Zbornik radova XLVII Konf za ETRAN, Herceg Novi, 8-13 juna 2003, tom I
Proc. XLVII ETRAN Conference, Herceg Novi, June 8-13, 2003, Vol. I

THE VHDL-AMS LANGUAGE IMPLEMENTATION IN ALECSIS SIMULATOR

Bojan Anđelković, Milunka Damnjanović, *Faculty of Electronic Engineering, Niš*

Abstract – *The simulator Alecsis is a mixed-signal and mixed-domain simulator with object-oriented HDL, AleC++ providing very useful modeling properties not found in standard languages. However, having in mind the convenient features that standardization brings, the VHDL-AMS language is incorporated in Alecsis. The details of implementation method and VHDL-AMS models integration are described in this paper.*

1. INTRODUCTION

Many of today's system designs are often mixed-signal and mixed electrical/non-electrical containing analog and digital subsystems, embedded software and sometimes optical, magnetic and/or micromechanical devices. For the development of such systems designers need both, powerful simulators and a uniform hardware description language (HDL) for modeling. With the approval of IEEE VHDL 1076.1-1999. standard (informally known as VHDL-AMS, where AMS stands for Analog and Mixed-Signal) [1], a high-level design language for both mixed digital and analog systems design, as well as for multi physics applications, is available. The major benefit of the standard language is the possibility of creating exchangeable models that can be used across companies and across design packages. Once developed models can be used in different simulation tools. Driven by these benefits, models of components are more and more developed using VHDL-AMS and many companies have been developing implementations of the language standard into their simulation packages. However, the standardization does not meet all demands in modern Application Specific Integrated Circuits (ASIC) and Systems-on-a-Chip (SoC) design. Recently developed standard HDLs, such as VHDL-AMS and Verilog-AMS, cannot be used for the description of software routines necessary in SoC designs. Also, some tool specific, non-standard HDLs have some very convenient modeling features, such as object-orientation, that are not found in standard languages. Therefore, many modern simulators often support a few HDLs (Advance MS from Mentor Graphics, SMASH from Dolphin Integration etc.). This gives designers freedom in modeling complex systems while still they can enjoy reusability of the models developed in standard languages.

One such approach is implemented in simulator Alecsis (Analog and Logic Electronic Circuit SImulation System) developed at the Faculty of Electronic Engineering Niš [2]. Alecsis provides a convenient environment for simulation of digital, analog, mixed-signal and multi domain systems. It has its own, object-oriented HDL, AleC++ (Analog and Logic Electronic C++) developed as a superset of a programming language C++. Therefore, AleC++ can be used for the description of software routines and their object-oriented features can be very useful in modeling. Although it has some advantages, the existence of standard HDLs and their importance in the design process cannot be neglected. Having that in mind, VHDL-AMS language has been incorporated into Alecsis simulation environment. This paper discusses process of the incorporation and AleC++/VHDL-AMS models integration and simulation.

A short overview of Alecsis organization will be given at the beginning. After that the concept of implementation of the new language into Alecsis will be explained in more details, as well as AleC++ and VHDL-AMS models integration. One simulation example will be given in order to illustrate the use of VHDL-AMS in Alecsis.

2. ALECSIS SIMULATION ENVIRONMENT

Organization of the simulator Alecsis is shown in Fig. 1. It contains three separated functional modules: compiler, linker/loader and simulation engine. Compiler translates models described in AleC++ into AleC++ object code (internal binary format of the simulator) that simulator can read. That code can be saved as a library file and invoked later by the simulator or distributed directly to the linker for further processing. The linker/loader resolves all global symbol names and generates all necessary data for the simulation engine. The simulation engine performs the process of the simulation according to the data given in the model and simulation control parameters. It contains virtual processor that interprets AleC++ object code. This approach of emulating a real processor is used in order to make the whole package portable between different platforms.

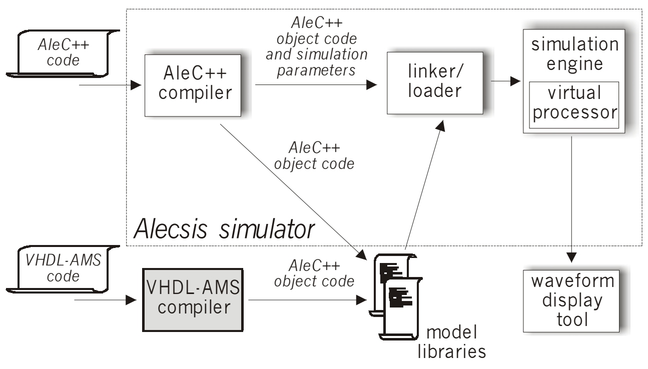

Fig. 1 *Organization of the simulator Alecsis*

* *This work was supported in part by the Ministry of Science, Technology and Development of Serbia, through the Project IT.1.02.0075.A realized in Technology Development area.*

3. VHDL-AMS COMPILER DEVELOPMENT

Having in mind the structure of the simulator Alecsis, shown in Fig. 1., it can be concluded that the simplest way to implement VHDL-AMS language was to keep the existing simulation kernel and to develop a new compiler suited for that language. The new, VHDL-AMS compiler, translates VHDL-AMS source code into AleC++ object code. In that way simulator does not make any difference between object files generated from AleC++ and VHDL-AMS compilers and compiled VHDL-AMS models can be used as any other AleC++ models.

VHDL-AMS compiler was designed using the classic front-end/back-end compiler structure [3] and its organization is shown in Figure 2.

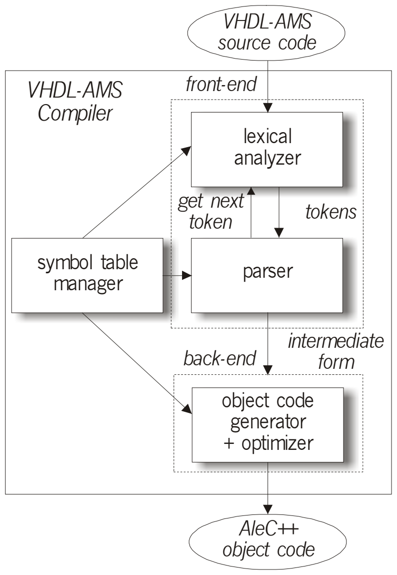

Fig. 2. *VHDL-AMS compiler structure*

Since AleC++ is based on standard HDLs, such as VHDL-AMS, correspondence between the two can be easily established and it is shown in Figure 3. It enables VHDL-AMS compiler to form appropriate data structures that can be converted into AleC++ object code by using the back-end part of the AleC++ compiler with some minor changes. In that way it is necessary just to construct the front-end part of VHDL-AMS compiler.

ALEC++		VHDL-AMS
module	$\rightleftarrows$	architecture
function	$\rightleftarrows$	function
function call	$\rightleftarrows$	function call
instance of module	$\rightleftarrows$	instance of component
node	$\rightleftarrows$	terminal
current	$\rightleftarrows$	through quantity
flow	$\rightleftarrows$	free quantity
simple eqn, across eqn, through eqn	$\rightleftarrows$	simple simultaneous statement
ddt	$\rightleftarrows$	dot
idt	$\rightleftarrows$	integ
———	$\rightleftarrows$	procedure
model card	$\rightleftarrows$	———
global variable	$\rightleftarrows$	———
———	$\rightleftarrows$	nature

Fig. 3. *AleC++/VHDL-AMS language correspondence. Shadowed items do not have appropriate counterparts*

Front-end part consists of two components: lexical analyzer (scanner) and syntax analyzer (parser) [3]. They perform a syntactic and semantic analyzis of the VHDL-AMS source code and generate the intermediate representations. Lexical analyzer reads the stream of characters making up the source code from left-to-right and groups them into the logical entities, called tokens, having a collective meaning. During this phase, the compiler detects identifiers, operators and keywords. The blanks separating the characters of the tokens and comments are ignored. In the input source code text there is a set of strings generating the same token as output. This set of strings is described by a rule, called pattern, associated with the token. Tokens produced by the lexical analyzer are then used by parser. Interaction between scanner and parser is implemented by making the lexical analyzer to be a subroutine to the parser. Lexical analyzer for VHDL-AMS compiler is automatically generated from the appropriate specification by the FLEX program. It is a Windows version of the program LEX, commonly used for this purpose under UNIX operating system. The output is a file in C programming language.

Syntax analyzer groups the tokens identified by the scanner into grammatical phrases that compiler uses for generating the target code. It is constructed from the VHDL-AMS language grammar given in BNF (Backus-Naur Form) notation. A few grammar simplifications had to be made to avoid ambiguities inherent within the VHDL-AMS BNF. The output of the parser is an intermediate representation of the source code containing structures called parser trees. Parser trees are used for representing constructs of the language such as statements, expressions etc. Parser for VHDL-AMS compiler is generated by Berkley's YACC (Yet Another Compiler-Compiler) that is a Windows version of the popular UNIX program YACC. Similarly to the scanner, the output is a C-file.

During these phases semantic analyzes is also performed. The main task in this analyzes is type checking. A data structure called symbol table is used for storing a record for each identifier together with its type, initial value and other important information.

VHDL-AMS compiler also has the possibility of recognizing syntactic errors in the VHDL-AMS data stream and reporting them to the user.

As mentioned before, back-end of the compiler is almost the same as the one used in AleC++ compiler.

VHDL-AMS uses the theory of Differential and Algebraic Equations (DAEs) for the continuous systems describing. For representing unknowns in the DAE's, VHDL-AMS provides a new class of objects, the *quantity* [1]. Quantities can also be used as ports of the model. Special kind of quantities, called *branch quantities*, are used for describing conservative systems, such as electrical circuits. Two kinds of branch quantities are introduced: *across quantities* representing effort like effects (voltage, pressure) and *through quantities* for flow like effects (current, fluid flow rate). They are declared with reference to two terminals. Terminal is another new object in VHDL-AMS, and it can be of different nature representing distinct energy domains (electrical, thermal). Terminals-specifying as ports of the model is used for constructing nodes in hierarchical descriptions when the models are instantiated.

AleC++ uses the element called *link* for representing the unknowns in the system of equations [2]. It can be of one from five different types: *node, current, flow, charge and signal*. Therefore, electrical across quantities in VHDL-AMS correspond to the difference of voltage nodes, nodes to terminals and electrical through quantities to current. *Free quantities* in VHDL-AMS have flows as their counterparts in AleC++. Since Alecsis does not make any difference between nonelectrical *across* and *through* quantities all of them can be treated as flows.

Relationships between quantities in VHDL-AMS are expressed by using simple simultaneous statements [1]. Similar constructs for describing equations exist in AleC++: one for non-conservative and two for conservative systems. Therefore, similar functions in both AleC++ and VHDL-AMS compilers can be used for determining contributions of the equations to the system matrix. Nodes to which the equation contributes are terminals for the appropriate across or through quantities. In order to determine terminals corresponding to the across or through quantity appearing in the equation, the terminal names are stored when quantities are declared together with the name and type of quantity – through or across. The system of equations is then formed by evaluating simple simultaneous statements and by resolving terminal and quantity declarations.

VHDL-AMS attributes for derivative and integration over time exist in AleC++, too, so they can be easily implemented in VHDL-AMS compiler.

VHDL-AMS provides conditional and selected forms of the simultaneous statement that allow choosing appropriate equations depending on some condition(s). AleC++ also provides such constructs, so they can be parsed in VHDL-AMS compiler by using similar functions as in AleC++ compiler.

4. VHDL-AMS MODELS INTEGRATION

As it can be seen in Fig. 3, every VHDL-AMS architecture with appropriate entity relates to one module in AleC++ and they are compiled to the same library objects. Another basic language element is function. Code combining under this level is forbidden. VHDL-AMS/AleC++ models interaction is enabled through the instantiation of the components and calling functions described in the other HDL. Thus, it is possible in VHDL-AMS models to use components and call functions defined in AleC++ and vice versa [4].

In order to integrate VHDL-AMS models with Alecsis it is also very important to establish data type correspondence between the HDLs. That task can be easily accomplished since both languages have the same machine representation of data types (Figure 4). Having in mind data type system in VHDL-AMS, branch quantities get their types from the nature of their plus and minus terminals.

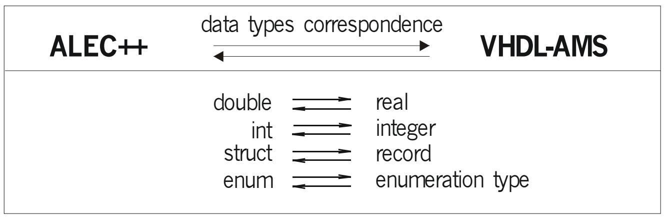

Fig. 4. *AleC++/VHDL-AMS data types correspondence*

5. SIMULATION EXAMPLE

As an example of VHDL-AMS models integration and simulation, a model of Butterworth filter based on the example found in [5] is presented here. RC net and voltage controlled voltage source for the filter model are described as separate VHDL-AMS components (Fig.5 and Fig.6). Both components are described by using simple simultaneous statements with branch quantities.

```
entity rcnet_e is
   generic (resin1, resin2: real;
   res1, res2, res3, res4, res5, res6,
   res7, res8, res9: real;
   resa: real;
   cap1, cap2, cap3, cap4: real);
   port (terminal t1, t2, t3, t4, t5, t6,
t7, t8, t9, t10, gnd: electrical);
end entity rcnet_e;
architecture rcnet of rcnet_e is
terminal t11: electrical;
   quantity v1 across i1 through t4 to
gnd;
   quantity v2 across i2 through t5 to
t4;
   ...
   quantity vc3 across ic3 through t6 to
t11;
   quantity vc4 across ic4 through t7 to
gnd;
   begin
   i1 == 1/res1 * v1;
   i2 == 1/res2 * v2;
   ...
   ic3 == cap3 * vc3'dot;
   ic4 == cap4 * vc4'dot;
end architecture rcnet;
```

Fig. 5. *VHDL-AMS model for RC net of the filter*

```
entity vcvsgen_e is
   generic (gain: real);
   port (terminal t1,t2,t3,t4:
electrical);
end entity vcvsgen_e;
architecture vcvsgen of vcvsgen_e is
   quantity vout across iout through t1
to t2;
   quantity vc across ic through t3 to
t4;
   begin
       vout == gain*vc;
end architecture vcvsgen;
```

Fig. 6. *VHDL-AMS model of voltage controlled voltage source*

These components are instantiated and the filter circuit is created (Fig. 7).

```
entity filter_e is
```

```
   port (terminal f1, f2, f3, f4, f5, f6,
f7, f8, f9, f10, fgnd: electrical);
end entity filter_e;

architecture filter of filter_e is
   component vcvsgen
      ...
   end component;
   component rcnet
      ...
   end component;
   begin
   g0: rcnet ...
   g1: vcvsgen ...
   g2: vcvsgen ...
end architecture filter;
```

Fig. 7. *VHDL-AMS model of Butterworth filter*

AleC++ code for the circuit verification is given in Figure 8, and appropriate simulation results are presented in Figure 9.

```
   #include <alec.h>
   #define Period 15 ms
   module filter (node f1, f2, f3, f4,
f5, f6, f7, f8, f9, f10, fgnd);
   module vcvsgen (node t1, t2, t3, t4)
action (double gain);
   module rcnet (node t1, t3, t4, t5, t7,
t8, t9, gnd)
   action(double resin1, double resin2,
   double res1, double res2, double res3,
   double res4, double res5, double res6,
   double res7, double res8, double res9,
   double resa, double cap1, double cap2,
   double cap3, double cap4);
   library "filter";
   library "rcnet";
   library "vcvsgen";
   root test() {
      filter f1;
      vsin vin;
      vin (n1, 0)
      {amp=10v;freq=159.1549431;}
      f1 (n1, n2, n3, v1, n5, n6, n7, v5,
n9, n10, 0);
```

Fig. 8. *AleC++ code for the filter verification*

6. CONCLUSION

VHDL-AMS is a standard HDL providing the mixed-signal solutions for the behavioural modeling of mixed analog-digital and multi physics systems. Having in mind its importance for portability and reusability of created models, it has been incorporated in simulator Alecsis. On that way, designers can use already developed VHDL-AMS models while exploiting at the same time good features of a non-standard HDL AleC++ to overcome limitations in standard languages.

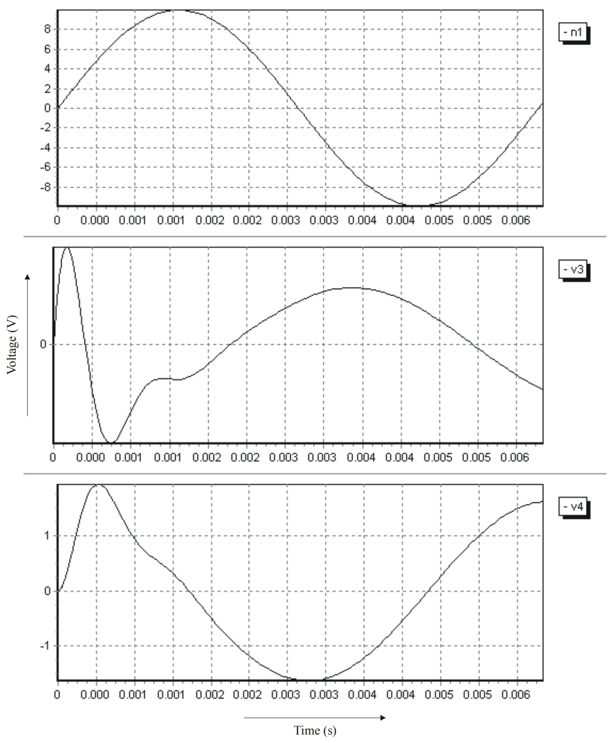

Fig. 9. *Butterworth filter simulation results. Traced signals are voltages in some nodes of the circuit*

REFERENCES

[1] ---, *IEEE Standard VHDL Language Reference Manual (Integrated with VHDL-AMS changes) – IEEE Std 1076-1, draft version*, New York IEEE, 1998.

[2] Ž. Mrčarica et al., *Alecsis 2.3, the simulator for circuits and systems. User's Manual*, Laboratory for Electronic Design Automation, Faculty of Electronic Engineering, University of Niš, Yugoslavia, LEDA – 1/1998.

[3] A. Aho, R.Sethi, J. Ullman, *Compilers – Principles, Techniques and Tools,* Addison Wesley Publishing Company, Reading, Massachusetts, 1986.

[4] V. Litovski, Ž. Dimić, M. Damnjanović and Ž. Mrčarica, "Electronic circuit simulation in a mixed-language environment", *Microelectronics journal,* vol. 29, No. 8, pp. 553-558, 1998.

[5] SEAMS Homepage, http://www.ececs.uc.edu/~mistie/

Sadržaj – Simulator Alecsis je hibridni simulator koji poseduje objektno orijentisani jezik za opis hardvera, AleC++, koji ima veoma korisne osobine u modelovanju kojih nema u standardnim jezicima. Međutim, imajući u vidu pogodnosti koje donosi standardizacija, jezik VHDL-AMS je ugrađen u Alecsis. U ovom radu opisan je metod implementacije i integracija VHDL-AMS modela.

IMPLEMENTACIJA JEZIKA VHDL-AMS U SIMULATORU ALECSIS

Bojan Anđelković, Milunka Damnjanović