SystemC-AMS Simulation of Conservative Behavioral Descriptions

SystemC-AMS simulation of conservative behavioral descriptions

Sara Vinco, Michele Lora, Mark Zwolinski

Abstract SystemC has recently been extended with the Analog and Mixed Signal (AMS) library, with the ultimate goal of providing simulation support for analog electronics and continuous time behavior. SystemC-AMS allows modeling of systems that are either conservative and low level or continuous time and behavioral, which is a limited range compared to other AMS HDLs. This work addresses this challenge by extending SystemC-AMS support to a new level of abstraction, namely *Analog Behavioral Modeling* (ABM), to cover models that are both behavioral and conservative. This leads to a methodology that uses SystemC-AMS constructs in a novel way. Full automation of the methodology allows proof of its effectiveness both in terms of accuracy and simulation performance, by applying the overall approach to a complex industrial Micro Electro-Mechanical System (MEMS) case study. The effectiveness of the proposed approach is further highlighted in the context of virtual platforms for smart systems, and adopting a C++-based language for MEMS simulation reduces the simulation time by about 2x, thus enhancing the design and integration flow.

1 Introduction

SystemC has long been considered the reference language for electronic system-level design, as it supports both hardware and software and the integration of multiple levels of abstraction, including Register Transfer Level (RTL) and transactional level [1]. However, the increasing presence, in embedded systems, of analog components and Micro Electro-Mechanical Systems (MEMS) limits the generality of

Sara Vinco
Politecnico di Torino, Turin, Italy e-mail: sara.vinco@polito.it

Michele Lora
Università di Verona, Verona, Italy e-mail: michele.lora@univr.it

Mark Zwolinski
University of Southampton, Southampton, United Kingdom e-mail: mz@ecs.soton.ac.uk

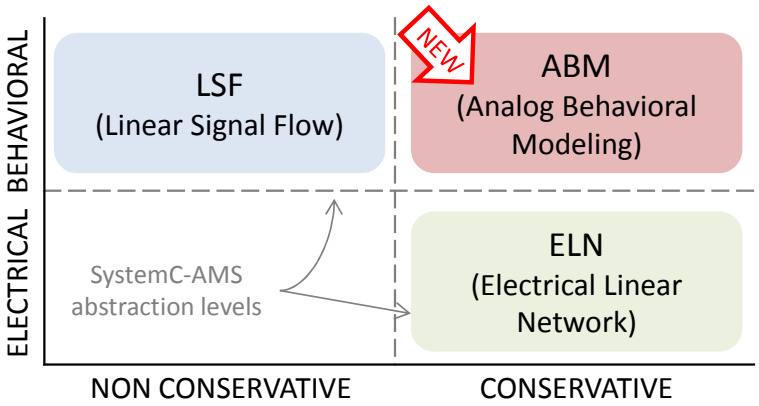

Fig. 1 Design space covered by the proposed approach *w.r.t.* SystemC-AMS. The methodology bridges the gap between LSF and ELN, by targeting behavioral and conservative descriptions (*ABM*).

SystemC [2]. Indeed, these types of component require the support of continuous time and conservative behaviors, which cannot be modeled with a discrete event simulator.

In response to this, Accellera has standardized the SystemC-AMS extension [3]. SystemC-AMS provides a number of predefined levels of abstraction that reproduce linear continuous time models with different degrees of accuracy and adherence to physical behaviors. Unfortunately, such abstraction levels (briefly outlined in Figure 1) do not model all types of analog models. The *Linear Signal Flow* (LSF) level of abstraction focuses on behavioral, continuous time systems, but it does not support the modeling of conservative systems. On the other hand, *Electrical Linear Network* (ELN) is conservative, but it does not support behavioral models. Furthermore, SystemC-AMS does not yet support non-linear modeling [9] and it can not therefore be considered a replacement for SPICE [4] or Verilog-AMS [5].

The resulting gap between ELN and LSF thus misses descriptions that are both behavioral and conservative, and that are commonly used for the modeling of MEMS and other analog components [6, 7]. The limited flexibility of SystemC-AMS forces designers to adopt other AMS HDLs (*e.g.*, Verilog-AMS) for modeling these kinds of components, thus reducing the applicability of SystemC-AMS.

The key idea of this work is to bridge the gap between LSF and ELN, to represent models that are both behavioral and conservative in SystemC-AMS. This new level of abstraction, called *Analog Behavioral Modeling* (ABM), is demonstrated with a methodology that exploits existing SystemC-AMS constructs. The goal is to show that SystemC, with its AMS extensions, can be used as a general embedded system modeling and simulation framework even in the presence of analog circuitry and MEMS. It is important to note that this work does not define new SystemC-AMS libraries, but it rather uses ELN primitives in an innovative way.

The main contributions of this work are:

- *identification of the ABM level of abstraction*, necessary for overall embedded system simulation in a SystemC-based environment;
- definition of a *sound methodology* for modeling ABM components in SystemC-AMS, by using existing primitives in a novel way;
- *validation of the ABM level against Verilog-AMS*, to show that those Verilog-AMS descriptions that do not fall into the ELN and LSF domains can now be correctly represented in SystemC-AMS;
- *automation of the proposed methodology* by automatically converting Verilog-AMS models into ABM SystemC-AMS models. This simplifies the application of the methodology to complex industrial case studies.

As a side effect of the proposed methodology, Verilog-AMS models can be automatically converted into SystemC-AMS code for easy integration into a virtual platform including analog models. This avoids the use of CPU intensive co-simulation frameworks, thus noticeably speeding up the simulation of a virtual platform.

The paper is organized as follows. Section 2 provides the necessary background on Verilog-AMS and SystemC-AMS. Sections 3 and 4 focus on the proposed methodology. Finally, Section 5 applies the proposed approach to an industrial case study and Section 6 draws some conclusions.

2 Background

This Section provides the necessary background on the adopted languages for analog and mixed signal modeling: Verilog-AMS (Section 2.1) and SystemC-AMS (Section 2.1).

2.1 Verilog-AMS

Verilog-AMS is one of the most widely used languages for analog and continuous time modeling [5]. The system solver is essentially the same as that used in SPICE [4]. A circuit is modeled in terms of an abstract graph of nodes (that can also be used for external connectivity) connected by branches [8]. The system state is defined in terms of voltages (`V()`) and currents (`I()`) associated with nodes and branches. The numerical values of potential differences and currents can be used in expressions with the access functions `V()` and `I()`. Relationships between nodes are modeled with differential algebraic equations (DAEs), written as *simultaneous statements*. The *contribution operator* `<+` models a simultaneous statement summing multiple contributions to the branch current (or voltage) as a function of other branch voltages and currents.

Conservative modeling is imposed by the requirement that the sum of currents leaving any node must be equal to zero at any time (thus reflecting Kirchhoff's Current Law). This condition is managed by the internal solver of the Verilog-AMS simulator, and thus must not be explicitly modeled by the designer.

The simulator internal solver uses the simultaneous statements and conservative conditions to build a system matrix. Numerical integration methods are used to solve the system of DAEs. Continuous time is modeled as a sequence of discrete time points, such that the time step is optimized to minimize errors while maximizing efficiency.

Non-linear equations are solved iteratively at each discrete time step to determine the system state over time. Typically, the Newton-Raphson method [10] is used for linearization

2.2 SystemC-AMS

SystemC-AMS is the extension of the SystemC framework for modeling analog and mixed-signal systems [3]. Its role is to provide a higher level view of mixed-signal and analog systems, to allow early simulation and validation of the overall system. For this reason, SystemC-AMS supports only linear and time-invariant descriptions, and is incapable of solving non-linear functions [9].

To cover a wide variety of domains, SystemC-AMS provides three different abstraction levels, supporting different communication styles and representations with respect to the physical domain:

- *Timed Data-Flow* (TDF) models are scheduled statically by considering their producer-consumer dependencies in the discrete time domain;
- *Linear Signal Flow* (LSF) supports the modeling of continuous time through a library of pre-defined primitive modules (*e.g.*, integration, delay), each associated with a linear equation;
- *Electrical Linear Network* (ELN) level models electrical networks through the instantiation of predefined primitives, *e.g.*, resistors or capacitors, where each primitive is associated with electrical equations.

A *SystemC-AMS internal solver* analyses the ELN and LSF components to derive the equations modeling system behavior, that are solved to determine the system state at any simulation time.

The main difference between LSF and ELN is in the adherence to physical laws. LSF is *non-conservative* and it expresses behaviors as directed flows of continuous-time signals or quantities. On the other hand, ELN is *conservative*, *i.e.*, the derived set of equations is extended by the internal solver to satisfy the conservation laws (Kirchhoff's laws).

3 Methodology overview

The goal of the proposed approach is to prove that conservative and behavioral descriptions can be modeled in SystemC-AMS. To this extent, the starting point of the methodology is a Verilog-AMS behavioral description, made up of a set of

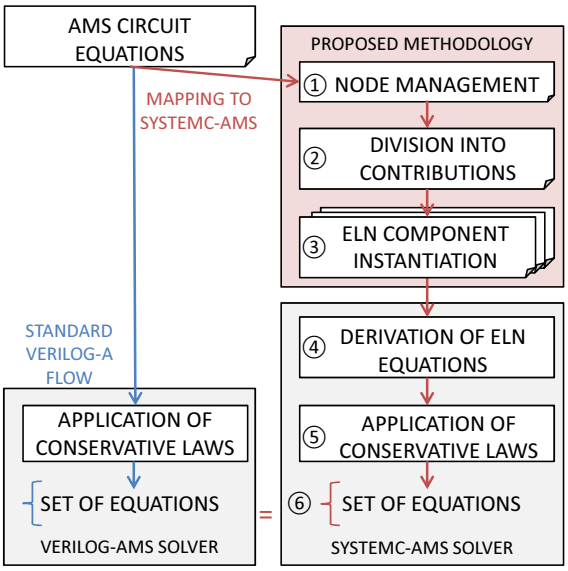

Fig. 2 Overview of the proposed methodology.

simultaneous statements that assemble voltages or currents to describe the state of the electrical circuit nodes. Due to the limitations of SystemC-AMS, the models are strictly linear and time-invariant.

The standard Verilog-AMS simulation flow is depicted on the left-hand side of Figure 2. The Verilog-AMS internal solver takes the simultaneous statements as the input and derives both the user defined equations and the conservative ones. The resulting equation set is used to build the numerical matrices that determine the system state.

The methodology to convert the Verilog-AMS code into SystemC-AMS is based on reproducing the final equation set in the SystemC-AMS environment, through the flow depicted on the right-hand side of Figure 2. First of all, Verilog-AMS nodes are mapped to SystemC-AMS nodes (①), and Verilog-AMS simultaneous statements are divided into basic contributions (②). Then, each contribution is mapped to a basic SystemC-AMS ELN element, where the equation associated with each ELN module is the same as the original Verilog-AMS contribution (③). The methodology determines how to connect the ELN modules (*i.e.*, in parallel or in series), so that the bindings describe the same relationship between voltages and currents as in the original Verilog-AMS simultaneous statement. The ELN system is then managed by the SystemC-AMS internal solver, that builds the corresponding equations (④) and adds conservative laws (⑤). The resulting equation system will thus reflect the Verilog-AMS one (⑥).

The choice of the ELN model of computation allows us to delegate the application of conservation laws to the internal solver. This is an important feature, as

adding conservative laws implies reconstructing the circuit topology from the AMS equations, which can be far from trivial.

It is important to note that both the basis of the methodology and the correctness of the proposed approach lie in the construction of the same equation set, that is then solved in the same way by the Verilog-AMS and SystemC-AMS solvers.

4 Methodology

The following sections describe the methodology in detail. This work focuses on the construction of the ELN system (steps ① to ③ in Figure 2). The remaining steps (*i.e.*, the bottom box on the right-hand side of Figure 2) are automatically performed by the SystemC-AMS internal solver. The visual representation of ELN modules adopted in the following figures is as defined by the SystemC-AMS standard [3].

4.1 The ABM abstraction level

The ABM abstraction level comprises descriptions mixing characteristics typical of digital behavioral models and of electrical conservative ones. ABM models are *behavioral* in that they do not directly reflect a hardware or circuit implementation, but they are used in the design process to simulate a component's behavior. At the same time, ABM models are *conservative* as they adopt circuit elements and constructs (*e.g.*, voltage and current values at circuit nodes), and thus abide by conservation laws.

These characteristics do not fit in any of the SystemC-AMS abstraction levels. Nonetheless, they are widely supported by other AMS HDLs for the design of components such as MEMS and analog circuitry [6, 7]. It is thus necessary to extend SystemC-AMS, to improve its coverage and effectiveness. To avoid the burden of implementing a new SystemC-AMS abstraction level (and thus new classes and libraries), this work proposes a methodology that converts ABM descriptions, modeled in other AMS HDLs, to SystemC-AMS ELN constructs. This guarantees the correctness of the underlying solution techniques, and it preserves compatibility with any SystemC-AMS description.

4.2 ELN terminology

ELN descriptions are based on the instantiation of an electrical network composed of electrical primitives (*i.e.*, *ELN modules*), connected together at *electrical nodes*. Each ELN module contributes to the equation system with a particular set of equations defining the mathematical relations at each node of the network.

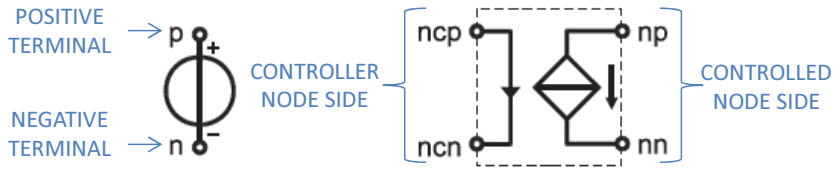

Fig. 3 ELN terminology applied to an independent source (left) and to a controlled source (right).

The interface of an electrical primitive is composed of a set of *terminals*. A negative (`n`)terminal and a positive (`p`) terminal are present on the interface of basic passive (linear) electrical components (*i.e.*, capacitor, inductor and resistor) and of voltage or current sources (left-hand side of Figure 3).

A particular sub-set of primitives, largely employed by the proposed methodology, is that of *controlled sources*. Controlled source primitives determine a current or voltage value on an output branch, whose generation linearly depends on the current or voltage value of an input branch. For this reason, controlled sources present two different interfaces, as depicted on the right-hand side of Figure 3: a *control* interface and a *controlled* interface. Each interface represents an *electrical branch*, *i.e.*, they are composed of a negative terminal and a positive terminal (*i.e.*, `ncn` and `ncp` terminals for the control interface, and `nn` and `np` for the controlled interface). Thus, the controlled sources introduce a set of relations where the value of voltage or current on the controlled branch is proportional to the value of voltage or current on the control branch.

4.3 Circuit node management

The first declaration added to the SystemC-AMS code is the instantiation of ground, declared as a node of type `sca_node_ref`. Verilog-AMS nodes are mapped to SystemC-AMS ELN circuit nodes (of type `sca_node`). Each node is then connected to ground through a 1 GΩ resistor, by using the ELN `sca_r` primitive. This is identical to the Gmin conductance that SPICE automatically inserts between each node and ground, and it helps to ensure the solution of the equation system specified by the circuit.

4.4 Division into contributions

SystemC-AMS is less expressive than Verilog-AMS, *i.e.*, it supports a more restricted range of constructs and ELN models can be composed only of instances of the predefined primitives [7, 10]. Furthermore, SystemC-AMS does not allow this set of predefined primitives to be extended. *E.g.*, in SystemC-AMS a voltage value can be controlled only by one voltage or current contribution, while Verilog-AMS

allows any number of contributions. Thus, a general Verilog-AMS simultaneous statement must be reproduced by connecting a number of ELN elements.

Given a Verilog-AMS description, our technique identifies the contributions comprising each simultaneous statement by finding the largest sub-equation that can be represented by a single ELN object. In linear and time-invariant descriptions this corresponds to breaking the equation into the single addends.

4.5 Mapping to ELN components

The remainder of this section shows how a set of template equations is mapped to ELN primitives to model their individual contributions and how such primitives are connected.

4.5.1 Voltage sources

Voltage source Verilog-AMS equations use a number of contributions to assign a voltage level to a circuit node. Contributions can be of three main types: independent, voltage-controlled and current-controlled. A complete example of a voltage source equation is shown in Figure 4.

Independent voltage sources assign a numerical voltage value, and they correspond to contributions like:

```
V(a) <+ 8.01
```

(*i.e.*, contribution 3 in Figure 4). They are implemented by using a `sca_vsource` ELN module, where the voltage value is an instantiation parameter (*i.e.*, +8.01). The module interface has only a positive terminal, connected to the controlled node (`a`), and a negative terminal, connected to ground (`gnd`).

A *voltage controlled voltage source* is a voltage source whose value depends on the voltage between a pair of circuit nodes. An example is contribution 1:

```
V(a) <+ +4.02 V(b)
```

This is implemented by using the `sca_vcvs` ELN module, where the scaling factor is an instantiation parameter (*i.e.*, +4.02). The module interface has a controlling node side (whose positive terminal is connected to `b`) and a controlled node side (whose positive terminal is connected to `a`). The negative controlling terminal is connected to ground.

A *current controlled voltage source* describes a voltage source whose value depends on the current through a circuit branch. An example is contribution 2:

```
V(a) <+ -3.72 I(c).
```

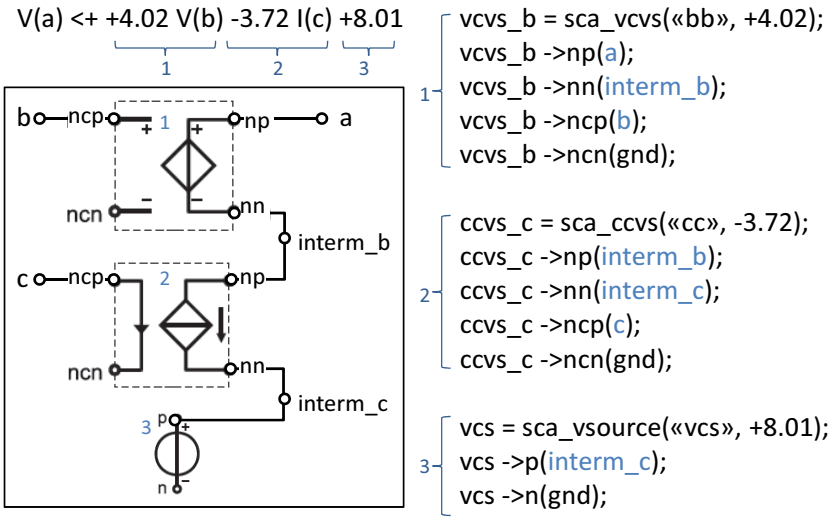

Fig. 4 Example of voltage source equation (top left) with the corresponding SystemC-AMS code (right) and ELN module connection (bottom left). Non-connected terminals are connected to ground.

Such contributions are implemented by using the `sca_ccvs` ELN module, connected to node `a` as the controlled node and to node `c` as the controlling node.

If a Verilog-AMS voltage source equation is made up of more than one contribution, SystemC-AMS instances are *connected in series*. This is achieved by creating intermediate nodes that connect the `nn` terminal of a primitive with the `np` terminal of the next primitive. In this way, voltage values add up and any new contribution is added in series with the former ones. In Figure 4, this is achieved by introducing intermediate nodes `interm_b` (that connects contributions 1 and 2) and `interm_c` (that connects contributions 2 and 3).

4.5.2 Current sources

Current source Verilog-AMS equations are the complement of voltage source equations, *i.e.*, they use a number of contributions to assign an input current to a circuit node. A complete example is shown in Figure 5.

An *independent current source* assigns a numerical current value and is implemented by using the `sca_isource` ELN module (contribution 3 in Figure 5). A *voltage controlled current source* defines a current source whose value depends on the voltage level at a certain circuit node (contribution 1 in Figure 5). These kinds of contributions are mapped to `sca_vccs` ELN modules. Finally, a *current controlled current source* describes a current source whose value depends on the current flow-

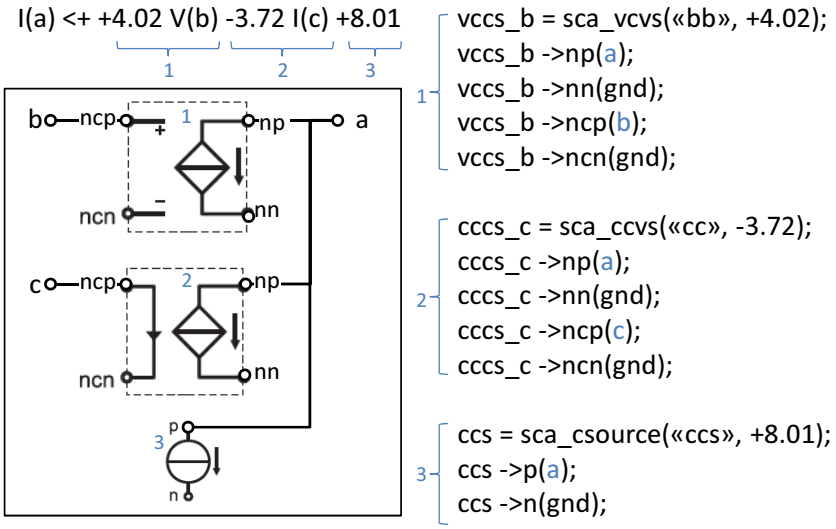

Fig. 5 Example of current source equation (top left) with the corresponding SystemC-AMS code (right) and ELN module connection (bottom left). Non-connected terminals are connected to ground.

ing through a certain circuit branch (contribution 2 in Figure 5). Such contributions are implemented by using `sca_cccs` ELN modules.

If a Verilog-AMS current source equation is made up of more than one contribution, SystemC-AMS instances are *connected in parallel*. The `ncp` terminal of each module is connected to the controlling node (`b` and `c`) and the `np` (or `p`) terminal is connected to the controlled node (`a`). In this way, the voltage is the same across all involved circuit branches and the current is summed at the controlled node `a`.

4.5.3 Differential constructs

Differential contributions are more complex than voltage or current ones, as they model a derivative (or integrative) relationship between the current or voltage of two separate circuit nodes. SystemC-AMS, however, restricts differential behaviors to dependencies on single network nodes, through the adoption of capacitors (`sca_c` ELN primitive) or inductors (`sca_l` ELN primitive). To overcome this limitation, it is necessary to introduce an intermediate node that has no physical correspondence in the circuit, but that is used for describing the differential dependence.

All the differential contributions are mapped using the generic topological pattern depicted in Figure 6, where colors depict the physical quantities involved: light blue for current and red for voltage, while yellow portions are dependent on the type of contribution to reproduce.

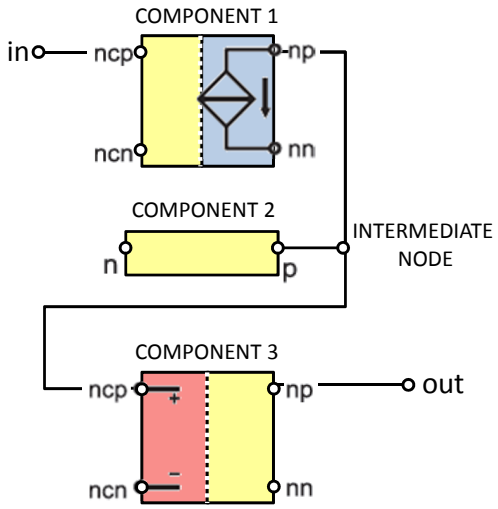

Fig. 6 Generic topological pattern used to implement differential contributions. All disconnected terminals are connected to ground.

Component 1 is a controlled current-source, as indicated by the controlled side in Figure 6 (in blue). The control side (in yellow) depends on the modeled contribution: if the argument of the derivative construct is a voltage, component 1 is a voltage controlled current source; else, if the argument is a current, component 1 is a current controlled current source.

Component 3 is a voltage-controlled source, as indicated by the control side in Figure 6 (in red). The controlled side (in yellow) reflects the target of the Verilog-AMS contribution statement: if the target is a voltage, component 3 is a voltage controlled voltage source; else if the target is a current, component 3 is a voltage controlled current source.

Component 2 (in yellow in Figure 6) is used to create the differential relation between the current value controlled by component 1 and the voltage value controlling Component 3. The component is an inductor whenever the differential contribution is derivative, and it is a capacitor in the case of an integrative contribution. Given $I_{np,nn}$ the current flowing through terminals `np` and `nn` of Component 1, and $V_{ncp,ncn}$ the voltage on the branch between the terminals `ncp` and `ncn` of Component 3, the relationship described by Component 2 is thus:

$$V_{ncp,ncn} = \int I_{np,nn} dt$$

in the case of a derivative contribution (*i.e.*, Component 2 is a capacitor), and

$$V_{ncp,ncn} = \frac{dI_{np,nn}}{dt}$$

in the case of an integrative contribution (*i.e.*, Component 2 is an inductor).

Table 1 Summary of the components employed to map differential contributions.

Contribution	Component 1	Component 2	Component 3
I(*out*)<+ k ddt(I(in_1))	Current Controlled Current Source sca_cccs	Inductor sca_l	Voltage Controlled Current Source sca_vccs
I(*out*)<+ k ddt(V(in_1))	Voltage Controlled Current Source sca_vccs	Inductor sca_l	Voltage Controlled Current Source sca_vccs
V(*out*)<+ k ddt(I(in_1))	Current Controlled Current Source sca_cccs	Inductor sca_l	Voltage Controlled Voltage Source sca_vcvs
V(*out*)<+ k ddt(V(in_1))	Voltage Controlled Current Source sca_vccs	Inductor sca_l	Voltage Controlled Vurrent Source sca_vcvs
I(*out*)<+ k idt(I(in_1))	Current Controlled Current Source sca_cccs	Capacitor sca_c	Voltage Controlled Current Source sca_vccs
I(*out*)<+ k idt(V(in_1))	Voltage Controlled Current Source sca_vccs	Capacitor sca_c	Voltage Controlled Current Source sca_vccs
V(*out*)<+ k idt(I(in_1))	Current Controlled Current Source sca_cccs	Capacitor sca_c	Voltage Controlled Voltage Source sca_vcvs
V(*out*)<+ k idt(V(in_1))	Voltage Controlled Current Source sca_vccs	Capacitor sca_c	Voltage Controlled Current Source sca_vcvs

Considering the derivative and the integrative operators of Verilog-AMS, we can restrict all possible configurations of the topological pattern to the eight cases summarized in Table 1. For each case, the table shows the SystemC-AMS primitives used to instantiate Components 1, 2 and 3. The remainder of this section shows the application to two example cases, *i.e.*, a derivative contribution of type $I(out) < +kddt(V(in_1))$ and an integrative contribution of type $I(out) < +kidt(V(in_1))$, respectively.

Derivative contributions

Consider a derivative contribution of the form of the second entry of Table 1:

```
I(a) <+ ddt(+4.02 V(b))
```

as exemplified in Figure 7. Given the derivative nature of the contribution, the circuit requires a new node (`interm`), that is connected to an inductor (*i.e.*, an instance of a `sca_l` ELN module). This adds the following equation (equation 2 in Figure 7):

```
V(interm) = ddt(I(interm))
```

Then, two equations are necessary to bind the values in `a` and `b` to the current and voltage values in the new node `interm`. In Figure 7, node `a` is modeled as a current source dependent on the voltage in `interm` (the dependency is implemented as an instance of `sca_vccs`). This adds equation 3:

```
I(a) = +4.02 V (interm)
```

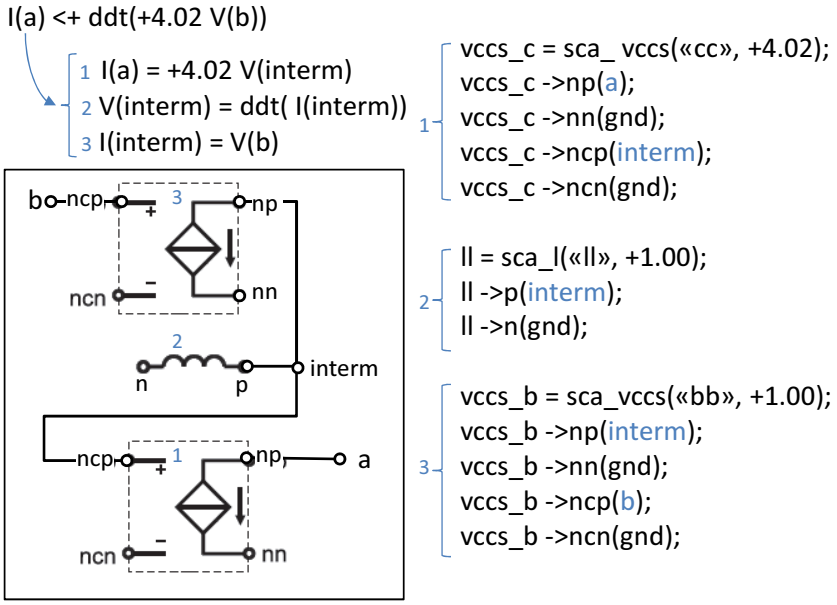

Fig. 7 Example of derivative equation with corresponding individual contributions and equations (top left), SystemC-AMS code (bottom left) and ELN module connection (right).

The current through `interm` is controlled by the voltage at `b` (the dependency is implemented as an instance of `sca_vccs`). This adds equation 1:

```
I(interm)= V(b)
```

The resulting system of equations will thus reconstruct the original dependency between nodes:

```
I(a) = +4.02V(interm)= +4.02ddt(I(interm))
                     = +4.02ddt(V(b))
```

This reflects the mapping defined in Table 1. The resulting SystemC-AMS subsystem is depicted on the left-hand side of Figure 7. The sub-system can be further connected to other ELN models if it is included in more complex simultaneous statements, by adopting either parallel or series compositions.

Integrative contributions

An integrative contribution illustrates the sixth case of Table 1:

```
I(a) <+ idt(+4.02 V(b))
```

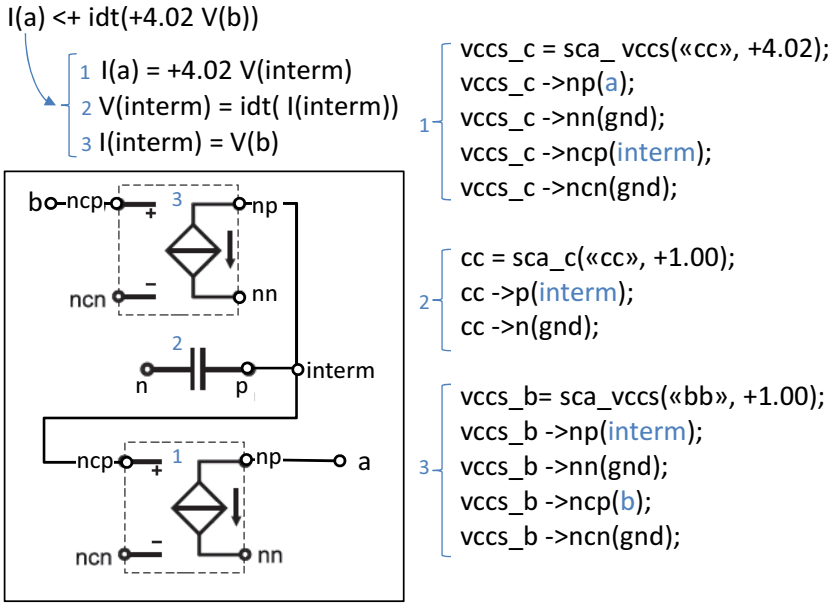

Fig. 8 Example of integrative equation with corresponding individual contributions and equations (top left), SystemC-AMS code (bottom left) and ELN module connection (right).

as shown in Figure 8. The circuit is extended with a new node (`interm`), used to represent the integrative dependency as a capacitor (an instance of the `sca_c` ELN module). This adds to the system equation 2:

```
V(interm) = idt(I(interm))
```

Then, two equations are necessary to bind the values in `a` and `b` to the current and voltage values in the new node `interm`, similarly to the solution proposed for derivative contributions:

```
I(a) = +4.02 V(interm)
I(interm) = V(b)
```

The resulting set of equations will thus reproduce the original dependency between nodes:

```
I(a) = +4.02V(interm)= +4.02idt( I(interm))
                      = +4.02idt( V(b))
```

The resulting SystemC-AMS sub-system is depicted on the left-hand side of Figure 8. The sub-system can be further connected to other ELN models if it is included in more complex simultaneous statements, by adopting either parallel or series compositions.

5 Experimental results

This section demonstrates the effectiveness of the proposed approach in terms of accuracy and simulation time. All experiments were evaluated on an i7 3.2GHz processor with 16GB RAM, running Ubuntu 14.04. Verilog-AMS descriptions have been simulated using Mentor's Questa 13.1 simulator [11].

5.1 Methodology automation

Manual application of the proposed methodology is a tedious error-prone process, and application to industrial case studies could be extremely difficult. For this reason, we implemented an automatic tool *ABACuS* (Analogue BehAvioural Conservative Systemc-ams). *ABACuS* leverages the academic license version of HIFSuite to ease the conversion process [12]. Verilog-AMS descriptions are analyzed and translated into the HIFSuite internal format (HIF). The code generated at this point is a tree-structured XML-like representation of the original code. *ABACuS* applies a number of processing steps to the HIF description to automate the methodology, including contribution identification and construction of the ELN system. This leads to a new HIF description, containing the instantiation and connection of the corresponding ELN primitives. The HIF description is then converted to SystemC-AMS by means of the HIFSuite *hif2sc* back-end tool.

5.2 Methodology validation

The first step to validate the propose methodology is to evaluate the accuracy of mapping single types of contribution, as detailed in Section 4. The accuracy is evaluated with 12 case studies, each targeting a single type of contribution. The case studies were implemented in Verilog-AMS and then converted to SystemC-AMS via *ABACuS*. The main characteristics of each case study are reported in Table 2, in terms of target contribution type and simulation time. Case studies are fed with sinusoidal inputs with 1KHz frequency, so that the outputs can be easily controlled and compared *w.r.t.* the expected theoretical results. In particular, the system of equations described using Verilog-AMS has been computed symbolically and solved for every time instant where a sample is collected by the SystemC-AMS execution. The SystemC-AMS simulation is run with an integration and sampling period of 10ns.

Simulation times in Table 2 refer to the amount of time needed to perform 1 second of transient simulation of the circuit implementing the given basic contribution. For all the cases depicted in Table 2 , the time needed for simulating the initial Verilog-AMS code matches that needed for the SystemC-AMS simulation. This is due to the fact that both the solvers (*i.e.*, SystemC-AMS and Questa) are solving the same set of equations, as described in Section 3.

Table 2 Validation of the mapping of each type of contribution to ELN constructs.

Case study	Target contribution	Simulation time (s)	Normalized RMSE
1	$\mathtt{V}(out) <+\ k_1\mathtt{V}(in_1) + \ldots + k_n\mathtt{V}(in_n) + c$	69.05	4.441e-7
2	$\mathtt{V}(out) <+\ k_1\mathtt{I}(in_1) + \ldots + k_n\mathtt{I}(in_n) + c$	70.59	4.441e-7
3	$\mathtt{I}(out) <+\ k_1\mathtt{V}(in_1) + \ldots + k_n\mathtt{V}(in_n) + c$	69.11	4.441e-7
4	$\mathtt{I}(out) <+\ k_1\mathtt{I}(in_1) + \ldots + k_n\mathtt{I}(in_n) + c$	69.01	4.441e-7
5	$\mathtt{I}(out) <+\ \mathtt{k\ ddt}(\mathtt{I}(in_1))$	51.15	4.733e-6
6	$\mathtt{I}(out) <+\ \mathtt{k\ ddt}(\mathtt{V}(in_1))$	52.08	4.733e-6
7	$\mathtt{V}(out) <+\ \mathtt{k\ ddt}(\mathtt{I}(in_1))$	51.64	4.733e-6
8	$\mathtt{V}(out) <+\ \mathtt{k\ ddt}(\mathtt{V}(in_1))$	51.88	4.733e-6
9	$\mathtt{I}(out) <+\ \mathtt{k\ idt}(\mathtt{I}(in_1))$	49.01	3.936e-9
10	$\mathtt{I}(out) <+\ \mathtt{k\ idt}(\mathtt{V}(in_1))$	48.70	3.936e-9
11	$\mathtt{V}(out) <+\ \mathtt{k\ idt}(\mathtt{I}(in_1))$	48.89	3.936e-9
12	$\mathtt{V}(out) <+\ \mathtt{k\ idt}(\mathtt{V}(in_1))$	49.16	3.936e-9

Table 2 reports also the level of accuracy *w.r.t.* the expected theoretical results. The error is given in terms of the *Room Mean Square Error*, *i.e.*, by normalizing the error to the mean of the measured values. Thus, it represents the *Coefficient of Variation* between the set of samples gathered during the simulation and the expected theoretical behavior. This proves that the error *w.r.t.* the theoretical results is extremely low, since the Coefficient of Variation between the theoretical reference and the simulation result is always less than $2e-05$, and in some cases it is as good as $4e-09$. This error is due to the precision issues of the numerical algorithms used by the simulator to perform continuous time simulation.

The similar simulation times imply that the proposed translation to SystemC-AMS does not provide any simulation speed up, as both the simulators solve the same set of equations with similar strategies. However, Sections 5.4 and 5.5 will highlight the effectiveness when handling more complex designs, including mixed analog and discrete descriptions.

5.3 Methodology scalability

In order to show the scalability of the proposed methodology another set of experiments was performed. Two circuits with a single contribution statement were simulated. One circuit has a single non-differential contribution (*i.e.*, $\mathtt{V}(out) <+\ k_1\mathtt{V}(in_1) + \ldots + k_n\mathtt{V}(in_n) + c$), while the second has a differential statement (*i.e.*, $\mathtt{I}(out) <+\ k_1\mathtt{ddt}(\mathtt{V}(in_1))$). Table 3 shows the results of this set of experiments focusing on the two particular kinds of contribution, but similar results apply also to the other case studies. The simulation time refers to the execution of 1 second of simulated time, while the accuracy is given in terms of the Normalized Root Mean Square Error used also for the experiments presented in Table 2.

The SystemC-AMS code is stimulated with three different sinusoidal inputs, with increasing maximum input frequencies. For each input, we simulated the code with

Table 3 Scalability of the proposed methodology *w.r.t.* the simulation timestep.

		Non-differential		Differential	
Input frequency	Adopted timestep	Normalized RMSE	Simulation time (s)	Normalized RMSE	Simulation time (s)
10 Hz	10 ns	4.53e-09	70.35	2.37e-09	51.23
	100 ns	4.53e-08	6.99	5.09e-10	5.09
	1 us	4.53e-07	0.78	1.71e-09	0.52
100 Hz	10 ns	4.44e-08	70.66	2.42e-09	50.98
	100 ns	4.44e-07	7.07	1.24e-09	5.12
	1 us	4.44e-06	0.73	1.66e-07	0.52
1 KHz	10 ns	4.44e-07	70.38	3.94e-09	50.64
	100 ns	4.44e-06	7.07	1.66e-07	5.47
	1 ns	4.44e-05	0.74	1.66e-05	0.52

different time steps, ranging from 10ns up to 1us. The simulation time decreases linearly with the length of the time step, with a speedup of approximately $100\times$ between a timestep of 10ns and a timestep of 1us. At the same time, accuracy is preserved, as the Coefficient of Variation between the theoretical reference and the simulation result is always less than $2e-05$. It is important to note that the error depends both on the adopted timestep, and also on the frequency of the sinusoidal inputs, especially in the non-differential case, where the highest accuracy (*i.e.*, 4.53e-09) is reached with the 10Hz input and the timestep of 10ns. On the other hand, the combination with the smallest frequency and the largest sample period performs worse than the others, with errors of $4.44e-05$ and $1.66e-05$ for the non-differential and the differential cases, respectively. This is due to the fact that the adopted time step is too coarse for the input frequency. Considering the differential contribution, it worths noticing that once a certain precision is reached, it does not scale linearly as in the non-differential case. However, this happens when an extremely high precision is reached. These considerations highlight the importance of choosing a suitable timestep for the simulation, but also that the generated SystemC-AMS code allows us to determine accuracy/simulation speed trade offs.

5.4 The MEMS accelerometer

In order to prove the effectiveness of the overall methodology on more complex designs, we applied the technique to a complex industrial case study, developed in the context of an industrially-funded project. The case study is a *2-dimensional MEMS accelerometer* modeled in Verilog-AMS by means of the MEMS design platform MEMS+. This supports automatic Verilog-AMS code generation [6], starting from 3-dimensional physical models such as that depicted in Figure 10. The choice of a MEMS design was guided by the consideration that MEMS behavioral modeling is based on differential and algebraic equations [7], thus following the Verilog-AMS structure assumed in this work. Table 4 reports the main characteristics of the

Table 4 Characteristics of the original Verilog-AMS MEMS design.

Lines of code		89
Equations	Voltage sources	10
	Current sources	15
Node declarations	Interface	14
	Internal	14
Contributions	Independent	4
	Voltage	59
	Current	0
	Derivative	12
	Integrative	0

MEMS design, both in terms of simultaneous statements and of types of contributions.

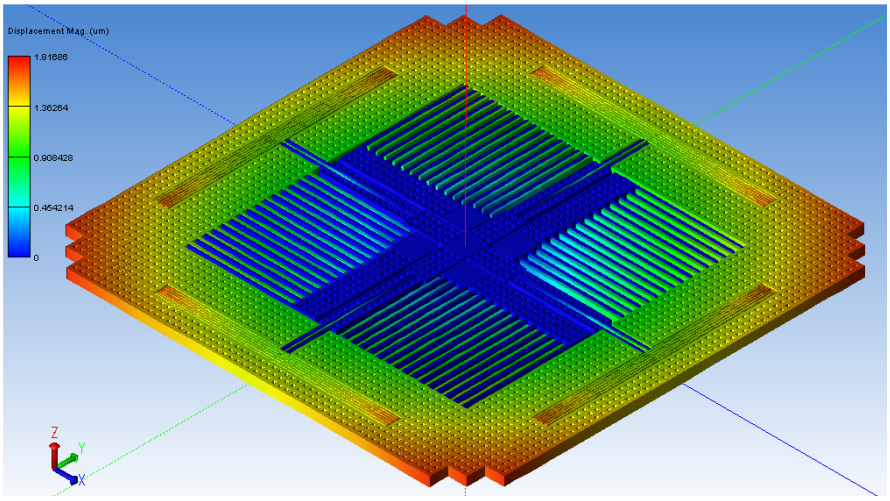

Fig. 9 3-dimensional model of the accelerometer in the MEMS+ design simulator.

Table 5 shows the results of the application of *ABACuS* to the MEMS design. The table shows the number of lines of code of the resulting SystemC-AMS implementation, the number of added nodes and of instances of SystemC-AMS primitives. The number of lines of codes is increased tenfold (precisely, 11.12×x), as the SystemC-AMS generated by the methodology is more verbose than Verilog-AMS. Each contribution requires the instantiation of the ELN primitive, plus the corresponding explicit port binding. Furthermore, the number of ELN primitives is higher than the number of Verilog-AMS contributions. This is due to the presence of 12 derivative contributions in the original Verilog-AMS code. Each such contribution determines the instantiation of three ELN primitives (as explained in Section 4.5.3). As a result, of the 188 resulting SystemC-AMS ELN instances:

Table 5 Characteristics of the generated SystemC-AMS MEMS design.

Lines of code		1,474
Added node declarations		12
SystemC-AMS primitive instantiations	`sca_r`	93
	`sca_vsource`	4
	`sca_vcvs`	32
	`sca_ccvs`	0
	`sca_csource`	0
	`sca_vccs`	48
	`sca_cccs`	0
	`sca_l`	12
	`sca_c`	0

Table 6 Characteristics of the execution of *ABACuS* on the MEMS design.

Overall		17.48s
HIFSuite tools	Conversion to HIF	1.86s
	Conversion to SystemC-AMS	7.81s
ABACuS	Node management	0.94s
	Division into contributions	0.29s
	ELN component instantiations	6.58s

- 93 correspond to resistors added to connect each SystemC-AMS node to ground;
- 59 correspond to voltage source contributions;
- 36 are generated by the 12 derivative constructs, that also require 12 additional internal nodes.

The numbers highlight that *ABACuS* strictly follows the presented methodology, in particular:

- one resistor is added for each circuit node;
- each non-derivative contribution determines the addition of one ELN primitive instance;
- each derivative contribution generates three ELN primitive instances.

Fast code generation is a major advantage of the proposed approach. Table 6 highlights that code generation is almost instantaneous (17.48s overall), and that most of the effort in spent in the HIFSuite conversions (55%). The most costly step of *ABACuS* execution lies in the mapping from Verilog-AMS contributions to ELN primitives and in their instantiation (37%). On the other hand, node management and the separation of Verilog-AMS equations into single contributions is almost immediate.

The generated code was validated by comparing its execution *w.r.t.* the original Verilog-AMS code. SystemC-AMS simulation was run by adopting the same input stimula as the Verilog-AMS implementation, and with a 1us timestep. SystemC-AMS simulation proved to be slightly faster than the Verilog-AMS execution (28.02s and 33.72s, respectively). At the same time, the average error in the computation of the MEMS outputs is 0.02%. This confirms the visual accuracy evident

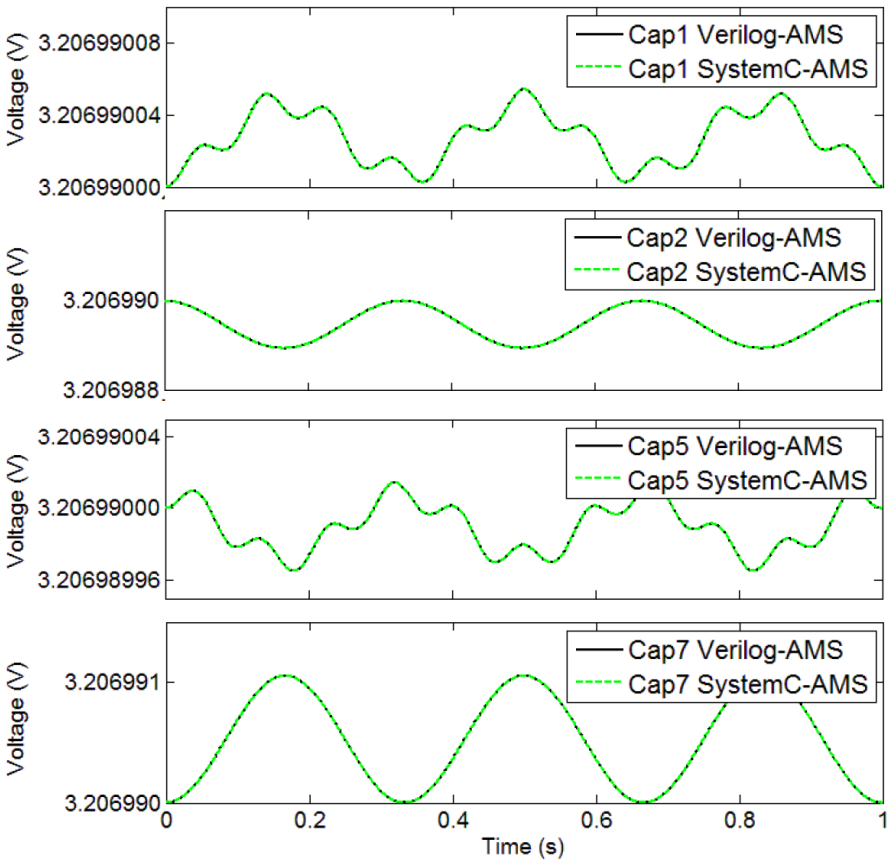

Fig. 10 Evolution of the MEMS outputs for Verilog-AMS (solid) and SystemC-AMS (dashed).

from Figure 10, where the Verilog-AMS and SystemC-AMS curves are almost totally coincident. The small error is due to the different management of time in the two simulators: SystemC-AMS adopts a fixed timestep, while Verilog-AMS can adapt the length of the timestep over time, thus reaching a higher accuracy. The low error rate highlights the effectiveness of the generated code, both in terms of accuracy and of simulation speed.

5.5 Effectiveness of the proposed approach

The most important advantage of modeling ABM models in SystemC-AMS lies in the ease of integration in more complex platforms and in the enhanced support for virtual platforms and system-level design, rather than in the pure accuracy or simulation speed.

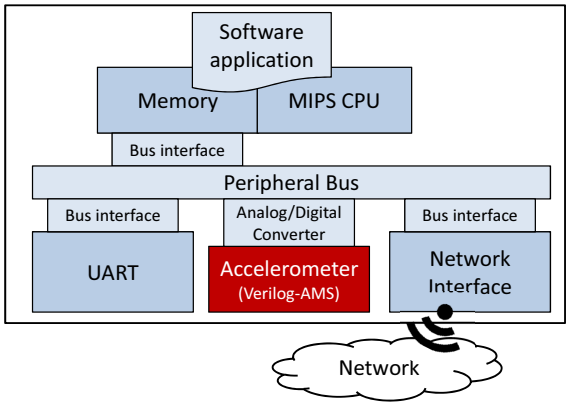

Fig. 11 Overview of the virtual platform containing the MEMS (*i.e.*, the Accelerometer) component. Digital components, implemented using SystemC, are colored in light blue. The MEMS (colored in red) is originally implemented in Verilog-AMS.

The SystemC-AMS scheduler is an extension of the discrete-event SystemC scheduler, and it thus allows simultaneous simulation of components belonging to heterogeneous domains. Furthermore, integration with a C++ system level description is eased, thus further removing computationally expensive interfaces and thus speeding up the simulation of mixed-signal systems. For these reasons, SystemC-based languages are a winning solution for the construction and validation of virtual platforms, and they are adopted by most of the currently available virtual platform environments [13–16]. Translating ABM models to SystemC-AMS thus allows their early validation, together with the interaction with other system components.

The MEMS accelerometer has been integrated into a virtual platform for smart systems. The structure of the platform is depicted in Figure 11. It includes (1) a 32-bit RISC processor (the *MIPS CPU*) executing (2) a *Software Application* elaborating data sensed by the accelerometer and stored in (3) a *Memory*. External communication is managed by (4) a Universal Asynchronous Receiver/ Transmitter (*UART*) and by (5) a *Network Interface*, used to send and receive data to and from other smart sensors. All system components, except the MEMS, are modeled in SystemC, and integrated in a virtual platform. Validating the integration of the original MEMS component in the overall system would thus require the construction of a simulation framework.

The first alternative to validate the integration of the MEMS component in the platform is to preserve the language heterogeneity, but within a single simulator. Thus, we adopted Questa [11], which handles both discrete-time and analog descriptions, and that natively provides SPICE-based constructs to connect analog and digital designs.

The second alternative is to adopt the methodology proposed in this work to convert the MEMS design to SystemC-AMS, integrate it in the virtual platform and to run the overall system with the SystemC simulator.

Table 7 Simulation time of the virtual platform by preserving the language heterogeneity and moving to SystemC-AMS.

Languages	Simulator	Simulation time (s)
SystemC and Verilog-AMS	Questa	215.47
SystemC and SystemC-AMS	SystemC-AMS kernel	97.59

Table 7 reports the time needed to simulate 100ms of real system execution, and it shows how the SystemC based simulation outperforms Questa (by 2.21×). This is mainly due to the heavy communication overhead induced by Questa to allow communication and synchronization between the discrete event and the Spice-base simulators used by Questa respectively for the SystemC and Verilog-AMS parts of the model. At the same time, SystemC-AMS provides a good level of accuracy (0.02%), thus constituting a valid alternative for early validation of the overall system and of the analog-digital communications.

6 Conclusions

The work described here proposes a methodology for representing models that are both conservative and behavioural in SystemC-AMS. We achieve this goal by adopting existing SystemC-AMS ELN primitives in a novel way. As a result, SystemC effectiveness is enhanced in the context of embedded system design, as it can cover a wider range of descriptions and components. Experimental results highlight the correctness of the proposed approach both on synthetic case studies, focusing on the single methodology steps, and on a complex industrial MEMS case study. Future work will focus on the identification of abstraction strategies to target the SystemC-AMS Timed Data Flow (TDF) level for improved simulation performance.

Acknowledgements This work has been partially supported by the European project SMAC FP7-ICT-2011-7-288827.

References

1. IEEE, “1666-2011 - IEEE Standard for Standard SystemC,” 2011, standards.ieee.org/findstds/standard/1666-2011.html.
2. R. Zafalon, “Smart system design: Industrial challenges and perspectives,” in *Proc. of IEEE MDM*, 2013, p. 3.
3. Accellera Systems Initiative, “SystemC-AMS and Design of Embedded Mixed-Signal Systems,” 2013, accellera.org/activities/working-groups/systemc-ams.
4. L. W. Nagel and D. O. Pederson, *SPICE: Simulation program with integrated circuit emphasis*. Electronics Research Laboratory, College of Engineering, University of California, 1973.

5. Accellera Systems Initiative, “Verilog-AMS,” 2014, accellera.org/downloads/standards/v-ams.
6. Coventor, Inc., “MEMS+: MEMS Simulation Software,” www.coventor.com/mems-solutions/products/mems.
7. P. Schneider, C. Bayer, K. Einwich, and A. Kohler, “System level simulation - A core method for efficient design of MEMS and mechatronic systems,” in *Proc. of IEEE SSD*, 2012, pp. 1–6.
8. S. Mijalkovic, “Advanced circuit and device modeling with Verilog-A,” in *Proc. of IEEE MIEL*, 2006, pp. 439–442.
9. P. Hartmann, P. Reinkemeier, A. Rettberg, and W. Nebel, “Modelling control systems in SystemC-AMS – Benefits and limitations,” in *Proc.of IEEE SOCC*, 2009, pp. 263–266.
10. R. Narayanan, N. Abbasi, M. Zaki, G. A. Sammane, and S. Tahar, “On the simulation performance of contemporary AMS hardware description languages,” in *Proc. of IEEE ICM*, 2008, pp. 361–364.
11. Mentor Graphics, “Questa Advanced Simulator,” www.mentor.com/products/fv/questa.
12. N. Bombieri, G. Di Guglielmo, M. Ferrari, F. Fummi, G. Pravadelli, F. Stefanni, and A. Venturelli, “Hifsuite: tools for hdl code conversion and manipulation,” *EURASIP Journal on Embedded Systems*, vol. 2010, pp. 4:1–4:20, Jan. 2010.
13. Synopsys, “Platform architect,” www.synopsys.com/Prototyping/ArchitectureDesign.
14. Cadence, “Virtual System Platform,” www.cadence.com/products/sd/virtual_system.
15. Imperas Software, “OVP - Open Virtual Platforms,” www.ovpworld.org.
16. Mentor Graphics, “Vista Virtual Prototyping for SystemC/TLM 2.0 and QEMU ,” www.mentor.com/esl/vista/virtual-prototyping.