New Aspects in HDL's Performance Evaluation

Bojan Anđelković, Vančo Litovski and Volker Zerbe

Abstract — **New aspects in Hardware Description Language's (HDL) performance evaluation such as object-orientation, system-level modeling, analog and mixed-signal modeling, software description and verification capabilities are analyzed in this paper. Features of mainstream HDLs and verification languages VHDL-AMS, Java, SystemC, AleC++, MLDesigner, OpenVera, the e language, PSL and SystemVerilog are compared in the context of these aspects.**

Keywords — **Hardware description languages, hardware verification languages, Java, SystemC, SystemVerilog, VHDL-AMS.**

I. Introduction

MODERN system-on-a-chip (SoC) designs require powerful modeling languages covering embedded software, system architecture, register-transfer-level (RTL) design and verification. System architects need HDLs that can represent complex hardware/software interactions from abstract levels to detailed descriptions of hardware and software modules. At the same time, hardware designers need a powerful HDL that can express various analog, digital and non-electronic components at different levels of abstraction. Finally, both types of designers require Hardware Verification Languages (HVLs) for hardware and software validation. Therefore, in today's HDL's performance evaluation it is necessary to take care of several new aspects such as: object-orientation, system-level modeling, analog and mixed-signal modeling, software description and verification capabilities. These aspects should be carefully examined during evaluation of usability of particular HDL.

Using examples and comparisons this paper analyzes strengths and weaknesses of some modern design languages in the context of the previously mentioned aspects. As broad as possible list of languages was considered. VHDL-AMS, Java, SystemC, AleC++ and MLDesigner and verification languages OpenVera, the e language and PSL are explored and compared. Also, SystemVerilog as a unified hardware description and verification language is explored. Modeling and verification features of these languages are contrasted and their role in modern mixed-signal SoC design and verification flow is analyzed.

Bojan Anđelković and Vančo Litovski are with the Faculty of Electronic Engineering, University of Niš, Serbia & Montenegro (e-mail: (abojan, vanco)@elfak.ni.ac.yu).

Volker Zerbe is with the Computer Science and Automation Faculty, Technical University of Ilmenau, Germany (e-mail: volker.zerbe@tu-ilmenau.de).

II. Analog, Mixed-Signal and SoC Design Languages

A. VHDL-AMS

VHDL-AMS is an Analog and Mixed-Signal extension to the VHDL (Very High Speed Integrated Circuits Hardware Description Language) [1]. It provides behavioral and structural description of both discrete and continuous hardware systems. Mixed-discipline models from different domains such as electrical, physical, and thermal can be described and simulated in a single language environment.

Modeling of continuous systems is based on the theory of DAEs (Differential and Algebraic Equations). For representing the unknown continuous variables in the system of DAEs, VHDL-AMS introduces a new class of objects, the *quantity*. Additional across and through branch quantities are provided to support conservation semantics of the systems like electrical circuits. Fig. 1(a) illustrates declaration of two branch quantities, voltage *v_in* and current *i_in*. They are declared with reference to two *terminals*, *t1* and *t2*. Terminals can be of different *natures* that represent distinct energy domains (electrical, thermal, etc.).

```
terminal t1, t2: electrical;
quantity v_in across i_in through t1 to t2;
```
(a)
```
i_in == v_in/100;
i_in == C*v_in'dot;
```
(b)

Fig. 1 VHDL-AMS examples. (a) terminal and branch quantity declarations. (b) Simultaneous statements

The system of DAEs can be described using *simultaneous statements* [1]. Fig. 1(b) shows simultaneous statements for resistor and capacitor, respectively.

VHDL-AMS also supports small-signal frequency domain and noise simulation. It allows a designer to define small-signal stimulus in the frequency domain and noise.

Since DAE solvers use numerical algorithms to solve the equation systems, VHDL-AMS enables a designer to specify individual tolerances for quantities, which must be satisfied by the simulator.

B. Java

Java is an object-oriented, general-purpose, concurrent, platform-independent programming language. It can be used for both software and the description of hardware [2].

Unlike C/C++, in Java concurrency can be explicitly implemented by threads as shown in an example of counter model shown in Fig. 2. It is assumed that the class *Counter* and its method *count* are defined previously.

```
  class Count extends Thread {
  Counter cnt;
  public Count(Counter c) {cnt=c; start();}
  public  void  run()  {for(int  i=0;  i<20;i++)
cnt.count();}
```

Fig. 2 Counter model in Java

C. *SystemC*

SystemC is a C++ class library used for system level modeling of concurrent systems [3]. It is a design language especially suitable for description of mixed hardware/software systems.

SystemC defines data types dedicated to hardware modeling such as bit and bit vector types. Hardware or software description is encapsulated inside a C++ class called module. Modules are similar to VHDL-AMS entity/architecture pairs and they represent basic building blocks of a hierarchical system.

A code fragment for counter model in SystemC is shown in Fig. 3. It is assumed that method *do_count* is defined later.

```
  class counter: public sc_module {
  public:
        sc_in<bool> clk, enable, reset, q;
  ...
  counter(sc_module_name cnt): sc_module(cnt) {
SC_METHOD(do_count);
  sensitive << clk.pos() << reset;
  }
```

Fig. 3 Counter model in SystemC

Ports are created from existing SystemC template classes. A SystemC model can be simulated by compiling it with a standard C++ compiler and it uses a discrete-event simulation model much like VHDL's.

D. *AleC++*

AleC++ (Analog and Logic Electronic C++) is a proprietary object-oriented HDL developed for use in the simulator Alecsis [4]. It can be used for modeling of both discrete and continuous hardware systems from various domains at different levels of abstraction. AleC++ provides some additional useful features, both for modeling hardware components and system-level descriptions, not found in other design languages [4].

VHDL-AMS uses process statements for defining synchronization of discrete-event models. For analog models processes are not used. AleC++ uses processes for both discrete and continuous parts of the model. Component definition, called module in AleC++, can contain any number of processes. Processes give the designer full control over the execution of the model.

AleC++ provides structural and behavioral modeling styles as well as the combination of the two. Contributions to the system of equations describing model can be defined by explicitly writing equations and modifying contributions of structurally connected built-in or previously defined models. The second approach is based on describing equivalent circuits of semiconductor components, so every engineer is familiar with this approach and it is easy to learn. One more important issue is that errors in such model can be easily detected comparing to the model consisting of equations.

In AleC++ it is possible to declare and define functions with a variable number of arguments as in C/C++. Besides this, the number of parameters passed to the model and the number of formal signals that are terminals of a module can be variable. It is useful for describing the regular structures and logic blocks with variable number of input/output signals.

A part of AleC++ code for modeling a diode is shown in Fig. 4.

```
  module new_diode::ndio(anode,cathode)
  action() {
  process per_iteration {
  diode_current = calculate_current(anode-
cathode);
```

Fig. 4 AleC++ model

Since AleC++ is a superset of C++, it can be used for modeling of hardware/software systems [4]. This gives designers an opportunity to describe both hardware and software components using one uniform design language.

III. Verification Languages

A. *Mission-level design and verification*

MLDesigner is the next generation system design tool. It is a multi-domain simulator that for the first time permits the seamless integration of the design flow from mission/operational level to implementation tradeoff and test of complex design [5]. The design flow may include for example terrain models, embedded system design and hardware/software partitioning. You can construct a system model through the graphical editor by using the Tcl/Tk command language.

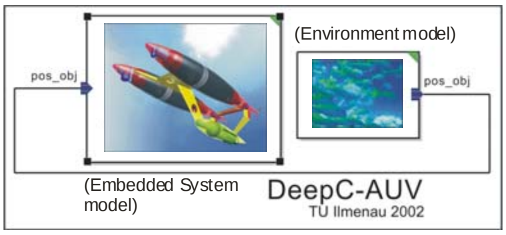

Fig. 5 System model in MLDesigner

You may specify the functionality of your modules by hierarchical block diagrams and principally by embedded C/C++ primitives. Multi-domain simulation modules can be combined and simulated together. Discrete event, finite state machines, continuous time/discrete event, dynamic data flow and high order function domains are supported. Fig. 5 shows the top level model of a multi domain virtual prototype embedded in an underwater environment Without a virtual prototype, design teams must accept significant risk in the development process. Proper verification requires modeling of the operational, or mission environment. Simply put, the mission environment ensures that the design is validated in the context of operational conditions.

B. Verification languages

Among standalone verification languages three most prominent are OpenVera, the e language and PSL [6]. They provide constrained random test case generation, designer-inserted assertions and automated functional coverage checking to quantitatively estimate how much of a design has been tested. Assertions are statements that describe designer intent. They can express complex temporal properties of the design under test which can be checked in simulations or in formal verification. Functional coverage checking is based on monitoring values of variables and state transitions. Each of these values are placed in "bins" and in the end the simulator reports which bins were empty indicating what behavior has yet to be exercised.

OpenVera is a concurrent, object-oriented language developed for writing testbenches to enable efficient verification of complex hardware designs [7]. It provides constructs for generating constrained random values to be applied to the hardware design under test. Also, it contains constructs for monitoring what values particular variables take on during the simulation that enables to estimate how much of a design's behavior has been checked in simulation. In addition, there is the ability to specify temporal assertions which enable to check whether certain system's behaviors ever appear during the simulation. SystemVerilog 3.1a incorporated much of these OpenVera's verification capabilities.

The e language is an object-oriented verification language that provides similar constructs like OpenVera for constrained random values generation, checking functional coverage and specification and checking of assertions [6].

The Property Specification Language, PSL [6], has narrower focus than OpenVera and e, since it is intended just to specify temporal properties (assertions), but it provides more formal semantics. PSL provides four levels of assertion constructs. The lowest, Boolean level enables to specify Boolean expressions on signals in the design under test. The second is temporal layer that allows a designer to specify properties that hold across multiple clock cycles. For example, `always !(a & b)` states that the signals `a` and `b` will never be true simultaneously in any clock cycle, while `always (a -> next[2] b)` means that `b` must be true two cycles after each cycle in which `a` is true. The third, verification level is related to a binding between assertions defined with expressions from the Boolean and temporal layer and modules in the design under test. The fourth, modeling layer enables to include Verilog, SystemVerilog or VHDL code inline in a PSL description and provide additional details about the design under test.

Some features of PSL for specifying assertions are incorporated into SystemVerilog 3.1a.

C. SystemVerilog

SystemVerilog is a set of extensions to the IEEE 1364-2001 Verilog [8]. These extensions provide features for system level modeling along with verification capabilities. The combination of Verilog HDL and such extensions make SystemVerilog to be Hardware Description and Verification Language (HDVL).

It incorporated some of the features already found in VHDL-AMS, such as strong type system, time units, enumerated types, record types (*structs*), multidimensional arrays, separate entity and architecture, iterated and conditional instantiations and configurations [8].

There are also features that have been requested by VHDL engineers that are readily available in SystemVerilog. *'ifdef* conditional compilation enables selection between different design implementations and testbench options. The fork-join statement allows for the spawning of multiple processes and optionally waiting for all the processes to complete before continuing execution of other processes and code. Multiple concatenation and replication enables replication of the contents of a bit or range of multiple bits. SystemVerilog also provides an object-oriented programming model. It supports virtual methods and classes, single inheritance, data and method overloading, static data members and constructors.

Additional SystemVerilog features not found in VHDL include logic-specific processes, implicit port connections, unions and interfaces. Logic-specific processes extend Verilog's *always* blocks for modeling combinational, latched or clocked processes.

Another important feature of SystemVerilog is the Direct Programming Interface (DPI). It allows designers to easily call C/C++ functions from within SystemVerilog and vice versa. It also gives an opportunity to enable using SystemC codes together with SystemVerilog.

SystemVerilog also includes many verification features found in OpenVera, e and PSL languages. Specifically, it contains constrained random values generation and functional coverage checking taken from OpenVera and temporal assertions coming from PSL.

An example of SystemVerilog assertions is shown in Fig. 6 [6]. It describes a temporal property stating that *ack* signal must rise between one and two cycles after each time *req* is true. The property samples *req* and *ack* signals at clock rising edge. After that it is necessary to create a checker to assert that this property holds.

```
property req_ack;
@(posedge clk)
  req ##[1:2] $rose(ack);
endproperty
as_req_ack: assert property(req_ack);
```

Fig. 6 SystemVerilog assertions

IV. HDL's Performance Evaluation

Table 1 shows a feature-by-feature comparison of different HDLs. Column *HVL* relates to hardware verification languages OpenVera, e and PSL. It presents capabilities of all analyzed languages and can be used to evaluate language's performances in various application areas.

Since modern, mixed-signal SoC designs include different electrical and nonelectrical components, as well as embedded software, models written in VHDL-AMS could become too low-level and the appropriate simulators too slow for validating a complete system.

TABLE 1: LANGUAGE FEATURES COMPARISON

Language /Feature	*VHDL-AMS*	*Java*	*SystemC*	*HVL*	*AleC++*	*System Verilog*
Analog & Mixed-Signal	+				+	
System design		+	+		+	+
Software		+	+		+	+
Time and hardware construct	+		+	+	+	+
Object oriented		+	+	+	+	+
Verification	Partial			+		+
RTL modeling	+				+	+
Software		+	+		+	Partial

VHDL-AMS is not appropriate for specifying software and components at higher levels of abstraction. In addition, it is not suitable to directly specify partial differential and algebraic equations necessary for modeling micro-electro-mechanical and microelectrofluidic systems. Because of component-level-oriented modeling features it can not be used to describe system-level behavior, as well.

SystemC fills a gap between traditional HDLs and software programming languages. It provides some advantages over general-purpose programming languages, such as C++ and Java. Such software programming languages are based on sequential programming and therefore they are not suited for the modeling of concurrent processes. Also, system and hardware components require a specification of delays, clocks and time that are not present in C++ and Java. Signals and ports used for communication in hardware models are different from constructs used in software programming. Data types existing in C++ and Java are not suitable for describing hardware implementation. SystemC provides the flexibility of operating with a general-purpose programming language together with data types and constructs necessary for hardware modeling. It can be used for architectural tradeoffs and early application software verification enabling much higher simulation speeds than possible with signal-oriented languages like VHDL-AMS or SystemVerilog. However, just as VHDL-AMS is not an optimal language for system-level modeling and high performance system prototypes, SystemC is not the right language for hardware description at gate level. Moreover, SystemC does not support modeling and simulation of continuous-time, mixed-signal and multidiscipline systems. There is currently an ongoing effort to enhance SystemC with appropriate constructs for analog and mixed-signal modeling similar to that found in VHDL-AMS [9]. Also, synthesis of SystemC designs is much restricted in terms of language coverage and vendor support and the language does not have support for verification.

AleC++ inherited object-orientation from C++. Since modeling is an object-oriented problem by its nature this is a very useful feature. VHDL-AMS does not support object-orientation in its formal definition, although there are some attempts to implement it. Being a superset of C++, AleC++ is suitable for description of software modules as well. It enables full-chip mixed-signal simulation at different levels of abstraction together with embedded software modules. Many of the features it provided at the time of the development now are included in standard design languages.

Standalone verification languages will probably disappear and their features will become parts of modern HDLs, like SystemVerilog.

SystemVerilog provides hardware designers with the ability to use Verilog to describe concise, synthesizable RTL. It also provides creation of efficient testbenches and assertions for simulation-based and formal property verification. With no third-party languages needed for verification, the simulators that support SystemVerilog should be faster than current approaches. Although it has system-level modeling capabilities similar to SystemC, it is considered that SystemC is more focused on systems and software and that the two languages are complementary. SystemVerilog supports modeling of only discrete systems and continuous and mixed-signal systems can not be described.

V. CONCLUSION

Successful mixed-signal integrated circuits and SoC design and implementation depends on the ability to combine the strengths of today's HDLs and HVLs in developing a complex design. Designers should be aware of presented new aspects in HDL's evaluation process while choosing the proper language for the design task at hand.

REFERENCES

[1] *IEEE Standard Definition of Analog and Mixed Signal Extensions to VHDL*, IEEE Standard 1076.1-1999, 1999.

[2] R. Helaihel, K. Olukotun, "Java as a Specification Language for Hardware-Software Systems," in *Proc. of the 1997 IEEE/ACM International Conference on Comupter-Aided Design*, Sun Jose, 1997, pp. 690-697.

[3] *Draft Standard SystemC Language Reference Manual*, Open SystemC Initiative, 2005.

[4] V. Litovski, D. Maksimović, and Ž. Mrčarica, "Mixed-Signal Modeling with AleC++: Specific Features of the HDL," *Simulation Practice and Theory 8*, pp. 433-449, 2001.

[5] V. Zerbe, "Mission Level Design of Complex Autonomous Systems", in *Proc. of XLVII ETRAN Conference*, Herceg Novi (Montenegro), 2003, pp. 55-59.

[6] S. A. Edwards, "Design and Verification Languages," Department of Computer Science Columbia University, New York, NY, Tech. Rep. CUCS-046-04, Nov. 2004.

[7] *OpenVera Language Reference Manual: Testbench, v1.4.2*, Synopsys, 2005.

[8] *SystemVerilog 3.1a Language Reference Manual, Accellera's Extensions to Verilog*, Accellera, 2004.

[9] H. Al-Junaid, and T. Kazmierski, "An Extension to SystemC to Allow Modelling of Analogue and Mixed Signal Systems at Different Abstraction Levels", in *Proc. of SoC Design, Test and Technology Seminar*, United Kingdom, 2004. Available: http://eprints.ecs.soton.ac.uk/9944