System-level modeling and simulation of microelectromechanical systems for multi-physics virtual prototyping in SystemC-AMS

Benoit Vernay

THÈSE DE DOCTORAT DE L'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique

École Doctorale

ED130 Informatique, Télécommunications et Électronique

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse

Modélisation et simulation haut-niveau de micro-systèmes électromécaniques pour le prototypage virtuel multi-physique en SystemC-AMS

Présentée par

Benoit Vernay

Soutenue le 16 Juin 2016

Devant le jury composé de

Prof. Habib Mehrez	Université Pierre & Marie Curie	Président
Prof. Gaëlle Lissorgues	ESIEE Paris	Rapporteur
Dr. Matthieu Moy	Grenoble INP	Rapporteur
Prof. Robert Sobot	ENSEA	Membre du jury
Prof. Skandar Basrour	Université Grenoble Alpes - TIMA	Membre du jury
Prof. François Pêcheux	Université Pierre & Marie Curie	Directeur de thèse
Dr. Marie-Minerve Louërat	Université Pierre & Marie Curie	Co-Directrice de thèse
Dr. Gerold Schröpfer	Coventor	Directeur de thèse
Dr. Arnaud Krust	Coventor	Invité

Benoît Vernay

System-level modeling and simulation of microelectromechanical systems for multi-physics virtual prototyping in SystemC-AMS

PhD Thesis

June 2016

University Pierre & Marie Curie
Computer Science Department
Paris

Abstract

Embedded systems have evolved to more complex assemblies, called Cyber-Physical Systems (CPS), mostly operating through networks and tightly interacting with the environment. As actuators or sensors, micro-electromechanical systems (MEMS) are essential elements in these systems where they are integrated along with control and signal processing units. Designing such solutions requires a multi-domain approach like virtual prototyping. Based on system-level models, this technique allows to anticipate the global behavior in early-design phases and to further refine it in more advanced steps. Integrated circuits were progressively designed with respect to this method, especially through Hardware Description Languages (HDLs) like VHDL or Verilog. By adopting a higher-abstraction degree, SystemC enabled the co-development of hardware/software specific applications. In parallel, the Analog and Mixed-Signal (AMS) extensions proposed in SystemC-AMS partly addressed the increasing amount of analog components and are considered as a promising alternative for the virtual prototyping of heterogeneous systems. To that end, this thesis addresses the system-level modeling and simulation of MEMS devices in SystemC-AMS. First, we evaluate the current capabilities of the standard and supported models of computation in SystemC-AMS. We demonstrate the limitations and the difficulty to elaborate equivalent models of MEMS devices whose geometry and internal coupling require more detailed descriptions. Second, we propose to directly integrate MEMS reduced models in SystemC-AMS. Model-order reduction is a mathematical technique to decrease the number of degrees of freedom and generate compact models from large-scale systems. We thus integrate the reduced models exported from the finite-element analysis tool *MEMS+* and propose an Application Programmable Interface (API) to insert these *ad hoc* models in SystemC-AMS. After reviewing the main API features, we discuss some improvements of both the standard and the presented solution. Finally, we verify our solution through the use case of an accelerometer and compare the results with the state of the art in terms of modeling accuracy and simulation performance. This thesis introduces a framework to integrate MEMS devices with the surrounding electronics in a unified system-level simulation environment.

Keywords Microelectromechanical systems, virtual prototyping, system-level simulation, Hardware Description Language (HDL), Analog and Mixed-Signal (AMS), SystemC, SystemC-AMS

Résumé

L'évolution des systèmes embarqués se traduit aujourd'hui par des ensembles complexes, dits systèmes cyber-physiques, opérant principalement en réseau et interagissant fortement avec leur environnement. Intégrés à des circuits de contrôle et de traitement du signal, les micro-systèmes électromécaniques, ou MEMS, jouent un rôle primordial dans ces ensembles en tant que capteurs ou actionneurs. La conception de tels systèmes requiert des solutions globales et pluri-disciplinaires telles que le prototypage virtuel. Basée sur des modèles haut-niveau, cette technique permet d'anticiper le comportement du système dès les premières phases de conception et de le raffiner lors de phases plus avancées. Ces méthodes ont progressivement été appliquées à la conception de circuits intégrés, notamment avec l'utilisation de langages de description matérielle, tels que VHDL ou Verilog. En adoptant un niveau d'abstraction supérieur, SystemC a largement contribué au développement concourant des parties matérielles et logicielles. Parallèlement, les extensions proposées dans SystemC-AMS répondent au nombre croissant de composants analogiques dans les circuits intégrés et constituent une base prometteuse pour le prototypage virtuel de systèmes hétérogènes. Pour cette raison, cette thèse traite de la modélisation et de la simulation haut-niveau de dispositifs MEMS en SystemC-AMS. Dans un premier temps, nous évaluons les capacités actuelles du standard et des modèles de calcul proposés dans SystemC-AMS. Nous démontrons les limites et la difficulté d'élaborer des modèles équivalents de dispositifs MEMS dont la géométrie et les couplages internes nécessitent des descriptions plus détaillées. Nous proposons donc, dans un deuxième temps, d'intégrer directement dans SystemC-AMS des modèles réduits de dispositifs MEMS. La réduction d'ordre de modèle est une méthode mathématique permettant de créer des représentations compactes de systèmes initialement très larges en termes de degrés de liberté. Ainsi, nous utilisons les modèles générés depuis l'outil d'analyse en éléments finis *MEMS+* et proposons une interface de programmation pour les insérer dans des modèles SystemC-AMS. Après avoir détaillé les principales fonctionnalités de l'interface, nous discutons les améliorations possibles du standard et de la solution présentée. Enfin, nous vérifions notre solution avec l'étude d'un accéléromètre et comparons les résultats avec l'état de l'art en termes de précision des modèles et de performances de simulation. Cette thèse propose ainsi une méthodologie complète pour intégrer des dispositifs MEMS dans un environnement de simulation haut-niveau.

Mot-clés Micro-systèmes, MEMS, prototypage virtuel, simulation haut-niveau, langage de description matérielle (HDL), analogique et signal-mixte (AMS), SystemC, SystemC-AMS.

Acknowledgments

"Je comprends que si je puis par la science saisir les phénomènes et les énumérer, je ne puis pour autant appréhender le monde."
Albert Camus, *Le Mythe de Sisyphe*, 1942.

I am grateful to my advisors, François Pêcheux and Marie-Minerve Louërat, for their guidance throughout my time at the University Pierre and Marie Curie and the autonomy in my work at LIP6. I want to express my most sincere gratitude to Gerold Schröpfer for the trust and encouragement he afforded me. I learned a lot from his insights on the MEMS industry and innovation management. I want to thank Arnaud Krust who significantly contributed to this research project and always gave me precious advice.

I would like to acknowledge the support received from the CIFRE program and the CATRENE organization which co-funded this work, realized in partnership between Coventor and LIP6, and part of the European project CA-701 H-Inception. I thank our European collaborators, especially the project leaders at ST Microelectronics, Serge Scotti and Olivier Guillaume. I want to thank Fabio Cenni for his great support and the numerous technical discussions we had. I am also grateful to Torsten Mähne for sharing his expertise and commitment though a fruitful collaboration at LIP6 during this project.

I was fortunate to be part of Coventor's development team and really enjoyed sharing such a great working environment. I hope this company, lead by Mike Jarowlovski, and the MEMS+ product, now driven by Stephen Breit and Stéphane Rouvillois, will both meet the success they deserve. It was a real pleasure to work with the *MEMS+* team in Paris and I want to thank its members. In alphabetical order, these are Aurélie Cruau, Igor Favorskiy, Gunar Lorenz, Arnaud Parent, Olivier Renaud, Alexandre Sinding and Daniele Timarchi.

I want to thank my workmates at LIP6, especially Liliana Andrade, Cédric Ben Aoun and Vanessa Tran with whom I shared great moments in our daily work and had a wonderful time in Greece. I also thank the former and current PhD students met at LIP6 laboratory: Jean-Baptiste Brejon, Alexandre Brière, Clément Devigne, Wilfried Dron, Cesar Fuguet, Zhi Hao, Thomas Hujsa, Laurent Lambert, Yao Li, Hao Liu, and Quentin Meunier. I wish them all the best in their future career and personal life.

In the end, life is a tremendous journey where love matters most. I would thus remember my parents from whom I have learned the value of work and the sense of integrity. I am incommensurately grateful to my sister, Marilyne, without whom I would not have been so far. I want to thank my family and friends for their unwavering support and the joy to spend time with them. Finally and above all, I want to thank my beloved, Sophie, for her devotion and patience throughout these years, teaching me that "real generosity towards the future lies in giving all to the present".

"L'esthétique, envisagée comme science des structures, mérite de devenir le guide sûr des actions humaines quelles qu'elles soient."
Henri Laborit, *Biologie et structure,* 1968.

Contents

List of Acronyms

3-D Three Dimensional *38, 62, 64, 72, 82, 135, 137*

AC Alternating Current *15–17, 51, 75, 76, 78, 79, 89, 135*

AMS Analog/Mixed-Signal *2, 3, 6, 8, 12, 22, 24, 28, 31, 32, 34, 37, 39, 41, 55, 57, 59*

API Application Programming Interface . *2, 5, 9, 20, 26, 36, 59, 68, 71–73, 78, 79, 82, 86, 87, 98, 99, 102, 139*

ASIC Application Specific Integrated Circuit *1, 2, 7, 42, 86, 87*

BEM Boundary-Element Method *7, 12–15, 18, 38*

CAD Computer Aided Design *7, 21, 39, 46*

CMOS Complementary Metal-Oxide Semiconductor *1, 2, 50*

DAE Differential Algebraic Equation *5, 22, 26, 35, 36, 42, 43, 45, 47*

DC Direct Current *15–17, 37, 50, 75, 76, 78, 79, 89, 135*

DE Discrete Event *4, 35, 36, 79*

DEVS Discrete EVent System specification *36, 103*

DoF Degrees of Freedom *7, 12, 14, 20, 22, 38, 53, 60, 86, 98, 139*

DTDF Dynamic Timed Data Flow *34, 73, 79, 102*

EDA Electronic Design Automation *1, 26*

ELN Electrical Linear Network *31–33, 35, 36, 50, 55*

ESL Electronic System-Level *5, 8, 9, 12, 24, 26–28, 39, 41*

FE Finite Element *21, 22, 25, 38, 42, 60*

FEA Finite Element Analysis *7–9, 11, 16, 18, 25, 26, 57, 59, 61, 63, 68*

FEM Finite Element Method *2, 7, 9, 11–15, 18, 25, 26, 38*

FMI Functional Mock-up Interface *4, 5*

HDL Hardware Description Language *2, 5, 8, 9, 11, 20–22, 24, 26, 27, 30, 31, 39, 59, 88*

HetSC Heterogeneous specifications in SystemC *36*

HW Hardware *1–9, 11, 19, 22, 26, 27, 31, 37–39, 41, 49, 57, 59, 84, 86, 88, 101–103*

I²C Inter-Integrated Circuit *93*

IC Integrated Circuit *1, 2, 5, 6, 8, 11, 12, 21, 22, 24, 27, 28, 36, 39*

IP Intellectual Property *2, 5, 28, 104*

LSF Linear Signal Flow *31–33, 35, 36, 50, 62*

LTI Linear Time Invariant *55, 84*

MARTE Modeling and Analysis of Real-Time and Embedded systems *3*

MDVP Multi-Domain Virtual Prototyping *3, 7, 36, 41, 57, 97, 103*

MEMS Microelectromechanical System .. *1–3, 6–9, 11–15, 17–22, 24–26, 31, 36–39, 41–43, 46, 48, 49, 52, 57, 59, 60, 71–73, 78, 79, 84, 86–88, 93, 94, 99, 101–103, 135, 140*

MIMO Multiple-Input Multiple-Output *47*

MoC Model of Computation *4–9, 28, 30–38, 41, 42, 46, 49, 50, 55, 57, 62, 73, 78, 79, 98, 101, 103*

MOR Model Order Reduction *7–9, 11, 12, 19–26, 38, 41, 59, 60, 68, 71, 84, 101, 103, 139*

ODE Ordinary Differential Equation *4, 5, 7, 12, 20, 22, 23, 26, 41, 45, 47, 48, 57*

PDE Partial Differential Equation *7, 12, 22, 23, 36, 38, 42, 45*

RF Radio Frequency *15, 41, 57*

RTL Register Transfer Level *5, 27, 28*

SCAX SystemC AMS extensions eXperiments *36, 46, 55*

SDF Synchronous Data Flow *32, 34*

SiP System in Package *1, 7, 8, 39*

SoC System on Chip *1, 2, 5, 7, 8, 39*

SPI Serial Peripheral Interface *93*

SVD Singular Value Decomposition *140*

SW Software *1–4, 6–9, 11, 19, 22, 26, 27, 31, 38, 39, 41, 49, 57, 59, 84, 86, 88, 101–103*

SysML Systems Modeling Language *3*

TDF Timed Data Flow *31–36, 50, 55, 62, 64, 67–69, 73, 78, 79, 93, 98, 103*

THELMA® THick Epipoly Layer for Micro-actuators and Accelerometers *63, 88, 135*

TLM Transaction Level Modeling *27–29, 31*

TPWL Trajectory Piecewise-Linear *24, 140*

UML Unified Modeling Language *3, 67, 73*

UVM Universal Verification Methodology *6, 28, 72*

VLSI Very Large Scalable Integration *22, 42*

List of Figures

List of Listings

List of Tables

Chapter 1

Introduction

"There is plenty of room at the bottom."
Richard Feynman, 1959 [1].

Embedded systems benefited over the last decade from the shrinking of subsystems, better integration into networks and more efficient autonomy. Microelectromechanical System (MEMS) devices have been massively integrated in embedded systems and more generally in Cyber Physical Systems (CPS) [2]. CPS integrate computing, communication, and storage capabilities with the monitoring and/or control of entities dependably, securely and efficiently [3]. They operate in real-time in the physical world, from the nano-world to large-scale wide-area systems of systems. These smart systems embed among others mechanical and electrical sub-systems, like MEMS [4]. These parts are connected to one another through Hardware (HW) and Software (SW) components responsible for their control and monitoring [5].

MEMS or microsystems are micro-scaled devices usually manufactured by lithographic technologies [6], initially developed for CMOS microchips and based on semiconductor process. These systems rely on the principle of energy conversion between mechanical movable parts and electrodes. They are packaged with microelectronics in order to integrate sensing and actuation, digital processing and control functions in a single package (SiP) or in some cases into a monolithic integration through single chip (SoC) [7, 8]. Developed for years, MEMS devices find a broad range of applications, for example inertial, fluid or pressure sensors [9, 10, 11], ink-jet headers [12] or micro-mirrors [13]. Moreover, the manufacturing processes enable the development of new kinds of devices such as microphones [14] or bio-sensors [15, 16, 17].

Historically, MEMS manufacturing relies on highly-specialized processes, custom-made for each device type. However the large amount of MEMS fabrication processes increases the production costs and extends the time to market. Simultaneously Application Specific Integrated Circuit (ASIC) and MEMS devices have been incorporated into continuously growing SoC and SiP batch productions [18]. MEMS technologies also tend to shorten the development time and reduce the related costs while meeting high-level performance and reliability requirements [19]. To this end, EDA vendors, foundries and fabless companies collaborate to standardize processes, likewise Integrated Circuit (IC) market [20].

To accelerate the design of MEMS devices, some virtual prototyping solutions propose to integrate MEMS devices into the CMOS traditional design flow through multidisciplinary simulation tools [21]. The MEMS designers can also exchange low-level information in a structured and efficient manner through MEMS Intellectual Property (IP) libraries and PDKs (Process Design Kits) delivered by foundries [22]. Additionally, the test engineers can specify some environment constraints and appropriate operating configurations. In practice, MEMS, HW and SW components are considered separately with low-level descriptions. For instance, ICs and SoCs are often implemented in languages such as C, assembly, or Hardware Description Languages (HDLs) like VHDL or Verilog. Similarly, MEMS are modeled in Finite Element Method (FEM) models which contain lots of details and may slow down the computing. These solutions are well fitted for the fabrication and the design of devices themselves, but need to be completed to also ensure their correct integration of into heterogeneous architectures. To this purpose, we investigate higher-level modeling methods to bind MEMS, HW and SW components into dedicated simulation frameworks. We consider the virtual prototyping offers novel opportunities at system level to tackle the integration of embedded systems.

In this thesis, we explore the question of **how to efficiently integrate MEMS devices within electronic system-level design environment while preserving modeling accuracy and simulation performance**. To solve this problem, we are interested in the definition of a unified simulation framework in which system-level models of ASIC and MEMS are merged in what we call **multi-physic virtual prototypes**. To this end, we integrate the behavioral description of MEMS within the Analog/Mixed-Signal (AMS) extensions of the system design language SystemC. Our solution consists in an Application Programming Interface (API) between the commercial tool *MEMS+*® [23] and the standard implementation SystemC-AMS [24]. We hope that higher level representations will lead to faster integration and provide strong robustness and efficiency guarantees, as well as an improved designer productivity. Moreover, we think that such an investigation is valuable to highlight some limitations and possible improvements of existing solutions.

1.1 Multi-domain virtual prototyping

Today, micro-electronics manufacturers use platforms to test processor applications with companion sensors. The key objective is to verify the system responses to software requests, e.g., interrupts or events assertion upon threshold overruns. Moreover, such tools aim to improve the power management and calibrate the embedded software applications. To this end, these platforms must include devices like MEMS to guarantee the coherency and the calibration of the overall system which depends for instance on the device mechanics or sensor positions. The state-of-the-art solutions aim to virtually reproduce the behavior of MEMS either through co-emulation or by reading results of experiments, as summarized in Figure 1.1. In co-emulation, the platform is directly interfaced with the physical sensors while the software is emulated on a virtual platform running on a server. The co-emulation solutions also require a complex setup and are limited to low parametric configurations. Alternatively, results of experiments on the actual devices can be recorded and stored in database in order to replay the corresponding sequence as input of the HW/SW platform. Despite an easier setup, this solution still remains limited to specific use cases. We therefore explore an alternative solution which is fully based on virtual prototypes in order to make the platforms more open and modular, improve the reproducibility of experiments and fulfill the requirements of the product assembler.

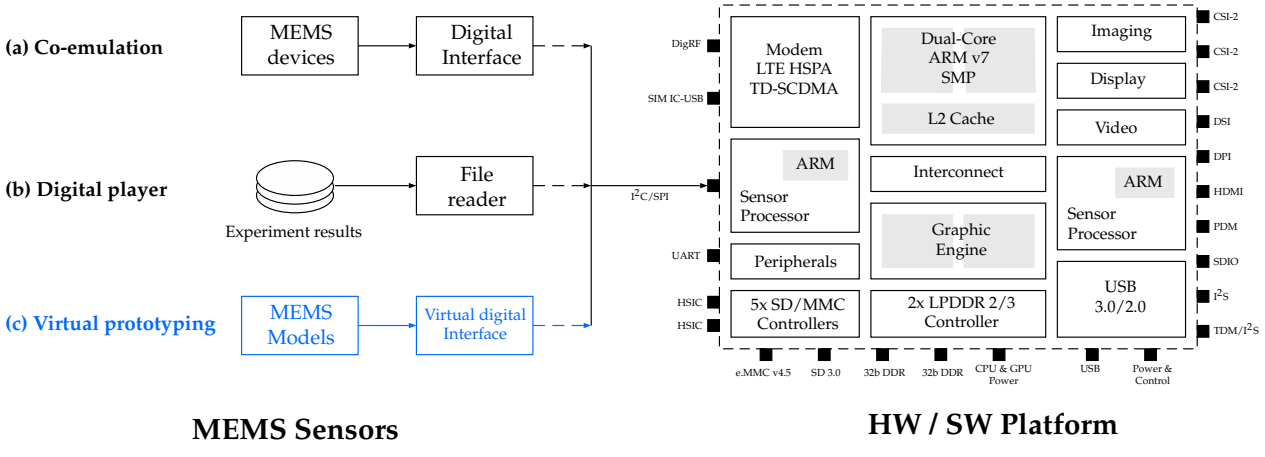


Fig. 1.1: State-of-the-art HW/SW platform interfaced with MEMS sensors. This platform was presented by STMicroelectronics during the European project H-Inception. The co-emulation solution (a) refers to the actual devices while the digital player (b) generates the stimuli observed in experiments and stored in databases. Our study aims to replace such infrastructures by a complete virtual prototype (c) which comprises both the MEMS devices and the AMS front-end and is connected to the rest of the HW/SW platform.

Our motivation is to find a lighter and more reproducible solution than the current ones employed by electronic manufacturers. To this end, we focus our study on the use of virtual prototypes able to represent complete solutions comprising MEMS, HW and SW parts. A virtual prototype is an analytical model of selected design properties. This allows the engineer to predict the system behavior without building expensive, inflexible, physical prototypes.

Model-based design produces a global view of the system and enables further refinements through high-fidelity mathematical models in specific area, e.g., kinematics, multibody dynamics or electronics. The related solutions setup the rules for syntax, semantics and pragmatics to define the representation, the meaning and the edition of the corresponding models, respectively. The Object Management Group standardized most of syntax used in model-based systems engineering [25] in regard to block-diagram principles. For instance, Unified Modeling Language (UML) [26] provides a generic visual syntax to represent systems as sets of interconnected blocks, or actors, with respect to connectivity and communication rules. SysML [27] restricts UML principles to systems engineering and targets the functional and structural verification of systems. To that purpose, SysML introduces an internal block diagram notation and the use of flow ports. The behavioral description of systems remains flexible enough to allow multiple behavioral definitions of the same entity. Alternatively, MARTE [28] supports performance analyses and quantitative predictions through a common way of modeling both hardware and software aspects. This enables a better interoperability between development tools used for the specification, design or verification steps. The aforementioned standards only address the syntactic modeling of systems and must therefore be complemented by simulation environments to fully anticipate their behavior.

The complexity of systems induces a tight interplay across domains (mechanics, electronics, software, communication networks ...) and requires more integrated tools to provide Multi-Domain Virtual Prototyping (MDVP) solutions. MDVP intends to merge domain-specific methods into unified modeling and

simulation environments to address the heterogeneous nature of embedded systems. Co-simulation and coupled simulation are the two complementary approaches commonly employed to this end.

On the one hand, co-simulation combines several domain-specific simulators. This approach defines interfaces between different modeling tools. Such frameworks support the necessary concurrency and communication mechanisms to orchestrate the overall simulation without the structural and abstraction constraints of hierarchy. Based on detailed models, this method ensures the accuracy of results but may lead to long computation time that is not acceptable in early-design phases. The deployment of standard implementations, e.g., High-Level Architecture [29] or Functional Mock-up Interface (FMI) [30], aims at the synchronization of different software tools. Dedicated interfaces, like Functional Mock-up Unit [31] in FMI, address the proper cross-tool integration. The corresponding programmable interfaces and software solutions also need to be actively maintained and documented to guarantee the large adoption and the global compatibility of the tool chain. Additionally, the combination of traditional tools can lead to incompatible syntax, misleading semantics and inconsistent pragmatics, as demonstrated by Lee in [32].

On the other hand, coupled simulation tackles the heterogeneity of systems by specifying a unified modeling language. Rather than addressing software problems of tool integration, the multi-paradigm modeling is focused on the semantics of inter-operation, i.e., the structural definition and the synchronization of multiple Model of Computations (MoCs). A MoC supports a particular definition of time and refers to domain-specific modeling and simulation rules. Nonetheless such a unified language cannot meet all requirements of a system modeling and specification tool. Therefore, the principle of MoC is generally applied in order to achieve strong semantics, yet address heterogeneity and provide mechanisms to allow heterogeneous models to interact concurrently [33]. In this case, the simulation core allows the addition of new MoCs to cover different physical domains. Such an architecture must preserve the integrity and coherence of simulations while staying scalable [34].

Ptolemy II [35, 36] provides a coordination framework and describes systems through the actor-based modeling technique. The actor-based description deepens the traditional use of block diagrams by adding information on the structural link that exists between components. Furthermore, this approach supports the notion of hierarchy directly in the model since an actor can itself contain several actors. The Ptolemy approach [37] influenced many of coupled simulation environments. For instance, SystemC architecture is extensible and could offer high-fidelity modeling framework with specific MoC structures and simulation facilities. Nevertheless, its current micro-kernel structure implies that any additional MoC must refer to the underlying Discrete Event (DE) simulator. Alternatively, ModHel'X [38] decouples the modeling from the execution process and refers to meta-modeling techniques in order to support component-oriented and hierarchical semantics. ForSyDe [39] rather extends the usage of MoCs in abstracting functions and features of complex heterogeneous systems. The low expressive and high-fidelity level of such models can address the abstraction and formal verification, but is not necessarily for HW/SW co-design.

Multiple tools already implement MoC-based fixed solutions to address system-level design of heterogeneous assemblies. MATLAB® [40] was first intended to provide routines to the powerful LINPACK [41] and EISPACK [42] libraries, developed to solve linear equations and eigenvalue problems, respectively. Built upon these initial matrix packages, MATLAB provides a general-purpose programming language that incorporates standard functions for the solution of Ordinary Differential Equations (ODEs). This language can be applied to simulation of continuous and discrete-time systems too. MATLAB/Simulink, or the open-

source equivalent Scilab/Scicos [43], indeed enables model-based design and system simulation. Both provide a graphical input to define models and handle hierarchical structure through block diagrams. Benefiting from the mathematical foundations of MATLAB, Simulink can dynamically solve complex matrix functions. This environment is extensively used to design applications in micro-mechatronics [44]. Moreover, toolboxes and APIs enable the co-simulation with external solvers, e.g., to test HW prototype or real-time software [45].

Alternatively, Modelica® [46] implements a modeling language with the intent to ease the description of multiphysics systems. This uses expressions essentially referring to the mathematical equations of the underlying model. Hence, Modelica models are usually based on a functional description of systems and refer to non-causal modeling with true Differential Algebraic Equations (DAEs) and ODEs [47]. As an object-oriented language, Modelica is well adapted to design purposes and aims to easily exchange models and model libraries. Moreover, Modelica, together associated with simulation tools such as Dymola or OpenModelica, supports the simulation of systems mixing continuous and discrete time descriptions in the FMI environment. LMS Amesim® [48] is another versatile physical modeling and simulation tool using primitive components and predefined libraries.

1.2 Electronic system-level design

HDLs shorten SoC design phases by modeling and verifying digital ICs through virtual prototypes. These languages are constructed on top of a single MoC that defines the elements, relationships and events further combined into models. The associated discrete-event simulator generates and updates the value of variables with a timestamp as defined by operations. Their relative cost can thus be evaluated regarding the underlying hardware obtained from logic synthesis. In industry, VHDL [49] and Verilog [50] are the de facto standards to address the digital simulation of ICs. In order to include analog front-end components, these HDLs were extended to analog simulation, e.g., with VHDL-AMS and Verilog-AMS [51]. Nevertheless, these solutions remain decoupled from the development of software running on the targeted architectures.

To address SoC growing complexity, specific modeling and analysis methodologies were developed. For instance, ad-hoc C/C++ based models are commonly used by designers to validate basic architectural choices. In addition, algorithms are defined and validated by complementary dataflow models targeting an optimal implementation, either on hardware or software. Moreover, IP reuse and associated techniques, such as platform-based design, notably accelerate SoC back-end implementation processes, i.e., from Register Transfer Level (RTL) to the layer definition [52]. These steps are now considered reliable and cost-efficient. As a consequence, the major design phases shift up in abstraction level to system level, denoted Electronic System-Level (ESL) design [53]. ESL enables the concurrent development of application-specific software and hardware systems. To that purpose, higher-level languages emerged over the last decade [54]. Involving multiple MoCs, they handle additional timing constraints and data types. Furthermore, these languages extend the capabilities of traditional HDLs with a better support for concurrency, communications and configurability of models [55].

SystemC became in the last few years a C/C++ modeling standard for systems architects [56]. This C++-based library enables the system-level simulation and the co-development of digital hardware and software. This is based on a single fixed MoC, the same as existing HDLs, namely the discrete event semantics.

Therefore a system is described as a set of interconnected modules processing threads or methods, each representing the system in a behavioral or functional manner [57]. These functions are then scheduled and executed by the SystemC event-driven simulation kernel. Besides the co-development of software and hardware, this standard enables the functional and behavioral modeling of ICs [53]. Furthermore, this simulation environment enables an early functional and architectural verification through dedicated frameworks like Universal Verification Methodology (UVM) [58].

SystemC AMS extensions enable the continuous systems simulation in SystemC [24]. This standard implements the synchronization between discrete-event and discrete-time solvers in charge of digital and analog components simulation, respectively. Moreover, SystemC AMS layered architecture allows the definition of additional MoCs [59], each dedicated to a specific physical domain. Another potential asset of SystemC-AMS lies in its further coupling with a verification process like UVM [60]. Nonetheless, SystemC AMS still remains limited to efficiently recover time-continuous behaviors, especially the nonlinear ones. Despite these drawbacks, SystemC AMS is a valuable support to simultaneously handle the simulation of software and digital hardware with additional non-electronic peripherals, in particular MEMS. This standard is used in the commercial software Coside® [61] which also supports VHDL-AMS to provide a complete design flow for AMS solutions.

Table 1.1: Overview of the frameworks, projects and software solutions for system-level modeling and simulation of embedded systems comprising HW, SW and multiphysics elements.

		Modeling			**Simulation**		
		Digital HW	Embedded SW	Multiphysics	MoC-based architecture	Single kernel	Co-simulation
Frameworks	ModHel'X	~	✗	✓	✓	✗	✗
	ForSyDe	✓	~	✓	✓	✗	✗
	SystemC	✓	✓	✗	✓	✓	~
	SystemC AMS	✓	✓	~	✓	✓	✗
	Ptolemy II	✓	✓	✓	✓	✓	~
Projects	RapidMPSOC	✓	✗	~	✗	✗	✓
	SMAC	✓	~	✓	✗	✗	✓
	Beyond Dreams	✓	✓	~	✓	✓	✗
	H-Inception	✓	✓	✓	✓	✓	~
Software	LMS Amesim	✗	✗	✓	✓	✓	✗
	Modelica	~	✗	✓	✓	✓	✓
	MATLAB/Simulink	~	~	✓	✓	✓	~
	Coside	✓	✓	~	✓	✓	~

The aforementioned solutions address the simulation of heterogeneous systems through specific software tools or dedicated frameworks. Nevertheless, the rise of SoC/SiP in embedded systems encouraged a digital-centric approach to build virtual prototypes. Alternative methods therefore emerged from the electronics community to bind HW/SW development with system simulation environments. We reviewed hereafter some of the research projects that contributed to the development of novel standards. RapidMP-SOC [62] and SMAC [63] projects were focused on co-simulation environments to enable the multi-domain simulation. One interesting result of the SMAC project relies on the automatic extraction of SystemC-AMS and C++ models from heterogeneous system descriptions as shown in [64]. This thesis was conducted during the H-Inception project that followed up the Beyond Dreams project which aimed to enlarge SystemC with AMS extensions [65]. One noticeable contribution of H-Inception is the proof of concept of a unified simulation environment for the MDVP with respect to SystemC and the MoC principles [66]. The aforementioned frameworks, projects and commercial software solutions are summarized in Table 1.1.

1.3 MEMS modeling and simulation

MEMS are generally designed and simulated in dedicated Computer Aided Design (CAD) tools, mostly through Boundary-Element Method (BEM) or FEM [67]. These fined-grain models have a large number of Degrees of Freedom (DoF) and are too complex to be further integrated at higher level in the models of complete heterogeneous systems. The complexity of such descriptions is indeed relatively high and results in more than thousands or millions possibly nonlinear coupled Partial Differential Equations (PDEs) or ODEs. The computation and solving of such systems is time consuming and cannot be directly used in a circuit-design environment. Therefore MEMS system-level design intends to produce functional or behavioral equivalent compact models to speed up the simulation.

Macro-modeling encapsulates the multiphysical behavior of MEMS (mechanics, electrostatics, fluidic damping) into small components. Specific commercial solutions [68] already implement this modular approach and rely on MEMS dedicated high-order Finite Element Analysis (FEA), which concatenates the multiphysics into one single equations system. This is in contrast to traditional FEM tools, which usually rely on solver coupling for multi-physical simulation. Nevertheless, the transient simulation of FEA is likely perceived as too slow for MEMS/ASIC co-design [69]. To address the system designer's needs for simulation speed while preserving the critical nonlinear characteristics of a device, the reference model must be reduced to a lower but self-sufficient number of DoFs, e.g., through Model Order Reduction (MOR).

MOR is a well-established mathematical theory and an efficient technique to generate compact and accurate models of large-scale dynamical systems [70]. This results in a reduced model, i.e., a small system of ODEs, that is derived and computed to recover the solution of original system. In MEMS design, MOR can be considered as solved in principle for linear systems regarding the support in simulation software of robust numerical algorithms in order to automate the process [71]. Latest research are focused on establishing error estimators to guarantee the quality of compact views from large-scale systems [72]. In addition configurable reduced models enable to preserve parameters and thus evaluate the design variations without performing MOR again [73]. Moreover, the MOR methods frequently handle nonlinearity of the system by approximation either through a polynomial of low degree or a translated quadratic bilinear form. An extensive review of MOR techniques in nonlinear cases is provided in [74]. The simplest technique remains the quadratic method which can be reinforced by the bilinearization method or variational

analyses which lead to lower errors. In case of strong nonlinearities, piece-wise linear approximations are preferred to handle the dynamic disparities of the system and refer to "snapshots" of intermediate states [75]. All these methods derive from frequency-domain MOR methods whose accuracy is measured by the related transfer functions.

Combining reduced models with HDLs is a promising way to enable the global system simulation. The digital part is usually described in HDL like VHDL or Verilog and can be broaden to analog parts thanks to AMS extensions. In addition, the AMS extensions of HDLs partly enable the simulation of continuous systems through discrete-time solvers [76]. In the case of MEMS, reduced models have been successfully integrated to AMS extensions of HDLs and found multiple applications as detailed in Section 2.3. Commonly based on VHDL-AMS or Verilog-AMS, we are interested in this work in higher-level descriptions in SystemC-AMS.

Besides reduced models, AMS extensions of HDLs are used to model microsystems through lumped elements. These models apply the fundamentals of electrostatics and mechanics in equivalent descriptions, for instance resonant systems like mass-spring-damper systems. The system-level simulation also raises the level of abstraction to decrease the amount of detail and enable fast simulations [77]. Despite faster computation, such models need to access parameters generally issued from more detailed models in FEA or collected from experiments. In addition to a cumbersome definition, the models have a limited accuracy and may produce severe approximations, especially in nonlinear cases. This matters since the complexity of MEMS devices is growing in terms of geometry and features. Lumped-element models are already supported by most AMS extensions of HDLs, but cannot be considered as accurate enough to appropriately tune hardware or software.

1.4 Contributions

In this thesis we propose several innovations for the system-level modeling of MEMS in order to provide suitable multi-physics virtual prototypes. In particular, we address the current limitations of SystemC-AMS with the exploitation of reduced models directly issued from the FEA tool *MEMS+*. We believe that our contributions improve the integration of MEMS devices in complex architectures and propose an alternative to the MoC-based approach by directly coupling SystemC-AMS to the commercial software solution *MEMS+*.

Evaluation of ESL-based system-level modeling

This work is focused on the simulation of MEMS once implemented with the surrounding IC, i.e. as SoC or SiP. SystemC applies ESL design principles and provides an efficient way to tackle HW/SW co-development. The digital-centered approach of SystemC is widely adapted from traditional HDLs. Moreover, the microkernel structure of SystemC allows flexibility and encourages the development of additional MoCs and solvers. For instance, AMS extensions have been implemented in order to model and simulate analog or non-electronic sub-parts. In the following, we specifically use the SystemC-AMS standard to model and simulate MEMS in complex architectures. The standard implementation of SystemC-AMS partly supports system-level modeling methodologies well-adapted to MEMS devices. In particular, we define state-space systems or transfer functions by applying first principles and implement these models in the standard SystemC-AMS MoCs. Additionally, we explore the capabilities of state-of-the-art extensions to provide equivalent descriptions through lumped-element circuits or bond graphs. Although these meth-

ods provide satisfactory results, they present strong limitations both in terms of modeling and simulation. This introductory study intends to evaluate the top-down approach supported by SystemC-AMS through high-level models of MEMS using common modeling methodologies by highlighting the difficulty to correctly handle the complex geometry of devices.

***MEMS+* API for the system-level simulation of reduced models in SystemC-AMS**

The generic definition, even of classical devices such as accelerometers or gyroscopes, may lead to approximations that cannot be acceptable when targeting the system verification. Since MEMS behavior is strongly conditioned by the geometry of the device, MEMS behavioral description must refer to the geometry and therefore requires in-depth information from experiments or detailed FEM models to ensure the correct configuration of models. We also support the idea that reduced models are more appropriated than lumped-element ones to elaborate efficient multi-domain virtual prototypes containing MEMS. To this purpose, we use the reduced models exported from *MEMS+* to correctly handle the coupling existing between mechanics and electrostatics in MEMS devices.

A programmable interface is proposed in order to directly integrate reduced-order models in SystemC-AMS. The reduced models are generated from preliminary FEA and spatial discretization of electromechanical coupling, here the FEA tool *MEMS+* [78]. These models are issued from a tested and already implemented feature that guarantees the preservation of electrostatic nonlinearities [79]. Furthermore, we analyze the influence of the simulation time step on its performance, especially with regard to the frequency distribution of models and the selected integration schemes that may require high sampling rate.

The presented interface enables SystemC-AMS, and to some extend C++ users, to instantiate and access reduced models through a set of functions and methods. We provide the necessary background to configure the model and import results in *MEMS+*. Additionally, we define functional simulation elements, such as test benches or stimuli generators. This serialized framework intends to increase the re-use of the code and enable multiple scenario of simulation. The proposed API aims to take advantage from SystemC-AMS without requiring any additional MoC and refer to accurate models initially created in *MEMS+*.

1.5 Outline

This thesis is structured as follows. Chapter 2 provides background on MEMS virtual prototyping and HW/SW co-development. We present the principles of ESL design and MOR techniques and review the state-of-the-art solutions to integrate reduced models in HDLs. SystemC and its extensions are also introduced. Chapter 3 details the system-level modeling of MEMS in SystemC-AMS. We provide a comparative study of existing modeling methods which are supported by the standard and the related MoCs. The limitations of equivalent representations of accelerometers and gyroscopes are discussed in order to justify the use of reduced models. In Chapter 4, we introduce the proposed API with *MEMS+* software. We compare our solution to the standard functions in order to assess the better performance and stability of the reduced models. In addition, we propose a generic framework to ease the definition of test benches and facilitate the reuse of the code. The proposed methodology is demonstrated and validated in Chapter 5. The full and reduced *MEMS+* models of an accelerometer are simulated and compared with regard to the state-of-the-art simulation tools and the SystemC-AMS standard. Chapter 6 concludes with a discussion of contributions, topics for future work, and closing thoughts.

Chapter 2

State of the art

2.1 Introduction

Virtual prototyping poses several challenges in modeling and simulation. To anticipate the fabrication and implementation of complex systems such as MEMS, an in-depth understanding of their behavior and interfaces is indeed required. Traditionally, to develop and test new devices, microsystem designers elaborate models either based on analytical formulas or refined in FEM. The outcome of either approach generally leads to contradictions with experimental results. A trial-and-error methodology is applied in order to recursively refine models and designs leading to development times of several years. This approach directly impacts the development with long and costly cycles. Hence modeling and simulation are still an essential need for MEMS designers to advance and enhance the technology [80].

The key-aim of modeling and simulation tools is to provide a reliable description of the targeted devices. The fabrication, design and implementation phases can benefit from more accurate and configurable models. First, by lowering the need for experimental verification, reliable simulation results can shorten the development phases from month to weeks. Second, robust and reliable models broaden the design space. For this reason, the designers can explore novel solutions and propose more aggressive choices both in terms of geometry and feature requirements. Third, the devices already manufactured and commercialized devices could be refined and improved from unexplored design solutions. Finally, beside the correct estimation of internal phenomena, MEMS designers must also ensure the good integration of MEMS with the surrounding electronics. To this end, we explore higher-level descriptions to be compliant with IC models and integrated in system-level simulation environment. Here we propose novel methods to create MEMS, HW, SW models at high level and allow in the end the simulation of complex systems.

In this chapter, we attempt to review the state-of-the-art techniques in MEMS system-level design. The fined-grain models initially created in FEM tools are too complex to be considered for higher-level simulation. The macromodeling aims to speed up the simulation by encapsulating the multiphysical behavior of MEMS (mechanics, electrostatics, fluidic damping) into small configurable components. After introducing the underlying physical principles of electrostatic transducers, we explore the capabilities of component-based FEA and further discuss the different simulation strategies to correctly address the system-level integration into larger assemblies. We then assume the benefits of even more compact descriptions by using MOR techniques. These methods produce accurate models that preserve the main characteristics of MEMS. Due to their small size, these models are well-suited for fast simulation and have been extensively used in the co-design of MEMS and IC through HDLs. Additionally, we consider SystemC as a standard

language to co-develop hardware and software. As the abstraction level progressively evolves with ESL design, we indeed envision SystemC-AMS as the preferred simulation environment for complete MEMS devices represented by both reduced electromechanical and IC models. SystemC AMS extensions are also introduced and discussed against alternatives.

2.2 MEMS system-level design

Transducer design requires finding the proper shape elements and structural dimensions of MEMS to fulfill all requirements at the given operational and environmental conditions [81]. To this end, modeling and simulation aim to describe the physical effects occurring in MEMS without building expensive prototypes. System-level and physical-level models are successively used during the design process.

The component-based modeling method, e.g., implemented in *MEMS+*, defines a preferred layout with preliminary structural dimensions reliant on the main physical parameters. These models are frequently based on lumped elements, such as parallel plate capacitors or simple beams. These are derived from physics fundamentals and yield small ODE systems that can be solved within a few seconds. These models also capture a lot of the typical MEMS behavior and address first conceptual evaluation of the design. However, to effectively build physical prototypes, refined models are required.

The multidisciplinary foundations of MEMS introduced coupled electrodynamics effects. The shrinking of systems also influences the force distribution and may change the material macro-properties. At micro scale, electrostatic forces indeed become very important as well as the damping due to the viscosity of surrounding fluid. Therefore, the dynamical characteristics are more dependent on surface and interface effects than bulk or volume forces. This can also lead to fringing effects directly impacting the performance of the microsystem. In consequence, the modeling and simulation of MEMS dynamical characteristics rely on accurate structural description.

Physical-level models describe the complete geometry of the device through three-dimensional PDEs. These are discretized and solved with the BEM or FEM. Such descriptions also address the tight coupling between electric and mechanical domains, especially the surface interactions between both related fields. Therefore, coupled solutions are generally preferred to the solutions dealing with the elastic and electrical parts separately. Furthermore, these equations are highly nonlinear and require specific solving methods to provide an effective investigation of MEMS dynamics, e.g., finding the maximum allowable voltage point to avoid pull-in effect. Due to the complexity and size of models with thousands of DoFs, these simulations require extensive computation resources in order to provide highly accurate results. Nevertheless, physical-level models are progressively employed to automatically generate system-level models through MOR techniques. We next review the evolution of physical modeling to macromodeling solutions in order to further use reduced models in a system-level simulation environment, such as SystemC-AMS.

2.2.1 Electrostatic transducers

Electrostatic, piezoelectric and magnetic forces are employed in MEMS devices to sense or act on the motion of mechanical parts. This study is centered on electrostatic forces that indeed appear at the surface of mechanical structures. On small scales, these forces are predominant compared to volume forces and have comparable performance to electromagnetic forces. They also benefit from low power consumption

properties and fast response time. Therefore the electrostatic transduction is one of the most common actuation and sensing method in MEMS devices due to its simplicity and high efficiency. To manufacture these devices, simple parallel-plate capacitors or comb-drive configuration are commonly employed. We discuss hereafter the basic mathematical and physical relationships related to these.

2.2.1.1 Fundamentals

We first consider the mechanical and electric problems individually. Boundary value problems are introduced with regards to domain equations and boundary conditions. This description yields the definition of energy functionals adapted from the variational principles. Energy functionals are also expressed and refined regarding the different fields of interest in both domains. These formulations are further combined in order to define the coupling problem occurring in MEMS. The associated boundary value problems are then solved through domain discretization, generally the BEM or the FEM. The literature on solid mechanics and coupled systems is extensive, and we refer the reader to dedicated books for additional details on methods and formulations related to the electromechanical coupling [67, 82, 83].

In solid mechanics, problems are defined by three basic components. First, the equations of constitution relate stresses to strains. Second, the equations of geometry of deformations relate displacements to strains. Lastly, the equations of equilibrium relate external traction and body forces to stresses. Respectively denoted as the constitutive, compatibility and equilibrium equations, these domain equations describe what is happening inside the body. In order to fully define the associated mechanical problem, boundary conditions also describe what is happening on the surface of the body. These boundary conditions comprise given data about the displacements and applied forces on the surface. The domain equations and associated boundary conditions define a so-called *boundary value problem*. By analogy, these fundamental equations and boundary conditions find equivalent definitions in electrostatics. Combined with each other, these equations govern the behavior of electrostatically actuated microsystems (see Appendix A). The resulting electromechanical coupling in micro-systems is expressed in terms of energy density as follows:

$$W = W^{int} - W^{ext} = \left(W_m^{int} - W_e^{int}\right) - \left(W_m^{ext} - W_e^{ext}\right), \tag{2.1}$$

where W^{int} and W^{ext} are the sum of internal and external energies which both consist in electrostatic members W_e^{int} and W_e^{ext} and mechanical ones W_m^{int} and W_m^{ext}.

The derivation and discretization of (2.1) yield computational expressions of the energy distribution [84]. The functionals related to each domain are combined with one another to define the energy density, with regard to the mechanical and electrostatic contributions. The classical approach, denoted *primal/primal* approach, is further introduced in [85]. Note the computation of electromechanical coupling in *MEMS+* derives from this method and is assumed in the following as the reference modeling technique for MEMS. Alternative strategies can be employed to solve the coupling problems. For instance, Rochus et al. [86] propose a *primal/dual* approach referring to the electrostatic dual functional to enrich the definition of surface forces definition and better address convergence issues.

2.2.1.2 Reference problem

To understand transducer operations, we consider the simplified model of a parallel plate capacitor shown in Figure 2.1. This is composed of two conducting plates with arbitrary shape. The electric field lines are assumed perpendicular to the plates, even near edges, which is equivalent to assuming infinite plates. Considering the voltage source, the charge of the capacitor reads:

$$Q = C\,V \text{ with } C = \frac{\varepsilon\,A}{g}, \tag{2.2}$$

where Q is the electrical charge, C is the capacitance, V is the voltage across the capacitor, ε is the relative permittivity of the gap space medium, A is the overlap area between the two electrodes, and g is the distance separating both electrodes. This relationship is obtained by applying the Laplace equation governing the electrostatic potential between the plates and neglecting the fringing [4]. For electrodes of simple geometry, an analytic expression can be found for C. However, for more complicated geometries, the capacitance is function of additional DoFs and the fringing effects can not be neglected anymore. FEM or BEM are also used and combined with curve fitting to characterize the relationship between C and these DoFs.

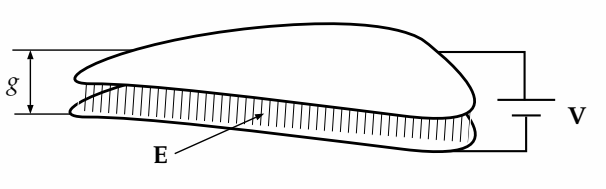


Fig. 2.1: Theoretical representation of an electric field (**E**) in a parallel plate capacitor with variable gap (g) actuated by a voltage source (V).

The electrostatic force can be derived from the expression of the co-energy stored in the capacitor which is the electrical potential energy W_e stored in the capacitor expressed as :

$$W_e(q) = \int_0^Q \frac{q}{C}\,\mathrm{d}q = \frac{Q^2}{2\,C} = \frac{1}{2}\,C\,V^2\,. \tag{2.3}$$

Hence, the energy varies in a quadratic way regarding the charge evolution and is inverse proportional to the gap variation. For instance, by positively varying the voltage at fixed gap, the charge will increase. Conversely, the gap will decrease if a positive external force is applied with a constant charge. Finally, the electrostatic force corresponds to the negative gradient of the potential energy, W, and depends on the electric field, $E = Q/(\varepsilon\,A)$, as follow :

$$F = \frac{1}{2}\,Q\,E = \frac{1}{2}\,\frac{Q^2}{\varepsilon\,A}\,. \tag{2.4}$$

The internal behavior of electrostatic microsystems has been introduced through boundary value problems with the governing equations of energy coupling. This also highlights the importance of surface effects occurring in MEMS. To address the modeling and simulation of such coupled systems, the previous

models are computationally solved through domain discretization, e.g., by BEM or FEM. The parallel plate capacitor finally illustrates a first application of these methods and is next assumed as a basic component of electrostatic actuators and sensors.

2.2.2 Component-based modeling

MEMS devices are composed of elementary micro-structures. For instance, electrostatic actuation and sensing mechanisms are mostly configured as either parallel plates or comb drives. Their mechanical response can be further approximated by components such as beams, plates or suspended structures. Therefore MEMS system-level modeling consists in the combination and the assembly of such basic elements in order to create more complex assemblies. We next introduce the principal components used at system-level and their related applications.

2.2.2.1 Microelectrostatic elements

The simplified model of parallel-plate capacitor depicted in Figure 2.2 can be part of an actuated mass-spring system. Its dynamic behavior is also governed by the electrostatic load, the restoring mechanical force and the damping force. The electrostatic load is composed of a Direct Current (DC) voltage and a smaller amplitude Alternating Current (AC) voltage. The DC polarization voltage generates an electrostatic force directly acting on the movable plate. Additionally the AC voltage makes the plate oscillate around the newly established equilibrium position. This simple configuration is widely used in both actuation and sensing applications.

Another capacitive configuration is referred as comb drives, which are widely used in MEMS design [87]. Comb drives present some advantages over parallel-plate actuators. They indeed have larger driving distance due to larger movement gap and avoid bi-stability [1] [89]. In addition, the actuation amplitude varies linearly with the applied driving voltage. Furthermore, comb drive can be configured laterally or transversely. Sliding film damping is more important in comb drives because of the orientation of the vibration which is parallel to the plane of substrate, either translational or rotational. By contrast, in parallel-plate, the squeeze-film damping is the dominant source of dissipation. Such damping distributions directly influence the dynamic performance of the MEMS [90].

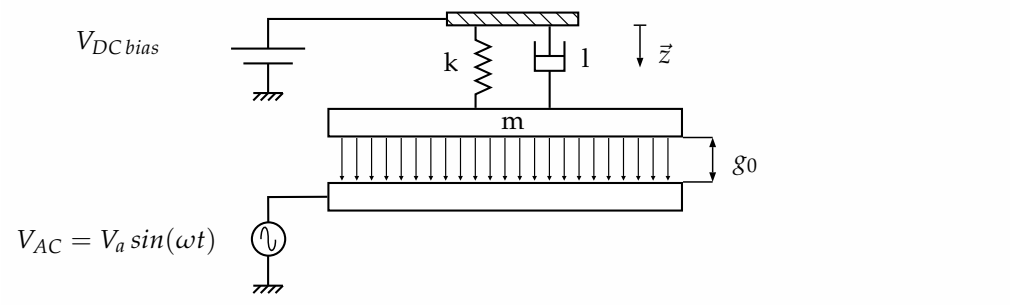
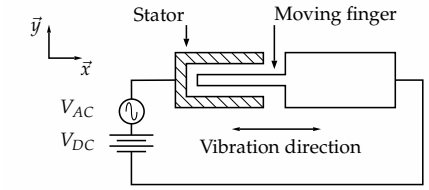


Fig. 2.2: Parallel plate.

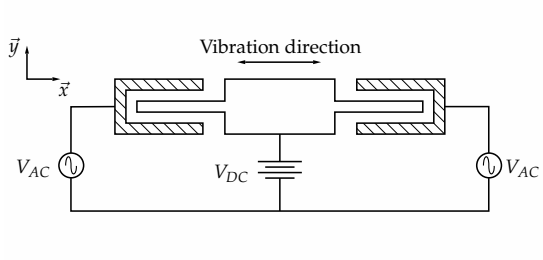
[1] Bi-stability offers the possibility to memorize a state without a continuous power supply, which is of interest to design, for instance, switches in Radio Frequency (RF) applications [88], but must be avoided in most of sensors or actuators.

In lateral configuration, a cell, i.e., a pair of fingers, consists in a movable part, the rotor, and a stationary part, the stator, as depicted in Figure 2.3(a). A driving voltage, composed of DC bias and AC drive, is applied between both electrodes to actuate the rotor. The comb drive configuration produces a second term in the electrostatic force that can be canceled trough push-pull actuation (Figure 2.3(b)).

The transverse comb drive is widely applied to sense very small displacements. This configuration indeed benefits from high capacitance sensitivity regarding the transverse movement, as compared with lateral comb structures (Figure 2.4(a)). Nevertheless side instability may occur in such a side comb and differential configurations are used to reduce such an effect as shown in Figure 2.4(b).

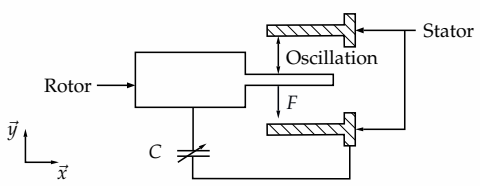


(a) Single lateral comb finger.

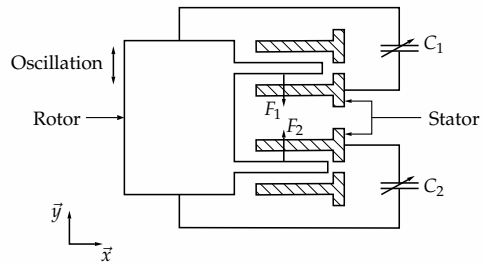


(b) Push-pull actuation

Fig. 2.3: Lateral comb finger arrays.



(a) Single transverse comb finger.



(b) Differential configuration.

Fig. 2.4: Transverse comb finger arrays.

Coupled field solutions are necessary to cope the nonlinearity of electrostatic forces occurring in such assemblies, as depicted in Sec. 2.2.1. This nonlinearity directly influences the stiffness constant of mechanical components and may result in a frequency shift [90]. For instance, the mechanical restoring forces are considered as linear in case of small deformations, but become nonlinear for large deformations. These induce stiffness nonlinearities that can be evaluated through FEA and expressed for example in the following cubic form of a resonant system like the beam illustrated in Figure 2.5:

$$F_e(y) \;=\; m\,\ddot{y} \;+\; b\,\dot{y} \;+\; k_1\,y + k_3\,y^3\;, \tag{2.5}$$

where $F_e(y)$ is the external electrical force, m is the mass, b is the damping, and k_1 and k_3 are respectively the mechanical linear and cubic stiffness coefficients. The cubic damping term has a direct influence on the usual Gaussian distribution of the deflection versus frequency curve. Hence, if $k_3 > 0$, the peak of the curve is bent towards right to higher frequency, and the non-linear spring of the system is denoted "hard spring". When $k_3 < 0$, the peak is bent towards left to lower frequency, and the non-linear spring of the system is denoted "soft spring". This phenomena is well described in [91, 92].

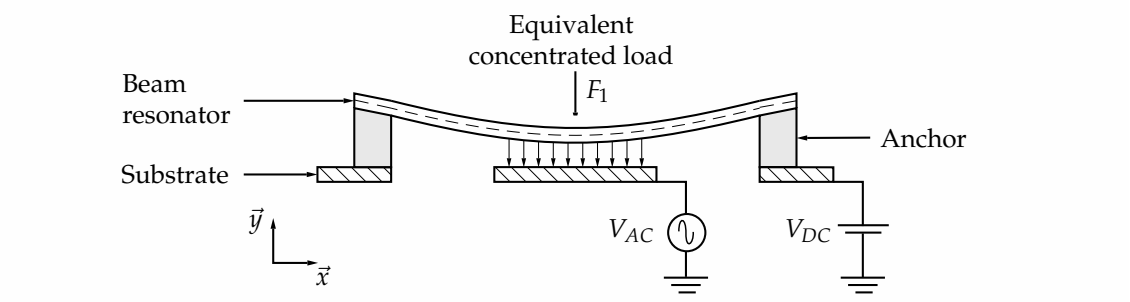


Fig. 2.5: Beam resonator.

The system description must therefore handle the corresponding stiffness softening or hardening to enable behavioral, dynamic analyses. For instance, pull-in is an instability that occurs in parallel-plate capacitors for any displacement over one-third of the initial gap. This effect pulls the movable plat all the way into contact with the fixed part. Within the rest of the gap, i.e. for small displacements, the movable plate can be positioned accurately. Additional effects due to the nonlinear dynamics, such as bifurcation or chaos, have also been covered in the literature on electrostatic forces.

A first application of parallel-plate capacitor consists in a beam resonator, as shown in Figure 2.5. The beam is deflected by the electrostatic force, generated by DC and AC applied voltages. The structure evolves under boundary conditions, such as clamped-clamped or cantilevered. In clamped-clamped configuration, the restoring mechanical force is composed of elastic, residual-stress and stretching forces. The elastic force evolves linearly with regard to Young's modulus and is dependent on the beam geometry [93].

As depicted above, MEMS devices with electrostatic actuation or capacitive sensing involve some nonlinearities, either electrostatic or structural. The system response tends to be nonlinear in particular configurations of the constitutive material or in case of large deflections compared to the device dimensions. The previous microelectrostatic elements are combined with models of basic micromechanical structures in order to recover such behaviors.

2.2.2.2 Micromechanical elements

MEMS design relies on common building blocks, such as beams, plates or suspended structures [94]. Classical theories of beams and plates, namely the Euler-Bernoulli beam theory and the Kirchhoff-Love plate theory, are widely employed to define one-dimensional problems with regards to structural mechanics. These methods permit the modeling of such structures by defining deformation in terms of a single field, i.e. the transverse deflection of a point on the neutral axis of the beam. The strains can be computed assuming the normal to the neutral surface remained normal after deformation. The resulting single governing differential equation is considered as an advantage although of high order. More general theories intended to define a single formulation which remains valid throughout the range of length to cross-section considerations. To this end, the Timoshenko theory [67] adds the transverse shear deformation to the other strains and is applicable on most geometries with refined mesh, i.e. with several integration points. Nevertheless, when applied to cases where the Euler-Bernoulli theory is viable, the Timoshenko theory can lead to locking effects which are observed when an element cannot interpolate a field properly with its nodal values

and the shape functions. These are counterbalanced with finite-element approximations which are useful when considering plate and shell problems [95].

Here the component-based approach is assumed as accurate enough despite some restrictions on the model definition. As an example, even in simple beams, residual stress may be observed. Although such an effect can be neglected in case of single-crystalline thin film, the stress residual may become significant enough in some MEMS designs to directly affect the dynamic behavior of the structure, e.g., the bending stiffness of the system. Additionally, the anchors induce stretching on structure boundaries which evolves in cubical way with displacement. Moreover, the stress within the beam are increased by the axial force loading that causes stretch at the neutral plane [96].

Based on the previous FEA, beams, plates or suspended structures can be represented as configurable elementary systems. These are commonly described with distributed parameters that can be solved by boundary value problems (Sec. 2.2.1). The linear analysis of such distributed models is approximated though closed-form solutions, applying the following assumptions. First, the displacements are small enough in amplitude to assume the strain-stress as linear. Second, the systems are considered as conservative in the sense there is no internal loss of energy nor external damping. Thirdly, external gravity is neglected. The resulting static and dynamic analyses aim to determine the eigenfrequencies as well as the amplitude of the system response [97].

2.2.3 Simulation strategies

The most efficient method to model electrostatic microsystems remains the variation approach previously introduced. This aims to properly estimate the electric field, i.e., the gradient of the electric potential. The quality of electrostatic models is therefore relative to the quality of computed electrostatic forces. In order to obtain a converged solution, the electric potential must also be computed with regard to the singularities observed in model, like mechanical or intrinsic nonlinearities as depicted in the above discussion. To this end, FEM and BEM *discretized solutions* are combined together either through co-simulation or coupled simulation.

In **co-simulation**, the structural and electric models are separately handled in two different codes. To compute the results, both codes must be recursively updated in each physical domain. The results are thus communicated in a synchronized manner between the corresponding staggered solvers. In practice, the mechanical displacements are sent to the electric model in order to update the geometry. Similarly, electric potentials or fields are sent to the structural model to generate electrostatic forces. Co-simulation enables the use of different discretization procedures and also specialized software can solve the distribution in each physical domain. For instance, structural FEM models can be co-simulated with an electrostatic BEM [98]. The drawback of co-simulation is the synchronization of data between models that also limits the computation of coupled derivatives.

In **coupled simulation**, all discretized fields are merged into a single, preferably using matching meshes. This requires a solver able to discretize all the physical fields at once, but leads to efficient solutions. In addition to more flexible models, the coupling expressions of internal behaviors are simplified and enable the computations of global properties, such as stiffness matrix or eigenfrequencies. Moreover, this method enriches the uncertainty quantification with more efficient stochastic analyses as demonstrated in [99].

Both co-simulation and coupled simulation environments rely on *solver configurations* classified in three categories that are the staggered, hybrid or monolithic solutions. Hannot [99] proposed the framework presented in Table 2.1. This classification intends to help designers in choosing appropriate electromechanical modeling strategies at physical level.

Staggered solutions involve solvers which are dedicated to each physical domain. These solvers solve the domain equations and communicate the updated version to the other solvers in a cyclic way. The systems to solve remain small and domain-specific, but the global solution suffers from slow convergence.

In **hybrid solutions**, the static analyses are first computed through staggered solutions. The static analyses aim at finding the boundary solutions required to bind both domains into a single expression. The dynamic analyses can then be computed on the unified system in a monolithic solution.

Monolithic solutions combine electrostatic and mechanical equations into one single system. This latter can be solved directly and enables the computation of elements needed to solve nonlinear equations, e.g., the tangent matrix. The large size of the system matrices may nevertheless lengthen the computation. Moreover, the estimation of scaling effects like the evaluation of squeeze film damping, is an important scaling element in three field problems.

In order to integrate MEMS models into architectures comprising HW and SW virtual prototypes, we investigate models that are suitable for fast simulation, especially to enable the transient analysis of the whole architecture. Based on the above comparison, MOR appears as the most adapted strategy and is achievable through monolithic solutions that enable both static and dynamic analyses in a coupled simulation environment.

Table 2.1: Simulation strategies for electromechanical system modeling and simulation [99].

		Co-simulation	*Coupled Simulation*		
Parameter	**Analysis**	**Staggered**	**Hybrid**	**Monolithic**	
Displacement	Load/Displacement	✓	✓	✓	*Static*
	Pull-in	✓	✓	✓	
	Pull-in S&S	✗	✓	✓	
Time	Transient	✓	✓	✓	*Dynamic*
	Dynamic Pull-In	✓	✓	✓	
	Dynamic Pull-In S&S	✗	✓	✓	
Frequency	(Damped) Frequency	✓	✓	✓	
	Frequency S&S	✗	✓	✓	
	MOR	✗	✗	✓	

2.2.3.1 Analyses

The pull-in voltage can be derived from load-displacement analysis which characterizes the quasi-static response of an electromechanical device. Load-displacement curves are obtained by varying the voltage, the charge or the displacement applied on the device applied on the device. As demonstrated in [4], these curves provide information on the working voltage and stability range of the device.

Dynamic modeling intends to address the transient and frequency analyses of MEMS devices. This kind of study is conditioned by the damping definition. It is assumed that no energy is lost in undamped case. If added, the damping will generate forces proportional to the velocity and thus induce friction. Taking into account this phenomenon highly depends on the purpose and design of the device. To further explore the design space, Sensitivity & Stochastic (S& S) analyses can be performed on both static and dynamic modeling. These intend to evaluate the impact on the design of the variation of parameters. Since finite differences are expensive and not robust, analytic methods aim to efficiently cope with the design space exploration, regarding a geometric description of devices. These methods are not presented in detail in this work, but the reader can refer to previous works for more detail [100].

Reduced modeling (MOR) aims to perform transient analysis on a compact view of the system. One major problem of these methods is the reduction of the nonlinear electrostatic forces. The related nonlinearity indeed depends on both the applied potential and the displacement. The dependency on the applied potential is quadratic and requires second order Taylor approximation. Nevertheless there is no optimal method to overcome the complicated dependency on displacement. MOR techniques are introduced in Section 2.3, and the selected method used in the API is described in Chapter 4.

Figure 2.6 depicts the interest of reduced models to overcome the complex description of devices in both co-simulation and coupled environments. Reduced models indeed enable the transient simulation of MEMS and are well fitted to be integrated in HDLs due to the limited number of ODEs to solve.

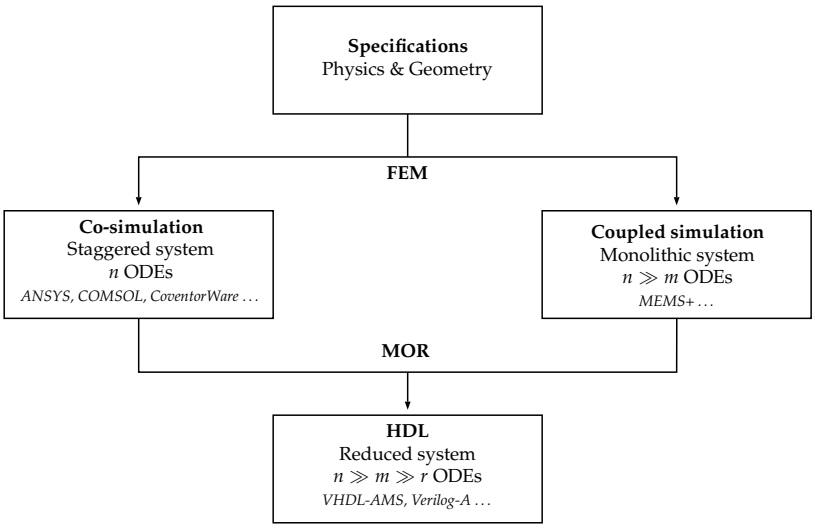


Fig. 2.6: MOR applied to MEMS simulation enables to decrease the number of DoFs of the considered model and to integrate such models in faster simulation environments such as HDLs.

2.2.3.2 Commercial software solutions

CAD software tools have been significantly improved the last two decades to design MEMS devices and structures, e.g. Finite Element (FE) solutions developed by ANSYS [101], COMSOL [102] or Coventor [103]. These tools address the nonlinear and multi-physics problems and enable the simulation of MEMS devices with complex shapes and configurations. To evaluate the performance of the system as well as to perform design refinements, the available software tools are mostly based on staggered solutions.

In the late 1990s, component-based solutions emerged from research projects like NODAS [97], SUGAR [104] or Lorenz's work [105]. These solutions are based on coupled simulation and consider MEMS as assemblies of basic elements. This approach encapsulates the behavior into components [106] and produce synthesized information of the system design. The method has been ported to CAD tools, e.g., *Architect3D* and *MEMS+* both developed by Coventor [78]. The use of pre-configured components is now considered in MEMS industry as an efficient way for designers to cope with the structural and electrostatic definition of the microsystems. In this work we refer to the component-based models created in *MEMS+* following the process detailed in Appendix B.

Upcoming challenges in MEMS modeling and simulation concern the acceleration of the simulation, the transient analysis, and the better integration within the IC design flow [107]. This latter concerns both low-level and system-level operations. At low level, the design rule check verify the layout is conformed to the proposed schematic. Besides low-level verification, ICs and systems designers must ensure the correct integration of non-electronic peripherals like MEMS through system-level modeling and simulation. To this end, we will assume MOR as the ground layer to address the system-level modeling and simulation of MEMS devices in HDLs.

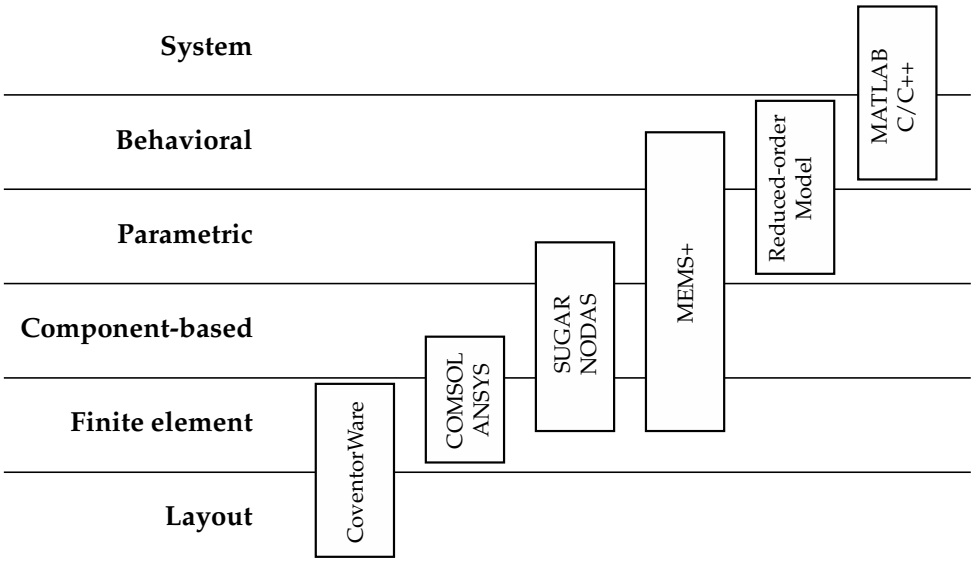


Fig. 2.7: Levels of abstraction supported by MEMS CAD tools and system simulators.

2.3 HDL-based reduced models

As the complexity of geometry and features of MEMS increases while their size is shrunk, different physical effects such as mechanical nonlinearities, electrostatic damping or thermal interaction cannot be ignored anymore. The design of micro-structures also requires new multi-physical models describing their complex internal behavior. The common approach for modeling MEMS devices relies on the coupling of electrostatic equations with mechanical equations through the variation method as previously stated.A spatial discretization of the electromechanical coupled PDEs leads to systems of very large state space dimensions that make the analysis and simulation unacceptably time consuming and expensive. Advanced nonlinear dynamic analysis is hard to be conducted on FE models. Therefore, the system-level simulation of MEMS mostly relies on DAEs and ODEs.

In this context, MOR appears as an efficient manner to approximate large-scale systems by much smaller models. These aim to capture the input-output behavior of the original system to a required accuracy and also preserve essential physical properties. With fewer DoFs, reduced models enable transient analyses to provide insight on various physical aspects and understand properly the behavior of MEMS devices. Furthermore, these models are highly interesting to design control systems and observe feedback effects.

AMS extensions of HDLs are geared for few DoF systems which includes reduced-order models. Traditional HDLs, i.e., VHDL and Verilog, enlarged their simulation capabilities to analog parts through AMS extensions. Therefore, IC models can be enlarged to peripherals, such as MEMS devices. This system-level approach also enriches the IC simulation with additional information on analog sub-parts and complements the low-detail FE modeling by accelerating the simulation. In the following, we review some of the state-of-the-art techniques in MOR and their use in traditional HDLs.

2.3.1 Model order reduction principles

We assume here that MOR is required to reduce the simulation time and enable the transient simulation of MEMS with larger systems integrating HW and SW parts. The general purpose of MOR is to approximate large-scale systems by smaller models of lower state-space dimension, but preserving their behavior. MOR finds multiple modeling applications in various domains [108] and is widely used in structural dynamics problems [109]. Most of the common methods are reported and deepened in dedicated books on the topic [70, 110].

In this section, we consider the system-level modeling and simulation of MEMS, where MOR techniques are similar to those developed for Very Large Scalable Integration (VLSI) design [111]. Mostly based on projection and moment matching techniques [74], these methods were developed for linear systems and extended by parametric methods [112]. The necessary mathematical background for matrix operations and basic MOR techniques is introduced along with numerical algorithms by Feng *et al.* in [71].

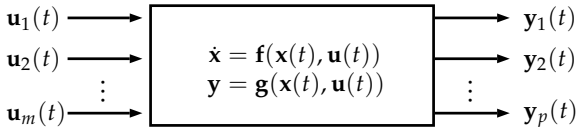


Fig. 2.8: Input-output system.

Dynamical coupled systems

We limit our study to ODEs and consider the general form of explicit finite-dimensional dynamical systems:

$$\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}(t), \mathbf{u}(t)) = 0, \tag{2.6}$$

where $\mathbf{x} : \mathbb{R} \mapsto \mathbb{R}^n$ is the state variable, $\mathbf{u} : \mathbb{R} \mapsto \mathbb{R}^m$ is the input, t is the time, and $\mathbf{f}$ is the function related to the behavior of the system. The observable outputs $\mathbf{y} : \mathbb{R} \mapsto \mathbb{R}^p$ are defined as follow:

$$\mathbf{y} = \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t)). \tag{2.7}$$

We consider the linearized system represented in Figure 2.8 that consists of n-dimensional state space vector $\mathbf{x}$:

$$\begin{aligned} \mathbf{E}\,\dot{\mathbf{x}} &= \mathbf{A}\,\mathbf{x}(t) + \mathbf{B}\,\mathbf{u}(t), \\ \mathbf{y} &= \mathbf{C}\,\mathbf{x}(t) + \mathbf{D}\,\mathbf{u}(t). \end{aligned} \tag{2.8}$$

Here, $\mathbf{A} = \partial_{\mathbf{x}} f \in \mathbb{R}^{n\times n}$ is the system matrix, $\mathbf{B} = \partial_{\mathbf{u}} f \in \mathbb{R}^{n\times m}$ is the input or control matrix, $\mathbf{C} = \partial_{\mathbf{x}} g \in \mathbb{R}^{p\times n}$ is the output matrix, $\mathbf{D} = \partial_{\mathbf{u}} g \in \mathbb{R}^{p\times m}$ is the feed-forward matrix, and $\mathbf{E} = \partial_{\dot{\mathbf{x}}} f \in \mathbb{R}^{p\times m}$ is the mass matrix.

Systems of the form (2.8) arise in electromechanical problems and are usually expressed by PDEs of second order in time. Moreover, they are influenced by damping effects that can be approximated by Rayleigh damping [113]. The term $\mathbf{D}\,\mathbf{u}(t)$ is usually not affected by the selected MOR method and generally null, except in piezoelectric structures where displacements may vary directly with the inputs [114]. For simplicity and since our study is focused on electrostatic devices, we assume $\mathbf{D} = 0$ and refer to the following system:

$$\begin{cases} \mathbf{E}\,\dot{\mathbf{x}} = \mathbf{A}\,\mathbf{x}(t) + \mathbf{B}\,\mathbf{u}(t), \\ \mathbf{y} = \mathbf{C}\,\mathbf{x}(t). \end{cases} \tag{2.9}$$

For linear dynamical systems, MOR aims to reduce the number of internal states, i.e., the size of $\mathbf{A}$, while preserving both input and output terminals.

Projection methods

In the following, we briefly review approximation methods based on projection, as depicted by Antoulas in [70]. These methods aim to truncate least important states from the original system in order to project the high-dimensional state space of the original model into a low-dimensional subspace. To this end, basis transformation is applied as in the Galerkin projection introduced hereafter.

Let $\Pi = \mathbf{V}\,\mathbf{W}^*$ the projection of the original n-dimensional state space onto a k-dimensional subspace, where $\mathbf{V}, \mathbf{W} \in \mathbb{R}^{n\times k}$ and $\mathbf{W}^*\,\mathbf{V} = \mathbf{I}_k$. The resulting reduced model is defined as follow:

$$\begin{cases} \tilde{\mathbf{E}}\,\dot{\tilde{\mathbf{x}}} = \tilde{\mathbf{A}}\,\tilde{\mathbf{x}}(t) + \tilde{\mathbf{B}}\mathbf{u}(t), \\ \quad \mathbf{y} = \tilde{\mathbf{C}}\,\tilde{\mathbf{x}}(t). \end{cases} \tag{2.10}$$

where $\tilde{\mathbf{x}} \in \mathbb{R}^k$ is the reduced state vector such as $\tilde{\mathbf{x}} = \mathbf{W}^*\mathbf{x}$, $\tilde{E} = \mathbf{E}\,\mathbf{V} \in \mathbb{R}^{k\times k}$, $\tilde{\mathbf{A}} = \mathbf{A}\,\mathbf{V} \in \mathbb{R}^{k\times k}$, $\tilde{\mathbf{B}} = \mathbf{B}\,\mathbf{V} \in \mathbb{R}^{k\times m}$ and $\tilde{\mathbf{C}} = \mathbf{C}\,\mathbf{V} \in \mathbb{R}^{p\times k}$. If $\mathbf{V} = \mathbf{W}$, i.e. the columns of $\mathbf{V}$ form an orthonormal set, then Π is orthogonal and is called a Galerkin projection. Otherwise, if $\mathbf{V} \neq \mathbf{W}$, we speak of a Petrov–Galerkin projection.

Galerkin projection has been further combined with other methods, especially Krylov subspace, balanced truncation and proper orthogonal decomposition. For a good introduction on these techniques, see Cardoso [115] and Vasylev [116].

Methods for nonlinear systems

The aforementioned methods have been extended to more complex systems, such as second-order, nonlinear or parametric systems as introduced in [71, 117]. In parametric systems, one or more of the constitutive matrices may vary with respect to a function of parameters $\mathbf{p}$. For instance, the time-dependent systems of the form 2.10 can be modeled as follow:

$$\mathbf{E}(p)\dot{\mathbf{x}} = \mathbf{A}(p)\mathbf{x} + \mathbf{B}(p)\mathbf{u}\,. \tag{2.11}$$

The model reduction mainly relies on interpolation methods, multivariate moment-matching or series expansion [118]. Parametric MOR is a efficient method for the space exploration and the evaluation of parameter influence on the structure, e.g., the variation of temperature on the frequency response [119].

Systems with many nonlinearities have too many inputs to be isolated and specific methods are thus employed. There are two main approaches for model order reduction of nonlinear systems. The first approach is based on trajectory reconstruction, i.e. interpolation between different snapshots of the systems. The related methods are proper othogonal decomposition, balancing and optimization, system identification, and Trajectory Piecewise-Linear (TPWL) [75]. The second approach consists of weakly nonlinear polynomial approximation through higher-order series expansion. To this end, bilinearization methods are commonly employed [74].

2.3.2 Implementation in HDLs

The above MOR techniques have been applied to MEMS design, especially to address the system-level integration of devices. To this end, the reduced models are commonly incorporated into HDLs, such as VHDL-AMS and Verilog-A. We discuss hereafter the evolution of traditional HDLs to ESL design solutions.

HDLs were first intended to realize the logic synthesis of digital ICs. For instance, VHDL [49] derived from the Ada programming language and defines a strong-typed language. This makes VHDL more verbose than Verilog, but intends to build self-documenting designs [120]. Moreover, additional coding is required to explicitly convert data type to another, e.g., integer to bit-vector. VHDL relies on a clear semantics to avoid unambiguous design. Alternatively, Verilog [50] introduced a weakly and limited typed language. Adapted from a former HDL, called Hilo, Verilog was mostly influenced by the C programming language. These languages progressively evolve to wider integrated solutions, especially to address ESL design [53].

Both languages also support the simulation of analog components in ICs as complemented by AMS extensions which are VHDL-AMS and Verilog-A, respectively. These extensions enable the continuous systems simulation through specific solving methods implemented along with the initial event-driven simulator [51]. More recently, Verilog has been extended by SystemVerilog [121], a hardware design and verification language already widely used by the digital community. But, despite its name, it has not been proven to be useful for system-level specification, design, or verification yet. Therefore, we did not consider SystemVerilog as an adapted environment for the simulation at system-level of complex systems. In the following, we review first applications in VHDL-AMS, Verilog-A and MATLAB of reduced models in order to simulate MEMS devices with surrounding electronics (Table 2.2).

Chen *et al.* [122] clearly stated the interest of MOR in MEMS design and depicted its advantage over lumped-element or traditional FEM models. Based on the method developed in [123], this study combines the Taylor series expansion with the Arnoldi method to automatically define reduced-order models for coupled energy domain nonlinear MEMS. The presented technique is tested on quadratic and third-order nonlinear models of fixed-fixed clamped beam. The results demonstrate that third-order models are far more accurate than linearized models, e.g., the steady-state error is thirty times lower, and is computationally more efficient compared to full-meshed solutions with a run time about five times faster. The proposed method is also discussed against the Karhnen-Loeve decomposition which can retrieve the behavior of weakly nonlinear device even in pull-in, see [124] for more details. Nevertheless, the authors highlight the difficulty to simulate such models directly in VHDL-AMS since the parameters of the corresponding Taylor expansion must be updated at each time step of the simulation.

Mähne *et al.* [125] used reduced models created in FEA tool to simulate the behavior of a yaw-rate sensor in VHDL-AMS. The selected MOR technique applies the modal superposition introduced in [126] and implemented in ANSYS. In order to integrate these models in VHDL-AMS, each mode of interest is treated and reduced separately from the original FE model in ANSYS. Therefore, two different reduced models are generated to handle the out-of-plane and in-plane rotations, respectively. The final implementation in VHDL-AMS couples the entries and outputs of each models in order to correctly reconstruct the nonlinearity of the device, especially the Coriolis force. The results confirm previous refined studies with an increased speed of computation. Such models are finally considered to efficiently evaluate different circuit concepts, e.g., driving, sensing and control circuits, or to reuse the device into new test benches.

Köhler *et al.* [127] extended a moment-matching MOR technique by automatically selecting the number of expansion points. In addition, a parametric version of the selected method is proposed in order to ease the design and space exploration of the device. Tested on the aforementioned yaw-rate sensor, this method presents competitive results in terms of computation and accuracy. However this remains inefficient in high-dimensional problems, even if the influence of few parameters can be well estimated.

Schlegel *et al.* [128] identified MOR as a promising support for system-level simulation of MEMS devices regarding the inaccuracy of equivalent lumped-element circuit models. Based on modal superposition [129], the selected method aims at identifying the most important modes to approximate the deformation state of the FE model by a series of weighted modal functions, i.e., the eigenmodes. Thus, the response of the system is approximated by the interpolation of the shape functions describing each operating mode. Based on ANSYS MOR feature, this study highlights the limitations of VHDL-AMS to deal with recursive algorithms that are needed to retrieve the nonlinearity of the system. Moreover, the interface of the models

may vary with the abstraction level. To avoid any incompatibility between low- and high-level models, the Multi-Modeling Architecture framework [130] is proposed. This top-down methodology defines the terminals and related quantities independently from the selected abstraction level. This allows for a test bench to be first defined with abstract models and further refined with reduced-order models without any modification to the interface. The proposed combination of EDA-tool with FEM simulator is validated on a sensor array case study with a low error on reduced models ($< 1\%$).

Verilog-A is the most popular alternative to VHDL-AMS. For instance, Hagleitner *et al.* [131] exemplified the use of Verilog-A to co-design analog front-end circuit and MEMS device. The study is centered on a scanning-probe storage device whose mechanical part is modeled as a micromechanical cantilever. The corresponding MOR method is based on Krylov subspace and instantiates a second-order system. Verilog-A is considered as well fitted for low-level, parameterized analysis since it can directly solve DAEs and ODEs systems. Moreover, the model takes into account the damping and noise effects and thus enables the coupling of electrical, mechanical and thermal domains.

Mehner *et al.* [132] further employed Verilog-A models to measure the impact of packaging on the microstructure. A parametric model of cantilever beam is proposed in order to characterize the thermomechanical coupling responsible of stress gradient in the MEMS structure. Additional shape functions are defined to correctly retrieve the mechanical nonlinearity and create multiple system snapshots.

Despite its lack of support for low-level circuit modeling, MATLAB/Simulink has been extensively used to simulate MEMS reduced models. Niessner *et al.* [133] proposed a framework to generate macromodels from FEA, integrate the solution in MATLAB/Simulink and adapt it into HDL. Following a first application on microphones [134], this study intended to automate the model generation and enable a tighter integration of system-level simulation environment with HDL.

Finally, Parent *et al.* proposed to automatically generate reduced-order models from FEA in *MEMS+* and export them either to Verilog-A [79] or Matlab [135]. The related MOR method is based on modal superposition and accelerates the simulation more than thirty times compared to full model. The results are detailed in Section 4.3 to support our work on an equivalent export to SystemC-AMS through a dedicated API.

Table 2.2: MOR applied to MEMS design and simulation in HDLs.

Reference	**Device**	**MOR technique**	**HDL**
Chen *et al.* [122]	Beam microstructure	Taylor series expansion & Arnoldi method	VHDL-AMS
Mähne *et al.* [125]	Yaw-rate sensor	Modal superposition	VHDL-AMS
Köhler *et al.* [127]	Yaw-rate sensor	Adaptive Moment Matching	VHDL-AMS
Schlegel *et al.* [128]	Vibration sensor	Modal superposition	VHDL-AMS
Hagleitner *et al.* [131]	Probe-storage device	Arnoldi method	Verilog-A
Mehner *et al.* [132]	Accelerometer	Modal superposition & Shape functions	Verilog-A
Niessner *et al.* [133]	RF switch	Galerkin projection	MATLAB
Bedyk *et al.* [134]	Microphone	Galerkin projection & Modal superposition	MATLAB
Parent *et al.* [79]	Gyroscope	Modal superposition	Verilog-A
Parent *et al.* [135]	Gyroscope	Modal superposition	MATLAB

Traditional HDLs demonstrate that low-level descriptions are not suitable for the modeling and simulation of large systems regarding their evolution to ESL methods. System-level solutions are therefore envisioned as a promising alternative, e.g., MATLAB or SystemC. In the following section, we also explore the capabilities of SystemC to address both the co-development of HW/SW specific applications and the simulation of analog and non-electronic components through SystemC-AMS.

2.4 SystemC, a system-level design language

Traditional HDLs successfully tackle the logic synthesis of ICs. Nevertheless, these solutions mostly remain decoupled from the development of software running on the targeted architectures. To better address the concurrent development of application-specific HW/SW, higher-level modeling and simulation languages, mostly based on C/C++, emerged over the last decade, with regard to ESL principles [136].

Originally introduced in the late 1990s [137] and supported by the Open SystemC Initiative, now part of the Accelera System Initiative consortium [2] [138]. SystemC was first intended to accelerate the simulation of digital systems. In practice, models were first created at high level in C/C++ and further adapted to RTL in Verilog or VHDL for the logic synthesis. Hence the designers referred to separate HDLs unable to cope the functional or behavioral description of the system initially defined [139, 140]. In order to smooth this refinement process, SystemC introduced a C++ class library to support various levels of abstraction [141] from RTL to transaction-based design (Figure 2.9). This evolved during the last decade to its current standard implementation [56].

SystemC is now considered as a reference system-level language for the co-development of digital HW/SW. Like Verilog and VHDL, SystemC supports hierarchical models, i.e., blocks containing input/output ports, internal signals and instances of other blocks. These blocks implement concurrently running imperative processes that are scheduled and executed by an event-driven simulation kernel. From a structural view point, a SystemC design consists in modules interconnected by channels. This structure clearly separates the computation units from the communication channels that also support Transaction Level Modeling (TLM) refinement. The event-driven simulation kernel thus interprets SystemC design as a set of synchronized, concurrent processes that are coordinated by events and transmitted through channels.

Although its standard implementation relies on a discrete-event MoC, SystemC architecture allows adding more MoCs [142]. Therefore multiple extensions were proposed to cover a wider range of systems. For instance, TLM is fully supported by the standard implementation and enables the abstraction of communications occurring in hardware architectures. Similarly, AMS extensions tackle the modeling and simulation of analog components in ICs [143].

Furthermore, SystemC intends to enable the verification of system-level models, since there is no structured nor unified corresponding methodology available for ESL design. For instance, the UVM in SystemVerilog is primarily targeting block/IP applications through RTL instead of system-level verification. Porting UVM to SystemC/C++ would enable an early functional and architectural verification [144]. This indeed allows the creation of more advanced system-level test benches and the reuse of verification components between system-level and block-level verification.

[2] Corporate Accelera members are AMD, ARM, Cadence, Ericsson, Intel, MentorGraphics, NXP, Qualcomm, ST Microelectronics, and Synopsys.

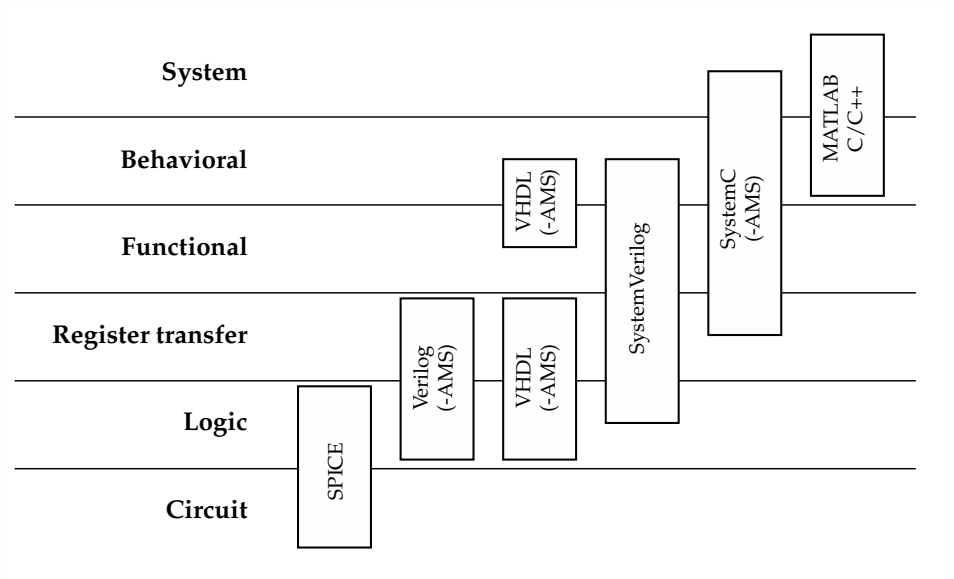


Fig. 2.9: Levels of abstraction supported by HDLs.

In this section, we first review the core principles of the SystemC structure and discrete-event simulation kernel. Then, we introduce the AMS extensions that are envisioned for continuous systems simulation. Finally, we review further extensions to the standard in order to build heterogeneous virtual prototypes. For an in-depth introduction to SystemC and its AMS extensions, the reader can refer to [57, 145].

2.4.1 Basic concepts

The SystemC language is a C++ subset and the standard is limited to the ground principles of hardware simulation. The core language, i.e., the semantics of SystemC, describes both the design structure and behavior through dedicated macros, classes and methods. The structure consists in modules, channels, ports and interfaces and the behavior is handled by events and processes. Additionally, SystemC provides hardware-specific data-types (bits, logic vectors, fixed point number ...). Moreover, some predefined channels, such as first-in first-out mechanism, enable the functional and signal modeling. Benefiting from C++ inheritance, the SystemC architecture enables the creation of additional design libraries or models. These can be complementary to more specific methodologies or MoCs not provided by the standard. We introduce hereafter some of the structural and behavioral aspects of SystemC [56].

Model structure

Metropolis [146], Ptolemy [37] and Ptolemy II [36] defined a multi-MoC framework which became a reference in the design of heterogeneous systems also targeted by SystemC. SystemC hierarchy relies on basic units called modules, similarly to actor-based MoC in Ptolemy [147]. This notion is well-adapted to object-oriented programming and allows for SystemC base classes to be overspecialized through inheritance. The basic structure of models rely on modules and the communication is coordinated through channels, ports, and interfaces, as shown in Figure 2.10.

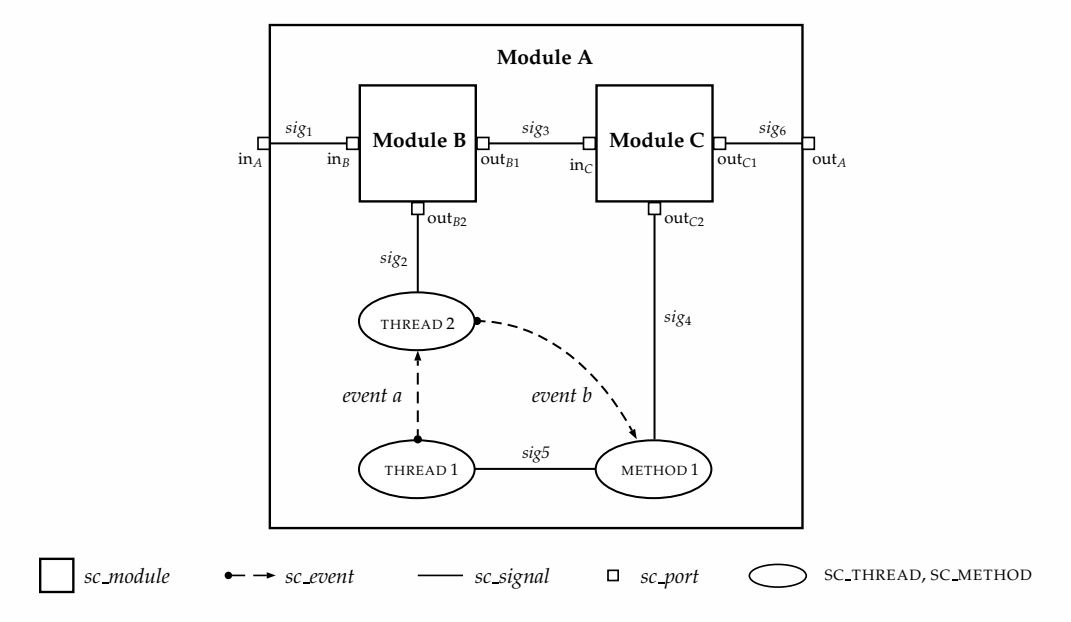


Fig. 2.10: Base classes in SystemC [57].

Modules are the fundamental building blocks in SystemC. The functional or behavioral description of a system is encapsulated in a module through internal processes. The corresponding state can be stored in local variables or computed by subordinated hierarchy. Furthermore, a module defines external connections, i.e. ports, to communicate with other instances through channels. A module can be instantiated either by the base class `sc_core::sc_module` or the macro `SC_MODULE`.

Modules are interconnected through interfaces, ports and channels. The communication principle consists in bounding each port (`sc_port`), i.e., input or output of a module, to an interface (`sc_interface`). Ports and interfaces also define which communication feature a module may use, whereas channels (`sc_channel`) implement these features. Therefore, a SystemC model is similar to a network of concurrent processes communicating over channels and synchronized by events. Moreover, SystemC is based on TLM and must therefore guarantee the efficient control and execution of concurrent transactions. To that end, it supports a set of primitive channels for the concurrency control, e.g., `sc_signal` or `sc_fifo`. The flexibility of the SystemC standard enables a modular communication framework illustrated in Figure 2.11.

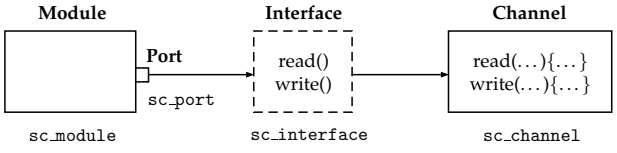


Fig. 2.11: SystemC communication model [148].

Model behavior

Events, sensitivity and notification are the key concepts of the SystemC execution which is based on event-driven simulation. The simulation kernel invokes processes when sensitive to an event and the event oc-

curs. These processes execute the behavior of the module and are instantiated either through methods or threads. A method process, `SC_METHOD`, executes its content in its entirety, i.e., without any interruption or step in its execution. Alternatively, a thread process, `SC_THREAD`, is a process which can be suspended at any time during its execution. Note that a thread is only started once at the beginning of the simulation, in contrast to method which can be invoked arbitrary often.

The sensitivity of these processes is either static or dynamic. Static sensitivity of both `SC_METHOD` and `SC_THREAD` is implemented at elaboration, i.e., within the module constructor. In contrast, the dynamic sensitivity enables a process to change its own sensitivity during the execution. Since a method cannot be interrupted, the change is specified by `sc_notify` and will be effective on the next simulation cycle. This mechanism is set by the `next_trigger` command. Similarly, the dynamic sensitivity of thread enables interruptions by calling the `sc_wait` function. The execution can be resumed on demand where the interruption occurs and refers to the internal state that has been stored by the process.

We reviewed above the modeling strategy in SystemC, further depicted in [57, 148]. To summarize, the model structure relies on base classes and templates which define the connectivity and communication of the system. The system behavior is then defined internally to modules through functions and procedures orchestrated by the simulation kernel whose the constituting phases are exposed in Figure 2.12.

Simulation kernel

The SystemC simulator executes a sequence of operations which mainly consists in two phases which are elaboration and execution [145, 149]. First, the elaboration consists in executing all statements prior to `sc_start()` function and contained in the `sc_main` method. This phase aims to establish the system hierarchy and connectivity with respect to the primitives of modules, channels and processes composing the model. In addition, all internal data structures are initialized. Once the model is built, the execution phase is operated by the SystemC simulator. This controls the simulation time, the execution of processes, the notifications of events and the update of primitive channels. In case of immediate notifications, the corresponding processes become ready to run and are directly executed. Like most HDLs, SystemC implements the delta-cycle notion to handle concurrency and establish an arbitrary order of simultaneous actions. Thus, the concurrent processes are first evaluated by executing internal methods and the corresponding changes are temporary stored. If there are no delta-delay notifications, the changes are propagated over channels by the update process. The simulation is advanced if no additional processes need to be evaluated or if there are no timed notifications. When no more simulation processes need to be run, the simulation ends. At the end of the execution, additional post-processing or cleanup operations can be performed. The aforementioned simulation kernel is decoupled from the modeling framework in SystemC which enables to extend the standard through specific MoCs and a broader range of applications.

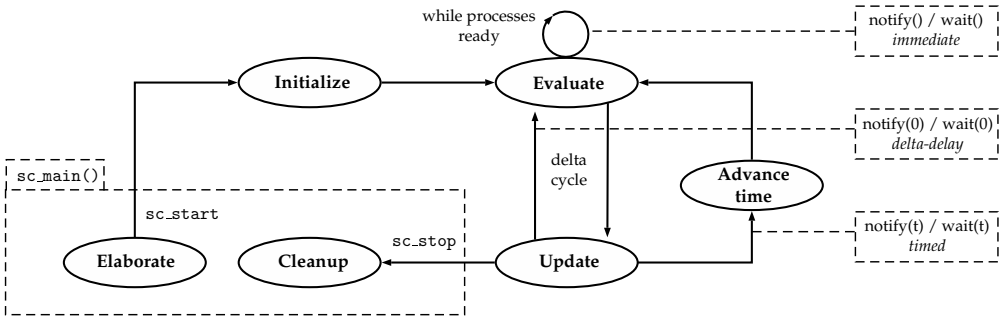


Fig. 2.12: SystemC simulation phases.

2.4.2 AMS extensions

SystemC supports a wide range of MoCs in the digital domain and is very well suited to design HW/SW systems [142]. However, in many applications, the digital HW and SW parts are coupled to analog components which interact with the continuous-time environment. To address the complexity and influence of these analog sub-parts, different low-level methods were proposed in traditional HDLs, e.g., VHDL-AMS or Verilog-A. Alternatively, the SystemC AMS extensions adopt a higher abstraction level and a more flexible modeling methodology to broaden the SystemC capabilities. These AMS extensions is compatible with existing SystemC (IEEE 1666–2005 specifications) and have been standardized in [24].

The AMS extensions enable to build system-level executable specifications to model and analyze analog components [150]. These models can further be refined to create virtual prototypes and proceed to space exploration or integration validation. SystemC AMS proposes a synchronization method between the discrete event and continuous-time solvers respectively in charge of digital and analog simulation. This mechanism enables the definition of additional MoCs [33], each dedicated to a specific physical domain. SystemC AMS first intended to provide a generic framework for the system-level simulation of complex systems. The standard interfaces indeed allows to define additional simulation methods as well as the integration with commercial simulators. The standard implementation supports the MoCs shown in Figure 2.13, i.e., Timed Data Flow (TDF), Linear Signal Flow (LSF) and Electrical Linear Network (ELN) MoCs. SystemC AMS still has some limitations to recover continuous-time behaviors, especially the nonlinear ones [151]. Nonetheless, we assume that SystemC AMS is one the most advanced solution to support the system-level simulation of digital software and hardware applications with additional analog peripherals, such as MEMS devices.

The AMS language standard [24] defines the execution semantics of the supported MoCs, i.e., TDF, LSF and ELN MoCs. Moreover, this provides the necessary elements to build the underlying enabling technology such as the synchronization layer, scheduler and linear solver. The interfaces and the class definitions are only defined in dedicated implementation. We refer in the following to SystemC-AMS, the implementation proposed by the Fraunhofer Institute [143].

Figure 2.13 summarizes the SystemC-AMS architecture, based on SystemC and extended to mixed-signal systems. Built upon SystemC, the AMS infrastructure interacts with the existing digital environment through the synchronisation layer. This allows mixed-signal designs in which analog components directly interact with pure digital units. The digital modules remain handled by the SystemC simulation kernel that activates the related processes to communicate and compute in a timely manner. As depicted above, SystemC efficiently supports discrete processes and covers a broad range of abstraction levels from gate level to software systems (see Figure 2.9). In addition, SystemC-AMS refers to the TLM framework to supervise the data exchange between the different processing elements as in SystemC. The user is thus free to focus on the behavioral definition of the processing methods inside modules, instead of defining how the data must be read or written between the different processing elements. Furthermore, SystemC-AMS tends to suppress low-level or component-specific details from models. The resulting description is centered on main parameters and aims to better understand the system behavior. The related lightweight models are also useful to explore the design space. While traditional design tools like SPICE give an accurate estimation of the circuit performance, they are inappropriate in early-design phase with regard to the cumbersome and rigid definition of models. Furthermore, SystemC-AMS benefits from the C++ language to define domain-

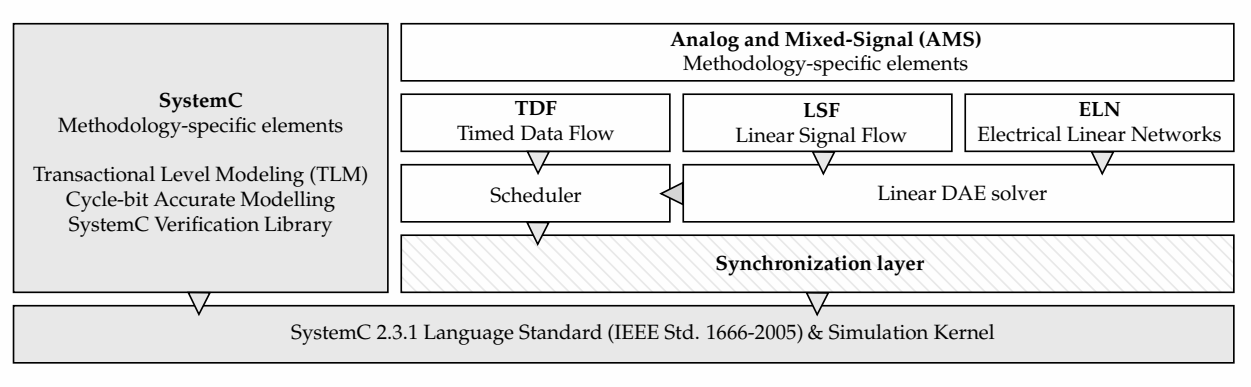


Fig. 2.13: Layered architecture of the SystemC standard with AMS extensions [24].

specific methods or extend the standard. Once the designer understands the behavior of new complex system, he may revert to tools like SPICE to estimate and optimize the performance characteristics at finer-abstraction levels.

SystemC-AMS exploits both discrete-time static non-linear (non-conservative behavior) and continuous-time dynamic linear (conservative and non-conservative behavior) model abstractions to fully support continuous-system modeling. In the one hand, non-conservative systems are modeled through unidirectional signals and mainly structured as block diagrams. For instance, this formalism is supported by TDF and LSF MoCs. In the other hand, conservative models contain bi-directional signals that carry two complementary quantities, i.e., the across and through values. These quantities are essential to compute the system energy and estimate its internal distribution. For example in ELN MoC, an electrical network is described through Kirchhoff laws which associate the voltage, i.e. the across value, with the current, i.e., the through value. From a global viewpoint, designers may define analog modules with interfaces connected to non-conservative signals while internally the model is refined through conservative signals. Thus the system appears from the outside as a fully directed model while preserving internal couplings.

Similarly to SystemC, the models created in SystemC-AMS instantiate the predefined modules, ports, terminals and signals [152]. The analog base class `sca_module` is derived from the standard SystemC `sc_module` class and provides common functionality to analog modules. The modules are interconnected through ports (`sca_port`) and signals (`sca_signal`) and form clusters, i.e., sets of instantiated AMS modules of the same MoC. To support each MoC formalism, the simulation process is declared in the underlying solver. Figure 2.14 summarizes the different supported MoCs by the standard SystemC-AMS and classified with respect to the definition of time and nature of signals. We review hereafter the main properties and specificity of the TDF, LSF and ELN MoC.

2.4.2.1 Timed Data Flow

Adapted from Synchronous Data Flow (SDF) [153], TDF is the most versatile formalism available in SystemC-AMS. This MoC allows the designer to set up specific analysis within custom instances of `sca_tdf::sca_module`. The internal behavior is defined in the member function `processing` and can be refined through detailed simulation steps. In TDF, the designer is not restricted to the predefined TDF modules and can add its own modules while specifying the related properties. The data flow is constrained by three

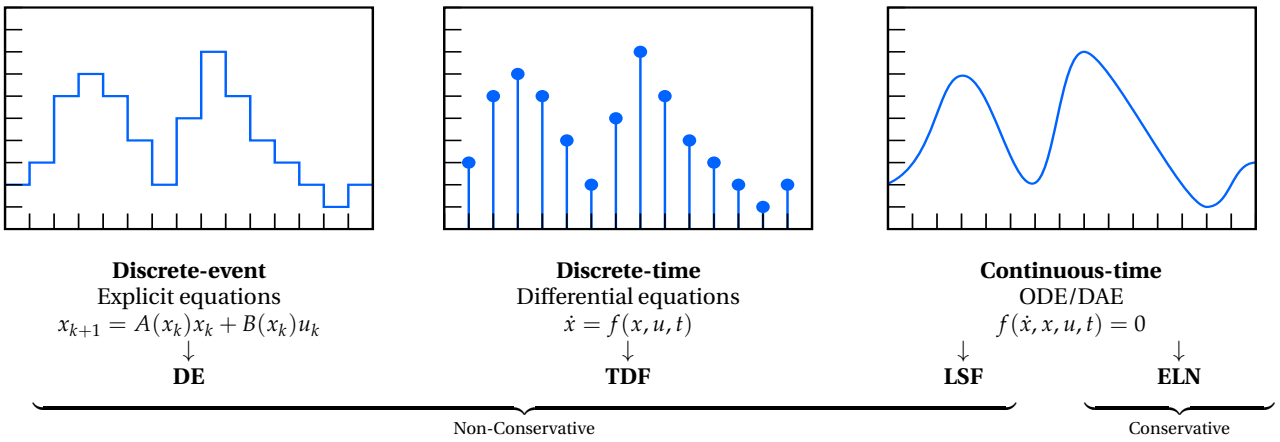


Fig. 2.14: Signal processing and equations supported by the models of computation in SystemC-AMS.

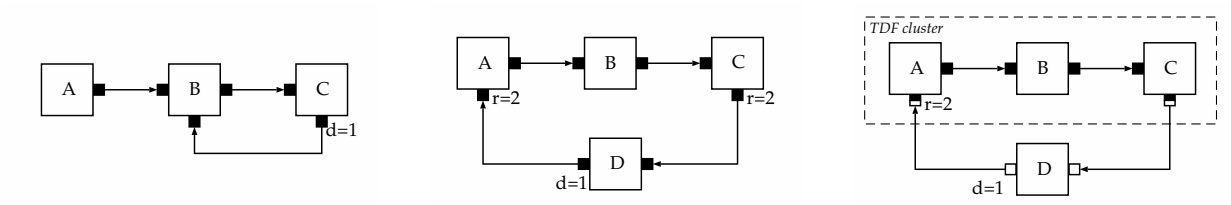
strict principles. First, the type of attributes assigned to ports and modules must be compatible, e.g., a unsigned-integer value cannot be written to a port supporting boolean. Second, the rate and time-step definition must ensure a coherent sampling time in the sending and receiving ports. Third, in case of feedback loops, at least one port of the loop must implement a delay to enable the overall synchronization of processes. Besides a flexible modeling formalism, the data are represented through signals sampled in time. In contrast to the tagged discrete events in SystemC or continous signals in LSF and ELN, this description allows to indifferently carry discrete or continuous information like signal amplitude despite being tagged discrete in time as shown in Figure 2.14.

A TDF model is a directed graph in which TDF modules are interconnected through signals and form a so-called TDF cluster. The mathematical functions implemented in each module are executed with respect to inputs and internal states. Note that the functions are only computed if the required number of input data is available. The results are then written to the output ports and tagged with time information regarding the selected time step. Additionally, the rates imposed on input and output ports allows the number of produced values to be different from the one of input values. The topology of TDF modules is illustrated in Figure 2.15. In Figure 2.15(a), a simple TDF loop is defined with a delay, denoted d, on the output port in order to correctly initialize and perform the simulation. A feedback loop is depicted in Figure 2.15(b) with the corresponding delay definition. Moreover, the rate on the input and output ports, denoted r, defines a multirate signal, i.e., the concatenation of multiple values on a single instance. To connect two different MoCs, converter ports are used, as represented by half filled-in boxes in Figure 2.15(c). The filled-in white are SystemC-specific ports on discrete-event modules. SystemC-AMS modules can also directly interact with pure SystemC modules allowing for mixed-signal analysis.

The elaboration and simulation phases in TDF MoC are detailed in Table 2.3. At elaboration, a time step is assigned to each module and port regarding the user-defined configuration on specific modules. In addition, the optional method `set_attributes` initialize the input/output delay and rate value. The method `initialize` is then activated to set the class specific data structures to user-defined or default values. During the simulation, the class-specific computation is performed through the `processing` method. The post-processing operations are finally executed with `end_of_simulation`.

Table 2.3: Elaboration and simulation in Timed Data Flow (TDF) MoC

Step	Action	Phase
Elaboration	1	Set TDF module attributes by executing `set_attributes` for each module in TDF cluster.
	2	Define the TDF time step to propagate through all modules on cluster and check its consistency.
	3	Define the cluster schedule and check its computation.
	4	Initialize all TDF modules in cluster by executing `initialize` method.
Simulation	5	Activate all TDF modules and start the simulation by executing recursively the `processing` method of each module
	6	Execute the TDF post-processing methods and finish the simulation through all `end_of_simulation` member functions



(a) Simple TDF model with internal loop and delay set on the related port.

(b) Feedback loop with delay and multirate signal.

(c) Connection to SystemC module through converter ports.

Fig. 2.15: TDF modeling topology.

Similarly to SDF, TDF formalism elaborates a static schedule prior to the simulation and executed regarding a fixed time step. This method leads to high-speed simulation performance and is well fitted for usual sampling methods such as communication protocols.

2.4.2.2 Dynamic Timed Data Flow

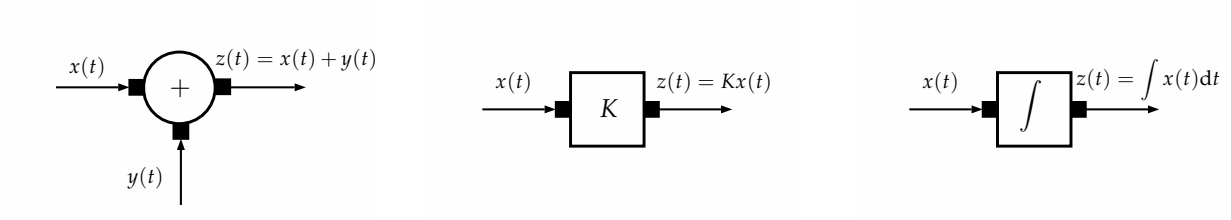
The fixed definition of the time step in TDF cannot address more reactive systems, like control units. Therefore, Dynamic Timed Data Flow (DTDF) aims to support dynamical change of the time step during the simulation as introduced in [154]. This feature enables the dynamic control of the time step through the `change_attributes()` function within the module that instantiates the `does_attributes_change()` method. The `request_next_activation()` method enables the related module to change the current time step to an updated value. The new value is accordingly propagated to the other modules of the TDF cluster which must implement the `accept_attributes_changes()` function. This enables to advance the simulation with the new time-step value. DTDF enriched the language semantics of the existing TDF MoC. This improves the time-accurate synchronization of AMS signal processing and control systems.

Despite the dynamical time-stepping in TDF, SystemC-AMS lacks of support for an automatic definition of the time step with regard to the dynamics of the system like in the state-of-the-art simulation tool MATLAB/Simulink. In SystemC-AMS, the user is responsible to define the time step which may lead to oversampling or inaccurate choices that directly impact the simulation quality and performance.

The above definition of dynamical time step has been further extended to LSF and ELN MoCs. An experimental version of the variable time-stepping method was proposed by Reuther and Einwich [155]. As introduced below, the simulation in LSF and ELN relies on a linear DAE solver and numerical integration schemes, like backward Euler or trapezoidal methods[3]. To avoid a refactorization of the system matrices, this solution implements the Woodbury formula to solve the DAE system. The system is also solved through the inverse of the system matrix instead of factorized coefficients which decreases the amount of operations to perform at each time step. Furthermore, this method supports wider time steps and thus speeds the simulation up while preserving the correctness and stability of the solution [157]. Nevertheless, its application remains limited to small linear systems and has not been successfully applied to larger or nonlinear systems.

2.4.2.3 Linear Signal Flow

The LSF MoC enables the modeling and simulation of non-conservative systems with continuous time. A LSF model consists in a real-valued signal flow which instantiates a linear DAE with respect to the relations between variables. The graphical representation of an LSF model is a block diagram, i.e., a set of blocks (LSF modules) interconnected by arrows (LSF signals). The related LSF cluster can directly interact with other MoCs, e.g., TDF or DE MoC, through inputs and outputs defined as converter ports. Note that the converter ports to SystemC DE MoC encapsulate a conversion to TDF and thus do not directly interact with the DE MoC. In contrast to TDF, the user cannot implement customized functions in LSF modules, but rather use a set of predefined LSF primitives (sum, multiplication, derivative, ...) as shown in Figure 2.16 and provided in the standard [152]. When creating an LSF model, the mathematical definition of each module and its interconnection are used to compose the overall equation system.



(a) Addition block. (b) Weighted multiplication block. (c) Time integration block.

Fig. 2.16: LSF modeling topology.

2.4.2.4 Electrical Linear Network

The ELN MoC introduces electrical primitives to model and analyze continuous-time, conservative electrical circuits. An ELN model consists in electrical primitives connected to nodes to form an electrical network. This network is interpreted through Kirchhoff's current and voltage laws in order to define an equivalent DAE system. ELN models are computed during the simulation by the linear DAE solver applying the modified nodal analysis [158]. Note that this method is also employed by low-level simulation tools like SPICE. Nevertheless, the aim of SystemC-AMS is to provide a set of method for high-level, rapid pro-

[3] For instance, the implicit or *backward* Euler method solves the equation $\dot{\mathbf{x}} = \mathbf{A}\,\mathbf{x}$ by involving both the current state $\mathbf{x}_n$ of the system and the later one $\mathbf{x}_{n-1}$ such as $\mathbf{x}_{n+1} = \Delta t(\mathbf{A} - 1)\mathbf{x}_n$ where Δt is the current time step.[156]

totyping. Therefore, the ELN MoC intends to get similar results of SPICE models, but with less details, i.e., approximating nonlinear electronic components by ideal ones. To this end, like LSF, ELN MoC is restricted to a set of predefined primitives connected via terminals to define an ELN cluster. Among others, the proposed primitives consist of independent sources (voltage, current), lumped elements (capacitor, resitor, ...), ideal amplifiers or switches.

2.4.3 Other extensions

Besides the standard AMS extensions, various attempts have been proposed to extend SystemC and address the system-level modeling and simulation of heterogeneous systems. Among the most recent contributions, SystemC MDVP [66] provides an extensible definition of interfaces in SystemC. This implementation supports a better definition of the interconnection between different MoCs Following her work on synchronization [159], Andrade [160] proposed a first implementation of the TDF MoC in this framework. The related proof of concept also appears as a promising basis for MDVP in SystemC.

SC-DEVS [161] implements the Discrete EVent System specification (DEVS) methodology to support continuous system simulation in SystemC. DEVS formalism relies on atomic definition of system components, separating the inputs from outputs and focusing on the computation of internal state changes [162]. This hierarchical, modular approach enables the simulation of concurrently running, time-discrete models. The proposed extension also implements different modes to process the transition functions and preserves the overall performance of SystemC, although additional steps in the elaboration and simulation phases.

SystemC-A [163] addresses the simulation of analog and mixed-signal components through an extended version of SystemC. SystemC-A provides an additional MoCs with DAE solver to model applications with distributed effects. In addition, Zhao [164] developed a PDE solver using spatial discretization of PDEs and finite difference analysis. Based on a coherent and robust definition, this framework has been applied on various mixed-signal systems [165].

The SystemC AMS extensions eXperiments (SCAX) library [166] implements, among others, a MoC based on the bond graph methodology. This energy-based method represents systems as bi-directional, conservative models and can be extended to different physical domains through analogies. SCAX implements a DAE solver and proposed a dynamical time-stepping algorithm to automatically adapt the solving method during the simulation. Despite good performance and first applications on MEMS [167], the current implementation of the solver does not support algebraic loops which limits the modeling of complex systems.

HetSC [168] intends to address the specification of complex architectures by defining consistent semantics and syntax. This methodology provides rules and guidelines to add new MoCs and allows heterogeneous modeling in SystemC through abstract MoCs. In addition, HetSC supports SystemC-AMS [169] and defines the synchronization mechanisms with SystemC-AMS MoCs. These are realized either through the DE MoC in SystemC or through dedicated border channels directly instantiated in HetSC. This work proves that, despite its initial purpose [142], SystemC still lacks of a standard definition of interfaces with new MoC.

Among the formalisms introduced above, the TDF MoC offers the most flexible and extensible modeling and simulation environment supported in SystemC-AMS. First, TDF allows the user to refine the predefined modules or to instantiate its own modules with custom internal, behavioral or functional definitions. Second, TDF directly interfaces the synchronization layer with the DE MoC and encloses a scheduler used by both LSF and ELN MoCs. Third, the converter ports between LSF or ELN and any other MoC encloses a TDF conversion to ensure the correct communication between each other. Finally, the TDF MoC provides additional features for dynamical time-step definition that are already supported by the standard. This argues in the further use of TDF MoC in our API to implement reduced models in SystemC-AMS in order to connect analog and mixed-signal IC components.

2.4.4 Signal conditioning

Before being exploited by digital HW components like micro-controllers or processors, the raw signals received from a sensor must be processed. To this end, MEMS devices are usually connected to signal conditioning circuits that mainly consist of amplifier and filters. In this section, we consider the principle of architectures for signal processing that are already supported in SystemC-AMS. A more detailed discussion of such circuits can be found in texts on electronic circuits and system control [170, 171].

Figure 2.17 shows the analog control of a differential measurement unit, similar to the capacitive configuration combs commonly applied in accelerometer, as depicted in Figure 2.4(b). Its principle is to control the position of the movable electrode through output voltages. The conditioning circuit first converts the capacitive variation of the sensing cell into a voltage. Since capacitive sensing is equivalent to variable capacitance, the charge variation is usually performed by operational amplifiers which convert the related input current into an output voltage. For instance, the refined modeling of operational amplifiers in SystemC-AMS has been proposed in [172].

Amplifiers and electronic circuits generate a thermal noise due to the heat turbulence and to material properties of the circuit. The material especially produce noise at low frequency and whose the spectral power density decreases in $1/f$. In order to reduce the influence of such parasitic signals, low-pass filters are employed in the feedback loop and aim to partially suppress the signal over a cut-off frequency. Besides a low-pass filter, the feedback loop consists in a gain module which is defined to ensures the stability of the system. The resulting signal aims finally to modulate the alternative sources in opposition phase, H_1 and H_2. The electrostatic forces generated by the feedback loop follow the Equation (2.4). Since the forces definition is quadratic regarding the charge, only very low voltage variations are needed to be applied around DC operating point in order to finely control the electrode. In this configuration, the feedback control can be approached by a linear approximation.

The previous electronic control unit can be implemented in SystemC-AMS through the various MoCs presented above. A good illustration of such signal conditioning in analog systems is provided in [173]. The system-level model of an inertial navigation system is indeed proposed in SystemC-AMS and tested against physical implementation. The system consists in an accelerometer and a yaw-rate sensor connected to an analog-to-digital conversion unit. The signal conditioning unit contains a control feedback loop, similar to the above implementation. The output signal is digitally processed to reconstruct a trajectory and can further be exploited by dedicated software applications. Based on different MoCs, the related model represents a good introduction to heterogeneous system modeling in SystemC-AMS.

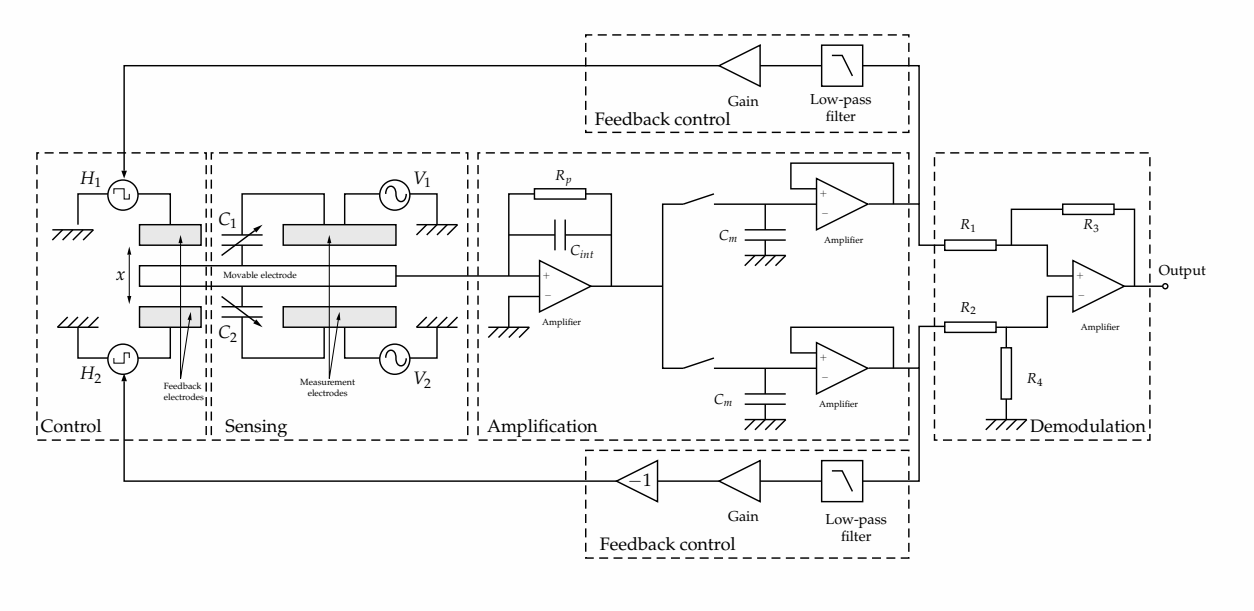


Fig. 2.17: Principle of an electronic unit with a control feedback loop on a movable electrode. This symbolic view of analog mixed-signal system is supported by SystemC-AMS through the different MoCs. In this case, the MEMS model remains limited to an approximated definition of the differential capacitive sensing.

Digital architectures with MEMS devices commonly integrate the ΣΔ conversion principle. In this case, the analog-to-digital converter is a synchronized comparator which casts the digital output signal into a specific number of digits. This kind of implementations enables to adapt the sampling frequency or add auto-zero features to increase the signal accuracy. A SystemC-AMS model of ΣΔ control loop is given in [174] which describes the further use of SystemC to exploit the converted digital signal.

The AMS extensions supported by SystemC-AMS enable the high-level modeling of applications involving analog, mixed-signal and digital subsystems. The aforementioned examples provide a first illustration of the signal conditioning related to MEMS devices and already supported by the different standard MoCs. The system-level modeling of MEMS described in Chapter 3 remains limited to approximated representations that could largely differ from actual devices, especially by avoiding some geometry details or electromechanical coupling arising in MEMS. In order to both improve the system-level modeling of MEMS and their integrate with HW/SW applications, we propose in Chapter 4 a solution based on the reduced models exported from *MEMS+* and compatible with the SystemC-AMS standard implementation.

2.5 Summary

MEMS are characterized by the tight coupling between the electrical and mechanical domains that occurs at micro scale. The related physical principles yield to large-scale systems of PDEs, generally discretized through BEM or FEM and solved by specific-domain algorithms. While being accurate, these models contain a high number of DoFs (typically several thousands or millions) and lead to computationally expensive analyses that are not suitable for first design phases and system-level simulation. To decrease the amount of DoFs , alternative methods consist to assemble predefined, configurable, elementary components. In contrast to traditional FE methods that mostly rely on coupling several domain-specific solvers,

the component-based approach yields a single matrix system solvable by a unique coupled solver. The component-based method was first limited to suspended structures because of the application of rigid body principles, but recent developments support nonlinear components, such as flexible elements, and the accurate modeling of damping and stiffening effects. Nevertheless, the transient analysis of such models still remains too slow to be considered for system-level analysis. Therefore, the compact and lightweight descriptions built upon original Three Dimensional (3-D) models through MOR techniques appear as an efficient manner to integrate MEMS models in system-level simulation environments. Thanks to several improvements during the last decade, MOR is now considered as a mature research area in applied mathematics.

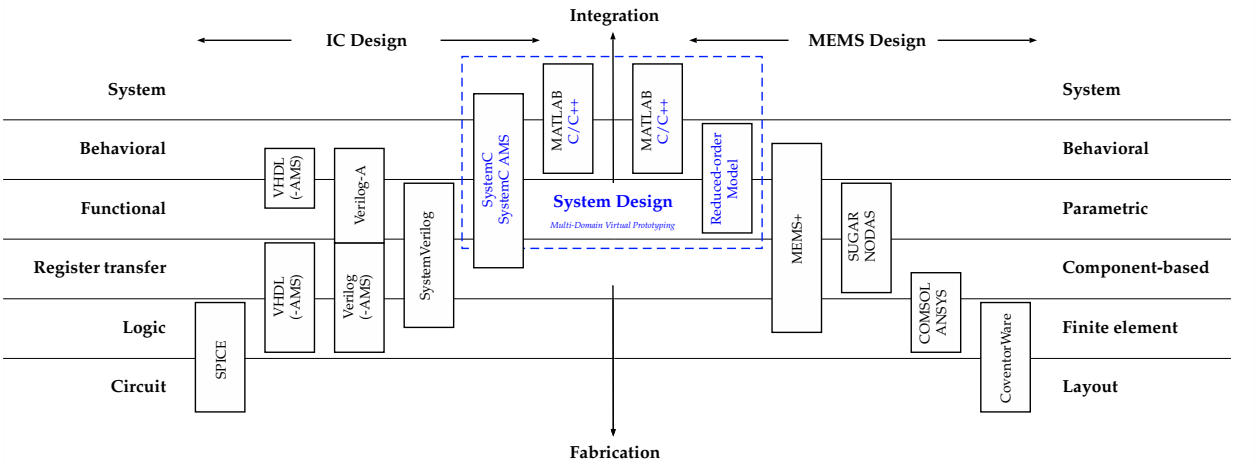


Fig. 2.18: IC and MEMS design flows with standard HDLs and traditional CAD tools converge at system-level into multi-domain virtual prototypes that allow the simulation of complex architectures that consist in MEMS, HW and SW sub-systems.

This PhD thesis is focused in the following on the use of reduced models in order to integrate MEMS reduced models directly in HDLs. First applications in VHDL-AMS or Verilog-A demonstrated the interest of such solutions to test the device with the surrounding electronics. Nevertheless, due to the low-level description of ICs provided by traditional HDLs, these solutions are not well fitted for large-scale system simulation. Moreover, the related languages remain decoupled from the development of software solutions. Therefore ESL design principles have been progressively incorporated into simulation environments, raising the level of abstraction of the overall simulation. SystemC is also considered as one of the most advanced solutions for the system-level modeling and simulation of SoC/SiP. By introducing conceptual representations for complex architectures, such as processors or micro-controllers, SystemC indeed enables the co-development of HW/SW specific applications. This C++ library benefited from recently standardized AMS extensions, i.e., SystemC AMS, already illustrated in several use cases. In this work, we thus envision to use the SystemC-AMS implementation to address the multi-domain virtual prototyping of systems integrating among others MEMS devices.

Chapter 3

ESL-based MEMS modeling

3.1 Introduction

In conceptual design, system-level models aim at evaluating the preferred layout with preliminary structural dimensions and related physical parameters. To model the electromechanical coupling, lumped elements such as parallel plate capacitor or simple beams are frequently employed to reproduce the electromechanical coupling. Derived from fundamentals of physical background introduced in the previous chapter, these models yield small ODE systems that can be solved in few minutes and are compatible with traditional ESL solutions. These models capture a lot of the typical MEMS behavior and enable a first evaluation of the design. In the following, we consider SystemC-AMS as solid basis to address the MDVP of MEMS devices with HW/SW specific applications.

By raising the abstraction level, SystemC provides a fast-simulation environment for electronics and notably improved the co-development of HW/SW solutions. The AMS extensions broaden the SystemC standard capabilities to the simulation of analog systems like RF or signal processing units. This chapter intends to demonstrate the ability of the current SystemC-AMS implementation to address the system-level modeling of MEMS, but also recall the limitations of a generic approach in terms of accuracy.

This chapter is also organized as follow. To make the chapter self-contained, we briefly review system-level modeling techniques usually employed. Section 3.2 also deals with the macro-modeling of MEMS devices. We introduce lumped-equivalent circuit and energy-based methods that represent MEMS through conservative systems to cope with internal physical couplings. These formalisms are then discussed against the traditional representation of MEMS as state-space description or transfer function. The specific use of the SystemC-AMS MoCs is further discussed with regard to MEMS system-level simulation requirements. To illustrate the different methods, macro-models of micro-machined inertial sensors, consisting of accelerometers and gyroscopes, are introduced in Section 3.3. We discuss the use of equivalent models to address the simulation of MEMS devices and their interface circuit with regard to the current limitations of a top-down modeling approach. To take into account the influence of the geometry and designer's choices on the behavior of the device, we finally argue for refined models built upon three-dimensional descriptions and sufficiently compact thanks to MOR methods.

3.2 Modeling methodologies

The monolithic integration of MEMS devices with ASIC requires the development of simulation and analysis tools that allow the coupling of circuit and micro-mechanical simulation [175]. This mainly relies on the development of models compatible with both micro-mechanical devices and control or signal-processing units. The modeling techniques of electronic components are now considered as mature, even in VLSI cases [176]. However, there is still a lack in developing models for MEMS with high level of abstraction. To describe the tight coupling of multiple energy domains in these systems, MEMS devices are usually represented by PDEs describing the motion of the structural members, by the characteristic equations of the transducer elements, and by a set of boundary conditions (see Section 2.2). The direct solution of these equations is obtained through fully meshed structures and remains computationally intensive. Thus, the system-level simulation and analysis of MEMS is still complicated to achieve with detailed models.

To overcome the tight computational cost of FE representations, different techniques have been proposed to build fast and efficient system-level models, called macromodels [177]. A common approach to construct lower-order device models is to develop semi-analytical macromodels. These models consist of equivalent circuit or system of DAEs whose parameters are determined through experiments or from detailed numerical simulations. Similarly, energy-based methods such as bond graphs [178] provide the necessary background to create conservative models based on the physical behavior of the device. Generating the forms of equivalent-circuit or energy-based models thus depend on designer's choice and requires an in-depth comprehension of the system. The modeling process may also take a long time. Nevertheless, both methods generate system-level models useful to evaluate the preferred layout with preliminary structural dimensions and the related physical parameters. These models generally recover the first- and second-order device behaviors and are effective to bind electronics and MEMS models in a unified simulation environment like SystemC-AMS. To this purpose, after introducing the principles of equivalent circuits and bond graphs, we explore in this section the capabilities of SystemC-AMS to implement such models in the different MoCs presented above.

Energy domain	**Effort** e	**Flow** f	**Generalized momentum** q	**Generalized displacement** p
Mechanical translation	Force F	Velocity $\dot{x}, v$	Momentum p	Position x
Mechanical rotation	Torque τ	Angular velocity ω	Angular momentum J	Angle θ
Electric circuit	Voltage V	Current i	…	Charge Q
Magnetic circuit	Magnetomotive force F_{mm}	Flux rate $\dot{\phi}$	…	Flux ϕ
Fluid flow	Pressure P	Volumetric flow Q	Pressure momentum Γ	Volume V
Thermal flow	Temperature T	Entropy flow rate $\dot{S}$	…	Entropy S

Table 3.1: Conjugate power variables, adapted from [4].

Lumped elements represent conservative systems as ideal physical entities connected to each other through bi-directional signals and ports. The exchange of energy between the device and the environment is achieved through ports. A port is defined by a pair $\{e, f\}$ of conjugate dynamic variables called the *effort* and the *flow*, respectively. These conjugate power variables can be specified for a broad range of physical domains. For instance, the most used variables in MEMS design are summarized in Table 3.1. Note the product of the conjugate variables is the power exchange through the port. The flow is given by the time derivative of the corresponding state variable. Figure 3.1 illustrates the energy exchange occurring in electromechanical transducers like electrostatic MEMS. The system is a two-port energy storage element. The electrical ports are characterized by the voltage-current pair $\{v(t), i(t)\}$, while the mechanical ports are defined by the force-velocity pair $\{F(t), v(t)\}$. These storage elements can be described by an energy function of two independent state variables each related to the mechanical and electric ports, i.e., the displacement x and the charge Q, respectively. The key-aim of lumped-element models is to clearly state the energy exchange and storage occurring in conservative systems such as MEMS. This approach is extended to other physical domains through energy-based methodologies, especially bond graph.

3.2.1 Equivalent-circuit representations

An often used lumped-parameter modeling approach describes the dynamics of the system through an equivalent electrical circuit. Once elaborated, this representation can be directly implemented in circuit simulators like SPICE in order to perform system-level simulation. Alternatively, the related DAE system can be extracted from the electrical network and solved in standard mathematical tools such as MATLAB.

In this modeling framework, mechanical elements are symbolized as electrical components with regard to equivalent conjugate power variables. With this approach, the stored potential energy in springs is represented by capacitors, the kinetic energy of the mass is represented by inductors, the dissipation of damping elements by resistors, and transforms between energy domains by transformers and gyrators. The development of such models is based on the analogy in the mathematical form of behavioral descriptions that exist in both domains. These analogies result from the formal similarities of the differential equations governing the behaviors of the electric and mechanical systems. For instance, the equation of motion which relates the force F and velocity v of a rigid mass m is given by $F = m\,\mathrm{d}v/\mathrm{d}t = m\,\mathrm{d}^2x/\mathrm{d}t^2$. From a mathematical viewpoint, this is similar to the constitutive equation of an electric inductor such as $v = L\,\mathrm{d}i/\mathrm{d}t = L\,\mathrm{d}^2q/\mathrm{d}t^2$. This direct analogy associates the force and voltage variables as well as the displacement and current variables together. A domain transformation is also considered as a dual relation equating an extensive variable (*effort*) to an intensive one (*flow*). The network topologies in mechanical and electrical domains are thus different from one another. In this case, a series connection in the electrical domain becomes a parallel arrangement in the equivalent mechanical system, and conversely. For example, a resonant system, i.e., a simple mass-spring-damper system, is modeled as RLC circuit, i.e., through capacitor, resistor and inductance assembled in series as shown in Figures 3.2(a) and 3.2(b). Additional resonances can be included by placing additional capacitor-resistor-inductance sets in parallel. Specific circuit elements appropriate to a wide range of linear MEMS sensor and actuator devices have been reviewed in [179]. Furthermore, to address nonlinear behaviors such as pull-in in MEMS, specific methods have been introduced in [180] and discussed by Senturia in [181]. This highlights the difficulty to correctly support the coupling between different energy domains through high-level modeling. Also, despite widely adopted, this method introduces two major issues in the modeling process.

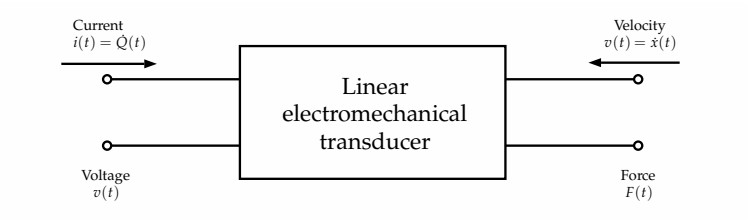
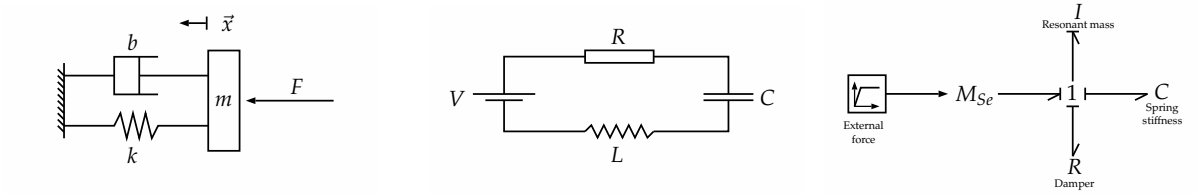


Fig. 3.1: Lumped-element representation of micromechanical transducer



(a) Resonant mechanical system. (b) RLC circuit. (c) Bond graph.

Fig. 3.2: Equivalent representations of a resonant mechanical structure.

On the one hand, the continuum device must be partitioned into an equivalent network of lumped elements which is particularly burdensome. Contrary to purely electric circuits, there is no clean mapping between geometry of general mechanical structures and corresponding network. Although more than a decade since first insights on the topic [80, 182], there is apparently no method nor tool to automatically generate an energetically correct lumped-element topology from an arbitrary device geometry. Moreover, the proposed equivalent representation would not necessarily be the right one and the definition of related parameters depends on experimental or simulation results. Therefore, even in early-design phase, designer input is required to build system-level models compatible with the targeted device.

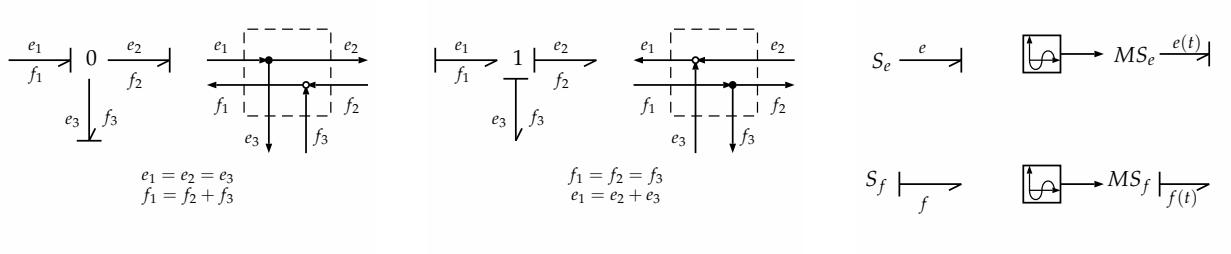
On the other hand, a numerical value must be defined for each parameter characterizing the different elements. The definition of descriptive variables is required to represent a three-dimensional device through lumped-equivalent system. In case of rigid bodies, these variables are the positions and orientations of constitutive elements. Similarly, the amplitude of deformation is observed for flexible entities through a shape function which is defined with regard to the structure and distributed loading of the element. For example, a flexible beam would be represented by a capacitor varying with regard to the load-displacement characteristic of the related spring element. Nevertheless, the energy variation in the spring, respectively in the capacitor, requires a shape function which is apparently not supported by a simple lumped equivalent. Defined from simulation or energy methods, the shape function represents the total stored energy in the element and aims at its minimization with respect to one or more undetermined parameters. Its correctness depends on the implementation of the stored energy computation. Regarding the previous limitations, energy methods like bond graph describe the coupling between different domains through transformers and gyrators. These bi-directional methods are considered as a good complement to lumped-element models in order to get first-order estimates of the device behavior, especially insights on the dependence of behavior on geometry and material properties.

3.2.2 Energy-based methodologies

As discussed in Chapter 2, the continuous behavior of dynamic systems is usually described in DAEs, ODEs and PDEs. The numerical simulation of these systems is performed by solving the set of equations and finding consistent initial conditions. To ease the definition of such representations, different techniques have been developed to decompose the systems into smaller lumped elements, as depicted above with equivalent circuits. Alternatively, bond graph provides a modeling formalism and a graphical notation that allows domain-independent description of the dynamic behavior of continuous systems and supports the hierarchical description of systems.

Bond graphs are based on the energy conservation principle and the use of a lumped approach. A bond graph consists of elements defining the system properties that are integrated through ideal connections. The related bi-directional model represents the energy exchanged in the system and determines the dynamics of the system through the estimation of power which is the product of effort and flow, i.e., the derivative of energy over time. This guarantees continuity since no energy is generated or dissipated.

Bond graphs are directed graphs whose edges represent physical processes and nodes are lumped elements. These latter exchange effort and flow though bonds with directions which represent the ideal exchange of energy in the system. Initially defined as non-causal models, bond graphs support the principle of causality in order to compute the exchange of power between elements, since the values of the two power variables, i.e., effort and flow, can not be computed at the same time. The causal analysis of bond graph also describes the model in computational terms and allows to derive the related set of equations. Therefore, a causal bond graph aims to identify which of the components causes the flow, or the effort.



(a) 0-junction node. (b) 1-junction node. (c) Source.

(d) Capacitor. (e) Inductor. (f) Resistor.

Fig. 3.3: Causality, equations and block diagram representation of main elements in bond graph.

Bond graphs consist in various standard components also called nodes, including junctions, capacitor (C-type), inductors (I-type), resistors (R-type), sources (Se, Sf), transformers (TF) or gyrators (GY). We briefly present hereafter some of these components, namely the junction, storage and resistive nodes (Figure 3.3). For more details on notation and methods of system analysis in bond graph, see [178, 183].

Junction nodes define the interactions between elements in a power-continuous way, i.e., by assuming no energy dissipation or storage. As set by Kirchhoff's laws in electrical networks, there are only two ways in which components exchange power. In the one hand, the *0*-junction represents a node where all the efforts of the connected bonds are equal, e.g., the parallel connections in electrical circuits. The sum of all the flows on the junction with regard to the power direction is zero, as defined by the Kirchhoff's current law in the electrical domain. In the other hand, the *1*-junction represents a node where all the flows of the connecting bonds are equal. In this case, the sum of efforts is assumed null corresponding to Kirchhoff's voltage law in the electrical domain or the force balance in the mechanical domain.

Storage nodes are characterized by port variables that are the generalized momentum q or the generalized displacement p. Depending on the imposed causality, these variables must be obtained either by the integration of the power variable with respect to time before being incorporated in the constitutive relation of the node, or by the differentiation of the the power variable with respect to time from a result of the constitutive equation. In the one hand, C-type nodes are related to q-type state variable, i.e, the integration of the flow or on the differentiation of the effort (Figure 3.3(d)). In the other hand, I-type nodes are related to p-type state variable, i.e., the integration of the effort or on the differentiation of the flow (Figure 3.3(e)).

Resistive nodes define components resisting to flow, e.g., current in the electrical domain, producing effort to be reduced between the input and output terminals, e.g., voltage in the electrical domain, while dissipating energy. This kind of element is used in various domains, i.e., resistors in the electrical domain or dampers in the mechanical problems. The constitutive equation is a linear relation between the effort and the flow such as $e = ri$, where r is the resistance value (Figure 3.3(f)).

As depicted above, every primitive of bond graph element defines one or more equations that involve the effort or flow variable values. These are handle by the two-signal connections, i.e., bonds, that connect the different nodes of the model and determine the bond causality with regard to the signal orientation and preferred integration scheme. Bond graphs support a wide scope of systems due to the multi- and inter-domain nature of this formalism. Moreover, the lumped approach allows an efficient evaluation and generation of design alternatives relying on the association features of bond and node components. Those attributes make bond graphs suitable for modeling mutli-domain systems or appropriate networks [184].

This formalism requires nevertheless a specific modeling knowledge. Moreover, like equivalent circuits, the definition of bond graphs relies on an in-depth understanding of the internal coupling of devices. The related parameters and constitutive properties must be obtained from a combination of analysis and numerical simulation. Despite recent support in CAD tools such as Modelica [46] or in SystemC-AMS with the development of the SCAX MoC [166], bond graphs remain difficult to connect directly to circuit simulators, which limits their applicability in the overall MEMS modeling environment. Since any lumped-circuit model or bond graph results in a set of coupled state equations, we explore in the following section how to directly construct model in the form of a set of coupled differential equations without any underlying circuit representation.

3.2.3 Transfer function and state-space system

As illustrated above through equivalent-lumped circuits and bond graphs, analytic models are derived from the fundamentals of physics and the principle of energy conservation. The related mathematical representation ultimately results in a set of coupled state ODEs or DAEs. In order to describe the entire dynamics of a system and its subsystems, it is necessary to introduce a vector, namely the state vector, whose components allow to describe the state of the system at any time. The state vector definition implicitly depends on the degree of complexity of the model which is set with regard to the underlying phenomena arising in the system. Since analytic models are always an approximation of real systems, it is primordial to correctly define the purpose of the model. For example, the same dynamical system will have a simulation model that differs from its control model. A first high-level model is sufficient to define the control synthesis. But, this control law must be verified with a simulation model whose the dynamical behavior is more realistic, and can be used as model in the loop for real time applications [185].

State-space representation and transfer function are two complementary approaches to analyze system behavior with regard to time and frequency domains, respectively. The time behavioral analysis concerns the transient response of the system to stimuli varying over time and potentially in space. The related differential system can be solved with traditional numerical tools. Alternatively, a system can be analyzed in frequency domain which is equivalent to a temporal analysis with trigonometric inputs, i.e., $u_0\, e^{j\omega_0 t}$. Frequency analysis introduces the notion of transfer between inputs and outputs through dedicated integral transformations (Laplace, Fourier). Although the same information can be retrieved in both representations, transfer approach has some limitations that we discuss below justifying our use of state-space systems.

Based on an external description, the definition of internal dynamics usually refers to multiple transfer functions. These functions can then be multiplied by one another which may introduce some simplifications of the system by so called pole-zero. The problem with the simplification by pole-zero is to assume the initial conditions as null which anneal some dynamical modes. This is a major inconvenient of the transfer approach since the system response relies on its past evolution, for instance, its initial conditions. Moreover, the input-output behavior may exclude internal dynamics that only affect the outputs. State-space representation overcomes this limitation by including all internal dynamics in the state vector. This method also supports Multiple-Input Multiple-Output (MIMO) systems which is not the case of the transfer function. Finally, transfer notion assumes the system linear in order to apply Laplace's transform. In case of nonlinear systems, Laplace's transform can not be defined or does not find any analytic expression. Therefore, the transfer notion does not apply to nonlinear systems.

The transfer approach allows to analyze linear systems from an input-output viewpoint. This external representation only relates inputs and outputs and may exclude some internal dynamics from the model. Moreover, the applicability of the concept of the transfer function is limited to linear, time-invariant, differential equation systems. As introduced above, its limitations reside in its difficulty to address the simplification by pole-zero, to set initial conditions, and to support multivariable systems. Furthermore, the Laplace's variable (s) does not enable a direct use of numerical simulation schemes. In order to simulate a transfer function, the transfer function must first be transformed into differential equations with null initial conditions, before to be solved by traditional methods.

The state-space representation proposes a unified analysis of the dynamical properties of systems. This method comprises a set of internal dynamics and takes into account the initialization of the system. Through the introduction of structural properties, this approach enables to study the problems of simplification by pole-zero or specific internal dynamics. Finally, it directly defines a set of ODEs easily solved by numerical algorithms.

The state-space sysem (2.8) is defined in the descriptor form, further studied in [186]. In the following, we consider the linear form of state-space systems that reads:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{3.1}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{3.2}$$

where $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times m}$, $\mathbf{C} \in \mathbb{R}^{p\times n}$, $\mathbf{D} \in \mathbb{R}^{p\times m}$. The state vector $x \in \mathbb{R}^n$, the input vector $u \in \mathbb{R}^n$, and the output vector $y \in \mathbb{R}^n$. Equation (3.1) is the evolution equation while Equation (3.2) is the observation equation. The matrix quadruplet $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ fully characterizes the state model. At any moment t_0, the value of an initial condition of the state $x(t_0) = x_0$ and of a piece-wise input vector $u(t) \in \mathcal{U}, t \geq t_0$, define in a unique way the trajectory $x(t) \in \mathcal{H}$ which is solution to (3.1) and thus the output trajectory $y(t) \in \mathcal{Y}, t \geq t_0$. The space $\mathcal{H}$ in which the trajectory $x(t)$ evolves is called the state space.

This state model can be represented by block diagram, as shown in Figure 3.4. This is based only on elements related to basic operations (addition, multiplication, integration). These components are well adapted to analyze linear systems and useful to simulation. The analysis and the control of such systems has been broadly covered in the literature [171, 187, 188].

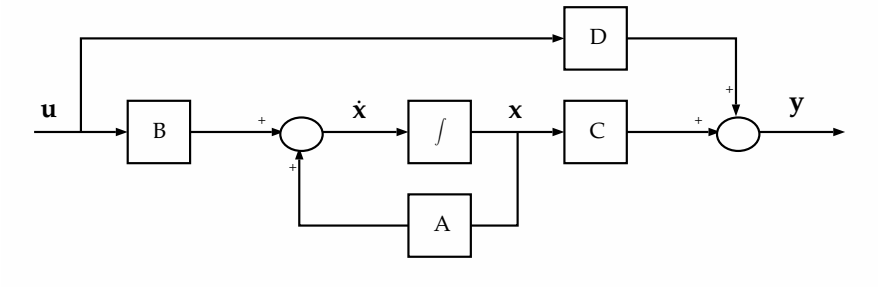


Fig. 3.4: Block diagram of a linear state-space system.

3.3 MEMS macromodels

Using the above rationale, we discuss below the system-level models of two widely used MEMS sensors that are accelerometers and gyroscopes. These micromachined inertial sensors represent the most important type of silicon-based sensors and are embedded in a wide range of applications, especially in inertial measurement units. Therefore, many models have already been proposed [189] and we limit our study to the generic definition of such devices. Apart from the description of these sensors with the previous methodologies, we have also devoted a small discussion on the signal processing related to analog out-

puts. In this section, we provide a first application of the modeling methods introduced above and that could be implemented in the different SystemC-AMS MoCs. The devices are also described by high-level descriptions, directly adapted from the underlying fundamental laws. This aims to define how sensor models can be developed using generalized field elements and understand the limitations of system models to design transducers and correctly tune the related HW/SW specific applications.

3.3.1 Accelerometers

Acceleration is the change in velocity for a given time period. In linear motion case, a mass that moves in straight line has positive acceleration with an increasing velocity. This is described by Newton's second motion law that establishes the relationship between force, mass and acceleration such as $F = m\,a$ where F is the force, m is the mass and a is the acceleration. An acceleration can thus be estimated by measuring the force applied on a mass through a scalar description that would be negative in case of decreasing velocity.

MEMS accelerometers are devices able to determine the force required to create a velocity change. The underlying principle can be approximated by mass-spring-damper system (Figure 3.2(a)). The corresponding system is sensitive to acceleration since the motion of the proof mass implies a variation of the spring length. The spring is indeed extended or contracted when acted upon respectively positive or negative acceleration. The observed displacement is generally linear with the force and is described by the equation $F = k\,x$, where k is the spring constant, also called stiffness coefficient. In most configurations, spring-mass systems are damped by drag and viscous forces which may induce nonlinear displacements. Therefore, bandwidth and usable frequency ranges must be correctly set according to these behavioral constraints.

The principle of accelerometers relies on a oscillating mass enclosed in a suspended frame (Figure 3.5). The mechanical input and the measured displacement may be different regarding the mounting resonance and damping characteristics. The accelerometer is based on a mass-spring-damper configuration and is characterized by a specific transfer function (3.4) from the case or package to moving mass. The aim of the device is to convert the sensed displacement of the frame into an electrical signal, e.g., through piezoelectric or capacitive measurement. This signal is then conditioned and processed by the interface circuit which operates the signal amplification, filtering and digital conversion. The governing equation for mass-spring-damper system is given by:

$$m\,\frac{\mathrm{d}^2x}{\mathrm{d}t^2} + b\,\frac{\mathrm{d}x}{\mathrm{d}t} + k\,x = u\;. \tag{3.3}$$

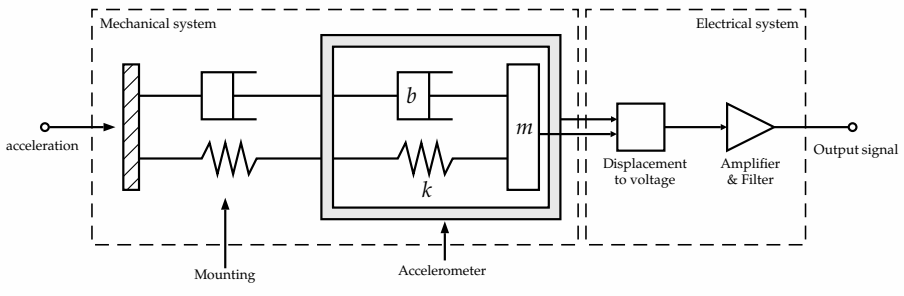


Fig. 3.5: Principle of the acceleration measurement by an accelerometer.

Hence the transfer function:

$$H(s) = \frac{X(s)}{Y(s)} = \frac{1}{s^2 + (b/m)\,s + (k/m)} = \frac{1}{s^2 + (\omega_0/Q)\,s + \omega_0^2}, \tag{3.4}$$

where b is the damping coefficient of the proof mass m, ω_0 is the resonant frequency defined by $\omega_0 = \sqrt{k/m}$ and Q is the quality factor defined by $Q = (m\,\omega_0)/b$. Note the mechanical sensitivity is inversely proportional to the square of the resonant frequency, i.e., proportional to $1/\omega_0^2$. This criterion is a fundamental design trade-off between device with good sensitivity or wide operating bandwidth.

Let the displacement be the output of the system, i.e., $y = x$, the equivalent state-space system is:

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} = \mathbf{C}\mathbf{x} \end{cases} \tag{3.5}$$

$$\text{where} \quad \mathbf{x} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} 0 \\ u \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 0 & 1 \\ -k/m & -b/m \end{bmatrix}, \quad \mathbf{B} = [0 \quad 1/m], \quad \mathbf{C} = [1 \quad 0]\,,$$

The above mechanical system has an equivalent lumped-element network where the speed $\mathrm{d}x/\mathrm{d}t$ is equivalent to an electrical current i, the mass m to an inductance L, the damping coefficient b to a resistor R and the stiffness coefficient k to the inverse of capacitance C. The electrical characteristic equation is given by:

$$L\,\frac{\mathrm{d}i}{\mathrm{d}t} + R\,i + \frac{1}{C}\int i\,\mathrm{d}t = V\,. \tag{3.6}$$

The mechanical description of the system (3.3), the related transfer function (3.4) or state-space system (3.5) and the equivalent electrical circuit (3.6) can all be implemented in SystemC-AMS in the different MoCs supported by the standard. For example, the state-space representation is supported in both TDF and LSF through the `sca_ss` class. Similarly, LSF MoC supports the definition of first- and second-order transfer functions with the `sca_lsf::sca_ltf` class. The above electrical network can be implemented in either ELN with a lumped-equivalent circuit or in LSF by defining the state-space equivalent to (3.6).

We limit our study to capacitive accelerometers and refer to [9] for an an in-depth introduction. Besides low fabrication cost, these devices are easily integrated with CMOS and have a high sensitivity as well as good temperature performance. They usually operate with natural frequencies in the range of 100 Hz to 20 kHz. The two most commonly used electrodes are parallel plates and finger comb arrays, as depicted below.

Figure 3.6 shows the basic configurations for capacitive displacement sensing through horizontal plates. This generally consists in structures composed of a pair of electrodes with one of them movable. Parallel plate electrodes are suitable for vertical sensing while comb fingers are preferred for sensing lateral displacement. The configuration in Figure 3.6(b) is generally preferred for DC offset and temperature variation cancellation thanks to the fully differential capacitance measurement. However, the configuration with a fixed reference capacitance shown in Figure 3.6(a) is much easier to implement.

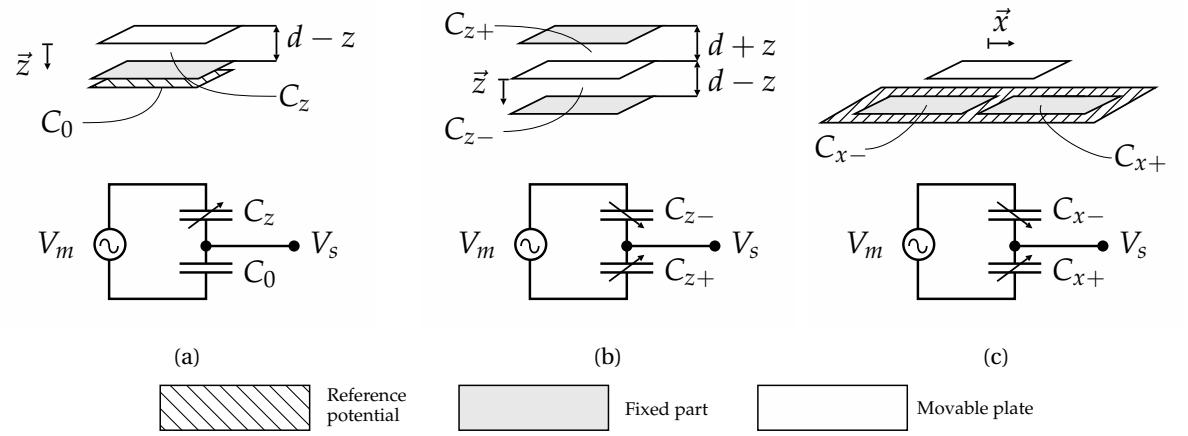


Fig. 3.6: Basic capacitive displacement sensing configurations with horizontal plates.

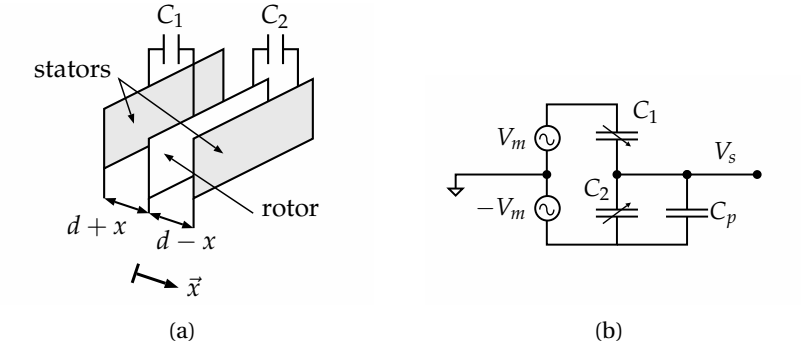


Fig. 3.7: Basic capacitive displacement sensing through sidewall capacitance of comb fingers.

Figure 3.7 illustrates the basic solution for capacitive displacement sensing with comb fingers. Comb-finger electrodes are normally used for lateral displacement sensing. As illustrated in Figure 3.7(a), each of the capacitors C_1 and C_2 formed by the three comb fingers include a sidewall parallel-plate and an edge capacitance. When the middle part, i.e., the rotor, moves along the **x**-axis, C_2 decreases while C_1 increases which is equivalent to the fully differential capacitive half-bridge depicted in Figure 3.6(b). Note that the parasitic capacitance C_P attenuates the signal and should be minimized. The resulting sensitivity to acceleration for this configuration is given by:

$$\frac{V_S}{a_{ext}} = \frac{1}{x\omega_r^2}\left(\frac{2V_m(C_1-C_2)}{C_1+C_2+C_P}\right) \approx \frac{2V_m}{\omega_r^2 d\left(1+\frac{C_P}{2C_0}\right)}, \tag{3.7}$$

where V_S is the output voltage, a_{ext} the external acceleration applied on the device, V_m a balanced AC modulation voltage, d is the gap between plates when the proof mass is not displaced, C_0 the value of the corresponding capacitance at steady state.

The resolution of inertial sensor may be limited by external parameters like temperature, power supply changes or electrical noise. For instance, the thermomechanical noise is related to the damping and is defined by its spectral density as follow:

$$\sqrt{\frac{a_n^2}{\Delta f}} = \frac{\sqrt{4k_B T b}}{m} \,, \tag{3.8}$$

where a_n is the Brownian equivalent acceleration noise, Δf is the system bandwidth usually set by the interface circuit, k_B is the Boltzmann's constant, and T is the absolute temperature in Kelvin. The unit of the thermal density is $m/s^2 Hz^{-1/2}$ or $gHz^{-1/2}$. Assuming the damping between the capacitor plates is mostly due to squeezed-film effect, the damping coefficient is:

$$b = \frac{\eta L h^3}{d^3} = \frac{4\eta C_0 h^2}{\varepsilon_0 d^2} \,, \tag{3.9}$$

where η is the viscosity of the fluid between the parallel plates, ε_0 is the permittivity of fluid, L is the total length of the plates in the system and h is the width of the plates. The damping is directly related to the aspect ratio of the plates. Let assume $h_C \ll L_C$ which is the case in lateral capacitive sensing, the Brownian noise with squeezed-film damping is also defined by:

$$\sqrt{\frac{a_n^2}{\Delta f}} = 4\sqrt{\frac{k_B T \eta}{\varepsilon_0}} \left(\frac{\sqrt{C_0}}{\rho A_m d}\right) \left(\frac{h}{h_m}\right) \,, \tag{3.10}$$

where ρ is the density of the proof mass, A_m is its area and h_m is its height. Finally, the Brownian noise can be expressed by introducing (3.7) in (3.10) with regard to the capacitive bridge output voltage V_S as follow:

$$\sqrt{\frac{V_S^2}{\Delta f}} \approx \frac{8V_m}{\rho A_m \omega_r^2 d^2} \sqrt{\frac{k_B T \eta}{\varepsilon_0}} \left(\frac{\sqrt{C_0}}{1 + \frac{C_p}{2C_0}}\right) \left(\frac{h}{h_m}\right) \,. \tag{3.11}$$

To decrease the noise, a larger mass would decrease its amplitude and the fluid damping would be annealed if the device operates in vacuum. In other fluids, especially air, smaller sense capacitors and larger sense gap would reduce the noise since the squeezed-film effect would be less important. However, this approximation is limited by the shear damping induced by the plate deflection which is not taken into account in the above macromodels. Different topologies are also employed to counterbalance the noise incidence on the capacitive interface, the most common being switched capacitor circuits [189].

The bias, i.e., offset, and gain stability are important characteristics of accelerometers. Bias issues may durably change the stress in microstructures. The corresponding stress gradients defined in Chapter 2 are also conditioned by the temperature of the structure and chip substrate. Moreover, external stresses may be induced by actions on the package such as temperature changes, external forces or manufacturing variations in mounting, wire-bonding or encapsulation in the chip. The resulting bias drift may directly impact the mechanical behavior of the device. Therefore, the study of thermal and packaging effects is required even at system-level, but needs for refined definition of the device [119].

3.3.2 Gyroscopes

Gyroscopes have been successfully designed as microsystems through the MEMS technology. Vibrating gyroscopes use oscillating mechanical elements to sense the rate or the angle of rotation. These sensors are based on the transfer of energy between two vibration modes of a structure induced by Coriolis force, well introduced in [190].

As illustrated in Figure 3.8, a gyroscope consists of a suspended mass connected to a fixed frame through two pairs of springs. The mass is thus free to vibrate in two mutually perpendicular directions and tilts around the **z**-axis. The mass oscillates in a sine wave pattern in the driving axis, here **x**-axis, and the rotation sensing is done on **y**-axis (see results simulation in Figure 3.10). If a rotation speed $\dot{\theta}$ is applied on around **z**-axis, the corresponding tilt induces an additional force on the mass with regard to the velocities in the **x**- and **y**-axis. This force is called Coriolis force, $F_{Coriolis}$ and is given by the following equations of motion:

$$\begin{aligned} m\ddot{x} + b_1\dot{x} + k_1 x &= F_1 + 2m\dot{\theta}\dot{y} \\ m\ddot{y} + b_2\dot{y} + k_2 y &= F_2 - 2m\dot{\theta}\dot{x}\ , \end{aligned} \tag{3.12}$$

where m is the mass, b_1, b_2 are the damping terms, k_1, k_2 are the stiffness of springs and F_1 is the excitation force applied on **x**-axis and F_2 is the sensing force measured on **y**-axis. Hence the state-space system:

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} = \mathbf{C}\mathbf{x} \end{cases}, \tag{3.13}$$

where

$$\mathbf{x} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -k_1/m & -b_1/m & 0 & 2\dot{\theta} \\ 0 & 0 & 0 & 1 \\ 0 & -2\dot{\theta} & -k_2/m & -b_2/m \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 & 0 \\ 1/m & 0 \\ 0 & 0 \\ 0 & 1/m \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1\,0\,0\,0 \\ 0\,0\,1\,0 \end{bmatrix}.$$

The motion in the sensing direction, here **y**-axis, is defined as follow:

$$\Delta y = \frac{F_{Coriolis} Q_y}{m\omega_y} \frac{1}{\sqrt{\left(\omega_x^2 - \omega_y^2\right)^2 + \frac{\left(\omega_x \omega_y\right)^2}{Q_y}}} \tag{3.14}$$

where Δy is the displacement variation, i.e., the measurement, in the **y**-direction, ω_y and ω_x are the natural pulsations in both directions respectively, and Q_y is the quality factor for the motion in the **y**-direction. The Coriolis force, $F_{Coriolis}$, is equal to $-2m\dot{\theta}\dot{x}$ and allows to compute the angular rate θ from the displacement in the **y**-direction. The quality factor Q_y for the motion in the **y**-direction can be written as:

$$Q_y = \frac{\sqrt{mk_y}}{b_2}\ . \tag{3.15}$$

We easily understand through this equation that a minimized damping increases the quality factor of the device. Therefore, the system is usually operated in vacuum [191].

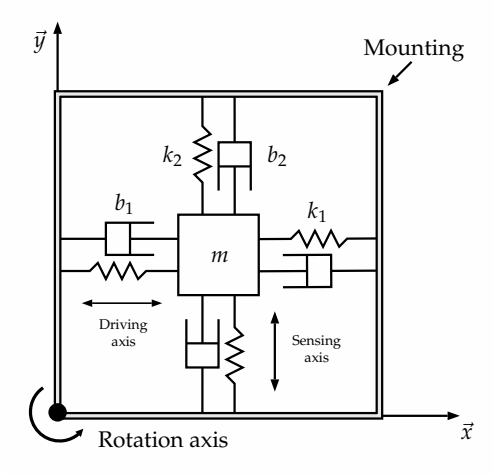
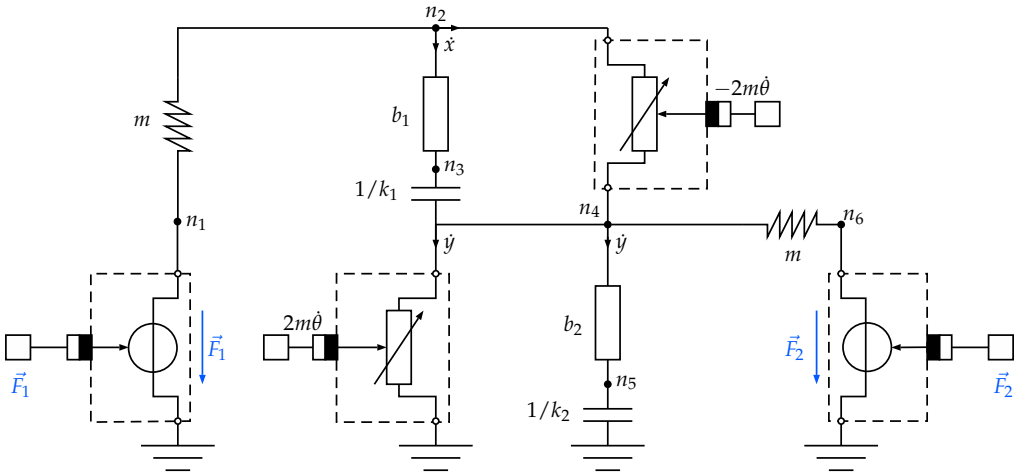
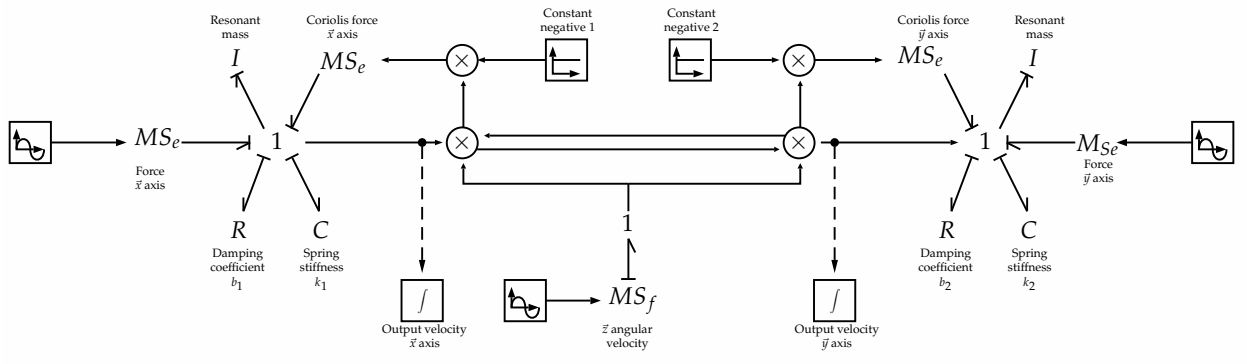


Fig. 3.8: Principle of the angular displacement measurement by a gyroscope.



(a) Equivalent electrical circuit.



(b) Bond graph.

Fig. 3.9: System-level representation of a gyroscope with Coriolis force modeling.

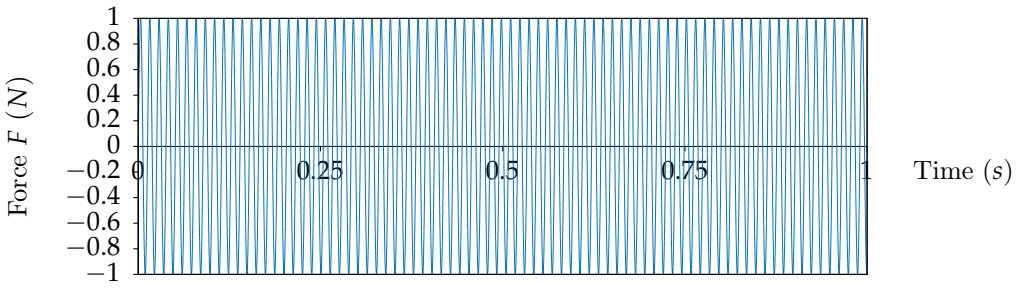
Figure 3.9(a) shows the equivalent circuit to the proposed gyroscope model. The mass-spring-damper system is also described by two complementary electrical resonant systems that consist of RLC lumped elements with regard to the aforementioned rules on analogy. This system represent the DoFs on both **x**- and **y**-directions, and the rotation about the **z**-axis is modeled through modulated resistors. The force inputs are defined as voltage sources.

The bond graph representing the gyroscope is given in Figure 3.9(b). Modulated sources of flow, MS_f, generates the rotation about the **z**-axis. This induces the Coriolis forces for both **x**- and **y**-axis as depicted by the central coupled binding. The resonant systems on both directions are modeled through I-, C- and R-type elements. Finally, the external forces, F_1 and F_2 are delivered by modulated source of effort, MS_e. We can notice that bond graph refers to the same model for both the mechanical and the electrical system. Additional modeling elements enable to access the integrated value of some variable, ,e.g., the output velocity on both axis. Furthermore, the related analysis of causality can be exploited to design control systems and test the design itself by simply tuning the coefficients [192]. In SystemC-AMS, the electrical network can be implemented in the ELN MoC using the standard predefined elements. Similarly, the bond graph model could be implemented in the SCAX library. To compute the Coriolis forces on both axes, the model defines a loop over the output velocities that are multiplied with the angular velocity and re-injected in the system. This also induces an algebraic loop which is nevertheless not supported by the solver yet. We therefore chose to instantiate the state space system (3.13) in the TDF MoC and thus refer to the standard AMS extensions. We discuss hereafter the simulation configuration and limitations.

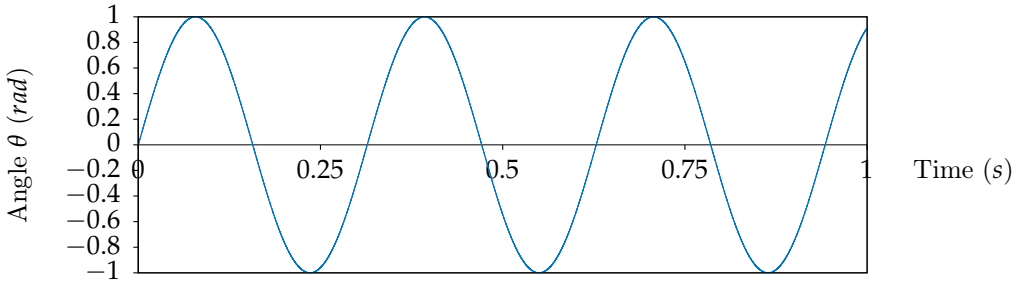
To solve this system, we used the `sca_tdf::sca_ss` class that accepts as attributes the system matrices, the state and input vectors and the current time step of the module. But, the `sca_ss` definition is limited to Linear Time Invariant (LTI) systems, i.e., with no variation in the constitutive matrices. Since the matrix $\mathbf{A}$ depends on θ variation over time, we must update the corresponding terms during the simulation. The SystemC-AMS vectors (`sca_vector`) and matrices (`sca_matrix`) correspond to data structures without support for mathematical operations. In order to solve the system, we use the *Eigen* library [193] which contains data structures and operations for linear algebra. In addition, we instantiated our own procedures to convert *Eigen* data structures into SystemC-AMS (`writeEigen2Sca`) and conversely (`writeSca2Eigen`). Note this conversion directly impacts and lengthens the simulation run-time. The proposed implementation is detailed in Appendix C.

The simulation results are given in Figure 3.10 and the numerical application is detailed in Table 3.2. Figure 3.10(a) shows a sinusoidal force applied along **x**-axis at the resonant frequency of the device, 495 rad/s. This input is modulated by the angular velocity $\dot{\theta}$ varying at a pulsation of 20 rad/s as shown on Figure 3.10(b). Finally, the displacement in **y**-direction obtained by the simulation is depicted in Figure 3.10(c).

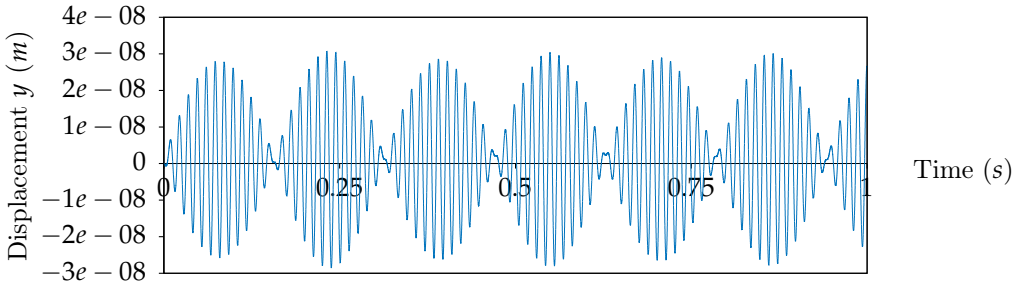
The system-level modeling of gyroscopes enables to quickly evaluate the behavioral response of such systems. Nevertheless, the coupling of the angular velocity with the displacement and velocity on both the sensing and the driving directions requires to tune the simulation in SystemC-AMS. Such models can further be connected to control units which are specifically detailed for gyroscopes in [194]. Despite some limitations in both the modeling and simulation processes, SystemC-AMS supports a wide range of standard procedures to cover the signal conditioning associated to global assemblies like inertial measurement units and introduced below.



(a) Applied force on x axis.



(b) Variation of the angular velocity.



(c) Displacement on y axis.

Fig. 3.10: Simulation result in SystemC-AMS for a gyroscope.

Variable	Symbol	Value	Unit
Mass	m	$9.425e^{-10}$	kg
Stiffness x-axis	k_1	$25e^{-03}$	-
Stiffness y-axis	k_2	$25e^{-03}$	-
Damping x-axis	b_1	0.0	-
Damping y-axis	b_2	3480.0	-
Pulsation x-axis	ω_x	495	rad/s
Pulsation y-axis	ω_y	495	rad/s
Angular velocity	$\dot{\theta}$	20.0	rad/s

Table 3.2: Numerical application of gyroscope, adapted from [195].

3.4 Conclusion

In this chapter, we provided the necessary background for the modeling and simulation of MEMS devices in SystemC-AMS through high-level representations. The SystemC AMS extensions enable the simulation of more complex systems and enrich the co-development of HW/SW specific applications with the modeling of, e.g., RF or signal processing units. The related MoCs allow the representation of systems as data flows, block diagrams or electrical networks. Furthermore, the modular definition of SystemC encouraged the development of additional MoCs to address the MDVP of heterogeneous systems.

Different techniques are envisioned to address the system-level modeling of MEMS. We also introduced modeling techniques to represent MEMS either as conservative or non-conservative systems. On the one hand, equivalent-lumped circuits or bond graphs define microsystems as conservative entities. These modeling methodologies take into account the description of internal couplings and provide equivalent representations applying analogies between the different physical domains involved in the system. On the other hand, state-space representation or transfer function provide a non-conservative and oriented definition of systems. The related set of ODEs enable to refine the system and control design through matrix analyses.

To illustrate the previous methodologies, we proposed macromodels of inertial sensors, consisting in accelerometers and gyroscopes. The different formalisms provide a basic representation of the devices which is acceptable for a first-cut design, but becomes irrelevant for complex geometry or specific feature realization. These models lack sufficient level of accuracy and may not predict complicated dynamical behaviors, such as the effect of higher order modes on the response of microstructures. Furthermore, parasitic phenomena like thermal noise are difficult to integrate in such models but directly impact the system behavior, e.g., packaging influence.

The top-down approach supported by the definition of additional MoCs in SystemC-AMS has also two main disadvantages. First, the underlying description may ignore important effects on the structure and will limit the refinement of HW/SW specific application like control and signal processing units. Second, the proposed models can not be automatically generated for a specific device. The assumptions made on the geometry may also lead to approximations that will not ensure the correct behavioral representation of the device. Designing MEMS remains a complex task requiring the designer's input. To overcome the limitation of traditional system-level modeling techniques, we further propose to integrate reduced-order models, initially created in FEA tool, in SystemC-AMS.

Chapter 4

System-level simulation API

4.1 Introduction

In the previous chapter, we introduced lumped-element and energy-based modeling methods to address the system-level simulation of MEMS devices. We demonstrated the ability of SystemC-AMS to support the definition and simulation of such models with respect to the standard implementation. This environment enables fast simulations with runtime of few seconds instead of minutes or hours in traditional HDLs which is highly valuable in early-design or integration phases The modeling and simulation of MEMS in SystemC-AMS primarily aim to correctly configure HW/SW dedicated applications. To this end, the models must verify good performance, smooth definition and sufficient accuracy.

The behavior of MEMS devices is highly dependent on the geometry and material properties. As depicted in Chapter 2, this is generally handled by large-scale systems for which transient simulation still remains hard to compute. In Chapter 3, we introduced lumped-element and energy-based models in which it is still hard to take into account such parameters. To overcome the above limitations, MOR techniques are the preferred alternative to extract the main parameters and generate compact, lightweight models from initial refined descriptions. This bottom-up approach is compatible with the system-level simulation supported by most HDLs, especially through AMS extensions. SystemC-AMS enables the co-development of digital HW/SW applications with analog or non-electronic components. We investigate here the integration of MEMS models into SystemC-AMS architectures.

In this chapter, we deal with reduced-order models automatically generated from the FEA tool *MEMS+*. We limit our investigation to the integration of these models in SystemC-AMS and the analog part of larger architectures. By using reduced models, we hope to cope with the geometry complexity and nonlinearity of devices with sufficient accuracy and thus provide a sustainable solution for MEMS system-level integration. This chapter follows the chronological development of the MOR feature implemented in *MEMS+* and developed in parallel to this work. First, we justify the use of reduced models in SystemC-AMS by presenting the initial version of the model export in *MEMS+*. This solution refers to a previous MOR implementation in *MEMS+* in which the reduced models are sampled at different operating points in order to reconstruct the nonlinearity of the system. In a second part, we detail our specific contribution with the implementation of an API in order to directly access the reduced models exported from *MEMS+* to SystemC-AMS. We reuse the feature initially developed for MATLAB/Simulink and based on the principles of an updated MOR technique refined to cope with the electrostatic nonlinearity of studied systems. We finally illustrate the use of the proposed API on test bench implementing a gyroscope.

4.2 Motivating example

We present below the first implementation of reduced models generated from *MEMS+* and integrated in SystemC-AMS that we presented in the following publications [196, 197].

4.2.1 Model definition

MEMS are nonlinear dynamic entities, hereby modeled through the single multi-physics system of equations assembled in *MEMS+*. Consider $\mathbf{F} : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ the sum of forces acting on the system at equilibrium defined as follow:

$$\mathbf{F}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{u}) = 0 \tag{4.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state vector representing the internal dynamics of the system, i.e., all electrical and mechanical DoFs, $\dot{\mathbf{x}} \in \mathbb{R}^n$ is its derivative and $\mathbf{u} \in \mathbb{R}^m$ is the input vector which includes all input voltages, external accelerations and angular velocities.

Linearization

The 1$^{\text{st}}$-order Taylor expansion of the conservation law (4.1) around a steady-state $(\mathbf{x}_i, \dot{\mathbf{x}}_i, \mathbf{u}_i)$ is given by:

$$\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{u}) \approx \mathbf{F}_0 - \mathbf{A}_i\mathbf{x} - \mathbf{B}_i\mathbf{u} + \mathbf{E}_i\dot{\mathbf{x}}\,, \tag{4.2}$$

with $\mathbf{A}_i = \partial_{\mathbf{x}}\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}_i, \mathbf{u}_i)$, $\mathbf{B}_i = \partial_{\mathbf{u}}F(\mathbf{x}_i, \dot{\mathbf{x}}_i, \mathbf{u}_i)$, $\mathbf{E}_i = -\partial_{\dot{\mathbf{x}}}\mathbf{F}(\mathbf{x}_i, \dot{\mathbf{x}}_i, \mathbf{u}_i), \mathbf{F}_0 = \mathbf{F}(\mathbf{x}_0, \dot{\mathbf{x}}_0, \mathbf{u}_0)$. Hence the linearized form of (4.2) is defined as follow:

$$\mathbf{E}_i\dot{\mathbf{x}} = \mathbf{A}_i\mathbf{x} + \mathbf{B}_i\mathbf{u} + \mathbf{r}_i, \tag{4.3}$$

where $\mathbf{E}_i \in \mathbb{R}^{n\times n}$ is the mass matrix, $\mathbf{A}_i \in \mathbb{R}^{n\times n}$ is the system matrix, $\mathbf{B}_i \in \mathbb{R}^{n\times m}$ is the input or control matrix and $\mathbf{r}_i = \mathbf{F}_0 - \mathbf{A}_i\mathbf{x}_i - \mathbf{B}_i\mathbf{u}_i + \mathbf{E}_i\dot{\mathbf{x}}_i$ is the force residual.

Model Order Reduction

The transient analysis of FE models is generally not possible due to the huge number of DoFs ($> 10^6$). MOR aims to decrease the number of DoFs of the system while preserving the principal dynamics of the system. This method is applied to produce smaller models, well suited for system-level simulation. In this section, we apply a first approximation based on the Petrov-Galerkin projection (see Section 2.3)

Let $V \in \mathbb{R}^{m\times n}$ with $m \ll n$ be the reduction matrix. It aims to approximate the manifold where the state vector $\mathbf{x}$ resides through a transformation of basis that replaces $\mathbf{x}$ by $\mathbf{V}\tilde{\mathbf{x}}$ in (4.3) with $\tilde{\mathbf{x}} \in \mathbb{R}^m$ leads to:

$$\mathbf{E}_i\mathbf{V}\dot{\tilde{\mathbf{x}}} = \mathbf{A}_i\mathbf{V}\tilde{\mathbf{x}} + \mathbf{B}_i\mathbf{u} + \mathbf{r}_i. \tag{4.4}$$

The above system has more equations than unknowns which can be solved through the Petrov-Galerkin projection. Let $\mathbf{W}$ a matrix in $\mathbb{R}^{n\times m}$, the corresponding reduced system reads:

$$\mathbf{W}\mathbf{E}_i\mathbf{V}\dot{\tilde{\mathbf{x}}} = \mathbf{W}\mathbf{A}_i\mathbf{V}\tilde{\mathbf{x}} + \mathbf{W}\mathbf{B}_i\mathbf{u} + \mathbf{W}\mathbf{r}_i. \tag{4.5}$$

To enhance the stability of the solution, we follow the method developed by Amsallem *et al.* [198] and further depicted in [79]. We remain in the case $\mathbf{W} = \mathbf{V}^T$ leading to the following linear reduced model:

$$\tilde{\mathbf{E}}_i\dot{\tilde{\mathbf{x}}} = \tilde{\mathbf{A}}_i\tilde{\mathbf{x}} + \tilde{\mathbf{B}}_i\mathbf{u} + \tilde{\mathbf{r}}_i, \tag{4.6}$$

where $\tilde{\mathbf{E}}_i = \mathbf{V}^T\mathbf{E}_i\mathbf{V}$, $\tilde{\mathbf{A}}_i = \mathbf{V}^T\mathbf{A}_i\mathbf{V}$, $\tilde{\mathbf{B}}_i = \mathbf{V}^T\mathbf{B}_i$, and $\tilde{\mathbf{r}}_i = \mathbf{V}^T\mathbf{r}_i$.

Nonlinearity Reconstruction

The reduction method defines a linear reduced model $(\tilde{\mathbf{E}}_i, \tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i, \tilde{\mathbf{r}}_i)$ around a specific operating point $(\dot{\tilde{\mathbf{x}}}_i, \tilde{\mathbf{x}}_i, \mathbf{u}_i)$. In order to reconstruct the nonlinear behavior of the whole system, several linear models are generated and interpolated. In this version, the order of approximation is fixed by the user at the export in *MEMS+* and determines the number of sampled linear systems required to further reconstruct the nonlinearity. The reduced nonlinear system to solve reads:

$$\sum_{i=0}^{k}\omega_i\tilde{\mathbf{E}}_i\dot{\tilde{\mathbf{x}}} = \sum_{i=0}^{k}\omega_i\tilde{\mathbf{A}}_i\mathbf{V}\tilde{\mathbf{x}} + \sum_{i=0}^{k}\omega_i\tilde{\mathbf{B}}_i\mathbf{u} + \sum_{i=0}^{k}\omega_i\tilde{\mathbf{r}}_i, \tag{4.7}$$

where ω_i are weighted coefficients computed according to the current value of the current input values $\mathbf{u}$, satisfying:

$$\sum_i \omega_i = 1, \quad \text{with} \quad \omega_i(\mathbf{u}) \in [0,1]. \tag{4.8}$$

After interpolation, the nonlinearity is thus preserved by updating the value of constitutive matrices regarding $\tilde{\mathbf{x}}$, and $\mathbf{u}$. The system reads:

$$\tilde{\mathbf{E}}\dot{\tilde{\mathbf{x}}} = \tilde{\mathbf{A}}\tilde{\mathbf{x}} + \tilde{\mathbf{B}}\mathbf{u} + \tilde{\mathbf{r}}, \tag{4.9}$$

with, $\tilde{\mathbf{E}} = \sum_{i=0}^{k}\omega_i\tilde{\mathbf{E}}_i$, $\tilde{\mathbf{A}} = \sum_{i=0}^{k}\omega_i\tilde{\mathbf{A}}_i$, $\tilde{\mathbf{B}} = \sum_{i=0}^{k}\omega_i\tilde{\mathbf{B}}_i$, and $\tilde{\mathbf{r}} = \sum_{i=0}^{k}\omega_i\tilde{\mathbf{r}}_i$.

Linearization, model reduction, and nonlinearity reconstruction are the complementary steps to define a system-level model in SystemC-AMS from FEA. In this case, the linearization and model reduction are directly performed in *MEMS+* as model export. Figure 4.1 shows how the nonlinearity is then reconstructed during the simulation in SystemC-AMS with respect to the current value of $\tilde{\mathbf{x}}$ and $\mathbf{u}$. These are read on the module's ports during the execution of the `processing` method. For simplicity's sake, we assume in the following the system matrices as reduced and don't use the tilde symbol to denote them anymore.

SystemC-AMS Implementation

SystemC-AMS defines two different modules for state-space representation, `sca_tdf::sca_ss` in TDF MoC and `sca_lsf::sca_ss` in LSF MoC [24]. On the one hand, the TDF MoC is used for procedural behavior processing samples tagged in time. The activation schedule of a set of connected TDF modules can be statically determined since the number of read and written operations is known and fixed. This MoC is thus well adapted for multi-rate signals. On the other hand, the LSF MoC is dedicated to the description of linear time-invariant and non-conservative systems using block diagram primitives. Switching and modulation of module parameters via external signals are allowed, making this MoC more suitable for simple continuous-time controllers and filters. The TDF MoC is preferred in this first case study as it also implements the synchronization tasks with SystemC .

The predefined TDF module, `sca_tdf::sca_ss`, assumes the following state-space system:

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \\ \mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}. \end{cases} \tag{4.10}$$

This differs from the 3-D model created in *MEMS+* which is exported to SystemC-AMS as a reduced-order model and represented by a state-space system in descriptor form:

$$\begin{cases} \mathbf{E}\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{r}, \\ \mathbf{y} = \mathbf{C}\mathbf{x}. \end{cases} \tag{4.11}$$

To respect the SystemC-AMS standard, we need to convert (4.11) to (4.10), i.e., to invert the mass matrix $\mathbf{E}$ and anneal the residual member $\mathbf{r}$. To this end, a left multiplication by $\mathbf{E}^{-1}$ is first applied. Because of the system nonlinearity, the matrix $\mathbf{E}$ is non-constant and so is its inverse. Therefore a LU (Lower Upper) decomposition needs to be performed at each iteration and this directly impacts the simulation performance. In the same way, a variable change can deal with the constrain $\mathbf{r} = 0$.

Let $\mathbf{z} = \mathbf{x} + \mathbf{A}\mathbf{E}^{-1}\mathbf{r}$. A similar variable change is performed on $\mathbf{y}$ which reads $\mathbf{y} = \mathbf{C}(\mathbf{z} - \mathbf{A}^{-1}\mathbf{E}^{-1}\mathbf{r})$. We finally get the system:

$$\begin{cases} \dot{\mathbf{z}} = \mathbf{E}^{-1}\mathbf{A}\mathbf{z} + \mathbf{E}^{-1}\mathbf{B}\mathbf{u}, \\ \mathbf{x} = \mathbf{z} - \mathbf{A}^{-1}\mathbf{E}^{-1}\mathbf{r}, \end{cases} \tag{4.12}$$

To overcome the limitations of the standard SystemC-AMS state-space representation, we proposed alternative functions declared in the `processing` method of a dedicated TDF module. The solving method deals with the exported system (4.12). As SystemC-AMS does not support matrix operations, we use *Eigen* [193], a C++ header library dedicated to linear algebra, to solve the state-space system. As we are interested in resonating sensors, i.e., potentially stiff systems, we have chosen the Crank-Nicolson time integration method.

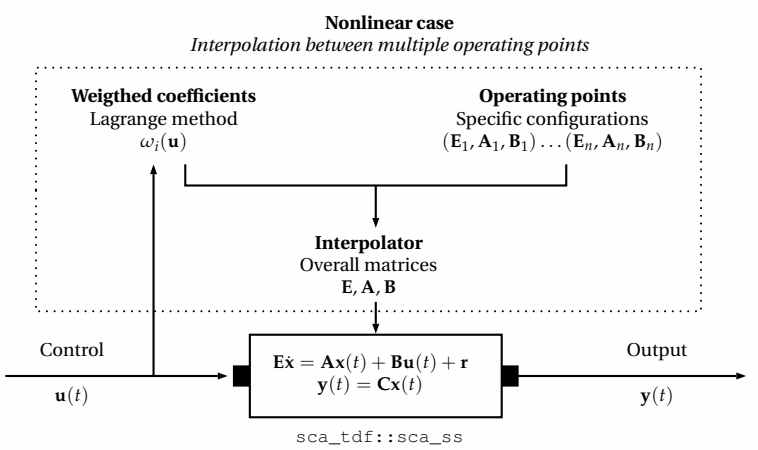


Fig. 4.1: First implementation of the model export with nonlinearity reconstruction during the simulation.

4.2.2 Use case

The selected MEMS is a biaxial accelerometer commercialized by ST Microelectronics, ST LIS332AR. It measures in-plane motion in two directions, x- and y-axis through capacitive electrodes. Its fabrication relies on THick Epipoly Layer for Micro-actuators and Accelerometers (THELMA®) micro-machining process, dedicated to suspended structures of relatively large thickness.

This sensor is composed of an inertial mass suspended to a frame through two pairs of springs as shown in Figure 4.3(a). Each spring is connected to a resonator that consists of an actuated beam vibrating at high frequency, here about $100kHz$. As demonstrated by Tocchio *et al.* [199], the use of an L-shaped resonator with high-resonant frequency aims to mechanically reduce the signal-to-noise ratio. Here, the first mechanical mode of the resonating beam is at a frequency of $105\,378Hz$ as shown in Figure 4.3(b). The resonant structure is also decoupled from the measurement one which allows for sensing applications. While operating, the in-plane displacements of the mass are captured through the folded springs. The resulting bending of the resonator is sensed by electrodes alongside the resonator and is measured as a variable capacitance. The signal can then be amplified and filtered by electronic components, not represented here.

The details provided by Comi *et al.* in [200] on the geometry and the sensing principle of the device are summarized in the Table 4.1. We created the model corresponding to the device in *MEMS+* and respect the design as well as the modal analysis defined in [201]. In *MEMS+*, we selected specific items from the component-based library as shown in the exploded diagram in Figure 4.2. For instance, the resonator is modeled as a Timoshenko beam, instead of a Bernoulli beam, in order to fully reconstruct its modal response. *MEMS+* supports the definition of perforations and thus avoids to find an equivalent density for the proof mass.

The FEA simulation performs the modal analysis of the proposed device. Table 4.2 lists the first six modes with the corresponding frequencies. The two lowest frequency modes correspond to the sensing modes, i.e. the translation in-plane modes. These modes have a frequency at $1\,620Hz$. The two following modes occur around $1\,976Hz$ and $7\,178Hz$ and correspond to the in-plane rotation around $\mathbf{z}$-axis and to out-of-plane translation respectively. Due to the symmetry and the differential measurement scheme of the device, these two modes do not influence the output signal as demonstrated in [201]. The other modes preserve the low sensitivity of the device since their frequencies are high enough, around $8\,860Hz$.

Table 4.1: Accelerometer Parameters

Parameter	Value Unit
Resonator length	352.0 μm
Resonator thickness	1.5 μm
Nominal gap at rest	2.0 μm
Main spring length	320.0 μm
Folded spring length	273.0 μm
Spring thickness	1.5 μm
Proof mass	6.1 e^{-09} kg

Table 4.2: Modal analysis realized in *MEMS+*

Mode	Description	Frequency (Hz)
1	In-plane translation (x-axis)	1 620
2	In-plane translation (y-axis)	1 620
3	In-plane rotation	1 976
4	Out-of-plane translation	7 178
5	Out-of-plane rotation (x-axis)	8 849
6	Out-of-plane rotation (y-axis)	8 873

Based on the above 3-D model, a reduced-order model is exported from *MEMS+* and encapsulated in a TDF module. The modes of interest, i.e., the **x**- and **y**-translations, are preserved by the reduction. To solve the state-space system, we follow either the SystemC-AMS standard or the proposed MOR implementation. The model is weakly nonlinear regarding the translational acceleration. This nonlinearity is reconstructed by an order-two interpolation on $\mathbf{a_x}$ and $\mathbf{a_y}$ inputs.

The inputs and outputs of the model represent both mechanical and electrical variables. The acceleration in both **x**- and **y**-axis is assumed to evolve in the range 0 g to 1 g. The driving and sensing voltages are considered as inputs. The driving voltage is fixed at 4 V. The sensing voltage is a sinusoidal signal biased at 4 V with a small amplitude of 0.025 V and oscillating at a frequency of 105 378Hz which corresponds to the resonance of the beam. The main outputs are the capacitances measured on each resonator. The displacement of the oscillating beams is also retrieved.

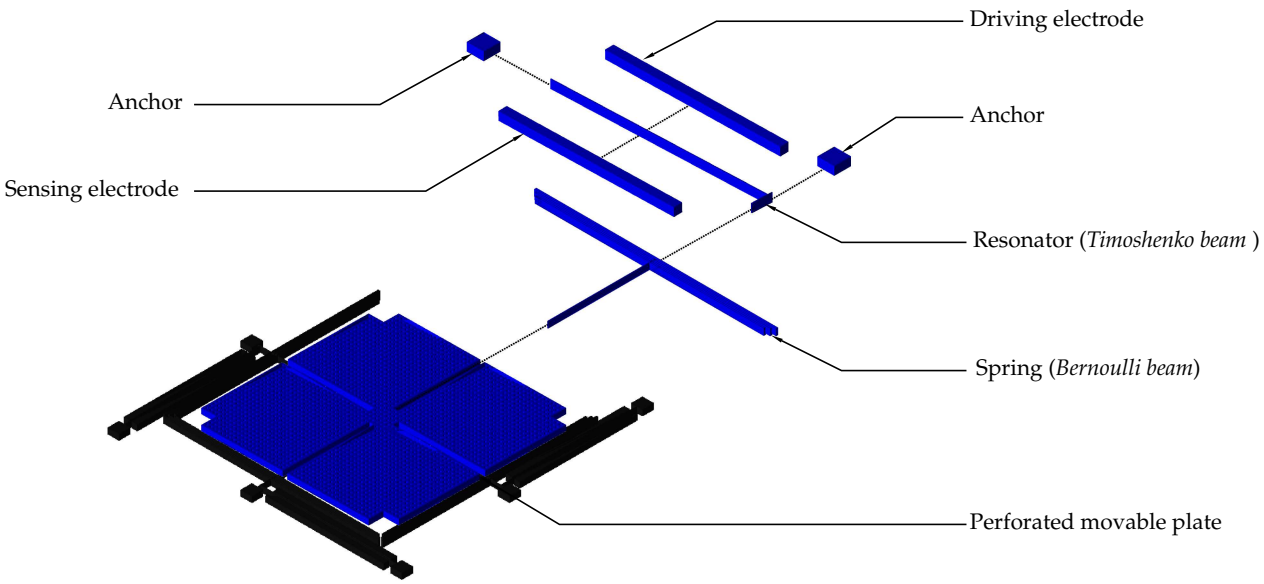
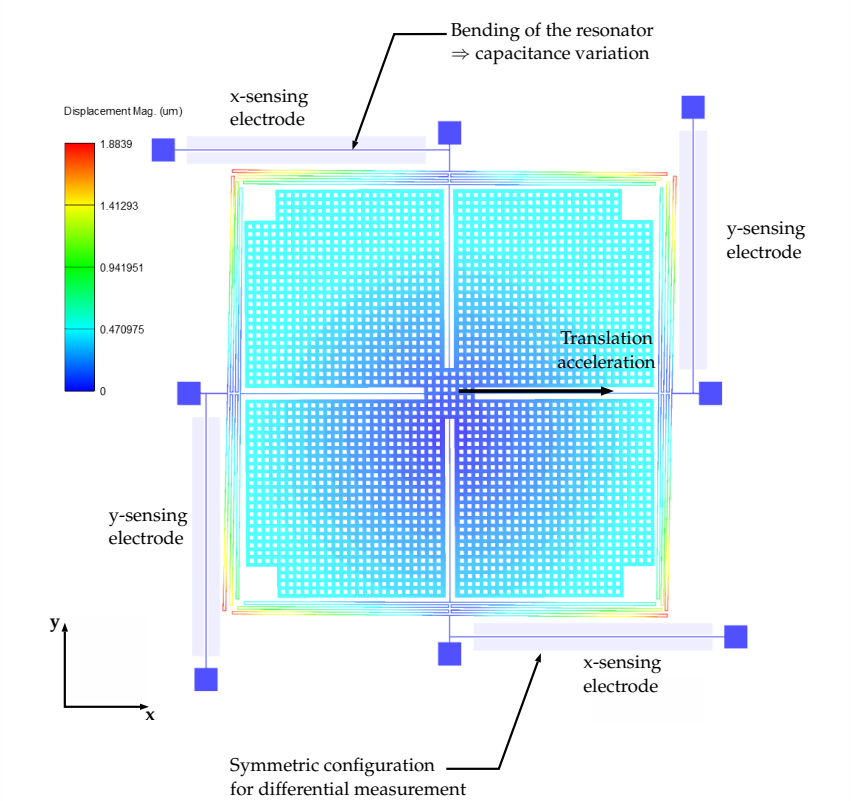
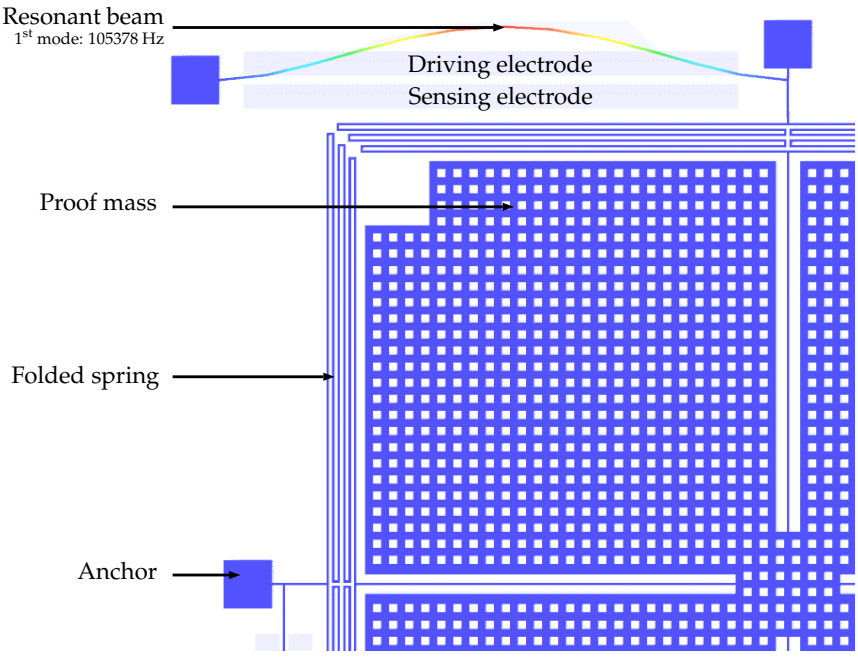


Fig. 4.2: Exploded diagram of the different *MEMS+* components used in the accelerometer ST LIS332AR.



(a) In-plane translation on **x**-axis sensed through the bending of the resonator.



(b) *L-shaped* resonator first mechanical mode used for sensing application.

Fig. 4.3: The principle of the biaxial accelerometer ST LIS332AR is based on a differential capacitive sensing. To this end, complementary external resonators are actuated and aim to sense translation acceleration. Here, an acceleration is applied in **x**-axis and moves the central plate laterally in the positive direction (a). This implies the bending of the resonant beam as illustrated in (b) for the first mode ($105\,378 Hz$).

Implementation

This version differs from the final implementation of the API presented in Section 4.3. The current export of the reduced model was realized from *MEMS+* by writing in a dedicated C++ source file the complete set of matrices and vectors, i.e., ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{E}$, $\mathbf{r}$), defining the system as well as its structure and the related solving methods. The corresponding model was instantiated as a C++ class which inherits from the base class `sca_tdf::sca_module` as shown in appendix in Listing E.1. This module defines the input and output ports of the system. Its internal behavior is defined in the `processing()` method which runs successive operations to solve the state-space system corresponding to the encapsulated data. First, the method reads the inputs in order to update the approximation weights. The current system matrices are then interpolated trough the `update()` and `interpolate()` methods. The state-space system is then solved by the `compute()` function and the current output values are written on the corresponding ports.

As depicted in Section 4.2.1, the system is defined by a quintuple of matrices ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$, $\mathbf{E}$). To reconstruct the nonlinearity, an interpolation is realized between k snapshots, each corresponding to the steady state of the system around a specific operating point, i.e., a matrix set, and where n is the order of approximation set by the user. In order to refine the model, the user may increase the approximation order n may which defines the number of operating points. With the use of the D-pointer pattern, this can also occur without changing anything to the structural description of the system, i.e., the number of inputs, states and outputs. Therefore, to preserve the structure and avoid unnecessary operations, we decoupled the data from the module.

To separate the data from the module definition, we applied the private data pattern, also known as opaque pointer. In C++, a common use of opaque pointer is the D-pointer which consists in a private data member of the class pointing to an instance of a structure, here the class that contains the different snapshots of the system (Figure 4.4). The D-pattern hides the class implementation from the user and protects the class state by minimizing the visibility of its attributes. For instance, the matrices and all related operations are defined and declared in the `Data` class, while the user will only use the `ModelExport` instance. The corresponding header file (Listing E.1) needs only to contain the declaration of the exported model and includes those other files needed for the class interface.

Experiment

In this case study, we are interested in two main tests. First we evaluate the accuracy of the reduced model with regard to the full description simulated in MATLAB/Simulink. We then investigate the performance of the reduced model simulated in SystemC-AMS in comparison to MATLAB/Simulink. The configuration of the different models used in this case study are indexed in Table 4.3. In the following, each model is denoted by its index and is implemented in the test bench defined in Figure 4.5.

In the one hand, we want to assess the reduced model exported from *MEMS+* is conformed to its full definition. The full model, i.e., Model 1, is created in *MEMS+* and instantiated in MATLAB/Simulink. In the following, Model 1 serves as reference in terms of accuracy since the modeling method used in *MEMS+* has been previously demonstrated with regard to fully meshed solutions like *CoventorWare* [78]. The reduced model is first exported in the *MEMS+* file format (*mrom*) first elaborated to import reduced models into MATLAB/Simulink. In Section 4.3, we propose to reuse this format in order to improve the presented solution

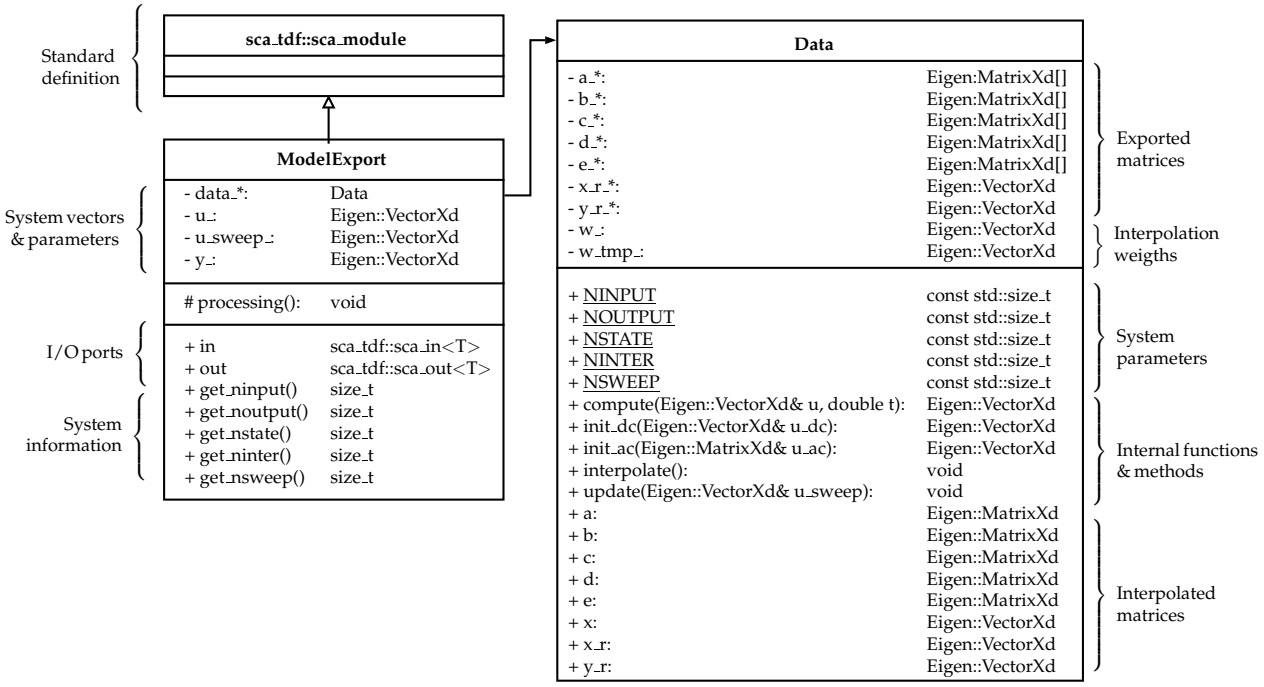


Fig. 4.4: UML diagram of the private data class pattern using an opaque pointer, i.e., D-pointer, in order to decouple the `Data` class instance that contains the model data from the main class which defines the model structure.

Table 4.3: Index of the models of the accelerometer LIS332AR in the different simulation environments.

Index	Simulation environment	Model	Integration scheme
Model 1	MATLAB/Simulink	Full	ode23
Model 2	MATLAB/Simulink	Reduced	ode23
Model 3	SystemC-AMS (standard)	Reduced	Euler-backward
Model 4	SystemC-AMS (modified)	Reduced	Crank-Nicolson

and thus ease the import in SystemC-AMS through a dedicated C++ API. The reduced model instantiated in MATLAB/Simulink is denoted Model 2 and aims to verify the correctness of the MOR technique. Model 2 serves as reference in terms of speed since it is the fastest model we can run in MATLAB/Simulink. Note that the transient simulation run in conventional FEA code would run hours or days if at all possible due to the high number of elements and solver-coupling. The two aforementioned models are computed in MATLAB/Simulink following the integration scheme `ode23` which corresponds to the Runge-Kutta method of order 2 used to solve non-stiff differential equations [156].

In the other hand, we use the reduced model in SystemC-AMS in order to compare the simulation performance to the-state-of-the-art simulator, i.e., MATLAB/Simulink. A first TDF module, denoted Model 3, is implemented in the SystemC-AMS standard. Model 3 solves the underlying state-space system through the `sca_tdf::sca_ss` class. As exposed in Section 4.2.1, the system matrices need to be updated at each time step of the simulation in order to handle the nonlinearity of the system. The SystemC-AMS standard does not support any algebraic operation. For example, there is no available matrix-vector product be-

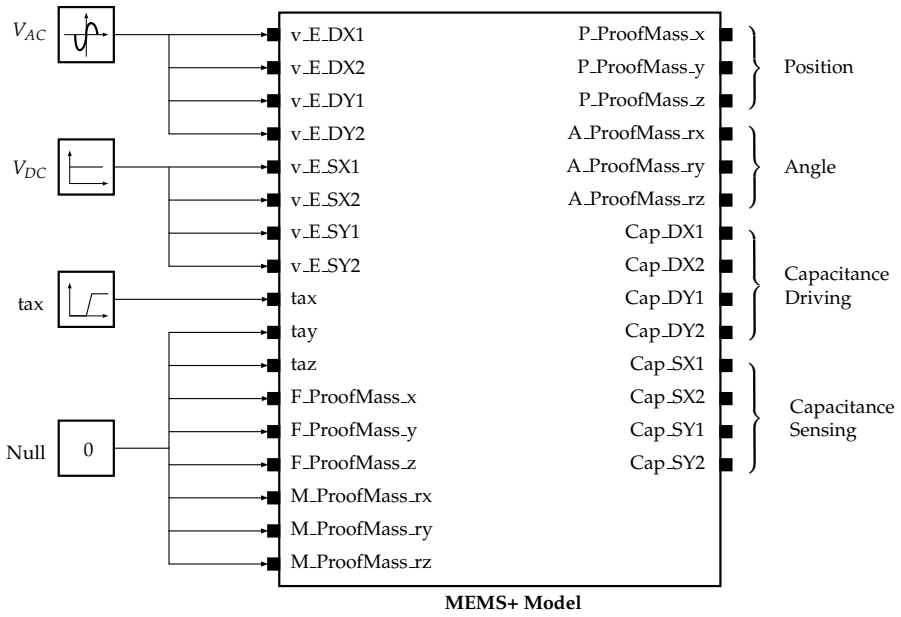


Fig. 4.5: Definition of the test bench implementing the *MEMS+* model of the biaxial accelerometer ST LIS332AR. In MATLAB/Simulink, Model 1 refers to the full model while Model 2 encapsulates the reduced model through a dedicated format file. In SystemC-AMS, Model 3 and Model 4 use the reduced model and are defined as TDF modules respectively based on the standard or modified solving method for state-space systems.

tween the `sca_util::sca_matrix` and `sca_util::sca_vector` classes. To overcome this limitation, we use the linear algebra library *Eigen* [193]. We provide dedicated conversions between the *Eigen* and the SystemC-AMS formats in order to use the standard class `sca_tdf::sca_ss`. As an alternative, we propose a modified version, denoted Model 4, that does not require to directly solve the state-space system by using only the *Eigen* library.

4.2.3 Results

The simulation is based on the test-bench defined in Figure 4.5 which connects the different models listed in Table 4.3 to the corresponding voltage and acceleration sources. It intends to verify the mechanical response of the accelerometer to a ramp impulse in acceleration against x-axis with an amplitude of 1 g.

The study first compares Model 1 against Model 2 in order to validate the accuracy of the MOR method. The relative error of the simulation is relatively low ($< 1\%$) and is assumed to be induced by the MOR process itself. This difference can notably be observed during the stabilization phase after the impulse as shown in Figure 4.6(c) where the reduced model has the same profile either simulated in MATLAB or SystemC-AMS. The capacitance response is given in Figure 4.6(a) for all simulations listed in 4.4.

Model 2 is then simulated in MATLAB/Simulink in two configurations, i.e., Simulation 2(a) and Simulation 2(b). In Simulation 2(a), MATLAB internal solver has a great performance with a simulation runtime about 3.7s and a time step automatically adapted during the simulation regarding a specific tolerance. In Simulation 2(b), the time step is set as fixed with the same integration scheme, `ode23t`. In this case, the solver

requires a much smaller time step ($1\mu s$) and several iterations to converge to the expected solution. This shows that if the time step control is left to the user, a sub-optimal choice can lead to huge performance decrease.

Model 3 has comparable performance to Model 2 in terms of speed and accuracy for a well chosen time step. If the selected time step is too large, the system may undergo numerical damping which has a direct impact on accuracy. The solver then fails to recover the variations and nonlinearity of the model. In contrast, a small time step would increase the number of iterations and thus lengthen the simulation runtime. Finding an acceptable time step is a task that should be automated with regard to an estimated error as depicted in [202]. Moreover, the integration scheme of SystemC-AMS (Backward Euler) limits the time-step size and directly impacts the performance of the simulation.

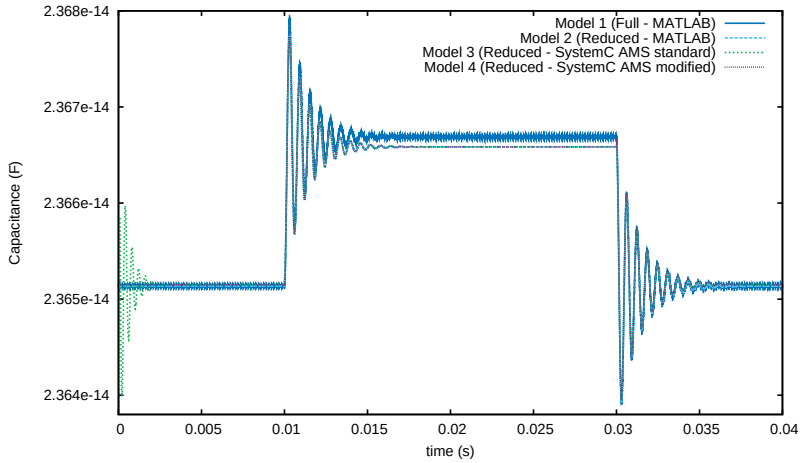
Our modified version in TDF (Model 4) demonstrates better results than the standard implementation of Model 3. Simulation 4 appears twice faster than Simulation 3 with the same time step, here $10\mu s$. This is mostly due to the use of *Eigen* library [193] to perform matrix-vector operations and solve the linear system. The conversion between Eigen format and SystemC-AMS format realized in Model 3 is indeed time consuming and would encourage for a better support from the standard for algebraic operations.

In Model 3, the solution oscillates at the beginning of the simulation as shown in Figure 4.6(b). These instabilities are due to the implicit integration scheme, i.e., the Backward Euler method, implemented in the standard. In Model 4, we therefore use an explicit scheme, i.e., the Crank-Nicolson method [156], which stabilizes the solution and prevents from non-physical oscillations. With an appropriate integration scheme and a proper use of dedicated libraries, solving state-space system in SystemC-AMS appears as an efficient and competitive alternative to traditional solvers, as demonstrated by the result of Simulation 4.

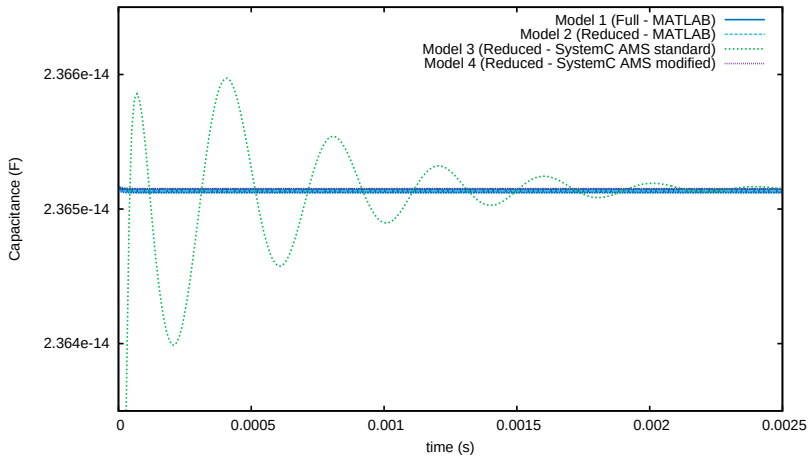
The above results highlight the ability of SystemC-AMS to correctly handle continuous-time systems with a discrete-event simulator. Nevertheless, SystemC-AMS still lacks proper time step control since the user is responsible for its definition which is a cumbersome task. This implies a lot of trials before finding a suitable fixed time step and may directly impact the simulation performance. The standard does not support linear algebra operations despite the definition of specific matrix and vector entities. We elaborated a modified version (Model 4) based on the external library *Eigen* [193] in order to get faster, more accurate and more stable results than the standard implementation initially proposed in Model 3.

Table 4.4: Simulation results for a ramp impulse in translation acceleration in **x**-axis (Amplitude: 1g)

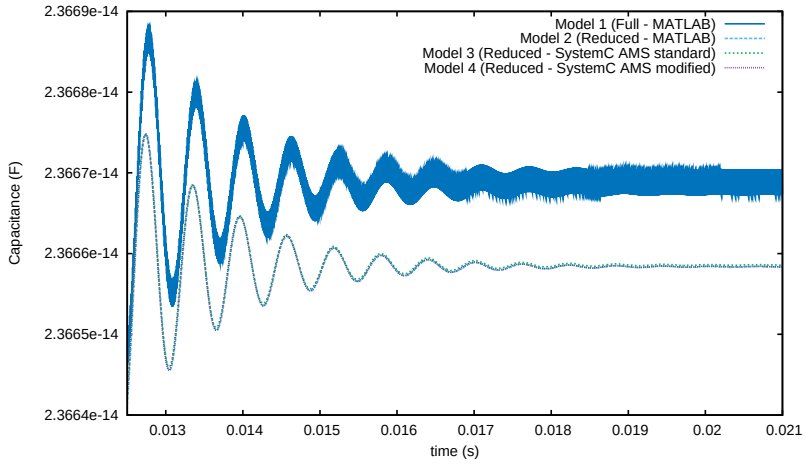
			Time step			
Environment	**Index**	**Model**	Type	Max. (μs)	Iterations	Time (s)
MATLAB/Simulink	Simulation 1	Model 1	variable	800.0	5 000	2 318.890
	Simulation 2(a)	Model 2	variable	800.0	43 893	3.725
	Simulation 2(b)	Model 2	fixed	1.0	400 000	797.036
SystemC-AMS	Simulation 3	Model 3	fixed	10.0	4 000	4.088
	Simulation 4	Model 4	fixed	10.0	4 000	1.833



(a) Global response to acceleration impulse.



(b) Initialization difference between models.



(c) Damped oscillation before steady state.

Fig. 4.6: The results above display the different results obtained with full and reduced models. Figure 4.6(a) highlights the numerical instability of the standard implementation at initialization. Moreover, Figure 4.6(b) compares the peak response observed at for each model and the delay observed in SystemC-AMS.

4.3 API Implementation

Directly interfacing *MEMS+* reduced models appears as a robust alternative to the SystemC-AMS standard solving methods to integrate and test MEMS devices in larger systems. The first implementation introduced in Section 4.2.1 has three main disadvantages. First, the system matrices need to be reconstructed and approximated at each time-step during the simulation through a specific evaluation of the current input values. Second, this approximation method as well as the initial value of system matrices can be directly accessed by the user. In order to preserve and protect the model, these data should be encrypted. Third, the export was not generic enough to refer to a standard set of functions and features. To address the aforementioned problems, we detail below the API implementation for the simulation of *MEMS+* reduced-order models in SystemC-AMS. In the following, we assume these models are built with respect to the MOR technique depicted in Appendix C.

The proposed API relies on a C++ dynamic library, i.e., a code package that aims at reusing the implemented functions and procedures in many programs. C++ libraries contain both header files and pre-compiled libraries. A header file (*.h*) defines the functionality of the library and is exposed to the program using it. The pre-compiled library contains the implementation of that functionality and is pre-compiled into the language. Libraries are of two kinds, namely static and dynamic.

A **static library** is a set of routines that are compiled and linked directly into the program. At compilation of a program using a static library, all the functionality of the static library becomes part of the executable. The advantage of static libraries is to deliver an executable that is easily shareable with other users. The downside of this approach is to copy the entire library in every executable which wastes memory space. Moreover, once the executable is compiled, it can not be easily upgraded to more recent versions. The executable needs also to be replaced in order to update the library. From a software production viewpoint, dynamic libraries offer more flexibility.

A **dynamic library** defines a set of routines that are loaded into the targeted application at run time. When compiling a program using a dynamic library, the library does not become part of the executable, but remains a separate unit. This approach enables multiple programs to share only one copy of the library, thus spare memory. Furthermore, the program can be updated to a newer version without replacing all of the executables that use it. Dynamic libraries are not linked into the program, but programs using dynamic libraries must explicitly load and interface the dynamic library. This can be confusing and requires a careful interfacing with the dynamic library, usually automated. The import of library therefore relies on an automated process to load and use the related dynamic library.

In C/C++, libraries rely on the declaration and definition of variables and functions. A declaration introduces a name and its type to the compiler without allocating any memory for it. In contrast, a definition provides details of a type's structure or allocates memory in the case of variables. The primary reason for creating an API is to hide any implementation details so that these can be changed without affecting existing clients [203]. There are two main categories of hiding techniques: physical and logical hiding. On the one hand, physical hiding means the private source code is simply not available to users. This essentially relies on storing internal details, i.e., definitions, in a separate file (*.cpp*) from the public interface (*.h*), i.e. declaration. On the other hand, logical hiding entails the use of language features to limit access to certain elements of the API, i.e., using `public`, `protected` and `private` declarations in a class.

APIs define libraries that are considered as minimally complete since they implement a limited set of functions and classes accessible through a unique entry point. An interface aims to separate the structural definition of core elements from their implementation. To access internal data, different methods are applied like callback functions, observers or notifications [203]. These functions enable to hide specific procedures or data structures from the users and thus prevent any external misuse. Furthermore, the implementation of APIs apply the principles of software architecture which mainly rely on pattern definition [204, 205]. APIs such as QT [206] or Eigen [193] provide a good illustration of well structured and documented libraries in the C++ language.

The targeted C++ interface between SystemC-AMS and MEMS+ aims to generate a system-level compatible model of MEMS devices. At system-level, the objective of the simulation is to consider a component, or entity, with regard to the design parameters. The device is virtually tested as illustrated in Figure 4.7. The corresponding test bench aims to verify different configurations of the device through specific stimuli sources. For instance, MEMS devices are characterized with regard to their response to static impulse, small oscillations and transient behavior. The simulation results can further be compared with a reference model on which a monitor inspects the behavior response through different analyses. A potential asset of SystemC-AMS is its further coupling with verification frameworks like UVM as illustrated in [60]. In the following, we focus our efforts on the definition of test benches for MEMS devices.

Our API enables to add reduced-order models exported from MEMS+ in SystemC-AMS test benches, as shown in Figure 4.8. To this end, we provide a limited number of classes and features to address the integration and the simulation of MEMS+ models. The 3-D model initially created in *MEMS+* can be exported to a specific binary file with an *mrom* extension. This format contains the structure and data of the corresponding reduced-order model and was initially developed for *MATLAB/Simulink* to accelerate the simulation [135]. Our objective is to re-use and read the same *mrom* files directly from C++ applications like in SystemC-AMS and access some of the features already implemented in *MEMS+* through callback functions. Once compiled, the corresponding executable (*.exe*) runs the SystemC-AMS simulation and generates a result file (*.trans*) that can be imported in MEMS+ to display results on the original 3-D model.

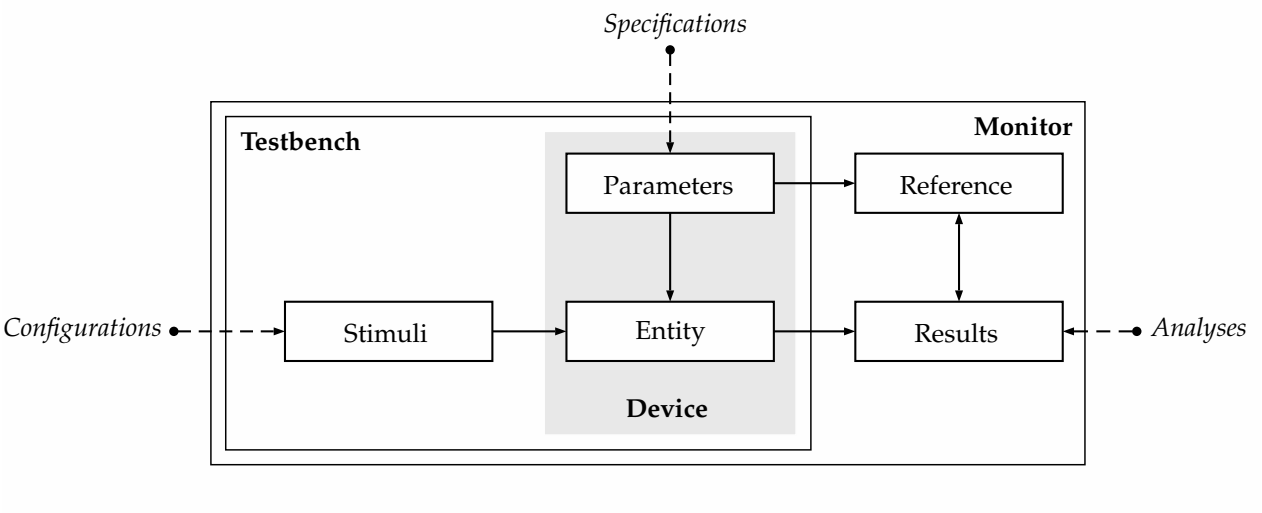


Fig. 4.7: Device under test in a system-level simulation environment.

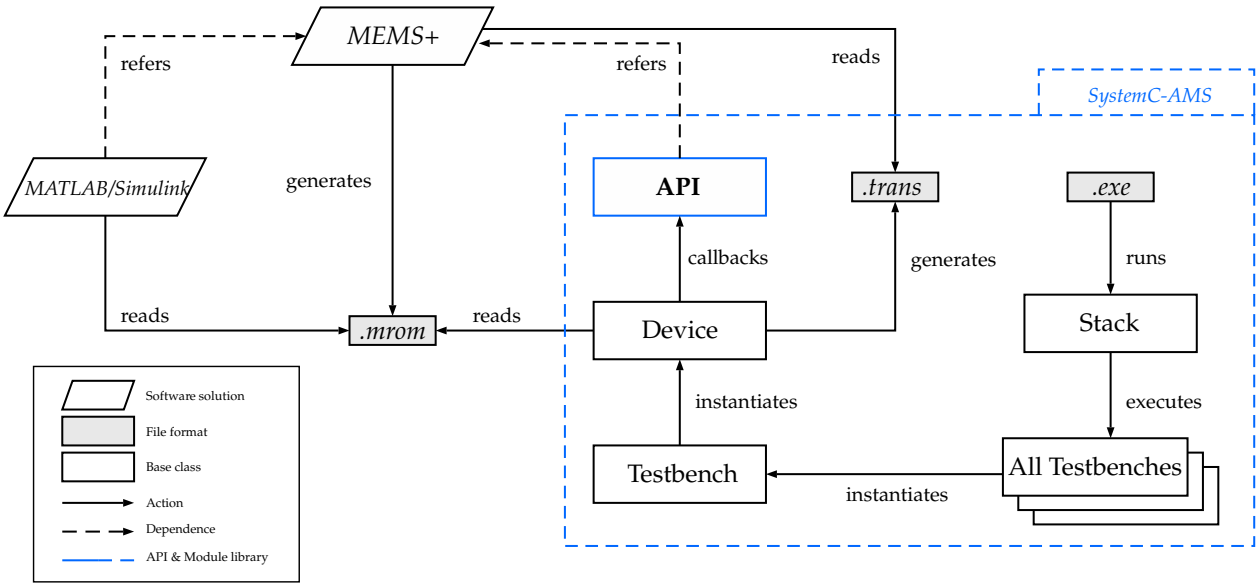


Fig. 4.8: Principle of the *MEMS+* API with SystemC-AMS and the import of *mrom* file in dedicated TDF modules. The *mrom* format can also be imported in MATLAB/Simulink.

The API core elements belong to a specific namespace `Memsplus::MROM` and are accessible through a single file (*SystemCMROM.h*) delivered alongside with MEMS+. This file contains the definition of the main class `SystemCMROM` whose UML diagram is shown in Figure 4.9. This class instantiates a private member `mromImporter_` which is a pointer to implementation, or *pimpl* [207], that refers to the *mrom* reader defined in the internal MEMS+ class `MROMImporter`. Based on object-oriented programming principles [208], the `SystemCMROM` class enables to instantiate devices either in SystemC, `ScModule`, or SystemC-AMS, `ScaModule`. The discrete-event nature of SystemC would require to define specific local clocks in order to recover the continuous behavior of MEMS devices. We therefore focus our study on the SystemC-AMS implementation in the TDF MoC through `sca_tdf::sca_module`. In order to define the input and output ports in a generic way, we use the standard class `sc_vector` [209] as data structure to dynamically implement ports. This method requires to define all the ports with same type, usually as `double`. To support the definition of quantity or unit on the different signals, conversion modules are required, as depicted in Section 4.3.3. The specific SystemC-AMS method `processing()` is specialized with the different methods required to initialize, update and solve the system, see Figure 4.11. In addition, the `set_attributes()` function supports DTDF through the `accept_attribute_changes()` function that allows to modify the time step internally or in another module during the simulation. This option is of interest to implement dynamic integration schemes with error estimation as introduced in [202]. We implemented the Crank-Nicolson method which is commonly assumed stable even with large time steps [210]. We provide in Section 4.4 an illustration of the influence of the time step on standard SystemC-AMS models compared to MEMS+ reduced models.

Although the API essentially provides a main class to correctly import *mrom* files in SystemC-AMS and define devices accordingly, we provide additional base classes to ease the edition of test benches in SystemC-AMS, especially for MEMS designers and system integrators. In the following, we introduce add-on objects of the API such as `Device`, `Testbench` or `Stack`. We also detail some of the classes and features shown in Figure 4.9 and useful to refine models, support units, or perform post-processing operations.

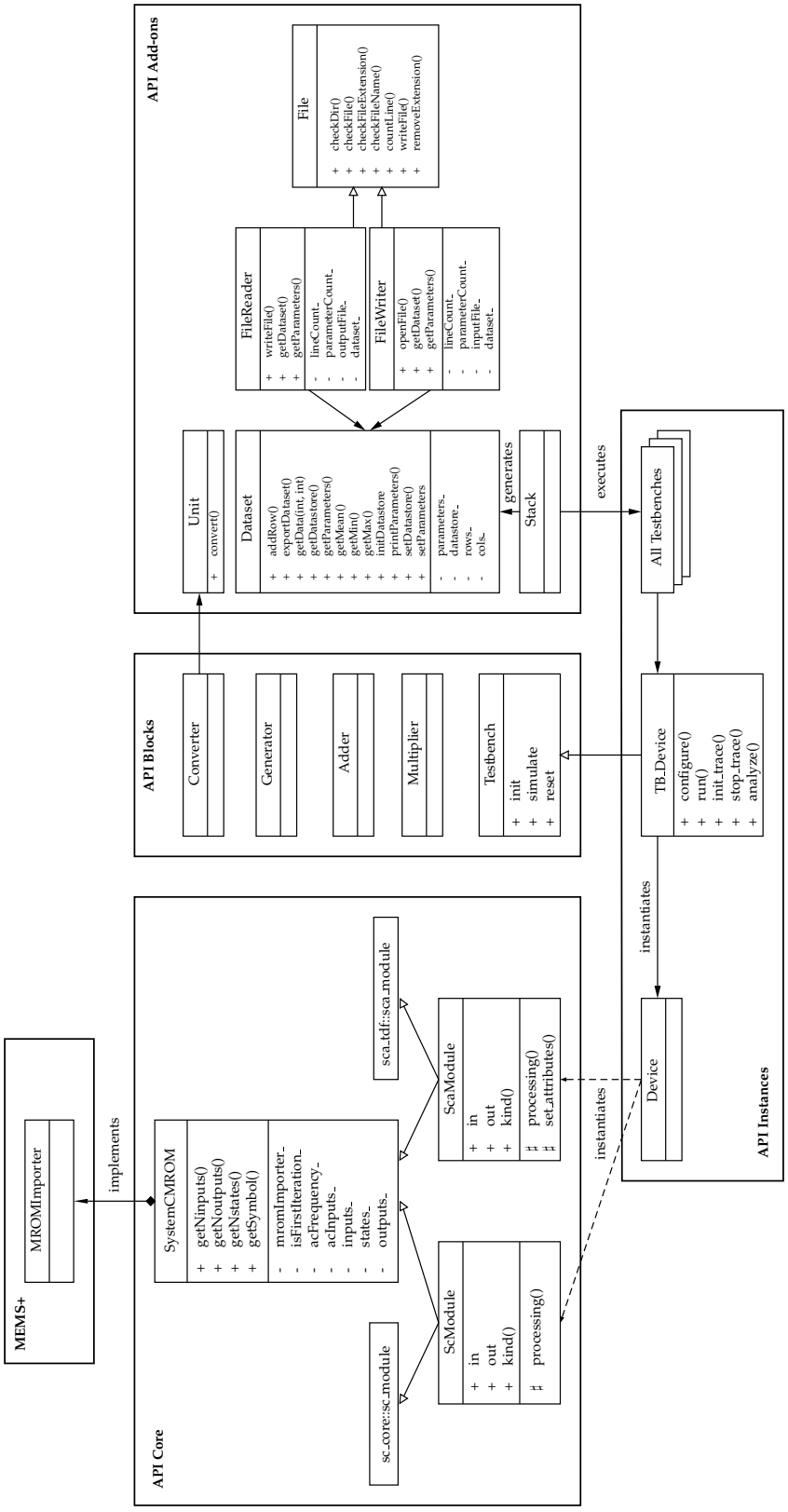


Fig. 4.9: UML diagram of *Memsplus::MROM* main classes.

4.3.1 Device

In contrast to `sc_module`, the `sca_module` class does not allow to enclose other instances of its own class [24]. In the following, we thus consider a device as an instance of `sc_module` class which encapsulates a `Memsplus::MROM::ScaModule` instance as follows:

Listing 4.1: Definition and declaration of an MROM instance in SystemC-AMS

```
template<T> // Templated definition of class allowing, e.g., the flexible typing of I/O.
class Device  : public  sc_core:: sc_module {
public:
  // Definition of I/O ports and signals
   sca_tdf:: sca_in<T> in;
   sca_tdf:: sca_out<T> out;
   sca_tdf:: sca_signal<T> *s_in;      // Dynamic definition of input signals
   sca_tdf:: sca_signal<T> *s_out;     // Dynamic definition of output signals
[...]
  // Defintion of sub-modules.
  Memsplus::MROM::ScaModule* m_mems; // Instance of the MROM module
[...]
  // Class constructor
  Device( sc_core::sc_module_name nm) {
    [...]
    std::string path = "~/tools/memsp/mems.mrom";         // Definition of the path to the MROM file.
    m_mems = new Memsplus::MROM::ScaModule("MEMS", path); // Declaration of MROM module in SystemC-AMS.
    [...]
  } [...]
}
```

The API proposes different callback functions to access the main parameters and get information on the *MROM* file. To be compliant with the initial model created in MEMS+, the device must respect the number of input and outpout ports. The number of inputs and outputs can also be retrieved through `getNinputs()` and `getNoutputs()`, respectively. For instance, they can be used to dynamically define the number of signals and the binding to the corresponding module ports as depicted below for outputs. In addition, a symbolic view can be generated outside the SystemC context by calling `getSymbol()` function which returns the map of inputs and outputs with their respective units. The output display of an *mrom* file in the SystemC-AMS simulation run-time is illustrated in Figure 4.10

Listing 4.2: Use of API callback functions

```
// Initialization of sc_vector that encapsulate a set of output ports
s_out = new  sca_tdf:: sca_signal<double>[m_mems->getNoutputs()];

// Binding of ports with output signals
for (int j=0; j < m_mems->getNoutputs(); ++j)
    m_mems->out[j](s_out[j]);
```

To correctly operate the device, the corresponding module defines sources that generate specific stimuli with regards to the model specifications. We review in Section 4.3.3 the functional definition of stimuli generators through our generic definition of `ScaGenerator` class. The initialization of the device is part of the processing method described in Figure 4.11 and aims to operate the device around an initial steady state, i.e., DC operating point, with small oscillations set by AC entries. For example, the initial oscillating state of a device can be specialized in a test-bench like in Listing 4.3.

Listing 4.3: AC analysis of the MEMS reduced-order model

```
// AC initialization for specific inputs.
// @param acInput_          : vector that defines the amplitude and phase of AC entries.
// @param drivingFrequency_ : frequency of small oscillations required to operate the device
m_mems->setAcValues(acInputs_, drivingFrequency_);
```

As defined in the standard [24], the `processing()` method of each `sca_module` is called at each time step of the simulation. In the `ScaModule` class, we implemented the algorithm depicted in Figure 4.11 which solves the Equation (2.8) by calling *MEMS+* internal functions. At initialization, the inputs are set to the values stored in the *mrom* file, and both DC and AC analyses are performed on the device to switch it on. The solving of the state-space system is realized through the following process. First, the input ports are read and the corresponding values stored in the private `inputs_` vector of the `Device` instance. Then, the matrix system is updated with regard to the current input and previous state values. The evolution equation, i.e., the differential relation between the state vector $\mathbf{x}$ and the input vector $\mathbf{u}$ is solved with regard to an integration scheme, here the Crank-Nicolson method. The outputs are computed by solving the trajectory equation, i.e., the linear relation between the state vector $\mathbf{x}$ and the output vector $\mathbf{y}$. The corresponding values are finally written on the related output ports and optionally exported to a *trans* result file.

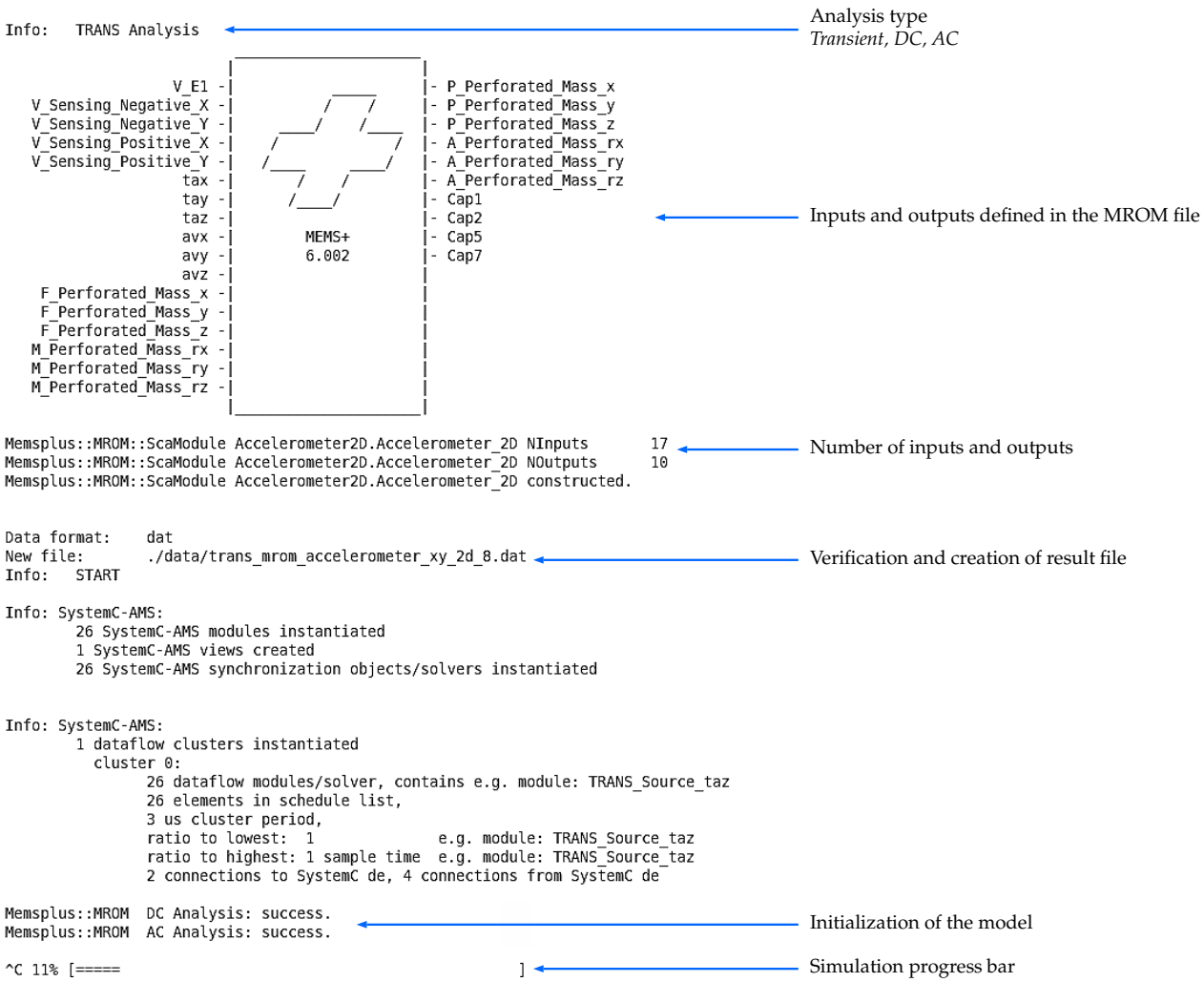


Fig. 4.10: SystemC-AMS runtime environment associated to MROM devices.

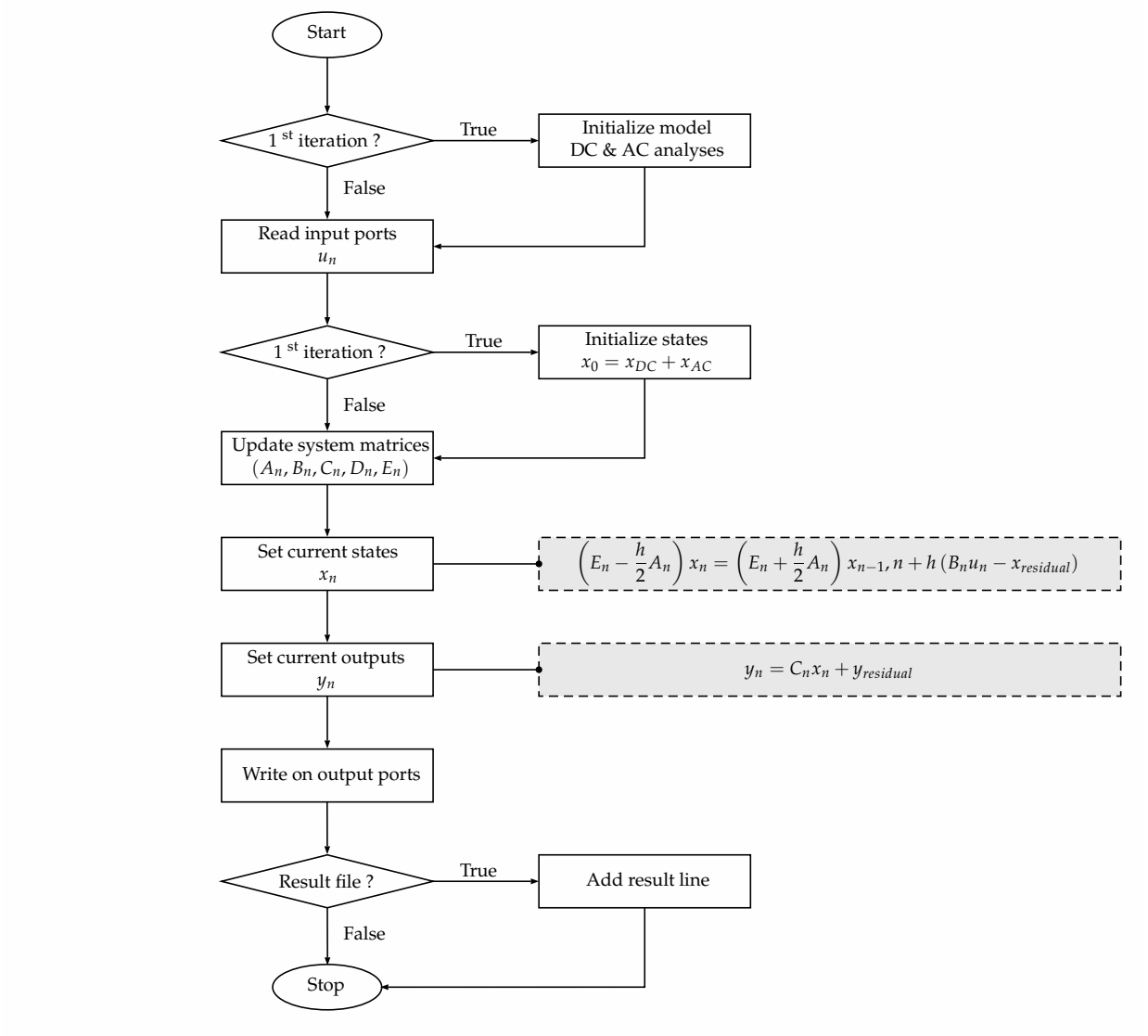


Fig. 4.11: Processing algorithm associated to MEMS+ MROM devices.

4.3.2 Test bench

In SystemC-AMS, any device is integrated into a test bench which is usually defined in the `sc_main` environment and run through a unique executable. Since `sc_main` refers to a specific configuration of a test bench, we propose a more generic definition in order to enable the recursive execution of multiple analyses on the same SystemC model. Listing 4.4 illustrates below the main functions inherited from the base class `Memsplus::MROM::Testbench` to perform a single analysis.

Listing 4.4: Basic simulation steps encapsulated in the base class *Memsplus::MROM::Testbench*

```
1    init_trace("./data/results.dat"); // Check the correct naming and create the result file.
2    simulate();                       // Run the simulation.
3    stop_trace();                     // Close tracefile.
4    reset();                          // Reinitialize the simulation context.
```

The `simulate()` function hides from the users the different steps of SystemC simulation, e.g., `sc_start()`, `sc_stop()`, and allows the designer to focus on the test-bench definition. In order to run multiple analyses on the same test bench, we need to call the `sc_main` environment several times within the same executable. Note that the SystemC model underlying a test bench refers to a particular structure and execution process registered in the `sc_simcontext` object. For more details on this function, we refer the reader to the SystemC standard [56]. The `Testbench` class enables to reinitialize the simulation context as follow:

Listing 4.5: Testbench reset function

```
1    void Testbench::reset() {
2      // Check if the simulation has been stopped.
3      if (not  sc_core::sc_end_of_simulation_invoked())
4         sc_core::sc_stop(); //< invoke end_of_simulation() function.
5
6      // Reset SystemC simulation context and allows the launch of further analyses on a test bench.
7       sc_core::sc_curr_simcontext = new  sc_core::sc_simcontext();
8       sc_core::sc_default_global_context =  sc_core::sc_curr_simcontext;
9
10     // Notification.
11     std::cout << "\nInfo:\t" << "RESET SIMCONTEXT" << std::endl;
12   }
```

The `Testbench` class also enables to define and simulate a test bench in different configurations. In the case of MEMS devices, we are mainly interested in three different analyses, namely the DC, AC and transient analyses. The DC analysis aims at characterizing the response of the system around a steady state defined by a specific operating point. The AC analysis is focused on the frequency response of the system. Finally, the transient analysis characterizes the behavioral evolution of the system over time, e.g., the response to an impulse or the linear evolution of a control input. The above analyses, i.e, DC, AC and TRANS, are declared as follow:

Listing 4.6: Analyses related to the base class *Memsplus::MROM::Testbench*

```
1    enum class AnalysisType {DC, AC, TRANS};
```

Since the previous analyses only induce small changes, the `AnalysisType` enable to easily define and overspecialize components dedicated to each analysis directly in the test-bench definition. Furthermore, to switch between different test benches, we provide a simple overwhelming class, denoted `Stack`, in which the `run` function executes test benches indexed by the user. As an example, the implementation of the accelerometer studied in Chapter 5 is given in Appendix F.

4.3.3 Add-ons

To complete the definition of test benches, our API provides some elementary blocks to build the whole systems in the TDF MoC. Mostly based on functional programming [211], the following base classes support the generation, the modulation and the unit typing of SystemC-AMS signals. We also provide some utility tools to further exploit simulation results.

Stimuli generation

The class `Generator` defines a module with an internal function, e.g., of time, producing a signal which is then transmitted to either SystemC or TDF instance through a unique output port. The proposed solution separates the general definition of the function from its execution in DE or TDF MoC, as shown in Figure 4.13. The `Generator` class has a template definition to type the output value of the function and automatically convert the output signal to the corresponding quantity. Moreover, the functional implementation of stimuli profiles let the user free to add profiles to the ones already provided and illustrated in Figure 4.12.

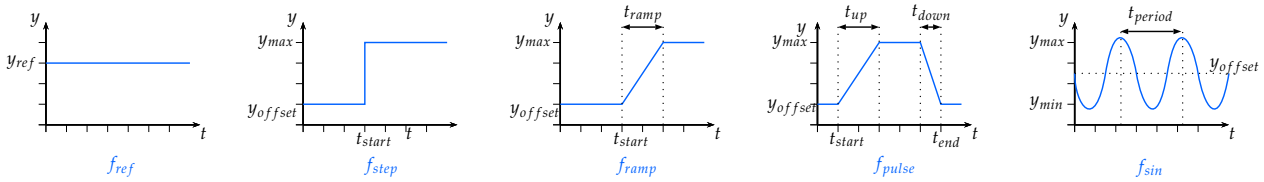


Fig. 4.12: Supported stimuli profiles.

`ScGenerator` is a SystemC module which samples the stimuli profile with regard to a specific time period. Similarly, the `ScaGenerator` is a `sca_tdf::sca_module` instance which gets the current time value when activated and generates the corresponding value of the function in a discrete-time manner. We setup two additional objects, i.e., `DCScaGenerator` and `ACScaGenerator` in order to respectively ease the DC and AC analyses of MEMS devices. These modules generate signals with regard to specific amplitude or frequency steps at a fixed-time period. This aims to characterize the response of the device at specific steady states or small oscillations.

Stimuli modulation

The `Multiplier` class allows to modulate any input signal by multiplying it with a specific function. Moreover, the template definition of the inherited objects from the `Multiplier` class allows to convert the output signal to a specific quantity. The `ScMultiplier` is a `sc_module` instance and supports the modulation of discrete-event signal. The `ScaMultiplier` is a TDF module which multiplies its entry by a variable gain which can optionally be converted into a specific physical quantity as shown in Figure 4.15(b). To enable the modulation of discrete-event signals to continuous ones, we introduce the `ScaDeInMultiplier` that instantiates an input converter port from DE to TDF MoC. Conversely, the `ScaDeOutMultiplier` handles initial continuous signals and convert them to discrete-event ones.

Based on the same principle of the `Multiplier`, the `Adder` class enables to add a specific time function to any input value. This allows to define gains or regulation modules by using different stimuli profiles. Besides simple operations like multiplication or addition, the `Integrator` class proposes a first implementation with the Crank-Nilcolson method. The functional definition of this class allows the user to choose other integration schemes. Furthermore, the DTDF option will allow to use nonuniform time steps and set a dynamic error checking during the simulation to ensure the convergence of the solution. The API base classes provide a flexible way to create block diagrams in the TDF MoC and ease the configuration of MEMS devices in SystemC-AMS.

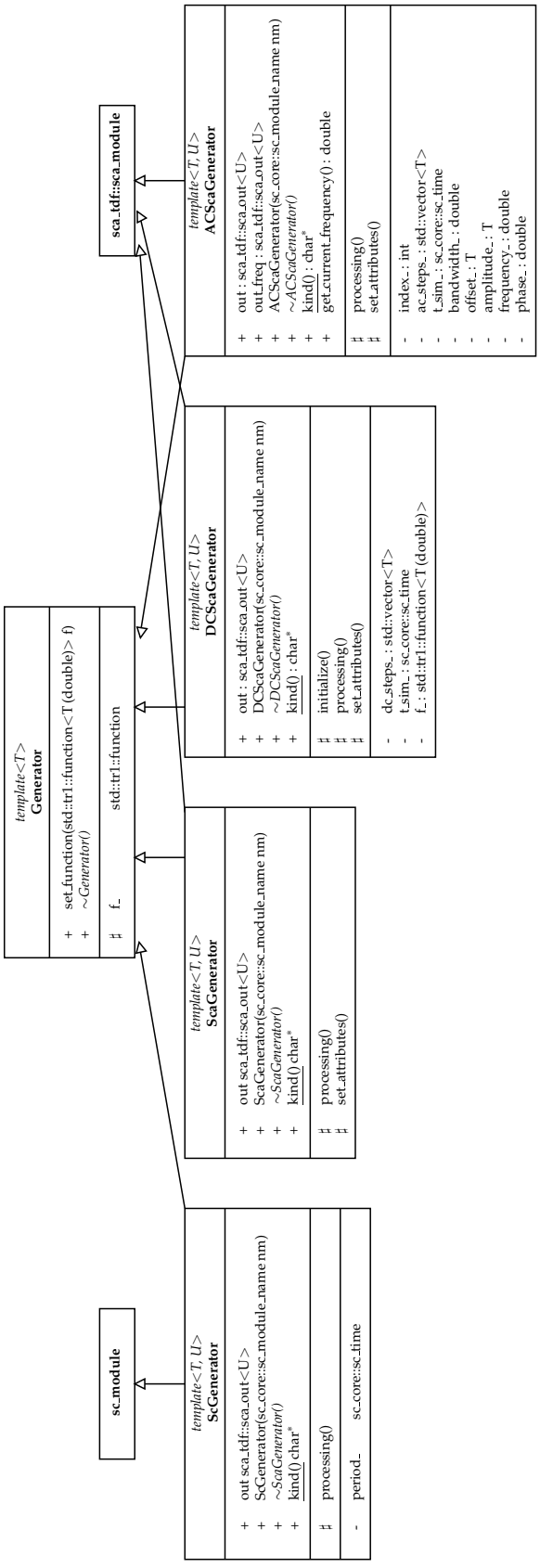


Fig. 4.13: Generator base class and inherited objects.

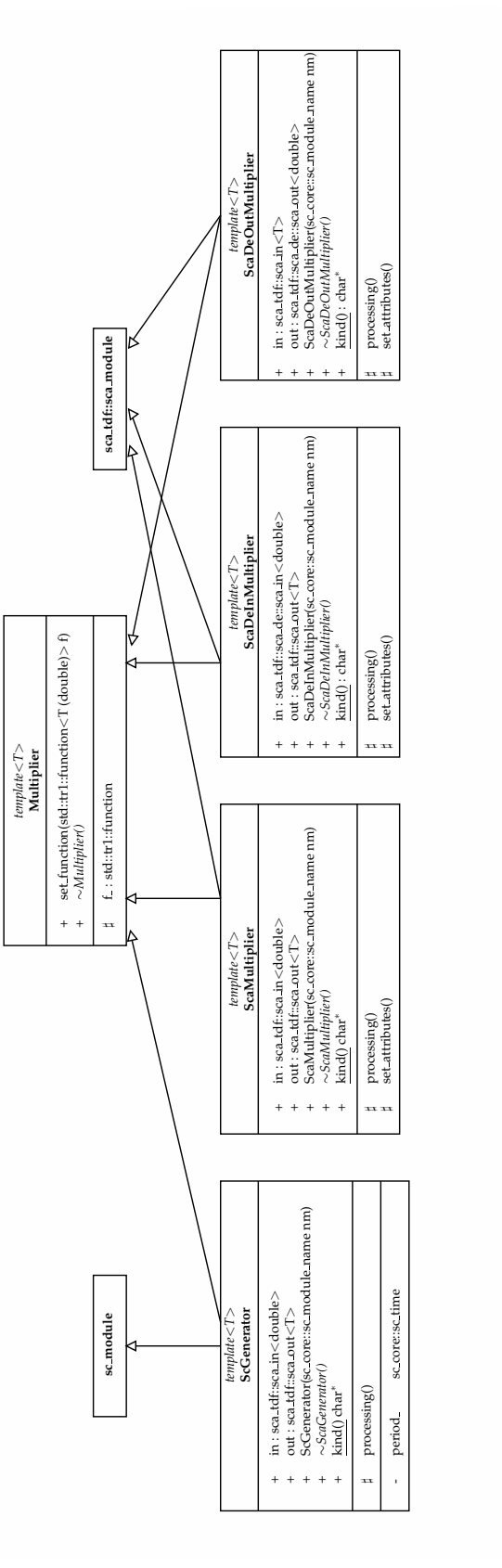
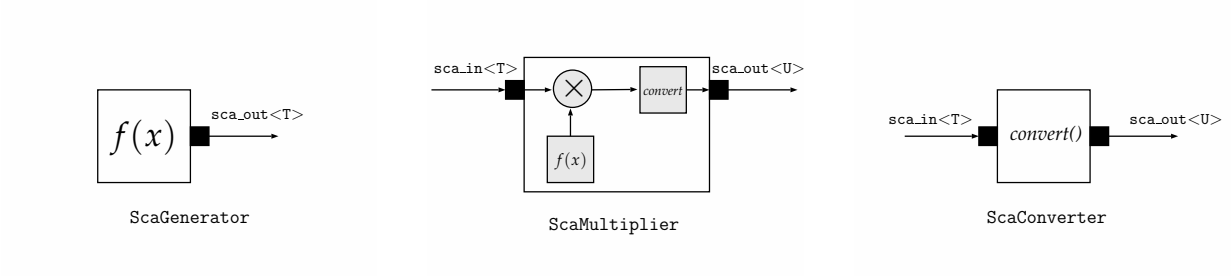


Fig. 4.14: Multiplier base class and inherited objects.



(a) Stimuli generation. (b) stimuli modulation. (c) Unit typing or conversion.

Fig. 4.15: API block modeling topology.

Physical quantities and units

As introduced by Mähne in [212], SystemC-AMS allows to use external libraries to support dimensional analysis. We also use the *Boost::Units* library [213] to annotate signals with units. This library relies on physical quantities, i.e., physical domains with their corresponding units. This allows to verify the coherency of typed signals and perform operations on them. We instantiate a list of reference types defining physical quantities. Moreover, the `Unit` class enables the automatic typing of values through the `convert()` function that adds the corresponding units to the selected signal. This feature is by default included in the aforementioned classes, i.e., `Generator`, `Multiplier` or `Adder`, and is specifically realized by the `Converter` class.

The `Converter` class is a `sc_module` or `sca_module` instance that automatically adds the unit to a value associated to the selected quantity. This especially allows the typing of input or output ports of a model exported from MEMS+ since the related module has only ports typed with double values. Moreover, from a system viewpoint, the use of *Boost::Units* guarantees the coherency and the compatibility of the different typed signals connected between each other. For example, the Listing 4.7 implements a `ScaGenerator` instance which produces a temperature signal with regard to a sinusoidal profile.

Listing 4.7: Generation of a sinusoidal temperature stimuli with internal unit conversion

```
1   // Definition of the temperature type
2   typedef quantity<celsius::temperature> temperature_celsius_type;
3   [...]
4
5   // Instance of signal handling a temperature value.
6    sc_core:: sc_signal<temperature_celsius_type> s_temperature;
7
8   // Sinusoidal stimuli profile defining the variation.
9   f_sin<double> sin_temperature(0, 120, 0.3, PHASE);
10
11  // Generator of the sinusoidal profile from the previously defined function and typing the output as a
        temperature expressed in Celsius degrees.
12  src_temperature =
13    new Memsplus::MROM::ScGenerator<double, temperature_celsius_type>("src_temperature", sin_temperature_);
14
15  // Write result on an output port
16  src_temperature->out(s_temperature);
17  [...]
```

Post-processing operations

The API offers the possibility to export the results in a *.trans* file, as defined in the `MROMImporter` class inherited from MEMS+. The function `addResultLine()` can also be called at the end of the `processing()` method of a `Device` instance. At the end of the simulation, the generated *trans* file can be directly opened in MEMS+ in order to display the results on the initial 3-D model and thus analyze graphically the transient behavior of the device. For instance, this facilitates the analysis of mechanical deformations or the evolution of parameters such as pressure.

By default, the SystemC-AMS standard enables to export results to different formats such as *vcd* or *dat*. To perform first analyses on such files, the `File` base class allows to load and manipulate the results through dynamic data structures, i.e., `Dataset`. In particular, the `FileReader` can load the results, extracts the list of the different recorded parameters and find characteristic values of specific parts of the dataset, e.g., minimum or mean values.

4.4 API use case

To illustrate the use of the API, we propose a first application on a dual mass gyroscope. The device consists of two perforated proof masses attached to a fixed structure through a system of beams that form a complex suspension. Figure 4.16 provides an exploded diagram of the device with the different components used in *MEMS+*. This sensor aims to detect the rotation of both masses about the **y** axis. The modes of interest are the out-of phase translations of the masses in **x** and **z** axis, i.e., the third and fourth modes of the device. They respectively represent the driving and sensing modes of the gyroscope, as shown in Figure 4.17.

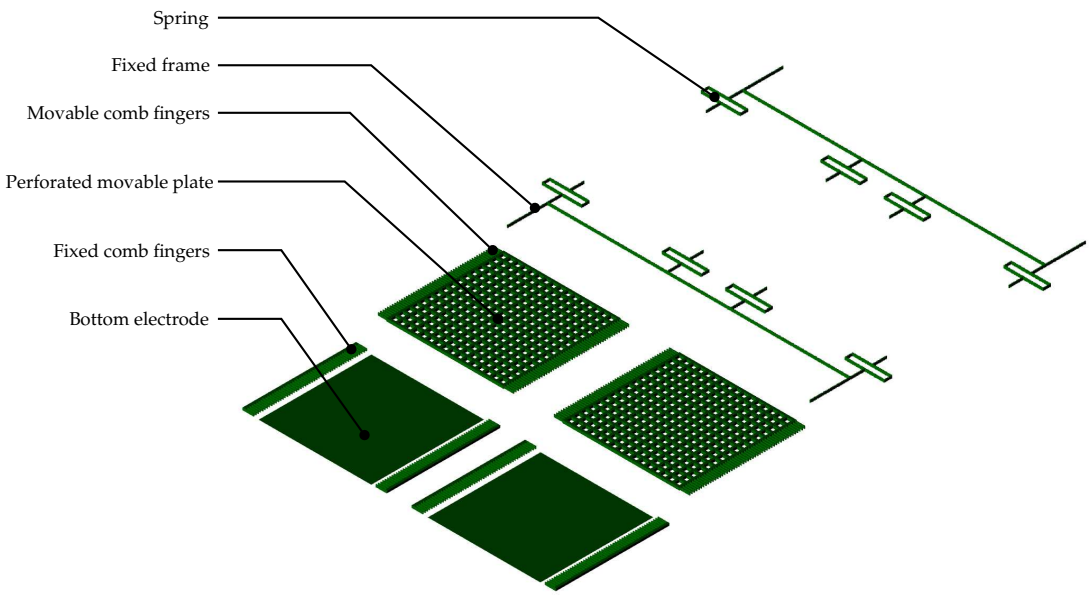
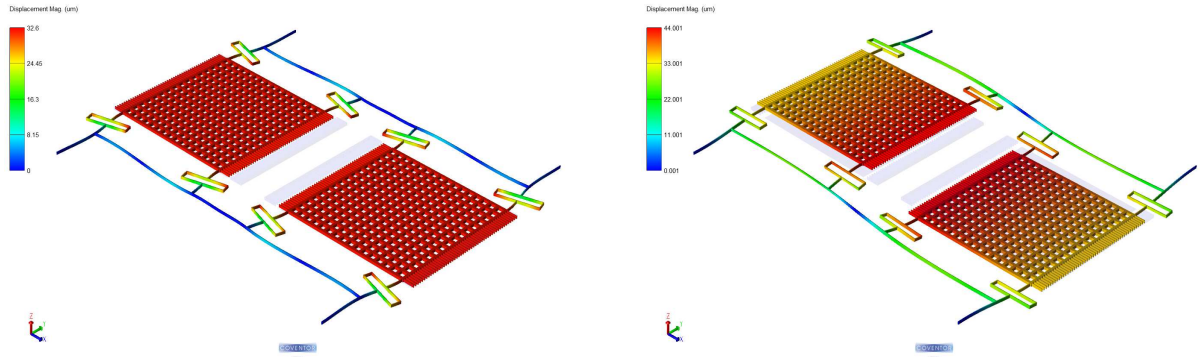


Fig. 4.16: Exploded diagram of the gyroscope double-mass.



(a) Driving mode (15 347Hz). (b) Sensing mode (15 940Hz).

Fig. 4.17: Modal analysis run in *MEMS+* and detailed for the driving and sensing modes of the gyroscope.

The gyroscope behaves as follows. The driving mode aims at maintaining the masses in oscillation in **x**-axis at constant amplitude. To this end, the comb drives are actuated with an alternative voltage at a bias of 14.7V and a resonant frequency of 15 347Hz which generates a 10μm oscillation in **x**-axis. A rotation along the y-axis will create an oscillating excitation of the sense mode, i.e., a displacement in **z**-axis. This movement is captured through the plate electrodes beneath each masses. These bottom electrodes induce capacitance variations of opposite sign, so it is possible to perform a differential measurement. The displacement amplitude is proportional to the angular velocity of the rotation thanks to Coriolis force. As introduced in Section 3.3.2, the Coriolis force, $F_{Coriolis} = -2m\dot{\theta}\dot{x}$ allows to compute the angular rate θ from the displacement in the **y**-direction. The coupling of the angular velocity, $\dot{\theta}$, and the linear one, $\dot{x}$, is responsible for the nonlinearity observed in the system response

4.4.1 Test-bench configuration

Figure 4.18 describes the test bench defined in SystemC-AMS to simulate the transient response of the gyroscope to an impulse signal. The device is actuated by the voltage sources listed in Table 4.5 generating small driving oscillations (V_{AC}) and setting up the sensing offset (V_{DC}). An impulse source of angular velocity around **y**-axis is also connected to the corresponding input (avy). These stimuli generators are all instances of the `ScaGenerator` base class introduced above. These are connected to the corresponding inputs of the reduced model exported from *MEMS+* and loaded through a `Memsplus::MROM::ScaModule` object.

Voltage source	**Bias (V)**	**Amplitude (V)**	**Frequency (Hz)**
Driving	14.7	$\pm$ 2.0	15 347
Sensing	0.1	-	15 940

Table 4.5: Actuation of the gyroscope electrodes.

Besides the integration of the reduced model in SystemC-AMS, we connect signal processing elements to the outputs of the device. First, the measurement of the capacitance is realized through a differential signal

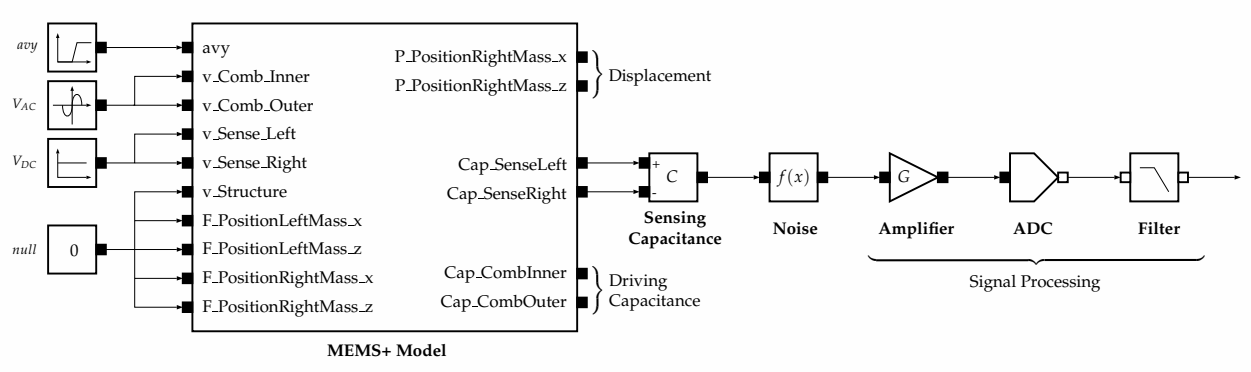


Fig. 4.18: Test-bench configuration of the gyroscope double-mass.

on which a white noise is added in order to reproduce a normal working environment. The signal is then amplified to be finally converted into 8-bit filtered signal, well fitted for further digital processing. This first application illustrates the ability of SystemC-AMS to handle signal processing and bit conversion as shown in Chapter 3. In the following, we assume that SystemC supports the co-design of the HW/SW parts and thus focus our study on the transient simulation of the MEMS device. Note that the different modules used for the signal processing are all based on a functional definition in order to reuse these elementary blocks.

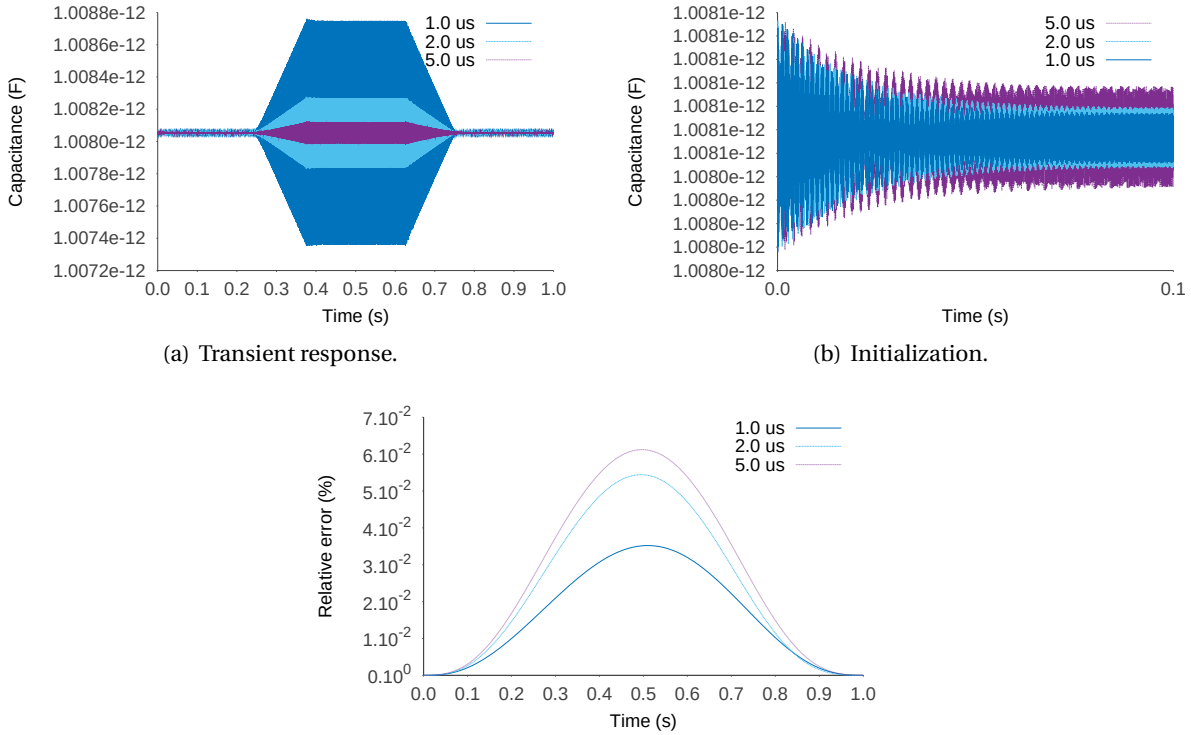
4.4.2 Simulation results

The simulation consists in generating an impulse stimuli of angular velocity around **y**-axis and measure the resulting capacitive change induced by the displacement along **z**-axis. We first verify that the reduced model is accurate enough with regard to the full description in *MEMS+*. This verification step is directly realized in MATLAB/Simulink and shows a relative error lower than 1% which validates the use of the reduced model encapsulated in the *mrom* file as reference. This difference is mainly due to the MOR process itself as verified in other formats, e.g., Verilog-A [79]. Although ensuring the model accuracy, the simulation of the reduced model in SystemC-AMS aims to provide a fast environment compared to the state-of-the-art, especially MATLAB/Simulink. Therefore the simulation run-time is considered as a major criterion.

As introduced in Section 4.3.1, we implemented the Crank-Nicolson method to solve the reduced model. Regarding the influence of the selected time step on the relative error, as shown in Figure 4.19(c), this choice may be discussed against other available methods. Explicit single-step methods are not envisioned since they do not ensure a sufficient stability in case of general linear models with unstable eigenvalues like the reduced models exported from *MEMS+*. On the other hand, the state transition matrix is a time-step independent method which is well fitted for LTI systems as depicted in [214]. Despite linearized, the reduced models may still include some nonlinearity as introduced in Appendix C. Therefore, we justify the use of an implicit scheme like the Crank-Nicolson method in order to guarantee the stability and the accuracy of the solution even for slightly nonlinear models. This implies to choose a “sufficiently small” time step, i.e., ensuring the solution remains in the region of absolute stability [158]. Further information on the eigenvalues of the main modes could be provided to let the user set a maximum time-step value.

The response of the gyroscope to an angular velocity impulse is presented in Figure 4.19(a). As introduced above, the choice of the time step influences the solution and implies a strong numerical damping due to

the fixed time stepping and the selected integration method. For instance, the initialization of the system is illustrated in Figure 4.19(b) where the stabilization of the solution is not achieved for a time step larger than $1\mu s$. Despite a longer computation time, inversely proportional to the time step, we assume the simulation with a time step of $1\mu s$ guarantees the correctness of results and can be compared to other simulators.



(a) Transient response.

(b) Initialization.

(c) Relative error.

Fig. 4.19: Simulation results highlighting the influence of the time-step selection.

4.4.3 Simulation performance

In order to compare the performance of SystemC-AMS to MATLAB/Simulink, the test bench illustrated in Figure 4.18 is configured in these environments. The simulation applies a ramp impulse of angular velocity around the **y**-axis and about 1 *rad* in amplitude. The results listed in Table 5.5 correspond to one physical second.

Table 4.6: Simulation results for a ramp impulse of angular velocity around the **y**-axis (Amplitude: 1 *rad*)

Environment	**Index**	**Model**	**Integration**	**Time step**	h (μs)	**Runtime** (s)
SystemC-AMS	Simulation 1	Reduced	Crank-Nicolson	fixed	1.0	222.65
MATLAB/Simulink	Simulation 2	Reduced	`ode23t`	variable	–	248.21
	Simulation 3	Reduced	`ode23t`	fixed	1.0	469.37
	Simulation 4	Full	`ode8`	variable	–	702.68

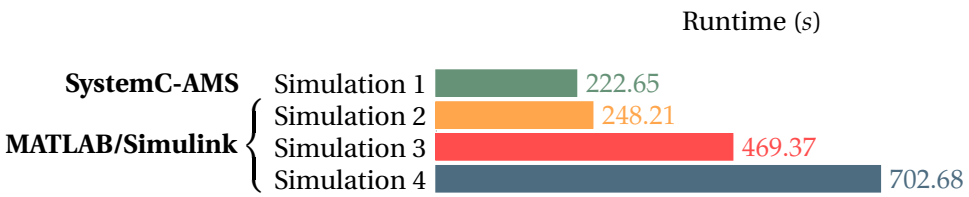


Fig. 4.20: Performance analysis between SystemC-AMS and MATLAB/Simulink when both simulations are configured with the same fixed time step ($1\mu s$).

The reduced model is run with a fixed time step of $1\mu s$ in SystemC-AMS (Simulation 1) and MATLAB/Simulink (Simulation 3). Both simulations are configured with equivalent time-stepping and integration methods. Simulation 4 performs the transient simulation of the full model in MATLAB/Simulink and serves as reference. Simulation 4 requires a finer time-step definition and cannot be computed with fixed time step by MATLAB. Simulation 3 appears almost twice faster than Simulation 4 in *MATLAB/Simulink*. This is due to the difference of DoFs to compute in the full and the reduced models. In SystemC-AMS, we observe that Simulation 1 is twice faster than in MATLAB/Simulink when imposing a fixed-time step resolution, here of $1\mu s$. In Simulation 2, we simulated the same reduced model in MATLAB/Simulink, but allows the solver to adopt a variable time step. In this case, the runtime is a bit less than $250s$ in this case, i.e., slightly slower than SystemC-AMS. The run-time of the above simulations is given in Figure 4.20.

4.5 Conclusion

The proposed API introduces a fully integrated solution for system-level simulation of MEMS devices. The evolution of the first implementation now allows system designers to fully benefit from SystemC-AMS as a co-development environment for MEMS devices connected to HW and SW sub-parts. Moreover, using *MEMS+* provides an efficient system-level modeling tool to include reduced models and thus ensure the simulation accuracy and speed. The add-on feature further help the user to be more productive in configuring and re-using its code by providing generic modules and base classes.

After verifying the accuracy and the coherence on a first use case of a gyroscope, we highlight the high capability of the reduced models to speed up the simulation, even for nonlinear systems. Moreover, the simulation in SystemC-AMS appears at least as fast as the current state-of-the-art solutions. Finally, the ability of SystemC to address software simulation sets this solution as a promising extension to complete the simulation of fully integrated MEMS solutions with the related ASIC and software sub-parts.

Chapter 5

Case study

5.1 Introduction

As argued in Chapter 3, we consider that high-level behavioral methods are too limited to correctly address the setting of MEMS and their related ASIC. This motivates the implementation of the API with *MEMS+* introduced in Chapter 4. This chapter concerns the application of the previous methodology on the 3-axis accelerometer ST SEM. This example intends to demonstrate the importance of the device geometry even at high-level and the efficient simulation in SystemC-AMS of reduced models exported from *MEMS+*.

The selected device has been tested in complementary studies during the SMAC project [215, 216, 217] and the H-Inception project [218]. Sanginario *et al.* [119] also explored the influence of the package on the device behavior, especially with regard to temperature variations. Moreover, Vinco *et al.* [219] proposed a conservative representation of the device in a dedicated system-level simulation framework based on the SystemC AMS standard.

We first describe the geometry and sensing principle of the device. We focus our study on the in-plane motion detection through the **x/y** accelerometer. An equivalent model is built in order to compare the reduced model with the SystemC-AMS standard modeling methods. The first simulation results highlight strong approximations in the high-level model considering the device as a simple movable plate. This approach is nonetheless commonly applied to accelerometers. We therefore provide an enhanced high-level representation which respects the device geometry and correctly represents its behavioral response. We detail the modeling procedure and use this model to evaluate the simulation performance of the SystemC-AMS standard with regard to the reduced model exported from *MEMS+*.

The device is integrated into a digital processing unit that reproduces the different steps of signal fitting to allow a further use in dedicated software applications. We analyze the device response to an acceleration impulse in both axes and verify the correct signal processing through the different test-bench components. Furthermore, we study the response of the device undergoing a step acceleration. We compare existing modeling procedures directly applicable in SystemC-AMS with the reduced models exported from *MEMS+*. Finally, the performance of the simulation is compared to state of the art and the full model initially created in *MEMS+*. This case study aims at justifying the use of reduced models instead of high-level descriptions that are disconnected from the design of the device. We also highlight some restrictions on the solver provided by the standard SystemC-AMS implementation.

5.2 Methods & Material

The following example aims at taking advantage of the integration of reduced models in SystemC-AMS in order to use specific output signals in dedicated software applications. We implement a test bench which performs the digital processing of the measured capacitance variation and illustrates the use of SystemC-AMS to interface HW/SW sub-parts. This study also demonstrates the limitations of an estimated representation of *a priori* simple devices like accelerometers. Furthermore, the system-level analysis conducted by Sanginario *et al.* [119] demonstrates the importance of HDLs-based simulation in the estimation of package effects on MEMS structures.

5.2.1 Device under test

The presented case study concerns the 3-axis accelerometer ST SEM fabricated in the THELMA® micromachining process [220]. The device is composed of two separated sensing elements that measure motions in the three main axes through capacitive electrodes. We focus our study on the in-plane motion in **x** and **y** axes by the accelerometer shown in Figure 5.1. The device has been created in *MEMS+* based on different elementary components detailed on the corresponding exploded diagram.

The **x**/**y** accelerometer consists of a perforated movable plate suspended through lateral springs. Two complementary sensing units are disposed in the central part of the device. Each of them consists of pairs of comb electrodes with two opposite parts, one movable attached to the perforated mass and one fixed on the underlying anchor. When the structure is exposed to an acceleration in **x** or **y** direction, the movable mass makes the electrodes move closer to their fixed part on one side while on the reverse quadrant they move away, respectively. The resulting capacitance values measured in symmetric quadrants are then summed into a differential signal which is linearly proportional to the acceleration of the frame.

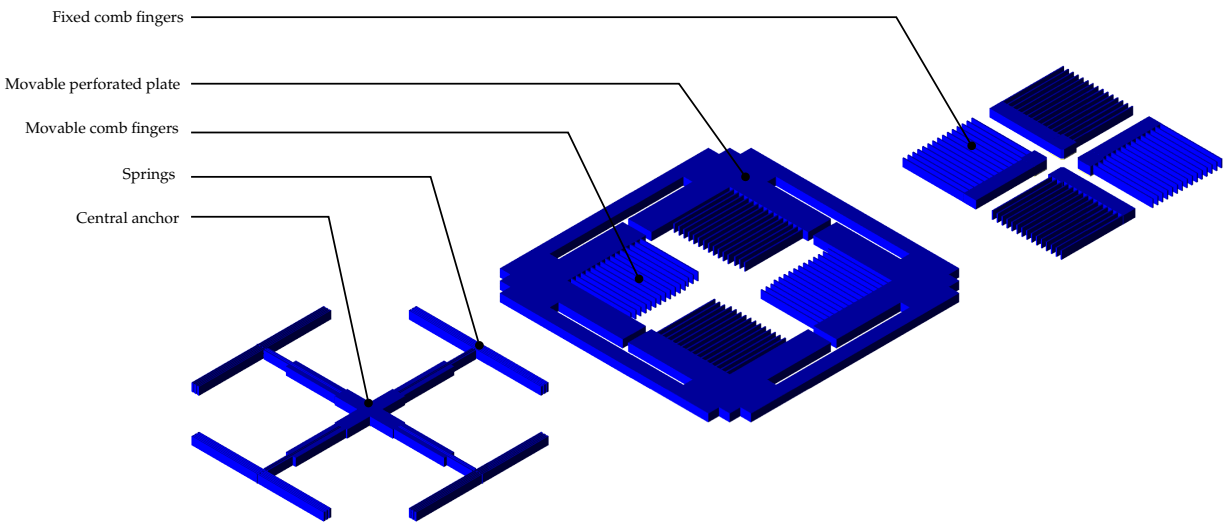
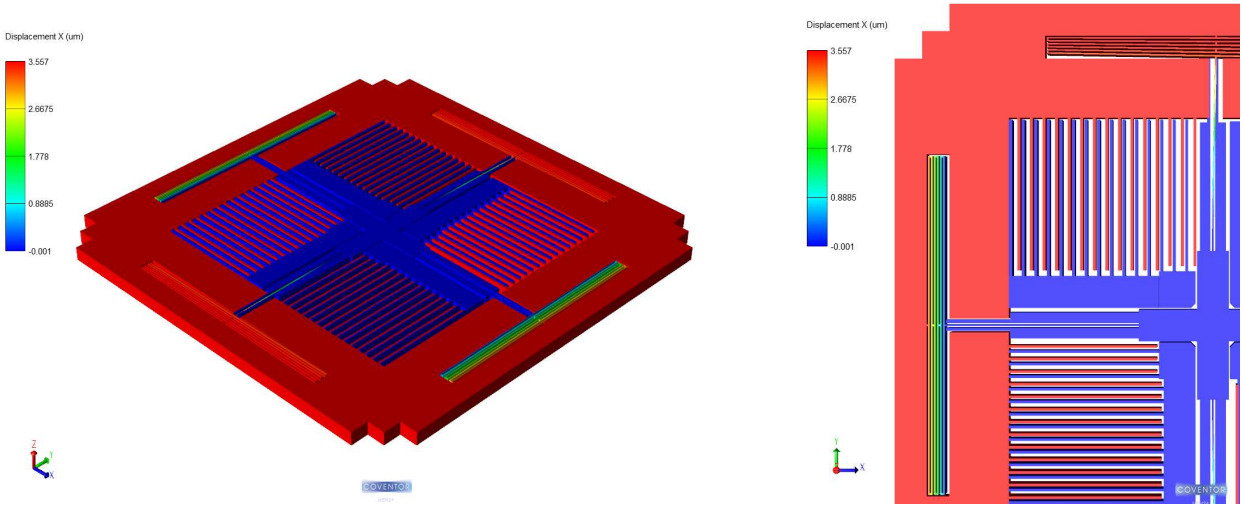


Fig. 5.1: Exploded diagram of the accelerometer **x**/**y** as defined with elementary components in *MEMS+*.

The modal analysis of the accelerometer is perfomed in *MEMS+* and summarized in Table 5.1. The vibrating structure is actuated by both a DC voltage, setting an offset position, and a AC voltage, provoking small oscillations of the sensing part. The corresponding bias and amplitude values of each voltage are listed in Table 5.2. The **x**/**y** accelerometer detects the in-plane motion in **x** and **y**-axis respectively illustrated in Figures 5.2(a) and 5.2(b). The symmetry of its design defines the same sensing frequency on both axes, i.e., 2 560Hz, which is used to actuate the perforated mass.



(a) Sensing mode (2 560Hz) for the sensing of translation acceleration in **x**-axis.

(b) Displacement of the upper left quadrant sensing the plate displacement through comb fingers.

Fig. 5.2: Sensing mode in **x**-axis for the accelerometer simulated in *MEMS+*.

Table 5.1: Modal analysis of the accelerometer.

Mode	**Description**	**Frequency** (Hz)
1	In-plane rotation (**z**-axis)	1 825
2	In-plane translation (**x**-axis)	2 560
3	In-plane translation (**y**-axis)	2 560
4	Out-of-plane rotation (**x**-axis)	13 148
5	Out-of-plane rotation (**y**-axis)	13 148
6	Out-of-plane translation (**z**-axis)	13 969

Table 5.2: Actuation of the accelerometer electrodes

Voltage source	**Bias** (V)	**Amplitude** (V)	**Frequency** (Hz)
Driving (**x**-axis)	2.0	0.025	1 825
Driving (**y**-axis)	2.0	0.025	1 825
Sensing (**x**-axis)	2.0	-	2 560
Sensing (**y**-axis)	2.0	-	2 560

5.2.2 Modeling procedure

We review below the different modeling steps that lead us to a correct high-level description of the studied device. Table 5.3 summarizes the different models employed below. Model 1 corresponds to a first approximation at high-level which consists in a movable plate and has been refined by an equivalent representation of comb fingers in Model 2. We compare these high-level descriptions to the model created in *MEMS+* either in its reduced form in Model 3 or its complete definition in Model 4. We assume here the reduced model as the reference since it has been validated in previous studies [119]. The objective of this section is to demonstrate the importance of the geometry in the definition of high-level models and warn against generic models that may induce errors in the design of the simplified system.

Table 5.3: Index of the models of the accelerometer ST SEM.

Description	**Index**	**Model**
High-level	Model 1	Equivalent movable plate
	Model 2	Equivalent comb fingers
MEMS+	Model 3	Reduced model
	Model 4	Full model

The different capacitive sensing configurations introduced in Section 3.3.1 and depicted in Figure 3.6 are usually applied to accelerometers. We build a first approximation with a simple movable plate capacitor in Model 1. The input of the system is an acceleration which is first converted into a displacement through the transfer function of a resonant system as follow:

$$H(s) = \frac{-1}{\omega^2 + \frac{\omega}{Q}s + s^2}, \tag{5.1}$$

where s is the Laplace variable, ω is the natural pulse and Q is the quality factor of the selected mode. This transfer function describes the mechanical response of the device. The related displacement enables to compute the capacitance C through Equation (2.2) in which $C = \varepsilon A/g$.

The step response of Model 1 is tested against the reduced model, denoted Model 3 which serves as reference. We compare both models with regard to the shift of displacement, respectively of capacitance, that is observed between the initial state and the final steady state of the system. The test is repeated on each model for different acceleration steps, i.e., with an amplitude of $1g$, $2g$, $4g$, $8g$ and $16g$.

Concerning the mechanical displacement of the device, a good accordance between the transfer function (5.1) and the reduced model is shown in Figure 5.3. Model 2 has a low error (0.29%) compared to Model 3 and is assumed conform the dynamics of the system. On the contrary, both models give different capacitance response as show in Figure 5.4. The observed error is up to 15% for an acceleration of $1g$ and tends to decrease with higher acceleration inputs since the capacitance is inverse proportional to the acceleration. The equivalent model of a movable plate is considered as poorly accurate to transform a displacement into a capacitance for small accelerations.

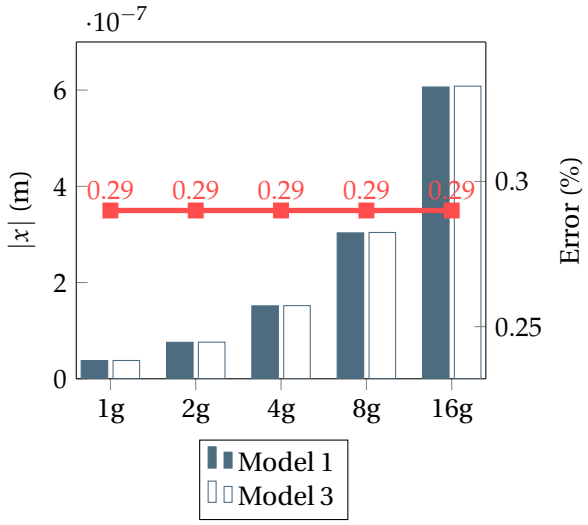


Fig. 5.3: Displacement error.

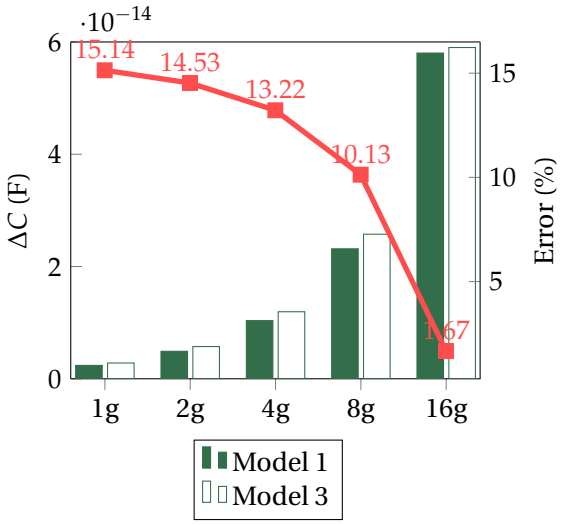
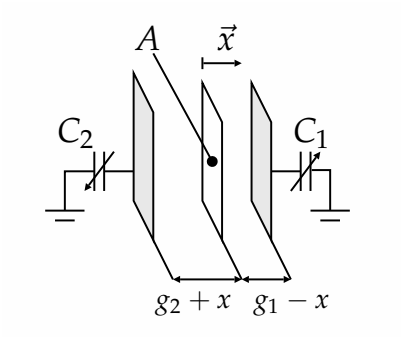


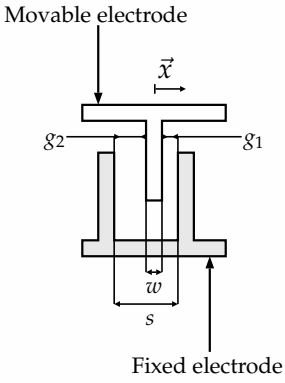
Fig. 5.4: Capacitance shift error.

We disqualify the use of Model 1 which is nonetheless employed to represent an accelerometer in a generic way. This first analysis allows us to isolate the problem in the high-level representation which only concerns the capacitance estimation. We assume in the following the correct reconstruction of the displacement through the transfer function defined in Equation (5.1) which represents the system as a second-order dynamical entity.

To enhance the description of the capacitance, we elaborate another solution considering the actual design of the device and denoted Model 2. The proposed model aims to better cope with the underlying sensing principle. The configuration relies on comb fingers which are approximated by two complementary movable plates with regard to a central fixed electrode. In this case, the comb fingers are asymmetric, i.e., the steady state gap between a finger and the fixed part differs on both sides. The principle of the comb finger is illustrated in Figure 5.5(a) and the parameters are exposed in Figure 5.5(b).



(a) Sensing principle.



Parameter	Value (μm)
Comb spacing (s)	10.7
Comb width (w)	3.1
Total gap ($g_1 + g_2$)	7.6
Offset	± 1.4
g_1	2.4
g_2	5.2

(b) Geometry of comb fingers with parametric definition.

Fig. 5.5: Refined solution in Model 2 matching the sensing principle of the 2-axis accelerometer.

The differential capacitance C_{diff} in Model 2 is given by:

$$C_1 = \frac{\varepsilon_0 A}{g_1 - x} \quad \text{and} \quad C_2 = \frac{\varepsilon_0 A}{g_2 + x}, \tag{5.2}$$

$$\text{thus} \quad C_{diff} = C_1 + C_2 = \varepsilon_0 A \left(\frac{g_1 + g_2}{(g_1 - x)(g_2 + x)} \right) \tag{5.3}$$

where g_1 and g_2 are the offset gaps on both sides of the comb finger at steady state, C_1 and C_2 are respectively the positive and negative capacitance measurement values, A is the side-comb area, x is the displacement of the movable mass and ε_0 is the permittivity of air. Both capacitance values C_1 and C_2 are summed into the differential signal C_{diff} which is then exploited, e.g., by digital sub-component. The corresponding block diagram of the comb finger is illustrated in Figure 5.6.

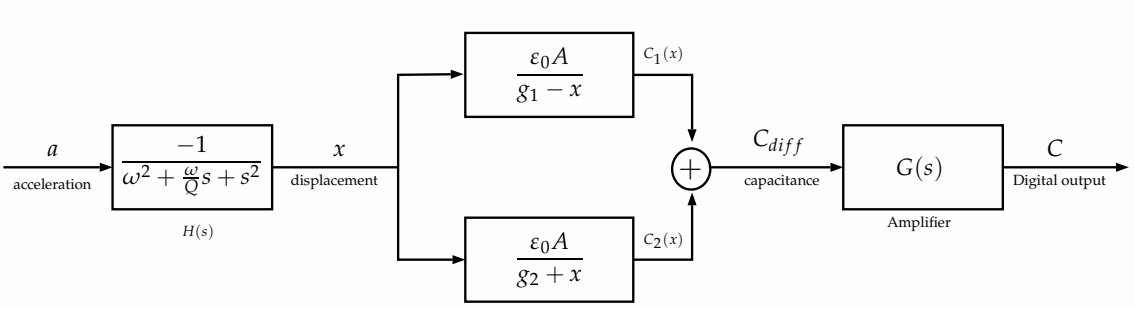


Fig. 5.6: Block diagram of Model 2 based on differential capacitive measurement.

We simulate the step response of Model 2 and Model 3 to an acceleration step of a $2g$ amplitude which presents an error about 14% on the capacitance shift in Model 1 as shown in Figure 5.4. The parameters of the step response are detailed in Figure 5.7 and the corresponding numerical values of the negative capacitance are summarized in Table 5.4 for Model 2 and Model 3. The step response contains qualitative information on the stiffness, the speed and the dynamics of the system. We compare below Model 2 to Model 3 in order to validate the further use of Model 2.

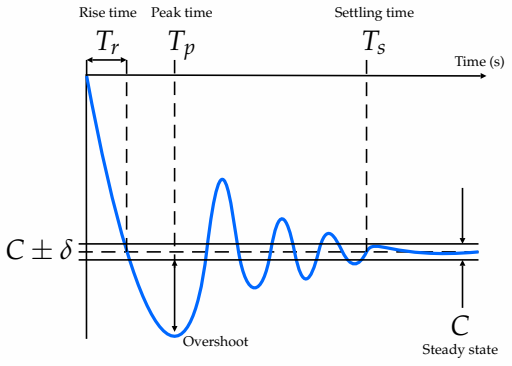


Fig. 5.7: Step response parameters.

Simulation	Model 2	Model 3
Initial state (F)	$3.427730e^{-13}$	$3.427731e^{-13}$
Peak (F)	$3.346286e^{-13}$	$3.376021e^{-13}$
Steady state (F)	$3.383855e^{-13}$	$3.400797e^{-13}$
Acceptable error δ (F)	$6.7677e^{-15}$	$6.8016e^{-15}$
Overshoot (F)	$3.7569e^{-15}$	$2.4776e^{-15}$
Rise time (s)	0.19	0.19
Peak time (s)	0.25019	0.25019
Settling time (s)	0.25097	0.25097

Table 5.4: Negative capacitance response to a $2g$ acceleration step.

The initialization of the system is correctly set in both models with similar value. Once the step is applied, the evolution of capacitance presents nevertheless some differences. The overshoot is twice higher in Model 2 than in Model 3, that means the electrostatic stiffness of the reduced model is not fully recovered by Model 2. The peak value also tends to be more important in the approximated model, but remains acceptable since the absolute error is about 0.9%. Moreover, the final steady state of Model 2 differs from Model 3 with an absolute error of $0.6\% < 1\%$ which is also assumed accurate enough.

The steady state value can be further used to design a control feedback loop, as introduced in Section 2.4.4. To this purpose, we introduce an acceptable error of 2% which implies the controlled response to stay within an interval $[C \pm \delta]$ where $\delta = 0.02C$ and C is the measured steady state in open-loop. In Model 2, the corresponding interval is twice wider than in Model 3 and would require to reduce the acceptable error to 1% in order to meet control design requirements.

Finally, the periodic response of the high-level model is analyzed to validate the frequency definition of the high-level model. The rise time T_r describes the response speed, the peak time T_p indicates the first reached maximum and the settling time T_s informs on the stabilization of the resonant system. All these three periods are equal in both models which ensures the preserved frequency behavior. In the following, the second high-level model presented above is assumed as accurate enough to run a trade-off with the reduced model on the simulation accuracy and performance.

5.2.3 Test-bench definition

The previous models are each encapsulated in a dedicated TDF module which defines a `Device` connected to further analog and digital components. The related `Testbench` implements the signal-processing open loop shown in Figure 5.8. This simulation reproduces the system response to an acceleration change and evaluates both the configurations of Model 2 and Model 3. The multi-physics part instantiates the MEMS device with the corresponding stimuli sources. These latter are all instances of the base class `ScaGenerator` and are connected to the electrical and mechanical inputs of the microsystem. The electrical entries refer to the voltage settings listed in Table 5.2 in order to correctly actuate the device.

The acceleration is generated by a specific source producing various stimuli profiles, e.g., step or square impulse, as detailed in the following section. The output signal of the device is then conditioned by the analog part. To match real operating conditions, a white noise is added to the differential capacitance measurement. The signal is then amplified and converted into a digital signal of 8 bits through dedicated `sca_tdf::sca_module`. This conditioning aims at representing the traditional digital-signal processing flow and connecting the presented solution to register banks and software emulation environments [167]. For instance, the filtered signal in output of the test bench can be further transferred to micro-controller or processor units through specific protocols like SPI or I^2C.

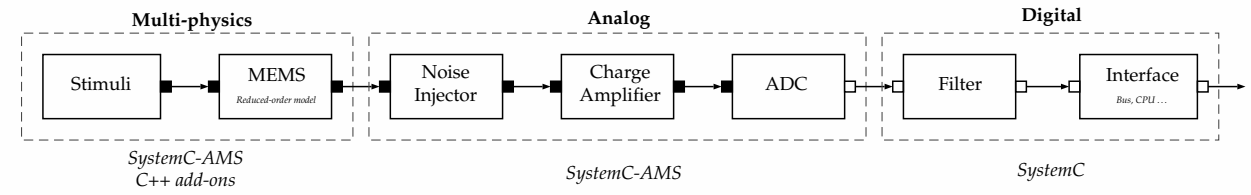


Fig. 5.8: Integration of the MEMS reduced-order or high-level model in a SystemC-based test bench.

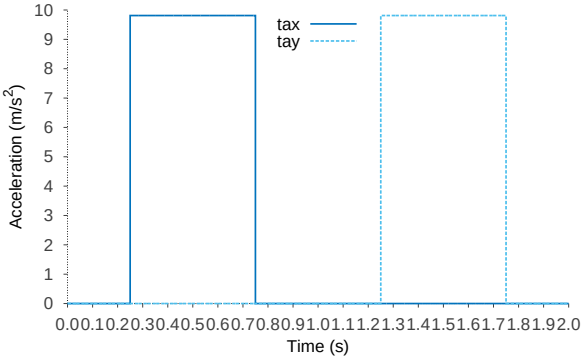
5.3 Experiment

The above test bench is tested in two different configurations. We first analyze the transient response of Model 3 to a square impulse of acceleration and provide a complete digital-signal processing flow. Then, the study is focused on the MEMS itself in order to verify the modeling accuracy and the simulation performance of Model 3 with regard to Model 4 and the simulation of Model 3 and Model 4 in the state-of-the-art solution MATLAB/Simulink. This second study only concerns the step response of the device.

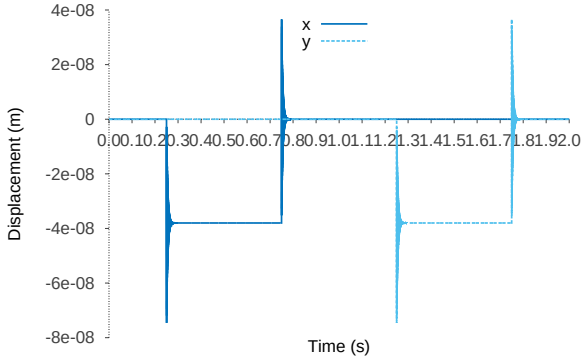
Square impulse

The accelerometer is submitted to a squared impulse of $1g$ acceleration successively in **x** and **y** axis as shown in Figure 5.9(a). The response of the accelerometer consists in a proportional displacement of the proof mass as shown in Figure 5.9(b). This results in a capacitance change in the symmetric quadrant responsible for the sensing of lateral motion. For instance, Figure 5.10(a) details the negative and positive values of the capacitance shift in **x**-axis. Note the displacement in **y**-axis induces a small change in capacitance as observed on the second part of the graph. This coupling between both axes is not reconstructed in Model 2 since each capacitor is isolated from one another.

The sensing principle of the accelerometer relies on the difference of these capacitance values as depicted in Figure 5.10(b) after the addition of a white noise that recreates real operating conditions. The analog signal is then amplified and converted into a digital signal which is combined with a full-scale ratio that varies from 2 to 16 bits as shown in Figure 5.10(b). Note the full-scale ratio conditions the accuracy of the signal by defining the number of digits on which the signal is encoded. The proposed example defines a full-scale ratio that doubles the available digits each $0.25s$ which thus decreases by twice the value of the transferred digital signal. Finally, the digital signal is filtered (Figure 5.10(d)) and fitted for external use through the output interface as shown in Figure 5.10(e).

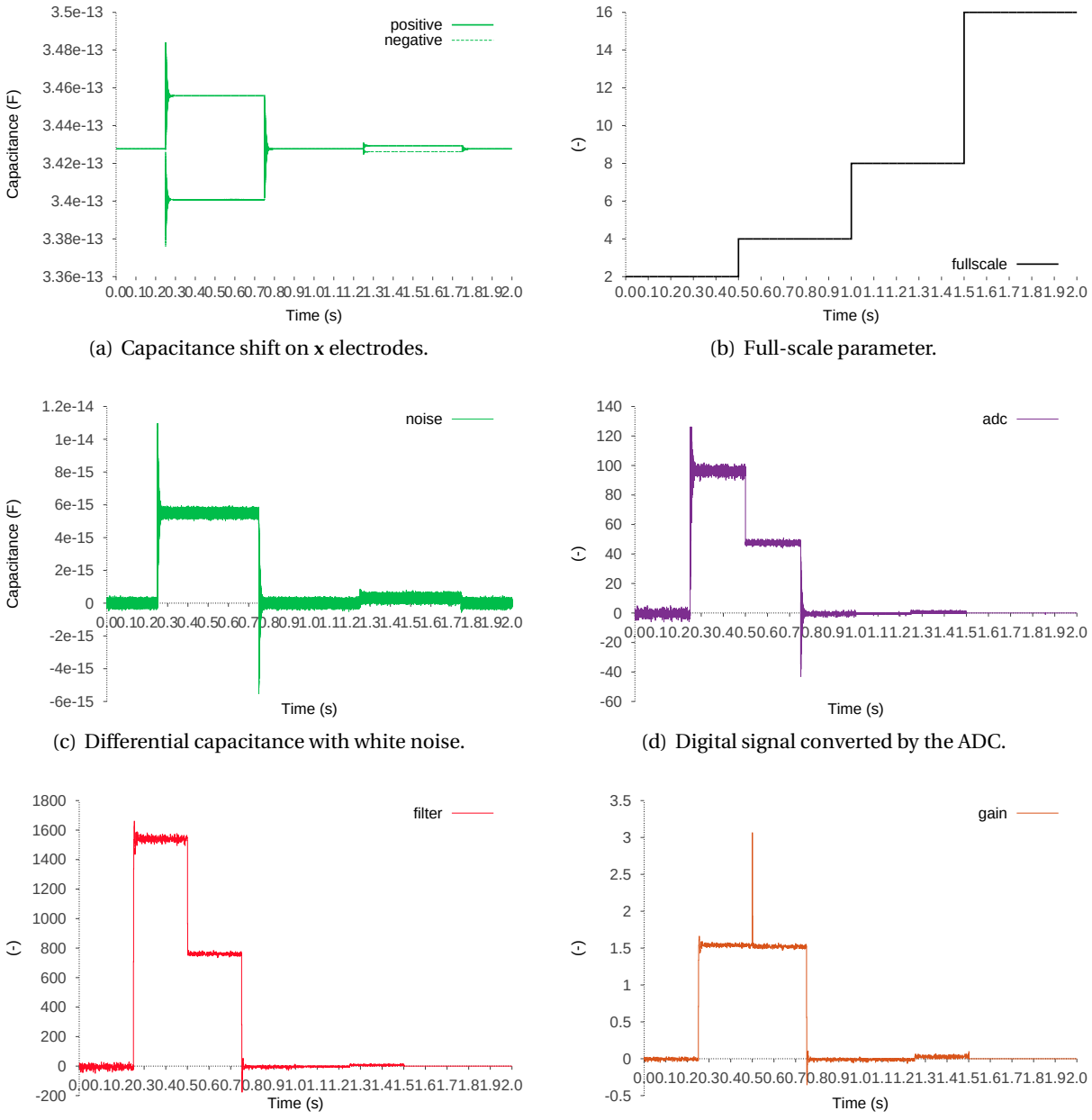


(a) Acceleration input in **x** and **y** axes.



(b) Displacement response in **x** and **y** axes.

Fig. 5.9: Mechanical response to an accleration impulse.



(a) Capacitance shift on **x** electrodes.

(b) Full-scale parameter.

(c) Differential capacitance with white noise.

(d) Digital signal converted by the ADC.

(e) Filtered digital signal.

(f) Fitted output digital signal.

Fig. 5.10: Signal processing in **x**-axis through the different test-bench components.

Step response

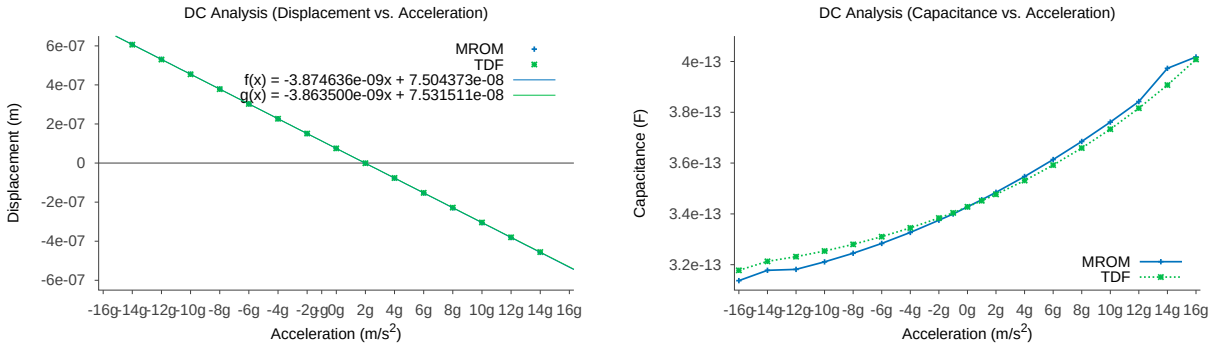
The step response of both high-level and reduced models, respectively Model 2 and Model 3, is compared in terms of modeling accuracy and simulation performance. To this end, a single stimuli source is instantiated with the `ScaGenerator` base class which produces an acceleration step in **x**-axis with a variable amplitude. We assume the same response would be observed in **y**-axis due to the symmetry of the device. The results associated to this second configuration are detailed in the following section.

5.4 Results

The following results are focused on the transient simulation of the accelerometer response to an acceleration step. We first verify the accuracy of Model 2 (TDF) through its step response to various acceleration amplitudes. Then, we analyze the simulation performance of Model 3 (MROM) with regard to the state of the art.

5.4.1 Modeling accuracy

As depicted in Figure 5.6, Model 2 clearly separates the mechanical response from the electrostatic one. To verify the proposed modeling procedure, we apply acceleration steps varying in amplitude from $-16g$ to $16g$. The transfer function correctly reconstructs the linear evolution of the displacement with regard to the acceleration (see Figure 5.11(a)). The accelerometer intends to produce a capacitance shift proportional to the acceleration or the force applied on the proof mass. The response in terms of capacitance is inversely proportional to the acceleration as defined by the Equation (5.3) and verified in Figure 5.11(b). We further analyze the difference between the high-level and reduced-order models.

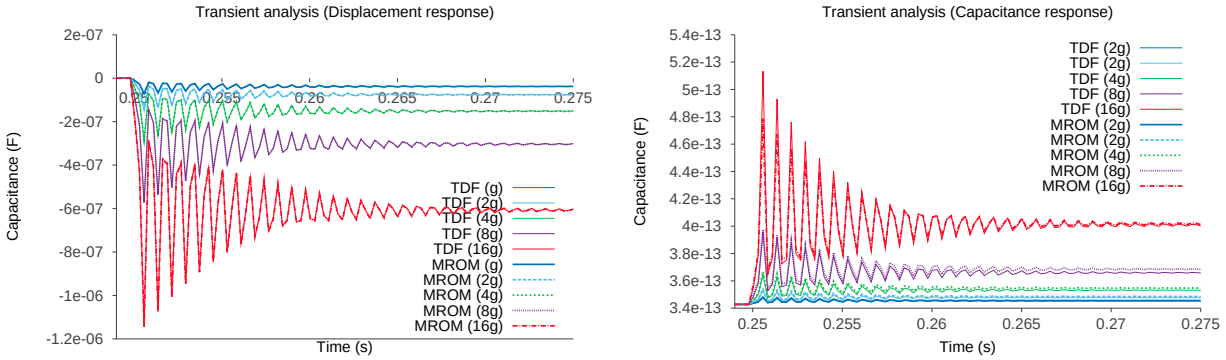


(a) Linear evolution of the displacement.

(b) Nonlinear evolution of the capacitance.

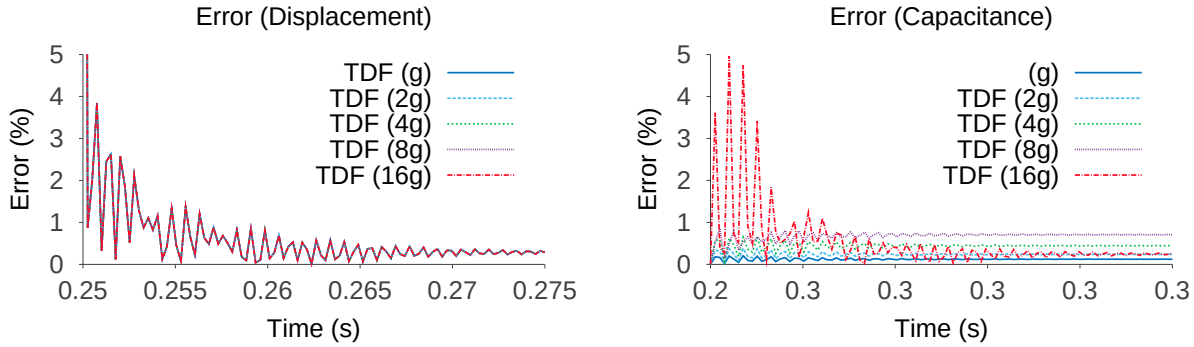
Fig. 5.11: Evolution of the system variables with regard to the acceleration amplitude.

The displacement response to various acceleration steps varying from $1g$ to $16g$ is traced in Figure 5.12(a). The related changes in capacitance are depicted in Figure 5.12(b). We discuss below the evolution of the relative error made on both variables. In Figure 5.13(a), we observe the relative error for displacement is relatively important during the transition period with a peak observed at 5% before returning to a 0.29% high error at steady state. This result confirms the observation made in Section 5.2.2 on the first high-level model. Moreover, the error made on the capacitance remains relatively low, i.e., under 1%, as shown in Figure 5.13(b). Note the response to high amplitudes tends to be more unstable, especially during the transition phase, but leads to an error on the steady state that remains low enough to be acceptable. For instance, the response at $16g$ is slightly impacted by the instability of the transition phase with a maximum error about 5% at peak, before decreasing to a relative error lower than 0.5% at steady state.



(a) Displacement variation. (b) Positive capacitance variation.

Fig. 5.12: Step response of the accelerometer to an acceleration step.



(a) Relative error on the displacement value. (b) Relative error on the capacitance value.

Fig. 5.13: Relative error observed for various acceleration step amplitudes.

Based on the previous observations, we conclude below on the correctness and the validity of the high-level model proposed in Section 5.2.2. Its refined version recovers the dynamic behavior of the device with a relative error that remains low enough, i.e., less than 0.5%, to ensure a correct simulation of the steady state of both the displacement and the capacitance. Moreover, the use of such a model for transient analysis is assumed acceptable since the response of the system to nonlinearities or discontinuities in input leads to a similar low-error range. Nevertheless, the observed errors on the first approximated model, i.e., through a single movable plate, disqualify a generic use of pre-configured models, even for system-level analysis.

We insist on the necessity to respect the actual geometry of the device in order to build correct descriptions with high level of abstraction. To this purpose, the device design must be known and we justify the use of reduced models since they encapsulate the information related to the inertial and spatial definition of the device. In order to correctly configure system-level models, these requirements would be either extracted from detailed descriptions or measured on the physical device itself. We also consider that a descendant approach relying on pre-defined models for each kind of devices, like accelerometers or gyroscopes, is inappropriate to build accurate-enough models. The inertial and the geometric definition of the device are indeed required to cope with the overall behavior of the device as demonstrated through the above accelerometer. Also, from a modeling viewpoint, we consider the reduced-order model as more adapted to efficiently address the MDVP objective to quickly bind, integrate and test different devices together.

5.4.2 Simulation performance

Numerical methods for integrating equations of motion are usually compared in terms of their accuracy and stability. We reviewed above the accuracy of the simulation results for the high-level and reduced models. In the following, we are interested in the stability of solutions, especially with regard to the time-step definition.

We noticed in Section 4.4 the influence of the time stepping in SystemC-AMS. The standard integration method in TDF MoC is an explicit method, i.e., the Euler-backward method, for which the stability is governed by the shortest natural period. The accuracy and stability usually depend upon the ratio of the time step, h, to the shortest natural period of the structural system [95]. The shortest natural period T_n can be much shorter than the fundamental natural period T_1. For instance, the sensing mode has a resonant frequency at 2 560 Hz, i.e., a related period of $T_1 = 0.39$ ms. The results presented above were obtained with a fundamental natural period of $T_n = 3$ μs which is the maximum time step for which the relative error of Model 2 remains under 1 %. The stability is also preserved by a ratio to the shortest natural period $T_n/T_1 > 10^3$. In contrast, Model 3 has a higher maximum time step equal to 10 μs, since the Crank-Nicolson method implemented in *MEMS+* API is an implicit method, i.e., defined as unconditionally stable.

Table 5.5: Simulation results for a ramp impulse in translation acceleration in **x**-axis (Amplitude: 1g)

Environment	**Index**	**Model**	**Integration**	**Time step**	h (μs)	**Runtime** (s)
SystemC-AMS	Simulation 1	Model 2	Backward-Euler	fixed	3.0	12.03
	Simulation 2	Model 3	Crank-Nicolson	fixed	3.0	103.08
	Simulation 3	Model 3	Crank-Nicolson	fixed	10.0	30.93
MATLAB/Simulink	Simulation 4	Model 3	`ode23`	fixed	10.0	152.48
	Simulation 5	Model 4	`ode8`	variable	–	335.46

In Simulation 1, the time step is constrained to 3 μs and one simulated second of Model 2 has a simulation runtime about 12 s. This is more than eight times faster than Simulation 2 that computes the equivalent simulation of Model 3 in 100 s. In contrast to Model 3, Simulation 2 can be run with a larger time step of 10 μs which decreases the run time to 30 s, i.e., twice longer than the high-level model.

This simulation performance is finally compared to the state of the art in Simulation 4. The configuration is similar to Simulation 3 with an implicit integration scheme and a time step of 10 μs. As noticed in Section 4.4, MATLAB/Simulink offers similar performance than in SystemC-AMS if the time stepping is configured as dynamic and the integration scheme is based on on a multistep method like Runge-Kutta. The selection of time-stepping and integration methods is an extensive area, well introduced by Burden *et al.* [156] and further discussed with regard to the stability and convergence criteria in [221]. Figure 5.14 shows the simulation runtimes of Model 3 in SystemC-AMS and MATLAB/Simulink. In addition, the full model (Model 4) is integrated to Simulation 5 to evaluate the performance of the simulation. Here the linearity of the system enables the simulation of the full model in MATLAB which is normally not possible due to the large number of DoFs. In SystemC-AMS, Simulation 3 is five times faster than Simulation 4 in MATLAB/Simulink and reduces by ten the simulation of the full model (Simulation 5).

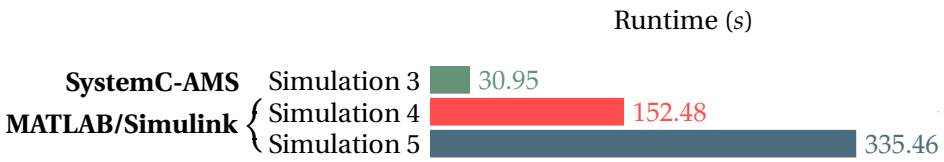


Fig. 5.14: Performance analysis between the different simulation environments highlighting a faster simulation in SystemC-AMS than in MATLAB/Simulink when both environments are configured with the same fixed time step and similar integration schemes.

5.5 Conclusion

The above example illustrates the use of the API between SystemC-AMS and *MEMS+* on a two-axis accelerometer and its integration into a full digital-signal processing flow. This study compared the modeling procedure and the simulation performance of the reduced-order model issued from *MEMS+* and higher-level representations built upon the SystemC-AMS standard. We first discussed the validity of transfer functions employed at high-level and concluded on the necessary knowledge of the operating principle to correctly recover the sensing capacitance of the device. High-level modeling approximations can lead to severe errors on the behavioral representation of the device if not matching the proper geometry and sensing principle of the device. To avoid such approximations, reduced models can be automatically generated from detailed MEMS design created in *MEMS+*. This eliminates the need for an equivalent modeling process as well as the extraction of specific parameters.

Based on a correct high-level description, we evaluated the simulation performance of the high-level and reduced models in SystemC-AMS. The relative error of the high-level model is acceptable while its simulation performance is twice faster than the reduced model. Nevertheless, the reduced-order model implements a more efficient integration method which allows for larger simulation time step and, in this case, leads to comparable simulation runtime. The use of a dynamic time-stepping is seen as a potential improvement in order to accelerate the simulation which is, in this case, already up to ten times faster than the state-of-the-art solution.

In a first pass, high-level models may grandly differ from the actual device design. We therefore argue for a direct use of reduced models in order to simplify the definition and maximize the re-use of system-level models of MEMS. This reproducible process is also independent of the nature of the MEMS device and avoids first approximations or unnecessary equivalent models.

Chapter 6

Conclusions

This dissertation proposed a novel approach for the system-level simulation of MEMS models within a state-of-the-art HW/SW co-development environment. Our study was focused on providing an efficient method to integrate reduced-order models within the system-level language SystemC-AMS.

6.1 Contributions

In the presented work, we have made major contributions to the field of MEMS system-level integration based on the SystemC-AMS standard and the commercial software solution *MEMS+*. We explored existing solutions to support the modeling and simulation of MEMS devices at system-level. We then evaluated the use of reduced models extracted from detailed descriptions to preserve the modeling accuracy. Finally, we elaborated a software solution to automate the MOR process and the interactions with *MEMS+*. The proposed solution implements an interface with the SystemC-AMS simulation environment and enables MEMS and system engineers to use a single model. By preserving the integrity and accuracy of the model, this method allows system engineers to verify that the targeted HW/SW applications are compliant with regards to peripherals, such as sensors or actuators.

Modeling and simulation of MEMS devices in SystemC-AMS

Based on different MoCs, SystemC-AMS supports the modeling of MEMS at system-level through equivalent circuit or lumped-element models and, to some extent, bond graphs. The lack of standardization in modeling MEMS devices at system-level does not allow complex or refined designs. The over-specialization of system-level models make them hard to refine and highly dependable on initial assumptions. Furthermore, the SystemC-AMS standard does not support some basic operations in linear algebra and does not provide a standardized implementation of integration and approximation methods. We also used external libraries to perform matrix-vector operations and implemented our own integration algorithms. In addition, the time-step definition is not automated and the correct simulation settings are cumbersome for the user. Despite the previous modeling and simulation limitations, SystemC-AMS correctly interfaces MEMS devices with digital HW/SW components.

Integration of MEMS reduced models in System-AMS

Instead of the classic top-down approach, we proposed an alternative mehtod based on a first detailed design in order to integrate MEMS in complex systems. A precise reduced-order model is automatically gen-

erated from *MEMS+*. We implemented a C++ API to integrate the reduced-order model into any SystemC-AMS test bench. It can then be connected to a full digital signal flow. In addition, the API provides basic blocks and add-on features to help the designer in the definition of new case studies and analyses. From a modeling viewpoint, the reduced models ensure a better accuracy than simple lumped-element models since they preserve the modal behavior and internal couplings of the device. The implemented integration scheme prevents the simulation from numerical damping as verified on a first gyroscope use case. We finally validated the modeling method and the simulation performance on a real accelerometer.

6.2 Future work

Validation of the simulation framework

Our study is focused on the system-level modeling and simulation of MEMS devices assuming the electronics, i.e., the HW and SW sub-parts, is already supported by the SystemC and SystemC-AMS standards. The simulation of complex architectures with MEMS, HW and SW sub-parts needs to be validated. This demonstration would aim at verifying the correct simulation of the different parts and evaluating the simulation performance of the overall. A good example would be the modeling and simulation of an inertial measurement unit that contain a microcontroller connected to a set of MEMS devices, usually an accelerometer, a gyroscope and a magnetometer. These units are massively integrated to embedded systems in order to measure their orientation and their position.

Time-stepping algorithms

The current solution could be improved by two complementary time-stepping methods. On the one hand, the use of a dynamic time-stepping is seen as a potential improvement in order to accelerate the simulation which is, in this case, already up to ten times faster than the state-of-the-art solutions. This could be further tested in the current implementation of SystemC-AMS through DTDF modules. Nevertheless, the current dynamic time stepping defined in DTDF forces the simulation to change the whole time stepping, and is not fitted for local changes. In addition, such an optimization would require to set a reasonable error regarding the global dynamics of the system which greatly differs from digital to continuous-time sub-parts. Therefore, specific conditions must preserve the simulation from recursive error testing and prevent potential slow down. On the other hand, multi-step integration schemes should enhance the stability of the solution. We demonstrate in the different use cases in our study the validity of an implicit method, like the Crank-Nicolson one. Additional integration methods would thus complement a dynamical time stepping and offers more flexibility to the user to address some unstable simulation cases.

Parametric reduced models

We proposed to predict the complicated dynamical behavior of MEMS devices through reduced models instead of simplified equivalent representations. In general, the MOR techniques have been improved over the last decade to reconstruct the nonlinearities of the complex systems for which they provide compact descriptions. The selected MOR method could be completed by further investigations on its ability to recover the mechanical nonlinearities besides the electrostatic ones already supported. Furthermore, the solution could benefit from a parametric definition of the reduced models in order to explore the design

space. This would allow the designers to test and optimize the design of MEMS devices by directly taking into account the need of the global system.

Continuous systems simulation

Finally, the proposed solution respects the SystemC-AMS standard implementation especially the synchronous formalism implemented in TDF. Nevertheless, the lack of interoperability between the discrete-event solver in SystemC and the TDF one appears as a major limitation in SystemC-AMS to correctly address the simulation of continuous-time systems. Therefore, the simulation environment could benefit from an additional MoC implementing the DEVS methodology. This intends to avoid a tight synchronization between the discrete-event and continuous-time systems since the coupling between solvers is realized through an state-based synchronization instead of the time-based formalism employed in SystemC-AMS. This principle has largely been discussed in the literature and is used, e.g., in Modelica [222].

6.3 Closing thoughts

The tight integration of MEMS components in complex systems implies to accurately predict their eletromechanical response. This especially allows the calibration of dedicated HW/SW applications. The related computer-aided analysis is an indispensable tool to predict and optimize the MEMS integration bottlenecks in mixed-signal electronics. But to be effective, such a design-driven method needs to be computationally efficient and must support a pragmatic design iteration. Although the computational efficiency is primarily driven by short product design cycles, the simulation speed is not sufficient to demonstrate the full capacity of a simulation solution to address the MDVP of heterogeneous systems. The virtual prototyping can be enforced by the physics-based modeling and simulation and perceived as an enabling design capability since the accuracy is necessary to verify complex systems.

We assumed in this work that SystemC is one of the most advanced solutions for the co-development of HW/SW applications at system-level. Its digital-centered approach relies on a discrete-event simulation kernel and partially allows to simulate continuous-time systems in SystemC-AMS. The object-oriented foundations of SystemC enables the definition of additional MoCs in order to build a unified MDVP environment. This aims at the modeling and simulation in several physical domains with regard to a satisfactory and high fidelity semantics.

The MoC-based approach remains questionable in the case of complex systems like MEMS that requires an in-depth understanding of the underlying multi-physics. In our case, we interfaced SystemC with an existing software solution arguing for domain-specific modeling methodologies and standard simulation environments. Such an hybrid solution avoids the cumbersome implementation of dedicated MoC and the inaccurate use of high-level models. Despite their limited investigation to date, hybrid methodologies are a promising path to improve the modeling flexibility and computational efficiency of existing solutions. Relying on the concurrent use of different modeling methodologies, hybrid methods allow different degrees of accuracy and complexity in the same simulation environment. This facilitates the development of comprehensive and accurate numerical models for heterogeneous systems. Regarding the presented solution, hybrid methods could rapidly tackle current industrial needs.

Finally, the successful integration of heterogeneous systems through hybrid methods mostly relies on the convergence of design flows and the collaboration between expert teams. Beyond novel modeling methodologies and simulation environments, the author would recommend to develop platforms to connect designers and system engineers. Such distributed solutions would facilitate the exchange of IPs and models, strengthen the interoperability between processes and speed up the integration of novel complex architectures.

Bibliography

[1] R. P. Feynman, *There's Plenty of Room at the Bottom*, pp. 63–76. Perseus Books, 1999.

[2] A. Heinig, M. Dietrich, A. Herkersdorf, F. Miller, T. Wild, K. Hahn, A. Gruenewald, R. Brueck, S. Kroehnert, and J. Reisinger, "System Integration - The Bridge between More than Moore and More Moore," IEEE, 2014. Design, Automation and Test in Europe Conference and Exhibition (DATE), Dresden, Germany, March 24-28, 2014.

[3] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*. 2nd ed., 2015.

[4] S. D. Senturia, *Microsystem Design*. Springer US, 2001.

[5] S. P. D. Manoj and H. Yu, "Cyber-Physical Management for Heterogeneously Integrated 3D Thousand-core On-chip Microprocessor," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 533–536, IEEE, 2013. IEEE International Symposium on Circuits and Systems (ISCAS), Beijing, China, May 19-23, 2013.

[6] S. Franssila and S. Tuomikoski, "MEMS Lithography," in *Handbook of Silicon Based MEMS Materials and Technologies* (M. Tilli, T. Motooka, V.-M. Airaksinen, S. Franssila, M. Paulasto-Kröckel, and V. Lindroos, eds.), Micro and Nano Technologies, pp. 427 – 443, William Andrew Publishing, 2nd ed., 2015.

[7] J. W. Judy, "Microactuators," in *MEMS* (O. Paul and J. G. Korvink, eds.), pp. 751 – 803, William Andrew Publishing, 2006.

[8] F. Laermer, "Mechanical microsensors," in *MEMS: A Practical Guide to Design, Analysis, and Applications* (J. Korvink and O. Paul, eds.), pp. 523–566, Springer, 2006.

[9] H. Xie, G. K. Fedder, and R. E. Sulouff, "Accelerometers," in *Comprehensive Microsystems* (Y. Gianchandani, O. Tabata, and H. Zappe, eds.), pp. 135 – 180, Elsevier, 2008.

[10] U. Bonne, "Gas sensors," in *Comprehensive Microsystems* (Y. Gianchandani, O. Tabata, and H. Zappe, eds.), pp. 375 – 432, Elsevier, 2008.

[11] A. D. Hennis and J. Chae, "Pressure sensors," in *Comprehensive Microsystems* (Y. Gianchandani, O. Tabata, and H. Zappe, eds.), pp. 101 – 133, Elsevier, 2008.

[12] J. Heinzl, "Ink jets," in *Reference Module in Materials Science and Materials Engineering* (S. Hashmi, ed.), pp. 335–368, Elsevier, 2008.

[13] K. Hane and M. Sasaki, "Micro-mirrors," in *Comprehensive Microsystems* (Y. Gianchandani, O. Tabata, and H. Zappe, eds.), pp. 1 – 63, Elsevier, 2008.

[14] R.-M. Chao, S.-Y. Li, C.-C. Hsu, and S. Y. Liang, "Characteristic evaluation of silicon-based MEMS acoustic/acceleration sensor," in *Proceedings of the ASME International Conference on Manufacturing Science and Engineering - 2007*, pp. 389–394, ASME, 2007. ASME International Conference on Manufacturing Science and Engineering, Atlanta, GA, Oct 15-18, 2007.

[15] V. Button, "New technological advancements in biomedical variables transducing," in *Principles of Measurement and Transduction of Biomedical Variables* (V. Button, ed.), pp. 351 – 360, Academic Press, 2015.

[16] Y.-H. Lin, T.-M. Pan, and M.-H. Wu, "Microfluidic technology and its biological applications," in *Comprehensive Biotechnology* (M. Moo-Young, ed.), pp. 141 – 157, Academic Press, 2nd ed., 2011.

[17] A. Vasudev and S. Bhansali, "Microelectromechanical systems (mems) for in vivo applications," in *Implantable Sensor Systems for Medical Applications* (A. Inmann and D. Hodgins, eds.), Woodhead Publishing Series in Biomaterials, pp. 331–358, Woodhead Publishing, 2013.

[18] T. Ryhänen and H. Pohjonen, "Impact of Silicon MEMS — 40 Years After," in *Handbook of Silicon Based MEMS Materials and Technologies* (M. Tilli, T. Motooka, V.-M. Airaksinen, S. Franssila, M. Paulasto-Kröckel, and V. Lindroos, eds.), Micro and Nano Technologies, pp. xix – xxxvii, William Andrew Publishing, 2nd ed., 2015.

[19] D. Tanner, "MEMS reliability: Where are we now?," *Microelectronics Reliability*, vol. 49, no. 9–11, pp. 937 – 940, 2009. 20th European Symposium on the Reliability of Electron Devices, Failure Physics and Analysis.

[20] R. Schneiderman, "EDA Sets the Standard for Complexity," in *Modern Standardization:Case Studies at the Crossroads of Technology, Economics, and Politics*, p. 288, Wiley-IEEE Standards Association, 2015.

[21] G. Schröpfer, M. McNie, M. da Silva, R. Davies, A. Rickard, and F. Musalem, "Designing manufacturable MEMS in CMOS compatible processes - Methodology and case studies," in *MEMS, MOEMS, and Micromachining* (Urey, H and El Fatatry, A, ed.), vol. 5455 of *Proceedings of the Society of Photo-optical Instrumentation Engineers (SPIE)*, pp. 116–127, SPIE, 2004. Conference on MEMS, MOEMS and Micromachining, Strasbourg, France, Apr. 29-30, 2004.

[22] G. Hoelzer, R. Knechtel, S. Breit, and G. Schröpfer, "3-D Process Modeling - A Novel and Efficient Tool for MEMS Foundry Design Support," in *2009 IEEE/SEMI Advanced Semiconductor Manucfacturing Conference*, pp. 200–205, IEEE, 2009. IEEE/SEMI Advanced Semiconductor Manufacturing Conference, Berlin, Germany, May 10-12, 2009.

[23] MEMS+®. http://www.coventor.com. Coventor.

[24] *Standard SystemC AMS extensions Language Reference Manual*. Accelera Systems Initiative, 2010.

[25] P. R. Wilson and H. A. Mantooth, *Model-based engineering for complex electronic systems*. Elsevier ; Newnes, 2013.

[26] M. Fowler, *UML Distilled*. Addison-Wesley, 2nd ed., 2000.

[27] J. Holt and S. Perry, *SysML for Systems Engineering*, vol. 7. The Institution of Engineering and Technology, 2008.

[28] F. Mallet and R. de Simone, "MARTE: A Profile for RT/E Systems Modeling, Analysis–and Simulation?," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pp. 43:1–43:8, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[29] "IEEE Standard for Modeling and Simulation & High Level Architecture (HLA)– Framework and Rules," *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*, pp. 1–38, Aug 2010.

[30] G. Lasnier, J. Cardoso, P. Siron, C. Pagetti, and P. Derler, "Distributed simulation of heterogeneous and real-time systems," in *Distributed Simulation and Real Time Applications (DS-RT), 2013 IEEE/ACM 17th International Symposium on*, pp. 55–62, Oct 2013.

[31] D. Broman, C. Brooks, L. Greenberg, E. Lee, M. Masin, S. Tripakis, and M. Wetter, "Determinate composition of FMUs for co-simulation," in *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, pp. 1–12, Sept 2013.

[32] C. Ptolemaeus, *System design, modeling, and simulation: using Ptolemy II*. Ptolemy.org, 2014.

[33] A. Jantsch, *Modeling embedded systems and SoC's concurrency and time in models of computation*. Morgan Kaufmann, 2004.

[34] D. J. Murray-Smith, "Issues of model quality and the validation of dynamic models," in *Modelling and Simulation of Integrated Systems in Engineering* (D. J. Murray-Smith, ed.), pp. 215 – 268, Woodhead Publishing, 2012.

[35] J. Liu, X. Liu, and E. Lee, "Modeling distributed hybrid systems in Ptolemy II," in *American Control Conference, 2001. Proceedings of the 2001*, vol. 6, pp. 4984–4985 vol.6, 2001.

[36] C. Brooks, E. Lee, S. Tripakis, and others, "Exploring models of computation with Ptolemy II," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2010 IEEE/ACM/IFIP International Conference on*, pp. 331–332, IEEE, 2010.

[37] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity - the Ptolemy approach," *Proceedings of the IEEE*, vol. 91, pp. 127–144, Jan 2003.

[38] F. Boulanger and C. Hardebolle, "Simulation of Multi-Formalism Models with ModHel'X," in *Software Testing, Verification, and Validation, 2008 1st International Conference on*, pp. 318–327, April 2008.

[39] I. Sander, A. Jantsch, and Z. Lu, "Development and application of design transformations in ForSyDe," *Computers and Digital Techniques, IEEE Proceedings on*, vol. 150, pp. 313–320, Sept 2003.

[40] MATLAB/Simulink®. http://www.mathworks.com. Mathworks.

[41] G. Stewart, "Research, Development, and LINPACK," in *Mathematical Software* (J. R. Rice, ed.), pp. 1–14, Academic Press, 1977.

[42] B. Garbow, "EISPACK - Package of Matrix Eigensystem Routines," *Computer Physics Communications*, vol. 7, no. 4, pp. 179–184, 1974.

[43] S. L. Campbell, J.-p. Chancelier, and R. Nikoukhah, *Modeling and Simulation in Scilab / Scicos*. Springer, 2006.

[44] V. Giurgiutiu, *Micromechatronics: modeling, analysis, and design with MATLAB*. Nano- and microscience, engineering, technology, and medicine series, CRC Press, 2nd ed., 2009.

[45] D. Quaglia, R. Muradore, R. Bragantini, and P. Fiorini, "A SystemC/Matlab co-simulation tool for networked control systems," *Simulation Modelling Practice and Theory*, vol. 23, pp. 71–86, Apr. 2012.

[46] P. A. Fritzson, *Introduction to modeling and simulation of technical and physical systems with Modelica*. Wiley : IEEE Press, 2011.

[47] L. Ljung and T. Glad, *Modeling of dynamic systems*. Prentice Hall information and system sciences series, PTR Prentice Hall, 1994.

[48] Arquimedes Canedo and Jan H. Richter, "Architectural Design Space Exploration of Cyber-physical Systems Using the Functional Modeling Compiler," *Procedia CIRP*, vol. 21, pp. 46 – 51, 2014. 24th CIRP Design Conference.

[49] "IEEE Standard VHDL Language Reference Manual," *IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002)*, pp. 1–620, Jan 2009.

[50] "IEEE Standard for Verilog Hardware Description Language," *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, pp. 1–560, 2006.

[51] F. Pêcheux, C. Lallement, and A. Vachoux, "VHDL-AMS and Verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, pp. 204–225, Feb 2005.

[52] B. Bailey, G. Martin, and A. Piziali, "Hardware implementation," in *ESL Design and Verification* (B. Bailey, G. Martin, and A. Piziali, eds.), Systems on Silicon, pp. 333 – 377, Morgan Kaufmann, 2007.

[53] R. Zurawski, ed., *Embedded systems handbook - Embedded Systems Design and Verification*. Industrial information technology series, CRC Press, 2nd ed., 2009.

[54] M. Radetzki, ed., *Languages for Embedded Systems and their Applications*, vol. 36 of *Lecture Notes in Electrical Engineering*. Springer, 2009.

[55] B. Bailey and G. Martin, *ESL Models and their Application*. Springer US, 2010.

[56] *IEEE Standard for SystemC Language Reference Manual*. Jan 2012.

[57] D. C. Black, B. Bunton, A. Keist, and J. Donovan, *SystemC - From the Ground Up*. Springer, 2nd ed., 2010.

[58] *Universal Verification Methodology (UVM) 1.2 - Class Reference*. Accelera Systems Initiative, 2014.

[59] H. D. Patel, S. K. Shukla, and R. A. Bergamaschi, "Heterogeneous behavioral hierarchy for system level designs," in *2006 Design Automation and Test in Europe (DATE), Proceedings of*, pp. 563–568, March 2006.

[60] T. Mähne, Z. Wang, B. Vernay, L. Andrade, C. Ben Aoun, J.-P. Chaput, M.-M. Louërat, F. Pêcheux, A. Krust, G. Schröpfer, M. Barnasconi, K. Einwich, F. Cenni, and O. Guillaume, "UVM-SystemC-AMS based Framework for the Correct by Construction Design of MEMS in their Real Heterogeneous Application Context," in *ICECS 2014*, December 2014.

[61] Coside®. http://www.coside.de. Fraunhofer IIS, Design Automation Division.

[62] T. Kirchner, N. Bannow, C. Kerstan, and C. Grimm, *System Specification and Design Languages: Selected Contributions from FDL 2010*, ch. A Framework for Interactive Refinement of Mixed HW/SW/Analog Systems, pp. 71–89. Springer New York, 2012.

[63] N. Bombieri, D. Drogoudis, G. Gangemi, R. Gillon, E. Macii, M. Poncino, S. Rinaudo, F. Stefanni, D. Trachanis, and M. van Helvoort, "SMAC: Smart Systems Co-design," in *Digital System Design (DSD), 2013 Euromicro Conference on*, pp. 253–259, Sept 2013.

[64] F. Fummi, M. Lora, F. Stefanni, and S. Vinco, "Code Generation Alternatives to Reduce Heterogeneous Embedded Systems to Homogeneity," in *FDL 2013*, 2013.

[65] I. O'Connor, "From performance modeling to design space modeling: an exploration method for heterogeneous systems," in *Ecole d'hiver francophone sur la technologie de conception de systèmes embarqués hétérogènes (FETCH)*, (Ottawa, Canada), January 2014.

[66] C. B. Aoun, L. Andrade, T. Mähne, F. Pecheux, M. M. Louerat, and A. Vachoux, "Pre-simulation elaboration of heterogeneous systems: The SystemC multi-disciplinary virtual prototyping approach," in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015 International Conference on*, pp. 278–285, July 2015.

[67] O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method for Solid and Structural Mechanics*. Elsevier, 6th ed., 2005.

[68] G. Lorenz and G. Schröpfer, "3D Parametric-Library-Based MEMS/IC Design," in *System-Level Modeling of MEMS*, ch. 17, pp. 405–424, Wiley-VCH Verlag GmbH & Co. KGaA, 2013.

[69] B. Vigna, "It Makes Sense: How Extreme Analog and Sensing Will Change the World," in *Solid-State Sensors, Actuators and Microsystems, 2012 Workshop on*, pp. 58–65, 2012.

[70] A. C. Antoulas, *Approximation of Large-Scale Dynamical Systems (Advances in Design and Control)*. Society for Industrial and Applied Mathematics, 2005.

[71] L. Feng, P. Benner, and J. G. Korvink, "System-Level Modeling of MEMS by Means of Model Order Reduction (Mathematical Approximations) – Mathematical Background," in *System-level modeling of MEMS*, pp. 53–93, Wiley-VCH Verlag GmbH & Co. KGaA, 2013.

[72] M. Rewieński, G. Fotyga, A. Lamecki, and M. Mrozowski, "Automated reduced model order selection," *IEEE Antennas and Wireless Propagation Letters*, vol. 14, pp. 382–385, 2015.

[73] H. Chang, Y. Zhang, J. Xie, Z. Zhou, and W. Yuan, "Integrated Behavior Simulation and Verification for a MEMS Vibratory Gyroscope Using Parametric Model Order Reduction," *Journal of Microelectromechanical Systems*, vol. 19, pp. 282–293, Apr. 2010.

[74] U. Baur, P. Benner, and L. Feng, "Model order reduction for linear and nonlinear systems: a system-theoretic perspective," tech. rep., Max Planck Institute Magdeburg Preprints, 2014.

[75] M. Rewienski and J. White, "A trajectory piecewise-linear approach to model order reduction and fast simulation of nonlinear circuits and micromachined devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, pp. 155–170, Feb. 2003.

[76] S. P. Levitan, S. Member, J. A. Martínez, S. Member, T. P. Kurzweg, A. J. Davare, M. Kahrs, M. Bails, and D. M. Chiarulli, "System Simulation of Mixed-Signal Multi-Domain Microsystems With Piecewise Linear Models," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 22, no. 2, pp. 139–154, 2003.

[77] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: orthogonalization of concerns and platform-based design," *Computer-Aided Design of Integrated Circuits and*

Systems, IEEE Transactions on, vol. 19, pp. 1523–1543, Dec 2000.

[78] G. Schröpfer, G. Lorenz, S. Rouvillois, and S. Breit, "Novel 3-D modeling methods for virtual fabrication and EDA compatible design of MEMS via parametric libraries," *Journal of Micromechanics and Microengineering*, vol. 20, pp. 64003–64003, June 2010.

[79] A. Parent, A. Krust, G. Lorenz, and T. Piirainen, "A novel model order reduction approach for generating efficient nonlinear verilog-a models of mems gyroscopes," in *Inertial Sensors and Systems (ISISS), 2015 IEEE International Symposium on*, pp. 1–4, March 2015.

[80] S. Senturia, "CAD challenges for microsensors, microactuators, and microsystems," *Proceedings of the IEEE*, vol. 86, no. 8, pp. 1611–1626, 1998.

[81] J. Mehner, "Modal-Superposition-Based Nonlinear Model Order Reduction for MEMS Gyroscopes," in *System-level modeling of MEMS* (T. Bechtold, G. Schrag, and L. Feng, eds.), pp. 291–309, Wiley-VCH Verlag GmbH & Co. KGaA, 1st ed., 2013.

[82] B. Simeon, *Computational Flexible Multibody Dynamics*. Springer, 2010.

[83] K. D. Hjelmstad, *Fundamentals of structural mechanics*. Springer, 2nd ed., 2005.

[84] J. Korvink, E. Rudnyi, A. Greiner, and Z. Liu, "MEMS and NEMS Simulation," in *MEMS: A Practical Guide to Design, Analysis, and Applications* (J. Korvink and O. Paul, eds.), pp. 93–186, Springer, 2006.

[85] V. Rochus, D. J. Rixen, and J.-C. Golinval, "Monolithic modelling of electro-mechanical coupling in micro-structures," *International Journal for Numerical Methods in Engineering*, vol. 65, pp. 461–493, Jan. 2006.

[86] V. Rochus and C. Geuzaine, "A primal/dual approach for the accurate evaluation of the electromechanical coupling in MEMS," *Finite Elements in Analysis and Design*, vol. 49, pp. 19–27, Feb. 2012.

[87] W. C. Tang, *Electrostatic Comb Drive for Resonant Sensor and Actuator Applications*. PhD thesis, University of California, Berkeley, 1990.

[88] J.-M. Sallese and D. Bouvet, "Principles of space-charge based bi-stable MEMS: The junction-MEMS," *Sensors and Actuators A: Physical*, vol. 133, pp. 173–179, Jan. 2007.

[89] M. Sulfridge, T. Saif, N. Miller, and M. Meinhart, "Nonlinear Dynamic Study of a Bistable MEMS: Model and Experiment," *Journal of Microelectromechanical Systems*, vol. 13, pp. 725–731, Oct. 2004.

[90] R. Lin and W. Wang, "Structural dynamics of microsystems—current state of research and future directions," *Mechanical Systems and Signal Processing*, vol. 20, pp. 1015–1043, July 2006.

[91] C. Comi, A. Corigliano, G. Langfelder, A. Longoni, and A. Tocchio, "On the nonlinear behaviour of mems resonators," in *Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE), 2011 12th International Conference on*, pp. 1–6, April 2011.

[92] A. Tocchio, A. Caspani, G. Langfelder, A. Longoni, and E. Lasalandra, "A Pierce oscillator for MEMS resonant accelerometer with a novel low-power amplitude limiting technique," in *Frequency Control Symposium (FCS), 2012 IEEE International*, pp. 1–6, IEEE, 2012.

[93] A. P. Kovacs and R. A. Ibrahim, "The nonlinear vibration analysis of a clamped-clamped beam by a reduced multibody method," *Nonlinear Dynamics*, vol. 11, no. 2, pp. 121–141, 1996.

[94] M.-H. Bao, "Basic mechanics of beam and diaphragm structures," in *Micro Mechanical Transducers, Pressure Sensors, Accelerometers and Gyroscopes* (M.-H. Bao, ed.), vol. 8 of *Handbook of Sensors and Actuators*, pp. 23 – 88, Elsevier Science B.V., 2000.

[95] K.-J. Bathe, "Finite Element Procedures," Prentice-Hall, 1996.

[96] R. B. Karabalin, L. G. Villanueva, M. H. Matheny, J. E. Sader, and M. L. Roukes, "Stress-induced variations in the stiffness of micro- and nanocantilever beams," *Phys. Rev. Lett.*, vol. 108, p. 236101, Jun 2012.

[97] Q. Jing, *Modeling and Simulation for Design of Suspended MEMS*. PhD thesis, Carnegie Mellon University, 2003.

[98] K. Liateni, D. Moulinier, B. Affour, A. Delpoux, M. A. Maher, and J. M. Karam, ch. Moving MEMS into Mainstream Applications: The MEMSCAP Solution, pp. 544–555. Springer US, 2000.

[99] S. D. A. Hannot, *Modeling strategies for electro-mechanical microsystems with uncertainty quantification*. PhD thesis, TU Delft, 2010.

[100] G. I. Schüeller, "Developments in stochastic structural mechanics," *Archive of Applied Mechanics*, vol. 75, no. 10, pp. 755–773, 2006.

[101] ANSYS. http://www.ansys.com.

[102] Comsol. http://www.comsol.com.

[103] Coventor. http://www.coventor.com.

[104] J. V. Clark, D. Bindel, N. Zhou, S. Bhave, Z. Bai, J. Demmel, and K. S. J. Pister, "Sugar: Advancements in a 3D multi-domain simulation package for MEMS," in *Proceedings of the Microscale Systems: Mechanics and Measurements Symposium*, 2001.

[105] G. Lorenz, *Netzwerksimulation mikromechanischer Systeme*. PhD thesis, Universität Bremen, 1999.

[106] G. K. Fedder and Q. Jing, "A hierarchical circuit-level design methodology for microelectromechanical systems," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 46, no. 10, pp. 1309–1315, 1999.

[107] P. Schneider, C. Bayer, K. Einwich, and A. Köhler, "System level simulation - a core method for efficient design of MEMS and mechatronic systems," in *SSD-12*, 2012.

[108] T. Hou, C. L. Bris, A. Patera, and E. Zuazua, eds., *Reduced Order Methods for Modeling and Computational Reduction*. No. 9 in MS&A - Modeling, Simulation and Applications, Springer, 2014.

[109] J. Mehner, W. Dötzel, B. Schauwecker, and D. Ostergaard, "Reduced Order Modeling of Fluid Structural Interactions in MEMS based on Model Projection Techniques," in *Tranducers 03*, pp. 1840–1843, 2003.

[110] W. H. Schilders, "Model Order Reduction: Theory, Research Aspects and Applications," Springer, 2008.

[111] E. Gad, M. Nakhla, and R. Achar, ch. Model-Order Reduction of High-Speed Interconnects Using Integrated Congruence Transform, pp. 361–401. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[112] B. N. Bond, *Stability-Preserving Model Reduction for Linear and Nonlinear Systems Arising in Analog Circuit Applications by*. PhD thesis, Massachusetts Institute of Technology, 2010.

[113] A. H. Nayfeh, M. I. Younis, and E. M. Abdel-Rahman, "Reduced-Order Models for MEMS Applications," *Nonlinear Dynamics*, vol. 41, pp. 211–236, Aug. 2005.

[114] M. Kudryavtsev, E. Rudnyi, J. Korvink, D. Hohlfeld, and T. Bechtold, "Computationally efficient and stable order reduction methods for a large-scale model of MEMS piezoelectric energy harvester," *Microelectronics Reliability*, vol. 55, pp. 747–757, Apr. 2015.

[115] M. A. Cardoso, *Development and application of reduced-order modeling procedures for reservoir simulation*. PhD thesis, Standford University, 2009.

[116] D. M. Vasilyev, *Theoretical and practical aspects of linear and nonlinear model order reduction techniques*. PhD thesis, Massachusetts Institute of Technology, 2007.

[117] J. Lienemann, E. Bertarelli, A. Greiner, and J. G. Korvink, *Linear and Nonlinear Model Order Reduction for MEMS Electrostatic Actuators*, pp. 263–289. Wiley-VCH Verlag GmbH & Co. KGaA, 2013.

[118] U. Baur, P. Benner, A. Greiner, J. Korvink, J. Lienemann, and C. Moosmann, "Parameter preserving model reduction for MEMS applications," *Mathematical and Computational Modelling of Dynamical Systems*, vol. 17, no. 4, pp. 297–317, 2011.

[119] A. Sanginario, A. Mehdaoui, S. Zerbini, G. Schröpfer, and D. Demarchi, "New design methodology for MEMS-electronic-package co-design and validation for inertial sensor systems," in *Design, Test, Integration and Packaging of MEMS/MOEMS (DTIP), 2015 Symposium on*, pp. 1–6, April 2015.

[120] S. Bailey, "Comparison of VHDL , Verilog and SystemVerilog," 2003.

[121] "IEEE Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language," *IEEE Std 1800-2005*, pp. 1–648, 2005.

[122] J. Chen, S.-M. S. Kang, J. Zou, C. Liu, and J. Schutt-Aine, "Reduced-order modeling of weakly nonlinear mems devices with taylor-series expansion and arnoldi approach," *Microelectromechanical Systems, Journal of*, vol. 13, pp. 441–451, June 2004.

[123] E. Hung and S. Senturia, "Generating efficient dynamical models for microelectromechanical systems from a few finite-element simulation runs," *Microelectromechanical Systems, Journal of*, vol. 8, pp. 280–289, Sep 1999.

[124] J. Chen, *Modeling and Simulation of Integrated Microstructures and Systems*. PhD thesis, University of Illinois.

[125] T. Mähne, K. Kehr, A. Franke, J. Hauer, and B. Schmidt, "Creating Virtual Prototypes of Complex MEMS Transducers Using Reduced-Order Modelling Methods and VHDL-AMS," in *FDL 2005*, pp. 135–153, 2005.

[126] F. Bennini, J. Mehner, and W. Dötzel, "Computational Methods for Reduced-Order Modeling of Coupled Domain Simulations," in *Transducers01*, 2001.

[127] A. Kohler, S. Reitz, and P. Schneider, "Sensitivity analysis and adaptive multi-point multi-moment model order reduction in mems design," in *Design, Test, Integration and Packaging of MEMS/MOEMS (DTIP), 2011 Symposium on*, pp. 64–71, May 2011.

[128] M. Schlegel, F. Bennini, J. Mehner, G. Herrmann, D. Muller, and W. Dotzel, "Analyzing and simulation of MEMS in VHDL-AMS based on reduced-order FE models," *IEEE Sensors Journal*, vol. 5, pp. 1019–1026, Oct. 2005.

[129] L. D. Gabbay, J. E. Mehner, and S. D. Senturia, "Computer-Aided Generation of Nonlinear Reduced-Order Dynamic Macromodels -I: Non-Stress-Stiffened Case," *Journal of Microelectromechanical Systems*, vol. 9, no. 2, pp. 262–269, 2000.

[130] M. Schlegel, G. Herrmann, and D. Muller, "Application of the multi architecture modeling design method to system level MEMS simulation," in *DTIP 2003*, pp. 5–7, 2003.

[131] C. Hagleitner, T. Bonaccio, H. Rothuizen, J. Lienemann, D. Wiesmann, G. Cherubini, J. Korvink, and E. Eleftheriou, "Modeling, Design, and Verification for the Analog Front-End of a MEMS-Based Parallel Scanning-Probe Storage Device," *Solid-State Circuits, IEEE Journal of*, vol. 42, pp. 1779–1789, Aug 2007.

[132] J. Mehner, V. Kolchuzhin, I. Schmadlak, T. Hauck, G. Li, D. Lin, and T. Miller, "The influence of packaging technologies on the performance of inertial MEMS sensors," in *Solid-State Sensors, Actuators and Microsystems Conference, 2009. TRANSDUCERS 2009. International*, pp. 1885–1888, June 2009.

[133] M. Niessner, J. Iannacci, G. Schrag, and G. Wachutka, "Experimental analysis and modeling of the mechanical impact during the dynamic pull-in of rf-mems switches," in *Advanced Semiconductor Devices Microsystems (ASDAM), 2010 8th International Conference on*, pp. 267–270, Oct 2010.

[134] W. Bedyk, M. Niessner, G. Schrag, G. Wachutka, B. Margesin, and A. Faes, "Automated Extraction of Multi-Energy Domain Reduced-Order Models Demonstrated on Capacitive MEMS Microphones," in *Solid-State Sensors, Actuators and Microsystems Conference, 2007. TRANSDUCERS 2007. International*, pp. 1263–1266, June 2007.

[135] A. Parent, A. Krust, G. Lorenz, I. Favorskiy, and T. Piirainen, "Efficient Nonlinear Simulink Models of MEMS Gyroscopes Generated with a Novel Model Order Reduction Method," in *Solid-State Sensors, Actuators and Microsystems (TRANSDUCERS), 2015 Transducers - 2015 18th International Conference on*, pp. 2184–2187, June 2015.

[136] B. Bailey, G. Martin, and A. Piziali, *ESL design and verification: a prescription for electronic system-level methodology*. The Morgan Kaufmann series in systems on silicon, Morgan Kaufmann, 2007.

[137] S. Liao, S. Tjiang, and R. Gupta, "An efficient implementation of reactivity for modeling hardware in the scenic design environment," in *Design Automation Conference, 1997. Proceedings of the 34th*, pp. 70–75, June 1997.

[138] Accelera Systems Initiative. http://www.accellera.org.

[139] G. Arnout, "SystemC Standard," in *Design Automation Conference, 2000. Proceedings of the ASP-DAC 2000. Asia and South Pacific*, pp. 573–577, June 2000.

[140] S. Liao, "Towards a new standard for system-level design," in *Hardware/Software Codesign, 2000. CODES 2000. Proceedings of the Eighth International Workshop on*, pp. 2–6, May 2000.

[141] P. Panda, "SystemC - a modeling platform supporting multiple design abstractions," in *System Synthesis, 2001. Proceedings. The 14th International Symposium on*, pp. 75–80, 2001.

[142] H. D. Patel and S. K. Shukla, *SystemC Kernel Extensions for Heterogeneous System Modeling*. Kluwer Academic Publishers, 2004.

[143] K. Einwich, P. Schwarz, C. Grimm, and C. Meise, "SystemC-AMS: Rationales, State of the Art, and Examples," in *SystemC Methodologies and Applications* (W. Müller, W. Rosenstiel, and J. Ruf, eds.), pp. 273–297, Springer, 2004.

[144] M. Mefenza, F. Yonga, and C. Bobda, "Automatic UVM Environment Generation for Assertion-Based and Functional Verification of SystemC Designs," in *Microprocessor Test and Verification Workshop (MTV), 2014 15th International*, pp. 16–21, Dec 2014.

[145] T. Grötker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*. Springer US, 2002.

[146] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: an integrated electronic system design environment," *Computer*, vol. 36, pp. 45–52, April 2003.

[147] E. Lee, "Heterogeneous actor modeling," in *Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on*, pp. 3–12, Oct 2011.

[148] P. Herber, *A Framework for Automated HW/SW Co-Verification of SystemC Designs using Timed Automata*. PhD thesis, TU Berlin, 2010.

[149] W. Müller, W. Rosenstiel, and J. Ruf, *SystemC: methodologies and applications*. Kluwer Academic Publishers, 2003.

[150] A. Vachoux, C. Grimm, and K. Einwich, "Towards Analog and Mixed-Signal SOC Design with SystemC-AMS," in *2004 IEEE International Field-Programmable Technology, 2004. Proceedings. Conference on*, pp. 0–5, 2004.

[151] P. a. Hartmann, P. Reinkemeier, A. Rettberg, and W. Nebel, "Modelling control systems in SystemC AMS - Benefits and limitations," *2009 IEEE International SOC Conference (SOCC)*, pp. 263–266, Sept. 2009.

[152] *SystemC AMS extensions User's Guide*. OSCI, 2010.

[153] L. Scheffer, L. Lavagno, and G. Martin, *Electronic Design Automation for Integrated Circuits Handbook*, vol. EDA for IC System Design and Testing. Taylor & Francis/CRC Press, 2006.

[154] M. Barnasconi, K. Einwich, C. Grimm, T. Mähne, and A. Vachoux, "Advancing the SystemC Analog / Mixed-Signal (AMS) Extensions Introducing Dynamic Timed Data Flow," tech. rep., 2011.

[155] C. Reuther and K. Einwich, "A SystemC AMS Extension for Controlled Modules and Dynamic Step Sizes," in *FDL 2012 Forum on Specification and Design Languages*, pp. 90–97, 2012.

[156] R. L. Burden and J. D. Faires, *Numerical analysis*. Brooks/Cole, 9th ed., 2011.

[157] E. Yip, "A Note on the Stability of Solving a Rank-p Modification of a Linear System by the Sherman-Morrison-Woodbury Formula," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 2, pp. 507–513, 1986.

[158] F. N. Najm, *Circuit simulation*. Wiley, 2010.

[159] L. Andrade, T. Mähne, A. Vachoux, C. Ben Aoun, F. Pecheux, and M.-M. Louerat, "Pre-simulation symbolic analysis of synchronization issues between discrete event and timed data flow models of computation," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*, pp. 1671–1676, March 2015.

[160] L. Andrade, *Principles and Implementation of a Generic Synchronization Interface between SystemC AMS Models of Computation for the Virtual Prototyping of Multi-Disciplinary Systems*. PhD thesis, Université Pierre & Marie Curie.

[161] F. Madlener, H. G. Molter, and S. A. Huss, "SC-DEVS: An efficient SystemC extension for the DEVS model of computation," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pp. 1518–1523, April 2009.

[162] F. E. Cellier and E. Kofman, *Continuous System Simulation*. Springer, 2006.

[163] H. Al-Junaid and T. Kazmierski, "Analogue and mixed-signal extension to systemc," *IEE Proceedings - Circuits, Devices and Systems*, vol. 152, pp. 682–690, Dec 2005.

[164] C. Zhao and T. J. Kazmierski, "An extension to systemc-a to support mixed-technology systems with distributed components," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pp. 1–6, March 2011.

[165] C. Zhao and T. Kazmierski, "Systemc-a modelling of mixed-technology systems with distributed behaviour," in *Specification Design Languages (FDL 2010), 2010 Forum on*, pp. 1–6, Sept 2010.

[166] T. Mähne, *Efficient Modelling and Simulation Methodology for the Design of Heterogeneous Mixed-Signal Systems on Chip*. PhD thesis, École Polytechnique Fédérale de Lausanne (EPFL), 2011.

[167] F. Cenni, O. Guillaume, M. Diaz-Nava, and T. Mähne, "SystemC-AMS/MDVP-based modeling for the virtual prototyping of MEMS applications," in *Design, Test, Integration and Packaging of MEMS/-MOEMS (DTIP), 2015 Symposium on*, pp. 1–6, April 2015.

[168] F. Herrera and E. Villar, "A framework for embedded system specification under different models of computation in systemc," in *Proceedings of the 43rd Annual Design Automation Conference*, DAC '06, pp. 911–914, ACM, 2006.

[169] F. Herrera, E. Villar, C. Grimm, M. Damm, and J. Haase, *Embedded Systems Specification and Design Languages: Selected contributions from FDL'07*, ch. Heterogeneous Specification with HetSC and SystemC-AMS: Widening the Support of MoCs in SystemC, pp. 107–121. Springer Netherlands, 2008.

[170] A. Antoniou, *Digital Signal Processing*. Mc Graw Hill, 2006.

[171] Norman S. Nise, *Control Systems Engineering*. Wiley, 6th ed., 2011.

[172] F. Paugnat, L. Fesquet, and K. Morin-Allory, "Model of a Simple yet effective Operational Amplifier - A Generic Model Managing the Nonlinearities with No Topological Assumptions," in *SMACD 2012*, pp. 165–168, 2012.

[173] E. Markert, M. Dienel, G. Herrmann, and U. Heinkel, "SystemC-AMS Assisted Design of an Inertial Navigation System," *IEEE Sensors Journal*, vol. 7, pp. 770–777, May 2007.

[174] S. Adhikari and C. Grimm, "Modeling Switched Capacitor Sigma Delta Modulator Nonidealities in SystemC-AMS," *FDL 2010*, pp. 14–16, 2009.

[175] A. Schroth, T. Blochwitz, and G. Gerlach, "Simulation of a complex sensor system using coupled simulation programs," *Sensors and Actuators A: Physical*, vol. 54, no. 1–3, pp. 632 – 635, 1996.

[176] B. Tutuianu, D. Lehther, M. Pandey, and R. Baldick, ch. Efficient RLC Macromodels for Digital IC Interconnect, pp. 293–304. Springer US, 2000.

[177] S. Senturia, N. Aluru, and J. White, "Simulating the Behavior of MEMS Devices," *IEEE Computational Science and Engineering*, pp. 30–43, 1997.

[178] W. Borutzky, *Bond Graph Methodology: Development and Analysis of Multidisciplinary Dynamic System Models*. Springer-Verlag, 2010.

[179] H. A. C. Tilmans, "Equivalent circuit representation of electromechanical transducers: I. Lumped-parameter systems," *Journal of Micromechanics and Microengineering*, vol. 6, no. 1, p. 157, 1996.

[180] H. A. C. Tilmans, "Equivalent circuit representation of electromechanical transducers: II. Distributed-parameter systems," *Journal of Micromechanics and Microengineering*, vol. 7, no. 4, p. 285, 1997.

[181] ch. Lumped-element System Dynamics, pp. 149–180. Springer US, 2001.

[182] B. F. Romanowicz, M. H. Zaman, S. F. Bart, V. L. Rabinovich, I. Tchertkov, C. Hsu, and J. R. Gilbert, ch. A Methodology and Associated CAD Tools for Support of Concurrent Design of MEMS, pp. 636–648. Springer US, 2000.

[183] P. Breedveld, *Bond Graph Modelling of Engineering Systems: Theory, Applications and Software Support*. Springer, 2011.

[184] Y. Liu, F. Bao, and Q. Cai, "Research on Bond Graph Simulation for Dynamic Performance Analysis of MEMS," in *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on*, vol. 2, pp. 1106–1109, May 2010.

[185] K. Popovici and P. J. Mosterman, *Real-time simulation technologies: principles, methodologies, and applications*. CRC Press, 2012.

[186] G.-R. Duan, *Analysis and Design of Descriptor Linear Systems*, vol. 23 of *Advances in Mechanics and Mathematics*. Springer New York, 2010.

[187] K. Ogata, *Modern control engineering*. Prentice-Hall electrical engineering series. Instrumentation and controls series, Prentice-Hall, 5th ed ed., 2010.

[188] P. H. Lewis and R. Kernan, *Basic Control Systems Engineering*. Prentice Hall, 1997.

[189] O. Brand and G. K. Fedder, eds., *CMOS-MEMS*. No. 2 in Advanced micro & nanosystems, Wiley-VCH, 2005.

[190] F. Ayazi, M. F. Zaman, and A. Sharma, "Vibrating gyroscopes," in *Comprehensive Microsystems* (Y. B. G. T. Zappe, ed.), pp. 181 – 208, Elsevier, 2008.

[191] H. Cao and H. Li, "Investigation of a vacuum packaged MEMS gyroscope architecture's temperature robustness," *International Journal of Applied Electromagnetics and Mechanics*, vol. 41, no. 4, pp. 495–506, 2013.

[192] D. Vink, *Aspect of Bond Graph Modeling in Control*. PhD thesis, University of Glasgow, 2005.

[193] Eigen, *C++ template library for linear algebra*. http://eigen.tuxfamily.org. version 3.2.4.

[194] M. Egretzberger, F. Mair, and A. Kugi, "Model-based control concepts for vibratory {MEMS} gyroscopes," *Mechatronics*, vol. 22, no. 3, pp. 241 – 250, 2012. Special Issue on Mechatronic Systems for Micro- and Nanoscale Applications.

[195] S. Das, *Mechatronic Modeling and Simulation Using Bond Graphs*. CRC Press, 2009.

[196] B. Vernay, A. Krust, T. Mähne, G. Schröpfer, F. Pêcheux, and M. M. Louërat, "A novel method of MEMS system-level modeling via multi-domain virtual prototyping in SystemC-AMS," in *Proceedings of the EDAA/ACM SIGDA PhD Forum at DATE*, March 2014.

[197] B. Vernay, A. Krust, G. Schröpfer, F. Pecheux, and M.-M. Louerat, "SystemC-AMS simulation of a biaxial accelerometer based on MEMS model order reduction," in *Design, Test, Integration and Packaging of MEMS/MOEMS (DTIP), 2015 Symposium on*, pp. 1–6, Apr. 2015.

[198] D. Amsallem and C. Farhat, "On the stability of projection-based linear reduced-order models: Descriptor vs non-descriptor forms," *ArXiv e-prints*, Sept. 2012.

[199] A. Tocchio, C. Comi, G. Langfelder, A. Corigliano, and A. Longoni, "Enhancing the Linear Range of MEMS Resonators for Sensing Applications," *IEEE Sensors Journal*, vol. 11, no. 12, pp. 3202–3210, 2011.

[200] C. Comi, A. Corigliano, G. Langfelder, A. Longoni, A. Tocchio, and B. Simoni, "A new biaxial silicon resonant micro accelerometer," in *Micro Electro Mechanical Systems (MEMS), 2011 IEEE 24th International Conference on*, pp. 529–532, Jan 2011.

[201] A. Caspani, C. Comi, A. Corigliano, G. Langfelder, and A. Tocchio, "Compact biaxial micromachined resonant accelerometer," *Journal of Micromechanics and Microengineering*, vol. 23, p. 105012, Oct. 2013.

[202] D. F. Rossi, W. G. Ferreira, W. J. Mansur, and A. F. G. Calenzani, "A review of automatic time-stepping strategies on numerical time integration for structural dynamics analysis," *Engineering Structures*, vol. 80, pp. 118–136, Dec. 2014.

[203] M. Reddy, *API design for C++*. Morgan Kaufmann, 2011.

[204] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern Oriented Software Architecture - A System of Patterns*, vol. 1. Wiley, 2001.

[205] M. Richards, *Software Architecture Patterns*. O'Reilly Media, 2015.

[206] A. Ezust and P. Ezust, *An Introduction to Design Patterns in C++ with Qt 4*. Prentice Hall, 2007.

[207] H. Sutter and A. Alexandrescu, *C++ coding standards: 101 rules, guidelines, and best practices*. Addison-Wesley, 2005.

[208] G. Booch, R. A. Maksimchuk, M. W. Engle, B. J. Young, J. Conallen, and K. A. Houston, *Object-Oriented Analysis and Design with Applications*, vol. 40. Addison-Wesley, 3rd ed., Mar. 2001.

[209] R. Görgen, P. A. Hartmann, and W. Nebel, "Automated SystemC Model Instantiation with modern C++ Features and sc_vector," in *Proceedings of DVCon Europe 2015*, Accellera Systems Initiative, 11 2015.

[210] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical Mathematics*. Springer, 2000.

[211] B. McNamara and Y. Smaragdakis, "Functional programming in C++," *ACM SIGPLAN Notices*, vol. 35, no. 9, pp. 118–129, 2000.

[212] T. Mähne and A. Vachoux, "Supporting dimensional analysis in SystemC-AMS," in *Behavioral Modeling and Simulation Workshop, 2009. BMAS 2009. IEEE*, pp. 108–113, Sept 2009.

[213] Boost library. http://www.boost.org/. version 1.59.0.

[214] K. Janschek and K. Richmond, *Mechatronic Systems Design: Methods, Models, Concepts*, ch. Simulation Issues, pp. 171–210. Springer Verlag, 2012.

[215] N. Bombieri, D. Drogoudis, G. Gangemi, R. Gillon, E. Macii, M. Poncino, S. Rinaudo, F. Stefanni, D. Trachanis, and M. van Helvoort, "SMAC: Smart Systems Co-design," in *Digital System Design (DSD), 2013 Euromicro Conference on*, pp. 253–259, Sept 2013.

[216] R. Gillon, G. Gangemi, M. Grosso, F. Fummi, and M. Poncino, "Multi-domain simulation as a foundation for the engineering of smart systems: Challenges and the smac vision," in *Electronics, Circuits and Systems (ICECS), 2014 21st IEEE International Conference on*, pp. 858–861, Dec 2014.

[217] M. Grosso, G. Gangemi, S. Rinaudo, F. Cenni, M. Crepaldi, A. Sanginario, and D. Demarchi, "Enabling smart system design with the smac platform," in *Design, Test, Integration and Packaging of MEMS/-MOEMS (DTIP), 2015 Symposium on*, pp. 1–6, April 2015.

[218] G. Schröpfer, G. Lorenz, A. Krust, B. Vernay, S. Breit, A. Mehdaoui, and A. Sanginario, "MEMS System-Level Modeling and Simulation in Smart Systems," in *Smart Systems Integration and Simulation* (N. Bombieri, M. Poncino, and G. Pravadelli, eds.), pp. 145–168, Springer International Publishing, 2016.

[219] S. Vinco, M. Lora, and M. Zwolinski, "Conservative behavioural modelling in SystemC-AMS," in *Specification and Design Languages (FDL), 2015 Forum on*, pp. 1–8, IEEE, 2015.

[220] A. Corigliano, B. D. Masi, A. Frangi, C. Comi, A. Villa, and M. Marchi, "Mechanical characterization of polysilicon through on-chip tensile tests," *Journal of Microelectromechanical Systems*, vol. 13, pp. 200–219, April 2004.

[221] P. Sehnalova, "Stability and Convergence of Numerical Computations," *Information Sciences and Technologies Bulletin of the ACM Slovakia*, vol. 3, no. 3, pp. 26–35, 2011.

[222] V. Sanz, A. Urquia, F. E. Cellier, and S. Dormido, "System modeling using the Parallel DEVS formalism and the Modelica language," *Simulation Modelling Practice and Theory*, vol. 18, no. 7, pp. 998 – 1018, 2010.

[223] B. Fraeijs de Veubeke, *Displacement and equilibrium models in the finite element method*, vol. 52, pp. 145–197. John Wiley & Sons, 1965.

[224] J. He, "Equivalent theorem of Hellinger-Reissner and Hu-Washizu variational principles," *Journal of Shanghai University (English Edition)*, vol. 1, no. 1, pp. 36–41, 1997.

Appendices

Appendix A

Functionals of coupled electromechanical systems

Let the mechanical and electric domains be symbolized by Ω_m and Ω_e (Fig. A.1). They are delimited by boundaries respectively denoted by $\delta\Omega_m = \Gamma_s \cup \Gamma_t$ and $\delta\Omega_e = \Gamma_e \cup \Gamma_d$. We define **S** as the mechanical strain tensor , **T** as the mechanical stress tensor, **E** as the electric field, and **D** as the electric displacement. The constitutive equations define the material response in terms of displacement, i.e., strains, to applied forces or fields as follows:

$$\mathbf{T} = \mathbf{H}\,\mathbf{S} \quad \text{and} \quad \mathbf{D} = \varepsilon\,\mathbf{E}\,, \tag{A.1}$$

where **H** is the stiffness matrix (Hooke's law) and ε is the permittivity matrix.

The compatibility equations ensure the continuity of the displacement field. By analogy, in electrostatics, Faraday's law is applied to obtain the electric potential, i.e., $\nabla \times \mathbf{E} = 0$ in Ω_e. The equations are summarized below:

$$\begin{cases} \mathbf{S} = \nabla_s \mathbf{u} & \text{in } \Omega_m \\ \mathbf{u} = \overline{\mathbf{u}} & \text{on } \Gamma_s \end{cases} \quad \text{and} \quad \begin{cases} \mathbf{E} = -\nabla\phi & \text{in } \Omega_e \\ \phi = \overline{\phi} & \text{on } \Gamma_e \end{cases}, \tag{A.2}$$

where **u** and ϕ are, respectively, the mechanical displacement vector and the electric scalar potential (the prescribed values are over-lined). The operator ∇ is the gradient operator. ∇_s is the matrix operator for mechanics.

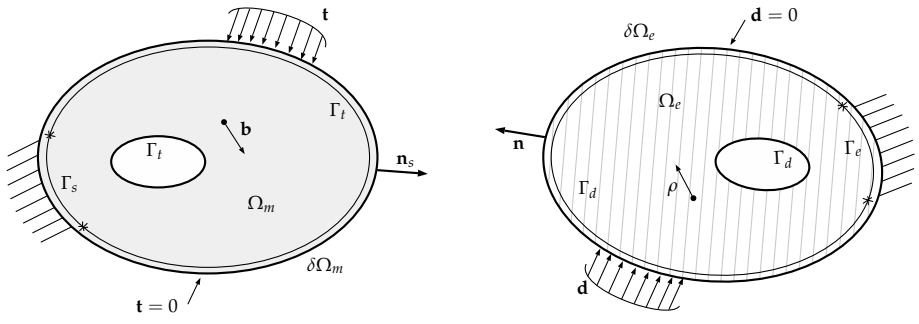


Fig. A.1: Domain definition for the mechanical and the electrostatic problems.

The equilibrium equations characterize the continuity of displacement field regarding external imposed forces or fields. The term *equilibrium* is extensively applied to electrostatics and refers to the Gauss's law, second term in (A.3). The equations for both domains are defined as follows:

$$\begin{cases} \nabla_s \cdot \mathbf{T} = \overline{\mathbf{f}} & \text{in } \Omega_m \\ \mathbf{n}_s \cdot \mathbf{T} = \overline{\mathbf{t}} & \text{on } \Gamma_t \end{cases} \text{ and } \begin{cases} \nabla \cdot \mathbf{D} = \overline{\rho} & \text{in } \Omega_e \\ \mathbf{n} \cdot \mathbf{D} = \overline{d} & \text{on } \Gamma_d \end{cases}, \tag{A.3}$$

where $\overline{\mathbf{f}}$ and $\overline{\rho}$ are respectively prescribed body forces and charge densities, $\overline{\mathbf{t}}$ and $\overline{d}$ are imposed surface tensions and normal electric displacement, $\mathbf{n}$ and $\mathbf{n}_s$ denote the normal vectors to boundary.

Energy functionals are commonly employed to solve the previous boundary value problems. To this aim, the principle of virtual work $\tilde{\mathscr{W}}$ is applied as as a direct consequence of the fundamental theorem of the calculus of variations as depicted below:

$$\tilde{\mathscr{W}} = \int_\Omega \overline{\mathbf{f}} \cdot \tilde{\mathbf{u}} \, \mathrm{d}\Omega + \int_\Gamma \overline{\mathbf{t}} \cdot \tilde{\mathbf{u}} \, \mathrm{d}\Gamma \,, \tag{A.4}$$

where the virtual displacements $\tilde{\mathbf{u}}$ are the values of the corresponding field either inside the body, i.e. the first term, or on the surface of the body, i.e., the second term.

In addition, the corollary of Vainsberg's theorem, also known as the *energy extremum principle*, states that if an energy principle exists for a certain differential equation, then one can find the energy functional from the associated virtual-work functional [83]. Therefore, energy principles can be formed with functionals that consider not only the displacement, but also the stress or strain field as the independent function.

The previous variational principle yields a reference 3-field functional initially defined by Fraeijs de Veubeke [223] and extended by the Hu-Washizu energy functional [224]. As depicted in (A.1), the constitutive equations relate strain field to stress field in both domains. By merging these equations in Hu-Washizu energy functional, Hellinger-Reisner variational principle [224] defines a 2-field energy functional that considers only the stress and displacement fields. Single field functionals are finally obtained by merging either the compatibility (A.2) or the equilibrium (A.3) equations in the Hellinger-Reisner energy functional.

These complementary approaches define respectively primal and dual energy functionals that are necessary to estimate the total energy of the system. On the one hand, the *primal* energy functionals, $\mathscr{F}_m$ and $\mathscr{F}_e$, are respectively function of the mechanical displacement vector and the electric scalar potential directly issued from the compatibility equations (A.2). They define the kinematic behavior of the system as they are based on the strain field. On the other hand, the *dual* energy functionals, i.e. $\tilde{\mathscr{F}}_m$ and $\tilde{\mathscr{F}}_e$, are focused on the static definition of the system. They indeed refer to the stress field, as given in the equilibrium equations. The definition of these functionals is summarized in Figure A.2.

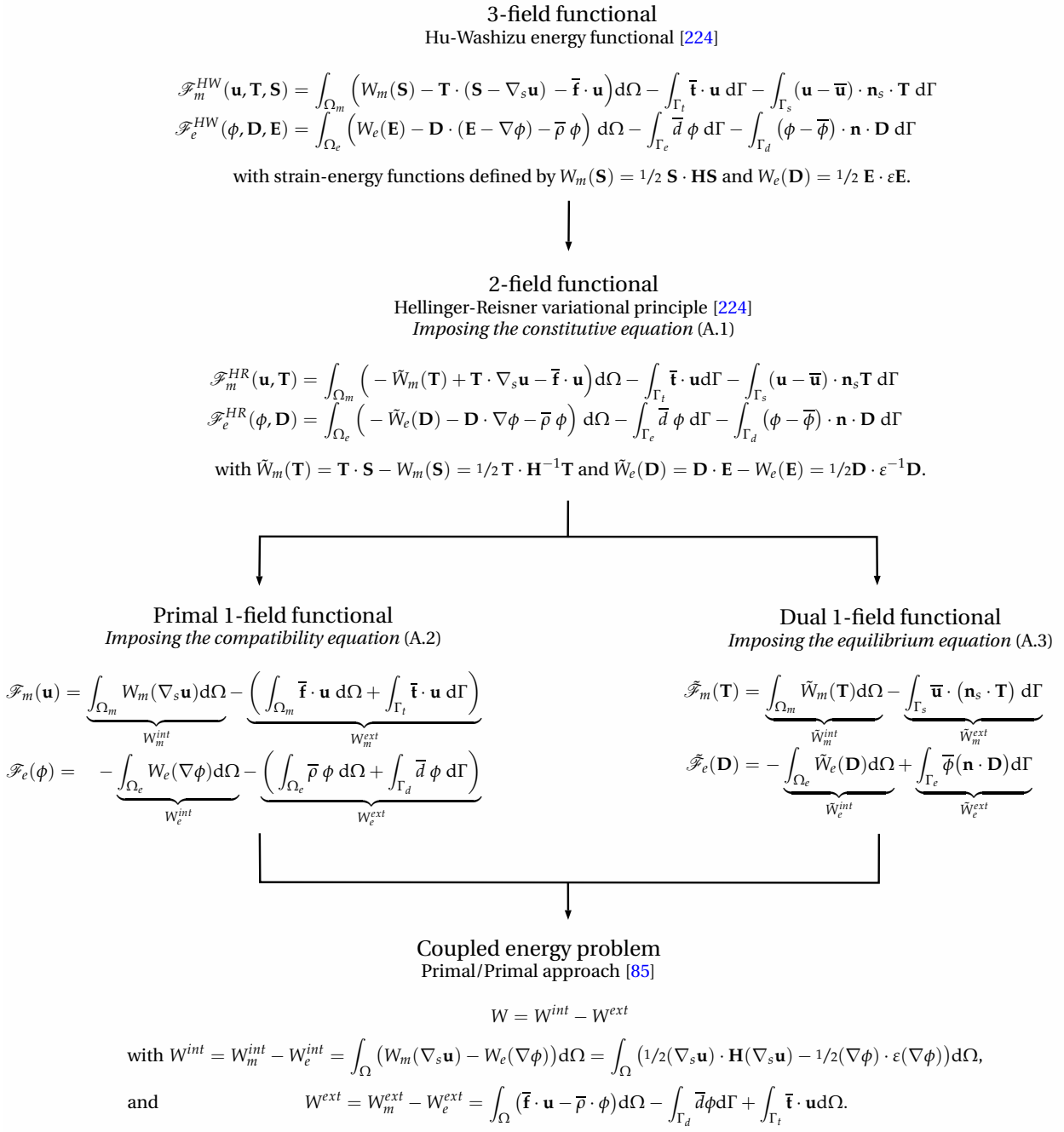


Fig. A.2: Definition of the primal and dual energy functionals based on the successive application of governing equations and boundary conditions in both mechanical (m) and electrostatic (e) domains. The electromechanical coupling is defined through energy density combining each contribution [85].

Appendix B

MEMS+ model definition

MEMS+ models are created in specific modeling and simulation plugins, directly inspired from the manufacturing and design of MEMS devices. These steps successively define the material properties (Figure B.1), the layered process (Figure B.2), the elementary components (Figure B.3) and the 3-D design of the device (Figure B.4). Moreover, *MEMS*+ allows static simulations like DC or modal analyses in order to verify the behavioral definition of the device (Figure B.5). Each plugin has a specific interface in which the user can define the properties and elements of the model. The system-level models are built upon the standard component library and the user'predefined components. Additional information on the environment like the temperature or pressure are required to provide consistent simulation results.

We illustrate below the use of the different *MEMS*+ plugins with the accelerometer LIS332AR used in Section 4.2 and defined in [201]. This reference provides information about the process specification, here the THELMA® process developed by ST Microelectronics. The geometry definition is also described by a microscope scan and a 3-D view of a first model defined in Comsol. Moreover, the inertial description as well as the definition of the driving and sensing modes are provided. We can finally compare our model to the reference one through analyses directly run in *MEMS*+, e.g., DC or AC analyses, and thus verify the expected modal response in terms of frequency and quality factors.

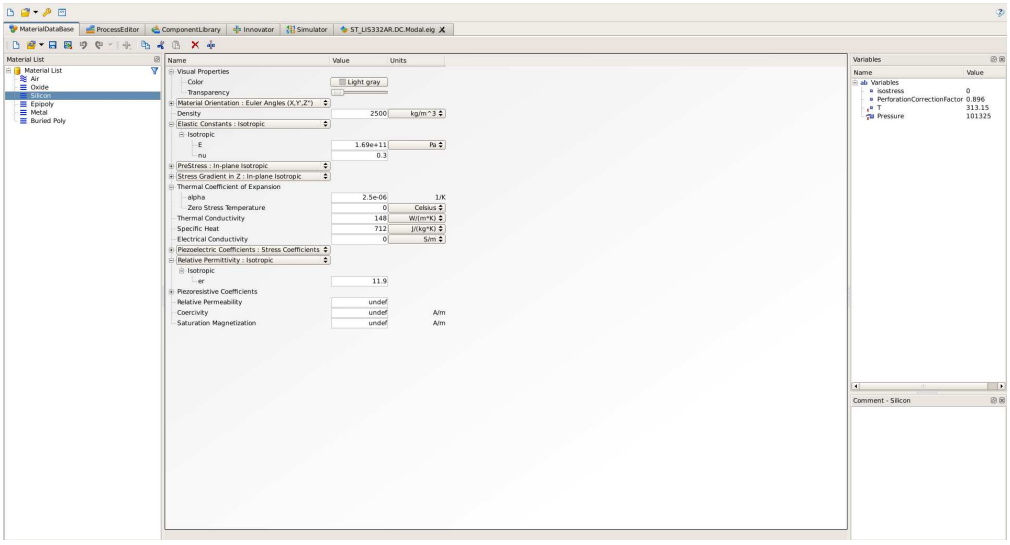


Fig. B.1: The material database stores information characterizing the different materials used during the process (density, inertia, crystal orientation, thermal or electrical conductivity…).

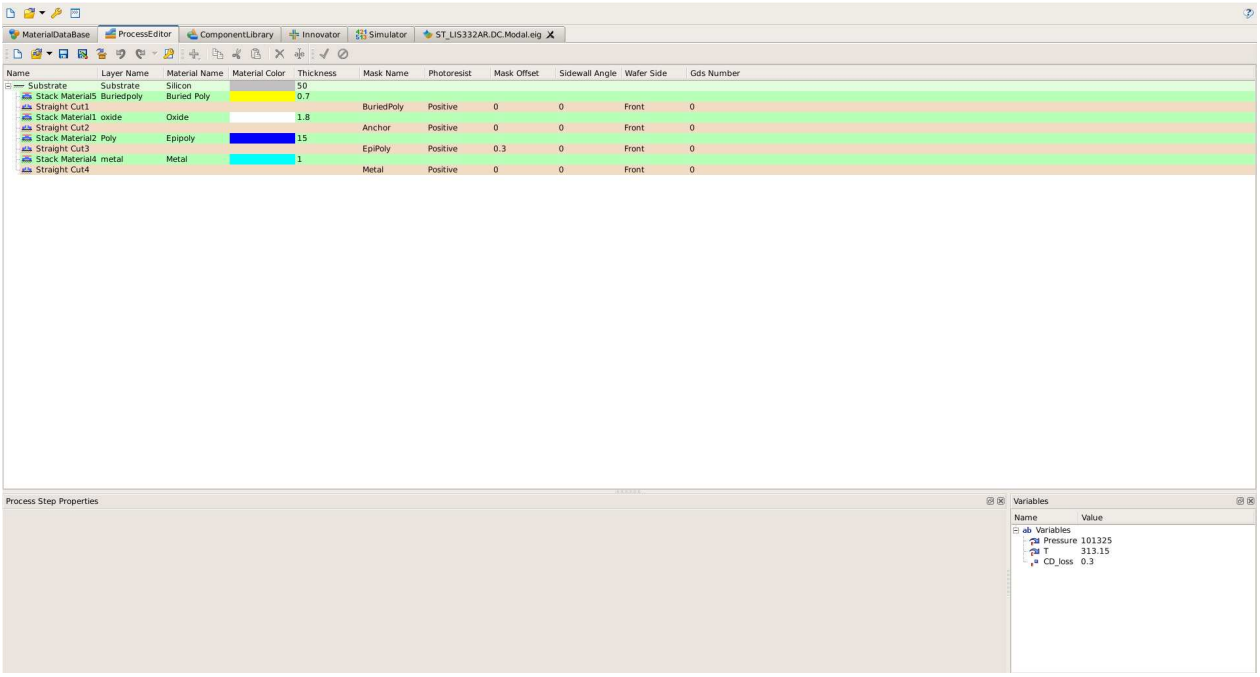


Fig. B.2: Each layer is described in the process editor by its thickness and the associated material.

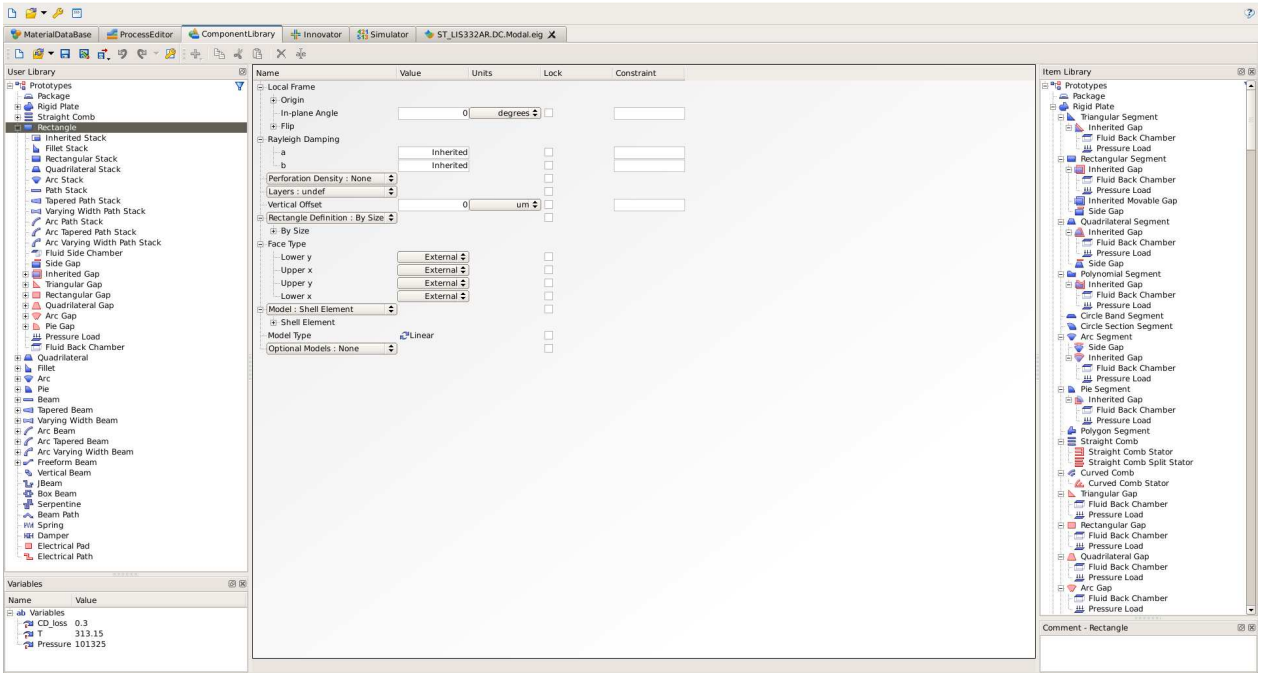


Fig. B.3: The component library regroups the standard or user-defined elements used to design novel devices. These components are fully configurable and associated to specific geometry and material properties.

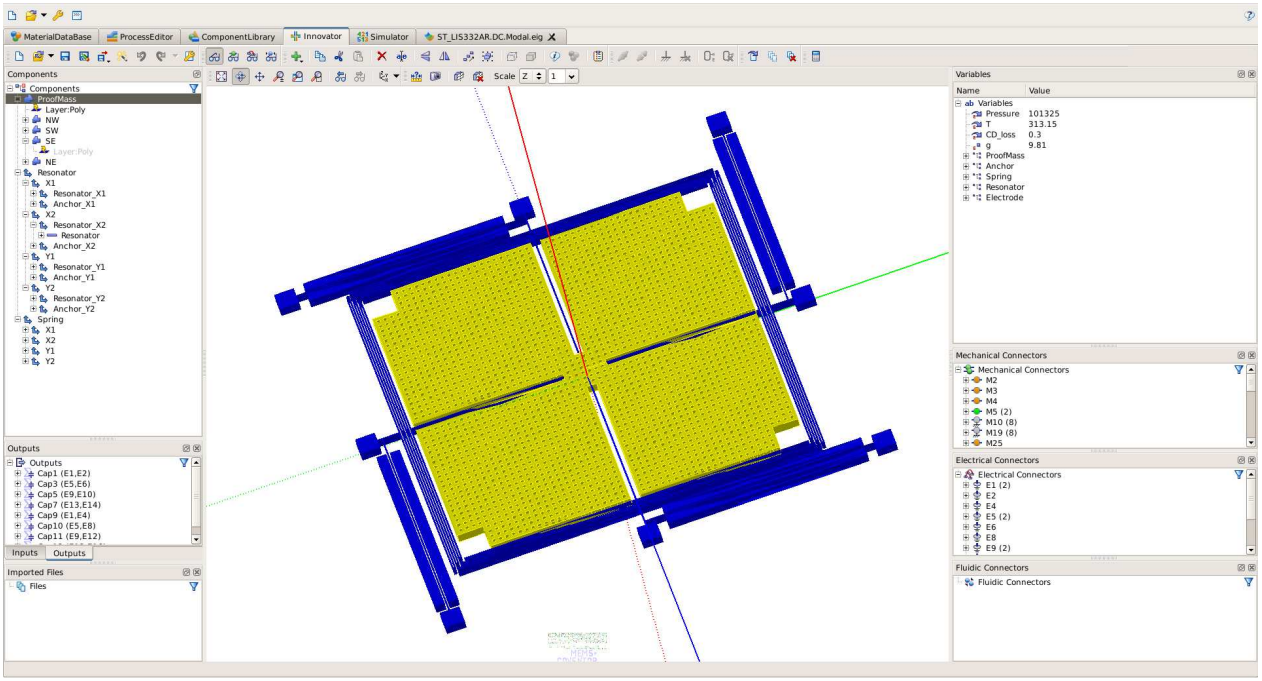


Fig. B.4: The main interface consist in a 3-D view where the user can assemble multiple elements in order to create the device geometrically. Here the movable plate is selected in the component list on the left and highlighted in the 3-D view.

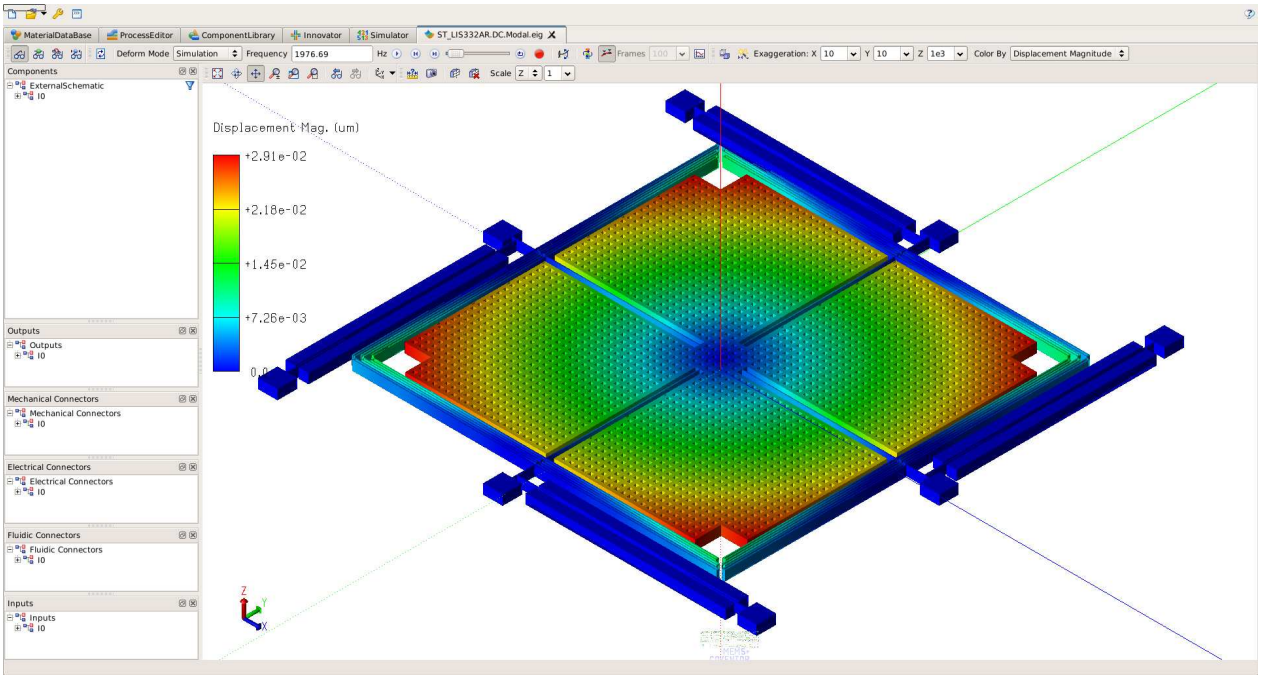


Fig. B.5: Different analyses (DC, DC Sweep, AC…) or exports (Matlab, Verilog-A…) can be performed from the Simulator plugin and the results can be displayed in additional windows like here for the modal analysis of the accelerometer LIS332AR.

Appendix C

Modeling procedures

The reduced models in *MEMS+* start from a system like (4.11) to which nonlinear force terms are added. We detail hereafter the MOR implemented independently from this work and developed in parallel to the API. This section is directly adapted from [218].

Reduction matrix

Definition

The accuracy and stability of the reduced system greatly depends on the choice of the matrix $\mathbf{V}$. The modal superposition method [81] ensures that certain eigenvalues of the system are preserved and stability of the reduced linear model is ensured, as long as the full model is stable. A vector $\mathbf{x}$ can be decomposed into:

$$\mathbf{x} = \sum_{i}^{n} \phi_i x_i \tag{C.1}$$

where the $\phi_i \in \mathbb{C}^n$ are the eigenvectors of the system.

Suppose, for simplicity, that we are interested only in the first two modes. Each mode has a real part $\mathscr{R}$ and an imaginary part $\mathscr{I}$. The columns of the matrix $\mathbf{V}$ are then $\mathscr{R}(\phi_1), \mathscr{I}(\phi_1), \mathscr{R}(\phi_2), \mathscr{I}(\phi_2)$. For accuracy reasons, we also want to make sure that the linearization point $\mathbf{x}_0$ belongs to the space spanned by the columns of $\mathbf{V}$ by simply including $\mathbf{x}_0$ as a new column of $\mathbf{V}$. Doing so, we can no longer guarantee stability of the system. It is however possible to preserve the block structure of the mechanical part of the system with a simple modification of $\mathbf{V}$. The full matrices $\mathbf{A}$ and $\mathbf{E}$ have the structure:

$$\mathbf{A} = \begin{bmatrix} \mathbf{K} & \mathbf{D} & \star \\ 0 & \mathbf{I} & \star \\ \star & \star & \star \end{bmatrix} \quad \text{and} \quad \mathbf{E} = \begin{bmatrix} 0 & \mathbf{M} & \star \\ \mathbf{I} & 0 & \star \\ \star & \star & \star \end{bmatrix} \tag{C.2}$$

with $\mathbf{K}$, $\mathbf{M}$, and $\mathbf{D}$ the stiffness, mass, and damping matrices. The stars $\star$ represent the electrical part of the system. The corresponding state vector can be written as:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}^p \\ \mathbf{x}^v \\ \mathbf{x}^e \end{pmatrix} \tag{C.3}$$

where $\mathbf{x}_p$ contains the displacement DoFs, $\mathbf{x}^v$ the velocity DoFs, and $\mathbf{x}^e$ the electrical DoFs. As the dimension of $\mathbf{x}^e$ is very small (only a few DoFs), we only need to reduce $\mathbf{x}^p$ and $\mathbf{x}^v$.

In the same fashion, we define $\mathbf{V}^p$ as the first rows of the original matrix $\mathbf{V}$. We ensure that ${\mathbf{V}^p}^T\mathbf{V}^p = \mathbf{I}$ with a Singular Value Decomposition (SVD) on $\mathbf{V}^p$. Consider the following reduction matrix:

$$\mathbf{V} = \begin{bmatrix} \mathbf{V}^p & 0 & 0 \\ 0 & \mathbf{V}^p & 0 \\ 0 & 0 & \mathbf{I} \end{bmatrix}. \tag{C.4}$$

This preserves the original block structure of $\mathbf{A}$ and $\mathbf{E}$:

$$\tilde{\mathbf{A}} = \begin{bmatrix} \tilde{\mathbf{K}} & \tilde{\mathbf{D}} & \star \\ 0 & \mathbf{I} & \star \\ \star & \star & \star \end{bmatrix} \quad \text{and} \quad \mathbf{E} = \begin{bmatrix} 0 & \tilde{\mathbf{M}} & \star \\ \mathbf{I} & 0 & \star \\ \star & \star & \star \end{bmatrix} \tag{C.5}$$

with $\tilde{\mathbf{K}} = {\mathbf{V}^p}^T\mathbf{D}\mathbf{V}^p$ and $\tilde{\mathbf{M}} = {\mathbf{V}^p}^T\mathbf{M}\mathbf{V}^- p$. The stars $\star$ are reduced as well. Compared to other methods, the chosen reduction matrix is expensive as it has twice as many columns, leading to twice as many unknowns in the reduced model. However, the gain in stability proved to justify the extra computational cost.

Higher-order terms

A wide range of methods can be applied to reduce a full model to a linear model and this field of research is now quite mature. However, a certain number of physical behaviors are nonlinear and cannot be recovered by such a simple model. For example, the electrostatic force is known to be quadratic with respect to voltage. Other examples are the rotating inertial forces (e.g., the Coriolis force), which a linear model cannot represent. Accurately modeling these nonlinearities in a ROM is a challenging task.

Interpolation between different linear models can be used to recover nonlinearities that are added to the reduced model, especially the electrostatic softening effect. This approach, although very accurate, tends to create very large models especially if multiple electrodes are involved [79]. A well-known application of interpolation is the TPWL method [75], which is quite successful at recovering mechanical nonlinearities of the MEMS structure. Another approach is to add polynomial terms in an analytical or a semi-analytical manner. Both approaches can successfully be mixed into a single reduced-order model [135].

System dynamics

Electrostatic forces

The input vector $\mathbf{u}$ can be decomposed into the voltage inputs $\mathbf{u}_e$ and all other inputs $\mathbf{u}_*$. For a gyroscope, $\mathbf{u}_* \in \mathbb{R}^3$ notably contains the angular rates. Decomposing the matrix $\tilde{\mathbf{B}}$ accordingly yields $\tilde{\mathbf{B}}\mathbf{u} = \tilde{\mathbf{B}}_e\mathbf{u}_e + \tilde{\mathbf{B}}_*\mathbf{u}_*$. Electrostatic forces are known to be quadratic terms with respect to voltage and can be written as:

$$\mathbf{F}_e(\mathbf{x}, \mathbf{u}_e) = \begin{bmatrix} \mathbf{u}_e^T C_1(\mathbf{x})\mathbf{u}_e \\ \vdots \\ \mathbf{u}_e^T C_n(\mathbf{x})\mathbf{u}_e \end{bmatrix}. \tag{C.6}$$

where $\mathbf{C}_1, \ldots, \mathbf{C}_n$ are the corresponding capacitance matrices. The dependence of these matrices with respect to displacement is nontrivial and needs to be approximated in a reduced model. In a first approximation, these matrices can be considered to be constant. This assumption removes cross terms between displacement and voltage which are known to cause the electrostatic softening effect.

In order to accurately recover electrostatic softening terms at a lower cost, i.e., without interpolation, we consider the matrices C_i to depend linearly on displacement. They are computed by finite differentiation of $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$. With $\tilde{\mathbf{C}}_i(\tilde{\mathbf{x}}) = \mathbf{C}_i(\mathbf{V}\tilde{\mathbf{x}})$, the electrostatic force term is reduced to:

$$\mathbf{V}^T\mathbf{F}_e(\tilde{\mathbf{x}}, \mathbf{u}_e) = \begin{bmatrix} \mathbf{u}_e^T\tilde{C}_1(\tilde{\mathbf{x}})\mathbf{u}_e \\ \vdots \\ \mathbf{u}_e^T\tilde{C}_n(\tilde{\mathbf{x}})\mathbf{u}_e \end{bmatrix}. \tag{C.7}$$

The model should no longer contain terms linear with respect to voltage. We also need to remove these terms from the matrix $\tilde{\mathbf{B}}$, which corresponds to removing $\tilde{\mathbf{B}}_e\mathbf{u}_e$. The resulting system of the reduced model is now:

$$\tilde{\mathbf{E}}\dot{\tilde{\mathbf{x}}} = \tilde{\mathbf{A}}\tilde{\mathbf{x}} + \tilde{\mathbf{B}}_*\mathbf{u}_* + \tilde{\mathbf{F}}_e(\tilde{\mathbf{x}}, \mathbf{u}_e) + \tilde{\mathbf{r}}, \tag{C.8}$$

This is enough to ensure that the first derivative with respect to displacement is exact at the point of extraction, yielding a correct softening.

Mechanical forces

The mechanical nonlinearity of the device behavior is mostly related to three inertial rotating fictitious forces to consider, i.e., the centrifugal force $\mathbf{F}_{centrifugal}$, the Coriolis force $\mathbf{F}_{Coriolis}$ and the Euler force $\mathbf{F}_{Euler}$, that are given by:

$$\mathbf{F}_{Centrifugal} = -\, m\,\Omega \times (\Omega \times \mathbf{r}), \tag{C.9}$$

$$\mathbf{F}_{Coriolis} = -\, 2m\,\Omega \times \dot{\mathbf{r}}, \tag{C.10}$$

$$\mathbf{F}_{Euler} = -\, m\,\frac{d\Omega}{dt} \times \mathbf{r}, \tag{C.11}$$

where $\mathbf{r} \in \mathbb{R}^3$ is the distance to the center of rotation and $\Omega \in \mathbb{R}^3$ the angular velocity of the reference frame. Let $\mathbf{F}_{inertial} = \mathbf{F}_{Centrifugal} + \mathbf{F}_{Coriolis} + \mathbf{F}_{Euler}$ and note that $\mathbf{r}$ and $\dot{\mathbf{r}}$ can be expressed in terms of $\mathbf{x}$. This inertial term can be analytically preserved and reduced to:

$$\tilde{\mathbf{F}}_{intertial}(\tilde{\mathbf{x}}) = \mathbf{V}^T\mathbf{F}_{inertial}(\mathbf{V}\tilde{\mathbf{x}}). \tag{C.12}$$

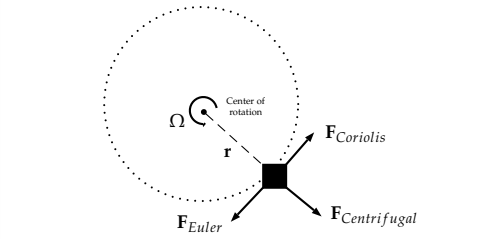


Fig. C.1: Mechanical forces acting on rotating device.

Appendix D

Source code - Gyroscope

Listing D.1: Gyroscope TDF Module Definition

```
#include <systemc-ams>
#include <systemc>
#include <Eigen/Dense>

#ifndef GYROSCOPIC_SENSOR_TDF_H_
#define GYROSCOPIC_SENSOR_TDF_H_

class GyroscopicSensor : public  sca_tdf:: sca_module
{
public:
  // Coefficients
  double m_;
  double k1_, k2_, b1_, b2_; //stiffness (k1, k2), damping (b1, b2)

  // Ports definition
   sca_tdf:: sca_in<double> forceAppliedX;
   sca_tdf:: sca_in<double> forceSensingY;
   sca_tdf:: sca_in<double> angularRotationVelocity;
   sca_tdf:: sca_out<double> displacementX;
   sca_tdf:: sca_out<double> displacementY;

  // Constructor
  GyroscopicSensor( sc_core::sc_module_name);

  virtual ~GyroscopicSensor();

  void writeEigen2Sca(Eigen::MatrixXd & mRef,  sca_util:: sca_matrix<double> & m);
  void writeEigen2Sca(Eigen::VectorXd & vRef,  sca_util:: sca_vector<double> & v);

  void writeSca2Eigen( sca_util:: sca_matrix<double> & mRef, Eigen::MatrixXd & m);
  void writeSca2Eigen( sca_util:: sca_vector<double> & vRef, Eigen::VectorXd & v);

  Eigen::VectorXd initializeDC();
  Eigen::VectorXd initializeAC();

protected:
  // SystemC-AMS internal methods
  //void set_attributes() {}
  //void initialize() {}
  void processing();

private:
  // Time-step defintion
  sca_core::sca_time tStep_;

  // State Space Representation
   sca_tdf::sca_ss stateSpace_;
   sca_util:: sca_matrix<double> a_;
   sca_util:: sca_matrix<double> b_;
   sca_util:: sca_matrix<double> c_;
   sca_util:: sca_matrix<double> d_;
```

```
52
53      // Vectors
54       sca_util:: sca_vector<double> x_;
55       sca_util:: sca_vector<double> y_;
56       sca_util:: sca_vector<double> u_;
57      double omega_;
58      bool first_call_;
59    };
60
61    #endif /* GYROSCOPIC_SENSOR_TDF_H_ */
```

Listing D.2: Gyroscope TDF Module Declaration

```
1     #include "gyroscopic_sensor_tdf.h"
2
3     GyroscopicSensor::GyroscopicSensor( sc_core::sc_module_name)
4         : m_(9.425E-10), k1_(0.00025), k2_(0.00025), b1_(0.0), b2_(3480.0), //damping effect with b2: 3480.0
5           forceAppliedX("AppliedForceX"), forceSensingY("SensingForceY"), angularRotationVelocity("
                AngularRotationVelocity"),
6           displacementX("DisplacementX"), displacementY("DisplacementY"),
7           tStep_(sca_core::sca_time(10.0,  sc_core::SC_US)),
8           //a_(Eigen::MatrixXd::Zero(4,4)), b_(Eigen::MatrixXd::Zero(4,2)), c_(Eigen::MatrixXd::Zero(2,4)), d_(
                Eigen::MatrixXd::Zero(2,2)),
9           x_(4), y_(2), u_(2), omega_(0.0),
10          first_call_(true)
11    {
12      /// State-space representation
13      /// Xdot = A.X + B.U
14      /// Y    = C.X + D.U
15      ///-------------------------------------------------------------------------------
16      /// X = (x, xDot, y, yDot)
17      /// U = (0, Fx_driving, 0, Fy_sensing)
18      /// Y = (xDot, yDot)
19      ///-------------------------------------------------------------------------------
20
21      a_.resize(4, 4);
22      a_(0, 0) = 0.0;       a_(0, 1) = 1.0;         a_(0, 2) = 0.0;     a_(0, 3) = 0.0;
23      a_(1, 0) = -k1_/m_;   a_(1, 1) = -b1_/m_;     a_(1, 2) = 0.0;     a_(1, 3) = 2*omega_;
24      a_(2, 0) = 0.0;       a_(2, 1) = 0.0;         a_(2, 2) = 0.0;     a_(2, 3) = 1.0;
25      a_(3, 0) = 0.0;       a_(3, 1) = -2*omega_;   a_(3, 2) = -k2_/m_; a_(3, 3) = -b2_/m_;
26
27      b_.resize(4, 2);
28      b_(0, 0) = 0.0;   b_(0, 1) = 0.0;
29      b_(1, 0) = 1.0/m_;   b_(1, 1) = 0.0;
30      b_(2, 0) = 0.0;   b_(2, 1) = 0.0;
31      b_(3, 0) = 0.0;   b_(3, 1) = 1.0/m_;
32
33      c_.resize(2, 4);
34      c_(0, 0) = 1.0;   c_(0, 1) = 0.0; c_(0, 2) = 0.0;   c_(0, 3) = 0.0;
35      c_(1, 0) = 0.0;   c_(1, 1) = 0.0; c_(1, 2) = 1.0;   c_(1, 3) = 0.0;
36
37      d_.resize(2, 2);
38      d_(0, 0) = 0.0;   d_(0, 1) = 0.0;
39      d_(1, 0) = 0.0;   d_(1, 1) = 0.0;
40
41      x_(0) = 0; x_(1) = 0; x_(2) = 0; x_(3) = 0;
42
43      u_(0) = 0; u_(1) = 0;
44
45      std::cout <<"\n\n" << this->kind() << " " << this->name() << " constructed" << std::endl;
46    }
47
48    GyroscopicSensor::~GyroscopicSensor()
49    {}
50
51    Eigen::VectorXd GyroscopicSensor::initializeDC() {
52      Eigen::MatrixXd a(4,4);
53      Eigen::MatrixXd b(4,2);
54
55      // DC Angular velocity
```

```
56      double omegaDC = 0.0; //angular velocity
57
58      // update coefficients in A-matrix
59      a_(1, 3) = 2*omegaDC;
60      a_(3, 1) = -2*omegaDC;
61      writeSca2Eigen(a_, a);
62      writeSca2Eigen(b_, b);
63
64      // DC Inputs
65      Eigen::VectorXd uDC(Eigen::VectorXd::Zero(2));
66      uDC(0) = 0.0; //forceAppliedX
67      uDC(1) = 0.0; //forceSensingY
68
69      // DC Analysis
70      return a.fullPivLu().solve((-1) * b * uDC);
71    }
72
73    Eigen::VectorXd GyroscopicSensor::initializeAC() {
74      Eigen::MatrixXd a(4,4);
75      Eigen::MatrixXd b(4,2);
76      Eigen::MatrixXd e = Eigen::MatrixXd::Identity(4,4);
77      Eigen::MatrixXd uAC(2,2);
78      Eigen::VectorXcd acInput(Eigen::VectorXcd::Zero(2));
79
80      // AC Angular velocity
81      double omegaAC { 1.0 };
82
83      // update coefficients in A-matrix
84      a_(1, 3) = 2* omegaAC;
85      a_(3, 1) = -2* omegaAC;
86      writeSca2Eigen(a_, a);
87      writeSca2Eigen(b_, b);
88
89      // AC Inputs
90      uAC(0,0) = 1.0; uAC(0,1) = 0.0; //forceAppliedX
91      uAC(1,0) = 0.0; uAC(1,1) = 0.0; //forceSensingY
92
93      // AC Analysis
94      // Calculation of the AC contribution to initial conditions
95      for (int i = 0; i < 2; i++) {
96        double phaseAC = uAC(i, 1) * M_PI / 180;
97        acInput(i) = uAC(i, 0) * std::complex<double>(cos(phaseAC), sin(phaseAC));
98      }
99
100     // Mode definition
101     double freq { 495.0 / (2.0 * M_PI) };
102     std::complex<double> s {};
103     s = std::complex<double>(0.0, 2.0 * M_PI * freq);
104
105     Eigen::MatrixXcd aComplex(a.cast<std::complex<double> >());
106     Eigen::MatrixXcd bComplex(b.cast<std::complex<double> >());
107     Eigen::MatrixXcd eComplex(e.cast<std::complex<double> >());
108     Eigen::MatrixXcd sEminusA(e.rows(), e.cols());
109     Eigen::VectorXcd rhsAC(2);
110     Eigen::VectorXcd xAC(2);
111
112     sEminusA = eComplex * s - aComplex;
113     rhsAC = bComplex * acInput;
114     xAC = sEminusA.fullPivLu().solve(rhsAC);
115
116     return xAC.imag();
117   }
118
119   void GyroscopicSensor::writeEigen2Sca(Eigen::MatrixXd & mRef,  sca_util:: sca_matrix<double> & m) {
120     for (int i=0; i<mRef.rows(); ++i) {
121       for (int j=0; j<mRef.cols(); ++j) {
122         m(i,j) = mRef(i,j);
123       }
124     }
125   }
```

```
126
127   void GyroscopicSensor::writeEigen2Sca(Eigen::VectorXd & vRef,  sca_util:: sca_vector<double> & v) {
128     for (int i=0; i<vRef.rows(); ++i) {
129       v(i) = vRef(i);
130     }
131   }
132
133   void GyroscopicSensor::writeSca2Eigen( sca_util:: sca_matrix<double> & mRef, Eigen::MatrixXd & m) {
134     for (std::size_t i=0; i<mRef.n_rows(); ++i) {
135       for (std::size_t j=0; j<mRef.n_cols(); ++j) {
136         m(i,j) = mRef(i,j);
137       }
138     }
139   }
140
141   void GyroscopicSensor::writeSca2Eigen( sca_util:: sca_vector<double> & vRef, Eigen::VectorXd & v) {
142     for (std::size_t i=0; i<vRef.length(); ++i) {
143       v(i) = vRef(i);
144     }
145   }
146
147   void GyroscopicSensor::processing() {
148     // Read sources actual value (update the Control Vector)
149     u_(0) = forceAppliedX.read();
150     u_(1) = forceSensingY.read();
151     omega_ = angularRotationVelocity.read();
152
153     a_(1, 3) = 2*omega_;
154     a_(3, 1) = -2*omega_;
155
156     if(first_call_) {
157       Eigen::VectorXd xInit(Eigen::VectorXd::Zero(4));
158       xInit += initializeDC(); //< perform DC analysis with input values
159       xInit += initializeAC(); //< perform AC analysis for initial oscillating state
160       writeEigen2Sca(xInit, x_);
161       first_call_ = false;
162     }
163
164     // Perform state-space representation
165     y_ =  stateSpace_(a_, b_, c_, d_, x_, u_, tStep_ );
166
167     //  Export and store outputs to output ports
168     displacementX.write( y_(0) );
169     displacementY.write( y_(1) );
170   }
```

Appendix E

Source code - Accelerometer ST LIS332AR

Listing E.1: TDF Module Definition of the accelerometer ST LIS332AR

```
#include "Data.h" // Reference to the data description of the model

namespace memsp_sc {

// TDF module automatically generated.
// The reduced-order model of the MEMS is encapsulated as a state-space system.
// Dedicated data structure and methods are associated to this C++ class.
class ST_LIS332AR : public  sca_tdf:: sca_module {

public:
  // C++ class definition, i.e., constructor and destructor.
  ST_LIS332AR( sc_core::sc_module_name);
  virtual ~ST_LIS332AR();
[...]

  // Input Ports
   sca_tdf:: sca_in<double> V_E_DX1;        // Driving voltage (x-axis)
   sca_tdf:: sca_in<double> V_E_SX1;        // Sensing voltage (x-axis)
   sca_tdf:: sca_in<double> tax;            // Translational acceleration (x-axis)
   sca_tdf:: sca_in<double> F_ProofMass_x;  // External force (x-axis)
   sca_tdf:: sca_in<double> M_ProofMass_x;  // Momentum (x-axis)
[...]

  // Output Ports
   sca_tdf:: sca_out<double> P_ProofMass_x; // Displacement (x-axis)
   sca_tdf:: sca_out<double> A_ProofMass_rx;// Angle (x-axis)
   sca_tdf:: sca_out<double> Cap_DX1;       // Driving capacitance (x-axis)
   sca_tdf:: sca_out<double> Cap_SX1;       // Sensing capacitance (x-axis)
[...]

protected:
  // SystemC-AMS method defining operations related to the sca_core::sca_module
  // Processing() is called at each time-step of the simulation.
  void processing() {
    // Update inputs with actual values of the control vector.
    u_(0) = double (E_DX1.read() );
[...]

    // Update weights and matrices regarding the inputs
    // in order to interpolate and build the whole system.
    data_->update(u_sweep_);
    data_->interpolate();

    // Return current output values computed from state-space system solving.
    y_ = data_->compute(u_, get_timestep().to_seconds() );

    // Export output values to corresponding ports.
    P_ProofMass_x.write(double(y_(0) ) );
  }
[...]

```

```
52      private:
53        // Data structure and methods exported from MEMS+
54        Data* data_;
55
56        // SystemC-AMS I/O vectors
57        Eigen::VectorXd u_;       // Input vector read during simulation
58        Eigen::VectorXd u_sweep_; // Input vector related to nonlinear variables
59        Eigen::VectorXd y_;       // Output vector written during simulation
60      [...]
61
62      }; // class ST_LIS332AR
63      }  // namespace memsp_sc
```

Appendix F

Source code - Accelerometer ST SEM

Listing F.1: Testbench Definition

```
1
2   /// @file   tb_mrom_accelerometer_2d.h
3   /// @brief  Testbench implementing an accelerometer 2-D for translation sensing
4   ///         in x- and y-axis.
5   ///
6   /// Related MEMS+ models and files:
7   ///  - Material Database:        ../memps/6.000_alpha/accelerometer_2d.mlib
8   ///  - Process:                  ../memps/6.000_alpha/accelerometer_2d.proc
9   ///  - 3-D model:                ../memps/6.000_alpha/accelerometer_2d.3dsch
10  ///  - Simulation configuration: ../memps/6.000_alpha/accelerometer_2d.msim
11  ///  - Reduced-Order Model:      ../memps/6.000_alpha/accelerometer_2d.MROM.mrom
12
13  #ifndef TB_MROM_ACCELEROMETER_2D_H_
14  #define TB_MROM_ACCELEROMETER_2D_H_
15
16  #include <cmath>
17  #include <ctime>
18  #include <iostream>
19
20  #include <systemc-ams>
21  #include <systemc>
22
23  #include "testbench.h"
24  #include "sc_generator.h"
25  #include "sca_generator.h"
26  #include "ac_sca_generator.h"
27  #include "dc_sca_generator.h"
28  #include "unit_util.h"
29
30  #include "mrom_accelerometer_2d.h"
31
32  /// This test bench implements a 2-D accelerometer sensing in x- and y-axis.
```

```
33    /// The design remains on a symmetric quadrant with two opposite sets of finger
34    /// combs in each axis in order to get a differential measurement of displacement.
35    class TB_MROM_Accelerometer2D : public Testbench
36    {
37    public:
38      //  DSP SETTINGS
39      /// Bits number of the digital buffer.
40      static const int NBIT_BUFFER = 16;
41
42      /// Number of bits supported by the ADC to encode the input signal to digital one.
43      static const int NBIT_ADC = 8;
44
45      /// Bits number of temperature signal.
46      static const int NBIT_TEMPERATURE = 12;
47
48      /// ADC frequency.
49      static const double ADC_FREQ;
50
51      /// Oversampling ratio defining sampling frequency significantly higher than
52      /// the Nyquist rate.
53      static const int OVERSAMPLING;
54
55      // STIMULI SETTINGS
56      /// MEMS frequency of main mode.
57      static const double FREQ;
58
59      /// Stimuli offset value.
60      static const double OFFSET;
61
62      /// Stimuli amplitude in x-axis.
63      static const double AMPLITUDE_X;
64
65      /// Stimuli amplitude in y-axis.
66      static const double AMPLITUDE_Y;
67
68      /// Signal delay.
69      static const double DELAY;
70
71      /// Signal phase.
72      static const double PHASE;
73
74      // PARAMETERS
75      /// Average number of bits (bitrate) per unit time passing between equipment
76      /// in a data transmission system.
77      static const int DATARATE[];
78      static const std::vector<int> v_datarate;
79
80      /// Full-scale ratio.
81      static const int FULLSCALE[];
82      static const std::vector<int> v_fullscale;
83
84      /// Powering of the device.
85      static const bool POWER[];
86      static const std::vector<bool> v_power;
87
88      /// Definition of the amplification.
89      static const float GAIN[];
90      static const std::vector<float> v_gain;
91
92      /// Definition of the scaling coefficient.
93      static const float SCALING[];
94      static const std::vector<float> v_scaling;
95
96      // SIGNALS
97      /// Signal binding acceleration in x-axis.
98       sca_tdf:: sca_signal<double> s_tax;
99
100     /// Signal binding acceleration in y-axis.
101      sca_tdf:: sca_signal<double> s_tay;
102
```

```
103     /// AC frequency (x-axis).
104      sca_tdf:: sca_signal<double> s_ac_freq_x;
105
106     /// AC frequency (y-axis).
107      sca_tdf:: sca_signal<double> s_ac_freq_y;
108
109     /// Global output signal in x-axis.
110      sc_core:: sc_signal<float > s_out_x;
111
112     /// Global output signal in y-axis.
113      sc_core:: sc_signal<float > s_out_y;
114
115     /// Signal defining the data rate.
116      sc_core:: sc_signal<int> s_datarate;
117
118     /// Signal defining the fullscale ratio.
119      sc_core:: sc_signal<int> s_fullscale;
120
121     /// Signal turning on/off the device under test.
122      sc_core:: sc_signal<bool> s_power;
123
124     /// Signal defining the temperature of the system [C].
125      sc_core:: sc_signal<hi::util::temperature_celsius_type> s_temperature;
126
127     // MODULES
128     /// Source defining the fullscale ratio.
129     hi::stimuli::ScGenerator<int, int>* src_fullscale;
130
131     /// Data rate source.
132     hi::stimuli::ScGenerator<int, int>* src_datarate;
133
134     /// Source turning the system on/off.
135     hi::stimuli::ScGenerator<bool, bool>* src_power;
136
137     /// Temperature source.
138     hi::stimuli::ScGenerator<double, hi::util::temperature_celsius_type>* src_temperature;
139
140     /// DC source with incremental acceleration steps (x-axis).
141     hi::stimuli::DCScaGenerator<double, double>* src_dc_tax;
142
143     /// Acceleration source with different frequency steps for AC analysis (x-axis).
144     hi::stimuli::ACScaGenerator<double, double>* src_ac_tax;
145
146     /// Source for transient analysis regarding x-axis acceleration.
147     hi::stimuli::ScaGenerator<double, double>* src_trans_tax;
148
149     /// DC source with incremental acceleration steps (y-axis).
150     hi::stimuli::DCScaGenerator<double, double>* src_dc_tay;
151
152     /// Acceleration source with different frequency steps for AC analysis (y-axis).
153     hi::stimuli::ACScaGenerator<double, double>* src_ac_tay;
154
155     /// Source for transient analysis regarding y-axis acceleration.
156     hi::stimuli::ScaGenerator<double, double>* src_trans_tay;
157
158     /// Module encapsulating the MROM file corresponding to Accelerometer2D.
159     hi::mems::mrom::Accelerometer2D<NBIT_ADC, NBIT_BUFFER>* m_accelerometer2D;
160
161     /// Tracefile
162      sca_util::sca_trace_file* atf;
163
164     /// Default constructor.
165     TB_MROM_Accelerometer2D();
166
167     /// Constructor to compute a specific @a analysis.
168     /// @param analysis defines either @c DC, @c AC or @c TRANS simulations.
169     TB_MROM_Accelerometer2D(const AnalysisType& analysis);
170
171     /// Constructor for transient analysis
172     /// @param config defines the selected transient profile definition.
```

```
173       TB_MROM_Accelerometer2D(const int& config);
174
175       /// Destructor
176       virtual ~TB_MROM_Accelerometer2D();
177
178       //  Member functions specialization
179       /// Initialize the configuration profiles for transient analysis and
180       /// the different parameters (power, fullscale, datarate).
181       void init();
182
183       /// Configure the parameter sof the selected @a analysis.
184       /// @param analysis defines either @c DC, @c AC or @c TRANS simulations.
185       void configure(const AnalysisType& analysis);
186
187       /// Perform the simulation wrt to its configuration (@see Testbench).
188       void run();
189
190       /// Perform post-processing operations on result file.
191       void analyze();
192
193       // TRACING
194       /// Defines the name of the results @a file.
195       /// @param file
196       void init_trace(const std::string& file);
197
198       /// Close the result file.
199       void stop_trace();
200
201     protected:
202       // METHODS
203       /// Select the @a analysis
204       /// @param analysis defines either @c DC, @c AC or @c TRANS simulations.
205       void set_analysis(const AnalysisType& analysis);
206
207       /// Select the transient profile corresponding to @a config.
208       /// @param config defines the selected transient profile definition.
209       void set_configuration(const int& config);
210
211     private:
212       // ANALYSES
213       /// Analysis mode (DC, AC or TRANS)
214       AnalysisType analysis_;
215
216       /// Simulation configuration for transient analysis.
217       int config_;
218
219       std::string filename_;
220
221       // PARAMETERS
222       /// Full-scale coefficient profile or steps.
223       std::tr1::function<int (double)> f_fullscale_;
224
225       /// Data rate profile or steps.
226       std::tr1::function<int (double)> f_datarate_;
227
228       /// Powering function.
229       std::tr1::function<bool (double)> f_power_;
230
231       // Stimuli profiles
232       std::tr1::function<double (double)> f_temperature_;
233
234       /// Acceleration stimuli profile (x-axis).
235       std::tr1::function<double (double)>* f_tax_;
236
237       /// Acceleration stimuli profile (y-axis).
238       std::tr1::function<double (double)>* f_tay_;
239     };
240
241     #endif // TB_MROM_ACCELEROMETER_2D_H_
242
```

```
243 /// Local Variables:
244 /// Mode: C++
245 /// End:
```

Listing F.2: Testbench Declaration

```
1
2  #include "tb_mrom_accelerometer_2d.h"
3  #include "file_reader.h"
4  #include "dataset.h"
5
6  #include "f_sin.h"
7  #include "f_step.h"
8  #include "f_pulse.h"
9  #include "f_multistep.h"
10
11 // CONSTANTS
12 const double TB_MROM_Accelerometer2D::ADC_FREQ =  6720;
13
14 const int TB_MROM_Accelerometer2D::OVERSAMPLING = 1;
15
16 const double TB_MROM_Accelerometer2D::FREQ = 2560.53; // si::hertz;
17
18 const double TB_MROM_Accelerometer2D::OFFSET = 0.0; // si::meter / si::second / si::second;
19
20 const double TB_MROM_Accelerometer2D::AMPLITUDE_X = 9.81;  // si::meter / si::second / si::second;
21
22 const double TB_MROM_Accelerometer2D::AMPLITUDE_Y = 9.81;  // si::meter / si::second / si::second;
23
24 const double TB_MROM_Accelerometer2D::DELAY = 0; // si::second;
25
26 const double TB_MROM_Accelerometer2D::PHASE = 0.0; // si::radian
27
28 const int TB_MROM_Accelerometer2D::FULLSCALE[] = {2,4,8,16};
29
30 const int TB_MROM_Accelerometer2D::DATARATE[] = {1344};
31
32 const bool TB_MROM_Accelerometer2D::POWER[] = {true, true, true, true};
33
34 const float TB_MROM_Accelerometer2D::GAIN[] = {55.36326e12, 27.68163e12, 13.840816e12, 4.6136e12}; // from 2
       ratio
35
36 const float TB_MROM_Accelerometer2D::SCALING[] = {1e-3, 2e-3, 4e-3, 12e-3};
37
38 const std::vector<int> TB_MROM_Accelerometer2D::v_fullscale (FULLSCALE, FULLSCALE + sizeof(FULLSCALE) /
       sizeof(FULLSCALE[0]) );
39
40 const std::vector<int> TB_MROM_Accelerometer2D::v_datarate (DATARATE, DATARATE + sizeof(DATARATE) / sizeof(
       DATARATE[0]) );
41
42 const std::vector<bool> TB_MROM_Accelerometer2D::v_power (POWER, POWER + sizeof(POWER) / sizeof(POWER[0]) );
43
44 const std::vector<float> TB_MROM_Accelerometer2D::v_gain (GAIN, GAIN + sizeof(GAIN) / sizeof(GAIN[0]) );
45
46 const std::vector<float> TB_MROM_Accelerometer2D::v_scaling (SCALING, SCALING + sizeof(SCALING) / sizeof(
       SCALING[0]) );
47
48
49 TB_MROM_Accelerometer2D::TB_MROM_Accelerometer2D()
50   : config_ (0)
51 {
52   // Initialization.
53   init();
54 }
55
56 TB_MROM_Accelerometer2D::TB_MROM_Accelerometer2D(const AnalysisType& analysis)
57   : analysis_ (analysis)
58 {
59   // Initialization.
60   init();
```

```
61  }
62
63  TB_MROM_Accelerometer2D::TB_MROM_Accelerometer2D(const int& config)
64    : config_ (config)
65  {
66    // Transient analysis
67    analysis_ = AnalysisType::TRANS;
68
69    // Initialization
70    init();
71  }
72
73  TB_MROM_Accelerometer2D::~TB_MROM_Accelerometer2D()
74  {}
75
76  void TB_MROM_Accelerometer2D::init()
77  {
78    // RUNTIME DEFINITION
79    //-------------------------------------------------------------------------
80    this->set_simulation_time( sc_core::sc_time(2.0,  sc_core::SC_SEC));
81    this->set_simulation_timestep( sc_core::sc_time(3.0,  sc_core::SC_US));
82
83    int n_config = 12;
84
85    f_tax_ = new std::tr1::function<double (double)> [n_config];
86    if (f_tax_ == nullptr) {
87      // error assigning memory. Take measures.
88      std::cout << "Please add function for" << this->f_tax_->target_type().name() << std::endl;
89    }
90
91    f_tay_ = new std::tr1::function<double (double)> [n_config];
92    if (f_tay_ == nullptr) {
93      // error assigning memory. Take measures.
94      std::cout << "Please add function for" << this->f_tay_->target_type().name() << std::endl;
95    }
96
97    for (int k =0; k < 5; ++k) {
98      f_pulse<double> impulse_X(0.0, pow(2, k) * AMPLITUDE_X, sca_core::sca_time(0.25,
           sc_core::SC_SEC), sca_core::sca_time(0.5,  sc_core::SC_SEC), sca_core::sca_time(0,
           sc_core::SC_MS), sca_core::sca_time(0,  sc_core::SC_MS));
99      f_pulse<double> impulse_Y(0.0, pow(2, k) * AMPLITUDE_Y, sca_core::sca_time(1.25,
           sc_core::SC_SEC), sca_core::sca_time(0.5,  sc_core::SC_SEC), sca_core::sca_time(0,
           sc_core::SC_MS), sca_core::sca_time(0,  sc_core::SC_MS));
100
101     f_tax_[k] = impulse_X;
102     f_tay_[k] = impulse_Y;
103
104     f_step<double> step_X(0.0, pow(2, k) * AMPLITUDE_X,  sc_core::sc_time(0.25,  sc_core::SC_SEC));
105     f_step<double> step_Y(0.0, pow(2, k) * AMPLITUDE_Y,  sc_core::sc_time(0.5,  sc_core::SC_SEC));
106
107     f_tax_[k+5] = step_X;
108     f_tay_[k+5] = step_Y;
109   }
110
111   f_sin<double> sin_X(OFFSET, AMPLITUDE_X, FREQ, PHASE);
112   f_sin<double> sin_Y(OFFSET, AMPLITUDE_Y, FREQ, 2*M_PI);
113
114   f_pulse<double> pulse_X(0.0, AMPLITUDE_X, sca_core::sca_time(0.25,
        sc_core::SC_SEC), sca_core::sca_time(0.25,  sc_core::SC_SEC), sca_core::sca_time(0.125,
        sc_core::SC_SEC), sca_core::sca_time(0.125,  sc_core::SC_SEC));
115   f_pulse<double> pulse_Y(0.0, AMPLITUDE_Y, sca_core::sca_time(1.25,
        sc_core::SC_SEC), sca_core::sca_time(0.25,  sc_core::SC_SEC), sca_core::sca_time(0.125,
        sc_core::SC_SEC), sca_core::sca_time(0.125,  sc_core::SC_SEC));
116
117   f_tax_[10] = sin_X;
118   f_tax_[11] = pulse_X;
119
120   f_tay_[10] = sin_Y;
121   f_tay_[11] = pulse_Y;
122
```

```
123       f_sin<double> sin_temperature(0, 120, 0.3, PHASE);
124       f_temperature_ = sin_temperature;
125
126       f_multistep<int> multistep_fullscale(0, v_fullscale, this->get_simulation_time());
127       f_fullscale_ = multistep_fullscale;
128
129       f_multistep<int> multistep_datarate(0, v_datarate, this->get_simulation_time());
130       f_datarate_ = multistep_datarate;
131
132       f_multistep<bool> multistep_power(true, v_power, this->get_simulation_time());
133       f_power_ = multistep_power;
134     }
135
136     void TB_MROM_Accelerometer2D::configure(const AnalysisType& analysis)
137     {
138       switch (analysis) {
139         case AnalysisType::DC : {
140           // Turn off other analysis profiles.
141           src_ac_tax = nullptr;
142           src_ac_tay = nullptr;
143           src_trans_tax = nullptr;
144           src_trans_tay = nullptr;
145
146           // Configure DC simulation range (x-axis).
147           std::vector<double> dc_steps_x;
148           double dc_amplitude_x_min = -16 * AMPLITUDE_X;
149           double dc_amplitude_x_max = 16 * AMPLITUDE_X;
150           double dc_step_x_size = 2 * AMPLITUDE_X;
151
152           // Configure DC simulation range (y-axis).
153           std::vector<double> dc_steps_y;
154           double dc_amplitude_y_min = -16 * AMPLITUDE_Y;
155           double dc_amplitude_y_max = 16 * AMPLITUDE_Y;
156           double dc_step_y_size = 2 * AMPLITUDE_Y;
157
158           // Number of steps to apply on each axis.
159           int n_steps_x = trunc((dc_amplitude_x_max - dc_amplitude_x_min) / dc_step_x_size);
160           int n_steps_y = trunc((dc_amplitude_y_max - dc_amplitude_y_min) / dc_step_y_size);
161
162           // Set the DC values.
163           for (int i = 0; i < (n_steps_x + n_steps_y); ++i) {
164             if (i < n_steps_x) {
165               dc_steps_x.push_back(dc_amplitude_x_min + i * dc_step_x_size);
166               dc_steps_y.push_back(0.0);
167             } else {
168               dc_steps_x.push_back(0.0);
169               dc_steps_y.push_back(dc_amplitude_y_min + (i - n_steps_x) * dc_step_y_size);
170             }
171           }
172
173           // Define the DC source (x-axis).
174           src_dc_tax = new hi::stimuli::DCScaGenerator<double, double>("DC_Source_tax", dc_steps_x, this->
                get_simulation_time());
175           src_dc_tax->out(s_tax);
176
177           // Define the DC source (x-axis).
178           src_dc_tay = new hi::stimuli::DCScaGenerator<double, double>("DC_Source_tay", dc_steps_y, this->
                get_simulation_time());
179           src_dc_tay->out(s_tay);
180
181           // Get info.
182           std::cout << "\nInfo:\t" << "DC Analysis" << std::endl;
183           break;
184         }
185
186         case AnalysisType::AC : {
187           // Turn off other analysis profiles.
188           src_dc_tax = nullptr;
189           src_dc_tay = nullptr;
190           src_trans_tax = nullptr;
```

```
191           src_trans_tay = nullptr;
192
193
194           // Define the AC source (x-axis).
195           src_ac_tax = new hi::stimuli::ACScaGenerator<double, double>("AC_Source_tax", this->
                  get_simulation_time(), FREQ/4, 0.0, 0.01 * AMPLITUDE_X, FREQ, 0.0);
196           src_ac_tax->out(s_tax);
197           src_ac_tax->out_freq(s_ac_freq_x);
198
199           // Define the AC source (y-axis).
200           src_ac_tay = new hi::stimuli::ACScaGenerator<double, double>("AC_Source_tay", this->
                  get_simulation_time(), FREQ/4, 0.0, 0.01 * AMPLITUDE_Y, FREQ, 0.0);
201           src_ac_tay->out(s_tay);
202           src_ac_tay->out_freq(s_ac_freq_y);
203
204           // Get info.r
205           std::cout << "\nInfo:\t" << "AC Analysis" << std::endl;
206           break;
207         }
208
209         case AnalysisType::TRANS : {
210           // Turn off other analysis profiles.
211           src_dc_tax = nullptr;
212           src_dc_tay = nullptr;
213           src_ac_tax = nullptr;
214           src_ac_tay = nullptr;
215
216           // Define the transient source.
217           src_trans_tax = new hi::stimuli::ScaGenerator<double, double>("TRANS_Source_taz", f_tax_[config_]);
218           src_trans_tax->out(s_tax);
219
220           src_trans_tay = new hi::stimuli::ScaGenerator<double, double>("TRANS_Source_tay", f_tay_[config_]);
221           src_trans_tay->out(s_tay);
222
223           // Get info.
224           std::cout << "\nInfo:\t" << "TRANS Analysis" << std::endl;
225           break;
226         }
227       }
228     }
229
230
231     void TB_MROM_Accelerometer2D::run()
232     {
233       // SIMULATION RUNTIME
234       //-------------------------------------------------------------------------
235        sc_core::sc_time TSIM = this->get_simulation_time();
236        sc_core::sc_time TSTEP = this->get_simulation_timestep();
237
238       // SUBMODULES
239       //-------------------------------------------------------------------------
240       // Stimuli
241       configure(analysis_);
242
243       // Parameters
244       src_fullscale = new hi::stimuli::ScGenerator<int, int>("src_fullscale", f_fullscale_, (TSIM / 2) /
              v_fullscale.size());
245       src_fullscale->out(s_fullscale);
246
247       src_datarate = new hi::stimuli::ScGenerator<int, int>("src_datarate", f_datarate_, (TSIM / 2) / v_datarate
              .size());
248       src_datarate->out(s_datarate);
249
250       src_power = new hi::stimuli::ScGenerator<bool, bool>("src_power", f_power_, (TSIM / 2) / v_power.size());
251       src_power->out(s_power);
252
253       src_temperature = new hi::stimuli::ScGenerator<double, hi::util::temperature_celsius_type>("
              src_temperature", f_temperature_,  sc_core::sc_time(1.0/(ADC_FREQ),
              sc_core::SC_SEC));
254       src_temperature->out(s_temperature);
```

```
255
256       // Accelerometer 2D
257       m_accelerometer2D = new hi::mems::mrom::Accelerometer2D<NBIT_ADC, NBIT_BUFFER>("Accelerometer2D",
              v_fullscale, v_gain, v_scaling, OVERSAMPLING, pow(2*1.1e-18, 2.0), 1825.12, 1, 0.4, ADC_FREQ);
258       m_accelerometer2D->in_datarate(s_datarate);
259       m_accelerometer2D->in_fullscale(s_fullscale);
260       m_accelerometer2D->in_power(s_power);
261       m_accelerometer2D->in_tax(s_tax);
262       m_accelerometer2D->in_tay(s_tay);
263       m_accelerometer2D->out_x(s_out_x);
264       m_accelerometer2D->out_y(s_out_y);
265
266       m_accelerometer2D->m_mems->set_timestep(TSTEP);
267
268       // SIMULATION
269       //---------------------------------------------------------------------------
270       init_trace("mrom_accelerometer_xy_2d.dat");
271       simulate();
272       stop_trace();
273       reset();
274     }
275
276     void TB_MROM_Accelerometer2D::analyze()
277     {
278       std::ostringstream path;
279       hi::data::FileReader* file_reader;
280       hi::data::Dataset dataset;
281       std::map<int, std::string> parameters;
282
283       // Create a new reader.
284       file_reader = new hi::data::FileReader(filename_);
285
286       // Index of parameters
287       int t_index = 0;
288       int entry_index = 0;
289       int disp_index = 0;
290       int cap_index = 0;
291
292       switch (analysis_) {
293           case AnalysisType::DC :
294           case AnalysisType::TRANS :
295             t_index = 1;
296             entry_index = 4;
297             disp_index = 6;
298             cap_index = 8;
299             break;
300
301           case AnalysisType::AC :
302             t_index = 1;
303             entry_index = 4;
304             disp_index = 8;
305             cap_index = 10;
306             break;
307       }
308
309       // List of parameters
310       parameters[0] = "Index";
311       parameters[1] = "Time (s)";
312       parameters[2] = "Acceleration (m/s^2)";
313
314       // CAPACITANCE
315       // ---------------------------------------------------------------------------
316       // Store maximum values
317       parameters[3] = "Max. Capacitance (F)";
318       dataset.setParameters(parameters);
319
320       path.str("");
321       path << file_reader->remove_extension(filename_) << "_cap_max.dat";
322       dataset.setDatastore(file_reader->getDataset().getMax(t_index, entry_index, cap_index));
323       file_reader->writeFile(path.str(), dataset);
```

```
324
325       // Store minimum values
326       parameters[3] = "Min. Capacitance (F)";
327       dataset.setParameters(parameters);
328
329       path.str("");
330       path << file_reader->remove_extension(filename_) << "_cap_min.dat";
331       dataset.setDatastore(file_reader->getDataset().getMin(t_index, entry_index, cap_index));
332       file_reader->writeFile(path.str(), dataset);
333
334       // Store mean values
335       parameters[3] = "Mean Capacitance (F)";
336       parameters[4] = "Mean Dev. Cap. (F)";
337       dataset.setParameters(parameters);
338
339       path.str("");
340       path << file_reader->remove_extension(filename_) << "_cap_mean.dat";
341       dataset.setDatastore(file_reader->getDataset().getMean(t_index, entry_index, cap_index));
342       file_reader->writeFile(path.str(), dataset);
343
344       // DISPLACEMENT
345       // ------------------------------------------------------------------------
346       // List of parameters
347       parameters[0] = "Index";
348       parameters[1] = "Time (s)";
349       parameters[2] = "Acceleration (m/s^2)";
350
351       // Store maximum values
352       parameters[3] = "Max. Displacement (m)";
353       dataset.setParameters(parameters);
354
355       path.str("");
356       path << file_reader->remove_extension(filename_) << "_disp_max.dat";
357       dataset.setDatastore(file_reader->getDataset().getMax(t_index, entry_index, disp_index));
358       file_reader->writeFile(path.str(), dataset);
359
360       // Store minimum values
361       parameters[3] = "Min. Displacement (m)";
362       dataset.setParameters(parameters);
363
364       path.str("");
365       path << file_reader->remove_extension(filename_) << "_disp_min.dat";
366       dataset.setDatastore(file_reader->getDataset().getMin(t_index, entry_index, disp_index));
367       file_reader->writeFile(path.str(), dataset);
368
369       // Store mean values
370       parameters[3] = "Mean Displacement (m)";
371       parameters[4] = "Mean Dev. Disp. (m)";
372       dataset.setParameters(parameters);
373
374       path.str("");
375       path << file_reader->remove_extension(filename_) << "_disp_mean.dat";
376       dataset.setDatastore(file_reader->getDataset().getMean(t_index, entry_index, disp_index));
377       file_reader->writeFile(path.str(), dataset);
378
379       delete file_reader;
380     }
381
382     void TB_MROM_Accelerometer2D::init_trace(const std::string& file)
383     {
384       // Plotting configuration
385       /* ------------------------------------------------------------------------
386        *  Index | DC Analysis         | AC Analysis         | TRANS Analysis      |
387        *  ----- | ------------------- | ------------------- | ------------------- |
388        *      1 | Time                | Time                | Time                |
389        *      2 | SourceAccelerationX | SourceAccelerationX | SourceAccelerationX |
390        *      3 | SourceAccelerationY | SourceAccelerationY | SourceAccelerationY |
391        *      4 | tax                 | AC_Frequency_X      | tax                 |
392        *      5 | tay                 | AC_Frequency_Y      | tay                 |
393        *      6 | Displacement_X      | tax                 | Displacement_X      |
```

```
394         *      7 | Displacement_Y      | tay                | Displacement_Y      |
395         *      8 | Capacitance Pos. X  | Displacement_X     | Capacitance Pos. X  |
396         *      9 | Capacitance Neg. X  | Displacement_Y     | Capacitance Neg. X  |
397         *     10 | Capacitance Pos. Y  | Capacitance Pos. X | Capacitance Pos. Y  |
398         *     11 | Capacitance Neg. Y  | Capacitance Neg. X | Capacitance Neg. Y  |
399         *     12 | NoiseInjector X     | Capacitance Pos. Y | NoiseInjector X     |
400         *     13 | NoiseInjector Y     | Capacitance Neg. Y | NoiseInjector Y     |
401         *     14 | ADC X               | NoiseInjector X    | ADC X               |
402         *     15 | ADC Y               | NoiseInjector Y    | ADC Y               |
403         *     16 | Filter X            | ADC X              | Filter X            |
404         *     17 | Filter Y            | ADC Y              | Filter Y            |
405         *     18 | Gain Amplifier X    | Filter X           | Gain Amplifier X    |
406         *     19 | Gain Amplifier Y    | Filter Y           | Gain Amplifier X    |
407         *     20 | Out X               | Gain Amplifier X   | Out X               |
408         *     21 | Out Y               | Gain Amplifier X   | Out Y               |
409         *     22 | Data rate           | Out X              | Data rate           |
410         *     23 | Full scale          | Out Y              | Full scale          |
411         *     24 | Power X             | Data rate          | Power X             |
412         *     25 | Temperature         | Power X            | Temperature         |
413         *     26 |                     | Temperature        |                     |
414         * ------------------------------------------------------------------------*/
415
416
417       // Path and prefix specification for result file.
418       std::ostringstream path;
419
420       switch (analysis_) {
421         case AnalysisType::DC : {
422           // Rename file and define the data directory.
423           path << "./data/dc_" << file;
424           filename_ = this->auth_file(path.str());
425           atf =  sca_util:: sca_create_tabular_trace_file(filename_.c_str());
426
427           // Select the input data to store.
428            sca_util:: sca_trace(atf, src_dc_tax->out, "SourceAccelerationX" );
429            sca_util:: sca_trace(atf, src_dc_tay->out, "SourceAccelerationY" );
430           break;
431         }
432
433         case AnalysisType::AC : {
434           // Rename file and define the data directory.
435           path << "./data/ac_" << file;
436           filename_ = this->auth_file(path.str());
437           atf =  sca_util:: sca_create_tabular_trace_file(filename_.c_str());
438
439           // Select the input data to store.
440            sca_util:: sca_trace(atf, src_ac_tax->out, "SourceAccelerationX" );
441            sca_util:: sca_trace(atf, src_ac_tay->out, "SourceAccelerationY" );
442
443           // Select the input data to store.
444            sca_util:: sca_trace(atf, src_ac_tax->out_freq, "FrequencyX" );
445            sca_util:: sca_trace(atf, src_ac_tay->out_freq, "FrequencyY" );
446           break;
447         }
448
449         case AnalysisType::TRANS : {
450           // Rename file and define the data directory.
451           path << "./data/trans_" << file;
452           filename_ = this->auth_file(path.str());
453           atf =  sca_util:: sca_create_tabular_trace_file(filename_.c_str());
454
455           // Select the input data to store.
456            sca_util:: sca_trace(atf, src_trans_tax->out, "SourceAccelerationX" );
457            sca_util:: sca_trace(atf, src_trans_tay->out, "SourceAccelerationY" );
458           break;
459         }
460       }
461
462       // Mechanical variables
463        sca_util:: sca_trace(atf, m_accelerometer2D->in_tax, "tax");
```

```
464         sca_util:: sca_trace(atf, m_accelerometer2D->in_tay, "tay");
465
466         sca_util:: sca_trace(atf, m_accelerometer2D->m_mems->out[0], "Displacement_X");
467         sca_util:: sca_trace(atf, m_accelerometer2D->m_mems->out[1], "Displacement_Y");
468
469       // Capacitances
470         sca_util:: sca_trace(atf, m_accelerometer2D->m_mems->out[8], "Cap_Sensing_Positive_X");
471         sca_util:: sca_trace(atf, m_accelerometer2D->m_mems->out[6], "Cap_Sensing_Negative_X");
472
473         sca_util:: sca_trace(atf, m_accelerometer2D->m_mems->out[9], "Cap_Sensing_Positive_Y");
474         sca_util:: sca_trace(atf, m_accelerometer2D->m_mems->out[7], "Cap_Sensing_Negative_Y");
475
476       // DSP flow
477         sca_util:: sca_trace(atf, m_accelerometer2D->m_noise_injector_x->out, "NoiseInjector_X");
478         sca_util:: sca_trace(atf, m_accelerometer2D->m_noise_injector_y->out, "NoiseInjector_Y");
479
480         sca_util:: sca_trace(atf, m_accelerometer2D->m_adc_x->out, "ADC_X");
481         sca_util:: sca_trace(atf, m_accelerometer2D->m_adc_y->out, "ADC_Y");
482
483         sca_util:: sca_trace(atf, m_accelerometer2D->m_filter_x->out, "Filter_X");
484         sca_util:: sca_trace(atf, m_accelerometer2D->m_filter_y->out, "Filter_Y");
485
486         sca_util:: sca_trace(atf, m_accelerometer2D->m_gain_amplifier_x->out, "GainAmplifier_X");
487         sca_util:: sca_trace(atf, m_accelerometer2D->m_gain_amplifier_y->out, "GainAmplifier_Y");
488
489         sca_util:: sca_trace(atf, s_out_x, "Out_X");
490         sca_util:: sca_trace(atf, s_out_x, "Out_Y");
491
492       // Parameters
493         sca_util:: sca_trace(atf, s_datarate, "Datarate");
494         sca_util:: sca_trace(atf, s_fullscale, "Fullscale");
495         sca_util:: sca_trace(atf, s_power, "Power");
496         sca_util:: sca_trace(atf, s_temperature, "Temperature");
497     }
498
499     void TB_MROM_Accelerometer2D::stop_trace()
500     {
501        sca_util::sca_close_vcd_trace_file(atf);
502       delete atf;
503     }
504
505     void TB_MROM_Accelerometer2D::set_analysis(const AnalysisType& analysis)
506     {
507       analysis_ = analysis;
508     }
509
510     void TB_MROM_Accelerometer2D::set_configuration(const int& config)
511     {
512       if (analysis_ == AnalysisType::TRANS) {
513         config_ = config;
514       } else {
515         std::cout << "The simulation configuration cannot be changed since no transient analysis is defined." <<
                std::endl;
516       }
517     }
```

Modélisation et simulation haut-niveau de micro-systèmes électromécaniques pour le prototypage virtuel multi-physique en SystemC-AMS

L'évolution des systèmes embarqués se traduit aujourd'hui par des ensembles complexes, dits systèmes cyber-physiques, opérant principalement en réseau et interagissant fortement avec leur environnement. Intégrés à des circuits de contrôle et de traitement du signal, les micro-systèmes électromécaniques, ou MEMS, jouent un rôle primordial dans ces ensembles en tant que capteurs ou actionneurs. La conception de tels systèmes requiert des solutions globales et pluridisciplinaires telles que le prototypage virtuel. Basée sur des modèles haut-niveau, cette technique permet d'anticiper le comportement du système dès les premières phases de conception et de le raffiner lors de phases plus avancées. Ces méthodes ont progressivement été appliquées à la conception de circuits intégrés, notamment avec l'utilisation de langages de description matérielle, tels que VHDL ou Verilog. En adoptant un niveau d'abstraction supérieur, SystemC a largement contribué au développement concourant des parties matérielles et logicielles. Parallèlement, les extensions proposées dans SystemC-AMS répondent au nombre croissant de composants analogiques dans les circuits intégrés et constituent une base prometteuse pour le prototypage virtuel de systèmes hétérogènes. Pour cette raison, cette thèse traite de la modélisation et de la simulation haut-niveau de dispositifs MEMS en SystemC-AMS. Dans un premier temps, nous évaluons les capacités actuelles du standard et des modèles de calcul proposés dans SystemC-AMS. Nous démontrons les limites et la difficulté d'élaborer des modèles équivalents de dispositifs MEMS dont la géométrie et les couplages internes nécessitent des descriptions plus détaillées. Nous proposons donc dans un second temps d'intégrer directement des modèles réduits de dispositifs MEMS dans SystemC-AMS. La réduction d'ordre de modèle est une méthode mathématique permettant de créer des représentations compactes de systèmes initialement très larges en termes de degrés de liberté. Ainsi, nous utilisons les modèles générés depuis l'outil d'analyse en éléments finis *MEMS+* et proposons une interface de programmation pour les insérer dans des modèles SystemC-AMS. Après avoir détaillé les principales fonctionnalités de l'interface, nous discutons les améliorations possibles du standard et de la solution présentée. Enfin, nous vérifions notre solution avec l'étude d'un accéléromètre et comparons les résultats avec l'état de l'art en termes de précision des modèles et de performances de simulation. Cette thèse propose ainsi une méthodologie complète pour intégrer des dispositifs MEMS dans un environnement de simulation haut-niveau.