations for VHDL scheduling imperatives, which rewrite world lines, as the composition of two functionalities:

1. a *forced mode* update corresponding to the imperative scheduling update of VHDL in which values are perturbed from their originals at a future time;
2. a subsequent *relaxed mode* update in which all quantities 'relax' after the perturbation to a quiescent configuration that is again consistent at every moment in time with the equations declared for them in the AMS extension.

Both actions take place across time (and space, in the sense of name-space), in accordance with the unmodified VHDL semantics. In the absence of any governing differential equations the semantics is exactly that of VHDL. In the presence of some differential equations, it is necessary to distinguish between different areas of the name space (see Table 1).

The first two entries in Table 1 are classic VHDL concepts and their semantics is unchanged in the AMS extension even in the presence of governing equations. VHDL-AMS introduces a new concept known as *quantities* and it is these alone that are updated in the relaxed mode phase of updates.

The major changes to our existing semantic model for VHDL are:

1. the discrete time domain has been replaced by a continuous time domain;
2. the functionality of VHDL assignments has been extended as described above to incorporate a phase in which the analog solution of differential equations is carried out;
3. the semantics now also carries an equational environment that governs the relaxed mode development of quantities.

The semantics has been prototyped in the functional language Gofer (Jones 1991), an interpreted variant of Haskell.

VHDL-AMS has just recently arrived on the scene and the authors are aware of only one other research activity related to formal methods and VHDL-AMS. That work develops a model of VHDL-AMS in evolving algebras (EA) (Sasaki 1997). That research extends an existing model for VHDL in terms of EAs given by Müller, Börger et al. in the collection Delgado Kloos & Breuer (1995).

An analysis of the process semantics of VHDL and VHDL-AMS is given in Section 2. A denotational semantics for individual processes is presented in Section 4. Sections 3 and 5 discuss the VHDL-AMS analog solver in, respectively, overview and detail.

2 A PROCESS ALGEBRAIC ANALYSIS

The semantic changes with respect to VHDL introduced by our model of VHDL-AMS fall into two subclasses: conservative extensions of existing parts of the semantics of VHDL, and the distinctly new semantics for the new introductions of VHDL-AMS, including a semantics for the *analog solver*.

The basic compositionality of VHDL is preserved in VHDL-AMS. A VHDL model consists of processes running in parallel, each containing imperative commands running sequentially in a continuous loop. According to the semantics given in Breuer et al. (1995), the functionality of the whole is the composition via parallel and sequential (and looping) operators of the parts. This basic compositionality is unchanged, apart from a change in the underlying semantic domains: time is now extended to a continuous domain of values. The following is an analysis in terms of process algebra of the modelling of VHDL and VHDL-AMS processes. We show formally that it is indeed possible to model them in such a way that the semantics of the complete system is given by the parallel synchronous composition of our semantics for each process, thus laying the basis for the rest of this paper.

According to both VHDL and VHDL-AMS standards, processes communicate uniquely through a managing kernel. That communication takes place via *signals*. Variables are local to a process*. Conceptually, signals correspond to the information carried on circuit wires, but VHDL offers a more opera-

*The 93 standard extends the 87 standard to also allow some sharing of variable at startup but we regard it as a mistaken extension and choose to ignore it here!

tional view of signals. A VHDL process schedules changes of level on a signal at a future time, thus achieving any desired waveform.

The VHDL standard views the simulation of a completed VHDL model in terms of a monolithic controlling kernel which executes a simulation cycle. It repeatedly advances time to that of the next scheduled event, then checks if the event unblocks any processes. If so, it executes those processes (in zero time), allowing them to schedule future events, until they all become blocked again, waiting on some event. A new cycle then starts.

VHDL-AMS augments this simulation cycle with an 'analog solver'. The solver appears as a special process that runs during the passive phase of each simulation cycle, when all normal processes are waiting for time to advance to the next scheduled event. The analog solver generates solutions for the quantities governed by differential equations. The solution points are distributed throughout the time interval leading up to the next due digital event. The analog solver also detects if any quantity crosses a predetermined threshold; if so, it signals that as a digital event (on one of a set of dedicated signals) and stops its calculations for the current cycle, giving back control to the kernel.

The following is an analysis of VHDL and VHDL-AMS simulation cycles in terms of CSP. We show first that a VHDL model has the semantics of the synchronous parallel composition of the semantics for its component processes. In other words, the VHDL kernel can be eliminated by incorporating a local partial copy into each process as demontrated below. We then show that the VHDL-AMS model can be treated in a similar way. Each process model gets a copy of a fragment of the kernel plus a copy of the VHDL-AMS analog solver.

2.1 VHDL

Let A be an alphabet of basic scheduling requests on signals that the VHDL simulation kernel K accepts. The kernel alphabet consists of these requests plus a global simulation clock, plus accesses to the system state Σ imparted to the various processes under its control:

$$\alpha(K) = A \sqcup \mathit{clk} \sqcup \Sigma \tag{1}$$

We suppose:

(i) that each VHDL process P_i uniquely schedules events on a *nonempty* set of output signals S_i that are its own, and the other VHDL processes only 'listen' to these signals. This is an approximation to the truth, but all VHDL codes can be reformatted to observe this restriction.*

*VHDL allows the use of *resolution functions* which combine several signals into one, so two processes *can* write to one signal. But resolved signals can be removed from a VHDL code via renaming and replacement of the resolved signal by the resolution function expression.

(ii) that each VHDL process *does* schedule a change in some signal it owns in every possible trace that it can engage in. This condition is satisfied by initializing the signals.

Every VHDL process has to initialize its signals. Condition (i) ensures there are some to initialize and therefore (ii) holds.

The kernel directly communicates state changes to all processes. The state Σ consists essentially of the kernel calculation of signal values. The kernel always accepts scheduling requests from process P_i on the signals S_i unique to P_i. Scheduling requests cannot block. Processes P_i do not try to schedule on any other signals apart from S_i.

To summarize, the alphabet of scheduling events A is the disjoint sum of alphabets A_i which deal with scheduling events concerning particular disjoint sets of signals S_i. The alphabet A_i is associated with P_i in the sense that S_i is exactly the set of signals scheduled by P_i.

$$A = \bigsqcup_i A_i \tag{2}$$

The process P_i writes scheduling requests A_i on its signals S_i to the kernel and reads directly via the kernel clock events and the state $\Sigma = \bigsqcup_i \Sigma_i$ of signals:

$$\alpha(P_i) = A_i \sqcup \Sigma \sqcup clk \tag{3}$$

The kernel is impartial. On receiving a request from process P_i, it modifies the state corresponding to the schedule for S_i in the way indicated by P_i, and communicates maturing changes to all other processes at the right time.

$$K = |||_i K_i \tag{4}$$

$$\alpha(K_i) = A_i \sqcup \Sigma_i \sqcup clk \tag{5}$$

where K_i reads a scheduling request from process P_i on signals S_i using alphabet A_i and writes to other processes the resulting state Σ_i. The alphabet of K_i is precisely A_i and the state Σ_i plus the simulation cycle counter (Figure 1(a)).

Interleaving semantics is appropriate because the order in which the kernel accepts requests from different processes is not important. It updates its internal state and moves on its internal counter only after having treated all pending requests. VHDL is deterministic in this respect.

For each process P_i, define a process P_i' which consists of P_i in synchronous communication with the kernel fragment K_i dealing with its signals. Both intercommunicate only on the signals S_i scheduled by P_i and on the clock:

$$P_i' = (P_i \ |[A_i \sqcup \Sigma_i \sqcup clk]| \ K_i) \backslash A_i \tag{6}$$

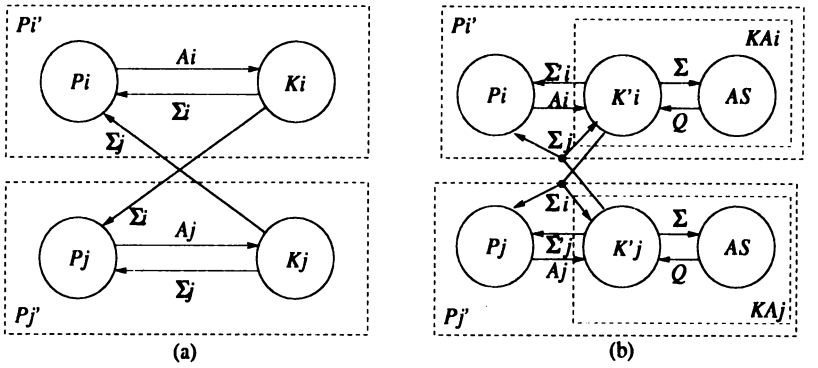

Figure 1 Communication in (a) VHDL and (b) VHDL-AMS process models. The digital kernel fragment K_i accepts scheduling requests A_i on the signals S_i of process P_i. The digital kernel communicates the state Σ_i of signals S_i to all processes. The analog kernel also communicates the state Q of quantities.

with alphabet:

$$\alpha(P_i') = \Sigma \sqcup clk \tag{7}$$

Proposition 1 *A VHDL model is the parallel synchronous composition of models P_i' for each of its processes. The process models do not intercommunicate on scheduling requests.*

Proof:

$$\begin{aligned}
\text{VHDL} &= (|||_i\ P_i)\ ||\ K \\
&= |||_i\ (P_i\ |[A_i \sqcup \Sigma \sqcup clk]|\ K) && \text{[|| distributes over |||]} \\
&= |||_i\ (P_i\ |[A_i \sqcup \Sigma \sqcup clk]|\ (|||_i K_i)) && \text{[definition of } K\text{]} \\
&= |||_i\ ((P_i\ |[A_i \sqcup \Sigma \sqcup clk]|\ K_i)\ ||| \\
&\qquad (P_i\ |[A_i \sqcup \Sigma \sqcup clk]|\ (|||_{j \neq i} K_j))) && \text{[distributivity again]} \\
&= |||_i\ (P_i\ |[A_i \sqcup \Sigma \sqcup clk]|\ K_i) && \text{[RH conjunct blocks by cond. (ii) on } P_i\text{]} \\
&= |||_i\ (P_i\ |[A_i \sqcup \Sigma \sqcup clk]|\ K_i) \backslash A_i && [K_j \text{ blocks on } A_i,\ j \neq i] \\
&= ||_i\ P_i' && \text{[equal alphabets]}
\end{aligned}$$

□

2.2 VHDL-AMS

Despite the description of the solver as a kind of process in the VHDL-AMS standard, the fact that it runs when other processes do not makes it easier to view it as part of the kernel, which also runs when all processes sleep. We view the kernel KA of the VHDL-AMS model as two cooperating processes:

$$KA = K' \,||\, AS \tag{8}$$

in which K' is the VHDL kernel with a slightly enlarged state space. The two parts communicate information Σ' on the state of the VHDL signals Σ and also information Q on the state of the quantities controlled by the solver:

$$\Sigma' = \Sigma \sqcup Q \tag{9}$$

Not considering the extension to state, the kernel has the same alphabet as in VHDL in that it accepts scheduling requests A from the processes on ordinary signals and reports to them on the state Σ', which now contains both signals and quantities. The solver does not accept scheduling requests.

$$\alpha(KA) = A \sqcup \Sigma' \sqcup \mathit{clk} \tag{10}$$
$$\alpha(K') = A \sqcup \Sigma' \sqcup \mathit{clk} \tag{11}$$
$$\alpha(AS) = \Sigma' \sqcup \mathit{clk} \tag{12}$$

The VHDL part K' is perfectly accepting with respect to quantity values, determined by the solver. The solver is perfectly accepting with respect to ordinary signal values, determined by the VHDL kernel calculation of when and what to update, given the scheduling requests that it has received.

The monolithic kernel can be broken into parts with a restricted alphabet:

$$\begin{aligned} KA &= K' \,||\, AS \\ &= (|||_i K'_i) \,|[\Sigma' \sqcup \mathit{clk}]|\, AS && \text{[definition of } K'\text{]} \\ &= |||_i (K'_i \,|[\Sigma' \sqcup \mathit{clk}]|\, AS) && \text{[}||\text{ distributes over }|||\text{]} \\ &= |||_i KA_i \end{aligned}$$

In which

$$KA_i = K'_i \,|[\Sigma' \sqcup \mathit{clk}]|\, AS \tag{13}$$
$$\alpha(KA_i) = A_i \sqcup \Sigma' \sqcup \mathit{clk} \tag{14}$$
$$\Sigma'_i = \Sigma_i \sqcup Q \tag{15}$$

and then exactly the same reasoning as before allows us to express VHDL-AMS semantics as the composition of semantic models for each process that

also incorporate within themselves a fragment of the monolithic kernel and a complete copy of the analog solver.

For each process P_i, define a process P_i' which consists of P_i in synchronous communication with the fragment KA_i of the VHDL-AMS kernel that deals with its signals. They communicate meaningfully only on the signals S_i scheduled by P_i plus all quantities and on the system clock (Figure 1(b)):

$$P_i' = (P_i \ \ [[A_i \sqcup \Sigma_i' \sqcup clk]] \ \ KA_i) \backslash A_i \tag{16}$$

with alphabet:

$$\alpha(P_i') = \Sigma' \sqcup clk \tag{17}$$

Proposition 2 *A VHDL-AMS model is the parallel synchronous composition of models P_i' for each of its processes.*

Proof: As in Proposition 1. □

The copies of the analog solver in the processes need not communicate quantity values, because each copy of the solver is deterministic, depending only on the signal values for its output, which are globally shared for reading.

3 THE ANALOG SOLVER IN OUTLINE

The analog solver is the main addition in VHDL-AMS, and it deserves special attention. The aim of this subsection is to give an initial abstraction of the behaviour of the analog solver, described in operational terms by the VHDL-AMS standard-to-be. Later sections (see 5) will go into more detail.

The analog solver calculates the value of all quantities at every moment in time. At every point on the trajectory through the state space formed by the state and time the governing differential equations are satisfied. Equations relate only the current values of quantities and their derivatives and the values of signals. In the following, *State* contains the values of the signals and quantities at every moment in time. Let s_0 be the current state at time t_0. Let t_1 be the due time of the next digital event beyond t_0. Let

$$s :: [t_0, t_1] \to State$$

be the unique analytic solution with $s_{t_0} = s_0$ on the quantities of interest to the differential equations *eqs* in the nonzero interval $[t_0, t_1]$, with t_1 sufficiently large. Let p be a predicate to detect a stopping condition given the current state and time. Then the VHDL-AMS solver has the following functionality:

$$\begin{aligned} &[-] :: EqnSet \to ((State, Time) \to Bool) \to (State, Time) \to (State, Time) \\ &[eqs]p\ (s_0, t_0) = (s_{t'}, t') \text{ s.t. } \ t' \in (t_0, t_1), \ \ p(s_{t'}, t'), \ \ \forall t \in [t_0, t') \cdot \neg p(s_t, t) \end{aligned} \tag{18}$$

or (s_{t_1}, t_1) if there is no time point in (t_0, t_1) satisfying p.

In the view taken by the VHDL-AMS standard-to-be, signal values remain constant for each run of the analog solver. I.e., the passive phase in which the solver runs lasts until at most the time t_1 of the next scheduled event. Therefore, the solver needs only the initial state of the signals, and not the whole of the signal drivers, for its proper functioning. There is no scheduled change of signal values in the interval over which the solver runs.

Our idea is generalize the solver to act over a longer time, taking into account the scheduled changes of signal state. We remove the stopping predicate p and run up to any desired time t by extending the solution s piecewise (analytically on the quantity values). Given the schedule $w_0 :: Time \to State$ and a current time t_0, let the sequence of changes in state due to occur according to the schedule be at $t_1, t_2, \ldots$ and let s_n^+ be the state of the signals scheduled at time t_n^+ in w_0. Let

$$s :: [t_0, \infty) \to State$$

be the unique solution to the differential equations *eqs* in the intervals $[t_n, t_{n+1})$ with $s_{t_0} = s_0$ and $s_{t_n} = s_n^+$ on signals, with the quantities taking certain values explained in Section 5 through the joins on the intervals. Then

$$[-]' :: EqnSet \to (Time \to State, Time) \to Time \to State$$
$$[eqs]'\ (w_0, t_0)(t) = s_t, \qquad t > t_0 \tag{19}$$

and we can recover the VHDL-AMS implementation of the solver over the interval $[t_0, t_1)$ from this function. Even if the extrapolation on the quantities is wrong, the expectation is that the extrapolation will we evaluated again before it is needed. Only the quantity at the current time can be accessed by processes, and the process must be in the digital part of its code when it makes the access, and therefore it must have passed a due event in the schedule to get to that point (the due event that woke it from the sleep during which the analog solver performs its work).

4 DENOTATIONAL SEMANTICS

This section describes our existing semantics for digital VHDL (Breuer et al. 1995) and extends the semantics to VHDL-AMS.

4.1 Domains

According to (Breuer et al. 1995), the state of a VHDL model is modelled by a *world line*, a term borrowed from general relativity, plus a *current time*

point. A world line represents the VHDL standard concept of a *signal driver.* A signal driver consists of a set of scheduled changes, and a world line consists of the time-varying values that result from applying the scheduled changes in the future plus the history of the recorded variation of the signal values. The current time point indicates the *current world.* It contains the current state of signals, and any other components of state.

$$WorldLine = Time \rightarrow State \tag{20}$$

$$State = Id \rightarrow Value \tag{21}$$

$$Semantics = (WorldLine, Time) \leftrightarrow (WorldLine, Time) \tag{22}$$

The basic semantic domain is a relation between world-line/time-point pairs. This is a domain with the empty relation as its top element and the full relation as its bottom element; the order is superset.

The semantics is relational in order to allow for non-determinism.* This will allow us to make a direct connection with the trace semantics of CSP and the process-level description set out in the previous section.

In VHDL, time is measured in integer multiples of a basic unit.

$$Time = Int$$

The minimal unit in VHDL is the femto second. No changes may take place in VHDL in any smaller real positive interval.*

The changes that have to be made in the above framework in order to accommodate VHDL-AMS are minimal. The semantics is also relational, but the system now includes a set of differential equations *EqnSet*. In the LRM this set of equations is considered as global, but, in order to facilitate the analysis here:

(i) we consider that the quantities are divided up in such a way that there is only one process that may change the set of equations that govern the evolution of a particular quantity;
(ii) we consider also that no more than one process attempts to read the value of one quantity.

The second is the major restriction. In other words, every quantity 'belongs'

*As it happens, VHDL is – intentionally – deterministic, but this is not directly built into even the operational semantics given in the VHDL standard, although it is stated as an aim. Determinism derives from the detailed semantics of the simulation cycle and, in fact, strict determinism was lost with the '93 revision of the standard, which allowed writing to shared global variables at start-up, but we are not considering that aspect of that revision here.

*In full VHDL, changes may also take place in zero time – so-called *delta time.* But still, no change may take place in any *positive* time less than that of the minimum unit.

to one and only one process. (i) says that only one process may 'write' to the quantity. (ii) says that only that one process may read the quantity. If the value is to be communicated to other processes, it must be signalled to them. The restriction is a convenience here that allows us to associate the governing equations for each set of quantities with one and only one process.

$$Semantics = (EqnSet, WorldLine, Time) \leftrightarrow (EqnSet, WorldLine, Time) \quad (23)$$

As implied, time now is continuous:

$$Time = \mathcal{R}eal \quad (24)$$

That is, changes may take place as close together as may be required. In addition, the *State* underlying *WorldLine* in the case of VHDL-AMS is the 'extended state' that includes information on quantity values as well as the signal values that comprise VHDL *State*.

The following subsections give the semantics for three kinds of statement: *wait*, *scheduled assignment*, and their *sequential* and *parallel* compositions. A process consists of a looped statement with at least one wait statement inside the loop. Parallelism is only one level deep in VHDL and in VHDL-AMS.

Because processes never terminate, it is appropriate to consider a semantics which runs a process to a point at which its execution may be momentarily suspended in order that its state may be examined. This semantics will be called the *suspension semantics* (Breuer et al. 1995). At the point of suspension the process has not yet terminated its execution. It may be restarted and allowed to continue to another point at which it may be suspended again.

Processes and statements both possess two distinct semantics: the suspension semantics and the standard semantics of execution to termination (*termination semantics*). These lie in the same modelling domain, *Semantics*, but (usually) differ for each element of the language. The rules of composition and the atomic statement semantics will be given in the following subsections.

4.2 Composition Semantics

Sequential composition in the termination semantics ($\mathcal{S}$) is just the composition of relations:

$$\mathcal{S}[a; b] = \mathcal{S}[a]; \mathcal{S}[b] \quad (25)$$

but the composition in the suspension semantics ($\mathcal{P}$) is more interesting:

$$\mathcal{P}[a; b] = \mathcal{P}[a] \cup \mathcal{S}[a]; \mathcal{P}[b] \quad (26)$$

That is, a sequence of two statements can *either* suspend in the first statement *or* the first statement can run to termination and then the second statement may run to suspension from that point.

This immediately gives rise to a suspension semantics for infinite loops – which is all we are interested in, because VHDL and VHDL-AMS processes are infinite loops:

$$\begin{aligned} \mathcal{P}[\texttt{while(true)do } a \texttt{ end}] &= \mathcal{P}[a] \cup \mathcal{S}[a]; \mathcal{P}[\texttt{while(true)do } a \texttt{ end}] \\ &= \mu R : \mathit{Semantics} \cdot R = \mathcal{P}[a] \cup \mathcal{S}[a]; R \end{aligned} \tag{27}$$

in which μ is the least fixpoint operator in the domain. This is the 'largest' (least specified) relation that satisfies the semantic equation (the full relation is the bottom element in the domain). The termination semantics of an infinite loop is the empty relation:

$$\mathcal{S}[\texttt{while(true)do } a \texttt{ end}] = \{\} \tag{28}$$

Note that there should be no loops inside a VHDL or VHDL-AMS process. The process itself is an infinite loop on its interior statements. One can legally write VHDL code that has interior loops, but in general it will not be synthesizable to hardware and therefore should not be written! VHDL processes are intended to represent simple finite state machines that can be implemented as circuit elements, and there is no point in them containing interior loops.

This semantics of the sequencing and looping operators is common to both VHDL and VHDL-AMS. The only difference is the underlying semantic domains on which the relations are based.

The semantics of parallelism is also the same in both VHDL and VHDL-AMS (only processes can legally be placed in parallel, but for completeness we give a hypothetical semantics to parallelism at the statement level too). We have shown in Section 2 that models in both languages are composed of the synchronous parallel composition of models for the individual processes which communicate on the signal (and quantity) states and the simulation time. Intersection of relations is therefore an adequate representation:

$$\mathcal{S}[a||b] = \mathcal{S}[a] \cap \mathcal{S}[b] \tag{29}$$

$$\mathcal{P}[a||b] = \mathcal{P}[a] \cap \mathcal{P}[b] \tag{30}$$

The only questionable element in this representation is that – in VHDL-AMS – it distinguishes processes by their action on governing equations, as well as by their actions on signals and quantities. We believe that to be appropriate.

4.3 Wait Semantics

A wait statement makes time advance. It schedules no changes in the driver, it just moves time forwards until a given predicate is satisfied.

In digital VHDL, there are three kinds of wait conditions: wait on a change in a signal, wait for a certain amount of time, and wait until an arbitrary predicate becomes true. The last subsumes the other two in the sense that the same semantic function can cover all these variations.

$$\begin{aligned}(e, w_0, t_0)\mathcal{S}[\textit{wait on } p](e, w_1, t_1) &= w_0 = w_1 \\ &\wedge\ wt_1 \models p \wedge \forall t \in [t_0, t_1) \cdot wt \not\models p \end{aligned} \tag{31}$$

$$\begin{aligned}(e, w_0, t_0)\mathcal{P}[\textit{wait on } p](e, w_1, t_1) &= w_0 = w_1 \\ &\wedge\ \forall t \in [t_0, t_1] \cdot wt \not\models p \end{aligned} \tag{32}$$

VHDL-AMS introduces a new kind of wait statement: a process can wait on a quantity crossing a certain threshold. This threshold crossing generates a digital signal event on a reserved signal and also aborts the solver early, but it is not necessary to emulate that particular signalling convention. We have direct access to the predicted values of quantities (and signals) via the extended world lines of the VHDL-AMS model. In our semantic model, quantities are scheduled along with signals by the assignment statement during the ordinary course of execution of a process and therefore the new *wait above* variant needs no new functionality in order to be able to wait on a quantity rather than a signal condition. The description above covers the VHDL-AMS extension.

4.4 Assignment Semantics

According to the language standard, digital VHDL signal assignment statements change the values scheduled in a *signal driver*. They are non-blocking, and take no real time to execute. The semantics given in Breuer et al. (1995) for VHDL therefore gives to them the suspension semantics of the empty relation (i.e., they cannot suspend).

$$\mathcal{P}[x \Leftarrow y \text{ after } \tau] = \{\,\} \tag{33}$$

The termination semantics is nonempty. It expresses a zero change in the current time point and a deterministic overwriting of the appropriate signal driver at all future time points at and beyond the scheduled point in time.

$$(w_0, t_0)\mathcal{S}[x \Leftarrow y \text{ after } \tau](w_1, t_1) = t_0 = t_1 \wedge w_1 = \textit{force\_assign}(w_0) \tag{34}$$

where the function *force_assign* performs the required signal update on x with the value of expression y (evaluated at t_0 in w_0) at every point $t \geq t_0 + \tau$.

Our denotation of assignment in VHDL-AMS covers quantity scheduling as well as signal scheduling. Some of the functionality of the analog solver is incorporated into each assignment statement. In detail, the local copy of the analog solver (see Section 2) in each process predicts the future quantity values whenever a scheduling assignment occurs. It does not matter that most of the values calculated will be replaced before they are ever seen. The aim is simply to capture the functionality concisely in the semantics.

We call the scheduling assignment to quantities *relaxed-mode* assignment, in contrast to the *forced-mode* assignment of signals. Scheduling assignments first schedule signals (preemptively – i.e., overwriting previously scheduled values). All quantities then 'relax' into a new configuration consistent with the scheduled signal and the current set of differential equations.

$$\begin{aligned}(e, w_0, t_0)\mathcal{S}[x \Leftarrow y \text{ after } \tau](e, w_1, t_1) &= t_0 = t_1 \\ &\wedge \; w_1 = \mathit{update}(e, t_0)(\mathit{force\_assign}(w_0))\end{aligned} \tag{35}$$

and the function *update* is described in (47) of Section 4.

5 THE ANALOG SOLVER

The semantic model set out here puts a copy of the analog solver into each process, along with a fragment of the digital kernel. In detail, the analog solver functionality is embedded in the semantics of the scheduling assignment.

This *relaxed-mode* update is based on a series of analytic extensions from an initial state. A set of equations first gives rise to a *local approximation*.

$$\begin{aligned}\mathit{LocalApprox} &= (\mathit{Time}, \mathit{State}) \leftrightarrow (\mathit{Time}, \mathit{State}) \\ \mathit{localApprox} &:: \mathit{EqnSet} \rightarrow \mathit{LocalApprox}\end{aligned} \tag{36}$$

This is a relation between two nearby points. Given an origin (t_0, s_0), a point lies on the approximation to the equations centered at (t_0, s_0) if it satisfies a certain derived equation. For example, if the differential equations are

$$\frac{\mathrm{d}q}{\mathrm{d}t} + 3q = 0$$

then a suitable approximation at origin $t = 0$, $q = 1$, is the plane

$$(q - 1)/(t - 0) + 3 = 0$$

No restriction is placed on how this approximation is calculated in general, but it has to be a tangent space to the solution space at the given origin. The VHDL-AMS Draft LRM explicitly says that the language is independent of

the analog solver used in a particular implementation, and different solvers may obtain different local approximations.

The approximation at origin (t_0, s_0) is valid in a small neighbourhood of (t_0, s_0). The local approximations centered at a sequence (t_0, s_0), (t_1, s_1), (t_2, s_2), (t_3, s_3), ... of nearby points are used to take an analytic extension as far as required by reincorporating improved estimates at every step. Given the starting point (t_0, s_0), we generate the local approximation space e_{t_0,s_0} at (t_0, s_0) and solve it to get a function $[e]_{t_0,s_0} :: \mathit{Time} \to \mathit{State}$ that gives approximate solution points $[e]_{t_0,s_0}(t)$ near to (t_0, s_0) as a function of time. In the example above, with origin $t = 0$, $q = 1$, the function $[e]_{(0,1)}$ is

$$q(t) = 1 - 3t$$

and in general this approximation function will be linear of the form

$$[e]_{(t_0,s_0)}(t) = s_0 + (t - t_0)\,\mathbf{K}_e(t_0, s_0) \tag{37}$$

in which the slope vector $\mathbf{K}_e$ varies with the origin of the approximation. In the example above, $\mathbf{K} = -3$ and $[e]_{(t_0,q_0)}(t) = q_0 - 3(t - t_0)$. This is the derivative of state with time when moving along a solution trajectory. The final state reached is the integral along the trajectory of the slope:

$$s(t) = s_0 + \int_{t_0}^{t} \mathbf{K}_e(t, s(t))\mathbf{d}t \tag{38}$$

and, within a short distance $\mathbf{d}t$ of t_0, $s(t)$ is approximated by $[e]_{(t_0,s_0)}$:

$$s(t) \sim [e]_{(t_0,s_0)}(t), \quad t \in [t_0, t_0 + \mathbf{d}t] \tag{39}$$

so that the integral (38) and the approximator (39) can be combined:

$$\begin{aligned} s(t) &= s_0 + \int_{t_0}^{t} \mathbf{K}_e(t, s(t))\mathbf{d}t \\ &\sim \begin{cases} [e]_{(t_0,s_0)}(t), & t \in [t_0, t_0 + \mathbf{d}t) \\ [e]_{(t_0,s_0)}(t_0 + \mathbf{d}t) + \int_{t_0+\mathbf{d}t}^{t} \mathbf{K}_e(t, s(t))\mathbf{d}t, & t \geq t_0 + \mathbf{d}t \end{cases} \end{aligned} \tag{40}$$

allowing us to obtain the following recursive approximation for $s(t)$ by naming $s_0 + \int_{t_0}^{t} \mathbf{K}_e(t, s(t))\mathbf{d}t$ as $ext_e(t_0, s_0)(t)$:

$$s(t) \sim ext_e(t_0, s_0)(t) \tag{41}$$

$$ext_e(t_0, s_0)(t) = \begin{cases} [e]_{(t_0,s_0)}(t), & t \in [t_0, t_0 + \mathbf{d}t) \\ ext_e(t_0 + \mathbf{d}t, [e]_{(t_0,s_0)}(t_0 + \mathbf{d}t))(t), & t \geq t_0 + \mathbf{d}t \end{cases}$$

In our example above, this approximation becomes:

$$ext_e(t_0, q_0)(t) = \begin{cases} q_0 - 3(t - t_0), & t \in [t_0, t_0 + \mathbf{d}t) \\ ext_e(t_0 + \mathbf{d}t, q_0 - 3\mathbf{d}t)(t), & t \geq t_0 + \mathbf{d}t \end{cases}$$

This approximator produces a world line $\lambda t.s(t)$ from an initial state s_0 at time t_0. It is convenient in terms of our semantic model, however, to express the extension in terms of an initial world line w_0 with $w_0\, t_0 = s_0$. Replace $ext_e(t_0, s_0)(t)$ in (41) with $extend(e, t_0)(w_0)(t)$ and generalize s_0 to $w_0\, t_0$, obtaining, for some world line w_1:

$$extend(e, t_0)(w_0)(t) = \begin{cases} [e]_{(t_0, w_0\, t_0)}(t), & t \in [t_0, t_0 + \mathbf{d}t) \\ extend(e, t_0 + \mathbf{d}t)(w_1)(t), & t \geq t_0 + \mathbf{d}t \end{cases} \tag{42}$$

where we require $w_1(t_0 + \mathbf{d}t) = [e]_{(t_0, w_0\, t_0)}(t_0 + \mathbf{d}t)$, to satisfy (41) and therefore may set $w_1 = [e]_{(t_0, w_0\, t_0)}$. By design:

$$extend(e, t_0)(w_0)(t) = ext_e(t_0, w_0 t_0)(t) \sim w(t) \tag{43}$$

where $w(t)$ is the intended future state at time t. Stating also that the world line is unchanged for $t < t_0$, we obtain:

$$\begin{aligned} &extend :: (LocalApprox, Time) \to WorldLine \to WorldLine \\ &extend\ (e_0, t_0)\ w_0 = w' \end{aligned} \tag{44}$$

$$w't = \begin{cases} [e](t_0, w_0 t_0)(t), & t \in [t_0, t_0 + \mathbf{d}t) \\ extend(e, t_0 + \mathbf{d}t)([e]_{(t_0, w_0\, t_0)})(t), & t \geq t_0 + \mathbf{d}t \\ w_0 t, & \text{otherwise} \end{cases}$$

Figure 2 illustrates the method. We can also write the above equations as:

$$\begin{aligned} extend\,(e_0, t_0)\, w_0 &= w' \\ w't &= w_n t \qquad t_n \leq t < t_{n+1} \\ t_n &= t_0 + n\mathbf{d}t \\ w_{n+1} &= [e](t_n, w_n\, t_n) \end{aligned} \tag{45}$$

Now we only have to take into account the previously scheduled values of the signals at each multiple of the VHDL minimum unit time in order to obtain the complete *update* function of (35):

$$\begin{aligned} &update :: (LocalApprox, Time) \to WorldLine \to WorldLine \\ &update\,(e_0, t_0)\, w_0 t = w_n t, \qquad n \leq t < n + 1 \\ &\qquad\qquad w_{n+1} = extend(e_0, \lfloor t \rfloor)(w_n \oplus signals) \end{aligned} \tag{46}$$

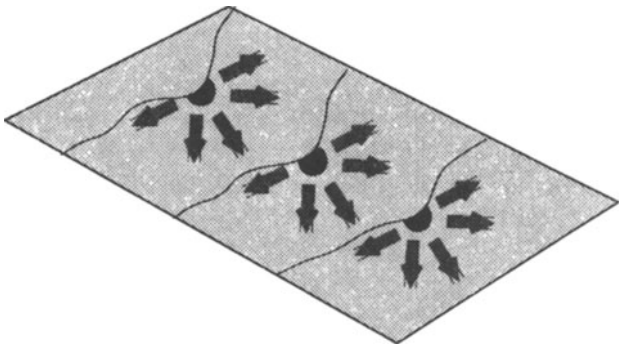

Figure 2 Local approximations, valid within a certain distance of the origin, are analytically extended to incorporate their improved estimates

in which ($\oplus$*signals*) overwrites with the signal drivers. That is to say, the solver is reinitialized at multiples of the VHDL minimum unit, in order to incorporate any changes in signal values. The equations governing quantities might refer to signal values. This mechanism serves also to reinitialize quantities after a discontinuity.

6 SUMMARY

We have given a semantic model for the kernel part of the VHDL-AMS language extension which we believe clarifies the semantic issues in the extension. Syntactic issues have not been the subject of this paper. Our treatment may be summarized as follows:

1. the VHDL-AMS analog solver has been embedded into VHDL-AMS processes as part of the semantics of scheduling assignment;
2. in that setting, it works as a speculative greedy update of a future schedule;
3. the extended semantics remains compositional;
4. the wait statement only changes the current time point, as in our treatment of VHDL in Breuer et al. (1995);
5. the scheduling assignment statement only changes the future schedule, not the time point, as in as in our treatment of VHDL in Breuer et al. (1995). In addition to the action on signals, it performs a 'relaxed-mode' extrapolation of quantity values into the future schedule.

ACKNOWLEDGMENTS

This work has been partially funded and carried out in the context of ESPRIT project 23015 COMITY and NATO project CARE4HW.

REFERENCES

Breuer, P. & Martínez Madrid, N. (1995), A Native Process Algebra for VHDL, *in* 'Euro-VHDL'95', IEEE Comp. Soc. Press, Los Alamitos, Calif.

Breuer, P., Delgado Kloos, C., Marín López, A., Martínez Madrid, N. & Sánchez Fernández, L. (1997), 'A Refinement Calculus for the Synthesis of Verified Hardware Descriptions in VHDL', *ACM Transactions on Programming Languages and Systems* **19**(4), 1–30.

Breuer, P., Sánchez, L. & Delgado Kloos, C. (1995), 'A Simple Denotational Semantics, Proof Theory and a Validation Condition Generator for VHDL', *Formal Methods for System Design* **7**(1 & 2), 27–51.

Delgado Kloos, C. & Breuer, P., eds (1995), *Formal Semantics for VHDL*, Kluwer, Amsterdam.

IEEE (1994), *IEEE Standard VHDL Language Reference Manual. ANSI/IEEE STD 1076-1993*, Institute of Electrical and Electronic Engineers, New York.

Jones, M. P. (1991), Introduction to Gofer, Technical report, Department of Computer Science, Yale University, USA. (part of the Gofer distribution, anonymous ftp from ftp.cs.nott.ac.uk in the directory nott-fp/languages/gofer, rev. Oct. 1994).

Sasaki, H. (1997), 'Semantic Analysis of VHDL-AMS by an EA-Machine and an Attribute Grammar', *URL: http://www.tamaru.kuee.kyoto-u.ac.jp/1076.1/se970307.prn.Z.*

7 BIOGRAPHY

Natividad Martínez Madrid is a research and teaching assistant at the Universidad Carlos III de Madrid. She received her diploma in Telecommunication Engineering from the Universidad Politécnica de Madrid in 1993.

Peter T. Breuer is an associate professor at the Universidad Carlos III de Madrid. He received his PhD in Enginering from the University of Cambridge in 1985.

Carlos Delgado Kloos is a full professor at the Universidad Carlos III de Madrid. He received his PhD in Telecommunication Engineering from the Universidad Politécnica de Madrid and in Computer Science from the Technical University of Munich in 1986.