ALECSIS / VHDL-AMS MIXED-LANGUAGE SIMULATION

Bojan Andjelkovic, Milunka Damnjanovic, and Vanco Litovski
Department of Electronics, Faculty of Electronic Engineering, Beogradska 14, 18000 Nis
{abojan; mila; vanco}@venus.elfak.ni.ac.yu

Abstract. *Mixed-domain simulator Alecsis has its own HDL, AleC++, suited for description of both digital and especially for complex analogue models, supporting the good features of object-orientation. Due to the importance of standard languages, VHDL-AMS is introduced as parallel input opportunity. VHDL-AMS / AleC++ cosimulation is the point of this paper.*

1. INTRODUCTION

Modern application-specific integrated circuits (ASICs) and system-on-chip (SoC) designs frequently contain both analog and digital subsystems, embedded software, and sometimes are used in conjunction with optical, magnetic and/or micromechanical devices. Therefore, we have not only analogue/digital (mixed-signal) designs, but also mixed-domain integrated systems, where different physical processes are interacting. For development of such systems a powerful simulator and appropriate modeling language are needed with the ability to describe and analyze all these kinds of subsystems in the most efficient way. Recent development in the field of mixed-signal hardware description languages (HDLs) has been determined by the strong need for standardization. IEEE VHDL 1076.1-1999 (informally known as VHDL-AMS where AMS stands for analog and mixed-signal) [1], [4], and Verilog-AMS standards have been issued and they are intended to be universal tools for modeling and documentation of both analog and digital devices and physical models from other domains (mixed-domain simulation). However, in the industrial community it is already clear that standardization is not going to solve all problems, such as, for example, the fact that VHDL-AMS and Verilog-AMS can not be used for hardware/software co-simulation necessary in SoC design. One convenient solution is to enable the simulator to accept and integrate models developed in different languages. Such language-neutral simulation environment let designers use codes (descriptions) already written in standard HDLs which are portable between different EDA tools, while taking good features of the other language(s) to describe and test the components which can not be described in standard HDLs. Finally, the whole complex system can be verified using only one simulation tool. Mixed-language simulators are already available in the market (ADVance MS from Mentor Graphics, SMASH from Dolphin Integration etc.). This paper describes one such approach. It discusses a method of co-simulation with our Alecsis simulator/language environment and VHDL-AMS pre-developed models.

Alecsis (Analogue and Logic Electronic Circuit Simulation System) [2] is a mixed-signal and mixed-domain simulator with its own object-oriented HDL named AleC++ suited for modeling and simulation of both digital and especially complex analogue models. Developed as a superset of C++, AleC++ supports the good features of object-orientation that enables modeling in a natural way. However, having

in mind the importance of standard languages (great number of designers that use it, portable models, growing number of already developed models) and convenience of using mixed-language simulation environment we have developed VHDL-AMS compiler for Alecsis. The compiler enables accepting of VHDL-AMS code and Alecsis simulation with as low designer intervention as possible.

2. ALECSIS / VHDL-AMS COMPILER SIMULATION SYSTEM OVERVIEW

Since AleC++, the hardware description language of simulator Alecsis, is based on the programming language C++, a straightforward solution would be to compile the HDL code into appropriate object code, link it into the simulation kernel and directly execute (compiled simulation). However, the model development is much simpler if the model code needs not to be compiled and linked to the simulation engine every time it is modified. The drawback of such approach is lack of code optimisation, so the model interpretation is slow. Since model code is executed many times during the simulation run, the code optimization is of great importance.

Alecsis combines good features of both approaches. From the user's point of view AleC++ models can be used both in an interpreted and compiled simulation. However, even if the user chooses interpreted mode, the code is firstly compiled into AleC++ object code and optimization is performed. After that the virtual processor interprets the object code. In compiled mode, the result of compilation is stored in the model libraries in AleC++ object code format. These compiled models can be later used in system description or in description of other models, and all global symbols will be resolved by the linker/loader before the simulation starts.

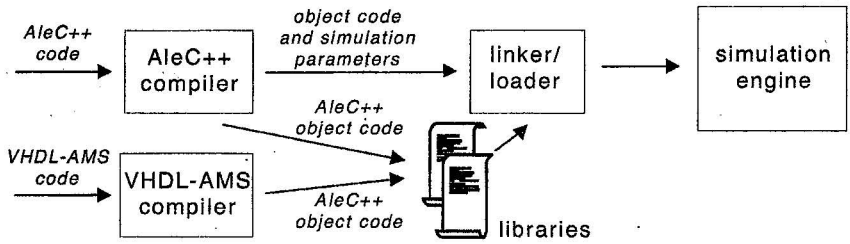

Figure 1. Organization of Alecsis simulator with VHDL-AMS compiler

The simplest way to achieve AleC++/VHDL-AMS co-simulation is to use the existing simulation kernel of Alecsis simulator and to develop a new compiler for VHDL-AMS language [3]. Figure 1 shows the co-simulation concept. At first the compiler front-end analyses VHDL-AMS source code and generates the intermediate data structures. It consists of two building blocks: lexical analyzer (scanner) and syntax analyzer (parser). The lexical analyzer carries out the simplest level of structural analysis and groups the individual characters of the source code text into the logical entities having a collective meaning. The syntax analyzer then groups the simple ele-

ments identified by the scanner into the larger language constructs, such as entities, architectures, statements, loops and functions. Also, in this phase the semantic analysis is performed which determines type of variables, signals, terminals and quantities, the size of arrays and so on. The compiler back-end takes generated intermediate data structures and produces AleC++ object code. Due to similarities between AleC++ and VHDL-AMS, as it will be shown in the following section, the code generation phase developed for AleC++ language is used nearly complete without changes for VHDL-AMS compiler, too.

Compiled VHDL-AMS models can be used in the same manner as any other AleC++ model. The only exception is that simulation control parameters must be obtained from AleC++ file. Therefore, VHDL-AMS models can not be used in the interpreted mode.

3. VHDL-AMS AND ALEC++ MIXED-LANGUAGE MODELING

Two problems must be solved in order to make AleC++/VHDL-AMS mixed-language simulation possible: on what level would be the code combining allowed and are the synchronizing primitives needed. Since the main goal of co-simulation here is to enable reuse of models developed in the other HDL there is no need for code combining inside a single language object (e.g. function, instance of component, etc.).

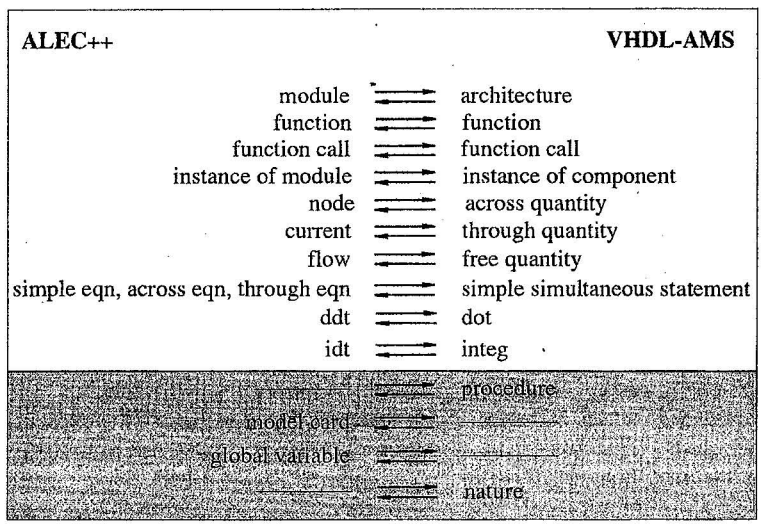

Figure 2. Correspondence between AleC++ and VHDL-AMS elements. Shadowed items do not have appropriate counterparts

Since AleC++ resembles the semantics of standard HDLs such as VHDL-AMS, the correspondence between language elements can be easily established (Figure 2). It enables VHDL-AMS compiler to form appropriate data structures which can be

translated into AleC++ object code and using of almost completely unchanged the back-end of the existing AleC++ compiler. A VHDL-AMS model consists of an entity describing the interface and one or more architectures containing the implementation of the model. When that model is instantiated in a structural description the designer specifies which of several architectures to use for each instance. Every architecture with appropriate entity in VHDL-AMS corresponds to one module in AleC++ and they are compiled into the library object of the same kind. Another basic language construct in both languages is function. Code combining under this level is forbidden. That means that it is not allowed to describe one process or equation in VHDL-AMS and another in AleC++ inside the same module/architecture. The mixed-language simulation is enabled through the instantiation of the subsystems (components in VHDL-AMS and modules in AleC++) and calling functions described in the other HDL. Thus, it is possible in VHDL-AMS descriptions to use components and call functions implemented in AleC++ and vice versa.

VHDL-AMS uses the theory of differential and algebraic equations (DAE's) for describing the continuous systems. A new class of objects, the quantity, is introduced to represent the unknowns in the system of DAE's. For notating DAE's a new class of statements known as simple simultaneous statements is introduced in VHDL-AMS. AleC++ has similar language constructs for writing equations. Three forms of describing equations are used: one for non-conservative and two for conservative systems with across and through quantities. Since the way of writing equations is almost the same in both languages VHDL-AMS compiler can easily determine contributions of the equations from VHDL-AMS model to the system of equations describing the whole design.

Through the instantiation, ports of the component are bound and parameters are passed to it. Also, through the function call actual arguments are passed to the function. Ports and parameters passed to the subsystem or function may belong to different types. Since, both languages, AleC++ and VHDL-AMS, have very complex and powerful data type system it is very important to establish the data type correspondence. Due to the same machine representation it is easy to accomplish that. If the designer creates new data type, the same name and the equivalent description has to be used in both languages. As opposed to earlier language versions VHDL-AMS introduces a new two-level type system: the types already inherent within VHDL'93 and the natures. Natures represent distinct energy domains (electrical, thermal etc.). Its declaration consists of specifying nature's "across" and "through" types. Therefore, type information in the intermediate structures is represented by type symbol and nature symbol instances. All instances of object (except for terminals), literal, subtype and subprogram symbols are linked to their corresponding type symbol and all instances of terminals and subnature symbols are linked to their corresponding nature symbol. Furthermore, nature symbol instances are linked to the symbol instances denoting their across and through types. Since a branch quantity is declared with reference to two terminals, and terminal is declared to be of some nature, the type of branch quantities is derived form the nature of their terminals.

Thanks to the fact that both languages produce the object code of the same format, the simulation engine and virtual processor do not know which compiler has prepared the information. Since the interference boundary of the two languages is limited to library objects, no additional synchronization mechanism is needed.

4. MIXED-LANGUAGE SIMULATION EXAMPLE

A simple example of a summer/limiter is given in order to illustrate AleC++/VHDL-AMS mixed-language description and simulation. Its architecture is described in VHDL-AMS and defines input/output transform as a single algebraic equation using a simple simultaneous statement. The role of the limiter is to clip the summer output if it exceeds a certain range. This model is instantiated and appropriate simulation control parameters are given in AleC++ file. VHDL-AMS model has to be compiled first into the AleC++ object code, and then the whole system is simulated using Alecsis. VHDL-AMS code describing the summer/limiter is shown in Fig. 3 and the corresponding code for the circuit verification is shown in Fig. 4.

```
entity summer_limiter is
  generic (gain1 : real; -- gain of in-
put 1
    gain2 : real; -- gain of input 2
    max_output: real; -- maximum output
    min_output: real; -- minimum output
  );

  port (quantity input1, input2 :
       in real; -- two input ports
    quantity output : out real -- output
port
    );
end entity summer_limiter;

architecture summer of summer_limiter is

quantity sum: real;

begin

  -- state defining equation
  sum: 1*sum - gain1*input1 -
              gain2*input2 == 0.0;

  if (sum > max_limit) use
    output: 1*output == max_limit;
  elsif (sum < min_limit) use
    output: 1*output == min_limit;
  else
    output: 1*output - 1*sum == 0;
  end use;

end architecture summer;
```

Figure 3 VHDL-AMS code describing the circuit

```
module summer (flow input1, input2,
output) action (double gain1, double
gain2, double max_limit, double
min_limit);

library "summer";

root summer_test() {
  flow in1, in2, outlim;

  summer sum;

  sum(in1, in2, outlim) {gain1=1.0;
  gain2=1.0; max_limit=0.0;
  min_limit=-2.0;}

  timing {tstop=Period;
  a_step=Period/1000; }

  plot {flow in1; flow in2; flow out-
lim;}

  /* sinusoidal excitation for input1
and input 2 */

  action {
    double excit_out;
    process per_moment {
      excit_out =
      1.0*sin(twopi*1000*now+3.141);
      eqn in1: {in1} = excit_out;
      eqn in2: {in2} = excit_out;
    }
  }
}
```

Figure 4 Code for the circuit verification

The mixed-language simulation results are shown in Figure 5.

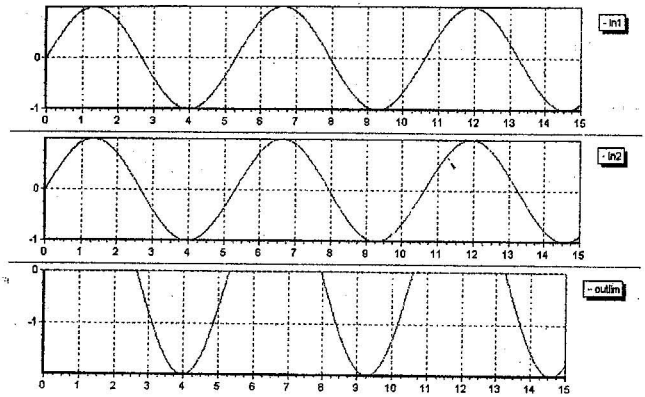

Figure 5. Simulation results of the summer/limiter. Traced signals are input and limited output voltages of the circuit

5. CONCLUSION

AleC++ is a HDL that has all properties of a programming language, too. This gives designers freedom in modeling very complex systems that are not conveniently covered by standard HDLs. However, having in mind the opportunities that standardization brings, a separate VHDL-AMS compiler for Alecsis simulator has been developed. In this sense, mixed-language descriptions and simulations are possible enabling one part of the design to be modeled in AleC++ and exploiting its important advantages, while the other part (pre-developed models in VHDL-AMS) may be given in standardized form. It gives designers the comprehensive environment they need to develop analog/mixed-signal circuits and SoC while they still can exploit the power of using portable models already developed in standard HDLs.

REFERENCES

[1] -----, "IEEE Standard VHDL Language Reference Manual (Integrated with VHDL-AMS changes) - Std 1076-1", draft version, New York, IEEE, 1999.

[2] D.Glozic 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] V. Litovski, Z. Dimic, M. Damnjanovic and Z. Mrcarica, "Electronic circuit simulation in a mixed-language environment", Miroelectronics Journal, Vol. 29, No. 8, 1998., pp. 553-558.

[4] E. Christen and K. Bakalar, "VHDL-AMS - A Hardware Description Language for Analog and Mixed-Signal Applications", IEEE Trans. CAS–II, Vol. 46, No. 10, 1999., pp.1263-1272.