

The Definition of a VHDL-AMS Subset for Behavioral Synthesis of Analog Systems *

Alex Doboli and Ranga Vemuri
{adoboli,ranga}@eecs.uc.edu
Department of ECECS
University of Cincinnati
Cincinnati, OH, 45221-0030

Abstract

This paper defines a VHDL-AMS subset for behavioral synthesis of analog systems. The subset includes language constructs for describing most of the functional aspects pertaining to analog systems. Besides, these constructs can be implemented with electronic circuits. Functional aspects, which can be expressed with the subset, relate to two interacting parts. The analog part continuously processes analog signals, while the control part generates control signals for configuring the flow of signals in the analog part. However, some language constructs have to be constrained or augmented, so that they become effective for synthesis. To motivate that constructs in the subset can be synthesized, we present, by means of an example, how VHDL-AMS programs are compiled into an intermediate format. The intermediate format can be either directly mapped to components from a component library, or used for further synthesis-related optimization steps. Finally, we discuss a complete experiment for specifying of functionality, compiling the specification, and its mapping to electronic components.

1 Introduction

The design of analog electronic systems is a very difficult task mainly because of complex constraints to be satisfied by the final product, and the high sensitivity of analog characteristics to outer-world perturbations. Nevertheless, the consumer's market offers a growing demand for analog and mixed analog-digital electronic systems. Previous research developed very effective computer-aided design (CAD) tools for automated design of digital systems [6]. However, while analog parts represent only 10% of a system functionality, they require 90% of the development time. Evidently, more recently, research has been focused on developing effective tools for automated synthesis of analog circuits [8] [10] [15] and systems [3].

Automated analog synthesis starts with input specifications for the system functionality and performance constraints, and automatically produces a

high-quality implementation. A typical behavioral-synthesis flow includes four design tasks. First, the system functionality (behavior) is *described* by using a specification language. Next, *architectural synthesis* and *component selection* pick a set of electronic components from a predefined library of component topologies, and explore which of their interconnections result in functionally correct implementations. Finally, the *sizing step* calculates physical dimensions (width, length) for the transistors of the picked components, so that existing design constraints are satisfied. Previous research on analog synthesis assumes a known circuit or system topology, and exclusively concentrates on the issue of using performance attributes for guiding the synthesis task. We believe that the absence of more substantial work on specifying analog functionality is mainly because of the lack of a standard specification language. Recently, Very High Speed Integrated Circuits Description Language for Analog Mixed Systems (VHDL-AMS) [1] emerged as a standard for specification of mixed analog-digital systems.

In this paper, we will define a subset of the VHDL-AMS language, and indicate how this subset is used for automated system synthesis. We will motivate what language constructs are necessary, so that *virtually* all functional aspects of a system can be described. Besides, these constructs must be implementable as electronic circuits. By analyzing a meaningful number of real-life examples, we have identified that analog systems can be modeled as consisting of two interacting parts. The *analog part* has different modes of functioning, and it performs all (signal) processing defined by the functionality. The *control part* generates signals for selecting the current functioning mode of the analog part. Our VHDL-AMS subset can describe any functional aspect pertaining to the analog or control parts. Nevertheless, being oriented towards simulation, some of the language constructs can not be synthesized, unless restrictions are imposed. Moreover, we believe that VHDL-AMS can not describe some specific system characteristics, i.e. low/high input/output impedance, but which are crucial for synthesis. In this paper, we present our language constraints/extensions for synthesis. Finally, by using an example, we show how our VHDL-AMS subset is com-

*This work was sponsored by the USAF, Wright Laboratories, Wright Patterson Air Force Base under contract number F33615-96-C-1911

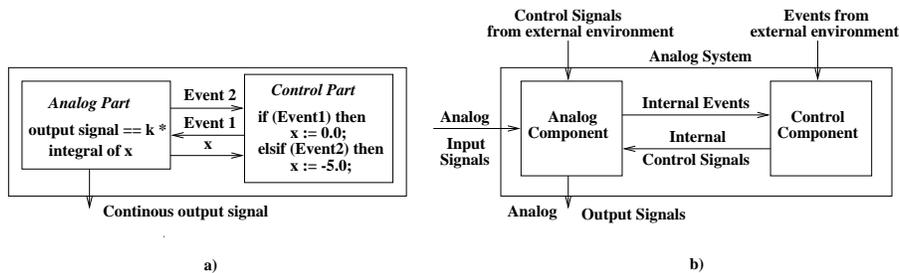


Figure 1: Behavioral model of an analog system

piled, and used for behavioral synthesis.

To the best of our knowledge, no previous results have been reported on defining a specification language for synthesis of analog systems. Nevertheless, defining a VHDL-AMS subset for synthesis is crucial for the development of any VHDL-based analog synthesis tool. We believe that our subset is general enough to express all functional aspects of analog systems. Besides, this subset can be easily extended for synthesis of mixed signal systems, as this is one of our future goals. In our work, we were concerned about preserving the resemblance between behavioral specifications for analog and digital parts, as this would help for repartitioning the functionality between the two components.

The VHDL-AMS subset we discuss in the paper, is used for developing system-level specifications, which are automatically synthesized with the VASE environment. The VHDL-AMS Synthesis Environment (VASE) [20] is a vertically integrated collection of software tools for synthesis of CMOS mixed-signal systems from specifications written in VHDL-AMS. Currently, the main components of VASE are: a *VHDL-AMS compiler*, a tool for *generation of performance constraints* [13], an *Analog Performance Estimator (APE)* [14], and a *library* of components [4]. Our compiler translates a VHDL-AMS program into a structural signal-flow representation, which is mapped to a net-list of components. The performance-constraints generation tool is a Genetic Algorithm based search engine, which computes ranges for the design parameters, so that system-level constraints are satisfied. Finally, the APE accepts design parameters (bias current, etc.) of an analog circuits along with its topology, and determines the performance parameters (area, UGF, slew rate, etc) of the circuit with the anticipated sizes of the circuit elements.

The paper is organized as six sections. We present some related work on system specification, in the following section. Section 3 introduces our behavioral modeling method, and Section 4 presents our VHDL-AMS subset for synthesis. Next, we discuss a behavioral synthesis experiment, that uses a system specification with our subset. Finally, we provide our conclusions, and intentions for future work.

2 Related Work

Existing CAD tools accept one of the following three specification styles: structural specifications,

implicit specifications and application-specific behavioral specifications. Structural specifications [15] describe an analog application by presenting its sub-components and the way they are interconnected. Implicit specifications [10] are used by tools specialized for the design of a certain class of analog circuits. Their application functionality is fixed, and implicitly decided by the class assumed for the tool. Nevertheless, implicit specifications express the performance constraints to be satisfied by the synthesized implementation. Application-specific behavioral specifications [3] explicitly represent the functionality of the system, but they are restricted to a fixed class of analog systems, i.e. analog filters. None of the above styles can claim to be a general, system-level behavioral specification method for analog systems. We believe that the main reason for this situation is due to the absence of a standardized, well-defined specification language for analog systems.

Recently, VHDL-AMS emerged as a standard language for simulation of analog systems [1]. VHDL-AMS related research refers to modeling and simulation [12] of analog systems or validation [17] of the language. To the best of our knowledge, no systematic research has been conducted to define a VHDL-AMS subset for analog synthesis. Recent work [9] stresses the importance of a behavioral notation for analog synthesis, but no systematic solution is offered. In [9] analog systems are described by using a limited subset of VHDL-AMS, but the necessity of a more powerful notation for specifying system functionality is indicated. Also, no rules are provided for transforming VHDL-AMS specifications into a representation suitable for conducting automated synthesis. This paper describes a VHDL-AMS subset for behavioral specification of analog systems, and discusses its relevance for automated synthesis. The effectiveness of such a research is also motivated by the successful application of VHDL for behavioral synthesis of digital systems [19].

3 Behavioral Modeling of Analog Systems

In this section, we discuss the technique, that we use for behavioral modeling of analog systems. We suggest that in the most general case, an analog system consists of two interacting parts: the **analog part** and the **control part**. The analog part has multiple modes of functioning (behavior), which perform

Example	Analog Part	Control Part	Events		Control signals		Model in	
			outside	intern	outside	intern	time	frequency
Function Generator	yes	yes	no	yes	no	yes	yes	no
Iter.Equat. Solver	yes	yes	no	yes	yes	no	yes	no
Temperature Controller	yes	yes	no	yes	no	yes	yes	no
Dual-slope A/D converter	yes	yes	yes	yes	yes	no	yes	no
Missile Solver	yes	no	yes	yes	no	no	yes	no
Telephone Set	yes	yes	yes	no	no	yes	yes	no
Power Meter	yes	yes	yes	yes	no	yes	yes	no
Rimoldi Diversity Receiver	yes	yes	yes	no	no	no	yes	no
Massey Diversity Receiver	yes	yes	yes	yes	yes	yes	yes	no
Example 1	yes	yes	yes	yes	no	yes	yes	yes
Example 2	yes	yes	yes	yes	no	yes	no	yes

Table 1: **The effectiveness of the behavioral model for real-life examples**

continuous-time processing of signals. The control part has an event-driven behavior, and it generates signals for selecting among different modes. We show the effectiveness of our model by presenting the results of 11 case studies, that we conducted. We used this model for selecting the language constructs in our subset.

For illustrating the model, we rely on the example of a ramp-signal generator, also because it was already used by another analog-synthesis related work [9]. The model of the ramp-signal generator consists of two functional parts, which are depicted in Figure 1a. The analog part continuously integrates a constant input signal x . It has two modes of functioning. In the first mode, it continuously integrates starting from the value of 0.0 to the upper limit of 5.0, while in the second mode, it integrates with a negative factor of -5.0 until the lower value of 0.0 is reached. Exceeding the threshold values of 0.0 or 5.0 represent **events**, and are denoted in Figure 1a as *Event1* and *Event2*. The two events are addressed to the control part. Depending on the type of the event, the control part will produce the appropriate **control signals** for switching the analog part to a new mode of functioning.

By generalizing the previous example, we suggest that an analog system consists of two interacting parts, as it is depicted in Figure 1b:

- The **analog part** performs continuous-time information processing according to multiple modes of functioning. This part is described functionally (behaviorally) by using sets of differential and algebraic equations (DAE), transfer functions, or algorithmically, by indicating the signal flow and computation algorithm they perform.
- The **control part** has an event-driven behavior, and produces control signals for the analog part. This part does not perform any signal processing, but instead, it generates control signals for selecting among the modes of analog functioning. The existence of this component is justified for analog systems with multiple modes of behavior.
- As a by-product of the processing in the analog part, **events** can happen, that determine the shift

of the analog part to a new mode of functioning. This shift is performed by the control component, which recognizes these events, and produces the required control signals.

- The control part produces **control signals** for changing the mode of behavior for the analog part.

The main advantage of this model is that it differentiates the essentially distinct parts of a system (with respect to their functionality and performance characteristics), and which will go through different synthesis steps, in our synthesis environment. Besides, the model can be naturally extended for describing also mixed-signal systems.

For emphasizing the effectiveness of the suggested model, we considered 10 other real-life systems, besides the ramp signal-generator. These examples were selected so that they represent essentially different processing aspects. *Temperature Controller* [2] is a control system for preserving temperature in a predefined range. The names of *Telephone Set* [18], *Power Meter* [7], *Iterative Equation Solver* [21], *Differential solver for a missile guider* [11] are self-explanatory for their functionality. *Dual-slope A/D converter* [5] is an analog to digital converter, *Rimoldy Diversity Receiver* [16] and *Massey's Diversity Receiver* [16] are modulation schemes used in telecommunication. We also developed two artificial examples for some missing functional aspects, and which are indicated by the generic names *Example1* and *Example 2*.

Table 1 presents the effectiveness of our analog-system model for representing the suggested examples. All examples include an analog part, and all, but one, have also a control part (thus, all, but one, have analog parts with multiple modes of functioning). Events for the control part and control signals for the analog part are produced either inside or outside the analog system. The functionality of all real-life examples is expressed in time, although behavior in frequency is also very common for analog systems [5]. The additional examples that we will develop address the issue of functionality in frequency.

The behavioral model points to some of the basic language constructs of any synthesis-oriented subset. A subset should include language constructs for describing the functional aspects pertaining to the analog and control parts, and their interactions. In the next section, we discuss our VHDL-AMS subset for synthesis.

4 VHDL-AMS Subset for Analog System Synthesis

This section concentrates on describing VASS (VHDL-AMS Subset for Synthesis), our subset for synthesis. When defining the subset, we targeted two distinct objectives. First, the subset must include all language constructs, which are required for expressing the functional aspects of an analog system. According to our behavioral model, VASS has constructs, that describe the functionality of the analog and control parts, and their interactions. Second, it should be possible to implement the subset constructs with electronic circuits. For each construct, we developed transformation rules, that convert it into a structural signal-flow representation. This representation can be directly mapped to components from a library [4], or used for the next synthesis steps. These transformation rules were embedded into a compiler for the VASS set.

Being oriented towards simulation, VHDL-AMS has to be partially "adjusted", so that it can be effectively used for synthesis. The adaption involves both restricting some of the language constructs, and augmenting the language with missing synthesis-oriented constructs. The necessity of restricting language constructs is exemplified by the use of a *for* instruction inside a *procedural* statement. Its semantics is defined in terms of a discrete counter, but which is difficult to be realized in a signal-flow structure. Instead, we impose that the number of iterations, for each *for* instruction, is statically known, so that its body can be unrolled for that number of times. Second, it is very common, that a system terminal must have a very low output impedance because of its external connections. This system characteristic can be achieved by synthesizing a specific output stage, but which can not be inferred, unless it is accordingly indicated (i.e. through annotations) in the specification.

The overall structure of an VASS program consists of entity declarations, architecture bodies, package declarations, and package bodies. In the rest of this section, we present the main language constructs in our subset, the constraints we impose for their synthesis, and their translation into a structural signal-flow graph. We also discuss our synthesis-related annotations, by which we augment a specification. Translation rules and the mapping to library components are illustrated by an example, in Section 5.

An entity declarations defines the interface of a system with its external environment. In VASS, we accept *signal*¹, *quantity*, and *terminal* ports. *Ter-*

¹To distinguish VHDL-AMS signals from physical signals, we will indicate the first in italics.

terminal ports describe the structural interconnection of a system with its external environment. Still, we impose that, for each terminal port, only one of its *through* (current) or *across* (voltage) quantities is used in the description. This accommodates the rest of the specification, in which only one of the facets (current/voltage) of an analog signal is utilized.

According to our behavioral model, the architectural body of an VASS program indicates the continuous-time behavior of the analog part, the event-driven behavior of the control part, and their interactions. The system functionality is expressed as relationships and computations on *free quantities* and on *signal* objects. Quantities define signals with a continuous-time behavior, and *signals* those with an event-driven behavior. VASS admits only quantities of *nature type* (floating-point or a composite type with elements of nature type), as they naturally represent analog signals. *Signals* are of nature or bit-vector types. Quantities can be composite (array or record type), but array quantities are indexed only by signals of bit-vector types. This accommodates well our model, where all non-analog computations are performed inside the control part.

Continuous-time behavior can be formulated in an *implicit* or *explicit* manner. In the implicit way, the behavior results from solving a set of differential and algebraic equations (DAEs). The explicit method defines the functionality either as a transfer functions or an algorithmic way, in which the flow of signals is indicated. Our subset supports both description styles. VASS accepts unrestricted DAE definitions as *simple simultaneous* and *simultaneous if/case* statements. Simultaneous statements can not be uniquely mapped into a signal-flow structure, as a set of distinct "solvers" correspond to each DAE set. Our synthesis environment considers all topologies that "solve" a DAE set, while searching for the best implementation.

More challenging for the subset definition is the explicit specification of continuous-time behavior by using *procedural* statements (procedurals). VASS accepts restricted definitions of procedurals, so that they can be translated into a realistic signal-flow path:

- Procedurals are described as a sequence of instructions, which refer to quantities, *signals* or variables. We infer the signal-flow path from the way in which quantities are assigned/referred by instructions (their data-dependencies), without considering any information about instruction sequencing. However, we interpret sequencing as a "redundant" information for verifying the correctness of a specification. Thus, in a sequence, a reference to a quantity must always be after an assignment to that quantity. Besides, due to their global nature, quantities are assigned at most once, and by only one procedural, in the specification.
- The simulation semantics of *while* loops can be preserved after synthesis, only if constraints are specified on their input (referred inside the loop) and output signals (assigned inside the loop). We impose that inputs for a loop change slower than

T_{max} , and it is sufficient that its outputs are produced at time steps greater than T_{max} . This is required as the simulation cycle assumes that a loop is always executed in zero time steps, while in reality, a delay of T_{max} time units can occur. Under these restrictions, a *while* loop describes a sampling functionality: input signals are constant while the loop body executes, and the produced structure is fast enough to implement the described behavior.

- *subroutine* calls are a powerful mechanism for sharing of functionality. Nevertheless, for continuous-time procedurals, functionality sharing is infeasible, unless we guarantee that all subroutine calls mutually exclude each other. In VASS, we accept function definitions and calls inside a procedural, under the following observations. If by analyzing the specification, mutual exclusiveness can not be guaranteed, then all calls to a function are expanded with the function body. If mutual exclusiveness is guaranteed, i.e. a function is only called from inside the alternatives of a branch, then we generate a structure, which multiplexes actual parameters to the inputs of the structure of the function body.

VASS includes *process* statements for specifying event-driven behavior. However, the peculiarities of our design problem can be exploited for restricting process definitions, so that they result in more effective structural descriptions. Accepting unrestricted interactions between processes (i.e. synchronization, communication), ultimately means that the VHDL-AMS simulation-cycle has to be explicitly realized in hardware. However, synchronization and inter-process communications are common for discrete-time systems, but they can be hardly accommodated with a continuous-time behavior. Thus, it is realistic to assume a simplified model of processes interactions, so that simpler hardware structures can be synthesized. We assume that *processes react to events, and after resuming, they execute their entire body (calculate control signals), and then suspend*. Process definitions do not include any *wait* statements. Events originate either in the analog part (events on 'above' constructs), or the outer environment (events on port signals).

The effectiveness of the synthesized process structure can be further increased, if specifications contemplate the following aspect with regard to *signal* use. Memory modules are expensive hardware (area, manufacturing), hence, whenever possible, our method avoids their use in the structural representation. General *signals* consist of a signal driver, apart from their current value. In order to reduce the amount of memory cells, we restrict the use of *signals*, so that they can be realized as one memory block. This is equivalent to avoiding to refer a *signal* after assigning a value to it. It is worth being mentioned, that the simplified process model also supports the constraint way of using *signals*.

As opposed to VHDL-AMS, our subset includes a declarative mechanism for describing properties of quantities and *signals*. This mechanism is required

for synthesis as the behavior is strongly heterogeneous with respect to the signal types and characteristics. An integrator circuit [5] performs its functionality with regard to its input voltage, but not its current. Besides, a declarative description style can be complementary to traditional description methods. Typically, transfer functions express the behavior of a filter. Nevertheless, if the transfer function is provided, then also the filter type, and its structure are decided. Instead, we could describe the properties of the signals along the signal path, i.e. frequency ranges, and let the synthesis tool infer the appropriate filter type. Currently, our annotation mechanism describes signal properties such as kind (voltage, current), value range, frequency range, and impedances at the terminal ports. Our ongoing work will explore a more systematic way of declaring the synthesis-related attributes of a design.

5 Application of the VHDL-AMS Subset for a Synthesis Experiment

This section details our experiment with the behavioral specification and synthesis of the real-life application of the *receiver module* of a telephone set [18]. This example is at the level of complexity which can be currently handled by our synthesis method. Besides, it corresponds to an interesting situation, where the control part is implemented by using only analog components. In this section, we present our modeling of the receiver functionality, and its specification with VASS (including some required annotations). Next, we describe the intermediate representation (signal-flow graph and control part) produced by our compiler. We manually mapped the representation onto library components, and then, sized their physical dimensions. We analyzed the behavior of the resulted design by simulating it, and observing its outputs.

The main function of the receiver module is to provide an audible signal to the telephone set earphone. It amplifies, with different gains, incoming signals transmitted from the calling part, and those produced by its own microphone amplifier and transmitter module. Besides, it automatically compensates losses introduced by different telephone line lengths. The output has a signal limiting capability, and is capable of driving a 270Ω load at 285 mV peak amplitude. A simplified VHDL-AMS specification of this example is depicted in Figure 2a. The output voltage *earph* is a weighted sum of the input voltages *line* and *local*. The resulted value is multiplied with a variable value *rvar*, which models the variable compensation resistance. The compensation algorithm is represented by a *process* statement, which, depending on the result of comparing quantity *line* and threshold voltage V_{th} , selects the corresponding compensation value. Quantities are annotated with attributes that indicate their kind (voltage), and output *earph* is augmented with information about its limiting and driving capabilities.

Our compiler translated the specification into the signal-flow graph in Figure 2b. The signal-flow graph details the sequence of continuous-time operations on the input signals by showing block functionality, their

```

ENTITY telephone IS
PORT (
  QUANTITY line: IN real; -- IS voltage
  QUANTITY local: IN real; -- IS voltage
  QUANTITY earph: OUT real) -- IS voltage
  -- limited
  -- drives 270 O
END ENTITY;

ARCHITECTURE behavioral OF telephone IS
  QUANTITY rvar: real;
  SIGNAL c1: bit;
(1) earph == (Aline * line + Alocal * local) * rvar;

(2) IF (c1='1') USE
  rvar == r1c;
  ELSE
  rvar == r1c + r2c;
  END USE;
  PROCESS (line'ABOVE(Vth)) IS
  BEGIN
  IF (line'ABOVE(Vth) = TRUE) THEN
  c1 <= '1';
  ELSE
  c1 <= '0';
  END IF;
  END PROCESS;
END ARCHITECTURE;

```

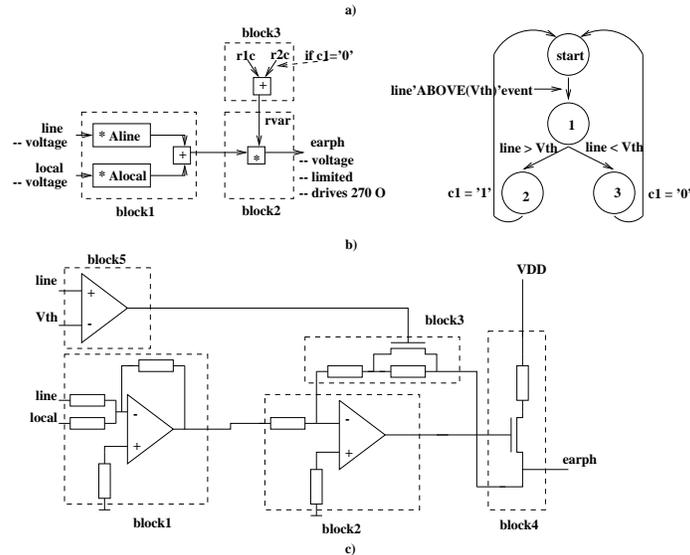


Figure 2: Specification and synthesis of the receiver module of a telephone set

interconnections, and signal and block attributes. The control part expresses how the signal-flow path is modified, due to external events. In our case, a proper compensation gain is selected depending on the comparison of the line voltage and a threshold voltage, V_{th} . Instruction 1 in the VASS program is translated into the sequence of blocks 1 and 2, Instruction 2, which regulates the value of $rvar$, is converted into block 3. The process statement is compiled into the Finite State Machine structure, in Figure 2b. Condition $line'ABOVE(Vth) = TRUE$ controls the arc between state *start* and *state1* to indicate that process resuming is controlled by signal $line'ABOVE(Vth)$. Depending on the result of comparing quantities $line$ and V_{th} , the value of signal $c1$ is set to either '1' or '0', so that a proper value for $rvar$ can be selected.

Next, we mapped the produced signal-flow representation to library components, so that the number of used op amps was minimized. The mapping procedure considers signal attributes and block functionalities, when deciding which components to be used from the library. The mapping assumes that op amps have a very high input impedance, and a very low output impedance. Also, loading effects are considered, when interconnecting two circuits from the library. For this example, the mapping was straightforward, and the resulted circuit structure is depicted in Figure 2c. The corresponding blocks in the signal-flow representation and in the circuit representation are annotated by similar names. Although the control part appears to be quite "sophisticated", its behavior can be realized by

a simple zero-cross detector [5], with a small hysteresis margin, so that repeated switchings are avoided. Its worthwhile mentioning, that *block4* was inferred from the attributes specified in the design, and not from the VHDL-AMS code. As opposed to the other blocks, *block4* does not process signals, but adapts the output stage of the system to the characteristics of the external environment. As we were concerned only about realizing functionality (with neglecting performance attributes), we picked confident values for the physical dimensions of the components. The produced design was described in SPICE, and then simulated. Simulations showed that the design functions correctly.

6 Conclusions and Future Work

This paper describes a complete VHDL-AMS subset for synthesis of analog systems. The identification of such a subset is crucial in the development of any behavioral synthesis tool for analog or mixed-signal systems. The subset includes language constructs for expressing continuous-time, and event-driven behavior, and their interactions. Nevertheless, some of the VHDL-AMS constructs are defined with respect to the simulation goal, and thus, they have to be restricted for being able to convert them into a realistic hardware structure. In this paper, we provided our language restrictions for synthesis. Moreover, we discuss the necessity of augmenting, in a declarative style, language constructs with their attributes, such as signal kind, value ranges, input/output impedances, fre-

quency bandwidth, etc. Such information is required not only for design parameter optimization, but also, as our experiment showed, for inferring a proper hardware structure for a VHDL-AMS specification. Each of the language constructs in the subset can be translated into a signal-flow based representation, which is mappable to library components, and then, sized for optimizing performance constraints. Our translation rules were already embedded in a VHDL-AMS compiler.

Our ongoing work is oriented towards two distinct objectives. We will explore a more systematic way of describing system attributes for synthesis. Besides, we are also concerned about the impact of the specification style on the quality of the produced implementation. Second, we plan to focus on the development of an algorithm for automatically mapping the structural representation into a net-list of components. This algorithm will separately consider the signal-flow part and the Finite State Machine, but it will care about their interdependence with respect to the performance attributes of the global system.

References

- [1] "IEEE Standard VHDL Language Reference Manual (Integrated with VHDL-AMS changes)", IEEE Std.1076.1.
- [2] R. Alur, C. Courcoubetis, T. Henzinger, P. Ho, "Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems", in R.L. Grossman, A. Nerode, A. Ravn, and H. Rischel, editors, *Hybrid Systems (Serie: Lecture Notes in Computer Science, vol. 736)*, Springer-Verlag, pp. 209-229, 1993.
- [3] B. Antao, A. Brodersen, "ARCHGEN: Automated Synthesis of Analog Systems", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol.3, no.2, pp.231-244, June 1995.
- [4] P. Campisi, "A CMOS Analog Cell Library for Analog Synthesis Systems", Master of Science Thesis, University of Cincinnati, 1998.
- [5] S. Franco, "Design with Operational Amplifiers and Analog Integrated Circuits", McGraw Hill, 1988.
- [6] D. Gajski, N. Dutt, S. Lin, "High-Level Synthesis", Kluwer Academic Publishers, 1992.
- [7] S. Garverick, D. McGrath, R. Baertsch, "A Programmable Mixed Signal ASIC for Power Metering", *IEEE Journal of Solid State Circuits*, vol.26, no.12, December 1991.
- [8] G. Gielen, H. Walscharts, W. Sansen, "Analog Circuit Design Optimization Based on Symbolic Simulation and Simulated Annealing", *IEEE Transaction on Solid-State Circuits*, vol.25, no.3, pp.707-713, June 1990.
- [9] C. Grimm, K. Waldschmidt, "Repartitioning and Technology Mapping of Electronic Hybrid Systems", *Proceedings of DATE'98*, IEEE CS Press, pp.52-58, 1998.
- [10] R. Harjani, R. Rutenbar, R. Carley, "OASYS: A Framework for Analog Circuit Synthesis", *IEEE Transactions on Computer-Aided Design*, vol.8, no.12, pp.1247-1266, December 1989.
- [11] D. J. Harris, "Analogue and Digital Computer Methods", Temple Press Books, 1964.
- [12] T. Kazimierski, "A formal description of VHDL-AMS analogue systems", *Proceedings of DATE'98*, pp.916-920, 1998.
- [13] N.R. Dhanwada, A. Nunez, R. Vemuri, "Component Characterization and Constraint Transformation based on Directed Intervals for Analog Synthesis", accepted for the *International Conference on VLSI Design*, 1999.
- [14] A. Nunez, R. Vemuri, "Performance Estimation for CMOS Analog Circuit Synthesis", Technical Report, DDEL, University of Cincinnati, March 1998.
- [15] E. Ochotta, R. Rutenbar, R. Carley, "ASTRX/OBLX: Tools for Rapid Synthesis of High-Performance Analog Circuits", *Proceedings of the 31st ACM/IEEE Design Automation Conference*, pp.24-30, 1994.
- [16] B. Rimoldi, "Five Views of Differential MSK: A Unified Approach", in R. Blahut, D. Costello, U. Maurer, T. Mittelholzer, editors, *Communications and Cryptography. Two sides of one Tapestry*, Kluwer Academic Publishers, 1994.
- [17] H. Sasaki, K. Mizushima, T. Sasaki, "Semantic Validation of VHDL-AMS by an Abstract State Machine", *Proceedings of BMAS*, IEEE CS Press, 1997.
- [18] J. Trontely, L. Trontelj, G. Shenton, "Analog Digital ASIC Design", McGraw-Hill Book Company, 1989.
- [19] J. Roy, N. Kumar, R. Dutta, R. Vemuri, "DSS: A Distributed High-Level Synthesis System", *IEEE Design & Test of Computers*, pp.18-32, June 1992.
- [20] R. Vemuri, N. Dhanwada, A. Nunez, P. Campisi, "VASE: VHDL-AMS Synthesis Environment Tools for Mixed-Signal Systems", in *Proceedings of the Analog & Mixed-Signal Application Conference*, San Jose, pp.1C77-1C84, 1997
- [21] B.R. Wilkins, "Analogue and Iterative Methods", Chapman and Hall Ltd, 1970.