ADVANCES IN SPECIFICATION AND DESIGN LANGUAGES FOR SOCS

ADVANCES IN SPECIFICATION AND DESIGN LANGUAGES FOR SOCS
Selected Contributions from FDL'05

Edited by
ALAIN VACHOUX
Ecole Polytechnique Fédérale de Lausanne

Kluwer Academic Publishers
Boston/Dordrecht/London

Contents

List of Figures

List of Tables

Foreword

This is a foreword to this volume...

Alain Vachoux
FDL'05 General Chair
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland, February 2006

Acknowledgments

This book compiles the best contributions to the Forum on Specification and Design Languages (FDL) 2005 at EPFL in Lausanne (Switzerland) from September 26–30 2005

I

GENERAL TOPICS

Specification driven design, formal verification techniques, mixed formal and simulation-based verification techniques, formal languages (B, CTL, Z, temporal logic, etc.), synchronous languages (Esterel, etc.), modeling concepts (e. g. StateCharts, Petri Nets, FSMs, dataflow models, etc.), models of computation. There are two chapters in this part: **??** and **??**.

II

C/C++-BASED SYSTEM DESIGN

III

DESIGN OF ANALOGUE AND MIXED SIGNAL SYSTEMS: STILL BLACK MAGIC?

The design of analogue and mixed-signal (AMS) systems has—unfortunately—never been done in a really systematic way. Until now, analogue design is done rather bottom up and in an intuitive way. Therefore, the design of analogue circuits has often been compared with "black magic". The lack of methodology was—and is—acceptable for small, standalone analogue circuits that are functionally well separated from digital components.

Today's AMS systems no longer fulfil these conditions. Therefore, AMS designers face a number of challenges:

- The shrinking of analogue circuits causes increasing process variations. This requires a more complete and more systematic verification, especially applying Monte Carlo Simulation, Corner Case analysis and regression tests. However, for reliable results many simulation runs (100–100,000) are needed. Considering the run time for numerical analogue simulation, new methods like importance sampling, symbolic analysis or even formal verification might become interesting complements.

- Analogue circuits are more closely coupled and functionally linked with digital hardware or even software. Therefore, design and verification requires an overall system simulation. Despite attractive languages and simulators like VHDL-AMS or co-simulation environments, the mixed-domain and mixed-level modeling and simulation is still an issue and requires especially appropriate modeling and verification methodologies.

- Many requirements (very low voltage, very low power, etc.) are hard to meet by the well known analogue circuit topologies. Available tools support the dimensioning and optimization of given topologies, but lack support for the more creative topology design. This task requires expert and application knowledge. Analogue topology synthesis might solve the problem in the future. Today's designers must re-use the topologies once designed and adapt them to new requirements.

Using "black magic" from SPICE days for the design of AMS systems results in low design productivity and frequent re-designs. However, the application of new tools and languages can also be a challenge without the right knowledge, methodology and design flow. The following part of the book contains some chapters that describe successful application of methodologies and tools. They give the reader valuable hints for solving the issues mentioned above.

Chapter 1 deals with the abstract modelling of micro mechanical components for system level verification. Here, a behavioural model is created by reduced-order modelling methods and formulated in the language VHDL-AMS. This permits an overall system simulation with—despite the complexity—sufficient simulation speed.

Chapter 2 introduces a new kind of analysis that goes beyond simulation: semi-symbolic simulation. Although not yet available in commercial simulators, the methods described seem to be an appropriate approach to deal with increasing process variations, and to get better verification coverage.

The Chapter 3 entitled "SystemC-WMS: A Wave Mixed-Signal Simulator" introduces an extension to SystemC—originally intended for system-level analysis of HW/SW systems. This extension allows designers to include analogue circuits into the system level simulation, modelling the overall system in a single language—SystemC-WMS.

Beside the co-simulation based on specific languages, simulator coupling is an important issue. For simulator coupling especially the interfaces between different languages and simulators requires a lot of effort. The Chapter 4, "Automatic Generation of Verification Platform for Heterogeneous System Designs", gives an overview of an approach that supports the automatic generation of interfaces.

Last, but not least, the application of UML for re-use of analogue circuits is introduced in the Chapter 5, "UML/XML-Based Approach to Hierarchical AMS Synthesis".

Christoph Grimm
FDL'05 AMS Chair
University of Hannover
Hannover, Germany, February 2006

Chapter 1

CREATING VIRTUAL PROTOTYPES OF COMPLEX MEMS TRANSDUCERS USING REDUCED-ORDER MODELLING METHODS AND VHDL-AMS

Torsten Mähne[1,3], Kersten Kehr[1], Axel Franke[1], Jörg Hauer[1], and Bertram Schmidt[2]

[1]*Robert Bosch GmbH*
Department CR/ARY
P. O. Box 106050
D-70049 Stuttgart
Germany
kersten.kehr@de.bosch.com
axel.franke@de.bosch.com
joerg.hauer@de.bosch.com

[2]*Otto-von-Guericke-University Magdeburg*
Institute for Micro- and Sensor Systems (IMOS)
P. O. Box 4120
D-39016 Magdeburg
Germany
bertram.schmidt@e-technik.uni-magdeburg.de

[3]*Ecole Polytechnique Fédérale de Lausanne (EPFL)*
Laboratoire de Systèmes Microélectroniques (LSM)
Bâtiment ELD, Station 11
CH-1015 Lausanne
Switzerland
torsten.maehne@epfl.ch

Abstract In this chapter the creation of "virtual prototypes" of complex micro-electro-mechanical transducers is presented. Creating these behavioural models can be partially automatised using a reduced-order modelling (ROM) method. It uses modal decomposition to represent the movement of flexible structures. Shape functions model the energy conservation and full coupling between the dif-

ferent physical domains. Both modal shapes and shape functions for strain energy and lumped capacitances of the structure can be derived in a highly automated way from a detailed 3D finite elements (FE) model available from earlier design stages. Separating the generation of the reduced-order models (ROM) from the same FE model but for different operation directions circumvents current limitations of the used ROM method. These sub models are integrated into a full model of the transducer. VHDL-AMS is used to describe additional strong coupling effects between the different operation directions, which are not considered by the used ROM method itself. The application of this methodology on a commercially-available yaw rate sensor as an example for a complex transducer demonstrates the practical suitability of this approach.

Keywords: micro-electro-mechanical systems (MEMS); surface micromachined (SMM) transducers, finite element method (FEM); model extraction; reduced-order modelling (ROM); modal decomposition; VHDL-AMS; geometry, circuit and system level simulation.

1. Introduction

Micro-electro-mechanical systems (MEMS) are characterised by a strong and non-linear interaction between the various physical domains. The consideration of only one domain during the design process is therefore not sufficient (Mehner, 2000). The physics of microsystems can be described using partial differential equations. These can be solved numerically with boundary or finite elements methods (BEM, FEM). This approach leads to very detailed models, which are used to determine the mechanical properties of flexible microstructures and the electrostatic field distribution between their electrodes, i. e. to support the design process of the different MEMS components. However, these models are, in terms of memory consumption and computing time, too expensive to be used for the simulation of the entire microsystem. Thus feedback effects of driving and sensing circuits can't be analysed at the detailed geometry level.

The whole system can only be described at higher levels of abstraction, like the circuit and system levels. On these levels only the degrees of freedom (DOFs) at the interfaces (ports) of the different components are of interest. The derivation of simplified and verified behavioural models is therefore necessary. Their manual creation is time consuming, error-prone and often implies strong simplifications (only 1^{st} and 2^{nd} DOFs). One better solution is the automatised generation of these models by extracting the necessary informations from detailed FE models that are already available from earlier design steps. This can be done using reduced-order modelling (ROM) methods that were in scope of several research efforts in the field of MEMS over the past years (Bechtold et al., 2003; Chen et al., 2004; Gabbay et al., 2000; Mehner et al., 2000; Reitz et al., 2004; Rudnyi et al., 2004; Rudnyi and Korvink, 2002).

The methods focus on various fields of application with different inter-domain coupling effects.

One of these ROM methods, developed by Lynn D. Gabbay, Jan Mehner et al., was evaluated regarding its applicability to complex MEMS that use electrostatic fields to excite the mechanical structure and to detect its movements. It was successfully used to create a fully coupled behavioural model of a commercially available micromechanical yaw rate sensor. A new approach is presented to circumvent the limitation of the chosen ROM method to structures moving primarily in one dominant direction. Separate models for the different operating directions are generated from the same FE model and afterwards coupled using VHDL-AMS to manually model the missing effects. The modelling flow is presented as well as a methodology to organise the models for efficient future maintenance and extension.

2. Theory of the Chosen Reduced-Order Modelling Method

The chosen ROM method (Bennini et al., 2001b) uses a weighted sum of n mode shapes (modal amplitudes q_i, eigenvectors $\varphi_i(x, y, z)$) of the mechanical structure to represent its deflection u:

$$u(t, x, y, z) \approx u_{\mathrm{eq}} + \sum_{i=1}^{n} q_i(t) \cdot \varphi_i(x, y, z) \tag{1.1}$$

where u_{eq} is the initial displacement caused by prestress in equilibrium state. Especially for MEMS a few eigenmodes are usually enough to accurately describe the dynamic response of the structure (Bennini et al., 2001a). The strain energy W_{mech} that is stored within the structure due to deflection or prestress is expressed as a function of the modal amplitudes q_i. Geometrical nonlinearities and stress-stiffening are considered by calculating the modal stiffness k_{ij} from the second derivatives of the strain energy with respect to the modal amplitudes:

$$k_{ij} = \frac{\partial^2 W_{\mathrm{mech}}}{\partial q_i \partial q_j} \tag{1.2}$$

The modal masses m_i and modal damping constants d_i are calculated from the eigenfrequencies ω_i of the modes i and the entries of the modal stiffness matrix k_{ij}:

$$m_i = \frac{k_{ii}}{\omega_i^2} \tag{1.3}$$

$$d_i = 2\xi_i \omega_i \cdot m_i \tag{1.4}$$

where ξ_i is the modal damping ratio of mode i. The modal damping ratios represent the fluidic damping of the structure and can be obtained from analytical

calculations (squeeze or slide film damping), numerical fluid dynamic simulations or measurements. The deflection of the mechanical structure changes the capacitances between the electrodes in a nonlinear manner. The capacitance C_{op} between the electrodes o and p is calculated as a function of the modal amplitudes and therefore provides the coupling between the mechanical and electrical quantities. The displacement current I_o through the electrode o can be calculated from the stored charge:

$$I_o = \frac{dQ_o}{dt} = \sum_p \frac{d\left(C_{op}(q_1, \ldots, q_n)(V_o - V_p)\right)}{dt} \tag{1.5}$$

where V_o and V_p are the voltages at the electrodes. The governing equation describing the whole electrostatically actuated micromechanical structure in terms of modal coordinates is:

$$F_{\mathrm{M},i} = m_i\ddot{q}_i + d_i\dot{q}_i + \frac{\partial W_{\mathrm{mech}}(q_1, \ldots, q_n)}{\partial q_i} - \frac{1}{2}\sum_r \frac{\partial C_{op}^{(r)}(q_1, \ldots, q_n)}{\partial q_i}(V_o - V_p)^2 + \sum_j \varphi_{ij}\lambda_j \tag{1.6}$$

where $F_{\mathrm{M},i}$ is the modal force and r the number of capacitances involved between the multiple electrodes. The λ_j are the reaction forces to the external forces $F_{\mathrm{N},j} = -\lambda_j$ at the master nodes j of the FE mesh that remain accessible in the behavioural model.

The reduced-order model (ROM) consists of the equations (1.1), (1.5) and (1.6), which fully describe the static and dynamic nonlinear behaviour of the flexible structure and its nonlinear coupling to the electrostatic domain. All missing parameters of the ROM can be derived from a detailed fully coupled FE model of the MEMS component in a highly automated manner. The eigenvectors φ_i and eigenfrequencies ω_i of the considered modes i are taken from the modal analysis of the mechanical structure. The shape function of the strain energy $W_{\mathrm{mech}}(q_i)$ as well as the functions of the capacitances $C_{op}(q_i)$ are expressed in a polynomial form. They are fitted to a set of sample points of strain energy and capacitances extracted from a series of static analyses of the FE model in which the structure is deflected to various linear combinations of its mode shapes. The ROM-Tool available in ANSYS/Multiphysics since Release 7 is one implementation of this method (ANSYS, 2002).

3. Micromechanical Yaw Rate Sensor

The yaw rate sensor (Figure 1.1) developed by Robert Bosch GmbH is manufactured in a surface micromachining (SMM) process (Funk, 1998). The mechanical part of the sensor consists of a flat polysilicon structure. This rotor

is fixed to the centre by a X-shaped spring and thus movable around all three axes of the coordinate system. Comb drive structures, which are placed in pairs at its perimeter, excite and detect the in-plane oscillation of the rotor around the vertical z-axis. If the whole chip with the oscillating rotor is rotated around the x-axis with the angular rate $\omega_{\mathrm{i},x}$ the law of the conservation of the angular momentum causes a torque $M_{\mathrm{D},y}$ around the y-axis:

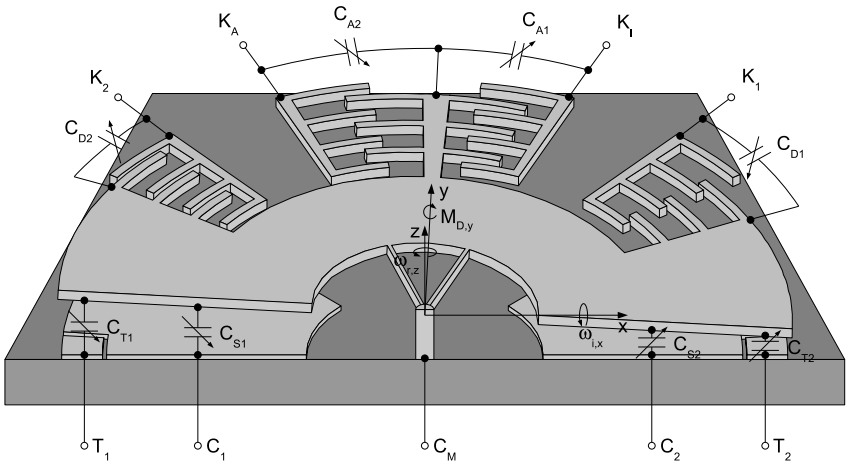
$$M_{\mathrm{D},y} = \left(J_z + J_y - J_x\right) \omega_{\mathrm{i},x} \omega_{\mathrm{r},z} \tag{1.7}$$

where J_x, J_y and J_z are the moments of inertia of the rotor around the axes of the coordinate system and $\omega_{\mathrm{r},z}$ is the current angular speed of the rotor. The rotor starts a tilting oscillation around the y-axis with an amplitude proportional to the angular rate $\omega_{\mathrm{i},x}$. This out-of-plane movement of the rotor is detected electrostatically using the electrodes below the structure.

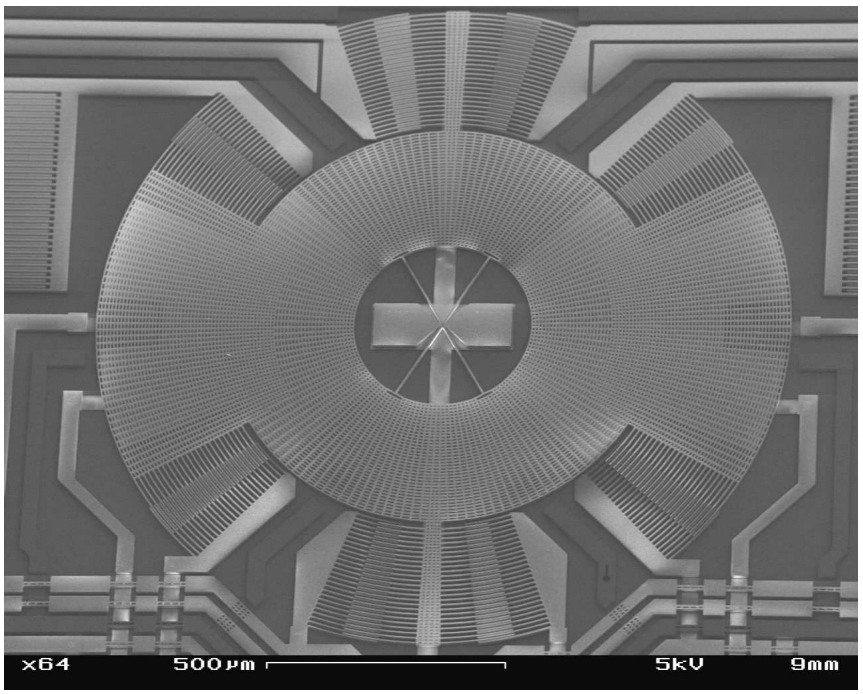
4. Preparation of Coupled FE Models for the ROM Method

To apply the described ROM method detailed FE models of the yaw rate sensor are needed, which model the structural as well as the electrostatic domains. To that end, the structural model available from earlier design steps of the device has to be extended to describe also the capacitances between the electrodes. To circumvent the limitation of the ROM method to systems moving primarily in one direction, two separate ROMs have to be generated for the yaw rate sensor: one representing the in-plane movement of the rotor and the other the out-of-plane movement. The two models will be coupled on the circuit level. To limit the model size and conserve simulation time during the generation pass, the FE models should only contain the details necessary for the particular ROM. Hence, two separate coupled FE models of the sensor are created from the same underlying structural model: One modelling the comb drive capacitances to excite and detect the in-plane movement of the rotor and the other modelling the capacitances of the rotor to the electrodes below the structure used to detect its out-of-plane movement. This approach also overcomes some implementation related limitations of the ANSYS ROM-Tool regarding the complexity of the models, especially the number of considered modes (≤ 9), conductors (≤ 5) and master nodes (≤ 10), that can be transformed into ROMs.

To model the capacitances of the comb drive structures and the capacitances between the movable structure and the subjacent electrodes 1-D electromechanical transducer elements of the type `TRANS126` were used (ANSYS, 2002). The nodes of these lumped elements have structural DOFs (displacement and force) as well as electrical DOFs (voltage and current) to fully describe the interaction between the structural and electrostatic domain in their proximity. The capacitance-displacement functions of the transducer elements are derived



(a) Principle



(b) SEM photography

Figure 1.1 Yaw rate sensor manufactured by Robert Bosch GmbH

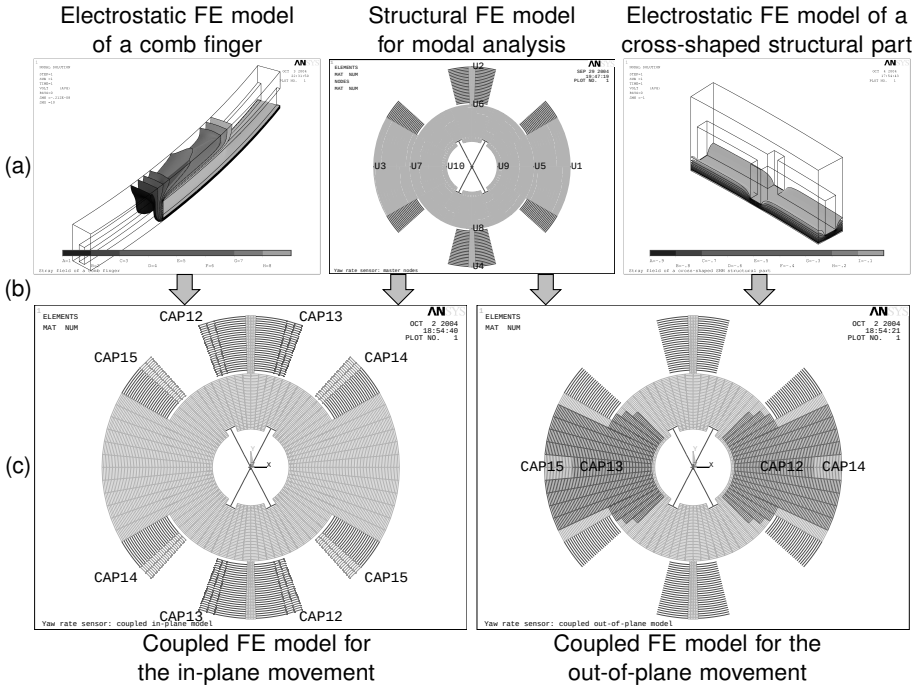


Figure 1.2 Steps to prepare the coupled FE models of the yaw rate sensor for the generation of its in-plane and out-of-plane ROMs: (a) single domain FE models, (b) preparation of the coupled FE models and (c) electromechanically coupled FE models

from an analytical approach using the parallel-plate capacitor assumption. The calculated characteristic curves can be corrected to account for the stray field using the results obtained from electrostatic field simulations with detailed cutaway FE models of one comb finger and a cross-shaped section of the moving SMM structure. In order to describe the total capacitance between two electrodes many transducer elements have to be connected in parallel, each describing one section of the space between the electrodes. In case of the comb drive structures and the capacitances below the rotor this results in one element at the end of each comb finger and at each cross section, respectively. Figure 1.2 illustrates all the steps necessary to prepare the coupled FE models for the ROM generation pass.

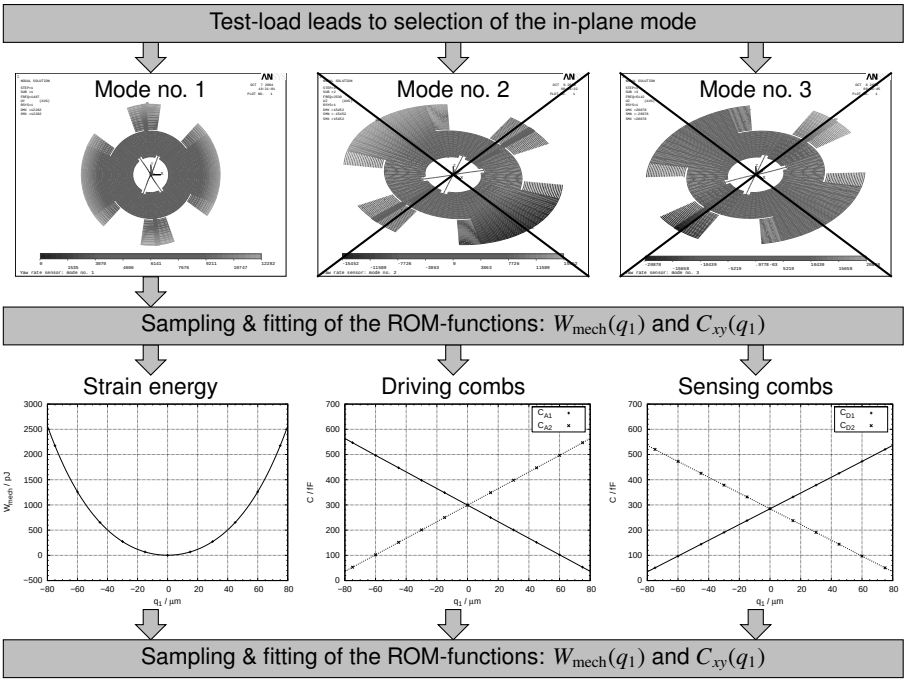
5. Generation of the Reduced-Order Behavioural Models

The prepared coupled FE models of the yaw rate sensor can now be simplified to the reduced-order behavioural models using the ANSYS ROM-Tool. This is done during the generation pass, which consists of a series of steps that are described in detail in (ANSYS, 2002; Mähne, 2004). Figure 1.3 illustrates the

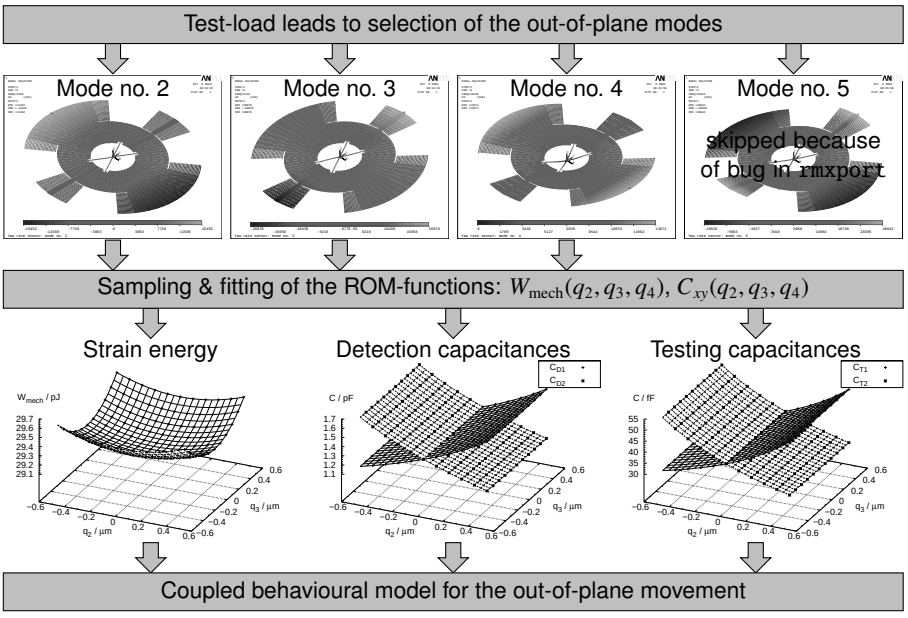
most important steps for the generation of the in-plane and out-of-plane ROMs of the sensor.

First the tool is initialised by establishing the main properties of the ROM: name of the FE model and its dimensionality (2-D or 3-D), working direction of the structure, capacitances and master nodes that should be retained in the ROM. Than a static test load is applied to the FE model to bring it in a typical deflection state that will be used to select the right modes for the ROM. The displacement of the structure through the test load is extracted from the FE model at the nodes that are selected to represent the neutral plane of the structure and stored in a file for later use. With a modal analysis of the structure its first nine eigenfrequencies and mode shapes (represented by the eigenvectors of the nodes in the neutral plane and of the master nodes) are extracted. After these preparatory analyses the modes for the ROM are selected automatically by calculating the contribution factor of each mode shape to resemble as close as possible the deflection state of the structure through the test load (Figure 1.3). The first eigenmode (oscillation around the z-axis) is identified as the only one contributing to the in-plane movement of the rotor and therefore selected for the in-plane ROM. All extracted higher eigenmodes contribute to the out-of-plane movement of the rotor but the calculated contribution factors show that the lower eigenmodes are dominating. These are the modes no. 2 and 3 (tilting oscillation around the y- and x-axes) as well as the modes no. 4 and 5 (butterfly shaped oscillation of higher order around the y- and x-axes). Due to a bug in the VHDL-AMS export function of ANSYS Releases ≤ 8.1, only three modes can currently be considered in the ROMs. For this reason only the modes no. 2, 3 and 4 are selected for the out-of-plane ROM. This does not influence the modelling of the sensor effect (conservation of the angular momentum) itself but the modelling of higher order interfering effects like prestress and direct structural coupling between the different operation directions. The most time consuming step is the following sampling pass where the strain energy and capacitances of the FE model are extracted for certain compositions of the scaled mode shapes. The sample routine had to be reimplemented to allow the extraction of the capacitances from the `TRANS126` elements. To this set of sample points the polynomials for the strain energy and capacitances are fitted. Type and order of the polynomials can be chosen individually for each shape function. The fitting step concludes the generation of the ROMs. They can be used afterwards within ANSYS (element type `ROM144`) or exported to an equivalent VHDL-AMS behavioural model (Schlegel et al., 2005).

The complete generation pass is automatised using APDL scripts so that the generation of the in-plane and out-of-plane ROMs can be run overnight in batch-mode. This is necessary since each mode was captured using 11 samples giving just 11 static analyses for the in-plane ROM but $11^3 = 1331$ static analyses for the out-of-plane ROM. Since each static analysis takes roughly



(a) In-plane model



(b) Out-of-plane model

Figure 1.3 Steps to generate the reduced-order models of the yaw rate sensor

1 min on recent PCs (P4 Xeon, 2.8 GHz, 4 GB RAM, Linux) both ROMs are generated within 23 h of CPU time.

6. Integration of the Reduced-Order Behavioural Models

The exported VHDL-AMS description of the ROMs are stored in a number of packages and entities that follow a fixed naming scheme (Mehner, 2004). Therefore they have to be separated into different design libraries. Each ROM defines a package `initial` that contains its characteristic constants, namely modal masses, modal damping ratios and eigenvectors of the master nodes. The polynomials for the strain energy $W_{\mathrm{mech}}(q_i, q_j, q_k)$ and the capacitances $C_{op}(q_i, q_j, q_k)$ are defined in separate packages called `s_ams_`ijk and `ca`op`_ams_`ijk, respectively, each defining the type of the particular polynomial, a flag if it should be inverted, the order of the polynomials with respect to the modal amplitudes q_i, q_j and q_k, scaling factors for the functions to overcome numerical problems, the number of polynomial coefficients and the coefficients themselves.

The entity `transducer` (the `ROM144` functional blocks in Figure 1.4) describes the interface of the ROM. Each DOF of the `ROM144` element is mapped to one of the across or through quantities of the terminals of the entity. At the modal terminals the modal amplitude q_i and modal force $F_{\mathrm{M},i}$ are available for the chosen modes. The master node terminals provide the displacement u_i and the inserted forces $F_{\mathrm{N},i}$ at these nodes. At the electrical terminals the voltages V_i and currents I_i are available for the electrodes of the system.

The architecture `behav` of this entity implements the complete behavioural model of the ROM. It declares all across and through quantities for the terminals of the entity. It also defines a function `spoly_calc()`, which calculates the strain energy and capacitances as well as their first derivatives with respect to the modal amplitudes q_i, q_j and q_k using the information of the polynomials defined in the packages `s_ams_`ijk and `ca`op`_ams_`ijk. The ordinary differential equations (1.1), (1.5) and (1.6) that describe the ROM are directly included in the architecture body as simultaneous statements using the `'dot` attribute to describe the derivatives with respect to the time.

An interesting detail of the ROMs is the use of an own system of units called µMKSV based on µm, kg, s and V. Its use is recommended for ANSYS to overcome numerical problems in MEMS FE models (ANSYS, 2002). Since the ROM method does not change the system of units of a model it has to be declared for the VHDL-AMS models too. The use of the IEEE standard packages for electrical (`ieee.electrical_systems`) and mechanical systems (`ieee.mechanical_systems`) is not possible since they are based on the SI system of units that is used for most other models on the circuit and system level. Hence, a new package `uMKSV` has been created that declares all important mechanical and electrical quantities as subtypes of `real` using their

own tolerance groups. The attributes `unit` and `symbol` are defined for each subtype naming their unit in a long and short form. These attributes document the declarations and are used for presentational purposes such as naming the axes in the signal plots of the simulator. The declared subtypes and natures can be named the same as their counterparts in the IEEE packages, allowing easy switching between the system of units, but only if the global name handling is implemented correctly within the VHDL-AMS simulation environment. Otherwise additional prefixes have to be used to prevent name clashes. Converter entities were implemented as an interface between the μMKSV and SI systems of units for the electrical (voltage, current) and mechanical quantities (displacement, force). They convert the across and through quantities between the terminals for the different system of units so that Kirchhoff's law remains valid.

To describe the complete yaw rate sensor the two generated ROMs of the structure have to be coupled to model the conservation of the angular momentum and a direct structural coupling between the operating directions of the sensor (Figure 1.4). The interface to this new entity representing the complete sensor consists of the electrical terminals that correspond to the pads on the sensor chip and three quantity input ports for the chip yaw rates $\omega_{\mathrm{i},x}$, $\omega_{\mathrm{i},y}$, $\omega_{\mathrm{i},z}$ around the x-, y- and z-axes, respectively. The coupling is done in the architecture `behavioural` using the VHDL-AMS support for combining structural and behavioural descriptions in a single architecture. One instance of the in-plane and one instance of the out-of-plane ROMs are created and their electrical terminals are connected through the system of units converter entities to the corresponding external electrical terminals. Since the external terminals of the sensor model use the declarations from `ieee.electrical_systems` it is fully compatible to other models of electrical components that use the SI system of units.

To model the conservation of the angular momentum the state of motion of the rotor with respect to the sensor chip as well as the one of the sensor chip itself with respect to an inertial coordinate system has to be known. The state of motion of the rotor is calculated from the first three modal amplitudes. These are directly proportional to the deflection angels $\sigma_{\mathrm{r},z}$, $\sigma_{\mathrm{r},y}$, $\sigma_{\mathrm{r},x}$ of the rotor around the z-, y- and x-axes, respectively. The first and second derivatives of the deflection angles with respect to the time give the current angular velocities $\omega_{\mathrm{r},z}$, $\omega_{\mathrm{r},y}$, $\omega_{\mathrm{r},x}$ and angular accelerations $\alpha_{\mathrm{r},z}$, $\alpha_{\mathrm{r},y}$, $\alpha_{\mathrm{r},x}$. The state of motion of the sensor chip itself is calculated from the yaw rate input signals by integrating and deriving them with respect to time to get the current rotation angles $\sigma_{\mathrm{i},x}$, $\sigma_{\mathrm{i},y}$, $\sigma_{\mathrm{i},z}$ an angular accelerations $\alpha_{\mathrm{i},x}$, $\alpha_{\mathrm{i},y}$, $\alpha_{\mathrm{i},z}$. With the known state of motion of the rotor and the sensor chip the torques M_x, M_y, M_z, can be calculated to

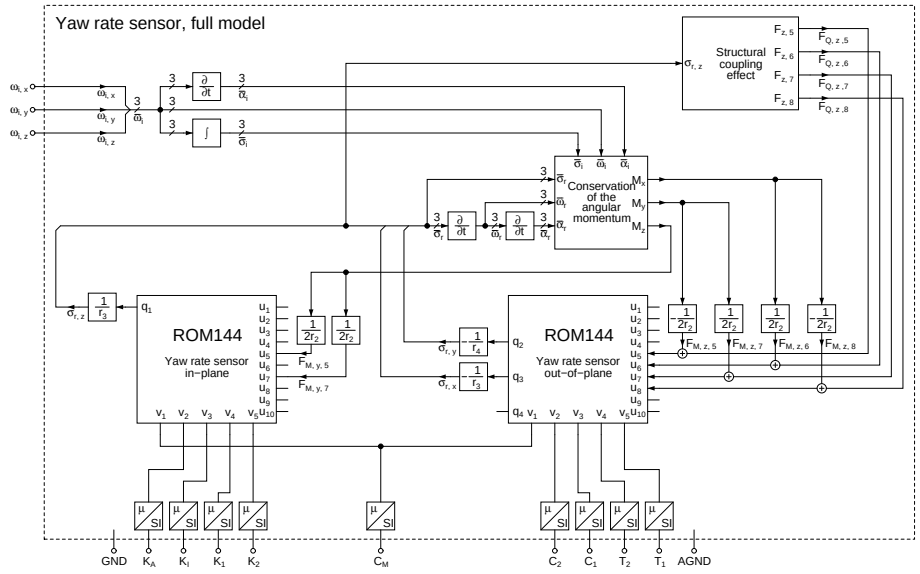


Figure 1.4 Structure of the yaw rate sensor full model with the in-plane and out-of-plane ROMs

conserve the angular momentum (Funk, 1998; Mähne, 2004):

$$\begin{aligned} M_x &= -\left[J_x\left(\alpha_{\mathrm{i},x} + \omega_{\mathrm{i},y}\omega_{\mathrm{r},z} - \omega_{\mathrm{i},z}\omega_{\mathrm{r},y}\right) + \left(\omega_{\mathrm{i},y} + \omega_{\mathrm{r},y}\right)\left(\omega_{\mathrm{i},z} + \omega_{\mathrm{r},z}\right)\left(J_z - J_y\right)\right] \\ M_y &= -\left[J_y\left(\alpha_{\mathrm{i},y} + \omega_{\mathrm{i},z}\omega_{\mathrm{r},x} - \omega_{\mathrm{i},x}\omega_{\mathrm{r},z}\right) + \left(\omega_{\mathrm{i},z} + \omega_{\mathrm{r},z}\right)\left(\omega_{\mathrm{i},x} + \omega_{\mathrm{r},x}\right)\left(J_x - J_z\right)\right] \\ M_z &= -\left[J_z\left(\alpha_{\mathrm{i},z} + \omega_{\mathrm{i},x}\omega_{\mathrm{r},y} - \omega_{\mathrm{i},y}\omega_{\mathrm{r},x}\right) + \left(\omega_{\mathrm{i},x} + \omega_{\mathrm{r},x}\right)\left(\omega_{\mathrm{i},y} + \omega_{\mathrm{r},y}\right)\left(J_y - J_x\right)\right] \end{aligned} \tag{1.8}$$

The torques are applied to the ROMs as force pairs at facing master nodes. The known deflection angles of the rotor can also be used to model the direct structural coupling between the in-plane and out-of-plane motion due to the non-rectangular sections of the beam springs supporting the rotor. To resemble the out-of-plane deflection of the rotor, forces $F_{\mathrm{Q},z,i}(\sigma_{\mathrm{r},z})$ are applied at the master nodes of the out-of-plane ROM. The relationships between forces and the in-plane displacement angle $\sigma_{\mathrm{r},z}$ are derived from the results of a series of static analyses of the structural FE model, to which polynomials are fitted. The implementation details of this effect are encapsulated in their own entity so that they can be changed easily to adapt the model e. g. to changes in the manufacturing process. The created sensor model could be easily extended to include further effects such as crosstalk between the electrical signal paths due to parasitic capacitances and resistors.

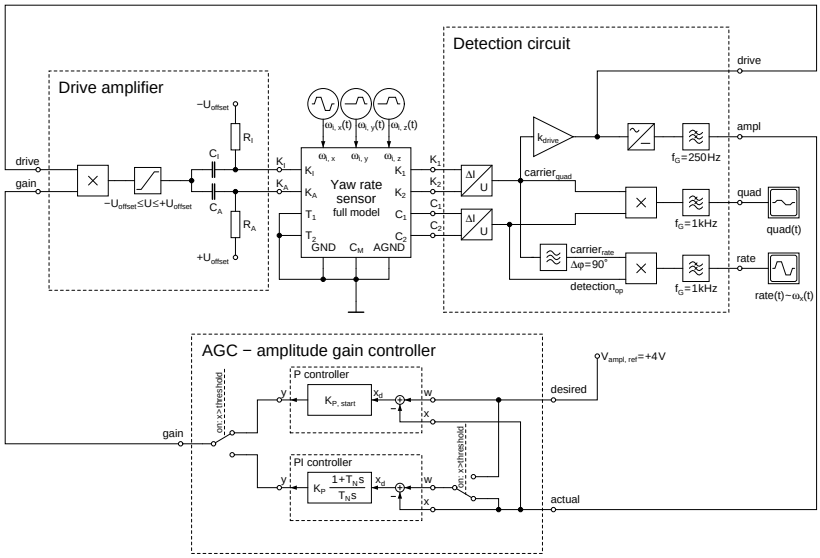


Figure 1.5 Structure of the testbench for the yaw rate sensor full model

7. Simulation of the Complete Sensor System

The created full model of the yaw rate sensor can be integrated in a testbench of the complete sensor system for verification (Figure 1.5). The testbench connects simple structural models of the drive amplifier, detection circuit and amplitude gain controller (AGC) to the sensor model. Although the models of the electrical components implement just their ideal behaviour they could be easily extended to be more realistic by adding a new architecture.

The created models of the sensor systems set high demands to the used VHDL-AMS simulator regarding its IEEE standard conformity and the quality of its solvers. This is a general problem when simulating MEMS and other multi-domain system that are common e. g. in the automotive sector (Haase, 2003; Schwarz, 2002). Because of the coupling of different domains very different time constants appear in the system creating stiff differential equations, which couple quantities of very different orders of magnitude. An additional problem is the need for integration of models using different systems of units as discussed in Section 6 creating the necessity of support for tolerance groups. For nonlinear systems with discontinuities in their description, like the switch discussed below, the support for the `break` statement to reinitialise the operating point is very important. Only the simulator SMASH of Dolphin Integration met all these demands and its version 5.2.1 was used for the analyses of the created model (Dolphin, 2003). Operating point and small signal analyses of the ROMs and the complete yaw rate sensor model, which were exposed to certain static

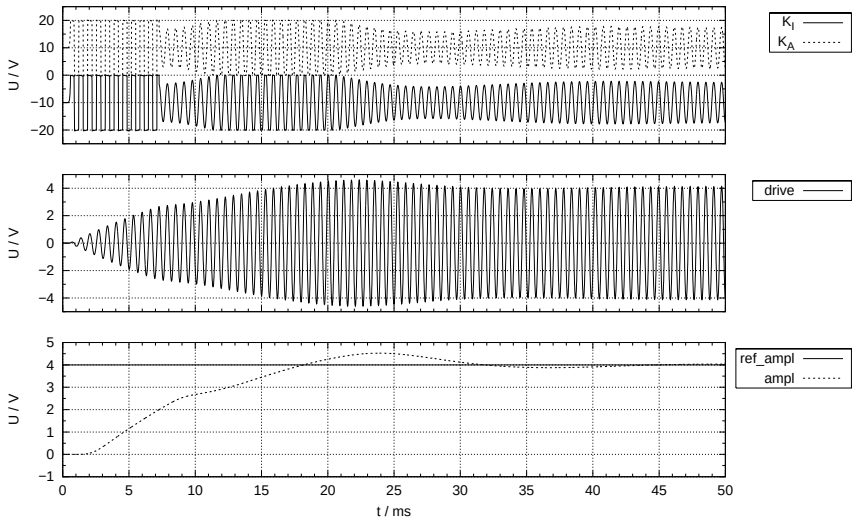
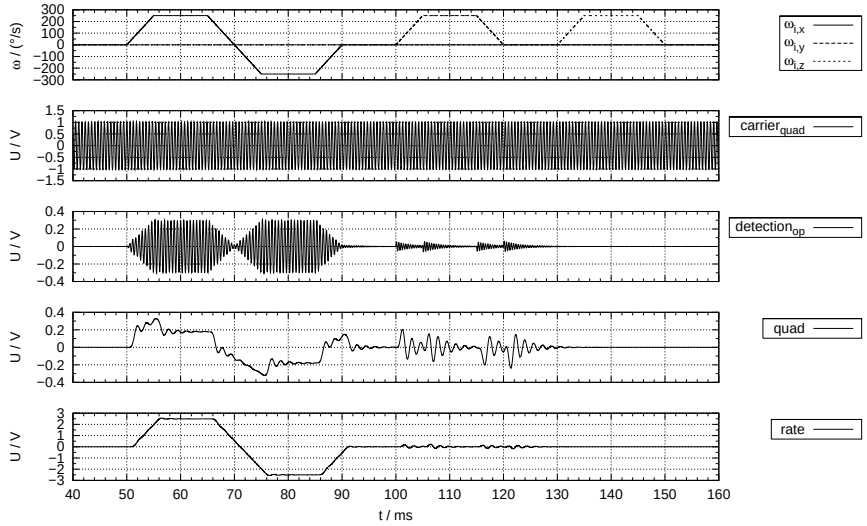


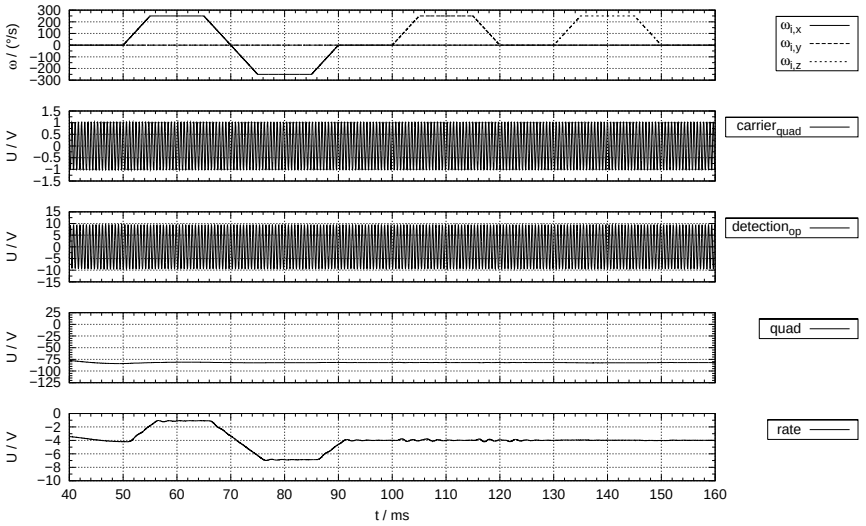
Figure 1.6 Simulated self-excitation of the yaw rate sensor

and small signal loads, showed that all important mechanical and electrical characteristic quantities (stiffness, masses, moments of inertia, eigenfrequencies, quality factors, capacitances) are very close (relative error ≤ 6 %) to the values of the coupled FE model and correspond well with the specifications of the real sensor element.

The drive amplifier uses a variable *gain* to amplify the detected *drive* signal and couples it capacitively on the drive combs. This signal excites the in-plane motion of the rotor. The movement is detected by measuring the displacement current at the detection combs, which is converted to a voltage and amplified to give the *drive* signal that is fed back to the input of the drive amplifier. The closed feedback loop leads to a resonant self-excitation of the in-plane movement that is stabilised using the AGC. It consists of two controllers: a P controller with a high proportional gain used to speed up the start-up phase and a PI controller with a lower proportional gain but with a reset time. The control is switched when the amplitude reaches a threshold so that the in-plane amplitude can stabilise on the desired value. The switch models need support for VHDL-AMS `break` statement within the used simulator to allow proper initialisation of the new operating point of the system. Figure 1.6 shows the results of the transient analysis of the self-excitation of the in-plane movement of the rotor. The first graph shows the feedback signals at the driving electrodes, the second one shows the detected in-plane signal and the third one shows the amplitude and reference signal at the AGC. The in-plane oscillation stabilises on the desired amplitude within 50 ms. After this start-up phase the



(a) Structural coupling effect neglected



(b) Structural coupling effect considered

Figure 1.7 Simulated rate detection of the yaw rate sensor

sensor is ready for detecting the yaw rate $\omega_{\mathrm{i},x}$. The results show also a frequency shift of the in-plane movement from the expected 1.487 kHz (first eigenfrequency of the rotor) to 1.680 kHz due to stress-stiffening of the X-shaped spring at large in-plane amplitudes and demonstrates the successful consideration of non-linear mechanical effects by the chosen ROM method.

The yaw rate $\omega_{i,x}$ at the input of the sensor causes an out-of-plane movement of the oscillating rotor, which is measured using the displacement currents at the subjacent detection electrodes. This out-of-plane signal contains the desired *rate* signal as well as the *quad* signal that is caused by the structural coupling between the in-plane and out-of-plane movement. The signals are 90° out of phase so that they can be clearly separated using synchronous amplitude demodulation with the *drive* signal as the demodulation carrier. It also allows a sign sensitive detection of $\omega_{\mathrm{i},x}$. The results of the transient analysis of this *rate* detection is shown in Figure 1.7 for two cases—one neglecting the structural coupling between the in-plane and out-of-plane movement and one considering the structural coupling. The first graph shows the yaw rate test signals applied to the sensor, the second one shows the *drive* signal used as demodulation carrier of the out-of-plane signal in the third graph, the fourth graph shows the *quad* signal, which corresponds to the amount of structural coupling between the in-plane and out-of-plane movement, and the fifth graph shows the demodulated *rate* signal. Without structural coupling the *rate* signal is the envelope of the detected out-of-plane signal. It also shows the low cross sensitivity of the sensor against yaw rates other than $\omega_{\mathrm{i},x}$. The small detected *quad* signal results from a cross demodulation error. The simulation results for the second case show that the out-of-plane movement caused by the yaw rate is much smaller than the much stronger movement caused by the direct structural coupling between in-plane and out-of-plane movement. However, the *rate* signal can still be extracted because of the fixed phase relationship between the two movements. The remaining offset error could be easily suppressed.

The results (Figures 1.6 and 1.7) agree well to older results published in Lorenz, 1999 and Reitz et al., 2004. The transient simulation of the whole sensor system including self-excitation and rate detection over a period of 120 ms with a maximum integration step width of 1 µs using SMASH took approximately 45 min on a recent PC (P4, 2.4 GHz, 512 MB RAM, Windows 2000). The newly created behavioural sensor model can be used to evaluate different circuit concepts (Funk, 1998; Rocznik, 2004) by creating new architectures and changing the configuration of the entities for the driving, sensing and control circuits or by creating entire new testbenches that integrate the sensor model.

8. Conclusions

In this paper a new approach for creating accurate fully coupled behavioural models (virtual prototypes) of complex MEMS was presented. A commercially available ROM method was utilised to automatically extract verified ROMs from available FE models of the component and VHDL-AMS to model missing effects. The whole modelling process was outlined using a micromechanical Yaw Rate Sensor as an example. It was shown how structural FE models available from earlier design steps can be extended to also model the coupling to the electrostatic domain and how the ROMs are generated from these prepared FE models using the ANSYS ROM-Tool. While the ROM method itself is still under research, the implementation used here is already useful, even though it has limitations. A careful partitioning of the problem and modelling of the missing coupling effects by hand can circumvent most of these limitations. This was demonstrated by generating separate ROMs for the different moving directions of the yaw rate sensor and their coupling on the circuit level. The modelling of the missing coupling effects can be done in a very natural way using VHDL-AMS powerful language constructs for behavioural and structural description. The realised partitioning of the full yaw rate sensor model offers an easy way to add/change the coupling effects between the different ROMs. The created full model of the sensor was included in a testbench for the complete sensor system adding the circuits for driving, sensing and controlling of the movement of the micromechanical element. Different analyses showed the successful modelling of all important mechanical and electrical properties of the sensor, the self-excitation of the in-plane movement, the yaw rate detection and the low cross sensitivity of the sensor. It was shown that VHDL-AMS is a good choice to model nonlinear, discontinuous and multi-domain systems even though their simulation imposes high demands on the used simulator, which were only met by SMASH for the presented project. In the future MEMS will get even more complex so that further research on ROM methods is required to improve the automation of the model extraction and extend the coverage of considered effects within the ROM. More effort is also needed on the tool side to improve the implementation and integration of the different program systems that are used in the MEMS design process.

Acknowledgments

The work presented in this paper was carried out within a diploma thesis project at the department CR/ARY of Robert Bosch GmbH in close collaboration with the IMOS of Otto-von-Guericke-University Magdeburg.

References

ANSYS (2002). *ANSYS 7.1—Coupled-Field Analysis Guide*. ANSYS, Inc.

Bechtold, Tamara, Rudnyi, Evgenii B., and Korvink, Jan G. (2003). Automatic generation of compact electro-thermal models for semiconductor devices. In *IEICE Trans. Electron.*, volume E86-C, pages 459–465. IEEE.

Bennini, Fouad, Mehner, Jan, and Dötzel, Wolfram (2001a). Computational methods for reduced order modeling of coupled domain simulations. In *11th International Conference on Solid-State Sensors and Actuators (Transducers 01)*, pages 260–263, Munich, Germany. IEEE.

Bennini, Fouad, Mehner, Jan, and Dötzel, Wolfram (2001b). A modal decomposition technique for fast harmonic and transient simulations of MEMS. In *International MEMS Workshop (IMEMS)*, volume 9, pages 477–484, Singapore.

Chen, Jinghong, Kang, Sung-Mo (Steve), Zou, Jun, Liu, Chang, and Schutt-Ainé, José E. (2004). Reduced-order modeling of weakly nonlinear MEMS devices with taylor-series expansion and arnoldi approach. *Journal of Microelectromechanical Systems*, 13(3):441–451.

Dolphin (2003). *VHDL-AMS in SMASH Release 5.2*. Dolphin Integration, 39, avenue du Granier, B. P. 65 ZIRST, 38242 Meylan, France.

Funk, Karsten (1998). *Entwurf, Herstellung und Charakterisierung eines mikromechanischen Sensors zur Messung von Drehgeschwindigkeiten*. Dissertation, Technische Universität München, München.

Gabbay, Lynn D., Mehner, Jan E., and Senturia, Stephen D. (2000). Computer-aided generation of nonlinear reduced-order dynamic macromodels—i: Non-stress-stiffened case. *Journal of Microelectromechanical Systems*, 9(2):262–269.

Haase, Joachim (2003). Anforderungen an VHDL-AMS-Simulatoren (Entwurf vom 1. Juli 2003). Technical report, Fraunhofer-Institut für Integrierte Schaltungen, Auenstelle EAS Dresden, Zeunerstrae 38, 01069 Dresden.

Lorenz, Gunar (1999). *Netzwerksimulation mikromechanischer Systeme*. Number D46 (Diss. Universität Bremen) in Berichte aus der Mikromechanik. Shaker Verlag, Aachen.

Mähne, Torsten (2004). Ordnungsreduktionsverfahren zur automatischen Generierung von Systemmodellen bei mikroelektromechanischen Systemen. Diplomarbeit, Otto-von-Guericke-Universität Magdeburg, Fakultät für Elektrotechnik und Informationstechnik, Postfach 4120, D-39016 Magdeburg.

Mehner, Jan (2000). *Entwurf in der Mikrosystemtechnik*, volume 9 of *Dresdner Beiträge zur Sensorik*. Dresden University Press, Dresden, München. Zugl.: Chemnitz, Techn. Univ., Habil., 1999.

Mehner, Jan (2004). *External System Simulation Based on VHDL-AMS*. CADFEM GmbH.

Mehner, Jan E., Gabbay, Lynn D., and Senturia, Stephen D. (2000). Computer-aided generation of nonlinear reduced-order dynamic macromodels—ii: Stress-stiffened case. *Journal of Microelectromechanical Systems*, 9(2):270–274.

Reitz, Sven, Döring, Christian, Bastian, Jens, Schneider, Peter, Schwarz, Peter, and Neul, Reinhard (2004). System level modeling of the relevant physical effects of inertial sensors using order reduction methods. In *DTIP of MEMS & MOEMS*, Montreux, Switzerland.

Rocznik, Marko (2004). *Optimierung des Entwurfs mikroelektromechanischer Drehratensensorsysteme*. Dissertation, Fakultät Elektrotechnik und Informationstechnik der Technischen Universität Ilmenau, Ilmenau.

Rudnyi, E. B., Lienemann, J., Greiner, A., and Korvink, J. G. (2004). mor4ansys: Generating compact models directly from ANSYS models. In *Technical Proceedings of the 2004 Nanotechnology Conference and Trade Show, Nanotech 2004*, volume 2, pages 279–282, Boston, Massachusetts.

Rudnyi, Evgenii B. and Korvink, Jan G. (2002). Review: Automatic model reduction for transient simulation of MEMS-based devices. In *Sensors Update*, volume 11, pages 3–33. WILEY-VCH Verlagsgesellschaft, Weinheim.

Schlegel, Michael, Bennini, Fouad, Mehner, Jan E., Herrmann, Göran, Müller, Dietmar, and Dötzel, Wolfram (2005). Analyzing and simulation of MEMS in VHDL-AMS based on reduced-order FE models. *IEEE Sensors Journal*, PP(99):1–8. Accepted for future publication.

Schwarz, Peter (2002). Modellierung und Simulation heterogener technischer Systeme. Technical report, Fraunhofer Institut für Integrierte Schaltungen Erlangen, Auenstelle Entwurfsautomatisierung Dresden, Zeunerstrae 38, D-01069 Dresden.

Chapter 2

MODELING UNCERTAINTY IN NONLINEAR ANALOG SYSTEMS WITH AFFINE ARITHMETIC

Wilhelm Heupke[1], Christoph Grimm[2], and Klaus Waldschmidt[1]

[1]*Department of Computer Engineering*
University of Frankfurt
Germany
heupke@ti.informatik.uni-frankfurt.de
waldsch@ti.informatik.uni-frankfurt.de

[2]*Institute of Microelectronic Systems*
University of Hannover
Germany
grimm@ims.uni-hannover.de

Abstract This chapter describes a semi-symbolic method for the analysis of mixed signal systems. Aimed at control and signal processing applications, it delivers a superset of all reachable values. The method that relies on affine arithmetic is precise for linear systems, but in the case of nonlinear systems, approximations are needed. As for each approximation a new term is added, the number of approximation terms increases during simulation and therefore slows down the simulation. This leads to a quadratic time complexity in the number of time steps. A method to avoid this and an example implementation based on SystemC-AMS are presented. Efficiency and time complexity of the improved semi-symbolic simulation are analyzed and discussed.

Keywords: affine arithmetic; intervals; SystemC-AMS; simulation; uncertainty; tolerance.

1. Introduction

Today's automotive, telecom, and ambient intelligence applications consist of sensors, actuators, analog and digital circuits, and a large portion of software. At the system level designers usually specify and model such applications by continuous-time block diagrams with directed signal flow. For the verification

and analysis of such systems, most notably a transient simulation is used: Input stimuli are specified and the simulator computes the output signals. The transient simulation allows designers to get important insight into the behavior of the modeled system and provides a basic functional verification. However, within the design process of many of the above mentioned systems, much time is spent for analyzing the impact of uncertainties:

- Static variations of operating conditions
  (e. g. production tolerances)
- Dynamic variations of operating conditions
  (e. g. temperature drift)
- Quantization and rounding errors in
  digital signal processing and analog/digital conversion
- Physical effects in analog circuits (e. g. noise).

One big problem is that some deviations are compensated and do not have a large influence on some output of interest, while another deviation of same magnitude will be amplified and thus violates the specification.

The established analog or mixed-signal simulators at the electrical level provide different methods that help designers to evaluate the impact of deviations: noise analysis, sensitivity analysis, worst case analysis, Monte Carlo analysis, AC analysis, and sometimes combinations thereof. These analyses are either based on the fact that analog circuits have a working point and can be linearized (AC analysis, noise analysis), require monotonicity (sensitivity analysis), or use a very high number of simulation runs to find corner cases (worst case simulation) and to compute statistical properties at the outputs (Monte Carlo analysis).

Although these analyses are very useful, they have several drawbacks: Methods based on linearizations are usually restricted to analog circuits and are not applicable to mixed-signal systems or even DSP software. In order to overcome these problems designers have to provide discrete models and additional models that are used for AC analysis. Furthermore, time domain simulations are used in combination with FFT methods to get information about the spectral distribution of noise, for example. Unfortunately, transient simulations and Monte Carlo methods do not provide information about the contribution of single sources of uncertainty to the total uncertainty at e. g. outputs in a direct and easy way; usually the interpretation is rather difficult.

The above mentioned classical analyses are aimed towards the electrical level and are based on linearization and linear equation solvers. The method proposed in this paper is intended for a system level simulation with a discrete time static dataflow model of computation, which is implied by the use of SystemC-AMS.

One should be aware that there is no automatic way to use the electrical level models at the system level or vice versa, yet.

Compared with purely numerical simulation, the symbolic or formal techniques provide designers with more information, e. g. about the origin of a deviation (Zhang and Mackworth, 1996; Henzinger, 1996; Hartong et al., 2002). Unfortunately, symbolic or formal techniques are far away from being applicable to the design of complex and heterogeneous systems (Stauner et al., 1997). In this situation, semi-symbolic techniques are very attractive if they combine advantages of symbolic techniques with the general applicability of simulation. A promising approach is the use of affine arithmetic (Andrade et al., 1994) for semi-symbolic analysis (Fang et al., 2003; Lemke et al., 2002) or even a semi-symbolic simulation (Heupke et al., 2003). A direct and easy interpretation is of particular interest for the case of design automation at system level.

Fang and Rutenbar (Fang et al., 2003) are doing a static analysis of rounding errors in DSP algorithms with affine arithmetic. In Heupke et al., 2003 and Grimm et al., 2004a we use affine arithmetic for semi-symbolic transient simulations of complex signal processing systems. The simulation result is a numerical output together with a symbolic, affine approximation of the contributions of different (symbolic) sources of uncertainty. An important advantage of the proposed method is the safe inclusion of all reachable values by the affine expression, therefore delivering reliable results. On the other hand the increasing number of terms and the resulting over-approximation caused by each nonlinear operation are a disadvantage.

In this chapter we introduce a method for semi-symbolic simulation with affine arithmetic that efficiently handles these approximation terms. Section 2 gives a brief introduction to affine arithmetic and semi-symbolic simulation with affine arithmetic as described in Heupke et al., 2003. Section 3 introduces a method to handle the over-approximation terms in semi-symbolic simulation based on affine arithmetic. This enables affine arithmetic to reach the same asymptotic time complexity conventional numerical simulation has. Section 4 demonstrates the applicability of the method by a simple example.

2. Semi-Symbolic Simulation with Affine Arithmetic

2.1 Basic Concepts of Affine Arithmetic

Affine arithmetic (Andrade et al., 1994), introduced by Comba et al., is a kind of improved interval arithmetic, and therefore allows us to compute with uncertain values. In each affine expression the influence of independent sources of uncertainty i to a variable $\hat{x}$ with the central value x_0 is represented by a symbolic sum of terms $x_i\epsilon_i$. Noise symbols ϵ_i represent arbitrary, but for one simulation run fixed values from the interval $[-1, 1]$. The partial deviations x_i

then scale these intervals. The ϵ_i are a symbolic representation and a certain value is never assigned to them.

$$\hat{x} = x_0 + \sum_{i=1}^{n} x_i \epsilon_i, \qquad \epsilon_i \in [-1, 1]$$

Basic mathematical operations are defined by

$$\hat{x} \pm \hat{y} := (x_0 \pm y_0) + \sum_{i=1}^{n} (x_i \pm y_i)\epsilon_i$$

and

$$c\hat{x} := cx_0 + \sum_{i=1}^{n} cx_i \epsilon_i$$

The operation $\mathrm{rad}(\hat{x})$ is the radius of the affine expression $\hat{x}$.

$$\mathrm{rad}(\hat{x}) = \sum_{i=1}^{n} |x_i|$$

The results of linear operations give accurate limits and have no over-approximation (no unnecessary expansion of the error interval).

2.2 Interval Arithmetic versus Affine Arithmetic

The subtraction of two affine expressions, which include the same noise symbols ϵ_m may reduce the partial deviation of the result, in contrast to the same values with different noise symbols. This allows us to model error cancellation, for example in feedback loops. In Table 2.1 the variables with a hat denote affine arithmetic variables while the ones written with a capital letter are corresponding interval variables. The diameter is obviously twice the radius for affine forms. In the case of intervals the diameter is the difference between supremum and infimum of the interval.

Table 2.1 shows the difference between affine arithmetic and interval arithmetic in the case of different or same source of uncertainty. The variables x and y share an uncertainty caused by the same source of uncertainty and therefore both have a term ϵ_1. For demonstration purposes also the influence of this uncertainty is of same magnitude and direction/sign. In contrast to that the variable z has a term ϵ_2 that shows that the uncertainty of z has a different source of uncertainty, although both have the same magnitude in example given in Table 2.1. The effect shows up in the subtraction of $x - y$. Interval arithmetic increases the interval diameter instead of bringing it to zero, while affine arithmetic keeps the correlation and delivers the precise result. This is because the

Table 2.1 Affine Expressions and their interval counterparts

affine arithmetic		*interval arithmetic*	
affine form	*diameter*	*interval*	*diameter*
$\hat{x} = 17.3 + 2.5\epsilon_1$	5.0	$X = [14.8, 19.8]$	5.0
$\hat{y} = 15.4 + 2.5\epsilon_1$	5.0	$Y = [12.9, 17.9]$	5.0
$\hat{z} = 15.4 + 2.5\epsilon_2$	5.0	$Z = [12.9, 17.9]$	5.0
$\hat{x} - \hat{y} = 1.9 + 0.0\epsilon_1$	0.0	$X - Y = [-3.1, 6.9]$	10.0
$\hat{x} - \hat{z} = 1.9 + 2.5\epsilon_1 - 2.5\epsilon_2$	10.0	$X - Z = [-3.1, 6.9]$	10.0

information contained in interval arithmetic is too limited, as the range of values is not the only important information that is needed to describe the influence of uncertainty.

This effect of interval arithmetic may be tolerated sometimes, but a simulation of a control loop, where a too pessimistic result is fed back in each time step, results in a diameter that is increasing with simulated time and depending on the system will increase exponentially in the worst case. This will deliver with interval arithmetic that the result is $[-\infty, +\infty]$ within a small number of simulation time steps (Heupke et al., 2003). For sure this is a safe inclusion, but would be useless pessimistic.

The concept described in this chapter can be extended to dynamic uncertainties and therefore to analyze effects like colored noise as described in Grimm et al., 2004b.

An important aspect is the guarantee that after each operation, the result is a superset of all reachable values. This is a problem especially for nonlinear operations. For nonlinear operations, different methods for over approximation are defined in Andrade et al., 1994. These methods add a new noisy symbol ϵ_{m+1} that describes the over approximation. For example, multiplication of two affine expressions is defined by:

$$\hat{x} \cdot \hat{y} := x_0 \cdot y_0 + \sum_{i=1}^{m} (x_0 \cdot y_i + x_i \cdot y_0)\epsilon_i + \mathrm{rad}(\hat{x}) \cdot \mathrm{rad}(\hat{y}) \cdot \epsilon_{m+1}.$$

In general, the error introduced by some nonlinear operation is over-approximated by a new noise symbol ϵ_{m+1}.

All nonlinear operations introduce new noise symbols and therefore some systems may present a problem, because of the permanently increasing number of terms. But some systems include strategies to reduce the influence of deviations (e. g. feedback). Caused by these strategies, the influence of these noise symbols converges to zero and for a stable system they are absolutely summable. Section 3 describes how this property delivers a solution for the problem of the increasing number of terms.

2.3 SystemC-AMS Based Implementation

For the implementation we chose SystemC-AMS (Vachoux et al., 2004), but the concept mentioned below can be implemented in every language that supports operator overloading, e. g., VHDL-AMS.

SystemC-AMS is an extension of the class library SystemC, aimed at supporting the modeling of mixed-signal (analog and digital) systems. It provides means to simulate analog, mixed-signal and signal processing systems as a block diagram. In contrast SystemC allows us to immediately reuse the code portion for these blocks, which have to be implemented in software later on. Additionally the code of the models that will be implemented in digital hardware, can be automatically synthesized to create, e. g., an ASIC or FPGA implementation. Only for the blocks, which model analog behavior, there is no automatic way for implementation. These blocks are specified by transfer functions or static nonlinear functions implemented in C++. Static dataflow is used as the model of computation.

The implementation of the affine arithmetic is based on the `libaffa` library (Gay, 2003), which defines linear and nonlinear operations on affine arithmetic variables in a class called `AAF` (affine arithmetic forms). It allows us to model computations with uncertainties in general.

Using the `AAF` class with SystemC-AMS is very simple: In SystemC-AMS, as in SystemC, signals are instantiated with a template parameter `T`. This parameter specifies the value type of the signal. For example by `sca_sdf_signal<double> my_signal` a signal with a value range of a variable with double precision is instantiated in SystemC-AMS. Of course, one can as well specify the template parameter `AAF` instead of **`double`**. This small change is all that is needed with SystemC-AMS to turn the numerical simulation into a semi-symbolic simulation based on affine arithmetic. Instead of using operators defined for **`double`** values, the compiler will use the operators defined in the `AAF` class, which overload the standard operators. The results of the simulation are affine expressions that semi-symbolically represent possible deviations.

For example, one can write the following code:

```
1 AAF a(2.0), b(3.0), c(2.0), y;
2
3 // constructor which adds a noise symbol
4 // x_i with partial deviation 0.1:
5 AAF uncertainty(Interval(-0.1, +0.1));
6 a = a + uncertainty;
7 y = (a + b) * (c + uncertainty);
8 cout << "y␣=␣" << y << endl;
```

This simple program produces the following output:

```
y = 10 + (e1*0.7) + (e2*0.01)
```

So after the uncertainty is introduced one can use a variable of type `AAF` like any other variable.

The advantage of semi-symbolic simulation compared with a pure numerical simulation becomes obvious if the uncertainties at the output of the simulated system exceed some specified range. In this case, the symbolic representation provides designers with the contribution of all sources of uncertainty to the deviation at the output. It also models the effects that are created by the combination of uncertainties. This together allows the designer to identify sources of uncertainty where improvements are most fruitful. As a long term perspective one day a mixed-signal synthesis system can be directed this way, where further optimization is needed.

3. Efficient Handling of Additional Terms in Feedback Control Systems

Each nonlinear operation approximation creates an additional term, as can be seen in the code example. These approximations are a problem for the affine arithmetic, as potentially a high number of very small and thereby insignificant terms in the symbolic expression is created.

This problem shows up especially if the system that is modeled contains a loop and this system has at least one component that creates an approximation in the path of this loop (e. g. by multiplication of two affine expressions).

Then any kind of memory (e. g. some modeled energy storage) in a block within the modeled system will contain most of the approximation terms that are created in each simulation cycle of this loop. If there is a constant number of approximations this means that in each simulation cycle the number of terms increases by this constant number.

To cope with that, we introduce a method that "cleans up" the affine arithmetic expression variables. It somewhat resembles the garbage collection concept, used to free unneeded memory of variables, which is used in some programming languages like e. g. Java.

If the number of noise symbols in the affine expressions increases above a certain level, the `simplification()` method is called. For all variables in the system, all terms smaller than a cut level l, set by the user, are summed up separately by the ones with a positive and the ones with a negative sign to two special noise terms.

Deleting the terms with an absolute value below this cut level could potentially lead to inaccurate results in the case of a high number of simulation time steps and certain functional blocks, e. g. integrators, because in this example they may grow to a big one over time. Therefore it is better to sum them. This way it delivers a safe inclusion, but it means that the correlation of the individual terms is lost. But it does not exhibit the same problem like interval arithmetic

does, as described above, because the correlation of this sum is still valid for all AAF variables in the future time steps and the terms with different signs are kept separate. Furthermore these uncertainties are usually far smaller than the nominal values and if l again is far smaller than the other uncertainties, any kind of over-approximation would not create a problem. So the influence of approximations decreases below the level l after several time steps.

Please note that if the simplification method would be called too often, the unneeded over-approximation could potentially show up significantly and in the above mentioned example the concept of feedback that makes these terms converge to 0, would not work. On the other hand if called not often enough, the computation time will increase significantly. Our experience is that the choice of the time point, when to call the simplification method, was for the example system not very critical.

The method resembles the typical strategy of leaving away smaller terms. But with affine arithmetic we do not have to really leave the smaller terms away, instead we can handle their sum as a new uncertainty. So not only the modeled uncertainties of the real system, but obviously also uncertainties caused by the modeling process, like these simplifications, are analyzed.

3.1 Implementation of the Simplification Method

In the present implementation the simplification method is invoked every $1{,}000^{\text{th}}$ simulation cycle, but later on it might be automatically invoked by some heuristic. For example the change in the highest index of the noise terms since the last simplification could be used as a criterion, when to call this method. The cut level l is set to a constant small fraction of the smallest explicitly introduced uncertainty by the user.

The change in an affine expression can be seen by the following example of a simplification with $l = 1.0 \cdot 10^{-4}$. First a variable was printed immediately before the simplification:

```
28.9796 + (e1*2.9925) + (e5*0.000856951)
+ (e6*1.14971e-006) + (e7*1.11085e-006)
+ (e8*-1.34821e-007) + (e9*1.07968e-006)
+ (e10*-1.12145e-007)
```

After the simplification the printed variable changed to:

```
28.9796 + (e1*2.9925) + (e5*0.000856951)
 + (e34*3.34024e-006) + (e35*-2.46966e-007)
```

Usually this happens with far more terms, but for demonstration purposes it would be difficult to show. In this case ϵ_{34} sums up the positive insignificant terms and ϵ_{35} sums up the negative insignificant terms.

By handling a list with pointers to all affine variables in the system, it is possible to access all `AAF` variables. This list is added as a static member of the `AAF` class, so that all `AAF` variables share it.

The `AAF` class saves the affine expression in one variable for the central value x_0 and two pointers to dynamically allocated arrays called `coefficients` and `indices`. In a first run across all `AAF` variables and across all coefficients in these variables, the significant terms are collected, based on the cut level l. A term x_i of an affine variable $\hat{x}$ is significant if it fulfills the condition:

$$|x_i| > l.$$

The second run goes again across all variables. For each of the variables it is determined how many significant terms are contained, based on the result of the first run. Then two new arrays for the coefficients and the indices are allocated and the significant terms are copied to the new arrays. After that the memory of the old arrays is freed.

3.2 Comparison of Efficiency

The following text analyses the effort to handle one variable. So the total effort also scales with the number of variables for all similar simulation methods.

Figure 2.1 shows a system we implemented as a test in order to validate the behavior by transient simulations with affine expressions as data type. It contains two elements that can be troublesome:

- The first is feedback. Another range arithmetic, the interval arithmetic Moore, 1966, will not deliver a meaningful result for the simulation of systems with feedback, while affine arithmetic works fine in this respect Heupke et al., 2003.
- The second is the emphasized nonlinear block in the system, which is interesting, as it creates additional approximation terms in each iteration through the loop. This results in a high number of terms that slows down the simulation more and more, if nothing is done about that.

Let us assume such a system with a loop, n be the number of total simulation time steps and c be a constant that describes the maximum number of nonlinear operations, along the path of the loop. Remember that these nonlinear operations add terms. Further let k be the number of explicitly introduced uncertainties.

With conventional simulation based on the static dataflow model of computation and with variables of type, e. g., **`double`** the space complexity is $O(1)$ and the simulation time is $O(n)$.

In contrast to that in the naive implementation the maximum memory needed for each affine variable is:

$$cn + k \subseteq O(n)$$

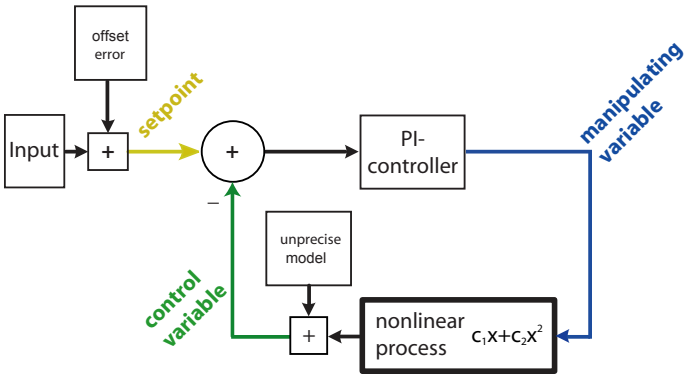


Figure 2.1 Simulated system with nonlinear block

because in each of the n steps c uncertainties are added, caused by over-approximations, and a maximum of k has been added intentionally at the elaboration phase. This means that the space complexity is $O(n)$.

Even worse is the resulting time complexity. This is because in each simulation time step each term of an affine variable needs to be handled, e. g. an arithmetic operation has to be performed for it by the CPU:

$$\sum_{i=1}^{n}(ci+k) = \frac{cn(n+1)}{2} + kn = \frac{c}{2}\cdot n^2 + \frac{c+k}{2}\cdot n \subseteq O(n^2)$$

System theory requires for every stable system that every bounded input delivers a bounded output. Obviously every technically meaningful system to be implemented is stable. Furthermore a discrete system is stable if and only if the impulse response is absolutely summable:

$$\sum_{i=-\infty}^{\infty} |h(i)| < \infty$$

This implies an important aspect: The impulse response of the opened control loop converges to zero. So every introduced over-approximation term will converge to zero with the number of iterations through the control loop in the given example. This means that we can apply a trick that copes with the terms caused by the over-approximation: From time to time we sum up all approximation terms that got extremely small (smaller than l) by a simplification method, thereby keeping the safe inclusion, but reducing the number of terms. On the other hand, this means, if the number of terms can not be reduced, we get a strong indication that the system might be unstable.

This simplification method, if called every m simulation time steps, is a substantial step forward regarding efficiency, because in the m time steps between

Table 2.2 Measured Computation Time

number of time steps	*computation time without simplification [s]*	*computation time with simplification [s]*
1,000	1	1
2,000	4	2
4,000	16	5
8,000	61	10
16,000	244	20
32,000	999	40
64,000	4,083	79
128,000	–	159
256,000	–	319
512,000	–	640
1,024,000	–	1,275

two simplification operations, a maximum of c terms adds in each time step. To this adds the number of k explicitly introduced terms. As c, m and k are all constants, we get asymptotically the same space complexity like pure numerical simulation:

$$cm + k \subseteq O(1)$$

The time complexity of the simulation with the simplification method needs $cm + k$ computations in one simulation time step in the worst-case of the time step before the next simplification method call. This happens in the worst-case n times. To this adds the effort of the simplification method, called n/m times. The simplification method itself needs in the first and the second pass to touch every term. This gives a total time complexity of $O(n)$, also the same complexity numerical simulation has:

$$(cm + k)n + \frac{2n}{m}(cm + k) = \left(cm + k + \frac{2}{m}(cm + k)\right) \cdot n \subseteq O(n)$$

4. Experimental Results

The system shown in Figure 2.1 was implemented in SystemC-AMS and the `AAF` class. With this setup transient simulation runs were performed.

Table 2.2 shows the time needed for the simulation with and without using the simplification method. The time interval that was simulated was the same for all values in the table and was scaled to deliver time results that are easy to interpret. Only the step width in time was changed for each row. The simplification method was called every 1,000$^{\text{th}}$ time step, respectively never in the case of no simplification.

The table shows the clear advantage of the simplification method, as the computation time increases linear with the number of simulated time steps, if the simplification method is used. It is very clear to see a quadratic increase of the needed computation time for the simulation without the use of the simplification method that shows up as a four fold increase of the required computation time for a two fold increase in the number of time steps. So it gets obvious that affine arithmetic would be much harder to use without this simplification method for long simulation runs in the presence of feedback and nonlinearity.

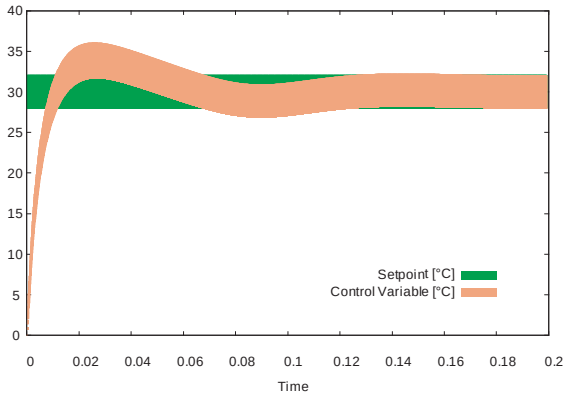
For a visual representation, we convert affine expressions to intervals, by use of the `rad` operator. These intervals can be plotted as shown in Figure 2.2 as a range. In the case of an uncertainty that is substantially smaller than the central value, two separate traces with different scalings are plotted. We use for both types of plots the program gnuplot, as usual waveform viewers do not support interval type signals. Figure 2.2a shows the step response of a feedback loop that contains a nonlinear control path, which is shown in Figure 2.1. Figure 2.2b shows the step response close to the stability border and Figure 2.2c the same system, but beyond the stability border. Interesting to note are typical chaotic effects of nonlinear systems near the stability border that show up very clearly in the uncertainty, and which are not linear with the central value in Figure 2.2b.

5. Conclusion

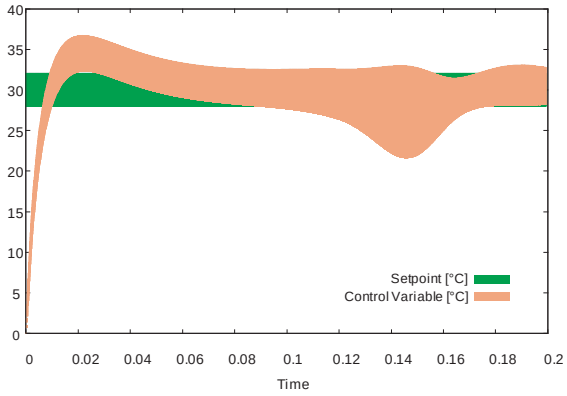
Without the described method semi-symbolic simulation with affine arithmetic has quadratic time complexity. On the other hand, with the presented method, simulation with affine arithmetic has linear time complexity, even in the presence of nonlinearities and feedback. This means that affine arithmetic is feasible for simulation even with a large number of time steps in nonlinear feedback systems.

Compared with purely numerical simulation these extensions allow designers to analyze the noise and sensitivity to different sources of uncertainty, such as thermal or quantization noise. Compared with analyses in “analog” simulators, the described method is applicable to digital signal processing methods and to discrete time approximations of analog circuits. This allows designers an analysis of heterogeneous systems that include large fractions of DSP software. The symbolic representation of the contributions to the deviations at the outputs can be interpreted easily and delivers a safe inclusion, an important aspect for the design of security critical systems, which could create otherwise dangerous situations if their deviation is too large.

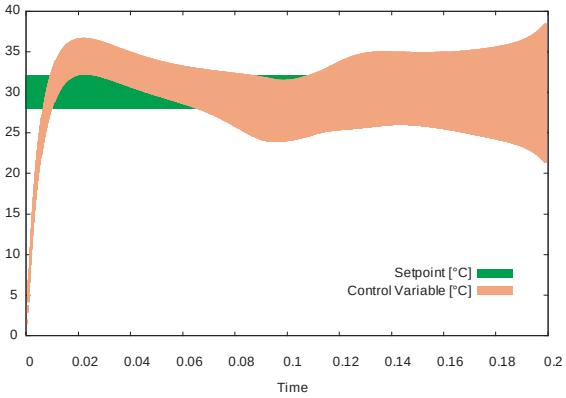
The improved efficiency of semi-symbolic simulation is a basic foundation for semi-symbolic simulation of more complex systems. For example, in Grabowski et al., 2006, semi-symbolic simulation is extended towards analog



(a) System within the stable area



(b) System near the stability border



(c) System beyond the stability border

Figure 2.2 Step responses of a feedback loop containing a nonlinear control path

circuit simulation. Without the method described in this paper, this would not have been possible.

References

Andrade, M. V. A., Comba, J. L. D., and Stolfi, J. (1994). Affine arithmetic (extended abstract). In *Proceedings of INTERVAL'94*, St. Petersburg, Russia.

Fang, C.F., Rutenbar, R.A., Püschel, M., and Chen, T. (2003). Towards efficient static analysis of finite-precision effects in DSP applications via affine arithmetic modeling. In *Design Automation Conference (DAC 2003)*, Anaheim, USA.

Gay, Olivier (2003). Libaffa—C++ affine arithmetic library for GNU/Linux. `http://savannah.nongnu.org/projects/libaffa/`.

Grabowski, D., Grimm, C., and Barke, E. (2006). Semi-symbolic modeling and simulation of circuits and systems. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS) 2006)*, Kos, Greece.

Grimm, Christoph, Heupke, Wilhelm, and Waldschmidt, Klaus (2004a). Refinement of mixed-signal systems with affine arithmetic. In *Design, Automation and Test in Europe 2004 (DATE'04)*, Paris, France.

Grimm, Christoph, Heupke, Wilhelm, and Waldschmidt, Klaus (2004b). Semi-symbolic modeling and analysis of noise in heterogeneous systems. In *Forum on Specification and Design Languages (FDL'04)*, Lille, France.

Hartong, Walter, Hedrich, Lars, and Barke, Erich (2002). Model checking algorithms for analog verification. In *Design Automation Conference (DAC 2002)*, New Orleans, Louisiana.

Henzinger, Thomas A. (1996). The theory of hybrid automata. *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*, pages 278–292.

Heupke, Wilhelm, Grimm, Christoph, and Waldschmidt, Klaus (2003). A new method for modeling and analysis of accuracy and tolerances in mixed-signal systems. In *Proceedings of the Forum on Design Languages (FDL'03)*, Frankfurt, Germany.

Lemke, Andreas, Hedrich, Lars, and Barke, Erich (2002). Analog circuit sizing based on formal methods using affine arithmetic. In *ICCAD 2002*.

Moore, R. E. (1966). *Interval Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey, USA.

Stauner, Thomas, Müller, Olaf, and Fuchs, Max (1997). Using HyTech to verify an automotive control system. In Maler, Oded, editor, *Hybrid and Real-Time Systems—International Workshop, HART'97*, volume 1201 of *Lecture Notes on Computer Science*, pages 139–153. Springer, Berlin.

Vachoux, Alain, Grimm, Christoph, and Einwich, Karsten (2004). Towards analog and mixed-signal SoC design with SystemC-AMS. In *IEEE Interna-*

tional Workshop on Electronic Design, Test and Applications (DELTA'04), Perth, Australia.

Zhang, Ying and Mackworth, Alan K. (1996). Specification and verification of hybrid dynamic systems with timed ∀-automata. In Alur, Rajeev, Henzinger, Thomas A., and Sontag, Eduardo D., editors, *Hybrid Systems III*, volume 1066 of *Lecture Notes on Computer Science*, pages 587–603. Springer, Berlin.

Chapter 3

SYSTEMC-WMS: MIXED SIGNAL SIMULATION BASED ON WAVE EXCHANGES

Simone Orcioni, Giorgio Biagetti, and Massimo Conti
Dipartimento di Elettronica, Intelligenza artificiale e Telecomunicazioni
Università Politecnica delle Marche
via Brecce Bianche, 12
I-60131 Ancona
Italy
SystemC-WMS@deit.univpm.it

Abstract This paper proposes a methodology for extending SystemC to mixed signal systems, aimed at allowing the reuse of analog models and to the simulation of heterogeneous systems. To this end, a general method for modeling analog modules using wave quantities is suggested, and a new kind of port and channel suitable to let modules communicate via waves have been defined. These entities are plugged directly on top of the standard SystemC kernel, so as to allow a seamless integration with the pre-existing simulation environment, and are designed to permit total interconnection freedom to ease the development of reusable analog libraries.

Keywords: SystemC; mixed signal simulation; system level simulation.

1. Introduction

SystemC is an emerging tool for the description and simulation of hardware and software at system level (OSCI, 2006), and it is not rare that this high level of abstraction could require the interfacing of both digital and analog parts. Such necessity of simulating a continuous-time analog part can arise, for example, in the area of power switching control as in the automotive or RF domains. To this aim it has been proposed (Einwich, 2002) to constitute an Open SystemC Initiative (OSCI) Working Group devoted to the development of an extension of SystemC to mixed-signal simulation: SystemC-AMS.

Vachoux et al., 2003 describes in detail the SystemC-AMS requirements and objectives. The first aspect considered is the need to encompass a variety of models of computation (MoCs), that can be used in order to describe any kind of system (discrete-event, data-flow, finite-state machines, analog signal flow, generic continuous-time, etc.). Furthermore, SystemC must also extend to heterogeneous domains of application (i. e. electrical, mechanical, fluidic), due to the increasing complexity of nowadays devices.

The OSCI Working Group claims that SystemC-AMS, besides being suitable for the description and the simulation of heterogeneous systems and supporting continuous-time MoCs, must also meet the following objectives: it must be an extension of the current SystemC; it must provide a (possibly generic) way to handle interactions between MoCs; it must provide appropriate views for the description of continuous-time models; and, finally, it must support the coupling with existing continuous-time simulators. A recent description of the state of the art of this initiative can be found in Vacoux et al., 2003.

The current implementation is structured into different layers. The solver layer provides simple but efficient solvers for linear differential equations and for explicit-form transfer functions. The synchronization layer provides a simple and fast synchronization scheme that executes analog solvers before the first delta cycle of each time step, scheduling them using static dataflow. Finally, a view layer provides means for specifying equations, for instance using netlists.

In addition to the activity performed by OSCI different papers (Bjørsen et al., 2003; Aljunaid and Kazmierski, 2004; Bonnerud et al., 2001), aimed to the extension of SystemC to analog environments, have been published. In Bjørsen et al., 2003 a mixed-signal simulation framework oriented to the simulation of signal processing dominated applications is presented. The library modules proposed do not provide a real continuous-time modeling but a discrete-time domain regulated by a virtual clock or a multi-phase clocking scheme. More recently, a mixed-signal extension using a general-purpose analog solver coupled with SystemC kernel via a lock-step synchronization algorithm has been proposed in Aljunaid and Kazmierski, 2004. This implied a modification of the SystemC 2.0 kernel to invoke and synchronize the operation of an analog solver together with that of the core kernel.

The basic SystemC methodology (OSCI, 2006) makes use of modules and interfaces to describe complex systems. Modules communicate through interfaces, implemented in channels, by calling methods in the channels themselves. Conversely, events occuring in a channel can activate modules connected to that channel. The present work proposes a methodology for the description of analog blocks using only such instruments and libraries. Taking advantage of this communication scheme and of the underlying SystemC kernel, we implement the various analog parts of a system as analog modules, which communicate by exchanging energy waves through wavechannel interfaces. The use of energy

waves permits the definition of a standard analog interface that allows the interconnection of modules belonging to different domains as well as of modules developed independently. Furthermore, interconnections of analog blocks giving rise to simple kinds of nonlinear Differential Algebraic Equations (DAEs) can also be simulated.

2. Description and Modeling of Analog Modules in SystemC

SystemC is essentially a library of C++ classes developed to build, simulate and debug a System on Chip described at system level. It provides an event-driven simulation kernel and the functionality of the system derives from the interaction of concurrent processes, that describe the behavior of individual modules subject to stimuli sent to them by other modules.

The core SystemC simulation paradigm assumes that modules have clearly defined inputs and outputs, and that they communicate between one another by means of appropriate channels. This paradigm allows the simulation to be carried out by a simple time-marching algorithm, that only needs to take care of interactions between modules and the channels directly connected to them, without the need of dealing with the global system topology.

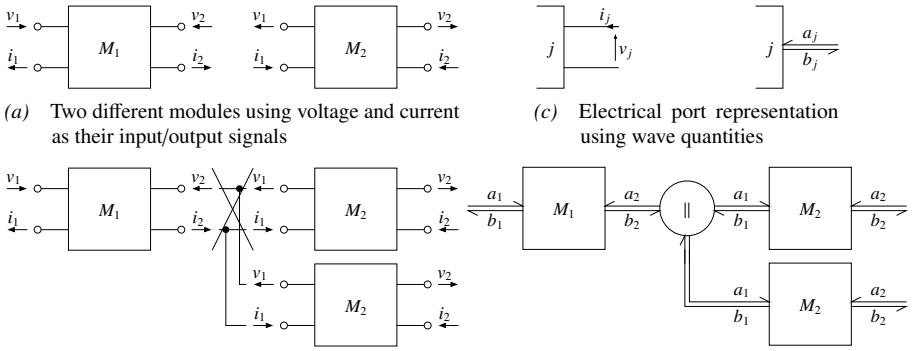
In order to be able to simulate systems containing analog modules some extensions to the base kernel are necessary. In cases where it is easy to obtain a signal flow graph representation of the system, this simulation paradigm can be coupled with an appropriate ODE solver as in Biagetti et al., 2004, thus enabling an efficient simulation of continuous-time analog modules described by a system of nonlinear ordinary differential equations of the following type:

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \\ \boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}) \end{cases} \tag{3.1}$$

where $\boldsymbol{f}$ and $\boldsymbol{g}$ are vector expressions describing system dynamics, while $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{u}$ are state, output and input vectors, respectively.

Equation (3.1) should describe a part of the system under consideration, like an N-port modeled at circuital level, or it may represent a high level macromodel describing the part functionality. This description is not able to take into account parts that need a DAE system to be described, neither conservative-law systems, however it is quite general, and will thus be used to describe the behavior of a single module.

Nevertheless, a signal-flow-graph representation is not always the most suited to model the interaction between modules representing analog units, since it can be hard to account for load effects or other interactions that might occur as they are interconnected. The goal of the next section is to propose an extension of



(a) Two different modules using voltage and current as their input/output signals

(c) Electrical port representation using wave quantities

(b) Some complex adaptor is needed to tie together the two output voltages

(d) Solution to the interconnection problem by a wave adaptor

Figure 3.1 Example of interconnection problem

this approach to cases where a signal-flow-graph representation of the modules is undesirable or excessively demanding.

2.1 Module Representation with a b Parameters

The first problem that needs to be solved is the possibility of interconnecting modules written independently. Figure 3.1a–3.1b depicts one possible problem that can arise trying to bind together electrical modules that use currents or voltages as their input/output signals. Whatever the designer's choice was regarding what to consider input or output, it would not be possible to simultaneoulsy be able to connect them in series or in parallel, as well as to cascade them.

Furthermore, in non-SFG representations there can be no physical clue on which quantity to consider input or output of a given module. Even if in principle it can be feasible to write a specialized channel that can handle all the possible combinations arising from a random choice, the resulting interconnection model would lack a physical meaning and would likely be cumbersome.

The use of an incident/reflected wave model (Kurokawa, 1965) for the description of analog modules allows us to avoid this difficulty since it can be mandated that modules use incident waves as inputs and produce reflected waves as outputs. This immediately solves the problem for cascaded modules, and the parallel or series connection can be accounted for by using an appropriate channel that dispatches waves to the modules it connects together, Figure 3.1d, and permits the formulation of a generic and standard analog interface usable across a variety of domains.

Such channel behaves similarly to the scattering junction of Wave Digital Filters (WDFs; Fettweis, 1973), which are digital models of analog filters, obtained

through the discretization of individual circuit components via a methodology that can also be extended to circuits in which mildly nonlinear elements are present (Sarti and De Poli, 1999).

The proposed approach makes use, like in the WDF theory, of the *a b* parameters as input/output signals and implements the duties of the scattering junction in a new entity called wavechannel, complying with SystemC conventions for channels. Furthermore, the user can choose the level of abstraction at which to model the system and the integration method (ODE solver) used to solve the continuous time system.

Without loss of generality, we can fix our attention to an N-port in the electrical domain, described through its port quantities v_j and i_j, $j = 1, \ldots, N$. Figure 3.1c depicts the situation for a single port. The relation between electrical quantities and wave quantities can be obtained from the following definition of incident (a_j) and reflected (b_j) wave:

$$\begin{aligned} a_j &= \frac{1}{2}\left(v_j/\sqrt{R_j} + i_j\sqrt{R_j}\right) \\ b_j &= \frac{1}{2}\left(v_j/\sqrt{R_j} - i_j\sqrt{R_j}\right) \end{aligned} \tag{3.2}$$

so that $a_j^2 - b_j^2$ is the instantaneous power entering port j and R_j is a normalization resistance. Similar relations hold for other domains as well. In the frequency domain, this representation leads to the commonly adopted description with a scattering matrix, and the normalization resistance can be assumed like the characteristic impedance of the transmission line connected to the port.

Solving the system Eq. (3.2) for the electrical quantities gives the inversion formulae:

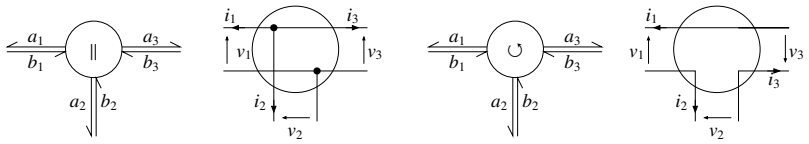
$$\begin{aligned} v_j &= \left(a_j + b_j\right) \cdot \sqrt{R_j} \\ i_j &= \left(a_j - b_j\right) / \sqrt{R_j} \end{aligned} \tag{3.3}$$

that can be useful when translating module descriptions from one set of quantities to the other.

Let us suppose that a port is defined by means of a relation of the type Eq. (3.1), where in the electrical domain $\{u, y\} = \{v, i\}$ (while in other domains we can find, for example, force and velocity or pressure and volume velocity as port variables). It is straightforward to build the representation of an N-port with the *a b* parameters by using Eq. (3.1) and Eq. (3.2), thus obtaining:

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{f}_1(\boldsymbol{x}, a) \\ b = g_1(\boldsymbol{x}, a) \end{cases}, \tag{3.4}$$

which are the state space equations written in wave quantities.



(a) parallel interconnection *(b)* series interconnection

Figure 3.2 Wavechannel symbols corresponding to parallel and series interconnections

2.2 Wavechannels

Wavechannels are the means by which modules described by wave quantities communicate. They can be thought of as a bunch of transmission lines connecting ports to a junction box, in which the lines are tied together, and their role is to model the scattering of waves that occures at the junction.

Consider a junction between N ports, each with its own normalization factor R_j. Let $\boldsymbol{v}$ and $\boldsymbol{i}$ be the voltage and current vector, respectively, and:

$$\begin{cases} \boldsymbol{A_v}\,\boldsymbol{v} = \boldsymbol{0} \\ \boldsymbol{A_i}\,\boldsymbol{i} = \boldsymbol{0} \end{cases} \tag{3.5}$$

be a complete and minimal set of Kirchhoff's equations describing the junction ($[\boldsymbol{A_v}]_{ij}, [\boldsymbol{A_i}]_{ij} \in \{0, \pm 1\}$). We maintain that letting:

$$\boldsymbol{A_x} = \boldsymbol{A_v} \operatorname*{diag}_{k=1,\ldots,N} R_k \quad \text{and} \quad \boldsymbol{A_y} = \boldsymbol{A_i} \operatorname*{diag}_{k=1,\ldots,N} 1/R_k \tag{3.6}$$

the scattering matrix $\boldsymbol{S}$ (such that $\boldsymbol{a} = \boldsymbol{S}\,\boldsymbol{b}$), by substituting Eq. (3.3) and Eq. (3.6) into Eq. (3.5), becomes:

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{A_x} \\ \boldsymbol{A_y} \end{bmatrix}^{-1} \begin{bmatrix} -\boldsymbol{A_x} \\ \boldsymbol{A_y} \end{bmatrix} \tag{3.7}$$

where $\boldsymbol{b}$ are the waves reflected by modules and thus entering the junction, whence $\boldsymbol{a}$ are scattered back from the junction to the modules thereby interconnected.

The above formulation can be used for any kind of junction. But, although there are many possible ways in which the lines can be tied together, the most common situation is to have parallel or series connections, as shown in Figure 3.2 for channels connecting three ports. From Kirchhoff's laws, a parallel connection is characterized by the equations:

$$\sum_{j=1}^{N} i_j = 0 \qquad v_1 = v_2 = \cdots = v_N \tag{3.8}$$

for which the scattering matrix described by Eq. (3.7) results in:

$$a_j = \frac{2\sum_{k=1}^{N} b_k/\sqrt{R_k}}{\sqrt{R_j}\sum_{k=1}^{N} 1/R_k} - b_j. \tag{3.9}$$

Similarly, for a series wavechannel we have:

$$\sum_{j=1}^{N} v_j = 0 \qquad i_1 = i_2 = \cdots = i_N, \tag{3.10}$$

which leads to:

$$a_j = b_j - \frac{2\sum_{k=1}^{N} b_k\sqrt{R_k}}{\sum_{k=1}^{N} R_k/\sqrt{R_j}}. \tag{3.11}$$

It may be worth noticing here that, if $N = 1$, equations (3.9) and (3.11) simply become $a_1 = \pm b_1$, and the two channel types are thus able to model the total reflection that takes place at an open circuit or at a shunt, respectively.

In the current implementation of wavechannels the propagation delay can be excluded, so that their connection to instantaneous blocks may result in the production of delay-free loops. This is accounted for by the standard SystemC delta cycle mechanism, which, without further intervention, would just use a fixed-point algorithm to search for the solution of the instantaneous loops, provided that the embedded ODE solver does not advance its state while iterating to find the fixed-point. The fixed-point solution is equivalent to the solution of Maxwell's equations in quasi-static conditions, i. e., when it is possible to model the circuit with lumped elements. The quasi-static condition is valid if the wave propagation delay τ is much smaller than the Δt used by ODE solvers. In our fixed-point solution method this τ is approximated with a null time.

Furthermore, to increase the convergence speed of the fixed-point algorithm, a damping effect has been introduced. This has been done on the basis that, in a time-marching simulation, states between successive time steps should not be very different, and thus the fixed-point solution may take advantage of a limitation in the amount of change allowed to the variables. Let $a^{(n)}[t]$ be the wave at the n-th delta cycle of the time step t. The evaluation of the module output functions, based on the values of the inputs $a^{(n)}[t]$ and of the state $x[t]$, yields the reflected wave $b^{(n+1)}[t]$. This is used in equations (3.9) and (3.11) to compute the scattering due to interconnections, let us call $\tilde{a}^{(n+1)}[t]$ the result. We then put:

$$a^{(n+1)}[t] = a^{(n)}[t] + \lambda(\tilde{a}^{(n+1)}[t] - a^{(n)}[t]) \tag{3.12}$$

where λ is a positive constant less than 1 (we obtained good results with values close to 0.9), governing the amount of damping. The update of a is skipped

altogether when the amount of change is below a predefined threshold related to the desired accuracy of the solution, so as to exit from the delta cycling and thus allowing the time to be incremented and the state of ODE solvers to be updated.

With this approach, it has been possible to obtain accurate simulations with a reasonable convergence speed of most of the systems containing delay-free loops provided they do not contain directly coupled state variables, that is, the circuit has a solution for every possible value of the state variables.

3. SystemC-WMS Class Library

To ease the implementation of complex systems containing analog blocks, a number of templates and classes have been designed and integrated in the SystemC-WMS (SystemC-Wave Mixed Signal) class library: a new kind of port to let modules communicate via wave quantities (`ab_port`), a channel that can interconnect them and that does the real computation of the scattering that occures at junctions (`ab_signal`), and a template base class (`wave_module`) that takes care of handling sensitivity lists and port declarations.

Ports expose an interface that allows users to read the incident wave value and to report (write) the reflected wave value, together with utility functions to poll for changes and to get other channel properties:

```
template <class T> struct ab_signal_if : virtual sc_interface
{
  virtual bool poll () const = 0;
  virtual const T read () const = 0;
  virtual void write (const T &) = 0;
  ...
};
```

The basic `wave_module` template looks like the following:

```
template <int n, class T> struct wave_module : sc_module, ...
{
  ab_port <T> port[n];
  sc_event activation;
  ...
};
```

where `port` is the array (or possibly a single variable if `n==1`) of ports used by the module to communicate. Of course, they can be freely mixed with standard SystemC ports. `activation` is an event that is signaled when some change occurs at the waves entering any of the ports, and the template parameter `T` must be associated to a structure, that essentially consists of a collection of typedefs, needed to define the underlying data type used for waves and to document the nature of the port. A number of predefined natures (electrical, mechanical, etc.) have been provided, and, of course, templates to ease the implementation of

transducers (that is, modules with ports of different natures) have also been defined and implemented.

With this library the only thing that the user needs to do in order to model an analog module is to implement the state derivative vector field f and output transformation function g as in Eq. (3.4):

```
struct example : wave_module <1, electrical>, analog_module
{
  // state variable x is inherited from analog_module
  void field (double *var) const;
  void calculus ();
  SC_CTOR(example) : analog_module(...)
  {
    SC_THREAD(calculus);
    sensitive << activation;
  }
};

void example::calculus ()
{
  x = 0;                      // state initialization here
  while (step()) {            // perform one ODE solver step
    double a = port->read(); // read incident wave here
    double b = g(x, a);       // compute reflected wave
    port->write(b);           // and send it out here
  }
}

void example::field (double *var) const
{
  double a = port->read();
  var[0] = f(x, a);           // evaluate state change
}
```

the `step` function is inherited from the `analog_module` and contains a simple time-marching ODE solver (Biagetti et al., 2004). Currently the user has a choice of Euler and Adams-Bashforth ODE solvers, but the implementation of other time-marching ODE solvers should be straightforward.

Finally, `ab_signal` takes care of making communication between modules possible. These signals can just be declared by specifying the nature of the ports and the kind of connection topology to make between them, with an optional default normalization resistance, for instance:

```
  ab_signal <electrical, parallel> test_signal_1(50 ohm);
  ab_signal <electrical, series>   test_signal_2(10 ohm);
```

and then connected to ports like ordinary SystemC signals.

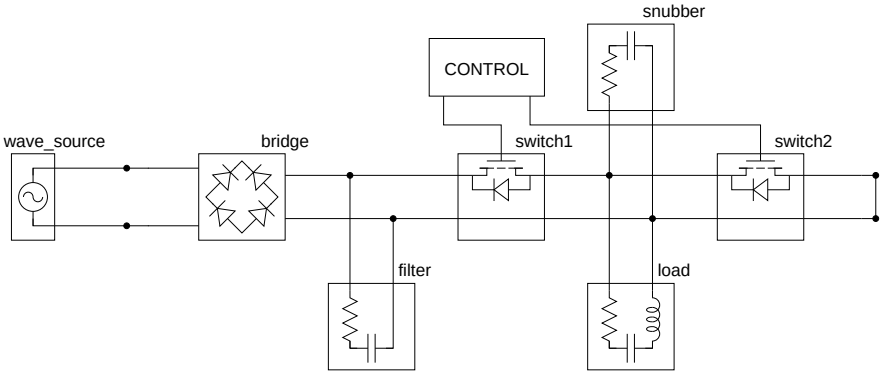


Figure 3.3 Half-bridge inverter: elctrical schematic diagram

4. Application Example

As an example of a possible application of this extension to a real problem a half-bridge inverter has been chosen. A simplified schematic of the circuit is shown in Figure 3.3. It is used to drive an *RLC* load for an induction-heating appliance. The function of the circuit is to regulate the delivery of power to a load. The amount delivered can be set by changing the duty cycle and/or the frequency of the signals controlling the two switches with a proper algorithm that can be implemented in digital hardware. Of course, the maximum output power corresponds to a 50 % duty cycle at the resonance frequency, but in the proposed example we have not modeled the details of the digital controller and have thus chosen a 48 % duty cycle, for safe operation of the switches, and a fixed frequency of 20 kHz.

The main components of the circuit are the switches (`controlled_rectifier`), the Graetz' bridge used to rectify the line voltage (`ideal_rectifier`), and the voltage source (`wave_source`) used to convert stimuli from standard SystemC signals or a signal flow graph representation to the wave representation. Furthermore, there are a couple of different passive reactive linear networks (`RC` and `RLC`). All of the analog stuff is connected together by means of wavechannels, as shown by the following code fragment illustrating the structure of the circuit under consideration. A brief description of the most important modules follows the code.

```
1 int sc_main (int argc, char *argv[])
2 {
3   sc_signal <electrical::wave_type> in;
4   sc_signal <bool> pulse1, pulse2;
5   ab_signal <electrical, parallel> mains;
6   ab_signal <electrical, parallel> rectified;
7   ab_signal <electrical, parallel> chopped;
```

```
8   ab_signal <electrical, series>   shunt;
9
10  generator signal_source("SOURCE1", 230 V, 50 Hz);
11  signal_source(in);
12
13  controller ctrl("CONTROL", 20 kHz);
14  ctrl(pulse1, pulse2);
15
16  source <electrical> wave_source("SOURCE2", cfg::across);
17  wave_source(mains, in);
18
19  ideal_rectifier bridge("BRIDGE");
20  bridge(mains, rectified);
21
22  RC filter("FILTER", 1 ohm, 5 uF);
23  filter(rectified);
24
25  controlled_rectifier switch1("SWITCH1");
26  switch1(chopped, rectified, pulse1);
27
28  controlled_rectifier switch2("SWITCH2");
29  switch2(shunt, chopped, pulse2);
30
31  RLC load("LOAD", 3 ohm, 80 uH, 740 nF);
32  load(chopped);
33
34  RC snubber("SNUBBER", 10 ohm, 10 nF);
35  snubber(chopped);
36
37  sc_start(150e-6, SC_SEC);
38  return 0;
39 }
```

The source class is a generic converter from standard SystemC signals to wave signals, and the second parameter to its constructor specifies an "across"-type (as opposed to a "through"-type) source, that is a voltage source in the electrical domain. The controlled_rectifier module models the behavior of an ideal switch, like a MOS switch with zero on resistance, coupled in parallel with a bypass ideal diode. It has been modeled as a 2-port module with an additional logical input to control the switch. For simplicity, the assumption that normalization resistances are the same for both ports has been made in its formulation, and so imposed in its constructor. That way, in its conducting state, whether it is due to the transistor switched on or to the diode, it simply lets waves through (like a transparent channel), otherwise it reflects them backwards (like a couple of open circuits). Its implementation is shown in the following:

```
1 struct controlled_rectifier : wave_module <2, electrical>
2 { // Ideal switch with integrated ideal diode
3   SC_HAS_PROCESS(controlled_rectifier);
```

```
4   controlled_rectifier (sc_module_name name);
5   void calculus ();
6   sc_in <bool> control;
7 };
8
9 controlled_rectifier::controlled_rectifier ...
10 {
11   SC_METHOD(calculus);
12   sensitive << activation << control;
13   // sets normalization resistances on both ports to be the ↵
         same:
14   port[0] <<= port[1] <<= 1;
15 }
16
17 void controlled_rectifier::calculus ()
18 {
19   double a0 = port[0]->read(), a1 = port[1]->read();
20   bool diode_on = a0 > a1, switch_on = control->read();
21   bool on = diode_on || switch_on;
22   port[0]->write(on ? a1 : a0);
23   port[1]->write(on ? a0 : a1);
24 }
```

When the switch is off, the detection of the state of the diode is done by looking at port voltages, but if the diode is off too, port voltages are proportional to incident waves, since $b_j = a_j \Rightarrow v_j = 2a_j\sqrt{R_0}$, so it is perfectly legal to test the latter ones. A similar formulation models the diode bridge in the `ideal_rectifier` module.

For what concerns the linear components, their are all modeled according to Eq. (3.4) and the example module reported in Section 3. In particular the equations governing the RLC circuit in wave quantities are:

$$\begin{cases} \dot{x}_0 = 2a\sqrt{R_0} - (R + R_0)x_0/L - x_1/C \\ \dot{x}_1 = x_0/L \\ b = a - x_0\sqrt{R0}/L \end{cases} \tag{3.13}$$

where the state vector $\boldsymbol{x}$ is composed of $x_0 = Li_L$ and $x_1 = Cv_C$ with obvious meaning of the symbols. This directly translates into the following state update and output computation functions:

```
1 void RLC::field (double *var) const
2 {
3   double a = port->read()
4   var[0] = 2*a*sqrt(R0) - state[0]*(R + R0)/L - state[1]/C ;
5   var[1] = state[0]/L ;
6 }
7
8 void RLC::calculus ()
```

```
9  {
10   R0 = port->get_normalization();
11   while (step()) {
12     double a = port->read();
13     double b = a - state[0]*sqrt(R0)/L;
14     port->write(b);
15   }
16 }
```

that, together with the obvious declaration of the `RLC` structure, complete the definition of the module. A model for the RC module can be derived similarly. The complete source code for this example, the full library, and other applications, are available from the authors' web site (Giorgio Biagetti and Orcioni, 2006).

4.1 Simulation Results

The circuit has been simulated using the proposed SystemC extension, which uses a fourth-order Adams-Bashforth ODE solver, and the results compared to a Matlab™ simulation done with the Simulink Power toolbox using the `ode15s` stiff ODE solver. Excessively long simulation times with the Matlab `ode113` solver (Adams-Bashforth), suited for nonstiff systems, led us to believe that testing both simulators with the same solver algorithm could be not very significant because of their different application contexts (which is a single module in our simulator).

Nevertheless, using an adaptive time step with the same maximum Δt of 5 ns, we obtained a simulation time of 11.3 s with SystemC-WMS and 2.3 s with Matlab™, both running on an Intel™ Pentium™ M processor at 1000 MHz.

Results are shown in Figure 3.4, where load current and chopped voltage are plotted as a function of time. The curves are completely overlapping.

5. Conclusion

The increasing complexity of systems and circuits asks for an easy way to model and simulate the overall behavior of a complex system spanning multiple domains. In order for SystemC to be able to cope with these requirements, an extension aimed at allowing the modeling and simulation of analog circuits is mandatory.

This work proposes an effective, and still not excessively complex framework, that simplify the modeling of the interaction between analog models belonging to heterogeneous domains, as well as model reuse. By using power waves as standard input/output signals for analog modules, these can be independently modeled and freely interconnected together in arbitrary topologies without having to deal with complex interface compatibility issues. Moreover, this

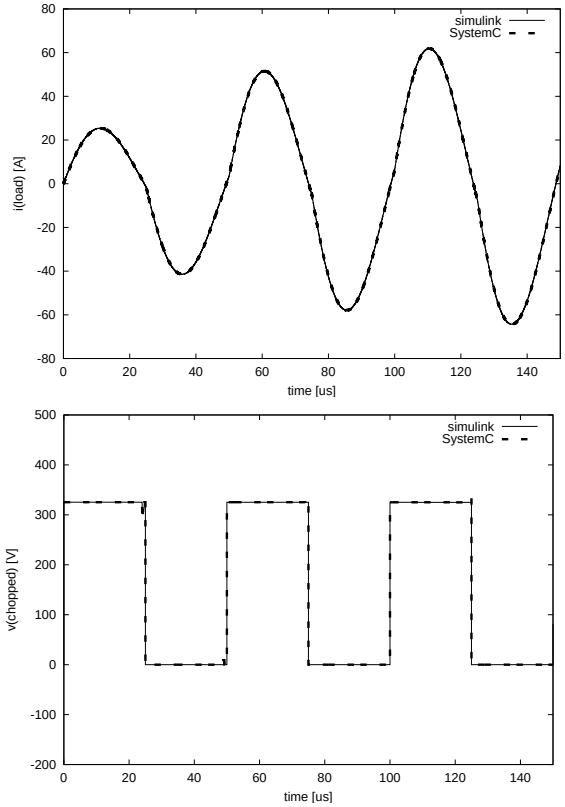


Figure 3.4 Simulation of the current into the load and voltage across the switches.

allows for a uniform treatment of heterogeneous domains, thus paving the way to the development of truly generic and reusable model libraries.

The first results are encouraging in terms of the accuracy of the simulation and, despite the simplicity of the algorithms employed, the variety of the class of circuits that can be simulated.

References

Aljunaid, H. and Kazmierski, T. J. (2004). SEAMS—a SystemC environment with analog and mixed-signal extensions. In *Proceedings of the 2004 International Symposium on Circuits and Systems (ISCAS'04)*, volume 5, pages V–281–V–284. IEEE.

Biagetti, Giorgio, Caldari, Marco, Conti, Massimo, and Orcioni, Simone (2004). Extending SystemC to analog modeling and simulation. In Grimm, Christoph, editor, *Languages for System Specification—Selected Contributions on UML,*

SystemC, System Verilog, Mixed-Signal Systems and Property Specifications from FDL'03, CHDL, chapter 15, pages 229–242. Kluwer Academic Publishers.

Bjørsen, Johnny, Bonnerud, Thomas E., and Ytterdal, Trond (2003). Behavioral modeling and simulation of mixed-signal system-on-a-chip using SystemC. *Analog Integrated Circuits and Signal Processing*, 34:25–38.

Bonnerud, T. E., Hernes, B., and Ytterdal, T. (2001). A mixed-signal, functional level simulation framework based on SystemC for system-on-a-chip applications. In *Proceedings of the 2001 IEEE Conference on Custom Integrated Circuits*, pages 541–544. IEEE.

Einwich, K. (2002). Analog mixed signal extensions for SystemC. White paper and proposal for the foundation of the SystemC-AMS OSCI working group, Fraunhofer-Institut für Integrierte Schaltungen IIS, Auenstelle Entwurfsautomatisierung EAS, `http://mixsigc.eas.iis.fhg.de/`.

Fettweis, A. (1973). Pseudopassivity, sensitivity and stability of wave digital filters. *IEEE Transactions on Circuit Theory*, CT-19:668–673.

Giorgio Biagetti, Massimo Conti and Orcioni, Simone (2006). SystemC-WMS home page. `http://www.deit.univpm.it/systemc-wms/`.

Kurokawa, K. (1965). Power waves and the scattering matrix. *IEEE Transactions on Microwave Theory and Techniques*, 13(2):194–202.

OSCI (2006). *SystemC Documentation*. The Open SystemC Initiative (OSCI), `http://www.systemc.org/`.

Sarti, Augusto and De Poli, Giovanni (1999). Toward nonlinear wave digital filters. *IEEE Transactions on Signal Processing*, 47(6):1654–1668.

Vachoux, Alain, Grimm, Christoph, and Einwich, Karsten (2003). SystemC-AMS requirements, design objectives and rationale. In *Proceedings of Design Automation and Test in Europe (DATE'03)*, Paris, France.

Vacoux, Alain, Grimm, Chistoph, and Einwich, Karsten (2003). Analog and mixed signal modelling with SystemC-AMS. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'03)*, volume 3, pages III–914–III–917. IEEE.

Chapter 4

AUTOMATIC GENERATION OF A VERIFICATION PLATFORM FOR HETEROGENEOUS SYSTEM DESIGNS

Suad Kajtazovic[1], Christian Steger[1], Andreas Schuhai[2], and Markus Pistauer[2]

[1]*Institute for Technical Informatics*
Graz University of Technology
Inffeldgasse 16/1
A-8010 Graz
Austria
kajtazovic@iti.tugraz.at
steger@iti.tugraz.at

[2]*CISC Semiconductor Design+Consulting GmbH*
Lakeside B07
A-9020 Klagenfurt
Austria
a.schuhai@cisc.at
m.pistauer@cisc.at

Abstract Complex microelectronic embedded systems are mostly subdivided into several subsystems, which are designed in different HDLs (Hardware Description Languages) to get best system performances. Verification of all subsystems in one environment presents a difficult task. One of the possibilities to solve this problem is using cosimulation techniques and standard HDL simulators. This chapter focuses on automatic building of cosimulation interfaces and model sources extracted from a HDL independent system description. Moreover, it presents a design methodology, which introduces advanced cosimulation techniques to be used in mixed-signal system design. The applied cosimulation technique allows set-up of system co-verification platforms by embedding different types of simulators into one environment. The proposed cosimulation technique and the automation methods have been applied on an application framework for multi-HDL system verification and later on evaluated by an example taken from the automotive industry.

Keywords: Heterogeneous System Design; Cosimulation; Design Automation; Distributed Computing.

1. Introduction

Verification plays an important role in design of microelectronic embedded systems that become more and more complex. The complex systems consist mostly of subsystems designed in different HDLs to get best performance provided by used HDLs (e. g. HW/SW subsystems). The verification of heterogeneous systems in one environment is very difficult. Currently, there are three possibilities to verify such systems. The first one is to translate corresponding subsystem descriptions from the foreign HDL into the target HDL, which can involve a degradation of model performance. The second possibility considers that the used simulator supports multi-HDL system design, which is preferred by many EDA tool vendors. A cosimulation presents the third possibility for the verification of heterogeneous systems. In a cosimulation, subsystems are simulated using HDL-specific simulators and the communication between them has been established by a cosimulation interface. In recent years use of cosimulation techniques becomes more and more interesting solution for the verification of heterogeneous system designs.

In this chapter we present an advanced cosimulation technique to be used in system design of microelectronic embedded systems. Automation in system design is one of the guidelines followed in this work. This work concentrates on design of a generic cosimulation interface and automatic building of a verification platform. Moreover, it presents a methodology to be used in multi-HDL system design.

This chapter is organized as follows. Section 2 describes cosimulation tools and methods, which are related to this work. The proposed design methodology has been described in Section 3. Section 4 describes the proposed generic cosimulation interface. One important topic in this work is the automatic code generation, which is described in Section 5. The proposed approach has been illustrated in Section 6 by an example taken from automotive industry. Section 7 concludes this work.

2. Related Work

In past years interfacing between different simulators and cosimulation techniques have been investigated intensively. Many different solutions have been found. A short overview about used technologies and tools is given here.

CosiMate (TNI-Software, 2005) is an EDA-tool, which uses a common cosimulation bus based on the open API interface. A graphical user interface supports the configuration of the cosimulation bus. Several simulators can be

coupled into one cosimulation environment: Matlab/Simulink, Saber, SystemC, VHDL/Verilog Simulators from MentorGraphics, Cadence, etc. CosiMate links these simulators via a configuration file and C interfaces to create the cosimulation bus.

Seamless (Mentor Graphics, 2005) provides a cosimulation with several ISS and HDL simulators. The cosimulation is based on the C-bridge API interface. The cosimulation backplane is tool-independent and supports several of today's popular simulators. The setup of the cosimulation must be done manually.

Link for ModelSim (The MathWorks, 2005) was presented by MentorGraphics and MathWorks as a solution for the cosimulation between Matlab/Simulink and the ModelSim simulator. It uses the standard communication layer (socket connection) for the data transfer. It integrates Matlab/Simulink into the hardware design flow for the development of Field Programmable Gate Array (FPGA) and Application-Specific Integrated Circuit (ASIC).

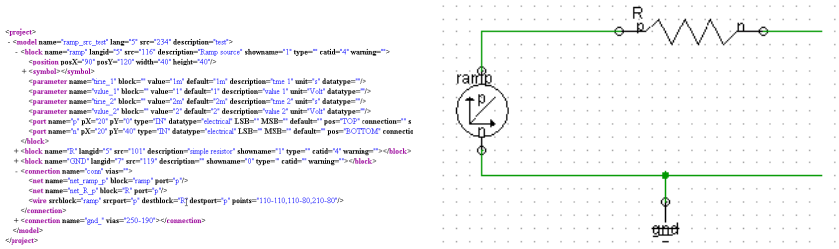
DCB (Distributed Cosimulation Backbone; de Mello and Wagner, 2002) is based on the HLA (High Level Architecture) method for the generation of distributed cosimulation interfaces. DCB serves as common interface for different simulator types. Each simulator can be connected via ambasadors to the DCB backbone. Ambasador controls the data exchange between DCB backbone and connected simulator. DCB supports both synchronous and asynchronous simulation. Therefore, rollbacks are possible. The DCB has been defined in the scope of the SIMOO project (Copstein et al., 1997).

Ptolemy II (Brooks et al., 2004) is a platform for modeling, simulation, and design of concurrent, real-time, embedded systems. Ptolemy II supports several computation models (e. g. Time-Discrete, Discrete-Event, Time-Continuous, etc.), which are called domains. Many pre-defined execution units, called actors are available as well as a graphical editor for the system modeling. Ptolemy II provides interfaces to most of today's popular simulators as well.

2.1 Summary

The two relevant approaches for implementing the cosimulation environment in this work are the Backplane based Simulation and the Direct Coupling. Almost every cosimulation framework recently proposed uses the simulation backplane. For example, the centralized scheme facilitates user interaction and the open API benefits in seamless coupling of different types of simulators. However, the backplane also has considerable drawbacks, such as the performance bottleneck caused by the centralized communication.

As a consequence of the backplane's drawbacks, this work bases on direct coupling of simulators without a central control unit. The proposed communication scheme is more flexible than a cosimulation backplane. However, the



(a) Using XML *(b)* Graphical representation

Figure 4.1 System description

cosimulation is supported by an automatic code generation, which examines the whole system and not only interfacing between two models.

3. Design Methodology

Based on top-down system design using IP (Intellectual Property) models the proposed methodology subdivides the system design into three different levels:

1 System design level

2 Language level

3 Simulator level

3.1 System design level

At the system design level a heterogeneous system is described using a language independent description semantic that enables integration of subsystems designed in different HDLs. At this level the functional descriptions of models are not necessary since the system has been designed using provided IP models. Concerning that, a system description becomes language independent. It opens a possibility to use the same description semantic for models written in different hardware description languages. For the system description, system hierarchy, and schematic representation XML (eXtensible Markup Language) has been used. The essential benefits of XML are language independency, legibility, compactness and support of hierarchical structures. An example of a system description using XML is depicted on Figure 4.1a and its graphical representation in a schematic editor on Figure 4.1b.

The simple model consists of three blocks (`ramp`, `R` and `GND`), which are connected with two connections (`conn` and `gnd_`). As depicted, at the system design level (in this case a top-level description) no functional description of

models has been used. Only the model references and model parameters have been used.

3.2 Language level

At the language level the system has been enhanced to meet requirements of target simulation environment. This includes splitting of the designed system into subsystems, which are later on grouped by its HDL or by required simulators (e. g. for parallel simulation), whereby the special cosimulation blocks are inserted between foreign subsystems. At this level the model sources including cosimulation interfaces have been generated. The performed modification does not have an influence on the behavior of the designed system.

3.3 Simulator level

At the simulator level the modified system description serves as an outline for code generation and setup of the cosimulation platform. Involved simulators communicate via integrated cosimulation interfaces. The data flow and simulation of the whole system is controlled by a synchronization mechanism.

An application framework has been developed to support system design based on the proposed design methodology. Language and simulator levels described above are design steps in a heterogeneous system design that are performed in this framework fully automatically.

4. Design of a cosimulation interface

The first and rather more challenging design effort of this work concentrates on a proper coupling mechanism that enables the incorporation of several different simulators into a heterogeneous cosimulation environment. The finally developed communication scheme allows flexible, reasonably efficient and time accurate heterogeneous cosimulation. The architecture of the implementation of the coupling mechanism should be designed as an easy to use, customizable and extensible framework. The developed concept is inspired by many aspects of the previously introduced cosimulation approaches. By means of evaluations of relevant simulation principles and resulting design decisions, this section presents the overall design of the so called “Cosimulation Interface”, which is the central component used to interconnect different simulators.

4.1 Interfacing between simulators

In general, there are two possibilities to interface different simulators. The first approach is based on file I/O primitives, which is sometimes used for less complex hardware/software cosimulation applications. The more common and flexible communication approach uses open interfaces provided by the partic-

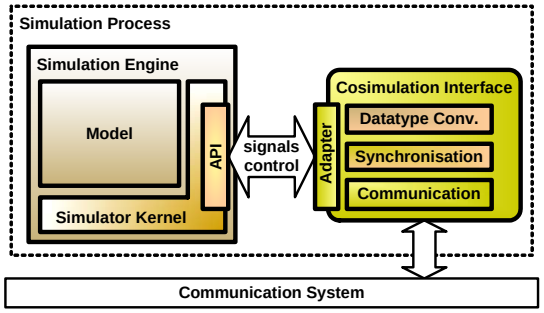


Figure 4.2 Simulator interfacing principle

ipating simulators. Communication by means of file I/O operations is rather simple and provides simulator independency, but it is inflexible and does not meet all intended requirements. Hence the coupling mechanism proposed by this work utilizes open interfaces of simulators. Prominent simulator interface examples are FLI (Model Technology, 2001) and PLI (Mittra, 1999). Figure 4.2 depicts the key principle of the simulator interfacing. The cosimulation interface component (`CsInterface`), hooks on the provided open C/C++ API. Due to no standardized simulator API has been developed yet, an adaptor is needed to adjust the basic cosimulation interface to the simulator specific API. All cosimulation interface components are interconnected by means of a communication system.

Communication system. Since the communication mechanism has a deep impact on simulation performance and accuracy, the communication system must be planned carefully and several aspects have to be considered. The following two requirements influence the design of the communication mechanism:

- Discrete event simulation uses a low time abstraction, therefore the communication should be fast and efficient (i. e. low overhead and short transmission times).
- Distributed simulation possibly on different platforms demands a network-compatible communication mechanism. The used communication system has to be portable to different platforms, e. g. Linux and Microsoft Windows.

There are two main paradigms for interconnecting simulators. One is based on shared resources (shared memory) and the other one lets simulators communicate through channels. The first one is highly efficient but it is not applicable for distributed simulation. The latter paradigm offers the choice between several

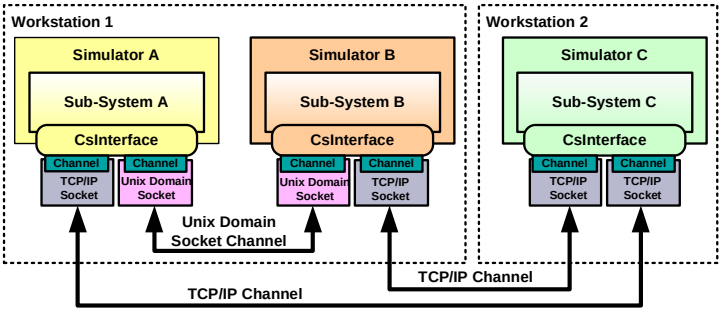


Figure 4.3 Proposed coupling mechanism

communication techniques, such as TCP/IP or UDP socket connections, RPC or higher abstract IPC mechanisms. The proposed communication mechanism of this work utilizes the idea of abstract communication channels, i. e. simulators communicate with each other through abstract channels. An abstract channel provides a uniform interface and encapsulates the underlying communication mechanism. Hence different channel implementations can be easily exchanged. This provides the opportunity to select the most appropriate communication mechanism for a given configuration. This work prefers low abstract methods to high abstract and convenient communication frameworks, such as CORBA.

Coupling mechanism. Summarizing, the proposed cosimulation environment uses a decentralized coupling mechanism. Communication, synchronization, data conversion and control functionality is distributed across all simulators and implemented by means of cosimulation interfaces. Cosimulation interfaces communicate through abstract, peer-to-peer channels. Therefore, each cosimulation interface establishes one connection to every other simulator. Figure 4.3 depicts the principle of the proposed coupling mechanism.

Abstract channels provide basic communication functionality by means of a simple uniform interface. Basically, the package oriented channel interface provides two operations: `readPackage` and `writePackage`. These operations must be of blocking nature. The example shows the needed peer-to-peer connections between three simulators. Simulator A and B are placed on the same workstation, hence they are connected via a shared memory channel. Assuming that workstation 1 and 2 are both sited in a network (LAN/WAN), the peer-to-peer connections between these workstations are built on a TCP/IP channel.

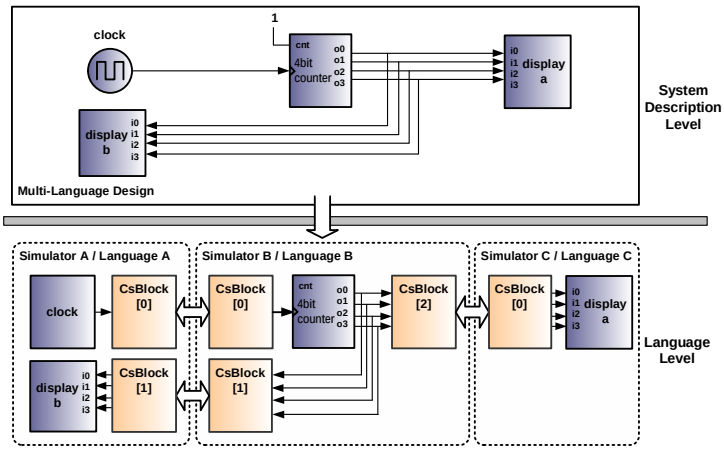


Figure 4.4 Simple example of the CsBlock concept

4.2 Communication

The basic coupling and communication mechanism builds the foundation needed by the three main tasks of the cosimulation interface component: Data communication, data type conversion and synchronization.

The `CsBlocks` modules are used to exchange signals between different simulation engines. It is obvious that data type conversion also takes place in the `CsBlock` module. One `CsBlock` is connected to local blocks with input and output ports and propagates signal changes to the corresponding `CsBlock` on a remote simulator. Clearly, cosimulation blocks always appear in pairs. Each `CsBlock` corresponds to a counterpart sited on a remote simulator. The simple example, shown in Figure 4.4, illustrates this concept of using cosimulation blocks to exchange signals between modules on separated simulators. At the system design level only the structure of the model is represented.

4.3 Datatype conversion

Considering data conversion more precisely, one have to distinguish between language level and simulator level conversion. At the language level, the cosimulation tool has to find equivalent data types for different languages. This usually happens at compile time (code generation phase). At the simulator level, the signal values have to be converted to the simulator's internal representation, which happens at run time. Converting data types is the next step done by the interface modules. This step is required to convert values from one data format to another. The data type conversion is predefined during the code generation phase at the language level. It has been implemented using a simple hash-table,

```
while unprocessed events remain do
        send and receive messages generated in the previous iteration
        LBTS = mini(N_i + LA_i)                    // N_i = time of next event in LP_i
        process events with time stamp <= LBTS     // LA_i = lookahead of LP_i
        barrier synchronisation
end while
```

Figure 4.5 Basic synchronization algorithm

which searches for equivalent data type for a certain language. In order to allow a coupling between several simulators with different data representations, it is a good practice to convert data values to a uniform intermediate format. This work uses the ASCII format for intermediate data representation.

4.4 Synchronization

The cosimulation environment structure is determined by the performance advantage of a decentralized communication scheme over a centralized one, so that the synchronization method has to be decentralized. The conservative protocols are relatively straightforward to implement and can be optimized by adjusting a few model specific parameters. Basically, they have a performance advantage over synchronous protocols. The chosen synchronization method is actually implemented as a decentralized, "synchronous", conservative protocol (Calinescu, 1995). The term synchronous refers to the implementation principle that is based on barrier synchronization. The basic algorithm is shown in Figure 4.5. Each iteration consists of the following steps:

- compute a lower bound on the time stamp (LBTS) of events that might be received later
- events with time stamp <= LBTS are safe to process
- process safe events, exchange messages
- global synchronization (barrier synchronization)

The barrier synchronization method descends from the BSP (Bulk Synchronous Parallel) model, which nowadays is a generic target for conservative discrete event simulations.

4.5 Cosimulation interface

Figure 4.6 depicts the rough architecture of entire simulator interface with its relevant components. `CsBlock` modules represent blocks simulated on a remote simulator. If a signal of a sensitive input port changes, the `CsBlock` module converts the signal value into an intermediate representation and packages it

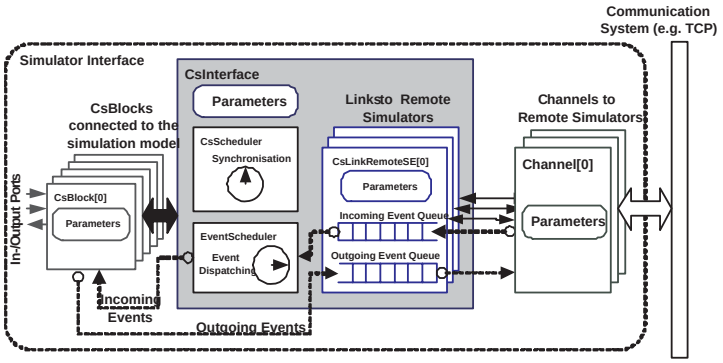


Figure 4.6 Simulator interface structure

into a remote event. Afterwards it forwards the event to the `CsInterface` component. A `CsBlock` also receives event messages and writes the packaged signal values to the corresponding output ports. `CsInterface` is the central component of the simulator interface responsible for event exchanging and synchronization. It contains a `CsLinkRemoteSE` component for each remote simulator, which accumulates all incoming and outgoing events for a certain remote simulator, stores them in a queue, and exchanges them with its counterpart. Each `CsLinkRemoteSE` is directly connected to a `Channel` responsible for data exchange.

The basic synchronization algorithm, implemented in the `CsScheduler` component, conducts the `CsLinkRemoteSE` components to send and receive external events in proper time steps. Furthermore, the `CsScheduler` embeds the cosimulation interface into the simulation process, i. e. it invokes the `CsInterface` procedures at specific simulation times. All incoming events are dispatched according to their timestamps and destination blocks by the `EventScheduler` component. Therefore, it fetches incoming events from the `CsInterface` and propagates them to the corresponding `CsBlock` module. All components are designed to build a general framework, which is easily extensible and adaptable to interconnect distinct simulators. The simulation performance can be optimized by adjusting simulation specific parameters of certain interface components.

5. Automatic code generation

One of the main tasks in this work is the automatic building of a cosimulation platform based on the proposed design methodology. The generation process of a verification platform consists of three main tasks: the cosimulation interface generation, the semiautomatic system partitioning and the hierarchical code

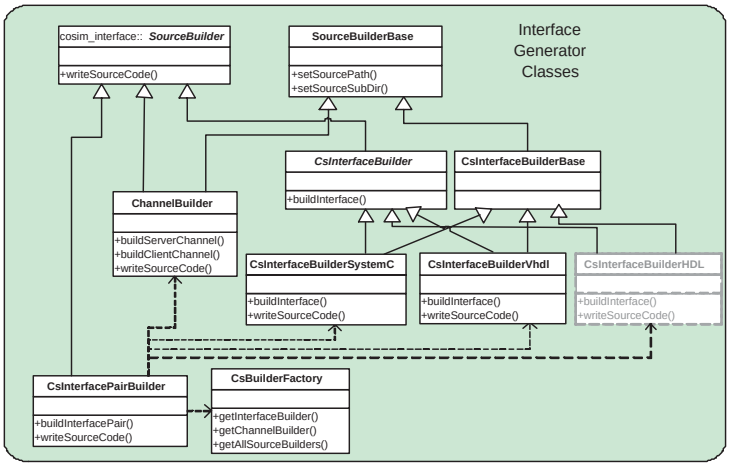


Figure 4.7 Class diagram of the cosimulation interface generator

generation. The automatic cosimulation interface generator creates `CsBlock`s, `CsInterface`s and `Channel`s, which are required to establish a connection between two `CsBlock`s. Figure 4.7 depicts the class diagram of the cosimulation interface generator.

As depicted the central component in the cosimulation interface generator is `CsInterfacePairBuilder`, which is derived from the `SourceBuilder` class. Two provided methods enables simple creation of a `CsBlock` pair: `buildInterfacePair(BlockA, BlockB)` and `writeSourceCode()`. The `buildInterfacePair`-Method creates two `CsBlock`s using simulator specific `CsInterfaceBuilder` classes. All cosimulation sources are generated by invoking the `writeSourceCode()` method. It is obvious that the cosimulation interface generator must have an abstract structure, which enables modular and generic implementation. The cosimulation interface generator serves for creating of `CsBlock` pairs, `CsInterface` control structure (Cs-API) and required channels. It does not create the whole verification platform, due to hold the system simple and clearly arranged. The cosimulation interface generator has been integrated in a more complex structure, which handles the code generation for the verification of the whole designed system. Figure 4.8 depicts the flow of the automatic code generation. The first step after getting the system description and the initialization phase in the code generator is to check the presence of a cosimulation.

In case of a cosimulation, the code generator executes a system partitioning that performs the creation of subsystems, which are later on clustered by simulator type. In this process, the cosimulation interface generator has been used to create required `CsBlock` pairs. At least the code generator invokes the

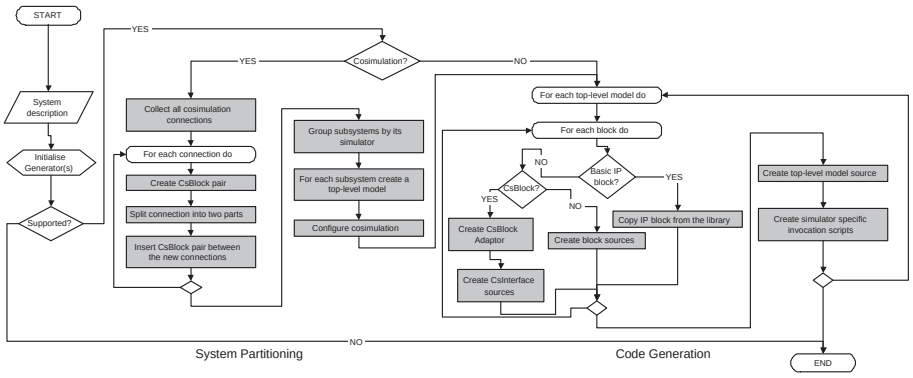


Figure 4.8 Flow of the automatic code generator

source generation for each created top-level description. Recursively all blocks are scanned and their source codes are generated. If a block is a basic IP block the source code will be taken from the provided IP library otherwise they will be generated. The last step in the code generation flow is the creation of the simulator specific invocation scripts. Each involved simulator uses specific call routines, which are defined in a simulator script.

Currently, the presented work provides automatic code generation for VHDL, |VHDL-AMS|, SystemC and SaberMAST models, whereby a cosimulation interface can be created between ADVanceMS, ModelSim and SystemC simulators.

6. Experimental example

An example of the code generation of a heterogeneous system using the methodology described in section 3 is presented in this section. Figure 4.9 depicts a system overview of an automotive power management system (APMS). The system controls the power needs of the automotive electromechanical loads and the charging of the battery and prevents a complete discharging of the battery at any time. This system should work in the large signal area and should contain all important analog nonlinear effects (voltage, current) and the algorithms of the controller. The energy consumption of the car depends first on static loads, controlled by the driver (driving cycle) and at second from the behavioral of the driver assistant systems (dynamic loads dependent on the environment like street condition etc.). It contains analog and digital components as well as software components. The microcontroller developed in SystemC as a state-machine represents a software part of the system. All

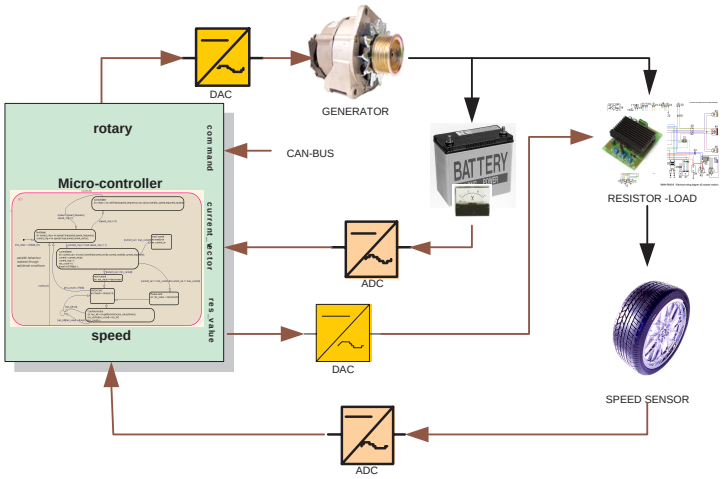
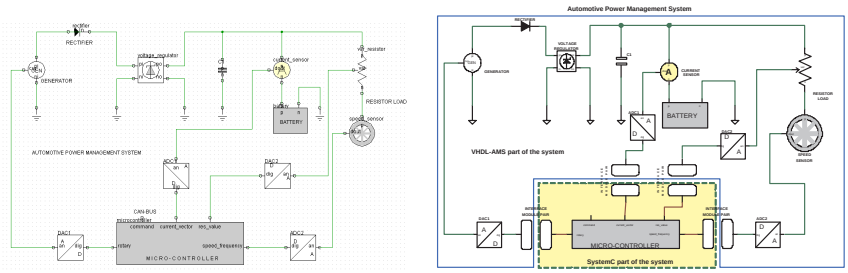


Figure 4.9 A system overview of an automotive power management system



(a) System design level *(b)* Language level

Figure 4.10 System overview

other components such as ADC/DAC, Generator, Speed-Sensor, Battery etc. are coded in VHDL-AMS.

6.1 Design steps

Concurrently developed subsystems are integrated using the schematic editor integrated in our application framework. Figure 4.10.a depicts a schematic overview of the APMS system at the system design level.

Two D/A converters and two A/D converters build a bridge between the microcontroller and the analog part of the system. To overcome the problems of the integration of models written in different languages, four `CsBlock` pairs have been inserted between the microcontroller unit and four converters. With

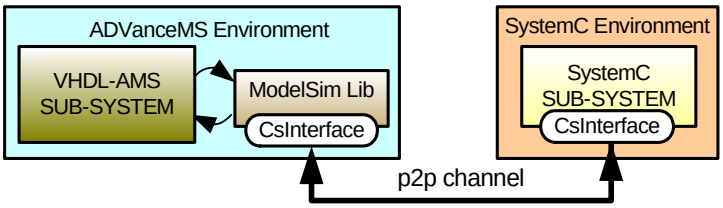


Figure 4.11 Configuration of the Cosimulation Platform

the insertion of `CsBlocks`, the APMS system has been subdivided into two language/simulator groups.

Next step is generating the top-level description of the generated language/simulator groups. In our example we have two language groups: SystemC and VHDL-AMS. The SystemC system consists of the microcontroller and four SystemC-interface modules. Respectively the VHDL-AMS subsystem consists of VHDL-AMS components and four VHDL-AMS interface modules. Figure 4.10b depicts the system overview at the language level with inserted interface modules. The A/D and D/A converters use standard logic vectors for communicating with the microcontroller unit. The VHDL standard logic vectors (`std_logic_vector`) are converted into logic vectors (`sc_lv`) on the SystemC side. Only one peer-to-peer connection channel between SystemC and VHDL-AMS simulation has been created, which is used to transfer the data of all interface modules. We use ADVanceMS simulator from MentorGraphics to simulate the VHDL-AMS subsystem. Since the current version of the ADVanceMS simulator does not provide the required FLI functions for the used synchronization method, we used the ModelSim interface of ADVanceMS to get access to FLI and via FLI to SystemC. The configuration of the cosimulation platform is depicted in Figure 4.11.

This configuration enables cosimulation of analog components with software components of the system. The interface module on the VHDL-AMS side was simulated using ModelSim library. Its entity definition is described in VHDL and the functional behavior of the interface module in C++. The remaining part of the VHDL-AMS side was simulated using ADVanceMS.

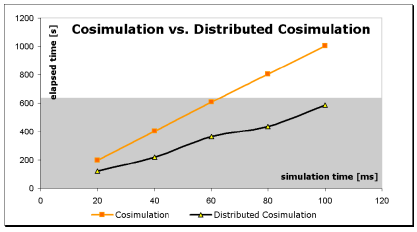
6.2 Results

The proposed cosimulation technique has been evaluated by used heterogeneous examples. Figure 4.12a depicts the comparison between performed cosimulations and its chart has been depicted on Figure 4.12b.

A cosimulation, which is distributed via TCP/IP using two workstations, has been compared with a cosimulation, whereby both simulators run on the same workstation. The best results have been retrieved by a distributed cosimulation

		Cosimulation	Distributed Cosim.
Num.	Simulation Time [ms]	Elapsed Time [s]	Elapsed Time [s]
1	20	197	119
2	40	401	220
3	60	606	364
4	80	804	434
5	100	1003	586
	Dist. Speed-Up	1	1,75
Cosimulation: ADVanceMS and SystemC			

(a) Measured data



(b) Chart

Figure 4.12 Cosimulation results

allowing parallel execution of independent simulation tasks. The table on Figure 4.12a shows the comparison of elapsed times for specific simulation time. At least the average speed-up has been calculated, which is recently dependent on many different factors. The applied synchronization method and `CsInterface` control structures have been improved by the results retrieved from the performed cosimulation, which are compared with the results from a simulation of the same system. To improve the implemented cosimulation approach, the MCU model has been developed in SystemC as well as in VHDL language. The computed data from both simulations are identical, which verifies the applied cosimulation method and used synchronization algorithm (Figure 4.13). The computed signals of a cosimulation are present on the upper side in the figure and the simulation results of a cosimulation are present on the lower side of the waveform viewer. It is obvious that the homogeneous simulation in this case must be faster than performed cosimulation, due to the delays of the communication and the used synchronization method.

However, the elapsed simulation time depends strongly on the system structure that is simulated, involved simulators and many other factors. The aim of this work was not the accelerating of a simulation but more to provide the possibility to design systems language independent and automatically generate required heterogeneous verification platform, which ensures cycle accurate cosimulation. Furthermore, the proposed cosimulation method supports building distributed and parallel simulation environment, to increase the simulation time. The ADVanceMS-SystemC cosimulation example has been advised selected here, to present the actually attractive and recently required combination, which integrates analog, digital as well as software components in one environment using today's most popular simulators. The proposed design methodology supports multi-HDL system design based on models provided by an IP library. Moreover, the verification platform has been generated fully automatically. The implemented automatic code generator reduces the system design time, which is an important issue.

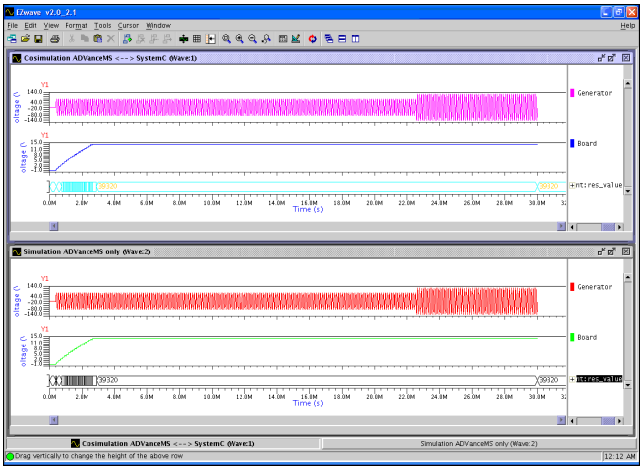


Figure 4.13 Signal comparison: Generator and board voltage in simulation and cosimulation

7. Conclusion

The proposed design methodology using cosimulation techniques has been applied on an application framework, which generates required verification platform fully automatically. Multi-HDL system design, generic interfacing as well as automatic generation of a verification platform including generation of cosimulation interfaces are the key benefits of this work. A generic and modular structured code generator has been implemented. The cosimulation method as well as the code generator it self have been evaluated by an example taken from automotive industry.

References

Brooks, Christopher, Lee, Edward A., Liu, Xiaojun, Neuendorffer, Steve, Zhao, Yang, and Zheng, Haiyang, editors (2004). *Heterogeneous Concurrent Modeling and Design in Java*, volume 1, 2 and 3. EECS, University of California, Berkeley, California, USA. Memoranda UCB/ERL M04/27, M04/16 and M04/17.

Calinescu, Radu (1995). Conservative discrete event simulations on bulk synchronous parallel architectures. Technical Report PRG-TR-16-95, Computing Laboratory, Oxford University.

Copstein, Bernardo, Wagner, Flávio Rech, and Pereira, Carlos Eduardo (1997). SIMOO-An environment for the object-oriented discrete simulation. In *9th European Simulation Symposium (ESS'97)*, Passau, Germany.

de Mello, Braulio Adriano and Wagner, Flávio Rech (2002). A standardized co-simulation backplane. In Robert, M., Rouzeyre, B., Piguet, C., and Flottes,

M.-L., editors, *SoC Design Methodologies*, volume 90 of *IFIP International Federation for Information Processing*, pages 181–192. Kluwer Academic Publishers.

Mentor Graphics (2005). *Seamless*. Mentor Graphics, Wilsonville, Oregon, USA. Online Documentation, `http://www.mentor.com/`.

Mittra, Swapnajit (1999). *Principles of Verilog PLI*. Kluwer Academic Publishers. ISBN: 0-7923-8477-6.

Model Technology (2001). *Modelsim 5.5f: Foreign Language Interface*. Model Technology, Portland, USA.

The MathWorks (2005). *Link for ModelSim*. The MathWorks, Natick, Massachusetts, USA. Online Documentation, `http://www.mathworks.com/`.

TNI-Software (2005). *CosiMate*. TNI-Software, Brest Cedex, France. Online Documentation, `http://www.tni-world.com/`.

Chapter 5

UML/XML-BASED APPROACH TO HIERARCHICAL AMS SYNTHESIS

I. O'Connor, F. Tissafi-Drissi, G. Revy, and F. Gaffiot
Laboratory of Electronics, Optoelectronics and Microsystems
Ecole Centrale de Lyon
36 avenue Guy de Collongue
F-69134 Ecully, France
ian.oconnor@ec-lyon.fr

Abstract This chapter explores the suitability of UML techniques for defining hierarchical relationships in AMS circuit blocks, and XML for storing soft AMS IP design rules and firm AMS IP design data. Both aspects are essential to raising the abstraction level in synthesis of this class of block in SoCs. The various facets of AMS IP are discussed, and explicit mappings to concepts in UML are demonstrated. Then, through a simple example block, these concepts are applied and the successful modification of an existing analogue synthesis tool to incorporate these ideas is proven. The central data format of this tool is XML, and several examples are given showing how this meta-language can be used in both AMS soft-IP creation and firm-IP synthesis.

Keywords: Analogue and mixed-signal; synthesis; IP; UML; XML.

1. Introduction

Cost, volume, power and pervasivity are all difficult constraints to manage in the design of new integrated systems (smart wireless sensor networks, ubiquitous computing, ...). Along with increasingly complex functionality and person-machine interfaces, they are driving the semiconductor industry towards the ultimate integration of complete, physically heterogeneous systems on chip. The coexistence of sensors, analogue/mixed and radio frequency systems (multi-

physics part commonly called AMS[1]) with digital and software IP[2] blocks causes significant design problems.

The difficulty centres on the concept of abstraction levels. To deal with increasing complexity (in terms of number of transistors), SoC[3] design requires higher abstraction levels. But at the same time, valid abstraction is becoming increasingly difficult due to physical phenomena becoming first order or even dominant at nanometric technology nodes. The rise in analogue, mixed-signal, RF[4] and heterogeneous content to address future application requirements compounds this problem. Efficient ways must be found to incorporate non-digital objects into SoC design flows in order to ultimately achieve AMS/digital hardware/software co-synthesis.

The main objective of such an evolution is to reduce the design time in order to meet the time to market constraints. It is widely recognized that for complex systems at advanced technology nodes, mere scaling of existing design technology will not contribute to reducing the "design productivity gap" between the technological capacity of semiconductor manufacturers (measured by the number of available transistors) and the design capacity (measured by the efficient use of the available transistors). Since 1985, production capacity has increased annually by between +41 % and +59 %, while design capacity increases annually by a rate of only +20 % to +25 %. The 2003 ITRS Roadmap clearly states that "cost [of design] is the greatest threat to continuation of the semiconductor roadmap". Only the introduction of new design technology (such as, historically, block reuse or IC implementation tools: each new technology has allowed design capacity to "jump" and to catch up with production capacity) can enable the semiconductor industry to control design cost. Without design technology advances, design cost becomes prohibitive and leads to weak integration of high added value devices (such as RF circuits). One of the next advances required by the EDA industry is a radical evolution in design tools and methods to allow designers to manage the integration of heterogeneous AMS content.

2. Definition of AMS IP Element Requirements for Synthesis Tools

Most analogue and RF circuits are still designed manually today, resulting in long design cycles and increasingly apparent bottlenecks in the overall design process (Gielen and Dehaene, 2005). This explains the growing awareness in industry that the advent of AMS synthesis and optimisation tools is a necessary

[1] Analogue and Mixed-Signal
[2] Intellectual Property
[3] System on Chip
[4] Radio-frequency

step to increase design productivity by assisting or even automating the AMS design process. The fundamental goal of AMS synthesis is to quickly generate a first-time-correct sized circuit schematic from a set of circuit specifications. This is critical since the AMS design problem is typically under-constrained with many degrees of freedom and with many interdependent (and often-conflicting) performance requirements to be taken into account.

Synthesisable (soft) AMS IP is a recent concept (Hamour et al., 2003) extending the concept of digital and software IP to the analogue domain. It is difficult to achieve because the IP hardening process (moving from a technology-independent, structure-independent specification to a qualified layout of an AMS block) relies to a large extent on the quality of the tools being used to do this. It is our belief that a clear definition of AMS IP is an inevitable requirement to provide a route to system-level synthesis incorporating AMS components.

Table 5.1 summarizes the main facets necessary to AMS IP. For the sake of clarity, a reference to VHDL-AMS concepts is shown wherever possible.

Figure 5.1 shows how these various facets of AMS IP should be brought together in an iterative single-level synthesis loop. Firstly, the performance criteria are used as specifications to quantify how the IP block should carry out the defined function. Performance criteria for an amplifier, for example, will include gain, bandwidth, PSRR, offset, etc. They can be considered to be the equivalent of generics in VHDL-AMS. They have two distinct roles, related to the state of the IP block in the design process:

1 as block parameters when the IP block is a component of a larger block, higher up in the hierarchy, in the process of being designed;

2 as specifications when the IP block is the block in the process of being designed (such as here). This role cannot be expressed with VHDL-AMS generics, although language extensions (Doboli and Vemuri, 2003; Hervé and Fakhfakh, 2004) have been proposed.

It will be shown in Section 3 that this dual role requires the definition of a new data type.

The comparison between specified and real performance criteria values act as inputs to the synthesis method, which describes the route to determine design variable values. It is possible to achieve this in two main ways:

1 through a direct procedure definition, if the design problem has sufficient constraints to enable the definition of an explicit solution;

2 through an iterative optimisation algorithm. If the optimisation process cannot, as is usually the case, be described directly in the language used to describe the IP block then a communication model must be set up between the optimiser and the evaluation method. A direct communication model

Table 5.1 AMS IP block facets

Property	*Short description*	*VHDL-AMS equivalent*
Function definition	Class of functions to which the IP block belongs	**entity**, behavioural **architecture**
Performance criteria	Quantities necessary to specify and to evaluate the IP block	**generic**
Terminals	Input/output links to which other IP blocks can connect	**terminal**
Structure	Internal component-based structure of the IP block	structural, **architecture**
Design variables	List of independent design variables to be used by a design method or optimisation algorithm	subset of **generic map**
Physical parameters	List of physical parameters associated with the internal components	**generic map**
Evaluation method	Code defining how to evaluate the IP block, i. e. transform physical parameter values to performance criteria values. Can be equation- or simulation-based (the latter requires a parameter extraction method)	(partly) **process** or **procedure**
Parameter extraction method	Code defining how to extract performance criteria values from simulation results (simulation-based evaluation methods only)	
Synthesis method	Code defining how to synthesize the IP block, i. e. transform performance criteria requirements to design variable values. Can be procedure- or optimisation based	
Constraint distribution method	Code defining how to transform IP block parameters to specifications at a lower hierarchical level	

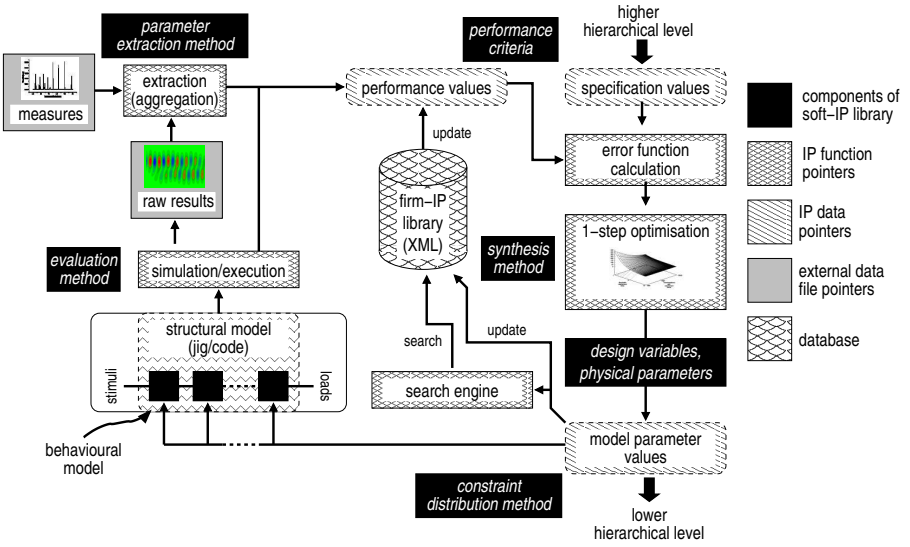


Figure 5.1 Single-level AMS synthesis loop showing context of AMS IP facet use

gives complete control to the optimisation process, while an inverse communication model uses an external process to control data flow and synchronization between optimisation and evaluation. The latter model is less efficient but makes it easier to retain tight control over the synthesis process.

The synthesis method generates combinations of design variables as exploratory points in the design space. The number of design variables defines the number of dimensions of the design space. The design variables must be independent of each other and as such represent a subset of IP block parameters (i. e. performance criteria, described above) in a structure definition. For example, a differential amplifier design variable subset could be reduced to a single gate length, bias voltage, and three transistor widths for the current source, matched amplifier transistors and matched current mirror transistors. Physical variables are directly related to design variables and serve to parameterise all components in the structure definition during the IP block evaluation process. These are represented by the generic map definitions in structural architecture component instantiations in VHDL-AMS. In the above example, the design variable subset would be expanded to explicitly define all component parameters.

The evaluation method describes the route from physical variable values to the performance criteria values and completes the iterative single-level optimisation loop. Evaluation can be achieved in two main ways:

1 through direct code evaluation, such as for active surface area calculations;

2 through simulation (including behavioural simulation) for accurate performance evaluation (gain, bandwidth, distortion, ...). If the IP block is not described in a modelling language that can be understood by a simulator, then this requires a gateway to a specific simulator and to a jig corresponding to the IP block itself. For the simulator, this requires definition of how the simulation process will be controlled (part of the aforementioned communication model). For the jig, this requires transmission of physical variables as parameters, and extraction of performance criteria from the simulator-specific results file. The latter describes the role of the parameter extraction method, which is necessary to define how the design process moves up the hierarchical levels during bottom-up verification phases.

Once the single-level loop has converged, the constraint distribution method defines how the design process moves down the hierarchical levels during top-down design phases. At the end of the synthesis process at a given hierarchical level, an IP block will be defined by a set of physical variable values, some of which are parameters of an IP sub-block. To continue the design process, the IP sub-block will become an IP block to be designed and it is necessary to transform the block parameters into specifications. This requires a definition of how each specification will contribute to an error function for the synthesis method and includes information additional to the parameter value (weighting values, specification type: constraint, cost, condition, ...).

3. UML in AMS Design

3.1 Reasons for Using UML in Analogue Synthesis

UML[5] is a graphical language enabling the expression of system requirements, architecture and design, and is mainly used in industry for software and high-level system modelling. UML 2.0 is due to be adopted as a standard by OMG[6] in 2005. The use of UML for high-level SoC design in general appears possible and is starting to generate interest in several research groups (Riccobene et al., 2005). A recent proposal (Carr et al., 2004) demonstrated

[5]Unified Modeling Language
[6]Object Management Group

the feasibility of describing AMS blocks in UML and then translating them to VHDL-AMS, building on other approaches to use a generic description to target various design languages (Chaudhary et al., 2004). This constitutes a first step towards raising abstraction levels of evaluatable AMS blocks. Considerable effort is also being put into the development of "AMS-aware" object-oriented design languages such as SystemC-AMS (Vachoux et al., 2003) and SysML (Vanderperren and Dehaene, 2005). However, further work must be carried out to enable the satisfactory partitioning of system-level constraints among the digital, software and AMS components. At the system level, the objective in SoC design is to map top-level performance specifications among the different blocks in the system architecture in an optimal top-down approach. This is traditionally done by hand in an ad hoc manner. System-level synthesis tools are lacking in this respect and must find ways of accelerating the process by making reasoned architectural choices about the structure to be designed, and by accurately predicting analogue/RF architectural specification values for block-level synthesis.

Therefore, to be compatible with SoC design flows, top-down synthesis functionality needs to be added to AMS blocks. Our objective in this work is to demonstrate that this is possible. Since UML is a strong standard on which many languages are based (SysML is directly derived from UML, and SystemC as an object-oriented language can be represented in UML also), it should be possible to map the work to these derived or related languages.

3.2 Mapping AMS IP Requirements to UML Concepts

In order to develop a UML-based approach to hierarchical AMS synthesis, it is necessary to map the AMS IP element requirements given in the previous section to UML concepts.

UML has many types of diagrams, and many concepts that can be expressed in each—many more, in fact, than are actually needed for the specific AMS IP problem. Concerning the types of diagram, two broad categories are available:

1 structural diagram, to express the static relationship between the building blocks of the system. We used a class diagram to describe the properties of the AMS IP blocks and the intrinsic relations between them. The tenets of this approach and how to generate UML-based synthesisable AMS IP will be described in this section, with an example in Section 5.

2 behavioural diagram, showing the evolution of the system over time through response to requests, or through interaction between the system components. We used an activity diagram to describe the AMS synthesis process. This will be described in further detail in Section 3,

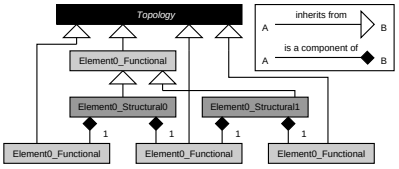


Figure 5.2 UML class diagram showing representation of hierarchical dependencies between AMS IP blocks

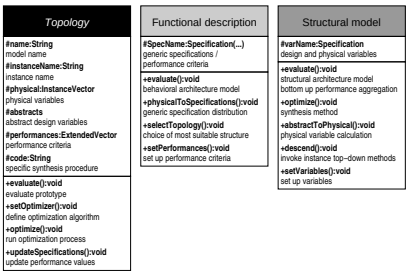


Figure 5.3 UML class definitions for hierarchically dependent AMS IP blocks

and extensions to an existing AMS synthesis tool to incorporate these concepts will be shown in Section 4.

Class Relationships. Firstly, it is necessary to establish a clear separation of a single function definition (entity and functional behavioural model for top-down flows) from n related structural models (for single-level optimisation and bottom-up verification). Each structural model will contain lower-level components, which should be described by another function definition. It is also necessary to establish functionality and requirements common to all structural models whatever their function. By representing all this in a single diagram (Figure 5.2), we are in fact modelling a library of system components; not the actual system to be designed itself. This can be done using an object diagram—however, in this work we will focus on the broader class diagram.

A class diagram constitutes a static representation of the system. It allows the definition of classes among several fundamental types, the class attributes and operations, and the time-invariant relationships between the various classes. From the above analysis, we require (cf. Figure 5.3):

1 a single, non-instantiable (abstract) class representing common functionality and requirements, in a separate publicly accessible package. We called this class `Topology`.

2 a single class representing the function definition, which inherits from `Topology`. An alternative solution would be to separate “Evaluatable” functionality and “Synthesisable” functionality through the use of interfaces. This is certainly a debatable point, but our view is that it would tend to overcomplicate the description process. Another point is that one can also be tempted to separate the entity aspect from the behavioural

Table 5.2 Mapping of AMS IP requirements to class structure

Property	*Class*	*Attribute Type*	*Method*	*Access*
Function definition			constructor	public
entity name	Functional	String		private
behavioural	Functional		evaluate()	public
architecture				
Performance criteria	Functional	Specification		protected
			setPerformances()	public
Terminals	Functional	DomainNode		protected
Structure				
structural	Structural	String		private
architecture name				
Design variables	Structural	Specification		protected
			setVariables()	public
Physical parameters	Structural	Specification		protected
Evaluation method	Structural		evaluate()	public
Parameter extraction				
method				
Synthesis method	Structural		optimize()	public
			abstractToPhysical()	public
Constraint	Structural		descend()	public
distribution method	Functional		physicalToSpecifications()	public

model aspect, which would then allow the entity class to become abstract. Again, this also appears to be somewhat overcomplicated to carry out.

3 *n* classes representing the structural models, which all inherit from the function definition class. Each structural variant is composed of a number of components at a lower hierarchical level, represented by a single function definition class for each component with different functionality. As the structural variant cannot exist if the component classes do not exist, this composition relationship is strengthened to an aggregation relationship.

AMS IP Requirement Handling Through Definition of Class Attributes and Methods. Having established how to separate particular functionality between common, functional and structural parts of an AMS hierarchical model, it is now necessary to define how to include each facet of the AMS IP requirements set out in Section 2. This is summarized in Table 5.2.

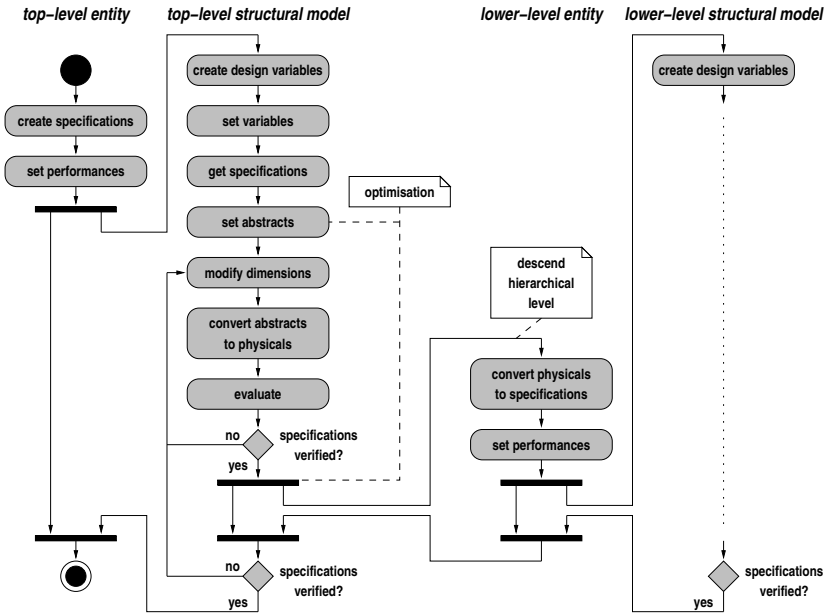


Figure 5.4 Activity diagram representing hierarchical AMS IP synthesis process for TIA block

Thus the performance criteria and variables are defined with type `Specification`. This is a specific data type, which plays an important role in the definition of AMS IP. It requires a name `String`, default value and units `String` as minimum information. When used as a performance requirement in a base class, it can also take on the usual specification definitions (<, >, =, `minimize`, `maximize`).

3.3 Modelling the Analogue Synthesis Process With Activity Diagrams

In UML, a behavioural diagram complements structural diagrams by showing how objects or classes interact with each other and evolve over time to achieve the desired functionality. Among these, the activity diagram is useful for showing the flow of behaviour (objects, data, control) across multiple classes as a sort of sophisticated dataflow diagram.

Figure 5.4 shows an example flow for two hierarchical levels. For a given hierarchical level, the process begins with specification definition—either from an external point (e. g. user) or from the design process at the hierarchical level immediately above. It then calls a number of internal methods (set performances, variables and abstracts), all of which must have been explicitly defined by the IP creator prior to synthesis. The optimisation process can then begin

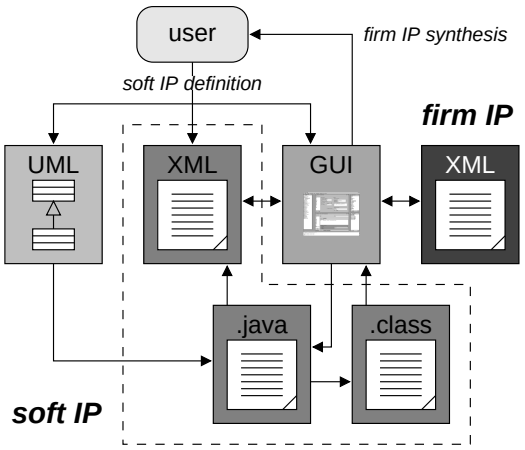


Figure 5.5 UML/XML use flow in runeII

with dimension (or variable) value modification according to the optimisation algorithm used, determination of the physical parameter values from the new design variable set, evaluation and comparison of achieved performance values with requirements. If the requirements are met, then the process can go down through the hierarchy to determine the parameters of lower hierarchical blocks, or if there are no lower levels then the verification process can begin. It should be noted that the sequence of events for a functional/structural model pair maps to the iterative loop shown in Figure 5.1.

4. Description of Extensions to Existing Analogue Synthesis Tool (runeII)

We have incorporated these concepts into an existing in-house AMS synthesis framework, runeII. This builds on a previously published version of the tool (Tissafi-Drissi et al., 2004). The main motivation behind this evolution was to improve the underlying AMS IP representation mechanisms, and to enhance the input capability of the tool. A schematic showing the various inputs and data files is given in Figure 5.5.

From the user's point of view, there are two main phases to AMS synthesis: AMS soft-IP definition, which can be done via UML, XML[7] or through a specific GUI[8]; and AMS firm-IP synthesis, which can be run from the GUI or from scenarios. XML is a text markup language for interchange of structured data specified by W3C. The Unicode Standard is the reference character set

[7]eXtensible Markup Language
[8]Graphical User Interface

for XML content, and because of this portable format and ease of use, it is fast becoming a de facto standard for text-based IP exchange.

4.1 AMS soft-IP definition

The aim of the first point is to create executable and synthesisable models (here, in the form of java `.class` files). We consider the central, portable format to be XML, which can be generated directly from the GUI and from `.java` source files.

A screenshot of the GUI enabling creation of such files from graphical format is shown in Figure 5.6. The various zones in the figure have been numbered and the corresponding explanation follows:

1 menu bar

2 database tree explorer (top nodes = entity/functional models; nested nodes = structural models). The user is able to process several actions: new, open, export, import, delete, rename, cut, copy, paste. These actions operate on the currently selected structure or function and are also available in the main frame toolbar

3 entity/functional model editor. Here, the IP creation process starts in earnest in defining the various performance criteria

4 structural model editor. This window allows the creation of design variables, physical parameters, evaluation procedures, ...

5 preset design plans (sequences of optimisation algorithms)

6 technology data files

7 message window. This is to output log information, e. g. detailed information about the operation which is being made, error descriptions etc.

4.2 AMS firm-IP synthesis

The second point exploits the created executable, synthesisable models in an iterative process aiming to determine the numerical parameter values necessary to optimally realize numerical performance requirements. Again, the database format was chosen as XML for reasons of portability. Here though, apart from capture of the numerical information itself in an XML document, a definition of the legal building blocks necessary to interpretation of the XML document structure is required. This is the purpose of a DTD[9], which can be declared

[9] Document Type Definitions

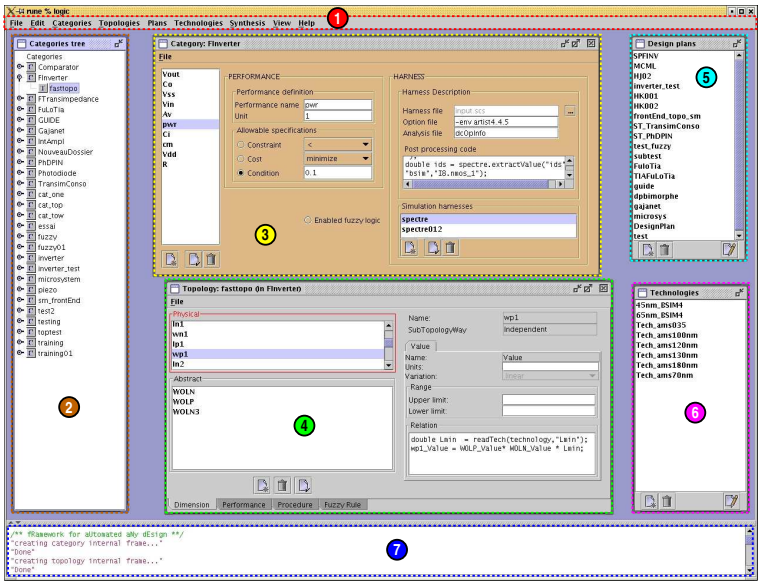


Figure 5.6 Screenshot of runeII GUI

Listing 5.1 Entity/Functional and Structural model DTD template

```
<!{ELEMENT} FunctionName (Structure1, Structure2*)>
  <!{ATTLIST} FunctionName
    PerformanceName1 CDATA ""
    ...
  >

<!{ELEMENT} StructureName (Component1, Component2, ...)>
  <!{ATTLIST} StructureName
    VariableName1 CDATA ""
    ...
  >
```

inline in the XML document, or as an external reference. We have chosen the latter approach, shown in Listing 5.1.

As mentioned previously, the synthesis process can be run either from the GUI or through the creation of scenarios. Scenarios are another type of class which instantiate and setup all the components necessary for synthesis in their constructor, much as in a traditional netlist, and then define the optimisation process in the main method. The scenario actually represents the final executable and, while more difficult to generate, avoids any constrictions imposed by the GUI.

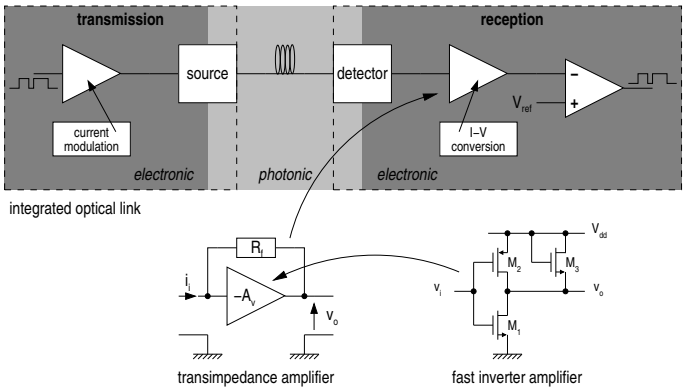


Figure 5.7 Context of TIA and internal amplifier in an integrated optical link

5. Example

We now introduce an example circuit to illustrate the concepts previously described. We focus on the representation of a resistive feedback TIA[10] (consisting of a non-differential inverting amplifier with feedback resistance) as part of a CMOS photoreceiver front-end (Figure 5.7; Tissafi-Drissi et al., 2003).

It is important to understand how a TIA is specified in the link. The main performance criteria for the TIA itself are the in-band transimpedance gain Z_{g0}, angular resonant frequency ω_0, quality factor Q, quiescent power dissipation and occupied surface area. The first three quantities express the capacity of the TIA to convert an input photocurrent variation to an output voltage variation according to a linear second-order transfer function. The latter two criteria (power and area) can only be accurately determined by synthesising down to transistor level, constituting the main difficulty in AMS IP formulation. To reach this level, the specific TIA structure (resistive feedback) is considered. The physical parameters consist of the feedback resistance value R_f, and the internal amplifier performance criteria (voltage gain A_v, output resistance R_o). Concerning the design variables, it has been shown in O'Connor et al., 2003 that only one is necessary: M_f, the ratio between R_f and R_o.

5.1 Class diagram example

This information suffices to start building a class diagram for the TIA structure (Figure 5.8).

For clarity, only the `TransimpedanceAmplifier` functional model class, defining the TIA performance criteria, and its derived `RFeedback` structural

[10]Transimpedance Amplifier

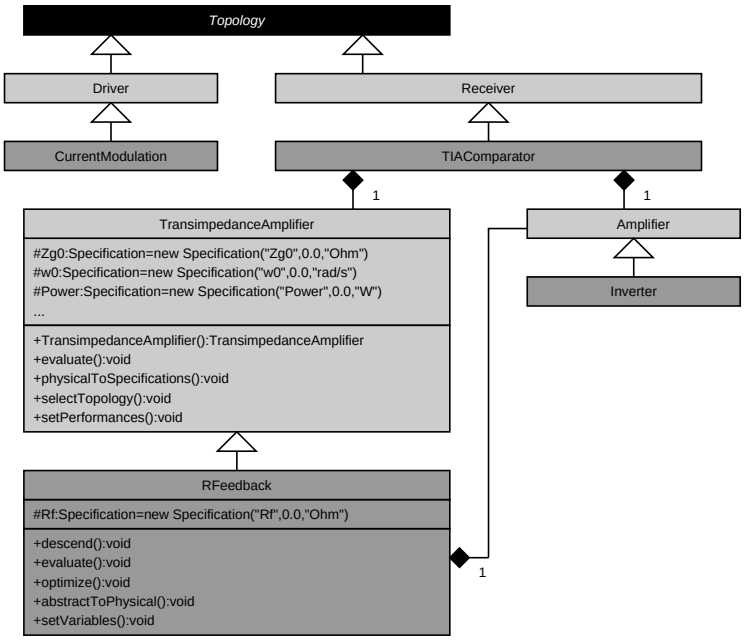


Figure 5.8 UML class diagram showing (in expanded form) TIA function and resistive feedback structure

model class, defining the physical and design variables, have been expanded. It should be noted that the physical variables related to the internal amplifier are defined in an aggregation relationship between the `RFeedback` class and the `Amplifier` functional model class. The other classes show their context in a class diagram representing an optical receiver circuit hierarchy. Some of the class methods shown are related to the specific implementation. In particular, some constructors require XML document inputs. These represent firm-IP data allowing the block to retrieve previously stored information, in the format described in Section 5.

5.2 Soft-IP XML file example

An example of an XML file representing (i) an entity/functional model is given in Listing 5.2, and (ii) a structural model is given in Listing 5.3. Both are based on specific DTD rules corresponding to the concepts set out in section 4 and illustrate the various facets of AMS IP defined in section 2.

5.3 Optimisation scenario example

As a simple example, Listing 5.4 shows the scenario (java source) to optimise the `RFeedback` object.

Listing 5.2 Entity/Functional model description output in XML

```
<category name="TransimpedanceAmplifier">
- <template name="Zg0" units="Ohm">
    <definitions constraint=">" cost="maximize" ↵
       condition="0.1"/>
  - <harness simulator="spectre" file="input.scs" ↵
     options="-env␣artist4.4.5" analysis="ac" selected="true">
    - <code>
       Zg0 = spectre.gainMax("ID:p","vo");
     </code>
    </harness>
  + <harness simulator="eldo" file="input.cir" options="" ↵
     analysis="ac" selected="false">
    </harness>
  </template>
+ <template name="QuiescentPower" units="W"></template>
...
```

Listing 5.3 Structural model description output in XML

```
- <topology name="TransimpedanceAmplifier-RFeedback" ↵
    instanceName="" categoryName="TransimpedanceAmplifier">
  + <physical type="dependent" name="Amplifier" ↵
     instanceName="A1" categoryName="Amplifier">
    </physical>
  + <physical type="independent" name="Resistance" ↵
     instanceName="Rf"></physical>
  - <abstract type="independent" name="Double" ↵
     instanceName="Mf">
    - <dimension name="Value" units="" lower="0.0010" ↵
       upper="100.0" variation="linear">
     </dimension>
    </abstract>
...
  - <performance name="Zg0" units="Ohm" heuristic="false" ↵
     enabled="false">
    - <equation>
       Zg0  =  ((Rf_Value * A1.Av())- A1.Ro())/(1 + A1.Av());
     </equation>
    </performance>
  + <performance name="QuiescentPower" units="W" ↵
     heuristic="false" enabled="false"></performance>
...
</topology>
```

Listing 5.4 TransimpedanceAmplifier/RFeedback optimisation scenario description in Java

```
1  package scenarios;
2  import basic.*; ...
3  public class S_RFeedback extends TestTIA {
4    public S_RFeedback() {
5      try {  // load specifications
6             Document TIADoc = ↵
                  ReadXML.loadDocument("/home/work/xmlFiles/TIA_specs.xml",↵
                   true);
7             // create RFeedback object with specifications. ↵
                  Sizing is done in the constructor.
8             // Assign it to tia object (defined in TestTIA base↵
                   class)
9             tia = new RFeedback("Rf",TIADoc);
10         } catch (Exception e) { e.printStackTrace(System.err);↵
             }
11     } // end constructor
12   public static void main(String[] args) {
13     try {  // create scenario object. Design process defined ↵
           and executed in constructor.
14            S_RFeedback scenario = new S_RFeedback();
15            // evaluation of resulting RFeedback object
16            scenario.getTIA().evaluate();
17            // store results in firm IP database
18            Document outputDocTIA = new ↵
                  Document(WriteXML.XMLTopology(scenario.getTIA()));
19             WriteXML.save("/home/work/xmlFiles/outxml/S_TIA_perfs.xml",outputDocTIA)
20         } catch (Exception e) { e.printStackTrace(System.err);↵
             }
21     } // end main
22 } // end S_RFeedback
```

Listing 5.5 Firm IP synthesis results in XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE GenericLink SYSTEM "Link_dtd.dtd">
<GenericLink BER="< 1.5E-18 Bit/s" DataRate="> 1.5e9 Gbit/s" ↵
    Abstract="True" toOptimize="True">
  - <OPPLink>
    + <Driver BiasCurrent="= 2e-3 A" ModulationCurrent="= ↵
        25e-6 A" Abstract="False" toOptimize="True">
      </Driver>
    + <WaveguideStructure Loss="2e-2 U" Length="2e-3 U"/>
    - <Receiver>
      + <Detector extinctionRatio="1.0 U" currentNoise="1.0 ↵
          U" Responsivity="0.8 A/W" />
      - <TIAComparator>
        ...
        <TransimpedanceAmplifier Cd="= 400.0E-13 F" Cl="= ↵
            150E-13 F" Zg0="= 1E3 Ohm" Q="= 0.7017"
        Abstract="True" toOptimize="True">
          <RFeedback Rf ="1390 Ohm">
            <Amplifier Av="10" Ro="500 Ohm" Cm="8.0 E-14 F" ↵
                Co="5.0 E-13 F" Ci="7.0E-13 F"
            QuiescentPower="0.5E-3 W" Abstract="True"/>
          </RFeedback>
        </TransimpedanceAmplifier>
        ...
        + <Comparator BW="= 3 GHz" QuiescentPower="= 164E-6" ↵
            Latence="= " refVoltage="= " Vl="= 0.1 V"
           Vh="= 0.8 V" Lmin="= 0.35E-6 m" Abstract="True" ↵
              toOptimize="True"/>
      </TIAComparator>
    </Receiver>
  </OPPLink>
</GenericLink>
```

5.4 Firm-IP XML output file example

The partial results, in the output XML format, of this synthesis process achieved for a 0.35 μm CMOS technology and with specifications given in the first line of the file, are shown in Listing 5.5.

6. Conclusion

In this paper, we have proved the feasibility of the use of UML for the representation of synthesisable hierarchical AMS IP blocks. A parallel between UML concepts and widely used concepts in AMS behavioural modelling languages (we used the VHDL-AMS example) was established, in particular:

- class diagrams to represent the various ways (structural architectures) of realizing a given function (entity and behavioural architectures)
- inheritance relations to identify the relationship between an entity/behavioural model (base class) and one or more structural architectures (derived classes)
- aggregation relations to identify the sub-components in a structural architecture.

We have successfully used these concepts to build class diagrams for a variety of AMS soft-IP blocks. While the approach is quite straightforward, the resulting diagrams can be quite large and unwieldy. Further work is necessary to determine how to make better use of package diagrams in soft-IP library management.

Several methods have to be written to render these model classes synthesisable (we associate UML with Java for this development task, but there is no technical reason why the same concepts cannot be developed with other OO[11] languages such as C++). We used this in the context of extending an existing AMS synthesis flow and as such have used it for low-level AMS blocks (TIA, amplifiers, filters and duplexers). XML was used in this respect to formulate soft-IP information and to store all generated numerical firm-IP. Future work will include the use of Pareto-sets to optimally reduce the amount of information stored, and data mining techniques to retrieve useful information. Application of this approach to more complex discrete-time and RF blocks is also a goal.

Acknowledgments

This work was partially funded by the European FP6 IST program under PICMOS FP6-2002-IST-1-002131, and by the French Rhône-Alpes region under OSMOSE (PRTP 2003-2005). The authors gratefully acknowledge discussions with Y. Hervé for valuable comments and insights.

References

Carr, C. T., McGinnity, T. M., and McDaid, L. J. (2004). Integration of UML and VHDL-AMS for analogue system modelling. *BCS Formal Aspects of Computing*, 16(80).

Chaudhary, V., Francis, M., Zheng, W., Mantooth, A., and Lemaitre, L. (2004). Automatic generation of compact semiconductor device models using

[11] Object-Oriented

Paragon and ADMS. In *Proceedings of the IEEE International Behavioral Modeling and Simulation Conference 2004*, page 107. IEEE.

Doboli, A. and Vemuri, R. (2003). Behavioral modeling for high-level synthesis of analog and mixed-signal systems from VHDL-AMS. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(11):1504–1520.

Gielen, G. and Dehaene, Wim (2005). Analog and digital circuit design in 65 nm cmos: End of the road? In *Proceedings of Design Automation and Test in Europe (DATE) 2005*, pages 36–41, Munich, Germany.

Hamour, M., Saleh, R., Mirabbasi, S., and Ivanov, A. (2003). Analog IP design flow for SoC applications. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS 2003)*, pages IV–676.

Hervé, Y. and Fakhfakh, A. (2004). Requirements and verification through an extension of VHDL-AMS. In *Proceedings of the Forum on Specification and Design Languages (FDL) 2004*, page 91, Lille, France. ECSI.

O'Connor, I., Mieyeville, F., Tissafi-Drissi, F., Tosik, G., and Gaffiot, F. (2003). Predictive design space exploration of maximum bandwidth CMOS photoreceiver preamplifiers. In *Proceedings of the IEEE International Conference on Electronics, Circuits and Systems (ICECS) 2003*. IEEE.

Riccobene, E., Scandurra, P., Rosti, A., and Bocchio, S. (2005). A SoC design methodology involving a UML 2.0 profile for SystemC. In *Proceedings of the Design Automation and Test in Europe (DATE) 2005*, pages 704–709, Munich, Germany.

Tissafi-Drissi, F., O'Connor, I., and Gaffiot, F. (2004). RUNE: platform for automated design of integrated multi-domain systems. application to high-speed cmos photoreceiver front-ends. In *Proceedings of Design Automation and Test in Europe (DATE) 2004—Designer's Forum*, pages 16–21, Paris, France.

Tissafi-Drissi, F., O'Connor, I., Mieyeville, F., and Gaffiot, F. (2003). Hierarchical synthesis of high-speed CMOS photoreceiver front-ends using a multi-domain behavioral description language. In *Proceedings of the Forum on Specification and Design Languages (FDL) 2003*, page 151, Frankfurt, Germany. ECSI.

Vachoux, A., Grimm, C., and Einwich, K. (2003). SystemC-AMS requirements, design objectives and rationale. In *Proceedings of Design Automation and Test in Europe (DATE) 2003*, pages 388–393, Munich, Germany.

Vanderperren, Y. and Dehaene, W. (2005). UML 2 and SysML: an approach to deal with complexity in SoC/NoC design. In *Proceedings of Design, Automation and Test in Europe (DATE) 2005*, pages 716–717, Munich, Germany.

IV

UML-BASED SYSTEM SPECIFICATION AND DESIGN