

UNIVERSITY OF ROSTOCK

DISSERTATION

Analog/Mixed-Signal Modeling Methodologies for Heterogeneous Virtual Prototypes and Their Impact on Smart Sensor Development

Author:

Alexandra KÜSTER

Dissertation

to obtain the academic degree

Doktor-Ingenieur (Dr.-Ing.)

of the Faculty of Computer Science and Electrical Engineering

at the University of Rostock

submitted by

Alexandra Küster

born on 12.08.1996 in Bad Soden a. Ts., Germany

from Reutlingen

https://doi.org/10.18453/rosdok_id00005687

Rostock, December 19, 2025

Reviewers:

- **Prof. Dr.-Ing. habil. Christian Haubelt**
Embedded Systems Chair
Institute of Applied Microelectronics and Computer Engineering
University of Rostock, Germany
- **Univ.-Prof. Dr. Daniel Große**
Institute for Complex Systems (ICS)
Johannes Kepler University Linz, Austria
- **Dr.-Ing. Rainer Dorsch**
Department for Engineering Advanced Products
Bosch Sensortec GmbH, Germany

Year of submission: 2025

Year of defense: 2025

“All models are wrong but some are useful.”

George E. P. Box

Acknowledgements

First of all, I would like to express my profound gratitude to Prof. Dr. Christian Haubelt for his willingness to supervise my work and the continuous support he has provided, even though I kept postponing my visit to Rostock over and over again. I always left our meetings with new food for thought and the reassurance that I was still on the right track even if not every initial idea led to the desired results. Moreover, your comments on a newly written paper have always been more than helpful to extract the essence of the contribution and clearly highlight it within the limited number of pages. Although our online cooperation worked well, I was happy to finally meet you in person and get to know my fellows in Rostock.

Secondly, and no less importantly, I would like to thank Rainer Dorsch, my supervisor at Bosch Sensortec, who was always on hand to give me advice and support, acquainted me with many other colleagues, kept advertising my work within the company but nevertheless showed understanding when I refused to spend too much time for project related work. I am convinced that we have found a good symbiosis to support my research and the projects at the same time. Thank you for the never ending support and the fruitful discussions.

Moreover, I would like to thank Univ.-Prof. Dr. Daniel Große who accepted to review my thesis three years after he had hosted the conference where I published my very first paper as corresponding author.

Last but not least, I want to thank the people that patiently endured my mental ups and downs, complaints, doubts and whatever comes to one's mind during the long path as a PhD student. This especially includes my colleagues who helped me to refuel energy and motivation during a coffee break, my fellow PhDs at Bosch who understood the pitfalls of an industrial doctorate like nobody else and of course my dear partner Nick who patiently listened to my enthusiastic but lengthy explanations when something worked out and kept me upright when it didn't.

All in all, this work would not exist without all of you and I truly appreciate all flavors of support you provided.

Abstract

MEMS sensors have become an integral part of everyday life. Although often invisible to us, we are surrounded by these small micro-electromechanical parts pervasively. They are hidden in smartphones, wearables, hearables, at home or in cars. Their applications are manifold ranging from simple mechanisms that let our display content rotate when rotating the phone up to complex features like (human) activity recognition. Modern sensor systems become more and more complex to fulfill the ever-increasing requirements and to enable emerging use cases. They are increasingly integrating microprocessors, on the one hand to enable continuous compensation and self-calibration, and on the other hand to calculate algorithms for sensor fusion or entire applications directly on the sensor resulting in *smart MEMS sensors*. At the same time, the fast changing consumer market puts high pressure onto the sensor developers to shorten time-to-market for future sensor generations.

These two contradicting developments require the adaption of old and the invention of new and more efficient development methodologies. A promising approach has been used for complex systems-on-chip for years. *Virtual prototypes* have evolved as an effective approach to support the hardware/software co-design. A virtual prototype thereby defines a system-level simulation model that enables the joint analysis of hardware and software. As a consequence, the software development can start earlier, becomes more efficient through the increased observability and debugging features of the virtual prototype, and early integration tests are enabled to ensure the proper interaction of hardware and software.

Virtual prototypes are nowadays already in use for the sensor development. However, the focus is still on the digital subsystem although the mechanical and analog components form a significant portion of the overall system. Hence, most existing virtual prototypes cover the inherently heterogeneous systems insufficiently. The limited coverage is especially problematic for the design of low-level firmware that interacts closely with the hardware. Test cases have to be adapted to the characteristics of the prototype, which reduces their reusability and induces new sources of error.

As a remedy, the thesis at hand deals with methodologies for the modeling and usage of heterogeneous virtual prototypes, i.e. system-level models that cover all domains of a smart MEMS sensor and enable the joint simulation of mechanical, analog/mixed-signal, digital and software components. The impact of such a model on the sensor development is investigated by examining different areas of development. It is demonstrated that heterogeneous virtual prototypes effectively support the development of firmware as well as trim and test algorithms. Thereby, requirements for these models are derived: early availability, high simulation performance and coverage of the analog state space. In a consecutive step, various new modeling methodologies are introduced to account for these requirements. Focus is thereby laid onto analog/mixed-signal models as digital virtual prototyping is already state-of-the-art. Strategies are developed to increase the solver efficiency of SystemC AMS and support the reuse of existing models for virtual platforms. Moreover, semi-symbolic simulation is introduced to increase the coverage of the analog state space.

Kurzfassung

MEMS-Sensoren sind aus dem alltäglichen Leben nicht mehr wegzudenken. Auch wenn es uns meistens nicht so bewusst ist, sind diese kleinen und unauffälligen mikroelektromechanischen Systeme unsere ständigen Begleiter, sei es im Handy, der Smartwatch, den Kopfhörern, zuhause oder im Auto. Ihre Anwendungen sind vielfältig und reichen von einfachen Mechanismen wie der Erkennung, wenn das Handy gedreht wird, um den Bildschirminhalt zu rotieren, bis zu komplexer Aktivitätserkennung. Dabei verändert sich der Konsumgütermarkt rasant, was einen hohen Druck auf die Hersteller ausübt, die Entwicklungszeit neuer Sensorgenerationen zu verkürzen. Trotzdem werden die Systeme moderner Sensoren immer komplexer, um den erhöhten Anforderungen und neuen Anwendungsfällen gerecht zu werden. Immer öfter integrieren Sensoren Mikroprozessoren, einerseits um kontinuierliche Kompensationen und Selbstkalibrierungen zu ermöglichen und andererseits, um Algorithmen zur Sensorfusion oder ganze Applikationen direkt auf dem Sensor zu berechnen, was in sogenannten *intelligenten MEMS-Sensoren* mündet.

Diese beiden konträren Verläufe erfordern effiziente Entwicklungsmethoden für moderne Sensoren. Ein vielversprechender Ansatz stammt hierbei aus dem Bereich komplexer Digitalsysteme. Virtuelle Prototypen haben sich als effektives Vehikel zur Unterstützung des Hardware-Software-Codesigns etabliert. Ein virtueller Prototyp definiert hierbei ein simulierbares Systemmodell, das die Funktionalität der digitalen Hardware widerspiegelt und eine gemeinsame Analyse von Hardware und Software ermöglicht. Dies unterstützt einen früheren Start der Software-Entwicklung (bevor die Hardware verfügbar wird), effizientere Fehlersuche durch die erhöhte Beobachtbarkeit des Prototypen sowie frühe Integrationstests, um sicherzustellen, dass Hardware und Software aufeinander abgestimmt sind.

Die Strategie virtueller Prototypen wird heutzutage bereits für die MEMS-Sensorentwicklung genutzt. Allerdings liegt auch hier der Fokus auf dem Digitalteil. Ein MEMS-Sensor beinhaltet allerdings neben dem Digitalteil ein mikroelektromechanisches System und eine analoge Ausleseschaltung. Dadurch ist das System inherent heterogen und wird nur unzureichend von einem digitalen virtuellen Prototypen abgebildet. Besonders bei Firmware, die oft sehr eng mit der Hardware interagiert, kann diese limitierte Abdeckung der Domänen zu Problemen führen. Testfälle müssen an die Begebenheiten des Prototypen angepasst werden, wodurch die Wiederverwendbarkeit leidet und neue Fehlerquellen geschaffen werden.

Die vorliegende Arbeit beschäftigt sich zu diesem Zweck mit Methoden für die Modellierung und Verwendung von heterogenen virtuellen Prototypen, also Systemmodellen, die alle Domänen eines MEMS-Sensors abdecken und die gemeinsame Simulation von mechanischen, analogen und digitalen Komponenten sowie der Software unterstützen. Sie untersucht den Einfluss eines solchen Modells auf die Sensorentwicklung, indem verschiedene Bereiche der Entwicklung betrachtet werden. Die Arbeit demonstriert, dass heterogene virtuelle Prototypen die Entwicklung von Firmware sowie Trim- und Testalgorithmen sichtbar unterstützen können. Dabei werden Anforderungen an diese Modelle abgeleitet, die im Anschluss durch verschiedene neu entwickelte Modellierungsmethoden unterstützt werden sollen. Der Fokus liegt dabei auf AMS (engl. analog/mixed-signal) Modellen, da die digitale Modellierung auf dieser Ebene bereits Stand der Technik ist. Zu diesem Zweck werden Strategien aufgezeigt, um die Simulationsgeschwindigkeit zu erhöhen und die Wiederverwendung existierender Modelle zu ermöglichen. Darüber hinaus werden semi-symbolische Simulationen verwendet, um eine bessere Abdeckung des Zustandsraums zu ermöglichen.

Contents

List of Abbreviations	viii
1 Introduction	1
1.1 Problem Statement	2
1.2 Goals and Scope of this Work	4
1.3 Contributions	5
2 Fundamentals and State of the Art	8
2.1 Smart Sensors	8
2.2 (Digital) Virtual Prototyping	17
2.3 Analog/Mixed-Signal Modeling	21
2.4 Heterogeneous Virtual Prototypes	27
3 The Impact of Heterogeneous Virtual Prototypes on the Smart Sensor Development	29
3.1 Related Work	29
3.2 Identification of Use Cases	32
3.2.1 Firmware Development	32
3.2.2 Trim and Test Procedures	35
3.3 Model Requirements	37
3.4 Implementation	40
3.5 Experimental Results	43
3.6 Impact on the Development Flow	44
3.7 Discussion	47
4 Choosing a Suitable Language for Analog/Mixed-Signal Modeling	50
4.1 Related Work	50
4.2 Choice of Simulation Language	52
4.3 Performance Analysis and Comparison of SystemC AMS and Real Number Modeling in SystemC	53
4.3.1 Update Mechanism	53
4.3.2 Scheduling	57
4.3.3 Dynamic Timing	59
4.3.4 ODE Performance	59
4.3.5 Application to a Virtual Sensor Prototype	60
4.4 Discussion	62
5 Increasing the Solver Efficiency in SystemC AMS	64
5.1 Related Work	65
5.2 Ordinary Differential Equations	67
5.3 Integration into the SystemC AMS Simulation Semantics	69
5.3.1 Method 1: PRECALC	72
5.3.2 Method 2: SEGMENT	73
5.4 Interference with Other Language Features	78

5.4.1	Feedback Loops	78
5.4.2	Reaction onto Incoming Events	80
5.4.3	Multiple Time Controlling Instances	82
5.5	Timestep Quantization: Overcoming the LU Decomposition Bottleneck	83
5.6	ELN Switching Efficiency	87
5.7	Implementation & User Interface	88
5.8	Experimental Results	91
5.8.1	Case 1: Abstract MEMS Accelerometer Model	92
5.8.2	Case 2: Reduced Order Modeling	95
5.8.3	Case 3: Switched-Capacitor Charge-to-Voltage Converter	99
5.9	Discussion	102
6	Early Availability Through Automated Translation from MATLAB/Simulink	104
6.1	Related Work	105
6.2	Structural Generation Flow	106
6.2.1	Monolithic Modules	107
6.2.2	Hierarchical Modules	111
6.3	Restrictions and Modeling Guideline	113
6.4	Experimental Results	114
6.4.1	Performance Evaluation	116
6.5	Discussion	119
7	Semi-Symbolic Simulation of Virtual Sensor Prototypes	121
7.1	Affine Arithmetic	121
7.2	Existing Fields of Application	124
7.3	Application to Sensor Trimming	125
7.4	Limitations	131
7.5	Proposed Extensions	132
7.6	Case Study: Binary Search	136
7.7	Discussion	139
8	Conclusion and Future Work	142
8.1	Discussion and Limitations	143
8.2	Future Work	146
	Bibliography	148
	List of Publications	162
A	Implementation Details on the Solver Enhancements in SystemC AMS	163
A.1	Variable-Step Solvers	163
A.2	Switch Monitoring	166
B	Illustrative Example of the Outcome of the Translation Process	169
C	Additional Notes to Affine Arithmetic	175
C.1	Affine Division	175
C.2	Examples of Approximations of Nonlinear Functions in AA2	176
C.3	Implementation of Special Operators	177

List of Abbreviations

AA	A ffine A rithmetic
AA2	A ffine A rithmetic with 2 nd order root-cause tracking extension
AADD	A ffine A rithmetic D ecision D iagram
ABM	A nalog B ehavioral M odeling
AC	A lternating C urrent
ADC	A nalog-to- D igital C onverter
AMS	A nalog/ M ixed- S ignal
API	A pplication P rogramming I nterface
ASIC	A pplication- S pecific I ntegrated C ircuit
AT	A pproximately T imed
BG	B ond G raph
BIST	B uilt- I n S elf- T est
CIC	C ascaded I ntegrated- C omb (filter)
CPU	C entral P rocessing U nit
CV	C harge-to- V oltage
DAC	D igital-to- A nalog- C onverter
DAE	D ifferential A lgebraic E quation
DC	D irect C urrent
DE	D iscrete E vent
DSP	D igital S ignal P rocessing
DTDF	D ynamic T imed D ata F low
DUT	D evice U nder T est
EC	E mbedded C oder
ELN	E lectrical L inear N etwork
FEM	F inite E lement M ethod
FIFO	F irst- I n F irst- O ut
FMI	F unctional M ock-up I nterface
FMU	F unctional M ock-up U nit
FPGA	F ield- P rogrammable G ate A rray
FW	F irm W are
GUI	G raphical U ser I nterface
HDL	H ardware D escription L anguage
HVP	H eterogeneous V irtual P rototype
HW	H ard W are

IMU	I nternal M easurement U nit
IoT	I nternet of T hings
IPC	I nter- P rocess C ommunication
ISS	I nstruction S et S imulator
I²C	I nter- I ntegrated C ircuit (communication protocol)
LSF	L inear S ignal F low
LT	L oosely T imed
LTF	L inear T ransfer F unction
LU	L ower- U pper (matrix decomposition scheme)
MEMS	M icro- E lectro M echanical S ystem
MoC	M odel of C omputation
NLN	N on L inear N etworks
NVM	N on- V olatile M emory
ODE	O rdinary D ifferential E quation
PLL	P hase- L ocked L oop
RAM	R andom- A ccess M emory
ROM	R educed O rders M odel or R ead- O nly M emory
RTL	R egister- T ransfer L evel
RQ	R esearch Q uestion
SoC	S ystem- o n- C hip
SPI	S erial P eripheral I nterface (communication protocol)
SS	S tate S pace
SW	S oft W are
TCI	T ime C ontrolling I nstance
TCP	T ransmission C ontrol P rotocol
TDF	T imed D ata F low
TLM	T ransaction L evel M odeling
VCO	V oltage- C ontrolled O scillator
VP	V irtual P rototype
XAA	e Xtended A ffine A rithmetic

Chapter 1

Introduction

The first sensors based on micro-electromechanical systems (MEMS) have already been presented in the 1970s [1]. Since then, the MEMS sensor market has steadily increased. Roughly twenty years ago, it has been dominated by automotive use cases, inkjet nozzles for printers and projection systems made of large arrays of micro-mirrors [2]. Thereby, the automotive market focused on safety features using accelerometers in airbags and gyroscopes for electronic stability programs (ESPs). With the emergence of smartphones, the importance of sensors for the consumer market has increased rapidly. Nowadays, the consumer sector is the biggest market for MEMS sensors in terms of volume and revenue [2]. Especially inertial sensors are nowadays widely adopted in smartphones for various applications like step counters, tilt detection and more. Within the past decade, additional fields for sensors in the consumer sectors have arisen including but not limited to hearables, wearables, virtual reality and individual health applications. Moreover, additional markets have arisen like Industry 4.0. The global smart sensor market is expected to further continue its growth for the next years [3].

The current market trends put high demands onto the development of these devices. Firstly, the evolution goes along with an increasing demand for sensor performance in terms of low offset, drift, noise and current consumption along with increased sensitivity and robustness. Secondly, the complexity of algorithms that are applied to the sensor data increases as well to enable new applications like image stabilization, activity recognition and others. In the meanwhile, the trend of the Internet of Things (IoT) pushes the demand for smart devices with computational power on the edge. This combination further drives the need for smart sensors that combine the recording of physical data and their algorithmic processing in a single device and directly provide high level information to the host. Illustrating examples are smart home systems and fitness trackers.

The thesis at hand focuses on smart sensors for the consumer market. One characteristic of the consumer sector is its fast changing pace with short product cycles. This puts high pressure onto the time-to-market requirement in the development flow. As a consequence, companies try to adopt strategies to shorten the product development cycles. One powerful concept in this domain is called *frontloading*. The idea is to shift tasks to earlier project stages such that they are removed from the timing critical path of the product development. Thomke and Fujimoto define frontloading as "a strategy that seeks to improve development performance by shifting the identification and solving of [design] problems to earlier phases of a product development process" [4]. This especially includes the definition of proper interfaces between multiple components. The concept is further based on the experience that, on average, the effort to fix a bug increases over time. For software bugs, this relation is often described as exponential [5]. Frontloading thus tries to bring the workload of identifying and fixing issues up to earlier project stages (the "front") to reduce their overall impact. Modern computer aided engineering techniques support these activities. Thereby, simulation models are widely used to validate concepts or estimate performance metrics in early stages. In modern complex systems-on-chip (SoCs), hardware/software (HW/SW) co-design is one major aspect of frontloading activities.

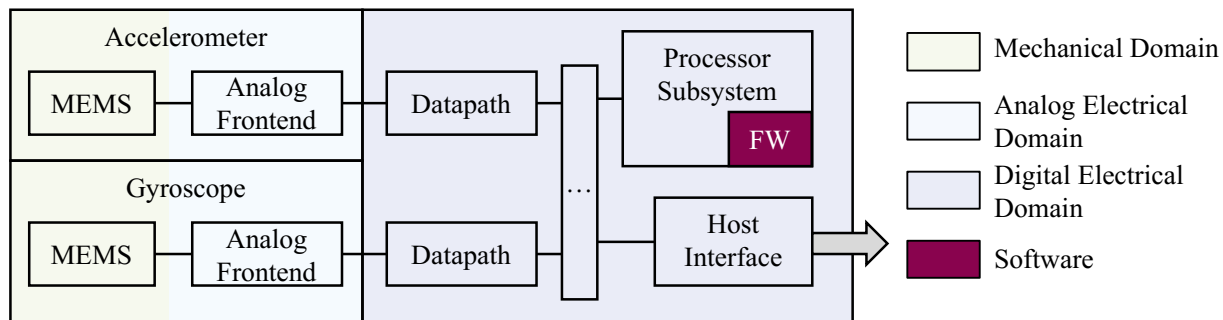


FIGURE 1.1: Schematic view of an inertial measurement unit (IMU). The colors denote the different domains integrated within the device.¹

Virtual prototyping evolved as a widely adopted approach in this manner [6, 7, 8]. A virtual prototype (VP) describes an abstract behavioral model of a system or product that allows to jointly simulate hardware and software. The simulation language SystemC is the de-facto standard in this regard. Its event-based simulation kernel and the commonly used high level of abstraction allow to create highly performant models. SystemC especially supports transaction-level modeling to enable abstract modeling of communication protocols and reduce the overall number of events. Moreover, it is a C++ library, i.e. it natively interfaces well with software written in C/C++. The microcontrollers are emulated using instruction set simulators. VPs can additionally be annotated with extra-functional properties as for example performance or power consumption to allow their analysis. Overall, VPs can be used to develop and test software within its (simulated) target environment. If the VP is already used during design-space exploration, it can support the HW/SW partitioning. During the design phase, it is used to develop and debug software before the first hardware becomes available [9]. Thereby, simulations on the VP act as continuous integration tests. Although often less accurate than physical prototyping, e.g. using an FPGA (field-programmable gate array), virtual prototyping is faster and less costly. Moreover, it can be directly derived from the specification due to its high level of abstraction. Thus, it is potentially earlier available than an FPGA implementation that requires the availability of a register-transfer level (RTL) design.

1.1 Problem Statement

Although often seen as an established holistic workflow, the VP usually covers only the digital hardware and software. The approach hence fits well for SoCs with complex computer architecture, multi-processor systems and similar applications. However, it offers only limited benefit for the smart sensor development. Many sensors for the consumer market are based on micro-electromechanical systems (MEMS). Figure 1.1 exemplarily shows the different components of a MEMS inertial measurement unit (IMU). It consists of an accelerometer and a gyroscope. Combining these two sensors allows proper motion tracking. The physical sensing is achieved via the MEMS sensing element that is connected to an analog read-out circuitry, called frontend. Digital signal processing is applied to filter the data and compensate nonidealities. As this work considers smart sensors, a processor subsystem is integrated including a microcontroller to run dedicated software on the device. All in all, the sensor integrates the mechanical, analog electrical, digital electrical and the software domain. A digital prototype thus covers only half of the domains inherent to a smart sensor. The remaining part of the system

¹For the sake of brevity, the MEMS sensing element is indicated as *MEMS* only in this and upcoming block diagrams. Still, the sensing element and not the entire sensor system is addressed.

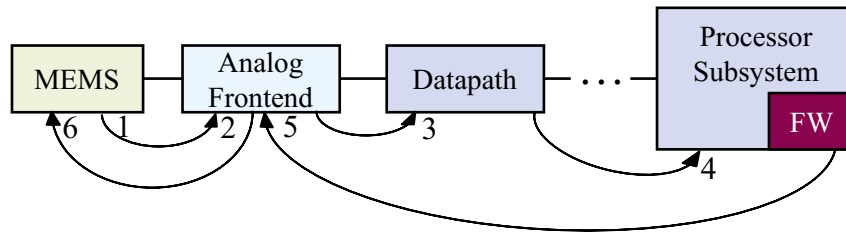


FIGURE 1.2: Schematic illustration of the interaction scheme of hardware and firmware. Physical stimuli are sensed by the MEMS (1), read-out by the analog frontend (2), pre-filtered by the datapath (3), and processed by the firmware (4). New configurations/stimuli are derived and applied to the frontend (5) that stimulates the MEMS (6). The roundtrip can execute multiple times.

is typically replaced by a simple data injection mechanism. Its usability for integration tests is therefore limited. Especially in the consumer market where time-to-market is usually short, the various components are developed in parallel. This harbors the risk of integration errors that are only discovered at a late stage. A simulation model that actually covers the interaction between all domains is therefore of utmost importance.

On top of that, modern smart sensors include not only software that post-processes sensor data but also firmware features that interact closely with the hardware. Especially advanced calibration and test features rely on bi-directional interactions between the firmware and the sensing element. Figure 1.2 represents a typical interaction scheme. First, physical stimuli are sensed by the MEMS sensing element (1). The read-out of samples is then performed by the analog frontend (2). Afterwards, a pre-filtering is conducted in the datapath (3) before the samples are processed by the firmware (4). Based on the evaluation of these data, the firmware chooses a configuration or stimulus to apply and sets corresponding configuration registers. These registers impact the functionality of the frontend (5), which in turn stimulates the MEMS (6). Finally, the firmware processes the newly generated samples (1-4). If the MEMS and frontend are replaced by a simple data injection mechanism as typically done, this kind of bi-directional interaction is not applicable to the VP. This makes the development and validation of corresponding firmware features complex and error-prone.

Besides, the development cycle of a smart sensor contains much more than pure hardware and software development. Testing, characterization and qualification consume a significant amount of resources. Moreover, the performance constraints can nowadays not be met without a variety of trimming algorithms that suppress deviations due to manufacturing variations or environmental effects. The simulation of these steps are out-of-scope for purely digital virtual platforms as most built-in trim and test equipment is located within the analog part of the system. Moreover, many nonidealities like offset, temperature dependent drifts, crosstalk between sensing axes etc. are generated by or within the MEMS. Emulating these effects by pure data injection requires to fit the injected data manually to each testcase and is not feasible for the large variety of effects.

Due to the lack of heterogeneity, classical (digital) virtual prototyping all in all offers limited benefits for the smart sensor development. These limitations motivate to expand the concept of virtual prototypes such that they actually reflect the basic behavior of the complete system. This requires the availability of multi-domain models on a similar abstraction level. The resulting fusion of a digital VP and additional models of computation is referred to as heterogeneous virtual prototype (HVP) throughout this work.

1.2 Goals and Scope of this Work

Several works have already been conducted to support multi-domain simulations. Many of these introduce frameworks for jointly managing and simulating different models of computations (MoCs) [10, 11, 12, 13]. This work does not target to add another framework to the existing list. Instead, the native connectivity between SystemC and SystemC AMS is used to allow multi-domain simulations. SystemC AMS offers multiple MoCs for the creation of analog/mixed-signal (AMS) models. The mechanical domain is reflected based on ordinary differential equations (ODEs) and thus covered by the same approach.

The overall goal of this work is to expand the scope and effectiveness of virtual prototypes for the smart sensor development. The starting point is the concept of heterogeneous virtual prototypes that integrate multiple domains into one system-level model. In the context of the thesis at hand, an HVP is represented by the combination of AMS models and digital SystemC models that support the integration of all domains given in Fig. 1.1 into a single model. The working hypothesis is that *HVPs can effectively support the development of smart sensors* throughout various stages of the development flow. Their impact is investigated at an industrial product development example of an inertial measurement unit. Specific use cases of the HVP are identified and requirements onto the modeling style are derived. The confirmation of the hypothesis relies on the existence of efficient analog/mixed-signal models. Therefore, this thesis introduces *novel AMS modeling methodologies*. They are chosen to address the requirements derived and aim to further increase the effectiveness of AMS modeling for HVPs. Thereby, three major requirements are addressed, namely an early *availability*, a high *simulation performance* and a sufficient *coverage* of the analog state space. The former two have been major drivers for the success of digital VPs. The extension towards HVPs shall thus not limit these inherent benefits. The latter is specific to analog models as the state space is continuous.

An HVP must fulfill the availability requirement to allow effective frontloading of development and integration tasks. It must fulfill the performance requirement as the speed of iteration cycles directly correlates with the time spent to simulate and check the current implementation of a feature. Meeting this requirement is especially challenging as continuously changing signals natively demand higher frequent updates than sporadically triggering events. Lastly, the HVP must fulfill the coverage requirement to account for the variability of the manufactured sensors and achieve confidence in the robustness of the developed features onto these variations.

The overall scope of the work can be summarized in terms of four central research questions (RQ1-4) that are addressed:

RQ1-Impact Where and to which extent can HVPs support the development of MEMS smart sensors?

The positive impact of VPs onto the hardware/software integration is well-known already. However, the author targets to explore additional use cases that are enabled by the heterogeneous extension and related to the development of MEMS smart sensors. Moreover, it shall be analyzed if the HVP can support software development even more effectively than a digital VP does.

RQ2-Performance How to optimize the simulation performance of AMS models for HVPs in the context of MEMS smart sensors?

Simulation is always subject to a trade-off between accuracy and performance. A model can either represent the system in every detail and be very accurate for the price of long simulation times or it can be abstract and fast. Thus, a suitable level of abstraction that is sufficiently accurate to cover any use case identified but still fast enough to speed up the development cycle must be found. Moreover, a suitable simulation language must be chosen that fits the

level of abstraction and provides high simulation performance. Additionally, enhanced solver techniques allow to further optimize the simulation performance.

RQ3-Availability How can an early availability of the HVP be ensured?

The effectiveness of virtual platforms scales heavily with the timeframe of their release. Many alternative solutions like simulation of RTL models or physical prototypes arise as soon as the implementation advances. Although the HVP keeps several benefits like observability and controllability which is essential for effective debugging, one major aspect of its success is grounded in the frontloading. Thus, strategies for rapid prototyping are key to the success of HVPs.

RQ4-Coverage To which degree can HVPs cover the analog state space with a finite set of simulations?

The combination of continuous states, manufacturing variations, environmental influences and aging effects causes MEMS sensors to have an inherent part-to-part variation. From a mathematical point of view, the state space representing the properties of a MEMS sensor is infinite. Consequently, the question arises whether a transient HVP is suitable to validate algorithms and features for the complete analog state space or if other strategies can be used to cover it sufficiently.

1.3 Contributions

The analyses and proposed methodologies presented in this work are in line with the research questions listed above. The concrete contributions are summarized below and related to the research questions stated. Major parts of the work have been previously published by the author of this thesis. The corresponding papers are introduced here and the author's role in their creation is highlighted. The thesis at hand sets these works into the overall context, refines and extends their content and creates a broader and more elaborate view on heterogeneous virtual prototyping in the context of smart sensor development than a single paper can cover.

Evaluation of the Impact of HVPs onto the Smart Sensor Development The first research question (RQ1-Impact) is addressed with the help of a large-scale industrial case study, in which the development of an inertial measurement unit has been accompanied by the provision of AMS models for the virtual platform. The impact of the resulting HVP on two different applications has thereby been investigated in more detail based on the feedback of respective model users. The use cases discussed are the firmware development and the preparation of trim and test procedures for the laboratory. The former has already been conducted on digital VPs in previous projects such that the differences between the old and the new approach are discussed. The former represents a use case that has not been in the scope of the digital VP at all. Requirements for the HVP are derived based on the presented case study. Moreover, RQ2-Performance is also partly addressed defining a suitable level of abstraction for the HVP. The impact is found to depend strongly on the point in time when the model becomes available. The earlier the model is released to other teams, the more benefit is taken out of it as no implementation exists at that point which could be used as reference instead. It is identified that the AMS extension of the VP broadens the model's scope to new use cases and strengthens the users' system insights.

The impact of HVPs onto the firmware development in smart sensors has already been published in [P1]. The case study presented in the paper has been fully designed, conducted, and evaluated by the author of the thesis at hand. She is also the corresponding author of [P1].

Performance Comparison to Evaluate which Modeling Language is Most Suitable for HVPs

As SystemC is the de-facto standard for virtual platforms, the thesis at hand takes its usage for the digital part as granted. Although existing co-simulation approaches are shortly discussed, the thesis at hand focuses on the two existing options that connect natively well to SystemC to avoid issues caused by simulator coupling. Hence, the main contribution in this regard is a performance comparison between real number modeling in SystemC, i.e. adding the analog content using SystemC itself, and the timed data flow MoC of SystemC AMS. The investigation represents an intermediate step to answer RQ2-Performance.

The results show that SystemC AMS is more performant for large clusters due to its static schedule. The interface between SystemC and SystemC AMS should be kept slim as each converter port introduces an overhead for the synchronization task.

Most aspects of the comparison have already been published in [P3]. The design of experiments, their execution and evaluation have been done by the author of the thesis at hand. She is also the corresponding author of [P3].

Enhancement of the SystemC AMS Simulation Performance In order to fully address RQ2 and increase the simulation performance of the chosen setup for the HVP further, the solver efficiency of SystemC AMS is enhanced introducing variable-step ordinary differential equation solvers. Two distinct methodologies to fit the mathematical concept of variable-step solvers into the execution semantics of SystemC AMS are presented. These concepts have been introduced and evaluated in [P6] for the timed data flow MoC. The methodology is further enhanced introducing a timestep quantization to improve the efficiency of implicit variable-step solvers. Implementations are conducted for all models of computation that are part of the SystemC AMS standard. They are realized extending the proof-of-concept implementation of SystemC AMS [14]. Another performance related contribution focuses on electrical linear networks with high switching activity. Here, a reuse of the underlying solver matrices is triggered to reduce the solver's computational cost.

The experimental results prove the methodologies to be effective. The exact speed up thereby largely depends on the experiment. The more heterogeneous the dynamics of a system throughout the simulation, the greater the benefit of the variable-step solver. The timestep quantization is beneficial in the presence of large differential equation systems.

An overview on the various approaches is given in [P7]. All methodologies have been designed, implemented and analyzed by the author of the thesis at hand. The same holds for the validating case studies and their evaluation. She is also the corresponding author of [P6] and [P7].

Automatic Model Translation from MATLAB/Simulink to SystemC AMS In order to tackle RQ3-Availability, the focus is laid on techniques to reduce the overall workload of creating AMS models on system level and enable re-use. It is self-evident that an early availability of the HVP directly relates to an early availability of corresponding engineering resources. However, resource planning is not in the scope of this work. Instead, a methodology is proposed that allows to fit existing AMS models to the integration into HVPs in an automated fashion. This is achieved through a translation procedure that generates SystemC AMS models from MATLAB/Simulink models. As MATLAB/Simulink is often used by analog designers to execute first exploration tests, abstract models in this language are well suited to allow efficient re-use. The large variety of libraries offered by this commercial tool allows rapid prototyping. The main advantage of the proposed translation is its preservation of hierarchical structures. In later project stages, individual subblocks may then be replaced by refined versions. The translation process including the model preparation takes only a few minutes and creates a highly modular SystemC AMS model.

The translation procedure has been published in [P2a, P2b]. Concept development, implementation of the automated tool as well as the evaluation have been done by the author of this work. She is also the corresponding author of [P2a] and [P2b].

Semi-Symbolic Simulation of AMS Models All contributions mentioned to this point are based on the underlying assumption that numeric, transient simulations are performed. This work additionally investigates the potential of other modeling styles to address RQ4-Coverage. The main contribution hereby is the investigation and adaption of the concept of semi-symbolic simulation with affine arithmetic for the application of sensor trimming simulation. The arithmetic is chosen because it allows to uniquely identify sources of variations in each affine term such that their propagation throughout the system becomes visible. An extension to the arithmetic is proposed that fits it better to the target of developing and validating trim procedures. The arithmetic which works on intervals proves to enhance the expressiveness of trimming simulations because a single simulation can result in the complete range of trim words that are required to trim any sensor that its physical parameters lie in a specified range. Moreover, a failing trim is easily investigated through the propagation of deviation symbols that identify sources of variations.

The concept and proposed extension have been introduced in [P5] and are evaluated in more detail in the thesis at hand. Concept, implementation and evaluation have been conducted by the author of this work. She is also the corresponding author of [P5].

Each major contribution as stated above is discussed in a distinct chapter. Hence, the rest of the thesis is structured as follows. Chapter 2 provides some fundamentals on smart sensors and virtual prototyping as a baseline. The impact of HVPs onto the smart sensor development is then elaborated in chapter 3. The performance requirement is addressed by the choice of the simulation language in chapter 4, and the work on the solver efficiency of SystemC AMS in chapter 5. Afterwards, the availability requirement is dealt with in chapter 6 and the semi-symbolic simulation is covered in chapter 7. Corresponding related work is discussed in each of these chapters. The work is finally concluded in chapter 8.

Chapter 2

Fundamentals and State of the Art

The objective of this chapter is to provide the reader with the fundamental concepts and principles that will be utilized throughout the remainder of this thesis. Section 2.1 introduces the working principles of smart sensors. Section 2.2 focuses on the subject of virtual prototyping and introduces the well-established concept of digital virtual prototypes. Subsequently, an overview of analog/mixed-signal modeling is provided in section 2.3, which introduces SystemC AMS as the modeling language used throughout this work and situates it within the context of established languages. Finally, section 2.4 combines the preceding concepts and covers existing approaches for heterogeneous virtual platforms that integrate multiple domains simultaneously.

2.1 Smart Sensors

In order to comprehend the challenges inherent to the development of smart sensors and investigate to which extent heterogeneous virtual prototypes can support this process, it is essential to gain an understanding of their fundamental operational principles and characteristics.

The set of physical effects that are instrumented as sensing principles is as diverse as the properties that are to be measured. To provide a few illustrative examples, piezoresistive effects are employed for the design of pressure sensors [15], thermal sensing is utilized in the development of accelerometers [16] or stress sensors [17], and optical effects support the measurement of angular rates [18], magnetic fields [19] and relative humidity [20]. Details on these and other sensing principles, including magnetic, chemical and piezocapacitive can be found in [21]. The application domains are similarly broad-ranging, encompassing automotive and consumer markets, as well as Internet-of-Things (IoT), Industry 4.0, biomedical applications, energy harvesting, and other fields [22].

This thesis focuses on the challenges facing low-cost consumer markets, which are dominated by MEMS sensors and characterized by rapid change. The term *micro-electromechanical system* (MEMS) describes a system that combines movable elements within the micrometer range with electrical measurement principles. An example of a MEMS is a mass suspended from a substrate by springs, which can be used to measure physical quantities through electrical readout strategies. The sensors described in this thesis employ a capacitive measurement principle. MEMS sensors are fabricated from silicon using surface micromachining. For further details on the manufacturing process, the reader is directed to the comprehensive elaboration by V. Kempe [23]. A variety of sensor types have been realized using MEMS since the 1970s [1], including but not limited to pressure sensors [24], gas sensors [25], and especially inertial sensors with high volumes [1, 26].

In the remainder of the thesis, mainly MEMS inertial sensors are investigated, namely accelerometers and gyroscopes. Accordingly, the working principles of these two types of sensors are presented in this chapter. Afterwards, an overview of the complete sensor system is provided to illustrate the heterogeneity of the design. In addition to MEMS and analog readout circuitry, modern sensors increasingly incorporate digital components to enhance the quality

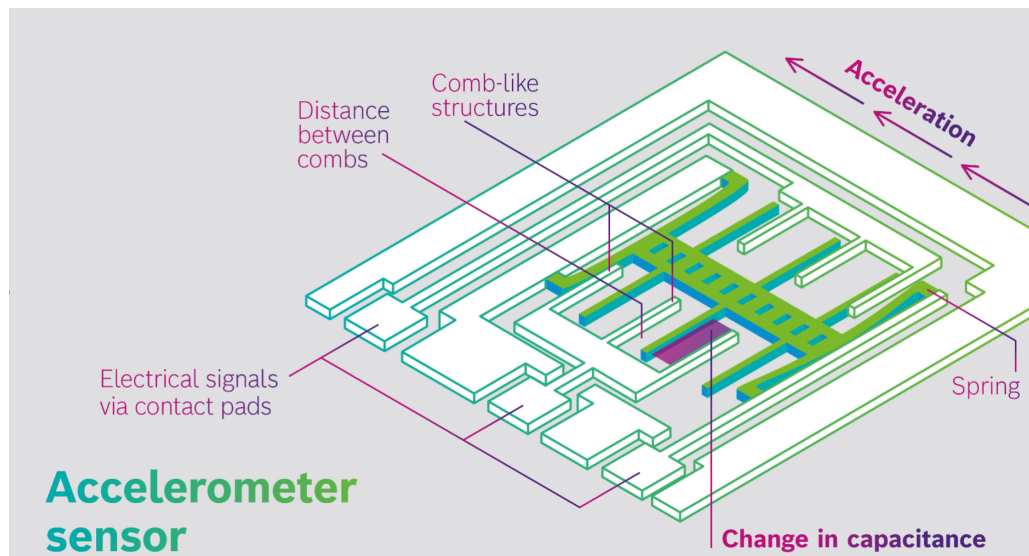


FIGURE 2.1: Schematic view of a MEMS accelerometer. It consists of a fixed frame (white) and a suspended mass (green) that together create comb finger electrodes. In presence of an acceleration, the suspended mass is deflected which creates a change in capacitance. © Bosch Sensortec GmbH

of the sensor data by post-processing steps and to account for the new market trends and the corresponding requirements. The term *smart sensor* indicates that modern sensors are more sophisticated than mere conduits for raw data. An increasing number of devices are incorporating microcontrollers, which enable the direct execution of firmware on the sensor. This, for instance, aligns with the trend of edge computing, which aims to preprocess data at the edge (i.e., the IoT device) and reduce the volume of data transmitted to the cloud, thereby facilitating the transmission of more meaningful data and an overall reduction of power consumption which is essential for battery powered devices [27]. Hence, modern sensors often support advanced features, such as the capability to calculate quaternions or perform activity recognition directly on the device [28].

MEMS Accelerometer

The following section outlines the fundamental working principle of MEMS accelerometers. A concise overview of the operational principles and existing architectural designs is presented by Langfelder et al. [26]. For a more detailed description, please refer to the cited literature (e.g., [29, 30, 31]). The relevant aspects to understand the working principle are given below. The basic architecture of a MEMS accelerometer is depicted in Fig. 2.1. The structure comprises a fixed frame, represented in white, which is formed by the substrate, and a suspended mass (green), which is connected to the frame by springs. When an acceleration is applied to the system, the resulting inertial force deflects the suspended mass. The combination of the frame and the movable mass is designed in such a way that it creates a comb-like structure of electrodes. The resulting capacitance depends on the distance between the combs and hence changes when an acceleration is applied. The capacitive change can then be read out using a charge-to-voltage (CV) converter to create an electrical signal (an analog voltage). Consequently, the sensing chain can be split in three distinct steps [26]:

1. x/F : A force F is mechanically converted into a deflection x .
2. C/x : The deflection x creates a change in capacitance C .
3. V/C : The capacitance C is converted into an analog voltage V as an electrical read-out.

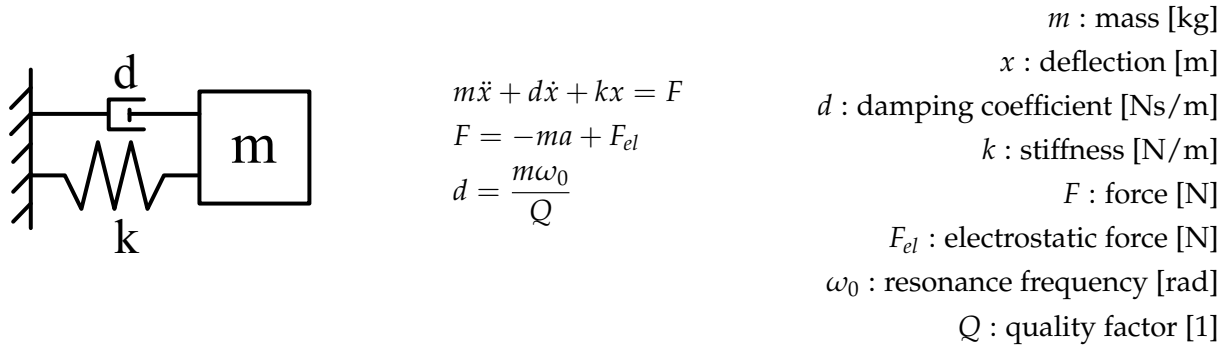


FIGURE 2.2: Mechanical equivalent circuit of a MEMS accelerometer. It represents a damped spring-mass system characterized by the given differential equation. Thereby, a force is transformed into a mechanical displacement.

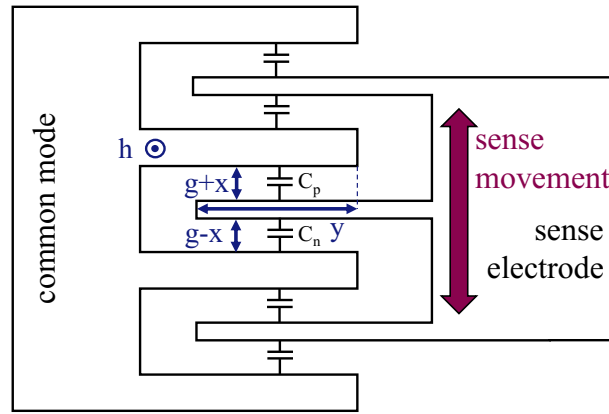


FIGURE 2.3: Schematic view of the electrode structure. The fingers of the common mode electrode and the sense electrode create a comb-like structure inducing capacitances between the fingers. The movement of the sense electrode changes the gap between the fingers which varies the capacitances.

Afterwards, the signal is converted to the digital domain using an analog-to-digital converter (ADC). The complete sensor system will be explained in a subsequent section. For now, focus is laid on the micro-electromechanical part. The first sensing step is realized through the suspended mass. Thereby, the mechanical system acts as a damped harmonic oscillator. Its equivalent circuit and the corresponding physical relations are given in Fig. 2.2. The suspended mass is connected to the anchor (the substrate) through a spring and a damper. The Laplace transfer function of the given differential equation evaluates to

$$H(s) = \frac{X(s)}{F(s)} = \frac{1}{m} \frac{1}{s^2 + \frac{\omega_0}{Q}s + \omega_0^2} \quad \text{with } \omega_0^2 = \frac{k}{m}. \quad (2.1)$$

The resonance frequency ω_0 is intentionally positioned above the bandwidth of the accelerometer. This ensures a flat transfer function within the region of interest and avoids high sensitivity errors.

The second step describes the behavior of the electrodes. A schematic view of the parallel plate electrodes is given in Fig. 2.3. Considering a single finger of the sense electrode, it creates two capacitances relative to the common mode. If the finger is perfectly centered in between the

fingers of the common mode electrode, both capacitances have an equal value

$$C_p = C_n = C_0 = \frac{\varepsilon_0 A}{g} \quad (2.2)$$

with ε_0 being the vacuum permittivity, $A = h \cdot y$ the electrode area resulting from height h and length y and g the gap between the electrodes at rest. As soon as the sense electrode starts to move, both capacitances change according to

$$C_p = \frac{\varepsilon_0 A}{g + x} \quad C_n = \frac{\varepsilon_0 A}{g - x} \quad (2.3)$$

with x once more denoting the deflection.

The third step is usually realized by a charge-to-voltage converter which outputs an analog voltage that is proportional to the differential capacitive change as given below.

$$\begin{aligned} C_p - C_n &= \varepsilon_0 A \left(\frac{1}{g + x} - \frac{1}{g - x} \right) \\ &= \varepsilon_0 A \frac{(g - x) - (g + x)}{(g + x)(g - x)} \\ &= \varepsilon_0 A \frac{-2x}{g^2 - x^2} \\ &\stackrel{x \ll g}{\approx} \varepsilon_0 A \frac{-2x}{g^2} = -\frac{2x}{g} C_0. \end{aligned} \quad (2.4)$$

It scales approximately linear with the deflection under the assumption that the deflection is much smaller than the total gap. This approximation inherently limits the full-scale range which is defined as the maximum range where the linearity error of the readout stays within defined limits. If the linearity shall be improved, a charge balanced frontend can be used where the analog output voltage becomes proportional to

$$\frac{C_p - C_n}{C_p + C_n} = \frac{\varepsilon_0 A \left(\frac{1}{g+x} - \frac{1}{g-x} \right)}{\varepsilon_0 A \left(\frac{1}{g+x} + \frac{1}{g-x} \right)} = \frac{g - x - (g + x)}{g - x + g + x} = \frac{-2x}{2g} = -\frac{x}{g} \quad (2.5)$$

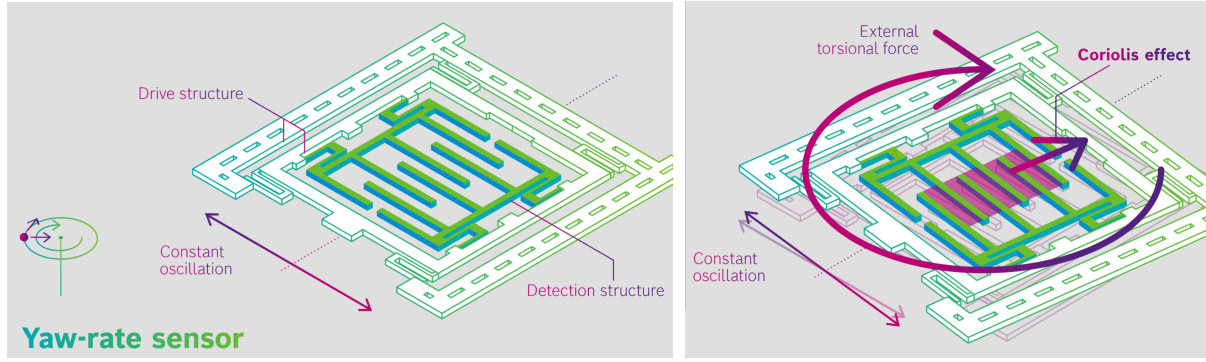
which is inherently linear to the displacement [32]. Please note that the displacement is not solely induced by the acceleration force. The electrodes additionally create electrostatic forces that act onto the system. A parallel plate capacitor with initial gap g and displacement x induces an electrostatic force

$$F_{el} = \frac{1}{2} \frac{dC}{dx} V^2 = \frac{1}{2} \frac{d}{dx} \left(\frac{\varepsilon_0 A}{g \pm x} \right) V^2 = \mp \frac{1}{2} \frac{\varepsilon_0 A}{(g \pm x)^2} V^2 \quad (2.6)$$

with V denoting the voltage across the plates [31]. As the force depends on the current deflection, it influences the behavior of the oscillator. Usually the effect is referred to as electrostatic spring softening as it can be interpreted as a change in the spring stiffness. The total stiffness k_{tot} is a sum of the electrostatic spring stiffness k_{el} and the raw mechanical stiffness k_{mech} according to

$$k_{el} = -\varepsilon_0 \frac{A}{(g \pm x)^3} V^2 \quad (2.7)$$

$$k_{tot} = k_{mech} + k_{el}. \quad (2.8)$$



(A) Operation without angular rate. The drive structure oscillates with fixed amplitude and frequency. (B) The combination of drive movement and angular rate creates a Coriolis force.

FIGURE 2.4: Schematic view of a MEMS gyroscope. It consists of a drive structure (white) and a suspended sense mass (green). (A) The drive structure is excited to oscillate in x-direction. (B) If a torsional force around the z-axis is applied, the Coriolis effect deflects the inner (sense) comb fingers along the y-axis, i.e. in orthogonal direction to the drive movement. © Bosch Sensortec GmbH

To be precise, the presented working principle defines an open-loop amplitude-modulated MEMS accelerometer, which is a widely used configuration in consumer applications. A more robust variant is the closed-loop amplitude-modulated accelerometer, in which displacements are suppressed through a feedback force that counteracts the apparent inertial force. The magnitude of the feedback force is then proportional to the acceleration. Given its greater size and cost, this topology is not typically utilized in the consumer market. However, it is a prevalent architecture in automotive applications. Furthermore, frequency modulated MEMS accelerometers have been presented in the literature [26]. These devices are capable of measuring the frequency change of a resonating mode. The resonance shift is induced by tensile or compressive stress applied to the resonating element (e.g., a beam). The stress is induced by the movement of a suspended mass. This topology will not be further considered in this work, as it has not been widely adopted.

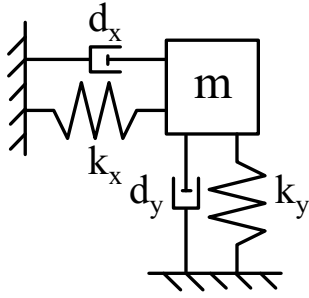
MEMS Gyroscope

A MEMS gyroscope measures angular rates based on the Coriolis effect. Its fundamental structure and working principle are illustrated in Fig. 2.4 and described below. For further reading, please refer to [30, 31, 33, 34]. Similar to the accelerometer presented above, a suspended mass is connected to an outer frame. In this configuration, the outer frame, designated as the drive structure, is actuated to undergo continuous oscillation (see Fig. 2.4a). Without loss of generality, the drive movement is defined to occur in x-direction for the following explanation and denoted by

$$\vec{v} = \begin{pmatrix} v_x \\ 0 \\ 0 \end{pmatrix}. \quad (2.9)$$

When an external torsional force (created by an angular rate) acts upon the system, the suspended mass is deflected through an apparent force called the Coriolis force. Assuming an angular rate $\vec{\Omega}$ rotating around the z-axis, the Coriolis force calculates to

$$\vec{F}_{Cor} = 2m_x \vec{v} \times \vec{\Omega} = 2m_x v_x \vec{e}_x \times \Omega_z \vec{e}_z = -2m_x v_x \Omega_z \vec{e}_y \quad (2.10)$$



$$m\ddot{x} + d_x\dot{x} + k_x x = F_x$$

$$m\ddot{y} + d_y\dot{y} + k_y y = F_y$$

$$F_x = F_{el,x} + F_{Cor,x}$$

$$F_y = F_{el,y} + F_{Cor,y}$$

m : mass [kg]

d_i : damping coefficient [Ns/m]

k_i : lateral stiffness [N/m]

F_i : force [N]

FIGURE 2.5: Mechanical equivalent circuit of a MEMS gyroscope. Two damped harmonic oscillators are shown, one for the drive and one for the sense direction. In reality, an additional cross-coupling term between both modes occurs due to manufacturing variations.

with m_x denoting the mass and Ω_z the scalar angular rate. The Coriolis force deflects the suspended mass towards the negative y -direction. The mechanical equivalent circuit of the presented structure yields a two-dimensional damped harmonic oscillator as illustrated in Fig. 2.5. In accordance with the direction of movement, the subscript x denotes the drive axis and the subscript y denotes the sense axis. Ideally, there is no coupling between these movements apart from the Coriolis force, which gives rise to the differential equations depicted in Fig. 2.5. The drive oscillation is induced by an electrostatic force based on sinusoidal or rectangular voltage schemes. The sense axis is subject to the apparent Coriolis force, whereas the electrostatic force is zero in the ideal case. For the sake of simplicity, other parasitic force terms are not considered here.

In practice, it is inevitable that a degree of cross-coupling between the axes will occur due to inherent manufacturing imperfections. These introduce additional off-diagonal terms in the stiffness and damping matrix when matrix notation is employed to account for the different directions [34]:

$$\begin{aligned} & \begin{bmatrix} m_x & 0 \\ 0 & m_y \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} + \begin{bmatrix} d_{xx} & d_{xy} \\ d_{yx} & d_{yy} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} + \begin{bmatrix} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \\ &= \begin{bmatrix} 0 & 2\Omega_z m_y \\ -2\Omega_z m_x & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} + \begin{bmatrix} F_{el,x} \\ F_{el,y} \end{bmatrix} \end{aligned} \quad (2.11)$$

Equation (2.10) demonstrates that the Coriolis force depends on the velocity of the driving movement. In order to infer the angular rate from the Coriolis force, it is necessary to have a well-defined velocity. Consequently, the drive movement is regulated such that it oscillates with constant amplitude $\hat{x}$ and frequency ω

$$x(t) = \hat{x} \sin(\omega t), \quad \hat{x} = \text{const.}, \quad \omega = \omega_0 = \text{const.} \quad (2.12)$$

such that

$$v_x(t) = \dot{x}(t) = \hat{x}\omega_0 \cos(\omega_0 t). \quad (2.13)$$

The Laplace transform of a damped harmonic oscillator (see Eq. (2.1)) allows the derivation of the amplitude and phase response, as illustrated in Fig. 2.6. It is evident that the greatest gain is attained in the resonant state. Thus, the drive frequency is regulated to match its resonance frequency ω_0 and enable minimal voltage levels to keep the drive oscillation at the desired amplitude. If the drive oscillation according to Eq. (2.12) is inserted into Eq. (2.10), the scalar

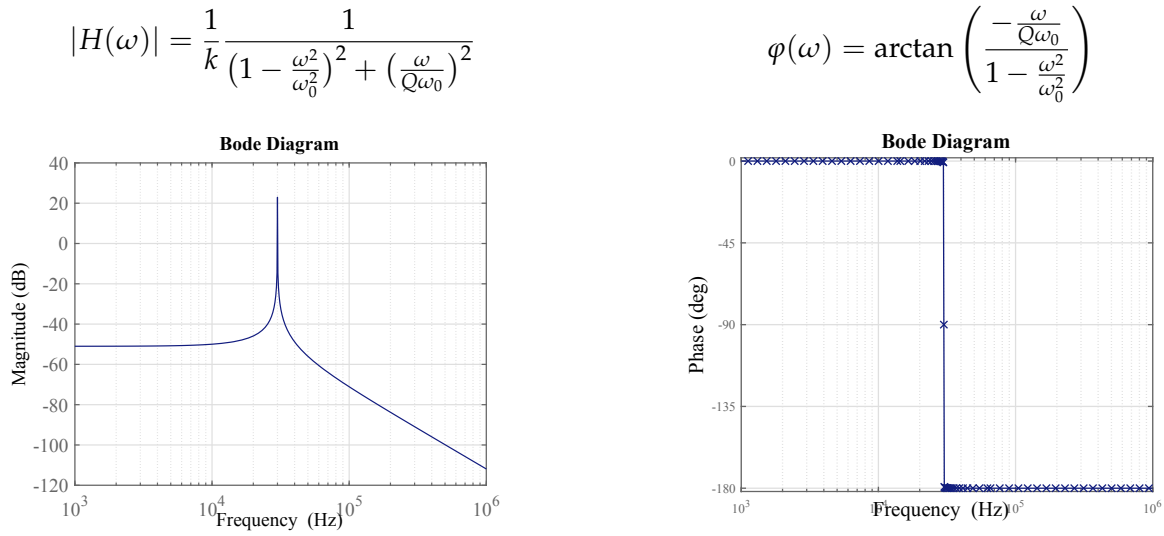


FIGURE 2.6: Transfer function of a damped harmonic oscillator. The gain $|H(s)|$ has a peak at resonance (here: $\omega_0 = 2\pi \cdot 30$ kHz). In this point, the phase delay is -90° and the ratio between resonance and DC amplitude reflects the quality factor (here: $Q = 5000 \approx 74$ dB).

Coriolis force results in

$$F_{Cor} = -2m_x v_x \Omega_z = -2m_x \dot{x} \omega_0 \cos(\omega_0 t) \Omega_z \quad (2.14)$$

which is modulated by the drive frequency. The cross-coupling term of the damping causes another force that is in phase with the Coriolis force whereas the cross-coupling stiffness causes a signal with a 90° phase shift:

$$F_{yx} = d_{yx} \dot{x} + k_{yx} x \quad (2.15)$$

$$= d_{yx} \dot{x} \omega_0 \cos(\omega_0 t) + k_{yx} x \sin(\omega_0 t) \quad (2.16)$$

Rewriting the latter as an equivalent angular rate yields

$$\Omega_Q = \frac{k_{yx}}{2m_x \omega_0} \quad (2.17)$$

which is called the mechanical quadrature. Thanks to the 90° phase shift, the rate and quadrature signal can be separated during demodulation.

With the help of Eq. (2.14) and the transfer function from Fig. 2.6, the Coriolis movement in y-direction can be derived to

$$x_y(t) = \frac{2m_x \dot{x} \omega_{0x} \Omega_z}{m_y \sqrt{(\omega_{0x}^2 - \omega_{0y}^2) + \frac{\omega_{0x}^2 \omega_{0y}^2}{Q_y^2}}} \sin(\omega_{0x} t). \quad (2.18)$$

The movement creates a capacitive change and is transformed into an analog voltage that is ideally in-phase with the displacement. The resulting voltage is then demodulated with a signal $\sin(\omega_{0x} t)$ which creates two frequency components. The term induced by the quadrature has a 90° phase shift such that the demodulation only causes a frequency component at $2\omega_{0x}$ and none within the sensor's bandwidth. The higher frequent components are suppressed by the subsequent filters.

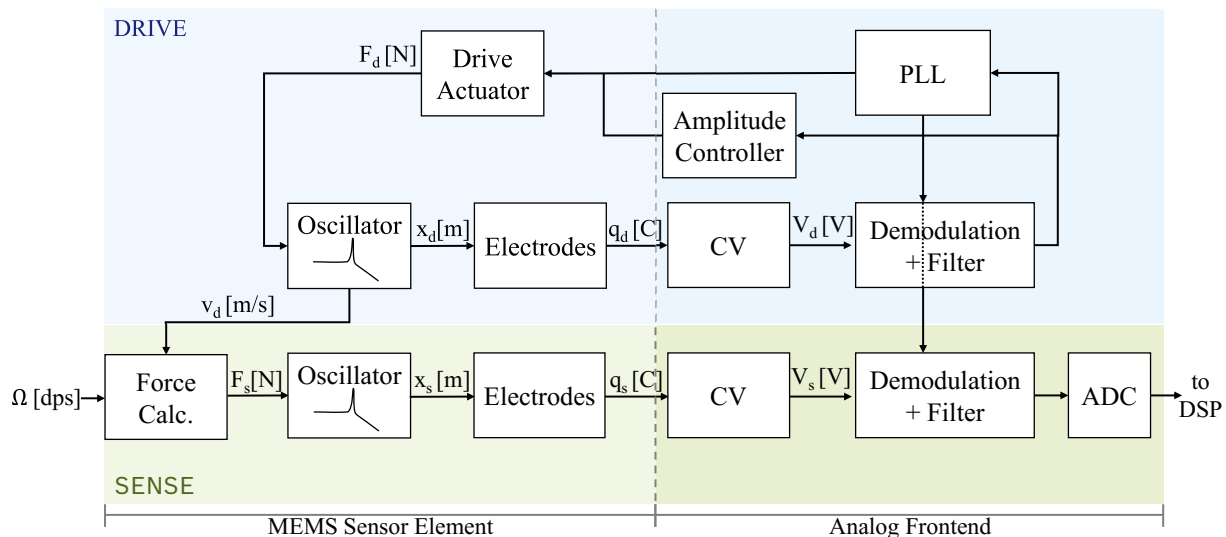


FIGURE 2.7: Block diagram of a single-axis MEMS gyroscope. The drive loop is given in blue at the top. An analog control loop actuates the drive mode to oscillate in resonance with constant amplitude. The sense path is illustrated in green at the bottom. The sense mass is deflected through the apparent Coriolis force and read out using a charge-to-voltage converter. The analog signal is demodulated and filtered before converting it to the digital domain where digital signal processing (DSP) is applied.

A block diagram of the complete analog and mechanical part of a MEMS gyroscope is given in Fig. 2.7. The diagram is divided into two sections: the left half represents the MEMS sensing element, and the right half represents the analog frontend (AFE). Drive and sense axes are separated vertically in the diagram. The drive axis creates a control loop between the MEMS and the AFE. The system comprises a phase-locked loop (PLL) that maintains a fixed phase difference of -90° between the actuating voltage and the output of the CV converter. This configuration stabilizes the frequency of the voltage-controlled oscillator (VCO), which is a component of the PLL, to the drive resonance frequency according to the phase response of the damped harmonic oscillator given in Fig. 2.6. The amplitude is regulated via the amplitude controller. The drive actuator fuses the amplitude and phase information to generate a driving voltage, which is then applied to the actuation electrodes. The resulting electrostatic force serves to actuate the MEMS.

The sense chain is exemplarily given for a single axis only. The configuration bears resemblance to that of the accelerometer frontend, albeit with the incorporation of an additional demodulation stage. In this block, the sinusoidal signal generated by the VCO is used to demodulate the sense signal, which was previously modulated by the Coriolis force within the MEMS. Assuming a constant angular rate, the output of the CV converter exhibits a single frequency term at drive resonance frequency ω_0 . The demodulation yields the desired DC component and an additional component at $2\omega_0$ in this case. The latter frequency component is eliminated through the application of successive low-pass filters following the ADC. The signal chain works similar for dynamic angular rates within the sensor's bandwidth.

Sensor System

A contemporary smart sensor is typically comprised of an application-specific integrated circuit (ASIC) that contains a substantial quantity of digital components. This paragraph broadens the preceding scope from mechanical and analog components to encompass the complete

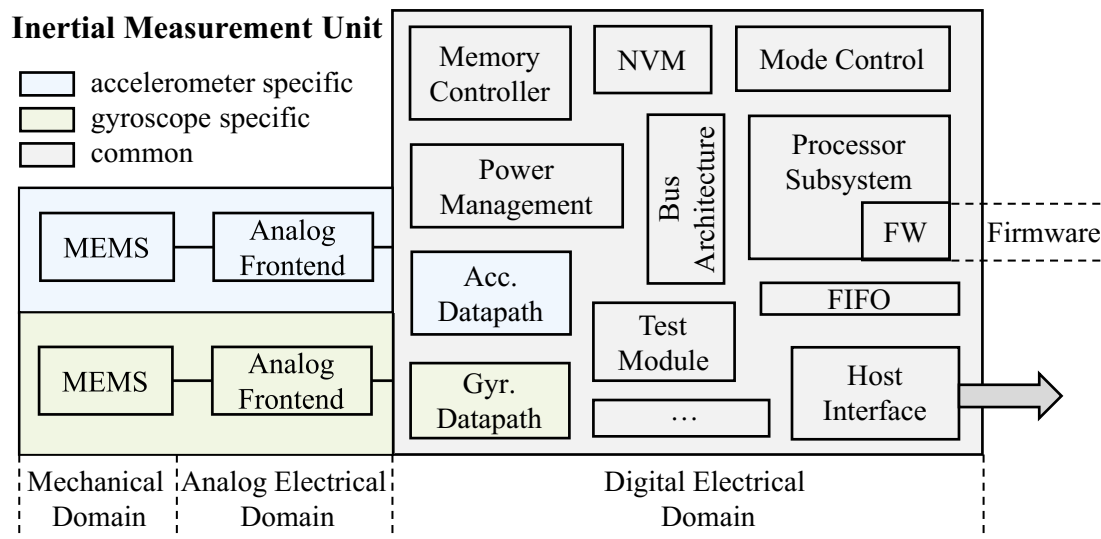


FIGURE 2.8: Schematic view of a MEMS inertial measurement unit. It integrates the mechanical and analog electrical part of an accelerometer (blue) and gyroscope (green) jointly with a set of digital modules. Raw data are postprocessed in a digital datapath, transferred via bus signals to the processor subsystem and the FIFO data buffer and streamed out to the host via SPI or I²C.

sensor system, with a particular emphasis on the digital content. It should be noted that the package, which constitutes an integral part of the overall sensor system, will not be addressed further in this discussion as it will not be reflected in the heterogeneous virtual prototype (HVP) that will be presented subsequently. As will be discussed in chapter 3, the HVP is most effective in reflecting the functionality of the system, rather than in predicting each and every non-ideal effect. Accordingly, the influence of the packaging on the system via its mechanical characteristics is not addressed.

Figure 2.8 illustrates the components of an inertial measurement unit. Besides the analog (mechanical/electrical) parts addressed in the previous sections, a set of digital modules is incorporated. These modules serve a variety of functions, including power management, data compensation and filtering within the datapaths, and the selection of sensor modes and configurations (e.g., full-scale range, output data rate, filter configuration). The internal communication is handled via a dedicated bus architecture. The samples are either transmitted to the processor for subsequent post-processing via firmware or directed towards the first-in, first-out (FIFO) data buffer, which is then accessed by the host. The communication with the host is usually performed using SPI (Serial Peripheral Interface) or I²C (Inter-Integrated Circuit) as communication protocols. Nowadays, also I3C (Improved Inter-Integrated Circuit) is supported by some sensors. The processor subsystem enables smart applications directly on the sensor. The processor subsystem facilitates the implementation of intelligent applications on the sensor itself. Moreover, the internal non-volatile memory (NVM) contains not only firmware but is also used to store trim values that have been defined during the final trim stage, prior to the product's delivery to the customer. In conclusion, the sensor system demonstrates a tight integration across multiple domains.

System-level modeling of MEMS

The analytical representation of the functionalities of MEMS, as previously outlined, provides an excellent foundation for system-level modeling of MEMS. However, the representation is significantly simplified, as it considers only the fundamental mode of each sensing axis. In

practice, however, a much larger set of modes can be excited. Its resonance frequencies lie beyond the fundamental modes. In order to obtain detailed information about the behavior of the mechanics, the finite element method (FEM) is employed for the purpose of simulating the layout of the MEMS sensing element. These fine-grained simulations are too time-consuming to be employed in system-level simulations. Consequently, a model order reduction is performed. The FEM simulation results are taken as the input, with the objective of reducing the order of the equation system while maintaining a high degree of fidelity to the initial behavior. A variety of model order reduction methods have been presented in the literature. Interested readers may refer to [35] for an overview on existing techniques. The application of this approach to MEMS system-level models has been presented by Blochmann et al. in 2017 [36]. The resulting model is called reduced order model (ROM). It can be represented in a state-space notation analogous to that of Eq. (2.11). In this representation, however, the dimensions correspond to different modes rather than different axes. The resulting equation system in the modal space can be written as follows:

$$M\ddot{X}_m(t) + D_m\dot{X}_m(t) + K_mX_m(t) = F_m(t) \quad (2.19)$$

with X_m the modal displacement vector, D_m the modal damping matrix, K_m the modal stiffness matrix, F_m the modal force vector and M the mass matrix. Usually, the system is normalized to the masses such that the mass matrix becomes the unity matrix. The resulting modal displacement is then finally converted into the displacement per sensing axis using corresponding weighting factors such that a classical model for the ASIC can be connected to the ROM.

2.2 (Digital) Virtual Prototyping

Historical Background

The term virtual prototyping was first introduced in the 1990s [37]. Focus was laid on strategies for hardware/software co-simulation, initially based on hardware description languages (HDLs) like Verilog or VHDL [38]. Hardware and software simulation were, for instance, connected using instruction set simulators (ISS) and inter-process communication (IPC) [39]. With the increasing complexity of SoCs and embedded firmware, the simulation performance of these setups became insufficient for two reasons. On the one hand, the level of abstraction of the hardware models was relatively low as they were derived from register transfer level (RTL) descriptions. On the other hand, the simulator coupling added an additional overhead. Over the years, these issues have been addressed incrementally through the introduction and standardization of SystemC and Transaction Level Modeling (TLM) interfaces. SystemC was first introduced in 2000 as a simulation language for system-level modeling [40]. In theory, it allows models to be created at a level similar to the register transfer level using signal ports as interfaces. However, its strength lies in a more abstract modeling style. SystemC gained much interest after the introduction of TLM 2.0 [41]. It provides an additional layer of abstraction for the communication between modules and a set of well-defined interfaces. Since then, SystemC has been widely adopted in the field of virtual prototyping and has become a de facto standard in this context [42].

Definition

The term virtual prototype (VP) has been used in various different fashions [43, 44]. In this work, a virtual prototype is defined as a simulatable computer model that reflects a physical product or system. Thereby, the model is usually split into components as is the system itself. A virtual prototype reflects the functionality of a system with timing information. The timing

accuracy can range from cycle-accurate up to loosely timed models. Moreover, extra-functional properties as energy or performance can also be simulated if the virtual prototype is annotated accordingly [45, 46]. In literature, the boundaries between a virtual prototype and a digital twin are fluid [47]. In this work, virtual prototypes are defined as system-level simulation models that are used during the development phase of a project, focusing thereby especially on hardware/software integration. The term digital twin is among others also used in the context of Industry 4.0 where the model runs during the operation of the physical system, i.e. in the field [48, 49]. Thereby, a continuous or at least regular bi-directional interaction between the physical twin and the digital twin takes place [50]. In contrast, the goal of a virtual prototype as defined in this work is to be a vehicle for product development and its application focuses on the development cycle, especially on early design phases when hardware is not yet available [8].

Purpose and Fields of Application

The original target of supporting hardware/software co-design is nowadays still the most popular application for VPs [51, 52]. Since SystemC is implemented as a C++ library, joint HW/SW simulation is natively supported. Instruction set simulators are typically used to emulate the processors' behavior. Virtual prototyping has gained much attention in this area in the context of RISC-V in the last years [53, 54].

A clear advantage of virtual prototypes is their early availability, which allows to frontload the software development [7]. Moreover, they are often still advantageous in the direct comparison with hardware models (e.g. emulations on FPGAs) because they support effective debugging due to high observability and controllability [6, 55]. A VP allows tracing of arbitrary internal signals, easy data injection, and fast development cycles to test the impact of structural or functional changes. Moreover, the insights gained on system level can then be taken to shape the RTL implementation [56, 57].

Other use cases include the use of virtual prototypes as reference models for verification purposes [43, 6, 58], fault injection [59, 60, 61], and power estimation [62, 45, 63, 46]. In addition, design-space exploration is performed, often in the context of HW/SW partitioning [64, 65, 66, 67], which then paves the way for high-level synthesis, which is also subject of continuous research to cope with the increasing complexity of modern SoCs [68, 69, 70].

A variety of solutions have been developed to support rapid prototyping in this context. Open Virtual Platforms (OVP) [71] provides a set of open source models to accelerate the development of virtual platforms for this use case. OVP provides several processor models (e.g. ARM, MIPS, Xilinx, OpenRisc), system component models (e.g., RAM, ROM, cache), peripheral models (e.g., FIFO, USB) and even some prebuilt platforms for running operating systems. It also provides a simulator called OVPsim. Wicaksana and Tang have demonstrated the applicability of virtual platforms on multiprocessor SoCs using OVP components and SystemC TLM 2.0 loosely timed modeling [72]. They show the benefit for various use cases as early software development, architectural exploration, estimated timing analysis, and functional verification. Commercial solutions are also available, such as the Virtualizer tool from Synopsys [73]. Some existing frameworks even support the combination of physical and virtual prototypes. For example, Kabir et al. [74] presented VIVE, a virtual prototyping framework for vehicle system-level testing. It connects a network simulator with electronic control units, sensors and actuators. The components can be either software models or physical prototypes such as a Raspberry Pi to emulate an electronic control unit. Communication among the clients is realized using the Transmission Control Protocol (TCP).

Much more work has been done in this area. Since the focus of this work is on heterogeneous VPs, they are not discussed further here.

SystemC

SystemC aims to support efficient system design especially for complex systems with hardware/software interaction. It is implemented as a C++ class library and standardized as **IEEE Std 1666™-2011** [40]. The architecture of a given system that shall be modeled is reflected by the hierarchical decomposition of the SystemC model into modules and the connectivity between them. Hence, the fundamental building blocks of a SystemC model are its *modules* which instantiate *ports* which are then connected by channels, either at the *signal* level or at the *transaction* level. Each module is assigned one or multiple *processes* which are triggered by events. These events may either be generated by signal changes or timeouts. Concurrency and timing behavior are handled by the dynamic scheduler.

Signal-level modeling allows the creation of cycle-accurate models. However, this usually results in a large number of events that slow down the simulation significantly. When this level of accuracy is required, modeling on RTL is often done instead using HDLs such as Verilog or VHDL. In contrast to that, SystemC is widely used for more abstract implementations based on transaction-level interfaces. TLM supports two coding styles, namely loosely timed (LT) and approximately timed (AT) modeling. The LT coding style utilizes the blocking transport of transactions. The AT coding style enables more accurate timing behavior through non-blocking transport at the cost of slightly increased simulation time.

Discrete Event Simulation Semantics

SystemC's simulation semantics are defined by the event-driven simulation kernel. A dynamic scheduler handles the activation of processes that are triggered by events, taking into account timing and execution order. Before the simulation starts, the *elaboration* phase sets up the module hierarchy and creates framework specific objects such as the scheduler. After elaboration, the composition of the simulation model is fixed and must not be changed. Then an initialization is performed before entering the main simulation loop. The execution semantics of SystemC are illustrated in Fig. 2.9. The main simulation loop consists of three phases as described below.

Evaluation Phase During evaluation, the list of runnable processes is executed. A process becomes runnable when its activation condition is fulfilled, i.e. when it is triggered through an event. An activated process runs until it is either completed or suspended. It is then removed from the list of runnable processes. Suspension occurs when a *wait* statement is executed. The argument of the *wait* statement defines the next activation condition of the given process. It can be sensitive to events or timeouts, such that the process is triggered either when the specified event is notified or when the specified simulation time has elapsed. Each process can notify (trigger) events. If the notification is performed immediately, all processes sensitive to that event are added to the set of runnable processes and executed in the same evaluation phase. A notification can also be delayed. Moreover, a process can request an update of a channel (e.g., a signal).

Update Phase The update phase handles all pending updates that were requested in the previous evaluation phase, such as the update of a port or signal value. An update of a channel schedules a delta notification that will be handled in the next delta cycle. Thereby, a delta cycle describes an infinitesimal small time step that is utilized by SystemC to order event chains at a given simulated time.

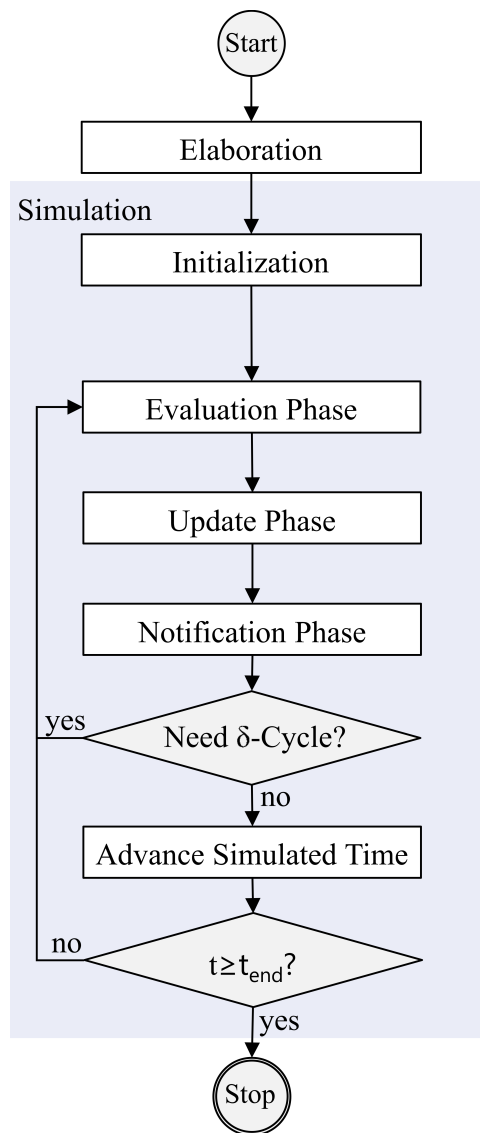


FIGURE 2.9: Simulation semantics of SystemC.

Notification Phase The notification phase splits up into two parts. First, the delta notification phase is executed. For each pending delta notification and timeout, processes that are sensitive to these events are added to the set of runnable processes (which has been empty at the beginning of this phase). If the number of runnable processes is non-zero, a delta cycle must be executed and the next evaluation phase is triggered. A delta cycle consists of another consecutive execution of the three phases explained (evaluation, update and delta notification phase). Simulation time has not advanced or rather only by an infinitesimal step as mentioned.

If the number of runnable processes is zero after a delta notification phase, the timed notification phase executes. The simulation time advances up to the time when the first pending notification occurs due to timeout. Afterwards, all sensitive processes are added to the set of runnable processes and a new evaluation phase starts.

The simulation ends if the list of runnable processes keeps empty after the timed notification phase.

2.3 Analog/Mixed-Signal Modeling

Languages and Abstraction Levels

The level of abstraction appropriate for a simulation model depends heavily on its intended purpose. Each level of abstraction is associated with a distinct set of requirements, benefits, and drawbacks. Consequently, a variety of simulation languages exist that are suited to different modeling styles. Figure 2.10 illustrates the available levels of abstraction for analog models as specified by Paugnat et al. [75] applied to the example of an inertial measurement unit. Thereby, suitable simulation languages are denoted for each level. The most abstract level of modeling is represented by *system-level* models. As this layer is similarly available in the digital domain, both analog and digital modules are presented together. Extracting a set of semantically tightly connected blocks yields the *functional unit level* which is also available for analog and digital units. In the given example, the analog frontend is defined as one analog functional unit because its components have the common objective to convert the change of charges at the MEMS electrodes into a sampled electrical signal. In the layer below, it is divided into its fundamental blocks, such as the charge-to-voltage converter. These blocks are referred to as modules which in turn are implemented from basic analog circuit blocks, such as the operational amplifier ("opamp"), as shown. Accordingly, these two levels are called *analog module level* and *analog module implementation level*. The latter is comparable to the RTL in the digital domain. Splitting these implementation components further ultimately yields the *transistor level*.

The thesis at hand discusses virtual prototypes ranging from the system level to the module level. As denoted in Fig. 2.10, both Simulink and SystemC AMS are capable of spanning these levels and are introduced below. SystemC AMS has proven to be an effective language for enabling rapid simulation, and will be the primary modeling language utilized throughout the dissertation. Therefore, its semantics are discussed in greater detail. The rationale behind this choice is explained in detail in chapter 4.

MATLAB/Simulink

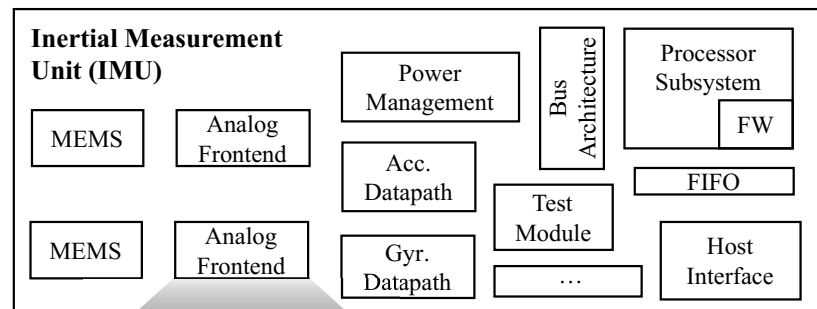
A widely adopted tool for AMS simulation is MATLAB/Simulink, a commercial software from MATHWORKS [76, 77]. MATLAB implements a numerical programming environment that focuses on matrix and array algebra. It provides a live editor for coding as well as capabilities for creating scripts, classes, and so on. Simulink, on the other hand, supports the modeling, simulation and analysis of systems in a graphical fashion. Models are built from block diagrams in which each block describes a computational unit, similar to a module in SystemC AMS, and the lines between them represent the channels that connect them. Simulink's semantics are based on signal flow graphs. To support the efficient modeling, the tool provides a rich set of libraries and primitives that can be used for rapid prototyping. Hierarchical systems are realized grouping basic blocks into *subsystems*. Simulink also supports a variety of solvers, including fixed-step and variable-step solvers for both discrete-time and continuous-time modeling styles. In addition, the *Embedded Coder* enables users to generate C/C++ code directly from MATLAB functions or Simulink models.

SystemC AMS

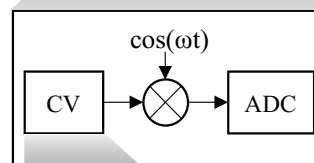
The analog/mixed-signal extensions to SystemC were specifically introduced to tackle the needs of heterogeneous systems where AMS blocks are intrinsically linked to digital HW/SW systems [78]. The language reference is standardized as **IEEE Std 1666.1™-2016** [79] and a proof-of-concept implementation is provided by ACCELLERA [14]. It introduces three distinct models of computation (MoCs), namely timed data flow (TDF), linear signal flow (LSF), and

System Level

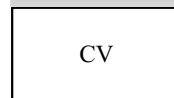
SystemC-AMS TDF, Simulink

**Analog Functional Unit Level**

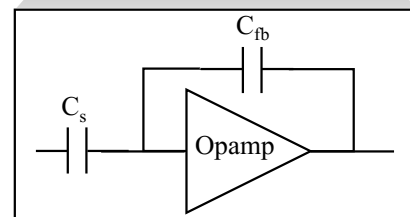
SystemC-AMS TDF, LSF, ELN, Simulink

**Analog Module Level**

SystemC-AMS TDF, LSF, ELN, Simulink, Verilog-A(MS), VHDL-AMS

**Analog Module Implementation Level**

SystemC-AMS ELN, Simulink, Verilog-A(MS), VHDL-AMS, Spice

**Transistor Level**

Spice

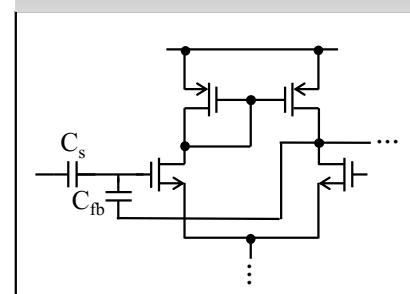


FIGURE 2.10: Analog abstraction levels at the example of a MEMS inertial measurement unit. Suitable simulation languages for each level are given. The system level considers the complete sensing chain from the physical input towards the data read by the host. It consists of multiple analog and digital functional units. The analog frontend is taken as example for an analog functional unit. Its modules are jointly responsible to transform the change of charges at the MEMS electrodes into an electrical signal. One of these modules is the CV converter. Its implementation includes basic circuit blocks as an operational amplifier ("opamp"). Splitting this further into transistors yields the lowest level of abstraction.

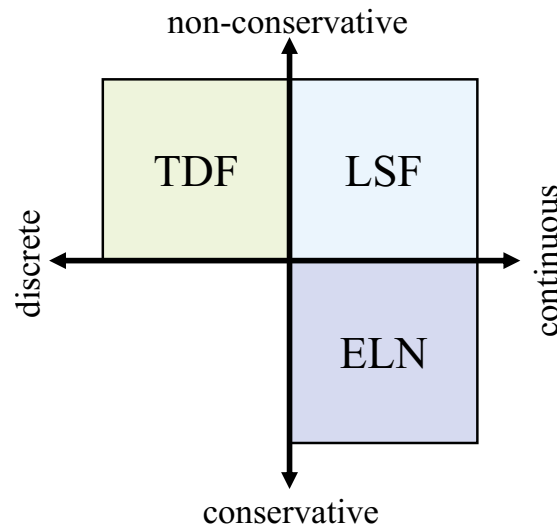


FIGURE 2.11: SystemC-AMS models of computation. The linear signal flow (LSF) MoC and electrical linear network (ELN) MoC both target linear continuous-time modeling. The timed data flow (TDF) MoC covers the discrete-time domain and supports nonlinearities. The ELN MoC reflects conservative systems.

electrical linear network (ELN). Their basic categorization based on timing behavior and semantics is illustrated in Fig. 2.11.

Timed Data Flow (TDF) Model of Computation

The semantics of the timed data flow MoC are based on the concept of synchronous data flow graphs [80], in which modules consume and generate tokens. The complete set of consumed and generated tokens must be in balance in order for the system to be schedulable. The TDF MoC enables the abstract modeling of non-conservative systems and is the sole MoC of SystemC AMS that permits the direct representation of nonlinearities. Each module is defined by a set of characteristics, including its timestep, in- and output port rates, and associated delays. The execution order of all modules within a given cluster is defined by a static schedule, which is calculated once during the elaboration phase. In this way, a cluster is defined as a set of connected modules. The timestep of a given module may be assigned explicitly or inherited from neighboring modules. In the case of multirate clusters, input and output rates are taken into consideration to derive inherited timesteps.

The scheduling procedure is illustrated at the example provided in Fig. 2.12. The cluster is comprised of three modules designated as A, B and C. In the absence of an explicit port rate specification, the default value of one is applied. The input rate of module B is equal to three and its output rate is one. Therefore, the module consumes three tokens (three input samples) for a single processing cycle and generates one output sample. In order for module B to be executed, module A, which has an output rate of one, must be executed three times until a sufficient number of tokens have been generated to run module B once. Consequently, the module timestep of module B is three times that of module A. Module C, which takes one input sample at a time, runs on the same timestep as module B. To ensure schedulability, module C must generate three tokens in a single processing step, allowing module A to be triggered three times in the subsequent step. As a result, a single cluster execution (once processing the entire schedule) includes three executions of module A and only one of the two other modules. Given that the system incorporates a feedback loop, it is necessary to introduce a delay to at least one module in order to make the system schedulable. In this case, the delay is applied to the output of module B, which is delayed by one sample. An initial value is assigned to

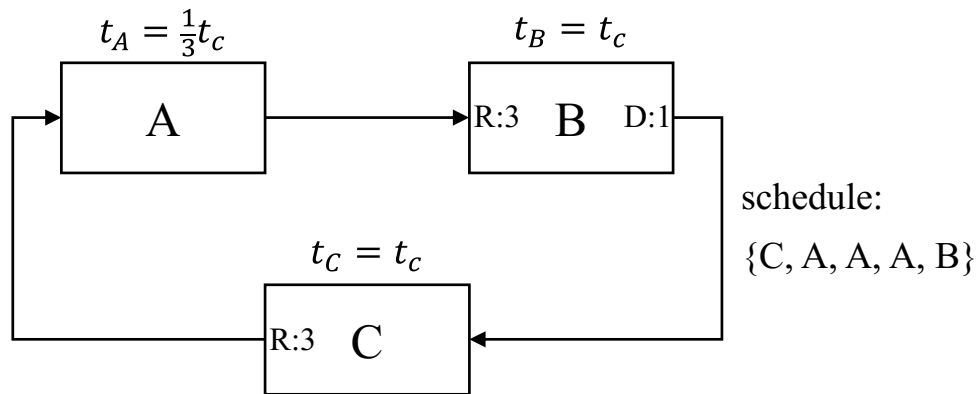


FIGURE 2.12: Exemplary TDF cluster consisting of three modules A, B and C with R denoting port rates and D port delays. Their module timesteps are given relative to the cluster timestep t_c . The delay is mandatory to break the algebraic loop.

the output of module B. Therefore, module C can commence its calculation immediately and is the first module to be executed in the schedule. The interval of time between two consecutive cluster executions is referred to as the cluster timestep and denoted with t_c in Fig. 2.12. It is derived from the module timesteps. To be precise, it represents the least common multiple of all module timesteps.

The semantics described above are established through a predefined interface that each TDF module is required to implement. This interface encompasses the following methods:

set_attributes() Optional method to be implemented. The method is invoked once during elaboration (before simulation start), enabling the user to define the module timestep, as well as input and output rates and/or delays. In the absence of a specified timestep or the absence of the method itself, the timestep is inherited from connected modules. Thus, each cluster must include at least one module that implements this method.

initialize() Optional method to be implemented. It is called once after the end of elaboration. It enables the user to initialize local variables and assign initial values to ports if the port delay is non-zero.

processing() Mandatory method to be implemented by each module. It is called in each timestep during simulation to process the module according to the static schedule. Thereby, the method computes the module's outputs based on the inputs and/or internal states.

ac_processing() Optional method to be implemented. It is only required for alternating current (AC) simulation as it implements the frequency behavior of the module and replaces the processing call for this type of simulation. A consistency check between a module's time- and frequency-domain behavior is not available for SystemC AMS.

Since the release of SystemC AMS 2.0, the constraint for a fixed timestep has been slightly softened. Firstly, it is possible to change the timestep during simulation, but only after each cluster execution. Secondly, an option has been implemented that triggers modules based on events. For that purpose, the TDF module interface was extended by additional methods as described below:

change_attributes() Optional method to be implemented if dynamic TDF features shall be used. The method is invoked once after the completion of each cluster execution. It enables the modification of the module's attributes (timestep, port rates and delays). It is imperative that the module explicitly indicates these alterations to the timestep by setting the corresponding flag (*does_attribute_changes*) in the *set_attributes()* method. Connected modules must accept these attribute changes by setting the *accept_attribute_changes* flag in order to ensure consistency. Moreover, an event may be registered that triggers the next module execution through a call of *request_next_activation()* within this method. When used in conjunction with the maximum timestep setting (*set_max_timestep()*), it enables the triggering of a module either by an event or latest after the maximum timestep.

reinitialize() Optional method to be implemented. It is called after the *change_attributes()* call to allow a proper reinitialization of ports whose properties have been changed.

In addition to the normal input and output ports, the TDF MoC also provides converter ports that connect a TDF cluster to the discrete event (DE) domain of SystemC. A DE-TDF input port reads from the discrete event domain. The interface can be designed to be either data or time synchronized [78]. In the case of data synchronization, the TDF module is activated independently of any events from the connected DE channel. It samples the DE signal at equidistant timesteps, regardless of whether the value has changed and an event has been fired. In contrast, time synchronization triggers the TDF module based on events happening on the connected DE channel. The TDF module sets a maximum timestep after which it will be triggered if no event occurs. Otherwise, the module is triggered earlier at the time of an event. A TDF-DE output port on the other hand writes from a TDF module to the discrete event domain. If it is connected to an *sc_signal*, an event is generated whenever the new sample differs from the previous one. If it is connected to an *sc_buffer*, an event is generated for every sample, no matter whether the value has changed.

Linear Signal Flow (LSF) Model of Computation

The linear signal flow MoC targets the modeling of linear non-conservative systems. It is based on the signal flow graph formalism firstly coined by Samuel J. Mason [81]. It describes a system as a set of nodes that are connected by directed branches. In SystemC AMS, an LSF module represents a node and the signals inbetween represent the branches. Predefined primitives as adders and subtractors, integrators and differentiators, gain stages, (de-)multiplexers and others are provided. Thereby, control signals, e.g. for the multiplexers, are steered either by TDF or DE signals. Moreover, converter ports to the TDF MoC are also provided that allow to mix TDF and LSF clusters within a single model. Direct conversion to the ELN MoC or the discrete event domain are not supported, connecting them to the LSF MoC always requires an intermediate TDF module. Moreover, the creation of custom primitives is not supported.

During elaboration, a differential equation system is set up that represents the LSF cluster. It is then solved during simulation using a differential algebraic equation (DAE) solver. To ensure a proper synchronization with the TDF time grid, LSF modules have a discrete timestep as well. It can either be set explicitly during elaboration or inherited from connected TDF modules. The DAE solver will generate solutions for at least the points in time specified by the discrete time grid. If necessary, it may generate additional solutions in between.

Electrical Linear Network (ELN) Model of Computation

The electrical linear network MoC implements the conservative counterpart to the LSF MoC. It is also based on predefined primitives and is simulated by DAE solvers. In contrast to the LSF

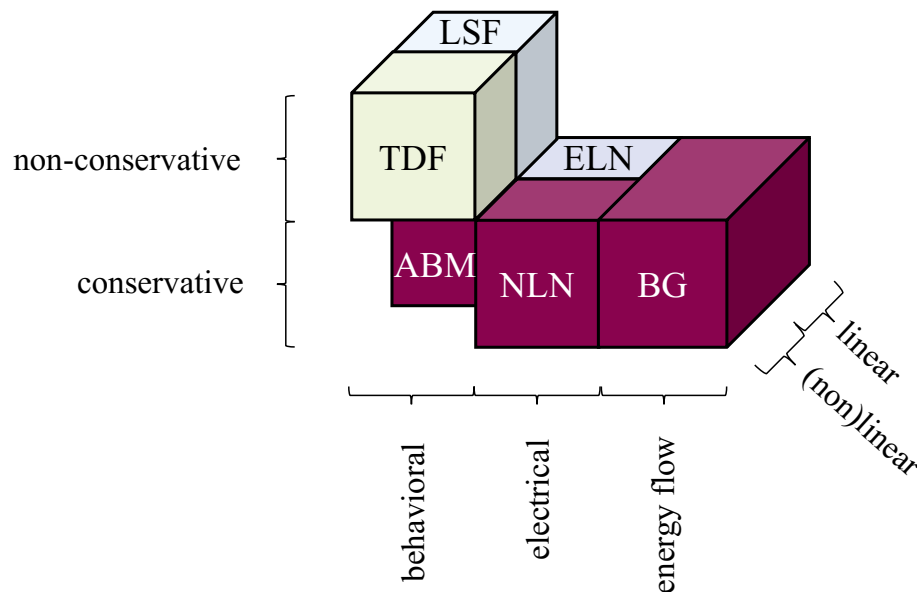


FIGURE 2.13: Existing MoCs for SystemC AMS. The violet ones are not part of the standard or the proof-of-concept implementation but have been presented in literature ([82, 83]). The analog behavioral modeling given in grey is not a fully new MoC but a new way of creating ELN models.

MoC, Kirchhoff's circuit laws apply. The primitives include linear passive components (resistors, capacitors, inductors), ideal switches, voltage and current sources, and others. This makes it well suited for filter design and similar applications that benefit from an implementation-centric modeling style. However, it also has the inherent limitation of linearity to avoid time-consuming nonlinear differential equation solvers. Similar to the LSF MoC, control signals are driven from the TDF MoC or the DE domain, e.g., a voltage-controlled current source supplies a current in the ELN MoC but the control voltage is represented by a TDF or DE signal. Again, converter ports are provided only to the TDF MoC.

During elaboration, a modified nodal analysis is conducted to set up the differential equation system that is then solved during simulation. The timestep requirements are equal to those of the LSF MoC.

Non-standard Extensions

Since the release of SystemC AMS, additional models of computation have been presented in literature that further extend the language. Figure 2.13 illustrates the domains covered. First, a model of computation based on the bond graph formalism was presented by Mähne et al. in 2008 [82]. Bond graph (BG) models represent the energy flow and can be applied to various physical domains. They are widely adopted to represent mechatronic systems. The new MoC has been integrated into SystemC AMS similarly to the existing MoCs.

In 2010, an extension of the ELN MoC towards nonlinear circuits (NLN MoC) has been proposed by T. Uhle and K. Einwich [83]. It fills the gap of nonlinear conservative modeling. In contrast to the ELN MoC, primitives are not predefined but created by the user. It additionally extends the modeling from the electrical domain towards other natures (e.g. kinematic). Since the extension requires the introduction of nonlinear solvers to SystemC AMS, the proper handling of nonlinear dynamic networks with discontinuous stimuli as induced by the DE domain is discussed by T. Uhle and G. Elst [84].

The third extension shown in Fig. 2.13 covers analog behavioral modeling (ABM). It was first presented by Vinco et al. in 2015 [85, 86]. In contrast to the BG and NLN MoC, it does not

intend to extend the SystemC AMS library itself. Instead, the authors propose to use ELN primitives in a novel way. The target is to support behavioral conservative modeling similar to VHDL-AMS or Verilog-AMS. They present a scheme that maps simultaneous statements, used in Verilog-AMS to describe relationships between nodes, to ELN primitives. Thereby, they make excessive use of controlled sources. To ease the handling of this scheme, an automatic translation of Verilog-AMS models into the corresponding ELN representation is implemented. As a result, it becomes possible for a user to start with behavioral modeling and end up with a SystemC AMS model that can then be integrated into a larger virtual prototype. However, the modeling is not done directly in SystemC AMS.

In addition to the introduction of new MoCs, support for dimensional analysis has been proposed for SystemC AMS [87]. The implementation is based on the open source *Boost.Units* library [88].

2.4 Heterogeneous Virtual Prototypes

The definition given in the previous section for virtual prototypes also applies to heterogeneous VPs. Regardless, the term heterogeneous should be clearly defined to allow for proper separation. Heterogeneity is used in two different ways in the context of virtual platforms. On the one hand, it refers to systems with a variety of different processor cores, i.e., purely digital heterogeneity. On the other hand, it is also used to describe systems that contain both digital parts and analog/mixed-signal or other physical domains. In this thesis, the term heterogeneous is used according to the second definition. Thus, a heterogeneous virtual prototype is defined as a computer model that reflects the functionality of a system that includes both digital and analog/mixed-signal blocks.

Existing Solutions and Fields of Applications

Over the past 20 years, heterogeneous virtual prototyping has been subject to numerous research to address the complexity of heterogeneous SoCs. Manifold simulation frameworks have been created to support the effective prototyping of these systems. In 2003, Ptolemy II was presented as one of the first heterogeneous modeling frameworks, which is still widely known today [10]. It structures the composition of various domains through a hierarchical heterogeneity concept. A detailed description of the approach, supported MoCs and a getting started guide can be found in [89]. Three years later, HetSC, a C++ library that builds upon the SystemC standard, was presented [11]. It defines a set of communication channels to create the semantics of different MoCs, but mainly focuses on untimed models.

In addition to the SystemC AMS 1.0 standard published in 2010 [90], the HetMoC framework was also introduced in the same year [12]. It combines the simulation of discrete event, continuous-time (CT) and synchronous/reactive (SR) models using adaptive ODE solvers for the CT simulation. In 2013, the updated SystemC AMS 2.0 standard was introduced, firstly incorporating the DTDF semantics [79]. Two years later, SystemC MDVP (multi-domain virtual prototyping) was presented by Aoun et al. [13] in response to the lack of a user-friendly interface for extending SystemC AMS with new MoCs. The underlying theory brought back the idea of hierarchical structuring of different MoCs. Two MoCs always realize a master-slave relationship, where the slave has to adapt to the semantics of the master and provide a proper interface. Although not explicitly stated, SystemC AMS follows a similar approach. The TDF MoC acts as master to both the LSF and ELN MoCs. As a consequence, the continuous-time MoCs provide solutions at each TDF timestep to ensure proper interfacing. The TDF MoC in turn acts as a slave to the DE semantics of SystemC. From a SystemC perspective, it appears as

a simple DE thread. To the best of the authors knowledge, SystemC MDVP, although easily extensible in theory, so far only supports MoCs for timed data flows and electrical networks [91]. The only semantic extension compared to SystemC AMS is therefore the existence of nonlinear primitives for the electrical domain.

The author of the thesis at hand does not claim that the set of frameworks presented above is complete, but it does give an impression of how much effort has been put into the topic. Within this broad landscape of heterogeneous modeling options, SystemC AMS has gained major interest. In recent years, it has been adopted in a wide variety of applications. In the first years after its release, this especially incorporated RF applications [92, 93, 94] and the modeling of different sensor types like MEMS sensors [95, 96], CMOS video sensors [97], chemical sensors [98], as well as complete sensor networks [99]. Moreover, it has been applied in biology for the simulation of lab-on-chip [100, 101] and microfluidic systems [102]. A few publications have been presented in other scientific areas such as chemistry [98] and optics [103]. In recent years, additional areas of application include electric vehicles and aircraft [104, 105, 106], cyber-physical production systems [107] and in-memory computation [108, 109]. Moreover, System AMS has also been deployed in combination with RISC-V through the development of the RISC-V-AMS platform [110].

Chapter 3

The Impact of Heterogeneous Virtual Prototypes on the Smart Sensor Development

The approach of using heterogeneous virtual prototypes is generally applicable to all kinds of heterogeneous systems. This chapter focuses explicitly on HVPs for the sensor development and addresses *RQ1-Impact*. It presents use cases for virtual prototypes that are not sufficiently covered by purely digital models and demonstrates how the use of HVPs positively influences the development flow of smart sensors. To evaluate this impact, the development of a high performance inertial measurement unit was used as an industrial case study. The author of the thesis at hand developed a SystemC AMS model including the MEMS and the analog frontend of the accelerometer and gyroscope of the IMU and integrated it into an existing digital virtual platform. Other project members were encouraged to make feature requests throughout the project phases in order to actively shape the model such that it meets real industrial needs. A complete sample phase of the project has been monitored. This industrial case study is the experimental basis for this chapter, which is structured as follows. First, related work is presented in section 3.1. The main use cases for the HVP that have been evaluated throughout the case study are presented in section 3.2, and the model requirements to best support them are discussed in section 3.3. The model implementation is described in section 3.4 to illustrate the appropriate level of abstraction, followed by some exemplary simulation results in section 3.5. Finally, the impact on the development flow is evaluated in section 3.6 before the concluding discussion is conducted in section 3.7. Parts of the presented work in this chapter, specifically regarding the impact of HVPs on firmware development, have been previously published by the author of this work [P1]¹.

3.1 Related Work

The improvement of development processes through virtual prototyping has been discussed in literature for years. Besides the digital-centric approaches that focus on HW/SW interaction [51, 52, 7, 111], several top-down design approaches for mixed-signal systems have been presented. Pagnat et al. describe the need to improve the analog refinement process and bring it closer to the digital one [75]. They propose a combined refinement process that starts with a SystemC model at the system level and introduces SystemC AMS for subsequent (refined) levels such as the analog module and block level. This allows joint simulation of the digital and analog domains during the top-down process, which then ends with implementation at the transistor level. Li et al. present another top-down approach based on SystemC AMS [112]. However, their definition of *system level* fits the functional unit level according to the definitions given earlier in Fig. 2.10 on page 22. They use SystemC AMS TDF models as containers for all

¹See section 1.3 page 5 for the author's contribution to this publication.

hierarchy levels. As SystemC AMS itself does not support transistor-level simulation, the TDF module calls an electrical simulator when the circuit level is targeted. They have created an interface to the CHAMS analog sizing engine (see [113] for reference), which initializes the circuit after the end of elaboration of SystemC AMS. During simulation, the *processing()* method then calls the electrical netlist simulator.

Another top-down development process based on SystemC AMS was presented by Barnasconi and Adhikari [114]. They divide the top-down process into three phases. The top level is the algorithmic or functional level. An executable specification is created that reflects the set of requirements. In this phase, highly abstract models are created in SystemC and SystemC AMS. In the second phase, the architecture is explored and concepts are developed. The digital components are modeled at the transaction level and TDF models are employed to implement the analog concepts. The third and lowest level is the implementation level, wherein the analog and digital designers construct the actual circuits. The data obtained during this process can then be utilized to refine the models that were created in earlier phases. In contrast to the other development processes presented, Barnasconi and Adhikari also consider the system validation. The aforementioned three steps represent the top-down breakdown of a system into components. As a second step, the validation is often done in a bottom-up manner where components are validated first and system-level validation is done in a consecutive step. To facilitate these kinds of top-down approaches, Genius and Apvrille present TTool, a tool that allows to start the top-down process from SysML models and to integrate SystemC and SystemC AMS TDF and ELN models for subsequent levels of hierarchy within the same tool [115]. They emphasize the visual aspect, for instance the generation of ELN models, and provide code generation capabilities.

The commonality between all of these methodologies presented is their focus on the hardware design process of mixed-signal systems. As no particular product area is taken into consideration, the use cases and development flows remain highly generic. The design flow presented by Barnasconi and Adhikari aligns well with the approach employed in this work [114]. The heterogeneous virtual prototype is refined throughout the project, though this process cannot be clearly delineated into two distinct stages, as the refinement occurs continuously. The implementation at the transistor level remains an isolated simulation environment. Nevertheless, a feedback of results to the HVP is targeted. This work differs from existing literature in that it identifies the particular requirements of sensor development and expands the scope beyond the mere engineering of hardware towards firmware development and trim and test procedures.

Some literature on virtual prototypes can also be found specifically in the context of sensors. It has been widely adopted for wireless sensor networks [116]. A variety of modeling approaches and frameworks have been presented that model a sensor node with SystemC and/or SystemC AMS, for example see the work of Vasilevski et al. [117], Du et al. [118] or Navarro et al. [119]. However, the underlying sensor models are typically quite simple, as the emphasis is placed on the network simulation. For instance, Vasilevski et al. simply model the sensing unit as a current source with a parallel load resistor [117]. Therefore, these works are not suitable for adaption to the context of sensor development.

Two works on the virtual prototyping of MEMS sensors that are closely related are given by Mähne et al. [95] and Cenni et al. [120]. Both works clearly highlight the necessity for HVPs, designated as multi-disciplinary VPs (MDVPs), for the efficient development of MEMS sensors. In this way, Mähne et al. demonstrate the benefits of an HVP exemplarily on the co-design of a MEMS vibration sensor and its configurable frontend [95]. Thereby, the sensing element is subjected to three stages of refinement. Initially, the sensing element is modeled in a simple manner using a Laplace transfer function in SystemC AMS. Subsequently, the model is refined towards an energy-conservative model to investigate the interactions between the mechanical

and analog domains. As non-linear conservative modeling is not supported by SystemC AMS, the authors make use of the bond-graph formalism that they had introduced in a previous publication [82]. The third step culminates in the modeling of the MEMS sensing element as a reduced order model (ROM), which is then integrated into SystemC AMS using state-space notation. The sensor frontend is configurable via a TLM interface that controls the sensor's register bank. As a result, the bidirectional interactions between the sensing element and the frontend can be effectively simulated. The study by Mähne et al. is primarily concerned with the issue of verification. Consequently, the model is embedded into a UVM-SystemC-AMS test environment.

Cenni et al. further develop this approach by integrating a microcontroller model, thereby enabling the joint simulation of the multi-disciplinary hardware with embedded software features [120]. The top-down approach, which contains a TDF model, a bond-graph representation, and a ROM for the MEMS, is analogous to that described in the previous publication. However, in this case, the authors have integrated a central processing unit (CPU) that interacts with the register banks. An illustrative example is provided in the form of a model of an IMU with an additional magnetometer. While the authors recognize the necessity of supporting the development and validation of embedded software features with HVPs, their work does not address this aspect further. Instead, they compare the various modeling techniques for the micro-electromechanical element. The thesis at hand builds upon and expands related work by explicitly identifying types of embedded software that necessitate the use of HVPs. It also discusses the specific requirements that must be met by the virtual prototype and provides evidence of the effectiveness of the approach, outlining both the benefits and drawbacks.

A number of other publications address the system-level modeling of MEMS sensors. In their discussion of system-level modeling of MEMS, Bechthold et al. examine various technologies [121]. The authors identify three categories that align well with the refinement levels presented in the aforementioned works. These include lumped-element models, with an emphasis on physics-based lumped-element models (which can be effectively modeled in the TDF MoC of SystemC AMS), model order reduction strategies, and modeling using generalized Kirchhoffian networks. This latter term describes the extension of Kirchhoff's conservation laws to other energy domains [122]. The bond-graph formalism is consistent with this theory. Another study examining the various abstraction levels of MEMS sensors has been presented by Shafique et al. [123]. Their system-level model can be classified as a lumped-element model and is implemented in VHDL AMS. Furthermore, they employ a ROM for additional refinement. The FEM simulations are considered the lowest level of the hierarchy, and their results are utilized to provide feedback on certain design parameters. A methodology for exporting ROMs from FEM simulations conducted in the MEMS+ tool to SystemC AMS can be found in the work of Vernay et al. [96]. An alternative methodology for the integration of ROMs into SystemC AMS is presented by Blochmann et al. [36]. The authors extend their solution to include the superposition of ROMs with fractional ordinary differential equations, thereby enabling the simulation of non-linear side-effects in [124]. As the integration of ROMs into SystemC AMS has already been addressed by various existing works, this topic will not be discussed in this thesis. For further information, please refer to existing literature on this topic. The remainder of this work will proceed under the assumption that ROMs can be integrated into SystemC AMS.

In their study, Schneider et al. examine an alternative aspect to MEMS modeling strategies [125]. The authors differentiate between three categories of modeling styles: white-, gray-, and black-box models. White-box models are easily interpretable, as their behavior is clearly and directly relatable to the parameters that have been selected. Modeling the fundamental physical relations, for example, results in the creation of a white-box model. In contrast, black-box models define input-output relations with a relatively fixed configuration, which can be challenging

to interpret. Reduced-order models are predominantly situated within this category. Therefore, a compromise between the ease of comprehension and the accuracy of the model must be reached. Gray-box models occupy a position somewhere between the aforementioned extremes. This perception of modeling styles is presented here as it will be a significant factor in the decision how the HVP is implemented to best support the use cases that will be presented in the following sections.

The thesis at hand examines two discrete use cases for HVPs in depth: the development of firmware and the creation of trim and test algorithms. In addition to the aforementioned work, the development of firmware for sensors has been the subject of particular attention by Rudolf et al. [46]. They propose a methodology for the automated generation of firmware from signal flow graphs. Thereby, the fundamental functional blocks still consist of hand-written code but they are integrated into a comprehensive firmware feature through the automated generation process. Furthermore, timing and power analyses are performed on the generated features. The approach is validated using a sensor node as an illustrative example. However, the model of the sensing element itself is exceedingly simple and lacks configurability. The difficulties associated with highly integrated firmware features are therefore not addressed.

New test or trim procedures are typically introduced in the context of their respective circuitry and therefore simulated at the circuit level only (see e.g., [126, 127]). To the best of the author's knowledge, no prior work has been presented that covers the development and simulation of trim and test procedures for sensors at the system level or within the context of virtual prototypes.

3.2 Identification of Use Cases

3.2.1 Firmware Development

Virtual prototyping is well-known as an effective method for HW/SW integration. As previously stated, the objective of this thesis is not to challenge the established workflows in this regard. Instead, this chapter explicitly focuses on the topic **firmware**, that is, low-level software running directly on the device. It considers the features of this type of software which are not well covered by the traditional digital approach. The fundamental distinction between software and firmware is that the latter has the capacity to access internal features that are not part of the host interface, which is known to the customer. These capabilities may contain the use of test modules, specific configuration registers and other features which facilitate use cases such as self-tests and self-calibration routines. These features are becoming ever more crucial to meet the ever-increasing requirements on sensor performance. It is evident that these features interact closely with the hardware. This necessitates cautious HW/FW co-design and rigorous early validation, which can be effectively achieved through the use of a virtual prototype.

Although the targeted outcome and physical concepts for these features are manifold, the majority of them adhere to the high-level procedure outlined below:

- | | | |
|--------------|---|----------------------------|
| 1. configure | } | may execute multiple times |
| 2. monitor | | |
| 3. evaluate | | |
| 4. conclude | | |

The abstract view is beneficial in identifying the commonalities between these features, which require the use of HVPs for validation. The configuration step prepares the sensor for the forthcoming firmware feature. To illustrate, the sensor may be configured to the appropriate mode,

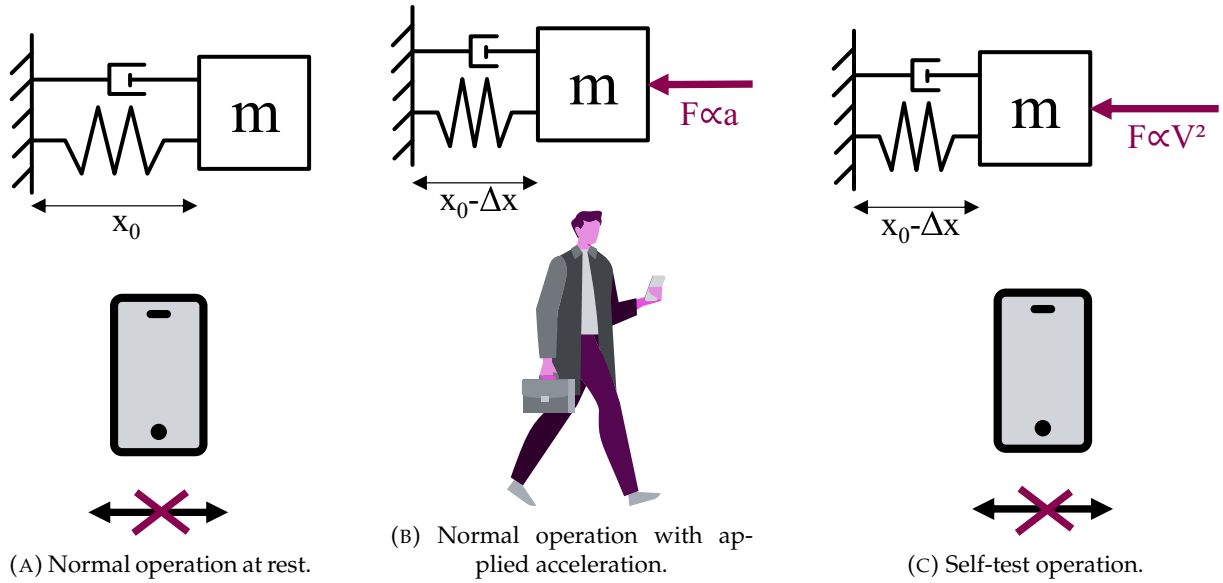


FIGURE 3.1: Working modes of a MEMS accelerometer integrated into a smartphone. In normal operation, the suspended mass is at rest when the device does not move (a). If an acceleration is applied, e.g. due to an accelerated movement of the smartphone's owner, the suspended mass is deflected due to the apparent inertial force (b). The built-in self-test emulates this behavior through an electrostatic force that acts onto the suspended mass. The effect is similar to an inertial force although the device is at rest (c).

which may be either a specific test mode or one of the standard modes (step 1). The configuration is initiated by the firmware but has an impact on the hardware, i.e., digital, and/or analog components. Subsequently, the sensor's response to the novel configuration is monitored (step 2). The incoming data are then evaluated according to the feature's target (step 3). The result of this evaluation may either define the next configuration to be applied and the aforementioned steps are repeated, or the algorithm has already gained all the information necessary and directly progresses to the final step of conclusion. The ultimate determination hinges on the specific feature under consideration. In the event of a self-test, a binary decision (pass/fail) may be taken. In the event of a self-calibration routine, the result may yield the requisite compensation value to satisfy the specified constraints.

In order to demonstrate the applicability of this abstract scheme to concrete features, the aforementioned steps are mapped to a built-in self-test (BIST) of a MEMS accelerometer in the following step. Figure 3.1 depicts the built-in self-test mode in contrast to the normal operation mode. In the given scenario, the sensor is integrated into a commercially available smartphone. In the absence of external forces, the suspended mass within the MEMS device maintains its equilibrium position (see Fig. 3.1a). If the device is then subjected to acceleration, for instance, due to the movement of a human carrying the smartphone, the acceleration creates an inherent inertial force that deflects the suspended mass, as illustrated in Fig. 3.1b. During self-test mode, depicted in Fig. 3.1c, the same deflection is achieved through an electrostatic force F_{el} , that is, the suspended mass m is once again subject to a force and measures a virtual acceleration

$$a_{virtual} = \frac{F_{el}}{m} \quad (3.1)$$

although the device is at rest. The resulting self-test procedure can thus be described by the following steps:

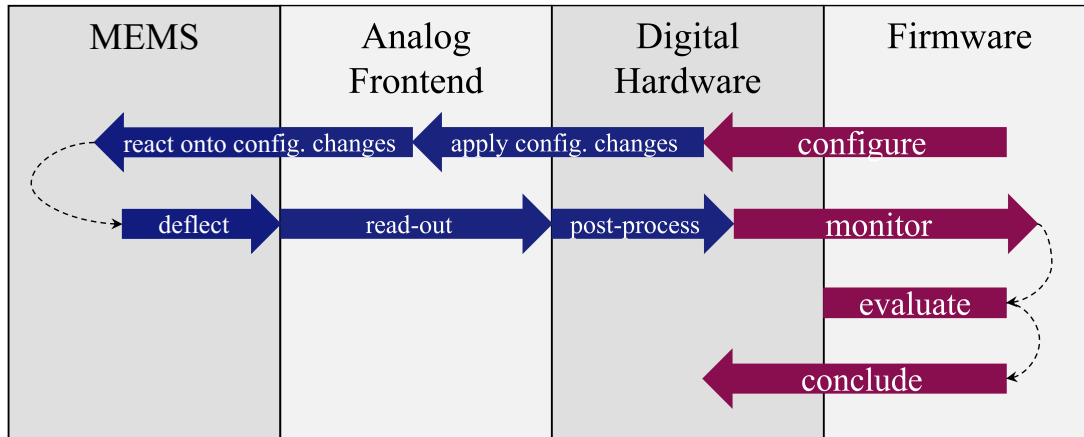


FIGURE 3.2: Self-test procedure of a MEMS accelerometer mapped onto the domains involved. A close interaction between the steering firmware and mechanical, analog and digital domain is visible.

1. **configure**: Configure the voltage level to be applied and set the sensor to self-test mode.
2. **monitor**: Collect a set of samples to measure the virtual acceleration.
3. **evaluate**: Compare the virtual acceleration to the (predefined) expected value.
4. **conclude**: Decide if the accelerometer works as expected and set the result to pass or fail.

In this scenario, a single measurement iteration is sufficient. As an additional option, both positive and negative excitation can be tested, which then requires a second configure-monitor-evaluate cycle. Although this example appears straightforward, a closer examination reveals the necessity for a heterogeneous virtual prototype to map the actions onto the involved domains. A graphical representation of this mapping is provided in Fig. 3.2. This graphical representation clearly demonstrates two crucial reasons why this specific type of firmware feature has been identified as a use case for HVPs. Firstly, the responses to the firmware actions are distributed across all domains, including the mechanical deflection of the physical sensing unit. Furthermore, and of greater significance, the procedure traverses domain boundaries in both directions. Nevertheless, if data are injected into a purely digital prototype, the sequence of events within the virtual prototype is unidirectional with regard to the MEMS and the frontend. In conclusion, the interaction between the domains is tight, necessitating complex workarounds to guarantee proper synchronization between the injected data and the VP in a purely digital context. These issues are inherently resolved when the complete system is reflected by an HVP.

In addition to the aforementioned issues, purely digital prototypes fail to adequately reflect the transient behavior of the physical and analog domains. One potential workaround to perform the BIST simulation is to implement a static offset in the event that the sensor is placed in self-test mode. As a result, the firmware could ascertain that the sensor has correctly responded to the electrostatic stimulus. As illustrated in Fig. 3.3, this approach may fail to identify potential issues in the firmware. The simulation results of a BIST with multiple excitations are presented for both an HVP with an electromechanical model and the aforementioned digital workaround. The graph demonstrates that the deflection of the sensing element and, consequently, the measured virtual acceleration does not attain its maximum amplitude for the shorter pulses. This is due to the fact that the MEMS under test exhibits a high degree of damping and the transient rise is insufficiently rapid. As a consequence, the amplitude evaluation is inaccurate, leading to a failed self-test despite the device functioning as intended. The digital workaround is unable to capture the transient behavior, thereby preventing the identification of this issue.

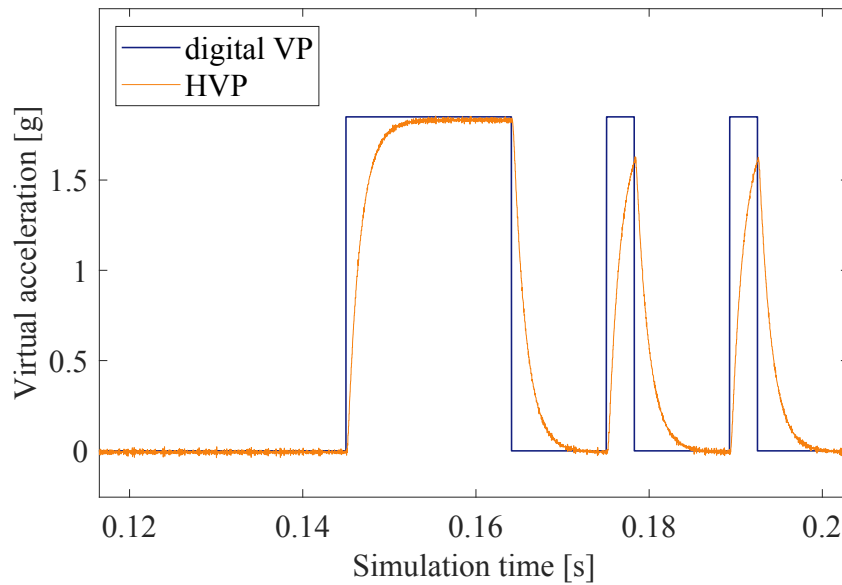


FIGURE 3.3: Simulation of a built-in self-test both with digital and heterogeneous VP. The MEMS is highly overdamped and cannot follow the excitation fast enough to reach its full deflection amplitude for short voltage pulses. The HVP reflects the transient behavior revealing the issue. The digital VP adds a static offset only.

3.2.2 Trim and Test Procedures

A newly manufactured sensor will most likely not directly meet the given specifications due to the inherent manufacturing variations that impact the behavior of the MEMS. To accommodate this, the production of MEMS sensors encompasses not only the initial manufacturing stage but also a final trimming phase. Figure 3.4 illustrates the objective of this additional trimming step. During the design phase, the MEMS and system parameters are carefully designed. Nevertheless, the effective parameters of a real device are subject to manufacturing variations and environmental effects. Consequently, a cohort of sensors encapsulates a more expansive range with respect to the parameter under examination. A trim procedure strives to align the parameter values with the design objective and mitigate the discrepancies between individual parts. Trimming is thereby realized either in the analog frontend, for instance by switching capacities on and off or varying the voltage levels, or in the digital part using arithmetic logic units to add offsets or insert gains.

Prior to the start of series production, it is necessary to develop the trim routines and utilize them for engineering samples in the laboratory. The implementation of these routines in form of code that can be executed on machines in the laboratory or in series production are called profiles. In addition to the trim profiles, test profiles are developed to characterize the sensors' behaviors. To the best of the author's knowledge, there is no existing literature addressing the simulation of these trim and test procedures at the system level.

Given the constraints of cost and time-to-market, it is not feasible to develop and debug these profiles directly on the measurement setups such as tilting tables or temperature chambers. As the throughput of a laboratory is typically limited by the availability of machines, it is of utmost importance to avoid occupying them with unsuccessful measurements that fail due to errors in the profile. Instead, the corresponding profiles are typically simulated using the ASIC toplevel model before they are actually applied in the laboratory. The term *ASIC toplevel model* hereby summarizes the set of Verilog or VHDL models which represent the digital components on

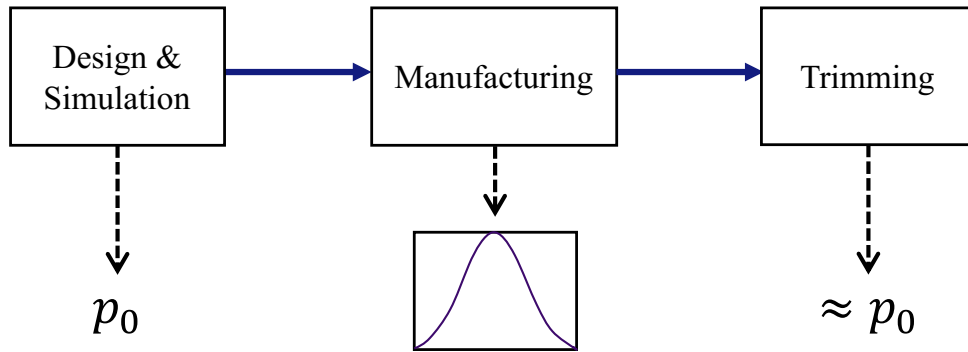


FIGURE 3.4: Graphical representation of the objective of sensor trimming. A parameter p is designed such that its value is p_0 . Due to manufacturing variations, the parameter's value differs from device to device. The subsequent trim step tries to adjust the behavior such that all sensors get close to the target behavior.

register transfer level and respective AMS models written in Verilog-A(MS). However, this approach gives rise to a number of issues that warrant further consideration. First and foremost, the extensive utilization of synthesis and layouting tools results in a short time frame between the establishment of a stable RTL model and the implementation on transistor level. Consequently, a stable toplevel model of the ASIC is typically only available shortly before tape-out. This places significant pressure on the team responsible for developing the trim and test procedures to be prepared as soon as the initial samples become available. The fast pace of the consumer market imposes the necessity to start the characterization of trimmed devices as fast as possible to reduce the time-to-market. Achieving this objective is challenging when simulations start later than anticipated. Furthermore, the simulations are typically time-consuming. The author of the thesis at hand has experienced a wait time of approximately 30 minutes solely for the purpose of completing the startup of a MEMS gyroscope, namely the time till the drive loop has settled, and retrieving the initial valid samples on the ASIC toplevel model. This is an inefficient approach for debugging purposes, where the same test case is executed multiple times to identify and resolve a bug. Furthermore, the ASIC simulation environment is considerably more intricate to handle and debug than the virtual prototype, which offers the convenient ability to step into the code or trace internal signals as required.

Moreover, relying exclusively on ASIC simulation presents another disadvantage with regard to the conceptualization process. In the initial stages of a project, the specific requirements for test features are collected and evaluated. Subsequently, the ASIC team develops implementation concepts and implements the requisite circuitry. Similarly, the trim circuitry is designed in a comparable manner. This process lacks an adequate feedback loop that would allow for the review of concepts with the relevant stakeholders. Even if a review is conducted based on a textual or graphical description, some issues may remain undetected. The use of an HVP serves to mitigate this risk. The higher level of abstraction permits the direct conveyance of concepts into a model, obviating the necessity to define all details of an implementation. Consequently, the HVP is capable of encapsulating features that have not yet been implemented. If the trim and test team initiates simulations at this preliminary stage, conceptual issues can be identified and rectified before the implementation is complete. Conversely, if a conceptual issue is identified at a later stage, it can either be addressed, albeit at the cost of a timeline extension, or left unresolved, which may necessitate ad hoc solutions for its management. These may impact various aspects, including measurement quality, test time, susceptibility to bugs due to increased complexity, and others.

In summary, HVPs can support both, the frontloading of activities and the collaboration between the different teams to achieve a proper interaction of the components. An illustrative

example is provided in the experimental results presented in section 3.5 after requirements and implementation of the model have further been discussed.

3.3 Model Requirements

In order to provide effective support for the aforementioned use cases, the HVP must fulfill a set of criteria. These have been collected during the course of the industrial case study. Some of them hold similarly for digital VPs whereas some requirements apply specifically to the AMS part.

In general, the module breakdown of the HVP should be consistent with that of the actual hardware. Each ASIC block is associated with a module target specification, which defines both the interface and the functionality of the block. The target specification is the most important source of information for the model development. Following the specification closely on the one hand ensures that the behavior of the HVP actually reflects the target behavior of the real sensor and on the other hand allows to use the specification likewise as model documentation. Nevertheless, all known discrepancies must be documented. Moreover, the interface of a SystemC/-AMS module typically provides only a subset of the ports accessible in the hardware, as some signals are omitted during abstraction. This primarily affects signals such as clocks or supply voltages. For the existing ports, it is advisable to implement a name-to-name matching between the hardware and the HVP to enhance the expressiveness.

The employment of AMS models introduces further complexities into the design of an HVP. While the digital state space is finite, and the behavior of the hardware can be defined precisely, the analog state space is infinite, and each individual part will behave slightly different, both due to manufacturing variations and external environmental factors. It is therefore essential to define a suitable modeling strategy that takes account of the inherent part-to-part variations. In order to derive this strategy, it is first necessary to identify the aspects that require validation. The development of a new feature, regardless of whether it is a firmware update, a trim, or a test procedure, necessitates the validation of multiple aspects. The overarching inquiry is whether the feature is capable of fulfilling the intended function and to what extent it exhibits the desired quality. To evaluate this, it is first necessary to ascertain whether the implementation of the feature aligns with its specifications. In the absence of such an evaluation, it is not possible to determine whether the overall concept is flawed or if the issue lies solely with the implementation. Secondly, the performance of the feature in terms of effectiveness (e.g., what is the remaining offset after trimming?), robustness (e.g., does the step counter still work if the smartphone containing the sensor is placed in a handbag?), efficiency (e.g., test time, power consumption?), and other relevant criteria must be evaluated. The aforementioned steps impose disparate requirements on the HVP. The following paragraph discusses the separation of these two steps and their related requirements, with the aim of providing an illustrative example based on the suitable modeling styles for a MEMS. Subsequently, the interference between the various use cases is addressed, as firmware features often have distinct requirements onto the HVP compared to trim and test procedures. This pertains primarily to the frontend model and can thus easily be explained in a separate step.

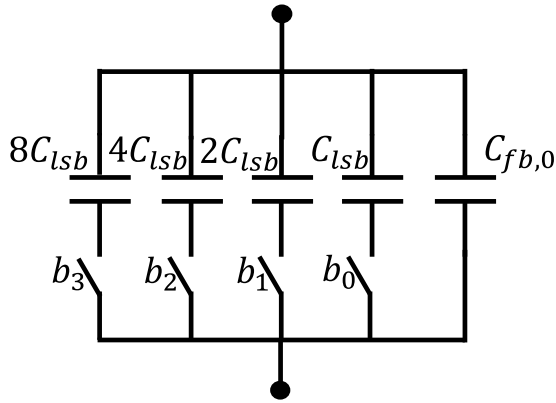
The first step deals with the concordance between the specification and the implementation. To assess this, it is of paramount importance that the simulation results are easily comprehensible and reproducible. Therefore, a white-box model is required that adheres to a known and comprehensible input-output relation. In the context of MEMS modeling, the use of a reduced

order model is thus not the optimal approach. The multitude of parasitic modes and nonidealities complicates the identification of the root cause when the simulation results fail to meet the expectations. Instead, the MEMS sensor element is represented as a simple damped harmonic oscillator for the fundamental mode, surrounded by a number of analytical formulas. These are invoked, for instance, to calculate the sum of applied forces and to translate the deflection of the movable mass into a capacitive change. This approach allows for a clear definition of the behavior of the MEMS. The parameterization is derived from the nominal values that are targeted by the design. At this juncture, it is sufficient for the HVP to qualitatively mirror the sensor's intended behavior.

Once the feature has been successfully applied to the initial test case, the second step is to ensure its quality with respect to the aforementioned aspects. It is therefore essential that the model can be manipulated in such a way that it does not perfectly reflect the nominal state. At this stage, the introduction of noise sources becomes pertinent. Moreover, the AMS model must be parameterizable in order to facilitate the implementation of multirun approaches for the assessment of a feature's robustness. However, the selection of parameters must be made with great care. The findings from the industrial project under examination indicate that parameters should be defined at the phenomenological level. This encompasses all properties that have a direct and readily assessable influence on the sensor output. This may include both physical quantities, such as the damping coefficient, which directly informs whether the system will overshoot and oscillate or slowly converge in the event of a sudden input change, and system-level properties, such as the electrical sensitivity of the MEMS, which is defined as the capacitive change per unit length of the deflection. Parameters that are less suitable for this purpose would include the gap between the positive and negative detection electrodes, as this influences the system behaviour in a number of ways (e.g. the capacitance at rest, the electrical sensitivity, the electrostatic forces), and is thus difficult to assess. In addition to being readily assessable by end users, phenomenological parameters offer the further advantage of being typically unambiguously defined and often documented within internal datasheets. This provides a reliable source of information for establishing a realistic initial parameterization. Furthermore, these quantities are typically subjected to continuous evaluation throughout the development phases, with predictions made using low-level models prior to the arrival of any hardware and measurements taken upon the arrival of the initial samples. Thus, these parameters are well suited to combine the top-down approach of virtual prototyping with a bottom-up approach that feeds back their values into the HVP similar to the approach by Barnasconi and Adhikari discussed within the related work [114].

So far, the analysis has been using the term *features* to account for various applications such as firmware, trim or test procedures. However, these use cases put distinct constraints onto the HVP. Firmware features are mainly designed for the in-field usage of a sensor. Thus, they rely on the standard functionalities of the hardware. In contrast to this, the test procedures that are designed for sensor characterization often make use of special test modes. Hence, these must explicitly be added to the HVP extending its compliance with the real hardware but likewise the modeling effort.

A similar situation holds for the trim circuitry. As the firmware is usually applied in the field, hence especially after final trim, the model is expected to behave like a trimmed sensor. This can easily be done reflecting the nominal behavior only and simply neglecting the existence of any trim circuitry. Contrarily, the proper evaluation of trim procedures obviously requires to provide their functionality. System characteristics which can be trimmed, for instance the capacitive offset of the micro-electromechanical part, become model parameters such that they can be directly set by the user. As a result, it is possible to analytically calculate the expected trim word and compare it to the results of the trimming simulation.



$$C_{fb} = C_{fb,0} + k_{fb}C_{lsb}$$

$C_{fb,0}$: default feedback capacitance,
limits the maximum gain

k_{fb} : 4-bit trim word ($b_3b_2b_1b_0$)

C_{lsb} : capacitance added for the
least significant bit of k_{fb}

FIGURE 3.5: Trim circuit in form of a binary ladder structure. Each bit of the trim word controls one switch of the ladder. The circuit is utilized to trim the feedback capacitance of a charge-to-voltage converter which directly affects the analog gain.

In summary, enabling this use case requires to decrease the level of abstraction slightly. Nevertheless, the HVP remains at the analog module level and does not require to switch to the analog module *implementation* level. The following example illustrates this fact. A CV converter of a sensor frontend is often implemented as switched-capacitor circuit. However, its nominal behavior is merely a linear input-output relation with a constant conversion factor from input charge q_{MEMS} to output voltage V_{out} . This conversion factor is usually determined by a feedback capacitor C_{fb} such that the behavior of the circuit can be described in an abstract manner following the equation

$$V_{out} = LPF \left\{ \frac{q_{MEMS}}{C_{fb}} \right\}. \quad (3.2)$$

Thereby, $LPF\{\cdot\}$ describes a low-pass filtering to account for the circuit's bandwidth limitation. Implementing this formula with a proper value for C_{fb} is already sufficient to create an HVP that is useful for firmware development. However, the feedback capacitance is used to trim the analog sensitivity, defined by the output voltage per Coulomb input charge. It is typically realized as binary ladder structure as illustrated in Fig. 3.5. The binary structure of the circuit allows the formula presented in Fig. 3.5 to effectively describe its behavior in a straightforward manner, eliminating the need for individual reflection of each switch. Therefore, the HVP maintains a reasonable level of abstraction to ensure both speed and comprehensibility. Moreover, it is still possible to develop the HVP based on the fundamental operational principle of each block, even before an actual implementation is available.

As previously stated, the firmware typically anticipates the sensor to be trimmed. Therefore, the enhancement of the HVP towards trimming functionality initially appears to be in opposition to the requirements of the firmware developers. In order to guarantee an adequate simulation environment for these purposes, it is essential to provide a default trim. When nominal values are used for parameterization, the result is typically the setting of trim words to the midpoint of their range. This is due to the fact that the system is designed with a maximum margin for deviations in both directions. At the beginning of the simulation, the default trim settings, if desired, are loaded into the non-volatile memory (NVM).

3.4 Implementation

In order to support the development of the IMU, a model of the MEMS sensor element and the frontend for both the accelerometer and the gyroscope have been constructed. Each sensor is reflected by a single multi-rate TDF cluster. For a comprehensive overview of the AMS models utilized in the HVP, please refer to Fig. 3.6. The model has been simplified somewhat due to proprietary considerations, but it accurately reflects the fundamental architecture and the setup with regard to parameterization and register connection. The frontend incorporates a variety of circuitry to accommodate the inherent manufacturing variations of the MEMS. To accurately represent this behavior, the relevant MEMS parameters are made accessible and configurable for users to inject these variations. The frontend model is linked to a set of registers that control the trim circuitry, enabling users to compensate for the injected deviations.

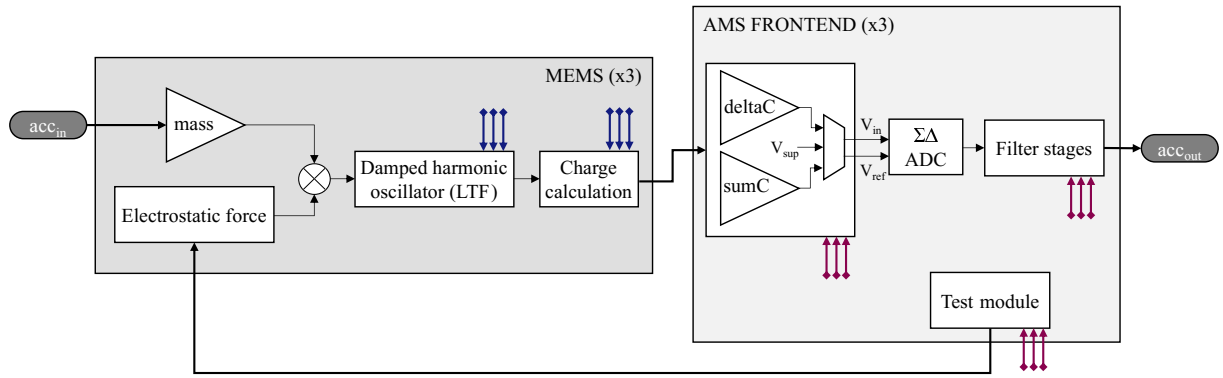
Table 3.1 provides a summary of the key data for the AMS models. The cluster for the accelerometer comprises nine modules per axis, in addition to two test modules, resulting in a total of 29 modules for the three-axis sensor. The gyroscope model is composed of 14 modules per sense axis, 15 modules for the drive loop, and two test modules, amounting to a total of 59 modules for the three-axis gyroscope.

It is also noteworthy that only a subset of the internal parameters are accessible within the top-level model. These are visible and configurable for users of the model. The remaining parameters are included solely for the purpose of increasing the reusability of the modules for subsequent projects; however, they are fixed for each individual project by the design. The set of accessed registers is a combination of mode configurations and trim registers. A single register can be split into multiple segments of bits that are semantically related. These segments are further referred to as fields and act like model parameters which can dynamically change during the simulation. Despite the gyroscope model being significantly larger and more complex than the accelerometer model, the latter has a greater number of parameters and accessible register fields. This is simply due to the gyroscope model only supporting a subset of the trimming functionality and focusing more on features used by firmware development. The accelerometer has been primarily used to evaluate the effectiveness of HVPs for trim and test procedures.

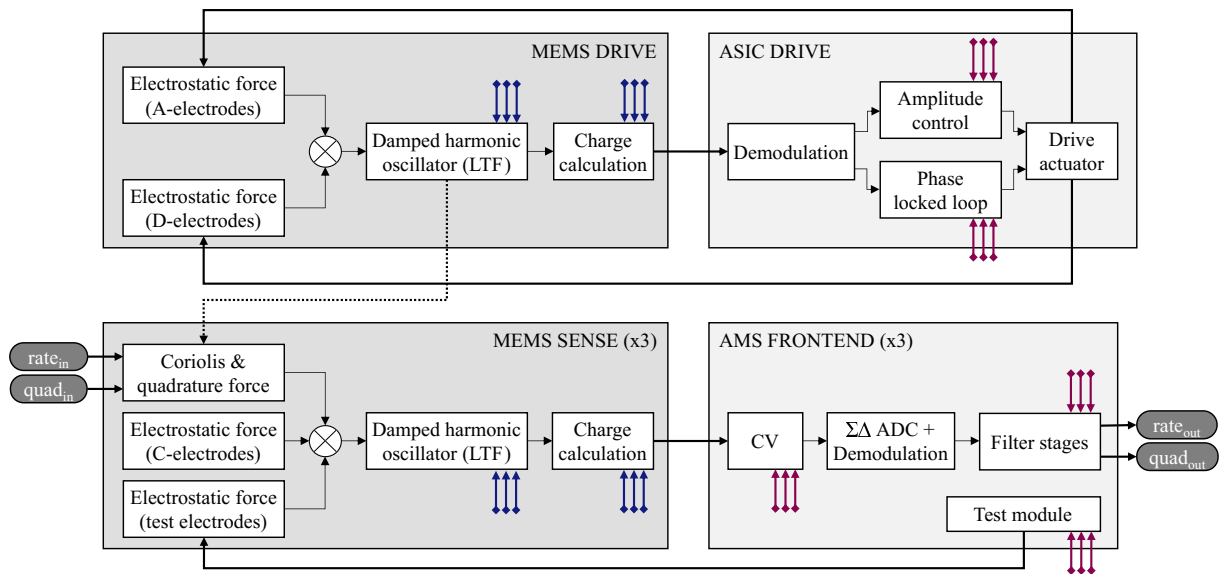
It can be observed that the model incorporates a few digital blocks, as the filter stages subsequent to the ADC are still written in SystemC AMS. Chapter 4 will provide a detailed comparison of the simulation semantics of the TDF MoC with that of SystemC. It will demonstrate that SystemC AMS offers a performance advantage when a module is subject to regular triggering, thereby enabling it to become part of the static schedule. In the aforementioned model, the ADC blocks operate at their designated frequencies, as do the subsequent filter stages. Therefore, they are still written in the SystemC AMS modeling language.

The implementation was conducted in two principal stages. First, the nominal functionality of the sensors was modeled in order to obtain a complete event chain as rapidly as possible, thereby enabling the firmware developers to commence utilization of the HVP. Subsequently, once the trim and test features had been defined and the HVP's potential of supporting the respective engineers had been identified, these were incorporated into the model.

The digital part of the VP introduces an additional 53 modules at the toplevel of the virtual prototype. These have mostly not been developed by the author of the thesis at hand and are not discussed further here. A testbench module is employed to simulate the host interface. Test scripts are created in Python by users for the purpose of controlling the simulation. The interaction with the host is specified via an API (application programming interface), whose functions are interpreted by the HVP through the use of a Python/C++ bridge. The fundamental operations include reading and writing registers, injection of inertial stimuli, and wait statements.



(A) TDF model of the accelerometer.



(B) TDF model of the gyroscope.

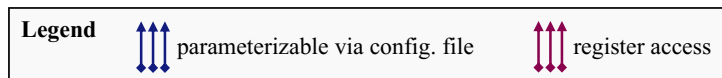


FIGURE 3.6: Overview of the architecture of the SystemC AMS model for the HVP of an IMU. The MEMS parameters are adjustable via a configuration file that is read in before simulation. The frontends can be configured and trimmed through corresponding registers. The outputs of the blocks shown are connected to digital blocks written in SystemC.

TABLE 3.1: Key characteristics of the SystemC AMS TDF models realizing the AMS part of an HVP of an industrial IMU.

Component	Modules	Parameters		Accessed register fields
		Internal	Top-level	
Accelerometer	29	96	21	36
Gyroscope	59	102	16	22

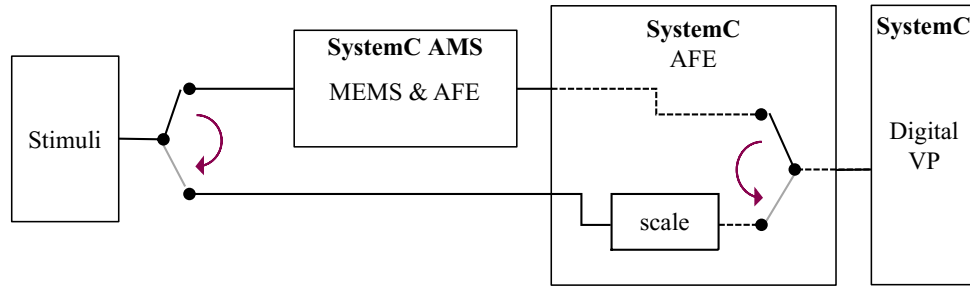


FIGURE 3.7: Integration of the SystemC AMS model into the virtual platform. It must be explicitly enabled. Otherwise, it is skipped and a dummy frontend replaces its functionality.

The virtual platform is designed in such a way that the AMS model can be included or excluded, depending on the necessity of the use case. For instance, if the objective is to test the functionality of the FIFO, incorporating an AMS model is not advantageous. Conversely, the validation process is most straightforward when the data intended for the FIFO can be directly injected into the digital signal processing (DSP) chain. In such a scenario, the AMS model can be omitted to simplify the validation of the test and increase the simulation performance.

Figure 3.7 illustrates the setup. The inertial stimuli are provided by a driver module that has the capacity to either read in a stimulus file or create the stimulus based on a predefined function. A command line option enables the AMS model, either the accelerometer, the gyroscope, or both. If for example the gyroscope is enabled, its toplevel module is instantiated during the elaboration phase. Otherwise, it is bypassed. To guarantee the consistency of the data in either scenario, a dummy analog frontend (AFE) is incorporated into the VP. In the absence of an AMS model, the stimuli are directly sourced from the driving module, scaled in accordance with the anticipated nominal output of the AFE, and conveyed to the subsequent module in the event chain. Therefore, the stimuli are interpreted as the acceleration or rotational rate that is **measured by the frontend**.

Otherwise, the stimuli are fed to the AMS model, which utilizes them as input for the MEMS sensing element. The SystemC AFE module merely traverses the outputs of the AMS model in their unaltered state. Nevertheless, the dummy frontend is responsible for handling certain events and triggering the subsequent module. Consequently, the process is not entirely bypassed. It should be noted that the meaning of the stimulus is altered when the AMS model is incorporated. In this case, the SystemC AFE module defines the **physical stimulus that the sensor is subject to**, which may deviate slightly from the values measured by the frontend.

The impact of the AMS models on the simulation performance has been assessed based on real-world examples. A regression test suite for firmware features has been utilized for this purpose. It has been developed for the high performance IMU project that is taken as industrial case study within this work. The firmware developers were emboldened to use the AMS models, i.e. the HVP, only if it has a clear benefit for their simulations. Otherwise, the purely digital VP was used. Therefore, test cases have usually not been executed both with and without AMS model. Although no systematic performance test has been conducted on this large-scale case study, the existing tests are sufficient to identify trends and ascertain the span of simulation speeds that is achieved by various tests. Usually, these tests are conducted on a server with parallel execution enabled. In order to obtain the simulation times in terms of wall-clock times, the tests were rerun in a sequential manner on a local computer. The results are depicted in Fig. 3.8. The performance factor is thereby defined as the ratio between the simulated time and the elapsed wall-clock time. A higher performance factor is consequently beneficial. Within the digital domain, the processor model has the strongest impact on the simulation speed. It employs clock gating to fasten the simulation when the processor is not active. As a result, the

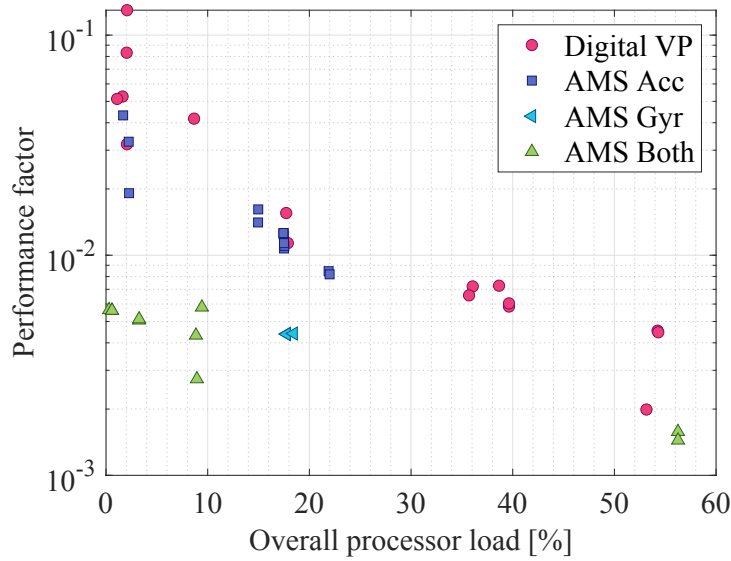


FIGURE 3.8: Performance of the presented HVP with and without enabling the SystemC AMS models for accelerometer and gyroscope. The performance factor describes the ratio between simulated time and elapsed wall-clock time.

simulation performance is highly dependent on the overall processor load. To account for that and allow a fair comparison between purely digital and heterogeneous simulations, the data is plotted against this load. It is visible that the performance factor decreases with increasing processor load. The effect of enabling the AMS model of the accelerometer is comparably small. The performance factors keep close to digital test cases with similar processor load. The gyroscope model is significantly larger and operates at a higher frequency, resulting in a discernible impact. For low processor loads, its relative contribution to the overall simulation time is large and the performance factor is decreased by roughly one order of magnitude. If the computational cost of the digital VP increases because the simulation requires to calculate many processor cycles, the relative contribution of the AMS model shrinks. All in all, the performance factor keeps within an acceptable range.

3.5 Experimental Results

To demonstrate the efficacy of trim simulation at the system level, an illustrative trim step is presented in the following section. In Chapter 2.1, the charge-balanced accelerometer frontend was introduced, which generates an electrical readout proportional to $\frac{C_p - C_n}{C_p + C_n}$, as this ratio is linear with respect to the acceleration. In order to achieve this, two charge-to-voltage converters are utilized. One of these delivers an output voltage that is proportional to the capacitive difference ($V_{\text{delta}C}$), while the other delivers an output voltage that is proportional to the sum of both capacitances ($V_{\text{sum}C}$). The subsequent ADC employs $V_{\text{delta}C}$ as the input voltage and $V_{\text{sum}C}$ as the reference voltage. The circuit is designed in such a way that the gain of each CV converter can be adjusted. The sum of the capacitances depends on both their initial values and their sensitivity to deflections. In order to ensure that the correct gain is maintained despite manufacturing variations, the algorithm under test targets to trim the gain of the CV converter responsible for $V_{\text{sum}C}$.

In order to decouple the effect from the differential CV stage, the sensor is set to a special test mode that routes $V_{\text{sum}C}$ to the input of the ADC and uses a constant supply voltage as reference voltage. Consequently, the output of the ADC becomes proportional to the output of the CV

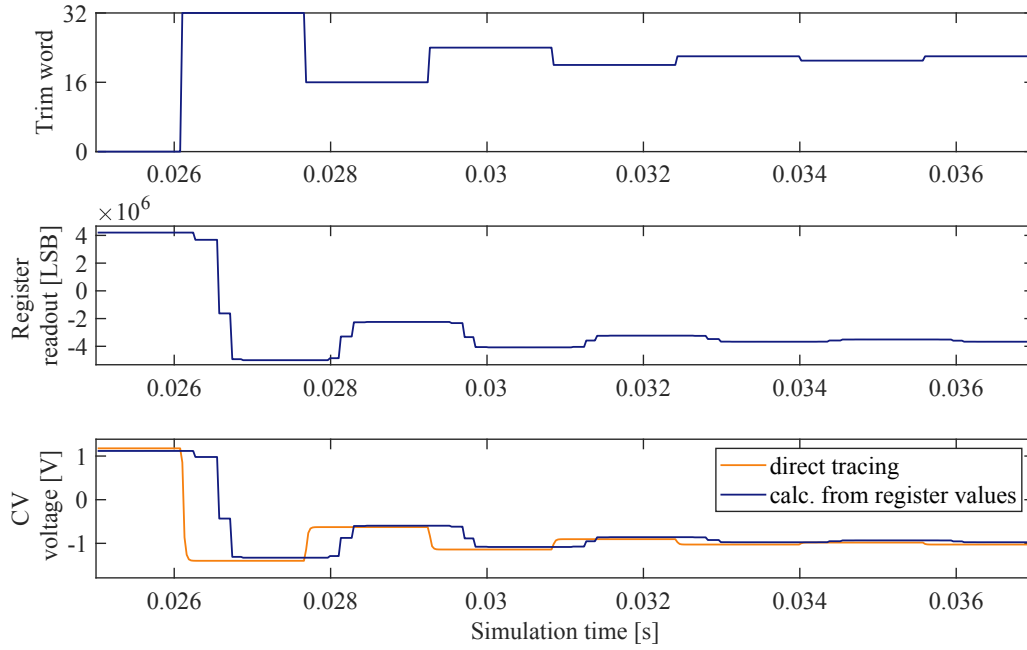


FIGURE 3.9: Simulation results of a binary search trim procedure. The voltage of the CV converter is calculated from the register values because the hardware does not directly provide to measure it. The HVP still allows to trace it directly for validation.

converter under test. The optimal trim word is identified through a binary search algorithm. Figure 3.9 illustrates the simulation results of this procedure. The initial row represents the trim word k , which has a bit width of six, thereby defining a trim range of $[0, 63]$. Prior to trimming, the value is set to its default of zero. Subsequently, the binary search commences at the midpoint of the interval ($k = 32$) and gradually converges upon the optimal step. The second row depicts the readout of the corresponding data register, which is utilized to assess the present output voltage of the CV converter. The bottom row depicts the signal under examination, namely the voltage V_{sumC} . The trim target is -1 V. The blue line represents the voltage level assumed by the trim algorithm. The calculation is derived from the register values and the corresponding scaling factors of the signal chain, such as filter gains. The orange graph depicts the actual output of the CV converter, which has been obtained through the tracing of the corresponding internal signal. It should be noted that access to this signal is not possible in the respective hardware. However, it is accessible in the HVP and eases the validation of the trimming procedure through simulation. Furthermore, it demonstrates the cumulative delays introduced by the ADC, the active filters, and the ODR configuration between the CV converter and the data registers. It should be noted that the data registers utilized are not part of the system depicted in Fig. 3.6. Rather, they are situated within the digital part of the VP. Figure 3.10 depicts an excerpt of the output log generated by the HVP during simulation. The testbench file for the trimming procedure creates a set of log messages to enable rapid evaluation of the simulation.

3.6 Impact on the Development Flow

The following section will evaluate the experiences derived from the industrial case study. These are then used to depict the overall development flow and highlight the impact of the HVP

```

[ 5932475 ns ] MSG: *****
[ 5932475 ns ] MSG: Accel trim procedure starting.
[ 5932475 ns ] MSG: *****

[ 27665975 ns ] MSG: int data_ch1 [lsb] for 1. trim_word (32): -5000891
[ 27665975 ns ] MSG: SumC measured voltage for 1. trim_word (32): 1.3759101792923893
[ 27665975 ns ] MSG: new diffch1 = 0.37591017929238935
[ 27665975 ns ] MSG: next word ch1 = 32 - 16

[ 29245175 ns ] MSG: int data_ch1 [lsb] for 2. trim_word (16): -2243102
[ 29245175 ns ] MSG: SumC measured voltage for 2. trim_word (16): 0.6171513986189895
[ 29245175 ns ] MSG: next word ch1 = 16 + 8

[ 30824375 ns ] MSG: int data_ch1 [lsb] for 3. trim_word (24): -4076336
[ 30824375 ns ] MSG: SumC measured voltage for 3. trim_word (24): 1.1215345818607165
[ 30824375 ns ] MSG: new diffch1 = 0.12153458186071653
[ 30824375 ns ] MSG: next word ch1 = 24 - 4

[ 32403575 ns ] MSG: int data_ch1 [lsb] for 4. trim_word (20): -3230354
[ 32403575 ns ] MSG: SumC measured voltage for 4. trim_word (20): 0.8887770102003596
[ 32403575 ns ] MSG: new diffch1 = 0.1112229897996404
[ 32403575 ns ] MSG: next word ch1 = 20 + 2

[ 33982775 ns ] MSG: int data_ch1 [lsb] for 5. trim_word (22): -3669359
[ 33982775 ns ] MSG: SumC measured voltage for 5. trim_word (22): 1.009561776007144
[ 33982775 ns ] MSG: new diffch1 = 0.00956177600714403
[ 33982775 ns ] MSG: next word ch1 = 22 - 1

[ 35561975 ns ] MSG: int data_ch1 [lsb] for 6. trim_word (21): -3508090
[ 35561975 ns ] MSG: SumC measured voltage for 6. trim_word (21): 0.9651913510760058
[ 35561975 ns ] MSG: final trim word ch1 = 22
[ 37141175 ns ] MSG: SumC measured voltage for optimal trim word: -1.0095207811890776
[ 37141175 ns ] MSG: *****
[ 37141175 ns ] MSG: Accel trim procedure finished.
[ 37141175 ns ] MSG: *****

[ 37141175 ns ] MSG: Simulation halted!

```

FIGURE 3.10: Terminal output of the trimming simulation. The script specifies a set of log messages that support the fast evaluation of results.

on it. The case study commenced with a firmware-centric modeling approach that exclusively reflected the nominal behavior of the system. A training session was conducted to introduce the new features of the VP (now HVP) and their user interface. This included information on how to enable the AMS model and how the semantics of the stimuli change. The model was subject to continuous improvements, including refinement, bug fixes, and additional features. In response to a feature request by the firmware team, the ability to trace data in a tabular format with a defined data rate was added to the HVP. At that time, only VCD (value change dump) tracing was implemented. The tabular format allowed the team to load data easily into MATLAB and/or Python for algorithmic development.

User feedback was regularly collected to track the efficacy of the HVP. One response highlighted the benefit to **flexibly generate transient data** with different configurations and parameterizations on their own. This accelerated the algorithmic development. Prior to this, data had been provided by ASIC designers from their respective simulations, resulting in a slow feedback loop. This involved requesting specific data, another party simulating and providing it, then evaluating the new test case. Frequently, a further set of data was requested afterwards. Moreover, feedback indicated that the AMS model **eased the transfer of test cases to the RTL environment**, as the model's overall architecture became more closely aligned with the actual hardware and hence with the ASIC top-level model. A similar HVP has already been requested to support the development of a first test chip of another project.

Following the incorporation of initial functionalities pertaining to trim and test into the HVP, a supplementary training session was conducted for the relevant team to facilitate their comprehension of these new capabilities. In order to derive some statistics about their usage behavior, a log file was created that tracked some information on each run. The log file was stored in a common folder, accessible to all instances of the HVP, regardless of user or hardware configuration. Firmware developers used a different setup such that the results are specifically focused on the trim and test team. The evaluation of these data revealed that roughly 1600 simulation runs have been conducted which enabled at least one of the two AMS models.

Thereby, a broad variety of test cases was simulated. A set of *standard characterization procedures* was defined within the project. At least half of them were identified as having been simulated on the HVP. This was determined through a process of parsing the names of the testscripts. It should be noted that the selected level of abstraction is capable of supporting 90% of the standard procedures. The remaining ten percent pertain to voltage measurements with external multimeters. The discrepancy between simulatable test cases and those that were actually simulated can be attributed to the delayed availability of the requisite features within the heterogeneous VP. This, in turn, was a consequence of the delayed specification of the necessary features within the pilot project. It is likely that the set of simulated procedures will be larger in a consecutive project where the HVP and its use cases are taken into consideration from the very beginning. According to the feedback from the respective team, the AMS model was denoted as "must have". It was particularly highlighted that the HVP allowed **rapid prototyping of algorithms even after the ASIC simulation environment became stable**. Thanks to an in-house developed tool that allows to automatically transfer the Python testscripts used on the HVP to the specific language required to control the measurement setups, the profiles could efficiently be transferred to the physical setups.

Figure 3.11 illustrates the novel development cycles applicable to firmware and trim and test profiles. The initial phase of the development process entails the creation and debugging of features on the HVP. The number of iterations required is contingent upon the complexity of the feature in question. The high simulation performance allows for the completion of these debug iterations in a relatively short period of time. Furthermore, the HVP is designated as a white box, signifying that it functions as a white-box model, thereby facilitating comprehensive system insights. This contributes to the effective debugging process. The simulation on RTL is not entirely superseded by the proposed methodology; rather, it constitutes a secondary verification phase. Once the feature has been successfully executed on the HVP, it is then transferred to the RTL environment for final verification. In the optimal scenario, the process is executed only once in the ASIC simulation setup. The simulation is designed to anticipate and address any potential issues arising from the abstraction made by the HVP. Nevertheless, it is probable that the number of issues identified on RTL is significantly reduced when the feature has been meticulously prepared on the HVP. This is especially important for the verification of firmware features that become part of the read-only memory (ROM) as these must be freezed till ASIC tape-out.

Figure 3.12 depicts a comprehensive overview of the development stages, taking into account the involvement of diverse working packages and groups. The project commences with a design space exploration, which can be assisted by the VP. Given that the support is mainly provided at the architectural level, the potential for an AMS model is constrained in this context. Consequently, the focus is placed on a highly abstract digital model at this stage. Once the partitioning and block-level specification begins, the development of the VP is intensified and the creation of the AMS model starts. Ideally, the HVP is already feature complete when the detailed implementation of the ASIC starts. However, the periods of specification and design usually overlap such that some features are added later. Nevertheless, the implementation and

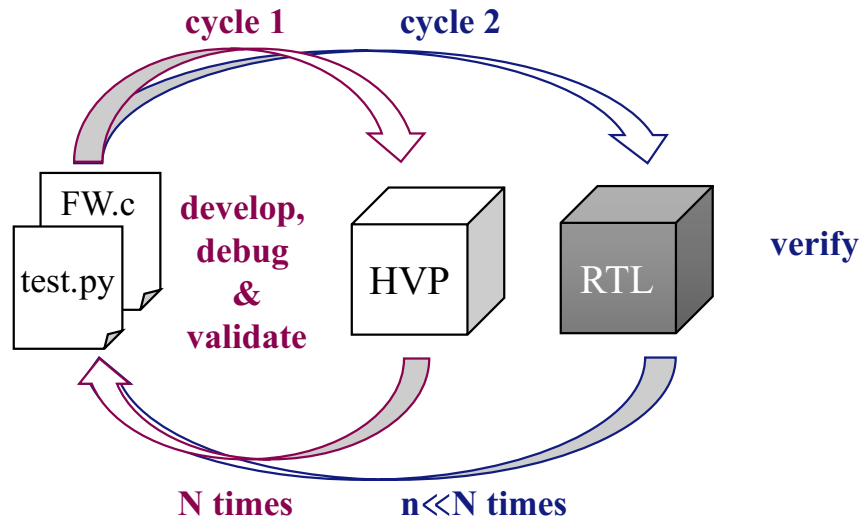


FIGURE 3.11: Development concept for firmware features and trim and test procedures. First, a fast debug cycle based on the HVP is used to develop the feature. When it runs successfully, it is transferred to the RTL model for verification.

debugging of firmware features may start in advance, as not all features of the sensor product are utilized in this phase. As illustrated in Fig. 3.11, the two-step approach facilitates the completion of the verification of firmware put into ROM until ASIC tape-out, as previously indicated. The firmware content for the random-access memory (RAM) is subject to less stringent timing constraints but nevertheless benefits from the enhanced debugging capabilities afforded by the HVP. The optimal deadline for the preparation of trim and test profiles is the arrival of samples. Even if trim and test circuitry is incorporated into the HVP in a subsequent phase after the fundamental functionality has been provided, the HVP can still be utilized for the development of trim and test profiles before the ASIC top-level model is available. The respective frontloading increases the probability that laboratory operations can start immediately following the arrival of samples.

3.7 Discussion

The chapter has demonstrated that heterogeneous virtual prototypes can effectively support the development of smart sensors. Although a digital VP is sufficient for digital-centric hardware/software co-design and can especially support the development of application software, the heterogeneous extension extends the scope towards firmware that closely interacts with the sensor element. Furthermore, the HVP facilitates the creation of entirely novel use cases that extend beyond the conventional scope of hardware and software development. The simulation of sensor trimming on the HVP has been successfully performed. The respective scripts are reused to program the measurement setups in the laboratory through automated translation into the machine-specific language. Furthermore, it has been demonstrated that 90% of the standard characterization procedures are suitable for evaluation on the HVP.

Given that firmware typically interacts with trimmed devices, it may be advantageous to link the two applications, namely, to simulate the trimming process first and then execute the firmware subsequently. Modifying the model's parameters allows the generation of a set of unique samples, each exhibiting a distinct trim outcome. The register configuration of the NVM can be saved at the conclusion of the simulation, resulting in the creation of a tuple comprising a parameterization and a corresponding trim configuration. Afterwards, the NVM can

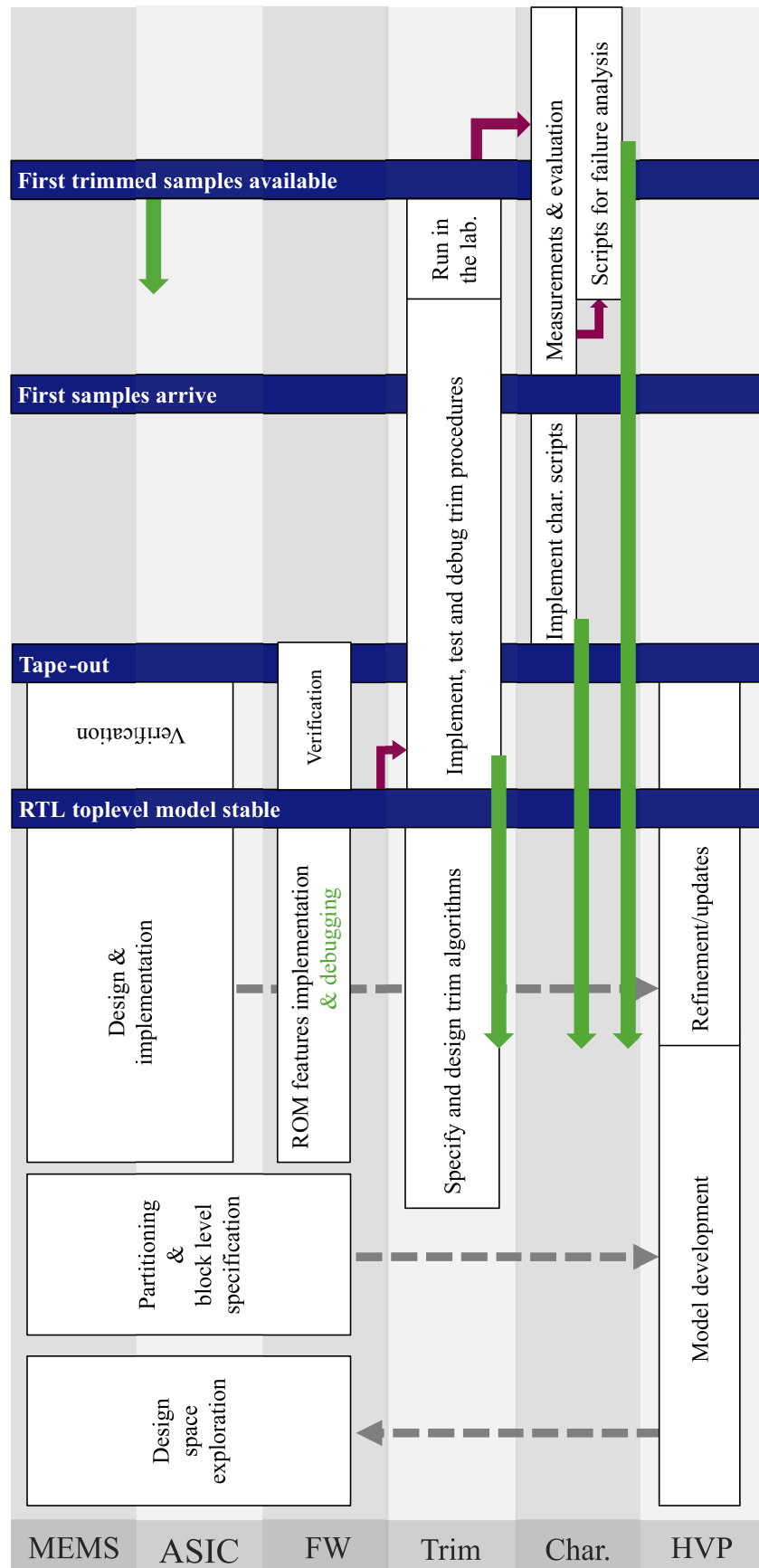


FIGURE 3.12: Schematic timeline of a sensor development separated into MEMS, ASIC and FW development, trim and characterization development for the laboratory, as well as the creation of the HVP. Red arrows indicate dependencies that are reduced by the HVP, green ones reflect timeline shifts induced by its usage.

be loaded by the HVP, such that the tuple can be transferred to the firmware team for subsequent simulations. In conclusion, this could create a *virtual lot* where each virtual device is fully specified by its parameterization file and the NVM. This could then enable further use cases, such as the preparation of characterization dashboards that illustrate the distribution of some given sensor characteristics.

In the context of the industrial case study, the set of procedures that were ultimately developed on the HVP was constrained by the delayed availability of certain features. Furthermore, it was demonstrated that the number of simulation runs can become exceedingly large if algorithms are debugged in a step-by-step manner. Therefore, even a modest improvement in simulation performance can result in a notable reduction in development time. In conclusion, the chapter encourages the continued advancement of AMS modeling techniques for HVPs, as discussed in the subsequent chapters.

Chapter 4

Choosing a Suitable Language for Analog/Mixed-Signal Modeling

The first step towards efficient heterogeneous modeling on system level is the choice of a suitable simulation language. As SystemC is the de-facto standard for virtual prototyping, the digital part of an HVPs is defined to be written in SystemC throughout this work. The previous chapter has already pointed out that SystemC AMS has been used as AMS language for the present thesis. This chapter aims to provide some information on the underlying reasons and pave the way to address *RQ2-Performance*. After introducing some related work, section 4.2 discusses some decision criteria. Afterwards, section 4.3 provides a comparison of the simulation performances of real number modeling in SystemC and SystemC AMS to classify the performances of these options.

4.1 Related Work

The question of which simulation languages are most suitable for the virtual prototyping of heterogeneous systems has been widely discussed in academic literature. Hervé and Legendre provide an overview of existing languages in this context [128]. VHDL, Verilog, SystemC, SystemVerilog, and SpecC are identified as digital languages but are deemed unsuitable for standalone modeling of mixed-signal systems. While SPICE is capable of conservative modeling of electronics, it lacks the functionality to support mixed-signal systems or other disciplines. In such cases, equivalent circuits must be derived. Furthermore, the high level of accuracy inherent to these simulations results in simulations being time-consuming. A more reasonable approach would be to utilize Verilog-AMS and VHDL-AMS. These languages are classified as AMS multi-field modeling languages [128]. SystemC AMS can be considered to fall into the same category. Both Verilog-AMS and VHDL-AMS are conservative modeling languages. The primary limitation of these approaches is their simulation performance. In a comparative analysis of analog modeling approaches, Georgouloupoulos and Hatzopoulos evaluate the simulation performance of various modeling languages [129]. As anticipated, the low-level modeling in SPICE is the slowest. Verilog-AMS and VHDL-AMS exhibit comparable performance. Furthermore, the authors examine the potential of real number modeling (RNM) and its capacity to facilitate high-speed simulations. While purely digital simulation is the fastest in the experiments, it is also the least accurate. To increase the accuracy while still remaining fast simulation, RNM describes a value-continuous event-based modeling style. As originally conceived in SystemVerilog, which has been extended with real number types for that purpose [130], the utility of this approach has been demonstrated in a variety of contexts. Phase-locked loops serve as a classic illustration of this concept [131]. The introduction of real-valued wires in Verilog-AMS (data type designated as "wreal") enables a comparable modeling approach within this language [132]. As SystemC is compatible with all C++ data types, it is able to support RNM as a native feature.

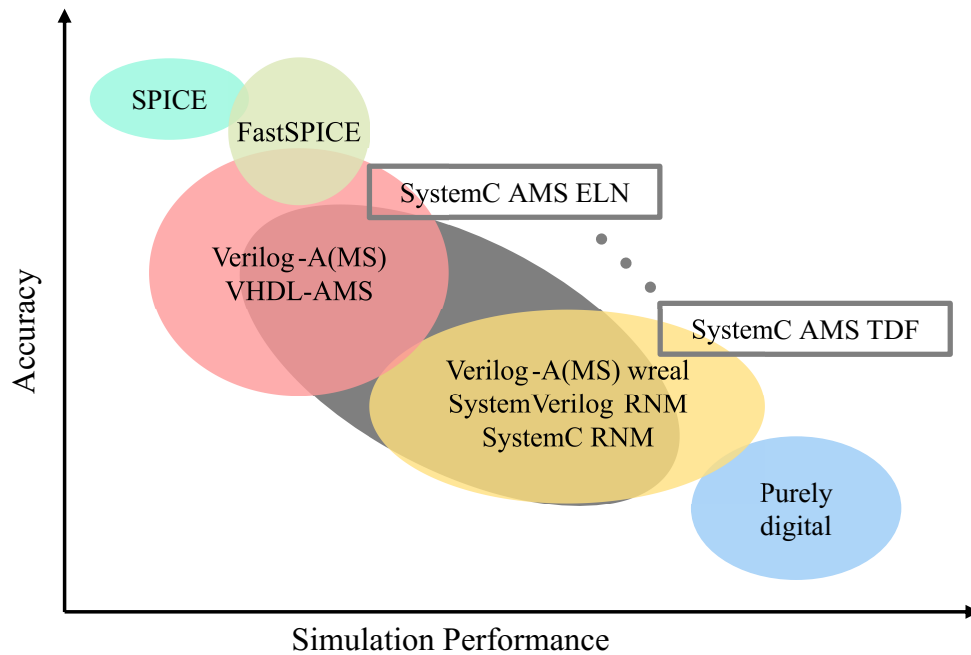


FIGURE 4.1: Overview of existing modeling languages according to their accuracy and simulation performance. Graphic based on [129, 130, 133].

Figure 4.1 illustrates the trade-off between accuracy and simulation performance for the languages mentioned (based on [129, 130, 133]). The graph additionally tries to classify SystemC AMS with its various MoCs according to its characteristics, the literature as presented below and the author’s own experiences.

In a series of experiments, Narayanan et al. compared the simulation speed of HSPICE with that of three AMS languages: Verilog-AMS, VHDL-AMS, and SystemC AMS [134]. In their experiments, the SystemC AMS model exhibits a similar execution time to that of the HSPICE model. However, the publication lacks sufficient details regarding the specific modeling techniques employed, the MoC utilized, and the selected timestep. Therefore, the results must be interpreted with caution. In a contradictory study published in 2020, Scherr and Einwich presented opposing results [135]. Benchmarking of RNM in SystemC and SystemVerilog was conducted in comparison to SystemC AMS ELN models, with the example of third-order low-pass filters. The RNM approaches were observed to run faster than the modeling of ELN networks, with SystemC exhibiting a slight superiority over SystemVerilog. Nevertheless, the observed performance loss of the ELN model was less than a factor of two compared to the RNM approaches in this particular case. The ELN MoC employs the use of ordinary differential equation solvers. The same holds for the LSF MoC such that Scherr and Einwich expect similar results for LSF models [135]. In contrast, the TDF MoC is based on purely analytical descriptions. A comparison of this MoC to the RNM modeling style has been presented by the author of the thesis in question in [P3] and will be discussed in this chapter.

One additional option for system-level modeling should be mentioned. MATLAB/Simulink is widely used for modeling because its graphical support and large module libraries ease rapid prototyping and collaboration. Pan et al. compare the capabilities of MATLAB/Simulink and SystemC using an automotive use case [136]. Their conclusion is that the simulation performance is comparable, but the languages are still not interchangeable. MATLAB/Simulink has its strength in graphical representation and is thus well suited for initial exploration testing and rapid prototyping. Integration tests are preferably performed on SystemC because it facilitates the joint simulation of hardware and software.

4.2 Choice of Simulation Language

Based on the target application of HVPs and the given constraints, a set of requirements has been derived that an AMS modeling language should fulfill. These are discussed below:

Interoperability with SystemC Since the implementation of the digital part of the VP is outside the scope of this discussion, the AMS language should operate well in combination with SystemC. A model that runs very fast standalone, but introduces a huge overhead when coupled to the SystemC simulation kernel, cannot result in an efficient overall HVP.

Level of Abstraction An HVP is a highly abstract but fully functional system-level model. The high level of abstraction is important to meet both the requirements of high simulation performance and early availability. It also allows the use of the HVP for design space exploration and keeps the modeling effort affordable. The simulation language must therefore support a suitable level of abstraction.

Simulation Performance The level of abstraction and simulation performance are closely related, since every simulation model is subject to the inherent trade-off between accuracy and speed. Nevertheless, both aspects are mentioned because simulation performance also depends on the efficiency of the underlying solvers. Especially for HVPs, which are intended to be used as tools for efficient debugging and are executed many times, the simulation performance is of utmost importance to enable efficient development.

Extensibility The variety of heterogeneous systems is endless. Specific applications are likely to have specific requirements. It is therefore advantageous to choose a language that is extensible and can be adapted to the specific needs of the heterogeneous system under test.

Based on these requirements, SystemC AMS has proven to be well suited for the development of HVPs for several reasons: Interoperability with SystemC is native by design. SystemC AMS provides a predefined interface to and from SystemC. This avoids the overhead of simulator coupling. However, the impact of the synchronization scheme between the discrete event and analog domains must be evaluated. The existence of three different MoCs helps to match the desired level of abstraction at different project stages. Related work has already pointed out the performance bottleneck for alternative approaches such as Verilog-AMS or VHDL-AMS [129]. In addition, SystemC AMS is open source. This allows the author of this work to develop additional approaches to further speed up simulations.

Nevertheless, an alternative approach is investigated, namely the use of SystemC for real number modeling. This approach fulfills the requirements for a high level of abstraction and circumvents any simulator coupling. It is open source and has been demonstrated to be a highly efficient solution. Therefore, the following section presents a comprehensive examination and comparison of these two modeling approaches. The analysis is centered on the TDF MoC. For a comparison with the ELN/LSF MoC, please refer to the work of Scherr and Einwich [135]. Furthermore, the investigation permits a more detailed examination of the implementation of SystemC AMS. It provides a foundation for the development of strategies to further accelerate simulations. These are presented in chapter 5.

4.3 Performance Analysis and Comparison of SystemC AMS and Real Number Modeling in SystemC

The SystemC AMS timed data flow model of computation is based on the synchronous data flow semantics and relies on a static schedule [79, 80]. The objective of this measure is to establish an efficient sample-based simulation, which will effectively circumvent the overhead associated with SystemC's dynamic scheduling as stated in the language's user manual [78]. Nonetheless, this implies that TDF models could also be formulated in SystemC. This section presents an analysis of the performance of the AMS extensions in comparison to the usage of pure SystemC, with the aim of motivating the usage of the AMS extensions. Most aspects of the given comparison have been previously published by the author of this work in [P3]¹. Subsequently, the outcomes are conveyed and assessed on the basis of a real-world example, namely a virtual sensor prototype of a gyroscope.

First of all a few words about the design of experiments: The investigation starts with fundamental experiments using minimal examples to distinguish the different influencing factors on the simulation runtimes. Both languages represent the core functionality of a module as a set of user-defined C++ statements. Consequently, all performance differences can be directly attributed to the characteristics of the underlying framework and the execution semantics of the language libraries. The experiments presented target to allow direct comparisons between the two languages. This puts minor constraints on the modeling style. The impact and implications of these constraints are stated and discussed.

The core functionality of a TDF module is embodied within its member function *processing()*. The behavior can be replicated in SystemC when a single process is attached to the module. Therefore, each SystemC module within the forthcoming simulation configurations is constrained to a single process. In addition to enhanced comparability, this is also due to the fact that the modules utilized do not encompass any sophisticated functionalities. In order to enable the analysis of scheduling effects, the user-defined components have been kept as concise as possible.

A SystemC process may be implemented as a method or as a thread. The two implementations are provided where it is deemed relevant to do so. In experiments that do not explicitly distinguish between these two implementations, the default approach is to use methods. The experiments were conducted on a Unix server, with ten simulation runs performed for each setup and for each value of the sweeping parameter. All data points shown in the upcoming graphs are averaged over these runs. The standard deviations remained within a few percent of the mean values in all cases. The open-source versions of SystemC AMS 2.3.0 and SystemC 2.3.3 were utilized in their original form, as provided by ACCELLERA [14].

In the following, SystemC ports and signals are referred to as discrete event (DE) ports and signals, respectively, in accordance with their timing properties. Similarly, SystemC AMS setups are identified via the abbreviation TDF to account for the model of computation employed.

4.3.1 Update Mechanism

The comparison is started with an investigation of the update mechanism. This refers to the time consumed by port accesses and signal updates. In order to isolate this mechanism from any scheduling effects, the simulation setup is limited to a minimum size. This corresponds to a single module where applicable. Input ports require a driver such that a second module is added that acts as source. The module under test does not incorporate any sophisticated functionality but solely implements port accesses. Three different types of ports are considered:

- DE in-/outputs (*sc_core::sc_in*, *sc_core::sc_out*)

¹See section 1.3 page 5 for the author's contribution to this publication.

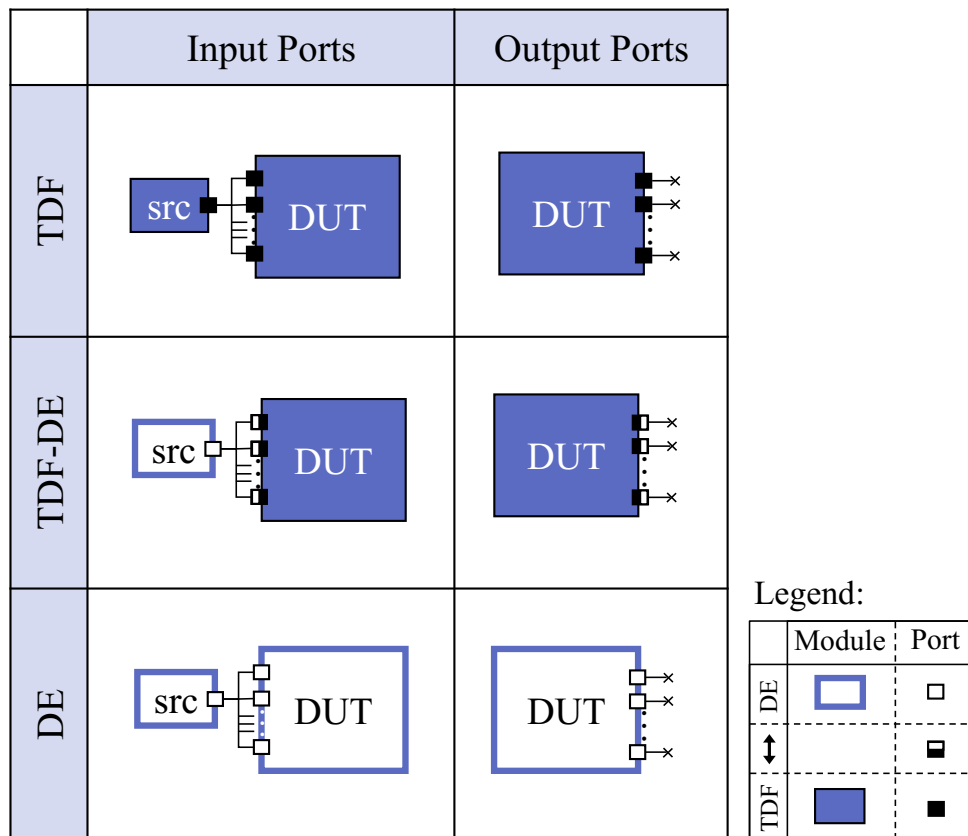


FIGURE 4.2: Cluster setups for the experiments targeting the update mechanism. Input ports require a driving signal which is provided by a source module that implements a simple counter.

- TDF in-/outputs (*sca_tdf::sca_in*, *sca_tdf::sca_out*)
- TDF-DE converter ports (*sca_tdf::sc_in*, *sca_tdf::sc_out*)

SystemC TLM ports are not discussed as no counterpart within SystemC AMS exists and they are not directly applicable to AMS modeling.

A total of six different port configurations were investigated which results in six distinct model setups for the experiments. The setups are depicted in Fig. 4.2. In each experimental setup, the device under test (DUT), which can be either a SystemC or a TDF module, is equipped with 100 ports of a single type. The high amount of ports is required to make the rather small deviations visible. Nevertheless, it is not implausible that a substantial number of ports may be present within the design of an SoC.

The objective is to compare the pure wall-clock times of port accesses (read/write operations) between the languages. This is accomplished by conducting multiple runs with different numbers of port operations. The DUT instantiates 100 similar ports in each run but only a subset of these ports is accessed (read/written). These ports are referred to as *active*. Multiple simulation runs are performed sweeping over the number of active ports. By plotting the simulation time in terms of wall-clock time over the number of active ports, the increase in runtime is directly related to the time consumed by the additional port accesses. The analysis of the results is based on the slope of the resulting graphs to cancel out the effects of the elaboration phase. Nevertheless, the same number of ports is always instantiated to keep the elaboration time constant and ease the optical inspection of the readers.

For the experiments focusing on output ports, it is sufficient to bind their ports to a set of signals. The experiments focusing on input ports additionally require a second module to drive

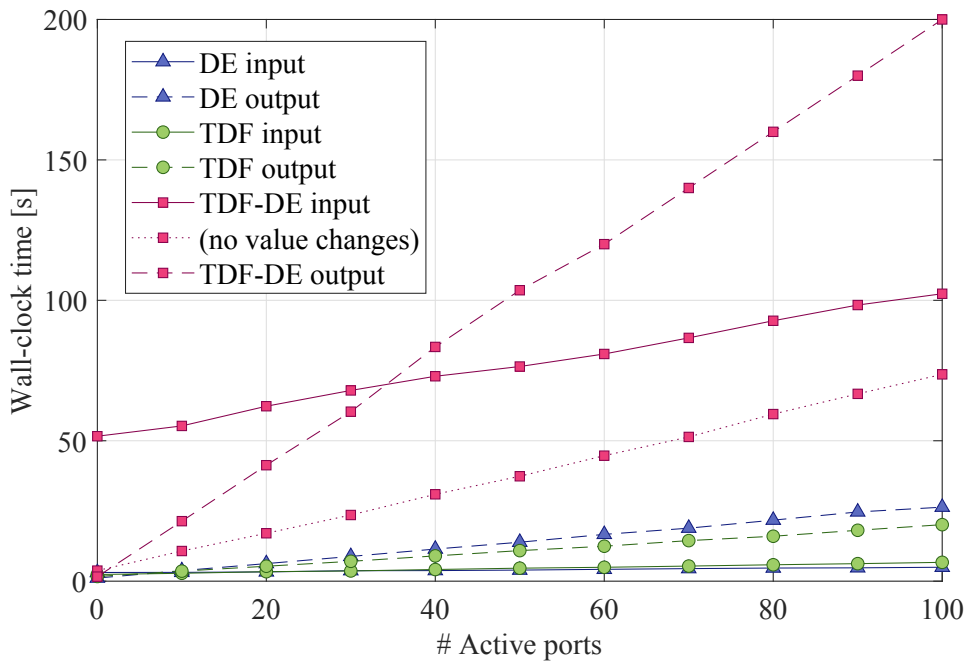


FIGURE 4.3: Impact of port operations on the simulation performance. Each setup consists of one module with 100 ports and a source module if required. Active ports are read/written, inactive ones are not accessed.

the connected signals. A source module is added as shown Fig. 4.2 which implements a simple counter. An alternative setup keeps the driving signal constant. Since the computational effort of the source module does not change with the sweeping variable, its impact on the evaluation of the slope is negligible, similar to the influence of the elaboration phase. The simulation results for a 10 s simulation with a timestep of $1.0 \mu\text{s}$ are shown in Fig. 4.3.

The DE and TDF input ports perform best. Reading an input simply assigns a new value to a variable. Writing to an output port consumes a bit more time because it triggers an update of the connected channel (signal). The DE output port introduces a slight additional processing burden due to the update mechanism inherent to SystemC. The new value of a TDF output port becomes valid instantaneously and can thus simply be written to the port and transferred to the channel. In contrast, the value of a DE port becomes valid only in the next delta cycle. Therefore, writing an output port triggers an update event that is executed after the list of runnable processes for the current timestep has been completed.

It is evident that the converter ports exhibit a notable inferiority in performance compared to the other ports. In addition to updating the value, writing a converter output port also initiates a synchronization event between the TDF scheduler and the DE kernel. This process consumes the additional wall-clock time that is visible in the graph. A similar effect diminishes the performance of the converter input ports. Nevertheless, the synchronization task does not explain why the wall-clock time remains high if all ports are inactive. A more thorough examination of the implementation reveals that the instantiation of a converter input registers a process that buffers all events occurring at the channel in preparation for the next invocation of the function *read()*. When the simulated time progresses, these events eventually become outdated if no port reading takes place. A garbage collection mechanism then deletes these events and resizes the corresponding buffer, which results in an increased simulation time. To demonstrate this effect, an additional setup has been introduced in which the source module provides a constant stimulus. Consequently, no *value-changed events* are triggered, and the buffer keeps empty. This causes the wall-clock time without active ports to become similar to that of the other setups. In

TABLE 4.1: Performance metrics related to port operations in SystemC and SystemC AMS. The profiling was conducted reading/writing the complete set of 100 ports per processing step.

	Calculations based on Eq. (4.1)		GPROF Profiling results for the processing() calls	
Inputs				
Variant	t_{read} [s]	Rel. Speed	Time/Call [s]	Total Time [%]
DE	1.99e-9	1	1.11e-7	59.7%
TDF	4.27e-9	0.466	2.20e-7	81.8%
TDF-DE	5.11e-8	0.039	6.94e-7	7.0%
Outputs				
Variant	t_{write} [s]	Rel. Speed	Time/Call [s]	Total Time [%]
DE	2.56e-8	1	2.29e-6	93.9%
TDF	1.82e-8	1.407	1.82e-6	92.6%
TDF-DE	1.98e-7	0.129	3.69e-06	26.9%

order to quantify the results, a formula is derived that extracts the wall-clock time of a single port operation. A simulation run with N cluster executions and n active ports conducts $n \cdot N$ port operations. Taking the difference of the wall-clock times of two different setups further allows to cancel out the share of the elaboration time. Keeping these relations in mind, one can derive

$$t_{read/write} = \frac{t_{sim,n} - t_{sim,0}}{n \cdot N} \quad (4.1)$$

with $t_{sim,i}$ being the average runtime of a simulation run with i active ports. To suppress any residue variations, the line of best fit of each graph has been taken to represent $\frac{t_{sim,n} - t_{sim,0}}{n}$. Table 4.1 summarizes the results which allow to calculate the relative speed of the different ports. The DE port is taken as baseline. In regard to the converter ports, the configuration utilizing the counter module as the source is employed. This represents the upper boundary for the wall-clock time, with a value-changed event triggered in each execution step.

In regard to the converter ports, the configuration utilizing the counter module as the source is employed. This represents the upper boundary for the wall-clock time, with a value-changed event triggered in each execution step. To further validate the results, the profiling tool GPROF was used to extract the wall-clock time consumed by the *processing()* calls [137]. In the case of DE and TDF ports, the results align with the calculations. The time per call is approximately a hundred times larger than the calculated value. As 100 ports are written per call and execution, this aligns with the expectations. In the case of the converter ports, the value obtained from profiling is considerably lower than the calculated one. However, it is evident that the percentage of wall-clock time consumed by the *processing()* method is considerably lower than in the other cases. This discrepancy can be attributed to the fact that the additional synchronization overhead is triggered by framework objects that the profiler is unable to directly correlate with the *processing()* call. Nevertheless, as the overhead is caused by the converter port, it is taken into account for the comparison. Still, the observations show that conventional profiling tools have only limited benefit in the context of SystemC. Therefore, the tool has not been used further in the upcoming experiments. All in all, the results show that the internal connections within a TDF cluster are considerably faster than those that connect to the digital domain. Both write and read operations between the TDF and the DE domain are more than ten times slower than their corresponding operation inside a TDF cluster.

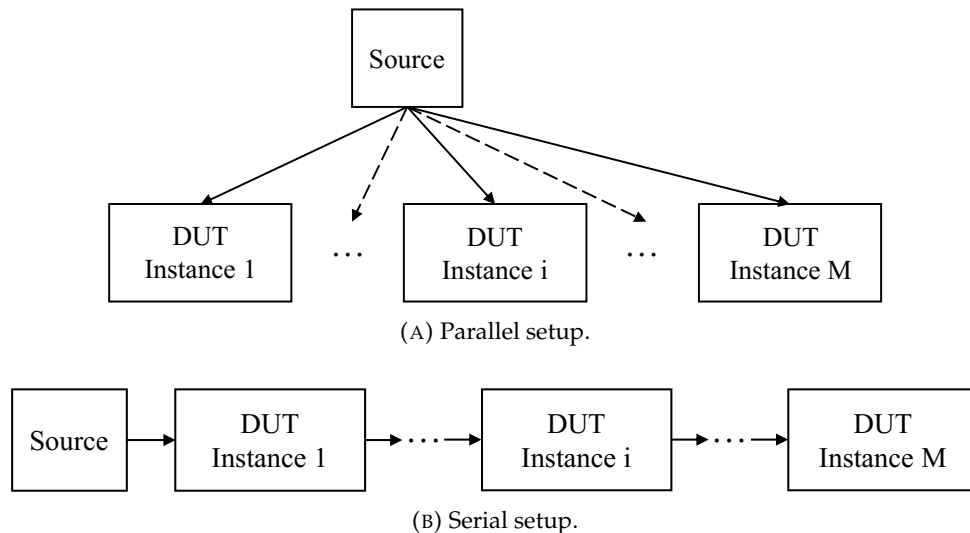


FIGURE 4.4: Cluster setups for the scheduling experiments. All modules besides the source are instances of the DUT having one in- and one output. The class can either be a SystemC or a TDF module.

4.3.2 Scheduling

As a next step, the minimal example is expanded to encompass basic cluster structures to examine the impact of scheduling effects. In theory, SystemC AMS should be advantageous in this regard due to the static schedule. In order to substantiate or refute this claim, two distinct cluster setups are taken into consideration. One of them consists of $M \geq 1$ parallel modules, as shown in Fig. 4.4a, which are supplied by a shared source module. The alternative setup reflects a series connection of these modules, whereby each module stimulates its successor. It is illustrated in Fig. 4.4b.

The primary difference between these configurations is the number of delta cycles that are necessary in SystemC until all modules have been executed once. Both setups execute the source module first. The class implementing the DUT consists of a single method that is sensitive to its single input ports. The execution of the source module notifies a value-changed event. As all instances of the DUT are sensitive to this event in the parallel setup, all of them are triggered within the first delta cycle. In contrast to this, the serial setup requires one delta cycle per instance as each instance triggers the next one for the subsequent delta cycle. The total number of delta cycles required is therefore equal to the number of instances.

A sweep over the number of modules is conducted in the experiment. The target is twofold. On the one hand, this approach allows to compare the scheduling efficiency of SystemC and its AMS extension for the extreme cases, namely the minimum and maximum number of delta cycles, to evaluate the boundaries within which real-world models will lie. On the other hand, it allows to separate the effect of delta cycles from other scheduling effects.

Figure 4.5 shows the simulation results for a 10 s-simulation and a timestep of 1 μ s. Even in the case of the parallel setup (solid lines), SystemC AMS performs significantly better than SystemC. Here, the static schedule pays off. Given that the execution order of the modules within one cluster is fixed, there is no need for them to occur as independent processes. The TDF cluster is handed over to the SystemC kernel as a single thread, which executes the entire cluster before it suspends. In contrast, the SystemC setup has M threads or methods if M represents the number of modules. Consequently, the SystemC AMS setup saves M threads (based on M instances and one source module) in comparison to the thread-based SystemC setup.

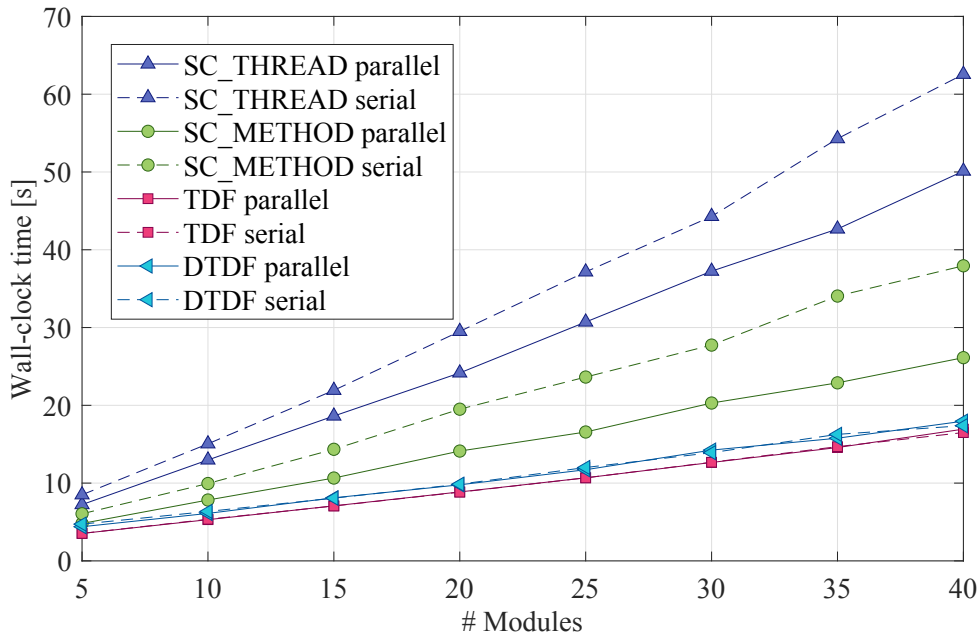


FIGURE 4.5: Scheduling effects in SystemC/-AMS. The serial setup in SystemC requires $M \delta$ -cycles with M being the number of modules not counting the source. The parallel setup requires only a single δ -cycle.

TABLE 4.2: Extracted wall-clock times based on the simulations shown in Fig. 4.5. The time per module is extracted from the parallel setup and the time per δ -cycle reflects timing differences between the parallel and the serial setup.

Setup	Wall-clock Time/Module [s]	Relative Speed
SC_THREAD	1.217e-7	1
SC_METHOD	6.09e-8	1.9967
TDF	3.78e-8	3.2183
DTDF	3.90e-8	3.1220
	Wall-clock Time/δ-Cycle [s]	Difference
SC_THREAD	3.29e-8	1.4e-9 (abs.)
SC_METHOD	3.15e-8	4.35% (rel.)

In order to quantify the impact, the average wall-clock time per module has been evaluated for the parallel setups. The results are calculated according to

$$t_{mod} = \frac{t_{parallel,M} - t_{parallel,m}}{(M - m)N} \quad (4.2)$$

and are presented in the upper part of table 4.2. In this context, m and M describe the number of modules with $m < M$ and N denotes the number of cluster executions. The quantity describes the slope of the sweeping line, normalized to the number of cluster executions. Once more, the line of best fit has been employed to generate the results depicted in table 4.2. The relative speed is calculated using the SystemC thread implementation as a baseline. The TDF setup exhibits a simulation speed that is more than three times faster than that of the implementation with SystemC threads. It is also noticeable that methods perform significantly better than threads. Nevertheless, SystemC AMS runs twice as fast as the method implementation.

Additionally, a dynamic TDF (DTDF) setup that changes its timestep dynamically has been

implemented to check how the simulation performance is impacted if clusters are triggered irregularly. The cluster alternates between two timesteps ($1.5\ \mu\text{s}$ and $0.5\ \mu\text{s}$) as a simple mechanism to create the worst case where the timestep is altered after each cluster execution. To ensure a fair comparison, the overall amount of executions is kept equal across the setups. As all port attributes (rates, delays) remain unaltered, the static schedule of SystemC AMS remains valid for the complete simulation. Only the timestep between two cluster executions is varied. As a consequence, the performance reduction is minimal, and the wall-clock time of the DTDF setup stays close to the static TDF case.

The serial setup performs similar to the parallel one when using SystemC AMS. Given that the schedule is static and consists of the same list of modules in both cases, any discrepancy in performance can reasonably be assumed to occur during elaboration time. This impact is negligible here. In contrast, the additional delta cycles for the SystemC setup have a discernible impact. Comparing the two setups reveals that the average wall-clock time for setting up an additional delta cycle can be derived to

$$t_\delta = \frac{t_{\text{serial},M} - t_{\text{parallel},M}}{(M-1)N} \quad (4.3)$$

with $t_{\text{serial},M}$ and $t_{\text{parallel},M}$ denoting the respective wall-clock times of the two types of setups with M being the number of instances of the DUT. Again, the line of best fit is taken for each setup to calculate the quantity. The results are presented in the lower half of table 4.2. The resulting wall-clock times per δ -cycle are expected to be independent of the type of the process because they are related to actions within the SystemC kernel. In the experiment shown, the wall-clock times per δ -cycle deviate only within a few percent between the implementation using threads and that using methods.

4.3.3 Dynamic Timing

As previously stated, the DTDF configuration applied in the preceding experiment solely modified the cluster timestep. The relations between the modules within the cluster have remained unaltered such that the static schedule kept valid throughout the simulation. The SystemC AMS language permits to change port rates and delays during the simulation. Changing the port rates triggers the calculation of a new schedule. Therefore, the experiments on the parallel structure from the previous section have been extended by a fifth setup in which the rate of the source module toggles between one and two. Again, the overall number of executions for the M modules in parallel fits to all previous setups to keep the comparison fair. Figure 4.6 illustrates the results. The new setup exhibits a significantly bigger runtime than the previous ones. As the schedule is recalculated after each cluster execution, the advantage of a static schedule turns into a bottleneck.

4.3.4 ODE Performance

All experiments conducted so far were designed in a manner that allows for a straightforward one-to-one mapping between the SystemC and the AMS implementation. However, the TDF MoC also permits the integration of ordinary differential equations (ODEs) into modules, which is an explicit extension in comparison to the capabilities of SystemC. The TDF MoC supports primitives for linear transfer functions (LTFs) and state-space (SS) equations for that purpose. The following experiment examines the simulation performance of this extension when varying step sizes are used. Its results will be of interest for the upcoming chapter. The device under test is a Chebyshev filter. The simulation was conducted using both a fixed

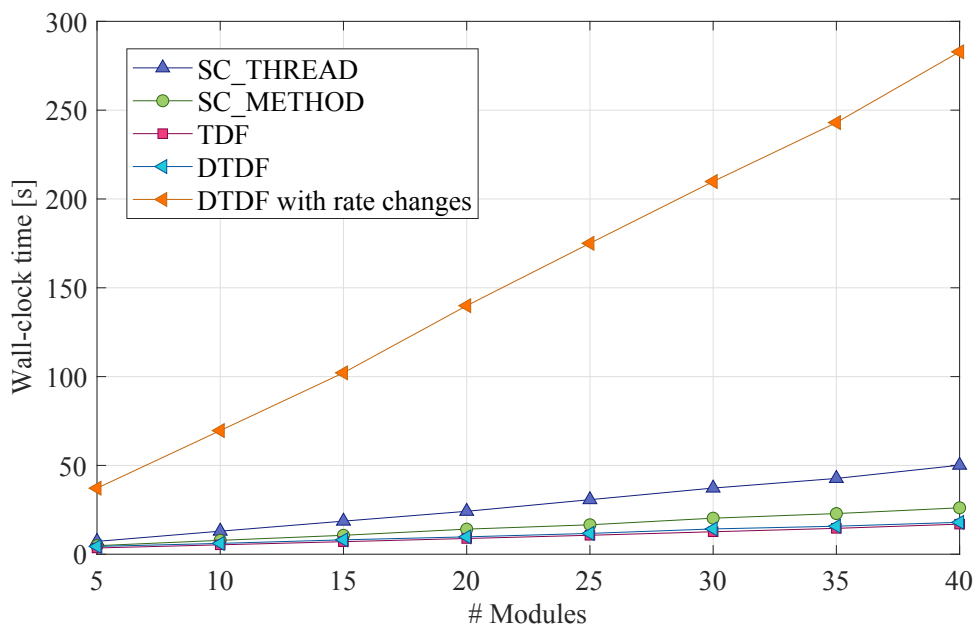


FIGURE 4.6: Simulation results of the parallel setups as shown in Fig. 4.5. An additional setup is given that modifies port rates after each cluster execution. This triggers the evaluation of a new static schedule each time.

timestep (TDF) and a varying timestep (dynamic TDF, DTDF). The number of cluster executions is equal across all experimental setups. To demonstrate the impact of the size of the differential equation system, a sweep over the filter order is performed. Figure 4.7 shows the resulting simulation times in terms of wall-clock times. The use of a fixed timestep results in a slight increase in runtime with the filter order, due to the elevated computational expense associated with solving the ODE. The DTDF setup exhibits significantly inferior performance due to the reinitialization of the underlying solver matrices with each execution, as these matrices are dependent on the specified timestep. A more detailed examination of this topic will be provided in the subsequent chapter. In the selected configuration, which employs only two distinct step sizes, the issue can be circumvented by instantiating two ODE instances and alternately directing the computations between them. Consequently, each instance operates with a single, fixed step size, thereby eliminating the need for reinitializations. Nevertheless, this approach is not viable when the step size is subject to arbitrary changes.

4.3.5 Application to a Virtual Sensor Prototype

A final performance comparison was conducted on a real-world example in order to classify the aforementioned results. The SystemC AMS model of the gyroscope described in the preceding chapter was employed for this purpose. An overview of the model is available in Fig. 3.6 on page 41. The objective of the experiment was to construct an equivalent SystemC model comprising the same blocks, supporting similar functionalities and delivering comparable data. The aim was to compare the simulation performance of this model to that of the TDF model. The modules were transferred based on a one-to-one mapping scheme, ensuring that the overall structure remained unaltered.

The majority of processes were selected to be responsive to value changes of their respective data input ports, thereby transforming the signal chain into an event chain operating with a fixed number of delta cycles. Furthermore, additional time-based trigger mechanisms were employed for the digital portion of the frontend, with the objective of ensuring the proper

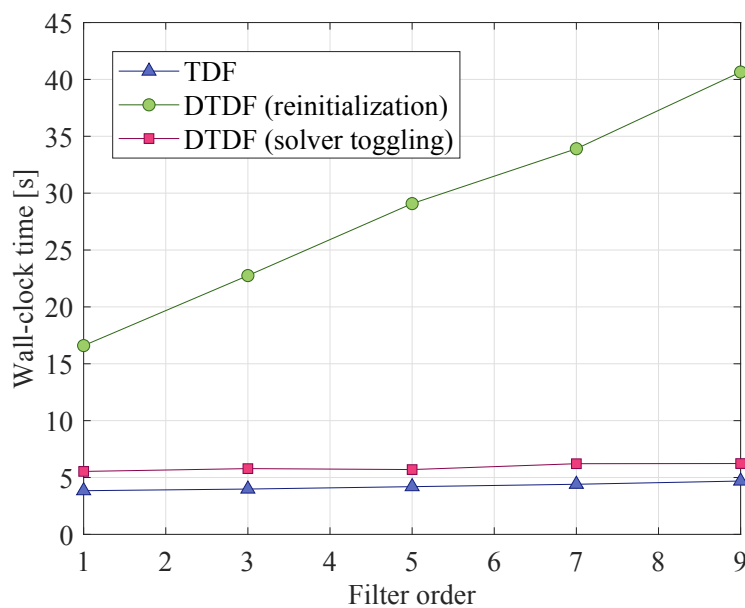


FIGURE 4.7: Wall-clock times of simulating Chebyshev filters of different orders for 10 s. The DTDF setups toggle the cluster timestep between 0.5 s and 1.5 s. The solver toggling setup thereby alternates between two ODE solver instances, each working on a fixed step size.

functioning of the filter stages. The ADC provides a bitstream that is subsequently fed into a CIC filter. It is possible for two consecutive bits of the bitstream to have the same value; however, the CIC filter must still trigger to be able to handle the samples correctly such that triggering based on value-changed events is not feasible. Furthermore, the drive loop must incorporate a delay element to prevent the generation of an infinite loop of delta cycles.

As SystemC lacks intrinsic capabilities for integrating ordinary differential equations, the library BOOST was employed to assess the ODE of the damped harmonic oscillator in each transient step [138]. The less crucial ODEs, namely the bandwidth-limiting filters in the CV converters, were transferred to the digital domain using a bilinear transform.

The TDF model is implemented as a multirate cluster due to the presence of downsampling operations within the filter stages. As the concept of port rates is inapplicable to discrete event based modules, each of the downsampling filter stages is triggered at each input sample and implements an internal counter to evaluate the point in time at which a new value should be written to the output. It should be noted that the SystemC setup has been designed without the explicit use of clock signals. The timing granularity is introduced by a thread with a timed wait statement within the model of the MEMS mechanics, which automatically affects the other modules due to their sensitivity lists.

Ultimately, both models exhibit comparable granularity and simulation results, operating with analogous sampling frequencies. Table 4.3 depicts the results of the performance comparison. The two models were each simulated for a period of one second, with a maximum module frequency of 2.3 MHz. Ten consecutive runs have been performed without showing significant deviations in the simulated times. The average values are given in the table. The models were developed using the commercial tool Coside, created by COSEDA TECHNOLOGIES [139]. The tool incorporates its own profiler, which is integrated directly into the language libraries. In addition to providing the overall wall-clock time, the tool also estimates the kernel and profiling time, thus enabling the user to extract the pure model execution time. Furthermore, it indicates the proportion of the overall simulation time allocated to each block. A performance

TABLE 4.3: Performance Results of a virtual prototype modeling the mechanics and the analog frontend of a MEMS sensor system. The overall wall-clock time was separated into model execution time and kernel and profiler time. The profiler from the tool Coside by COSEDA TECH. was used [139].

Model Version	Wall-clock Time [s]	Model Exec. Time [s]	Kernel & Profiler Time [s]
TDF	28.85	24.41	4.44
TDF (less LTFs*)	24.69	20.33	4.36
SystemC	26.80	22.49	4.31

*LTFs removed from CV converters, still used for MEMS model.

bottleneck was identified with the help of the profiler, caused by the dynamic allocation of memory for local vector-based variables. The results presented in Table 4.3 reflect the final status following the optimization of the implementation. As the profiler indicated that the blocks with the highest relative share were precisely those including LTFs, a second version of the TDF model was tested with LTFs within the frontend replaced by analytical formulas based on the bilinear transform, similar to the SystemC model. The MEMS model was still modeled using the SystemC AMS internal ODE solvers to ensure sufficient precision in the modeling. The results demonstrate that the overall discrepancy in simulation performance is minimal. Including the transfer functions in the four CV stages, the TDF model is approximately 8% slower than the SystemC model. Conversely, the TDF model with digital filters is approximately 8% faster than the SystemC model. It should be noted that all 22 accessed register fields (see Table 3.1 on page 41) are represented as DE-TDF converter ports within the TDF version of the model. However, these values change only sporadically. All outputs of the model are implemented as TDF-DE converter ports, because the model is intended to be simulated jointly with the digital components of the sensor. In contrast to the experimental setups previously described, the real-world example is dominated by the computational cost of the module implementations, rather than by the simulation kernel. Moreover, all variants of the model are primarily impacted by the MEMS modules because the corresponding ODE solvers insert significant computational cost. These observations collectively indicate that the impact of scheduling effects is relatively minor within the context of this experimental setup. However, the modeling procedure was found to be more straightforward when utilizing SystemC AMS, as its semantics align more closely with the requirements of AMS models at the selected level of abstraction. The availability of ODE solvers and the sample-based approaches to port rates and delays facilitate the reflection of analog systems in a convenient and comprehensible manner.

4.4 Discussion

The present chapter dealt with the choice of a suitable simulation language for AMS modeling of HVPs. It was stated that SystemC AMS fits well to the requirements on such a language because it allows to develop fast and abstract models, connects well to SystemC and is extensible thanks to being open source.

The simulation performance of SystemC AMS TDF models in comparison to the real number modeling approach applicable to SystemC has been examined. A series of artificial experiments was presented to illustrate the discrepancies in the semantics of the languages with regard to update mechanisms, scheduling, and dynamic timing. It was observed that converter ports exhibited significantly slower performance than those that remained within their designated domain. The results also demonstrated the beneficial impact of the static schedule of SystemC AMS, provided that the schedule remains unaltered for the majority of cluster executions. However, the application of this approach to a real-world example, namely a MEMS gyroscope model, did not yield conclusive evidence of a significant performance-related advantage of

SystemC AMS compared to an equivalent SystemC model. In contrast, the wall-clock times of both model versions varied within a few percent only. They were primarily constrained by the computational cost associated with solving the differential equations inherent to the mechanical part of the model.

Nevertheless, the results of the experiments provide a reasonable guideline for modelers with very stringent performance requirements. Since the impact of the scheduling increases with the number of modules and the depth of the event chain, it may become significant for very large models with low computational cost within distinct modules. Moreover, the benefit of the static scheduling is relativized by the impact of converter ports. Therefore, the interface between the AMS components and the digital part should be kept as lean as possible.

With regard to the application under examination, the impact was found to be on the order of a few percent only. Although the performance of both model versions was comparable, the modeling procedure was found to be significantly more straightforward when using SystemC AMS. Accordingly, the language is still regarded as the preferred option and will be utilized throughout the remainder of this work. Furthermore, the analysis presented in this chapter provides a solid basis for enhancing the simulation efficiency of SystemC AMS through the incorporation of new features, which will be discussed in greater detail in the subsequent chapter.

Chapter 5

Increasing the Solver Efficiency in SystemC AMS

Developing and especially debugging a feature on a simulation model often requires a large number of simulation runs. Therefore, increasing the simulation performance is an important step in speeding up development tasks and the question how to achieve that is one of the research questions of this work (*RQ2-Performance*). The biggest lever for the simulation speed in TDF models is their discrete time step. It introduces an inherent trade-off between accuracy and performance. The release of the SystemC AMS 2.0 standard in 2016 introduced dynamic language features that allow a module's timestep and/or port rates and delays to be modified at runtime [79]. In theory, these features offer a wide range of potential for optimizing simulation performance. However, the timestep remains subject to the designer's choice which requires a good understanding of the system dynamics and the underlying solver in order to achieve the desired accuracy while minimizing simulation time.

In many physical and analog models, the fundamental dynamics of the system are defined and described in terms of ordinary differential equations (ODEs). SystemC AMS provides primitives for linear transfer functions (LTFs) and state-space (SS) equations to describe them. These are available for both the TDF and LSF MoC. Furthermore, ODEs are introduced indirectly by the user by connecting primitives of the LSF or ELN MoC. In summary, the LSF and ELN MoCs always employ ODE solvers, whereas TDF models may utilize them, see Fig. 5.1. The definition of physical or analog characteristics in terms of transfer functions is a common approach. In the field of MEMS applications, the micro-electromechanical component is commonly abstracted as a set of damped harmonic oscillators, which allow for a convenient description using ODEs. If a reduced order model is used for more detailed modeling, it is again represented by an ODE system. Another common area of application is filtering. Thus, many TDF clusters include one or multiple differential equations.

When both aspects are considered together, it becomes evident that there is a potential for the introduction of variable-step ODE solvers into SystemC AMS in order to speed-up the simulation of heterogeneous virtual prototypes. In particular, if the ODEs represent the principal dynamics of the system, as it is the case for MEMS modeling, the corresponding solvers are well suited to control the step size of the overall cluster. In light of the above, this chapter introduces measures to incorporate variable-step ODE solvers into the semantics of SystemC AMS. Based on the results, a timestep quantization approach has been developed with the objective of further accelerating the simulation for variable-step implicit solvers. The approach is then adapted and transferred to provide support for ELN circuits with high switching activity. The resulting approach is referred to as switch monitoring.

Section 5.1 sets the presented methodologies into the context of related work. Mathematical basics of ODEs and their solvers are given in section 5.2. The integration schemes into SystemC AMS are discussed in section 5.3 before discussing their interference with other language features in section 5.4. Furthermore, additional optimizations regarding timestep quantization

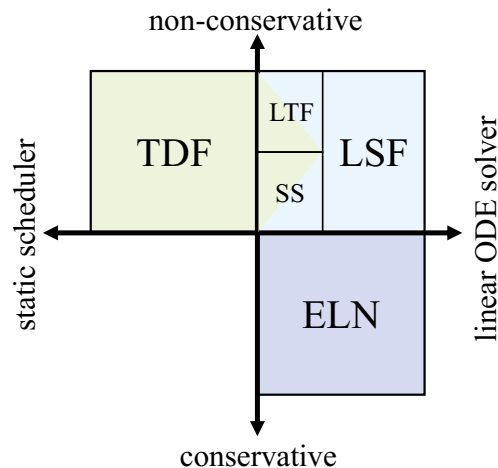


FIGURE 5.1: SystemC AMS models of computation (MoCs) and their respective solvers. Linear transfer functions (LTFs) and state-space (SS) equations are available in the timed dataflow (TDF) and linear signal flow (LSF) MoC. The third MoC is called electrical linear network (ELN).

and switching efficiency are introduced in sections 5.5 and 5.6. Subsequently, section 5.7 gives insights into the implementation and the user interface of all developed measures. Finally, section 5.8 presents experimental results before the concluding discussion is given in section 5.9. The fundamental integration schemes as presented in section 5.3 have been previously published by the author of this work in [P6]. A more comprehensive summary of the approaches introduced in this chapter is available in [P7]¹.

5.1 Related Work

A number of related studies have been conducted with the objective of identifying or resolving issues pertaining to the semantics and implementation of SystemC AMS. An early investigation of the abilities of SystemC AMS has been presented by Adhikari and Grimm in 2010 [140]. These issues have been partially addressed by the update to SystemC AMS 2.0. The authors had posited the necessity of analog models to react to events, a claim that has been resolved by the incorporation of a trigger mechanism into the dynamic TDF models. Other limitations, such as the purely linear behavior of LSF models, remain in place. Recommended primitives for saturation and alike are today still not part of the LSF MoC. Instead, these behaviors are mostly modeled through proper interactions between the linear MoCs and supporting TDF modules. A more recent and comprehensive examination of the capabilities and limitations of SystemC AMS can be found in the work of Fraccaroli and Vinco, published in 2022 [107]. The authors analyze the language from the context of cyber-physical production systems with the help of multiple case studies, for instance the modeling of a wind turbine. The authors categorize their findings into four distinct groups: schedulability issues, solver issues, MoC-related issues, and implementation issues. The latter group focuses on the C++ realization of instantiation and static scheduling and is not further addressed here. The others are briefly discussed below.

Two significant schedulability issues were identified in their work. Firstly, SystemC AMS is only capable of recognizing whether a cluster is schedulable at runtime. Furthermore, the information provided regarding the source of the problem is limited to a set of modules that have been scheduled thus far. This issue has already been partially addressed by Andrade et

¹See section 1.3 page 5 for the author's contribution to both publications.

al., who present a method for the static detection of synchronization issues between the DE and TDF domains [141]. The issue of TDF internal schedulability remains unaddressed. Secondly, SystemC AMS is unable to resolve algebraic loops. Therefore, it is necessary to explicitly break algebraic loops by introducing delay elements. In the event that the dependency loop cannot be calculated sequentially, it must be resolved in a preceding step and the solution is then modeled in SystemC AMS. For instance, Fraccaroli and Vinco address the cyclic dependencies of their wind turbine model symbolically prior to constructing the actual TDF model [107]. While addressing these issues would enhance the convenience of creating a TDF model, they do not directly impact the stated research questions of the thesis at hand. For this reason, they will not be explored further in this study.

The proof-of-concept implementation of SystemC AMS [14] is based on two implicit solvers, namely the backward Euler method and the trapezoidal method. Hartmann et al. have studied their limitations in the context of a crane controller [142]. The application of these rather low-order solvers to the control system of a crane does not yield a stable result. Instead, the crane position is oscillating around the setpoint. Using the same model with higher-order solvers has shown quick settling. Another solver issue has been presented by Rizzi et al. in 2021 [143]. The authors model a resistive crossbar array in the ELN MoC, hence utilizing the differential equation solvers of SystemC AMS. However, the variable resistances result in a dynamic state matrix for the system. The underlying implicit solver must invert the state matrix to solve the equation system. As a result, alterations to the entries in this matrix result in repeated inversions. In the proof-of-concept implementation [14], this process is already optimized using lower-upper (LU) decompositions. However, the issue still creates a performance bottleneck for large equation systems with frequent changes in the state matrix. Rizzi et al. propose to replace the variable resistors with equivalent circuits comprising controlled sources such that the varying entries do not affect the state matrix but the source vector [143]. This workaround enables the authors to accelerate their simulations considerably for their specific use case. However, it necessitates adaptations from the user and does not enhance the SystemC AMS implementation in general. The thesis at hand demonstrates alternative cases that suffer from regular changes in the state matrix. The timestep quantization and switch monitoring approaches proposed in this work are both based on this issue and seek to enhance the SystemC AMS implementation in order to address it.

Fraccaroli and Vinco have collected issues specific to each MoC [107]. With regard to the ELN MoC, the authors highlight the absence of support for non-linear elements and alternate current simulations. The primary limitation of both the TDF and LSF MoC is the selection of the timestep. As previously discussed, choosing a fixed timestep imposes an inherent trade-off between accuracy and simulation speed, which is challenging to optimize when the timestep is set manually. The situation gets even worse as changing the timestep imposes a reinitialization of the solver matrices, similar to the state matrix changes discussed above. In their study, Reuther and Einwich address this issue by employing the Woodbury formula [144]. The formula permits the fast calculation of the inverse of a matrix based on the inverse of another matrix that differs only in a few entries. This effect is employed to accelerate the recalculation process when step sizes are modified. However, the approach may create stability issues. Porras et al. attempt to circumvent the limitations of a fixed timestep through the implementation of a gradient-based timestep control and a threshold crossing detection mechanism [145]. The former method scales the timestep inversely with respect to the gradients of the signals, while the latter reduces the timestep locally when the signal value approaches the threshold and increases it again after the crossing has occurred. However, the correlation coefficients for the gradients and margins for the threshold detection are user-defined and challenging to determine.

The present thesis proposes an alternative methodology that partially circumvents the constraint of fixed step sizes. The approach is based on variable-step ODE solvers. To the best of

the author's knowledge, it represents the first solution presented for SystemC AMS that varies the timestep based on error estimation and internal timestep evaluation. Consequently, the user is merely required to specify the desired level of accuracy, after which the timestep is automatically adjusted to achieve this accuracy with optimal efficiency. The approach requires a proper synchronization between the ODE solver and the discrete time grid of the TDF scheduler. Two synchronization schemes are presented for that sake. The present study does not address the synchronization between continuous-time (CT) models and the DE kernel directly. This issue has recently been addressed by Fernandez-Mesa et al., who propose a novel synchronization scheme between CT and DE models [146]. However, this does not align with the hierarchical structure of SystemC AMS, where ELN/LSF modules can only be linked to the DE domain through an intermediate TDF module.

5.2 Ordinary Differential Equations

An ordinary differential equation describes a relation

$$\dot{y} = f(t, y(t), x(t)), f : \Omega \subset \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n \quad (5.1)$$

with time t , input vector x , output vector y and initial values

$$x(t = t_0) = x_0 \quad (5.2)$$

$$y(t = t_0) = y_0. \quad (5.3)$$

In transient simulations, these equations are usually solved numerically, and the continuous time is discretized into a fine grid. Each point in time with a valid solution is written as t_k , $k \geq 0$ whereby $k = 0$ denotes the initial state. The time interval between two valid solutions is referred to as step size and denoted with h further on.

The set of numerical solvers is among other criteria divided up into *explicit* and *implicit* solvers. Explicit solvers calculate the next sample y_{k+1} at time t_{k+1} based on the states of the current sample. The simplest explicit solver is a single step approach called forward Euler method [147]. It calculates each sample according to

$$y_{k+1} = y_k + h\dot{y}_k = y_k + hf(t_k, y_k, x_k) \quad (5.4)$$

which basically represents a linear extrapolation. Starting from the initial state set (t_0, y_0, x_0) , the iterative calculation of all subsequent steps is straightforward. A widely used class of explicit solvers is the Runge-Kutta family [148]. It is usually defined in the context of initial value problems, in which no input is considered, and follows the general scheme

$$y_{k+1} = y_k + h \sum_{i=1}^s b_i k_i \quad (5.5)$$

with

$$\begin{aligned} k_1 &= f(t_k, y_k) \\ k_2 &= f(t_k + hc_2, y_k + h(a_{21}k_1)) \\ k_3 &= f(t_k + hc_3, y_k + h(a_{31}k_1 + a_{32}k_2)) \\ &\vdots \\ k_s &= f(t_k + hc_s, y_k + h(a_{s1}k_1 + a_{s2}k_2 + \dots + a_{s,s-1}k_{s-1})). \end{aligned} \quad (5.6)$$

Thereby, s denotes the number of stages and a_{ij} , b_i and c_i represent sets of solver specific coefficients that are typically presented in the form of Butcher tableaux (see [147] for further reference). They are chosen according to stability constraints. The Runge-Kutta scheme can similarly be used with an additional input that is extrapolated to obtain the intermediate solutions of the differentials k_i .

A key characteristic of ODE solvers is their order, here denoted with p , which is a measure for the scaling of the local truncation error. Please note that the order p and the number of stages s are two distinct concepts although somewhat related. Assuming a solver with order p and step size h , the local truncation error ε_{tr} scales with

$$\varepsilon_{tr} \propto h^{p+1}. \quad (5.7)$$

Given that the number of steps for a given time interval scales inversely with the step size, the total accumulated error hence scales with

$$\varepsilon_{acc} \propto h^p. \quad (5.8)$$

The forward Euler method represents the simplest explicit solver and is a first-order method. In order to achieve higher-order solvers, it is necessary to employ multiple stages. The most prevalent Runge-Kutta solver is the RK4 method comprising four stages and being of fourth order ($s=p=4$). As a step of an explicit solver is computationally cheap, these solvers are typically very fast.

For some equation systems however, explicit solvers face stability issues. In these cases, extremely small step sizes are required to achieve convergence, and the step size is not limited by accuracy but rather by stability constraints. If the step size is set above the stability limit, high frequent changes disturb the solution. Systems that follow these characteristics are referred to as being *stiff* [149].

An implicit method tries to solve an equation where the new output vector y_{k+1} occurs on both sides. As these methods are A-stable (i.e. the region of stability covers the left half of the complex plane, cf. [150]), they are well suited to be applied to stiff systems. The simplest implicit method is the backward Euler method which follows the equation

$$y_{k+1} = y_k + h\dot{y}_{k+1} = y_k + hf(t_{k+1}, y_{k+1}, x_{k+1}). \quad (5.9)$$

As the names already indicate, the explicit or forward Euler method implements a left-sided ("forward") approximation whereas the implicit or backward Euler describes a right-sided ("backward") approximation. Solving Eq. (5.9) for y_{k+1} is apparently computationally more expensive than the pure analytical calculation of its explicit equivalent as described in Eq. (5.4). However, in the context of stiff systems, implicit solvers can use significantly bigger step sizes as they do not face stability issues. Consequently, they are typically still more performant than explicit solvers in these cases.

Both, implicit and explicit solvers can be designed as *fixed-step* or *variable-step* solvers. In fixed-step solvers, the step size is constant over the complete simulated time, such that

$$h_k = h_j \quad \forall \quad k, j > 0, \quad (5.10)$$

whereas variable-step solvers change the step size dynamically defining

$$h_k = t_k - t_{k-1}. \quad (5.11)$$

In the latter case, the choice of the step size is performed based on a local error estimation. In order to enable a proper approximation, two solvers of orders p and $(p - 1)$ are used to solve

the equation system and their results are compared. To minimize the computational overhead, these methods share intermediate steps. In the case of Runge-Kutta solvers assuming a higher-order method with s stages and order p that behaves according to Eq. (5.5), the lower order step calculates to

$$y_{k+1}^* = y_k + h_{k+1} \sum_{i=1}^s b_i^* k_i. \quad (5.12)$$

The method employs the same intermediate results, k_i , as the higher-order method but utilizes an alternative set of coefficients b_i^* . If it has fewer stages than the other solver, coefficients with indices above its number of stages are zero. As a result, the difference between the two solutions is given by

$$e_{k+1} = y_{k+1} - y_{k+1}^* = h_{k+1} \sum_{i=1}^s (b_i - b_i^*) k_i \quad (5.13)$$

and its evaluation creates negligible computational cost. A common formula for the estimation of the normalized local error is

$$\varepsilon = \frac{\|y_k - y_k^*\|}{\varepsilon_{abs} + \varepsilon_{rel}(a_x \|y_k\| + a_{\dot{x}} h_k \|\dot{y}_k\|)} \quad (5.14)$$

with ε_{abs} and ε_{rel} being user defined accuracy constraints for absolute and relative tolerance, a_x and $a_{\dot{x}}$ being the corresponding weights and $\|\cdot\|$ denoting the maximum norm [151]. The solver's target is to keep the normalized error below one. A classical formula used to determine the step size accordingly is

$$h_{new} = \begin{cases} h_{old} \cdot \max(0.9\varepsilon^{\frac{-1}{p-2}}, 0.2) & \text{if } \varepsilon > 1 \\ h_{old} \cdot \min(0.9\varepsilon^{\frac{-1}{p}}, 5.0) & \text{if } \varepsilon \leq 0.5 \\ h_{old} & \text{else} \end{cases} \quad [151]. \quad (5.15)$$

The notation with index k is avoided here because h_{new} can have different meanings regarding the absolute simulated time. As the choice of the step size is only an estimate, it can happen that the local estimated error exceeds its boundaries ($\varepsilon > 1$) and violates the user given accuracy constraints. As a result, the solution is rejected and a new step size is tried for the current step. Taking $h_k = h_{old}$ as a starting point, h_{new} would then describe an updated value for h_k . In the alternative case where the solution of the current step is accepted ($\varepsilon \leq 1$), the value h_{new} becomes the estimate for the next step size h_{k+1} .

5.3 Integration into the SystemC AMS Simulation Semantics

SystemC AMS provides primitives to explicitly introduce ordinary differential equations with input $x(t)$ and output $y(t)$ in two distinct formats, namely linear transfer functions and state-space equations. A linear transfer function describes an ODE in the Laplace domain according to

$$H(s) = \frac{Y(s)}{X(s)} = \frac{\mathcal{L}\{y(t)\}}{\mathcal{L}\{x(t)\}} = \frac{a_m s^m + a_{m-1} s^{m-1} + \dots + a_1 s + a_0}{b_n s^n + b_{n-1} s^{n-1} + \dots + b_1 s + b_0} \quad (5.16)$$

using coefficients a_i and b_i and the complex variable s . Thereby, $\mathcal{L}\{\cdot\}$ represents the Laplace transform. Contrarily, SS equations describe the system in the time domain according to

$$\dot{s}(t) = As(t) + Bx(t) \quad (5.17)$$

$$y(t) = Cs(t) + Dx(t) \quad (5.18)$$

with $s(t)$ denoting an internal state vector, and A , B , C and D being the state matrix, the input matrix, the output matrix and the direct feedthrough matrix.

The thesis at hand focuses on concepts related to heterogeneous virtual prototypes, which are defined to contain at least one analog and one digital module. Given that the ELN and LSF MoC do not provide direct conversions to the discrete event domain of SystemC, the assumption that at least one TDF module is instantiated can be made without loss of generality. ELN/LSF instances within the name spaces *sca_eln::sca_de* and *sca_lsf::sca_de* seem to provide direct conversions at first glance but internally define TDF-DE converter ports. For instance, the LSF sink *sca_lsf::sca_de::sca_sink* that provides an output to the DE domain internally instantiates a TDF-DE output port *sca_tdf::sca_de::sca_out*. This language restriction gives rise to two implications. On the one hand, the static scheduler is an indispensable component. If an ODE solver is additionally used, the proper synchronization between the ODE solver and the static scheduler must be ensured. On the other hand, handling the aforementioned synchronization properly automatically ensures the synchronization between the ODE solver and the discrete event kernel as this is always done in a two-step manner (solver-TDF, TDF-DE) and the TDF-DE synchronization is already covered by the SystemC AMS standard. In the following, the current time of the ODE solver is referred to as solver time t_s , the current time of the static scheduler as TDF time t_{TDF} and the time of the discrete event kernel as DE time t_{DE} . This section covers the synchronization schemes between the solver time and the TDF time. Thereby, the thesis at hand imposes two fundamental rules:

1. The solver time can move forward and backward. However, moving backward is limited to the rescission of a single timestep to allow the rejection of invalid solutions that do not meet the accuracy constraints.
2. The TDF time cannot move backward.

The current implementation of SystemC AMS permits the execution of arbitrary C++ operations within a TDF module and does not make any internal states accessible by the framework objects. Thus, it would be the user's responsibility to guarantee the proper reset of all internal states if a step was backtracked. The approaches presented in this chapter aim to ease the handling of SystemC AMS simulations and do not intend to put this burden onto the user. Furthermore, the process of backtracking a complete TDF cluster can be considerably time-consuming, particularly in cases where the cluster is large or the processing of the modules is computationally expensive. Thus, TDF time rollbacks are also avoided for the sake of enhanced simulation performance.

Figure 5.2 illustrates the integration of the the variable-step solver into the TDF cluster. The block which implements the ODE is called the **time controlling instance (TCI)**. The TCI can either represent a TDF module that contains an LTF or SS equation instance (see Fig. 5.2a), or a complete ELN or LSF cluster as shown in Fig. 5.2b. Both scenarios are interchangeable from the perspective of the static scheduler. An ELN/LSF cluster is conceptually equivalent to a single TDF module with unit in- and output rate.

The variable-step solver is internally used to solve the differential equation of the TCI and determines its step size $h_{k,TCI}$. The other modules of the TDF cluster follow the dynamic step size changes according to their port rates. For modules directly connected to the TCI and port

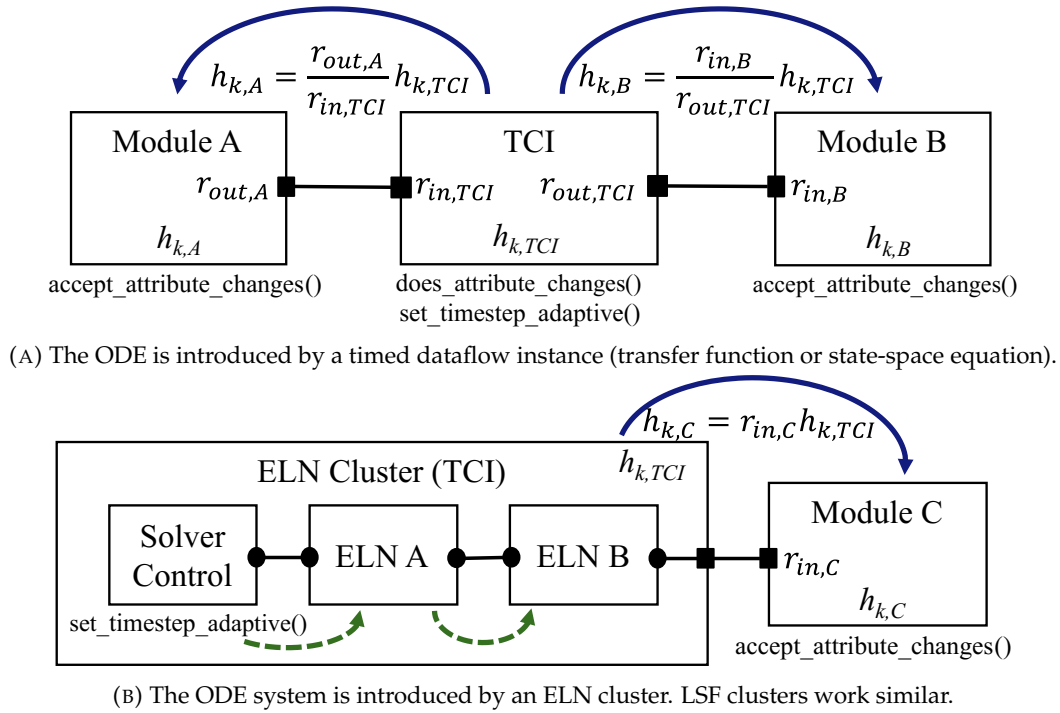


FIGURE 5.2: Exemplary TDF cluster with ODE solver instance. The block including the ODE is called time controlling instance (TCI) and controls the timestep of the complete cluster. The TCI is either represented by a TDF module with LTF or SS equations (A) or by an ELN/LSF cluster (B). From the perspective of the static scheduler, the TCI is handled as single block with module timestep and in-/output rates to the TDF cluster in either way. © 2025 IEEE.

rates denoted with r , the step sizes result in

$$h_{k,*} = \begin{cases} \frac{r_{out,*}}{r_{in,TCI}} h_{k,TCI} & \text{if module } * \text{ is connected to an input of the TCI} \\ \frac{r_{in,*}}{r_{out,TCI}} h_{k,TCI} & \text{if module } * \text{ is connected to an output of the TCI} \end{cases} \quad (5.19)$$

The SystemC AMS standard requires to explicitly state that a module will change the step size dynamically. Hence, the TCI must set the *does_attribute_changes* flag. Similarly, TDF modules must explicitly allow that other modules can influence their step sizes dynamically. Thus, all other TDF modules within the cluster must set the *accept_attribute_changes* flag. If a module shall run on a constant timestep, it can be split from the controlled cluster using decoupling ports to create another cluster. The synchronization between the two resulting TDF clusters is then either based on a sample-and-hold scheme or by interpolation based on the user's choice. According to the current language reference, only TDF modules can modify their timesteps dynamically during the simulation. This is achieved through the use of the *change_attributes()* method. In contrast, ELN/LSF modules are only allowed to set their timesteps during elaboration and must not actively alter timesteps during the simulation phase. Instead, they inherit propagated timesteps after the end of elaboration. The approach presented in the thesis at hand relaxes this restriction and extends the language specification to permit ELN/LSF clusters to actively change timesteps if they act as TCI.

The fundamental principle of variable-step solvers is based on the trial-and-error method. An appropriate step size is estimated and applied. In the event that the solution does not satisfy the accuracy constraints, it is rejected and the solver reverts the last step. While this is a

straightforward and common approach for ODE systems, where a dedicated state vector can be saved and restored if necessary, this approach is not suitable to globally rollback a complete TDF cluster because the internal and external states of each single module must be (re-)stored. Accordingly, the integration schemes outlined herein are designed to preclude TDF time rollbacks, thereby mitigating the overhead associated with unsuitable step sizes (see rule 2). This thesis presents two distinct methodologies. In the remainder of this work, these methodologies will be referred to as the PRECALC and SEGMENT methods. Both of them are integrated into the cluster setup as discussed above. Their working principles are explained below. The subsequent investigation of language interferences in section 5.4 gives clear indications when to choose which of the proposed methods. The optimal approach is dependent on the characteristics of the cluster.

5.3.1 Method 1: PRECALC

The first method is designated as PRECALC due to its reliance on the effect of precalculation. In order to restrict rollbacks locally to the TCI, it is necessary that the step size does not immediately impact the TDF time. This is accomplished by shifting the calculation by one sample into the future. Figure 5.3 illustrates the effect with a timing diagram. The simulated time is split into discrete intervals with t_0 denoting the start of the simulation. The diagram depicts a single input x and a single output y for the sake of readability. However, the approach works similarly for multiple in- and output ports as they are mathematically equivalent to single in- and output vectors. An initial value is assigned to the output. The TCI calculates each subsequent output sample one step in advance relative to the TDF time, such that the output sample y_{k+1} is evaluated at TDF time t_k . However, the sample is only valid at t_{k+1} . Therefore, the sample must be realigned to the TDF time. This is achieved through the use of an output delay. As these delays are sample-based in SystemC AMS, the time shift is automatically adjusted to the step size, shifting the output sample from t_k to t_{k+1} as intended.

Figure 5.3 clarifies the inherent trial-and-error approach at the example of the second step which starts at simulated time t_1 . A step size h_2^* is initially estimated. A corresponding solution for the output is evaluated to y_2^* and the associated error is estimated to ε_2^* . However, the resulting error exceeds its limit in this example ($\varepsilon_2^* > 1$) and the solution is rejected. As the processing of the TCI has not yet been completed, the TDF time has not yet advanced. Consequently, solely the state vector of the ODE system is reset to its most recent value, thereby facilitating a local backtracking within the module. Subsequently, a new, smaller step size h_2 is selected in accordance with Eq. (5.15). A second evaluation of the ODE is conducted, accompanied by a further local error estimation.

If the new estimated error is still beyond target, a second backtracking iteration will be conducted. In this example, the solution is valid ($\varepsilon_2 \leq 1$) and the result for y_2 is accepted. The value is applied to the module's output and the processing step ends. In the absence of an output delay, the newly generated output value would be immediately valid, allowing a subsequent module to read it at the current TDF time. This is a fundamental difference to SystemC where a value gets valid only within the next delta cycle. Here, the newly calculated value is targeted for the next timestep, thus a delay of one sample is registered for the output port. It should be noted that the rejected step size h_2^* has never actually been applied to the simulated TDF time but has only existed locally within the ODE solver to calculate an output sample that was rejected and reverted. Thus, its temporary choice has not been visible to any connected modules and does not influence the further simulation in any other way than its utilization to calculate the valid, smaller step size h_2 .

After the execution of the TCI has finished, all remaining modules of the static schedule are processed for the current cluster execution time (here: t_1). When the cluster execution is completed, the chosen step size is finally applied and the TDF time advances to $t_{TDF}=t_S=t_2$ such

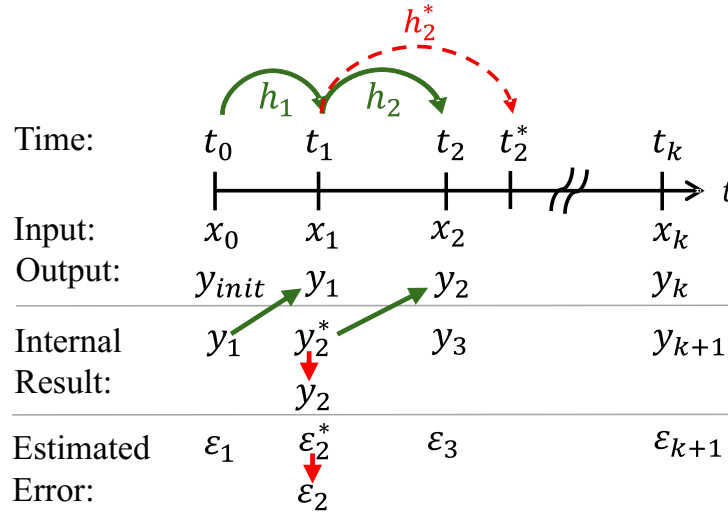


FIGURE 5.3: Timing diagram of the PRECALC method. Each sample is calculated one step in advance. If a solution is rejected, a smaller step size is chosen to increase the accuracy. © 2025 IEEE.

that TDF and solver time become realigned. If an ELN/LSF cluster acts as TCI, the complete variable-step solving scheme is invisible for the user and handled internally using framework objects. However, if a TDF module acts as TCI, the implementation presented in this work follows the standard DTDF scheme and requires to explicitly set the new value for the timestep within the *change_attributes()* call. The trial-and-error loop of the variable-step solver is handled during the processing call. Figure 5.4 illustrates the internal calculations of the TCI and the mapping to the standard TDF methods.

The implementation details and user interface of the newly introduced features are further explained in section 5.7.

5.3.2 Method 2: SEGMENT

In order to calculate a single sample in advance, it is necessary to make a commitment to the forthcoming step size. Such a restriction limits the spontaneity in the reaction to unforeseen events. The subsequent section provides a comprehensive examination of the respective scenarios and their associated impacts. In this section, an alternative integration scheme is presented, which calculates each sample at the targeted point in time. Irrespective of the specific format of the TCI, its execution is part of the static schedule. As it cannot be assumed that the TCI will always be the first instance within the schedule, it is possible that several TDF modules may have already been evaluated when the TCI is called. Therefore, the current TDF time is already fixed, which in turn means that the step size from the last to the current point in time is fixed as well. Consequently, the step size cannot be changed arbitrarily without the need for global backtracking. As this option is beyond the scope of this study, an alternative solution must be identified to address the scenario in which the solution of the ODE does not meet the accuracy constraints.

The scheme is once more explained at the example of a TCI with input port x and output port y and its timing diagram is given in Fig. 5.5. In this scenario, the calculation of output sample y_2 is performed at TDF time t_2 (and not t_1), which has been reached by a step size h_2 that has been estimated to be the optimal one. If the estimation is successful and the local estimated error remains below the threshold, the solution is accepted and a new step size for the upcoming step is evaluated. However, if the timestep estimation has failed and the local error exceeds the threshold, it is not feasible to reduce the step size by shifting t_2 into the past.

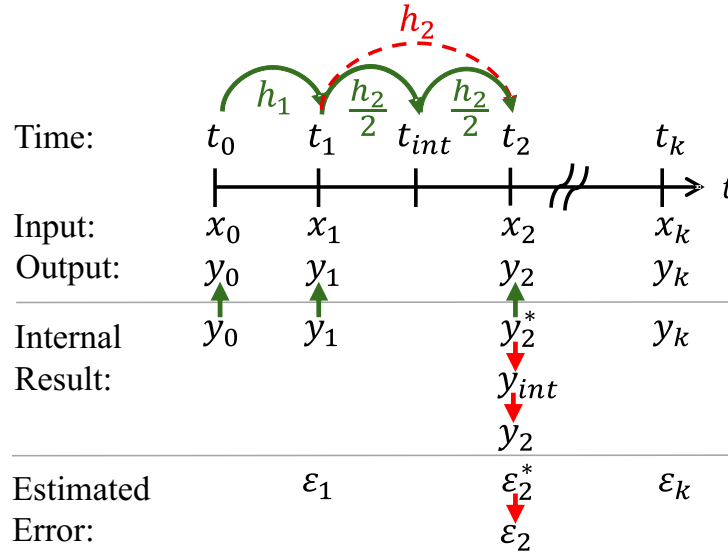


FIGURE 5.5: Timing diagram of the SEGMENT approach. Each sample is calculated in-time. If a solution is rejected, one or multiple supporting points are calculated to increase the accuracy. © 2025 IEEE.

Instead, the overall step size is kept unchanged, but it is divided into multiple segments. In the given example, the initial solution y_2^* yields an estimated error $\epsilon_2^* > 1$ and is rejected. As a result, the step size h_2 is divided into two segments ($N = 2$). The rejected solution and the solver time are locally backtracked. Afterwards, the ODE is solved again, this time with a reduced step size h_2/N . This yields an intermediate result y_{int} which pertains to the time point $t_1 + h_2/N < t_2$. Starting from this time and value, the next segment is calculated. As no dedicated value for the input at this point in time (here: $t_1 + h_2/2$) is available, the input is estimated via linear interpolation between x_1 and x_2 . In the example given in Fig. 5.5, the second step already recovers the solver time to meet the TDF time t_2 . If further segments are employed, the procedure is repeated until the entire interval has been traversed. Information pertaining to the supporting points is only accessible within the TCI and its associated solver instance; it is not accessible to other modules within the cluster. After the calculation of the new sample has been completed, the error estimation is repeated based on the new solution to ascertain whether the segmentation has indeed enhanced the accuracy. In the given example, the target is met ($\epsilon_2 \leq 1$), and the solution y_2 is accepted and written to the output. If the accuracy constraints are still not met, the procedure is repeated with an increased number of segments.

The number of segments selected is contingent upon the degree to which the estimated error exceeds the target. Considering a classical initial value problem, the local truncation error scales with h^p when the solver is of order $p - 1$. In this case, it was sufficient to use

$$N = \lceil \sqrt[p]{\epsilon} \rceil \quad (5.20)$$

segments, with $\lceil \cdot \rceil$ denoting an upper Gaussian bracket, to reduce the error below the targeted threshold. However, the impact of the input and its unknown values at the intermediate points increases the actual error. If the input is simply interpolated between the two surrounding values, its error scales proportionally to the step size. To account for this effect, Eq. (5.20) is modified by the exclusion of the root such that

$$N = \lceil \epsilon \rceil \quad (5.21)$$

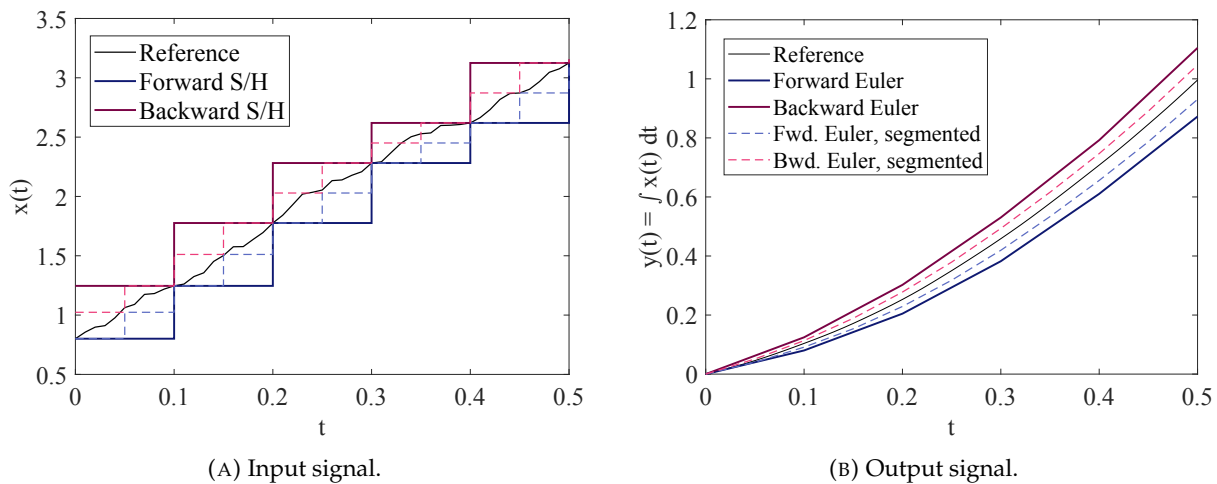


FIGURE 5.6: An input reference signal $x(t)$ is sampled with a sample-and-hold scheme. Adding more samples by splitting each interval into two segments and interpolating the value increases the approximation accuracy. The output of the differential equation $\dot{y}(t)$ becomes closer to the reference using the segmented inputs.

is utilized. This adds a robust safety margin, particularly for higher-order solvers, and reduces the computational cost to calculate N . Figure 5.6 illustrates the positive effect of the segmentation on the error. The transient input is depicted in Fig. 5.6a. The staircase plots demonstrate the manner in which the input is interpreted by a forward or backward Euler solver with step size $h = 0.1$. It results in a forward or backward sample-and-hold scheme. The dotted lines illustrate the interpretation of the input when each step is divided into two segments. The corresponding values are based on a linear interpolation and thus do not precisely align with the reference. Nevertheless, if the time grid is sufficiently fine, the interpolation still enhances the accuracy compared to the pure sample-and-hold scheme. The differential equation under test is a basic integrator such that

$$\dot{y}(t) = x(t). \quad (5.22)$$

The results presented in Fig. 5.6b demonstrate that the segmentation improves the accuracy of the solver. Nonetheless, if the input signal is insufficiently sampled and varies strongly between samples, increasing the number of segments does not necessarily increase the accuracy, as the enhancement is limited by the degree of matching between the interpolation and the actual transient behavior. To avoid numerical instabilities and lengthy simulations, the algorithm is equipped with an upper limit for the number of segments. If a discontinuity occurs within the input signal, the solver might thus be unable to meet the requisite accuracy constraints at that specific point. In this case, the algorithm issues a warning and retains the solution obtained with the maximum number of segments.

Figure 5.7 depicts the sequence diagram of the SEGMENT method at the example of a TDF module as TCI. The integration into the TDF member functions is similar to the PRECALC method (see Fig. 5.4). Again, the solver locally iterates within the processing call until a valid solution is found. In contrast to the PRECALC approach, no delay is assigned to the output port as each sample is calculated at its targeted point in time. The estimation of the next timestep is done based on the initial estimated error. Otherwise, the upcoming step size would most likely yield inaccurate results as well.

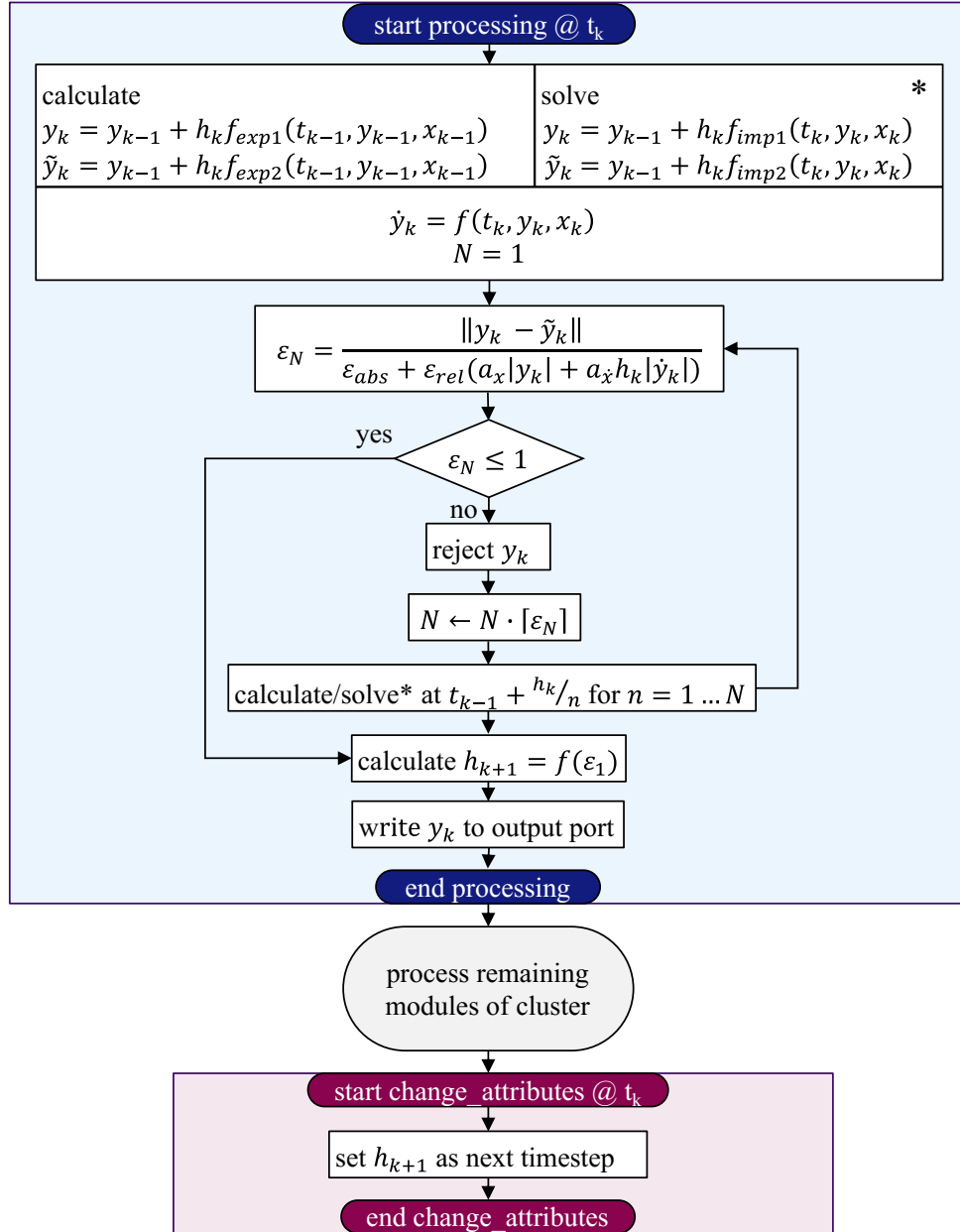


FIGURE 5.7: Sequence diagram of the SEGMENT method at the example of a TDF module as TCI. The calculations are given for both explicit solvers (f_{exp*}) and implicit ones (f_{imp*}). The iterations of the variable-step solver are executed locally within the processing call of the TCI increasing the number of segments for the given time interval until the accuracy constraints are met. When a valid sample has been obtained, the cluster execution proceeds until the schedule has been fully executed. The next step size is estimated based on the initial error ($N = 1$) and set as TDF timestep using the *change_attributes()* method.

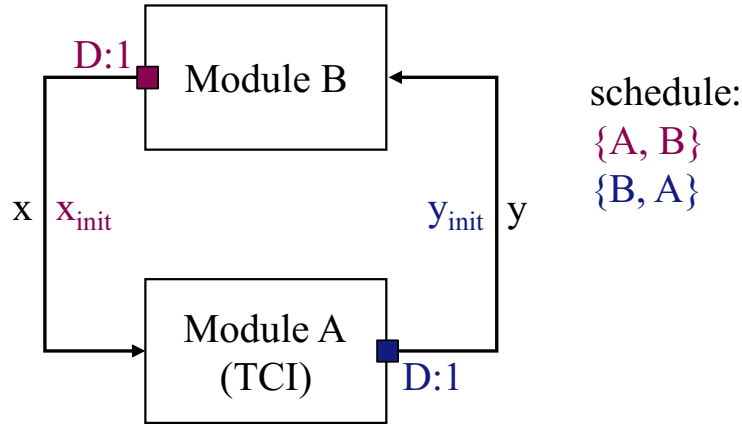


FIGURE 5.8: Feedback loop consisting of two modules A and B. Module A acts as the TCI. A delay element is necessary to break the algebraic loop. The two colors mark the possible positions and the resulting static schedules.

5.4 Interference with Other Language Features

Two distinct methodologies have been presented for integrating variable-step ODE solvers into the execution semantics of a SystemC AMS TDF cluster and synchronizing it with the static scheduler, respectively. Both of them interfere with other language features of SystemC AMS. In this section, these interferences are identified and explained in order to derive a guideline which will assist the users in determining which approach is most suited to their specific simulation models.

5.4.1 Feedback Loops

One intrinsic limitation of SystemC AMS is its inability to deal with algebraic loops. In order to ensure the schedulability of a TDF cluster, it is necessary to introduce delay elements to break all feedback loops. In this regard, the PRECALC approach offers the inherent benefit of requiring a delay anyway, which allows for the straightforward creation of feedback loops without the necessity of additional delay injection. Opposing to that, the SEGMENT approach does not expect a port delay and faces disturbances inside a feedback loop. As the k -th output y_k is calculated using step size h_k , delaying the sample to $t_{k+1} = t_k + h_{k+1}$ potentially distorts the signal because the time interval between the newly calculated sample and the previous one is h_{k+1} instead of h_k . If both step sizes differ, the signal's slope is distorted. To illustrate the effect, the reader may consider an exemplary loop consisting of two modules A and B, as shown in Fig. 5.8. Module A realizes the TCI and includes a linear transfer function

$$H(s) = \frac{1}{s} \quad (5.23)$$

representing a simple integrator. Transforming it from the Laplace to the time domain yields

$$\dot{y} = x \quad (5.24)$$

for y being the output and x being the input of the TCI. Module B implements the simple analytical formula

$$x = 2y + 1. \quad (5.25)$$

Inserting Eq. (5.25) into Eq. (5.24) yields the differential equation

$$\dot{y} = 2y + 1, \quad (5.26)$$

which can be solved analytically to

$$y = 0.5(e^{2t} - 1), \quad (5.27)$$

assuming an initial condition $y_0 = 0$. However, SystemC AMS does not provide any algorithms to resolve the algebraic loop and hence does not recognize this solution. For the sake of simplicity, the solver is assumed to use a backward Euler step only, i.e. the TCI calculates

$$y_k = y_{k-1} + h_k \dot{y}_k \stackrel{(5.24)}{=} y_{k-1} + h_k x_k \quad (5.28)$$

in each step whereas module B is interpreted as

$$x_k = 2y_k + 1. \quad (5.29)$$

Adding a unit delay to module A in order to break the algebraic loop, Eq. (5.28) transforms to

$$y_{k+1} = y_k + h_k x_k \quad (5.30)$$

and the resulting cluster schedule is {B, A}. Approximating the differential for y yields

$$\frac{y_{k+1} - y_k}{h_{k+1}} = \frac{y_k + h_k x_k - y_k}{h_{k+1}} = x_k \frac{h_k}{h_{k+1}} \quad (5.31)$$

such that the distortion gets visible through the undesired factor $\frac{h_k}{h_{k+1}}$. A naive approach to solve this issue would be to apply the output delay to another module within the loop. However, this does not resolve the distortion; rather, it merely relocates the source within the loop. To demonstrate the effect, the delay is applied to module B instead. The new resulting schedule is {A, B} and the equations transform to

$$y_k = y_{k-1} + h_k x_k \quad (5.32)$$

$$x_{k+1} = 2y_k + 1. \quad (5.33)$$

The calculation of the discrete derivative of y looks fine in the first place:

$$\frac{y_k - y_{k-1}}{h_k} = \frac{y_{k-1} + h_k x_k - y_{k-1}}{h_k} = x_k. \quad (5.34)$$

However, doing the same for signal x ,

$$\frac{x_{k+1} - x_k}{h_{k+1}} = \frac{2y_k + 1 - (2y_{k-1} + 1)}{h_{k+1}} = \frac{2(y_k - y_{k-1})}{h_{k+1}} \stackrel{(5.34)}{=} 2x_k \frac{h_k}{h_{k+1}}, \quad (5.35)$$

it gets obvious that the source of distortion has only been shifted. As the exact solution for the derivative using the chain rule calculates to

$$\dot{x} = \frac{\partial x}{\partial y} \frac{\partial y}{\partial t} = 2x, \quad (5.36)$$

the term $\frac{h_k}{h_{k+1}}$ in the approximation again represents the distortion factor. As the solution of y directly depends on the distorted input x , y becomes distorted as well. Figure 5.9 shows the

corresponding simulation results for the given example. The in-time calculation as performed by the SEGMENT approach creates distortions for both positions of the delay. When the delay is assigned to the output of the TCI and the PRECALC method is chosen, the output becomes smooth. Please note that it comes close to but does not precisely fit the exact behavior derived in Eq. (5.27) (black line). This could only be achieved solving the algebraic loop manually by replacing it with a single module implementing the analytical solution directly.

5.4.2 Reaction onto Incoming Events

Upon the introduction of DTDF semantics into SystemC AMS, a method *request_next_activation()* was added that allows to trigger a module (and therewith the complete cluster) in response to an incoming event from the DE domain. If this feature is jointly used with the method *set_max_timestep()*, which sets the maximum time interval between two successive module activations, the respective module is triggered either by an incoming event or latest after the maximum timestep has elapsed.

The SEGMENT approach aligns well with this feature. The timestep selected by the variable-step solver is merely a proposal and has not yet been used. It can therefore be taken as a parameter for the call to *set_max_timestep()*. If a reactivation is triggered by an event prior to reaching the selected maximum time interval, the resulting timestep will be smaller than that chosen by the solver. This increases the accuracy of the current step and enhances the probability that the accuracy constraints will be met in the first iteration, obviating the need to recalculate with multiple segments. When the TCI is called, the SEGMENT approach notices the actual step size and continues its calculations based on the new value. In conclusion, the approach reacts to dynamic activations without any problems.

In contrast, the PRECALC method has already utilized the next step size for its calculations when it is ultimately applied to the TDF cluster. If an incoming event reduces the actual timestep, the precalculated sample will occur at an incorrect point in time. As illustrated in Fig. 5.10, this has a discernible impact on the solution. Given that rollbacks of the TDF cluster are forbidden, it is not possible to prevent the incorrect sample, as it has already been calculated in the previous step and becomes available at the output as soon as the cluster is reactivated. Nevertheless, a **step recovery** can be conducted in order to prevent the accumulation of these errors over the course of the simulation. The step recovery works as follows. The solver recognizes the actual timestep that has been taken from the previous to the current point in time. It compares it with the step size that has been proposed by the ODE solver. If both step sizes match, the solver continues normally. Otherwise, the difference between the proposed timestep h_{prop} and the applied timestep h_{app} is calculated. Please note that the difference $\Delta h = h_{prop} - h_{app}$ is always non-negative as incoming events cannot postpone a cluster activation. The solver skips the calculation of a new output sample. Instead, it writes the same value(s) as before to the output port(s). The difference $\Delta h > 0$ is proposed as next timestep. As a result, the sample that was unintentionally shifted by the incoming event is reapplied at its targeted point in time. Figure 5.10 illustrates the effect. Using the step recovery, the global solution remains unaffected and only the local disturbance event retains its visibility. As incoming events are typically sparse and not uniformly distributed, it is not anticipated that this approach will generate any colored noise.

It should be noted that the step recovery approach is only meaningful in cases where the incoming event does not directly influence the ODE. If the input to the ODE is suddenly changed or the incoming event induces coefficient changes of the ODE itself, the solution will inevitably be disturbed. Consequently, the precalculated sample will become invalid at the targeted point in time as well. In such a scenario, it is inadvisable to employ step recovery; instead, the solver

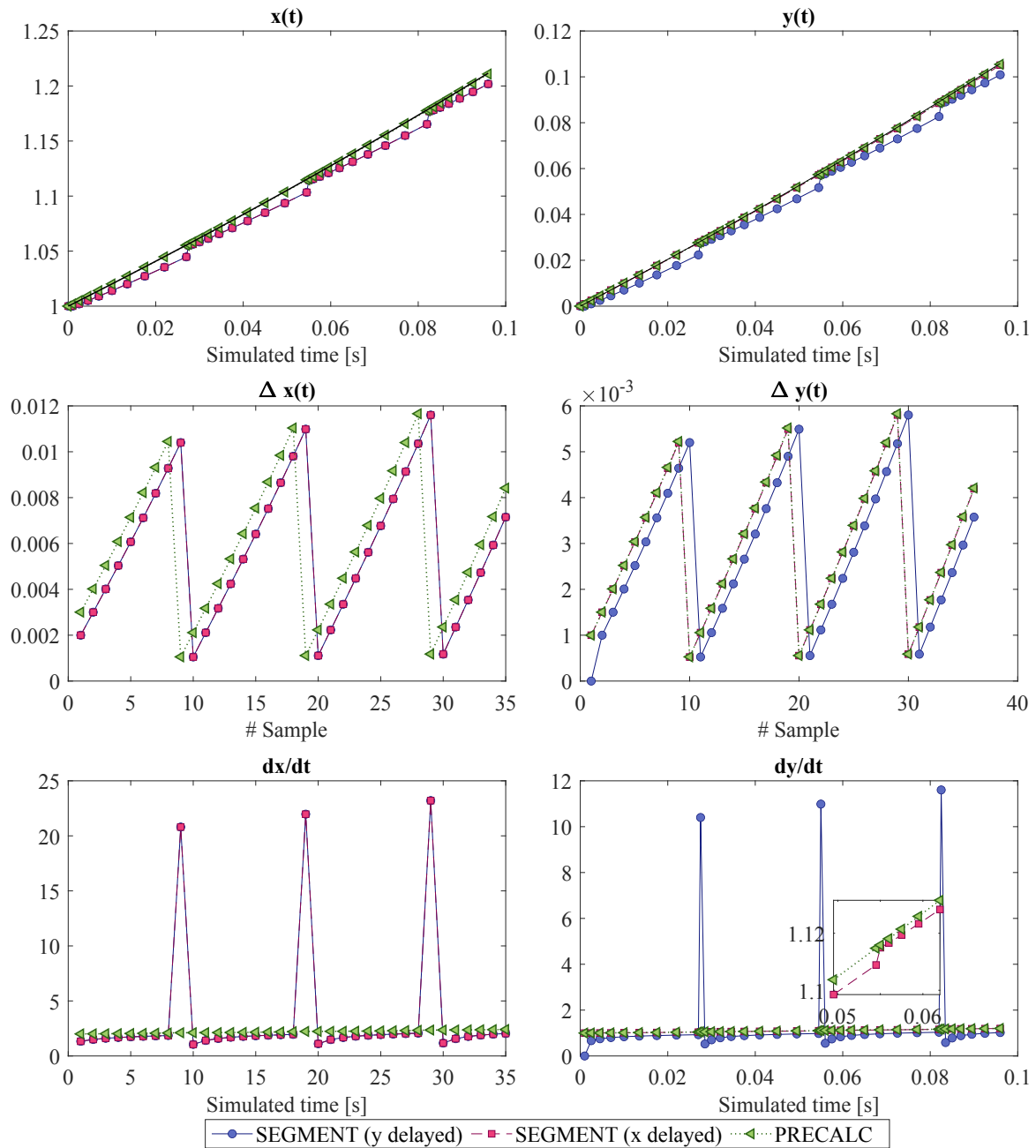


FIGURE 5.9: Simulation results for the structure shown in Fig. 5.8 implementing the equations $\dot{y} = x$ and $x = 2y + 1$. Signal y is distorted independent of the placement of the delay if the SEGMENT approach is used.

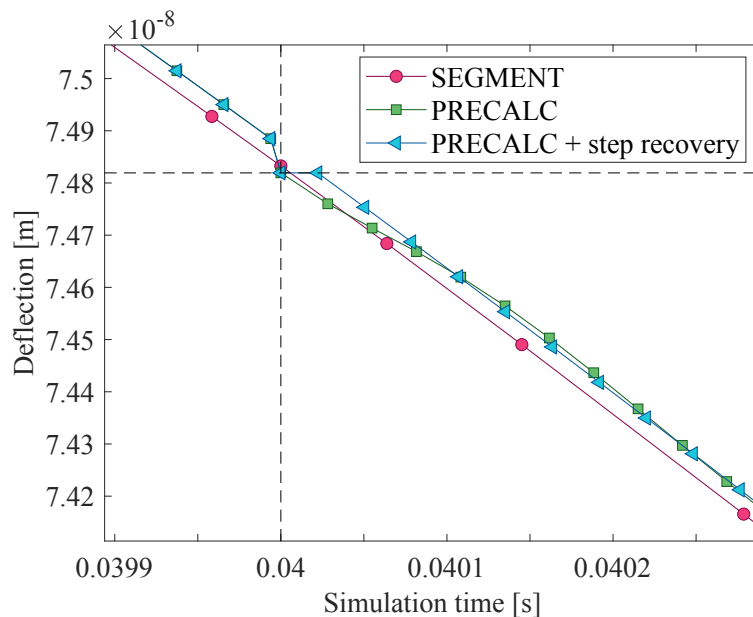


FIGURE 5.10: Impact of a cluster reactivation due to an incoming event at $t = 0.04$ s. The SEGMENT approach adapts nicely to the changed timestep. The PRECALC approach is distorted as the applied step size does not fit the estimated one which was used for calculating the sample. The step recovery applies the calculated output again at its intended point in time. Still, the single disturbance cannot be avoided without TDF rollbacks. © 2025 IEEE.

will use a minimal step size to facilitate an expedient reaction to the novel system configuration. Nevertheless, it is not an implausible scenario that the event does not directly influence the ODE. One illustrative example is the simulation of a MEMS sensor, wherein the mechanical component is represented by a differential equation and configuration changes within the frontend trigger events. The MEMS itself is thus not affected by the event; rather, the event affects the interpretation of the MEMS output, for instance because the full-scale range has been switched.

5.4.3 Multiple Time Controlling Instances

Up to this point, the underlying assumption was made that the system is controlled by a single variable-step solver operating on a single ODE system. However, it is not uncommon for heterogeneous virtual prototypes of cyber-physical systems to contain multiple sensors or other physical components and thus to incorporate multiple ODE systems. In the context of smart sensor development, a common example is an inertial measurement unit, which contains an accelerometer and a gyroscope. Both incorporate a MEMS sensing element that is represented by a set of ODEs. Furthermore, it is uncertain which of them is more critical in terms of accuracy because this depends highly on the physical stimuli applied to them. In this special case, it is straightforward to divide the model into two distinct clusters, one representing the accelerometer and the other the gyroscope. However, in other instances, such as modeling the drive loop of a gyroscope, this approach is less viable. In this example, the mechanics are once more represented by an ODE, and an additional one is introduced by the phase-locked loop (PLL) that is used to stabilize the drive oscillation at its resonance frequency. During the start-up phase, the control signals of the PLL change rapidly and their accuracy constraints are harsh to ensure a proper stabilization of the oscillation. When a stable operation has been reached, the step

size is then primarily constrained by the oscillation itself, as the input signal of the controller exhibits only minor fluctuations.

In this and other cases, it might be of interest to nominate multiple modules/clusters as time controlling instances. Nevertheless, the connected TDF cluster can only inherit a single step size. Thus, the scheduler will throw an error if the TDF cluster is not schedulable due to the specification of multiple inconsistent timesteps. It is therefore required to synchronize the two (or more) TCIs properly.

One possible solution to this problem is to estimate the step sizes for all TCIs and then take the minimum value from the resulting set. This approach ensures that accuracy constraints are considered for all differential equation systems within the model. The actual timestep observed by a TCI is either aligned with its estimation or reduced in accordance with the constraints imposed on another solver instance. Therefore, the concept is analogous to that of the previous section. Solely the root cause of the diminished step size has changed from a DE event to a second solver instance. Consequently, the SEGMENT approach is favorable again and natively works well with this approach. The PRECALC method, however, is not an appropriate choice for this concept. In the case of sparse incoming events, a step recovery is a reasonable method for handling local disturbances. In contrast, the use of the PRECALC method for multiple TCIs will result in the perpetual creation of disturbances in the ODE solutions, which cannot be disregarded.

5.5 Timestep Quantization: Overcoming the LU Decomposition Bottleneck

This section focuses on a bottleneck inherent to implicit solvers, a phenomenon that has been reported in the context of SystemC AMS multiple times in literature [107, 143, 144] and became visible in the experiments of the previous chapter (see Fig. 4.7 on page 61). Although variable-step solvers are generally known to be more efficient than fixed-step solvers in a multitude of applications, modifying the step size in implicit solvers requires caution. In the following, a mathematical formulation of the problem statement is derived. As a starting point, the ordinary differential equation system is represented in state-space notation as

$$\dot{s}(t) = As(t) + Bx(t) \quad (5.37)$$

$$y(t) = Cs(t) + Dx(t) \quad (5.38)$$

with $s(t)$ being the state vector, $x(t)$ the input vector, and $y(t)$ the output vector. The system characteristics are given by the state matrix A , the input matrix B , the output matrix C and the feedthrough matrix D . As the proof-of-concept implementation of SystemC AMS is based on the (implicit) trapezoidal method, we focus on this solver scheme to derive the problem statement. The local differential under the assumption of step size h_k using the trapezoidal method yields

$$\dot{s}_k = 2 \cdot \frac{s_k - s_{k-1}}{h_k} - \frac{s_{k-1} - s_{k-2}}{h_{k-1}}, \quad (5.39)$$

such that Eq. (5.37) transforms to

$$2 \cdot \frac{s_k - s_{k-1}}{h_k} - \frac{s_{k-1} - s_{k-2}}{h_{k-1}} = As_k + Bx_k \quad (5.40)$$

$$\iff \left(\frac{2}{h_k} I - A \right) s_k = \frac{2}{h_k} s_{k-1} + \frac{1}{h_{k-1}} (s_{k-1} - s_{k-2}) + Bx_k, \quad (5.41)$$

with I denoting the identity matrix [144]. To solve the equation for s_k , it is necessary to apply the inverse matrix Z_k^{-1} defining

$$Z_k := \frac{2}{h_k} I - A. \quad (5.42)$$

As the matrix depends on the timestep, each alteration of the timestep requires a recalculation of the inverse. Within SystemC AMS, a lower-upper (LU) decomposition is used as computationally most expensive step in this regard. An LU decomposition of an $n \times n$ -matrix using Gaussian elimination and partial pivoting can be computed in $\mathcal{O}(n^3)$. However, optimized solutions are available for sparse matrices which basically scale with the number of non-zero elements instead of the matrix dimension [152]. As a consequence, the impact of the recurring LU decompositions depends highly on the characteristics of the ODE's state matrix.

Figure 5.11 illustrates the conceptual correlation between the simulation performance and the system characteristics. The ordinate represents the ratio of the wall-clock times for a fixed-step and a variable-step approach, which reflects the speed-up achieved through the use of a variable-step solver. If certain accuracy requirements are set, the requisite timestep to meet these may fluctuate throughout the simulation due to the evolving dynamics in the transient signals. The variable-step solver is capable of adapting to these changes and varies the timestep accordingly. Contrarily, the fixed-step solver is required to use the minimum of these timesteps in order to meet the requirements at all times during the simulation. Therefore, the profound assumption is made that the number of cluster executions N is higher for the fixed-step solver than for the variable-step solver. As a result, the theoretical maximum of the performance gain achievable by a variable-step solver is equal to the ratio between these numbers of cluster executions. This theoretical maximum is contingent upon the assumption that the computational cost for a single output sample of the ODE is equivalent for both solver approaches. If Δc describes the difference of the computational cost, it holds

$$\lim_{\Delta c \rightarrow 0} \left(\frac{t_{wall,fix}}{t_{wall,var}} \right) = \frac{N_{fix}}{N_{var}}, \quad (5.43)$$

with $t_{wall,*}$ denoting the simulation time in terms of wall-clock times. In practice, this maximum cannot be achieved, as the variable-step solving introduces additional computational costs for the second solver method, the local error estimation, the timestep calculation, and potentially a certain amount of rollbacks. Nevertheless, these effects are typically small. For explicit solvers, this indicates that the performance enhancement can become close to the maximum value presented. In the case of implicit solvers, however, the recurring LU decompositions must be taken into account as an additional source of overhead. It should be noted that the computational cost associated with each LU decomposition has been set to an arbitrary but fixed value for the graphs shown. The dark blue curve in Fig. 5.11 represents a common implicit variable-step solver behavior. If the connected TDF cluster is sufficiently small and/or computationally inexpensive, the recurring LU decompositions will dominate the simulation time. In this case, the variable timesteps even deteriorate the simulation performance. In the case of larger and/or computationally more expensive clusters, the impact of the LU decomposition is comparably small. Therefore, the simulation performance asymptotically approaches the theoretical maximum.

This section introduces a measure to increase the simulation performance for systems with large ODEs and significant impact of the LU decomposition. The concept is referred to as **timestep quantization**. The objective is to decouple the amount of LU decompositions from the number of simulation steps. To this end, the number of possible step sizes is limited to

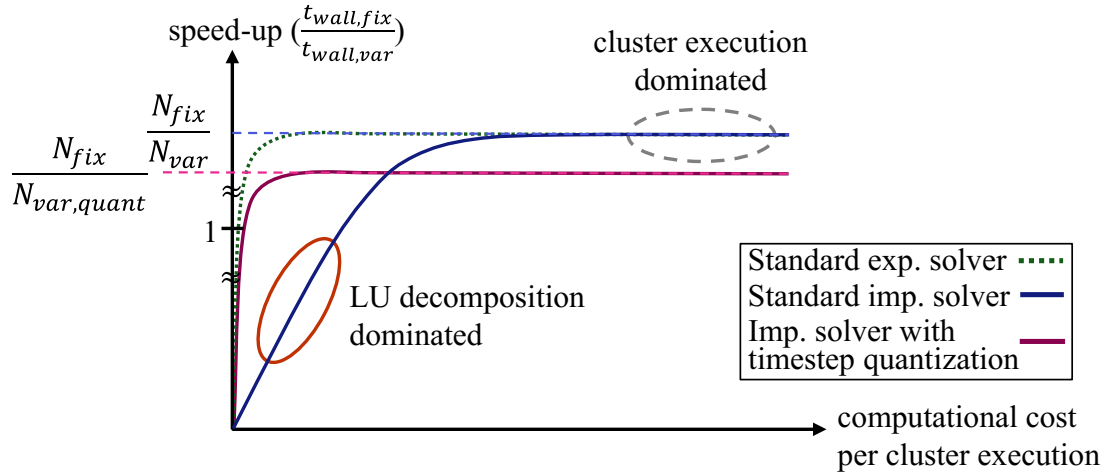


FIGURE 5.11: Dependency of the simulation performance of variable-step solvers on the relation between computational cost per cluster execution and per LU decomposition. If the impact of the decompositions is negligible, the maximum speed-up compared to a fixed-step solver equals the relation of the number of cluster executions performed. © 2025 IEEE.

a finite amount such that the decomposed matrices can be stored and reused multiple times during simulation. A discrete grid of permitted timesteps is created for that purpose. Generally spoken, the solver specifies a set of valid timesteps

$$\mathcal{H} = \{h_1, h_2, \dots, h_n\}, n \in \mathbb{N} \quad (5.44)$$

with each timestep being within a certain range

$$h_{min} \leq h_i \leq h_{max} \quad \forall i. \quad (5.45)$$

As a consequence, the number of different matrices Z_k is limited to the cardinality of $\mathcal{H}$, referred to as $H := |\mathcal{H}|$. The existence of a minimum ensures that the solver time cannot get stuck at a discontinuity or rapid change of the input. The maximum timestep correlates with the reaction speed onto increasing signal gradients. The assumption is made that the coefficients of the state matrix are constant. This ensures that equal timesteps are tantamount to equal matrices Z_k . Semantically, this limits the approach to linear systems.

During simulation, the solver step size is permitted to switch between the elements of $\mathcal{H}$ arbitrarily often. If a step size is employed for the first time, an LU decomposition is conducted in order to solve the equation system. The decomposed matrix is stored. When the same step size occurs another time during the simulation, the decomposed matrix is reused. Consequently, the number of LU decompositions performed within a single simulation run is limited to H and decoupled from the number of simulation steps.

The purple line in Fig. 5.11 illustrates the qualitative impact of the concept on simulation performance. As the number of LU decompositions is no longer proportional to the number of transient steps but rather to the number of valid step sizes, its impact is significantly reduced. Therefore, the simulation performance converges to the theoretical maximum even when the computational cost of the TDF cluster is relatively small. The fast convergence is paid by a reduction of the maximum achievable speed-up. In order to satisfy the accuracy constraints of the ODE solver and prevent a high number of rollbacks, the quantized timestep must not exceed the estimated one. In each simulation step, the step size is estimated as previously described (see Eq. (5.15)) and aligned with the grid of valid timesteps defined by $\mathcal{H}$ in a subsequent step. Thereby, the mapping from the estimated step size h^* to the valid step size h can be formulated

as a constrained optimization problem:

$$\begin{aligned} & \max h \\ & \text{s.t. } h \in \mathcal{H} \\ & h \leq h^*. \end{aligned} \tag{5.46}$$

For the proof of concept, the implementation of the solution is kept simple. An ordered list is used to store the valid step sizes. The mapping of a given timestep h^* is performed iterating through the list as long as $h \leq h^*$ holds. This allows to easily find the maximum timestep that fulfills the constraints as defined in Eq. (5.46). The step can further be optimized using a binary search or a directed search starting from the last discrete step size because the majority of step size variations are usually small. All in all, the quantization increases the overall number of steps that are conducted in a simulation compared to a classical variable-step solver. The extent of this increase depends strongly on the granularity of the chosen grid. The choice of a suitable cardinality depends on the span between h_{min} and h_{max} to be covered and on the simulated time. The more overall transient steps are performed, the higher the gain of inserting additional valid step sizes.

The remaining question to be addressed is that of how to select the individual elements of the set. A natural approach would be to place them equally distant from one another. The relative change from a step size h_i to its neighboring step size h_{i+1} then derives to

$$\frac{|h_{i+1} - h_i|}{h_i} = \frac{\Delta}{h_i} \tag{5.47}$$

with Δ being a constant. The equation reveals that the relative change between two neighbouring points in the grid depends on h_i , such that it becomes larger in the region of small step sizes. As the relative change between two step sizes is limited according to Eq. (5.15) to avoid numerical instabilities, taking linearly equidistant step sizes does not work well. Instead, the author of the thesis at hand proposes to use a **logarithmically equidistant** grid, such that

$$\log_{10}(h_{i+1}) = \log_{10}(h_i) + \log_{10}(\Delta) \tag{5.48}$$

with Δ again being a constant representing the granularity of the grid. As a result, the relative change between two neighbouring step sizes is independent of the actual element taken. For a given minimum and maximum step size and a defined number of valid steps, the logarithmic distance derives to

$$\log_{10}(\Delta) = \frac{|\log_{10}(h_{max}) - \log_{10}(h_{min})|}{H - 1}. \tag{5.49}$$

Please note that the granularity must be chosen such that $\Delta \leq 5$ when following Eq. (5.15). Otherwise, the step size is stuck as the timestep change is limited to a factor of five, and even a single quantization step cannot be overcome.

One more word to the scope and limitations of this approach. First of all, it works well with the PRECALC approach but is not realistically suited for the SEGMENT approach as the latter would still introduce new (invalid) step sizes when splitting a valid one into segments. Secondly, the approach is explicitly intended for models with large ODE systems that are highly dominated by the computational cost of the LU decompositions. The sole constraint imposed is the immutability of the state matrix. Nevertheless, it is theoretically feasible to further mitigate this limitation, provided that the set of state matrices employed within a simulation is

finite. In this case, the matrix Z_k is a function of both, the state of A and the current timestep. In order to utilize the aforementioned approach in an efficacious manner, it is imperative to identify a computationally inexpensive solution for determining whether the solver matrix should be treated as a new entity or mapped to its pre-registered equivalent. Comparing each entry of the matrix to each stored instance creates a significant overhead and is hence not desirable. However, if for instance only a limited number of entries are affected by changes, comparing these entries may be a sufficient method for identification. This may be realized by registering a callback to write operations on the state matrix that tracks entry changes. Although these general considerations are not deepened in the thesis at hand, one broadly applicable case is presented in the following section. It is of particular interest in the context of sensor simulation.

5.6 ELN Switching Efficiency

The electrical linear network MoC of SystemC AMS provides primitives for switches that are controlled via a boolean TDF or DE signal. Each switch is represented internally as a resistance that switches between two distinct values r_{on} and r_{off} with $r_{on} \ll r_{off}$. The resistive value constitutes one entry in the state matrix, whereby each switching action triggers a matrix change. Given that a switch implements only two valid states (on/off), the number of potential switching states for a circuit with S switches is limited to 2^S and is hence finite. In practice, some of the switches are often correlated, which reduces the total number of switching states. Moreover, in many cases, the number of valid states is especially smaller than the number of switching actions as valid states occur multiple times, often even periodically. One type of circuit that adheres to this specification is a switched-capacitor circuit. Its switches are controlled by clocked signals with periodic states. The high level of switching activity induces recurring LU decompositions. Again, the number of decompositions is proportional to the number of executed simulation steps, resulting in a considerable performance bottleneck.

The thesis at hand presents a method to overcome this bottleneck. It is referred to as **switch monitoring**. Basically spoken, the switching states are monitored to identify recurring states and reuse the respective decomposed matrices. Figure 5.12 illustrates the concept. Whenever an ELN switch (`sca_eln::sca_tdf::sca_rswitch`, `sca_eln::sca_de::sca_rswitch`) is instantiated, it registers itself at the **switch monitor** (see Fig. 5.12a). The monitor assigns each switch a unique index in ascending order based on its registering sequence. The index indicates which bit of the **state mask** is associated with the current switch. Thereby, an open switch is represented as zero and a closed switch as one. Consequently, the switching state mask uniquely characterizes the current switching state. In the illustrated example, all switches are initially open, thus the initial state mask is zero.

Each switch registers a callback to its switching function that updates the assigned bit of the state mask. Once all switches have been processed for the current timestep, the ODE solver instance is invoked to solve the resulting equation system. At this juncture, the switch monitor is triggered to assess the novelty of the current switching state mask. This entails a search for an equivalent within the mask storage. If the mask is new, the solver performs an LU decomposition for the matrix Z_k . Subsequently, the current state mask is appended to the mask storage and linked to the decomposed matrix. Otherwise, the existing mask in the storage directly links to a previously stored decomposition result and the computationally expensive step is circumvented.

Figure 5.12b illustrates an exemplary simulation sequence. In the initial state, all switches are open resulting in an all-zero initial mask. In the first processing step, switch S_3 is turned on. The new switching state mask thus toggles its third bit. As the state occurs for the first time, it is added to the mask storage. The solver conducts its LU decomposition, stores the result and links it to the corresponding mask. In the second cluster execution, switch S_3 is turned off

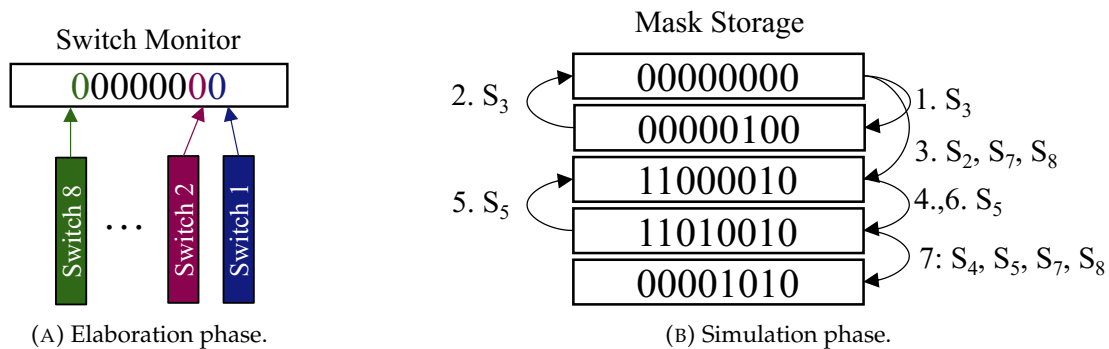


FIGURE 5.12: Switch monitoring concept. (A) Each switch registers itself at the switch monitor and is mapped to a unique bit of the state mask. (B) Each switching action changes the current state mask. At each cluster timestep, the current mask is stored in the mask matrix if not yet available. Corresponding decomposed matrices are stored and linked via a common index. © 2025 IEEE.

again and the state mask is changed back to the all-zero state. As this has already been used beforehand, no new mask is added to the mask storage and no LU decomposition is conducted. During the third timestep, multiple switches are triggered. Although they are internally processed sequentially, they semantically happen in the same timestep. Thus, the new mask is only evaluated after all switching activities have been completed and the intermediate results are not added to the mask storage. Only the final mask including three non-zero bits is processed and only one LU decomposition is performed.

Similar to the timestep quantization in section 5.5, the switch monitoring decouples the number of LU decompositions from the number of executed simulation steps. This time, the number of conducted LU decompositions becomes equal to the number of occurring switching states. Although it is theoretically feasible to combine this approach with the quantized timesteps from the previous section, it is currently implemented for fixed-step simulations only. The presented approach is intended to be used for circuits with high switching activity and recurring switching states. These circuits usually include clocked control signals for the switches and are thus not applicable to variable-step solvers anyway in order not to disturb the switching frequencies. It should be noted that the limitation of having a constant state matrix A , that was introduced in the previous section, is still valid except for the entries related to switches. Within the context of ELN models, this especially forbids variable passive elements (resistors, capacitors, inductors) as these can induce arbitrary modifications in the state matrix. Controlled sources however do not influence the state matrix but only the source vector dynamically and are thus applicable. The work of Rizzi et al. presents an approach to create an equivalent model that replaces variable resistors by controlled sources [143]. In case a few variable passive elements shall be used, this is an effective approach to circumvent this limitation.

5.7 Implementation & User Interface

All approaches presented in this chapter have been embedded into the proof-of-concept implementation of SystemC AMS provided by ACCELLERA [14]. This section describes the implementation concepts, with a particular focus on the user interface. For further details on the implementation, please refer to Appendix A.

First, the integration of variable-step solvers for TDF modules is discussed. The author of the thesis at hand intends to keep the user interface as close as possible to the known structure to ensure an intuitive use for experienced SystemC AMS modelers. TDF modules provide

three distinct primitives that trigger the instantiation of an ODE solver, two of them specifying linear transfer functions, either in nominator-denominator scheme (*sca_tdf::sca_ltf_nd*) or in pole-zero form (*sca_tdf::sca_ltf_zp*), and one specifying the ODE in terms of state-space equations (*sca_tdf::sca_ss*). All of them implement the same public member functions. The proposed extension adds multiple additional member functions to each of these classes:

set_stepper_type() Sets the stepper type of the underlying solver instance. Default is a fixed-step solver ("fixed"). Other options are the PRECALC approach ("precalc"), optionally with timestep quantization ("precalc_quant"), and the SEGMENT approach ("segment"). The stepper type shall be fixed throughout the entire simulation. If the method is called after the end of elaboration, an error is thrown.

set_stepper_params() Sets the parameters that specify the behavior of the selected stepper type. For variable-step solvers, this includes the absolute and relative error constraints (ϵ_{abs} , ϵ_{rel}), their weights (a_x , $a_{\dot{x}}$) as well as minimum and maximum step sizes (h_{min} , h_{max}). In the current implementation, the granularity Δ is stipulated. As the fixed-step approach does not require additional parameters, calling this method has no impact if the stepper type is set to "fixed". The stepper parameters shall be constant throughout the entire simulation. If the method is called after the end of elaboration, an error is thrown.

set_ode_solver() Sets the type of ODE solver used. The default setting is the standard SystemC AMS solver, which employs the trapezoidal method. If a variable-step size is used, the backward Euler method is used as second solver. Currently, one other option is implemented that uses (explicit) Runge-Kutta solvers of order four and five ("runge_kutta54"), also known as Dormand-Prince solver. If only one solver is required, the higher order solver is used. The solver type shall be fixed throughout the complete simulation. If the method is called after the end of elaboration, an error is thrown.

set_timestep_adaptive() Variable-step equivalent to the method *set_timestep()*. Instead of explicitly specifying a new step size, the step size calculated by the variable-step solver is applied to the module. Since the timestep is evaluated internally, no argument must be provided. If the method is called outside the *change_attributes()* environment, an error is thrown. The method also fails if the module has not set the flag *does_attribute_changes* or a fixed-step solver is used.

set_max_timestep_adaptive() Variable-step equivalent to the method *set_max_timestep()*. Instead of explicitly specifying a new step size, the step size calculated by the variable-step solver is applied as new maximum step. It may be used jointly with the method *request_next_activation()*. If this function is called outside the *change_attributes()* environment, an error is thrown. The method also fails if the module has not set the flag *does_attribute_changes* or a fixed-step solver is used.

get_estimated_error() Debug feature. Returns the current local estimated error of the underlying ODE solver in order to trace the solver's behavior.

The implementation of the explicit solver utilizes the open source library BOOST that provides classes for Runge-Kutta solvers [138]. The code snippet in listing 5.1 represents an exemplary TDF module that models an oscillator and employs a variable-step solver to illustrate the usage of the new member functions.

LISTING 5.1: TDF module of a simple oscillator acting as TCI.

```

SCA_TDF_MODULE(oscillator)
{
public:
    sca_tdf::sca_in<double> pos;
    sca_tdf::sca_out<double> force, dbg_eps;

    oscillator(){...}

private:
    sca_util::sca_vector<double> num, denom;
    sca_tdf::sca_ltf_nd ltf;

    void set_attributes()
    {
        ltf.set_stepper_type("precalc");

        // stepper params: eps_abs, eps_rel, ax, axp, hmin, hmax
        ltf.set_stepper_params(1e-5, 1e-3, 1.0, 1.0, 1e-9, 1e-3);

        does_attribute_changes(); // still required
        pos.set_delay(1); // sync timing for precalc approach
    }

    void initialize()
    {
        pos.initialize(0.0);

        // initialize ODE through numerator, denominator
        num(0) = ...
    }

    void processing()
    {
        double x = ltf(num, denom, force.read()); // same format as fixed-step

        pos.write(x);
        dbg_eps.write(ltf.get_estimated_error()); // for user reference only
    }

    void change_attributes()
    {
        ltf.set_timestep_adaptive();
    }
};

```

The SystemC AMS language reference manual states that ELN and LSF modules are only permitted to set their timesteps during elaboration [79]. They implement a method *set_timestep()* for this purpose that throws an error if called after the end of elaboration. The thesis at hand relaxes this restriction in order to facilitate variable-step solving with ELN/LSF clusters as the time controlling instance. Nevertheless, as the TCI is always represented by the entire cluster, individual modules are still prohibited from independently setting timesteps. Therefore, the existing primitives remain unaltered for the solution presented here. Instead, the thesis at hand introduces a new primitive for each of the two MoCs. The ELN version is denoted as *sca_eln::sca_solver_control* and its LSF equivalent is called *sca_lsf::sca_solver_control*, respectively.

As already indicated in Fig. 5.2b, this primitive must be connected to the cluster that shall become the TCI. Thereby, any node except the reference node may be used, and the solver control module does not modify the equation system. To ensure that the solver and stepper parameters remain static after the end of elaboration, they can only be set once within the constructor. For the ELN instance, two constructors are implemented, one for variable-step solvers and one for switch monitoring as these features are orthogonal to each other. If the constructor is called without any arguments, all parameters fall back to the default case that employs a standard fixed-step solver. The constructors for the ELN instance are given in listing 5.2.

If a variable-step solver is used, the solver control instance induces a similar effect as the method *set_timestep_adaptive()* in the TDF MoC. In case switch monitoring is enabled, the respective solver creates an instance of the newly introduced class *switch_monitor*. It includes three static attributes, one for the current mask, one for the complete mask storage (a vector of masks) and one indicator for the mask length or the number of registered switches, respectively. Each switch creates its own instance of the corresponding child class *switch_handler* which inherits the static attributes but additionally gets a unique index to identify which bit position refers to this switch. Additionally, it provides a method to update the current mask after a switching event has happened. Each switch registers a callback to this method. Please note that the current implementation limits the number of switches to 64 as the mask is internally represented by a 64-bit word but can easily be extended to more switches by splitting the complete mask into multiple 64-bit segments. The implementations of both the *switch_monitor* and the *switch_handler* are provided in Appendix A. All in all, the user simply connects an instance of the *sca_solver_control* module class to the desired cluster and sets the respective parameters in the constructor to enable variable-step solving or switch monitoring. The decision to handle the timestep control of TDF modules more explicitly via dedicated functions was taken to follow the current style of the language reference.

LISTING 5.2: Extract from the newly introduced primitive for the ELN MoC. It controls the solver settings of the cluster.

```
class sca_eln::sca_solver_control: public sca_eln::sca_module
{
public:
    sca_eln::sca_terminal p; // terminal to connect to the TCI cluster

    // constructors
    sca_solver_control(sc_core::sc_module_name, std::string stepper_type_="fixed",
                      std::string solver_type="trapez", double eps_abs_=1e-6, double eps_rel_=1e-3,
                      double ax_=1.0, double axp_=1.0, double h_min_=1e-12, double h_max_=1e-3);

    sca_solver_control(sc_core::sc_module_name, bool enable_switch_monitoring);
    ...
private:
    virtual void matrix_stamps(); // registers change_timestep as POST_SOLVE callback
    void change_timestep();
    ...
};
```

5.8 Experimental Results

In this section, the methods introduced above are applied to various case studies for evaluation. Three cases from the context of virtual sensor prototypes are shown to illustrate the various approaches that have been discussed.

5.8.1 Case 1: Abstract MEMS Accelerometer Model

The first case study concerns a highly abstract model of a MEMS accelerometer as shown in Fig. 5.13. This kind of model is applicable to early project stages, when implementation details are not yet specified. It enables the early and rapid qualitative evaluation of a sensor system. The model comprises a force calculation block, which derives the sum of all forces acting on the damped harmonic oscillator. Both the acceleration force and the electrostatic force are considered. The electrostatic force acting on a plate capacitor calculates to

$$F_{el} = \frac{dE}{dx} = \frac{d\left(\frac{1}{2}CV^2\right)}{dx} = \frac{1}{2} \frac{dC}{dx} V^2 \quad (5.50)$$

with

$$\frac{dC}{dx} = \frac{d\left(\frac{\epsilon_0 A}{d+x}\right)}{dx} = -\frac{\epsilon_0 A}{(d+x)^2} \approx \text{const. if } x \ll d. \quad (5.51)$$

Thereby, E denotes the energy stored within the capacitor that depends on the capacitance C and the voltage V across the plates. The deflection is specified as x and impacts the distance d between the plates with area A . The vacuum permittivity is given by ϵ_0 . If the deflection is sufficiently small compared to the plates' distance, the sensitivity dC/dx is sometimes assumed to be constant. If not, the current sensitivity is fed back to the force calculation block (see Fig. 5.13, dashed line), thereby establishing a feedback loop within the system. The dynamics of the MEMS movement are represented as damped harmonic oscillator with movable mass m , quality factor Q and resonance frequency ω_0 . In order to test the variable-step solver for both the TDF and LSF MoC, the oscillator is either represented as a linear transfer function instantiated within a TDF module or by a corresponding cluster of LSF primitives. The ELN and LSF MoC use the same underlying solver implementation such that it is sufficient to prove the correctness of the solver's behavior at the example of one of these MoCs. The stopper module (dotted lines) is not part of the model for now.

First of all, the behavior of the synchronization mechanisms is investigated. Figure 5.14 shows some exemplary simulation results for the accelerometer model. The sensitivity is considered as constant for this experiment. The synthetic stimulus is designed in a way that allows for the examination of the various influencing factors on the behavior. It starts with a decaying oscillation. The multiplication of a high-frequent oscillation and a slowly decaying function allows to distinguish the influences of the absolute and the relative tolerances. The sudden step at $t_{sim} = 0.02$ s challenges the robustness of the algorithms. Afterwards, the signal remains constant for a short period of time before it finally slowly decays. The experiment has been conducted on both variants of the MEMS implementation, a single TDF module with LTF or an LSF cluster.

In addition to the stimulus and the reaction of the MEMS, the timesteps and local estimated errors are also plotted. The results demonstrate that all four variants adhere to the same fundamental principles and, as a consequence, exhibit comparable behavioral patterns. The high-frequency oscillation results in continuous alterations to the timestep, due to the relative accuracy constraint. The influence of the absolute accuracy constraint is evident in the overall trend of an increase in the timestep for a reduction in the amplitude of the signal. The discontinuity in the stimulus causes a sudden and strong decrease in the timestep, which then recovers rapidly due to the constant signal. The timestep is constrained to a maximum value of $t_s = 0.1$ ms. Consequently, it cannot increase further and the local estimated error, which has been maintained at a value close to one, begins to decrease. For the TDF simulations, the local estimated error reaches zero in a single step, resulting in a discontinuity in the logarithmic

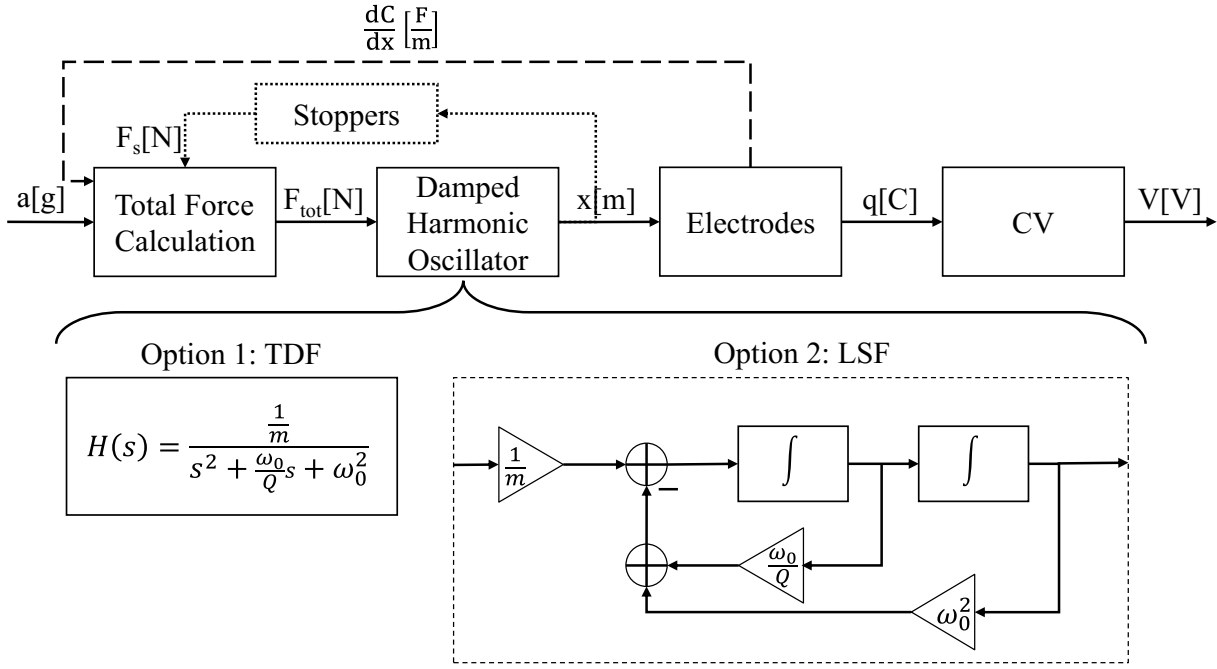


FIGURE 5.13: Schematic view of the accelerometer model used for the case study. The MEMS is represented as damped harmonic oscillator (PT2). It is either represented as TDF module with linear transfer function or composed from LSF primitives. A charge-to-voltage converter represents the readout circuitry.

© 2025 IEEE.

plot. The gradually decaying signal enables the use of larger timesteps. It is furthermore noticeable that the SEGMENT approach shows more severe downward peaks in the timestep. As the timestep cannot be reduced arbitrarily (for example, by 10%), but is instead at least halved whenever the local estimated error exceeds its threshold, the changes in timesteps appear more rapid and pronounced.

In a consecutive step, the sensitivity is modeled according to Eq. (5.51) and the feedback loop is incorporated into the model. The synthetic stimulus is applied to the system once more. Figure 5.15 shows the corresponding simulation results for the overshoot that occurs at the sudden jump of the input signal. The preceding results from the loop-free configuration are provided for reference. At first glance, the overall simulation results seem fine. The adaption of the sensitivity and hence the shifts in electrostatic force influence the amplitude of the deflection. The model's modification thus explains the two distinct stable levels that are reached by the traces. Moreover, exact timing of the overshoot varies a bit due to the relatively big timesteps that are conducted beforehand. This issue, that is inherent to variable-step solvers in general, will be further discussed in the subsequent experiment. In this setup, the focus is on the presence of signal distortions. As previously discussed in section 5.4.1, the SEGMENT approach creates signal distortions in presence of feedback loops because it does not consider the delay element that is required in that case. The zoom window of Fig. 5.15 reveals this issue. Without feedback loop, both synchronization schemes produce smooth outputs. The PRECALC approach incorporates a delay anyway and still produces smooth results when the loop is added. The SEGMENT approach however shows the expected distortions when two consecutive timesteps differ significantly (see especially $t_{sim} = 0.0206$ s).

Finally, the option to request cluster activations based on incoming events is evaluated based on the given accelerometer model. The testcase investigates the sensor's mechanical durability.

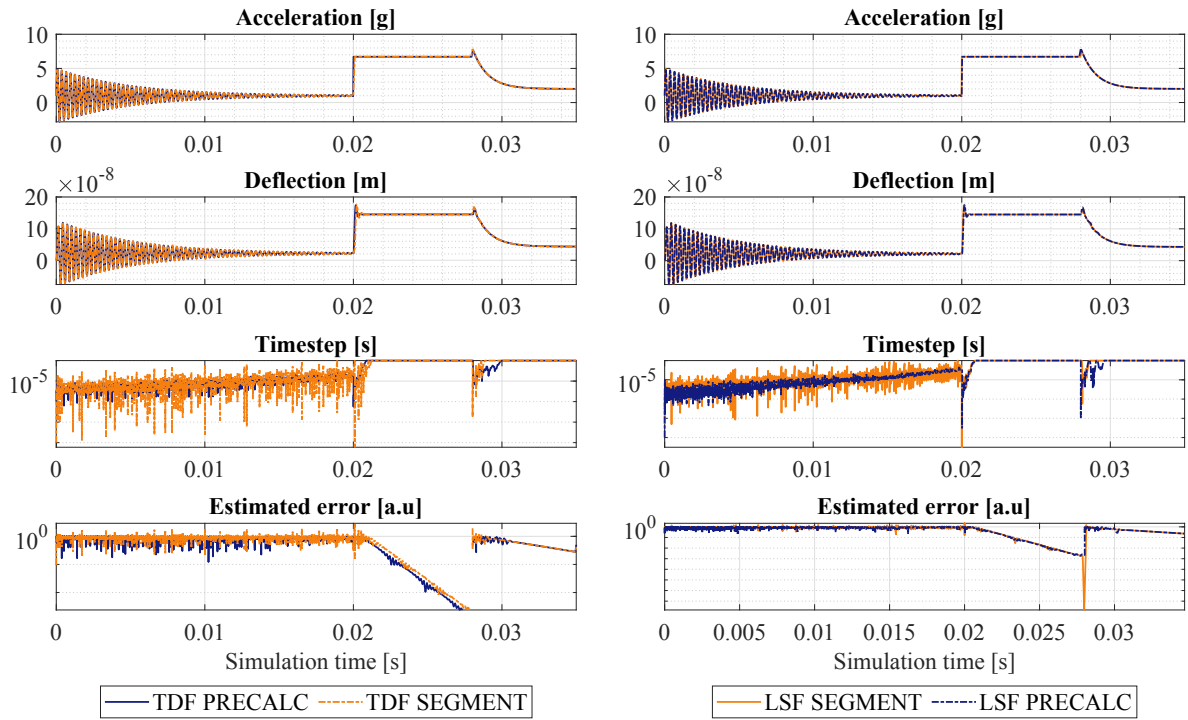


FIGURE 5.14: Simulation results for an accelerometer model subject to a synthetic stimulus. Besides the in- and output of the respective ODE, the timestep of the variable-step solver and its local estimated error are given. The experiment has been conducted using a TDF module (left) or an LSF cluster (right) as TCI. Both show similar results. © 2025 IEEE.

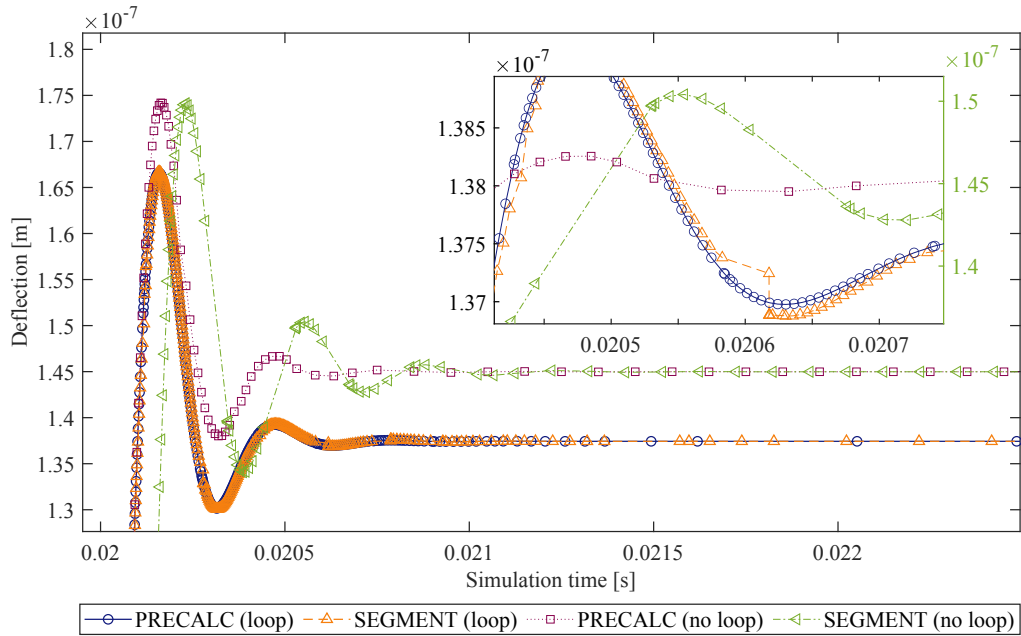


FIGURE 5.15: Simulation results for the overshoot caused by a sudden input change. The results from Fig. 5.14 are shown and compared to a version with variable sensitivity. The latter introduces a feedback loop that causes disturbances for the SEGMENT approach.

If a sensor is mounted inside a smartphone, it will most likely be exposed to mechanical shocks, for example when the user accidentally drops his or her phone. Therefore, a repeated shock test is conducted on the sensor, which serves as the test case in this investigation. The sensor is exposed to a series of mechanical shocks with high accelerations (here: 20 kg) but brief durations. Inbetween these shocks, the acceleration is zero. Thus far, the MEMS sensing element has been modeled simply via the differential equation for a damped harmonic oscillator. In reality, the deflection of the MEMS cannot reach arbitrarily high values as the suspended mass will hit the frame at some point. Usually, specific stoppers are incorporated into the design with the objective of preventing stiction and ensuring that the suspended mass returns to its original position. To account for this, the accelerometer model is extended by an additional stopper module (see Fig. 5.13, dotted lines) that adds a huge resilience when the suspended mass comes close to a deflection of $x = 10 \mu\text{m}$. As this again introduces a feedback loop, the PRECALC approach is preferable and hence primarily used for this experiment.

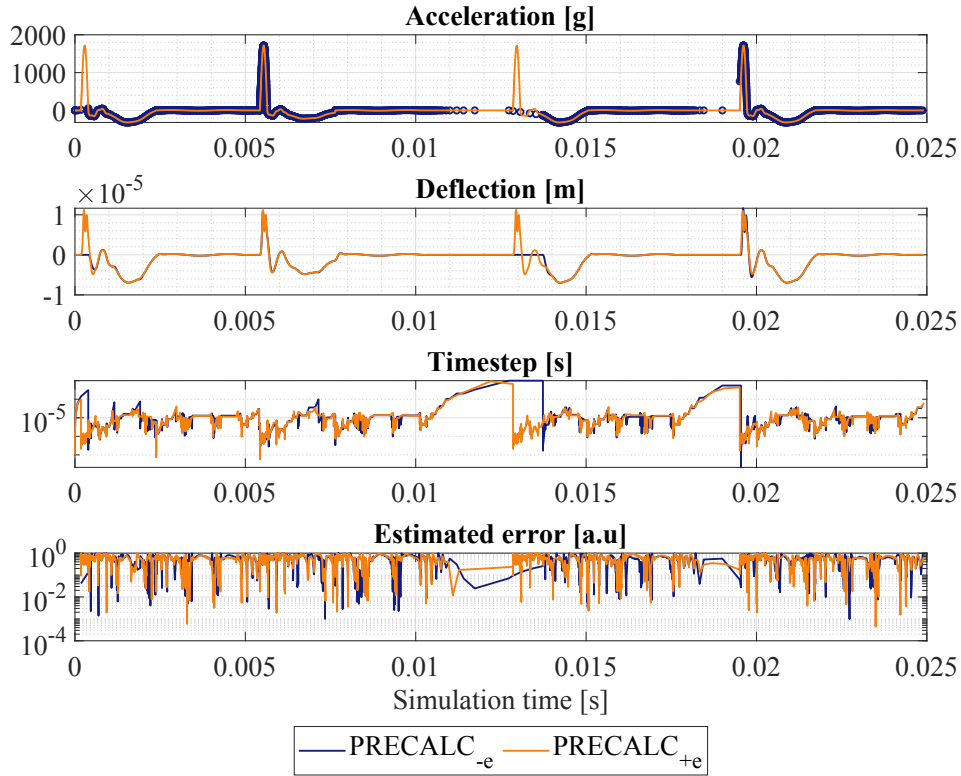
The results of the repeated shock test simulation are presented in Fig. 5.16. Two distinct variants of the PRECALC synchronization are employed. The first is the classical approach, which sets the timestep of the TCI adaptively. The second variant additionally listens to incoming events. The TCI does not set the timestep but only the maximum timestep as described in section 5.4.2. The module responsible for generating the stimulus is provided with an additional DE output, which indicates the beginning of a new shock. The TCI is sensitive to this event and triggers the module when a new shock is notified. If no event occurs until the maximum timestep is reached, the module is triggered by the timestep chosen by the variable-step solver as per usual practice. The stimulus creates four shocks within the presented simulation. Whether the solver reacts onto events is denoted with an index " $\pm e$ ".

Due to the sparsity of the stimulus, the timestep converges towards its maximum value between two shocks. Consequently, it is not possible to guarantee the accurate detection of the start time of a new shock without the use of event-triggering techniques. In the simulation presented, the combination of the selected maximum timestep and the narrowness of the stimulus causes the solver to fail to identify the first and third shock. The fourth shock is identified with a delay. The use of an event-based approach to signal the occurrence of a new shock effectively addresses this issue. As the single event-triggered module execution already faces a non-zero stimulus, the solver recognizes the abrupt transition and automatically reduces the timestep, thereby facilitating the accurate simulation of the shock.

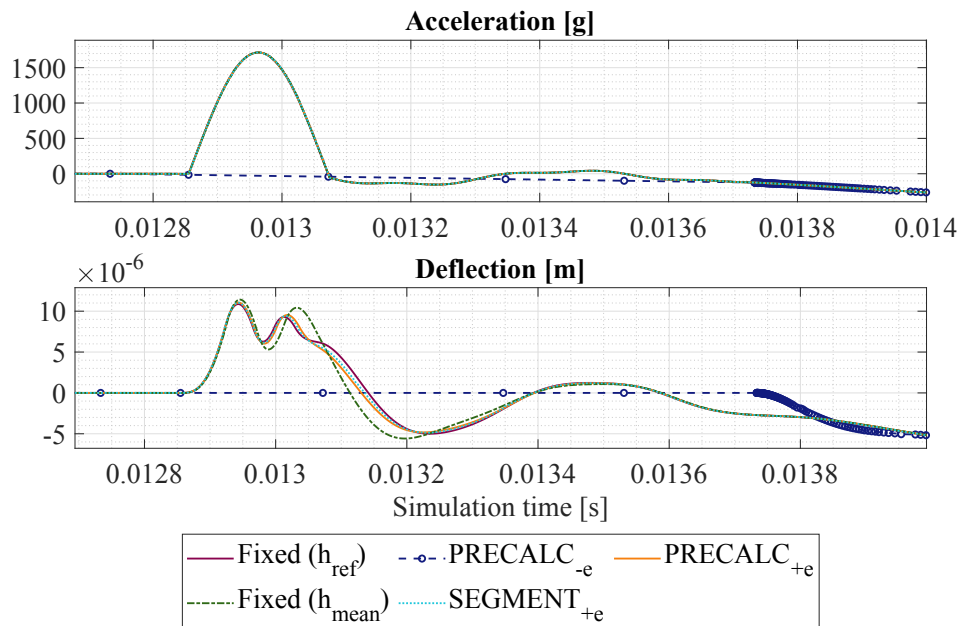
Figure 5.16b depicts a zoom onto the third shock. As previously stated, the PRECALC_{-e} trace completely misses the shock. In contrast, the PRECALC_{+e} run shows good accuracy compared to the reference run that has been conducted using a small, fixed step size ($h_{ref} = 0.01 \mu\text{s}$). In order to show the effectiveness of the variable-step solver, another fixed-step run has been conducted that uses the average step size compared to the PRECALC_{+e} approach, such that both runs conduct the same overall number of cluster executions. The mean step size calculates to $h_{mean} = 1.01 \mu\text{s}$ in this example. The fixed-step solver shows significant deviations compared to the reference. The variable-step solver is more effective as it concentrates most of the executions onto the sections with high-frequent changes and saves unnecessary calculations during the resting times. In section 5.4.2, it has been discussed that the reaction onto incoming events can cause signal distortions for the PRECALC synchronization method. However, in this specific case, the signal is flat when the event occurs. Consequently, the impact becomes negligible. To demonstrate that the same approach also works with the SEGMENT method, another run has been conducted. The results align closely to the reference.

5.8.2 Case 2: Reduced Order Modeling

The case presented above has focused on a simple representation of a MEMS that only considers the fundamental mode of each axis. The upcoming experiment utilizes a reduced order



(A) Overall simulation. Without event triggering mechanism, the first and the third shock are missed and the fourth one detected late.



(B) Zoom onto the third shock. Both variable-step solvers show good accuracy to the reference when adding the trigger.

FIGURE 5.16: Transient simulation of a repeated shock robustness test. The sparsity of the stimulus leads to big timesteps and can cause the solver to miss the short pulses. The issue is solved triggering the cluster based on an event that notifies the start of a new pulse (denoted with "+e"). Simulating the same number of overall steps that the PRECALC_{+e} approach conducts with a fixed-step solver ($h_{\text{mean}} = 1.01\mu\text{s}$) shows significant deviations from the reference that uses a fixed step size of $h_{\text{ref}} = 0.01\mu\text{s}$. © 2025 IEEE.

model to get a more detailed description of the electromechanical behavior. The objective of this experiment is to evaluate the timestep quantization approach. The one-dimensional linear transfer function used so far transforms into a multi-dimensional state-space equation system. The resulting input vector² $u(t)$, state vector $s(t)$ and output vector $y(t)$ representing the MEMS sensing element in modal space are

$$\vec{u}(t) = \begin{pmatrix} F_1 \\ \vdots \\ F_m \end{pmatrix} \quad \vec{s}(t) = \begin{pmatrix} x_1 \\ \vdots \\ x_m \\ \dot{x}_1 \\ \vdots \\ \dot{x}_m \end{pmatrix} \quad \vec{y}(t) = \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix} \quad (5.52)$$

with m denoting the number of modes, F_i , $i = 1 \dots m$ being the forces applied to each mode and x_i the resulting modal deflections. Rewriting the differential equation of a multi-modal damped harmonic oscillator according to the general form of SS equations

$$\dot{s}(t) = A_s \cdot s(t) + B_s \cdot u(t) \quad (5.53)$$

$$y(t) = C_s \cdot s(t) + D_s \cdot u(t) \quad (5.54)$$

yields exemplarily for $m = 5$ modes

$$A_s = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ & & -K_m & & & & & -D_m & & \end{pmatrix} = \begin{pmatrix} \mathbf{0} & I_5 \\ -K_m & -D_m \end{pmatrix} \in \mathbb{R}^{2m \times 2m} \quad (5.55)$$

$$B_s = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ I_5 \end{pmatrix} \in \mathbb{R}^{2m \times m} \quad (5.56)$$

$$C_s = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} I_5 & \mathbf{0} \end{pmatrix} \in \mathbb{R}^{m \times 2m} \quad (5.57)$$

$$D_s = \mathbf{0} \in \mathbb{R}^{m \times m}. \quad (5.58)$$

with K_m denoting the modal stiffness matrix and D_m denoting the modal damping matrix. If all modes are assumed as independent, K_m and D_m evolve to diagonal matrices. As a result, the

²Usually referred to as $x(t)$, here replaced by $u(t)$ to avoid confusion with the displacement x .

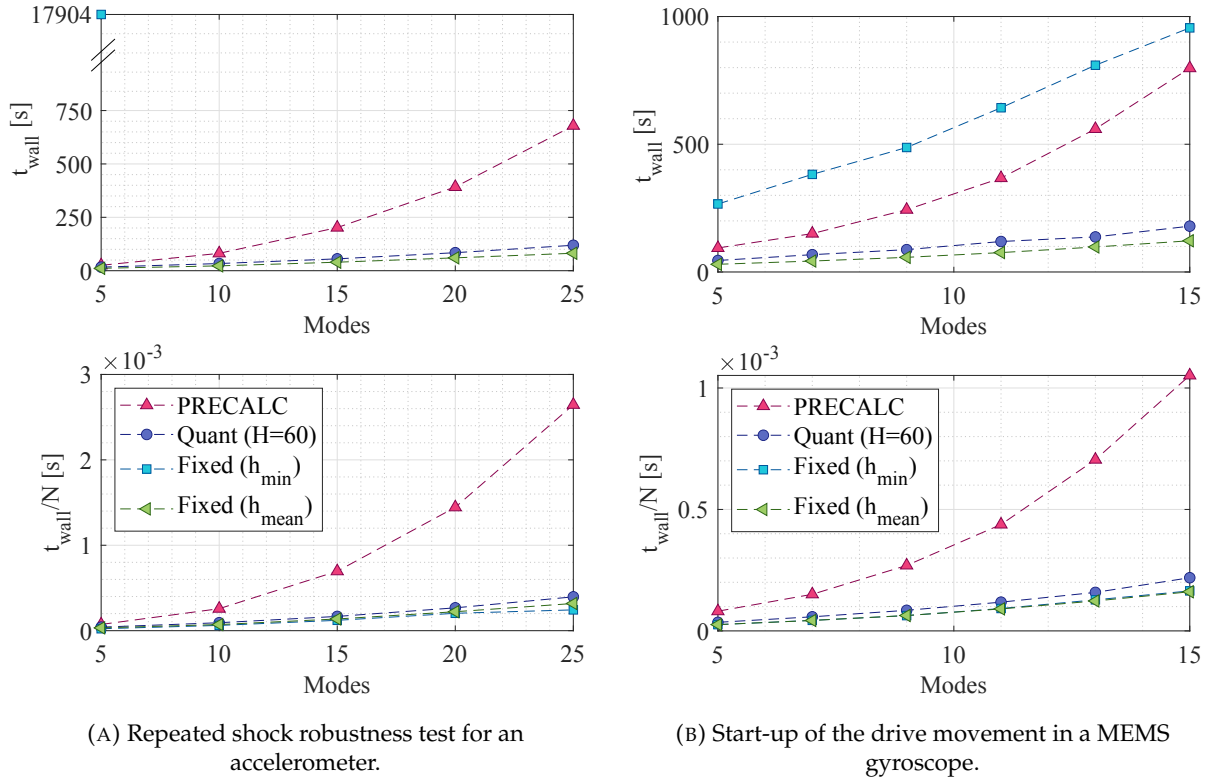


FIGURE 5.17: Wall-clock times of the simulations of reduced order models depending on the number of modes considered. The higher the number of modes, the more costly is each LU decomposition. The timestep quantization approach significantly decreases the simulation time. © 2025 IEEE.

state matrix is sparse and the LU decomposition is fast. In this experiment, couplings between the modes are taken into account such that $d_{i,j} \neq 0 \forall i, j$ which increases the computational cost noticeably.

Two different testcases are simulated. A ROM of an accelerometer is created and the shock robustness test from the previous section is repeated. Additionally, a ROM of a MEMS gyroscope is considered to simulate the start-up of the drive oscillation. Both testcases are run multiple times utilizing various solver approaches and numbers of modes. The wall-clock times are measured in order to visualize the impact of the LU decompositions. The simulation results are given in Fig. 5.17. The PRECALC based variable-step solver is simulated first and taken as a reference for the subsequent runs. The minimum and average step sizes are extracted and reapplied in the consecutive fixed-step runs. Moreover, the PRECALC approach is rerun with activated timestep quantization. The upper plots illustrate the absolute simulation times in terms of wall-clock times for the simulation runs plotted against the number of modes considered. The bottom plots depict the wall-clock times normalized to the overall number of cluster executions. The experimental results substantiate the aforementioned claims for the presented approaches:

1. In all cases, the variable-step solver exhibits superior performance in terms of runtime compared to the fixed-step solver with a minimum timestep. As the minimum timestep would approximately be required to meet the same accuracy constraints with the fixed-step solver that were applied to the variable-step run, this demonstrates the ability of variable-step solvers to enhance the simulation performance of SystemC AMS.

2. The absolute value of the speed-up factor is contingent upon the specific case in question. The more diverse the dynamics of the system, the higher the gain. The drive oscillation of the gyroscope causes a narrow range for the timestep, which results in minimal performance gain of the variable-step solver. As the overall accuracy is better with the fixed minimum step size, the PRECALC approach offers no discernible benefit in the test case illustrated in Fig. 5.17b. In contrast, the fixed-step run for the shock robustness test takes approximately five hours for five modes and more than two days for 25 modes (not shown in the plot for the sake of readability). In this case, the variable-step approach is highly advantageous, as it significantly reduces the simulation time to a few minutes. Doing so, many debug iterations could be done in a single day whereas the same procedure may take weeks to complete when using the fixed-step setup.
3. The timestep quantization effectively suppresses the negative impact of recurring LU decompositions. The wall-clock time of the PRECALC run increases heavily with the number of modes. The computational cost of an LU decomposition scales with $\mathcal{O}(m^3)$. The number of LU decompositions increases with the number of simulated steps, it scales with $\mathcal{O}(N)$ with N being the number of cluster executions because the setup is single-rate and the ODE solver is called once per cluster execution. The timestep quantization reduces the wall-clock time of each cluster execution close to the one exhibited by the fixed-step runs. As each quantized timestep is equal or smaller than it would have been without quantization, the overall number of cluster executions increases. Still, the total wall-clock times are close to the fixed-step runs with average timesteps. Hence, the impact of this effect is small and the efficacy of the method is demonstrated.

The previous point is further illustrated by Fig. 5.18, which plots the number of cluster executions for the PRECALC approach with and without timestep quantization in comparison to the fixed-step run with minimum step size. The number of valid step sizes has been chosen to be $H = 60$, spanning an overall range of four orders of magnitude. Consequently, the relative difference between neighboring valid step sizes calculates to

$$\log_{10}(\Delta) = \frac{4}{60 - 1} \implies \Delta = 1.17, \quad (5.59)$$

revealing a fine grid size. Hence, the number of cluster executions increases only slightly when activating the timestep quantization. However, using a fixed-step approach creates a huge amount of additional cluster executions severely degrading the simulation performance. The varying overall number of execution steps is caused by the changing dynamics of the ROM versions.

5.8.3 Case 3: Switched-Capacitor Charge-to-Voltage Converter

The final case study focuses on the switch monitoring presented in section 5.6. Switched-capacitor circuits are commonly used in sensor frontend architectures. Therefore, they serve as a good example to evaluate the effectiveness of the approach. A switched-capacitor charge-to-voltage converter is utilized as the device under test. It has been designed for a MEMS accelerometer frontend for the automotive market. The circuitry has been firstly presented by Lanniel et al. [153]. It consists of four main stages, namely a Gm-stage, an integrator, a hold stage and a feedback logic as shown in Fig. 5.19. In SystemC AMS, all stages are composed of ELN primitives and jointly form the ELN cluster. It includes a total of 43 switches. For further information on the circuitry including a complete schematic and detailed explanation of the functionality, the reader is referred to [153].

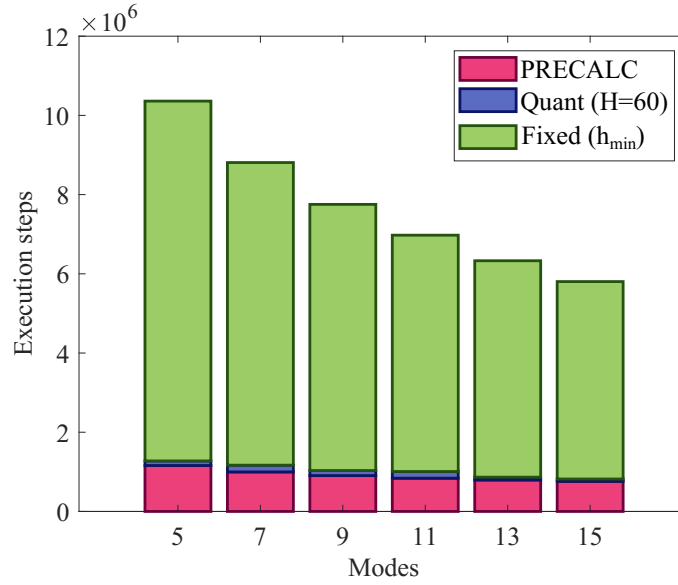


FIGURE 5.18: Stacked number of simulation steps for the PRECALC approach, its extension with timestep quantization and a fixed-step run with minimal timestep. © 2025 IEEE.

TABLE 5.1: Wall-clock times for the simulation of the CV converter for 1s simulated time and chopper frequency $f_{chop} = 4\text{MHz}$. The relative wall-clock time of the solver instances are additionally shown.

Setup	Equations	Overall wall-clock time* [s]	Solver share	Relative Speed [†]
Reference	98	6251.29 [75.16]	99.9%	1
Split Clusters	94 + 10	5412.85 [153.34]	98.5% + 1.3%	1.17
Switch Monitoring	94 + 10	116.89 [2.35]	54.7% + 38%	53.48

*average [standard deviation]

[†]Relative Speed = $(t_{\text{wall,reference}})/t_{\text{wall}}$

The set of phases required to control the switches of the CV converter are provided by a TDF module. Although 43 switches can theoretically create $2^{43} \approx 8.8 \cdot 10^{12}$ different switching states, the interdependent control signals create only 21 different states for a nominal execution. The resulting switching scheme is depicted in Fig. 5.20. Initially, all switches are open. Afterwards, ten distinct switching states are shown. The introduction of an additional high-frequency chopper clock results in a doubling of the number of these switching states, thus increasing the overall number of valid states to 21. In terms of switch monitoring, this implies that a maximum of 21 LU decompositions are required throughout the course of a complete (arbitrarily long) simulation. The solver is configured to utilize the presented primitive `sca_eln::sca_solver_control` which can be connected to any node within the cluster except the reference node (see Fig. 5.19). In this setup, the model of the MEMS sensing element is selected to be as abstract as possible. The MEMS is solely represented by a pair of variable capacitors. The ELN MoC provides primitives for variable capacitors that are controlled by TDF signals, which are fed by a TDF stimulus module in the experiment. This high level of abstraction of the MEMS is a common practice when testing a new circuit concept. However, it has already been discussed in section 5.6 that the state matrix must not be subject to any entry changes except the ones induced by switching actions. The variable capacitors violate this constraint. To resolve this conflict, the MEMS representation is split from the ELN cluster representing the CV converter. The overall setup

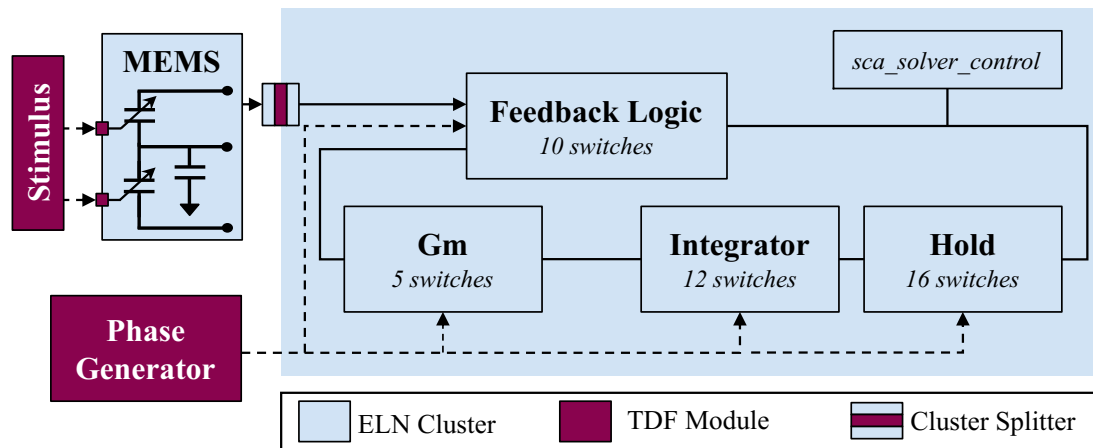


FIGURE 5.19: Schematic of the charge-to-voltage converter circuit. It implements a switched-capacitor design and includes 43 switches in total. © 2025 IEEE.

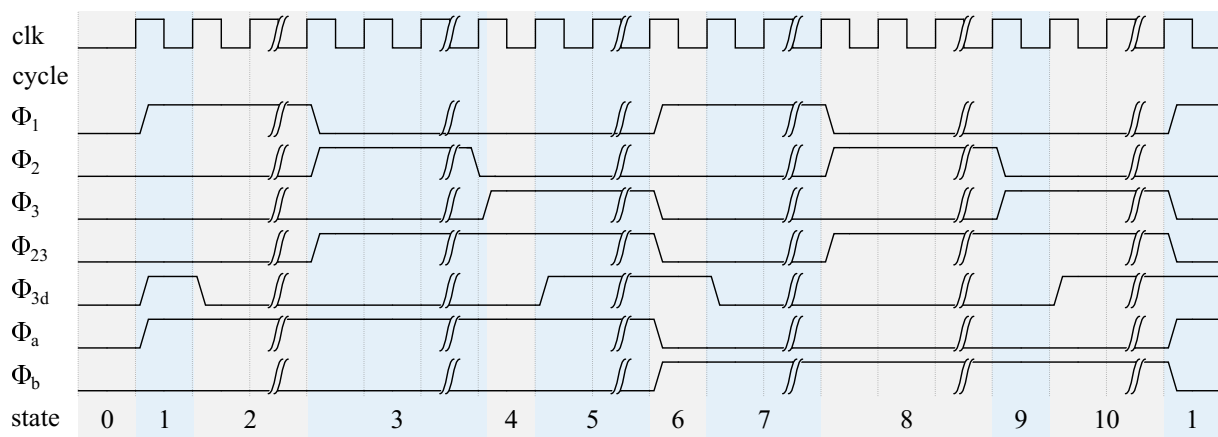


FIGURE 5.20: Switching pattern of the circuit under test. A set of phases Φ implements ten distinct switching states besides the initial state. The additional high-frequency chopper clock is not shown here but doubles the number of valid switching states. © 2025 IEEE.

hence contains one ELN cluster for the MEMS sensing element, a *cluster splitter* composed of an ELN-TDF converter and a TDF-ELN converter, and the second ELN cluster for the CV converter. In principle, the various MEMS models from the previous experiments can be readily connected to the CV converter for joint simulation without introducing any complications. The ELN-based setup is deliberately selected for the purpose of elucidating the constraints of the approach and exploring potential avenues for their mitigation.

Finally, the MEMS cluster is composed of three capacitors (two variable, one fixed) and three voltage sinks that connect the output nodes to the TDF MoC. This yields a total of six modules and ten equations for the cluster. The CV cluster is considerably more extensive, comprising 94 equations for 70 modules. Although the number of LU decompositions for the MEMS cluster scales with $\mathcal{O}(N)$ with N being the number of cluster executions, its impact is relatively small as the matrices are small. In contrast, the CV cluster realizes a state matrix that is close to 10x larger. Here, the switch monitoring provides support such that the number of LU decompositions scales with $\mathcal{O}(1)$ instead of $\mathcal{O}(N)$. The corresponding wall-clock times for the different simulation setups are presented in Table 5.1. They clearly demonstrate the applicability and effectiveness of the approach. The switch monitoring improves the simulation performance by

53x compared to the reference implementation where the MEMS and frontend cluster are directly connected (no intermediate TDF cluster) and still 45x compared to the separated clusters without switch monitoring.

5.9 Discussion

In the given chapter, three novel methodologies were presented with the objective of enhancing the solver efficiency of SystemC AMS. First, *variable-step ODE solvers* were integrated into the semantics of SystemC AMS. Thereby, two distinct synchronization methods were introduced that align the ODE solver time with the discrete time grid of the TDF MoC. The PRECALC approach calculates each sample of the ODE one step ahead of the TDF time such that a step can be backtracked and replaced by a smaller one before the TDF time advances. This approach is well suited in the presence of feedback loops as both, breaking an algebraic loop and realigning the precalculated value, require an output delay. In contrast, the SEGMENT synchronization scheme calculates each sample for the current point in time such that the timestep is already defined by the current TDF time. If a calculated solution does not meet the accuracy constraints and needs to be rolled back, the timestep is split into multiple segments. Thanks to the interpolation of the input vector, calculating each segment sequentially yields an increased accuracy. This approach is particularly advantageous when the variable-step cluster is to be additionally triggered by incoming events or when multiple TCIs are to be employed.

The conducted case studies have proven the applicability of variable-step solving in the context of MEMS sensor models. It was shown that the performance improvement depends heavily on the diversity of the system dynamics throughout the simulation. While the improvement kept small for the gyroscope simulation, it reached factors $>100x$ for the repeated shock robustness test with sparse stimuli.

It should be noted that the method can be applied to all models that contain an ODE that sufficiently reflects the dynamics of the system to control the overall step size. The proof of its effectiveness for other domains is therefore a subject to future work.

Furthermore, the results of the ROM simulation have demonstrated that varying the timestep in implicit solvers is associated with a considerable number of LU decompositions, which have the potential to impair performance. The second method, therefore, addresses this issue. The proposed solution is referred to as *timestep quantization*. It restricts the set of applicable step sizes to a finite number that are logarithmically equidistant between the minimum and maximum step size. Consequently, it is possible to store the decomposed matrices and reuse them when the same timestep is encountered again within a simulation. As a result, the overall number of LU decompositions is not proportional to the number of simulation steps but rather to the cardinality of the set of applicable step sizes, and thus significantly reduced. The case study with the reduced order models has demonstrated the efficacy of the approach. For a ROM with 25 modes (creating a state matrix of size 50×50 with sparse upper half), the wall clock time has been reduced by a factor of seven compared to the variable-step run without timestep quantization. It is anticipated that for larger matrices, such as those induced by a large ELN circuit, the speed-up will be even greater.

Switch monitoring was presented as the third novel method in this chapter. It is based on the same underlying idea of reusing decomposed matrices as the timestep quantization. However, the idea is transferred to a distinct use case. It is typically not feasible to apply variable-step solvers to ELN circuits with clock-based switching activities. Nevertheless, these circuits are subject to the same bottleneck associated with recurring matrix decompositions, given that each switching action results in a change to an entry of the state matrix. Consequently, a switch monitor was developed to record the occurrence of switching states, thereby facilitating the reuse of decomposed matrices. The switched-capacitor circuit under test demonstrated the

efficacy of this approach, exhibiting a reduction in wall-clock time by a factor of 45 relative to an equivalent run without switch monitoring.

Both approaches that reuse the solver matrices impose the limitation that the state matrix is not modified. Switching actions are the only exception to that rule. In theory, this restriction could be relaxed, provided that the number of occurring states is finite and significantly smaller than the number of conducted executions. However, the presented methods use lightweight mechanisms for comparing the current state with previously used ones, either based on the current step size or on the current switching state mask. If other state matrix alterations are to be permitted, it will be necessary to develop an alternative mechanism that enables the rapid identification of states. This topic will not be further addressed here but might be of interest for future work.

Chapter 6

Early Availability Through Automated Translation from MATLAB/Simulink

In the course of the discussion on the impact of a virtual sensor prototype on the development flow in chapter 3, the software engineers were identified as early users of the HVP, particularly given the need to deliver firmware for the read-only memory until ASIC tape-out. It is therefore of utmost importance to deliver an HVP that reflects the complete system as early as possible. At that stage, details on the actual implementation may not yet be available, and thus the initial model may not provide the necessary features to support trimming. However, it already provides a qualitatively correct representation of the entire signal chain.

MATLAB/Simulink has been identified as a tool that is widely adopted by analog hardware designers for preliminary exploration tests. This aligns with the conclusion of Pan et al. that MATLAB/Simulink is well suited for the partitioning phase where the system is broken down into its components, as its extensive libraries and interactivity allow for rapid modeling [136]. These models are typically sufficiently abstract to be reused for a first generation of a heterogeneous virtual prototype.

The following section on related work will present a series of existing co-simulation approaches and translation flows. While a number of translation tools are currently available for coupling the simulation of SystemC and Simulink models, they are usually based on the creation of a single black box model, which can then be used either within or jointly with the other simulator. It is important to note that a virtual prototype is not merely a tool for conducting simulations; it is also a means of fostering a more profound comprehension of the system and the interdependencies between its various modules and domains. Furthermore, it can serve an instrumental role in disseminating this knowledge across different development departments. The latter is merely supported by the provision of a black box model.

The thesis at hand proposes a new translation flow that reflects a structural generation of SystemC AMS models based on the structure of an initial Simulink model in order to ensure the early availability of an HVP and address *RQ3-Availability*. In this context, the term *structure* refers to the module breakdown of the system. The Simulink software provides the capability to organize blocks within subsystems, thereby enabling the creation of multiple hierarchical viewing layers. The objective of the presented approach is to preserve these layers.

In light of the various levels of analog modeling that have been introduced in the fundamentals on AMS modeling (see Ch. 2, Fig. 2.10 on page 22), it is probable that a Simulink model encompasses multiple of these layers. This chapter focuses on models that have been used for initial exploration tests. It is unlikely that they will extend below the analog module level, as implementation details are not yet available at that stage. The analog modules are then grouped into functional units and integrated into the system-level view. It seems thus reasonable to posit that a scenario comprising three levels of hierarchy would be a realistic representation of this approach. The structure of the model thereby effectively provides insights into the partitioning of the real hardware system.

The objective of the proposed flow is to maintain the visibility of these layers within the generated model, thereby facilitating more comprehensive system insights for users. Furthermore, the maintenance of the initial structure enhances the modularity of the generated model. As the project progresses, individual modules can easily be substituted with more sophisticated versions. The question of whether these refined modules have been natively written in SystemC AMS or translated is thereby irrelevant. Additionally, translating the analog module blocks independently enables the tracing of intermediate signals for the SystemC AMS model.

Following the presentation of related work in the subsequent section, the proposed flow will be elucidated in detail in section 6.2. Afterwards, section 6.3 addresses the model requirements and limitations, which are mainly induced by the use of the Embedded Coder, a tool for C++ code generation provided by MATHWORKS itself [154]. A modeling guideline is derived based on these constraints and the target and working principle of the translation flow. Then, implementation and experimental results are discussed in section 6.4. The chapter is brought to a close with the discussion presented in section 6.5. The entire development was carried out using MATLAB R2022b [155]. The chapter is based on a previous publication of the author of this work [P2a, P2b]¹.

6.1 Related Work

One variant of automated generation of SystemC AMS models has been addressed by Lora et al. [156]. The authors put forth a methodology for the bottom-up reuse of models. Verilog-A models are automatically translated into SystemC AMS ELN models. Thereby, the usual constraint of ELN models for linearity is imposed. An additional abstraction flow creates C++ code to support virtual prototyping. In a more holistic abstraction flow, Fraccaroli et al. present a method for automatically splitting a mixed-signal HDL model into digital and analog parts, abstracting both of them independently towards C++ models, and restoring their synchronization on the system level [157]. The analog abstraction is based on the work of Lora et al. [156]. A summary of existing bottom-up translation flows is provided in another work by Lora et al. [158]. While these approaches are promising for supporting the validation phase and tracing information gained during low-level modeling and implementation into system-level models, they do not meet the requirements of the thesis at hand. In this work, the focus is on early availability for frontloading activities. Therefore, the HVP should be available before low-level models exist.

Given that both SystemC and Simulink are employed in the context of system-level modeling, considerable effort has been dedicated to establishing a connection between the two. G6rgen et al. aim to translate VHDL code into Simulink blocks, employing SystemC as an intermediate model, which is subsequently wrapped for the utilization in Simulink [159]. The other direction is provided by the company MATHWORKS who market MATLAB/Simulink. They provide the Embedded Coder [154]. It facilitates the generation of C++ code from MATLAB code and/or Simulink models. Moreover, the tool is capable of directly generating SystemC models with a TLM interface [154]. The generation of signal interfaces or SystemC AMS models is not supported. Another export from Simulink has been presented by Zhang et al. [160]. They generate SystemMoC modules based on a module library for basic blocks. The necessity for a corresponding library block however limits the scalability. In conclusion, the aforementioned export options lack the capacity to facilitate signal-based AMS modeling. Kleen et al. have proposed a methodology for the export of Simulink models to SystemC AMS TDF models [161]. They use Java Emitter Templates to manipulate the code generation for that purpose. Although the source and destination of the procedure are similar to the approach presented here, their

¹See section 1.3 page 5 for the author's contribution to these publications.

approach is limited to the translation of complete model files. In contrast, the approach presented in the present thesis is considerably more flexible, as any subsystem can be translated to a SystemC AMS module, and hierarchical models can be created to connect them. Therefore, the overall structure of the original model is preserved. Solely the lowest level of hierarchy is translated using the Embedded Coder and creating a SystemC AMS wrapper. The upper hierarchy levels are recreated by composing hierarchical SystemC AMS modules that instantiate the lower-level modules and create the same interconnections as the source.

Besides the existing translation flows, a set of co-simulation approaches has been presented to interface SystemC and MATLAB/Simulink. Mendoza et al. create a server-client relationship where the server incorporates the SystemC model and connects to a client that is realized as S-function (a way of integrating C++ code into Simulink supported by the tool itself) and handles the Simulink models [162]. The connection between server and client is done in various fashions. They either share the same process or utilize TCP inter-process communication. Glass et al. connect Simulink and SysteMoC synchronizing both processes via a co-simulation interface [163]. The interface is implemented as S-function such that Simulink becomes the master of the simulation. Hylla et al. do not mix SystemC and Simulink models but verify Simulink blocks with the help of SystemC testbenches [164]. Again, Simulink is used as master and the SystemC blocks are integrated using S-functions. Furthermore, the CODIS tool supports the co-simulation of Simulink and SystemC with specific interfaces and three distinct synchronization schemes [165, 166]. The tool has been specifically designed for co-simulation purposes. It is able to create these interfaces in an automated fashion. However, all these approaches suffer from frequent context switches between C++ and the Simulink simulator. Mendoza et al. address the question of simulation performance by stating that the impact of the TCP connection on the wall-clock time is significant [162]. By creating C++ based analog models with SystemC AMS interface as presented in this work, any simulator coupling can be avoided. As a result, coupling related performance overheads are prevented and the synchronization between the models is inherently handled by the SystemC AMS simulation kernel.

Another option for co-simulation that has gained considerable traction is the Functional Mock-up Interface (FMI) standard [167]. It defines standardized interfaces for the interconnection of simulation models originating from disparate domains. Of particular interest is the approach of co-simulation, whereby each functional mockup unit (FMU, a model component that adheres to the specified interface) contains its own solver and is therefore capable of running standalone. A scheduler must be developed to facilitate synchronization between multiple FMUs. Exemplary works on FMI implementations can be found in [168, 169]. The usability and performance of these approaches are contingent upon the specific implementation, as the only elements that are standardized are the interfaces.

6.2 Structural Generation Flow

The structural generation is comprised of two parts: the monolithic and the hierarchical translation. Figure 6.1 illustrates the partition at a generic example. The exemplary Simulink model is partitioned into two subsystems, which represent functional units. Each functional unit is, in turn, composed of multiple blocks, resulting in a total of three levels of hierarchy. The module level is selected as the lowest level of hierarchy for mapping to the SystemC AMS model. It is therefore selected for the *monolithic* translation. Consequently, a TDF module is generated for each of the green blocks. Subsequently, the upper layers of the hierarchy, comprising the functional units and the complete model, are translated into *hierarchical* models. These models instantiate and interconnect their components. The two translation steps are integrated and automated via a MATLAB application, which offers a graphical user interface for configuration

settings and translation flow execution. The following sections provide further descriptions of the two translation processes.

6.2.1 Monolithic Modules

The translation of monolithic modules builds upon the Embedded Coder provided by MATHWORKS [154]. The term *monolithic module* refers to the type of module that is created in SystemC AMS, i.e. a classical TDF module implementing at least the member function *processing()*. With regard to the Simulink side, various options are available with respect to the source block. A monolithic module can be generated from all components that support the usage of the Embedded Coder. This includes

1. MATLAB functions
2. stateflow charts
3. atomic subsystems and
4. complete models (file with ending *.slx*).

If a complete model is taken, the approach becomes somewhat similar to the one of Kleen et al. and all internal structure gets lost [161]. Stateflow charts are typically not part of analog models and not further discussed here. Therefore, the usage of subsystems is considered for the subsequent explanation as it allows to group blocks based on the system semantics. The limitation to *atomic* subsystems is not severe as each subsystem in Simulink can be set to atomic in principle. If a subsystem is marked as atomic, it is treated as a single entity during scheduling. This entails that the complete set of modules within the subsystem is executed one after another, without the execution of other blocks of the model occurring in between. MATLAB functions may supplement the set of monolithic blocks although not explicitly stated. Their translation follows a similar procedure. Given that atomic subsystems are taken as least level of interest, the Simulink model has at least one more layer than the final SystemC AMS model, namely the content of the atomic subsystems. In theory, these subsystems can also comprise multiple viewing layers. In case of the system shown in Fig. 6.1, module A may represent a filter that is implemented through multiple gain and integrator blocks. Creating each of them as individual SystemC AMS modules does not provide any valuable insights into the system partitioning. Therefore, the user may decide to take module A as least *relevant* level of hierarchy and start the monolithic generation at that layer. Prior to initiating the automated translation process, users are thus required to select the layer at which they wish to disassemble the SystemC AMS model. The lowest layer is designated by assigning all relevant subsystems the atomic status. It should be noted that setting a subsystem to atomic may influence the behavior of the overall model slightly because the order of execution may be altered, resulting in a slight impact on transient behavior. In the majority of cases, this effect is inconsequential; however, it is imperative that the user verifies that the modified model continues to function as anticipated. Additionally, the restrictions for the scheduling process may lead to the formation of algebraic loops. As SystemC AMS is unable to resolve them, the user is required to incorporate a delay element between the subsystems.

Once the model has been successfully prepared, the monolithic translation is ready to start. The proposed flow is depicted in Fig. 6.2. It uses the option of the Embedded Coder to generate a C++ class from Simulink blocks. The direct export option to SystemC is not applicable, given that only TLM interfaces are supported [170]. However, the AMS model is designed to operate at the signal level. Initially, the Embedded Coder is configured in accordance with certain model properties and a suitable base configuration established for the translation flow.

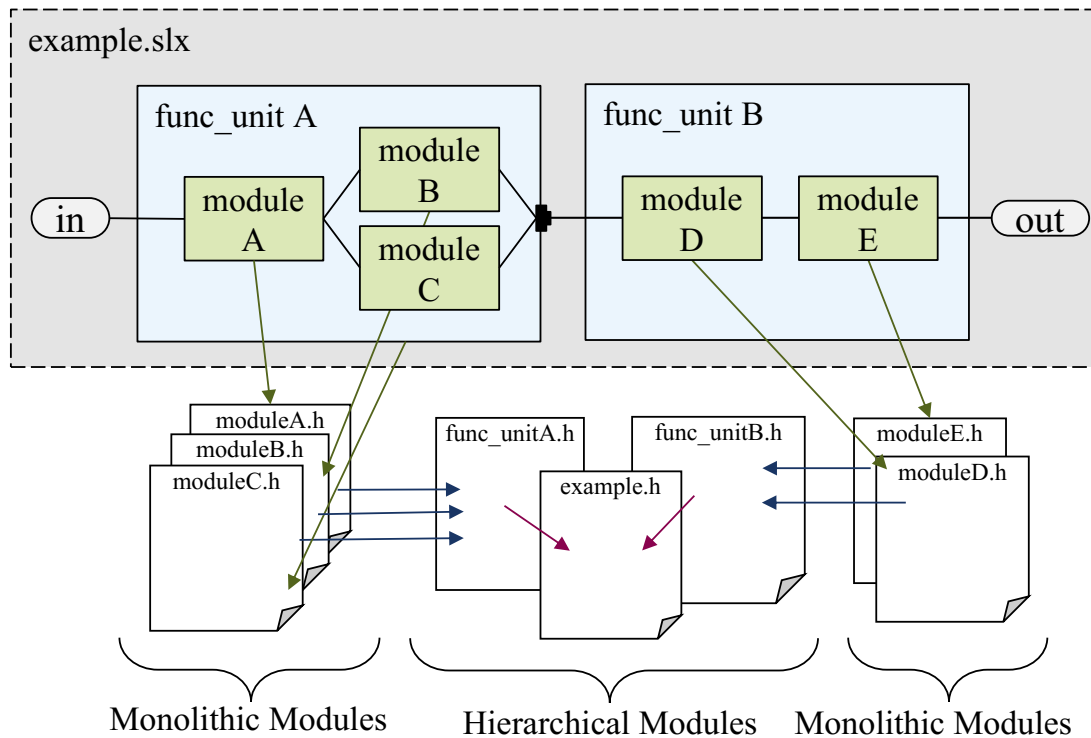


FIGURE 6.1: Overview of the structural generation flow. The exemplary model is composed of two functional units, which in turn are composed of multiple modules. Each module (A-E) is an atomic subsystem and translated using the Embedded Coder. Overall, five monolithic modules are generated. Two hierarchical modules are created for the intermediate level and one as toplevel model representing the entire Simulink model file.

This ensures that all generated code is compliant to the wrapper that is created in a consecutive step. The most important configurations include to set the target language to C++ (instead of C) and to set the interface to the mentioned C++ class. Moreover, the configuration may either set the parameters to tunable or inlined. In the latter case, the parameter values are fixed, which reduces the model's flexibility. The result of the former case depends on the setup of the Simulink model. If parameters are part of the base workspace, they are translated into global variables. If they are situated within the model workspace, they become static class attributes. To prevent any potential conflicts between multiple translated blocks, it is imperative that parameters within Simulink are consistently situated within the model workspace. In the configuration phase, the solver type and the timestep of the model are selected. The Embedded Coder only supports fixed-step solvers, and the timestep must be set prior to code generation. The user may elect to incorporate ordinary differential equation solvers into the generated code or to utilize discrete solvers exclusively. The former approach typically yields more accurate results, although the latter generates faster models. All components of the Simulink model must comply with the selected solver. For instance, if a discrete solver is chosen, no continuous states are permitted.

Once the Embedded Coder has been configured, the code generation process is initiated. The Embedded Coder generates C++ code (headers and source files), an HTML report that provides a summary of the properties of the created code, and a MATLAB workspace object file (.mat) that contains a struct with some important code information, including model ports and parameters. These will subsequently serve as a basis for the hierarchical generation process. The methods of the generated C++ code fit well into the simulation semantics of SystemC AMS.

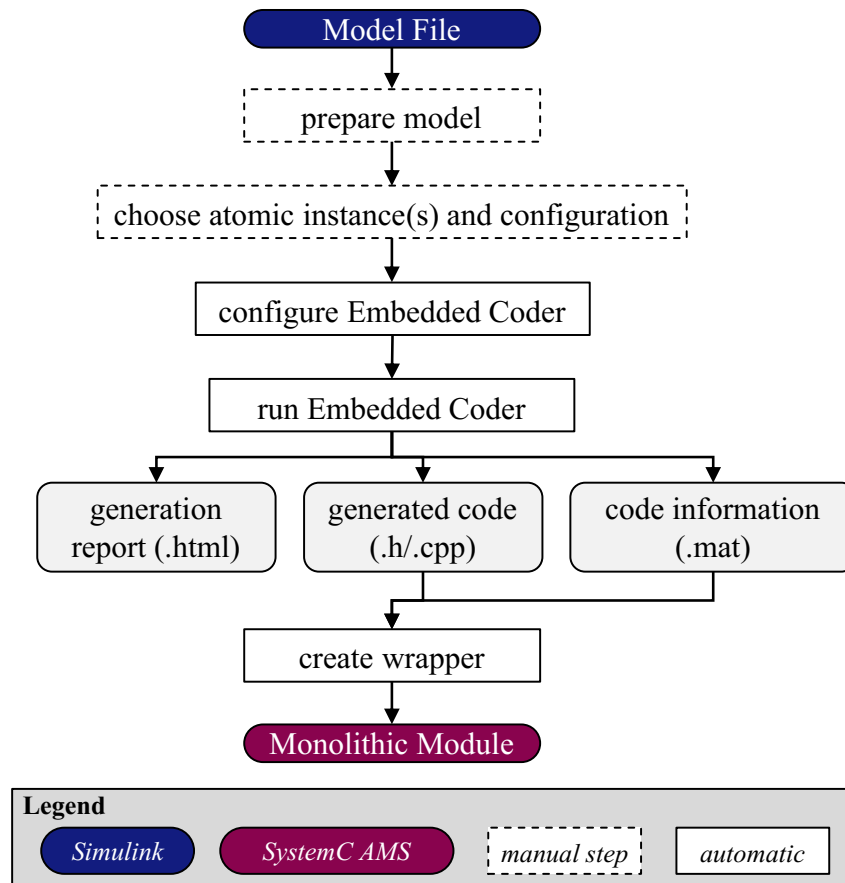


FIGURE 6.2: Monolithic translation procedure. The Embedded Coder is used to generate C++ code which is then embedded into a SystemC AMS wrapper to create a TDF module. The configuration specifies timestep, solver and parameter handling.

Figure 6.3 illustrates the semantics of the generated code in comparison to those of a TDF module. The similarity permits a one-to-one mapping of the methods. A SystemC AMS wrapper is generated, which creates a TDF module with an instance of the generated C++ class as an attribute. The TDF methods call the corresponding methods of the class instance. Furthermore, the processing call reads the input ports of the TDF module and transfers their values to the class object. Subsequently, the function *step()* is invoked and the output data are written back to the TDF output ports. The function *step()* is designed such that it expects a fixed step size as specified during the configuration of the Embedded Coder. The proper synchronization between the class object and the TDF module is ensured by setting the same timestep within the *set_attributes()* method of the wrapper.

The utilization of fundamental data types, such as integers and doubles, within the Simulink model is sufficient for the generation of a functional SystemC AMS module. Basic data types are directly mapped from Simulink to their C++ equivalents. However, SystemC and SystemC AMS provide some specific data types. These shall be used by the wrapper to achieve an interface that is similar to and connects well with handwritten SystemC AMS code. This paves the way for the integration with other SystemC or AMS models. The following list enumerates all cases that are explicitly recognized and handled during wrapper generation:

Complex Numbers Simulink maps complex numbers onto a structured data type comprising two members, one for the real part and one for the imaginary part. SystemC AMS provides its

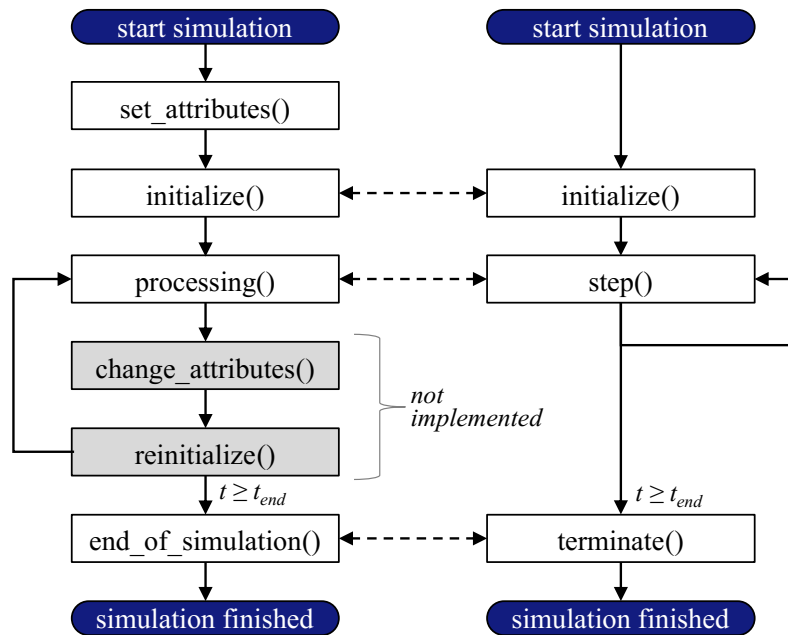


FIGURE 6.3: Sequence diagram of a single TDF module (left) in comparison to the code generated by the Embedded Coder (right). A one-to-one mapping of the methods is applicable.

own data type for complex numbers (*sca_util:sca_complex*). Consequently, the wrapper translates the C++ structure into the specified data type.

Fixed-point Numbers The Embedded Coder represents fixed-point numbers in integer form. The built-in C++ integer type with the smallest sufficient number of bits is employed for this purpose. To illustrate, a fixed-point number with 12 digits is mapped onto an *int16* type, while a 32-digit fixed-point number is mapped onto an *int32* type. The generated code interprets these integers as fixed-point numbers and scales them accordingly. Nevertheless, the actual scaling is not discernible at the interface. In contrast, the SystemC language offers explicit data types for fixed-point numbers. The wrapper generates TDF ports in the desired fixed-point format. Within the *processing()* function, the input values are scaled onto the corresponding integer representation and conveyed to the generated code. After its execution, the outputs are scaled back to the fixed-point format and written to the TDF output ports. This measure significantly increases the expressiveness of the module.

Buses Another special case occurs if bus signals are used inside the Simulink model. Simulink distinguishes between virtual and nonvirtual buses. A virtual bus serves solely to define a structuring within the graphical representation, without affecting the model semantics. In contrast, a nonvirtual bus implies the existence of an actual bus structure. It imposes the restriction that all signals within the bus must have identical sampling times. Virtual buses are disregarded by the Embedded Coder. All signals are treated as independent entities and mapped onto their corresponding data types. Nonvirtual buses are translated into C++ structs. Two options exist for the generation of the respective wrapper. The struct is either split into its individual signals within the wrapper, such that all SystemC AMS ports utilize basic data types, or a TDF port that uses the struct as the data type is defined. In the latter case, the SystemC AMS tracing function must be overloaded in order to enable the tracing of the bus signals. If the bus structure is not named within Simulink, the SystemC AMS naming is derived from the name of the output signal of the bus creator (*bus_[signalName]*). In the absence of a designated name,

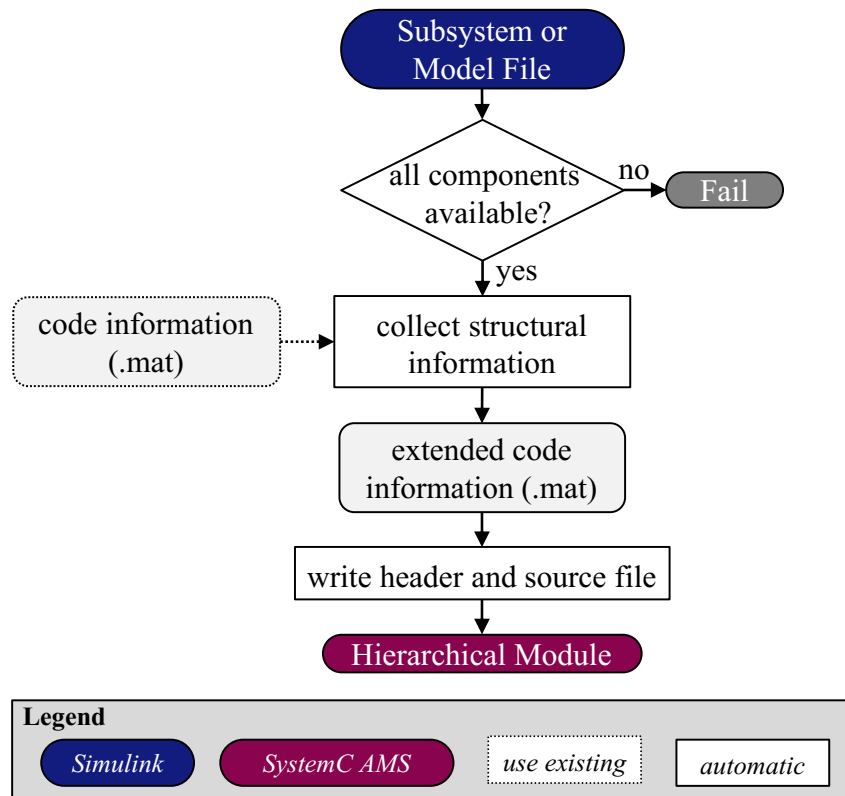


FIGURE 6.4: Hierarchical translation procedure. The existence of all component models is required to generate a hierarchical model.

a generic name *bus_[n]* is employed with *n* denoting an integer that ensures uniqueness when multiple unnamed bus structures coexist.

6.2.2 Hierarchical Modules

Once the atomic subsystems have been translated using the aforementioned monolithic flow, the hierarchical modules are ready to be generated. Please note that the final SystemC AMS toplevel model does not have to be the Simulink toplevel (the model file) but can also be a subsystem. For example, functional unit A in Fig. 6.1 may already be available in SystemC AMS, such that only functional unit B is to be translated. In this case, modules D and E are generated monolithically, and functional unit B is created as a hierarchical module. Figure 6.4 illustrates the proposed flow for the hierarchical generation.

The translation tool is designed to identify all components of the Simulink subsystem or model file that is to be translated. In order to commence the hierarchical generation process, these components must be atomic subsystems that have been generated via the monolithic approach, such that mat-files containing code information are available, other previously generated hierarchical modules, or specific primitives that will be explained later. The mat-files serve two purposes: firstly, they indicate that the component has already been generated; secondly, they provide information for the hierarchical setup. Among others, the mat-file indicates the name and data type of each port within the block. In the event that models for all components are available, the tool proceeds to collect structural information on how these components are connected to one another. In the generic example illustrated in Fig. 6.1, the translation of functional unit A as a hierarchical module necessitates the interconnection of the output of module A with

the inputs of module B and module C via a signal. The tool identifies the necessity for this signal and designates a name for it. The mat-files of the interconnected components are then augmented with the pertinent signal information. This is achieved by appending the signal name to each structure that defines a connected port within these files.

As usually not all signals within a Simulink model are named, a naming convention has been defined that ensures the uniqueness of the signal names. The thesis at hand makes use of data flow related signal properties for this purpose. A signal may have none, a single, or multiple recipients but it must always have exactly one driver. Consequently, the variable name of each unlabeled signal is derived from its driver to ensure its uniqueness. As each block may have multiple output ports and multiple blocks are allowed to have similar port names, both, block and port name are used for identification. As a result, each unnamed signal in Simulink will result in a SystemC AMS signal following the naming convention *s_[blockName]_[portName]*. In the given example of functional unit A, the name of the internal signal derives to *s_moduleA_out*. It is then added to the mat-files of module A, B and C as part of the port information section. The input of module A is connected to the input of the upper level of hierarchy. Here, port-to-port binding is done and the name of the hierarchical input port is added to the mat-file replacing a signal name.

When the complete structure has been parsed, the set of mat-files includes all relevant structural information to generate the hierarchical module. The actual code generation is ready to start. The tool generates the code for a SystemC module by instantiating all components (monolithic TDF modules and other hierarchical modules) and conducting a port binding based on the collected structural information. Module instances are designated according to the naming scheme *[blockName]_i*.

Supported Primitives Outside of Subsystems

In the example given in Fig. 6.1, the model is entirely decomposed into subsystems without any intermediary primitives. In theory, restricting the flow to these types of models is not a limitation, as even a single primitive can be embedded into a subsystem. Still, the author of the thesis at hand has identified a small set of primitives for which it is semantically beneficial to place them outside of subsystems in certain cases. The translation tool is able to recognize these primitives and utilize templated SystemC AMS library blocks to incorporate them during the process of hierarchical generation. These primitives are:

- memory and delay blocks
- type converters
- gain stages
- bus creators and selectors.

On top of that, the use of scopes and to-File blocks is permitted but these primitives are simply ignored during hierarchical generation. To restore their functionalities, the tracing option inherent to SystemC AMS may be enabled.

As previously stated, setting subsystems to atomic may result in the creation of algebraic loops within Simulink, necessitating the introduction of a delay element to resolve this issue. To guarantee the schedulability of the hierarchical models in SystemC AMS, it is imperative to break algebraic loops at each level of the hierarchy, even in instances where Simulink may not recognize them. This may happen if subsystems are not set to atomic, and thus do not exert any influence on the scheduling process. If they are then translated hierarchically, they become atomic within the SystemC AMS environment and impede the schedulability of the respective cluster. In Simulink, memory and delay blocks can be employed to break algebraic

loops. It is therefore beneficial, and in some cases imperative, to situate these blocks outside of the subsystems. The equivalent SystemC AMS library block comprises an input and an output with a templated data type. It applies the input value to the output without any change except the insertion of a unit delay. As an alternative, the translation flow could be modified such that no explicit delay module is incorporated into the SystemC AMS cluster, but rather an input delay is assigned to any port driven by the Simulink memory/delay block. Please note that shifting the delay to the output of the driving module is generally not applicable as the output port might drive multiple inputs of which only a subset shall be delayed. The proposed strategy offers two key advantages. Firstly, it draws attention to the presence of an algebraic loop, and secondly, it ensures that the semantics of the subsystems remain consistent with their equivalent blocks in Simulink.

Similarly, type conversions and gain stages incorporate primitives in SystemC AMS, where in- and output data types are templated. The gain module multiplies the input by a parameterized gain factor. The type conversion entails the reading of the input, its conversion to a different type, and the subsequent writing of the converted value to the output. The scheme is only applicable to C++ data types that provide a built-in conversion scheme between them. If multiple subsystems are provided by disparate groups or designers, it is possible that their interfaces may not initially align. In such a scenario, it is beneficial to insert a type conversion and/or scaling stage between the two to align their respective interfaces. In order to enable the separation of the component modeling from the integration task, it is permitted to insert type converters and gain stages outside atomic subsystems.

Lastly, bus creators and selectors adhere to a similar objective. A bus creator situated between the subsystems enables the aggregation of data from multiple blocks and its transmission to a subsequent block via a unified bus signal. The SystemC AMS library block implements an array of input ports with a templated number of ports and merges them to a single output whose data type is the corresponding struct. Bus selectors work vice versa.

Referenced Subsystems

A referenced subsystem occurs if a Simulink model file (*.slx*) is used as a subsystem within another model. The speciality in this approach is that it allows for the use of multiple referenced subsystems within one model, each referencing the same model file. This results in the occurrence of multiple instances of the same component. The translation tool is capable of recognizing the doubling of the component and generates the monolithic module for the subsystem only once. During the hierarchical construction step, multiple instances of this monolithic module are generated. Accordingly, the nomenclature is augmented to include the designation *[blockName]_i[n]*, where *n* represents an integer counting from zero onwards. During the collection of structural information, the mat-file is assigned multiple signal names for each port as an array. In the course of code generation, each instance is associated with a specific index within the array. In Simulink, each referenced subsystem is permitted to have its own unique parameterization. The Embedded Coder, however, realizes tunable parameters as *static* class attributes such that each instance in SystemC AMS ends up to have the same parameterization. As this is often not the desired outcome, the header of the monolithic module is parsed and the static property is removed for all parameters.

6.3 Restrictions and Modeling Guideline

The usage of the Embedded Coder imposes some restrictions on the translation process. The most significant limitation is the lack of compatibility with variable-step solvers, which can significantly impair the simulation performance. Furthermore, it is incumbent upon the modeler

to ensure that the Simulink model does not contain any blocks that are exclusively designed to operate with variable-step solvers. The majority of library blocks are compatible with both solver types, thereby rendering this restriction irrelevant for a significant proportion of models. S-functions are typically less flexible and must be handled with caution. A significant restriction to the modeling style is the prohibition of global *Goto* blocks. Bus element ports are supported and may be employed as a substitute for the purpose of sharing signals between distant blocks without the necessity of actual routing. In addition, certain blocks from various Simulink toolboxes are not applicable. As it is not feasible to provide an exhaustive list here, the reader is directed to the MATLAB documentation for further information [171]. A summary of additional modeling guidelines derived is listed below:

- Group blocks into semantically meaningful subsystems to enable hierarchical generation.
- Use expressive names for subsystems that are valid C++ variable names.
- Use referenced subsystems for components that occur multiple times.
- Reset the option *minimize algebraic loop occurrences* for atomic subsystems to hinder Simulink from resolving algebraic loops. It eases the identification whether the resulting SystemC AMS model will be schedulable.
- Add a delay block between atomic subsystems to break any identified algebraic loop and run the Simulink simulation again to check if the simulation results are still consistent.
- If the delay element impacts the transient behavior significantly, consider to generate a monolithic module for the entire loop instead. Alternatively, regroup the blocks to resolve the algebraic loop if possible.
- Avoid to mix single-dimensional and multi-dimensional ports with (de-)multiplexers. Interfaces become inconsistent for hierarchical generation. Switch to buses if possible.
- If buses shall be translated in a structured fashion, set the bus to nonvirtual directly during the generation of the Simulink model. It ensures that all sampling times are consistent.
- Designate the output signal of a bus creator with inherited type to ensure an expressive name for the resulting structure.

6.4 Experimental Results

In the following, the procedure is validated at the example of a MEMS accelerometer model with three relevant levels of hierarchy. The block diagram is depicted in Fig. 6.5. The model file comprises three principal components: the accelerometer itself, an environmental block that supplies the sensor with external stimuli, and an evaluation block. They jointly constitute a testbench for the accelerometer model. The sensor model without testbench is to be reused for the respective HVP. It is divided into two functional units, namely the MEMS sensor element and the frontend. The mechanics are further subdivided into three sensing axes, while the frontend contains multiple analog modules as the charge-to-voltage converter, the ADC, and a cascaded integrator-comb (CIC) filter. Moreover, it supplies the necessary voltages to operate the MEMS. These blocks are considered as the least relevant level of hierarchy. Internally, they are composed of multiple primitives that are provided by Simulink's libraries. The CIC filter for instance is realized by a set of adders and discrete delay blocks.

The MEMS channels are classified into two categories: those moving in-plane (x-,y-axis) and the one moving out-of-plane (z-axis). As x- and y-axis work similar, a referenced subsystem

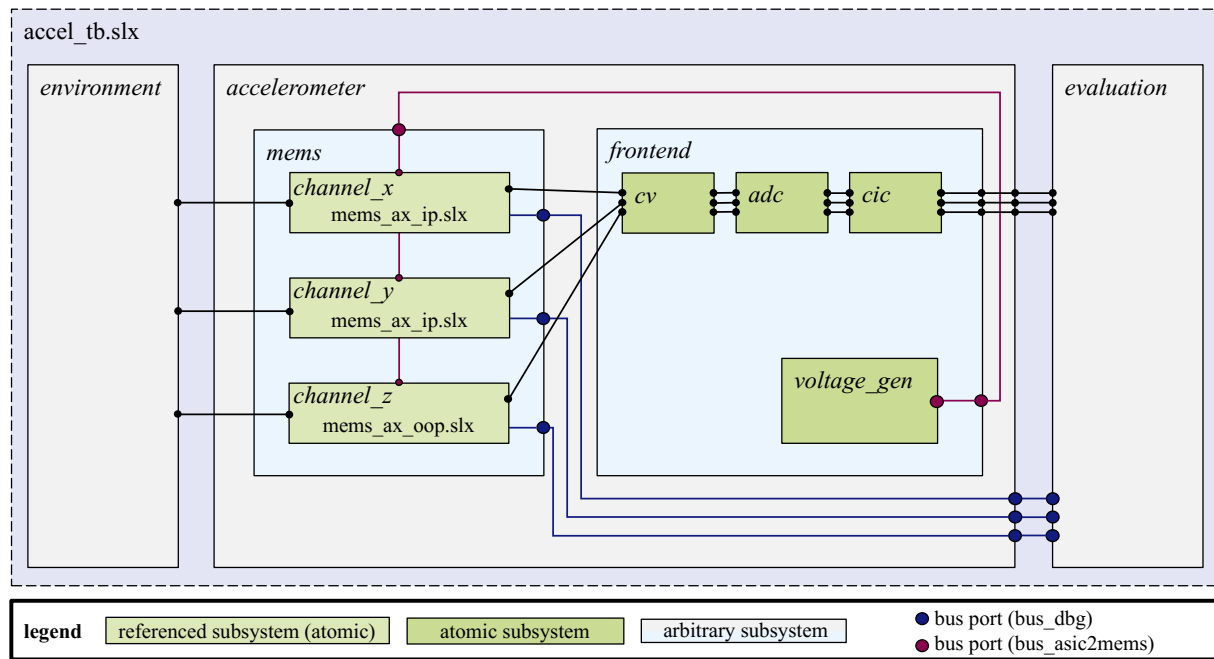


FIGURE 6.5: Model under test for the translation procedure. It represents a MEMS accelerometer within a simple testbench. The accelerometer shall be translated preserving three levels of hierarchy. The subsystems illustrated in green denote the lowest relevant level of hierarchy and are translated monolithically. MEMS, frontend and the complete accelerometer are generated hierarchically. Testbench specific modules (environment, evaluation) are skipped.

is used to reuse the same model for both blocks. The interface from ASIC to MEMS is implemented by a single bus containing all requisite voltages to supply the MEMS electrodes. Figure 6.6 shows the graphical user interface (GUI) of the translation tool. It is realized as MATLAB application and can only be started if a Simulink model is loaded. The underlying reason is that the application searches for all subsystems within the model during initialization in order to create a corresponding module list in the GUI. Moreover, the current solver settings from the schematic are utilized as the default values for the configuration. All non-virtual buses are identified and listed. Their names are editable within the GUI. The button *Show* highlights the corresponding bus within the schematic. The translation procedure section enables the user to select the desired procedure, the monolithic one, the hierarchical one, or both. The monolithic generation process is only applicable to atomic subsystems. The hierarchical generation process is available for subsystems whose components are either already available (including the associated code information in the mat-file) or have been selected for monolithic translation. Subsystems that are not applicable are grayed out.

The blocks that shall be translated are denoted with checked boxes in Fig. 6.6. However, the Simulink model as illustrated in Fig. 6.5 will not create a schedulable SystemC AMS model. The MEMS drives signals for the inputs of the frontend, and the frontend in turn drives signals for the inputs of the MEMS. As a consequence, SystemC AMS would identify an algebraic loop comprising these two modules and the system would not be schedulable. The Simulink scheduler does not identify any algebraic loop because the frontend subsystem is not set to atomic and the voltage generator is not dependent on any signals originating from the MEMS. A delay is added to the capacitive outputs of the MEMS to break the algebraic loop. Then, the hierarchical translation procedure is ready to start. Listing 6.1 shows a code snippet of the resulting toplevel accelerometer model. Comments denoted with `/* */` have been added manually for

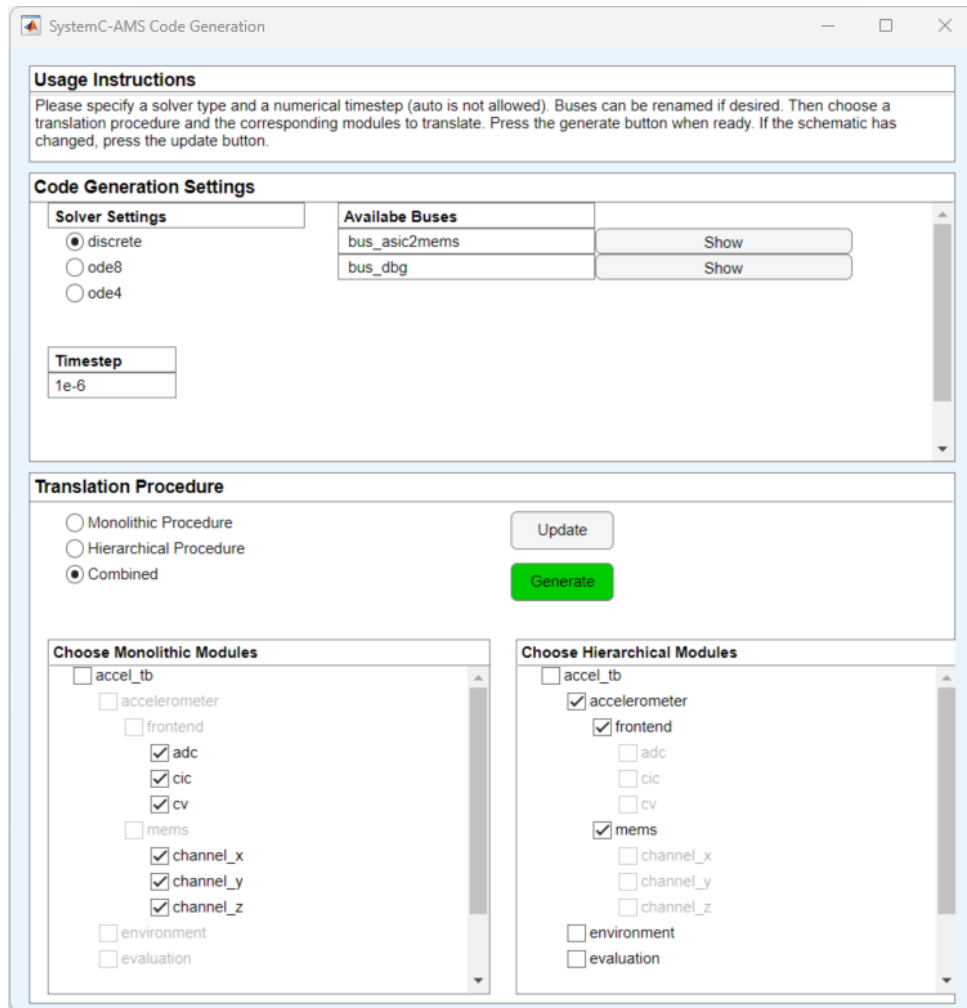


FIGURE 6.6: Graphical user interface of the presented translation tool. It is implemented as a MATLAB application. Its exact appearance depends on the model under test as it lists available bus structures and subsystems within the model.

explanation. For further code examples, please refer to Appendix B which contains the generated code for the monolithic module *mems_ax_oop* and the hierarchical MEMS module.

The complete translation process including the adaption of the Simulink model has been performed within a few minutes. Figure 6.7 compares the simulation results of the Simulink model to those of the generated SystemC AMS code for a sinusoidal acceleration stimulus. The models show exact accordance. Their different appearance in Fig. 6.7b is due to the fact that the Simulink trace is interpolated between its sampling points and the SystemC AMS trace follows a sample-and-hold scheme. Still, the values of the discrete samples align well which validates the correctness of the translation process.

6.4.1 Performance Evaluation

In order to get an impression of the simulation performance of the generated code in comparison to handwritten SystemC AMS models, the $\Sigma\Delta$ -ADC from ACCELERA's applications examples has been rebuilt in Simulink [14]. Figure 6.8 illustrates the resulting model. The loop filter is implemented using continuous integrators, and the solver is set to *ode8*, which is a Runge-Kutta solver of eighth order. A second version employs discrete integrators and a discrete solver. Both versions were translated to SystemC AMS using the proposed methodology.

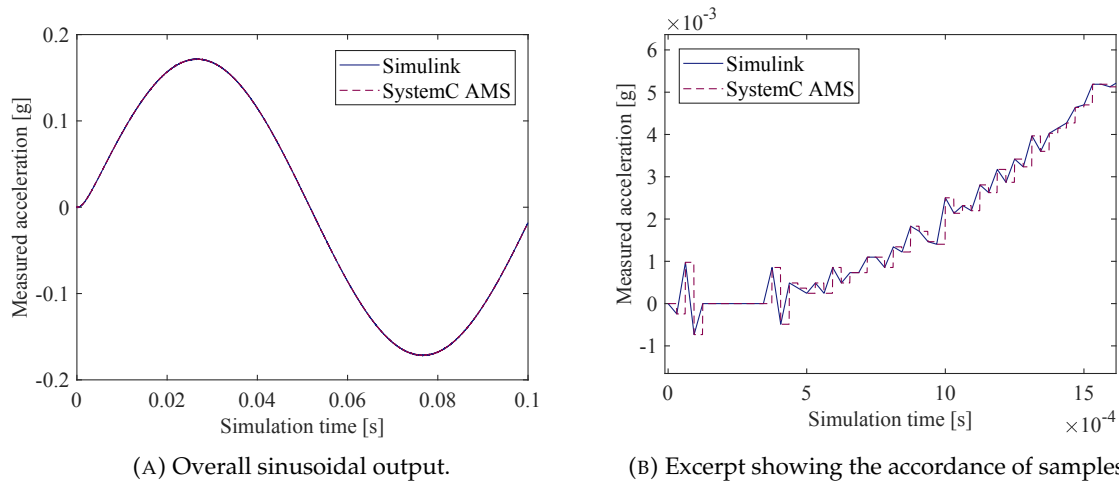


FIGURE 6.7: Simulation results of the accelerometer model in presence of a sinusoidal stimulus. The traces of the original Simulink model and the generated SystemC AMS model align well.

Moreover, a number of handwritten SystemC AMS models have been taken into consideration. The continuous version implements the loop filter as an LSF model and establishes connections with TDF modules for the quantizer and the digital-to-analog converter (DAC). One of the discrete versions replaces the LSF implementation of the loop filter with a third TDF module and connects all TDF blocks to a single hierarchical model. The other version creates a single monolithic TDF model encompassing the entire ADC.

LISTING 6.1: Excerpt from the outcome of the hierarchical translation procedure (*accelerometer_module.h*).

```
class accelerometer_module : public sc_core::sc_module
{
public:
    // ports
    sca_tdf::sca_in<double> acc_x;
    sca_tdf::sca_in<double> acc_y;
    sca_tdf::sca_in<double> acc_z;
    sca_tdf::sca_out<double> acc_measured_x;
    [...]
    sca_tdf::sca_out<double> dbg_x_x;
    sca_tdf::sca_out<double> dbg_x_Fel;
    sca_tdf::sca_out<double> dbg_y_x;
    [...]

    // instances
    mems_module *mems_i;
    frontend_module *frontend_i;
    delay<double> *delay_i0; /* primitive with templated data type */
    delay<double> *delay_i1;
    delay<double> *delay_i2;

    // signals
    /* s_[blockName]_[portName] with [portName] = [busName]_[sigName] */
    sca_tdf::sca_signal<double> s_frontend_asic2mems_vc1x;
    sca_tdf::sca_signal<double> s_frontend_asic2mems_vc2x;
    sca_tdf::sca_signal<double> s_mems_dCx; /* similar for y and z */
    sca_tdf::sca_signal<double> s_delay0_out; /* similar for other two delays */
}
```

```

[...]
```

```

accelerometer_module(sc_core::sc_module_name nm)
: sc_module(nm)
, mems_i(new mems_module("MEMS"))
, frontend_i(new frontend_module("FRONTEND"))
, delay_i0(new delay<double>("DELAY0"))
, delay_i1(new delay<double>("DELAY1"))
, delay_i2(new delay<double>("DELAY2"))
{
    mems_i->acc_x(acc_x); /* port to port binding */
    mems_i->acc_y(acc_y);
    mems_i->acc_z(acc_z);
    mems_i->asic2mems_vc1x(s_frontend_asic2mems_vc1x); /* port to signal binding */
    [...]
    mems_i->dC_x(s_mems_dCx);
    mems_i->dC_y(s_mems_dCy);
    mems_i->dC_z(s_mems_dCz);
    [...]
    frontend_i->cap_x(s_delay0_out);
    frontend_i->cap_y(s_delay1_out);
    frontend_i->cap_z(s_delay2_out);
    frontend_i->asic2mems_vc1x(s_frontend_asic2mems_vc1x);
    frontend_i->asic2mems_vc2x(s_frontend_asic2mems_vc2x);
    [...]
    delay_i0->in(s_mems_dCx); /* delay inserted between mems and frontend */
    delay_i0->out(s_delay0_out);
    delay_i1->in(s_mems_dCy);
    delay_i1->out(s_delay1_out);
    [...]
}
};

```

Table 6.1 provides a summary of the wall-clock times. Each configuration was subjected to a one-second simulation with a step size of 1 μ s. The Simulink model was simulated in normal mode, exhibiting the lowest speeds in both the continuous and discrete cases. Please note that Simulink offers a *Rapid Accelerator Mode* which performs code generation and simulates the resulting code. If the build time is disregarded, the resulting model runs significantly faster. The objective of the experiment is to evaluate the simulation performance of the SystemC AMS versions such that the Simulink performance is given for reference only.

In the continuous case, the handwritten model exhibited a runtime approximately twice that of the generated one. The generated code employs an ODE solver of order eight, whereas the SystemC AMS library utilizes a second-order ODE solver. Therefore, the additional computational cost is accompanied by a reduction in the accumulation of errors. In the discrete case, the generated code exhibits a slight advantage in speed over the handwritten one. Furthermore, it is evident that the monolithic version exhibits superior performance compared to the hierarchical one, as it eliminates the need for some function calls. The enhanced performance is paid by a reduction in the expressiveness of the model. In conclusion, the performance discrepancies between handwritten and generated code remain within acceptable limits, ensuring the feasibility of the approach.

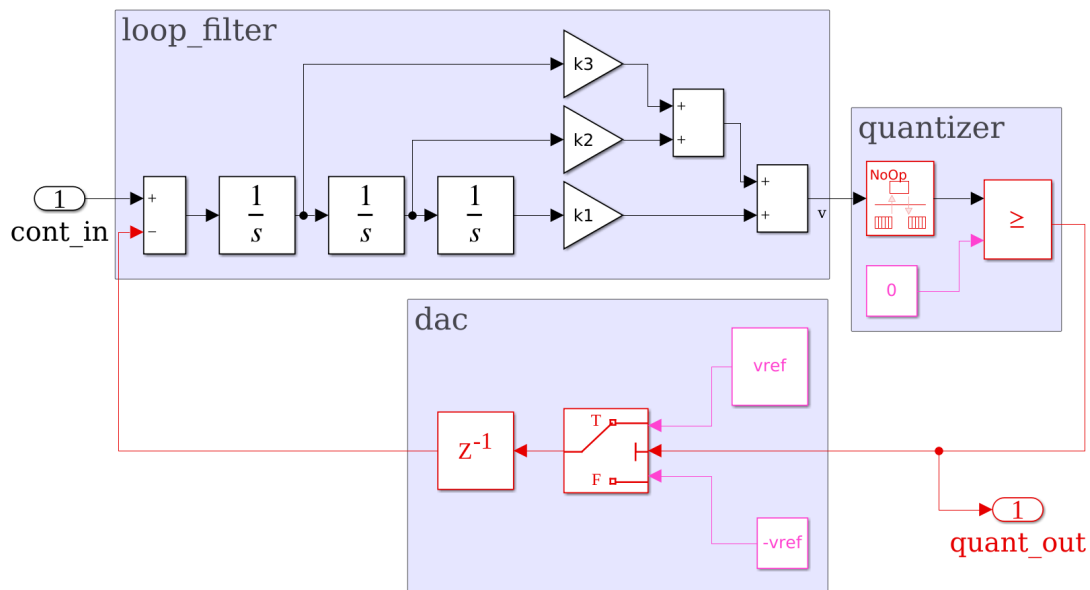


FIGURE 6.8: Continuous $\Sigma\Delta$ -ADC implementation in Simulink. For the discrete version, the continuous integrators of the loop filter have been replaced by their discrete counterparts. The color black indicates the continuous part of the system, whereas red indicates a discrete timestep and pink refers to constants.
© IEEE 2024.

TABLE 6.1: Simulation performance of different modeling styles of a $\Sigma\Delta$ -ADC. One second has been simulated with a step size of $1.0\ \mu\text{s}$.

	Language	Model Variant	Generated	Wall-clock time
cont.	Simulink	see Fig. 6.8, ode8 (Dormand-Prince)	×	19.79 s
	SystemC AMS	loop filter as LSF	×	0.78 s
	SystemC AMS	C++ class with TDF wrapper	✓	1.57 s
discrete	Simulink	discretized version of Fig. 6.8	×	4.8 s
	SystemC AMS	three connected TDF modules	×	0.90 s
	SystemC AMS	single analytical TDF module	×	0.40 s
	SystemC AMS	C++ class with TDF wrapper	✓	0.31 s

6.5 Discussion

In the course of this chapter, a two-step translation procedure was presented to generate SystemC AMS models from Simulink models or portions thereof. The distinction between monolithic and hierarchical translation allows the original model's structuring to be maintained, ensuring that the resulting code accurately reflects the module breakdown. As a result, the expressiveness of the model is enhanced, and it becomes more than a pure black box model. Furthermore, intermediate signals are available and can be traced in various formats to ease the debugging process. As a major advantage, the hierarchical translation increases the modularity of the resulting model. As the model is targeted for first rapid prototyping and components will most likely be replaced by refined versions in later stages, modularity is of utmost importance. With the presented approach, components can be modeled and replaced step-by-step without disrupting the signal chain. The presented methodology hence enables an agile workflow and a continuous improvement of the HVP.

Nevertheless, the readability of the monolithic modules is constrained by the usage of the Embedded Coder. More severely, the restriction on fixed-step solvers precludes the compatibility

between this flow and the variable-step solving methodology presented in the previous chapter. The latter can only be applied once all monolithic modules originating from Simulink have been replaced by refined versions. Hierarchical modules can still be used without any issue. In general, the simulation performance of the very generic early models is likely to be high anyway. The addition of further functionality, for instance to facilitate trim and test operations, may result in a reduction in the efficiency of the HVP. Therefore, it is still reasonable to introduce variable-step solving in a posterior project stage.

Future research may concentrate on the elimination of the limitation imposed on fixed-step solvers. To achieve this objective, a promising approach would be to combine this work with that of Kleen et al. [161]. Their utilization of Java Emitter Templates enables the manipulation of the outcome of the Embedded Coder. Should a timestep parameter be introduced into the generated code and utilized by integrators and other time-dependent blocks, the generated code would be capable of adhering to the timestep alterations produced by a SystemC AMS internal variable-step solver, without compromising the integrity of the results.

Chapter 7

Semi-Symbolic Simulation of Virtual Sensor Prototypes

This chapter addresses the coverage of the analog state space which is examined by *RQ4-Coverage*. This chapter deals with this property by introducing a semi-symbolic simulation scheme that operates on intervals rather than concrete values. It is founded upon the principles of affine arithmetic. In the context of this work, the scheme especially targets to enhance the level of confidence in the validation of trim features.

The chapter is structured as follows. Firstly, the characteristics and underlying mathematical principles of affine arithmetic are introduced and existing fields of application are presented in sections 7.1 and 7.2. In order to provide the necessary context, related work is introduced in these sections. Subsequently, the applicability of affine arithmetic to sensor trimming is discussed in section 7.3. The properties of affine arithmetic match well to the challenges of trimming simulation and are leveraged beyond the pure scope of range analysis. Based on this use case, limitations are pointed out in section 7.4. In order to partially overcome these limitations, an extension to classical affine arithmetic is proposed in section 7.5. A case study on a binary search algorithm is presented as a proof of concept for the applicability on a real-world example. It is given in section 7.6. The chapter is summarized in section 7.7. Parts of the results presented in this chapter have been previously published in [P5]. The author of this work is also the corresponding author of the referenced paper¹.

7.1 Affine Arithmetic

The concept of affine arithmetic (AA) was first introduced by Stolfi and Comba in 1993 [172]. Its semantics are based on classical interval arithmetic, which was first formalized approximately 30 years earlier by R.E. Moore [173]. Interval arithmetic operates on ranges. This enables the consideration of uncertainties, whether in the device under test or in the model itself. The objective of interval arithmetic is thus to enable effective range analysis and to track errors that a model introduces. However, interval arithmetic has a significant drawback: it strongly overestimates intervals because it does not include any information on correlations between quantities. This can be illustrated with an easy example. Taking the interval

$$\hat{x} = [0, 10] \quad (7.1)$$

and performing a simple subtraction using classical interval arithmetic (IA)

$$\hat{x} - \hat{x} \stackrel{IA}{=} \left[(\min(\hat{x}) - \max(\hat{x})), (\max(\hat{x}) - \min(\hat{x})) \right] = [-10, 10] \quad (7.2)$$

¹See section 1.3 page 5 for the author's contribution to this publication.

reveals an interval $[-10, 10]$ although the equation should obviously result in zero. Affine arithmetic tries to reduce the overestimation rewriting each interval to an affine form

$$\tilde{x} = x_0 + \sum_{i=1}^m x_i \varepsilon_i, \quad \varepsilon_i \in [-1, 1] \quad \forall i, \quad m \geq 0 \quad (7.3)$$

with x_0 being the center value of the interval and each ε_i being a deviation symbol which forms a deviation term together with the corresponding weight x_i called partial deviation. A deviation symbol is defined to lie within the interval $[-1, 1]$ but is never actually assigned during simulation. For that reason, simulations using affine arithmetic are referred to as symbolic or semi-symbolic. Affine forms are designed such that the boundaries of the interval spanned by an affine form $\tilde{x}$ can easily be evaluated to

$$x_{min} = x_0 - \sum_{i=1}^m |x_i|, \quad x_{max} = x_0 + \sum_{i=1}^m |x_i|. \quad (7.4)$$

This keeps range analysis simple and computationally cheap. In order to illustrate the benefits of affine arithmetic compared to classical interval arithmetic, please refer again to the simple subtraction example. The equation

$$\tilde{x} - \tilde{x} = x_0 - x_0 + \sum_{i=1}^m (x_i \varepsilon_i - x_i \varepsilon_i) = 0 \quad (7.5)$$

yields zero this time because the deviation symbols clearly identify the correlation between minuend and subtrahend. Consequently, two distinct affine forms $\tilde{x}$ and $\tilde{y}$ can either be (partially) correlated if they (partially) depend on the same deviation symbols ($\exists i \ x_i \neq 0 \wedge y_i \neq 0$) or uncorrelated if their sets of deviation symbols are distinct ($x_i \neq 0 \Rightarrow y_i = 0, y_i \neq 0 \Rightarrow x_i = 0$). Affine arithmetic is defined on linear operations as these operations again yield affine forms. Therefore, they are also referred to as affine operations. Assuming two affine forms $\tilde{x}$ and $\tilde{y}$, m the overall number of deviation symbols, and a scalar c , these operations evaluate to

$$\tilde{x} \pm \tilde{y} = x_0 \pm y_0 + \sum_{i=1}^m (x_i \pm y_i) \varepsilon_i \quad (7.6)$$

$$c\tilde{x} = cx_0 + c \sum_{i=1}^m x_i \varepsilon_i \quad (7.7)$$

and can be merged to any linear combination of $\tilde{x}$ and $\tilde{y}$. In contrast to this, nonlinear operations do not yield affine forms and are therefore referred to as non-affine operations, for example

$$\tilde{x} \cdot \tilde{y} = x_0 y_0 + \sum_{i=1}^m (x_0 y_i + x_i y_0) \varepsilon_i + \sum_{i=1}^m \sum_{j=1}^m x_i y_j \varepsilon_i \varepsilon_j \quad \text{forbidden.} \quad (7.8)$$

As a consequence, these operations must be approximated if they shall be used within a simulation. Different solutions have been presented in literature to handle this issue [174]. One common approach is the incorporation of new deviation symbols. Keeping the example of a multiplication, the equation then yields

$$\tilde{x} \cdot \tilde{y} = x_0 y_0 + \sum_{i=1}^m (x_0 y_i + x_i y_0) \varepsilon_i + \varepsilon_{m+1} \sum_{i=1}^m \sum_{j=1}^m x_i y_j \quad (7.9)$$

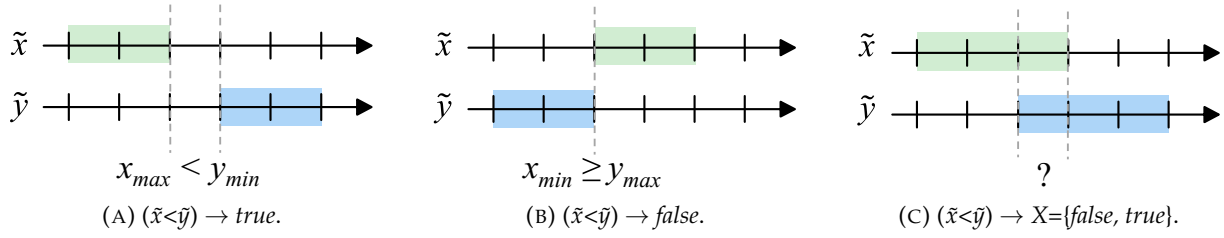


FIGURE 7.1: Affine conditions and their three possible results.

with ε_{m+1} being the newly introduced deviation symbol that accounts for the originally non-affine term. It is reasonable to posit that this approach will rapidly result in an exponential increase in the number of deviation symbols when employed in transient simulations. If it is assumed that each timestep requires a single multiplication and that the simulation performs N simulation steps, then the number of deviation symbols scales with $\mathcal{O}(N)$. It is evident that this approach is not feasible for long transient simulations. Accordingly, the present thesis employs Hansen's formulation, as previous research on symbolic simulation with SystemC AMS has done [175]. Hansen's form dispenses with the introduction of new deviation symbols. Instead, it drops the non-affine terms and tracks resulting approximation errors in an additional offset term e as shown below:

$$\tilde{x} \cdot \tilde{y} = x_0 y_0 + \sum_{i=1}^m (x_0 y_i + x_i y_0) \varepsilon_i + \underbrace{\sum_{i=1}^m \sum_{j=1}^m \delta_{ij}}_e \quad (7.10)$$

$$\delta_{ij} := \begin{cases} [-|x_i||y_j|, |x_i||y_j|] & i \neq j \\ [0, x_i y_i] & i = j, x_i * y_i > 0 \\ [x_i y_i, 0] & i = j, x_i * y_i < 0. \end{cases} \quad (7.11)$$

The offset term falls back into classical interval arithmetic. Consequently, the interval boundaries (7.4) transform to

$$x_{min} = x_0 - \sum_{i=1}^m |x_i| + \min\{e_x\}, \quad x_{max} = x_0 + \sum_{i=1}^m |x_i| + \max\{e_x\} \quad (7.12)$$

considering e_x being the offset interval of the affine form $\tilde{x}$.

Affine Arithmetic Decision Diagrams

Affine arithmetic covers only analog behavior as an affine form represents a coherent interval by definition. In the context of mixed-signal simulations, discontinuous operations are required. Such discontinuities may be introduced by means of if-else branching, the use of affine loop conditions or other means. To illustrate the issue, consider the following example. A comparison $\tilde{x} < \tilde{y}$ between two affine forms, designated as $\tilde{x}$ and $\tilde{y}$, has three possible results: The result is *true* when the range of $\tilde{x}$ lies entirely below the range of $\tilde{y}$, such that the upper limit of $\tilde{x}$ is less than the lower limit of $\tilde{y}$ (Fig. 7.1a). Conversely, it is evaluated as *false* when the lower limit of $\tilde{x}$ is equal to or greater than the upper limit of $\tilde{y}$ (Fig. 7.1b). In all other cases, the result is unknown and referred to as $X = \{false, true\}$ because both cases are possible, depending on the actual values of the deviation symbols (Fig. 7.1c).

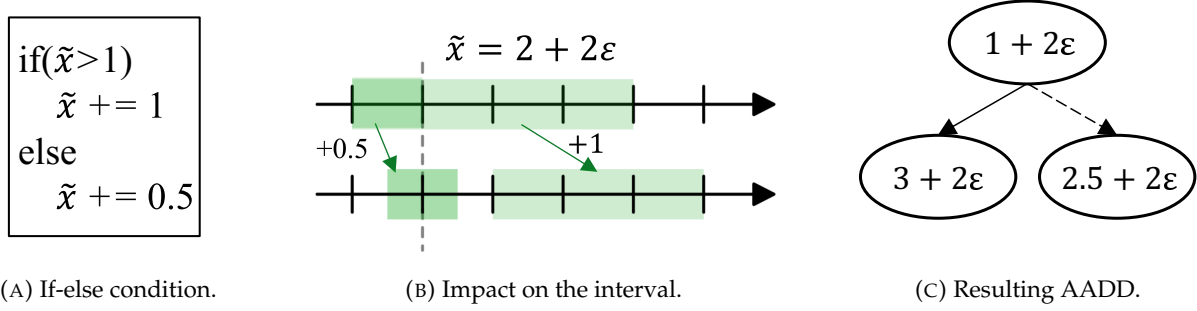


FIGURE 7.2: Example of an AADD. The if-else condition can evaluate to both, true or false, depending on the actual value of the deviation symbol. An AADD is created that reflects both options and splits the initially coherent interval into two distinct ones. The internal node reflects the corresponding condition. The explicitly stated condition ($\tilde{x} > 1$) is thereby transformed to ($\tilde{x} - 1 > 0$) within the AADD and implied by the affine form $\tilde{x} - 1$. © IEEE 2024.

To expand the range of applications of affine simulations, Radojicic and Grimm have proposed an extended affine arithmetic (XAA) that incorporates supplementary terms

$$\omega_i \in \{-1, 1\}, i \geq 1 \quad (7.13)$$

representing discontinuous uncertainties [176]. More recent works adopt an alternative approach. They employ the use of Affine Arithmetic Decision Diagrams (AADDs) [177]. The term defines ordered binary trees where each node refers to an affine form. The leaves of an AADD represent the potential outcomes (intervals) of the current evaluation of the given affine form whereas the internal nodes reflect the decisions that have been made in order to reach this interval. If an internal node contains the affine form $\tilde{z}$, this represents the condition $\tilde{z} > 0$. Fig. 7.2 illustrates the working principle at the example of a simple if-else condition. The result is only *true* if the deviation symbol is greater than -0.5 . Otherwise, the boolean expression evaluates to *false*. Consequently, both results are possible and the initially coherent interval $\tilde{x}$ splits into two distinct intervals that reflect the different branches. The resulting AADD consists of one internal node containing the affine form $\tilde{x} - 1$ to reflect the condition $\tilde{x} - 1 > 0$ and two leaves that represent the two distinct intervals. The thesis at hand uses AADDs to represent discontinuities due to the higher flexibility compared to XAA.

7.2 Existing Fields of Application

Although the field of affine arithmetic has a long history, it has only recently begun to garner significant attention within the context of electronic system-level simulations. In 2011, Schupfer et al. proposed a refinement flow that takes range-based system models as a starting point for system analyses and utilizes the results to refine the system parameters [178]. Repeating this process with different parameters and deviation ranges can enhance the system's resilience. One year later, Radojicic et al. proposed to perform verification combining affine arithmetic and assertions [179]. In the following years, the corresponding research group developed various strategies for modeling discontinuities [180], both with extended affine arithmetic [176] and AADDs [175]. Furthermore, Zivkovic et al. have thereby also discussed a methodology to trigger discrete events based on symbolic activation conditions [181]. Thereby, AADDs are employed such that all assignments executed within the activated process are solely valid within the branch that initiated the activation. Symbolic time values have not been addressed in literature so far. Moreover, they have refined their verification approach towards a hierarchical

scheme [182]. Verification of the components is conducted through the application of either formal methods or multirun methods (e.g. Monte-Carlo analysis). The knowledge gained about the components is then transferred to the system level by setting realistic ranges for the parameters used within the system-level model that is instrumented with affine arithmetic. Subsequently, system-level properties can be derived from the results of the range-based simulation. Other researchers have taken over the arithmetic as well. In 2017, Ding et al. [183] conducted worst-case analyses using affine arithmetic. They point out that sampling-based methods, such as Monte-Carlo simulation, only yield results for a subset of potential outcomes. Consequently, these methods are unable to reliably identify the true worst-case boundaries within a finite time frame. Therefore, the range encompassed by these methods is always slightly narrower than the actual worst-case scenario. Affine arithmetic approximates the worst-case boundaries from an alternative perspective, namely that of over-approximation, which introduces a degree of safety margin. Nevertheless, the degree of overestimation is less pronounced than that observed in interval arithmetic. A particular focus of their work is a worst-case analysis within the frequency domain. In 2020, Esteban et al. employed affine arithmetic to estimate the round-off noise in fixed-point algorithms [184]. In the same year, Lauter and Volkova applied semi-symbolic simulation onto models using floating-point arithmetic [185]. The objective is to analyze the floating-point error introduced in the context of Deep Learning. A completely different approach has been presented by Tarraf and Hedrich in 2021 [186]. The authors focus on a bottom-up approach that generates abstracted SystemC AMS models from SPICE netlists. The method relies on state-space sampling. To secure the abstraction, modeling errors are considered using affine forms such that the resulting ranges envelope the potential behaviors of transistor level simulations.

The following section introduces a novel application. This new approach differs from previously conducted works in that it does not only use affine arithmetic for range analysis, but also specifically harnesses the existence of deviation symbols to enable generic compensation that works independent of the actual value of the deviation term.

7.3 Application to Sensor Trimming

The thesis at hand uses AADDs to semi-symbolically simulate trimming algorithms. Simulations are performed using SystemC AMS and the open-source AADD library for C++ [187]. In the following, the approach is explained at the example of a simple charge-to-voltage converter trim. The simplicity of the example allows for a detailed analysis of the simulation's behavior through the lens of underlying analytical calculations, as well as a comprehensive explanation of the working principle. Thereby, benefits and limitations of the approach are discussed.

Fig. 7.3 illustrates the initial situation. The device under test is a MEMS accelerometer. The movement of the suspended mass influences a differential capacitance ΔC . Ideally, the difference is zero at rest. In reality, the MEMS is subject to manufacturing variations and the differential capacitance will most likely be non-zero at rest. Hence, it can be modeled as

$$\Delta C|_{a=0} = C_{off,0} + \varepsilon_1 C_{off,1} \quad (7.14)$$

which expresses the uncertainty of the actual offset value due to process variations. The level of deflection in presence of an acceleration depends on the sensitivity of the system. The sensitivity is subject to variations as well. Here, the effects are considered as uncorrelated, thus a second deviation symbol ε_2 is inserted and the overall formula for ΔC derives to

$$\Delta C = C_{off,0} + \varepsilon_1 C_{off,1} + (S_0 + \varepsilon_2 S_2)a, \quad (7.15)$$

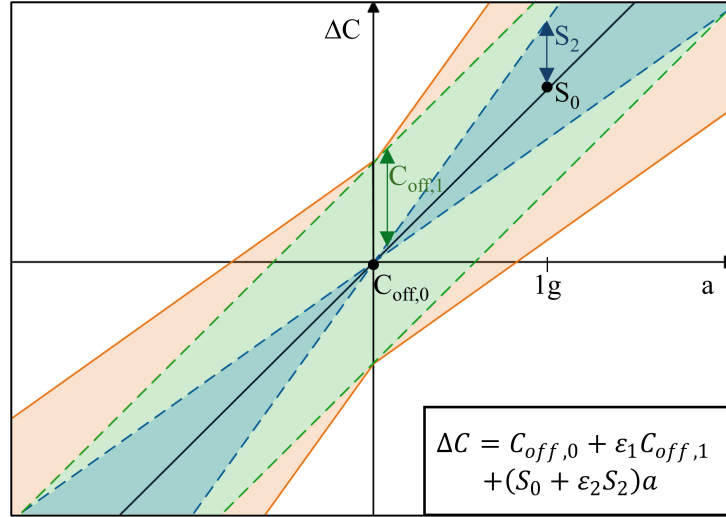


FIGURE 7.3: Differential capacitance of a MEMS accelerometer in the presence of process variations. The black line represents the ideal behavior. Superposing the potential offset error (green) and sensitivity error (blue) yields the total range given in orange. © IEEE 2024.

with $S := dC/da$ denoting the sensitivity and a being the acceleration. The colored fields in Fig. 7.3 illustrate the resulting range of the differential capacitance. It should be noted that additional effects, such as nonlinearity, have been omitted for the sake of simplicity.

The variations induced by the MEMS shall be compensated by the consecutive CV stage. The high-level input-output relation of the CV converter can be considered as

$$V_{cv} = V_{sup} \frac{\Delta C - C_{otr}}{C_{fb}} = V_{sup} \frac{\Delta C - k_{otr} C_{otr,lsb}}{C_{fb,fix} + k_{fb} C_{fb,lsb}}, \quad (7.16)$$

with V_{sup} denoting the supply voltage, C_{otr} the offset trim capacitance and C_{fb} the feedback capacitance that configures the gain. Both capacitances are realized as binary weighted capacitor banks and their switches are controlled by corresponding trim words (k_{otr} , k_{fb}). Hence, their values per least significant bit (LSB) are used. The feedback capacitance additionally consists of a fixed capacitor $C_{fb,fix}$ that is always connected and limits the maximum gain. The steps required to find the correct trim words are given below. Formulas indicated in blue are those that are specifically presented for explanatory purposes and are typically processed internally by the simulation or derive from measurements when using a physical sensor.

1. **Initial configuration:** Set k_{otr} and k_{fb} to zero.
2. **Offset measurement:** Measure the output voltage of the CV stage at rest resulting in

$$V_0 := V_{cv}|_{a=0, k_*=0} = V_{sup} \frac{\Delta C|_{a=0}}{C_{fb,fix}} = V_{sup} \frac{C_{off,0} + \varepsilon_1 C_{off,1}}{C_{fb,fix}}. \quad (7.17)$$

3. **Offset trim:** Calculate and apply the offset trim word

$$k_{otr} = \frac{C_{fb,fix}}{V_{sup} C_{otr,lsb}} V_0 = \frac{C_{off,0} + \varepsilon_1 C_{off,1}}{C_{otr,lsb}}. \quad (7.18)$$

The offset trim word is derived based on the expected behavior of the CV stage according to eq. (7.16) and the measurement of V_0 .

4. **Offset trim check, part 1:** Measure the output voltage of the CV stage a second time and verify that it has been reduced to zero:

$$\begin{aligned} V_{cv}|_{a=0, k_{fb}=0} &= V_{sup} \frac{\Delta C - k_{otr} C_{otr,lsb}}{C_{fb,fix}} = V_{sup} \frac{C_{off,0} + \varepsilon_1 C_{off,1} - \frac{C_{off,0} + \varepsilon_1 C_{off,1}}{C_{otr,lsb}} C_{otr,lsb}}{C_{fb,fix}} \\ &= 0 \\ &\neq f(\varepsilon_1). \end{aligned} \quad (7.19)$$

5. **Offset trim check, part 2:** Check the interval of k_{otr} to validate that it fits into the provided bit length of its hardware realization.

Up to this point, the offset trim has been applied successfully. Although the initial capacitive offset has been subject to uncertainty, the evaluation states that the trimmed output voltage will be zero **in all cases**, irrespective of the actual value of the deviation symbol. In this context, the strength of the deviation symbols becomes evident. As the trim word is calculated based on measurement results that contain the deviation symbol ε_1 , the trim word itself contains the aforementioned deviation symbol as well. Consequently, the effects of ΔC and k_{otr} are mutually cancelling, obviating the necessity for knowledge of the actual value of the initial offset.

An additional verification of the interval boundaries of k_{otr} is required to ascertain whether the current implementation is indeed capable of encompassing the full range of input variations. Within the semi-symbolic simulation run, the trim word is represented as a real value. In practice, however, it is represented by a fixed number of bits. For example, a 5-bit trim word can represent any integer between 0 and 31. Accordingly, the real-valued version of k_{otr} should not exceed the interval $[0, 31.5)$ for the trim circuit to be considered as correctly dimensioned. The interval was selected such that rounding would yield valid trim values. Negative values are not considered, as the polarity is typically addressed through the use of an additional sign bit. In the given example, the correctness of the trimming algorithm is examined and the quantization error is ignored to keep the results easily interpretable. Section 7.5 introduces a rounding function that enables to analyze the granularity of the trim circuit by adding a new deviation symbol to account for the rounding error.

To return to the example, the next step is to perform a sensitivity trim based on the following steps:

6. **Sensitivity evaluation:** Apply an external stimulus to the system (e.g. $a = 1g$) and measure the output voltage of the CV converter. Then, the sensitivity is calculated to:

$$\frac{V_{cv}|_{a=1g} - V_{cv}|_{a=0g}}{1g} = V_{sup} \frac{S_0 + \varepsilon_2 S_2}{C_{fb,fix}}. \quad (7.20)$$

7. **Sensitivity trim:** Calculate and apply trim word k_{fb} to reach the target sensitivity.

$$\frac{V_{cv}|_{a=1g, k_{fb}=0}}{V_{cv}|_{a=1g, target}} = \frac{C_{fb,fix} + k_{fb} C_{fb,lsb}}{C_{fb,fix}} = 1 + k_{fb} \frac{C_{fb,lsb}}{C_{fb,fix}} \quad (7.21)$$

$$\begin{aligned} \Rightarrow k_{fb} &= \left(\frac{V_{cv}|_{a=1g, k_{fb}=0}}{V_{cv}|_{a=1g, target}} - 1 \right) \frac{C_{fb,fix}}{C_{fb,lsb}} \\ &= \left(\frac{S_0 + \varepsilon_2 S_2}{S_0} - 1 \right) \frac{C_{fb,fix}}{C_{fb,lsb}} \\ &= \frac{\varepsilon_2 S_2}{S_0} \frac{C_{fb,fix}}{C_{fb,lsb}} \end{aligned} \quad (7.22)$$

8. **Sensitivity trim check, part 1:** Measure the trimmed circuit in presence of stimulus and compare the result to the target.

$$\begin{aligned}
 V_{cv}|_{a=1g} &= V_{sup} \frac{S_0 + \varepsilon_2 S_2}{C_{fb,fix} + \frac{\varepsilon_2 S_2}{S_0} \frac{C_{fb,fix}}{C_{fb,lsb}}} \cdot 1g \\
 &= V_{sup} \frac{S_0 + \varepsilon_2 S_2}{C_{fb,fix} (1 + \varepsilon_2 \frac{S_2}{S_0})} \cdot 1g \\
 &\stackrel{\text{C.1}}{=} V_{sup} \frac{S_0}{C_{fb,fix}} \cdot 1g \neq f(\varepsilon_2) \\
 &= V_{cv}|_{a=1g, target}
 \end{aligned} \tag{7.23}$$

9. **Sensitivity trim check, part 2:** Check the interval of k_{fb} to validate that it fits into the provided bit length of its hardware realization.

Once more, the outcome no longer incorporates any deviation symbols, and the sensitivity trim has been successful. All of the aforementioned steps are inherently performed using a single transient simulation run. Thereby, the conducted checks can be implemented as assertions. Using a single semi-symbolic simulation, the CV converter trimming has been validated for the complete state space spanned by offset and sensitivity errors. In the case of more complex algorithms, this approach helps to minimize the risk of overlooking corner cases, which can easily happen if only a limited number of concrete test cases are simulated. Fig. 7.4 illustrates the simulation results from the example given above. Both trim steps serve to reduce the possible output voltages to concrete values (the target), thereby suppressing the initial variations.

So far, it has been discussed in which manner affine arithmetic helps to validate the correctness of a trimming procedure. Beyond that property, the arithmetic also supports the debugging process if the trimming has not (fully) been successful. To illustrate the effect, the preceding example has been slightly modified as highlighted in purple in the subsequent formulas. It is assumed that the CV converter itself introduces an offset voltage that does not scale with the feedback term, such that its new behavior is given by

$$V_{cv} = V_{sup} \frac{\Delta C - k_{otr} C_{otr,lsb}}{C_{fb,fix} + k_{fb} C_{fb,lsb}} + V_{off,0} + \varepsilon_3 V_{off,3}, \tag{7.24}$$

with $V_{off,i}$ being the components of the affine offset voltage. As a result, the offset trim word k_{otr} evaluates to

$$k_{otr} = \frac{C_{off,0} + \varepsilon_1 C_{off,1}}{C_{otr,lsb}} + \frac{C_{fb,fix}}{V_{sup} C_{otr,lsb}} (V_{off,0} + \varepsilon_3 V_{off,3}), \tag{7.25}$$

getting an additional term due to the modified outcome for the initial measurement of V_0 . Up to now, the offset trim is once more successful as the additional term cancels out the newly introduced offset voltage. Neither of the deviation symbols occurs in the output voltage at rest. However, the new term contains the untrimmed feedback capacitance. Consequently, the sensitivity trim disturbs the offset trim and the final output voltage of the CV converter without

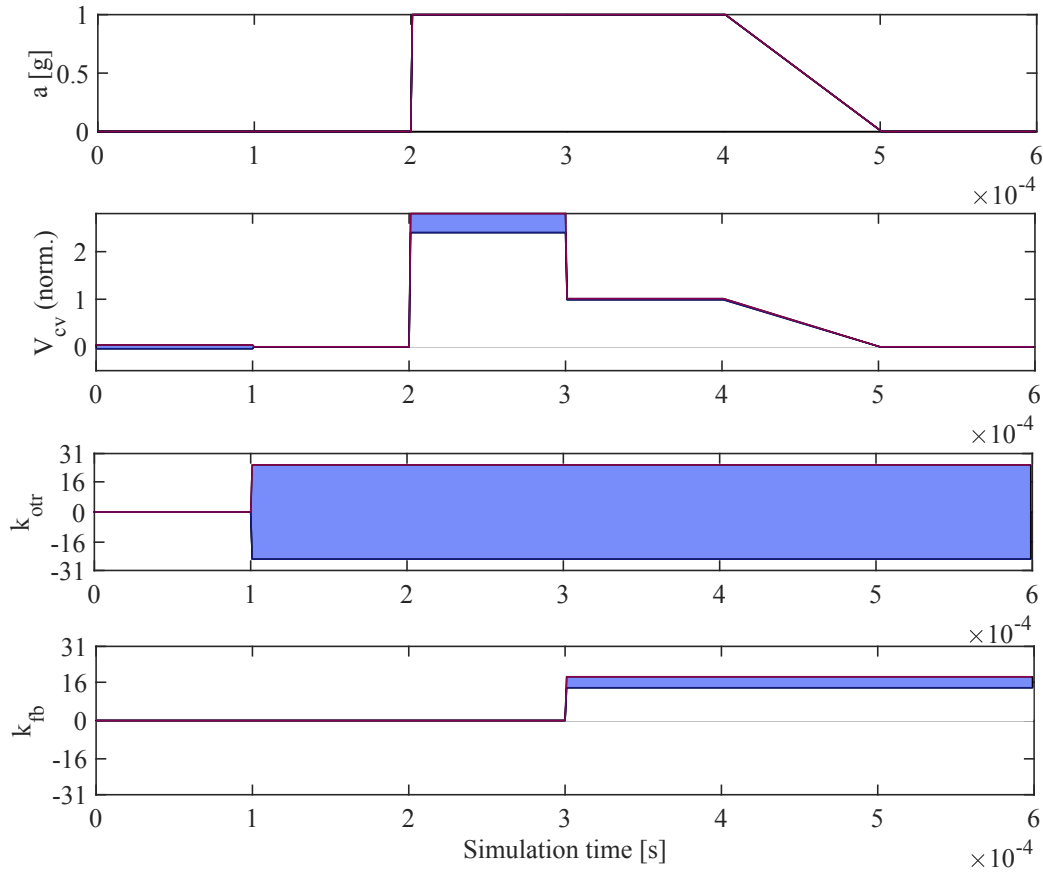


FIGURE 7.4: Simulation results of a CV converter offset and sensitivity trim. Colored areas describe the intervals spanned by affine forms. The CV output voltage is normalized to its target at $a = 1\text{g}$. Both, initial offset and sensitivity are subject to uncertainty. After trimming, the normalized output voltage becomes zero at rest and one at $a = 1\text{g}$, such that their concrete target values are reached for both cases, and dependencies on deviation symbols are suppressed. This clearly identifies the success of the trimming procedure.

any stimulus yields

$$\begin{aligned}
 V_{cv}|_{a=0\text{g}} &= V_{sup} \frac{-\frac{C_{fb,fix}}{V_{sup}C_{otr,lsb}}(V_{off,0} + \varepsilon_3 V_{off,3})C_{otr,lsb}}{C_{fb,fix} + k_{fb}C_{fb,lsb}} + V_{off,0} + \varepsilon_3 V_{off,3} \\
 &= (V_{off,0} + \varepsilon_3 V_{off,3}) \left(1 - \frac{C_{fb,fix}}{C_{fb,fix} + k_{fb}C_{fb,lsb}} \right) \\
 &= (V_{off,0} + \varepsilon_3 V_{off,3}) \frac{k_{fb}C_{fb,lsb}}{C_{fb,fix} + k_{fb}C_{fb,lsb}} \\
 &= f(\varepsilon_3).
 \end{aligned} \tag{7.26}$$

Here, the characteristics of affine arithmetic facilitate efficient debugging. The final output voltage contains the deviation symbol ε_3 , but does not include the MEMS related uncertainties. Consequently, the user is directed to the source of the error, and the analysis is conducted with greater precision, thereby reducing the time required. Once more, the scenario has been simulated. The simulation results are presented in Fig. 7.5. Tracing is designed such that it

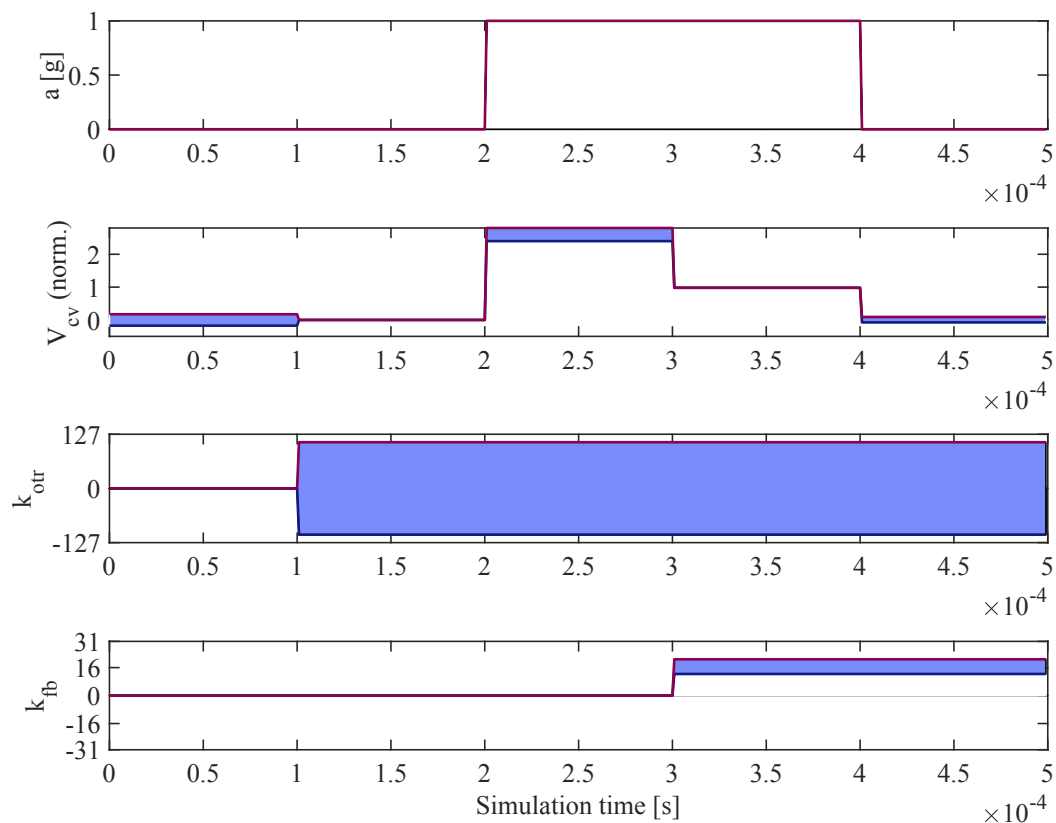


FIGURE 7.5: Simulation results of a CV converter offset and sensitivity trim. Colored areas describe the intervals spanned by affine forms. The CV output voltage is normalized to its target at $a = 1\text{g}$. In contrast to Fig. 7.4, the offset trim is disturbed by the subsequent sensitivity trim. Hence, the output voltage of the CV stage still spans an interval at the end of the simulation indicating a residual error.

traces the interval boundaries. An additional printout option of the AADD library enables the user to investigate which deviation symbols are present and to divide the interval into its contributors.

All in all, the semi-symbolic simulation of trimming algorithms provides the developer with comprehensive support throughout the entire development process, offering three principal contributions:

1. **Error propagation analysis:** To design an effective and robust algorithm, the propagation mechanism of uncertainties must be understood first. Here, affine arithmetic facilitates to monitor the propagation of deviation symbols throughout the system.
2. **Identification of error sources:** If a trimming algorithm does not yield the desired result, the contributing deviation symbols clearly identify the error source.
3. **Validation:** If the trimming algorithm yields the desired result, the level of confidentiality is higher than that of a concrete test case. This is due to the fact that the semi-symbolic simulation has inherently covered areas of the analog state space, as opposed to focusing on single points.

The latter contribution will additionally be demonstrated in the experimental results section. There, a binary search is presented that makes use of decision diagrams and identifies all possible trajectories within a single simulation run.

7.4 Limitations

The previous example has shown that affine arithmetic is able to effectively support the development and validation of trimming procedures. However, the approach is subject to a set of limitations. Their sources can be divided up into arithmetic specific limitations and implementation specific limitations. Both of them are discussed in this section.

Arithmetic Specific Limitations

Affine arithmetic is, by definition, a linear arithmetic that allows for the inclusion of deviation terms up to first order only. Therefore, the constraints imposed by this approach are largely determined by the linearization process. In accordance with Hansen's formulation, each non-linear deviation term is ultimately incorporated into the offset interval. If a huge number of transient steps is performed, these errors will cumulate, leading to an increase in the degree of overestimation. For the majority of trimming algorithms, the accumulation of errors is not a big issue. Deviation terms are actively canceled, and the number of non-linear operations remains within reasonable limits. Nevertheless, particular attention must be devoted to the linearization of the functions executed in each simulation step. Especially for relatively large deviation symbols, these linearizations can induce errors that have a discernible impact on the behavior of the simulation. For instance, trimming the analog voltage-controlled oscillator (VCO) of a MEMS gyroscope using semi-symbolic simulation is an ineffective method. The scenario is as follows. The oscillating voltages that actuate the drive of a MEMS gyroscope are generated using an analog VCO, which produces a sinusoidal wave. The VCO output frequency follows the equation

$$f_{VCO} = f_0 + k_{ctrl}v_1 + k_{trim}v_2, \quad (7.27)$$

with v_1 being the output voltage of a PI controller dynamically regulating the VCO frequency. The second term enables the adjustment of the center frequency of the VCO. When the gyroscope is enabled prior to trimming, the phase-locked loop (PLL) is able to lock due to the influence of the control term $k_{ctrl}v_1$. The time required for the PLL to reach the locked state is referred to as the start-up time. Subsequently, a VCO trim is performed, which shifts the center frequency of the VCO in proximity to the MEMS resonance frequency. This results in v_1 becoming close to zero in the locked state. The objective is to maximize the margin for potential frequency drifts of the MEMS that the PLL can still track. Such drifts may result from changes in temperature or stress levels, for instance. In order to simulate the trim step with affine arithmetic, the MEMS resonance frequency must be subject to uncertainties. This results in the generation of uncertain control voltages v_1 in order to reach the locked state. Subsequently, the appropriate (affine) trim word can be evaluated in theory. However, the VCO is required to continuously approximate the sinusoidal oscillation, that is to say, it must apply a sinusoid to an affine form. The AADD library provides a function that approximates the sinusoid based on the least mean squares method [187]. Figure 7.6 illustrates the resulting intervals for a single deviation symbol with weight (a) $\pi/16$ and (b) $\pi/4$. While the phase shift is clearly visible, the amplitude of the sinusoidal function is overestimated. In the case of minor uncertainties, the magnitude of the interval keeps close to the boundaries $[-1, 1]$. However, in instances of greater variation, it begins to exceed these limits to a considerable extent. The use of an uncertain frequency will result in an increase in the phase shift over time, which will ultimately

lead to a significant issue. This effect reduces the mathematical plausibility of the solution and disturbs the amplitude control.

Nonlinearities that are less invasive, such as the division by the feedback capacitance in equation (7.16), do not impede the use of affine arithmetic. The primary limitation of nonlinear operations is the absence of root cause tracking. If two deviation symbols interfere with one another, the resulting higher-order terms invariably end up in the offset term. This ultimately results in the loss of the root cause, that is, the information pertaining to the specific deviation symbols that have interfered to create this offset. Therefore, root cause tracking is only applicable to first-order terms. It should be noted that this issue is not exclusive to Hansen's form; the same phenomenon occurs when a new deviation symbol is introduced.

Implementation Specific Limitations

A fundamental restriction associated with the utilization of a particular arithmetic is that it precludes the employment of numerous standard libraries that are accessible for C++. Consequently, all operators that are required for the new arithmetic must be implemented specifically. In the present case, the control structures are also redesigned. The present thesis is largely based on the functionalities available in the existing AADD library [187]. In order to accommodate specific use cases, a number of operators has been added. The aforementioned elements are introduced in the following section.

In addition to this apparent limitation, the scalability of the approach must be taken into account. Affine arithmetic decision diagrams offer a means of overcoming the limitations of purely analog simulations, but their use requires careful consideration. A typical transient simulation comprises thousands or even millions of simulation steps. The execution of decisions and loops in each and every step makes the size of these trees explode easily. It is therefore necessary to implement appropriate reduction rules. Thereby, many concepts familiar from binary trees can be applied to AADDs in a similar manner [177]. The existing AADD library implements some simple reduction measures but still suffers from large and redundant trees. The definition and implementation of an exhaustive set of reduction rules for AADDs is beyond the scope of this work. The thesis at hand aims to prove the extensibility of HVPs beyond purely numerical simulations and to identify and enable corresponding use cases. The present work does not address graph theory in general nor tree data structures in particular.

7.5 Proposed Extensions

In the context of trimming algorithms, second-order terms warrant particular attention for a number of reasons. Firstly, it is more likely that a developer will fail to identify a deviation that originates from system-spanning interferences than one that is the result of a simple propagation of a single deviation. Therefore, semi-symbolic simulation is of greater benefit in this context. Secondly, second-order terms will invariably arise when both the uncertainty associated with the property to be trimmed and the uncertainty inherent in the trim circuit itself are taken into account. The author of this thesis thus proposes an extension of Hansen's form of affine arithmetic towards second-order root cause tracking. This is referred to as AA2. Furthermore, additional operators have been incorporated into the AADD library to align with the specific requirements of trimming algorithms.

AA2: Second-Order Root Cause Tracking

The objective of AA2 is to facilitate second-order root cause tracking while maintaining the fundamental simplicity of range analysis. A central tenet of affine arithmetic is that the boundaries of intervals are determined by the simple addition or subtraction of each partial deviation

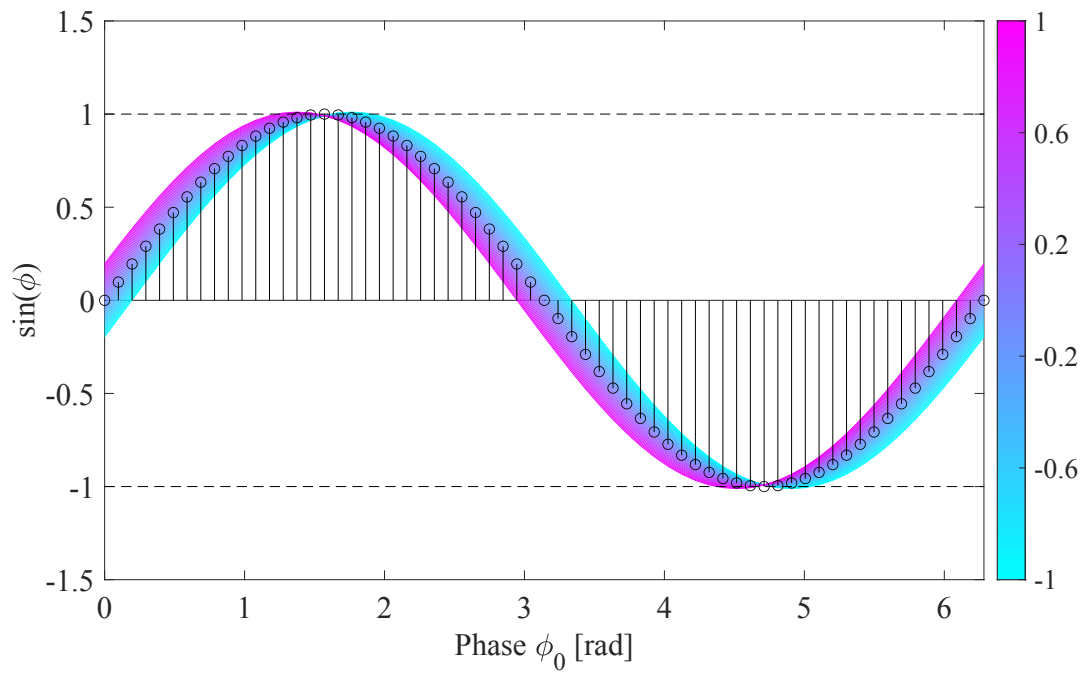
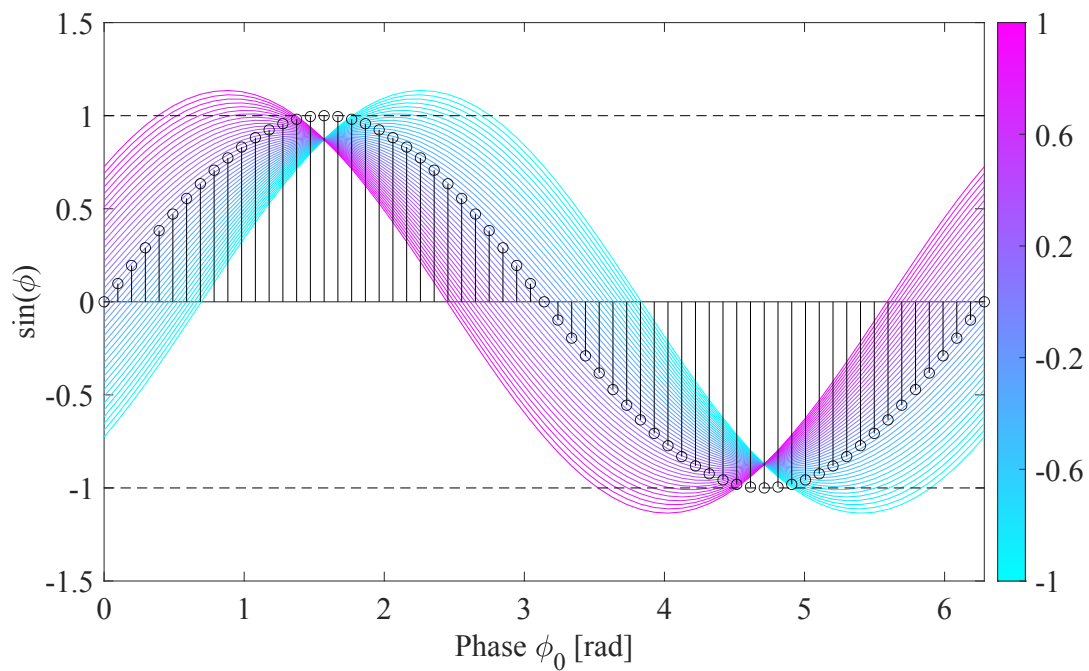
(A) Resulting uncertainty for $\phi_1 = \pi/16$.(B) Resulting uncertainty for $\phi_1 = \pi/4$.

FIGURE 7.6: Affine approximation of the sinusoidal function. The phase $\phi = \phi_0 + \epsilon_1 \phi_1$ includes one deviation term. The color map represents the value of the deviation symbol ϵ_1 . For large partial deviations, the interval boundaries significantly exceed the nominal amplitude of a sinusoidal.

as given by Eq. (7.4). AA2 adheres to the concept of Hansen's form, whereby each nonlinear term is incorporated into an overall offset term. However, the author of the thesis at hand proposes an alternative implementation of the offset term, which is represented not only by a pair of numbers denoting the minimum and maximum offset, but rather by a matrix that identifies the sources of subterms. To be precise, the offset term transforms to an upper triangular matrix E given by

$$E = \begin{pmatrix} e_{11} & \dots & e_{1m} \\ & \ddots & \vdots \\ \mathbf{0} & & e_{mm} \end{pmatrix}, \quad (7.28)$$

where e_{ij} represents the partial offset introduced by the product of ε_i and ε_j . Doing so, evaluating the interval boundaries keeps simple as they derive to

$$x_{min} = x_0 - \sum_{i=1}^m |x_i| - \sum_{i=1}^m \sum_{\substack{j=1 \\ j \neq i}}^m |e_{ij}| + \sum_{i=1}^m \chi_i e_{ii} \quad (7.29)$$

$$x_{max} = x_0 + \sum_{i=1}^m |x_i| + \sum_{i=1}^m \sum_{\substack{j=1 \\ j \neq i}}^m |e_{ij}| + \sum_{i=1}^m (1 - \chi_i) e_{ii} \quad (7.30)$$

$$\text{with } \chi_i := \begin{cases} 0 & \text{if } e_{ii} > 0 \\ 1 & \text{else} \end{cases}. \quad (7.31)$$

The approach has two dedicated benefits. Firstly, it enables a clear identification of deviation symbols that caused significant offsets, therewith supporting a more effective analysis of simulation results. If the interference of two arbitrary deviation symbols creates much higher offset terms than the other combinations do, the developer should evaluate whether modifications could be made to the system or the trimming algorithm to enhance its resilience to this specific form of cross-coupling.

Secondly, the degree of overapproximation is diminished to a slight extent. Upon examination of a single nonlinear approximation, such as a multiplication as introduced in Eq. (7.11), the level of overestimation remains the same as for the original form. However, the sign of each particular addend e_{ij} is stored, such that offsets do not necessarily add up, but can also compensate each other. To illustrate the effect, consider the following example, which employs three affine forms

$$\begin{aligned} \tilde{x} &= x_0 + \varepsilon_1 x_1 \\ \tilde{y} &= y_0 + \varepsilon_2 y_2 \\ \tilde{z} &= z_0 + \varepsilon_2 z_2 \end{aligned} \quad (7.32)$$

where $\tilde{y}$ and $\tilde{z}$ are correlated and depend on the same deviation symbol. The equation

$$\tilde{a} = \tilde{x} \cdot \tilde{y} + \tilde{x} \cdot \tilde{z} \quad (7.33)$$

shall be evaluated. Using classical affine arithmetic and evaluating multiplications first and the addition secondly, the final result will be

$$\begin{aligned}\tilde{a} &= \left(x_0 y_0 + \varepsilon_1 y_0 x_1 + \varepsilon_2 x_0 y_2 + [-|x_1 y_2|, |x_1 y_2|] \right) + \left(x_0 z_0 + \varepsilon_1 z_0 x_1 + \varepsilon_2 x_0 z_2 + [-|x_1 z_2|, |x_1 z_2|] \right) \\ &= \underbrace{x_0 y_0 + y_0 z_0}_{=:a_0} + \varepsilon_1 \underbrace{(y_0 x_1 + z_0 x_1)}_{=:a_1} + \varepsilon_2 \underbrace{(x_0 y_2 + x_0 z_2)}_{=:a_2} + [-|x_1 y_2| - |x_1 z_2|, |x_1 y_2| + |x_1 z_2|],\end{aligned}\tag{7.34}$$

where the offsets of each multiplication have cumulated. Both offsets originate from a term that, if allowed, contained the product $\varepsilon_1 \varepsilon_2$. Therefore, AA2 identifies their correlation leading to a final result

$$\tilde{a}_{AA2} = a_0 + \varepsilon_1 a_1 + \varepsilon_2 a_2 + [-|x_1 y_2 + x_1 z_2|, |x_1 y_2 + x_1 z_2|]\tag{7.35}$$

which is equal to the one above if y_2 and z_2 have the same sign but smaller otherwise as the offsets partially compensate each other (triangle inequality). The example is illustrated in Fig. 7.7. The result of the product $\tilde{a}$ is plotted against the two deviation symbols. If the rules of affine arithmetic are ignored and the exact result is calculated, the surface is slightly curved. Using the arithmetic with or without extension, the surfaces become flat because second-order terms become part of the offset which is handled independent of the deviation symbols. In the given example, $y_2 = 0.5$ and $z_2 = -0.8$ are chosen such that their effects partially compensate each other. As a result, the overestimation of AA2 is significantly smaller than for classical AA. In this particular example, the interval boundaries even match the minimum/maximum values of the exact result.

In conclusion, AA2 offers two forms of support. On the one hand, it improves the root cause tracking by extending it towards second-order terms. On the other hand, it reduces the degree of overestimation of intervals. The increase in computational cost thereby keeps small. Evaluating the interval boundaries requires to sum the $\frac{m(m+1)}{2}$ offset terms first with m being the number of deviation symbols. Moreover, the calculation of offsets within a nonlinear function is somewhat more costly. If the approximation is based on a Taylor series, for instance, it is necessary to evaluate the second-order term in order to derive the corresponding offset terms within the matrix. For illustrative purposes, a number of approximations for AA2 are provided in appendix C.2.

Special Operators

In the following, two operators are described that have been added to the existing AADD library [187]. They address particular needs within the context of trimming simulation.

Rounding The representation of trim words as ranges requires a final step of rounding to account for the finite granularity of the trim circuit. The maximum rounding error for any given real value is 0.5. Hence, the default rounding method given in the AADD library adds ± 0.5 to the offset interval. For this particular operation, the thesis at hand proposes the to use the alternative approach to AA, which involves introducing a new deviation symbol. This approach allows for the differentiation between approximation errors introduced by the arithmetic and rounding errors occurring due to the trim circuit. The final result of a trim simulation then reveals the granularity of the trimming and can be validated against the specification. Please note that a new deviation symbol is introduced only if the rounding of the minimum and maximum of an affine form yield distinct integers. Otherwise, as for example for an affine form $\tilde{x} = 3.2 + 0.2\varepsilon_1$, all deviation terms are dropped and rounding results in a concrete value (here:

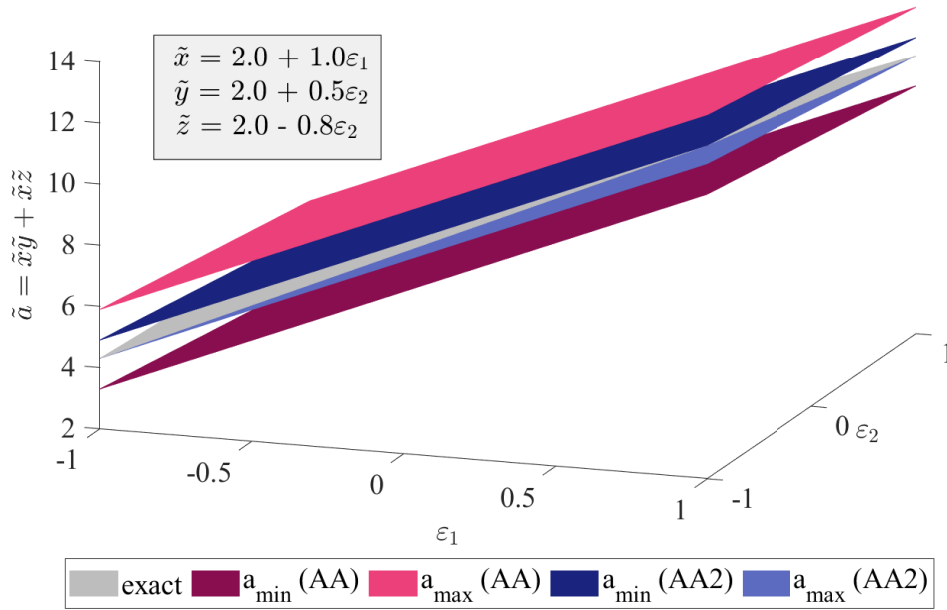


FIGURE 7.7: Affine summation of summands containing offset terms. The exact solution is given as reference. It does not cut off the second order terms originating from the multiplications. The subscripts *min* and *max* refer to the interval boundaries taken for the offset term when either using affine arithmetic in classical Hansen’s form (AA) or with the proposed extension (AA2). The offset interval is narrower for AA2 because it takes the correlation between the offset terms resulting from $\tilde{x}\tilde{y}$ and $\tilde{x}\tilde{z}$ into account which partially compensate each other.

$\text{round}(\tilde{x}) = 3$). In case of an AADD, the rounding is applied to each leaf. If multiple leaves end up in the same affine form, reduction rules can be applied. The implementation of the function for a single leaf or affine form is given in appendix C.3.

Bit Access In the simple example given above, the trim words control capacitor banks with binary ladder structure. Consequently, they can be regarded as a single word, with all bits adhering to a uniform semantic structure. It should be noted, however, that other encodings exist, including thermal encoding and combinations of multiple differently dimensioned ladder structures for instance. In these cases, it is necessary to extract single bits or partial bit sequences from a trim word. The author of the thesis at hand proposes an AADD based version of a bit access operator for that purpose. A function $x.\text{getBit}(N, i)$ has been introduced that interprets an affine form $\tilde{x}$ as N -bit word and extracts the i^{th} bit out of it. If the interval boundaries of $\tilde{x}$ exceed the range of an N -bit word, an error is thrown. If $x_{\max}$ is smaller than 2^{i-1} , the result is 0. Otherwise, an AADD is created where each leaf has either the value 1 or 0. Internal conditions are designed such that the rounded values within each interval result in the value given in the dedicated leaf. For a detailed documentation of the implementation, please refer to the appendix C.3.

7.6 Case Study: Binary Search

In order to demonstrate the efficacy of the approach for more intricate algorithms, a binary search has been performed. The scenario is once more a sensitivity trim using the feedback

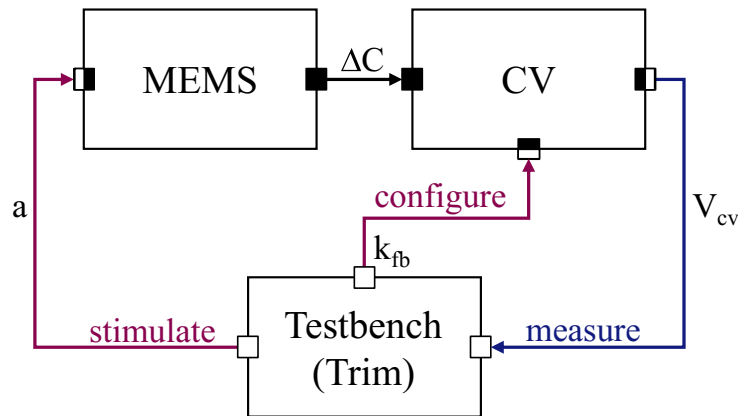


FIGURE 7.8: Simulation setup for the binary search trim. The testbench module replaces the hardware of the laboratory. It provides stimuli and executes the trim procedure. The MEMS model is the source of uncertainties which are then compensated by the trim circuitry of the charge-to-voltage converter. The complete model is written in SystemC/-AMS using affine data types and symbolic control structures from the AADD library.

capacitance of a CV stage. The simulation setup is given in Fig. 7.8. The system is comprised of three principal components. The MEMS is implemented in SystemC AMS dataflow semantics and its sensitivity to the acceleration stimulus is subject to uncertainty. Similarly, the CV converter is implemented in SystemC AMS dataflow semantics and is designed to compensate for deviations in order achieve nominal sensitivity for its output voltage. The trim procedure is encapsulated in the testbench module. The testbench stimulates and monitors the system under test, comprising the MEMS and CV stage. To be precise, it supplies the acceleration and the trim word, which would otherwise be provided by a register within the digital part of the system that is not modeled here. The testbench is implemented as a SystemC module such that sequential operations can be defined. Each time a new stimulus or trim configuration has been applied, a *wait* statement is employed to allow for the system under test to reach a steady state before the next measurement is taken.

The trimming procedure works as follows: The feedback capacitance is adjusted by means of a 6-bit trim word denoted as k_{fb} . Accordingly, the preliminary trim word is 32. It should be noted that this is a specific, concrete value. Similarly, the target output level is also a concrete value. In the simulation results given in Fig. 7.9, the output voltage is normalized such that the target evaluates to one. The binary search compares the actual output voltage to the target and checks whether it is

- (a) too high and the feedback capacitance must be increased
- (b) too low and the feedback capacitance must be decreased or
- (c) within the optimal range.

In the latter case, the binary search is terminated. In the other cases, the upper or lower remaining search interval is bisected and a new trim word is applied. Although both the target value and the trim word are initially concrete, the actual output voltage reflects the uncertainty introduced by the MEMS. Consequently, the comparison of actual and target value might reveal that multiple cases are possible depending on the precise value of the deviation symbol. The simulation is capable of identifying and executing all potential trajectories within a single run. Figure 7.9 illustrates the procedure. The upper limits of the CV voltage are indicated by the data points in pink, while the lower limits are highlighted in green. In any case, namely for all possible values of the deviation symbol, the output voltage is too high given the initial trim

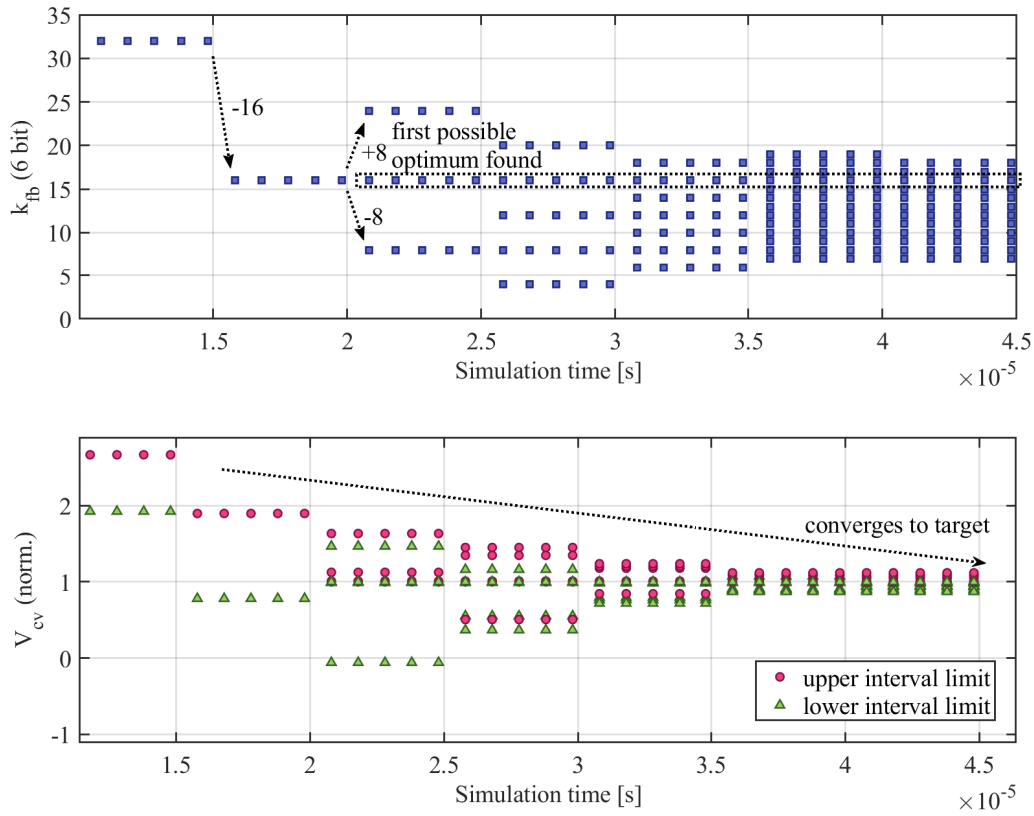


FIGURE 7.9: Simulation results for a sensitivity trim conducted as binary search. The trim word becomes an AADD throughout the simulation. As a result, all possible trajectories of the search procedure are executed within a single simulation run. The CV output voltage is normalized to its target. Circular markers represent the upper limits and triangles the lower limits. One interval per leaf of the AADD is shown. All of them converge towards the target.

word. Therefore, the lower search range of k_{fb} is bisected, resulting in a new trim word of 16, which remains concrete. The new output voltage is measured and found to contain the target value within its interval boundaries. As a consequence, it is necessary to consider all three options in the next step. The testbench model employs symbolic control structures (denoted with an additional suffix "S"), which facilitate the execution of multiple branches in this scenario. The trim word is then transformed into an AADD comprising three leaves, each containing a distinct value: 24, 16, and 8. The trim value of 16 has been identified as a potential optimum, and this branch will not be considered in subsequent simulation steps. The next measurement does not yield a single interval, but rather three distinct intervals for the CV output voltage, each corresponding to one trim word. The lower two intervals overlap. Once more, all potential outcomes of the comparison between the actual and targeted voltages are taken into account, and the possible trim words are further divided. Upon completion of the procedure, each distinct interval has converged towards the optimum. The remaining deviations from the target voltage are attributed to the coarse granularity of the trim circuit in this example. The leaves of the AADD representing the trim word have concrete values corresponding to one possible trim word each. From a semantic perspective, they can be merged into one coherent interval spanning the range [7, 18]. The simulation validates that

- (a) The complete range of expected sensitivities can be trimmed towards the target.

- (b) The trim circuitry has been designed with a sufficient safety margin to accommodate potential variations in sensitivity beyond the anticipated range.
- (c) The algorithm functions as anticipated. Potential indications of anomalies would include the omission of a specific trim word within the spanned range or the occurrence of outliers.

The entire trimming procedure is realized within a single SystemC thread. A condensed representation of the thread is provided in the code snippet in listing 7.1. The symbolic data type *doubleS* represents an AADD and *boolS* a classical binary tree. Initially, both variables have only a single leaf and no internal nodes. They grow during the simulation. The event *ev_apply* triggers a method that sets the output ports of the testbench module to apply a new stimulus or trim configuration. The *abs()* function call evaluates the absolute value of the argument. In general, it holds

$$|x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{else.} \end{cases} \quad (7.36)$$

If the function is applied to an AADD, it is iteratively applied to each leaf. If the leaf contains an affine form $\tilde{z}$, it keeps unchanged if $\min(z) \geq 0$, switches its sign if $\max(z) < 0$ and is replaced by an internal node with two leaves otherwise. All in all, the necessity for AADDs is evident as a simulation of purely coherent intervals is infeasible for this kind of application. The case study demonstrates that AADDs are well suited to evaluate the correctness of trimming procedures that do not only rely on analytical formulas but also on search algorithms with dedicated control structures.

7.7 Discussion

In the present chapter, the concept of affine arithmetic was introduced. It is a semi-symbolic arithmetic that operates on intervals. Two major contributions were presented. Firstly, the applicability and efficacy of affine arithmetic for the application of sensor trimming simulation were demonstrated. The approach benefits from the concept of deviation symbols, which enable the expression of correlations in a way that allows for the trimming of uncertainties without the need to define their actual concrete values. As a result, the trim algorithm is evaluated for the entire range of deviations encompassed by the deviation term. Therefore, a single simulation covers an area within the analog state space, rather than a single point. In the event that any deviation term is not fully compensated, the respective uncertainty symbol serves to indicate the root cause to the user.

Secondly, an extension to affine arithmetic, designated AA2, was developed that enables to identify the underlying cause of deviation terms up to second-order effects. In this context, second-order effects are defined as those resulting from the interference of two deviation symbols. In classical affine arithmetic, these terms are summed to form a single offset term that may contain contributions from all interferences. In AA2, these contributions are maintained as separate entities, thereby facilitating the tracking of root causes. Moreover, an implementation of the rounding operator that does not follow Hansen's form and a bit access operator were introduced.

LISTING 7.1: Extract from the SystemC thread that implements the binary search based trim procedure.

```

// initialize
doubleS k_scfb = 32;
boolS optimum_found = false;
int n = 4;
[...] // specify target voltage vtarget, ...

acc = 1.0; // set stimulus of 1g
ev_apply.notify(); // event that triggers a process to apply the new stimulus

// start binary search
whileS(!optimum_found && (n >= 0))
    vdiff = v_cv.read() - vtarget; // compare actual value with target
    dev_old = abs(vdiff);
    k_scfb_opt = k_scfb;

    ifS(abs(vdiff) < vlsb) // deviation smaller than granularity of trim circuit
        optimum_found = true; // --> optimum found, stop
    endS

    ifS(!optimum_found)
        ifS(vdiff > 0)
            k_scfb = k_scfb + pow(2,n); // upper half
        elseS
            k_scfb = k_scfb - pow(2,n); // lower half
        endS
    endS
    n = n - 1;

    ev_apply.notify(); // event that triggers a process to apply the new trim word
    wait(AFE_UNIT_DELAY, sc_core::SC_SEC); // wait till frontend has settled
endS

// assert that optimum has been found and apply optimum trim bit
[...]
```

The illustrative examples and experimental results have demonstrated that affine arithmetic is an effective method for simulating trim algorithms. The current limitations are scalability and integration into a complete HVP.

The scalability is constrained by two primary factors. On the one hand, the use of control structures in each transient simulation step harbors the risk that the tree sizes increase exponentially. The handling of AADDs necessitates the careful implementation of binary tree reduction rules, and may still result in huge trees that negatively impact the simulation. On the other hand, the scalability is limited by the quality of approximation schemes that are applied to non-affine (non-linear) functions. Once more, the issue scales with the number of transient steps, as the approximation errors of each transient step accumulate.

In the course of this work, an HVP was developed for an industrial inertial measurement unit to examine its applicability to real-world examples, as described in chapter 3. To ensure a consistent development flow, it would be beneficial to integrate the semi-symbolic simulation seamlessly into the HVP. The AADD library utilized for the experiments presented in this chapter provides the ability to switch between semi-symbolic and numerical simulation through the use of aliases for control structures and data types. However, the current setup of the HVP

employs a C++/Python bridge, whereby test scripts are developed within the Python environment. Accordingly, an interpreter was necessary to transform the behavior of a Python test bench into either concrete or symbolic C++ commands, contingent on the current configuration. This matter was not pursued further in the present thesis, and thus the model presented in this chapter is not connected to the numeric HVP that was discussed in the preceding chapters. The seamless integration of the two model types is thus subject to future work.

Chapter 8

Conclusion and Future Work

The thesis at hand examined the impact of heterogeneous virtual prototypes on the development of smart sensors. Thereby, the firmware development and the preparation of trim and test scripts for the laboratory were identified as use cases that benefit clearly from the heterogeneous model nature. The most appropriate level of abstraction for these applications was discussed. The model must be sufficiently abstract to perform the modeling tasks before an actual implementation is available, while still covering the necessary details to ensure qualitatively correct transients, both for normal operation and specific test modes. The overall methodology was evaluated based on a large-scale industrial case study. The development of a commercial inertial measurement unit for the consumer market was utilized for that purpose and provided with an HVP. In this context, a set of requirements were defined for the HVP that significantly impact its success. These include the HVP's early availability to enable frontloading, a high level of simulation performance to shorten development cycles, and a wide coverage of the analog state space to identify hidden corner cases.

SystemC AMS was used as the modeling language to reflect the AMS part of the system. Firstly, it is designed to seamlessly integrate with the digital components written in SystemC. Secondly, the semantics of the TDF MoC enable the generation of fast and abstract AMS models. Furthermore, it is open source and thus extensible. To illustrate its ability in terms of simulation performance, a comparison was conducted between the performance of the timed dataflow semantics of SystemC AMS and a real number modeling approach in SystemC. Subsequently, the efficiency of TDF models in presence of ordinary differential equations was enhanced through the incorporation of variable-step ODE solvers. Two distinct methodologies were presented to integrate the variable timing into the language's semantics while simultaneously preventing rollbacks of the TDF time grid. The reuse of internal solver matrices was introduced as a means of enhancing efficiency. The concept was adapted and applied to implicit variable-step solvers in the form of timestep quantization and to electrical linear networks with high switching activity in the form of switch monitoring.

The thesis at hand presented the observation that the commercial tool MATLAB/Simulink is widely used for the rapid modeling of AMS content due to an extensive library of available components and its illustrative nature. Hence, it was used as a starting point to support the early availability of AMS models for HVPs. A translation tool was presented that automates the generation of SystemC AMS TDF models from Simulink sources. Thereby, the structure of the initial Simulink model, comprising multiple layers of subsystems, was preserved. TDF modules were constructed for each block on the lowest layer of interest. The layers above were constructed in a hierarchical manner through the composition of their constituent subblocks. As a consequence, the resulting models were highly modular, allowing for the replacement of individual modules with more sophisticated versions at a later stage of the project.

The coverage of the analog state space was addressed, extending the scope from numerical transient simulations to semi-symbolic simulation using affine arithmetic. The objective was to simulate the process of sensor trimming. The uniqueness of deviation symbols within the arithmetic provides the ability to compensate for uncertain deviations through similarly uncertain trim words, thereby ensuring the reliability of the trim result across a comprehensive range of deviation values. To enhance the method's efficacy, an augmentation towards second-order root cause tracking was proposed, which further divides deviation terms into their root causes up to the second order of interference.

8.1 Discussion and Limitations

In the very beginning of this work, a set of research questions was formulated. The following section will examine how and to what extent this work has addressed them based on the contributions made. The thesis at hand in no way claims to answer these questions in a complete and exhaustive manner. Consequently, the limitations of this thesis are also highlighted in this section.

RQ1-Impact Where and to which extent can HVPs support the development of MEMS smart sensors?

The potential for HVPs in the context of MEMS smart sensors has been demonstrated through the development of an industrial high-performance inertial measurement unit as presented in chapter 3. It was shown that a comprehensive coverage of the heterogeneous system increases the expressiveness of the virtual platform and supports the development of firmware that closely interacts with the hardware in a more effective and seamless manner than purely digital VPs. Furthermore, the applicability of HVPs for the preparation of measurements and trimming in a laboratory was validated. This was accomplished by means of a simulation of the respective procedures on the HVP. These results address the given research question, clearly demonstrating the positive impact that an HVP can have on the development of smart sensors. Moreover, a targeted development flow was depicted, which illustrated the frontloading effect of the HVP. The features, comprising both firmware and trim or test scripts, were initially developed on the HVP and verified on RTL in a consecutive step. Hence, the majority of debugging cycles were conducted on the rapid white-box model (the HVP) that also permitted an in-depth examination of the system semantics. In contrast, the number of time-consuming simulations on RTL models was significantly reduced. Consequently, the overall design flow becomes more agile and enables a reduction in time-to-market.

While the HVP has proven to accelerate the development of sensors, the applicability of the approach is primarily limited to ensuring functional correctness. Therefore, it is feasible to validate the operational principle of a trimming algorithm; however, it is not possible to predict the untrimmed distribution of parameters in advance. Similarly, the modeling of noise is feasible to challenge the robustness of firmware algorithms but the prediction of the noise level must be performed separately. It is therefore necessary to obtain this information from more detailed component models. The present thesis does not address the issue of a bottom-up approach that refines the HVP based on the results of low-level simulations in detail. In general, the modularity of the HVP and the parameterization of the modules permit the model to be adapted in closer alignment with the actual behavior step-by-step.

Another area that has not been addressed is that of model verification. In particular, when the development of the HVP is initiated at an early project stage, the implementation of the ASIC is not yet available. Consequently, the model must be verified against the specification directly. The question of how to efficiently accomplish this and how to measure the verification coverage is beyond the scope of this work.

The industrial case study benefited to a limited extent from the HVP, due to the rather late availability of certain features. This happened because the project was used to explore new use cases, and the overall strategy evolved during the course of the project. As a result, there is still a lack of quantitative evidence to demonstrate the full potential of the HVP. Furthermore, corporate management-related aspects, such as resource planning, were not within the scope of this work.

RQ2-Performance How to optimize the simulation performance of AMS models for HVPs in the context of MEMS smart sensors?

To address RQ2, a performance comparison was conducted between real number modeling and the timed dataflow semantics. The results revealed the performance-related benefits of the static schedule of SystemC AMS, which allows for the merging of a complete TDF cluster into a single SystemC process. The dynamic TDF features do not negate the benefit, provided that only the timesteps are adjusted and the static schedule remains unaltered. In contrast, the introduction of rate and/or delay changes necessitates a recalculation of the schedule, which should be employed only on an occasional basis. A potential performance bottleneck is the synchronization between the discrete-event kernel and the TDF time, which is triggered whenever a converter port is accessed. Therefore, the interface should be kept as concise as possible. However, the overall impact appeared to be small when applied to a real-world example.

Nevertheless, the analysis demonstrated that SystemC AMS is a promising language to create fast AMS models for HVPs. Furthermore, it is open source, which allows the author of this work to extend it with additional features. Two synchronization schemes were presented that facilitate the integration of variable-step ODE solvers within the SystemC AMS semantics. It was observed that the resulting performance enhancement is contingent upon the specific use case, but it has the potential to reach improvement factors $>100\times$ when the proposed integration is extended by the timestep quantization approach. A related approach was presented for the enhancement of the simulation performance of ELN circuits with high switching activity, designated as switch monitoring. In this case, a factor of 45 was attained for the switched-capacitor CV converter under examination.

The variable-step solver adapts the timestep based on an error estimation. However, this approach is constrained to TDF clusters that include a differential equation, which must additionally reflect the system dynamics accurately. While this is an intrinsic limitation of the approach, it is typically satisfied in the context of MEMS sensors, as the mechanical component is predominantly modeled in this manner. In order for the timestep quantization and the switch monitoring to be applicable, it is necessary that the set of occurring state matrices during a simulation is finite. The timestep quantization approach permits alterations to the state matrix solely as a consequence of the introduction of a new timestep. The switch monitoring methodology assumes that switching actions represent the sole source of state matrix modifications. These constraints preclude the application of the methodologies presented to systems that model nonlinearities based on state parameter changes. Furthermore, variable passive elements are prohibited in ELN circuits and must be rewritten as equivalent controlled sources.

RQ3-Availability How can an early availability of the HVP be ensured?

It is evident that the early availability of a simulation model is contingent upon the early availability of the specification and the allocation of sufficient engineering resources for its creation. The present study did not address these project management-related issues. In lieu of this, a translation tool was presented that generates SystemC AMS TDF models from MATLAB/Simulink models. It was demonstrated that the tool enables the generation of highly modular AMS models suitable for HVPs, as it preserves the initial hierarchy of the Simulink

model. Consequently, individual subblocks can be effortlessly substituted with more sophisticated versions. The assertion that MATLAB/Simulink models are frequently accessible at an early stage was derived from the findings of the industrial case study and the input of numerous ASIC designers. These models were employed for preliminary exploration prior to the commencement of the actual hardware implementation.

The approach is inherently constrained by the capabilities of the Embedded Coder, which restricts the utilization of specific blocks in Simulink and supports the generation of models using fixed-step solvers exclusively. Moreover, it would be erroneous to assume that MATLAB/Simulink models are available at an early stage for all companies and areas of application. Nevertheless, the methodology aligns well with the development flow that was examined in this study. Moreover, MATLAB/Simulink is a widely adopted software in industry and academia, suggesting that its applicability extends to a range of other environments.

RQ4-Coverage To which degree can HVPs cover the analog state space with a finite set of simulations?

The final research question addressed the extent of coverage afforded by the analog state space. The continuous space of potential parameters gives rise to an infinite state space. Therefore, it is challenging to sufficiently cover this area with a limited set of simulations, even if the HVP is parameterizable. The methodology presented for addressing this issue employed semi-symbolic simulation with the use of affine arithmetic. Consequently, simulations were conducted on intervals rather than on specific values. This approach encompassed the variation of a parameter within an anticipated range through a single simulation. By defining a limited set of parameters as affine forms, it became possible to encompass finite regions of the state space rather than a single point with a single simulation. This enhanced the coverage, facilitated the identification of root causes, and thus enhanced the overall confidence in the trim algorithm under test.

It is inherent to linear operations that they are affine; however, nonlinear operations must be approximated in order to be incorporated into the arithmetic. This constrains the expressiveness of a simulation that employs numerous nonlinear functions due to overestimations, but particularly because nonlinear interference of deviation terms results in an offset term that lacks any information regarding the source of the deviation. This limitation was partially addressed in the present work through the development of AA2, an extension of affine arithmetic that allows for root cause tracking up to second-order terms. Nevertheless, repeated concatenations of deviation symbols that result in higher-order terms are not taken into account and remain part of the common offset term. Moreover, affine arithmetic decision diagrams, a form of binary trees, have the potential to attain considerable levels of growth in the absence of appropriate reduction rules.

The semi-symbolic simulation model illustrated in chapter 7 has not been integrated into the overall HVP that was presented in chapter 3. Instead, it was implemented as a separate model limiting the interoperability between the semi-symbolic approach and the other methodologies presented in preceding chapters. Besides, alternative ways to cover the analog state space with formal or multi-run methods, such as Monte Carlo simulations for instance, have not been considered.

It bears noting that the set of experiments used to validate the presented AMS modeling methodologies was drawn entirely from the application domain under test, namely MEMS smart sensor development. In particular, the solver enhancements and translation procedure are similarly applicable to other areas of application. However, this was not specifically demonstrated in the present work and is a subject to future investigation.

8.2 Future Work

The results of the thesis at hand have demonstrated the feasibility of extending the scope from virtual platforms to multi-domain simulations, and have shown that this extension has a beneficial impact on the development flow of smart sensors. These results encourage further research in this area and the investigation of additional use cases. In the context of sensor development, an interesting research question could be which new use cases could be enabled when more detailed MEMS models are used, as for example reduced order models. Does the introduction of nonlinearities and parasitic modes in the MEMS model enable to analyze their impact on system-level even with the rather abstract modeling style of the electronics? Future work may investigate benefits and limitations of this approach and explore the impact of these refined models on current use cases of HVPs. The firmware development for example sometimes requires deterministic and fully explainable models to distinguish between sensor nonidealities and bugs in the firmware. Here, at least two variants of the HVP would then be required where the ROMs might be introduced for the more detailed view and utilized for a final check of an algorithms accuracy or stability.

Another potentially interesting application could be the preparation of data evaluation scripts for qualification and failure analysis. These typically combine, visualize, and assess data from a multitude of measurements to provide insights into the distribution of specific sensor parameters, the quality of compensations, and other related matters. The development of these scripts necessitates the utilization of multi-run methods, wherein each run represents a discrete component with a unique set of parameters. It is therefore essential that the format of the output data from the simulation is aligned with that of the actual measurements. Ultimately, a virtual distribution could be plotted in preparation for the evaluation of real measurements. It is still an open research question which level of detail the HVP would require to support these activities and how to handle the constrained random parameterization. Still, it is considered as an interesting future research field as this approach could facilitate closer integration between the work of the firmware developers and the trim development engineers. An *untrimmed virtual sensor* from the set of simulations would be fully specified by its set of parameters. Trim procedures could be applied by simulation and the resulting trim words would be written to the NVM. The NVM configuration could be saved for subsequent simulations, thus enabling the specification of a *trimmed virtual sensor* as a tuple comprising the set of parameters and the NVM. These tuples could subsequently be employed by the firmware developers for their simulations on trimmed devices.

Besides additional use cases, future work may also encompass the extension of the scope of the proposed solver enhancements. As previously stated, these set clear constraints on the manner in which state matrices are permitted to change. In principle, the fundamental concept of reusing solver matrices is applicable to all scenarios with a finite set of state matrices, irrespective of the underlying cause of the change. One potential avenue for further research would be to relax these conditions. Nevertheless, if each entry of the new state matrix is to be compared with all stored versions, it is unlikely that the overall performance will be enhanced, but rather reduced. Consequently, an efficient method for identifying and matching old and new states was required. In addition, the proposed extensions could contribute to extend the SystemC AMS standard.

The most important limitation of the translation methodology outlined in chapter 6 is the exclusion of variable-step solvers. The generated code employs a step function that is called in each timestep. If the Embedded Coder could be manipulated such that all internal timings within this function were based on a timestep variable, it would become feasible to use the generated SystemC AMS modules in variable-step setups as well. Such an approach would facilitate a native connection between the presented solver enhancements and the translation flow.

Lastly, the efficiency of development and validation of trim and test features could be further enhanced if it was possible to simulate the same test script with the numeric HVP and the semi-symbolic version. As previously discussed, the current implementation of the numeric HVP harnesses Python test scripts to simulate the interaction between the host and the sensor. Creating an entire HVP that is switchable between concrete and semi-symbolic simulation would require to develop a Python/C++ bridge that is able to map the Python commands either onto concrete or symbolic commands in accordance to the current configuration of the HVP. Additionally, the new setup would require a twofold register implementation that can handle both numeric and affine data. The development of strategies to enable a seamless integration of numeric and semi-symbolic virtual prototypes is hence an open research question.

In conclusion, the thesis at hand has laid the foundation for the usage of HVPs for the sensor development and introduced measures to make the modeling of their AMS subsystems more efficient and expressive.

Bibliography

- [1] Jiri Marek. "MEMS for automotive and consumer electronics". In: *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*. 2010, pp. 9–17. DOI: [10.1109/ISSCC.2010.5434066](https://doi.org/10.1109/ISSCC.2010.5434066).
- [2] Pierre Delbos, Pierre-Marie Visse, and Khrystyna Kruk. *The MEMS Industry: Looking Back at the Last 20 Years of Innovation and Growth*. Yole Intelligence. Sept. 15, 2023. URL: <https://www.yolegroup.com/strategy-insights/mems-industry-looking-back-at-the-last-20-years-of-innovation-and-growth/>.
- [3] Mordor Intelligence. *Smart Sensors Market Size & Share Analysis - Growth Trends & Forecasts (2024 - 2029)*. Accessed: 2024-01-18. URL: <https://www.mordorintelligence.com/industry-reports/global-smart-sensors-market-industry>.
- [4] Stefan Thomke and Takahiro Fujimoto. "The Effect of "Front-Loading" Problem-Solving on Product Development Performance". In: *Journal of Product Innovation Management* 17.2 (Mar. 2000), pp. 128–142. ISSN: 1540-5885. DOI: [10.1111/1540-5885.1720128](https://doi.org/10.1111/1540-5885.1720128).
- [5] Craig S. Wright and Tanveer A. Zia. "A Quantitative Analysis into the Economics of Correcting Software Bugs". In: *Computational Intelligence in Security for Information Systems*. Springer Berlin Heidelberg, 2011, pp. 198–205. ISBN: 9783642213236. DOI: [10.1007/978-3-642-21323-6_25](https://doi.org/10.1007/978-3-642-21323-6_25).
- [6] R. Leupers, Frank Schirrmeister, Grant Martin, Tim Kogel, Roman Plyaskin, Andreas Herkersdorf, and M. Vaupel. "Virtual platforms: Breaking new grounds". In: *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Mar. 2012. DOI: [10.1109/date.2012.6176558](https://doi.org/10.1109/date.2012.6176558).
- [7] Jungyun Choi, Kyungsu Kang, Byunghoon Lee, Sangho Park, and Jaewoo Im. "Early HW/SW Co-Verification Using Virtual Platforms". In: *2021 18th International SoC Design Conference (ISOCC)*. IEEE, Oct. 2021. DOI: [10.1109/isocc53507.2021.9613958](https://doi.org/10.1109/isocc53507.2021.9613958).
- [8] Thomas Nirmaier, Manuel Harrant, Marc Huppmann, Wendy You, and Georg Pelz. "Virtual Prototyping: Closing the digital gap between product requirements and post-Si verification". In: *2022 IEEE International Test Conference (ITC)*. IEEE, Sept. 2022. DOI: [10.1109/itc50671.2022.00072](https://doi.org/10.1109/itc50671.2022.00072).
- [9] Thomas Eckart and Martin Schnieringer. "Development and Verification of Embedded Firmware using Virtual System Prototypes". In: *2006 International Symposium on System-on-Chip*. IEEE, Nov. 2006. DOI: [10.1109/issoc.2006.322007](https://doi.org/10.1109/issoc.2006.322007).
- [10] J. Eker, J.W. Janneck, E.A. Lee, Jie Liu, Xiaojun Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Yuhong Xiong. "Taming heterogeneity - the Ptolemy approach". In: *Proceedings of the IEEE* 91.1 (Jan. 2003), pp. 127–144. ISSN: 0018-9219. DOI: [10.1109/jproc.2002.805829](https://doi.org/10.1109/jproc.2002.805829).
- [11] F. Herrera and E. Villar. "A framework for embedded system specification under different models of computation in SystemC". In: *Proceedings of the 43rd annual conference on Design automation - DAC '06*. DAC '06. ACM Press, 2006. DOI: [10.1145/1146909.1147140](https://doi.org/10.1145/1146909.1147140).

- [12] Jun Zhu, I. Sander, and A. Jantsch. "HetMoC: heterogeneous modelling in SystemC". In: *2010 Forum on Specification and Design Languages (FDL 2010)*. IET, 2010. DOI: [10.1049/ic.2010.0139](https://doi.org/10.1049/ic.2010.0139).
- [13] Cedric Ben Aoun, Liliana Andrade, Torsten Maehne, Francois Pecheux, Marie-Minerve Louerat, and Alain Vachoux. "Pre-simulation elaboration of heterogeneous systems: The SystemC multi-disciplinary virtual prototyping approach". In: *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE, 2015. DOI: [10.1109/samos.2015.7363686](https://doi.org/10.1109/samos.2015.7363686).
- [14] Accellera. *SystemC*. Accessed: 2024-08-18. URL: <https://www.accellera.org/downloads/standards/systemc>.
- [15] Xing Pang, Qi Zhang, Xiaoya Liang, and Yulong Zhao. "MEMS Pressure Sensor Based on Piezoresistive Effect of MoS₂ Film". In: *2023 IEEE SENSORS*. IEEE, Oct. 2023. DOI: [10.1109/sensors56945.2023.10324905](https://doi.org/10.1109/sensors56945.2023.10324905).
- [16] C.S. Silva, J. Pontes, J.C. Viana, L.A. Rocha, and J. Gaspar. "A fully integrated three-axis thermal accelerometer". In: *2013 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. IEEE, May 2013. DOI: [10.1109/i2mtc.2013.6555558](https://doi.org/10.1109/i2mtc.2013.6555558).
- [17] Wei Gao, Binghe Ma, Jian Luo, and Jinjun Deng. "High Sensitive Flexible-Based Single-Wall Carbon Nanotubes Thermal Shear Stress Sensor for Underwater Applications". In: *2019 20th International Conference on Solid-State Sensors, Actuators and Microsystems & Eurosensors XXXIII (TRANSDUCERS & EUROSENSORS XXXIII)*. IEEE, June 2019. DOI: [10.1109/transducers.2019.8808702](https://doi.org/10.1109/transducers.2019.8808702).
- [18] Zhuo Wang, Guochen Wang, Santosh Kumar, Carlos Marques, Rui Min, and Xiaoli Li. "Recent Advancements in Resonant Fiber Optic Gyro—A Review". In: *IEEE Sensors Journal* 22.19 (Oct. 2022), pp. 18240–18252. ISSN: 2379-9153. DOI: [10.1109/jsen.2022.3195502](https://doi.org/10.1109/jsen.2022.3195502).
- [19] Zi Jian Liu, Yong Sen Yu, Xuan Yu Zhang, Chao Chen, Cong Cong Zhu, Ai Hua Meng, Shi Mei Jing, and Hong Bo Sun. "An Optical Microfiber Taper Magnetic Field Sensor With Temperature Compensation". In: *IEEE Sensors Journal* 15.9 (Sept. 2015), pp. 4853–4856. ISSN: 1558-1748. DOI: [10.1109/jsen.2015.2429911](https://doi.org/10.1109/jsen.2015.2429911).
- [20] Yaqi Tang, Chao Wang, Shuangchen Ruan, Chi Chiu Chan, and Xinyong Dong. "Virtual Vernier effect-based high sensitivity optical fiber humidity sensor". In: *2021 19th International Conference on Optical Communications and Networks (ICOON)*. IEEE, Aug. 2021. DOI: [10.1109/icocn53177.2021.9563721](https://doi.org/10.1109/icocn53177.2021.9563721).
- [21] Ekbert Hering, Karl-Ernst Biel, Ulrich Guth, Martin Liess, and Winfried Vonau. "Physical Effects of Sensor Use". In: *Sensors in Science and Technology: Functionality and Application Areas*. Ed. by Ekbert Hering and Gert Schönfelder. Wiesbaden: Springer Fachmedien Wiesbaden, 2022, pp. 5–145. ISBN: 978-3-658-34920-2. DOI: [10.1007/978-3-658-34920-2_2](https://doi.org/10.1007/978-3-658-34920-2_2). URL: https://doi.org/10.1007/978-3-658-34920-2_2.
- [22] Alessandro Morcelli, Simone Ferri, and Anton Hofmeister. "MEMS: From a Bright Past Towards a Shining Future". In: *Silicon Sensors and Actuators*. Springer International Publishing, 2022, pp. 945–978. ISBN: 9783030801359. DOI: [10.1007/978-3-030-80135-9_27](https://doi.org/10.1007/978-3-030-80135-9_27).
- [23] Volker Kempe. "MEMS technologies". In: *Inertial MEMS: Principles and Practice*. Cambridge University Press, 2011, 152–204.
- [24] Artyom T. Tulaev, Aleksei S. Kozlov, Jacob V. Belyaev, Vera V. Loboda, Mikhail A. Bellavin, and Alexander S. Korotkov. "MEMS Pressure Sensors Design, Simulation, Manufacturing, Interface Circuits: A Review". In: *IEEE Sensors Journal* 24.6 (2024), pp. 7395–7405. DOI: [10.1109/JSEN.2024.3358951](https://doi.org/10.1109/JSEN.2024.3358951).

- [25] Muhammad Izzudin Ahmad Asri, Md. Nazibul Hasan, Mariatul Rawdhah Ahmad Fuaad, Yusri Md. Yunos, and Mohamed Sultan Mohamed Ali. "MEMS Gas Sensors: A Review". In: *IEEE Sensors Journal* 21.17 (2021), pp. 18381–18397. DOI: [10.1109/JSEN.2021.3091854](https://doi.org/10.1109/JSEN.2021.3091854).
- [26] G. Langfelder, M. Bestetti, and M. Gadola. "Silicon MEMS inertial sensors evolution over a quarter century". In: *Journal of Micromechanics and Microengineering* 31.8 (July 2021), p. 084002. ISSN: 1361-6439. DOI: [10.1088/1361-6439/ac0fbf](https://doi.org/10.1088/1361-6439/ac0fbf).
- [27] Feichi Zhou and Yang Chai. "Near-sensor and in-sensor computing". In: *Nature Electronics* 3.11 (Nov. 2020), pp. 664–671. ISSN: 2520-1131. DOI: [10.1038/s41928-020-00501-9](https://doi.org/10.1038/s41928-020-00501-9).
- [28] BHI360. *Programmable, low power, Inertial Measurement Unit with integrated sensor fusion library*. Rev. 1.0. Bosch Sensortec. Apr. 2023. URL: <https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bhi360-ds000.pdf>.
- [29] Volker Kempe. "Accelerometers". In: *Inertial MEMS: Principles and Practice*. Cambridge University Press, 2011, 283–363.
- [30] Derek K. Shaeffer. "MEMS inertial sensors: A tutorial overview". In: *IEEE Communications Magazine* 51.4 (Apr. 2013), pp. 100–109. ISSN: 0163-6804. DOI: [10.1109/mcom.2013.6495768](https://doi.org/10.1109/mcom.2013.6495768).
- [31] Giorgio Allegato, Lorenzo Corso, and Carlo Valzasina. "Inertial Sensors". In: *Silicon Sensors and Actuators*. Springer International Publishing, 2022, pp. 439–475. ISBN: 9783030801359. DOI: [10.1007/978-3-030-80135-9_13](https://doi.org/10.1007/978-3-030-80135-9_13).
- [32] Tommaso Ungaretti, Sergio Pernici, Daniele De Pascalis, Deyou Fang, Mario Maiore, and Giovanni Pelligra. "Electronic Sensors Front-End". In: *Silicon Sensors and Actuators*. Springer International Publishing, 2022, pp. 734–735. ISBN: 9783030801359. DOI: [10.1007/978-3-030-80135-9_22](https://doi.org/10.1007/978-3-030-80135-9_22).
- [33] Volker Kempe. "Gyroscopes". In: *Inertial MEMS: Principles and Practice*. Cambridge University Press, 2011, 364–459.
- [34] Mikko Saukoski. "System and circuit design for a capacitive MEMS gyroscope". PhD thesis. 2008. URL: <https://urn.fi/URN:ISBN:978-951-22-9297-4>.
- [35] Wilhelmus H. A. Schilders, Henk A. Vorst, and Joost Rommes, eds. *Model Order Reduction: Theory, Research Aspects and Applications*. Springer Berlin Heidelberg, 2008. ISBN: 9783540788416. DOI: [10.1007/978-3-540-78841-6](https://doi.org/10.1007/978-3-540-78841-6).
- [36] Tino Blochmann, Stephan Gerth, and Peter Schneider. "Integration of MEMS models in simulations on electronic system level". In: *MikroSystemTechnik 2017; Congress*. 2017.
- [37] V.K. Madiseti and T.W. Ego. "Virtual prototyping of embedded microcontroller-based DSP systems". In: *IEEE Micro* 15.5 (1995), pp. 9–21. ISSN: 0272-1732. DOI: [10.1109/40.464577](https://doi.org/10.1109/40.464577).
- [38] C. Passerone, L. Lavagno, M. Chiodo, and A. Sangiovanni-Vincentelli. "Fast Hardware/-software Co-simulation For Virtual Prototyping And Trade-off Analysis". In: *Proceedings of the 34th Design Automation Conference*. IEEE, 1997. DOI: [10.1109/dac.1997.597178](https://doi.org/10.1109/dac.1997.597178).
- [39] D. Becker, R.K. Singh, and S.G. Tell. "An engineering environment for hardware/software co-simulation". In: *[1992] Proceedings 29th ACM/IEEE Design Automation Conference*. IEEE Comput. Soc. Press. DOI: [10.1109/dac.1992.227848](https://doi.org/10.1109/dac.1992.227848).
- [40] "IEEE Standard for Standard SystemC® Language Reference Manual". In: *IEEE Std 1666-2023 (Revision of IEEE Std 1666-2011)* (2023), pp. 1–618. DOI: [10.1109/IEEESTD.2023.10246125](https://doi.org/10.1109/IEEESTD.2023.10246125).

- [41] OSCI TLM-2.0 Language Reference Manual. Open SystemC Initiative (OSCI). July 2009. URL: https://www.accellera.org/images/downloads/standards/systemc/TLM_2_0_LRM.pdf.
- [42] Tim Kogel. "Virtual Prototyping of Processor-Based Platforms". In: *Handbook of Computer Architecture*. Springer Nature Singapore, 2022, pp. 1–41. ISBN: 9789811564017. DOI: [10.1007/978-981-15-6401-7_44-1](https://doi.org/10.1007/978-981-15-6401-7_44-1).
- [43] J.A. Rowson. "Virtual prototyping". In: *Proceedings of CICC 97 - Custom Integrated Circuits Conference*. CICC-97. IEEE. DOI: [10.1109/cicc.1997.606591](https://doi.org/10.1109/cicc.1997.606591).
- [44] G. Gary Wang. "Definition and Review of Virtual Prototyping". In: *Journal of Computing and Information Science in Engineering* 2.3 (Sept. 2002), pp. 232–236. ISSN: 1944-7078. DOI: [10.1115/1.1526508](https://doi.org/10.1115/1.1526508).
- [45] Gereon Onnebrink, Rainer Leupers, Gerd Ascheid, and Stefan Schurmans. "Black box ESL power estimation for loosely-timed TLM models". In: *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. IEEE, July 2016. DOI: [10.1109/samos.2016.7818374](https://doi.org/10.1109/samos.2016.7818374).
- [46] Jens Rudolf, Daniel Gis, Sebastian Stieber, Christian Haubelt, and Rainer Dorsch. "SystemC Power Profiling for IoT Device Firmware using Runtime Configurable Models". In: *2019 8th Mediterranean Conference on Embedded Computing (MECO)*. IEEE, June 2019. DOI: [10.1109/meco.2019.8759994](https://doi.org/10.1109/meco.2019.8759994).
- [47] Md Rafiul Kabir and Sandip Ray. "Virtual Prototyping for Modern Internet-of-Things Applications: A Survey". In: *IEEE Access* 11 (2023), pp. 31384–31398. ISSN: 2169-3536. DOI: [10.1109/access.2023.3262499](https://doi.org/10.1109/access.2023.3262499).
- [48] Mohsin Raza, Priyan Malarvizhi Kumar, Dang Viet Hung, William Davis, Huan Nguyen, and Ramona Trestian. "A Digital Twin Framework for Industry 4.0 Enabling Next-Gen Manufacturing". In: *2020 9th International Conference on Industrial Technology and Management (ICITM)*. IEEE, Feb. 2020. DOI: [10.1109/icitm48982.2020.9080395](https://doi.org/10.1109/icitm48982.2020.9080395).
- [49] Hao Qin, Hongwei Wang, Yu Zhang, and Libin Lin. "Constructing Digital Twin for Smart Manufacturing". In: *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, May 2021. DOI: [10.1109/cscwd49262.2021.9437791](https://doi.org/10.1109/cscwd49262.2021.9437791).
- [50] Barbara Rita Barricelli, Elena Casiraghi, and Daniela Fogli. "A Survey on Digital Twin: Definitions, Characteristics, Applications, and Design Implications". In: *IEEE Access* 7 (2019), pp. 167653–167671. ISSN: 2169-3536. DOI: [10.1109/access.2019.2953499](https://doi.org/10.1109/access.2019.2953499).
- [51] Sunha Ahn and Sharad Malik. "Automated firmware testing using firmware-hardware interaction patterns". In: *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*. ESWEEK'14. ACM, Oct. 2014. DOI: [10.1145/2656075.2656080](https://doi.org/10.1145/2656075.2656080).
- [52] Baekgyu Kim, Yusuke Kashiba, Siyuan Dai, and Shinichi Shiraishi. "Testing Autonomous Vehicle Software in the Virtual Prototyping Environment". In: *IEEE Embedded Systems Letters* 9.1 (Mar. 2017), pp. 5–8. ISSN: 1943-0671. DOI: [10.1109/les.2016.2644619](https://doi.org/10.1109/les.2016.2644619).
- [53] Vladimir Herdt, Daniel Große, and Rolf Drechsler. "Towards Specification and Testing of RISC-V ISA Compliance". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2020, pp. 995–998. DOI: [10.23919/DATE48585.2020.9116193](https://doi.org/10.23919/DATE48585.2020.9116193).
- [54] Milan Funck, Vladimir Herdt, and Rolf Drechsler. "Virtual Prototype driven Design, Implementation and Evaluation of RISC-V Instruction Set Extensions". In: *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. 2022, pp. 14–19. DOI: [10.1109/DDECS54261.2022.9770108](https://doi.org/10.1109/DDECS54261.2022.9770108).

- [55] Hoang M. Le, Daniel Große, and Rolf Drechsler. "Scalable fault localization for SystemC TLM designs". In: *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2013, pp. 35–38. DOI: [10.7873/DATE.2013.022](https://doi.org/10.7873/DATE.2013.022).
- [56] G. Economakos, P. Oikonomakos, I. Panagopoulos, I. Poulakis, and G. Papakonstantinou. "Behavioral synthesis with SystemC". In: *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*. 2001, pp. 21–25. DOI: [10.1109/DATE.2001.914995](https://doi.org/10.1109/DATE.2001.914995).
- [57] Vladimir Herdt, Hoang M. Le, Daniel Große, and Rolf Drechsler. "Towards fully automated TLM-to-RTL property refinement". In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2018, pp. 1508–1511. DOI: [10.23919/DATE.2018.8342253](https://doi.org/10.23919/DATE.2018.8342253).
- [58] Kyungsu Kang, Sangho Park, Byeongwook Bae, Jungyun Choi, SungGil Lee, Byunghoon Lee, and Jong-Bae Lee. "Seamless SoC Verification Using Virtual Platforms: An Industrial Case Study". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Mar. 2019. DOI: [10.23919/date.2019.8715128](https://doi.org/10.23919/date.2019.8715128).
- [59] Bogdan-Andrei Tabacaru, Moomen Chaari, Wolfgang Ecker, Thomas Kruse, and Cristiano Novello. "Fault-effect analysis on system-level hardware modeling using virtual prototypes". In: *2016 Forum on Specification and Design Languages (FDL)*. IEEE, Sept. 2016. DOI: [10.1109/fdl.2016.7880368](https://doi.org/10.1109/fdl.2016.7880368).
- [60] Douglas Lohmann, Alexis Huf, Djones Lettnin, Frank Siqueira, and Jose Luis Guntzel. "A Domain-specific Language for Automated Fault Injection in SystemC Models". In: *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, Dec. 2018. DOI: [10.1109/icecs.2018.8617838](https://doi.org/10.1109/icecs.2018.8617838).
- [61] Ebrahim Nouri, Nooshin Nosrati, Hanieh Totonchi Asl, Mozhgan Rezaie Manavand, and Zainalabedin Navabi. "Multi-Level Fault Injection Methodology Using UVM-SystemC". In: *2023 IEEE East-West Design & Test Symposium (EWDTS)*. IEEE, Sept. 2023. DOI: [10.1109/ewdts59469.2023.10297034](https://doi.org/10.1109/ewdts59469.2023.10297034).
- [62] Nidhi Chandoke and Ashish Kumar Sharma. "A novel approach to estimate power consumption using SystemC transaction level modelling". In: *2015 Annual IEEE India Conference (INDICON)*. IEEE, Dec. 2015. DOI: [10.1109/indicon.2015.7443519](https://doi.org/10.1109/indicon.2015.7443519).
- [63] Zineb El Hariti, Abdelhakim Alali, and Mohamed Sadik. "Virtual platform for modeling the power consumption of a soft-core processor by the SystemC/TLM". In: *2018 4th International Conference on Optimization and Applications (ICOA)*. IEEE, Apr. 2018. DOI: [10.1109/icoa.2018.8370590](https://doi.org/10.1109/icoa.2018.8370590).
- [64] Y. W. Hau and Mohamed Khalil-Hani. "SystemC-based HW/SW co-simulation platform for system-on-chip (SoC) design space exploration". In: *2008 International Conference on Electronic Design*. IEEE, Dec. 2008. DOI: [10.1109/iced.2008.4786734](https://doi.org/10.1109/iced.2008.4786734).
- [65] Christian Haubelt, Thomas Schlichter, Joachim Keinert, and Mike Meredith. "System-CoDesigner: automatic design space exploration and rapid prototyping from behavioral models". In: *Proceedings of the 45th annual Design Automation Conference*. DAC '08. ACM, June 2008. DOI: [10.1145/1391469.1391616](https://doi.org/10.1145/1391469.1391616).
- [66] Ting-Shuo Hsu, Chao-Chih Wu, Che-Wei Hsu, Chih-Tsun Huang, Jing-Jia Liou, Yao-Hua Chen, and Juin-Ming Lu. "Design space exploration with a cycle-accurate system-C/TLM DRAM controller model". In: *2017 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. IEEE, Apr. 2017. DOI: [10.1109/vlsi-dat.2017.7939674](https://doi.org/10.1109/vlsi-dat.2017.7939674).
- [67] Jintaek Kang, Changjae Yi, Keonjoo Lee, Seungwook Lee, Soojung Ryu, and Soonhoi Ha. "Fast Simulation of a Many-NPU Network-on-Chip for Microarchitectural Design Space Exploration". In: *2021 24th Euromicro Conference on Digital System Design (DSD)*. IEEE, Sept. 2021. DOI: [10.1109/dsd53832.2021.00029](https://doi.org/10.1109/dsd53832.2021.00029).

- [68] Philippe Coussy, Dominique Heller, and Cyrille Chavet. "High-Level Synthesis: On the path to ESL design". In: *2011 9th IEEE International Conference on ASIC*. IEEE, Oct. 2011. DOI: [10.1109/asicon.2011.6157400](https://doi.org/10.1109/asicon.2011.6157400).
- [69] Hung-Yi Liu, M. Petracca, and L. P. Carloni. "Compositional system-level design exploration with planning of high-level synthesis". In: *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Mar. 2012. DOI: [10.1109/date.2012.6176550](https://doi.org/10.1109/date.2012.6176550).
- [70] Santosh Shetty and Benjamin Camon Schafer. "Enabling the Design of Behavioral Systems-on-Chip". In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, Dec. 2021. DOI: [10.1109/dac18074.2021.9586263](https://doi.org/10.1109/dac18074.2021.9586263).
- [71] Open Virtual Platforms. *Technology*. Accessed: 2024-09-17. 2024. URL: <https://www.ovpworld.org/technology>.
- [72] Arya Wicaksana and Chong Ming Tang. "Virtual Prototyping Platform for Multiprocessor System-on-Chip Hardware/Software Co-design and Co-verification". In: *Computer and Information Science*. Springer International Publishing, May 2017, pp. 93–108. ISBN: 9783319601700. DOI: [10.1007/978-3-319-60170-0_7](https://doi.org/10.1007/978-3-319-60170-0_7).
- [73] Synopsys. *Virtualizer*. Accessed: 2024-09-17. 2024. URL: <https://www.synopsys.com/verification/virtual-prototyping/virtualizer.html>.
- [74] Md Rafiul Kabir, Neha Mishra, and Sandip Ray. "VIVE: Virtualization of Vehicular Electronics for System-level Exploration". In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. Vol. 14. IEEE, Sept. 2021, pp. 3307–3312. DOI: [10.1109/itsc48978.2021.9564397](https://doi.org/10.1109/itsc48978.2021.9564397).
- [75] Franck Pagnat, Katell Morin-Allory, and Laurent Fesquet. "A refinement process for top-down mixed-signal designs thanks to SystemC-AMS". In: *2011 IEEE 9th International New Circuits and systems conference*. IEEE, 2011. DOI: [10.1109/newcas.2011.5981249](https://doi.org/10.1109/newcas.2011.5981249).
- [76] MathWorks. *MATLAB*. Accessed: 2024-10-11. 2024. URL: <https://www.mathworks.com/products/matlab.html>.
- [77] MathWorks. *Simulink*. Accessed: 2024-10-11. 2024. URL: <https://www.mathworks.com/products/simulink.html>.
- [78] *SystemC Analog/Mixed-Signal User's Guide*. Accellera Systems Initiative. Jan. 2020.
- [79] "IEEE Standard for Standard SystemC(R) Analog/Mixed-Signal Extensions Language Reference Manual". In: *IEEE Std 1666.1-2016* (2016), pp. 1–236. DOI: [10.1109/IEEESTD.2016.7448795](https://doi.org/10.1109/IEEESTD.2016.7448795).
- [80] Edward A. Lee and David G. Messerschmitt. "Synchronous data flow". In: *Proceedings of the IEEE* 75.9 (1987), pp. 1235–1245. DOI: [10.1109/PROC.1987.13876](https://doi.org/10.1109/PROC.1987.13876).
- [81] Samuel Mason. "Feedback Theory-Some Properties of Signal Flow Graphs". In: *Proceedings of the IRE* 41.9 (Sept. 1953), pp. 1144–1156. ISSN: 0096-8390. DOI: [10.1109/jrproc.1953.274449](https://doi.org/10.1109/jrproc.1953.274449).
- [82] Torsten Maehne, Alain Vachoux, and Yusuf Leblebici. "Development of a bond graph based model of computation for SystemC-AMS". In: *2008 Ph.D. Research in Microelectronics and Electronics*. IEEE, 2008. DOI: [10.1109/rme.2008.4595729](https://doi.org/10.1109/rme.2008.4595729).
- [83] Thomas Uhle and Karsten Einwich. "A SystemC AMS extension for the simulation of non-linear circuits". In: *23rd IEEE International SOC Conference*. IEEE, 2010. DOI: [10.1109/socc.2010.5784751](https://doi.org/10.1109/socc.2010.5784751).

- [84] Thomas Uhle and Günter Elst. "Efficient Transient Simulation of Non-linear Dynamic Networks with Discontinuous Forcing". In: *IFAC Proceedings Volumes* 45.2 (2012). 7th Vienna International Conference on Mathematical Modelling, pp. 62–66. ISSN: 1474-6670. DOI: <https://doi.org/10.3182/20120215-3-AT-3016.00010>. URL: <https://www.sciencedirect.com/science/article/pii/S1474667016306437>.
- [85] Sara Vinco, Michele Lora, and Mark Zwolinski. "SystemC-AMS Simulation of Conservative Behavioral Descriptions". In: *Languages, Design Methods, and Tools for Electronic System Design*. Springer International Publishing, 2016, pp. 151–173. ISBN: 9783319317236. DOI: [10.1007/978-3-319-31723-6_7](https://doi.org/10.1007/978-3-319-31723-6_7).
- [86] Sara Vinco, Michele Lora, and Mark Zwolinski. "Conservative behavioural modelling in SystemC-AMS". In: *2015 Forum on Specification and Design Languages (FDL)*. 2015.
- [87] Torsten Maehne and Alain Vachoux. "Supporting dimensional analysis in SystemC-AMS". In: *2009 IEEE Behavioral Modeling and Simulation Workshop*. IEEE, 2009. DOI: [10.1109/bmas.2009.5338878](https://doi.org/10.1109/bmas.2009.5338878).
- [88] *Boost.Units 1.1.0*. Matthias Schabel and Steven Watanabe. 2017. URL: https://www.boost.org/doc/libs/1_65_0/doc/html/boost_units.html.
- [89] Claudius Ptolemaeus, ed. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014. URL: <http://ptolemy.org/books/Systems>.
- [90] Karsten Einwich. "Introduction to the SystemC AMS extension standard". In: *14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*. IEEE, Apr. 2011. DOI: [10.1109/ddecs.2011.5783036](https://doi.org/10.1109/ddecs.2011.5783036).
- [91] François Pêcheux, Liliana Andrade, Marie-Minerve Louerat, Ilias Bournias, Roselyne Chotin, and Daniela Genius. "Virtual Prototyping of Open Source Heterogeneous Systems with an Open Source Framework Featuring SystemC MDVP Extensions". In: *2020 Forum for Specification and Design Languages (FDL)*. IEEE, 2020. DOI: [10.1109/fdl50818.2020.9232947](https://doi.org/10.1109/fdl50818.2020.9232947).
- [92] Junqing Guan and Renato Negra. "Modelling and simulation of an RF-DAC based transmitter at architectural level in SystemC/AMS". In: *6th Conference on Ph.D. Research in Microelectronics & Electronics*. 2010.
- [93] Wei Li, Dian Zhou, Minghua Li, Binh P. Nguyen, and Xuan Zeng. "Near-Field Communication Transceiver System Modeling and Analysis Using SystemC/SystemC-AMS With the Consideration of Noise Issues". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 21.12 (Dec. 2013), pp. 2250–2261. ISSN: 1557-9999. DOI: [10.1109/tvlsi.2012.2231443](https://doi.org/10.1109/tvlsi.2012.2231443).
- [94] Yifan Wang, Zhimiao Chen, Christoph van Meersbergen, and Stefan Heinen. "System simulation of adaptive I/Q mismatch compensation method using SystemC-AMS". In: *6th Conference on Ph.D. Research in Microelectronics & Electronics*. 2010.
- [95] Torsten Maehne, Zhi Wang, Benoît Vernay, Liliana Andrade, Cédric Ben Aoun, Jean-Paul Chaput, Marie-Minerve Louërat, François Pêcheux, Arnaud Krust, Gerold Schröpfer, Martin Barnasconi, Karsten Einwich, Fabio Cenni, and Olivier Guillaume. "UVM-SystemC-AMS based framework for the correct by construction design of MEMS in their real heterogeneous application context". In: *2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, Dec. 2014. DOI: [10.1109/icecs.2014.7050122](https://doi.org/10.1109/icecs.2014.7050122).
- [96] Benoît Vernay, Arnaud Krust, Gerold Schropfer, Francois Pecheux, and Marie-Minerve Louerat. "SystemC-AMS simulation of a biaxial accelerometer based on MEMS model order reduction". In: *2015 Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS (DTIP)*. IEEE, Apr. 2015. DOI: [10.1109/dtip.2015.7160987](https://doi.org/10.1109/dtip.2015.7160987).

- [97] Fabio Cenni, Serge Scotti, and Emmanuel Simeu. "A SystemC AMS/TLM platform for CMOS video sensors". In: *Proceedings of the 2011 Conference on Design & Architectures for Signal & Image Processing (DASIP)*. IEEE, Nov. 2011. DOI: [10.1109/dasip.2011.6136873](https://doi.org/10.1109/dasip.2011.6136873).
- [98] Fabio Cenni, Emmanuel Simeu, and Salvador Mir. "Macro-modeling of analog blocks for SystemC-AMS simulation: A chemical sensor case-study". In: *2009 17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, Oct. 2009. DOI: [10.1109/vlsisoc.2009.6041358](https://doi.org/10.1109/vlsisoc.2009.6041358).
- [99] Michel Vasilevski, Nicolas Beilleau, Hassan Aboushady, and Francois Pecheux. "Efficient and refined modeling of wireless sensor network nodes using SystemC-AMS". In: *2008 Ph.D. Research in Microelectronics and Electronics*. IEEE, June 2008. DOI: [10.1109/rme.2008.4595730](https://doi.org/10.1109/rme.2008.4595730).
- [100] Amr Habib, François Pêcheux, and Marie-Minerve Louërat. "SystemC-AMS modeling of a PCR-CE lab-on-chip for multithreaded DNA analysis". In: *2010 International Conference on Microelectronics*. IEEE, Dec. 2010. DOI: [10.1109/icm.2010.5696194](https://doi.org/10.1109/icm.2010.5696194).
- [101] Morgan Madec, Francois Pecheux, Yves Gendrault, Loic Bauer, Jacques Haiech, and Christophe Lallement. "EDA inspired open-source framework for synthetic biology". In: *2013 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, Oct. 2013. DOI: [10.1109/biocas.2013.6679717](https://doi.org/10.1109/biocas.2013.6679717).
- [102] Víctor Fernández, Elier Wilpert, Herique Isidoro, Cédric Ben Aoun, and François Pêcheux. "SystemC-MDVP modelling of pressure driven microfluidic systems". In: *2014 3rd Mediterranean Conference on Embedded Computing (MECO)*. IEEE, June 2014. DOI: [10.1109/meco.2014.6862665](https://doi.org/10.1109/meco.2014.6862665).
- [103] Fabien Teyseyre, David Navarro, Ian O'Connor, Francesco Cascio, Fabio Cenni, and Olivier Guillaume. "Fast Optical Simulation from a Reduced Set of Impulse Responses using SystemC-AMS". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*. DATE 2015. IEEE Conference Publications, 2015. DOI: [10.7873/date.2015.0284](https://doi.org/10.7873/date.2015.0284).
- [104] Donkyu Baek, Yukai Chen, Jaemin Kim, Naehyuck Chang, Enrico Macii, Sara Vinco, and M. Poncino. "SystemC-AMS Simulation of Energy Management of Electric Vehicles". In: *2018 IEEE International Telecommunications Energy Conference (INTELEC)*. IEEE, Oct. 2018. DOI: [10.1109/intlec.2018.8612387](https://doi.org/10.1109/intlec.2018.8612387).
- [105] Jian FU, Jean-Charles MARE, Liming YU, and Yongling FU. "Multi-level virtual prototyping of electromechanical actuation system for more electric aircraft". In: *Chinese Journal of Aeronautics* 31.5 (May 2018), pp. 892–913. ISSN: 1000-9361. DOI: [10.1016/j.cja.2017.12.009](https://doi.org/10.1016/j.cja.2017.12.009).
- [106] Yukai Chen, Donkyu Baek, Jaemin Kim, Santa Di Cataldo, Naehyuck Chang, Enrico Macii, Sara Vinco, and Massimo Poncino. "A SystemC-AMS Framework for the Design and Simulation of Energy Management in Electric Vehicles". In: *IEEE Access* 7 (2019), pp. 25779–25791. DOI: [10.1109/access.2019.2900505](https://doi.org/10.1109/access.2019.2900505).
- [107] Enrico Fraccaroli and Sara Vinco. "Modeling Cyber-Physical Production Systems with SystemC-AMS". In: *IEEE Transactions on Computers* (2022), pp. 1–13. ISSN: 2326-3814. DOI: [10.1109/tc.2022.3226567](https://doi.org/10.1109/tc.2022.3226567).
- [108] Seyed Hossein Hashemi Shadmehri, Ali BanaGozar, Mehdi Kamal, Sander Stuijk, Ali Afzali-Kusha, Massoud Pedram, and Henk Corporaal. "SySCIM: SystemC-AMS Simulation of Memristive Computation In-Memory". In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Mar. 2022. DOI: [10.23919/date54114.2022.9774749](https://doi.org/10.23919/date54114.2022.9774749).

- [109] Hanwen Gong, Hu He, Bin Gao, Jianshi Tang, Qingtian Zhang, He Qian, and Huaqiang Wu. "ACCLAIM: An End-to-End SystemC-AMS Simulation Framework for Analog In-Memory-Computing". In: *2023 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA)*. IEEE, Oct. 2023. DOI: [10.1109/icta60488.2023.10364310](https://doi.org/10.1109/icta60488.2023.10364310).
- [110] Sallar Ahmadi-Pour, Vladimir Herdt, and Rolf Drechsler. "RISC-V AMS VP: An Open Source Evaluation Platform for Cyber-Physical Systems". In: *2021 Forum on specification and Design Languages (FDL)*. IEEE, 2021. DOI: [10.1109/fdl153530.2021.9568387](https://doi.org/10.1109/fdl153530.2021.9568387).
- [111] Oliver Bringmann, Wolfgang Ecker, Andreas Gerstlauer, Ajay Goyal, Daniel Mueller-Gritschneider, Prasanth Sasidharan, and Simranjit Singh. "The Next Generation of Virtual Prototyping: Ultra-Fast Yet Accurate Simulation of HW/SW Systems". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*. DATE 2015. IEEE Conference Publications, 2015. DOI: [10.7873/date.2015.1105](https://doi.org/10.7873/date.2015.1105).
- [112] Yao Li, Ramy Iskander, Farakh Javid, and Marie-Minerve Louërat. "A unified platform for design and verification of mixed-signal systems based on SystemC AMS". In: *Proceeding of the 2012 Forum on Specification and Design Languages*. 2012. URL: <https://ieeexplore.ieee.org/document/6336988>.
- [113] Ramy Iskander, Marie-Minerve Louërat, and Andreas Kaiser. "Hierarchical sizing and biasing of analog firm intellectual properties". In: *Integration* 46.2 (Mar. 2013), pp. 172–188. ISSN: 0167-9260. DOI: [10.1016/j.vlsi.2012.01.001](https://doi.org/10.1016/j.vlsi.2012.01.001).
- [114] Martin Barnasconi and Sumit Adhikari. "ESL Design in SystemC AMS". In: *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017. DOI: [10.1145/3061639.3072951](https://doi.org/10.1145/3061639.3072951).
- [115] Daniela Genius and Ludovic Apvrille. "Hierarchical Design of Cyber-Physical Systems". In: *Proceedings of the 11th International Conference on Model-Based Software and Systems Engineering*. SCITEPRESS - Science and Technology Publications, 2023. DOI: [10.5220/0011654400003402](https://doi.org/10.5220/0011654400003402).
- [116] Mohammed Sulaiman BenSaleh, Raoudha Saida, Yessine Hadj Kacem, and Mohamed Abid. "Wireless Sensor Network Design Methodologies: A Survey". In: *Journal of Sensors* 2020 (Jan. 2020), pp. 1–13. ISSN: 1687-7268. DOI: [10.1155/2020/9592836](https://doi.org/10.1155/2020/9592836).
- [117] Michel Vasilevski, Francois Pecheux, Hassan Aboushady, and Laurent de Lamarre. "Modeling heterogeneous systems using SystemC-AMS case study: A Wireless Sensor Network Node". In: *2007 IEEE International Behavioral Modeling and Simulation Workshop*. IEEE, Sept. 2007. DOI: [10.1109/bmas.2007.4437517](https://doi.org/10.1109/bmas.2007.4437517).
- [118] Wan Du, Fabien Mieleveille, David Navarro, and Ian O Connor. "IDEA1: A validated SystemC-based system-level design and simulation environment for wireless sensor networks". In: *EURASIP Journal on Wireless Communications and Networking* 2011.1 (Oct. 2011). ISSN: 1687-1499. DOI: [10.1186/1687-1499-2011-143](https://doi.org/10.1186/1687-1499-2011-143).
- [119] D. Navarro, F. Mieleveille, W. Du, M. Galos, and I. O'Connor. "Towards a design framework for heterogeneous Wireless Sensor Networks". In: *2011 1st International Symposium on Access Spaces (ISAS)*. IEEE, June 2011. DOI: [10.1109/isas.2011.5960925](https://doi.org/10.1109/isas.2011.5960925).
- [120] Fabio Cenni, Olivier Guillaume, Mario Diaz-Nava, and Torsten Maehne. "SystemC-AMS/MDVP-based modeling for the virtual prototyping of MEMS applications". In: *2015 Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS (DTIP)*. IEEE, Apr. 2015. DOI: [10.1109/dtip.2015.7160972](https://doi.org/10.1109/dtip.2015.7160972).

- [121] T. Bechtold, G. Schrag, and L. Feng. "Enabling technologies for system-level simulation of MEMS". In: *2013 14th International Conference on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE)*. IEEE, Apr. 2013. DOI: [10.1109/eurosime.2013.6529994](https://doi.org/10.1109/eurosime.2013.6529994).
- [122] Gabriele Schrag and Gerhard Wachutka. *System-Level Modeling of MEMS Using Generalized Kirchhoffian Networks – Basic Principles*. Feb. 2013. DOI: [10.1002/9783527647132.ch2](https://doi.org/10.1002/9783527647132.ch2).
- [123] Mohammad Shafique, Aric Menon, Kashif Virk, and Jan Madsen. "System-Level Modeling and Simulation of MEMS-based Sensors". In: *2005 Pakistan Section Multitopic Conference*. IEEE, 2005. DOI: [10.1109/inmic.2005.334503](https://doi.org/10.1109/inmic.2005.334503).
- [124] Tino Blochmann, Stephan Gerth, Peter Schneider, and Roland Jancke. "Enhanced behavioral models of MEMS elements for system level verification". In: *ANALOG 2018; 16th GMM/ITG-Symposium*. 2018.
- [125] Peter Schneider, Christian Bayer, Karsten Einwich, and Andreas Kohler. "System level simulation — A core method for efficient design of MEMS and mechatronic systems". In: *International Multi-Conference on Systems, Signals & Devices*. IEEE, Mar. 2012. DOI: [10.1109/ssd.2012.6198066](https://doi.org/10.1109/ssd.2012.6198066).
- [126] Udo Sobe, Enno Bohme, and Karl-Heinz Roochy. "Methodology of virtual testing of trimmable analog circuits". In: *2010 IEEE 16th International Mixed-Signals, Sensors and Systems Test Workshop (IMS3TW)*. IEEE, 2010. DOI: [10.1109/ims3tw.2010.5503010](https://doi.org/10.1109/ims3tw.2010.5503010).
- [127] Po-Chang Wu, Bin-Da Liu, Sheng-Hsiang Tseng, Hann-Huei Tsai, and Ying-Zong Juang. "Digital Offset Trimming Techniques for CMOS MEMS Accelerometers". In: *IEEE Sensors Journal* 14.2 (Feb. 2014), pp. 570–577. ISSN: 1558-1748. DOI: [10.1109/jsen.2013.2284284](https://doi.org/10.1109/jsen.2013.2284284).
- [128] Yannick Hervé and Arnaud Legendre. "Functional Virtual Prototyping for Heterogeneous Systems". In: *Design Technology for Heterogeneous Embedded Systems*. Springer Netherlands, 2012, pp. 223–253. DOI: [10.1007/978-94-007-1125-9_11](https://doi.org/10.1007/978-94-007-1125-9_11).
- [129] Nikolaos Georgouloupoulos and Alkiviadis Hatzopoulos. "Parameterizable Real Number Models for Mixed-Signal Designs Using SystemVerilog". In: *Journal of Electronic Testing* 37.5–6 (Dec. 2021), pp. 685–700. ISSN: 1573-0727. DOI: [10.1007/s10836-021-05977-7](https://doi.org/10.1007/s10836-021-05977-7).
- [130] Sathishkumar Balasubramanian and Pete Hardee. "Solutions for Mixed-Signal SoC Verification Using Real Number Models". In: *Cadence White Paper* (2012).
- [131] Mina Louis, Mohamed Dessouky, and Ashraf Salem. "PLL Real Number Modeling in SystemVerilog". In: *2019 16th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*. IEEE, 2019. DOI: [10.1109/smacd.2019.8795233](https://doi.org/10.1109/smacd.2019.8795233).
- [132] Accellera Systems Initiative Inc. *Verilog-AMS Language Reference Manual*. May 2014.
- [133] Nikolaos Georgouloupoulos and Alkiviadis Hatzopoulos. "Efficiency evaluation of a SystemVerilog-based real number model". In: *2018 7th International Conference on Modern Circuits and Systems Technologies (MOCAS)*. IEEE, May 2018. DOI: [10.1109/mocast.2018.8376642](https://doi.org/10.1109/mocast.2018.8376642).
- [134] Rajeev Narayanan, Naeem Abbasi, Mohamed Zaki, Ghiath Al Sammane, and Sofiene Tahar. "On the simulation performance of contemporary AMS hardware description languages". In: *2008 International Conference on Microelectronics*. IEEE, 2008. DOI: [10.1109/icm.2008.5393509](https://doi.org/10.1109/icm.2008.5393509).

- [135] Wolfgang Scherr and Karsten Einwich. "Beyond real number modeling: Comparison of analog modeling approaches". In: *2020 Forum for Specification and Design Languages (FDL)*. IEEE, 2020. DOI: [10.1109/fdl50818.2020.9232932](https://doi.org/10.1109/fdl50818.2020.9232932).
- [136] Xiao Pan, Carna Zivkovic, and Christoph Grimm. "Virtual prototyping of heterogeneous automotive applications: matlab, SystemC, or both?". In: *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. ASPDAC '19. ACM, Jan. 2019. DOI: [10.1145/3287624.3287629](https://doi.org/10.1145/3287624.3287629).
- [137] Jay Fenlason and Richard Stallman. *GNU gprof - The GNU Profiler*. Accessed: 2024-10-28. URL: https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_mono/gprof.html.
- [138] Karsten Ahnert and Mario Mulansky. *Boost.Numeric.Odeint*. Accessed: 2024-10-16. 2024. URL: https://www.boost.org/doc/libs/1_85_0/libs/numeric/odeint/doc/html/index.html.
- [139] COSEDA Technologies GmbH. *System Level Design Environment COSIDE®*. Accessed: 2024-10-23. 2024. URL: <https://www.coseda-tech.com/coside-overview>.
- [140] S. Adhikari and C. Grimm. "Modeling switched capacitor sigma delta modulator non-idealities in SystemC-AMS". In: *2010 Forum on Specification and Design Languages (FDL 2010)*. IET, 2010. DOI: [10.1049/ic.2010.0155](https://doi.org/10.1049/ic.2010.0155).
- [141] Liliana Andrade, Torsten Maehne, Alain Vachoux, Cédric Ben Aoun, François Pêcheux, and Marie-Minerve Louërat. "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*. IEEE Conference Publications, 2015. DOI: [10.7873/date.2015.0758](https://doi.org/10.7873/date.2015.0758).
- [142] Philipp A. Hartmann, Philipp Reinkemeier, Achim Rettberg, and Wolfgang Nebel. "Modelling control systems in SystemC AMS - Benefits and limitations". In: *2009 IEEE International SOC Conference (SOCC)*. IEEE, Sept. 2009. DOI: [10.1109/soccon.2009.5398043](https://doi.org/10.1109/soccon.2009.5398043).
- [143] T. Rizzi, E. Perez-Bosch Quesada, Ch. Wenger, C. Zambelli, and D. Bertozzi. "Comparative Analysis and Optimization of the SystemC-AMS Analog Simulation Efficiency of Resistive Crossbar Arrays". In: *2021 XXXVI Conference on Design of Circuits and Integrated Systems (DCIS)*. IEEE, 2021. DOI: [10.1109/dcis53048.2021.9666193](https://doi.org/10.1109/dcis53048.2021.9666193).
- [144] Christiane Reuther and Einwich Karsten. "A SystemC AMS extension for controlled modules and dynamic step sizes". In: *Proceeding of the 2012 Forum on Specification and Design Languages*. 2012.
- [145] Liliana Lilibeth Andrade Porras, Torsten Maehne, Marie-Minerve Louërat, and François Pêcheux. "Time Step Control and Threshold Crossing Detection in SystemC AMS 2.0". In: *Huitième colloque du GDRSOC-SIP du CNRS* (June 2013), p. 3.
- [146] Breytner Fernandez-Mesa, Liliana Andrade, and Frederic Petrot. "Synchronization of Continuous Time and Discrete Events Simulation in SystemC". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.7 (2021), pp. 1450–1463. DOI: [10.1109/tcad.2020.3019204](https://doi.org/10.1109/tcad.2020.3019204).
- [147] Benjamin A. Stickler and Ewald Schachinger. "Ordinary Differential Equations: Initial Value Problems". In: *Basic Concepts in Computational Physics*. Springer International Publishing, 2016, pp. 63–83. ISBN: 9783319272658. DOI: [10.1007/978-3-319-27265-8_5](https://doi.org/10.1007/978-3-319-27265-8_5).
- [148] Allan Struthers and Merle Potter. "Numerical Methods for Differential Equations". In: *Differential Equations : For Scientists and Engineers*. Cham: Springer International Publishing, 2019, pp. 373–407. ISBN: 978-3-030-20506-5. DOI: [10.1007/978-3-030-20506-5_6](https://doi.org/10.1007/978-3-030-20506-5_6). URL: https://doi.org/10.1007/978-3-030-20506-5_6.

- [149] M.N. Spijker. "Stiffness in numerical initial-value problems". In: *Journal of Computational and Applied Mathematics* 72.2 (Aug. 1996), pp. 393–406. ISSN: 0377-0427. DOI: [10.1016/0377-0427\(96\)00009-x](https://doi.org/10.1016/0377-0427(96)00009-x).
- [150] Randall J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial and Applied Mathematics, 2007, p. 171. DOI: [10.1137/1.9780898717839](https://doi.org/10.1137/1.9780898717839). eprint: <https://epubs.siam.org/doi/pdf/10.1137/1.9780898717839>. URL: <https://epubs.siam.org/doi/abs/10.1137/1.9780898717839>.
- [151] *Steppers*. Accessed: 2024-10-16. 2024. URL: https://www.boost.org/doc/libs/1_85_0/libs/numeric/odeint/doc/html/boost_numeric_odeint/odeint_in_detail/steppers.html.
- [152] Jennifer Scott and Miroslav Tůma. "Sparse LU Factorizations". In: *Algorithms for Sparse Linear Systems*. Springer International Publishing, 2023, pp. 89–111. ISBN: 9783031258206. DOI: [10.1007/978-3-031-25820-6_6](https://doi.org/10.1007/978-3-031-25820-6_6).
- [153] Alice Lanniel, Tobias Boeser, Thomas Alpert, and Maurits Ortmanns. "Low-Noise Read-out Circuit for an Automotive MEMS Accelerometer". In: *IEEE Open Journal of the Solid-State Circuits Society* 1 (2021), pp. 140–148. ISSN: 2644-1349. DOI: [10.1109/ojsscs.2021.3116125](https://doi.org/10.1109/ojsscs.2021.3116125).
- [154] Inc. The MathWorks. *Embedded Coder*. Accessed: 2024-08-12. URL: <https://www.mathworks.com/products/embedded-coder.html>.
- [155] *R2022b at a Glance*. The MathWorks, Inc. 2022. URL: https://www.mathworks.com/products/new_products/release2022b.html.
- [156] Michele Lora, Sara Vinco, Enrico Fraccaroli, Davide Quaglia, and Franco Fummi. "Analog Models Manipulation for Effective Integration in Smart System Virtual Platforms". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.2 (2018), pp. 378–391. DOI: [10.1109/tcad.2017.2705129](https://doi.org/10.1109/tcad.2017.2705129).
- [157] Enrico Fraccaroli, Michele Lora, and Franco Fummi. "Automatic Generation of Analog/Mixed Signal Virtual Platforms for Smart Systems". In: *IEEE Transactions on Computers* 69.9 (Sept. 2020), pp. 1263–1278. ISSN: 2326-3814. DOI: [10.1109/tc.2020.2970699](https://doi.org/10.1109/tc.2020.2970699).
- [158] Michele Lora, Sara Vinco, and Franco Fummi. "Translation, Abstraction and Integration for Effective Smart System Design". In: *IEEE Transactions on Computers* 68.10 (2019), pp. 1525–1538. ISSN: 2326-3814. DOI: [10.1109/tc.2019.2909209](https://doi.org/10.1109/tc.2019.2909209).
- [159] Ralph Gorgen, Jan-Hendrik Oetjens, and Wolfgang Nebel. "Automatic integration of hardware descriptions into system-level models". In: *2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, Apr. 2012. DOI: [10.1109/ddecs.2012.6219034](https://doi.org/10.1109/ddecs.2012.6219034).
- [160] Liyuan Zhang, Michael Glaß, Nils Ballmann, and Jürgen Teich. "Bridging algorithm and ESL design: Matlab/Simulink model transformation and validation". In: *Proceedings of the 2013 Forum on specification and Design Languages (FDL)*. 2013.
- [161] Henning Kleen, Shangkun Xiao, Ralph Gorgen, Nico Bannow, and Wolfgang Nebel. "Automatische Übersetzung von MATLAB/Simulink-Modellen nach SystemC-AMS". In: *ITG GMM GI*. Feb. 2011.
- [162] F. Mendoza, C. Kollner, J. Becker, and K. D. Muller-Glaser. "An automated approach to SystemC/Simulink co-simulation". In: *2011 22nd IEEE International Symposium on Rapid System Prototyping*. IEEE, 2011. DOI: [10.1109/rsp.2011.5929987](https://doi.org/10.1109/rsp.2011.5929987).
- [163] Michael Glaß, Jürgen Teich, and Liyuan Zhang. "A co-simulation approach for system-level analysis of embedded control systems". In: *2012 International Conference on Embedded Computer Systems (SAMOS)*. IEEE, July 2012. DOI: [10.1109/samos.2012.6404200](https://doi.org/10.1109/samos.2012.6404200).

- [164] Kai Hylla, Jan-Hendrik Oetjens, and Wolfgang Nebel. "Using SystemC for an extended MATLAB/Simulink verification flow". In: *2008 Forum on Specification, Verification and Design Languages*. IEEE, 2008. DOI: [10.1109/fdl.2008.4641449](https://doi.org/10.1109/fdl.2008.4641449).
- [165] Gabriela Nicolescu, Faouzi Bouchhima, and Luiza Gheorghe. "CODIS – A Framework for Continuous/Discrete Systems Co-Simulation". In: *IFAC Proceedings Volumes 39.5* (2006), pp. 274–275. ISSN: 1474-6670. DOI: [10.3182/20060607-3-it-3902.00052](https://doi.org/10.3182/20060607-3-it-3902.00052).
- [166] F. Bouchhima, M. Briere, G. Nicolescu, M. Abid, and E. Aboulhamid. "A SystemC/Simulink Co-Simulation Framework for Continuous/Discrete-Events Simulation". In: *2006 IEEE International Behavioral Modeling and Simulation Workshop*. IEEE, Sept. 2006. DOI: [10.1109/bmas.2006.283461](https://doi.org/10.1109/bmas.2006.283461).
- [167] *Functional Mock-up Interface*. Accessed: 2024-10-18. 2024. URL: <https://fmi-standard.org/>.
- [168] Edmund Widl, Florian Judex, Katharina Eder, and Peter Palensky. "FMI-based co-simulation of hybrid closed-loop control system models". In: *2015 International Conference on Complex Systems Engineering (ICCSE)*. IEEE, Nov. 2015. DOI: [10.1109/complexsys.2015.7385981](https://doi.org/10.1109/complexsys.2015.7385981).
- [169] Robert Lajos Bucs, Luis Gabriel Murillo, Ekaterina Korotcenko, Gaurav Dugge, Rainer Leupers, Gerd Ascheid, Andreas Ropers, and Markus Wedle. "Virtual hardware-in-the-loop co-simulation for multi-domain automotive systems via the functional mock-up interface". In: *2015 Forum on Specification and Design Languages (FDL)*. 2015. URL: <https://ieeexplore.ieee.org/document/7306355>.
- [170] *Transaction Level Model Generation*. Accessed: 2024-07-09. URL: https://de.mathworks.com/help/hdlverifier/transaction-level-model-generation.html?s_tid=CRUX_lftnav.
- [171] Inc. The MathWorks. *Blocks Supported for C/C++ Code Generation*. Accessed: 2024-10-18. 2024. URL: <https://www.mathworks.com/help/referencelist.html?type=block&capability=codegen>.
- [172] J. Stolfi and J. Comba. "Affine arithmetic and its applications to computer graphics". In: *Proc. SIBGRAPI'93 — VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens*. Recife, Brazil, 1993, pp. 9–18.
- [173] R.E. Moore. "Interval Analysis". In: *Prentice-Hall*. 1966.
- [174] Carna Radojicić. "Symbolic Simulation of Mixed-Signal Systems with Extended Affine Arithmetic". PhD thesis. University of Kaiserslautern, Sept. 2016.
- [175] Carna Zivkovic and Christoph Grimm. "Symbolic Simulation of SystemC AMS Without Yet Another Compiler". In: *2018 Forum on Specification and Design Languages (FDL)*. IEEE, Sept. 2018. DOI: [10.1109/fdl.2018.8524061](https://doi.org/10.1109/fdl.2018.8524061).
- [176] Carna Radojicic and Christoph Grimm. "Formal verification of mixed-signal designs using extended affine arithmetic". In: *2016 12th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*. IEEE, 2016. DOI: [10.1109/prime.2016.7519482](https://doi.org/10.1109/prime.2016.7519482).
- [177] Carna Zivkovic and Christoph Grimm. "AADD-Based Symbolic Simulation of SystemC AMS". In: *Lecture Notes in Electrical Engineering*. Springer International Publishing, 2019, pp. 135–160. DOI: [10.1007/978-3-030-31585-6_8](https://doi.org/10.1007/978-3-030-31585-6_8).
- [178] Florian Schupfer, Carna Radojicic, Joseph Wenninger, and Christoph Grimm. "System refinement design flow based on semi-symbolic simulations". In: *IEEE Africon '11*. IEEE, 2011. DOI: [10.1109/afrcon.2011.6072111](https://doi.org/10.1109/afrcon.2011.6072111).

- [179] Carna Radojicic, Florian Schupfer, Michael Rathmair, and Christoph Grimm. "Assertion-based verification of signal processing systems with affine arithmetic". In: *Proceeding of the 2012 Forum on Specification and Design Languages*. 2012.
- [180] Carna Radojicic, Christoph Grimm, Javier Moreno, and Xiao Pan. "Semi-symbolic analysis of mixed-signal systems including discontinuities". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2014. IEEE Conference Publications, 2014. DOI: [10.7873/date.2014.029](https://doi.org/10.7873/date.2014.029).
- [181] Carna Zivkovic and Christoph Grimm. "Nubolic Simulation of AMS Systems with Data Flow and Discrete Event Models". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019. DOI: [10.23919/date.2019.8715278](https://doi.org/10.23919/date.2019.8715278).
- [182] Carna Zivkovic, Christoph Grimm, Markus Olbrich, Oliver Scharf, and Erich Barke. "Hierarchical Verification of AMS Systems With Affine Arithmetic Decision Diagrams". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.10 (2019), pp. 1785–1798. DOI: [10.1109/tcad.2018.2864238](https://doi.org/10.1109/tcad.2018.2864238).
- [183] Tongyu Ding, Liang Zhang, Riccardo Trinchero, Igor S. Stievano, and Flavio G. Canavero. "Worst-Case analysis of electrical and electronic equipment via affine arithmetic". In: *2017 International Conference on Electromagnetics in Advanced Applications (ICEAA)*. IEEE, 2017. DOI: [10.1109/iceaa.2017.8065425](https://doi.org/10.1109/iceaa.2017.8065425).
- [184] Luis Esteban, Juan Antonio Lopez Martin, and Alberto Regadio. "Round-off noise estimation of fixed-point algorithms using Modified Affine Arithmetic and Legendre Polynomials". In: *2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*. IEEE, 2020. DOI: [10.1109/dcis51330.2020.9268668](https://doi.org/10.1109/dcis51330.2020.9268668).
- [185] Christoph Lauter and Anastasia Volkova. "A Framework for Semi-Automatic Precision and Accuracy Analysis for Fast and Rigorous Deep Learning". In: *2020 IEEE 27th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2020. DOI: [10.1109/arith48897.2020.00023](https://doi.org/10.1109/arith48897.2020.00023).
- [186] Ahmad Tarraf and Lars Hedrich. "Analog Circuit Abstraction to SystemC-AMS Secured by Affine Forms". In: *SMACD / PRIME 2021; International Conference on SMACD and 16th Conference on PRIME*. 2021.
- [187] Christoph Zivkovic Carna/Grimm. *Github AADD*. Accessed: 2024-09-26. URL: <https://github.com/tukcps/AADD>.

List of Publications

[P1] Alexandra Küster, Rainer Dorsch, Christian Haubelt, and Karsten Einwich. "Virtual Prototyping in SystemC AMS for Validation of Tight Sensor/Firmware Interaction in Smart Sensors". In: *2022 Forum on Specification & Design Languages (FDL)*. 2022, pp. 1–8. DOI: [10.1109/FDL56239.2022.9925660](https://doi.org/10.1109/FDL56239.2022.9925660).

[P2a] Alexandra Küster, Rainer Dorsch, and Christian Haubelt. "Structural Generation of Virtual Prototypes for Smart Sensor Development in SystemC-AMS from Simulink Models". In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2023, pp. 1–2. DOI: [10.23919/DATE56975.2023.10137224](https://doi.org/10.23919/DATE56975.2023.10137224). [Extended abstract]

[P2b] A. Küster, R. Dorsch and C. Haubelt. "Structural Generation of Virtual Prototypes for Smart Sensor Development in SystemC-AMS from Simulink Models". In: *2023 Smart Systems Integration Conference and Exhibition (SSI)*. 2023, pp. 1–6. DOI: [10.1109/SSI58917.2023.10387948](https://doi.org/10.1109/SSI58917.2023.10387948). [Full paper]

[P3] A. Küster, R. Dorsch and C. Haubelt. "On the Performance Differences of SystemC and SystemC AMS: A Guideline for Real Number Modeling and Beyond". In: *DVCon Europe 2023; Design and Verification Conference and Exhibition Europe*. 2023, pp. 21–26.

[P4] A. Küster, R. Dorsch and C. Haubelt. "Heterogeneous Virtual Prototypes for Smart Sensor Development: Requirements and Impact," MBMV 2024; 27. Workshop, Kaiserslautern, 2024, pp. 1-4.

[P5] A. Küster, R. Dorsch, and C. Haubelt. "Symbolic Simulation of Sensor Trimming Algorithms Using Affine Arithmetic". In: *2024 IEEE International Symposium on Inertial Sensors and Systems (INERTIAL)*. 2024, pp. 1–4. DOI: [10.1109/INERTIAL60399.2024.10502057](https://doi.org/10.1109/INERTIAL60399.2024.10502057).

[P6] A. Küster, R. Dorsch and C. Haubelt. "Adaptive ODE Solvers for Timed Data Flow Models in SystemC-AMS," 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE), Valencia, Spain, 2024, pp. 1-6, DOI: [10.23919/DATE58400.2024.10546509](https://doi.org/10.23919/DATE58400.2024.10546509).

[P7] A. Küster, R. Dorsch and C. Haubelt. "Towards Fast Heterogeneous Virtual Prototypes: Increasing the Solver Efficiency in SystemC AMS," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, DOI: [10.1109/TCAD.2025.3554612](https://doi.org/10.1109/TCAD.2025.3554612)

Appendix A

Implementation Details on the Solver Enhancements in SystemC AMS

The following section provides further details regarding the implementation of the solver enhancements introduced in chapter 5. They are integrated into the proof-of-concept implementation of SystemC AMS, which is provided by ACCELLERA [14].

A.1 Variable-Step Solvers

The incorporation of the variable-step ODE solvers into SystemC AMS requires to provide two distinct implementations. The ELN MoC and the LSF MoC share a common solver implementation structure, which results in a similar implementation for both. However, the TDF instances that trigger ODE solvers are handled individually. Solely the fundamental mathematical operations within the analog solver are shared between the two. In consequence, the extension for the newly introduced features is analogous for the ELN and LSF MoC, but somewhat disparate for the TDF MoC.

Figure A.1 depicts an excerpt of the class diagram for the SystemC AMS implementation. The three TDF instances that employ ODE solvers are illustrated alongside their underlying implementing classes. Those utilized directly by the user (the upper row) define the interfaces as described in the language reference manual [79], and each possess an instance of their underlying proxy as an attribute. These proxies implement the actual functionality. A significant proportion of the functions accessible to the user merely invoke their equivalent method of the proxy. The proxy for the LTF instance in zero-pole form is derived from the proxy for the numerator-denominator type LTF. The SS proxy is distinguished from other proxies by its treatment of coefficients as matrices and states as vectors. However, the underlying working principle is similar across all proxies. The methods and attributes indicated in blue have remained unaltered from the proof-of-concept implementation. Those in purple have been adapted, and those in green have been added. Thereby, any adaptations do not impact the behavior of the standard features of SystemC AMS.

The new public methods of the upper classes have already been introduced in section 5.7. They call the respective methods in their proxies in order to set the newly introduced parameters that configure the solver. The most noteworthy methods for the processing of the ODE are the following two:

initialize_equation_system() As the name already indicates, this method initializes the equation system and prepares the underlying solver matrices and triggers the LU decomposition of the solver matrix Z_k (see Ch. 5, Eq. (5.42)). Hence, it is called each time either a coefficient or the timestep of the ODE changes. When timestep quantization is used, it derives the quantized step size from the given estimation, checks whether this timestep has already been used and restores the corresponding decomposed matrices if applicable.

calculate() This method handles the main calculation steps to determine a new output sample for the ODE. If a variable-step solver is used, it calls its adaptive counterpart denoted as *calculate_adaptive()*. The latter includes the complete processing loop as illustrated in Figures 5.4 and 5.7. Thereby, it calls the method *initialize_equation_system()* whenever a new timestep shall be tried. Moreover, it calls either the built-in trapezoidal solver or a Runge-Kutta solver from the Boost.Numeric.OdeInt library [138].

In the event that an ELN or LSF cluster is utilized as TCI, the solver control is managed by a novel primitive. It is called *sca_solver_control* and enables the user to set the solver parameters exclusively within the constructor. This guarantees that they remain fixed throughout the entire simulation. It is derived from either the *sca_eln::sca_module* or the *sca_lsf::sca_module* class. Both versions of the *sca_module* class contain a virtual method, *matrix_stamps()*, which each derived primitive is required to provide. Typically, the method is employed to incorporate the component's equations into the state matrix in accordance with the tenets of a modified nodal analysis. In contrast, the novel primitive employs this method to configure the ODE solver, transmitting the values of its local attributes to the respective solver. Additionally, a *change_timestep* method is registered as *POST_SOLVE* callback, such that it is executed after each solver calculation. For reference, the implementation of the virtual method is provided in Listing A.1.

The remaining actions, including the execution of the solver's try-and-error loop, estimation of the local error, and calculation of the next step size, are performed by an instance of the class *sca_linear_solver*, which invokes the same underlying functions of the analog solver as the TDF proxies.

LISTING A.1: Implementation of the method *matrix_stamps* for the new primitive *sca_eln::sca_solver_control*. The method is required for each ELN and LSF primitive. It usually adds equations to the state matrix. Here, it configures the ODE solver instead.

```
namespace sca_eln
{
void sca_solver_control::matrix_stamps()
{

    // no influence on matrix itself

    // control solver
    set_solver_parameter("stepper", stepper_type);
    set_solver_parameter("eps_abs", eps_abs);
    set_solver_parameter("eps_rel", eps_rel);
    set_solver_parameter("ax", ax);
    set_solver_parameter("axp", axp);
    set_solver_parameter("h_min", h_min);
    set_solver_parameter("h_max", h_max);
    set_solver_parameter("quant_sw", (monitor_switching?1.0:0.0));

    if(std::strcmp(stepper_type.get().c_str(), "fixed") != 0)
    {
        add_method(POST_SOLVE, SCA_VMPTR(sca_solver_control::change_timestep));
    }
}
}
```

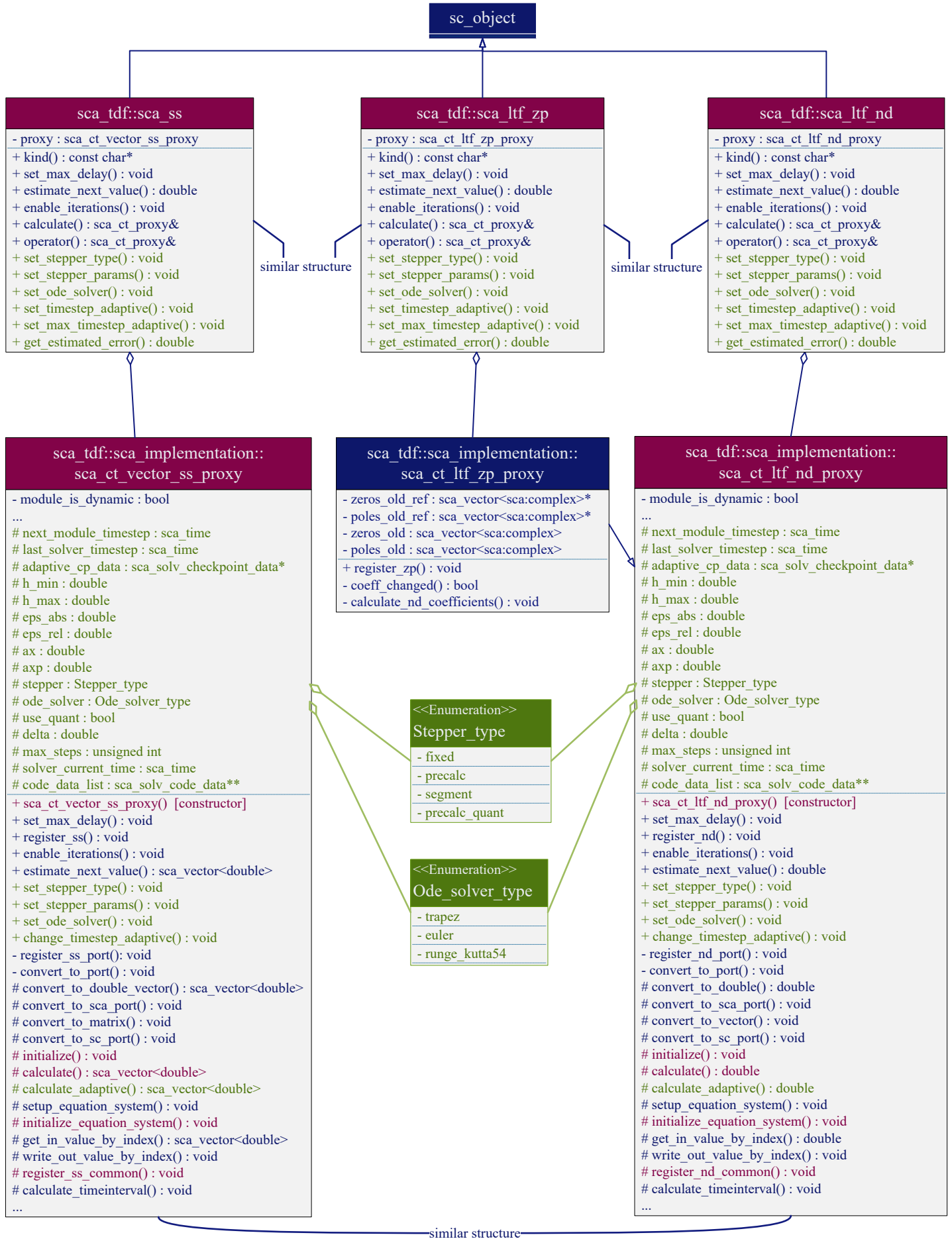


FIGURE A.1: Excerpt from the SystemC AMS class diagram denoting manipulated content (purple) and new one (green).

A.2 Switch Monitoring

In order to implement the switch monitoring approach, two new classes have been introduced, namely one for the switch monitor itself and a child class inherent to each switch that handles the respective updates whenever a switch changes its state. The implementation of these classes is provided in Listing A.2 and A.3. The current implementation is based on the data type `uint64_t` natively limiting the approach to a maximum of 64 switches. Please note that the implementation could similarly be done with the `std::bitset` type to enable even higher numbers of switches.

LISTING A.2: Implementation of the switch monitor. A single instance is created when switch monitoring is enabled.

```
class switch_monitor
{
public:
    switch_monitor();

    uint64_t get_current_mask(); // return mask
    void insert_mask_to_list();
    int find_mask_in_list(); // return index (-1 if mask not in list)
    unsigned int get_state_length();
    void print_statistics(); // for debug purposes

protected:
    unsigned int add_switch(bool init_state);
    void update_mask(bool state, unsigned int idx);

private:
    static uint64_t mask;
    static unsigned int state_length;
    static std::vector<uint64_t> masks_occured;
};

switch_monitor::switch_monitor() : mask(0), state_length(0)
{
}

uint64_t switch_monitor::get_current_mask()
{
    return mask;
}

void switch_monitor::insert_mask_to_list()
{
    masks_occured.push_back(mask);
}

int switch_monitor::find_mask_in_list()
{
    int idx = 0;
    for(auto it = begin(masks_occured); it != end(masks_occured); it++)
    {
        if((*it ^ mask) == 0)
        {
            return idx;
        }
    }
}
```

```
        idx++;
    }
    return -1;
}

unsigned int switch_monitor::get_state_length()
{
    return state_length;
}

void switch_monitor::print_statistics()
{
    std::cout << "Total number of registered switches: " << state_length << "\n"
    << "Total number of registered states: " << masks_occured.size() << std::endl;
}

unsigned int switch_monitor::add_switch(bool init_state)
{
    if(state_length < 64)
    {
        update_mask(init_state, state_length);
        state_length++;
        return (state_length-1);
    }
    else
    {
        std::ostringstream str;
        str << "Number of switches exceeds maximum mask length (64). This feature is
            currently not supported.";
        SC_REPORT_ERROR("SystemC-AMS", str.str().c_str());
    }
}

void switch_monitor::update_mask(bool state, unsigned int idx)
{
    if(state)
    {
        mask |= uint64_t(1) << idx;
    }
    else
    {
        mask &= ~(uint64_t(1) << idx);
    }
}
```

LISTING A.3: Implementation of the switch handler. Each switch comprises its own instance of this class.

```
class switch_handler : public switch_monitor
{
public:
    switch_handler(bool init_state);

    void trigger_mask_update(bool val);

private:
    unsigned int idx;
};

switch_handler::switch_handler(bool init_state) : idx(add_switch(this->init_state))
{
}

void switch_handler::trigger_mask_update(bool val)
{
    this->update_mask(val, idx);
}
```

Appendix B

Illustrative Example of the Outcome of the Translation Process

Section 6.4 discusses an illustrative example to demonstrate the behavior of the automated translation process from MATLAB/Simulink to SystemC AMS. The code snippets below provide exemplary outcomes of the automated procedure. Listing B.1 shows the header that is generated by the Embedded Coder for the out-of-plane MEMS channel. Its TDF wrapper module is provided in Listings B.2 and B.3, respectively. Additionally, the code of the hierarchical module comprising the entire MEMS with all three channels is given in Listing B.4.

LISTING B.1: mems_ax_oop.h (Generated by the Embedded Coder [154].)

```
#ifndef RTW_HEADER_mems_ax_oop_h_
#define RTW_HEADER_mems_ax_oop_h_
#include "rtwtypes.h"
#include "mems_ax_oop_types.h"

// Macros for accessing real-time model data structure
#ifndef rtmGetErrorStatus
#define rtmGetErrorStatus(rtm)      ((rtm)->errorStatus)
#endif

#ifndef rtmSetErrorStatus
#define rtmSetErrorStatus(rtm, val) ((rtm)->errorStatus = (val))
#endif

// Class declaration for model mems_ax_oop
class mems_ax_oop final
{
// public data and function members
public:
// Block states (default storage) for system '<Root>'
struct DW {
real_T Integrator1_DSTATE; // '<S1>/Integrator1'
real_T Integrator_DSTATE;  // '<S1>/Integrator'
};

// External inputs (root inport signals with default storage)
struct ExtU {
real_T vc1;           // '<Root>/vc1'
real_T vc2;           // '<Root>/vc2'
real_T vcm;           // '<Root>/vcm'
real_T acc_SI;        // '<Root>/acc_SI'
};

// External outputs (root outports fed by signals with default storage)
```

```

struct ExtY {
real_T dC;                // '<Root>/dC'
bus_dbg dbg;              // '<Root>/dbg'
};

// Parameters (default storage)
struct P {
real_T Constant2_Value[2]; // Expression: CpParaC0
// Referenced by: '<Root>/Constant2'

real_T Constant_Value;     // Expression: d_gapX_electr0
// Referenced by: '<Root>/Constant'

real_T Constant1_Value;    // Expression: C0
// Referenced by: '<Root>/Constant1'

real_T Integrator1_gainval; // Computed Parameter: Integrator1_gainval
// Referenced by: '<S1>/Integrator1'

real_T Integrator1_IC;     // Expression: 0
// Referenced by: '<S1>/Integrator1'

real_T Integrator1_UpperSat; // Expression: d_gapX_2_bumper-xOffs
// Referenced by: '<S1>/Integrator1'

real_T Integrator1_LowerSat; // Expression: -d_gapX_2_bumper+xOffs
// Referenced by: '<S1>/Integrator1'

real_T Constant_Value_f;   // Expression: xOffs
// Referenced by: '<S1>/Constant'

real_T Gain2_Gain;         // Expression: 1/mass
// Referenced by: '<S1>/Gain2'

real_T Integrator_gainval; // Computed Parameter: Integrator_gainval
// Referenced by: '<S1>/Integrator'

real_T Integrator_IC;     // Expression: 0
// Referenced by: '<S1>/Integrator'

real_T Gain1_Gain;        // Expression: dfriect/mass
// Referenced by: '<S1>/Gain1'

real_T Gain_Gain;         // Expression: kHooke/mass
// Referenced by: '<S1>/Gain'

};

// Real-time Model Data Structure
struct RT_MODEL {
const char_T * volatile errorStatus;
};

// Copy Constructor
mems_ax_oop(mems_ax_oop const&) = delete;

// Assignment Operator

```

```

mems_ax_oop& operator= (mems_ax_oop const&) & = delete;

// Move Constructor
mems_ax_oop(mems_ax_oop &&) = delete;

// Move Assignment Operator
mems_ax_oop& operator= (mems_ax_oop &&) = delete;

// Real-Time Model get method
mems_ax_oop::RT_MODEL * getRTM();

// External inputs
ExtU rtU;

// External outputs
ExtY rtY;

// Tunable parameters
static P rtP;

// model initialize function
void initialize();

// model step function
void step();

// model terminate function
static void terminate();

// Constructor
mems_ax_oop();

// Destructor
~mems_ax_oop();

// private data and function members
private:
// Block states
DW rtDW;

// Real-Time Model
RT_MODEL rtM;
};

#endif                                     // RTW_HEADER_mems_ax_oop_h_

```

LISTING B.2: mems_ax_oop_module.h

```

#ifndef _MEMS_AX_OOP_MODULE_H_
#define _MEMS_AX_OOP_MODULE_H_

#include <systemc-ams>
#include "mems_ax_oop.h"

class mems_ax_oop_module : public sca_tdf::sca_module
{
public:

```

```

sca_tdf::sca_in<double> vc1;
sca_tdf::sca_in<double> vc2;
sca_tdf::sca_in<double> vcm;
sca_tdf::sca_in<double> acc_SI;
sca_tdf::sca_out<double> dC;
sca_tdf::sca_out<double> dbg_x;
sca_tdf::sca_out<double> dbg_Fel;

mems_ax_oop_module(sc_core::sc_module_name nm);

void set_attributes();
void initialize();
void processing();
void end_of_simulation();

private:
mems_ax_oop modelObj;
};

#endif // _MEMS_AX_OOP_MODULE_H_

```

LISTING B.3: mems_ax_oop_module.cpp

```

#include "mems_ax_oop_module.h"

mems_ax_oop_module::mems_ax_oop_module(sc_core::sc_module_name nm)
: vc1("VC1")
, vc2("VC2")
, vcm("VCM")
, acc_SI("ACC_SI")
, dC("DC")
, dbg_x("DBG_X")
, dbg_Fel("DBG_FEL")
{}

void mems_ax_oop_module::set_attributes()
{
    set_timestep(1e-6, sc_core::SC_SEC);
}

void mems_ax_oop_module::initialize()
{
    modelObj.initialize();
}

void mems_ax_oop_module::processing()
{
    modelObj.rtU.vc1 = vc1.read();
    modelObj.rtU.vc2 = vc2.read();
    modelObj.rtU.vcm = vcm.read();
    modelObj.rtU.acc_SI = acc_SI.read();

    modelObj.step();

    dC.write(modelObj.rtY.dC);
    dbg_x.write(modelObj.rtY.dbg_x);
}

```

```

    dbg_Fel.write(modelObj.rtY.dbg.Fel);
}

void mems_ax_oop_module::end_of_simulation()
{
    modelObj.terminate();
}

```

LISTING B.4: mems_module.h

```

#ifndef _MEMS_MODULE_H_
#define _MEMS_MODULE_H_

#include "mems_ax_ip_module.h"
#include "mems_ax_oop_module.h"

class mems_module : public sc_core::sc_module
{
public:
    sca_tdf::sca_in<double> acc_x;
    sca_tdf::sca_in<double> acc_y;
    sca_tdf::sca_in<double> acc_z;
    sca_tdf::sca_in<double> asic2mems_vcm;
    sca_tdf::sca_in<double> asic2mems_vc1x;
    sca_tdf::sca_in<double> asic2mems_vc1y;
    sca_tdf::sca_in<double> asic2mems_vc1z;
    sca_tdf::sca_in<double> asic2mems_vc2x;
    sca_tdf::sca_in<double> asic2mems_vc2y;
    sca_tdf::sca_in<double> asic2mems_vc2z;

    sca_tdf::sca_out<double> dCx;
    sca_tdf::sca_out<double> dbg_x_x;
    sca_tdf::sca_out<double> dbg_x_Fel;
    sca_tdf::sca_out<double> dCy;
    sca_tdf::sca_out<double> dbg_y_x;
    sca_tdf::sca_out<double> dbg_y_Fel;
    sca_tdf::sca_out<double> dCz;
    sca_tdf::sca_out<double> dbg_z_x;
    sca_tdf::sca_out<double> dbg_z_Fel;

    mems_ax_ip_module *mems_ax_ip_i0;
    mems_ax_ip_module *mems_ax_ip_i1;
    mems_ax_oop_module *mems_ax_oop_i0;

    mems_module(sc_core::sc_module_name nm)
    : sc_module(nm)
    , mems_ax_ip_i0(new mems_ax_ip_module("MEMS_AX_IP_0"))
    , mems_ax_ip_i1(new mems_ax_ip_module("MEMS_AX_IP_1"))
    , mems_ax_oop_i0(new mems_ax_oop_module("MEMS_AX_OOP_0"))
    {
        mems_ax_ip_i0->vc1(asic2mems_vc1x);
        mems_ax_ip_i0->vc2(asic2mems_vc2x);
        mems_ax_ip_i0->vcm(asic2mems_vcm);
        mems_ax_ip_i0->acc_SI(acc_x);
        mems_ax_ip_i0->dC(dCx);
        mems_ax_ip_i0->dbg_x(dbg_x_x);
    }

```

```
    mems_ax_ip_i0->dbg_Fel(dbg_x_Fel);

    mems_ax_ip_i1->vc1(asic2mems_vc1y);
    mems_ax_ip_i1->vc2(asic2mems_vc2y);
    mems_ax_ip_i1->vcm(asic2mems_vcm);
    mems_ax_ip_i1->acc_SI(acc_y);
    mems_ax_ip_i1->dC(dCy);
    mems_ax_ip_i1->dbg_x(dbg_y_x);
    mems_ax_ip_i1->dbg_Fel(dbg_y_Fel);

    mems_ax_oop_i0->vc1z(asic2mems_vc1z);
    mems_ax_oop_i0->vc2z(asic2mems_vc2z);
    mems_ax_oop_i0->vcm(asic2mems_vcm);
    mems_ax_oop_i0->acc_SI(acc_z);
    mems_ax_oop_i0->dC(dCz);
    mems_ax_oop_i0->dbg_x(dbg_z_x);
    mems_ax_oop_i0->dbg_Fel(dbg_z_Fel);
}
};

#endif // _MEMS_MODULE_H_
```

Appendix C

Additional Notes to Affine Arithmetic

C.1 Affine Division

Division is inherently not a linear operation. Hence, an affine approximation must be applied. One common approach is to interpret a division as a multiplication with the inverse. The inverse is calculated performing a Taylor series expansion around the center value up to first order.

$$\frac{1}{\tilde{x}} \stackrel{AA}{\approx} \frac{1}{x_0} - \sum_{i=1}^m \frac{x_i \varepsilon_i}{x_0^2} + e_x \quad (C.1)$$

$$(C.2)$$

Taking a similar approach, the affine approximation of the division operation yields

$$\begin{aligned} \frac{\tilde{x}}{\tilde{y}} &= \tilde{x} \cdot \frac{1}{\tilde{y}} \\ &\stackrel{AA}{\approx} \frac{x_0}{y_0} + \sum_{i=1}^m \frac{x_i \varepsilon_i}{y_0} - \sum_{i=1}^m \frac{x_0 y_i \varepsilon_i}{y_0^2} + e_{xy} \\ &= \frac{x_0}{y_0} + \sum_{i=1}^m \left(\frac{x_i}{y_0} - \frac{x_0 y_i}{y_0^2} \right) \varepsilon_i + e_{xy} \end{aligned} \quad (C.3)$$

Various approaches exist to evaluate the offset term. The *minimum range approximation* evaluates the minimum range of the final result. Its major objective is to minimize the over approximation of the interval boundaries [174]. In contrast, the *Chebyshev approximation* is adopted when sensitivity analysis is the major driver for the simulation.

Please note that the cancellation of deviation symbols by division still works except that it creates an offset term:

$$\begin{aligned} \frac{\tilde{x}}{\tilde{x}} &\stackrel{AA}{\approx} \left(x_0 + \sum_{i=1}^m x_i \varepsilon_i \right) \cdot \left(\frac{1}{x_0} - \sum_{i=1}^m \frac{x_i \varepsilon_i}{x_0^2} + e_x \right) \\ &\stackrel{AA}{\approx} 1 + \sum_{i=1}^m \frac{x_i \varepsilon_i}{x_0} - \sum_{i=1}^m \frac{x_i \varepsilon_i}{x_0} + e_{xx} \\ &= 1 + e_{xx} \end{aligned} \quad (C.4)$$

However, this special case can easily be caught. A better implementation of affine division tries to express the nominator as multiplicative of the denominator first, i.e. it checks whether

$$\frac{x_i}{y_i} = \frac{x_j}{y_j} = c \quad \forall \quad 0 \leq i, j \leq m, c \in \mathbb{R} \quad (C.5)$$

If a common real-valued factor c is found, it directly becomes the result of the division. Otherwise the quotient is evaluated according to (C.3).

C.2 Examples of Approximations of Nonlinear Functions in AA2

All consecutive examples are based on an affine form

$$\tilde{x} = x_0 + \sum_{i=1}^m x_i \varepsilon_i, \quad \varepsilon_i \in [-1, 1] \quad (\text{C.6})$$

with all coefficients $x_0, x_i \in \mathbb{R}$ and $m \geq 0$. It may additionally be related to an offset matrix

$$E_x = \begin{pmatrix} e_{x,11} & \cdots & e_{x,1m} \\ & \ddots & \vdots \\ \mathbf{0} & & e_{x,mm} \end{pmatrix} \quad (\text{C.7})$$

which is handled separately for the upcoming calculations.

Example 1: Inversion

$$\tilde{y} = f(\tilde{x}) = \frac{1}{\tilde{x}} \quad (\text{C.8})$$

The approximation of the function above is based on a Taylor expansion around the center value. For any function $f(\tilde{x})$, the Taylor series around the center value derives to

$$f(\tilde{x}) \approx f(x_0) + \frac{f'(x_0)}{1!} \sum_{i=1}^m x_i \varepsilon_i + \frac{f''(x_0)}{2!} \left(\sum_{i=1}^m x_i \varepsilon_i \right)^2 + \dots + \frac{f^n(x_0)}{n!} \left(\sum_{i=1}^m x_i \varepsilon_i \right)^n. \quad (\text{C.9})$$

The affine form $\tilde{y}$ represents the expansion to first order only. The second order is still calculated and mapped onto the corresponding offset matrix E_y as shown below:

$$\begin{aligned} \tilde{y} &\approx \frac{1}{x_0} - \sum_{i=1}^m \frac{x_i}{x_0^2} \varepsilon_i + \sum_{i=1}^m \sum_{j=1}^m \frac{x_i x_j}{x_0^3} \varepsilon_i \varepsilon_j \\ &\stackrel{AA}{\approx} \frac{1}{x_0} - \sum_{i=1}^m \frac{x_i}{x_0^2} \varepsilon_i \\ e_{y,ij} &\stackrel{AA2}{=} \frac{1}{e_{x,ij}} + \frac{x_i x_j}{x_0^3} \quad \forall i, j \leq m \end{aligned} \quad (\text{C.10})$$

Example 2: Square-root

The second example evaluates the square-root of an affine form such that

$$\tilde{y} = f(\tilde{x}) = \sqrt{\tilde{x}}. \quad (\text{C.11})$$

This nonlinear function can similarly be approximated based on a Taylor series. The Taylor expansion derives to

$$\begin{aligned}
 \tilde{y} &\approx \sqrt{x_0} + \sum_{i=1}^m \frac{x_i}{2\sqrt{x_0}} \varepsilon_i - \sum_{i=1}^m \sum_{j=1}^m \frac{x_i x_j}{8\sqrt{x_0^3}} \varepsilon_i \varepsilon_j \\
 &\stackrel{AA}{\approx} \sqrt{x_0} + \sum_{i=1}^m \frac{x_i}{2\sqrt{x_0}} \varepsilon_i \\
 e_{y,ij} &\stackrel{AA2}{=} \sqrt{e_{x,ij}} - \frac{x_i x_j}{8\sqrt{x_0^3}} \quad \forall i, j \leq m.
 \end{aligned} \tag{C.12}$$

C.3 Implementation of Special Operators

Rounding

The rounding operation as suggested deviates from Hansen's form as it introduces a new deviation symbol to account for the additional uncertainty. Doing so, the granularity of the trim circuit becomes visible. Listing C.1 shows the implementation of the operator as part of the class AAF which specifies affine forms.

LISTING C.1: Affine rounding operation.

```

AAF AAF::round(){
    double min = this->getMin();
    double max = this->getMax();
    int length = this->getLength();

    AAF *temp = new AAF();

    if(std::round(min) == std::round(max)) // rounding reduces interval to single value
    {
        temp->cvalue = std::round(min);
    }
    else
    {
        temp->cvalue = this->cvalue;
        temp->indexes = new unsigned[length+1];
        temp->deviations = new double[length+1];

        // copy uncertainties from source
        for(int i = 0; i < length; i++)
        {
            temp->indexes[i] = this->indexes[i];
            temp->deviations[i] = this->deviations[i];
        }
        // add new deviation symbol
        temp->indexes[length] = next_index(); // add lowest index not yet in use
        temp->deviations[length] = 0.5;
    }
    return (*temp);
}

```

Bit Access

The bit access operator as suggested is a bit more complex as it requires to split a coherent interval into multiple segments. To illustrate the concept, an example is provided that derives the corresponding AADD. An affine form

$$\tilde{x} = 4 + 2\varepsilon_1 - \varepsilon_2 \quad (\text{C.13})$$

represents an interval $[1, 7]$. If it is interpreted as 4-bit word and the bit at index two is to be evaluated, the corresponding function call is then given by the expression $x.\text{getBit}(4, 2)$. It is evident that the bit is set exclusively if $\tilde{x} \geq 4$. Consequently, the expected AADD must comprise one internal node that reflects this condition and two leaves with values of one and zero, respectively. In classical arithmetic, the evaluation of whether a bit is set is a straightforward process, which can be performed using the following formula:

$$b_i = \left\lfloor \frac{x}{2^i} \right\rfloor \% 2 \quad (\text{C.14})$$

with b_i denoting the i^{th} bit and $\%$ the modulo operator. Here, this formula is used in a modified fashion. First the weighting is done calculating

$$\frac{\tilde{x}}{2^i} \quad (\text{C.15})$$

It should be noted that this does not require an approximation, as the denominator is a concrete value. Afterwards, the floor operation is applied to minimum and maximum of the interval spanned by the resulting affine form. In the given example, this evaluates to

$$\tilde{a} = \frac{\tilde{x}}{2^2} = 1 + 0.5\varepsilon_1 - 0.25\varepsilon_2 \quad (\text{C.16})$$

$$\lfloor a_{\min} \rfloor = \lfloor 1 - 0.5 - 0.25 \rfloor = 0 \quad \lfloor a_{\max} \rfloor = \lfloor 1 + 0.5 + 0.25 \rfloor = 1 \quad (\text{C.17})$$

Given that the difference between these integers is one, it is evident that further subdivision is unnecessary. Consequently, the AADD will possess a depth of two, comprising a single internal node and two leaves. If $\tilde{a} \geq 1$, the bit will be one because $1 \% 2 = 1$, otherwise it will be zero. Please note that the condition $\tilde{a} \geq 1$ and the condition $\tilde{x} \geq 4$, which was derived in the beginning, are equivalent.

For larger intervals, the situation becomes a bit more complex. The second affine form under test is

$$\tilde{x} = 16 + 4\varepsilon_1 + 4\varepsilon_2 - 4\varepsilon_3 \quad (\text{C.18})$$

that spans the interval $[4, 28]$. Once more, the bit at index two is to be evaluated. The condition $\tilde{x} \geq 4$ still holds. However, this time it is a necessary but not a sufficient condition. For instance, the interval $8 \leq \tilde{x} < 12$ fulfills the condition, yet b_2 equals to zero in that case. Therefore, the resulting AADD has multiple levels of depth. Adopting the same approach as previously described yields

$$\tilde{a} = \frac{\tilde{x}}{4} = 4 + \varepsilon_1 + \varepsilon_2 - \varepsilon_3 \quad (\text{C.19})$$

$$\lfloor a_{\min} \rfloor = \lfloor 4 - 1 - 1 - 1 \rfloor = 1 \quad \lfloor a_{\max} \rfloor = \lfloor 4 + 1 + 1 + 1 \rfloor = 7. \quad (\text{C.20})$$

Given that the final operation is modulo two, it follows that the final interval length must be no

greater than one. Thus, the interval is bisected in order to proceed with the subsequent layer of depth.

$$\lfloor a_{mid} \rfloor = \left\lfloor \frac{\lfloor a_{min} \rfloor + \lfloor a_{max} \rfloor}{2} \right\rfloor = \left\lfloor \frac{1+7}{2} \right\rfloor = 4 \quad (\text{C.21})$$

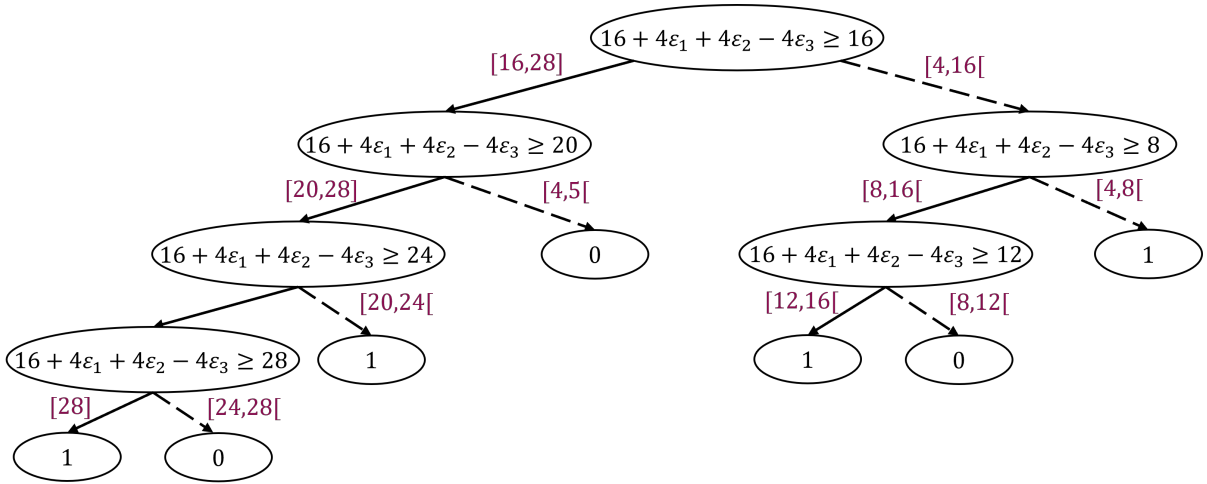
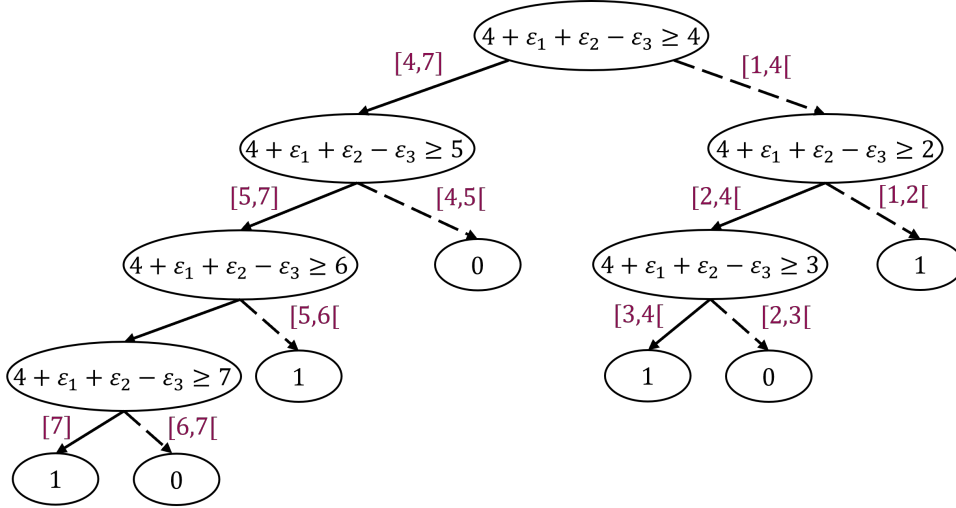
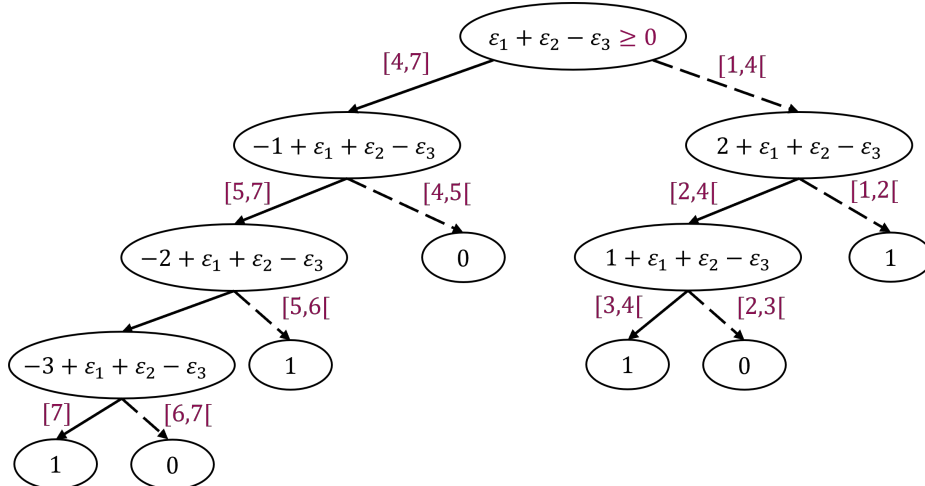
The first condition to store is hence $\tilde{a} \geq 4$. If the branch where this condition is true continues with the interval $[4,7]$, the other branch with the interval $[1,4)$. Again, these intervals are too broad and are split further following the same sequence until the intervals span only two neighbouring integers. The resulting AADD is given in Fig. C.1. For purposes of clarity, two additional transformation steps are denoted, which do not alter the logical meaning but serve to elucidate the relationship to the initial affine form. The implementation of the function described is given in Listing C.2. Thereby, $ITE(c, t, f)$ represents a function that creates an AADD with internal condition c , AADD subtree t if the condition is true and AADD subtree f else.

LISTING C.2: Affine bit access operator.

```
AADD AAF::getBit(numBits, idx)
{
    double factor = pow(2,idx);
    AAF a = (*this)/factor;
    int min = floor(a.getMin());
    int max = floor(a.getMax());
    int start = ceil((max-min/2));

    if(min < -1*factor || max > factor-1)
    {
        std::cout << "Error: The interval extends the range of an " << numBits << "-bit
            word!" << std::endl;
        return NULL;
    }
    AADD *temp = build_tree(a, min, max);
    return (*temp);
}

AADD* build_tree(AAF a, int minimum, int maximum)
{
    AADD *temp = new AADD;
    if(maximum-minimum == 1)
    {
        if(minimum %2 == 0)
        {
            temp = ITE(a-maximum, 1.0, 0.0);
        }
        else
        {
            temp = ITE(a-maximum, 0.0, 1.0);
        }
    }
    else
    {
        int mid = floor((maximum-minimum)/2.0);
        temp = ITE(a-mid, build_tree(a, minimum, mid), build_tree(a, mid, maximum));
    }
    return temp;
}
```

(A) Decision tree representing the value of bit b_2 when interpreting the affine form $\tilde{x}$ as binary word.(B) Mathematical equivalent view dividing each side of each condition by $2^i = 4$.(C) Representation of the decision tree from (b) as AADD. The condition (≥ 0) becomes implicit and only the affine form is stored in internal nodes.FIGURE C.1: Derivation of the resulting AADD when applying a bit access operation to affine form $\tilde{x} = 16 + 4\varepsilon_2 + 4\varepsilon_1 - 4\varepsilon_3$ using index $i = 2$. (A) Intuitive representation as decision tree. (B) Mathematical Reduction. (C) Resulting AADD.

Declaration of Aids Used

I hereby acknowledge the use of DeepL Write¹ in supporting me to review and polish my own writing in terms of grammar and academic style. The tool was exclusively used for paraphrasing such that the structure and the content of the presented work are entirely my own creation. Furthermore, the outcome of DeepL Write was critically reviewed to ensure that the intent of my original text had not been changed, and revised using my own expressions.

Notice on Copyright of Certain Figures

Some figures within this thesis have been previously published by the author of this work in the papers given in the list of publications. Although created by the author of this thesis, they are thus subject to IEEE Copyright and therefore marked as such. The author of this work is permitted to reuse these figures according to the IEEE Copyright regulations². The author acknowledges Bosch Sensortec GmbH for the right to use the schematic drawings of an accelerometer and gyroscope (see Fig. 2.1 on page 9 and Fig. 2.4 on page 12).

¹<https://www.deepl.com/en/write>

²<https://journals.ieeeauthorcenter.ieee.org/choose-a-publishing-agreement/avoid-infringement-upon-ieee-copyright>

Declaration of Authorship

I hereby declare that I have written this thesis independently and entirely on my own and that I have used only the sources and aids indicated. All statements from other writings, whether directly quoted or paraphrased, have been identified and are denoted as such. This thesis has not previously been submitted for examination in the same or a similar version.

Reutlingen, December 19, 2025

Alexandra Küster