A COMPARATIVE ANALYSIS OF DESIGN LANGUAGES FOR HARDWARE-SOFTWARE SYSTEMS

Bojan Anđelković, Vančo Litovski, *Faculty of Electronic Engineering Niš*

Abstract – *In this paper a comparative analysis of design languages VHDL-AMS, Java, SystemC, SystemVerilog and AleC++ is given. Modeling features and capabilities of all languages are explored and compared using appropriate benchmark examples. Also, advantages and restrictions of these languages for the description of various hardware-software systems at different levels of abstraction are pointed out.*

1. INTRODUCTION

The increase in the development of complex, mixed-signal integrated circuits, micro-opto-electro-mechanical systems and systems-on-a-chip (SoC) has leaded in a need for powerful Hardware Description Languages (HDLs) capable for modeling and simulation of these systems in different domains and at various levels of abstraction. Such languages should provide the description of both hardware components and software routines.

Modern hardware-software design languages are based either on software programming languages or hardware description languages. In order to achieve hardware modeling, software languages are usually extended or special libraries are provided. The Java programming language enables modeling of hardware and software modules using object-oriented programming techniques. VHDL-AMS is a hardware description language optimized for continuous, discrete and mixed-signal hardware modeling. Hardware modeling at Register-Transfer Level (RTL) is too low as an abstraction level for design of very complex systems. It is also necessary to describe an entire system, including embedded software, which is beyond the capabilities of existing HDLs. Therefore, special languages for SoC design have been developed. SystemC is a C++ subset for system-level modeling. SystemVerilog is an extension of the Verilog language to support system level modeling and object-orientation. AleC++ is an object-oriented HDL developed for use in simulator Alecsis. It enables structural and behavioral description of analog, logic and mixed-signal systems at any level of abstraction and in different domains. Since it is a superset of C++, AleC++ can also be used for description of software routines.

This paper presents a comparative analysis of design languages used for modern hardware-software systems modeling. Advantages and drawbacks of different languages, specifically VHDL-AMS, Java, SystemC, SystemVerilog and AleC++ are pointed out. Also, modeling capabilities and usage of these languages in different domains are explored. Section 2 presents main features of VHDL-AMS. In Section 3 application of Java as a hardware and software description language is described. In Section 4 and Section 5 the system-level languages SystemC and SystemVerilog are introduced, respectively. Section 6 describes main features of AleC++. Finally, Section 7 gives some conclusions.

2. VHDL-AMS

VHDL-AMS is a hardware description language that provides behavioral and structural description of both discrete and continuous hardware systems from different domains. The language is an Analog and Mixed-Signal extension to the Very High Speed Integrated Circuits Hardware Description Language (VHDL) [1].

VHDL-AMS allows for a clear separation betweeen the interface of a model, called *entity*, and its internal functionality, called *architecture*. The model can have one or more architectures and when it is instantiated in a structural description the designer specifies which of several architectures to use for each instance.

Modeling of continuous systems is based on the theory of Differential and Algebraic Equations (DAEs) [2]. VHDL-AMS also has the ability to describe non-electrical physical phenomena. Mixed-discipline models with different domains such as electrical, physical, and thermal can be described and simulated in a single design environment.

A structure of a VHDL-AMS model and an overview of the language elements and statements are given with the help of a simple circuit with a diode shown in Fig. 1.

```
ENTITY Diode_Cir IS
    PORT (TERMINAL n1, n2: electrical);
END;

ARCHITECTURE behav OF Diode_Cir IS
    QUANTITY v_in     ACROSS i_out   THROUGH n1 TO electrical_groun
    QUANTITY u_D      ACROSS i_D     THROUGH n1 TO n2;
    QUANTITY u_r1000  ACROSS i_r1000 THROUGH n2 TO electrical_groun

BEGIN
    v_in==1000.0 * sin (6.28 * now * 1000.0);

    i_r1000    == u_r1000/1000; -- resistor
    IF v_in > 0.00 USE -- diode
        u_D == 0.0;
    ELSE
        i_D == 0.0;
    END USE;
END;
```

Fig. 1 *VHDL-AMS model*

For representing the unknown continuous variables in the system of DAEs, VHDL-AMS introduces a new class of objects, the *quantity* [2]. Additional across and through branch quantities are provided to support conservation semantics of the systems like electrical circuits. In the example, the branch quantities are diode and resistor voltage and current. They are declared with reference to two *terminals*. Terminals can be of different *natures* that represent distinct energy domains (electrical, thermal, etc.).

The system of DAEs can be described using *simultaneous statements* [2]. VHDL-AMS also provides two special simultaneous statements, called *simultaneous if* and *simultaneous case*, to change the set of equations.

For implementing A/D conversions, VHDL-AMS provides quantity attribute *above*. When the value of a quantity Q crosses a threshold E the Boolean signal

Q'Above(E) evaluates to TRUE if Q>E and FALSE if Q<E [2]. A special construct called the *break* statement is used to represent discontinuities in VHDL-AMS model descriptions and new initial conditions.

VHDL-AMS also supports small-signal frequency domain and noise simulation using special *source* quantities. They allow 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 the designer to specify individual tolerances for quantities, which must be satisfied by the simulator.

However, since systems on chips more and more 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. Also, the language does not have high-level programming constructs, and this makes it difficult to specify software and systems at higher levels of abstraction. In addition, VHDL-AMS is not suitable to directly specify partial differential and algebraic equations necessary for modeling micro-electro-mechanical and microelectrofluidic systems, as shown in [3]. Because of component-level-oriented modeling features it can not be used to describe system-level behavior, as well.

3. 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 [4].

It is a platform-independent language because its run-time environment is an abstract machine called the Java virtual machine containing its own instruction set called bytecodes. Java programs consist of multiple classes, which are compiled into a bytecode representation called class file format. Classes include data fields and methods. If they are declared as static, they are shared by all instances of that class, while non-static fields and methods are duplicated for each new instance. Data types in Java are primitive types such as integers, floats and characters and references to class instances and arrays.

C/C++ does not provide explicit language constructs to express concurrency. In Java concurrency can be explicitly implemented by threads as shown in an example of counter model (Fig. 2).

Threads are created by extending the *Thread* class and overriding its *run* method to describe the thread behavior. Then, when instances of this class are created, they call their *start* methods to start executing the objects' *run* methods. Synchronizing a method or block uses a per-object lock to resolve the situation when two or more threads attempt to access the same object simultaneously. When a thread attempts to access an object owned by another thread it will be blocked until the access to the object is released.

Unlike C/C++, Java doesn't use pointers and its automatic garbage collection frees the programmer from memory management. However, Java does not support class templates and operator overloading provided in C++ that results in a need for numerous procedure calls.

```
class Counter {
int value;
boolean present = false;
public Counter() {
value = 5;
}
public Counter(int arg) {
value = arg;
}
public synchronized void count() {
try { while (present) wait(); }
catch (InterruptedException e) {}
value++; present = true; notifyAll();
}
public synchronized int read() {
try { while (!present) wait(); }
catch (InterruptedException e) {}
present = false; notifyAll();
return value;
}
}
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*

4. SYSTEM C

SystemC is a C++ class library used for system level modeling of concurrent systems [5]. It is a design language especially suitable for description of mixed hardware/software systems.

SystemC 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 those used in software programming. Data types existing in C++ and Java are not suitable for describing hardware implementation.

SystemC defines data types dedicated to hardware modeling such as bit and vector types. Hardware or software description is encapsulated inside a C++ class called module.

An example of counter model in SystemC [5] is shown in Fig. 3.

Modules are similar to VHDL-AMS entity/architecture pairs and they represent basic building blocks of a hierarchical system.

All modules in SystemC are derived from the existing base class *sc*_module. They consist of concurrent processes describing their behavior. Modules communicate with each other through channels and ports. Ports are created from existing SystemC template classes. Channels are generalized form of signals in VHDL-AMS. Simulation instructions such as channels to be traced, the simulator's resolution, top level instance, simulation time and other are located inside a function called *sc_main* similar to *main* function in Java. A SystemC model can be simulated by compiling it with a standard C++ compiler and it uses a discrete-event simulation model.

```
class counter: public sc_module {
  int value;
public:
  sc_in<bool> clk;
  sc_in<bool> count;
  sc_in<bool> reset;
  sc_out<int> q;

  SC_HAS_PROCESS(counter);

  counter(sc_module_name nm): sc_module(nm), value(0)
      SC_METHOD(do_count);
      sensitive<< clk.pos() << reset ;
  }
protected:
  void do_count() {
    if (reset ) { value = 0; }
    else if (count) {
      value++;
      q.write(value);
    }
  }
};
```

Fig. 3 *Counter model in SystemC*

SystemC fills a gap between traditional HDLs and software programming languages such as C++ and Java. It provides the flexibility of operating with a general-purpose programming language but it is primarily intended for system-level descriptions. However, VHDL-AMS is more appropriate for low-level physical descriptions than SystemC. 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 in VHDL-AMS [6].

5. SYSTEM VERILOG

SystemVerilog is a set of extensions to the IEEE 1364-2001 Verilog to enable a higher level of abstraction for modeling and verification with the Verilog HDL. It is intended to become an IEEE standard in 2005.

It incorporated some of the features already found in VHDL, such as strong data typing, time units, enumerated types, records, multidimensional arrays, separate entity and architecture, iterated and conditional instantiations and configurations [7].

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. An example of SystemVerilog model is shown in Fig. 4. The example demonstrates the use of interfaces that enable efficient system-level descriptions.

However, SystemVerilog supports modeling of only discrete systems. Continuous and mixed-signal systems can not be described. Also, it is not an optimal language for system-level modeling and building high performance system prototypes, as SystemC.

```
interface chip_bus (input wire clk);
wire request, ready;
wire [31:0] address;
wire [31:0] data;
modport cpu(input clk, output request...)
modport ram(input clk, output request...)
end interface

module CPU (chip_bus .cpu io);
...
endmodule
module RAM (chip_bus .ram pins);
...
endmodule
module TOP;
wire clk;
chip_bus a(clk);
CPU CPU(a.cpu);
RAM RAM(a.ram);
end module
```

Fig. 4 *SystemVerilog model*

6. ALEC++

AleC++ (Analog and Logic Electronic C++) is an object-oriented design language developed for use in the simulator Alecsis [8]. 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 [8].

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. Writing equations in AleC++ is very similar to that in VHDL-AMS. There are three *eqn* statements (simple, through and across) equivalent to simultaneous statements in VHDL-AMS that enable description of equations containing quantities. The second approach of modifying contributions of connected submodels is not found in VHDL-AMS, but it is very user-friendly. It is based on describing equivalent circuits of semiconductor components using built-in or previously defined models. Every engineer is familiar with this approach, so 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.

AleC++ provides using of different built-in elements similar to SPICE components. VHDL-AMS does not provide such compatibility with the SPICE netlist format, although it has all necessary language constructs to build a library of SPICE elements models.

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. A simple model of a diode using object-oriented features in AleC++ is shown in Fig. 5.

```
class new_diode {
double is;
double eta; //model parameters
public:
new_diode(); //constructor
~new_diode(); //destructor
>new_diode(); // PREPROCESSOR
double calculate_current(double);
friend module ndio;
}
module new_diode::ndio (anode, cathode)
...
action() {
process per_iteration {
double diode_current;
diode_current =
calculate_current(anode-cathode);
...
}
}
}
```

Fig. 5 *AleC++ model*

AleC++ also has some useful features for the description of digital systems, which do not have their counterparts in VHDL and VHDL-AMS. 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.

AleC++ supports the declaration of user-defined signal attributes of any legal type. They can be used for setting signal values in the preparation phase of the simulation and for modeling parasitic capacitance in simulation of digital circuits. VHDL-AMS does not provide the use of user-defined signal attributes.

Since AleC++ is a superset of C++, it can be used for modeling of hardware/software systems [8]. This gives designers an opportunity to describe both hardware and software components using one uniform design language.

7. CONCLUSION

This paper presented a comparative survey of nowadays design languages for hardware-software systems. Since modern mixed-signal SoC contain both analog and logic components as well as embedded software there is no a uniform language capable for the description of such system. Also, in order to simulate such complex system that language should be developed having in mind simulation algorithms. VHDL-AMS is a hardware description language for continuous, discrete and mixed-signal systems. Java is a software programming language that provides some useful features, such as object-orientation, that can be used for hardware modeling, as well. SystemC and SystemVerilog offer object-oriented constructs together with hardware specific features for system level descriptions. AleC++ is an object-oriented HDL that can be used for description of various hardware and software components and systems. Many of the features it provided at the time of the development now are included in standard design languages.

REFERENCES

[1] IEEE Computer Society, *IEEE Standard VHDL-AMS Language Reference Manual*, IEEE Computer Society, 1999.

[2] P. Ashenden, G. Peterson, D. Teegarden, *The System Designer's Guide to VHDL-AMS*, San Francisco: Morgan Kaufmann Publishers, 2003.

[3] T. Zhang, K. Chakrabarty, R. Fair, "Behavioral Modeling and Performance Evaluation of Microelectrofluidics-Based PCR Systems Using SystemC", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 6, pp. 843-858, June 2004.

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

[5] *Introduction to SystemC Tutorial*, Esperan 2005., http://www.esperan.com

[6] H. Al-Junaid and T. Kazmierski, "An Extension to SystemC to Allow Modelling of Analogue and Mixed Signal Systems at Different Abstraction Levels", http://eprints.ecs.soton.ac.uk/9944/

[7] C. Cummings, "SystemVerilog – Is This The Merging of Verilog & VHDL?", www.sunburst-design.com/papers/CummingsSNUG2003Boston_SystemVerilog_VHDL.pdf

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

Sadržaj – U ovom radu predstavljena je uporedna analiza jezika za projektovanje VHDL-AMS, Java, SystemC, SystemVerilog i AleC++. Istražene su i upoređene mogućnosti i osobine ovih jezika korišćenjem odgovarajućih testnih primera. Takođe, ukazano je na prednosti i ograničenja ovih jezika za opisivanje različitih hardver-softver sistema na različitim nivoima apstrakcije.

UPOREDNA ANALIZA JEZIKA ZA PROJEKTOVANJE HARDVER-SOFTVER SISTEMA

Bojan Anđelković, Vančo Litovski