

Mixed-Signal Extensions for SystemC

K. Einwich

P. Schwarz

Fraunhofer IIS/EAS

Zeunerstraße 38

D-01069 Dresden, Germany

Karsten.Einwich@eas.iis.fhg.de

Ch. Grimm

K. Waldschmidt

Univ. Frankfurt

Robert-Mayer-Straße 11-15

D-60054 Frankfurt, Germany

grimm@informatik.uni-frankfurt.de

Abstract

SystemC supports a wide range of Models of Computation (MoC) and is very well suited for the design and refinement of HW/SW-systems from functional down to register transfer level. However, for a broad range of applications the digital parts and algorithms interact with analog parts and the continuous-time environment. Due to the complexity of these interactions and the dominance of the analog parts in respect to the system behavior, it is essential to consider the analog parts within the design process of an Analog and Mixed Signal System.

Therefore simulation performance is very crucial - especially for the analog parts. Thus different and specialized analog simulators must be introduced to permit the use of the most efficient solver for the considered application and level of abstraction. In this paper we describe possible areas of application and formulate requirements for analog and mixed-signal extensions for SystemC.

1. Introduction and Motivation

SystemC 2.0 [1] provides a very flexible methodology for the design and refinement of complex digital HW/SW-systems. This methodology is strongly influenced by the communication model introduced by Gajski [10]. In this methodology *modules* which consist of other modules or algorithms implemented in methods communicate via *channels*. A set of methods for communication is specified in an *interface*. These methods are realized in the channel. Modules can call methods of a channel, and events in a channel can activate methods in a module connected to the channel. This concept is generic enough to describe systems using various models of computation, including static and dynamic multirate dataflow, Kahn process networks, communicating sequential processes, and discrete events. We call such systems *discrete systems*.

However, the discrete approach to modeling and simulation is not applicable to systems, whose behavior is specified by differential and algebraic equations. In the following, we call such systems *analog systems*. Note, that the equations can also be given implicitly by an electrical netlist, by a mechanical model description or by transfer functions, for example. For many classes of systems, the analog parts will become more dominant in respect to performance parameters, power consumption, silicon cost and yield. Analog systems are simulated by mathematical methods that compute a solution for the underlying set of equations. For simulation of analog systems, a simulator ("*solver*") computes a solution of the differential and algebraic equations,

using linear and non-linear equation solving algorithms and numerical integration techniques to compute signal values for a time interval. For combined analog/discrete (“mixed-signal”) simulation, discrete event simulators are coupled and synchronized with analog solvers.

In the following, we give an overview of the analog and mixed-signal extensions for SystemC discussed within the “SystemC-AMS working group”[5] (AMS=Analog and Mixed-Signal). First approaches for analog extensions are described in [2,3,4]. The synchronization mechanisms discussed in this paper are well known [6,7,8,9]. The main focus of this paper is to give an overview of the requirements for AMS extensions for SystemC and first approaches for their integration in SystemC:

- Which concept for AMS extensions corresponds well to the levels of abstraction, on which SystemC is used?
- How can AMS extensions be integrated in SystemC?

In section 2, areas of application and the level of abstraction, on which AMS-extensions should be applied is described. Section 4 gives an overview of the concept and introduces the layered approach for the integration of the AMS extensions into SystemC.

2. SystemC-AMS: Areas of Application and Requirements

Analog and mixed-signal systems can be found in many applications. The requirements for modeling and simulation depend both on the area of application and the level of abstraction of the model. SystemC is used for system-level design tasks, such as the modeling and refinement of hardware/software – systems. In such a context, analog models are used most notably for the following tasks:

Executable Specification: Analog models are often used as an executable specification of signal processing functions. Currently, interactive tools with a graphical or textual interface such as Matlab/Simulink are used for such tasks.

Behavioral Modeling: In the design of analog systems, there is always a large “bottom-up” fraction. Behavioral modeling of existing analog netlists allows the simulation of analog circuits in a reasonable time.

Co-Simulation with Environment: On system level, the analog (continuous-time) environment is co-simulated with the embedded system. This allows a rough validation of an executable specification. Furthermore, many designs can only be validated in such a co-simulation.

In difference to digital systems, analog systems often combine different physical domains and are very application-specific. Therefore, on one hand, the use of analog and mixed-signal components in concrete applications must be considered. On the other hand, the approach must still be open for new applications and methodologies. For the discussion of application-specific requirements, three application fields are discussed: signal processing, RF/wireless and automotive applications.

In telecommunication and multimedia applications, the modeling of signal processing functions is dominant. These systems are mostly sampled or even over sampled using constant time steps. The system is modeled by a block diagram with directed signal flow. The blocks are described by linear transfer functions and weak or static non-linear functions. Often, such a system level description is combined with linear networks, which are used for macro modeling or for modeling the system environment.

In RF and wireless communication applications, systems are also specified by block diagrams with directed signal flow. A necessary feature for RF applications is the ability to simulate the baseband behaviour.

In the automotive domain, analog and mixed-signal systems are often non-linear systems, and usually embrace multiple physical domains (electrical, mechanical, fluidic, etc.). In difference to telecommunication and multimedia applications, control systems in the automotive domain are often systems with very different time constants (“stiff systems”). Nevertheless, executable specifications and executable prototypes are also modeled as block diagrams with directed signal flow.

The above mentioned requirements are partially contradictory: On the one hand, rather simple block diagrams connected with directed signal flow and some external components seem to be sufficient for many applications on a high level of abstraction. On the other hand, some requirements can only be fulfilled by solutions, which are specific for concrete applications and levels of abstractions. For example, for the simulation of electronic circuits a designer might want to use dedicated circuit simulators, such as SABER or SPICE, and for the precise simulation of mechanical components a dedicated simulator for mechanical systems, and so on.

The situation discussed above leads us to an *open approach*. In such an approach, SystemC-AMS extensions are used for the simulation of executable specifications, behavioral models and the environment as far as possible. Furthermore, the AMS extensions provide a synchronization mechanism that permits the easy integration of additional solvers. These additional solvers can be specific for (maybe new) applications and levels of abstractions that are not covered by the “built-in” solvers. The resulting scenario is shown in Figure 1.

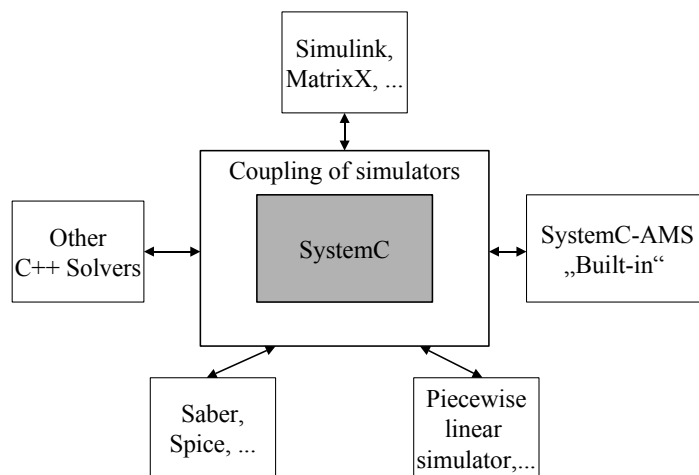


Figure 1: Use of SystemC-AMS in a heterogeneous tool-environment.

In addition to the requirements derived from the applications, AMS extensions for SystemC depend on the design and existing features of SystemC. SystemC already provides a generic and flexible concept for the modeling of computation and communication in discrete, heterogeneous systems. The existing SystemC is based on a layered structure of a C++ class library. Basically, SystemC consists of a generic discrete event simulator kernel, data types for modeling digital hardware, and templates for modeling communication and synchronization in different discrete models of communication (MoCs). In order to make communication and synchronization flexible, signals and ports use (communication) interfaces. For this reason, SystemC is more than just a discrete-event simulator. SystemC allows users to add new models of computation, as long as this is feasible with the discrete kernel. Such additional models of computation can be provided by

libraries, for example, which define new classes of signals. The good object-oriented design of SystemC even invites users to make such extensions by simple inheritance.

In this context, mixed-signal extensions for SystemC should provide a basic, object-oriented framework, where users can integrate appropriate and domain-specific “analog MoCs” – and not only an analog simulator which is coupled with the digital SystemC kernel. The mixed signal extensions shall allow users the integration of new solvers or alternative synchronization schemes in an as generic and flexible way as in the discrete domain. To allow the user this flexibility, a well designed object oriented structure of the mixed-signal extension library will be required.

In order to meet the different requirements, the AMS extension library is structured in four different layers. The structure of this “layered approach” is shown in Figure 2. On top of the existing standard SystemC kernel, a *synchronization layer* provides methods to couple and synchronize different analog solvers and the discrete event kernel of SystemC. All communication between different MoCs takes place via this synchronization layer. On top of the synchronization layer, different analog MoCs can be realized by different analog solvers (*solver layer*) and the discrete event SystemC kernel. The *view layer* provides convenient user interfaces for the analog solvers. The view layer converts user views, for example netlists, to a set of differential or algebraic equations in the solver layer. The interaction of solver- and view layer can be compared to a model/view architecture, where one object provides an abstract data structure, another objects provide graphical views to the data structure.

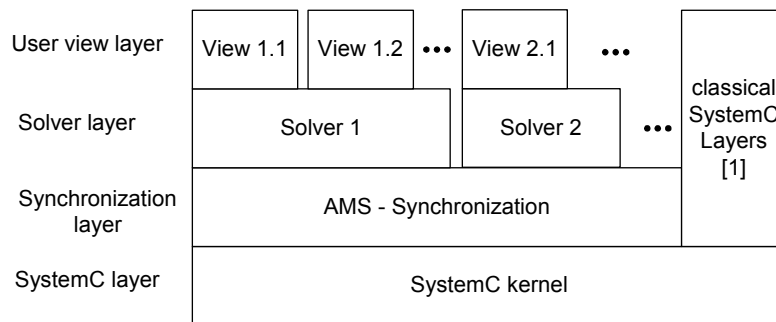


Figure: 2 Layered approach for SystemC-AMS extensions [5].

3. The Layered Approach in Detail

The most critical part of the AMS extensions is the synchronization layer. The synchronization layer encapsulates the analog solvers as far as possible, and allows the connection of blocks simulated by different solvers or instances of the same solver for IP (intellectual property) encapsulation. In general, the synchronization layer handles the following tasks:

- The synchronization layer determines a step width for the simulated time, respectively the points in time, at which the solvers and the discrete event kernel are synchronized.
- The synchronization layer determines an order, in which the single solvers simulate.
- Cyclic dependencies can represent additional equations, which have to be solved. The synchronization layer handles this.

The concrete realization of the above functionality is rather application-specific. In order to limit the complexity, we make some restrictions:

1. For the communication via the synchronization layer, we suppose a directed signal flow. In the considered areas of application, the signal flow between system-level components is

directed. Therefore, the realization of a directed signal flow on the synchronisation layer is sufficient. Note that - if necessary - a bidirectional coupling (e. g. for a coupling on netlist-level) can be realized by two separate signals in both directions. Furthermore, netlists or systems that are strongly coupled can (and should!) be modeled and simulated using one single solver without communication via the synchronization layer.

2. We do not plan to support backtracking (setting the simulated time back to the past). This would require the ability to save a state and to restore the saved state. We cannot expect solvers in general to support such methods. For this reason, the synchronization layer must only increase the simulated time.

In the telecommunications or RF domain, digital signal processing functions are usually oversampled with known and constant time steps. Therefore, a synchronization of the analog solvers and the digital kernel in constant time steps is sufficient. The order in which the solvers are computed is usually determined by the signal flow. This allows us to determine the order of execution of the individual simulators before the simulation. The underlying synchronization principle is the static dataflow model of computation.

In the automotive domain, systems are often nonlinear and stiff systems. The use of a constant step width could lead either to a large error, or to a very bad simulation performance. Furthermore, the coupling of analog and digital components is often very close. Therefore, the step width has to be determined during simulation by the synchronisation layer, controlled by parameters such as the local quantization error and/or synchronization requests and events of single simulators.

In order to provide a maximum of flexibility, a generalized, abstract “coordinator interface” provides access to the synchronization layer. This interface is by default realized by the rather simple, but stable static data-flow approach. In order to realize more complex synchronization schemes, the interface methods can be replaced by other, more sophisticated methods.

Simulation with the default method: Static dataflow

In order to determine an appropriate order of execution, the methods known from static dataflow MoC can be used. In the static dataflow MoC, cyclic dependencies are transformed into a pure linear dependency by assuming delays in each cycle as shown in Figure 3. The static dataflow MoC defines the order, in which the analog modules are simulated (1-4).

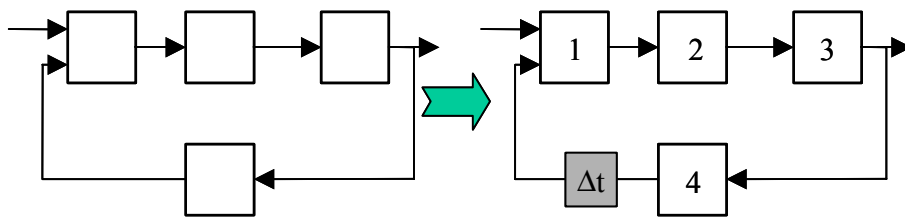


Figure 3: Synchronization of four analog modules with the static dataflow MoC.

However, the static dataflow MoC is an untimed model - the delay introduced to break the cyclic dependency has no numerical value. In order to simulate analog systems, time is mandatory. Therefore, we have to determine a concrete value for the introduced delay. A simple method is to ask each solver for the next point in the simulated time, when a synchronisation is required, and to select the minimum of all responses for all solvers. Then this value determines the value of the delay Δt . Furthermore, a user can specify a minimum and a maximum step width of simulators.

The above mentioned method for synchronisation looks very simple, but fulfills the requirements: At runtime, no scheduling effort has to be done, which makes computation of large structures with a large data volume possible, for example in telecommunication applications. All simulators can define synchronisation points, and no simulator is required to support backtracking. This allows the integration of very different solvers for the simulation of heterogeneous systems.

Possible improvements of the synchronisation methods

The default methods for synchronisation are rather general methods, that permit the integration of nearly all solvers. Nevertheless, a user might need a faster method that reduces the number of points for synchronisation, maybe on the cost of simulation precision or with the risk of making errors – whatever is appropriate for his application. In order to reduce the number of synchronisations required, a single solvers can be allowed to simulate for larger time steps than the step width chosen by the synchronisation layer. In this case, it might be necessary to *locally* do a backtracking within this solver. Note, that this backtracking is only local backtracing within one solver.

Other possible improvements are related to the way, in which the equations introduced by the structure of simulators are solved. When several analog models are coupled, the overall system can be seen as a more complex analog model. The simulation of this overall analog model would require methods that solve the overall set of equations. In a system, where very different approaches for modeling and simulation are combined, this is not applicable. In Figure 3, we have introduced a delay to explain the computational model used for the synchronization of the simulators. This method can also be seen as a relaxation method that computes a solution of the overall set of equations (The scheduling with the static dataflow approach can be seen as a sorting of the matrix according to the dataflow dependencies). Stability and convergence can then be improved by known methods that improve stability and convergence for relaxation, for example by a method that extrapolates a possible future waveform from the known waveform.

In general, the synchronization layer provides an open platform for programmers and experienced designers to include alternative solvers and maybe alternative synchronization methods.

4. Solver Layer and User View Layer

A solver provides methods that simulate an analog system, e. g. a solver for a linear DAE-system, or another solver for a non-linear system, or a solver optimized for power electronics. A solver must be able to interact with the synchronization layer. For interaction with the synchronisation layer, he must provide at least methods that

- determine the next required synchronisation point, if a variable step width synchronization is required.
- resume simulation with a given input waveform to a specified point in the simulated time.

The restrictions made in the synchronization layer lead to the following restrictions for the solvers:

- A solver may not set the simulated time of the overall simulation to a point in the past, because the synchronization layer does not support backtracking.
- A solver must accept synchronisation points from other simulators the synchronisation layer resp., because the synchronization layer determines the synchronization points.

Solvers simulate models, that are specified by differential and algebraic equation. Those systems of equation are usually represented in a solver specific form, e.g. as constant or time dependent matrices. Examples for solvers that fulfill the above requirements are presented in [2, 3]. However, usually designers do not specify systems in such rather solver-specific data structures.

A designer should describe systems using the *(user) view layer*. The view layer provides the solvers with the above mentioned matrices. This representation of a system of equation can be generated from a network using the modified nodal analysis, or from a behavioral representation like transfer function or state space equations, for example. The same view can be useful for different solvers (e.g. linear/nonlinear DAE's); nevertheless, the realization must take into account that the mapping to the solver layer is different. At least the following views should be part of the SystemC-AMS extension library:

- *Netlist view*: This view should be common to all underlying solvers.
- *Equation view*: This view should allow a user to formulate behavioral models or functional specifications in a direct way as a differential algebraic equation.

Further views could be a possibility to describe finite element systems, for example – in general, the library should allow the user to add new and application specific views to a solver.

5. Conclusion

We have discussed requirements for analog and mixed-signal extensions for SystemC in a broad range of applications, for example in telecommunication, multimedia and automotive applications. The layered approach covers nearly all requirements for system-level design in these applications and can be extended in an easy way. The layered approach structures the extensions into a synchronization layer, a solver layer and a view layer.

Considering the requirements on system level, an open, general, but maybe less precise synchronization concept is more appropriate than a complicated precise, but closed concept. With the possibility to overload single methods of the synchronisation layer, the synchronization layer can be changed to more efficient synchronization methods – if required. The possibility to add solvers or external simulators in the solver layer allows us to include application-specific simulators in an easy and convenient way.

The proposed structure and concept of an analog mixed-signal extension library can significantly improve the generality of the SystemC approach: SystemC is currently restricted to the design of pure discrete systems. The proposed AMS extensions can make it “open” for the design of heterogeneous systems.

References

1. An Introduction to System-Level Modeling in SystemC 2.0. Technical report of the Open SystemC Initiative, 2001. http://www.systemc.org/technical_papers.html
2. Karsten Einwich, Christoph Clauss, Gerhard Noessing, Peter Schwarz, and Herbert Zojer: "SystemC Extensions for Mixed-Signal System Design". Proceedings of the Forum on Design Languages (FDL'01), Lyon, France, September 2001.
3. Christoph Grimm, Peter Oehler, Christian Meise, Klaus Waldschmidt, and Wolfgang Fey. "AnalogSL: A Library for Modeling Analog Power Drivers with C++". In Proceedings of the Forum on Design Languages", Lyon, France, September 2001.
4. Thomas E. Bonnerud, Bjornar Hernes, and Trond Ytterdal: "A Mixed-Signal, Functional Level Simulation Framework Based on SystemC System-on-a Chip Applications". Proceedings of the 2001 Custom Integrated Circuits Conference, San Diego, May 2001. IEEE Computer Society Press.
5. K. Einwich, Ch. Grimm, A. Vachoux, N. Martinez-Madrid, F. R. Moreno, Ch. Meise: "Analog Mixed Signal Extensions for SystemC". White paper of the OSCI SystemC-AMS Working Group.
6. L. Schwoerer, M. Lück, H. Schröder: "Integration of VHDL into a Design Environment", in Proc. Euro-DAC 1995, Brighton, England, September 1995.
7. M. Bechtold, T. Leyendecker, I. Wich: "A Dynamic Framework for Simulator Tool Integration", Proceedings of the 2nd International Workshop on Electronic Design Automation Frameworks, Charlottesville, 1990.
8. M. Bechtold, T. Leyendecker, M. Niemeyer, A. Ocko, C. Ocko: "Das Simulatorkopplungsprojekt", Informatik-Fachbericht 255, Springer Verlag, Berlin .
9. G. Nössing, K. Einwich, C. Clauss, P. Schwarz: "SystemC and Mixed-Signal – Simulation Concepts", in Proc. 4th European SystemC Users Group Meeting, Copenhagen, Denmark, October 2001.
10. D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, S. Zhao: „SpecC Specification Language and Methodology“, Kluwer Academic Publisher 2000