VHDL-AMS and Verilog-AMS as Alternative Hardware Description Languages for Efficient Modeling of Multi-Discipline Systems

F. Pêcheux, C. Lallement, *Member*, *IEEE*, and A. Vachoux, *Member, IEEE*

***Abstract*—This paper focuses on commonalities and differences between the two mixed-signal hardware description languages VHDL-AMS and Verilog-AMS in the case of modeling heterogeneous or multi-discipline systems. The paper has two objectives. The first one consists of modeling the structure and the behavior of an airbag system using both the VHDL-AMS and the Verilog-AMS languages. Such a system encompasses several time abstractions (i.e. discrete-time and continuous-time), several disciplines, or energy domains (i.e., electrical, thermal, optical, mechanical, and chemical), and several continuous-time description formalisms (i.e., conservative-law and signal-flow descriptions). The second objective is to discuss the results of the proposed modeling process in terms of the descriptive capabilities of the VHDL-AMS and Verilog-AMS languages and of the generated simulation results. The tools used are Advance-MS from Mentor Graphics for VHDL-AMS and AMS Simulator from Cadence Design Systems for Verilog-AMS. The paper shows that both languages offer effective means to describe and simulate multi-discipline systems, although using different descriptive approaches. It also highlights current tool limitations since full language definitions are not yet supported.**

***Index Terms*—** VHDL-AMS, Verilog-AMS, Mixed-signal Simulation, Accelerometer, EKV MOS Model, Generalized Kirchhoff Laws, Optical link, Chemical system, Airbag, Thermo-electrical Interactions

I. Introduction

For the past three decades, hardware description languages (HDLs) have been widely used to model and simulate systems belonging to various engineering fields, from digital and analog electronics to mechanics and chemistry. For a long time, all these fields have been completely separated, each scientific community having its own design methodologies, tools and idiosyncrasies. Dedicated HDLs supporting the description and the simulation of the logical structure of a system belonging to a particular engineering domain were, and are still available. They allow the description of a system as a possibly hierarchical connection of predefined submodels, or primitives. For example, in the electrical/electronic domain, the SPICE simulator and all its derivatives allow the description of the netlist of a circuit using electrical primitives such as resistors, capacitors, sources and transistors. In an attempt to support the modeling and simulation of non-electrical systems as well, several modeling methods using energy equivalences between the electrical domain and other domains such as mechanical, thermal or fluidic domains have been proposed [1][2][3].

With the advent of nano-technologies, the design of innovative integrated devices, like Micro-Opto-Electro-Mechanical Systems (MOEMS), has shifted from vertical only to both vertical and horizontal integration. Using the benefit of all the experience acquired in incremental design, MOEMS design now involves strong "horizontal" interaction of different application-field parts on the very same chip (e.g., mechanical, electrical, thermal, fluidic parts), with partial close coupling between these fields. Neglecting the interaction effects or the cross coupling between parts may have disastrous consequences on the final design in terms of a loss in performance or an increase in design time.

One way of addressing this issue is to use a consistent modeling and simulation framework that allows for the description of systems from different disciplines and for the description of interactions between these systems. This is where the VHDL-AMS HDL [4][5][6] and the Verilog-AMS HDL [7][8] come in as effective backbones of the aforementioned framework.

In this paper, we therefore propose to compare the modeling power of these two HDLs. In order to illustrate this, we present a meaningful case study that consists in modeling the structure and the behavior of an airbag system using both the VHDL-AMS and the Verilog-AMS languages. Such a system is well suited for our purpose as it encompasses several time abstractions (i.e. discrete-time and continuous-time), several disciplines, or energy domains (i.e., electrical, thermal, optical, mechanical, and chemical), and several continuous-time description formalisms (i.e., conservative-law and signal-flow descriptions). In addition, the case study allows us to highlight the key modeling capabilities offered by both analog

Manuscript received June 6, 2002.

F. Pêcheux is with the LIP6 Integrated Systems Architecture Department, Université Pierre et Marie Curie, 12, rue Cuvier, 75252 Paris Cedex 05, France (phone: 00-33-1-4427-6528; fax: 00-33-1-4427-7280; e-mail: francois.pecheux@lip6.fr).

C. Lallement, is with the ERM-PHASE/ENSPS Laboratory, Pôle API, Bld S. Brant, BP. 10413, F-67412 Illkirch Cedex, France (e-mail: christophe.lallement@ensps.u-strasbg.fr).

A. Vachoux is with the LSM Microelectronic Systems Laboratory, Ecole Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland (e-mail: alain.vachoux@epfl.ch).

and mixed-signal HDLs in terms of expressive power and quality of simulation results.

The paper does not intend to provide a complete presentation of the two languages. Rather, it focuses on specific analog and mixed-signal modeling and simulation aspects and on how the two design languages can actually address them.

The paper is organized as follows. Section 2 introduces the context of the work, i.e. the modeling and simulation aspects that are addressed in the paper. It gives a short presentation of the VHDL-AMS and Verilog-AMS design languages. Section 3 presents a complete multi-discipline case study, consisting of an airbag system, and discusses selected VHDL-AMS and Verilog-AMS models. Section 4 gives relevant simulation results obtained with the VHDL-AMS and the Verilog-AMS tools. Section 5 summarizes the insights we can learn from the case study. Finally, section 6 gives some conclusions.

II. CONTEXT OF THE WORK

A. Modeling and Simulation aspects

Modeling is at the core of any design process. This task essentially consists in developing abstract descriptions of some physical reality in such a way that they are useful for the design process. Models may be used to validate characteristics of some part or of the whole of the designed system, e.g. its functionality or its performances. Such models are simulation models, or executable models that produce a response when acted upon by stimuli. These are the kind of models this paper is dealing with. To be complete, models may also be used as a support for a synthesis process, i.e., the progressive refinement process that starts from an abstract description of what the system should do and in which conditions, i.e. its specifications, and that ends when a detailed description of a realization that meets the specifications is obtained.

Models may describe the behavior and/or the structure of the designed system at various levels of details, or levels of abstraction. Selecting the appropriate level is, on the one hand, a matter of compromise between model accuracy and model performance, and on the other hand, a means to cope with system complexity. Abstraction may be applied to different aspects of a model. One aspect is how time is considered. Time may be abstracted as pure causal effects, as synchronous cycles, as integer multiple values of a minimum resolution time (discrete time), or as real values (continuous time). Another aspect is how the values of model data are represented. They can be tokens that merely represent the existence or the absence of data at some point in time and at some place in the system, or they can be bits or bit words that represent logic information, or they can be real values. As a data abstraction level is usually related to a time abstraction level, this paper considers the so-called *discrete-time models* as models in which data is abstracted as logic functions of a discrete variable that represents the time, and the so-called *continuous-time* models in which the data is abstracted as real-valued functions of a real-valued independent variable that represents the time [4]. Discrete-time behaviors are generally expressed as logical Boolean equations or as communicating processes that are triggered upon events, while continuous-time behaviors are generally expressed as Differential Algebraic Equations (DAEs). For electrical systems, the term *digital* is more often used to denote a discrete-time characteristic and the term *analog* is more often used to denote a continuous-time characteristic.

Hierarchy is also a means to cope with complexity by decomposing a complex behavior or structure into a set of smaller manageable pieces, the latter being possibly further recursively decomposed up to any appropriate level of deepness. *Leaf components* are those components that are at the deepest level of the hierarchy.

Each decomposed model piece must therefore include additional information on how it relates to other pieces. In the case of a hierarchical structure, this amounts to define a set of interconnected components that communicate or share data through interface elements called *ports*. The nature of the ports may be abstracted either as directional signal-flows, for both discrete-time and continuous-time models, or as satisfying conservative-law relationships between *quantities*, for continuous-time models only. Conservative-law relationships assume the existence of two classes of specialized quantities, namely *across quantities* that represent an effort (e.g., a voltage for electrical systems or a velocity for mechanical systems), and *through quantities* that represent a flow (e.g., a current for electrical systems or a force for mechanical systems). They also state that so-called *General Kirchhoff Potential Law* (GPL) and *General Kirchhoff Flow Law* (GFL) are met. These two laws are essentially the Kirchhoff's laws for electrical circuits generalized to any kind of conservative energy systems, e.g., mechanical, thermal, or fluidic systems.

The interconnection of components in a level of hierarchy is done through *nets* that link two or more ports. The nature of a net is derived from the nature of the ports it links together, so it can be either a discrete-time (digital) net or a continuous-time (analog) net. Nets only define topological relationships. *Signals*, as hierarchical collections of nets, are the model objects that actually carry data. This general definition models all kinds of physical quantities which are able to exchange energy or information between submodels [9] and should not be mistaken with the more specific VHDL definition of a signal. Depending on the nature of nets, signals may be either discrete-time (digital) or continuous-time (analog) signals. A *mixed-signal model* is a model that uses both discrete-time and continuous-time signals.

The simulation of discrete-time models relies on event-driven techniques for which the internal states of the model are only, and selectively, reevaluated when signals change their values. The reevaluation process includes the execution of Boolean functions or the execution of sequences of instructions. A so-called *discrete-time (digital) simulation kernel* is responsible for the management of the events and the

selective reevaluation of the model states.

On the other hand, the simulation of continuous-time models uses a number of, usually numerical, techniques to solve the set of DAEs that come from the behavior of the system and its signal-flow or conservative-law connection semantics. The set of simulation techniques used form the so-called *continuous-time (analog) simulation kernel*. As differential equations are discretized using a fixed or a variable time step, the true continuous-time behavior of the solutions is actually approximated as piecewise linear functions of time. The times at which the solutions are computed are called *analog solution points* or ASPs. The quality of the solution therefore depends on tolerances that define the discretization time step and other characteristic values related to the numerical techniques used [10]. Another important characteristic of the simulation of continuous-time models is that a consistent initial (quiescent) operating point is required. Without it, it is likely that inaccuracies or non-convergence issues will arise during the rest of the simulation.

The simulation of mixed-signal models requires in addition mechanisms to convert between discrete and continuous representations of time and signal values, and a synchronization mechanism between the discrete-time and the continuous-time simulation kernels to handle mixed-signal interactions such as threshold crossings. Last but not least, a consistent quiescent operating point, this time possibly involving some discrete-time signals as well, is still required.

A hardware description language (HDL) is a programming language for developing executable simulation models of hardware systems. In this paper, the considered hardware systems are not only electrical or electronic systems, but more generally heterogeneous systems. An HDL that can be used to describe models in several domains is said to be “multi-disciplines” or “mixed-disciplines”.

B. Two competitive Mixed-Signal HDLs

VHDL-AMS is the result of an IEEE effort to extend the VHDL language to support the modeling and the simulation of analog and mixed-signal systems. The effort culminated in 1999 with the release of the IEEE standard 1076.1-1999 [4]. Verilog-AMS, on the other hand, is intended to be an extension of the Verilog HDL language to also support the modeling and the simulation of analog and mixed-signal systems. Verilog is a digital HDL that has been released in 1995 as IEEE standard 1365-1995. The Verilog-AMS language reference manual is currently being completed under the auspices of the Accellera consortium [11]. It has not been submitted yet to IEEE for standardization.

Both the VHDL-AMS language and the Verilog-AMS language are mixed-signal HDLs. Both extend their respective digital HDL with new behavioral and structural language constructs and new simulation mechanisms. We give below a brief presentation of distinguished features of each language. More detailed insights will be given later when we discuss the case study.

Each language supports continuous-time behavior, but uses different formulations. VHDL-AMS provides a notation for describing DAEs in a fairly general way. The == operator and the way the quantities (bound to ports or free) are declared allow the designer to write his equations mathematically, and both implicit and explicit formulations are supported. Verilog-AMS provides a notation based on the so-called *probe-source network* formulation, that distinguishes between assignments of free quantities (with the operator =) and assignments of potential/flow quantities (with the operator <+) in the equations themselves with access functions. Both languages introduce the concept of a continuous-time, or analog, simulation kernel and both support the specification of initial conditions, the definition of piecewise-defined behavior (i.e., the definition of different regions of operations between which the model can dynamically switch during simulation), and can handle discontinuities.

Each language supports the description of networks as conservative-law and signal-flow networks. As such, they support the description and the simulation of multi-discipline systems at these two levels of abstraction. VHDL-AMS does not provide any automatic mechanism to insert mixed-signal interfaces in a model, although it provides all the required language constructs to do it manually. Verilog-AMS supports the automatic insertion of so-called *connect modules*.

Both languages have a canonical, tool independent, mixed-signal simulation cycle that defines how to simulate a mixed-signal description. VHDL-AMS has in addition a formal definition of how to initialize a mixed-signal model, while Verilog-AMS still lacks such a definition. Both languages support continuous-time analyses such as time-domain, DC, small-signal AC and noise analyses.

Due to historical reasons, a continuous-time only, or analog only, subset of Verilog-AMS is defined and referred to as Verilog-A [12]. It is very likely that this subset will eventually disappear as tools will start to support the full Verilog-AMS language. There is no analog subset for VHDL-AMS.

Verilog-AMS provides some kind of compatibility with the SPICE netlist format by defining so-called preferred primitive names, parameter names and port names for a limited number of SPICE elements. VHDL-AMS does not provide such kind of compatibility neither in the language definition nor in some standard library, although it has all the necessary language constructs to build a library of SPICE element models.

VHDL-AMS provides a fairly generic support for defining tolerances. It allows for grouping models unknowns into classes of tolerances, each class being given a different name. The language does not define how the tolerance class names are mapped to actual simulator tolerance values, so the mapping is tool dependent. Verilog-AMS, on the other hand, does associate numerical tolerance values to the model unknowns using SPICE-like `abstol` and `reltol` tolerance specifications.

Any VHDL-AMS design unit may be compiled separately and stored in a library. In addition, VHDL-AMS allows for a

clear separation between the interface of a model and its internal description and provides a mechanism to select the submodels to use through *configuration*. Both capabilities hence allow for much flexibility when it comes to model large complex hierarchical systems. Verilog-AMS models require that all textual information has to be available in the model to simulate, possibly by including it at appropriate places in the source code.

The EDA tools used for implementing and simulating the models are Advance MS from Mentor Graphics for VHDL-AMS models and AMS Simulator 2.0 from Cadence Design Systems for Verilog-AMS models. Both tools do not yet support the full language definitions so it is likely that the models presented in this paper could be written differently when the full power of each language becomes available.

III. AIRBAG CASE STUDY

This section describes the model of an airbag system [13] that mixes two time abstractions, namely discrete-time and continuous-time, five disciplines, namely mechanical, electrical, optical, thermal and chemical, and three connection semantics, namely conservative, continuous signal-flow, and digital signal-flow.

A. Airbag system description

Fig. 1 presents the synoptic view of the airbag system we are going to discuss. This figure acts as a reference throughout the rest of the paper. Each box references a submodel, with its name and its interface connectors. To each connector is associated a discipline, i.e. ME for Mechanical, EL for Electrical, TH for Thermal, OP for Optical, and a connection semantics, i.e. CS for Conservative-law, SF for Signal-Flow, and D for digital. For instance, the laser diode is a conservative system as far as its electrical and thermal behaviors are concerned, and a signal-flow model as far its optical behavior is concerned.

From a physical point of view, the airbag, located in the steering wheel, has to be inflated when a major deceleration of the car occurs (usually more than 4 G). To protect the driver from the car wreck, the 60 liters of the airbag envelope have to be filled with inert gas (N_2) in less than 5 milliseconds. Such a spectacular action can only be achieved by means of a violent chemical reaction which converts solid material into gas, initiated by the explosion of a propergol capsule, also located in the steering wheel.

From a systemic and chronological point of view, the acceleration of the car is modeled as a 10 KHz sine wave, described in the submodel `src_force` in Fig. 1. Due to frictional resistance and structural elasticity, the resulting force applied to the seismic mass of the accelerometer is a damped sine-wave, with a peak amplitude that exceeds 4G in case of a front car impact. The `src_volt` submodels are used to supply the top and bottom electrodes of the electrical part of the acceleration sensor model, called `accelerometer`, with a 1 MHz sine-shaped signal. Thanks to the `amplifier` submodel, the middle electrode delivers an amplified 1 MHz sine-shaped signal which amplitude is proportional to the acceleration. This signal is then continuously compared to an acceleration threshold to detect the car impact condition thanks to the `comparator` submodel. When the car acceleration exceeds the airbag threshold, the comparator outputs a 1 MHz pulsed signal. To be sure that the airbag is not untimely triggered due to a spurious perturbation, 10 periods of the comparator output signal are counted up before the airbag collision signal is generated (10 periods means the acceleration must exceed the airbag threshold for more than 10 µs). This task is performed by the `trigger` submodel that also converts the digital collision information, a simple rising edge, into its analog equivalent. The trigger output is then inverted by means of the `inverter` submodel that is based on the EKV transistor model. The CMOS inverter drives a laser-diode, described in the `laser_diode` submodel, which emits in turn the requested collision information (a falling edge) into an optical fiber, described in the `xmedium` submodel, as an emission of monochromatic light. The laser-diode being a very temperature sensitive device located near the CMOS inverter, we seized the opportunity to introduce a thermal network, described in the `th_network` submodel, which models thermo-electronic interaction between electrical and optical devices. After a small delay due to light propagation through the optical fiber, a pin photo-diode, described in the `photo_diode` submodel, extracts the propagated information at the other extremity of the fiber and uses the falling edge to actually trigger the airbag chemical reaction. The triggering of the chemical reaction is described in the `chemtrig` submodel, which would be the place for modeling the propergol explosion in a more realistic model. For now, the falling edge of the triggering signal propagated through the optical fiber directly acts as a firing event for the chemical reactions. Last but not least, the airbag inflation is modeled as a thermo dynamical kinetic process described in the `chemsys` submodel. Considering the airbag as an adiabatic and isotherm system, the inert gas N_2 needed to inflate the airbag follows the perfect gas law, and therefore the gas pressure and volume can be expressed in terms of evolution of the number of moles of N_2 as a function of time, provided by a set of three chemical kinetic equations**.**

Note that the medium used to propagate the command to inflate the airbag from the acceleration sensor to the actuator relies on the use of an optical fiber, which can not actually be found in any real car. However, in order to experience the modeling and signal-flow interconnecting capabilities of the studied HDLs, we made the assumption that the accelerometer sensor is located far from the airbag actuator.

In the rest of the section, we compare the two HDLs on meaningful parts of source codes. Each comparison starts with a description of the modeled functionality. Then the two corresponding source codes that realize the functionality are presented and discussed.

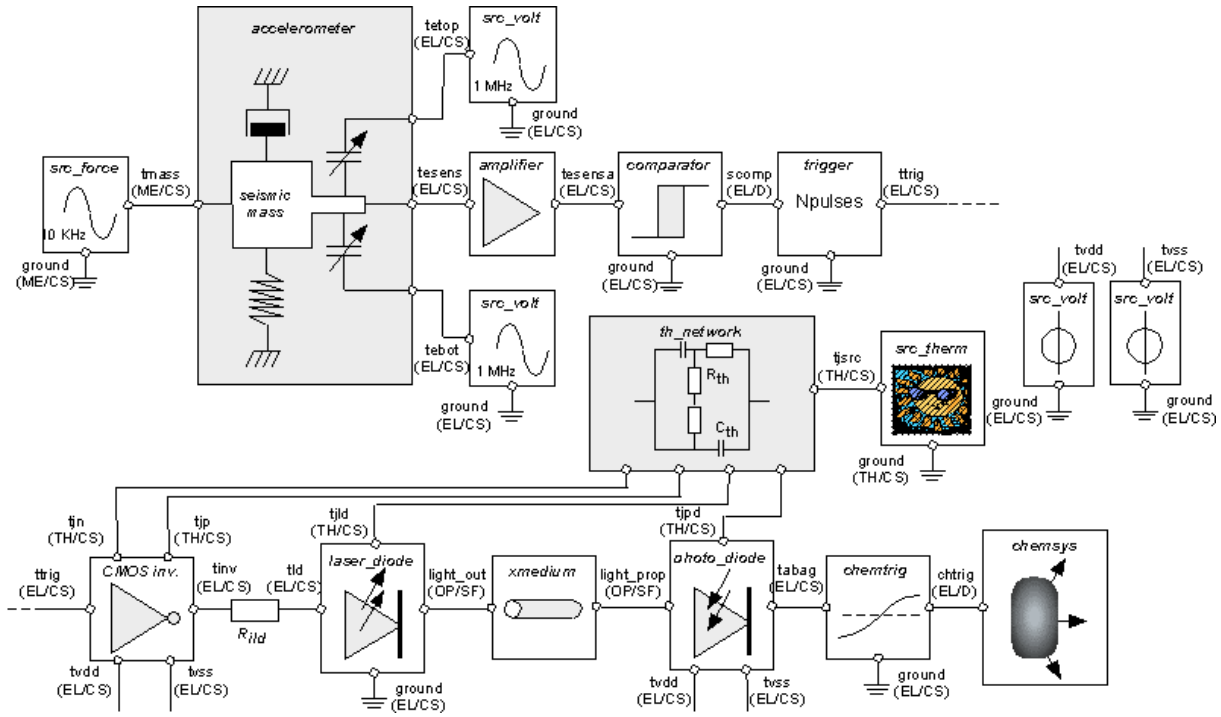

Fig. 1. Synoptic view of the airbag system.

B. The `accelerometer` submodel

The accelerometer submodel is an interesting example that mixes two disciplines, mechanics and electronics. The accelerometer is interfaced with the upstream acceleration stimulus sine-wave generator and with the downstream analog electrical comparator. Therefore, as both a mechanical and electrical device, the accelerometer is subject to conservative general Kirchhoff's laws GPL and GFL that state energy conservation.

The accelerometer has the structure presented in Fig. 2 [14].

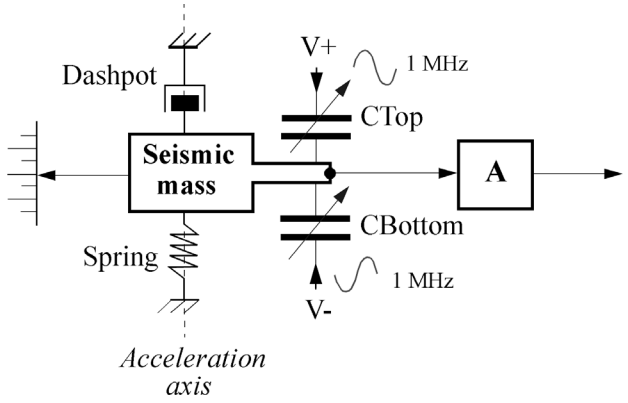

Fig. 2. Synoptic view of the accelerometer.

Acceleration sensing is performed in two steps. First, the car acceleration is transformed into a mechanical displacement, thanks to the use of a primary transducer composed of a pendulous or tethered seismic mass. Second, the mechanical displacement is converted into an electrical signal by a secondary transducer, which can rely on piezoelectric or capacitive properties. In the primary transducer, the micro flexural structure can be modeled as a damped harmonic oscillator, i.e., as a 2nd order differential equation such as:

$$F(t) = M\frac{d^2x}{dt^2} + D\frac{dx}{dt} + kx \tag{1}$$

where F is the force applied to the seismic mass, x is the displacement of the mass M, D is the damping coefficient and k is the spring stiffness. The secondary transducer uses the seismic mass as the middle plate of a differential capacitance circuit. The displacement x of the seismic mass modifies the gap between plates and hence the differential capacitance values according to the following equations:

$$C_{top} = \frac{\varepsilon A}{d-x} \quad C_{bottom} = \frac{\varepsilon A}{d+x} \tag{2}$$

where ε is the vacuum permittivity constant, A the surface of the capacitance plate, and d the initial gap between two plates. When a 1 MHz sine-shaped voltage is applied to the top and the bottom electrodes, the currents through the two capacitances follow the electro kinetic equations:

$$I_{top} = C_{top}.\frac{dV_{top}}{dt} \qquad I_{bottom} = C_{bottom}.\frac{dV_{bottom}}{dt} \tag{3}$$

(1) to (3) describe the overall ideal electro-mechanical behavior of the accelerometer. A more realistic model would include other effects such as nonlinear capacitors or electrostatic forces.

1) VHDL-AMS description

The VHDL-AMS source code of the accelerometer is given in Fig. 3. To implement the set of multi-discipline differential equations, one has to make reference to the definitions related to electrical and mechanical systems (lines 1 to 4). These packages contain all the information needed to declare the two conservative translational ports `tmass` and `tmref` (line 16), and the three electrical ports `tetop`, `temid`, and `tebot` (line 17) and to use physical constants such as `EPS0`, the permittivity of the vacuum (lines 36 and 37). The default values of generic parameters (lines 9 to 14) make use of predefined scale factors from the `energy_systems` package to improve readability. The name of the library `ieee_proposed` means that the definitions contained in these packages are not completely stable yet. They should be however defined as some IEEE standard in a near future.

The model interface is described in the `entity` declaration (lines 6 to 18) and is decomposed into the specification of generic parameters (lines 8 to 14) and of the interface ports (lines 16 and 17). The generic parameters make the submodel reusable in other models than only the airbag system model. The actual values of mechanical and geometrical parameters may be changed for each specific instance of the accelerometer model. The port interface defines conservative-law mechanical and electrical connection points or terminals.

The accelerometer behavior is defined in a separate architecture body called `bhv` (lines 20 to 41). Lines 22 to 28 declare *quantities*, which correspond in VHDL-AMS to the unknowns of the system of equations to be solved by the analog solver. In the lines 22 to 24, a number of *branch quantities* are declared. Branch quantities are defined between two ports (terminal in VHDL-AMS) and represent it's across or through aspects. The first across/through set of branch quantities (line 22) belongs to the mechanical discipline, while the other two sets belong to the electrical discipline. These quantities are used to express equations (1) to (3). Then, a number of so-called *free quantities*, that is quantities not bound to any terminal, are declared (lines 26 to 28). These quantities are mainly used to break down complex relationships into more manageable and understandable pieces.

VHDL-AMS provides a fairly general notation for expressing DAEs based on the so-called *simultaneous statements*. In essence, simultaneous statements denote the constitutive equations in a model that will be gathered into a system of equations to be solved. From the user viewpoint, VHDL-AMS does not make any difference between explicit and implicit equations. The simple simultaneous statement simply states that, once the values of the unknowns (quantities) are computed at some time point, the evaluation of the left hand side expression of the "==" statement minus the right hand side expression gives a value close to zero, within the defined tolerances.

The accelerometer behavior is described using eight simultaneous statements (lines 31 to 40). Note the use of the `'dot` attribute to denote a first order time derivative of the quantity prefix. One could have rather selected the velocity as the across mechanical quantity instead of the position, but this would have enforced writing (1) with an integral term. Also, the use of the intermediate quantity `cd_vel` to hold the velocity allows to only use first order time derivatives. This is usually preferred as higher order time derivatives are usually computed with less accuracy. Anyway, the notation `cd_pos'dot'dot` is perfectly legal. As one can see, the VHDL-AMS equations of lines 31 to 40 perfectly match the physical equations (1) to (3).

```
(1)   library ieee_proposed;
(2)     use ieee_proposed.energy_systems.all;
(3)     use ieee_proposed.electrical_systems.all;
(4)     use ieee_proposed.mechanical_systems.all;
(5)
(6)   entity cap_sensor is
(7)     generic (
(8)       -- mechanical properties
(9)       M : mass      := 0.16*NANO;          -- seismic mass
(10)      D : damping   := 4.0*MICRO;          -- damping coefficient
(11)      K : stiffness := 2.6455;             -- spring stiffness
(12)      -- geometrical properties
(13)      A : real      := 2.0*MICRO*110.0*MICRO;  -- capacitor area
(14)      D0: real      := 1.5*MICRO);         -- initial position
(15)    port (
(16)      terminal tmass, tmref      : translational;
(17)      terminal tetop, temid, tebot: electrical);
(18)  end entity cap_sensor;
(19)
(20)  architecture bhv of cap_sensor is
(21)    -- branch quantities
(22)    quantity cd_pos across cd_force through tmass to tmref;
(23)    quantity vtm    across itm      through tetop to temid;
(24)    quantity vbm    across ibm      through tebot to temid;
(25)    -- free quantities
(26)    quantity cd_vel: velocity;       -- comb drive velocity
(27)    quantity dtm, dbm: displacement; -- comb drive displacements
(28)    quantity ctm, cbm: capacitance;  -- capacitances
(29)  begin
(30)    -- compute displacement of comb drive
(31)    cd_vel == cd_pos'dot;
(32)    cd_force == K*cd_pos + D*cd_vel + M*cd_vel'dot;
(33)    dtm == D0 + cd_pos;
(34)    dbm == D0 - cd_pos;
(35)    -- compute change in capacitances
(36)    ctm == A*EPS0/dtm;
(37)    cbm == A*EPS0/dbm;
(38)    -- compute generated current
(39)    itm == ctm*vtm'dot;
(40)    ibm == cbm*vbm'dot;
(41)  end architecture bhv;
```

Fig. 3. VHDL-AMS model of the accelerometer.

2) Verilog-AMS description

The Verilog-AMS source code for the accelerometer is

given in Fig. 4. The first two lines include statements that define the `kinematic` and `electrical` disciplines as well as physical constants such as `P_EPS0`, the permittivity of vacuum. As the constant is defined as a macro in the "`constants.h`" file, it has to be used with the back tick prefix, as shown in lines 33 and 34.

The model interface and the model body are included in the same `module` statement. The interface ports are defined in three steps: the port names (line 4), the port directions (line 6), and the port disciplines (lines 8 to 11). As all interface ports belong to a discipline, they are declared of mode `inout` (this declaration is not actually required as `inout` is the default mode). The default values of generic parameters (lines 14 to 19) make use of predefined scale factors to improve readability. Verilog-AMS is case sensitive so the "m" scale factor means 10^{-3} while the "M" scale factor means 10^{6}! Lines 19 and 20 declare a number of real variables that hold intermediate results.

Verilog-AMS encapsulates the description of continuous-time behavior in so-called *analog blocks*. The language imposes to have only one analog block in a module. For that reason, the accelerometer behavior is defined in a single `analog` statement containing a sequence of several statements (lines 24 to 42). All the statements are executed sequentially, and their order is relevant. The contribution statements that define the currents in the capacitors have to be at the end of the block (lines 40 and 41) otherwise the current values would not be correct. Simple assignments to integer or real variables are achieved using the "=" sign, while two forms for expressing continuous-time behavior are provided.

The first form, used in this submodel, is the so-called *contribution statement*, with the contribution operator "`<+`" that may only affect quantities referred to by *access functions*. The second (implicit) form is called *indirect branch assignment* and allows for describing equations using the state-space formulation.

In the model, it is possible to use the contribution statement since all equations may be written in explicit form. The access functions for the kinematic discipline are "`Pos`" for the potential/across quantity and "`F`" for the flow/through quantity. There is also a `kinematic_v` discipline available where the potential/across quantity is a velocity, but the model uses the position instead for the same reasons mentioned in the description of the VHDL-AMS model. Quantities like "`V(tetop, temid)`" (a potential) are qualified as *probes*, as they appear at the right-hand side of the contribution statement, while other quantities, such as "`I(tetop, temid)`" (a flow) would have been qualified as *sources* as they appear on the left-hand side. The same quantity cannot be both a source and a probe. The "`<+`" sign also clearly indicates that the operation is *additive*. The `ddt` operator computes the first order time derivative of an *expression*. This contrasts with the VHDL-AMS `'dot` attribute that can only be applied to a quantity. There are however two restrictions imposed by the Cadence implementation on the use of the `ddt` operator. First, nesting ddt operators is not allowed and second, this operator may only be applied to nodes. This forces to declare additional nodes in the model (lines 10 and 11) and to add contribution statements (lines 26 to 29, and 36 to 41). It is not clear, though, whether these restrictions are fundamental to the Verilog-AMS language definition or only a consequence of the Cadence implementation.

```
(1)   `include "disciplines.h"
(2)   `include "constants.h"
(3)
(4)   module cap_sensor (tmass, tmref, tetop, temid, tebot);
(5)
(6)     inout tmass, tmref, tetop, temid, tebot;
(7)
(8)     kinematic tmass, tmref;
(9)     electrical tetop, temid, tebot;
(10)    electrical cd_vel, cd_accel;
(11)    electrical vdiff_tm, vdiff_bm, vdiff_dtm, vdiff_dbm;
(12)
(13)    // mechanical properties
(14)    parameter real M  = 0.16n;    // seismic mass
(15)    parameter real D  = 4u;       // damping coefficient
(16)    parameter real K  = 2.6455;   // spring stiffness
(17)    // geometrical properties
(18)    parameter real A  = 220f;     // capacitor area
(19)    parameter real D0 = 1.5u;     // initial position
(20)
(21)    real cd_pos;
(22)    real dtm, dbm, ctm, cbm;
(23)
(24)    analog begin
(25)      // compute displacement of comb drive
(26)      cd_pos = Pos(tmass);
(27)      V(cd_vel) <+ ddt(Pos(tmass));
(28)      V(cd_accel) <+ ddt(V(cd_vel));
(29)      F(tmass, tmref) <+ K*cd_pos + D*V(cd_vel) + M*V(cd_accel);
(30)      dtm = D0 + cd_pos;
(31)      dbm = D0 - cd_pos;
(32)      // compute change in capacitances
(33)      ctm = A*`P_EPS0/dtm;
(34)      cbm = A*`P_EPS0/dbm;
(35)      // compute generated current
(36)      V(vdiff_tm) <+ V(tetop, temid);
(37)      V(vdiff_bm) <+ V(tebot, temid);
(38)      V(vdiff_dtm) <+ ddt(V(tetop, temid));
(39)      V(vdiff_dbm) <+ ddt(V(tebot, temid));
(40)      I(tetop, temid) <+ ctm*V(vdiff_dtm);
(41)      I(tebot, temid) <+ cbm*V(vdiff_dbm);
(42)    end
(43)
(44)  endmodule // cap_sensor
```

Fig. 4. Verilog-AMS model of the accelerometer.

C. *The* `trigger` *submodel*

The trigger submodel has mostly a digital behavior that can be conceptually modeled as a single process, sensitive to the digital comparator output changes. As the output signal has to be analog for interconnection reasons with the CMOS inverter based on EKV MOST models, some D/A interface must be provided in the model. A 50 KHz clock, internal to the trigger submodel, serves as a duty cycle and is anded with the

comparator output signal, according to Fig. 5, to count up to 10 periods. When 10 periods have been counted up, the collision output signal toggles.

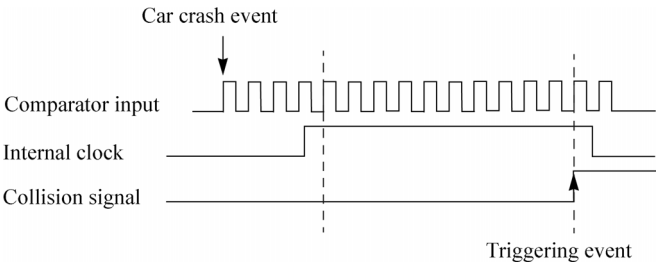

Fig. 5. Counting 10 pulses of the comparator to produce the collision signal.

1) VHDL-AMS Description

The VHDL-AMS source code for the trigger is given in Fig. 6. The model first references predefined packages that include the definitions of the logic value system to be used for digital signals, essentially the `std_logic` type, and the definitions related to the electrical discipline, essentially the `electrical` nature (lines 1 to 5).

The model interface declares both event-driven (line 15) and continuous-time (line 16) ports. The latter functionally represent model outputs, although nothing in the interface declaration actually stresses that.

The trigger behavior is defined in a separate architecture body called `bhv` (lines 20 to 47). Local declarations include the digital internal clock signal `intclk` (line 22), the discrete-time representation of the output (line 23) and two branch quantities that represent an ideal source connected to the terminal ports `tp` and `tm` (line 25). The model body is decomposed in two concurrent processes and one simultaneous statement. The first process (line 29) generates the internal clock, the second process (lines 31 to 43) is sensitive to an event on either the internal clock or on the digital input `din` and counts until it reaches the requested number of pulses. Then, the discrete-time signal `sout` is inverted. The use of this real-valued signal is useful to detect the triggering event accurately. The simultaneous statement (line 45) defines the ideal source as a voltage source whose value is the analog equivalent of the internal signal `sout`, but with ramping transitions between states. The rise and fall times of the ramps are identical and are defined by the generic parameter `TT`. Note that it is very important to define a non-default initial value for signal `sout` to avoid convergence problems during the computation of the quiescent point of the model. In fact, a real-valued signal object gets a default initial value of `real'left`, that is the largest negative floating-point value, and this is usually not a good value to start iterations at DC. This explains why the signal `sout` has an initial value of 0.0 (line 23). As a general rule, all real-valued signals involved in simultaneous statements should have a non-default value equal to 0.0 or any other meaningful value.

```
(1)   library ieee;
(2)     use ieee.std_logic_1164.all;
(3)
(4)   library ieee_proposed;
(5)     use ieee_proposed.electrical_systems.all;
(6)
(7)   entity trigger is
(8)     generic (
(9)       VOAMPL : voltage := 2.5;      -- output voltage amplitude
(10)      ICLKPER: time    := 20 us;    -- internal clock period
(11)      NPULSES: natural := 10;       -- #pulses to count
(12)      TT     : real    := 100.0e-9  -- output transition time
(13)    );
(14)    port (
(15)      signal   din   : in std_logic;
(16)      terminal tp, tm: electrical
(17)    );
(18)  end entity trigger;
(19)
(20)  architecture bhv of trigger is
(21)
(22)    signal intclk: std_logic := '0';
(23)    signal sout  : real := 0.0;
(24)
(25)    quantity vout across iout through tp to tm;
(26)
(27)  begin
(28)
(29)    iclkgen: intclk <= not intclk after ICLKPER/2;
(30)
(31)    process
(32)      variable count: natural := 0;
(33)    begin
(34)      wait on din, intclk;
(35)      if din'event and din = '1' and intclk = '1' then
(36)        count := count + 1;
(37)        if count = NPULSES then
(38)          sout <= 1.0 - sout;
(39)        end if;
(40)      end if;
(41)      if intclk'event and intclk = '0' then  -- reset
(42)        count := 0;
(43)      end if;
(44)    end process;
(45)    vout == VOAMPL*sout'ramp(TT);
(46)
(47)  end architecture bhv;
```

Fig. 6. VHDL-AMS model of the trigger.

2) Verilog-AMS Description

The Verilog-AMS source code for the trigger is given in Fig. 7. The model starts with a timescale directive which defines the time unit and the time precision used in the model. In our case, the internal clock period `ICLKPER` is defined as a multiple of 100 ns, while the time precision is 1 ns. A second include statement essentially makes the definition of the `electrical` discipline available to the model.

The model interface declares both event-driven (lines 5 and 7) and continuous-time (lines 6 and 8) ports. The discipline of the digital port `din` is actually implicitly defined as being

`logic` through the `wire` net type. Then come a number of generic parameters whose actual values may be changed for each instance of the model (lines 10 to 13), and a number of local declarations (lines 15 to 17). One can see that there is no real difference between variables and signals in Verilog-AMS. The `reg` type variables have logic values (namely 0, 1, X or Z) and may only be assigned from within a process.

The model body is decomposed into three processes and one analog statement. The `initial` process (lines 19 to 23) is executed only once at the beginning of the simulation to initialize some objects, while the two other `always` processes (lines 25 and 27 to 34) execute cyclically as soon as an event occurs on sensitive signals, namely `intclk` for the first one and `din` and `intclk` for the second one. One process is dedicated to the internal clock generation (line 25) and the other process actually counts the number of input pulses and triggers the output signal when the requested number of pulses is reached. As the output port `tp` is in the continuous-time domain, some D/A interface must be used to convert the internal digital variable `sout` that exhibits steep transitions to an analog equivalent that does not. This is done with an analog process (lines 36) that uses the `transition` filter to generate ramps on the output voltage. The rise and fall times are identical and are controlled by the parameter `TT`.

```
(1)   `timescale 100ns/1ns
(2)   `include "disciplines.h"
(3)
(4)   module trigger (din, tp);
(5)     input din;
(6)     inout tp;
(7)     wire din;
(8)     electrical tp;
(9)
(10)    parameter real VOAMPL  = 2.5;  // output voltage amplitude
(11)    parameter      ICLKPER = 200;  // internal clock period
(12)    parameter real NPULSES = 10;   // #pulses to count
(13)    parameter real TT      = 100n; // output transition time
(14)
(15)    reg intclk;
(16)    integer count;
(17)    real sout;
(18)
(19)    initial begin
(20)      intclk = 0;
(21)      count = 0;
(22)      sout = 0;
(23)    end
(24)
(25)    always #(ICLKPER/2) intclk = !intclk;
(26)
(27)    always @(posedge din or negedge intclk)
(28)    begin
(29)      if ((din == 1) && (intclk == 1)) begin
(30)        count = count + 1;
(31)        if (count == NPULSES) sout = 1.0 - sout;
(32)      end
(33)      else if (intclk == 0) count = 0;
(34)    end
(35)
(36)    analog V(tp) <+ VOAMPL*transition(sout, 0, TT);
(37)
(38)  endmodule // trigger
```

Fig. 7. Verilog-AMS model of the trigger.

D. The CMOS `inverter` *submodel*

The CMOS inverter is composed of one nMOS and one pMOS transistor and is connected to its direct environment as shown in Fig. 8.

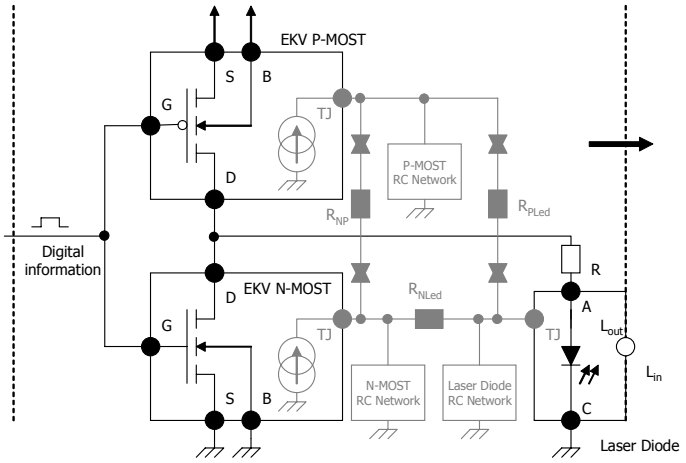

Fig. 8. The CMOS inverter and its direct environment.

In the airbag system, the nMOS transistor has electrical connections with the trigger output, with the pMOS transistor to form the inverter, and with the laser diode. When located on the same substrate, thermo-electronic interactions take place between these devices. Heat diffusion through the corresponding materials can be modeled by different means [15][16]. In our model, we do this by sourcing dissipated power into a thermal RC network [16], which represents the material properties of the different layers (thermal resistances and capacitances of the heat sink and the package). The thermal network is composed of thermal capacitors, resistors and an ambient temperature generator. All the devices are thermally interconnected through coupling thermal resistances: R_{NLed}, R_{PLed}, and R_{NP}.

The pMOS and nMOS transistor behaviors are described using the EKV MOST model, an accurate analytical model for deep submicron designs [17][18]. It is a charge-based compact model that consistently describes effects on charges, transcapacitances, drain current and transconductances in all regions of operation of the MOSFET transistor (weak, moderate, strong inversion) as well as conduction to saturation. The modeled effects include all the essential effects present in sub micron technologies. Several electrical parameters are highly dependent on temperature, namely the threshold voltage, the mobility, the thermal voltage, etc... Their respective temperature variations are taken into account by appropriate coefficients in the model equations [19].

1) VHDL-AMS Description

The VHDL-AMS source code for the CMOS inverter is given in Fig. 9. It is a structural model that instantiates two components: one pMOS transistor called `PMOS` (lines 22 to 36) and one nMOS transistor called `NMOS` (lines 38 to 52). Both the generic parameters and the port associations use the named association mechanism for improved readability.

It is possible to develop a single transistor model that is valid for both pMOS and nMOS transistors. The trick is to change signs of quantities and parameters appropriately. In this paper, we present a simplified version of the EKV MOST model as the full version would have needed several pages of code. Fig. 10 gives the simplified VHDL-AMS model of one transistor. The `MTYP` generic parameter allows for defining the type of the MOS transistor and also the sign of some relevant voltages and parameters (defined in line 8, used in lines 54 and 68). Note that some actual parameters in the MOS instances must anyway have the right sign (e.g., `VT0` in Fig. 10). The interface ports are the four standard electrical pins of a MOSFET transistor, plus an additional thermal pin to account for dynamic thermal exchanges between the transistor and its environment (line 25). The order in which terminals are specified in a branch quantity declaration defines the direction of the flow. Considering the thermal port, the temperature `temp` is measured between the port and the thermal reference (line 39), while the heat `gpower` is flowing out the device from the thermal reference to the port (line 38). This way, the thermal interaction is really bi-directional and the self-heating behavior of the device is properly taken into account (line 69).

The electrical behavior of the EKV MOST model is actually procedural so it is more efficient to use the sequential statements proposed by VHDL-AMS. The simultaneous `procedural` statement could be used, but, as it is not yet supported in the Mentor tool, we use a function instead, namely the `f_id` function, to implement the computation of the drain current (lines 47 to 65). The equation of the drain current is then implemented in a single simultaneous statement with the appropriate signs for the function arguments to account for the actual model type (line 68). Note that all terminal potentials are defined relatively to the bulk terminal, a specificity of the EKV MOST model.

```
(1)   library ieee_proposed;
(2)     use ieee_proposed.energy_systems.all;
(3)     use ieee_proposed.electrical_systems.all;
(4)     use ieee_proposed.thermal_systems.all;
(5)
(6)   entity cmos_inv is
(7)     generic (
(8)       WN: real := 15.0*MICRO;
(9)       LN: real := 0.15*MICRO;
(10)      WP: real := 15.0*MICRO;
(11)      LP: real := 0.15*MICRO
(12)    );
(13)    port (
(14)      terminal tin, tout, tvdd, tvss: electrical;
(15)      terminal tjn, tjp          : thermal
(16)    );
(17)  end entity cmos_inv;
(18)
(19)  architecture str of cmos_inv is
(20)
(21)  begin
(22)    PMOS: entity work.mos(ekv_simple)
(23)          generic map (
(24)            MTYP => -1.0,
(25)            WEFF => WP,
(26)            LEFF => LP,
(27)            VT0  => -0.4,
(28)            TCV  => -1.5*MILLI
(29)          )
(30)          port map (
(31)            td => tout,
(32)            tg => tin,
(33)            ts => tvdd,
(34)            tb => tvdd,
(35)            tj => tjp
(36)          );
(37)
(38)    NMOS: entity work.mos(ekv_simple)
(39)          generic map (
(40)            MTYP => 1.0,
(41)            WEFF => WN,
(42)            LEFF => LN,
(43)            VT0  => 0.4,
(44)            TCV  => 1.5*MILLI
(45)          )
(46)          port map (
(47)            td => tout,
(48)            tg => tin,
(49)            ts => tvss,
(50)            tb => tvss,
(51)            tj => tjn
(52)          );
(53)  end architecture str;
```

Fig. 9. VHDL-AMS structural model of the CMOS inverter.

```
(1)   library ieee; use ieee.math_real.all;
(2)   library ieee_proposed;
(3)     use ieee_proposed.energy_systems.all;
(4)     use ieee_proposed.electrical_systems.all;
(5)     use ieee_proposed.thermal_systems.all;
(6)   entity mos is
(7)     generic (
(8)       MTYP : real := 1.0;         -- NMOS: 1.0, PMOS: -1.0
(9)       -- geometrical parameters
(10)      WEFF : real := 1.0*MICRO;    -- effective channel width
(11)      LEFF : real := 0.15*MICRO;   -- effective channel length
(12)      -- threshold voltage and substrate body effect parameters
(13)      VT0  : real := 0.4;         -- long channel thresh. voltage (NMOS!)
(14)      PHI  : real := 0.97;        -- bulk Fermi potential
(15)      GMA: real := 0.71;          -- body effect parameter
(16)      -- mobility parameters
(17)      KP   : real := 453.0*MICRO;  -- transconductance parameter
(18)      THETA: real := 50.0*MILLI;   -- mobility reduction coefficient
(19)      -- temperature coefficients
(20)      TCV  : real := 1.5*MILLI;    -- temp. coef, of thres. voltage
(21)      BEX  : real := -1.5         -- temp. coef. of trans. parameter
(22)    );
(23)    port (
(24)      terminal td, tg, ts, tb: electrical;
(25)      terminal tj            : thermal
(26)    );
(27)  end entity mos;
(28)
(29)  architecture ekv_simple of mos is
(30)    constant KOQ    : real := K/Q;
(31)    constant TEMPREF: real := 300.15;
(32)    -- electrical branch quantities
(33)    quantity vg across tg to tb;
```

```
(34)      quantity vd across td to tb;
(35)      quantity vs across ts to tb;
(36)      quantity ids through td to ts;
(37)      -- thermal branch quantities
(38)      quantity gpower through thermal_ref to tj;
(39)      quantity temp   across  tj to thermal_ref;
(40)
(41)      function i_v (constant v: real) return real is
(42)        variable x: real;
(43)      begin
(44)        return (log(1.0 + 0.5*exp(v)))**2;
(45)      end function i_v;
(46)
(47)      function f_id (temp, vg, vs, vd: real) return real is
(48)        variable id, vt, ratio, eg, egref: real;
(49)        variable vto_th, kp_th: real;
(50)        variable vgprime_0, vgprime, vp, iff, irr, beta, n: real;
(51)      begin
(52)        vt := KOQ*temp + 1.0e-6;
(53)        ratio := abs(temp/TEMPREF + 1.0e-6);
(54)        vto_th := MTYP*(VT0 - TCV*(temp - TEMPREF));
(55)        kp_th := KP*(ratio**BEX);
(56)        vgprime_0 := vg - vto_th + PHI + GMA*sqrt(PHI);
(57)        vgprime:=0.5*(vgprime_0+sqrt(vgprime_0*vgprime_0+1.0e-3));
(58)        vp := vgprime - PHI
(59)      - GMA*(sqrt(vgprime + 0.25*GMA*GMA) - 0.5*GMA);
(60)        iff := i_v((vp - vs)/vt);
(61)        irr := i_v((vp - vd)/vt);
(62)        beta := kp_th*(WEFF/LEFF)*(1.0/(1.0 + THETA*vp));
(63)        n := 1.0;
(64)        return 2.0*n*beta*vt*vt*(iff - irr) + 1.0e-10;
(65)      end function f_id;
(66)
(67)    begin
(68)      ids == MTYP*f_id(temp, MTYP*vg, MTYP*vs, MTYP*vd);
(69)      gpower == abs(ids*(vd - vs));
(70)    end architecture ekv_simple;
```

Fig. 10. VHDL-AMS model of a simple EKV MOS model.

2) Verilog-AMS Description

The Verilog-AMS source code for the CMOS inverter is given in Fig. 11. It is also a structural model that instantiates two MOSFET transistor components. Both the generic parameter and the port associations use the named association mechanism for improved readability. Note that the include statement in line 3 should be typically inserted in the top-level model of the airbag system only. Including the same file in several places is a common mistake and compiler directives are usually included to prevent this.

Fig. 12 gives the Verilog-AMS model of the simplified EKV MOST model. The use of an analog block is natural here as the EKV model is inherently procedural (lines 32 to 53). Local variables to hold terminal voltages are used in the block (lines 33 to 35). This is done to avoid scattering access functions throughout the code and also to use the right signs depending on the MOS type. A utility macro "`I_V`" is defined to realize a smooth interpolation function (line 4). The macro is expanded in-line in the code with the appropriate parameter substitution (lines 45 and 46). Note the use of the predefined `limexp` function, which implements an exponential function whose value is limited during the solution iterations. This is very useful to prevent arithmetic overflow in intermediate computations that may typically occurs in semiconductor models. At the end of the analog block, the computed drain current `id` is assigned as a flow contribution between the `td` and `ts` electrical terminals (line 50), and the self-heating heat is assigned as a flow contribution to the thermal terminal `tj` through the `Pwr()` access function (line 52). The flow is defined from the thermal ground, as declared in lines 9 and 10 to the terminal `tj`.

```
(1)     `include "disciplines.h"
(2)
(3)     `include "mos_ekv_simple.v"
(4)
(5)     module cmos_inv (tin, tout, tjn, tjp, tvdd, tvss);
(6)
(7)       inout tin, tout, tjn, tjp, tvdd, tvss;
(8)       electrical tin, tout, tvdd, tvss;
(9)       thermal tjn, tjp;
(10)
(11)      parameter real WP = 60u;
(12)      parameter real WN = 30u;
(13)      parameter real LP = 0.15u;
(14)      parameter real LN = 0.15u;
(15)
(16)      mos_ekv         #(.MTYP(-1.0),
(17)                      .WEFF(WP),
(18)                      .LEFF(LP),
(19)                      .VT0(-0.4),
(20)                      .TCV(-1.5m))
(21)              mp      (.td(tout),
(22)                      .tg(tin),
(23)                      .ts(tvdd),
(24)                      .tb(tvdd),
(25)                      .tj(tjp));
(26)
(27)      mos_ekv         #(.MTYP(1.0),
(28)                      .WEFF(WN),
(29)                      .LEFF(LN),
(30)                      .VT0(0.4),
(31)                      .TCV(1.5m))
(32)              mn      (.td(tout),
(33)                      .tg(tin),
(34)                      .ts(tvss),
(35)                      .tb(tvss),
(36)                      .tj(tjn));
(37)    endmodule // cmos_inv
```

Fig. 11. Verilog-AMS structural model of the CMOS inverter.

```
(1)     `include "disciplines.h"
(2)     `include "constants.h"
(3)
(4)     `define I_V(v) pow(ln(1.0 + 0.5*limexp(v)),2)
(5)
(6)     module mos_ekv (td, tg, ts, tb, tj);
(7)       inout td, tg, ts, tb, tj;
(8)       electrical td, tg, ts, tb;
(9)       thermal tj, th_gnd;
(10)      ground th_gnd;
(11)
(12)      parameter real MTYP  = 1.0;      // NMOS: 1.0, PMOS: -1.0
(13)      // geometrical parameters
(14)      parameter real WEFF  = 1.0e-6;   // effective channel width
(15)      parameter real LEFF  = 0.15e-6;  // effective channel length
(16)      // threshold voltage and substrate body effect parameters
(17)      parameter real VT0   = 0.4;      // long channel threshold voltage
(18)      parameter real PHI   = 0.97;     // bulk Fermi potential
```

```
(19)      parameter real GMA = 0.71;     // body effect parameter
(20)      // mobility parameters
(21)      parameter real KP    = 453.0e-6; // transconductance parameter
(22)      parameter real THETA = 50.0e-3;  // mobility reduction coefficient
(23)      // temperature coefficients
(24)      parameter real TCV   = 1.5e-3;   // temp. coef, of threshold voltage
(25)      parameter real BEX   = -1.5;     // temp. coef. of transcond. param.
(26)
(27)      real vt, temp, tempref, ratio, eg, egref;
(28)      real vto_th, kp_th, vgprime_0, vgprime, vp;
(29)      real vg, vs, vd;
(30)      real iff, irr, beta, n, id, gpower;
(31)
(32)      analog begin
(33)        vg = MTYP*V(tg, tb);
(34)        vs = MTYP*V(ts, tb);
(35)        vd = MTYP*V(td, tb);
(36)        tempref = 300.15; temp = Temp(tj);
(37)        vt = `P_K*temp/`P_Q + 1.0e-6;
(38)        ratio = abs(temp/tempref + 1.0e-6);
(39)        vto_th = MTYP*(VT0 - TCV*(temp - tempref));
(40)        kp_th = KP*pow(ratio,BEX);
(41)        vgprime_0 = vg - vto_th + PHI + GMA*sqrt(PHI);
(42)        vgprime =0.5*(vgprime_0+sqrt(vgprime_0*vgprime_0+1.0e-3));
(43)        vp = vgprime - PHI
(44)             - GMA*(sqrt(vgprime + 0.25*GMA*GMA) - 0.5*GMA);
(45)        iff = `I_V((vp - vs)/vt);
(46)        irr = `I_V((vp - vd)/vt);
(47)        beta = kp_th*(WEFF/LEFF)*(1.0/(1.0 + THETA*vp));
(48)        n = 1.0;
(49)        id = MTYP*2.0*n*beta*vt*vt*(iff - irr) + 1.0e-10;
(50)        I(td, ts) <+ id;
(51)        gpower = abs(id*(vd - vs));
(52)        Pwr(th_gnd, tj) <+ abs(id*(vd - vs));
(53)      end
(54)    endmodule // mos_ekv
```

Fig. 12. Verilog-AMS model of a simple EKV MOS model

E. The `optical link` *submodel*

An optical fiber provides numerous advantages over electrical interconnection: signals degrade less, there is less interference, and lower-power transmitters can be used instead of the high-voltage electrical transmitters needed for copper wire. From a modeling point of view, optical fibers obey signal-flow rules. An optical fiber is a device with infinite input impedance, and null output impedance, and its dynamic characteristics do not depend on the applied load. Moreover, the quantity of light produced by a laser diode remains the same, whatever the number of optical fibers connected to it. To put emphasis on the solutions provided by VHDL-AMS and Verilog-AMS to deal with signal-flow modeling, we now detail how the negative edge of the collision signal is converted into monochromatic light, propagated with the optical fiber transmission medium, and converted back into an electrical signal with a photodiode.

From an electrical point of view, the anode of the laser diode is electrically connected to a current limiting resistor, which other end is connected to the inverter output, as shown in Fig. 13 [20]. The cathode is directly connected to electrical ground. The intrinsic behavior of the diode is modeled by the classical equation for which the current is exponentially dependent on the potential V_d [21]. If necessary, series and leakage resistors could also be added, for improved modeling accuracy.

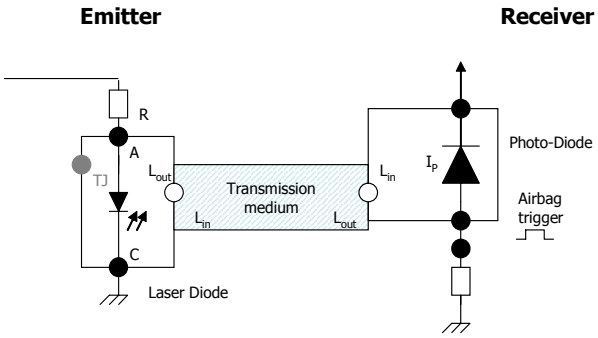

Fig. 13. The optical link.

From an optical point of view [20], the laser diode aims at producing light with controlled light power in relation to its forward current and wavelength. To obtain the laser effect, the laser diode needs a threshold current I_{th} which is temperature dependent. If the forward current is less than the threshold current, no light is emitted. If the forward current is higher then the relation between the light output power and the forward current is linear with a ratio slope η. The dynamic behavior of the laser diode is assumed to be the response as a first order low pass filter with time constant τ as formulated in the following Laplace transfer function:

$$\frac{L_{out}}{L_{in}} = \frac{1}{1+\tau \cdot s} \tag{4}$$

where L_{in} is the light power calculated. L_{out} gives the output power. The laser diode is very sensitive to temperature, and the ratio between absorbed power and emitted light power is low. The exceeding power is transformed into heat. To model the temperature evolution, we have added a thermal terminal to the laser diode so that the temperature of the device can change with the variation of the output value. We assume a linear variation of I_{th} and η with respect to the temperature as a good approximation within an appropriate temperature interval (about 70°C).

A more accurate model would take into account the produced wavelength, electromagnetic modes and the associated opto-geometrical aspects [22].

The transmission medium is assumed to be ideal, i.e. no geometrical/electromagnetic or dispersion effects are taken into account. Only a pure delay and a reduction coefficient are considered here.

The photodiode model [20] contains an input connector for light, two electrical terminals (anode and cathode) and a thermal terminal dedicated to thermodynamical exchanges with other devices in the transmission receiver. The photodiode is a device that generates a current when it receives light on its input signal-flow connector. The photo-current is globally proportional to light intensity, but it is modified by

three factors: the dark current (sensitive to temperature), the diffusion capacitance of the diode and the leaking resistance, as stated in Fig. 14. When the photodiode is polarized in reverse mode it can be modeled as a current generator I_p in parallel with the diffusion capacitance C_d and the leak resistor. The current I_p is proportional to the optical power received by the photodiode and has an offset current I_{dark} called the dark current. The temperature dependency of the dark current is also taken into account. More accurate models of photodiodes can be found in [23][24].

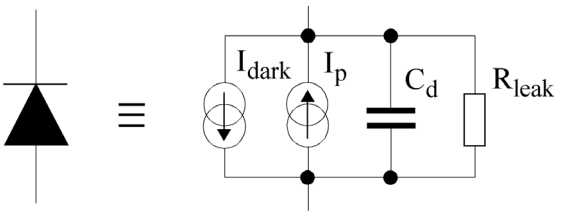

Fig. 14. The photo-diode model.

1) VHDL-AMS Description

The VHDL-AMS source code for the laser diode is given in Fig. 15. The model interface has two electrical ports, namely the anode and the cathode (line 19), one thermal port (line 20) and one signal-flow port (line 21), which receives the light emitted by the laser-diode. The signal-flow port is realized by an *interface quantity*. In the model body, the function `idval()` computes the diode current as a function of the temperature (lines 34 to 39), and the function `lightpwr` calculates the emitted light power as a function of the temperature and the diode current (lines 41 to 52). The diode current, the temperature sensitivity of the current threshold I_{th} and the ratio slope η are therefore easily implemented (lines 45 and 46). For (4), the model uses the predefined attribute `'ltf` to define a Laplace transfer function and to apply it to the `lpwr` quantity (line 58). The arguments of the attribute are the numerator and denominator coefficients of the transfer function indexed by their order in the respective polynomials. Here again, the named association mechanism, e.g., "`0 => 1.0`", improves the readability of the model.

The VHDL-AMS model of the transmission medium is given in Fig. 16. The model interface is purely of the signal-flow kind, with an input and an output *quantity ports* (lines 10 and 11). The model is very simple as it only delays the input with some possible attenuation (line 19). It is legally possible to write it as a single statement, but the Mentor tool does not yet support the application of the `'delayed` attribute to a quantity port.

The VHDL-AMS model of the photo-diode is given in Fig. 17. As for the laser diode model, the model interface includes one signal-flow port for the light input, two electrical ports for the anode and the cathode, and one thermal port for the thermal interactions (lines 19 to 21). The model body implements the temperature dependency of the dark current and the model of Fig. 14. If the temperature aspects were neglected, it could have been possible to avoid the declaration of the free quantities `idark`, `ip`, `ic`, and `ir` (line 32) and to declare them as through branch quantities at line 27 with the quantity `id` removed. The sum, currently expressed as an explicit statement (line 42), could have therefore been implicit thanks to the branch quantity declaration.

```
(1)   library ieee;
(2)     use ieee.math_real.all;
(3)
(4)   library ieee_proposed;
(5)     use ieee_proposed.energy_systems.all;
(6)     use ieee_proposed.electrical_systems.all;
(7)     use ieee_proposed.thermal_systems.all;
(8)
(9)   entity laser_diode is
(10)    generic (
(11)      IRR     : real := 1.0*PICO;     -- inverse diode current
(12)      ITH0    : real := 10.0*MILLI;     -- current thres. at ambient temp.
(13)      TAU     : real := 0.3*NANO;     -- output light time constant
(14)      ETA0    : real := 0.32; -- prop. coefficient at ambient temp.
(15)      ITHSFT: real := 0.1*MILLI;   -- temp. sensitivity of current thres.
(16)      ETASFT: real := 2.0*MILLI -- temp. sensitivity of prop. coeff.
(17)    );
(18)    port (
(19)      terminal tan, tca: electrical;
(20)      terminal tj     : thermal;
(21)      quantity olight  : out real
(22)    );
(23)  end entity laser_diode;
(24)
(25)  architecture bhv of laser_diode is
(26)
(27)    quantity vd across id through tan to tca;
(28)
(29)    quantity power through thermal_ref to tj;
(30)    quantity temp  across  tj         to thermal_ref;
(31)
(32)    quantity lpwr: real;
(33)
(34)    function idval (temp, vd: real) return real is
(35)      variable vt: real;
(36)    begin
(37)      vt := K*temp/Q;
(38)      return IRR*(exp(vd/(2.0*vt + 1.0e-20) - 1.0));
(39)    end function idval;
(40)
(41)    function lightpwr (temp, id: real) return real is
(42)      variable tempc, ith_eff, eta_eff: real;
(43)    begin
(44)      tempc := temp - 273.0;
(45)      ith_eff := ITH0 + ITHSFT*tempc;
(46)      eta_eff := ETA0 + ETASFT*tempc;
(47)      if id > ith_eff then
(48)        return eta_eff*(id - ith_eff);
(49)      else
(50)        return 0.0;
(51)      end if;
(52)    end function lightpwr;
(53)
(54)  begin
(55)    id == idval(temp, vd);
(56)    power == id*vd;
(57)    lpwr == lightpwr(temp, id);
(58)    olight == lpwr'ltf((0 => 1.0), (0 => 1.0, 1 => TAU));
(59)  end architecture bhv;
```

Fig. 15. VHDL-AMS model of the laser diode.

```
(1)   library ieee_proposed;
(2)     use ieee_proposed.energy_systems.all;
(3)
(4)   entity xmedium is
(5)     generic (
(6)       XDEL : real := 5.0*NANO;  -- transmission delay
(7)       XCOEF: real := 0.5        -- reduction coefficient
(8)     );
(9)     port (
(10)      quantity lin : in  real;
(11)      quantity lout: out real
(12)    );
(13)  end entity xmedium;
(14)
(15)  architecture ideal of xmedium is
(16)    quantity qin: real;
(17)  begin
(18)    qin == lin;  -- 'delayed on port qties not yet supported
(19)    lout == XCOEF*qin'delayed(XDEL);
(20)  end architecture ideal;
```

Fig. 16. VHDL-AMS model of the transmission medium.

```
(1)   library ieee;
(2)     use ieee.math_real.all;
(3)
(4)   library ieee_proposed;
(5)     use ieee_proposed.energy_systems.all;
(6)     use ieee_proposed.electrical_systems.all;
(7)     use ieee_proposed.thermal_systems.all;
(8)
(9)   entity photo_diode is
(10)    generic (
(11)      CD        : real := 1.0*PICO; -- diffusion capacitance
(12)      RLEAK     : real := 1.0*MEGA; -- leakage resistance
(13)      SENSITIVITY: real := 0.13;    -- diode sensitivity
(14)      IDARK0    : real := 1.0*NANO; -- dark current at nominal temp.
(15)      IDARK_DT  : real := 45.0;     -- temperature variation
(16)      COOLINGG  : real := 1.0*MILLI -- cooling conductance
(17)    );
(18)    port (
(19)      quantity ilight  : in real;
(20)      terminal tan, tca: electrical;
(21)      terminal tj      : thermal
(22)    );
(23)  end entity photo_diode;
(24)
(25)  architecture bhv of photo_diode is
(26)
(27)    quantity vd across id through tan to tca;
(28)
(29)    quantity power through thermal_ref to tj;
(30)    quantity temp  across tj to thermal_ref;
(31)
(32)    quantity tempc, idark, ip, ic, ir: real;
(33)
(34)  begin
(35)    tempc == temp - 273.0;
(36)
(37)    ir == vd/RLEAK;
(38)
(39)    idark == IDARK0;
(40)    ic == CD*vd'dot;
(41)    ip == -SENSITIVITY*ilight;
(42)    id == idark + ip + ic + ir;
(43)
(44)    power == abs(id*vd) + (tempc*COOLINGG - ilight);
(45)
(46)  end architecture bhv;
```

Fig. 17. VHDL-AMS model of the photo-diode.

2) Verilog-AMS Description

The Verilog-AMS source code for the laser diode is given in Fig. 18. The model interface has two electrical ports `tano` and `tcat`, one thermal port `tj` and one optical port `olight` (lines 4 to 9). Verilog-AMS uses a simplified discipline definition for modeling signal-flow ports. Since there is no predefined signal-flow optical discipline in the `disciplines.h` file, we have to define our own in the model. The following code gives the discipline definition:

```
nature Illuminance
   units  = "Cd";
   access = LP;
`ifdef CHARGE_ABSTOL
   abstol = `CHARGE_ABSTOL;
`else
   abstol = 1e-14;
`endif
endnature

discipline optical_sf
   potential Illuminance;
enddiscipline
```

The `optical_sf` discipline definition is stored in a file named `optical_sf.v`, which is included in the top-level airbag system model since three models are using the discipline. Light is then described by a potential (`Illuminance`) and managed by the access function `LP()`. In the model body, one can notice the call to the `cross` function without the execution of any related statement (line 30). The goal is to force the analog simulation kernel to have a simulation point inserted at the time of the crossing and then to accurately switch between the two possible expressions for the `lightpwr` variable (lines 31 and 32). The contribution to the light output is expressed through the Laplace filter `laplace_nd` as given by (4) (line 34).

The Verilog-AMS model of the transmission medium is given in Fig. 19. The model interface is purely signal-flow (lines 1 to 4) and the model body makes use of the predefined function `transition` to implement the fiber delay (line 9). Verilog-AMS does also support the function `absdelay` but it is not used here as it would force too many simulation timepoints. It has to be noted, however, that the transition function is used without rise or fall times to again avoid too many simulation time points. The price to pay is a less accurate output waveform, although still meaningful at the system level.

The Verilog-AMS model of the photo-diode is given in Fig. 20. As for the laser diode model, the model interface includes one signal-flow port for the light input, two electrical

ports for the anode and the cathode, and one thermal port for the thermal interactions (lines 1 to 8). The model body implements the temperature dependency of the dark current and the model of Fig. 14. Instead of using the variable `id` in line 25, it could have been possible to use the contribution operator <+ four times in a row.

```
(1)   `include "disciplines.h"
(2)   `include "constants.h"
(3)
(4)   module laser_diode (tano, tcat, tj, olight);
(5)
(6)     inout tano, tcat, tj, olight;
(7)     electrical tano, tcat;
(8)     thermal tj;
(9)     optical_sf olight;
(10)
(11)    parameter real IR      = 1p;   // inverse diode current
(12)    parameter real ITH     = 10m;  // current thresh. at ambient temp.
(13)    parameter real TAU     = 0.3n; // output light time constant
(14)    parameter real ETA     = 0.32; // prop. coeff. at ambient temp.
(15)    parameter real ITHSFT = 0.1m;  // temp. sens. of current thresh.
(16)    parameter real ETASFT = 2.0m;  // temp. sensitivity of prop. coeff.
(17)
(18)    real vd, id, vt;
(19)    real tempc, ith_eff, eta_eff, lightpwr;
(20)
(21)    analog begin
(22)      tempc = Temp(tj) - 273.0;
(23)      ith_eff = ITH + ITHSFT*tempc;
(24)      eta_eff = ETA + ETASFT*tempc;
(25)
(26)      vt = `P_K*Temp(tj)/`P_Q;
(27)      vd = V(tano, tcat);
(28)      id = IR*(limexp(vd/(2.0*vt + 1.0e-20) - 1.0));
(29)
(30)      @(cross(id - ith_eff, 0));  // enforces a time point at crossing
(31)      if (id > ith_eff) lightpwr = eta_eff*(id - ith_eff);
(32)      else              lightpwr = 0.0;
(33)
(34)      LP(olight)   <+ laplace_nd(lightpwr, {1}, {1, TAU});
(35)      I(tano, tcat) <+ id;
(36)      Pwr(tj)      <+ id*vd;
(37)    end
(38)
(39)  endmodule // laser_diode
```

Fig. 18. Verilog-AMS model of the laser diode.

```
(1)   module xmedium (lin, lout);
(2)
(3)     inout lin, lout;
(4)     optical_sf lin, lout;
(5)
(6)     parameter real XDEL  = 5n;   // transmission delay
(7)     parameter real XCOEF = 0.5;  // reduction coefficient
(8)
(9)     analog LP(lout) <+ XCOEF*transition(LP(lin), XDEL);
(10)
(11)  endmodule // xmedium
```

Fig. 19. Verilog-AMS model of the transmission medium.

```
(1)   `include "disciplines.h"
(2)
(3)   module photo_diode (ilight, tano, tcat, tj);
(4)
(5)     inout ilight, tano, tcat, tj;
(6)     electrical tano, tcat;
(7)     thermal tj;
(8)     optical_sf ilight;
(9)
(10)    parameter real CD        = 1p;   // diffusion capacitance
(11)    parameter real RLEAK     = 1M;   // leakage resistance
(12)    parameter real SENSITIVITY = 0.13; // diode sensitivity
(13)    parameter real IDARK0    = 1n;   // dark current at nominal temp.
(14)    parameter real IDARK_DT  = 45.0; // temperature variation
(15)    parameter real COOLINGG  = 1m;   // cooling conductance
(16)
(17)    real tempc, ir, ic, ip, idark, id;
(18)
(19)    analog begin
(20)      tempc = Temp(tj) - 273.0;
(21)      ir = V(tano, tcat)/RLEAK;
(22)      ic = CD*ddt(V(tano, tcat));
(23)      ip = -SENSITIVITY*LP(ilight);
(24)      idark = IDARK0*pow(10.0,(tempc/IDARK_DT));
(25)      id = ir + ic + ip + idark;
(26)      I(tano, tcat) <+ id;
(27)      Pwr(tj) <+ tempc*COOLINGG - LP(ilight);
(28)      Pwr(tj) <+ abs(id)*V(tano, tcat);
(29)    end
(30)
(31)  endmodule // photo_diode
```

Fig. 20. Verilog-AMS model of the photo-diode.

F. The chemical reaction submodel

Literature on airbag technology state that three concurrent chemical reactions (step reactions) take place at the same time, once the propergol capsule has been ignited, namely:

$$2NaN_3 \rightarrow 2Na + 3N_2 \quad (5)$$

$$10Na + 2KNO_3 \rightarrow K_2O + 5Na_2O + N_2 \quad (6)$$

$$K_2O + Na_2O + SiO_2 \rightarrow K_2Na_2SiO_4 \quad (7)$$

(5) states that, when the capsule blasts, molecules of sodium azide NaN3 are totally transformed into Sodium Na and Azote gas N_2. The Azote gas inflates the airbag envelope as requested, but sodium azide is highly toxic and ignites when mixed with water. The chemical reaction (5) has to be performed in less than 5 milliseconds. (6) acts as a sodium neutralizer, as it reacts with potassium nitrate KNO_3 to produce more N_2 and some moles of potassium oxide K_20 and sodium oxide Na_2O. (7) states that the two latter chemical products reacts with silicon dioxide to form a specified quantity of alkaline silicate $K_2Na_2SiO_4$, which is nothing more than a harmless glass powder.

"For the three reactions to occur, the reactants must meet at one place at one time. The probability of a reactant to be at any given place is proportional to its concentration, and the probabilities of the different reactants are stochastically independent of each other" [25]. The equation set (5)-(7) can be rewritten to introduce the necessary time parameter. Using the Van't Hoff theory on kinetic equations on the equation set with appropriate stoichiometric coefficients, we get the following reaction rate equations with eight unknowns representing the concentration of chemical products over time:

$$\frac{d[NaN_3]}{dt} = -2 \cdot k_1 \cdot [NaN_3]^2 \quad (8)$$

$$\frac{d[Na]}{dt} = 2 \cdot k_1 \cdot [NaN_3]^2 - 10 \cdot k_2 \cdot [Na]^{10} \cdot [KNO_3]^2 \quad (9)$$

$$\frac{d[N_2]}{dt} = 3 \cdot k_1 \cdot [NaN_3]^2 + k_2 \cdot [Na]^{10} \cdot [KNO_3]^2 \quad (10)$$

$$\frac{d[KNO_3]}{dt} = -2 \cdot k_2 \cdot [Na]^{10} \cdot [KNO_3]^2 \quad (11)$$

$$\frac{d[K_2O]}{dt} = k_2 \cdot [Na]^{10} \cdot [KNO_3]^2 - k_3 \cdot [K_2O] \cdot [Na_2O] \cdot [SiO_2] \quad (12)$$

$$\frac{d[Na_2O]}{dt} = 5 \cdot k_2 \cdot [Na]^{10} \cdot [KNO_3]^2 - k_3 \cdot [K_2O] \cdot [Na_2O] \cdot [SiO_2] \quad (13)$$

$$\frac{d[SiO_2]}{dt} = -k_3 \cdot [K_2O] \cdot [Na_2O] \cdot [SiO_2] \quad (14)$$

$$\frac{d[K_2Na_2SiO_4]}{dt} = k_3 \cdot [K_2O] \cdot [Na_2O] \cdot [SiO_2] \quad (15)$$

k1 to *k3* represent probability constants, so called reaction rate constants.

This chemical model expects NaN_3 to exponentially decrease with time, and N_2 gas to increase, in order to inflate the airbag. The Na should first increase with the first reaction of (5) and then decrease with the second reaction of (6). However, Na should never grow too much.

Like the other models in the airbag system, the chemical reaction kinetics is only roughly described by the eight equations (8)-(15). This is because our chemical model is the direct emanation of our "macroscopic" vision of chemistry as engineers with a standard EECS curriculum. As stated by [25], "literature on chemical reaction kinetics concentrates on molar concentrations and their time derivative exclusively, and it ignores the energy and its conservation entirely. Many references on the subject do not introduce the chemical potential as a system property at all".

1) VHDL-AMS Description

The VHDL-AMS model of the `chemsys` block that realizes the chemical reactions is given in Fig. 21. The model interface only includes a single digital signal `strig` that represents the trigger signal that will initiate the reactions (line 11). The model body implements the equation set (8)-(15) (lines 26 to 37). In lines 39 to 46, the model defines the initial conditions that have to hold before the trigger signal becomes active. The `break` statement on line 23 notifies the analog simulation kernel when an event on the discrete-time trigger signal occurs, therefore forcing an analog simulation point at the time of the event. This allows the simulator to compute a smooth transition between the two regions of operations of the model that are defined by the `if` statement (lines 25 to 47).

```
(1)   library ieee;
(2)     use ieee.std_logic_1164.all;
(3)
(4)   entity chemsys is
(5)     generic (
(6)       K1: real := 14000.0;  -- reaction rate for eq.(5)
(7)       K2: real := 1.0;      -- reaction rate for eq.(6)
(8)       K3: real := 1.0       -- reaction rate for eq.(7)
(9)     );
(10)    port (
(11)      signal strig: in std_logic
(12)    );
(13)  end entity chemsys;
(14)
(15)  architecture bhv of chemsys is
(16)
(17)    -- chemical concentrations
(18)    quantity cNaN3,cNa,cN2,cKNO3, cK2O, cNa2O, cSiO2,
(19)           cK2Na2SiO4: real;
(20)
(21)  begin
(22)
(23)    break on strig;
(24)
(25)    if strig = '1' use
(26)      cNaN3'dot == -2.0*K1*(cNaN3**2);
(27)      cNa'dot   == 2.0*K1*(cNaN3**2) –
(28)         10.0*K2*(cNa**10)*(cKNO3**2);
(29)      cN2'dot   == 3.0*K1*(cNaN3**2) +
(30)         K2*(cNa**10)*(cKNO3**2);
(31)      cKNO3'dot == -2.0*K2*(cNa**10)*(cKNO3**2);
(32)      cK2O'dot  == K2*(cNa**10)*(cKNO3**2) –
(33)         K3*cK2O*cNa2O*cSiO2;
(34)      cNa2O'dot == 5.0*K2*(cNa**10)*(cKNO3**2) –
(35)         K3*cK2O*cNa2O*cSiO2;
(36)      cSiO2'dot == -K3*cK2O*cNa2O*cSiO2;
(37)      cK2Na2SiO4'dot == K3*cK2O*cNa2O*cSiO2;
(38)    else
(39)      cNaN3     == 5.0/3.0;
(40)      cKNO3     == 1.0/3.0;
(41)      cN2       == 0.0;
(42)      cNa       == 0.0;
(43)      cK2O      == 0.0;
(44)      cNa2O     == 0.0;
(45)      cSiO2     == 1.0/6.0;
(46)      cK2Na2SiO4 == 0.0;
(47)    end use;
(48)
(49)  end architecture bhv;
```

Fig. 21. VHDL-AMS model of the chemical system.

2) Verilog-AMS Description

The Verilog-AMS model of the `chemsys` block that realizes the chemical reactions is given in Fig. 22. The model interface only includes a single digital signal `strig` that represents the trigger signal that will initiate the reactions (line 13). The model body implements the equation set (8)-(15) (lines 24 to 48) in their integral form using the predefined analog operator `idt`. This form is preferred as the `idt` operator provides a way to specify a reset condition, which in the model is handled by the value of the digital signal `strig`. The indirect branch assignment also offered by Verilog-AMS is not appropriate here as we need to trigger the integration at some point in time. When the signal `strig` has a nonzero value, the operator returns the initial value specified as the second argument. The integration therefore starts when the signal value becomes zero. Another point is the definition of

the chemical signal-flow discipline `chem_sf` and its associated nature `ChemQ` that is required to declare the quantities to integrate. It is not allowed to use real variables for that purpose.

```
(1)   nature ChemQ
(2)     units  = "-";
(3)     access = CH;
(4)     abstol = 1e-14;
(5)   endnature
(6)
(7)   discipline chem_sf
(8)     potential ChemQ;
(9)   enddiscipline
(10)
(11)  module chemsys (strig);
(12)
(13)    input strig;
(14)
(15)    parameter real K1 = 14000.0;  // reaction rate for eq. (5)
(16)    parameter real K2 = 1.0;      // reaction rate for eq. (6)
(17)    parameter real K3 = 1.0;      // reaction rate for eq. (7)
(18)
(19)    // chemical concentrations
(20)  chem_sf cNaN3,cNa,cN2,cKNO3,cK2O,cNa2O, cSiO2,
(21)          cK2Na2SiO4;
(22)
(23)    analog begin
(24)      CH(cNaN3) <+ idt(-2.0*K1*pow(CH(cNaN3),2),
(25)                   5.0/3.0, !strig);
(26)      CH(cNa)   <+ idt(2.0*K1*pow(CH(cNaN3),2)
(27)                   -10.0*K2*pow(CH(cNa),10)*pow(CH(cKNO3),2),
(28)                   0.0, !strig);
(29)      CH(cN2)   <+ idt(3.0*K1*pow(CH(cNaN3),2)
(30)                   + K2*pow(CH(cNa),10)*pow(CH(cKNO3),2),
(31)                   0.0, !strig);
(32)      CH(cKNO3) <+ idt(-2.0*K2*pow(CH(cNa),10)*
(33)                   pow(CH(cKNO3),2),
(34)                   1.0/3.0, !strig);
(35)      CH(cK2O)  <+ idt(K2*pow(CH(cNa),10)*
(36)                   pow(CH(cKNO3),2)
(37)                   - K3*CH(cK2O)*CH(cNa2O)*CH(cSiO2),
(38)                   0.0, !strig);
(39)      CH(cNa2O) <+ idt(5.0*K2*pow(CH(cNa),10)*
(40)                   pow(CH(cKNO3),2)
(41)                   - K3*CH(cK2O)*CH(cNa2O) *CH(cSiO2),
(42)                   0.0, !strig);
(43)      CH(cSiO2) <+ idt(-K3*CH(cK2O)*CH(cNa2O)*
(44)                   CH(cSiO2),
(45)                   1.0/6.0, !strig);
(46)      CH(cK2Na2SiO4) <+ idt(K3*CH(cK2O)*
(47)                   CH(cNa2O)*CH(cSiO2),
(48)                    0.0, !strig);
(49)    end
(50)  endmodule  // chemsys
```

Fig. 22. Verilog-AMS model of the chemical system.

IV. Simulation Results

As is, the airbag system model can be used to obtain numerous valuable simulation results. This section presents only a small set of result examples.

A. Mechanical-Electrical Interactions

Fig. 23 presents a screen dump of the VHDL-AMS simulation. Fig. 24 presents similar results obtained with the corresponding Verilog-AMS model. The upper plot represents the simulated acceleration of the seismic mass and the lower plot shows the proportional evolution of the amplitude at the accelerometer output. Chronograms at the bottom of the figure also show the digital output of the comparator. While the acceleration amplitude exceeds a given threshold, the comparator output `d` toggles at a frequency of 1 MHz. When 10 pulses have been reached, the `chtrig` signal is asserted and the chemical reaction is initiated.

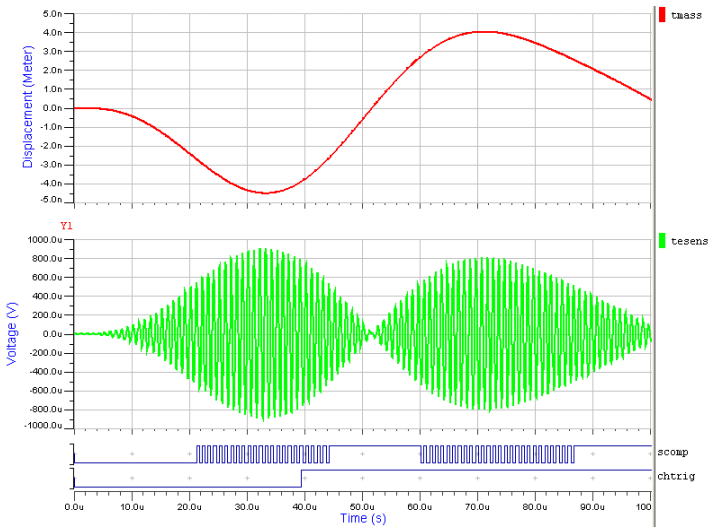

Fig. 23. Mechanical-electrical interactions (VHDL-AMS).

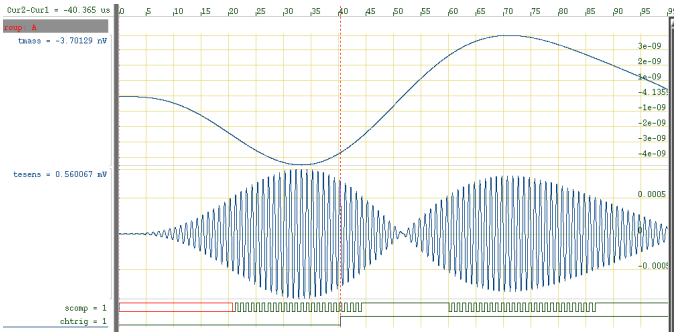

Fig. 24. Mechanical-electrical interactions (Verilog-AMS).

B. Electrical-Thermal Interactions

Fig. 25 details the simulation results, obtained with Advance-MS, of the thermal coupling between the n-MOST and the p-MOST when the inverter is stimulated by a square wave stimulus and connected to the thermal RC networks given in Fig. 8 [17]. The thermal network represents the thermal constants of the various material layers, and the values of the capacitances and resistances have been deliberately oversized to highlight the thermal effects. The figure shows the temperature evolution in the inverter for two different values of R_{NP}. As expected, for small values of R_{NP}, the temperatures in the N and P transistors are tightly linked (curves 1 and 2 in

the figure). For higher values, the temperature evolution of each transistor becomes more and more decoupled (curves 3 and 4 in the figure). The dissipated power is produced during transitions, as expected.

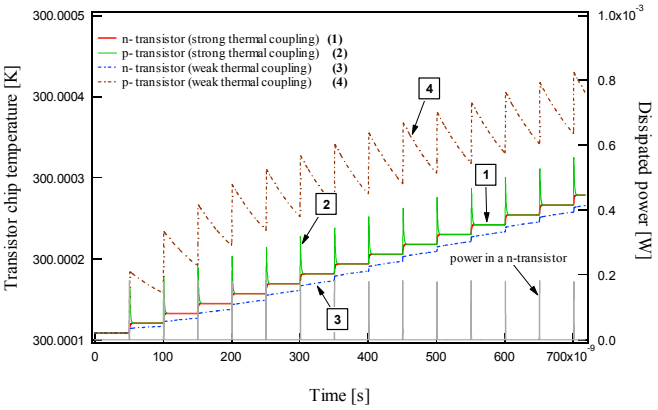

Fig. 25. Thermal coupling between the nMOST and pMOST for different values of coupling [17].

C. *Electrical-Optical Interactions*

Fig. 26 shows the falling edge of the collision signal generated by the CMOS inverter, transformed into light and propagated through the optical fiber, with the appropriate delay and reduction coefficient. The plot has been extracted from an Advance-MS simulation.

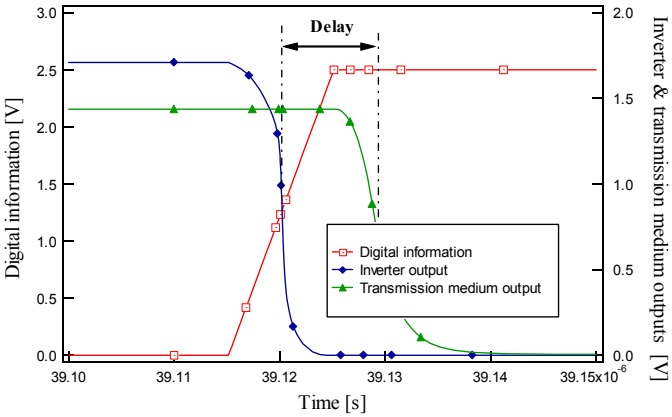

Fig. 26. Propagation delay in the transmission medium (Advance-MS).

D. *Chemical Reactions*

Fig. 27 shows typical chemical kinetic reaction behaviors from the VHDL-AMS model. Reactive chemical product concentration of NaN_3 decreases exponentially with time, and one can see that the 2.5 moles of N_2, providing the 60 liters of necessary gas according to the perfect gas law, are generated in less than 5 milliseconds. The volatile sodium Na is instantly generated when the airbag inflation order is triggered, and then decreases during the sodium neutralization phase. The time scale of this plot should also be noticed: all the mechanical-electrical-optical signal transmission time is negligible compared to the time needed to inflate the airbag.

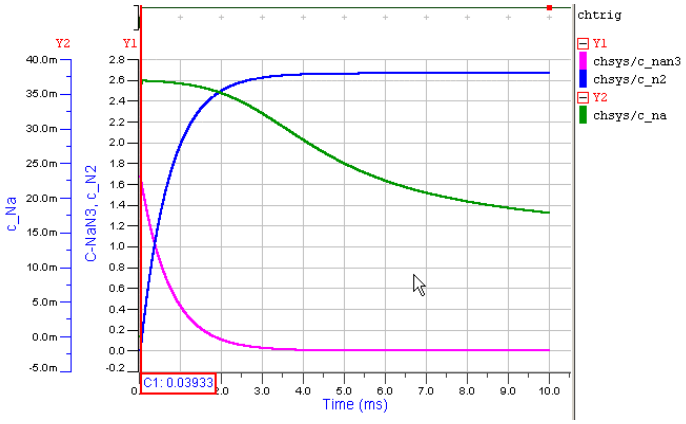

Fig. 27. Chemical reactions (Advance-MS).

E. *Global System Behavior*

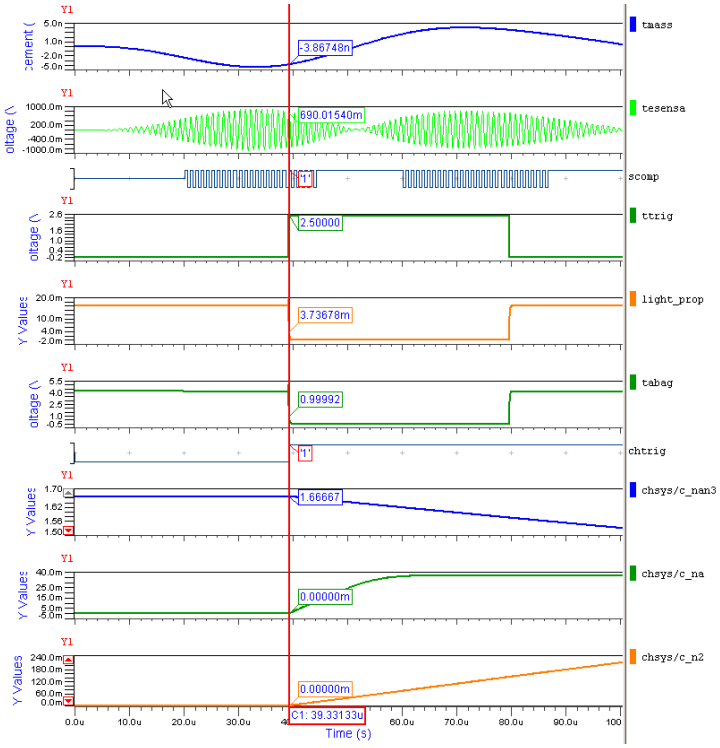

Fig. 28. Airbag system model simulation results (Advance-MS).

Fig. 28 and Fig. 29 give a snapshot of the behavior of the whole airbag system, obtained with the VHDL-AMS and the Verilog-AMS model, respectively.

From top to bottom, we find successively the chronograms for the acceleration stimulus of the exciting the seismic mass `tmass`, the amplified modulated accelerometer output signal `tsensa`, the comparator output `scomp`, the trigger signal `ttrig`, the propagated light `light_prop`, the photo-diode output signal `tabag`, the chemical reaction trigger signal `chtrig` and the evolutions of the concentrations of the NaN_3, N_2 and Na elements.

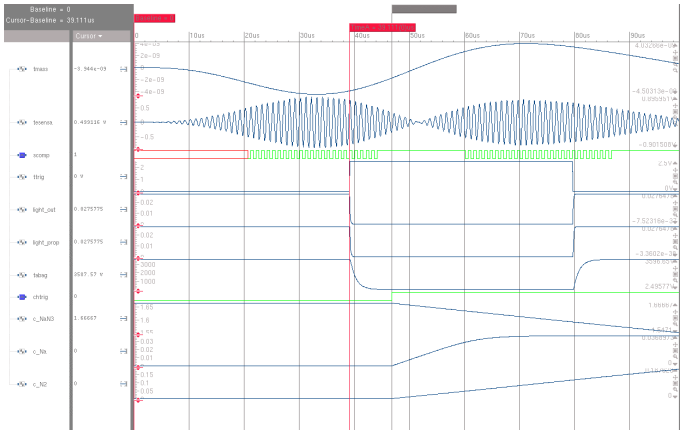

Fig. 29. Airbag system model simulation results (Cadence).

V. Discussion

The various models of the airbag system developed in the paper are neither complete nor accurate. The activity of modeling is mostly directed by which aspects of a physical behavior have to be taken into account and how a particular modeling notation, e.g. a hardware description language, can actually express these aspects properly through simulation. Additionally, the development of a complete model of the airbag system is beyond the scope of the paper. All the models discussed in this paper are kept intentionally simple to avoid hiding key aspects in too many lines of code. All the models of Fig. 1 have been designed and simulated with appropriate stimuli in both languages and concordant results have been obtained (cf. Figs 28 and 29). Not all the models have been discussed in the paper, but the full set of VHDL-AMS and Verilog-AMS models, including test benches and appropriate simulation control commands, are available [26].

In section 2.2, we listed a number of distinguished features of the VHDL-AMS and the Verilog-AMS languages to give a first insight into how each of them could support the development of models of mixed-signal and multi-discipline systems. Now we went through the example of modeling an airbag system, we can expand the list further. Table I gathers a number of feature classes and indicates how each language actually supports the features in each class. Some items have further associated notes that are given below the table to avoid cluttering the table too much.

The paper shows that the same complex system can be successfully modeled with both HDLs. Some workarounds have been however required to address the partial support of the languages in current tools, although the alternative solutions worked as well. For instance, the procedural construct of VHDL-AMS is not yet supported by Advance MS. The model of the EKV MOSFET (section 3.4) could have been more naturally described without using functions.

The main difference between the two HDLs is the way equations are handled. On the one hand, sequencing the equations is mandatory in Verilog-AMS, because the language forces the designer to use sequential statements in an analog block, and because only certain forms of implicit equations are supported. On the other hand, in VHDL-AMS, sequencing the equations is a transparent task thanks to the powerful notation for equations provided by simultaneous statements. Verilog-AMS models have to be coded with a nodal formulation in mind and therefore fit better with current simulator implementations. In the same context, Verilog-AMS allows only one analog block per module, while any number of concurrent and simultaneous statements can cohabit in a single VHDL-AMS architecture. This reinforces the more circuit level approach of Verilog-AMS where each behavioural module is basically seen as a leaf cell which has to be hierarchically connected in some upper level. VHDL-AMS also supports this approach, while providing more degrees of freedom when writing complex DAEs. It should be noted, however, that this feature might be less efficiently handled in current VHDL-AMS simulators that are still built on circuit simulators. What would be ideally needed is an efficient blend of an event-driven/circuit simulator and a, possibly symbolic and numerical, equation solver. The model writer would then be free to select the most appropriate method to describe equations. Last, but not least, one should not forget the mandatory need for event-driven modeling capabilities, not only to describe the event-driven/logic part of the system, bus also to simplify some analog parts and/or make their simulation more efficient.

Probably the chemical subsystem used in the airbag system exemplifies many of the modeling issues one has to deal with when addressing multi-discipline systems. In section 3.6, we deliberately used a macroscopic approach that does not fully take all physical aspects into account. If we consider the same chemical reactions from a microscopic viewpoint, things appear totally different. Chemistry turns into thermodynamics, and the equations (5)-(7) concerning mass transfer only are replaced by equations concerning the couple mass/energy. As a consequence, the chemical submodel involves more elements, interactions between these elements take place at a very low level, and conservative laws pervade the previous signal-flow approach. For each chemical reactant, “two different goods are actually traded, namely, mass and energy” [25] and the GKLs apply. [25] details perfectly this process, and shows how the chemical potential [J $mole^{-1}$] and the molar flow [$mole\ sec^{-1}$] can respectively be defined as effort and flow quantity for each reactant. The product of the two quantities has the dimension of a chemical power. Drawing an analogy with the electrical domain, reactants and products are modeled by capacitive sources, one for each chemical species, where moles accumulate like charges. The reaction equations are modeled by multi-port transformers with transform ratios determined by stoichiometric coefficients (also represented by transformers), which balance and redistribute chemical power between the chemical species. All these elements are interconnected and yield a chemical network isomorphic to the

electrical or thermal networks described in the paper. [9] defines this approach as object-oriented modeling, i.e. "the construction of strictly hierarchical, modular models. Each terminal element has its own implemented sequential algorithm. Setting up the equations describing the whole system is the task of the simulator. All elements communicate via message passing, coordinate their behavior, and so the simulation of the entire system is carried out". [25] [27] also propose to use bond graphs for that purpose, but they are too complicated most of the time. Our experience on this chemical issue shows that this phenomenological approach, while elegant and consistent, hides lots of difficulties: a thorough methodological background is needed to design the leaf submodels because the reasons for analogies between physical theories [28] are not always simple to understand.

Feature class	**Feature**	**VHDL-AMS**	**Verilog-AMS**
Language aspects	Definition	IEEE Std 1076.1-1999 Strict extension to IEEE Std 1076 (VHDL)	Accellera standard version 2.1 (January 2003) Extension to IEEE Std 1364 (Verilog HDL)
	Inheritance	Ada-like Case insensitive	C-like Case sensitive
	Modularity	Separation of external/interface views (entities) and internal views (architectures), packages, configurations	Modules include both external/interface and internal views. Include files for shared definitions
	Genericity	Parameters, generate statements[1]	
	Library management	Yes (pre-compiled design units)	No
	Analog subset	No[2]	Yes (limited support of SPICE primitives)
Expression of structure	Ports	Event-driven and continuous	
		Conservative and non conservative (signal-flow)	
		Continuous ports are modeless	Continuous ports are of mode **inout**
	Composition	Hierarchical instantiation of components	
	Conservative semantics	Natures define energy domains, subtypes define nature attributes; no predefined natures.	Disciplines define energy domains, natures define discipline attributes; predefined set of disciplines
		Terminal and branch quantities	Access functions and named branches
Expression of behavior	Objects	Terminals, quantities, signals, variables, constants	Variables, wires, registers
	Statements	Concurrent, sequential, continuous (simultaneous and procedural)	
		Continuous statements can be freely mixed with concurrent statements	Continuous statements have to be grouped in an analog block. Only one analog block per module
	Expression of DAEs[3]	Explicit and implicit form of equations[4] supported Simultaneous[5] and procedural[6] formulations Derivative attribute 'dot only possible on quantities. Attribute can be chained for higher order derivatives[7] Mathematical functions defined in separate standard IEEE 1076.2	Explicit form of equations supported; limited support of implicit form of equations[8] Procedural formulation Derivative operator ddt may be applied to expressions. Operator cannot be chained for higher order derivative; must declare local nodes[9] Pre-defined mathematical functions
		Piecewise defined behavior supported[10]	
	Discontinuity	Discontinuities must be explicitly announced in the model	

Feature class	Feature	VHDL-AMS	Verilog-AMS
	handling	User-defined re-initialization after discontinuity supported	Order of discontinuity may be specified
Conservative semantics	Energy domains	Natures define energy domains and subtypes define nature attributes. No predefined natures[11] Branch quantities[12]	Disciplines define energy domains and natures define discipline attributes. Pre-defined set of disciplines. Access functions and branch objects[13]
	Formulation	Equation-oriented formulation with simultaneous statements[14] No specific circuit graph representation enforced	Circuit-oriented formulation with probe and source branches[15] and additive contribution statements[16] Nodal formulation enforced
Signal-flow semantics	Model interface	Directional interface (free) quantities[17]	Module ports associated to a discipline with only one nature (signal-flow discipline) Signal-flow ports are interpreted as grounded potential probes or sources
	Functional blocks	Laplace and z transforms[18]	
Mixed-signal aspects	Interfaces	A/D and D/A interface language attributes ('ramp, 'slew, 'above) No direct port association[19]	A/D and D/A interface language filters and event detectors Automatic insertion of connect modules (infra- or inter-disciplines)
	Behavioral interaction	Access of discrete signals in continuous context Access of continuous quantities in discrete context	Access of discrete nets and variables in continuous context Access of continuous nets and variables in discrete context
Simulation controls	Solvability	Solvability check done at design unit level[20]	No solvability checks
	Timestep	Timestep size may be bounded	
	Tolerances	Generic string annotation not formally linked to simulator[21]	SPICE-like tolerances defined in natures
Tool used		Mentor AdvanceMS 1.5 Full IEEE 1076.1 standard not yet supported[22]	Cadence AMS Simulator 2.0 Proprietary implementation of a standard in development

[1] Generate statements offer macro-like capabilities in the text of the model. Verilog-AMS also supports pre-processing directives.
[2] It is possible to develop packages to support SPICE level modeling. No standard packages are existing yet.
[3] DAE = Differential Algebraic Equations.
[4] An equation in the explicit form looks roughly like an assignment, e.g. x = f(y,z), while an equation in the implicit form typically requires iterations to compute the unknowns, e.g. x = f(x,y,z).
[5] Simultaneous statements are basically equations that may be given in any order in the model.
[6] Procedural statements have to be given in a particular order. The VHDL-AMS tool used did not support simultaneous procedural statements yet, so we used functions instead.
[7] To maintain good numerical accuracy it is recommended to hold higher order derivatives in local quantities and to only use first order derivatives.
[8] An equation in explicit form is written with the contribution statement, while an equation in implicit form requires a special syntax with limitations. Only state-based equations may be expressed in implicit form.
[9] It is not clear whether this is a limitation of the tool or of the language definition. The net effect is to clutter the model with unnecessary node declarations and contribution statements.
[10] Continuous behavior can be defined by regions of operation.
[11] A draft VHDL-AMS standard package for multiple energy domain support is currently under IEEE ballot.
[12] The direction of the flow in the branch and of the potential difference is defined in the branch quantity declaration.
[13] As access functions may be scattered throughout the whole model, it is recommended to hold probe branches in local variables at the beginning of the analog block.

[14] Quantities are the unknowns. As far as the language is defined, the order in which the simultaneous statements is not important. We anyway faced some non-convergence issues with "misplaced" simultaneous statements (tool issue).
[15] Roughly speaking, probe branches measure some voltage or current, while source branches generate some voltage or current.
[16] The order in which the contribution statements are given is important.
[17] Direction is used for solvability checks.
[18] Verilog-AMS offers more variants than VHDL-AMS.
[19] It is not allowed to associate formal and actual ports of different natures or types. Explicit interface code has to be added in the model when pre-defined attributes are not enough. It is also expected that tools may help in inserting proper interface code when working at schematic level.
[20] This basically checks that there is an equal number of unknowns and of equations. Although the rules that define what is considered as an unknown and what is considered as an equation are clearly defined in the language reference manual, it may become pretty hard to figure out what is missing when a complex model such as the full EKV MOS model (with more than 100 quantities) does not comply with the solvability condition. In addition, the current implementation of the Mentor tool imposes to have the same number of simple simultaneous statements in each branch of a conditional or selective simultaneous statement.
[21] Current VHDL-AMS simulators are using their own tolerances that may be set in the tool's environment.
[22] Full digital VHDL 1076 support requires coupling with Modelsim simulator.

Table I. Key features of VHDL-AMS and Verilog-AMS.

VI. Conclusion

In this paper, the VHDL-AMS and Verilog-AMS languages have been compared and contrasted in an objective manner. A complex case study has been used as a representative mixed-signal issue and solution to demonstrate some language similarities and differences, and indicated that the system modeled in one language can also be modeled in the other, although with different coding techniques that are not only related to syntax.

Whatever the state of the language, standard or not, all-defined mixed-signal capabilities are not yet implemented or still subject to interpretation. Anyway, we hope that tool limitations will be progressively removed so the model writer has access to the real expressive powers of both languages.

Therefore, the choice of the language is mostly guided by the idiosyncrasies of the developing team, the EDA tool suite available, and commercial and time to market issues. VHDL-AMS can operate at various levels of abstraction. Both VHDL-AMS and Verilog-AMS prove their efficiency at the circuit-level. At this level, however, solid knowledge on analogies between physical theories is required [25].

As systems on chips (SoC) more and more include analog, RF and non-electrical parts (not to mention the growing importance of embedded software), it is possible that models written in VHDL-AMS or Verilog-AMS become too low-level and their relative simulators too slow for validating a complete system. An ongoing effort is for example intending to enhance SystemC to support the description and the simulation of continuous-time, and a fortiori mixed-signal and multi-discipline, systems [29]. A seamless path to VHDL-AMS or Verilog-AMS modeling levels will have anyway to be provided to keep a link to more physical design aspects.

VII. Acknowledgments

The authors would like to express their thanks to Dr Y. Leroy, Prof. A. Goltzené, Prof. B. Allard, Prof. F.E. Cellier, and Prof. C. Fretigny for their help on modeling the chemical reaction kinetics.

[1] N.R. Swart, A. Nathan, "Flow-Rate Microsensor Modeling and Optimization using SPICE," Sensors and Actuators, A34, pp. 109-122, 1992.
[2] N.R. Swart, A. Nathan, "Mixed-Mode Device Circuit Simulation of Thermal-Based Microsensors," Sensors and Materials, Vol. 6, No. 3, pp. 179-192, 1994.
[3] D. C. Hamill, "Lumped equivalent circuits of magnetic components: the gyrator-capacitor approach," Power Electronics, IEEE Transactions on, Volume: 8 Issue: 2 , pp. 97-103, April 1993
[4] 1076.1-1999 IEEE Standard VHDL Analog and Mixed-Signal Extensions Language Reference Manual, IEEE Press, ISBN 0-7381-1640-8.
[5] P. Ashenden, G. D. Peterson, D. A. Teegarden, The System Designer's Guide to VHDL-AMS, Morgan Kaufman Publishers, 2002.
[6] E. Christen, K. Bakalar, "VHDL-AMS-a hardware description language for analog and mixed-signal applications," IEEE Trans. on Circuits and Systems, part II, Vol. 46 Issue: 10, pp. 1263 –1272, Oct. 1999.
[7] P. Frey, D. O'Riordan," Verilog-AMS: Mixed-signal simulation and cross domain connect modules," Proc. 2000 IEEE/ACM International Workshop on Behavioral Modeling and Simulation (BMAS), 2000, pp. 103 –108.
[8] http://www.eda.org/verilog-ams/
[9] P. Schwarz, "Physically Oriented Modeling of Heterogeneous Systems," 3rd IMACS Symposium of Mathematical Modelling (MATHMOD), Wien, 2-4 Feb. 2000, pp. 309-318 (vol1).
[10] E. Christen, "Selected Topics in Mixed-Signal Simulation," Proc. 2002 Forum on Specification and Design Languages (FDL), 2002.
[11] http://www.accellera.org/
[12] D. Fitzpatrick, I. Miller, Analog Behavioral Modeling with the Verilog-A Language, Kluwer Academic Publishers, 1998.
[13] L. Zimmermann, J.Ph. Ebersohl, F. Le Hung, J.P. Berry, F. Baillieu, P. Rey, B. Diem, S. Renard, and P. Caillat, " Airbag application: a microsystem including a silicon capacitive accelerometer, CMOS switched capacitor electronics and true self-test capability," Sensors and Actuators A, 46-47, pp. 190-195, 1995.
[14] E. Christen, K. Bakalar, A. M. Dewey, E. Moser, "Tutorial 2: Analog and Mixed-Signal Modeling Using the VHDL-AMS Language," 36th Design Automation Conference, New Orleans, June 21-25, 1999.

[15] H. A. Mantooth, E. Christen, "Modeling and simulation of electrical and thermal interaction," Modeling in Analog design, Vol. 2, in series on Current Issues in Electronic Modeling, Ch. 5, pp. 93-120, Eds: Jean-Michel Bergé, et al., Kluwer Academic Publishers, 1995.
[16] C. Lallement, R. Bouchakour and T. Maurel, "One-dimensional Analytical Modeling of the VDMOS Transistor Taking Into Account the Thermoelectrical Interactions," IEEE Transactions on Circuits and Systems, Part. I, Vol. 44, N° 2, pp. 103-111, Feb. 1997.
[17] C. Lallement, F. Pêcheux, Y. Hervé, "A VHDL-AMS Case study: The incremental Design of an Efficient 3rd generation MOS Model of Deep Sub Micron Transistor," SOC Design Methodologies, Ed. M. Robert, B. Rouzeyre, C. Piguet, M.-L. Flottes, Kluwer Academic Publishers, Boston, Hardbound (ISBN 1-4020-7148-5), pp. 349-360, July 2002
[18] M. Bucher, C. Enz, F. Krummenacher, J.-M. Sallèse, C. Lallement, A.-S. Porret, "The EKV 3.0 Compact MOS Transistor Model : Accounting for Deep-Submicron Aspects (invited paper)," International Workshop on Compact Models, Modeling and Simulation of Microsystems (MSM 2002), pp. 670-673, San Juan, Puerto Rico, April 22-25, 2002.
[19] M. Bucher, C. Lallement, C. Enz, F. Théodoloz, K. Krummenacher, "The EPFL-EKV MOSFET Model Equations for simulation, version 2.6," http://legwww.epfl.ch/ekv/
[20] W. Uhring, Y. Hervé, F. Pêcheux, "Model of an instrumented optoelectronic transmission system in HDL-A and VHDL-AMS," Proceedings of SPIE Vol. 3893, pp. 137–146, Design, Characterization, and Packaging for MEMS and Microelectronics, Editor(s): B. Courtois, S. N. Demidenko. Published: 10/1999.(ISBN 0-8194-3494-9)
[21] F. Pêcheux, C. Lallement, "VHDL-AMS and Verilog-AMS as Competitive Solutions for the High Level Description of Thermodynamical Interactions in Opto-Electronic Interconnection Schemes," FDL'02 (Forum on specifications & Design Languages), France, September 2002
[22] T. Nagahori, K. Miyoshi, Y. Aizawa, Y. Kusachi, Y. Nukada, N. Kami, and N. Suzuki, "An Analog Front-End Chip Set Employing an Electro-Optical Mixed Design on SPICE for 5-Gb/s/ch Parallel Optical Interconnection," IEEE Journal Of Solid-State Circuits, Vol. 36, No. 12, pp. 1984-1991, December 2001
[23] Wei-Jean Liu; Chen, O.T.-C.; Li-Kuo Dai; Ping-Kuo Weng; Kaung-Hsin Huang; Far- Wen Jih**, "**A CMOS photodiode model,**"** Behavioral Modeling and Simulation, 2001. BMAS 2001. Proceedings of the Fifth IEEE International Workshop on , 2001, pp. 102 –105
[24] Jau-Ji Jou; Cheng-Kuang Liu; Chien-Mei Hsiao; Huang-Hsiang Lin; Hsiu-Chih Lee, "Time-delay circuit model of high-speed p-i-n photodiodes," IEEE Photonics Technology Letters, Volume: 14, Issue: 4, pp. 525 -527, April 2002
[25] F.E. Cellier, "Continuous System Modeling," Springer-Verlag NY, (ISBN 0-387-87502-0), 1991.
[26] http://lsmwww.epfl.ch/tcad_paper/.
[27] A. Blundell, "Bond Graphs for Modelling Engineering Systems," Ellis Horwood Publishers, Chichester, United Kingdom, and Halsted Press, New-York, 1982.
[28] E. Tonti, "The reasons for analogies between physical theories," Appl. Math. Modeling 1(1976), 37-50.
[29] A. Vachoux *et al.*, "Towards Analog and Mixed-Signal SOC Design with SystemC-AMS," Proceedings of the Second IEEE International Workshop on Electronic Design, Test and Applications (DELTA 2004), 0-7695-2081-2/04, Feb. 2004.

François Pêcheux received in 1988 the M.S. degree from Université Pierre et Marie Curie, Paris, France, and the Ph.D. degree from the same University in 1992, all in Computer Science and Electrical Engineering. From 1991 to 1992, he worked as a research assistant in the MASI Laboratory, Paris, France. He developed several CAD tools and dedicated software for the efficient modeling of macrocell generators. In 1995, he joined the Ecole Nationale Supérieure de Physique de Strasbourg (ENSPS), Strasbourg, France as an Associate Professor, and the CNRS Laboratory for Physics and Applications of Semiconductors (PHASE). He participated in the development of several mixed-signal models for submicron devices, taking into account physical interactions with their direct environment. In September 2002, he joined back the LIP6 Laboratory, in the Integrated Systems Department, Paris, France. His current researchs focus on System-level modeling and the use of efficient principles, languages and tools for integrating mixed-signal designs on chip.

Christophe Lallement (M'96) received the M.S. degree from Science University of Nancy I, France, in 1989, and the Ph.D. degree from Ecole Nationale Supérieure des Télécommunications (ENST), Paris, France, in 1993, all in engineering. From 1989 to 1993, he was a research assistant with the Electronics Department, ENST-Paris. From November 1994 to September 1997, he was a Postdoctoral Research Scientist at the laboratory of Electronics, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, working on the characterization and modeling of the MOSFET transistor in the development team of the EKV MOST model. In September 1997, he joined the University of Strasbourg (ULP) as an associate Professor, and the CNRS Laboratory for PHysics and Applications of Semiconductors (PHASE laboratory). Since September 2003, he is Professor at the ENSPS Engineering School of Strasbourg. His current researches focus on modeling and characterization of the deep submicron MOSFET transistors with the EKV compact model, on the study of deep submicron effects in the MOSFET transistor, and on the modeling of mixed-signal systems with VHDL-AMS and Verilog-AMS.

Alain Vachoux (M'81) received the M.S. and Ph.D. degrees in electrical engineering from the Swiss Federal Institute of Technology Lausanne (EPFL) in 1980 and 1988, respectively.

From 1981 to 1998 he was with EPFL as a Research Assistant where he worked on the development of EDA tools for filter design and large-scale circuit simulation, and on VHDL-based design. From 1998 to 2001, he was with XEMICS, a Swiss fabless semiconductor company that designs mixed-signal ASICs and ASSPs. He was responsible for the deployment of a proprietary low-power standard cell and memory library as well as of a low-power 8-bit RISC core. In 2001, he joined Antrim Design Systems to work on mixed-signal modeling and characterization plans for pipeline A/D converters. He also managed the development of a mixed-signal layout generator for retargeting purposes. In 2002, he joined back EPFL to work in the Microelectronic Systems Laboratory to develop R&D activities on design flows and methodologies and on the use of languages for analog and mixed-signal design. Dr. Vachoux has been actively involved in the development of the analog and mixed-signal extensions to VHDL since 1992. He participated to the VHDL-AMS language design task and served as the Chair of the IEEE working group between 1997 and 1999. He conducted the IEEE ballot process until the first release of the IEEE 1076.1 Standard in March 1999.