SystemC-AMS Steps towards an Implementation

Karsten Einwich

Fraunhofer IIS/EAS Dresden

Abstract

In several publications, the extension of SystemC for the design and refinement of Analog and Mixed-Signal Systems were motivated, discussed and first concepts were introduced. On the basic of these publications and the work of the SystemC-AMS study group, this paper will discuss first concepts of a prototype implementation for generalized SystemC-AMS extensions.

1. Introduction and Motivation

SystemC will become increasingly a wide spread methodology for the design and refinement of complex digital hard- and software systems. However, state of the art systems consisting besides large digital hard- and software components of analog blocks and an analog environment. These analog modules must be considered in such early as possible design stages to prevent cost intensive design cycles from lower levels or silicon. Additional a tight impact of software and analog blocks is very crucial. These and other application specific motivations were leading to the development of a couple of proprietary solution for modeling of analog and mixed-signal components within SystemC [2,3,4].

In February 2002 a SystemC-AMS study group was founded to collect the requirements for SystemC-AMS extensions and generalize the existing experiences to define a generic and extendable SystemC-AMS framework. The first outcome was a “White paper” which describes beside the requirements a first structure for SystemC-AMS extensions [5].

On the basic of these concepts, some other publications showed possible examples for the description and refinement of systems from different application domains [11].

This paper will discuss first ideas how the described concepts can be realized by a C++ class library, how these extensions will be fit into SystemC and how those generic extensions can be synchronized with the discrete event SystemC-kernel. The resulting syntax and semantic will be demonstrated by a example motivated from telecommunication application.

2. Brief Overview to the SystemC-AMS Concepts

The focus of SystemC-AMS is on higher abstraction levels - in the specification and system design phases. However, SystemC-AMS must provide paths to lower levels of abstraction. Requirement for these design phases is besides an efficient and flexible system description style a high simulation performance. This simulation performance is required to simulate complete application scenarios within an overall model. To achieve this performance SystemC-AMS must be able to use different application and abstraction optimized algorithms (Models of Computation). Consequently, the system characteristics will be used to speed up the simulation significantly. An example is the use of the over sampled principle of many telecommunication systems. In this case for the simulation of the analog components usually simple and thus fast integration algorithm can be applied [2]. An other important aspect is the encapsulation of models. One objective of SystemC-AMS is to provide a framework for connecting different complex models (which can be protected IP’s also) to an overall system model. Thus it must be possible to reduce the interaction of the different models so far as possible to the communication over the connections (in general analog modules which will be simulated within the same simulator can influence each other also by the step width control or numerical effects like the truncation error calculation). Due to the fact, that a complex system is in general heterogeneous, different description styles and Models of Computation has to be combined within one system. To meet this requirements and objectives a layer structure (fig. 1) on top of the kernel layer of the existing SystemC was suggested. This layer structure consists of a synchronization, solver and view layer. The synchronization layer has to synchronize at one hand the discrete event SystemC-kernel with the analog solvers and at the other hand the analog solvers among each other. The solver layer provides different algorithm for solving a more or less specialized class of problems. The solvers communicate only via the synchronization layer. Thus, the synchronization layer encapsulates the solvers. A solver has a specific interface by which the analog equations can be provided. This interface will be used by the view layer, which provides the user with methods/classes for the system description. This can be methods for the description of analog transfer functions or analog networks. In the case of networks, the view layer has to setup the equation system (e.g. by the Modified Nodal Analysis – MNA) and provide the equations usually in the form of matrices to a solver.

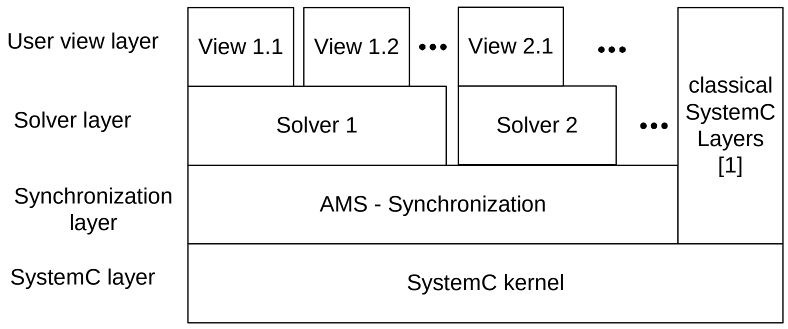

Figure: 1 Layered approach for SystemC-AMS extensions [5].

3. Ideas for an Implementation

In this paragraph, the problems and possible solution for the implementation of the introduced concepts will be discussed. Therefore, the following main problems were figured out:

- Integration into SystemC
- Synchronization Layer
- Solver Layer

Integration into SystemC / View Layer

The SystemC-AMS extensions must be fit with the description style provided by the currently available SystemC. This is especially important for the description of hierarchical models. It must be possible to describe a hierarchical model, which contains sub models of different analog and digital domains.

However, the elaboration and simulation phases for analog modules are quite different. The simulation of SystemC is based on the communication of processes. This approach cannot be used generally for analog simulation. For analog simulation, usually an equation system using module specific information and their connectivity (structural information) has to be setup. In general, this equation system has to be solved globally. In opposite the SystemC kernel performs no structural analysis and the simulation is performed by locally module/process computation and communication / synchronization. An other issue is the time progress in SystemC, which is equal

for all modules. This will be a hard restriction for implementing efficient synchronization and solver algorithms.
Thus, at one hand, the description facilities of SystemC must be re-used but at the other hand the SystemC-AMS, extensions must be properly encapsulated from the SystemC-kernel and a separate much more complicated elaboration phase is required. In this elaboration phase, powerful methods for the system partitioning and structure exploration are essential.
To achieve a homogenous description style, the analog module and interface/channel base classes will be derivate from the corresponding SystemC-classes. The basic elements for structural description in SystemC are: Modules, Ports, Interfaces and Channels. Figure 2 shows the principally derivation of analog extension base classes (prefixed by sca_) from standard SystemC classes (prefixed by sc_ and gray printed). Currently we assume that a sca_ - Module instance is assigned to one solver instance only and is always used only for primitive modules. Hierarchical modules, which can contain sca_ - Modules, are always "classical" SC_MODULE's.

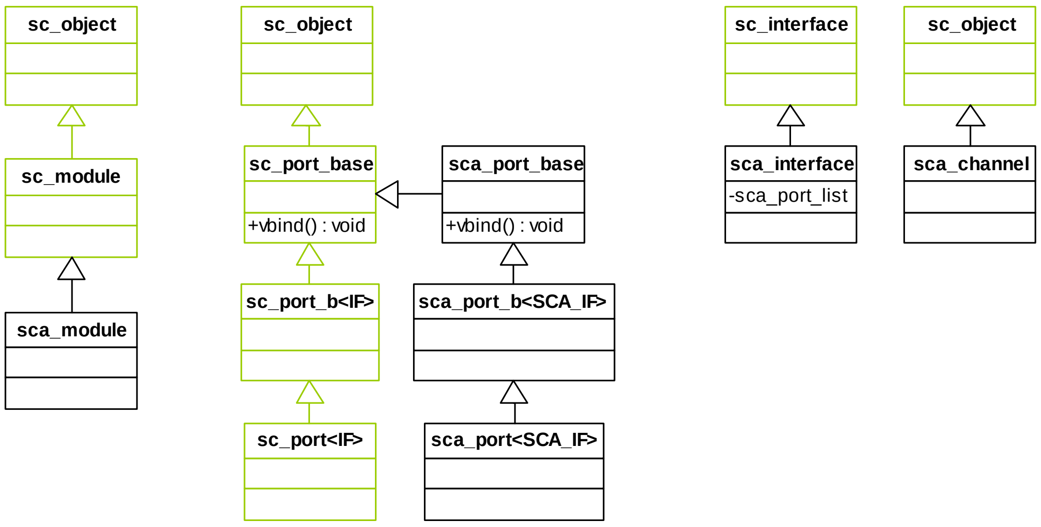

Figure 2: Simplified UML diagrams for Integration of AMS base classes into SystemC

The sca-module base class will be used for the implementation of different view-layers. Thereby we have to distinguish between views, which primitives has to setup an equation system by the use of a solver interface and sca-module primitives which implementing an own independent solver. For the first case, the splitting into interface and channel is not required. The interface/channel represents a connection, which is used to set up the equation system instead of a communication channel that can be refined. However, for compatibility reasons this splitting will be retained. In the second case the communication between modules or solvers is similar to the communication between classical SystemC – modules. Thus, the splitting into interface and channel will be useful, whereby the channel has to be implementing the communication protocol and the interface to the synchronization layer.
For analog domains we established new base classes, a solver interface base class and a solver base class. A solver implements one or more interfaces. The solver interfaces will be used by sca_modules. This interface will provide methods to set up the equation system. Thus, the sca_module classes correspond to the view layer and the solver to the solver layer of figure 1. The two layers communicate via this interface.
To demonstrate the application of this base classes the implementation of a resistor and a capacitor for a linear analog solver will be shown. For the description of those networks the

modules of the basic elements, ports for these modules and hierarchical modules and connectors (wires) including a special wire which signs the reference node (ground) are required. For the simulation of those networks, a solver is required, which implements a solver interface, which is used by the modules of the basic elements.

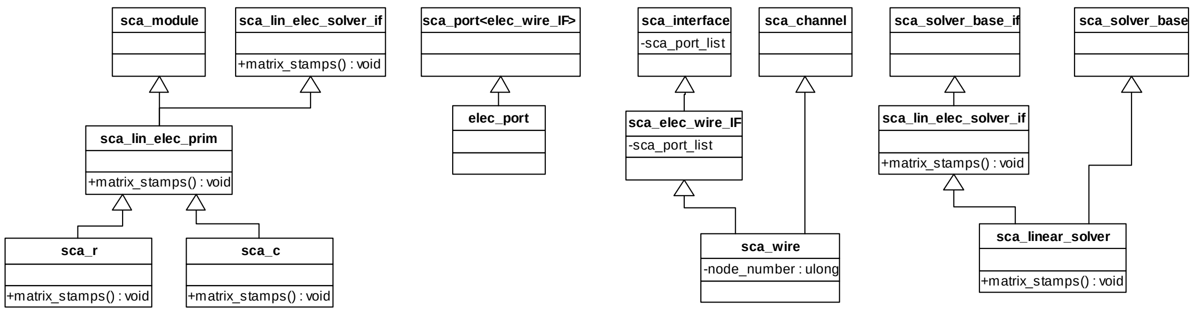

Figure: 3 Simplified UML – diagram for the definition of a linear electrical domain

To set up the equation system of an electrical network consisting of linear elements like resistors, capacitors and inductors usually the Modified Nodal Analysis (MNA) [12] is used. Thereby every element adds so called matrix stamps into global matrices. The position of the matrix stamps of a certain instance depends from their connection – the node numbers. Thus, every element must use the solver interface to provide their specific stamps.
The following code examples showing how the class structure will be used for the definition of a linear electrical primitive domain.

```
//definition of a linear electrical primitive base class
class sca_lin_elec_prim: public sca_module,  sca_lin_elec_solver_if
{
             :
  virtual void matrix_stamps();    //system of equations contributions
             :                         //for a Modified Nodal Analysis (MNA)
  SCA_CTOR}( sca_lin_elec_prim) { …. }
};

//implementation of a resistor
class sca_r : public sca_lin_elec_prim
{
 public:
  elec_port a;
  elec_port b;

  double value ;

  void matrix_stamps()
  {
    sca_a(a,a) +=  1.0/value;  //by operator overloading this realizes
    sca_a(a,b) += -1.0/value; // sca_a(a.get_node_number(),b.get_node_number())
    sca_a(b,a) += -1.0/value; //sca_a is provided by sca_lin_elec_solver_if
    sca_a(b,b) +=  1.0/value;
  }
           :
};
```

```
//use of the defined primitive models inside a hierarchical model
SC_MODULE(rc_net)  //standard SystemC macro
{
   elec_port node1;
   elec_port node2 ;

  //definition of reference node (special wire)
 elec_gnd  gnd;
//definition of internal connector
elec_wire w1;

//definition of used instances
 sca_r *r1, *r2;
 sca_c *c1;

 SC_CTOR(rc_net) //standard SystemC constructor
 {
    r1=new sca_r("r1");
       r1->a(node1);
       r1->b(gnd);
       r1->value=100.0;

  c1=new sca_c("c1");
     c1->a(node1);
     c1->b(w1);
     c1->value=1e-6;
             :
  }
};
```

Synchronization Layer

The synchronization layer must encapsulate the analog solvers, the discrete event SystemC-kernel and the analog solvers among each other. In dependency of the application, different principles must be supported. Thus, constant and variable time step synchronization will be provided. The communication will be always directed. For feasibility reasons a backtracking of a solver or the digital part to a previous time must be prevented.

For an efficient simulation and implementation are loops a crucial problem. To ensure the encapsulation, delay-less loops over solver instances should not allowed. If such a tight interconnection is required, the whole loop has to be modeled inside a solver instance. To determine the order of execution and the next synchronization time point the loop delay must known by the synchronization layer.

In opposite to the connection of network elements the communication between solver instances among each other and with the discrete event SystemC kernel is directed and comparable with the communication between classical SystemC-modules. However, we must differentiate between a communication among analog solvers and with the SystemC-kernel.

A communication with the SystemC-kernel is event driven. Thus, the signal value will change (jump) at discrete time points. In dependency of the analog solver, signals to the analog domain will be sampled or they lead to a re-initialization of the analog solver. Generally it will be a task of the analog solver how it handles changes at discrete event inports.

For signals from the analog domain to the discrete event SystemC – kernel a kind of threshold or sampling mechanism is required to restrict the number of events.

More complicated is a general solution for the communication among analog solver instances. Analog signals are time and value continuous, however for a numerical simulation they are calculated at discrete time points and the value pattern among this time points is usually assumed

as linear. In general, the time distances of the calculation points are different for all solver instances and they can change dynamically. Thus, a mechanism is needed which provides the solver inputs with the appropriate value of an arbitrary time point. Therefore, usually linear interpolation is used.
If we consider a loop-free connection of analog solver instances, the synchronization will be done by consecutive calls of the solvers, following the directed graph. Thereby always, a time interval (which should be so large as possible) will be calculated. Thereby the data items, which will be exchanged, are generally describing a wave as an arbitrary number of time value pairs.
Much more critical is the case of a non-loop free connection of solver instances. In this case, a kind of delay in the loop back is required to prevent iterations and thus re-setting of solver instances. A kind of delay means, that at a solver instance output a time interval in the future compared to the input time is available. In this case, the solver scheduling can be done similar to the loop – free case.
Currently we assume that the implementation of a delay-less loop over solver instances makes especially of performance and feasibility reasons no sense. Such loops have to be encapsulated into one solver instance. There they can be solved by the corresponding solver by setting up the equation system.
An open problem is the detection of the loop – delay by the synchronization layer. Currently we use a manual assignment.
Using this restriction, for the synchronization of analog solvers a kind of dataflow scheduling is required.
Presently we prefer one synchronization object, which provides a restricted number of methods to setup the structure and to perform the synchronization.
For the first prototype solution and experiments, we support constant time step synchronization only. In this simple case the view layer has to provide the synchronization layer with information about the time distances of the solver in- and outputs and if a loop of solvers with the loop delay. In this case, the solvers can be scheduled by the static dataflow model of computation, which leads to a high simulation performance, due the scheduling can be done once before simulation.

Solver Layer

A solver computes an analog equation system. Therefore, it will realize a more or less application specific (and thus efficient) algorithm. In this context, a solver can be a complex nonlinear DAE-solver or in the simplest case, some sequential C-statements that calculate output values may in dependency of the time, input values and states. In this case the solver is not a separate object, instead they is implemented in the model.
Generally, the solver needs at one hand an interface for providing the equation by the modules and at the other hand it must be able to provide the interface to the synchronization layer with the required information to manage the time control and the data exchange.
Generally, it must be possible to restrict the simulation interval to each synchronization time step. A backtracking over synchronization time points is not allowed.
In the simple case of constant time step synchronization, each solver has to calculate always a constant time interval. The solver reads inputs and writes the calculated outputs at equidistant time points.

4. Demonstration of the use of the introduced implementation ideas

In this section, a static dataflow oriented description style will be defined. This description style is very efficient for the simulation of signal processing oriented applications and is influenced by tools like COSSAP and SPW. In those applications, the sampling rate is usually much higher than the system time constants. In such a case, the application of constant time step solvers is very efficient. Additional the differential equations are mostly linear, which allows the application of very simple and thus fast integration algorithm [2].

```
//definition of a static dataflow base class which provides the methods for dataflow calculations and
//methods for the calculation of embedded analog functions like transfer functions
class sdf_module : public sca_module,
                              sca_synchronization,   //provides access to the synchronization
                              sca_linear_behavior    //provides methods for transfer function, state
                                                      //space systems, …

{

  virtual void init() {};            // initialization method
  virtual void sig_proc();           // simulation method
  virtual void post_proc() {};  // post processing method

   sdf_module(sc_module_name nm)
   {
   //registration of data-flow methods to simple synchronization layer
   //provided by sca_synchronization
     SCA_SDF_INIT(init);                    //initialization method
     SCA_SDF_RUN(sig_proc);             //signal processing method
     SCA_SDF_POST(post_proc);          //post processing methods
   }
};

//definition of ports for static dataflow modules
typedef sca_port<T,sca_sdf_synchronization_in_if<T> > sdf_in<T>;
typedef sca_port<T,sca_sdf_synchronization_out_if<T> > sdf_out<T>;

//ports for synchronizing with standard SystemC - signals
typedef sca_port<T,sc_signal_in_if<T> >   sc2sdf_in<T> ;
typedef sca_port<T,sc_signal_out_if<T> > sdf2sc_out<T> ;

//the interfaces sca_sdf_synchronization_in_if  and sca_sdf_synchronization_in_if will be implemented
//by the channel sdf_signal<T>, this channel will provide the synchronization with the required sample-
//times and will establish the communication between the modules

//example for the use of the defined dataflow description style
struct low_pass: sdf_module
{
 //ports
 sdf_in<double>    in;  //port declaration
 sdf_out<double> out;

sc2sdf<bool>      gain_6dB; //event driven control signal

 //parameter for cut-off frequency
 double fc;

 void init() {
                  B[0]=1.0;                  //Coefficients of transfer function
                 A[0]=1.0;
                 A[1]=1.0/(2.0*M_PI*fc)
```

```
               }
 void sig_proc()  {
                          double gain;
                          if(gain_6dB.read()) gain=2.0;
                         else                        gain=1.0;

                         out= gain * sca_ltf(A,B,S,id,in);    //transfer function method provided
                       }                                               //by sca_linear_behavior

  private:
            sca_vector        A, B, S;     //vector class holds coefficients and states
           sca_dae_id        id;            //signs system of equations
    };                                        //if more than one used per module
```

//those dataflow modules can be connected with above described networks by the definition of network //elements with dataflow in-/outports like a voltage source which is driven by a dataflow signal

```
class sca_vsdf : public sca_lin_elec_prim
{
  public:
   elec_port a;                       //electrical ports
   elec_port b;

  sdf_in<double> voltage;   //dataflow inport

   double value ;

   double v_t()
   {
     return voltage.read();
   }

  long opt_eq ;  //number for additional equation (required for voltage source by MNA)

   void matrix_stamps()  //matrix stamps of voltage source
   {
     opt_eq=sca_get_add_eq();

     sca_a(a,opt_eq)  =   1.0;
     sca_a(b,opt_eq)  = -1.0;
     sca_a(opt_eq,a)  =  1.0;
     sca_a(opt_eq,b)  =  -1.0;

    sca_q(opt_eq)     = v_t;  //assign method which will be called by the solver
   }
             :
};
```

5. Conclusion

The paper presented first ideas for an implementation of a generalized and generic SystemC-AMS extension. An efficient implementation seems to be feasible. The complexity is manageable by a good encapsulation of the solvers by simple and restricted synchronization schemes. The existing SystemC has not been changed. However, it has been evaluated, that all used interfaces are defined in the Language Reference Manual available for a short time. For the definition of a generalized and applicable solver - synchronization further work has to be done.

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 Circuts 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
11. Ch. Grimm: Modeling and Refinement of Mixed Signal Systems with SystemC. In SystemC: Methods and Applications. Kluwer Academic Publisher (KAP), April 2003.
12. Ho, C.W, Ruehli, A.E., Brennau, P.A., „The Modified Nodal Approach to Network Analysis" IEEE Transactions on Circuits and Systems CAS22 (1975) June, 504-509