A SystemC AMS Extension for the Simulation of Non-linear Circuits

Thomas Uhle, Karsten Einwich
Fraunhofer Institute for Integrated Circuits IIS
Design Automation Division
Zeunerstr. 38, 01069 Dresden, Germany

Abstract
This paper proposes a new model of computation for SystemCAMS with similar features like in VHDL-AMS or Verilog-A/AMS to enable the modelling of analogue, non-linear parts of cyber-physical systems. It integrates smoothly into the current architecture of SystemC/ SystemCAMS and, thus, allows joint simulations along with other SystemC methodologies like TLM for abstract modelling of digital hardware/software systems. Finally, it is illustrated that our implementation of the proposed extension is already in use for some industrial examples.

1. Introduction
New technologies permit the integration of more and more features on one or more silicon dies. At the same time integrated systems have to become more flexible to cover a wider range of use cases because of the heavily increasing design and production set-up costs.

Our world and how we interact with it is analogue. More and more highly integrated systems contain analogue components—ITRS estimated that 80% of today’s Systems on Chip (SoCs) are mixed-signal chips [9]. However, analogue circuits cannot be scaled as easily to the smaller lithographic dimensions of newer technologies as digital circuits. And it is even more difficult to achieve the kind of configurability needed for different application requirements. Because of those obstacles, the number and complexity of analogue components are being reduced. The remaining analogue parts are then “assisted” by digital components as much as possible. In this way, the performance and configurability of a typical analogue front-end is increased through a tight interaction with a digital regulation and control subsystem. Sophisticated signal processing is applied to compensate the analogue imperfections. Nowadays, we talk more and more about “digital assisted analogue” [17].

A similar trend can be observed in overall systems. In automotive applications for example, more and more mechanics is being either substituted or assisted by electronics. Consider the brush-less motor: its mechanics are simple and inexpensive, but it requires therefore tightly coupled sophisticated electronics. The consequence of these design trends is that it is getting ever harder, in some cases even impossible, to consider the analogue components independent of the digital hardware and software or to design any integrated circuit and software algorithm without detailed knowledge of the full system environment. This is why executable overall system-level models have become design-essential for an increasing number of SoCs.

SystemC is a C++ based hardware description language with the focus on system architecture design for large systems. In particular, it supports methodologies that allow the description of the interaction of hard- and software as well as models at high levels of abstraction to facilitate high-performance system-level simulations.

Bearing the aforementioned trends in mind, it seems to be reasonable that modelling and simulation of analogue circuits, also at higher levels of abstraction, rapidly gain importance. So SystemC is being extended to the analogue domain. Following the SystemC philosophy, the SystemCAMS extensions focus on abstract modelling to permit overall system-level simulations of “real-time” application scenarios [11]. Therefore, the current SystemCAMS standard [12] includes means for data-flow and linear network modelling. Although restricted, these models of computation (MoCs) provide enough facilities to support abstract descriptions for a wide range of applications, especially for communication systems. And so SystemCAMS achieves the necessary simulation performance [3]. The restrictions implied by the MoCs of the SystemCAMS standard usually do not limit system-level modelling. However, especially for those applications, where the interaction between front- or back-end stages and the system environment is very tight (e.g. driver stages with pulse width modulation), the existing MoCs do not facilitate suitable system models with the required accuracy. We are facing those applications in particular in the automotive domain. One example may be an airbag system where the squib driver is part of a non-linear control loop with a squib equivalent circuit.

Anyway, those parts of such systems that need highly detailed model descriptions are very small. Usually, there are just some devices at the front- or back-end of a circuit. Moreover, the internal complex functionality can be modelled and simulated very efficiently by the existing Sys-
temC/ SystemC AMS MoCs and methodologies.

An imaginable solution might be to model the whole system with languages like VHDL-AMS or Verilog-A/AMS. However, those languages do not support modeling facilities at higher levels of abstraction like data-flow or transaction level modelling (TLM) [13]. As discussed before, these facilities are required to achieve the necessary simulation performance. The speed-up that can be achieved by abstract SystemC/ SystemC AMS based modelling techniques has been reported to be in the region of 10,000 to 100,000 compared to VHDL/ VHDL-AMS behavioural models [16] if these techniques are suitable for the considered application and its model’s use cases. Other techniques like the utilisation of specialised algorithms [5, 10] or model generation [7] are restricted to only a few areas of application.

In this paper, we propose an additional model of computation for SystemC AMS that provides capabilities similar to those of languages like VHDL-AMS and Verilog-A/AMS. This extension of SystemC AMS enables the modelling of non-linear networks and is natively embedded in SystemC/ SystemC AMS. So all features of these languages are retained and a joint simulation along with the already existing MoCs is ensured. The presented approach is optimised to handle highly detailed models of small analogue parts of a complex mixed-signal hardware/software system that are coupled with models of parts of this system at higher levels of abstraction like data-flow or TLM. This approach looks out for the sizes of equation systems by exploiting system and model properties. For instance, multiple solvers are instantiated wherever applicable. Hence, each solver instance deals with a rather small non-linear equation system and can deliberately determine its step sizes independent of other solver instances whenever possible. That means that small time constants in one component may have no influence on the step sizes of solvers of other components with large time constants. This way, different parts of the system are encapsulated and we are able to observe only a moderate slow-down in simulation performance.

Anyway, it is not our intent to supersede VHDL-AMS or Verilog-A/AMS simulations because our non-linear solver cannot be faster than other simulators when handling a system with only one large analogue subsystem.

II. ESSENTIAL NEW LANGUAGE FEATURES

The SystemC AMS standard defines a set of pre-defined MoCs. But the language architecture of SystemC AMS has also been defined in an extensible way. So there are base types for signals, ports and modules for instance. The different MoCs are each assigned to a distinct namespace. The namespace of the timed data-flow MoC is sca_tdf for instance. The types or classes within these namespaces extend the base types of SystemC AMS with features specific to the corresponding MoC. The proposed extension of SystemC AMS for non-linear networks follows the same rules.

It is common knowledge that real circuits can be described by Kirchhoff networks or multipole networks for instance in order to simulate their behaviour. Regarding the latter approach, a multipole network is an interconnection of various \( n \)-poles, which are multipoles with a set of \( n \) terminals and a constitutive relation that describes the terminal behaviour. These \( n \)-poles represent circuit elements. The interconnections of these \( n \)-poles can be specified by identification of their terminals by means of an equivalence relation. The representatives of its equivalence classes are called nodes [20].

Models of circuit elements are encapsulated in SystemC modules. In SystemC it is differentiated between two kinds of modules: primitive modules (basic models, so-called elementary \( n \)-poles) and hierarchical modules (macro models). Macro models are made up of interconnected basic models.

The entity of each module in SystemC is described by a SC_MODULE class declaration with a port and terminal list. An interconnection of some \( n \)-pole modules is realised by binding their terminals to nodes. In order to utilise the built-in binding mechanism of SystemC, terminals have to be instances of a type derived from sca_core::sca_port and nodes instances of a type derived from sca_core::sca_interface.

All new language constructs populate the namespace sca_nln unless specifically stated otherwise. Please keep in mind that the statement

```cpp
using namespace sca_nln;
```

is always implied for the following examples of this paper without any further notice.

A. Model Entity

The capabilities of SystemC AMS are enriched by new language constructs quite similar to the ones for electrical linear networks (ELN MoC). In contrast to the latter, it is intended not to model only electrical networks but also networks of other physical domains (natures). Therefore, the new classes are template classes with an appropriate nature as template argument (e.g. electrical, or kinematic). This way, the template classes may inherit different methods to access the across value for instance.

Instances of class sca_terminal<nature> are terminals of \( n \)-pole modules, whereas instances of class sca_node<nature> are nodes and instances of class sca_node_ref<nature> reference nodes respectively.

Identifying terminals and nodes via SystemC’s binding mechanism, their natures have to match pairwise, for example:


```cpp
sca_node<electrical> n1;
sca_node_ref<electrical> gnd;
sca_terminal<electrical> t1;

my_module mod1;
mod1.b(t1);
mod1.c(n1);
mod1.e(gnd);
```

As can be seen in this example the same holds for parent port binding, namely when a terminal of the parent module (here: t1) is to be bound to a terminal of a child module (here: mod1.b).

Macro models, that means hierarchical modules, are simply classes derived from `sc_core::sc_module` as usual in SystemC.

### B. Elementary n-poles

Basic models, that means primitive modules, are classes derived from `sca_module` and, thus, inherit most notably the callback methods `initialize()` and `equations()`. The macro `SCA_NLN_MODULE` is provided for simple module declarations.

These basic models may have an internal structure (skeleton) which designers can define by using internal nodes and branches. The classes `sca_internal_node<nature>` and `sca_branch<nature>` are provided for their declarations. Head and tail of a branch may point to any terminal or internal node as long as their nature match. Here is an example of a branch declaration:

```cpp
sca_internal_node<electrical> n1;
sca_terminal<electrical> t1;
sca_branch<electrical> br_1(t1, n1);
```

In this example, the branch `br_1` is directed from terminal `t1` to node `n1`.

### C. Formulation of Model Equations

The model equations shall describe the terminal behaviour [18, 19] of the corresponding model. It is much appropriate to set up model equations with respect to the rules of a modified nodal analysis (MNA) [6]. This way, one has the freedom to incorporate also implicit equations. Such an equation system has the form

\[
i = g(t, u, \dot{u}, x, \dot{x}), \quad (1)
\]

\[
0 = h(t, u, \dot{u}, x, \dot{x}), \quad (2)
\]

where \(t\) denotes the time of value, \(i\) the vector of terminal currents, \(u\) the vector of terminal voltages and \(x\) the vector of additional variables (so-called internal variables, e.g. branch currents, charges, fluxes, etc.; also for substitutions to fulfill equation 1). In practise, no restrictions on the formulation of model equations are implied. If they cannot be expressed in the form of an admittance description, then equation 2 can be seen as a generalised Belevitch form [1].

Using a globally defined virtual node for the description of the terminal behaviour of a model is discouraged. Instead the designer shall extend the model’s entity with an additional terminal.

After applying Kirchhoff’s current law to each node of the network except the reference nodes, the resulting equation system generally consists of implicit differential and algebraic equations (DAE)

\[
0 = f(t, u, \dot{u}, x, \dot{x}),
\]

which can be discretised by implicit integration methods (e.g. linear multistep methods) and then be solved by nonlinear solvers [15].

Additional network variables (components of vector \(x\)) can be declared by using class `sca_variable` to which a `quantity`, that means an instance of class `sca_quantity_type`, may be assigned. There are already lots of quantities pre-defined. Here are some examples of variable declarations:

```cpp
sca_variable x1;
sca_variable q2(electrical::charge);
sca_variable f3(kinematic::force);
```

The value of such a variable can be accessed by the method `value()` and its derivative by method `dv()`. In addition, the value of a variable is also returned by `operator double()` for the designers’ convenience.

The across value of `sca_terminal<electrical>` or `sca_branch<electrical>` and its derivative can be accessed by the methods `v()` and `dv_dt()` respectively, and its through value by the method `i()`. With regard to `sca_terminal<kinematic>` or `sca_branch<kinematic>` the across value (the position) and its derivative (the velocity) are accessed by the methods `x()` and `vx()` respectively, and the through value (the force) is accessed by the method `f()`. Note the different semantics of the method `v()` within the two different contexts. This way, model equations in source code look quite similar to handwritten equations, which may prevent the designers from typing errors.

As SystemC and so SystemC AMS as well are just class libraries written in C++, models have to be compiled by a C++ compiler. The drawback of this approach is the lack of information which another simulator might get by parsing the model descriptions itself. That is why the designers of SystemC AMS models have to give additional hints within the source code. An example:

A branch, whose constitutive relation matches a voltage source, can be labelled with the attribute `vsource`. Hence, one of the branch’s nodes is controlled by the other one and the impressed voltage waveform. Then, the controlled node can be eliminated from the combined vector of network variables \((u, x)\).
D. Examples

The new language constructs introduced in the last sections are a first starting point to begin writing models for elementary 

1. Cascaded Voltage Sources:

For our first example, imagine a tree of voltage sources. The subcircuit in figure 1 serves as a good example for the following considerations.

A reference node can be arbitrarily picked (e.g. n0). Then the impressed voltage across source \( v_1 \) determines the voltage value at node n1 with respect to the reference node n0. Furthermore, the voltage values at nodes n2, ..., n4 are fixed by the impressed voltages across the other voltage sources in turn. Thereby, it does not matter whether the impressed voltages are time-varying or constant. The effect is that all nodes n1, ..., n4 are controlled by node n0 which results in the fact that these nodal voltages are no longer unknowns of the network equation system. The corresponding nodal equations are eliminated and, thus, affect the nodal equation of node n0, too. In general terms, Kirchhoff’s current law is also applicable to arbitrary cut-sets of the graph of a network and in particular to the cut-set resulting from the tree of voltage sources,

\[
i_1 + \cdots + i_4 = 0. \tag{4}\]

This equation replaces the original nodal equation of node n0. The following listing exemplarily shows the model description of one of these four voltage sources:

```
SCA_NLN_MODULE(vsrc_const)
{
  sca_terminal<electrical> p, n;
  sca_branch<electrical> br;
  double voltage_value;
  vsrc_const( sc_module_name nm,
               double value = 1.0 ) :
    br(p, n), voltage_value(value)
  {  
    sc_attribute<string> attr =
      new sc_attribute<string>("vsource");
    attr->value = "constant";
    br.add_attribute(*attr);
  }
  void equations()
  {  
    br.v() == voltage_value;
  }  
  [...]  
};
```

2. Simple DC Motor:

The following example of a DC motor’s model (cf. figure 2) demonstrates what a model combining two different natures looks like. Two internal network variables, the motor current \( i_{\text{coil}} \) and the coil’s magnetic flux \( \phi \), and corresponding additional equations are defined. The additional equations are introduced by the macro \texttt{EQN(var)} \(< \ldots \text{...} \)
III. SystemC / SystemC AMS SIMULATION CYCLE

It is already indicated above in this paper that multiple solvers are instantiated where it is applicable. In the case of several analogue components being separated by intermediate digital components, the solvers assigned to these analogue parts are partitioned into different clusters of the system. Each cluster is managed by its own SystemC process that is sensitive to events of the input signals. Thus, a tight synchronisation between digital and analogue parts of a cyber-physical system can be established.

The synchronisation method of SystemC AMS ensures that the analogue solvers run ahead of the SystemC scheduler. The maximum step sizes of the solvers are limited by the next incoming events. Each solver may stop its integration method at threshold crossings of signals that generate digital events and reports the time of the event to the synchronisation method. In this case, other solvers may have to backtrack if their inputs are affected by this new event. In the case that there are no occurrences of intermediate events, the last time steps of the solvers are accepted. If only network models and discrete-event models are coupled, the synchronisation method is similar to the VHDL-AMS simulation cycle. In the case of additional timed data-flow models the synchronisation becomes way more complex [paper in preparation].

IV. CIRCUIT EXAMPLE

A really complex example of a cyber-physical system is a car’s window lifter. Parts of its model are shown in figure 3. The top level of the system can be seen in figure 3(a). Next to digital and data-flow models the subsystem WINDOW LIFTER CONTROL also contains a model of a microcontroller. There are two models of it at different levels of abstraction. Variant 1 is a bus-cycle-accurate behavioural model of the microcontroller with the real software algorithms running on it. Variant 2 is a TLM based algorithmic model. The model of the non-linear mechanical subsystem is presented in more detail in figure 3(b). Mainly, it consists of the motor, its driving circuit, the gear, the window pane and the limiters of the window frame.

In figure 4 the results of a simulation of the window lifter system are shown. The upper waveform illustrates the position of the upper edge of the window pane during the simulation. It can be seen that this upper edge hits an obstacle after about eight seconds. The software algorithms recognise it and the window pane is immediately lowered. After removing this obstacle, the window
is closed until the pane reaches the upper limitation of the window frame. The waveform of the motor shaft’s
torque is shown in the middle section. Finally, the lower
waveform shows the course of the corresponding motor
current which is monitored by the software algorithms to
detect obstacles. It can be observed that the current ex-
ceeds a threshold when the obstacle is hit.

It took our simulator about 230 min on a Linux system
with 2.1 GHz Opteron CPU to simulate this scenario with
variant 1 of the microcontroller for 15 sec of real-time. But
the simulation of the same scenario with variant 2 took
only 58 sec. Hence, it can be reasoned that the analogue
solver for the small non-linear parts does not dominate the
overall performance of the simulation.

V. CONCLUDING REMARKS

In this paper we proposed an extension of SystemC AMS
for the simulation of non-linear networks that integrates
smoothly into the existing language architecture. This ex-
tension was prototypically implemented and simulation re-
sults were presented.

We are encouraged by the results obtained so far to go
on developing a simulation platform that handles those
large complex cyber-physical systems at higher levels of
abstraction. In the next steps we would like to provide
block-oriented networks for taking advantage of the in-
herent, repetitive modular structure of a system for the
computational efficiency during simulation. The result-
ing equation systems of those networks can be solved by
means of simulation techniques based on non-linear
Gauß-Seidel and multi-level Newton iterations [4, 8].

REFERENCES

Francisco 1968.
[2] Leon O. Chua, Charles A. Desoer, Ernest S. Kuh: Linear and
the Specification of a Complex Wired Telecommunication System.
Proc. of the Forum on Design Languages (FDL05), pp. 49–60,
Lausanne 2005.
Analysis of Large-Scale Nonlinear Networks. Proc. of Euro-
[5] Christoph Grimm, Peter Oehler, Christian Meise, Klaus Wald-
schmidt, Wolfgang Fey: AnalogSL: A Library for Modeling Analog
Power Drivers with C++. Proc. of the Forum on Design Languages
(FDL01), Lyon 2001.
Nodal Approach to Network Analysis. IEEE Trans. on Circuits and
[7] Stefan Hölldampf, Daniel Zaum, Markus Olbrich, Erich Barke, Ing-
mar Neumann, Sebastian Schmidt: Methodologies for High-Level
Modelling and Evaluation in the Automotive Domain. Proc. of the
Forum on Design Languages (FDL08), pp. 73–77, Stuttgart 2008.
Method Using System Dynamic Behavior. Circuits, Systems, and
modelling platform for digital and mixed-signal hardware/soft-
ware co-design. Proc. of the Forum on Design Languages
duction to Modeling Embedded Analog/Mixed-Signal Systems
using SystemC AMS Extensions. 7th Symposium on Electronic
SystemCAMS Extensions Language Reference Manual. 2010,
http://www.systemc.org/downloads/standards/amsl01/
http://www.systemc.org/downloads/standards/tlm20/
[14] François Pécheux, Christophe Lallement, Alain Vachoux:
VHDL-AMS and Verilog-AMS as alternative hardware description
languages for efficient modeling of multidiscipline systems. IEEE
Trans. on Computer-Aided Design of Integrated Circuits and
[15] Lawrence T. Pillage, Ronald A. Rohrer, Chandramouli Viswes-
[16] Monica Raffaia, Christian Decker, Christoph Grimm, Karsten
Einwich, Thomas Markwirth, Georg Pelz: New Methods for
System-level Verification using SystemCAMS Extensions: Ap-
plication to an Automotive ECU. Proc. of the 12th Workshop “Methoden
der Beschreibungssprachen zur Modellierung und Verifikation von
Keynote 45th Design Automation Conference (DAC’08), Anaheim
Proc. of European Conference on Circuit Theory and Design
[19] Albrecht Reibiger: Terminal Behaviour of Networks, Multipoles
and Multports. Proc. of 4th Vienna International Conference on
[20] Albrecht Reibiger: Foundations of Network Theory. Proc. of Inter-
national Symposium on Theoretical Electrical Engineering
(ISTET’09), Lübeck 2009.