Design, Automation and Test in Europe, DATE'03, Munich, March 3-7, 2003, pp. 388-393

SystemC-AMS Requirements, Design Objectives and Rationale

Alain Vachoux
Swiss Fed. Inst. of Tech.
Microelectronic Systems Lab.
Lausanne, Switzerland
alain.vachoux@epfl.ch

Christoph Grimm
University Frankfurt
Technische Informatik
Frankfurt, Germany
grimm@ti.informatik.uni-frankfurt.de

Karsten Einwich
Frauenhofer IIS/EAS
Design Automation Dept.
Dresden, Germany
karsten.einwich@eas.iis.fhg.de

Abstract

SystemC is emerging as a de-facto standard for system design but it still lacks support for continuous-time models of computation and multi-domain systems. This becomes an issue as the systems under consideration are often heterogeneous. An effort has recently started to allow the SystemC framework to address this need by the definition of the so-called SystemC-AMS (Analog and Mixed-Signal) extensions. This paper defines the context of the extensions with respect to requirements and inferred design objectives. It also gives a rationale for each design objective. Finally, it presents preliminary seed works that will influence the development of the SystemC-AMS extensions and provides some information on the next steps.

1. Introduction

System-on-chip design is a complex task as the targeted systems are more and more heterogeneous. Heterogeneity occurs in the underlying models of computation (MoCs) that are used to describe hardware and software components of the system (e.g. discrete-event, dataflow, FSMs, sequential, continuous-time). Heterogeneity also occurs in the nature of the components of the system (e.g. different disciplines such as electrical, mechanical, fluidic). One way to cope with heterogeneity is to work in a consistent design framework. SystemC is emerging as a de-facto standard for system design but it still lacks support for continuous-time MoC and multi-nature systems [10]. This document presents the foundations on which the mixed-signal extensions to SystemC, named *SystemC-AMS*, will be developed.

This paper is organized as follows. Section 2 discusses the requirements that motivate the extension of the SystemC environment to support analog and mixed-signal systems. Section 3 gives the design objectives that are derived from the requirements. A rationale is given for each of them. Section 4 presents a number of application examples that already started to explore the SystemC capabilities to model and simulate analog and mixed-signal systems. Section 5 draws some conclusions and outlines the next steps.

2. Motivations and Requirements

The systems we are considering in this paper are heterogeneous in nature in that they may include electronic and non-electronic parts. The electronic parts may include digital or analog hardware components or software components. As an example, a typical mobile communication system today includes an RF front-end, analog and mixed-signal components for signal amplification, clock/data recovery, and A/D-D/A conversions, digital components (possibly including processors and software) for data processing. The non-electronic parts may include sensors or actuators. As an example, an antilock brake system includes speed sensors and hydraulic components, in addition to electronic parts. Heterogeneous systems therefore include a great deal of components whose behavior is continuous in time.

The design of such heterogeneous systems encompasses many different engineering disciplines and raises the issue of delivering correct and efficient products in ever shortening time frames. Design techniques that can address the issue include the capability to develop a description (a model of) of the whole system, its parts, and its environment such that key characteristics can be extracted and verified. Extraction and verification are usually performed by static analysis, dynamic simulation or formal proof. Due to increasing system complexity, it is required to support modeling and simulation at high levels of abstraction. On the other hand, accurate verification also usually requires co-simulation with circuit-level models.

Three application domains are considered for the requirements, namely signal processing dominated applications (telecommunications and multimedia), RF/wireless communications, power electronics and automotive.

Signal processing dominated applications are essentially executing operations such as (de)coding, compress-

ing, or filtering data streams with fixed sampling rates. Data processing makes extensive use of arithmetic functions. A static scheduling of operations may be usually derived from the data dependencies to achieve regular and compact system architectures. As modern signal processing systems more and more include both programmable and dedicated components, there is a need to use design technologies that are capable of mapping applications to heterogeneous architectures [13].

RF/wireless applications are essentially realized using an RF front-end part and a baseband part. The design of a RF transceiver at system level, i.e. taking into account both the analog and the digital components and their interactions, is usually done using dataflow models to improve simulation efficiency while still achieving an acceptable level of accuracy [18].

Both signal processing and RF/wireless applications require to model and to simulate both the time-domain and the frequency-domain behavior of key components (amplifiers, mixers, oscillators, etc.). In addition, many frequency-based simulation methods have been developed to overcome limitations of time-domain methods when designing RF circuits [12].

Power electronic and automotive applications share the distinguished requirement to design multi-domain, or multi-discipline, systems, i.e. systems including non electronic parts (mechanical, fluidic, thermal, etc.) [11]. Such systems usually lead to *stiff* nonlinear models that exhibit time constants whose values differ by several orders of magnitude. This property imposes strong numerical constraints to simulation algorithms.

The design of automotive systems more and more requires to develop virtual prototypes including software-in-the-loop and hardware-in-the-loop components [5]. The latter kind of prototype also implies real-time modeling and simulation capabilities, meaning that models must execute in time steps that are bounded by some maximum execution time or response time. It should be noted that real-time capabilities may also be required in signal processing applications.

3. Design Objectives

Design objectives for the SystemC-AMS extensions are inferred from the requirements discussed in Section 2. It is apparent from that discussion that the domain covered by the AMS extensions is pretty large. It will be therefore necessary to proceed by levels or phases when developing the extensions. This issue will be addressed in Section 5.

Design objectives define the context in which the extensions will be designed as well as the goals and the constraints it will have to meet. A rationale is given for a design objective where some more information not explicitly linked to the requirements discussed in Section 2 is required. The given rationales are not intended to bind the objectives to particular implementations. If it seems to be the case, the implementation aspects should be considered as illustrative only. The formulation of the design objectives is inspired from similar work done for the IEEE 1076.1 (VHDL-AMS) hardware description language [17].

SystemC-AMS must be suitable for the description and the simulation of heterogeneous systems.

SystemC-AMS is primarily intended to support the development of *executable specifications*. Support for synthesis, i.e. the process of deriving an implementation from an abstract description, is out of the scope as automated synthesis of analog, mixed-signal, and mixed-technology systems are not yet mature enough.

SystemC-AMS is also primarily targeted towards *system design*. This means that it has to be effective at managing complexity, both in terms of descriptive capabilities and simulation performances. Traditional ways to address this issue is to support abstraction and hierarchy.

The development of system-level executable specifications of continuous-time parts include the modeling of signal processing functions, abstract behaviors (equations), hierarchical structures, and the environment in which the modeled system is intended to work.

SystemC-AMS must be an extension of the SystemC language.

SystemC provides a consistent definition of how both structure and behavior of discrete time systems can be described and simulated. The SystemC simulation semantics is defined by a scheduler and an execution model that support both hardware-oriented and software-oriented modeling [10].

The so-called SystemC *core language* provides a general-purpose framework that supports a variety of models of computation (MoCs), abstraction levels, and design methodologies used in system design. Roughly speaking, a model of computation is a set of (semantic) rules that define the interactions between components of the model. The kind of model components and rules depend on the level of abstraction considered. For example, the *discrete event* (DE) MoC views a system as a set of concurrent processes interacting through signals. Processes are activated when signals whose values are read in the processes experience a value change, a.k.a. events. The rules define how signals get and hold their values and how processes are activated. DE models are typically suitable for RTL hardware modeling.

As another example of MoC, the *dataflow* (DF) MoC views a system as a directed graph where the vertices represent computations and the edges represent totally ordered sequences (or streams) of tokens. In the particular case of *static* or *synchronous dataflow* (SDF), the scheduling of the operations is static and one cycle of the scheduling consists in traversing the graph until all required nodes have been visited and their corresponding computations executed. DF models are typically suitable for signal processing applications.

One distinguished aspect of a MoC is how time is abstracted. Time can be handled as clock ticks, as an integer multiple of a base time (a.k.a. the minimum resolvable time), or as a real value. Model components may also interact in a timeless way through causality rules in so-called untimed functional models.

What is currently missing in the SystemC design framework is the capability to model and simulate continuous-time systems. Analog and mixed-signal extensions for SystemC are currently scheduled for the release 4.0 whose delivering date is not yet defined (the current release of SystemC is 2.0).

SystemC-AMS must support continuous-time models of computation.

Continuous-time (CT) MoCs are based on the theory of differential and algebraic equations (DAEs) that have the following form:

$$F(\dot{x}, x, y, t) = 0 \quad (1)$$

where F is a vector of expressions, x is a vector of differential variables (unknowns), y is a vector of algebraic variables (unknowns), $\dot{x}$ is a vector of derivatives of the x unknowns with respect to time, t is the time (independent variable).

DAEs have been studied extensively by numerical mathematicians for almost 30 years [3]. Most DAE systems have no analytical solutions, so only an approximated solution can be found using numerical techniques. The order of the highest time-domain derivative in a DAE system relates to the so-called index of the system. An ordinary differential equation (ODE) system is a DAE system of index 0. Several numerical solution methods exist for low index DAEs and methods have been developed to reduce the index by augmenting the number of equations. This is mainly required to solve DAEs generated from mechanical system models.

Several tools for continuous system simulation have been developed using languages derived from the Continuous System Simulation Language (CSSL) specification [1]. Most of them support the description of the behavior of a dynamic system as first-order ODEs of the form:

$$\dot{x} = F(x, u, t) \quad (2)$$

where u denotes the input vector of the system. The $\dot{x}$ are discretized using an explicit numerical integration formula such as the Forward Euler or the Runge-Kutta formulas and the equations are sorted to get a sequence of assignments that will be used repeatedly to compute the values of the unknowns over time for any set of input values. In case of algebraic loops exist in the system of equations, meaning that there is a cyclic dependency between unknowns such that it is impossible to define a sequence of assignments, iterative numerical methods such as the Newton-Raphson method has to be used [4].

For a lot of applications in system design, modeling the continuous-time behavior as linear ODEs is sufficient. Typical formulations that produce linear ODEs are transfer functions, state-space equations, or equation formulation of linear electrical networks. In addition, the resulting system of equations can be solved without iterations [6]. This fact will allow to have a first version of SystemC-AMS that already provides useful functionalities while avoiding the need to implement a full-featured DAE solver. It is planned that subsequent releases of SystemC-AMS will support nonlinear DAEs and equations in implicit form.

One important aspect of equations (1) and (2) is that initial conditions must be provided for some or all unknowns to correctly model the dynamic behavior of a continuous-time system. A related issue is the handling of discontinuities that has two aspects, namely the detection of the discontinuities and the specification of new initial conditions to apply just after the discontinuities occur. It is planned that the first release of SystemC-AMS will provide a limited support for initial conditions and won't support discontinuities.

Continuous-time MoCs actually include several kinds of analyses. Some of them are called *static* as they do not require any input stimulus, others are called *dynamic* as they compute the response of the system when input stimulus are applied. Static analyses include the computation of the DC operating point, or quiescent state, transfer functions of the system, and small-signal linear frequency-domain analysis (including noise analysis). Dynamic analyses include the time-domain (transient) and large-signal nonlinear frequency-domain analyses. SystemC-AMS will naturally support time-domain analysis, first as it is one of the most used kind of analysis for continuous-time systems, and second as it may synchronize well with discrete-time MoCs (this is discussed below). SystemC-AMS will also have to support at least small-signal linear frequency-domain analysis, as the frequency-domain characteristics of a system is also important, particularly for signal processing applications. This should not require additional language element as the frequency-domain model can be derived from the time-domain description (1) or (2).

SystemC-AMS must support the description and the simulation of continuous-time systems as signal-flow and conservative-law models.

There are two ways to model a continuous-time system, namely the conservative-law model and the signal-flow model. Conservative-law models define the behavior as close as possible to the physical behavior of the modeled system. Physical quantities are abstracted as so-called *across* and *through quantities* that respectively represent an effort and a flow. The behavior is then described as relations between across and through quantities. For example, for electrical circuits, across quantities are voltages, through quantities are currents, and conservative laws are Kirchhoff's laws. The formulation is quite generic as it can be applied to the model of any kind of physical systems, e.g. mechanical, thermal, optical, etc. systems.

Signal-flow models define the behavior of continuous-time systems as mathematical relations between quantities that represent real-value functions of an independent variable, usually the time. The underlying principle of signal-flow modeling is a directed graph. Each edge represents a quantity and each vertex represents a relation (usually an assignment). Signal-flow models have long been used in many areas of engineering, from the theory of linear networks to automatic control, signal processing, and data communication. As it provides an appropriate level of abstraction for system design with regard to modeling power and simulation efficiency, signal-flow modeling is the best candidate to be supported by SystemC-AMS. As we'll see later, it also provides a natural interface to the world of discrete time MoCs.

Supporting conservative-law modeling is required as well to support multi-domain systems. Conservative-law models interface themselves less directly with discrete-time models as signal-flow models. This is however still feasible, for example either by embedding conservative-law models into signal-flow models [14], or by providing the appropriate interface models (mixed-signal or mixed-domain interfaces).

It is planned to only support signal-flow modeling in the first release of SystemC-AMS.

SystemC-AMS must provide a, possibly generic, way to handle interactions between MoCs.

SystemC 2.0 provides a very flexible way to model the communication of systems by specifying a model for communication and synchronization in a channel and providing an interface for the channel, which can be used independently from a specific realization of that channel. This is generic enough to describe systems using various MoCs, including both discrete-time and continuous-time MoCs.

Following the layered approach advocated in SystemC, it is planned that SystemC-AMS will eventually support several specialized continuous-time MoCs and their associated solvers, e.g. a solver for linear DAE/ODE systems, a solver for static DC/AC/noise analysis, a solver for nonlinear DAE systems, a solver optimized for modeling and simulating electrical power systems or mechanical systems.

In all generality, SystemC-AMS has to address the interactions both between continuous-time MoCs and between continuous-time and discrete-time MoCs. On the one hand, interactions between continuous-time MoCs, such as the coupling between a static DC/AC/noise solver and a dynamic linear DAE solver, may be non existent as each continuous-time solver may implement all required numerical methods. An example is a linear DAE solver that can compute the DC operating point of the system of equations.

On the other hand, interactions between continuous-time and discrete-time MoCs has to be formally defined. Here comes the concept of a dedicated manager, let us call it the *synchronization layer*, in the SystemC-AMS framework. An example of a formal definition of the synchronization between an event-driven solver and a continuous-time solver is given in the definition of the VHDL-AMS hardware description language [19]. Another example of more general mixed discrete-time/continuous-time synchronization is implemented in the Ptolemy II environment [16].

For the first release of SystemC-AMS, it is planned to support synchronization between the synchronous dataflow (SDF) MoC and the CT MoC implemented as a linear ODE solver for signal-flow models. These are the MoCs that can be interfaced in the most natural and easy way. SDF models are dataflow models in which each vertex consumes and produces a fixed number of tokens per activation. They have the nice property that a finite static scheduling can always be found. Linear ODE systems of equations can be solved using a fixed integration time step that can be synchronized with the rate at which samples are handled by the SDF model.

Using constant time steps is appropriate for signal processing systems, most of which being oversampled systems. The simulation of control systems, however, usually requires solving stiff nonlinear systems of equations. This will require to also support nonlinear DAE solvers and variable integration time steps in SystemC-AMS. Ultimately, the synchronization layer will be formally defined to allow supporting more mixed discrete-time/continuous-time synchronization schemes whenever possible or making sense.

It is important to note that the synchronization also requires the formal definition of a consistent initial (quiescent) state for the whole mixed-signal system, otherwise

the simulation of the continuous part of the model may either fail or be inaccurate at best.

SystemC-AMS must provide appropriate views (or description layers) for the description of continuous-time models.

The interface layer provides the solver with the system of equations to solve. This system of equations can be, for example, generated from a network using the Modified Nodal Analysis method or from a behavioral representation like a transfer function or state-space description. The same interface can be useful for different solvers (e.g. linear /nonlinear DAEs). The realization must however take into account that the mapping to each solver layer is different. At least the following interfaces should supported: a netlist interface that should be common to all underlying continuous-time MoCs, and an equation interface that should allow a user to formulate behavioral models or functional specifications in a more natural way as a set of DAEs.

SystemC-AMS must support the coupling with existing continuous-time simulators.

SystemC-AMS will be essentially, as any other SystemC extension over the core language, a library of C++ classes and methods that allow designers to develop system-level executable specifications of mixed-signal (analog-digital) and mixed-domains (e.g. electro-mechanical) designs. It will be by no means designed to replace existing circuit-level or system-level continuous-time simulators/solvers. Rather, it will provide an open architecture in which existing, mature, simulators or solvers may be plugged in and coupled with discrete-time MoCs.

4. Seed Works

A number of research works already took advantage of the programming capabilities offered by SystemC to develop own analog extensions and to get a mixed-signal simulation framework "for free". All the works presented here have developed their own specialized C++ classes and methods as well as their own libraries of modules.

In [2], Bonnerud *et al.* present such a mixed-signal simulation framework with an application to the design of pipelined A/D converters. The approach proves to be useful to model a circuit-level technique, the digital noise cancellation technique, to allow an efficient exploration of pipelined architectures at a more abstract level, while achieving comparable accuracy to MATLAB. The module library includes functional models of relatively complex mixed-signal elements (e.g. flash ADC, switched capacitor DAC, or operational amplifier).

In [6], Einwich *et al.* discuss the synchronization between synchronous dataflow and linear continuous-time MoCs using a fixed time step. The module library includes primitive electrical elements (R, L, C, sources) and transfer functions. The framework also allows the simulation of mixed-signal system in the frequency domain, provided frequency-domain models are added to the discrete-time components in the system.

In [8], Grimm *et al.* present a framework for simulating power electronic components. This is an example of a dedicated framework as it provides for an efficient simulation of a specific family of power circuits, namely power drivers with capacitive or inductive loads. The coupling with the discrete-time world remains simple and efficient thanks to the limited number of supported power circuit architectures. The module library includes primitive electrical elements (R, L, C, sources, transistors).

In [9], Grimm *et al.* go a step further and present a top-down modeling and simulation methodology based on a refinement process. The issue here is not the module library per se, but the synchronization mechanism between synchronous dataflow and continuous-time models at different levels of abstraction, from high-level mathematical models to more physical, pin-accurate, models. The refinement process takes advantage of existing object-oriented features of SystemC.

Last, but certainly not least, the work on the Ptolemy II framework [16], although not supporting SystemC, deserves a special attention. The way discrete-time and continuous-time MoCs and their relative synchronization mechanisms are implemented in this environment can provide useful insights on how developing AMS extensions to SystemC.

5. Conclusions and Future Work

This paper described the context in which the analog and mixed-signal extensions to SystemC, called SystemC-AMS, will be developed. A number of design objectives have been defined from requirements related to different application domains.

As the application domains of AMS extensions are pretty diverse in their requirements, it is contemplated that the development will go over three phases, each phase adding new capabilities:

1. Support of signal processing dominated applications. This includes:
   - The support of linear continuous-time MoCs with behaviors expressed at the signal-flow level as a sequence of assignments.
   - Time-domain simulation.
   - Mixed-signal synchronization will be provided with the synchronous dataflow MoC using fixed time

steps. The coupling between discrete-time and continous-time MoCs is expected to be "weak", e.g. through control or status signals only.
- A library of primitive elements, e.g. R, L, C, sources, and special functions, e.g. transfer functions.

2. Support of RF/wireless applications. This includes:
   - The support of non linear DAEs and their simulation using variable time steps.
   - The formulation of implicit equations, e.g. true simultaneous statements.
   - Frequency-domain simulation.
   - An enriched mixed-signal library with more complex functional (signal-flow) models, e.g. amplifiers, converters.

3. Support of automotive applications. This includes:
   - Specialized continuous-time MoCs, e.g. for power electronics or mechanical systems.
   - Support of conservative-law models.
   - Enrichment of the mixed-signal library with conservative-law mixed-domain models.
   - Definition of a generic synchronization mechanism between discrete-time and continuous-time MoCs, including software MoCs.

To achieve these goals, a proposal [7] to form an OSCI Working Group to develop SystemC-AMS has been submitted recently (July 2002) to the OSCI Board of Directors.

6. References

[1] D.G. Augustin *et al.*, "The SCi Continuous System Simulation Language (CSSL)", Simulation, 9, pp. 281-303, 1967.

[2] T.E. Bonnerud, B. Hernes, T. Ytterdal, "A Mixed-Signal, Functional Level Simulation Framework Based on SystemC for System-on-a-Chip Applications", Proc. IEEE 2001 Custom Int, Conf. (CICC), 2001.

[3] K.E. Brennan *et al.*, *Numerical Solutions of Initial-Value Problems in Differential-Algebraic Equations*, Classic Appl. Math. 14, SIAM, 1996.

[4] F.E. Cellier, *Continuous System Modeling*, Springer-Verlag, 1991.

[5] J.M. Cho *et al.*, "Design and Implementation of HILS System for ABS ECU of Commercial Vehicles", Proc. IEEE ISIE Conference, 2001.

[6] K. Einwich *et al.*, "SystemC Extensions for Mixed-Signal System Design", Proc. 2001 Forum on Design Languages (FDL'01), 2001.

[7] K. Einwich *et al.*, "Analog Mixed Signal Extensions for SystemC", White paper and proposal for the foundation of an OSCI Working Group (SystemC-AMS working group), June 2002, http://mixsigc.eas.iis.fhg.de/.

[8] C. Grimm *et al.*, "AnalogSL: A Library for Modeling Analog Power Drivers in C++", Proc. 2001 Forum on Design Languages (FDL'01), 2001.

[9] C. Grimm *et al.*, "Refinement of Mixed-Signal Systems with SystemC", Submitted to DATE'03.

[10] T. Grötker *et al.*, *System Design with SystemC*, Kluwer, 2002.

[11] J.G. Kassakian, D.J. Perreault, "The Future of Electronics in Automobiles", Proc. IEEE International Symp. on Power Semiconductor Devices 2001, pp. 15-19, 2001.

[12] K. Kundert, "Simulation Methods for RF Integrated Circuits", Proc. IEEE ICCAD'97, pp. 752-765, 1997.

[13] P. Lieverse *et al.*, "A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems", Proc. IEEE Workshop on Signal Processing Systems, pp. 181-190, 1999.

[14] J. Liu, *Continuous-Time and Mixed-Signal Simulation in Ptolemy II*, UCB/ERL Memo M98/74, Univ. of California, Berkeley, 1998.

[15] Open SystemC Initiative, http://www.systemc.org/.

[16] Ptolemy, University of California, Berkeley, http://ptolemy.eecs.berkeley.edu/ptolemyII/.

[17] C.-J. R. Shi, A. Vachoux, "VHDL-A Design Objectives and Rationale", CIEM, Vol. 2: Modeling in Analog Design, pp. 1-30, Kluwer, 1995.

[18] G. Vanderspeeten *et al.*, "A methodology for efficient high-level dataflow simulation of mixed-signal front-ends of digital telecom transceivers",Proc. IEEE DAC 2000, pp. 440-445, 2000.

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