Automatic synthesis of analog systems using a VHDL-AMS to HSPICE translator

G. Doménech-Asensi (1), T.J. Kazmierski (2), R. Ruiz-Merino (1)

(1) Gines.Domenech@upct.es, Ramon.Ruiz@upct.es, Departamento de Electronica, Tecnologia de Computadoras y Proyectos. Universidad Politecnica de Cartagena, Spain
(2) tjk@ecs.soton.ac.uk, Department of Electronics and Computer Science, University of Southampton, UK

Abstract
This paper describes the initial development of a translator from VHDL-AMS code into HSPICE, as a contribution to the group of CAD-EDA tools in the field of analog design. Analog synthesis tools are not as developed as digital ones but the arise of VHDL-AMS, a superset of VHDL, can help to their development. In this work, we present some features of a translator capable to synthesise analog circuits described in a hierarchical and/or behavioural way from standard VHDL-AMS to an electronic HSPICE representation.

1. Introduction
VHDL-AMS language, regulated under standard IEEE 1076.1 [1], has been conceived initially as a tool for specification of digital, analog and mixed systems. From this standard, one can develop simulators to verify the behaviour of a wide variety of physical systems (not only electronic ones) and nowadays it is extending its utility towards synthesis in different levels for all kind of electronic systems. Unlike digital systems, in which system development process from high level specification to physical implementation over a wide set of platforms, is everyday more standarized and automated, analog systems have the disadvantage their synthesis is highly dependent upon the designer experience, relying only on a few tools to help in this task. Whatever effort to develop refinement methodologies and tools in the specification of analog and mixed systems, in particular at highest levels, constitutes a very interesting aim. In order to reach this objective, we need to find the way to introduce this experience in different aspects that compose every design & verification process, to optimise translations under different criteria (cost, speed, power dissipation ...), to make it safe and reliable and to fit each development to the application domain particularities. We need to have a common and wide specification frame, to allow descriptions of a same system in different levels, before facing this objectives. VHDL-AMS language can be an useful tool in this process, supplying us an adequate frame due to its inherent descriptive capability.
On the other hand, among all electronic design support tools, those related to verification hold a relevant place, where simulators are particularly important to ensure functional specifications from highest levels to the physical implementation. Synthesis tools must allow generation of interpretable descriptions for respective simulators, each one in its own description level. On analog design, the more accepted simulation tool at electronic level is SPICE on its different alternatives being HSPICE the more extended nowadays.
In relation to links between analog circuits description & synthesis tools and simulators, currently strong efforts are done to develop simulators able to interpret VHDL-AMS code, carrying out the syntactic correction of input files and simulating behaviour of systems described within this language. Now, first commercial versions of this type of simulators are arising , and their use is possible on the highest levels of abstraction.
If we descend to electronic level, problem of simulation of analog subsystems must be focused in other way. It is more convenient to translate VHDL-AMS descriptions towards and from files written for more powerful and popular simulation tools, like HSPICE, rather than develop new simulation tools for VHDL-AMS code in this level.
We can note some efforts have been done in the development of computer programs able to obtain VHDL-AMS code from HSPICE descriptions [2]. Despite its potential utility, there is no doubt this is the easiest direction in this way between specification languages. Description of a structure of connected components in a library, whose behaviour is defined by algebraic and/or differential equations, is directly translatable to a specification language such as VHDL-AMS.
The inverse direction of this way, however, is not so automatic. The main difficulty in this process is the own descriptive richness of VHDL-AMS. Every function in VHDL-AMS must have its associated electronic structure, expressed in terms of proper connections of a set of components of a predefined library.
We want to study in depth and contribute with solutions in this field of analog synthesis, although we are not going to generalise to whatever function in whatever application. We will rather limit to analog systems which implement biologically inspired neuronal structures. Restricting the set of
functions to such a field of application will allow us
to complete the modules definition procedure in an
area of study of special interest nowadays.
In this contribution, we give an overview of features
reached with our translator from VHDL-AMS to
HSPICE. In section 2 differences between
hierarchical and behavioural descriptions are
presented, and then translation of hierarchical
descriptions is explained in detail. Section 3 focuses
on behavioural descriptions and their translation.
Finally section 4 describes conclusions obtained
from the development of this work. Our contribution
starts from a previous work [3] developed in the
University of Southampton, consisting on a Web
parser for VHDL-AMS.

2 Hierarchical descriptions

From the point of view of the translator design, the
kind of descriptions that VHDL-AMS includes can
be classified into two main classes:
a) Hierarchical descriptions
b) Behavioural descriptions

Hierarchical descriptions are more hardware like
than behavioural ones which do not have easy
equivalents structures in hardware. The kind of
statements used to define hierarchically
interconnected blocks are concurrent statements, in
their various forms.

On the other hand, behavioural descriptions, include
statements that define behaviour of a certain system
regardless of its structure. In this group we can
include the so called sequential statements, already
defined in original VHDL [4] and also simultaneous
statements. VHDL-AMS includes simultaneous
statements as one of the main contributions to the
original standard. This kind of statements allow
description of both algebraic and differential
equations as well as some additional forms that will
be described in the following sections. Equations
describing behaviour of analog systems and circuits
have the form of algebraic or differential equations
and this is the reason this kind of statements are so
interesting for describing such systems.

Simultaneous statements could also be considered as
descriptors of a system structure due to their
condition of simultaneous. However in this paper
they will be considered as behavioural statements
because of its potential ability to describe algebraic
and differential equations as aforementioned.

In this section we are going to show translation of
hierarchical descriptions and then we will apply it to
a common and well known example like a small
neural network, to clarify the explanation.

Translation of hierarchical or structural designs is
based on the use of a VHDL-AMS primitives
library. This library includes entities (primitives) which are directly translated into a HSPICE

Translator recognises a hierarchical description because a component instantiation statement, which defines a subcomponent of the design entity, is found in the VHDL-AMS code inside the architecture body associated to a given entity. This component instantiation is translated as a subcircuit node call (i.e. node X1 in figure 2), whose name is the component name and whose interface list is taken from the port map of the component instantiation. The number given to node ‘X1’ is created by the translator and increased in each new component instantiation found in a same architecture. At the same time, this subcircuit is created in the HSPICE file, using the same component name and taking the interface from the port map (i.e. .SUBCKT in figure 2).

1.- Numbering of nodes is easier working with
subcircuits, and moreover same names used in
VHDL-AMS as variables are used in netlist as
nodes.
2.- HSPICE netlist obtained in this way is clear and
easy to understand.
To create the subcircuit body, translator moves to next level of hierarchy, looking for the entity of the component instantiated. Then it processes its associated architecture as explained before, which makes this translation a recursive processing when successive component instantiation statements are found. These steps are repeated until a primitive is instantiated, in that case we have arrived to the lowest level of hierarchy. If this happens, such primitive is translated directly to a HSPICE subcircuit, as explained before. In fact, translator does not process next level until it does not finishes translation of all sentences in current architecture.

We can see full capability of translator in next example in which we will detail a XOR neural network described in a hierarchical structure of four levels. First level corresponds to the network itself, which is composed by three neurons.

```vhdl-ams
package weights is
type byte_vector is array( INTEGER range<> ) of BIT_VECTOR ( 7 downto 0);
end weights;
use weights all;
entity network is
  port( quantity xor_in: in real_vector(1 to 3));
  port( quantity weight_in: in byte_vector(1 to 9));
nenetwork_1;
cf entity network_1;
architecture structural of network is
  components declaration
  quantities declaration
begin
  neuron1: neuron
    port map( input(1 to 3) => xor_in(1 to 3), output => internal1, weight(1 to 3) => weight_in(1 to 3));
  neuron2: neuron
    port map( input(1 to 3) => xor_in(1 to 3), output => internal2, weight(1 to 3) => weight_in(4 to 6));
  neuron3: neuron
    port map( input(1) => internal1, input(2) => internal2, input(3) => xor_in(3), output => xor_out, weight(1 to 3) => weight_in(7 to 9));
end architecture behaviour;
```

These neurons are placed in the second level of description. Each neuron contains several blocks inside, one of them which performs scalar product of inputs and weights. This block belongs to third level of hierarchy. Finally, this block is composed by primitives, which constitute the forth and lower level of the system.

Figure 3 shows a piece of VHDL-AMS code representing a description of the aforementioned neural network. This network works with normalised values for inputs and outputs (0 to 1) and weights (-1 to +1). The network (figure 4) has two input vectors, an analog one for the inputs themselves and the other for the weights, which have to be stored in a digital memory in the real implementation, and one analog output.

In the following paragraphs, the hierarchy levels as well as translation process are explained with more detail, following this example.

Translator starts processing architecture behaviour of entity neuron, and it finds three component instantiation statements. These instantiations are translated as already explained, creating three nodes and three HSPICE subcircuits.

Then translator continues processing entity neuron and its associated architecture, moving so to level 2 in the structure. This second level corresponds to the neuron model. The neuron model used in this description is the conventional one composed by a scalar product of an input vector and a weight vector module followed by a sigmoid function and a voltage follower as the output stage. General schema of this neuron can be seen in figure 5.

Description of the aforementioned scalar product belongs to third level of hierarchy, shown in figure 6, in which we can see this module composed by three Digital x Analog multipliers and one analog adder. The DA multiplier performs multiplication
between an analog input (corresponding to the neuron input) and a digital value, corresponding to the input weight. We suppose neuron weights are stored in digital format, in a mixed signal design, and in this case an 8 bit sign-magnitude representation has been used.

The forth level of this hierarchical design includes entities used as primitives for HSPICE translation. In this case, we have used primitives for Sigmoid, Opamp, Adder, and DA multiplier, that we have used as basic building blocks.

![Diagram](image)

**Fig. 6.** Scalar multiplication. Level 3 in the structure.

Automatic net naming is performed by the VHDLAMS-HSPICE translator.

Figure 7 shows a sample of the HSPICE file generated by our translator. The naming of the nodes can be compared with the original notation of the VHDL-AMS code. Translator has been designed to use original notation, in an attempt to clarify generated HSPICE code. As it can be seen the structure of the VHDL-AMS code is respected in the HSPICE one, and each entity has its corresponding subcircuit, except at the top level one. Besides, VHDL-AMS vectors in the entity port definitions are split into single ports in the translation to HSPICE subcircuits, to accomplish with its syntactical rules.

![Diagram](image)

**Fig. 7.** HSPICE code generated by the translator.

Some lines are added to the HSPICE code to supply it with stimuli to perform a transient analysis of the neural network, whose results are shown in figure 8. We have used HSPICE models from AMS 0.6µ technology process.

![Graph](image)

**Fig. 8.** Simulation of the neural network

### 3 Simultaneous Statements

As aforementioned, VHDL-AMS descriptions could be basically divided in two main categories, the previously studied hierarchical descriptions and the one we will explain now: behavioural descriptions. **Sequential statements** are similar than in VHDL and they are used to define algorithms for the execution of a subprogram or process [4]. Synthesis of this kind of sentences for digital systems has been already solved by different vendors and we have not worked on them at the moment, although it will be included in this translator later.

On the other hand **simultaneous statements** are peculiarly interesting to describe analog systems as we have previously explained. Standard VHDL-AMS defines several kinds of simultaneous statements. A **simple simultaneous statement** is the basic form of the simultaneous statement and represents the equality of two expressions. Some examples of simple simultaneous statements are:

\[ a + b = c + d + e; \]
\[ i_c = \frac{V_t}{\tau} \left[ \frac{d}{dt} V(t) \right]; \]

where ‘Dot’ represent a time derivative.

**Simultaneous if statement** and **simultaneous case statement** are similar to the if and case statements defined in VHDL, but adapted to simultaneous behaviour. The **simultaneous procedural statement** allows sequential notation, similar to a function body but without using it. Finally, the evaluation of a **simultaneous null statement** has no effect on the description of the system.

Tool described in this paper is able to translate successfully most of these forms of simultaneous statements. In this section we are going to focus our
The way to translate a simultaneous if statement is by means of switches. The condition of the statement can contain one of the following operators: >, <, =, in which '=' has not been implemented due to the difficulty that means its analog interpretation. Whenever a check of this kind:

if a = b use

must be performed, it will be rather used:

if (a - b) < value use

Electronic implementation of this kind of sentences is performed following simple rules. For each if or elsif clause, a comparator is set whose inputs are the operands of the condition clause and whose output is a control signal driving certain CMOS switches. The way in which this switches are open or closed, driven by corresponding control signals makes HSPICE electronic description behaves like VHDL-AMS description. One switch enables or disables a simple simultaneous statement. If there are nested simultaneous if statements, then serial switches will be set and all of them should be closed to enable a certain simple simultaneous statement.

Let us suppose we have to translate this VHDL-AMS code:

```vhdl
if a>b use
    output==a;
else
    output==b;
endif;
```

having a simultaneous if statement and two simple simultaneous statements, each one associated to an opposite condition.

The translator will create a comparator, with a and b as its inputs and a signal named control_0 as output. It will also create a couple of complementary switches driven by this signal which will respectively connect a to output and b to output. Figure 9 shows an electronic representation of this translation:

![Fig. 9. Electronic implementation](image)

When translating to HSPICE description, translator must supply with proper names to different nets and nodes created in this process. It also must use correct names for subcircuits used in this description. The potential of the translator can be seen with another example.

Text in figure 10 describes an entity and its corresponding architecture with several nested simultaneous if statements sentences. This description performs the discretization of an analog input into six possible values, from 0 to 1 volt with a step of 0.2 volts.

```vhdl
entity test is
    port (quantity input: in real_vector;
          quantity output: out real);
end entity test;
architecture behaviour of test is
begin
    if input(1)>0.6 use
        if input(1)>1.0 use
            output==1.0;
        elsif input(1)>0.8 use
            output==0.8;
        else
            output==0.6;
        end use;
    elsif input(1)>0.4 use
        output==0.4;
    else
        if input(1)>0.2 use
            output==0.2;
        else
            output==0.0;
        end use;
    end use;
end architecture behaviour;
```

Fig. 10. VHDL-AMS model of a discretizer.

The first if condition creates a comparator that drives a control signal that also works over different switches. Text line created in the HSPICE file is:

```vhdl
X0 input_1 0.6 Control_0 Comparator
```

Second if condition, nested inside the previous one, is translated as follows:

```vhdl
X1 input_1 1.0 Control_1 Comparator
```

Then, next simultaneous statement is processed:

```vhdl
output==1.0;
```

that is translated as:

```vhdl
X2 output net_3 Control_0 Switch_1
X3 net_3 1.0 Control_1 Switch_1
```

As it can be noted, in order to join output node with 1.0 node it is needed that both switches driven by control_0 and control_1 respectively are closed. This coincides with VHDL-AMS description:
if input(1)>0.6 use
    if input(1)>1.0 use
        output=1.0;

Basically this is the way to proceed in the translation process. Translation of behavioural descriptions is fully compatible with translation of hierarchical ones, and there is no limit in the number of levels of hierarchy or in this case in the number of nested simultaneous if statements.

HSPICE text in figure 12 is the final result obtained for the whole architecture:

X0 input_1 0.6 Control_0 Comparator
X1 input_1 1.0 Control_1 Comparator
X2 output net_3 Control_0 Switch_1
X3 net_3 1.0 Control_1 Switch_1
X4 input_1 0.8 Control_4 Comparator
X5 output net_6 Control_0 Switch_1
X6 net_6 net_7 Control_1 Switch_0
X7 net_7 0.8 Control_4 Switch_1
X8 output net_9 Control_0 Switch_1
X9 net_9 net_10 Control_1 Switch_0
X10 net_10 0.6 Control_4 Switch_0
X11 input_1 0.4 Control_11 Comparator
X12 output net_13 Control_0 Switch_0
X13 net_13 0.4 Control_11 Switch_1
X14 input_1 0.2 Control_14 Comparator
X15 output net_16 Control_0 Switch_0
X16 net_16 net_17 Control_11 Switch_0
X17 net_17 0.2 Control_14 Switch_1
X18 output net_19 Control_0 Switch_0
X19 net_19 net_20 Control_11 Switch_0
X20 net_20 0.0 Control_14 Switch_0

Fig. 12. HSPICE code generated by the translator.

Next figure shows result obtained in HSPICE representation. Both nomenclature for nodes and subcircuits is presented. Switch_0 and Switch_1 subcircuits are CMOS switches which are respectively closed and open for active control signals.

If a proper stimulus in form of analog ramp is added to the HSPICE code, we can do a simulation (figure 14) of the circuit obtained and verify that its behaviour is similar to the system described through VHDL-AMS. Again, we have used an AMS 0.6µ technology to do the simulation.

In the simulation, voltage ramp from 0 to 1.4 v is applied to node input_1, and the output (with steps shape) is read from node output.

Fig. 14. HSPICE simulation

4 Conclusions

We have presented a tool to perform translations from an emerging description language as VHDL-AMS to an electronic simulation description code as HSPICE. Although features currently reached are only a few, the optimal working of this tool in the examples shown and its capability to work with both hierarchical and behavioural descriptions make us to be optimistic on the aim to obtain a too able to synthesise analog systems from higher or lower description levels.

We have shown examples using a 0.6µ AMS process. However, this translator is able to work with several technologies using a different primitives library in each case, which ensures applicability of this tool in a wide variety of designs.

References