Towards Analog and Mixed-Signal SOC Design with SystemC-AMS

Alain Vachoux
EPFL-LSM, Switzerland
alain.vachoux@epfl.ch

Christoph Grimm
Univ. Frankfurt, Germany
grimm@cs.uni-frankfurt.de

Karsten Einwich
Fraunhofer IIS/EAS, Germany
karsten.einwich@eas.iis.fhg.de

Abstract

Systems-on-Chip (SoCs) are heterogeneous by nature as they may integrate digital, analog, RF hardware as well as software components or non electrical parts such as sensors or actuators. The increasing level of complexity for designing SoCs in a reasonable amount of time and resources asks, among other capabilities, for powerful modeling and simulation means. SystemC is emerging as a de facto standard for digital system design, but is still lacking a standard support of continuous-time and mixed discrete-event/continuous-time systems. This paper presents the first elements of extensions to SystemC, called SystemC-AMS, that are proposed to fill the gap.

1. Introduction

A system on chip (SOC) is defined as a complex integrated circuit that integrates the major functional elements of a complete end-product into a single chip or chipset [1]. One big challenge in today's SOC designs is their heterogeneity. They encompass digital, analog and mixed-signal hardware and software components and even non electrical parts such as sensors and actuators (MOEMS: micro opto-electronic mechanical systems). The design and the verification of such systems therefore require powerful modeling and simulation means to address all aspects consistently and efficiently.

SystemC is emerging as a de facto standard for digital system design. It provides an open framework for describing the structure and the behavior of discrete-event hardware systems, possibly also including software [2][3]. SystemC however still lacks a standard support for analog and mixed-signal systems, or, more generally, for the modelling and the simulation of continuous-time systems and so of mixed continuous-time/discrete-event systems. To fill this gap, a SystemC-AMS Study Group has been formed in 2002 with the mission to develop the so-called analog and mixed-signal (AMS) extensions to SystemC, hence the name SystemC-AMS. The work of the Study Group is driven by a number of objectives [4] and is building on several preliminary works [5][6].

The goal of this paper is to present an overview of the first version of the SystemC-AMS functional specification and to show how they integrate with the existing SystemC 2.0 implementation. Section 2 gives a brief overview of SystemC 2.0 and highlights the features that are exploited and extended in SystemC-AMS. Section 3 presents the context in which SystemC-AMS is being developed. Section 4 summarizes a first attempt to develop a consistent set of functional specifications for SystemC-AMS. Section 5 gives a couple of illustrative, but non definitive, examples. Section 6 brings some conclusions and highlights future work.

2. SystemC 2.0 overview

SystemC is a set of C++ classes and methods that provides a powerful means to describe the structure and the behavior of hardware/software systems from abstract specifications to register transfer level (RTL) models. A simulation semantics is defined by a scheduler that supports several possible execution models or models of computation (MoC) such as static and dynamic multi-rate dataflow and discrete event. The latest release 2.0 of SystemC offers constructs for generalizing the modeling of communication and synchronization as well as the support of transaction-level modeling and system-level verification [7].

In SystemC, modules encapsulate concurrent processes and have ports to communicate with the outside world. Processes encapsulate sequential behavior, but may execute asynchronously (concurrently) between each others.

Each port is associated with an interface which defines the possible *abstract* operations (e.g., read or write). A channel implements an interface by defining how the operations are performed. There might be several different channels for the same interface, hence supporting one form of refinement from abstract communication such as FIFO queues to complex protocols with embedded processes.

Primitive channels belong to a special class of channels that supports the *request-update* communication mechanism at the heart of discrete event simulation. This mechanism basically allows for simulating concurrency and guarantees that the result of simulation does not depend on the order in which processes are executed.

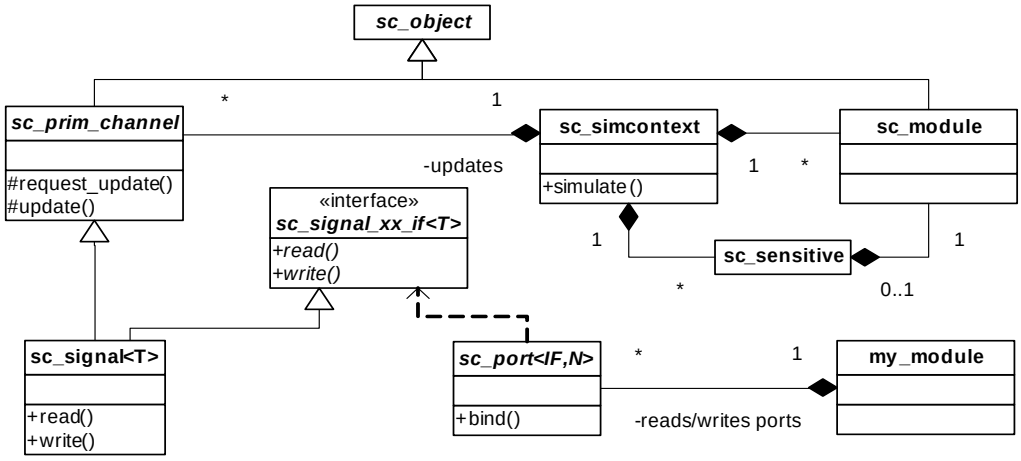

Figure 1. Simplified SystemC class organisation in UML notation.

SystemC's object-oriented approach provides the most essential features to support system-level design. *Overloading*, *specialization* and *refinement* allow for progressively refining an abstract specification to a detailed model that can be used as input for (behavioral) synthesis or software compilation, while *polymorphism* allows for hiding the refinement process behind unique interfaces.

The SystemC class hierarchy is rooted to the class sc_object that provides methods for the basic management of object attributes such as name and kind. Figure 1 gives a simplified view of the SystemC class organization in UML notation. The class sc_module defines methods to manage concurrent processes and provides a context to support structural composition through ports. The class sc_prim_channel implements methods that realize the basic request-update simulation mechanism of the discrete-event MoC. A user-defined module has typically ports that belong to a particular interface. The abstract class sc_port defines the binding of the port to the interface. There are several possible interfaces available for the discrete event MoC that are represented in Figure 1 by the class sc_signal_xx_if<T> (it is actually not existing under this name; xx must be replaced by in, inout or out). This class declares the virtual methods read and write to operate on the associated channel. Finally, the class sc_signal realizes the actual channel and implements the read and write methods. This channel is typically used to model digital systems at the Register Transfer Level (RTL) or at the gate level.

The primitive channel class provides an abstract framework for discrete event MoC. SystemC also provides other specialized primitive channels than sc_signal for that MoC that support other kind of communication, namely intra-module communication, semaphores or bounded FIFO queues. All these different communication mechanisms can be bound to the same module ports since all of them implement the same interface.

3. SystemC-AMS objectives

The main objective of SystemC-AMS is to provide an efficient means for modelling and simulation of heterogeneous systems. The support of continuous-time systems shall be seamlessly integrated in the existing SystemC framework. To that end, the development of the AMS extensions is planned in three phases [4]:

- Phase 1 is currently ongoing and is addressing signal processing dominated applications. This is done by providing a support for linear dynamic continuous-time modelling, predefined linear operators (transfer functions, state-space equation formulation), linear network elements, and a synchronization between continuous-time and discrete-event model parts using a static dataflow scheduler with fixed time steps.
- Phase 2 will address RF/wireless applications by providing a support of nonlinear differential algebraic equations (DAEs) simulation using variable time steps and frequency-domain simulation.
- Phase 3 will address automotive applications by providing a support of conservative-law systems and a generic synchronization mechanism between continuous-time and discrete-time model parts.

4. Functional specifications

4.1. Layered approach

The AMS extensions to SystemC are being defined using a layered approach [8] (Fig. 2). The base layer is the existing SystemC 2.0 kernel. On top of the base layer, two sets of layers are defined: on the one hand there is the set of the existing SystemC 2.0 layers, e.g. discrete-event channels,

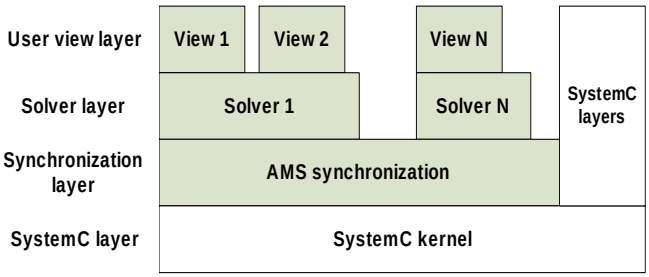

Figure 2. SystemC-AMS layered structure.

and on the other hand there is the new set of layers related to AMS extensions, namely, from the top:

- The *user view layer* provides different descriptive methods to write the executable continuous-time models, namely: procedural behavior, equations, transfer functions, state-space formulation, netlists of primitives.
- The *solver layer* provides different implementations of solvers that are required to simulate specific AMS descriptions. The solvers are related to the available views. For example, a view consisting of a netlist of electrical RLC primitives will need a linear solver using an equation formulation method such as the Modified Nodal Analysis method, while a view consisting of equations will need a more general nonlinear DAE solver. Specialized solvers, e.g. for the simulation of power networks or mechanical parts, are also considered.
- The *synchronization layer* implements a mechanism to organize the simulation of a SystemC-AMS model that may include different continuous-time views and discrete-event parts. It also defines a generic interface in which continuous-time solvers can be plugged.

4.2. Phase 1 semantic model

Phase 1 of the AMS extensions considers continuous-time descriptions to be embedded in dataflow clusters. A dataflow cluster may include any number of dataflow blocks whose execution order is statically scheduled. Each dataflow cluster is embedded in a separate discrete-event process called a *cluster process* that is managed by a *coordinator* (Fig. 3).

A dataflow cluster may be then selected and run at a constant time step during the AMS simulation. The time step is defined by the sampling rate of the signals in the cluster. It is assumed that the sampling rate of signals in dataflow clusters is much higher than twice the maximum frequency of the discrete-event signals in order to minimize quantization and interpolation effects. This assumption is realistic as most of signal processing applications use oversampling.

The embedding of dataflow clusters in discrete-event processes is done at elaboration time and is therefore not

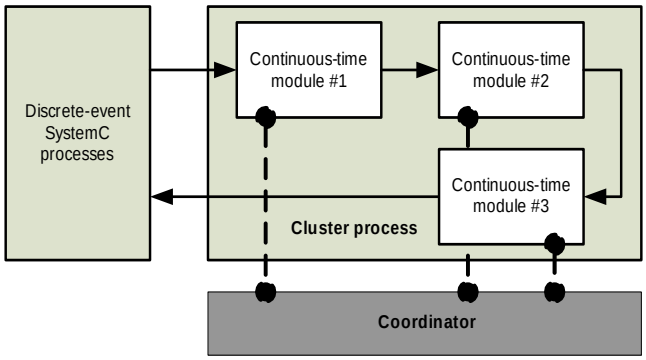

Figure 3. Continuous-time blocks embedded in a dataflow cluster.

explicitly expressed in the text of the model. SystemC does not formally has an elaboration phase as VHDL has, although it is considered that elaboration occurs when the SystemC library binds each port, and its related interface, to a designated channel.

A dataflow block encapsulates some continuous-time behavior in the form of a new kind of module called an *AMS module*. The synchronization between different continuous-time solvers and the event-driven SystemC kernel is done using the coordinator and *generalized signals* or *generalized channels*. The coordinator has the task of registering continuous-time modules and signals, and of defining the methods to handle the synchronization (e.g. time step selection).

Generalized signals provide a generic mechanism to define the coupling between continuous-time and event-driven modules. More importantly, they allow true object-oriented model refinement from abstract specifications to detailed implementations.

4.3. The AMS module

An AMS module is described with the macro SCA_MODULE as follows (the sca_ prefix denotes an AMS extension to distinguish from regular SystemC 2.0 item names which start with the sc_ prefix):

```
SCA_MODULE(module-name)
   // ports, internal data, member functions
   SCA_CTOR(module-name) {
      // constructor for SDF synchronization
      SCA_SDF_ATTR(attribute-function)
      SCA_SDF_INIT(initialization-function)
      SCA_SDF_SIGPROC(signal-processing-function)
      SCA_SDF_POST(post-processing-function)
   }
};
```

The SCA_MODULE macro is actually a short-hand for the definition of the class sca_module as:

```
class sca_module : public sc_module { ... };
```

The module may register at most four member functions: an attribute function, an initialization function, a signal processing function and a post-processing function. Only the signal processing function is required. The other functions are optional. As a consequence, no discrete-event processes are allowed in an AMS module.

The attribute function defines the values of useful attributes for the static dataflow simulation, namely the time step, the delay at the output ports and, in a future release, the rate of the multi-rate simulation. The initialization function is called once after elaboration is done and before simulation starts. The signal processing function defines the continuous-time behavior of the module in the form of procedural assignments. The post-processing function is called once and before simulation finishes.

The hierarchical composition of AMS modules is possible using regular SystemC modules (SC_MODULE). As a consequence, the macro SCA_MODULE defines continuous-time primitives.

4.4. Generalized signals

The modules in dataflow clusters communicate via directed signals. Signals define the model of communication but do not contribute directly to the system behavior. This allows for formally separating function from communication. Signals can also be used to provide appropriate couplings between different models of computation. In our case, a new class of signals must provide an interface between discrete-event and dataflow modules.

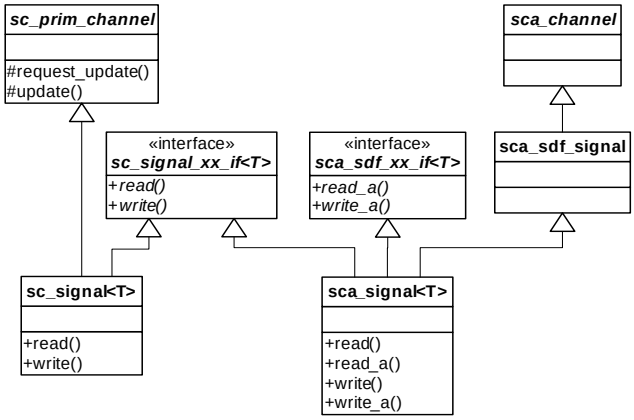

Figure 4. Simplified UML diagram of generalized signals and interfaces.

Figure 4 gives a simplified UML description of the new classes. The class sca_signal defines a generalized signal that implements methods for accessing both discrete-event signals (through methods read() and write()) and continuous-time signals (through methods read_a() and write_a()). This class also inherits from the class sca_sdf_signal which implements the static dataflow communication scheme.

As far single rate dataflow MoC is concerned, simple buffers to store continuous-time signal values are enough. It would be however possible to have more sophisticated dataflow communication mechanisms, e.g. using FIFOs or supporting multi-rate dataflow systems.

4.5. AMS ports

AMS modules may have two kinds of ports. One kind is defined by a new abstract class called sca_port, that supports continuous-time dataflow signals. Specialized (dataflow) signal ports are then derived form the abstract class, namely sca_in, sca_inout and sca_out and defined as (class sca_out is identical to sca_inout):

```
template <class T>
class sca_in : public sca_port<sca_sdf_in_if<T> > ...;

template <class T>
class sca_inout : public sca_port<sca_sdf_inout_if<T> > ...;
```

Another kind of abstract class, called sca_d_port, supports the use of a regular SystemC signals as control signals for the module. Specialized classes sca_d_in and sca_d_inout are defined as (again, class sca_d_out is identical to sca_d_inout):

```
template <class T>
class sca_d_in : public sca_port<sca_signal_in_if<T> > ...;

template <class T> class sca_d_inout :
   public sca_port<sca_signal_inout_if<T> > ...;
```

It should be noted that no regular SystemC ports are allowed in an AMS module as this would break the discrete-event/continuous-time synchronization.

4.6. Coordinator

The coordinator is responsible for the following tasks. During elaboration, it has to built the dataflow clusters from the structural organization of the AMS modules. As there should be one process per cluster, the coordinator has to instantiate a number of cluster processes. It also has to compute, if not defined, the time step for each module in the dataflow clusters. Single-rate dataflow simulation uses the same time step for all modules in a cluster. Finally, it has to determine a static scheduling of the modules in the dataflow clusters. Cyclic dependencies in a cluster must be broken by inserting a time step delay.

During simulation, the coordinator executes all signal processing functions in the order defined by the static scheduling and notifies the SystemC kernel to reactivate the computation at the next time step.

4.7. Mixed-signal simulation cycle

The SystemC 2.0 simulation cycle is proposed to be extended as follows to support the execution of the dataflow clusters:

1. *Initialization*. The initialization methods registered in AMS module are executed. This includes the definition of initial conditions.
2. *Evaluation*. Cluster processes are only executed at delta 0 in the order defined by the static scheduling (delta cycles provide a standard way to emulate concurrency when simulating discrete-event models.) The cluster processes will be reactivated, always at delta 0, at every time step defined for the cluster.
3. Repeat step 2 while there are still processes ready to run, else go to step 4.
4. *Update*. Signals are updated with their new values.
5. Go to step 2 if the signal updates generated events with zero delay (delta cycle), else go to step 6.
6. Finish simulation if there are no more pending events, else go to step 7.
7. Advance to the time of the earliest pending event.
8. Determine processes that are ready to run and go to step 2.

It should be noted that the proposed changes to the standard SystemC 2.0 simulation cycle are minimal as far as only the support of signal processing dominated applications is concerned. The support of a more general simulator coupling mechanism might require more substantial modifications of the simulation cycle, possibly affecting the existing SystemC kernel.

5. Examples

The first example shows how to embed the continuous-time behavior of a simple second-order lowpass filter in a static dataflow module. The transfer function of such a filter is given by Equation 1:

$$H(p) = H_0 \cdot \frac{\omega_p^2}{p^2 + \frac{\omega_p}{Q_p} p + \omega_p^2} \qquad (1)$$

The code in the right column shows one possible description of the lowpass filter module in SystemC-AMS. The transfer function is modeled using the predefined function sca_ltf which is assumed to be referenced in the include file sca_basic_lib.h. It is planned to enrich the so-called basic library with other functions such as state-space equations, sources, sinks, converters and arithmetic modules. The registered initialization function init computes the numerator and denominator coefficients while the registered signal processing function compute does the computation through the function sca_ltf. Note the selection of the actual gain with the control signal gain6db.

```
#include "sca_basic_lib.h"  // for accessing sca_ltf function
SCA_MODULE(lp_sdf) {
   // sdf ports
   sca_in<double>    LPIN;    // filter input
   sca_out<double>   LPOUT;   // filter output
   // control port
   sca_d_in<bool>    GAIN6DB;
      // if false, gain = 1.0, else gain = 2.0
   // parameters
   const double FC, QF;  // cut-off frequency, quality factor
   sc_vector<double> N, D;
      // LTF numerator and denominator
   // sdf methods
   void init(){
      double wp = 2.0*M_PI*FC;
      N[0] = wp*wp;
      D[0] = wp*wp;
      D[1] = wp/QF;
      D[2] = 1.0;
    }
   void compute(){
      double gain = 1.0;
      if (GAIN6DB.read()) gain = 2.0;
      LPOUT.write_a(gain*sca_ltf(N, D, LPIN.read_a()));
   }
   // constructor
   SCA_CTOR(lp_sdf) {
      SCA_SDF_INIT(init);
      SCA_SDF_SIGPROC(compute);
   }
};
```

The second example shows how linear electrical networks may be described. The support of linear electrical elements has not been discussed in this paper since it is not yet completely defined. However, as it is planned to be included in phase 1 it is good to get a first insight into how it could be done in SystemC-AMS. The use of linear electrical network descriptions at system level might seems a bit contradictory and overwhelming. They are usually used to specify blocks as simplified macromodels which may include parameters to allow design space exploration.

Figure 5 shows a simple linear network whose actual behavior depends on a control switch signal called hook. The following code shows one possible description of the network module in SystemC-AMS. It is assumed that the behavior of electrical primitives are defined in classes referenced in the include file sca_prim_lib.h. The detailed implementations for theses classes are not given here, but they essentially define the contribution of each element to a system matrix to be solved by an external solver [5]. The example also shows the use of a new kind of connection points called sca_node and one specific node called

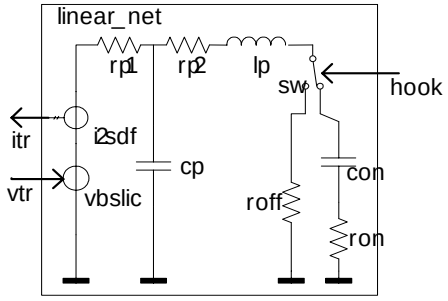

Figure 5. Simple linear network.

```
#include "sca_prim_lib.h"  // for electrical primitives
SCA_MODULE(linear_net) {
    // sdf ports
    sca_in<double>    VTR;     // network input
    sca_out<double>   ITR;     // network output
    // control port
    sca_d_in<bool>    HOOK;
    // local nodes
    sca_node n1, n2, n3, n4, n5, n6, n7;
    sca_reference gnd;
    // primitive instances
    sca_r      *rp1, *rp2, *roff, *ron;
    sca_c      *cp, *con;
    sca_l      *lp;
    sca_sw     *sw;
    sca_i2sdf   *i2sdf;
    // constructor
    SCA_CTOR(linear_net) {
        vbslic = new sca_vsdf("vbslic",w1,gnd,VTR);
        rp1    = new sca_r("rp1",w4,w2,60.0);
        rp2    = new sca_r("rp2",w2,w3,40.0);
        cp     = new sca_c("cp",w2,gnd,1e-12);
        lp     = new sca_l("lp",w3,w4,1e-3);
        sw     = new sca_sw("sw",w4,w5,w6,HOOK);
        roff   = new sca_r("roff",w5,gnd,600.0);
        ron    = new sca_r("ron",w5,w7,1e3);
        con    = new sca_c("con",w7,gnd,1e-6);
        i2sdf  = new sca_i2sdf("i2sdf",w1,w4,ITR);
    }
};
```

sca_reference. The full definitions for these new classes were not yet available by the time this paper was written.

6. Conclusions

The paper presented first elements of phase 1 of the extensions of the SystemC framework to support analog and mixed-signal systems, or more generally continuous-time and mixed discrete-event/continuous-time models of computations. The approach used in phase 1 is to embed continuous-time descriptions into discrete-event modules as a cluster of dataflow components. The simulation of dataflow components is done using a single-rate static scheduling. Although not explicitly shown in the paper, the definition of generalized signals and channels will support true object-oriented model refinement from abstract specifications to detailed implementations.

The concept of a global coordinator which manages the simulation of the continuous-time parts and the synchronization with the discrete-event systemC kernel is a first step towards a more generalized mechanism that will support the coupling of different kinds of solvers.

However, a lot of work is still to be done to complete Phase 1. One issue is the support of small-signal AC modeling and simulation, an important requirement for signal processing applications. Another issue is the support of linear electrical primitives and of their corresponding solver(s). Prototypes have already been developed for these features, but it is now required to harmonize these works into a common and consistent framework which has been partially presented in this paper.

7. References

[1] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, L. Todd, *Surviving the SOC Revolution: A Guide to Platform-Based Design*, Kluwer Academic Publishers, 1999.

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

[3] T. Grötker, S. Liao, G. Martin, S. Swan, *System Design with SystemC*, Kluwer Academic Publishers, 2002.

[4] A. Vachoux, C. Grimm, K. Einwich, "SystemC-AMS Requirements, Design Objectives and Rationale", Proc. 2003 Design Automation and Test in Europe (DATE 2003), Munich, Germany, 2003.

[5] K. Einwich, P. Schwarz, C. Grimm, C. Meise, "SystemC-AMS: Rationales, State of the Art and Examples", in SystemC: Methodologies and Applications, W. Müller, W. Rosenstiel, J. Ruf, Eds., Kluwer Academic Publishers, 2003.

[6] C. Grimm, "Modeling and Refinement of Mixed-Signal Systems with SystemC", in *SystemC: Methodologies and Applications*, W. Müller, W. Rosenstiel, J. Ruf, Eds., Kluwer Academic Publishers, 2003.

[7] W. Müller, W. Rosenstiel, J. Ruf, Eds, *SystemC: Methodologies and Applications*, Kluwer Academic Publishers, 2003.

[8] K. Einwich, "SystemC-AMS: Steps towards an implementation", Proc. FDL 2003, Francfort, Germany, 2003.