DISSERTATION

Insight in Analog Systems: A Semi-Symbolic Approach using enhanced Deviation Traceability

Submitted at the Faculty of Electrical Engineering and Information Technology,
TU Wien in partial fulfillment of the requirements for the degree of
Doktor der technischen Wissenschaften (equals Ph.D.)

under supervision of

Univ.-Prof. Dr. Christoph Grimm
Fachbereich Informatik - Entwicklung Cyber-Physikalischer Systeme
Technische Unversität Kaiserslautern

and

Prof. Dr.-Ing. Lars Hedrich
Institut für Informatik - Professur für Entwurfsmethodik
Johann Wolfgang Goethe Universität Frankfurt

and

Apl. Prof. Dr.-Ing. habil. Helmut Gräb
Fakultät für Elektrotechnik und Informationstechnik
- Lehrstuhl für Entwurfsautomatisierung
Technische Universität München

by

Dipl.-Ing. Michael Rathmair
Matr.Nr. 0426124
Veronikagasse 34/4, 1170 Vienna

Date of Defense: 23.03.2018

Abstract

Technological innovations in the field of embedded systems are pushing the development of devices even further. The achieved performance is significantly influenced by inner systems characteristics such as robustness, accuracy, and reliability. These characteristics are potentially affected by uncertainties which are represented by unpredictable, unspecified deviations of parameters.

The goal of this thesis is to consider such parameter deviations during model-driven system development processes. Besides classical approaches, where numerical simulations are executed multiple times, I focus in semi-symbolic methods based on affine Arithmetic forms. They enable a new spectrum of analysis features which evaluate the propagation of deviations through functional blocks, give a more precise understanding of deviation impacts, and enable efficient identification of potential optimization. The development of such new analysis techniques, consequencing the mentioned benefits for system development, is the primary scientific contribution of this thesis. To achieve this, a set of new concepts needed to be developed, and were implemented as an open source software framework.

Finally, I apply the developed concepts on demonstration applications. These are defined on different levels of abstraction, ranging from deviant models described through high-level behavioral specifications to the modeling of parameter variations in semiconductor devices. The presented examples underline the versatile use of the developed software framework, and significant enhancements of deviation tracing features.

The results of this thesis are meant to motivate designers to integrate semi-symbolic modeling, simulation and analysis techniques in the system development design flow. Future applications make the use of semi-symbolic modeling, simulation, and analysis absolutely essential to handle the rising system complexity and increasing number of uncertain system parameters.

Kurzfassung

Technologische Innovationen im Bereich von eingebetteten Computer-Systemen treiben den Entwicklungsstand von elektronischen Geräten immer weiter voran. Die Performance eines Systems, welche unter anderem durch innere Systemeigenschaften wie Robustheit, Genauigkeit und Zuverlässigkeit definiert ist, kann dabei durch Unsicherheiten wesentlich vermindert werden. Unsicherheiten wie zum Beispiel Schwankungen in Versorgungsspannungen, Toleranzen in Herstellungsprozessen von Bauelementen und Temperaturveränderungen, resultieren in unvorhersehbare, unspezifizierte Schwankungen in Systemparametern.

In dieser Arbeit werden Parameterschwankungen bereits in der modellgetriebenen Entwurfsphase eines Systems berücksichtigt. Neben den existierenden klassischen Ansätzen, bei denen numerische Simulationen mehrfach ausgeführt werden, konzentriere ich mich auf semi-symbolische Methoden basierend auf Affiner Arithmetik. Mittels semi-symbolischer Methoden wird das Verfolgen von Abweichungen und somit deren Fortpflanzung über Funktionsblöcke in Simulationsprozessen abgebildet. Dies eröffnet ein völlig neues, erweitertes Spektrum an Analysemöglichkeiten, welche Entwickler helfen Parameterschwankungen und deren Auswirkungen bzw. Effekte im System besser zu verstehen, und somit gezielt Optimierungspotentiale aufzuzeigen. Die Entwicklung von Analyseprozessen ist der primäre wissenschaftliche Beitrag dieser Arbeit. Um dies zu erreichen, musste eine Reihe neuer Konzepte erstellt werden, welche anschließend als Open-Source-Software-Framework implementiert wurden.

Schließlich zeige ich die praktische Anwendbarkeit der entwickelten Methoden anhand von demonstrativen Beispielen. Diese sind dabei auf verschiedensten Abstraktionsebenen, von high-level Verhaltensbeschreibungen bis zur Modellierung von Parametervariationen in Halbleiterstrukturen, definiert. Die Beispiele unterstreichen die vielseitige Verwendung des entwickelten Software-Frameworks, sowie die Vorteile der erwähnten Verfolgbarkeit von Unsicherheiten im Systemmodell.

Die Ergebnisse dieser Arbeit sollen Entwickler zur Integration von semi-symbolische Modellierungs-, Simulations- und Analysetechniken im System Design Flow motivieren. Für zukünftige Anwendungen werden derartige Modellierungs-, Simulations- und Analyseprozesse absolut notwendig, um den Anstieg an Systemkomplexität und die zunehmende Anzahl unsicherer Systemparametern zu bewältigen.

Preface

This Ph.D. thesis is about enhancements in simulation and analysis processes for electronic circuits and systems. Uncertainty represented as parameter deviations are considered during design time processes such as modeling, simulation, and analysis. Especially new analysis features arise, which enables designers to understand a system better and enhance the insight. This additional knowledge is subsequently used for optimization and refinement of the design.

The rest of this thesis is organized as follows:

Chapter 1 introduces into the topic of range based system simulation, highlights current trends and applications. The increasing functional density and simultaneously decreasing silicon circuit structures enhances the impact of uncertainties. In this chapter, a motivation is formulated for research on enhanced range based methods for modeling, simulation and analysis.

In Chapter 2 I discuss state of the art and related publications including approaches and trends of range based system simulation, modeling, and analysis processes.

Chapter 3 describes the simulation framework implemented for this work in general. Approaches for representing uncertainties and enhanced features for their traceability are described in detail.

Chapter 4 is about extended analysis and verification features. The described processes highlight critical structures and operation periods. Results enhance the system insight and identify potentially critical deviation effects.

Chapter 5 evaluates the proposed simulation and analysis methods in demonstration examples. Results illustrate the proposed enhancements as well as efficiency and performance of the implemented framework.

Chapter 6 concludes the thesis by recapitulating main results and highlighting the main contribution to the research field. Finally, future research questions and possible expansions of the realized simulation and analysis framework are identified.

Acknowledgements

I would like to acknowledge and thank all people that encouraged and supported me during my Ph.D. time. My special thanks go to my supervisors and reviewers Prof. Christoph Grimm, Prof. Lars Hedrich, and Prof. Helmut Gräb. Thank you also to Prof. Axel Jantsch and Prof. Hermann Kaindl from the Institute of Computer Technology for their help and support in Vienna.

I am very grateful to my colleagues and friends Florian Schupfer, Carna Radojicic, Friedrich Bauer, Christoph Luckeneder, and many more people from the Institute of Computer Technology who gave helpful suggestions and valuable discussion through these intense and exciting years of scientific work and thesis writing.

Last but most important I like to thank my parents, family and close friends who made all that possible. They offered never-ending motivation and unconditional support during all the time.

Thank you!

Table of Contents

Abbreviations

AA	Affine Arithmetic
AADD	Affine Arithmetic Decision Diagram
AAFA	Affine Arithmethic assertion
AAF	Affine Arithmetic form
ABV	Assertion-based verification
ACC	Adaptive Cruise Control
ADC	Analog to digital converter
ADT	Abstract data type
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
ASK	Amplitude-Shift Keying
AST	Abstract syntax tree
BCD	Bipolar-CMOS-DMOS
BER	Bit Error Rate
BIST	Built-In Self-Test
BSIM	Berkeley Short-channel Insulated Gate FET Model
CC	Cruise Control
CMOS	Complementary Metal-Oxide-Semiconductor
CPS	Cyber-physical system
CTL	Computation Tree Logic
DC	Distance Control
dll	Dynamic-Link Library
DoE	Design of Experiments
DSP	Digital Signal Processor
EMC	Electromagnetc Compability
ESL	Electronic System Level
FET	Field effect transistor
FFT	Fast Fourier Transformation
FIR	finite impulse response
FPGA	Field Programmable Gate Array
FSM	Finite state machine
GCC	GNU Compiler Collection
HDL	Hardware description language
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IIR	Infinite impulse response
IoT	Internet of Things
IP	Interlectual property
LGPL	Lesser General Public License
LoA	Level of Abstraction

LSB	Least Significant Bit
LTL	Linear-Time Temporal Logic
MC 8	Model Computer 8
MC	Monte Carlo
MoC	Model of Computation
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MOS	Metal Oxide Semiconductor
MSB	Most Significant Bit
NRE	Non-recurring Engineering
NTBI	Negative Bias Temperature Instability
OAT	One-at-a-Time
pdf	Probability density function
PLC	Power-line communication
PL	Power line
PPTL	Probabilistic Propositional Temporal Logic
PSL	Property specification language
PTL	Propositional Temporal Logic
PVT	Process, Voltage and Temperature
RF	Radio Frequency
SA	Sensitivity Analysis
SDR	Signal to Deviation Ratio
SESYD	Semi-Symbolic System Discovery
SNR	Signal to Noise Ratio
SoC	System on Chip
SPICE	Simulation Program with Integrated Circuit Emphasis
STL	Standard Template Library
TDF	Timed Data Flow
3PIP	Third party intellectual property
ULP	Ultra Low Power
USB	Universal Serial Bus
VHDL	Very High Speed Integrated Circuit Hardware Description Language
XAAF	Extended Affine Arithmetic form

1 Introduction

Today, "Making Things Smarter" is definitely one of the biggest trends in the development and design of new electronic devices. Modern society demands a wide variety of new applications and services. This has the consequence that digital electronic technology is significantly driven by "smart" device characteristics. For manufacturers, new opportunities in application areas like consumer electronics, automotive applications, communication infrastructure and industrial applications may arise. Smart electronics are integrated everywhere from household appliances to industrial production plants. Devices get conjoint to the so-called Internet of Things (IoT), where millions of nodes are linked to a computer cloud network [KG13]. Besides local computational functionalities, IoT nodes are designed to satisfy a collaborative task. However, but what is it that makes a device "smart"? - In the following paragraphs, I want to clarify this question by discussing some selected technological drivers.

Communication: IoT applications mainly rely on the assumption of a pervasive service infrastructure providing a communication channel. Application data is continuously shared between nodes resulting in a steadily rising volume of traffic. For example, the enterprise communication bandwidth required by smart mobile devices is continually growing by a factor of three almost every two years [Qui13]. The implemented communication channels have to be flexible and use a wide variety of different physical media (wireless, optical fiber, copper, etc.). As a consequence, data transfer parameters (delay, errors, bandwidth, etc.) highly depend on the localization of the device and its surrounding environment.

Computing power: As already mentioned the IoT is a network of embedded processor nodes acting as data sources (sensors), sinks (actuators) as well as storage, routing and digital computation elements [KG13, ABC+11]. Efficient execution of the realized software functionalities also requires enhanced computational hardware-power. Mainly, standard processors are integrated into complex multi/many-core system architectures. In 2017 ARM represents the leading architecture for processors, and according to [Cou17] it's an absolute must for a foundry to have this architecture in the product portfolio. From a full platform perspective, manufacturers aim to provide highly reusable (static) application platforms to cover various types of IoT applications. Thus, system design not necessarily enables optimal hardware utilization. In terms of software behavior, this may result in varying computational delays, performance, accuracy, robustness, etc.

Customization: Early devices based on mechanical or electro-mechanical components behave more or less uniformly over time (if we set aside the rudimentary configuration possibilities). For

modern embedded IoT systems potential customization and reconfiguration options are playing an increasingly important role (e. g. situation-aware behavior). This has a further significant impact on the integration of standard processor units (e. g. ARM technology). Through software updates of embedded processors, smart devices may be customized to be used in an entirely new way [KG13]. In architectures where functions are realized by digital or analog customized ASIC (Application Specific Integrated Circuit) circuitry (e. g. audio/video decoding, data encryption, etc.) customization is more or less applied at the design level. In order to bring products to the market as fast as possible, designers are increasingly reusing components from previous projects and integrating external third party intellectual property cores (3PIP). This design practice, however, entails a higher risk of uncertainty effects, as the behavior of IP cores may vary depending on the target platform and technology.

Energy efficiency and mobile devices: A further driving force for smart mobile devices is efficient management of supply energy. If power is provided by battery packs or energy harvesting units, a minimized average consumption is a strict requirement [KG13]. Solutions for achieving this goal may be architectural optimization of components (low power technologies) or behavioral modification of hardware functionalities (voltage- and frequency scaling, etc.). In a technological perspective in November 2017 Samsung proposed a 40% lower power consumption achieved through a change from 14 nm to 10 nm FinFET technology [7]. These trends result in varying electrical circuit characteristic which may also influence the performance, robustness or accuracy of a full system.

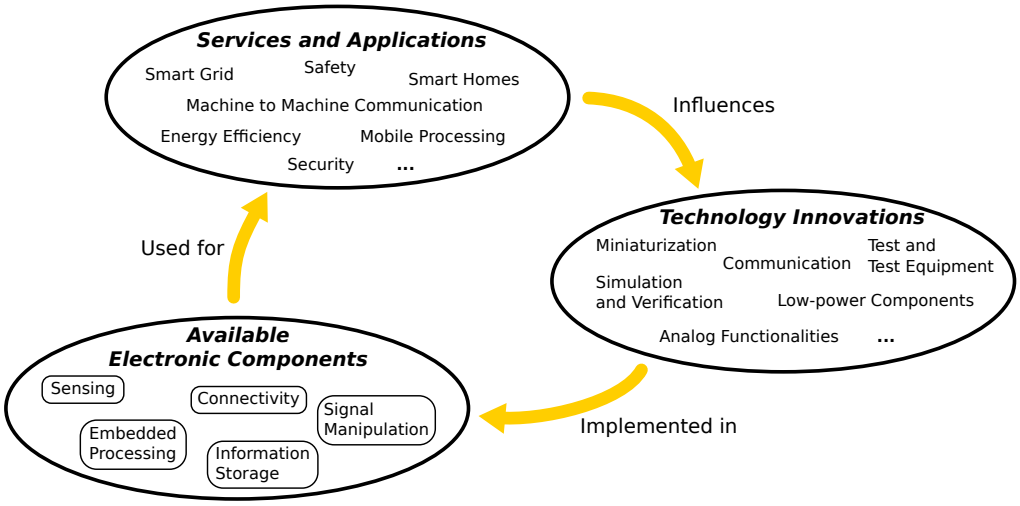


Figure 1.1: Internet of Things technology circle indicating the interaction of innovations, available components and services. [KG13]

Figure 1.1 illustrates the three parts of the IoT technology puzzle [KG13]. New services and applications required by customers motivate scientists to advance their research towards the latest technological trends. Innovations are adapted for being integrated into industrial manufacturing processes. As a result, new components implied in application areas like sensing, embedded processing, connectivity, information storage, signal manipulation, etc. are integrated into "smart" devices. Such new hardware modules in combination with appropriate software applications achieve the smart functionality aimed for [KG13]. Each cycle of this sequence takes electronic components and devices to a "smarter" level within the collaborative architecture of the IoT.

The functional density of systems will be continuously rising. In principle, the functional set of a system can be split into tasks implemented in hardware or software. From a cost effort-benefit perspective, according to [8] at a system design using 10 nm technology, approximately 62.5 % ($120 million) of total cost are invested in hardware- and 37.5 % ($72 million) in software development. One challenge in designing state of the art embedded systems is to deploy tasks to hard- and software implementations concerning given performance requirements. My focus is on the design of hardware components. However, complex software requires enhanced architectures of hardware resources for efficient execution (standard processors, hardware accelerators, memory interfaces, etc.). Structural feature sizes are shrinking and technologies which do not necessarily scale according to "Moore's Law" have to be taken into account [ABC+11, Cou17], [10].

For Internet of Things applications, the so-called More than Moore principle provides additional value in different ways. As a first part, I discuss an application perspective. According to [ABC+11] the following innovations on an application level following this principle and will significantly influence future chip properties:

- ***Non-digital functionalities are migrated*** from board-level into the package of a chip. Analog components are reallocated to the same silicon die as digital structures. This results in a mixed signal architecture where, on chip level, analog and digital domains are combined. Due to the functional conjunction, the resulting structure is called a mixed signal System on Chip (SoC). New challenges as electrical signal interactions between digital and analog domain will arise. In a technology perspective, a significant task is to enhance the compatibility of the standard CMOS (Complementary Metal-Oxide-Semiconductor) production and non-digital technologies such as analog filters, signal converters, RF (Radio Frequency) components, etc. [ABC+11].
- More than Moore also refers to an ***enhanced capability of interaction*** with the environment of the chip. This includes communication between devices (Machine to Machine communication) and with the user (user interface). Unfortunately, manufacturers use a large repertoire of communication languages (protocols) and varying communication channels. A potential solution, may be software-defined radio, where a RF front end is configured by software instructions [40].
- As already mentioned, mobile applications may be subjected to tight constraints of ***power consumption***. Improved innovations in architectural design (e. g. voltage and frequency scaling) and silicon technology are introduced in so-called Ultra Low Power (ULP) components. For example, [FDC13] promises a decrease of leakage currents and delays by 23 % and 29 % respectively for arithmetical circuits realized using FinFET instead of planar transistor technologies.

As a second part, I give an overview of current advances in silicon technology to highlight the continuous downsizing trend of silicon structures. As a consequence, components suffer from a lower reliability due to increased occurrences of so-called "soft errors" and PVT (Process, Voltage and Temperature) variability [ZMR17, DPAC17, MBDG12].

Structural sizes for analog circuits. Where structural sizes for digital hardware functionalities decrease continuously, for analog functions smaller does not necessarily mean better. Limitations are caused by double/multi pattering at finer processes which have a significant impact on signal uncertainty and sensitivity characteristics. Extra masks for manufacturing and lower supply voltages because of design constraints expense the development of analog functional blocks

[Qui13]. Thus, analog design does not scale as fast as digital structures, and mass production in 2016 is still at 28 *nm* and above [Qui13, OSY+16], [15, 4]. In addition, analog structures have to be interfaced to high-density digital functionalities, which means that the gap between digital and analog technology nodes will be widened with each new generation of digital technology [Qui13].

For ***digital functionalities*** such as processors, memory chips, interconnects, etc. downsizing is like a race. Planar FETs (Field effect transistors) are more or less limited to structural sizes of approximately 20 *nm* and above [Qui13], [9]. The problem with a planar sub-22 *nm* process is a poor short-channel electrostatic potential leading to a degraded channel characteristic and high leakage current [FDC13]. However, due to relatively low IC design costs of approximately $30 million, planar 22 *nm* is still in the race for high volume production and according to [9] TSMC announced a new energy efficient 22 *nm* bulk planar process in 2017 [9]. In general the market has shifted from planar technology to FinFET structures 16/14 *nm* and beyond [9]. By the end of 2017 Global Foundries, Intel, Samsung and TSMC were ready to ship 10 *nm* and/or 7 *nm* chips [11]. A 7 *nm* design requires an effort of $271 million and 500 man-years to bring out a mid-range SoC (System on Chip) for production [11, 8]. But according to [8] a 7 *nm* design compared to 16/14 *nm* provides a speed-up of 35 %, consumes 65 % less power and has a 3.3x density improvement. In 2018, however, the race is still not over and manufacturers plan to go for 5 *nm* and 3 *nm* so called "gate-all-arround" FETs in 2020 and 2022 [12]. Figure 1.2 shows a road map estimation (originally published in [Cou17]) for the future foundry market. Beyond 2020 this prediction is very uncertain. Unpredictable multiple effects in lithography, silicon structure, etc. and many other unknowns will pose hard challenges for semiconductor technology research [12].

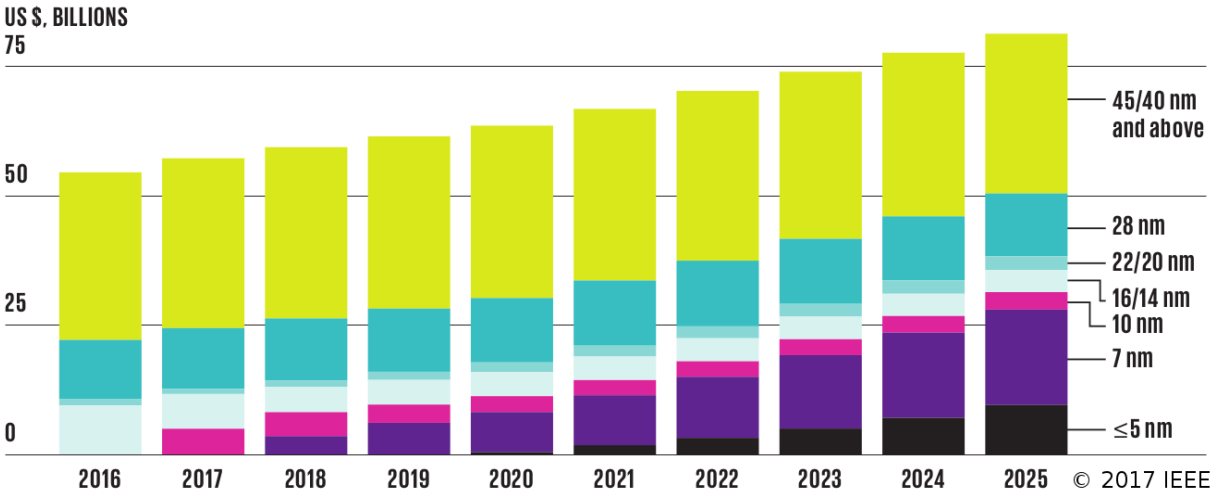


Figure 1.2: 10 *nm* will come first but business strategies of customers will force the 7 *nm* technology in the next decades [Cou17].

I would like to sum up, and elaborate different kind of uncertainty causes for IoT applications using leading-edge silicon devices. On the one hand, variability is caused by semiconductor technology itself, as explained in the previous paragraph. Today, chip makers are struggling with several problems in high-end FinFET processes. Open challenges, significantly influencing uncertainty in devices, are for example: lithography processes, unexpected contact resistances, killer defects significantly impacting the yield, PVT variations, structural inaccuracies (FIN dimensions), Negative Bias Temperature Instability (NTBI) and electromigration effects, complexity and cost of design processes, etc. [ZMR17, DPAC17, Wie17], [8, 11, 12]. These uncertainties on

silicon level are propagated from silicon level to application level because they affect the electrical characteristics of components (drain current in on and off state, gate-source voltages, delay, etc.). The final consequences are uncertain parameters and device characteristics. On the other hand, I have already mentioned variability caused by the application domain (e. g. uncertain delays in wireless communication channels, environmental temperature, misuse, etc.) Some of these application-level uncertainties are partially/fully driven by uncertainties in the silicon (e. g. computation delay influenced by FET switching delays) and vice versa (e. g. environmental over temperature of a device may significantly affect silicon material effects). Thus, there are various potential circular dependencies between high and low level uncertainty causes, resulting in a direct link from process- to performance variations [TM13]. In any case, they all affect the inner parameters of a system resulting in uncertain behavior, performance measures and robustness.

1.1 Scientific Challenges and Tasks

Scientific challenges motivating this thesis are primarily driven by the described trends in smart devices. As mentioned, consequent downsizing of silicon structures, changing environmental conditions, potential misuse of devices, etc. all enhance variability. They may significantly influence the behavior, but at least definitely impact the performance of a system. The following subsections motivate this work propose new methodologies in modeling, simulation and especially analysis to take into account potential variations already in the design phase.

1.1.1 Motivation

A major goal for state of the art mixed signal systems design is effective verification during all development phases. Bugs, design errors, specification mismatches, undocumented behavior, etc. should be detected as early as possible. Redesigns or redefinition loops during the design cycle of a product cause increased costs and delay market launch. The main objective of this thesis is not classical verification of functional requirements. The focus is on the verification and analysis of system characteristics caused by the variability of parameters. As already stated in the introduction (Section 1), modern IoT systems face various uncertain parameters which may significantly influence the behavior and performance of the system. For the efficient design of such systems besides functional verification, additional consideration and analysis of parameter deviations during development may result in significant benefit.

The following detailed discussion about the motivation for this thesis (and also the main part) is sectioned into three pillars:

- Models for systems featuring deviant inputs and parameters.
- Efficient simulation techniques and management of simulation results.
- Enhanced analysis of the simulation results, which increases insight. The evaluation of deviation related system characteristics is embedded into a modeling, simulation and analysis flow.

For design verification processes, a projected design is represented by an appropriate model (model-driven system design). The abstraction level of models may range from a coarse block-oriented system view to a fully refined description of the hardware structures [Dre10, Chapter

Introduction]. For future designs which imply large functional diversification one system models will not be sufficient [ABC+11]. Therefore, a set of models are constructed. Potential problems caused by consistency, version management etc. are not part of this work. Verification models for uncertainty analysis so-called range based models may be part of this model set. Models for the analysis of parameter deviations in principle have similar requirements as models for functional verification (e. g. efficiency, complexity, accuracy, expressiveness, etc.) [RSRG12b, RSRG12a].

In addition to modeling adapted simulation techniques have to be developed. There are already some published range based simulation techniques based on different types of models. Especially semi-symbolic approaches derived from computer graphic applications reduce the computational effort dramatically compared to classical multi-run techniques [Rad16]. Thus, the main part of this thesis deals with semi-symbolic modeling and applied enhanced range based simulation techniques. Based on the available frameworks SystemC [23] and SystemC AMS [38] we already have an effective and approved simulation environment for mixed signal systems. Due to open source licensing of these tools the presented modeling simulation and analysis procedures are implemented as an extension framework for SystemC/SystemC AMS.

Commonly used numerical simulation techniques are adapted and extended by arithmetic functionalities enabling the handling of semi-symbolic representation of deviation. This creates extra value as new possibilities for analysis enhancing the system insight. Insight is defined as the effective monitoring and tracing of internal signals and their corresponding effects (gaining, attenuation, correlation, etc.). Sophisticated consideration of additionally achieved knowledge may improve the design processes of future electronic systems. As proposed the robustness of a system represented as dedicated signal sensitivities against their deviation causes may impact (and eventually damage) the full behavior. Hence, the third, and one of the main motivation pillar of this thesis, is to enable enhanced analysis of inner system characteristics. This has the goal of gaining performance and decreasing the behavioral impact of deviant system inputs.

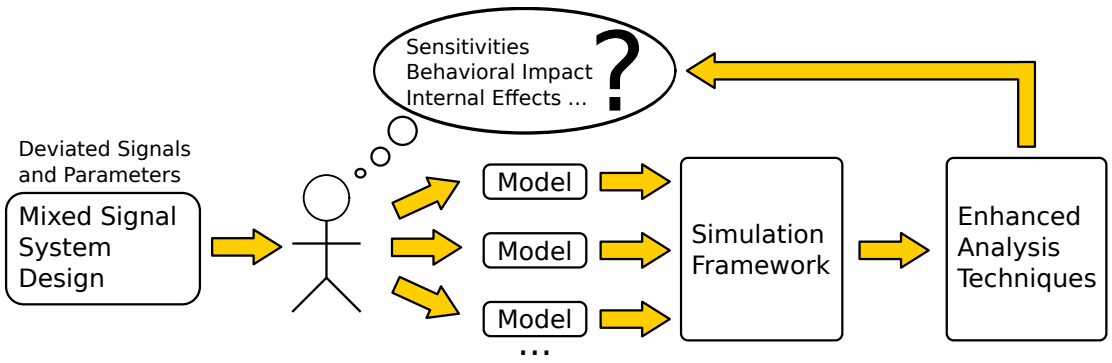


Figure 1.3: Analysis run illustrating a human verification engineer interested in enhanced insight into a mixed signal system design with deviated inputs. In this thesis selected semi-symbolic models representing such deviations are generated and handed over to a SystemC AMS simulation environment. Subsequent analysis enables optimization processes to adjust signal sensitives, monitor internal deviation effects, etc.

Figure 1.3 depicts an abstracted system simulation and analysis sequence. A human verification engineer intends to verify and evaluate potential optimizations of a mixed-signal system design. A central condition of this verification process is that input signals and parameters of the system

deviate from the ideal value. Verification and analysis goals deal with questions regarding inner characteristics of the system under test (e. g. signal sensitivities with respect to input deviations, behavioral correctness inside the range of tolerances, internal uncertainty gaining or attenuation effects, etc.) Then a set of models is transformed or deduced from the initial (exact) design. These models are simulated under consideration of the defined uncertainties. Finally, range based analysis techniques are applied to the resulting simulation outputs. The increased internal knowledge and insight close the loop by refinement and optimization of the original design. As a result, range based analysis may raise the robustness of the design, guarantee correct behavior under specified signal tolerances (safety properties) and may guide design optimization.

1.1.2 Problem Description Based on an Example

In this subsection, I want to give a short demonstration example illustrating the importance of considering uncertainties and their associated impact during the design time.

In 2016 a company selling electronic cash register systems asked for help at our institute. Their system is implemented on a USB flash drive, and they had run into increasingly reported problems that flash drives dying unexpectedly after a few weeks of operation. The error seemed to occur randomly and did not affect all drives. Putting the situation in the terminology of this thesis, there was an unspecified, unacceptable variability in lifetime and reliability significantly impacting the performance of the product. Time pressure before market launch forced designers to focus their effort on getting the system running, with the consequences that test and verification procedures get a shortened time slot within the project plan. System analysis of potential uncertainty effects and their impact was not carried out at all.

The errors reported could have various causes and engineers involved in the project had different educated guesses and starting points for locating the problem. One possible approach I proposed the project team is so-called inverse propagation variability modeling (see Figure 1.4 and Section 2.2). Based on lab experiments, a functional model (for hard- and software) is derived. Configuration and parameters are evaluated from measurements (see Figure 1.3). The goal of the process is to increase the insight regarding possible uncertainties and thus locate the cause(s) of the variability in the lifetime (see Figure 1.4). The main advantage of a model-based analysis, instead of exhaustive lab testing, is that "what if" questions can be answered with less effort (e. g. time saving for a high number of virtual versus real flash memory read/write cycles). In addition, by explicitly modeling uncertainty causes their impact and propagation can be traced during virtual test processes, while actual lab testing obfuscate such effects.

The modeling process itself follows a top-down approach as illustrated in Figure 1.4. In the first iteration behavioral models are highly abstract, but cover the full application context including selected variabilities of environmental properties, user interaction parameters, software behavior, analog circuit properties, etc. (e. g. environment temperature, unspecified frequent plug in/off of the flash drive, variable hotspot times regrading read/write executions, etc.) In the next iteration the model is refined to a component/IC/module level of abstraction. The internal chips of the USB flash drive [41], software components, as well as analog circuitry, are described using a block-oriented functional model. Typical variabilities within this model are component parameter drift, supply voltage parameters, IC temperature, etc. (e. g. power supply peak at plug in/off

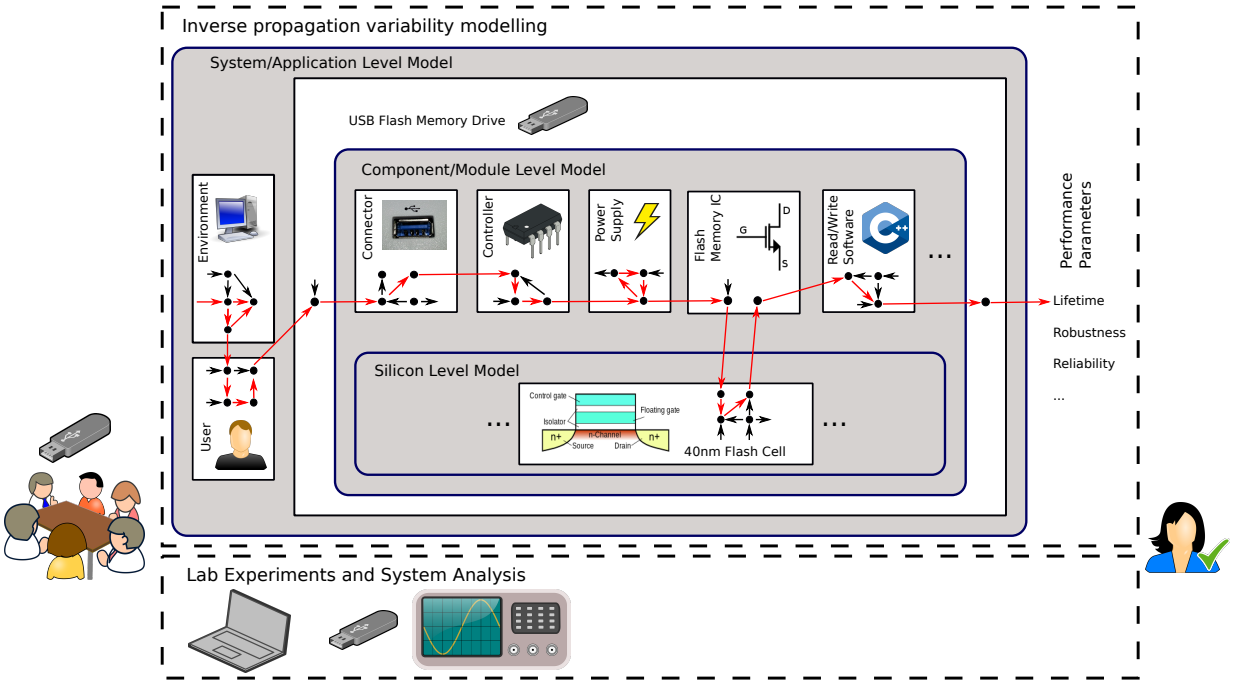


Figure 1.4: System analysis example of an USB flash memory application. For inverse propagation variability modeling I define different levels of abstraction. An essential tool for variability analysis is comprehensive impact tracing as illustrated by the red marked path.

operations, variabilities and distortion of the supply voltage, unspecified variabilities in the heat map of ICs, offset drift, gain variability and nonlinearities in analog circuit components, etc.) The final refinement step, having the lowest level of abstraction is a model representing behavior in silicon level. In the present example, this could be a model of the flash memory cells (e. g. [KH05, PSN04, TUS16]). Variabilities which may be considered at this level include: cell capacity parameters, aging and breakdown characteristics, read/write delay, soft-errors such as Negative Bias Temperature Instability (NTBI), etc. As a result, this approach allows a comprehensive verification and analysis of various uncertainties in the full application. However, the effort for refining a specific submodel to a lower abstraction level is high. The spent workforce directly correlated with a significant increase of the systems insight and knowledge regarding variations. In a modeling perspective, it is essential that the selected methodology allow detailed tracing of variabilities and their associated effects as well as chain-linking over specified abstraction levels. In Figure 1.4 this is highlighted by the red path which traces the propagation of a variability caused in the environment domain through the various models to its impact resulting in reduced hardware liftetime. One major goal of the described approach is an exhaustive cause and effect analysis.

Regarding the faulty USB flash drives, the reduction of lifetime was ultimately found to have been caused by a combination of flash memory aging effects and poor application design (high number of read/write cycles). However, these causes were discovered by best practice lab experiments without any resulting knowledge of a variability cause-effect characteristic. Detailed consideration and enhanced effort in modeling/simulation and analysis, also considering variabilities already during the design phase, however, would have made this application more robust.

1.1.3 The Focus of this Work

As mentioned, the work is about the analysis and evaluation of inner system characteristics which are caused by parameter deviations. Besides classical approaches (multi-run, Interval Arithmetic), a semi-symbolic modeling, simulation, and analysis flow is presented. The focus of this thesis is enhanced analysis techniques enabled by deviation traceability features of the semi-symbolic approach (e. g. cause-consequence analysis, correlation tracking, temporal and structural tracing, etc.). Traceability allows detailed impact tracking of selected deviations from their causes to their consequences in the system's behavior. For the implementation and management of semi-symbolic traceability features, already published frameworks have to be extended. This thesis presents an academic approach but discusses in detail why a semi-symbolic methodology with its specific deviation traceability character may improve the future industrial design and verification tools.

1.1.4 Hypothesis

The research effort invested for this work concentrates on methodological approaches and processes for increasing analysis capabilities of mixed signal systems during the design phase. The described techniques enable the integration of range values into a system modeling, simulation, analysis and verification flow. For this thesis I have formulated the following three hypothesis statements:

a) For enhanced behavioral analysis HDL models defined in SystemC AMS, VHDL, Verilog, etc. have to be extended to ***accept range inputs***. There is a need for augmented creation of models for different types of analysis and verification tasks to examine uncertainty. Especially semi-symbolic modeling approaches demonstrate their advantages in expressiveness when the number of uncertainties rises and chained ***deviation effects receive an increased impact*** onto the designed system behavior.

b) For simulation of enhanced models, associated computation concepts need to be integrated into a software framework usable for C++-based simulation environments. An ***extendable object-oriented framework architecture*** provides detailed information about the ***propagation of deviation causes through the system structure*** (tracing information).

c) Evaluation of deviation tracing information during a simulation run enables ***new analysis methods*** which may entail a significant increase in system insight. So-called ***inner systems characteristics***, such as value bounds, sensitivities, cause-and-effect chains, stability issues, etc., can be analyzed individually for each included component. Furthermore, evaluated analysis results may guide subsequent optimization processes and thus integrate range based modeling, simulation and analysis techniques into a ***comprehensive enhanced design-time verification flow***.

1.2 Contribution to the Research Field

My contributions to the research field started in 2012, but Semi-symbolic simulation using Affine Arithmetic forms (AAFs) in general had already been studied prior to that (see [SKG+10,

GHW04, BGG+09] etc.). This thesis is partially based on the following selected publications:

RATHMAIR, M. ; SCHUPFER, F. ; RADOJICIC, C.; GRIMM, C.: Extended framework for system simulation with Affine Arithmetic. In: *Forum on Specification and Design Languages (FDL), 2012* - [RSRG12b]
An object-based implementation of AAF data types in C++ is presented. Extended functionalities such as deviation symbol management, tracing of uncertainties and application-specific approximation algorithms are introduced.

RADOJICIC, C. ; SCHUPFER, F. ; RATHMAIR, M.; GRIMM, C.: Assertion-based verification of signal processing systems with Affine Arithmetic. In: *Forum on Specification and Design Languages (FDL), 2012* - [RSRG12a]
Introduces assertion-based system verification using semi-symbolic system simulation methods. Functions for verification applications such as temporal operators, relation operations, analog frequency operators, etc. are defined for Affine Arithmetic forms.

RATHMAIR, M. ; SCHUPFER, F. ; GRIMM, C.; RADOJICIC, C.: Simulationsgestützte Analyse der inneren Eigenschaften von Mixed-Signal Systemen (german). In: *Proceedings of 16. Workshop - Analog Circuits 2014* - [RSGR14]
In this publication sensitivity-, correlation-, and interaction effect analysis procedures for AAF based system simulations are introduced. The main outcome of this paper is that semi-symbolic analysis and the subsequent knowledge of inner properties of a system may improve design and optimization procedures.

RATHMAIR, M.: Range Based Analysis of Inner Systems Characteristics. In: *Design, Automation and Test in Europe (DATE) 2015, EDAA PhD Forum* - [Rat15]
In the submitted extended abstract and the poster for the PhD Forum, I present the full context of this thesis. As a conclusion, I highlight the motivation of increased system insight enabled by an extended simulation framework including deviation tracing features and associated analysis processes.

RATHMAIR, M. ; SCHUPFER, F.: Dealing with Uncertainties in Electronic Systems Simulation. In: *VSS - VIENNA young SCIENTISTS SYMPOSIUM 2015* - [RS15]
In this publication types of uncertainties, associated correlation effects and their representation using Affine Arithmetic forms are identified. This is shown using an adapted BSIM3 Transistor model which is used in an inverter stage. The impact of production uncertainty on the inverter's propagation delay is evaluated.

GRIMM, C. ; RATHMAIR, M.: Dealing with Uncertainties in Analog/Mixed-Signal Systems: Invited. In: *Proceedings of the 54th Annual Design Automation Conference 2017* - [GR17]
In the paper we describe the deviation modeling and documentation approach using AAF and extended AAF forms in detail. Besides uncertain values also uncertain events (control flow deviations) are introduced. We illustrate the proposed approach using a PLC (Power-line communication) data transmission system.

RADOJICIC, C. ; GRIMM, C. ; JANTSCH, A. ; RATHMAIR, M.: Towards Verification of Uncertain Cyber-Physical Systems: In: *Proceedings 3rd International Workshop on Symbolic and Numerical Methods for Reachability Analysis - 2017* - [RGJR17]
In this paper, we focus on modeling uncertain cyber-physical systems and how these models can be verified. Therefore, we show how partially existing methods defined for different MoC (Model of Computation) are adapted and integrated into a standard system design process.

RATHMAIR, M. ; LUCKENEDER, C. ; KAINDL, H. ; RADOJICIC, C.: Semi-symbolic Simulation and Analysis of Deviation Propagation of Feature Coordination in Cyber-physical Systems. In: *Proceedings of the 51st Hawaii International Conference on System Sciences - HICSS 18* - [RL+18] - Best paper awarded
In this publication, we use the AAF approach and associated analysis features in the context of automotive feature coordination. A model of ACC (Adaptive Cruise Control) including unpredictable deviations (e. g. speed profile of the car in front) is created. Results describe the impact on the feature's performance (distance and speed control) in detail.

1.3 What this Thesis is not About

The main method in this thesis for representing uncertainties is to model them using abstract symbols. This semi-symbolic method has the advantage that signal ranges of outputs can be computed within one simulation step. Multi-run approaches, which require various simulation runs, as well as classical interval arithmetic are implemented in the proposed framework but not studied in detail. Such methods are state of the art and essential for discussion as well as for the evaluation of advantages and disadvantages of semi-symbolic approaches.

Analysis methods are based on simulation results evaluated by semi-symbolic simulation. They are applied during the design phase of a system. Thus, models for verification and analysis procedures are pre-synthesized and/or pre-compiled functional representations of a piece hard-/software. This thesis is not about testing processes or the evaluation of adequate test patterns for embedded systems. Proposed analysis and verification methods are applied (but not limited) to abstract models.

Approaches for system refinement and optimization processes (loops in the design cycle) as focused in [Sch13] are not part of this thesis. The concepts proposed aim primarily to expand the inner knowledge and increase the insight into system concerning the impact of parameter uncertainties.

1.4 Discussion of Expected Results

Expected Results of this thesis are enhancements in range based system analysis processes mainly enabled by semi-symbolic modeling and simulation. Consideration of parameter deviations and uncertainties in circuits and systems may become a significant co-existing technique to state of the art functional circuit verification. The focused strength of a semi-symbolic verification approach lies in its ability to trace the propagation of deviations from their cause to their consequences. This should significantly enhance the insight and provide added value for subsequent debugging and optimization procedures. This work is expected to be a further contribution towards the future integration of semi-symbolic approaches into an industrial design flow.

2 State of the Art and Background

This chapter gives an overview and background of state of the art approaches for modeling, simulation and analysis techniques considering uncertainty. Figure 2.1 sketches some selected fields and methods described in the following subsections.

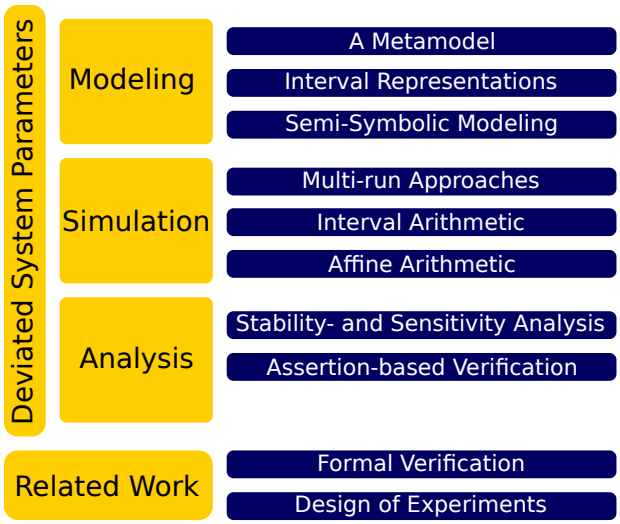


Figure 2.1: State of the art approaches and related work in the field of system-modelling, -simulation and -analysis considering deviated parameters.

2.1 Definitions and Terminology of Uncertainty

This preceding section is a short survey of frequently/commonly used definitions and terms such as "system", "uncertainty", "deviation" etc. Some of them may slightly vary in other research fields. However, for the rest of this thesis technical terms and phrases are used as defined in the following.

A System in the context of this work

The term "system" is widely used in various fields of technical applications ranging from meteorology via mechanical and electrical engineering to biology. The Oxford Dictionary [34] holds a general but for this work pretty well fitting definition of: "A set of things working together as parts of a mechanism or an interconnecting network" [34].

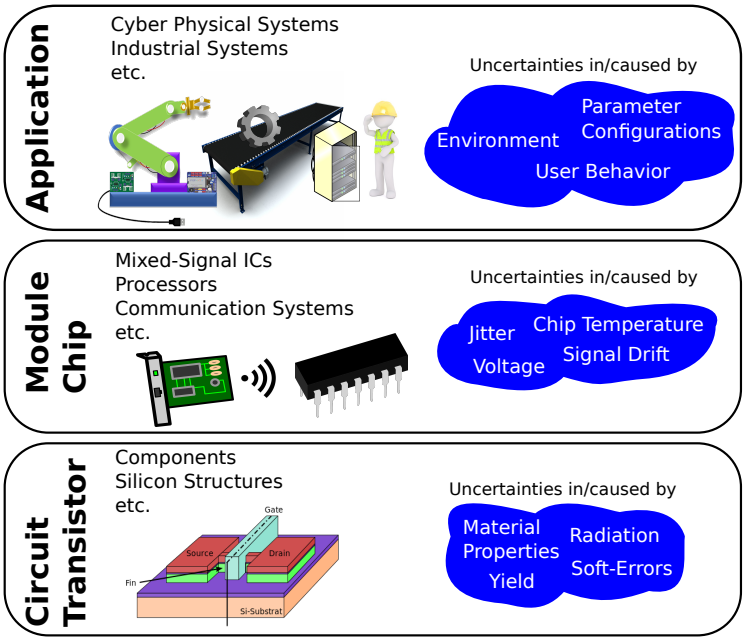


Figure 2.2: Layered definition of the term "system" used for this work. An application includes chips and modules based on transistor circuits.

As shown in Figure 2.2 for this thesis a system, in general, may be assigned to one of the illustrated layers. The layers also identify a level of abstraction for uncertain parameters and associated deviation effects.

CPS (cyber-physical system) Applications are included in the highest layer of this stack. They are highly abstracted descriptions of functional behavior including components as computers (with software), sensors, communication equipment, analog devices etc. in an ESL (Electronic System Level) style. The next lower level is defined as chip, module, IC level. Main components of this level are refined descriptions of hardware/software functions on a block level (e. g. multiplexer, adder, memory registers, analog filters, mixers, etc.). System on Chip (SoC) and System in Package are components, where several sub-components (analog or digital) are integrated on/in the same chip/package, are also assigned to this abstraction level. The most refined systems, and thus the lowest layer in the presented stack, are netlists/circuits and transistors.

For the rest of this work the term "system" is used as described in this presented layered definition. Default, the full thesis is focused on the middle layer where a system includes a bunch of analog/digital modules described by appropriate models.

Besides the definition of a system itself some associated terms independent of its level are described in the following [PKB07]:
Input: Signals which are located at the border of the defined system representation that drives or force the associated internal state to respond. Inputs may change each simulated time-step.
Parameter: Parameters are defined as a super set including system inputs. In general, parameters represent a numerical variable which can be constant or configurable. Parameter may have some variations (e. g. caused by changing environmental conditions), which are basis of enhanced system uncertainty analyses.

Output: Output variables/signals are of particular interest at any kind of system analysis. They characterize how the system presents its reactions based on the given inputs and/or the system's history. Inputs and Outputs are used for interaction with other systems.
System State: A set of variables models the current internal state of a system. These variables may be modified at each time-step during the simulation process (driven by input events). The full state of the system is defined by the internal state in combination with the variables representing in- and output signals.

Uncertainty

Within this work, I will focus on uncertainties occurring in technical (engineering) applications. However, uncertainties face in many situations in our everyday life, ranging from the departure time of a bus taking for the way to work, to biological systems, where uncertainty in the DNA is essential for new populations. In a generalized and intuitive perspective, we encounter uncertainties as tolerances, which define that a value is located unpredictable inside a specified confidence interval [Dro09]. In a technical application where data is evaluated from experiments, measurements or computed from a model a value holds ubiquitous uncertainty [Wol09]. Also, it is worth to mention that storing data digitally adds additional inaccuracies. Typically, IEEE 754 standardized floating-point numbers are used which round an answer per definition if it is inexact. There is an exciting research field for replacing floats by so-called "posits" which potentially overcome disadvantages of standard float types [GY17]. Designing a system under given uncertainties addresses additional requirements. Uncertainty is strictly speaking, unavoidable, and significantly important in highly optimized, robust and flexible systems [Neu04]. This forces designers to treat uncertainties in the full development life-cycle, especially during design time.

The mentioned tolerance window is located around a single numerical best estimate value, where calculated/measured samples may be clustered. This best estimate value may be obtained from numerical or analytic models, and represents a single value indicating an exact numerical solution. But the best estimate value is still an estimation of the reality because modeling a system is, in any case, a simplification and hence adds unavoidable uncertainty (see Section 2.2). For the rest of the work, the context are ***uncertain computer models*** and ***uncertain data***, caused by parameter variations.

According to [WHR+03] the nature of uncertainty is that it is never known whether a specific sampled value is greater or less than the "exact", best estimate value [WHR+03]. In fact, the quality of a scientific computer model may be characterized by the difference between computed results and measured output values derived from the corresponding experiments on a real system implementation [Cel91]. Within this work models and associated simulation methods are proposed where besides a numerical exact value, bounds for tolerances are evaluated under given deviated inputs/parameters.

Correlated uncertainty

According to [Dro09] correlated uncertainty is defined by the existence of a dependency between uncertain components. Otherwise, uncertainties are uncorrelated. In a further perspective, uncertainty consequences are affected by multiple causes which depend on each other (e. g. the drift of two clock signals based on the frequency-temperature drift of a single oscillator crystal). Thus, correlated system uncertainties are partially inferred from a common unpredictable effect (physically, component variability, semiconductor material properties etc.) [Dro09].

There are different types of correlations: Direct correlation, where an uncertain event directly affects multiple values, and indirect correlations, where effects may be chained and consequences

are not directly in touch. Reciprocal correlation can be found where an increasing characteristic in uncertainty A consequences a decreasing effect in uncertainty B [Dro09].

For a measure of correlation, the so-called Pearson product moment correlation coefficient r_{xy} is defined. It is a normalized and dimensionless value between -1 and 1. Bounds of this range represent total correlation (reciprocal or direct), or uncorrelated for $r_{xy} = 0$. Further, details for the product-moment correlation coefficient can be found in [Dro09, p.117] and [SCS+00, p.23].

According to [WHR+03] a system uncertainty or a common component of multiple uncertainties are called random or statistical if they are totally uncorrelated. A probability distribution function of randomly sampled values located around the best estimate indicates the characteristic for covering a specific random uncertainty effect [WHR+03]. An outstanding example of totally uncorrelated (normally distributed) uncertainty effects applied on single data points are defined by observing effects of radioactive materials.

Variability
For this thesis, the term variability is defined according to [WHR+03]. Variability describes the spread of an uncertain value set. Thus, variability is defined as the maximum and minimum values uncertain data may reach. Variability is well known while uncertainty is based on lack of scientific or technical knowledge. For example, if we consider the throwing coin experiment, data variability is given by the two results which are heads or tails. If the experiment is considered randomly, the frequency of heads and tails results is entirely unpredictable denoting a so-called epistemic uncertainty [WHR+03]. For standard uncertainties where the possible value space is defined by normal distributions the variability can be determined by standard deviation or median measures.

Deviation
For this work, the term deviation is not used as a synonym for uncertainty. As described in [Dro09] and other textbooks for statistics, the sign of a deviation is known. The deviation of a dedicated sample indicated by its magnitude, the distance to the best estimate value, and the sign represents whether a sample is too small or too large [Dro09, Neu04]. Thus, the deviation is a measure of how precise the given point is to the exact/best estimate value [Dor10]. For correction purposes of a single data-point or the full model, the deviation of continuously sampled values is a significant characteristic. For example, let's assume the results of a simulation model, computing best estimate values for given test cases, consistently produces too small values compared to real measurements. Based on the deviation (magnitude and sign) between real values and computed best estimate values the simulation model may be corrected to improve its quality. Analogous to uncertainty, multiple deviations may be correlated [Dro09]. According to the previously mentioned example concerning the uncertain frequency of an oscillator, all derived clock frequencies are too small/large if the common oscillating frequency is too small/large.

Confidence interval
In general, a defined confidence interval is a range of data samples we trust with a given level of confidence [Dor10]. Thus, a defined confidence interval is not a probability that sampled data are located within this range. A defined confidence interval is located around the true/exact value a model may produce. The peak confidence of 100% trust level is given if the exact estimated value is observed. For spanning a bounded confidence interval around this peak indicating the best estimate sample a dedicated confidence interval has to be defined (e. g. 95% is often chosen for Gaussian distributed data) [Dor10].

Outliers
Outliers, also called flyers, are data points which lie at an abnormal distance to the location of

the rest of the data. In fact, it has to be defined in principle, what an abnormal characteristic is concerning the distribution of all defined data values. This mainly depends on the probability distribution of inspected/computed data related to the distance of the outlier value to the best estimate value [Dro09], [29]. The occurrence of an outlier may be mainly caused by two possibilities [Dro09]:

- The model which computes an outlier data-point is wrong. Due to unconsidered modeling circumstances (undesired input constellations, wrong model parameters, inaccurate implementation of algorithms, etc.) the computed value may face an outlier.
- The value of the data-point itself is wrong and hence indicate an abnormality (e. g. the interpretation of a produced data value is insufficient, scaling factors are erroneous, etc.).

Thus, it is advisable to consider produced outliers sufficiently, to enhance the quality of the modeling and design process or to identify errors which may rarely occur in the deployed system. Proposed options for dealing with outliers according to [Dro09] are:

- Identify the outlier values and correct them. Check the input values applied to a model carefully and debug implemented algorithms in a white-box manner. If the cause of a produced outlier is identified the model may be corrected or extended to avoid/cutoff the abnormal computation value.
- Simply exclude the outlier observation, document the occurrence and eventually accept an increased model inaccuracy.
- Select a larger variability for the data set produced by a model and include the produced outlier value to the range based system analysis. Dependent on the used uncertainty model (see Section 2.2) the best estimate value has to be corrected.
- In a design perspective, the observation of an outlier may highlight uncovered asymptotic poles of the system characteristic. Detailed inspection and analysis of the model besides the obtained range based analysis may increase the performance of the design.

However, dealing with outliers and associated analysis of abnormalities in inspected/generated data is a state of the art research topic and covered in several publications such as [YWS17], [LYX+10], [WJCK16].

2.2 Modeling of System Uncertainties

Creating computer models and virtual prototypes for the design of technical applications is a commonly used procedure for almost all technical development processes. A model defines an abstracted representation which behaves within a specified modeling context accordingly to the projected application. Abstraction of modeling refers to simplification in structure, functional coverage, behavior, etc. in predefined operational bounds. Modeled systems are embedded into a simulation environment where virtual execution is performed. Test vectors are applied, and the correct behavior of the model is evaluated. A high number, and eventually risk critical, constellations of input valuations can be applied during the design process (answer so-called what-if questions) [WHR+03].

Creating application models (at a specific level of abstraction) during design time has the following selected advantages:

- Enables high-level functional verification (formal and informal) against a given system specification.
- Performance measures can be evaluated at a very early stage in the design process.
- Fast and cheap regression testing after changes and optimization of the design.
- Analysis methods enable the evaluation of coverage measures to estimate the progress of verification and confidence before moving to lab prototypes and fabrication.

In addition to implementing planned model behavior, the consideration of uncertainties in specific components and modules is of increased interest as introduced. Unpredictable deviations in signals, parameters, configurations, behavioral descriptions, etc. are represented in this system analysis processes. According to [AAC12] two different challenges can be identified at the usage of deviation models.

Forward propagation describes a technique where uncertainty ranges are specified based on the analysis of the constructed behavioral model. Thus, variations are defined before a specific real implementation of the model is built. Analysis processes mainly verify if the system behaves according to the specified properties under assumed worst-case operation (reflected by the deviation ranges).

Inverse propagation uses experimental results from a final implemented system or a prototype implementation. Evaluated outputs are used to identify and locate uncertainties in the constructed computer model. Extended observation may also successively increase the knowledge about real system behavior, refine the modeled functionalities and evaluate optimization to maximize system performance [KO01]. By using enhanced (symbolic) representations for uncertainties, in the model potential correlations of deviation effects deduced from a real implementation can be represented.

2.2.1 A Metamodel for Uncertainties in Computer Models

[KD07] presents a conceptional mathematical description of considering uncertainties, applicable for any type of computer models. This abstract metamodel (a generalized model for modeling uncertainties) is finally used to identify and locate causes of uncertainty and of course their impact. Detailed analysis allows a categorization into location, level, and nature of deviations as discussed in [WHR+03, KD07] and [WJBK10].

At the proposed abstract description a computer model has a defined vector of inputs $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ and outputs $\mathbf{y} = (y_1, y_2, \ldots, y_M)$. Single vector elements represent signals, parameters, configurations, etc. Valuations of inputs are within strictly defined value ranges as given by the system specification. Elements of $\mathbf{x}$ take their values out of a corresponding set of basic random variables $\mathbf{X} = (X_1, X_2, \ldots X_N)$. An input sub-model $f_{\mathbf{x}}(\mathbf{X}, \mathbf{\Theta}_f)$ defines the probabilistic distribution of $\mathbf{X}$. Functions g_i, $i = 1, 2, \ldots M$ defines the relation between inputs and outputs of the model $\mathbf{y} = g_i(\mathbf{x}, \mathbf{\Theta}_g)$. The parameter $\mathbf{\Theta}_g$ denotes uncertainties of the model itself.

For forward propagation parameters $\mathbf{\Theta}_f$ and $\mathbf{\Theta}_g$ are defined during the specification phase of the system, while for inverse propagation these characteristics are deduced from experimental results.

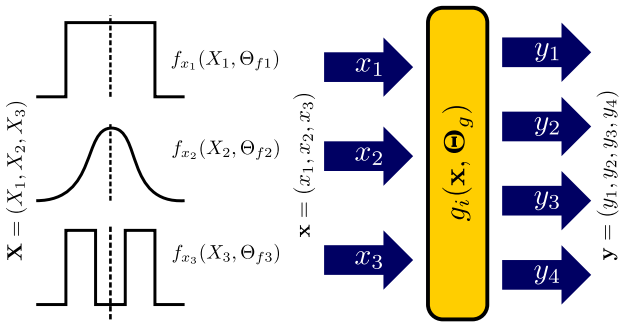


Figure 2.3: A metamodel for considering uncertainties in any type of computer models according to [KD07].

Figure 2.3 illustrates the described metamodel where three exemplary inputs $\mathbf{x} = (x_1, x_2, x_3)$ are selected by the probabilistic sub-models $f_{x_1}(X_1, \Theta_{f1}), f_{x_2}(X_2, \Theta_{f2}), f_{x_3}(X_3, \Theta_{f3})$. Model outputs $\mathbf{y} = (y_1, y_2, y_3, y_4)$ are computed under their corresponding functions g_i. Model uncertainties are covered by the parameters Θ_g. This metamodel is independent of the obtained application model and a mathematical construction for the consideration of uncertainties in any type of computer models.

2.2.2 Classification and Origin of Uncertainties

Using the definitions of the presented metamodel, uncertainties in computer models can be mapped to one of the following categories concerning its cause [KD07, AAC12]:

1. Uncertainties in the set of available random numbers $\mathbf{X}$ for the specification of input variables. These uncertainties are deduced from an inadequate design specification or evaluated from experiments on an implementation. For example, if a range of input signal values in the set $\mathbf{X}$ is not fully covered as defined in a system specification, which results in a lack of possible model input valuations.

2. Uncertainty model errors caused by the selection of a probabilistic submodel represented by the function $f_{\mathbf{X}}(\mathbf{x}, \mathbf{\Theta}_f)$. This function describes deviations in the distribution of basic variables.

3. Uncertainties in the physical in/output submodel described by the function $\mathbf{y} = g_i(\mathbf{x}, \mathbf{\Theta}_g)$. These are caused by inaccuracies in dedicated model implementations.

4. Statistical uncertainties which are caused by parameter estimations of $\mathbf{\Theta}_f$ and $\mathbf{\Theta}_g$ in the corresponding submodels. These estimation functions are directly related to the quality of available system information. For forward propagation $\mathbf{\Theta}_f$ and $\mathbf{\Theta}_g$ are defined by the system specification. They specify the model respecting that the following implementation operates correctly. Inversely, whether a real implementation is represented by a model an increased amount of observed and analyzed experimental data allow a more accurate estimation of model parameters [WHR+03].

5. There are additional uncertainties in the case that the model is evaluated by inverse propagation. Observation process for $\mathbf{\Theta}_f$ and $\mathbf{\Theta}_g$ may be highly erroneous.

6. Uncertainties in the representation of numbers and iterative calculations in the model. Computational errors are caused by limited bit widths of digital number representations and rounding errors [GY17].

As a result of the presented metamodel and the identification of uncertainty causes mainly two critical factors for deviations in computer models are faced. First, parameter and input uncertainty described as value ranges and associated probability distributions. They are either context specific input signals or statically parameterize the behavior of the modeled system. Second, Model inadequacy describes uncertainties in the structure, accuracy, type, etc. of the model itself. A model is a simplified and abstracted representation of reality. Hence, a model is not perfect and exhaustively representing the full behavior of an implementation [KO01, AAC12].

The so-called three-dimensional concept presented in [WHR+03] classifies uncertainty in location, nature, and level (see Figure 2.4). The main idea of this classification is a mapping of uncertainty into a dedicated category. This has further consequences in the documentation of uncertainty and their associated effects (discussed in more detail later). A drawback of the following categorization is that uncertainty underlays a lack of knowledge and may behave in a completely random way.

1. **Location:** Describes where the uncertainty occurs within the full model. Full model in this context means all described modeling information including the context, boundaries, model structure uncertainties, technical uncertainties, inputs, parameters, etc.

2. **Level:** The level of uncertainty highly correlates with the general level of knowledge (insight) of the system. The level of uncertainty may range from complete deterministic understanding to total ignorance and unpredictability.

3. **Nature:** [WHR+03] proposes for the nature of uncertainties two different types. Eptimistic uncertainty is given by the lack of knowledge, and variability uncertainty is defined by inherent deviation of parameters from their perfect value. Precise evaluation of uncertainty nature in general increases the knowledge and guides designers to consider, and finally reduce them achieving a more efficient system design.

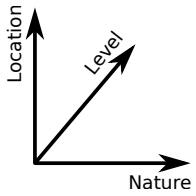


Figure 2.4: Classification of uncertainty in Nature, Location and Level [WHR+03].

The context of this thesis is electronic circuit and system design. Classification of uncertainty effects and their causes has been already discussed in detail in the following theses [Rad16, Sch13]. I just give a short selected summary of which uncertainties are modeled typically within a circuits and systems context:

- **Input variations:** voltage variation of input signals, supply voltage variations, ground bounce, voltage level distortion, signal noise, jitter in voltage phase and frequency, offset voltage drift, initial value variations etc.
- **Component variations:** component characteristics tolerances, temperature dependencies, aging effects, etc.
- **Circuit level variations:** gain deviations (opamp, mixer, filter, etc.), attenuation feedback loop variations, time delay variations, ADC (Analog to digital converter) quantization errors, round off errors in DSPs (Digital Signal Processors), etc.
- **Silicon level-, and process variations:** electron/holes mobility variations, NTBI (Negative Bias Temperature Instability) soft-errors, aging effects, process variability (e. g. MOS gate dimensions), etc.
- **Modeling variations:** Abstraction errors, value quantization errors, rounding and truncation, lack of parameter correlations, etc.
- **Environment variations:** unspecified usage, unpredictable events, unplanned decisions, security and safety issues, etc.

The next subsections focus in particular state of the art methods for modeling the itemized uncertainties. The presented methods differ in expressiveness, as well as in their abilities to describe uncertainty effects.

2.2.3 Interval Representations

Exact scalar numbers , are commonly used for representing values within a precise mathematical context. For an application in the real world, physical values have unavoidable (sometimes unpredictable) errors. This has the consequence that they deviate from an exact value. However, it does not matter whether this error comes from measurement processes, approximations, variabilities, representation as a limited bit-width digital number, etc. Using a single scalar number is not longer sufficient, the described error is covered by introducing a tolerance measure.

A popular method to describe tolerances is using interval representations. A set of two numbers indicating upper and lower numeric bounds ($\underline{X}$ and $\overline{X}$) is defined, where an inclusion of the exact number (x) is guaranteed $X = [\underline{X}, \overline{X}] = \{x : \underline{X} \leq x \leq \overline{X}\}$ [HJVE01, MKC09, Pop98]. An exemplary interval on a number ray is illustrated in Figure 2.5. Based on this definition the number set of all valid interval representations $\mathbb{I}$ is defined according to

$$\mathbb{I} = \{[\underline{X}, \overline{X}] | \underline{X}, \in \mathbb{R} \wedge \overline{X} \in \mathbb{R} \wedge \underline{X} \leq \overline{X}\} \tag{2.1}$$

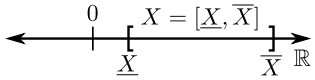


Figure 2.5: Interval specification

This pairwise description is also called endpoint notation for tolerances [MKC09]. Endpoint notation as presented, indicates a “closed” interval where upper and lower bounds are included

in the defined interval range. If upper and lower bounds are represented by non-infinite values, the interval is called bounded. It may happen that an interval gets unbounded caused by a mathematical operation (e. g. division where zero is an element of the divisor) [HJVE01].

2.2.4 Semi-Symbolic Modeling of Uncertainties

Semi-symbolic approaches for modeling uncertainty are an extension of standard interval representations introduced in Subsections 2.2.3 and 2.3.2. Deviations are modeled by using symbols indicating uncertainty causes. This solves the traceability problem (described in detail in the next sections) and overcomes several further disadvantages of Interval Arithmetic and multi-run based simulation methods. In this work, I will use Affine Arithmetic forms (AAF) for semi-symbolic modeling, simulation and system analysis processes.

Affine Arithmetic (AA) is a range based computation methodology which was originally developed for computer graphics [CS93]. It can be used for symbolic modeling of parameter deviations in any fields of technical calculations. Scalar exact values are extended by linear superimposed deviation parts. The central value (sometimes also called nominal value) and deviations are merged into a compound data type called an Affine Arithmetic form (AAF) [SF03, GGB06].

An AAF according to [SF97, SF03] and used for this thesis is mathematically defined as:

$$\hat{x} = x_0 + \sum_{i=1}^{N} x_i \epsilon_i = x_0 + \sum_{i \in \mathcal{N}_{\hat{x}}} x_i \epsilon_i \tag{2.2}$$

$$\text{with } \hat{x} \in \mathbb{A} \ , \ x_0, x_i \in \mathbb{R} \ , \ i \in \mathbb{N}^+ \ , \ \epsilon_i \in \mathbb{U}$$

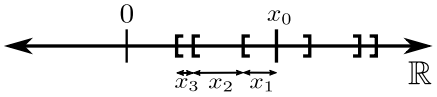
$$N \in \mathbb{N} = max(\mathcal{N}_{\hat{x}})$$

$$\mathcal{N}_{\hat{x}} = \{i | x_i \neq 0\}$$

In this work, a variable representing an AAF is indicated by a hat sign (e. g. $\hat{x}$). The scalar center value x_0 ($x_0 \in \mathbb{R}$) denotes the exact value which is present if the deviation is 0. This is the equal numerical value as used for standard numeric simulation without considering any uncertainty.

The full deviation range of an AAF is partitioned in a sum of partial deviations x_i. x_i are scalar numbers with $x_i \in \mathbb{R}$. Each partial deviations is multiplied by a deviation symbol (sometimes called noise symbol) ϵ_i. These have an unpredictable value between -1 and $+1$. Thus, Affine Arithmetic forms hold no information about a statistical probability distribution of deviations. Symbols just represent that their value is guaranteed, but unpredictable, and unknown between -1 and 1. ϵ symbols are also representable as a closed interval with the range $[-1, 1]$; $\epsilon_i \in \mathbb{U} = [-1, 1] \subseteq \mathbb{R}$ [PLV10]. x_i scale this range for the given AAF. All partial deviations of an AAF are superimposed, and the sum describes the total variability around the numerical center value x_0. Due to the symmetry of ϵ symbols all modeled deviations are located symmetrically around the center value. A deviation symbol may represent a physical uncertainty effect which influences signals or system parameters (e. g. manufacturing tolerances, temperature, electromagnetic influences, ...) [GGB06, Sch13] and enable new possibilities in simulation and analysis of a system. i is the counting index for the sum of partial deviations. It must be taken into consideration that an AAF can contain different deviations which associated noise symbol index i is not sequentially increasing (e. g. $50 + 3\epsilon_2 - 5\epsilon_4$). One method to avoid this is to allow sequentially counting up indices to the limit of N. An extra definition specifies that non impact

uncertainties have a partial deviation value x_i of zero (e. g. $50 + 0\epsilon_1 + 3\epsilon_2 + 0\epsilon_3 + 0\epsilon_4 - 5\epsilon_4$). An further method is to define a specific set $\mathcal{N}_{\hat{x}}$ which contains only indices where the corresponding partial deviation is not zero. $i \in \mathcal{N}_{\hat{x}}$ and $i \in \mathbb{N}^+$ (e. g. $50 + 3\epsilon_2 - 5\epsilon_4$, $\mathcal{N}_{\hat{x}} = \{2, 4\}$) [SF03]. In this work this latter sum method will be used for representing AAFs. With the definitions expressed in equation 2.2 for all valid Affine Arithmetic forms a specific number set $\mathbb{A}$ is defined. All real numbers are subset of $\mathbb{A}$ ($\mathbb{R} \subseteq \mathbb{A}$). Standard Affine Arithmetic forms as used in this thesis include real and no complex values ($\mathbb{C} \nsubseteq \mathbb{A}$).



$$\hat{x} = x_0 + x_1\epsilon_1 + x_2\epsilon_2 + x_3\epsilon_3$$

Figure 2.6: Graphical number ray representation of an AAF

In Figure 2.6 a graphical representation of an AAF is shown on a number ray. The included sub ranges (defined by partial deviation values) are plotted symmetrically around the central value in an accumulated style. Thus, the presentation is a worst-case scenario where positive and negative interval bounds are specified by ϵ values of ± 1. The order of the sub intervals is not significant due to the commutativity of the sum operation (in most of the cases the index i is increasing from inside to outside located sub intervals). As already mentioned, and illustrated in the figure sub intervals are never overlapping.

For the description of variation in time an Affine Arithmetic signal $\hat{x}(t)$ is defined as a (maybe large) sequence of AAF samples [GR17]:

$$\hat{x}(t) = x_0(t) + \sum_{i \in \mathcal{N}_{\hat{x}}} x_i(t)\epsilon_i \tag{2.3}$$

The dependency of partial deviations x_i on time is optional. If x_i varies in time or is dependent on other values (e. g. x_0) a deviation is called dynamic. Otherwise, a deviation is called static (constant).

Figure 2.7 illustrates an AAF signal. The dashed red line indicates the center value. The blue lines represent the bounds for the deviation 1 associated with the symbol ϵ_1. This partial deviation is static and thus has a constant value over time. The partial deviation associated with the symbol ϵ_2 is dynamic, and its value depends on the value of x_0. Boundaries for the accumulated partial deviation 1 and 2 are plotted in green. The order for plotting and accumulation of the two included partial deviations is in this case defined as ϵ_1,ϵ_2 but is in general arbitrary. For an AAF signal each point in time is defined by a single Affine Arithmetic form sample. In Figure 2.7 this is indicated for $t = 0.2$.

As a summary, the following advantages and disadvantages of representing uncertainty by Affine Arithmetic forms are discussed.

Advantages:

- A selected type of uncertainty can be modeled by using a freely assignable deviation symbol ϵ_i.

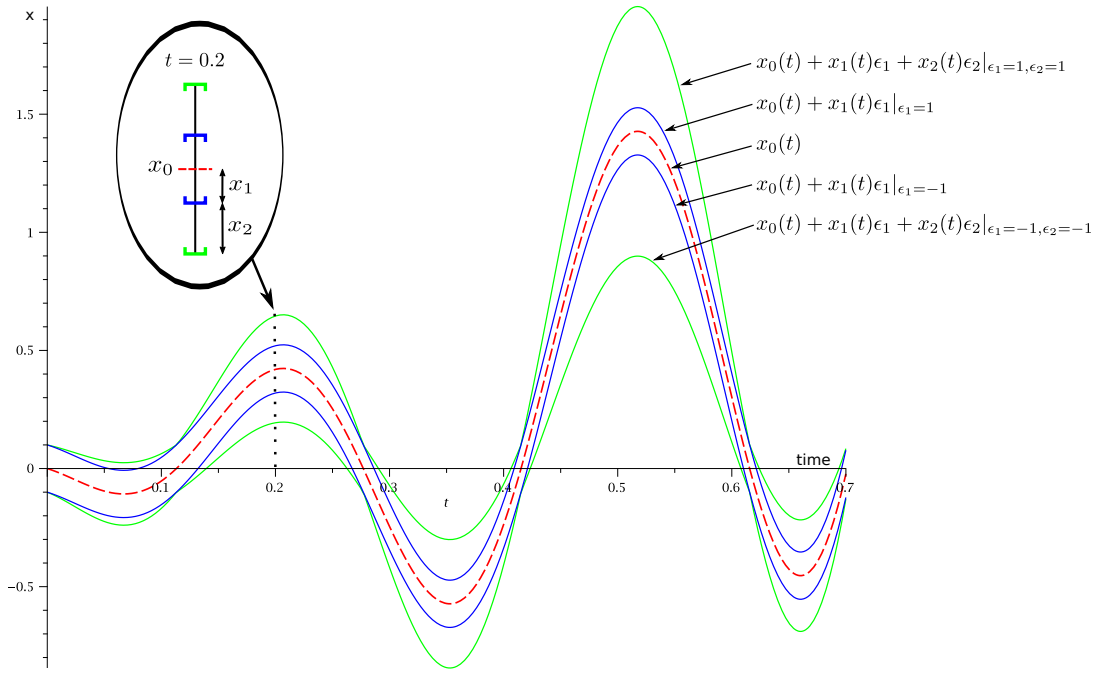


Figure 2.7: Illustration of an AAF signal

- The description of correlated uncertainty effects is provided by multiple usage of deviation symbols in Affine Arithmetic forms.
- Partial deviations x_i have a sign and can be either positive or negative. This describes their impact if they are interrelated with correlated forms. Thus, for deviations gaining and attenuation effects (e. g. amplifiers or feedback loops) can be modeled.

Disadvantages:

- Used symbols have to be managed by an adequate modeling framework. Unmanaged dynamic creating and deleting of symbol instances will lead to disarrayed structure and maybe lack of correlation representation.
- A limit of uncertainty modeling capabilities is given by the closeness and symmetry of AAFs. Partial deviations are due to their ϵ definition closed symmetric intervals.
- Standard Affine Arithmetic forms hold no information about the statistical probability distribution of deviation inside the defined ϵ interval.
- For the presentation of results and especially temporal traces Affine Arithmetic forms have to be unpartitioned to scalar traces. Traces of AAF datatype cannot be plotted by standard waveform viewer tools by default.
- For sure there is an enhanced model complexity which forces a system architect analyzing deviation effects, their origin and correlation effects in detail. This is time-consuming and requires an increased effort during the design phase.

2.3 Simulation Techniques for considering Range Inputs

In this section I will describe how the techniques for uncertainty modeling, presented in Section 2.2 are used for system simulation. Strictly speaking, basic mathematical operations and associated functions (e. g. for transferring intervals to AAFs and vice versa) are described Multi-run, Interval Arithmetic and Affine Arithmetic simulation methods are motivated, and their advantages/disadvantages for comprehensive system simulation are highlighted.

2.3.1 Multi-run Approaches

A commonly used approach to cover uncertainties of signals, component characteristics, parameters, etc. at simulation are so-called multi-run techniques. Single numerical simulation runs are executed multiple times at varying scalar input valuations. Hence, for each run a single value, limited by an uncertainty tolerance window, is selected for each uncertain variable. The deterministic model used for this approach is equal to the initial system design used for classical functional simulation [Rub81]. The evaluated uncertainty is considered within the simulation process itself, but not within the model. Thus, there is no specific modeling required for performing this type of simulation. Main common characteristic of the following described multi-run methods is the repeated execution of single simulation runs, under sequential changes of input valuations [Ray08].

Monte Carlo simulation
Monte Carlo (MC) is a type of multi-run simulation where this value is selected randomly, considering a specified probability density function (pdf). This pdf represents a model for the total variance and distribution of uncertain inputs and parameters. Thus, MC simulation allows the definition of a statistical distribution of deviations. The Monte Carlo method can be seen as a *what-if analysis* resulting in statistical characteristics about the reaction of the system under inputs deviating from their ideal value [PKB07, HA08, Rub81, SCS+00].

According to [Ray08] the methodological steps when performing a MC simulation can be identified (illustrated in Figure 2.8):
Model Generation: As mentioned the model used for MC simulation includes no modifications from the exact initial design. For uncertainty analysis, there is no need for adaption of the original model used for design purposes (functional simulation, verification, high-level synthesis, etc.).
Input Distribution Identification: As a second step deviated inputs are described. First, it has to be defined which inputs of the model are considered to be under variations. Second, the corresponding characteristic of the variation is determined. Since deviations originate from a stochastic manner, underlying input probabilistic distributions have to be identified. Proposed input characteristics are derived from historical data or analysis of a real-world application.
Random Variable Generation: Based on the variable distributions identified at the previous step a set of random input variables is generated. The set identifies one specific operation point of the system. Thus, the simulation is pure numerical with scalar inputs. Outputs, which are mapped to the generated input variable set, are monitored and collected for further processing. This step, which is the core part of the classical sequential Monte Carlo simulation procedure, is repeated N times. N represents the accuracy which is proportional to the sample density of the simulation run.
Analysis and Decision Making: As a final step simulation results are post-processed. Applied algorithms calculate standard sample statistical measures like mean, variance, skewness, histograms, sensitivity analysis parameters etc.

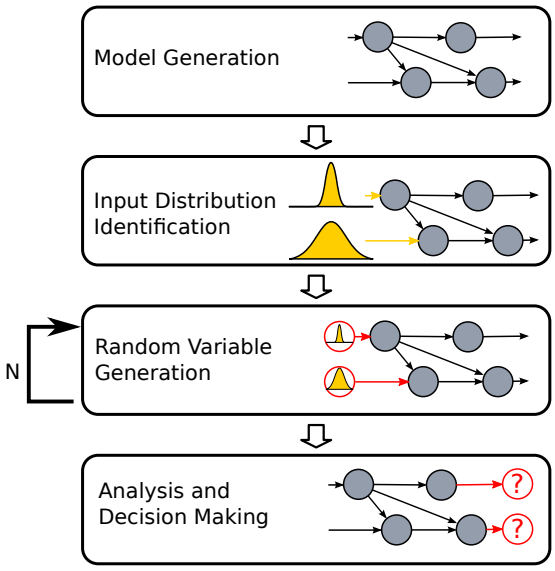


Figure 2.8: Steps of a full Monte Carlo simulation process. First, the input model has to be defined. Second, the probability distribution of the varying input nodes are defined. Third, as a main step, random input valuations are generated N times. Last, the generated simulation results may be analyzed in a statistical way.

Enhancements in Monte Carlo methods:
Due to the huge popularity of Monte Carlo methods and their integration in commercial simulation tools several scientific publications are enhancing the MC simulation method. Research fields reach from circuit simulation, solid-state structures, biomedical applications to financial analysis and many more. Common goals are to speed up simulation processes and enhance the accuracy.

[SB15] gives a survey of sampling techniques for a defined reliability problem. The failure probability is given by integration of a probability density function (pdf) where the result is not known as a closed form. Thus, MC integration, which samples the according pdf, is used. [SB15] presents first simple sampling techniques (antithetic variate sampling, Latin Hypercube Sampling) and second, intelligent techniques (importance sampling, asymptotic sampling, etc.) These methods are compared and benchmark measures are evaluated based on specified use cases. Especially importance sampling is a significant technique in the context of simulation applications. [CL07] presents an illustrative example in the introduction section of the publication which highlights the power of importance sampling: "For an event occurring with probability 10^{-4}, one expects the occurrence of 1 event every 10000 simulation runs. [...] To simulate a small probability $P(A)$, importance sampling changes the measure P to Q under which A is no longer a rare event [...]." [CL07] This addresses the problem that the number of simulation runs N has to be very large to give good coverage of the full deviation range [CGM07]. Wights are defined by the ratio between the P and Q distribution. The according function over the full range of the distributions is then called importance function. Finding this importance function is essential for the method. [CGM07] discusses this problem in detail and presents approaches using normalized distributions in combination with estimators and coordinated re-sampling.

Worst case simulation
Worst-case simulation is a sub-type of multi-run simulations where parameters are not selected randomly, but lower and upper boundary values are considered. Hence, the number of single runs can be pre-calculated and depends on all defined possible combinations of boundary values. The

number of runs grows exponentially by the number of considered uncertainties [Rub81]. Worst-case simulation is a well applicable simulation method for applications where the system response can be considered as monotonic. If monotonicity is fulfilled, worst-case behavior is defined as a combination of worst-case uncertainty combinations [Dro09].

Implementation of Multi-run Methods in commercial tools
[JHY11] reports about implementing MC methods in their Matlab/Simulink tool. The proposed method uses Matlab's random number generators and stores result in M-dimensional arrays where M is the number of MC rounds. Systems illustrated in the example section of [JHY11] are described by a set of formulas and not within a hierarchical structure of modules. Thus, a reset of the model is done by re-initialization of variables and MC execution is realized by looped calculation of the defined formulas. For result analysis, they use Matlab to calculate statistical parameters and create handsome diagrams.

[HWC08] is about implementing MC simulation with Modelica [17]. The discussed approach is close to the standard J2748 [18] which describes random pattern handling within VHDL-AMS. [HWC08] reports in detail how different types of random number generators are implemented and integrated within the Modelica simulation framework. The model calculation (a differential amplifier) is called within a loop structure. None of the two proposed frameworks provide functionalities for MC round management and efficiently controlling the full Multi-run simulation behavior.

SPICE (Simulation Program with Integrated Circuit Emphasis) implements Multi-run simulation and analysis techniques natively in the simulation environment [Cad10]. Functions for MC simulation, worst case analysis, parameter sweeping, etc. are provided per default. Developers of SPICE based tools follow more and more the trend towards full system simulation (e. g. [28]). Besides circuit elements, models may also include abstract behavioral blocks as transfer functions, signal processing algorithms, processor cores including software, etc. [Cad10].

2.3.2 Interval Simulation Techniques

Interval representations for modeling uncertainties are introduced in Subsection 2.2.3. For the development of an Interval Arithmetic based simulation framework mathematical operations have to be defined. An interval can be seen as a compound interval-type which is given by its upper and lower bounds [MKC09]. Intervals have to be bounded (specific non-infinite lower and upper limits) for the definition of this arithmetic. According to [MKC09] basic operations for intervals are defined. x and y are representative values located inside the specified intervals X and Y.

$$X \oplus Y = [x \oplus y : x \in X, y \in Y], \oplus = + - \cdot /$$

Resulting intervals defined by a $\oplus$ operations are calculated using the operand's upper and lower limits [Kol93, MKC09].

$$X \pm Y = [\underline{X} \pm \underline{Y}, \overline{X} \pm \overline{Y}]$$
$$X \cdot Y = [min(S), max(S)], S = \{\underline{X} \cdot \underline{Y}, \overline{X} \cdot \underline{Y}, \underline{X} \cdot \overline{Y}, \overline{X} \cdot \overline{Y}\} \quad (2.4)$$
$$X/Y = X \cdot (1/Y), (1/Y) = [1/\overline{Y}, 1/\underline{Y}], 0 \notin Y \quad (2.5)$$

Additional interval parameters associated for the following discussions are width $w(X)$, absolute value $|X|$ and midpoint $m(X)$.

$$|X| = max\{|\underline{X}|, |\overline{X}|\} \tag{2.6}$$
$$w(X) : \mathbb{I} \mapsto \mathbb{R} \tag{2.7}$$
$$w(X) = |\overline{X} - \underline{X}| \tag{2.8}$$
$$m(X) : \mathbb{I} \mapsto \mathbb{R} \tag{2.9}$$
$$m(X) = (1/2)(\underline{X} + \overline{X}) \tag{2.10}$$

Using these characteristics every interval can be represented alternatively by a midpoint/width form [Kol93, MKC09]

$$X = m(X) + \left[-\frac{1}{2}w(X), \frac{1}{2}w(X)\right] \tag{2.11}$$
$$X = m(X) + \frac{w(X)}{2}[-1, 1] \tag{2.12}$$

According to these definitions several publications [HJVE01, MKC09, Kol93] present a complete Interval Arithmetic (also including order relations, union-, intersection operations, etc.). Most cited Interval Arithmetic implementation for C++ is "Boost" [24]. For Matlab which is widely used "b4m" is developed and provided as an add-on toolbox [6]. In general, Interval Arithmetic is well embedded into simulation environments and there are a lot of publications using IA for circuit and system case studies. Authors mainly address runtime advantages in contrast to multi-run techniques as Monte Carlo and worst-case simulation. The focus of this work is enhanced analysis features by deviation tracing. This is just fractionally given in the IA calculus and motivates to extend for Affine Arithmetic as given in Subsections 2.2.4 and 2.3.3.

For interval arithmetic [HJVE01] presents the following set of algebraic properties:
Correctness for Interval Arithmetic, describes that the result of an operation covers all individually possible result values. This is valid for arbitrary point-wise scalar execution of any scalar inside the argument intervals.
Totality is given if an operation is defined for all possible arguments. For interval arithmetic, this is just partially fulfilled (unbounded result by a division of an interval containing 0).
Closeness defines that an interval operation results from a valid interval. This is also just partially fulfilled (exception is a division where the dividend interval contains the value 0). According to [HJVE01] this problem can be alternatively described by distributing the interval as $\frac{1}{[\underline{X},\overline{X}]}$ where $\frac{1}{[\underline{X},0_-]} = [-\infty, \frac{1}{\underline{X}}]$ and $\frac{1}{[0_+,\overline{X}]} = [\frac{1}{\overline{X}}, \infty]$. As a concequence, the division results in union of $[-\infty, \frac{1}{\underline{X}}]$ and $[\frac{1}{\overline{X}}, \infty]$. This set has more information than the single form joint interval result of $[-\infty, \infty]$.
Optimality defines that the resulting interval of an operation is not wider than necessary. In some cases, optimality is defined by including an endpoint to a specific interval or not (open or closed interval definition).
Efficiency is mainly driven by higher complex algorithms. It may happen that various subroutine calls slow down simulation speed, especially in execution systems without a floating-point arithmetic unit.

For conclusion, I discuss advantages and disadvantages for interval representations and their associated simulation techniques:

Advantages:

- Intervals provide an easy-to-use possibility representing uncertainty as ranges by upper and lower worst-case value definition.
- Interval representations are well embedded into industrial applications and a standard for representing tolerance measures.
- Under consideration of bounded intervals IA simulation is fast and provides a range result within a single simulation step.
- Mathematical operations are well defined, and in general, the result of an operation (at least for basic $+, -, \cdot, /$) is a bounded interval [DLM07].
- For plotting simulation results, lower/higher bounds can be used to illustrate potential worst-case behavior. In principle, two plots represent a single signal. Both plots are standard numerical sequences, which can be processed by standard waveform viewer tools.
- The already identified idea of an enhanced datatype representing sets of disjoint intervals would enable new possibilities for deviation modeling in the circuits and systems context [HJVE01, DLM07].

Disadvantages:

- Intervals are defined in a conservative nature for covering all achievable values resulting from an exact calculation. Thus, intervals resulting form sequential calculations may get disappointingly large (conservative propagated uncertainty) [Pop98].
- Uncertainty causes represented by interval definitions cannot be identified uniquely [HJVE01]. This has following disadvantages in analysis and tracing of intervals. Uncertainty represented by intervals is obfuscated in the output forms, and its impact cannot be traced back to specific causes.
- Correlated uncertainty effects cannot be represented. At default representation, interval forms can not be linked.
- As a result of the previous two items, an interval may just get larger at sequential calculations. For circuits including feedback loops, it may happen that deviation bounds are affected by attenuation effects.

2.3.3 Semi-Symbolic Calculation Methodology

Using Affine Arithmetic forms for simulation, operators for AAFs have to be defined the so-called affine arithmetic based on $\mathbb{A}$. The proposed goal is just to replace scalar values in a numerical computer model by Affine Arithmetic forms for considering uncertainty. The modeling methodology using Affine Arithmetic forms is introduced in Subsection 2.2.4.

First, I introduce basic linear operations which do not require approximation of results. This means that the result of an operation can be represented by a valid AAF. For $\mathbb{A}$ these are the affine

addition, subtraction and all basic operators where one operand is a scalar (see equations 2.13 and 2.14).

$$\hat{x} \oplus_{lin} \hat{y}, \hat{x} \in \mathbb{A}, \hat{y} \in \mathbb{A} \text{ where } \oplus_{lin} = +, - \tag{2.13}$$

$$\hat{x} \oplus_{lin} y, y \in \mathbb{R} \text{ where } \oplus_{lin} = +. - . \cdot ./ \tag{2.14}$$

Using the definition of an AAF given in equation 2.2 these linear operations are defined as:

$$\hat{x} \pm \hat{y} = x_0 \pm y_0 + \sum_{i \in \mathcal{N}_{\hat{x}}} x_i \epsilon_i \pm \sum_{i \in \mathcal{N}_{\hat{y}}} y_i \epsilon_i \tag{2.15}$$

$$= x_0 \pm y_0 + (x_i \pm y_i)\epsilon_i + (x_{i+1} \pm y_{i+1})\epsilon_{i+1} + \ldots \tag{2.16}$$

$$\hat{x} \pm y = x_0 \pm y + \sum_{i \in \mathcal{N}_{\hat{x}}} x_i \epsilon_i \tag{2.17}$$

$$\hat{x} \cdot y = x_0 \cdot y + \sum_{i \in \mathcal{N}_{\hat{x}}} (x_i \cdot y)\epsilon_i \tag{2.18}$$

$$\hat{x}/y = x_0/y + \sum_{i \in \mathcal{N}_{\hat{x}}} (x_i/y)\epsilon_i \tag{2.19}$$

As a result, it can be evaluated that these linear operations handle correlations correctly. Strictly speaking, partial deviation of both operators associated with equal deviation symbols (representing a common deviation cause) are operated correctly. In IA impacts of equal causes, are represented, and computed independently. Thus, origins are obfuscated and incorporated in an accumulated interval representation.

Besides these linear operations also other mathematical operations have to be specified. They are non-linear because the result is no longer representable by a valid Affine Arithmetic form, according to equation 2.2.

$$\hat{x} \oplus_{nonlin} \hat{y} = z, \hat{x} \in \mathbb{A}, \hat{y} \in, z \notin \mathbb{A} \text{ where } \oplus_{nonlin} = \cdot, /, \text{etc.} \tag{2.20}$$

In standard Affine Arithmetic forms only linear deviations can be expressed. A quadratic AAF approach, which is not used in this work has been published in [GGB06]. Non-linear functional operations on two or more AAFs can produce polynomial terms with an order greater than one. For example, the multiplication of two Affine Arithmetic forms may contain linear combinations and quadratic expressions of deviation symbols. For further ongoing calculation, it is required that such non-linear operations return a valid AAF. High order partial deviations have to be converted to linear terms. Therefore, appropriate approximation algorithms for multiplication, division, inversion, exponential function, trigonometric functions, etc. have to be integrated into an AAF calculation framework [SF03]. The conventional method of approximations is to add an additional deviation symbol (in this thesis this is called a system deviation) covering the non-linear characteristics of the operation. Already published approximation schemes for non-linear operations in $\mathbb{A}$ are:

Strict linearization (Taylor form): The non-linear characteristic of an operation can be linearized at any operation point within the deviation range. Therefore, often the central value is

used, but also other operation points where the tangent is calculated are possible. The result is a linear function which can be represented by an AAF according to equation 2.2. This first order Taylor approximation adds no extra system deviation symbols but leads automatically to a systematic calculation error due to under-approximation [SF03]. An advantage of this method is that the returned AAF solution has an exact functional value and correct partial deviations (sensitivities) at the point of linearization. A disadvantage is that the calculation error has an unpredictable sign for values unequal to the point of the linearization. The absolute value and the sign of the error depend on the second derivation of the exact mathematical form. This approximation method can be used for applications where small absolute errors close to the point of linearization are given. The approximation is only accurate if nearly all ϵ values are predictable in a limited range around the point of linearization (e. g. operation point of a transistor circuit).

Chebyshev approximation: The goal of the Chebyshev approximation method is to calculate a minimum-area parallelogram. The parallelogram have to enclose the exact (non-linear) result function in the given ϵ intervals of $[-1, 1]$ [SF03]. The slope of the non-vertical parallelogram sides is computed from the first derivation of the non-linear function. This additional deviation represented by the area of the parallelogram is modeled by an additionally added system deviation symbol. Thus, the vertical height of the parallelogram defines the additional partial deviation value [RK15], [Gra09]. Due to the property that deviations are located symmetrically around the AAF's center value for this approximation also a shifted center value is required. An advantage of this approximation method is that full inclusion of the exact result at any specific ϵ value (-1 to $+1$) is guaranteed. The area of the parallelogram defined by the additional partial deviation approximation value depends on the nonlinearity of the operation [Gra09]. A disadvantage of this approximation method is that there is over-approximation at least one of the interval bounds. Another disadvantage is that the center value is shifted at this method. For a repeated execution of operations (within a loop) the exact central value must be reminded. This is accomplished by the simultaneous implementation of approximations within the AAF class (see Section 3.2). Applications for Chebyshev approximation are simulations where inclusion of the exact resulting value must be guaranteed for every single value of included deviation symbols.

Minimal range approximation: The minimal range approximation is defined as a minimization problem. An AAF is constructed which fulfills the requirement that the radius of the form is a minimum [Gra09], [RK15]. A disadvantage of this approximation is that all deviations (also user deviations which represent the sensitivity of the output in respect to parameter deviations) are manipulated, and exact computable linear factors of the result are overwritten (and lost for further calculations). An advantage of minimal range approximation is that there is no over--approximation at the interval bounds [GGB06].

Figure 2.9 shows the Taylor, Chebyshev and minimal range approximation of a non-linear function (bold line in both figures). In Figure 2.9-a the linear function labeled with crosses is an AAF Taylor approximation at the central value. The gray shaded area is the parallelogram spanned by a Chebyshev approximation of the non-linear function. The grayed area at Figure 2.9-b illustrates the full inclusion parallelogram of a minimal range approximation. As a result, the figures indicate over-approximation, shifted central values and deviation sensitivity properties of each discussed method.

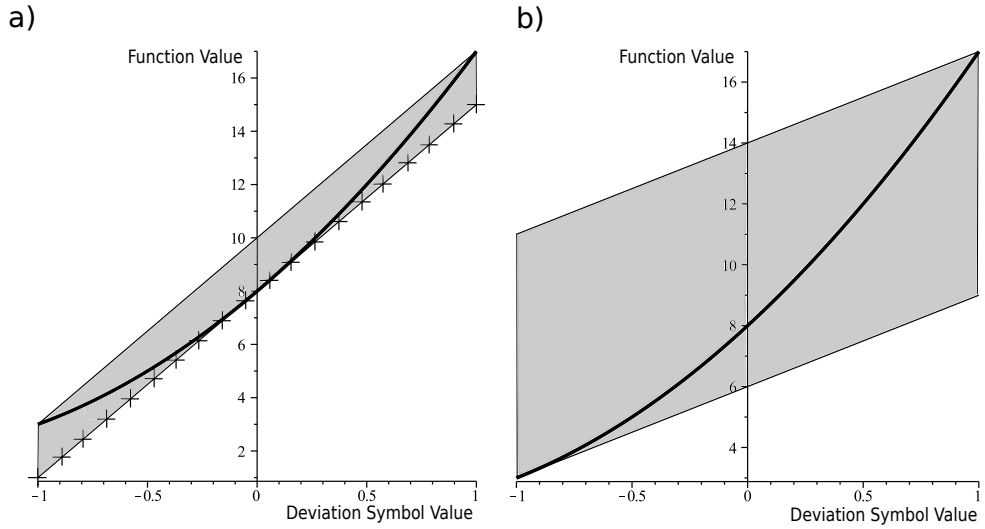


Figure 2.9: Approximations of a non-linear function (bold curve). a) Illustrates a linearization (Taylor approximation) and the spanned area resulting from a Chebyshev approximation. b) Illustrates the spanned area at an approximation with the minimal range method.

Besides definition of an arithmetic on $\mathbb{A}$ I give a brief overview and discuss some selected additional functions defined on AAFs.

Hansen's form

Hansen's form is a combination of AAF and interval representations initially defined in [Kra06]:

$$\tilde{x} = x_0 + \sum_{i \in \mathcal{N}_{\hat{x}}} x_i \epsilon_i + [\underline{X}, \overline{X}] \tag{2.21}$$

$\mathbb{I}$ and $\mathbb{A}$ are subsets of all valid Hansen's forms. The interval can be zero, and the set $\mathcal{N}_{\hat{x}}$ can be empty. Hansen's forms have the advantage that potential widening of AAF ranges, caused by approximation of non-linear operations, can be specified by the interval part of the form. No additional symbol has to be added to the affine part of the result. Thus, linear and non-linear operations defined on Hansen's forms always result in a Hansen's form. Potential advantages of this representation, in general, are given by constant time and space complexity [Rad16]. I will use an approach similar to the Hansen's form for motivating a new type of approximation (see Subsection 3.2.2).

Radius of an AAF

The radius of an AAF is defined by its deviation bounds. Basically, the bounds are given by summing up all included partial deviation values x_i of a form independent of its sign. Thus, the radius represents the worst-case variability resulting from individual $-1{,}1$ settings of deviation symbols. As given in equation 2.22 the radius of a AAF is the absolute sum of all partial deviation values [Gra09]. The radius calculation function mathematically expressed as rad, is a function which operated from the set $\mathbb{A}$ the set of affine forms to $\mathbb{R}^+$ a real scalar positive number (including 0):

$$rad(\hat{x}) : \mathbb{A} \mapsto \mathbb{R}^+$$
$$\mathbb{R}^+ = \{\gamma | \gamma \in \mathbb{R} \wedge \gamma \geq 0\}$$

The radius of a scalar is due to non-existing deviation parts defined as 0. $rad(z) = 0$ for $z \in \mathbb{R}$. For valid AAF the radius is a positive real value defined according to equation 2.22.

$$rad(\hat{x}) = \sum_{i \in \mathcal{N}_{\hat{x}}} |x_i| \tag{2.22}$$
$$\hat{x} \in \mathbb{A} \ , \ x_i \in \mathbb{R} \ , \ rad(\hat{x}) \in \mathbb{R}^+$$

In addition to this definition of the radius function for Affine Arithmetic forms the following properties are discussed:

- *rad* is a non-injective function. A *rad* value of the co-domain is not uniquely leading to a single AAF element of its domain [Die03, p.55]. Due to a custom partition of deviations one radius value can be assigned to the same AAF. $rad(\hat{x}_1) = rad(\hat{x}_2) \Rightarrow \hat{x}_1 = \hat{x}_2$ is not fulfilled. Radius equivalence do not result in identical affine forms.
- The *rad* function is surjective, because each element of the co-domain is assigned to at least one element of its domain [Die03, p.56].
- The *rad* function is not bijective (just surjectivity is fulfilled) [Die03, p.56]. This means that the *rad* function is not uniquely invertable.
- Each functional composition $rad \circ rad$ results in zero. Assume $\zeta = rad(\hat{x})$, $\zeta \in \mathbb{R}$ and $rad(\zeta) = 0 \Rightarrow rad(rad(\hat{x})) = 0$.
- The *rad* function is not linear. According to

$$rad(a \cdot \hat{x}) = a \cdot rad(\hat{x})$$
$$\sum_{i \in \mathcal{N}_{\hat{x}}} |a \cdot x_i| = a \cdot \sum_{i \in \mathcal{N}_{\hat{x}}} |x_i|$$
$$\text{due to} \quad |a \cdot x_i| = a \cdot |x_i|$$
$$a \in \mathbb{R} \ , \ \hat{x} \in \mathbb{A}$$

  it fulfills homogeneity. But it is not additiv:

$$rad(\hat{a} + \hat{b}) \neq rad(\hat{a}) + rad(\hat{b})$$
$$\sum_{i \in \{\mathcal{N}_{\hat{a}} \ \cup \mathcal{N}_{\hat{b}}\}} |a_i + b_i| \neq \sum_{i \in \mathcal{N}_{\hat{a}}} |a_i| + \sum_{i \in \mathcal{N}_{\hat{b}}} |b_i|$$
$$\text{due to the triangle inequality} \quad |a_i + b_i| \neq |a_i| + |b_i|$$
$$\hat{a} \in \mathbb{A} \ , \ \hat{b} \in \mathbb{A}$$

  For linearity both properties have to be fulfilled [Wil03], so it is not linear.

Maximum and minimum Functions

Maximum and minimum functions return the bounds of an AAF under the consideration of its accumulated modeled uncertainty. The functions are mathematically expressed as *max* and *min* and assign an element in the set $\mathbb{A}$ to a scalar number in $\mathbb{R}$.

$$min(\hat{x}) : \mathbb{A} \mapsto \mathbb{R}$$
$$max(\hat{x}) : \mathbb{A} \mapsto \mathbb{R}$$

Maximum and minimum of an AAF can be calculated using the radius function rad.

$$min(\hat{x}) = x_0 - rad(\hat{x}) \tag{2.23}$$
$$max(\hat{x}) = x_0 + rad(\hat{x}) \tag{2.24}$$
$$\hat{x} \in \mathbb{A} \ , \ min(\hat{x}) \in \mathbb{R} \ , \ max(\hat{x}) \in \mathbb{R}$$

min and max of a real scalar value are equal to the value itself. An AAF with radius 0 has the property that all partial deviations are 0 with the consequence that minimum and maximum values are equal to its center value.

$$min(y) = max(y) = min(\hat{x}) = max(\hat{x}) = y$$
$$\text{if} \quad y \in \mathbb{R} \ , \ \hat{x} \in \mathbb{A} \ \text{ with } \ x_0 = y \ \text{ and } \ rad(\hat{x}) = 0$$

Due to the symmetry of the deviations around the center value minimum and maximum of an AAF can be directly transferred to each other.

$$max(\hat{x}) = min(\hat{x}) + 2 \cdot rad(\hat{x}) \tag{2.25}$$
$$min(\hat{x}) = max(\hat{x}) - 2 \cdot rad(\hat{x}) \tag{2.26}$$

The min/max function has the following mathematical properties:

- min, max functions are not injective because a maximum element of the co-domain can be mapped form several Affine Arithmetic forms in the domain of the functions.
- The min, max functions are surjective because each element of the co domain can be mapped at least to one element of the domain.
- Due to the non-ijectivity the minimum, maximum functions are not uniquely invertable. A given scalar value representing a maximum/minimum can not be mapped to a single specific AAF.
- Each functional composition $max \circ max$, $min \circ min$ results, and also multiple functional compositions result in the max/min value of the innermost operation (the max/min of a scala is the value itself).

  $$max(max(\hat{x})) = max(\hat{x})$$
  $$min(min(\hat{x})) = min(\hat{x})$$
  $$\forall \hat{x} \in \mathbb{A}$$

  Thus, idempotence with respect to a functional composition is fulfilled for min and max.
- Linearity is not given for the min, max functions. On the first hand, homogeneity is fulfilled due to the homogeneity of the radius function (see equations 2.25 and 2.26). On the other hand additivity is not fulfilled due to the non-linearity of radius function. Thus, min and max are also non-linear.

Transformation functions between $\mathbb{I}$ ***and*** $\mathbb{A}$

Using the previously defined functions (rad, min, max, m w), transformation functions between

intervals $\mathbb{I}$ and Affine Arithmetic forms $\mathbb{A}$ can be defined. The transformation Γ and its pseudoinverse transformation Θ maps an element of $\mathbb{A}$ to $\mathbb{I}$ and vice versa.

$$\Gamma(\hat{x}) : \mathbb{A} \mapsto \mathbb{I}$$
$$\Theta(X) : \mathbb{I} \mapsto \mathbb{A}$$

For the definition of the transformation functions m, w for intervals and min, max, rad functions for AAFs are used.

$$\Gamma(\hat{x}) = [min(\hat{x}), max(\hat{x})] = X \tag{2.27}$$
$$\Theta(X) = m(X) + \delta = \hat{x} \tag{2.28}$$
$$\text{where } \delta = \sum_{i \in \mathcal{N}_{\hat{x}}} x_i \epsilon_i \ \wedge \ rad(\hat{x}) = \frac{w(X)}{2}$$
$$X \in \mathbb{I} \ , \ \hat{x} \in \mathbb{A}$$

The transformation function Γ is not injective, but surjective. This has the consequence that Γ is not uniquely invertable. min and max functions used for the definition of Γ equation 2.27 are not uniquely invertable. Strictly speaking, multiple combinations of partial deviations (subintervals of an AAF) can lead to equal boundary values which represent an interval. However, for applications the so-called pseudo inverse transformation Θ makes sense $\Theta \neq \Gamma^{-1}$. Θ may return one possible AAF representation of an interval. The interval width $w(X)$ can be arbitrarily partitioned to multiple partial deviations.

2.3.4 Selected Affine Arithmetic Properties on a Dense Number Space

In this section I discuss selected topological aspects of $\mathbb{A}$. In general, a binary relation between two AAF elements of the set $\mathbb{A}$ is defined as a subset of the Cartesian Product of $\mathbb{A} \times \mathbb{A}$. For the statement $A(\hat{x}, \hat{y})$ a relation R is defined as $R = \{(\hat{x}, \hat{y}) \in \mathbb{A} \times \mathbb{A} | A(\hat{x}, \hat{y})\}$. This can be also written as $\hat{x} R \hat{y}$ [Her06, p.28]. Specific definitions of $\mathbb{A}$ can be used to define an order of AAFs (order relation). In the following itemization selected order relations are defined.

- $<$ and $\leq$: The less relation is defined in a way that an AAF has to be smaller than an other under worst case considerations (bounds of the AAF). This is expressed as $R_< = \{(\hat{x}, \hat{y}) \in \mathbb{A} \times \mathbb{A} | max(\hat{x}) < min(\hat{y})\}$. The $\leq$ relation expresses that bounds of the AAF may touch (be equal) at a single point $R_\leq = \{(\hat{x}, \hat{y}) \in \mathbb{A} \times \mathbb{A} | max(\hat{x}) \leq min(\hat{y})\}$.
- $>$ and $\geq$: These relations are defined similar to $<$ and $\leq$, $R_> = \{(\hat{x}, \hat{y}) \in \mathbb{A} \times \mathbb{A} | min(\hat{x}) > max(\hat{y})\}$ and $R_\geq = \{(\hat{x}, \hat{y}) \in \mathbb{A} \times \mathbb{A} | min(\hat{x}) \geq max(\hat{y})\}$.
- $==$: For the equivalence relation using AAFs two types of equivalence are defined: *identical* and *equal*. The $==$ checks if two AAFs are identical. This means, that central values are equal and all deviations are equal (partial deviation values and associated symbols) $R_{==} = \{(\hat{x}, \hat{y}) \in \mathbb{A} \times \mathbb{A} | x_0 = y_0 \wedge x_i = y_i \forall i \in \mathcal{N}_{\hat{x}} \wedge \in \mathcal{N}_{\hat{y}}\}$.
- $\diamond =$: This is an AAF equal relation. Two AAFs are equal according to $\diamond =$ if central values are identical and worst-case deviations, their radius, is identical. Partial deviations and deviation symbols may be different (two AAFs are equal if they are identical). $R_{\diamond=} = \{(\hat{x}, \hat{y}) \in \mathbb{A} \times \mathbb{A} | x_0 = y_0 \wedge rad(\hat{x}) = rad(\hat{y})\}$.

- $\diamond$: The $\diamond$ relation is an overlapping relation. This relation can be defined using $>$ and $<$ relations of $\mathbb{R}$ applied on the AAF's bounds. $R_{\diamond} = \{(\hat{x}, \hat{y}) \in \mathbb{A} \times \mathbb{A} | (min(\hat{b}) < min(\hat{a}) \wedge min(\hat{a}) < max(\hat{b}) < max(\hat{a})) \vee (max(\hat{b}) > max(\hat{a}) \wedge min(\hat{a}) < min(\hat{b}) < max(\hat{a}))\}$.
- $\diamond\diamond$: The $\diamond\diamond$ relation defines that an AAF is completely enclosed by the other. $R_{\diamond\diamond} = \{(\hat{x}, \hat{y}) \in \mathbb{A} \times \mathbb{A} | max(\hat{x}) > max(\hat{y}) \wedge min(\hat{x}) < min(\hat{y})\}$.

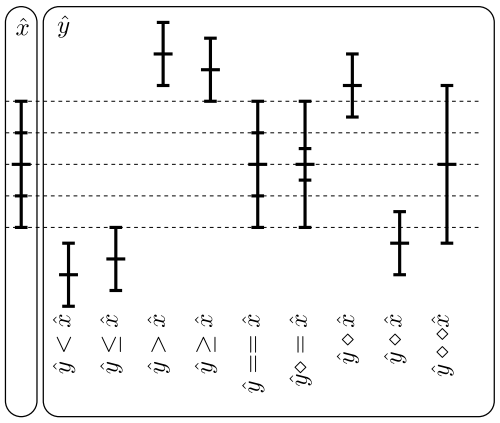


Figure 2.10: Graphical presentation of previously defined relations in $\mathbb{A}$.

These defined relations are often used in the form of an operator as $\hat{x} \oplus \hat{y} : \mathbb{A} \times \mathbb{A} \mapsto \mathbb{B}$ with $\oplus = <, \leq, >, \geq, ==, \diamond =, \diamond, \diamond\diamond$. These operators (and the associated relations) are visualized using abstract examples in Figure 2.10.

Handling comparison operators where the result is a binary statement are easy to evaluate and well implemented in the AAF framework presented by Darius Grabowski [Gra09]. The software framework implemented for this thesis offers the operators itemized above.

Very interesting but unfortunately significantly harder to evaluate are binary operators where the result is an AAF or an AAF is compared with a scalar (e. g. analog to digital conversion), $\hat{x} \oplus \hat{y} : \mathbb{A} \times \mathbb{A} \mapsto \mathbb{A}$. For example, the computation of a minimum form of two given AAFs, $min(\hat{x}, \hat{y})$, can not be reduced to the computation of $<$ operators (as we know that from numerical values). I address this topic and its implementation in Subsection 3.2.3 and in an example presented in Subsection 5.4.

2.4 Discussion of Modelling and Simulation Techniques considering Uncertainty

In this section I recap advantages, disadvantages and properties of uncertainty representations and calculation methodologies. The listed items are discussed specifically for electronic system modeling and simulation applications (as used in this thesis).

Already published library implementations for AAF based modeling and simulation

A basic version of an AAF library was implemented in [31]. The C++ library is based on the original AAF work from Stolfi and Figueired [SF97] and is open source licensed under GNU LGPL (Lesser General Public License). An extended implementation is presented by Darius

Grabowski in [Gra09]. It features enhanced functions for circuit and system simulation and is basis for further extensions. This library implementation is frequently used with the SystemC and SystemC AMS simulators.

[Sch13] presents a Fourier transformation algorithm for Affine Arithmetic forms. This enables system analysis in the frequency domain which is a significant enhancement for the analysis of telecommunication applications.

[GGB06, GOB08] presents a range based solver for circuits described on transistor level. Occurring uncertainty in transistor models may influence the full circuit behavior significantly. Differential equations are solved numerically. For non-linear transistor characteristics an approximation scheme adds additional deviation symbols to guarantee a enclosure of the potential exact result.

A building block library for communication systems is presented in [OSG11]. For analysis purposes, the authors use Affine Arithmetic forms for the consideration of range inputs and parameter variations. The building block library includes templates for signal sources, arithmetic components, filters, and channels.

One of the recently published extensions based on AA is XAAF (Extended Affine Arithmetic form) [RGJR17]. It extends the described AAF analog variability modeling with functions and methods for digital behavior. Discrete behavior is represented by using a constrained decision tree called AADD (Affine Arithmetic Decision Diagram). A basic description of AADD representations is given in Subsection 5.4.

For the implementation of the range based assertion-based verification methodology using SystemC AMS we used the AAF library of Grabowski [Gra09, RSRG12a]. This analysis method is further described in Section 4.4.

The AAF library implementation from Grabowski is a good basis for many AAF extensions implementations and modeling applications. However, for enhanced analysis requirements (especially deviation tracing) and functional expandability which is one of the main contribution of this thesis a reimplementation using new software structures is a valuable approach and detailed discussed in Chapter 3 and 4.

The following Table 2.1 holds requirements on modeling and simulation on the left side and discuss advantages(+)/disadvantage(-) of the presented methodologies considering uncertainty (multi-run methods, Interval Arithmetic, Affine Arithmetic forms) in columns.

Table 2.1: Summary of advantages and disadvantages of described simulation methodologies considering uncertainties.

	Multi-run approaches	Interval Arithmetic	AAF
Initial model modification	+ Simulation models used for functional verification can be identically used for uncertainty analysis. - Pattern generators have to be included in the simulation model and connected to the model's interface. - For stimulation of inner signals data path have to be cut and temporary connected to pattern generators.	+ Numerical models derived from functional simulation models can be used. Calculating worst-case outputs is reduced to numeric processing input bounds. - Scalar variables have to be replaced by a tuple representing bounds (tolerances).	- Numerical values represented by a scalar in original models have to be replaced by an appropriate compound AAF type. + AAF modeling forces designers for initial awareness of uncertainties and strict uncertainty documentation.
Computation time	-Simulation time which directly reflects the number of single runs, mainly depends on the number of uncertain inputs and the selected calculation accuracy. + The multi-run simulation run can be aborted after each single run.	+ An output interval is computed within a single computation of the model. Due to the reduced complexity of operations IA is the fastest simulation methodology within the discussed methods.	+ AA produces a range result within a single computation of the full model. - Due to the increased complexity of operations and enhanced information included in the AAF datatype a computation is in principle slower than using IA.
Mathematical operations	+ Operators are scalar values and thus all standard operations offered by the C ++ language can be used. + The result of an operator is in general a scalar value, which requires no approximation.	- Using intervals as a compound datatype requires redefinition of mathematical operations within the simulation framework. + Operations result in valid interval representations (however bounded or unbounded), and require no approximation forms.	- Affine Arithmetic forms requires a strict redefinition of all used operations. In principle, the exact center values can be calculated using default scalar operators. - Results of non-linear operations require approximated forms. Approximation types have specific properties which may be a basis for customized uncertainty analysis.

	Multi-run approaches	Interval Arithmetic	AAF
Traceability of uncertainty	+ At the inputs of a model specific generator represents an uncertainty uniquely. - Uncertainty is considered by varying input and parameter values sequentially. Uncertainty itself is not specifically modeled and thus obfuscated by the execution of multiple simulation runs.	- Uncertainty is explicitly modeled by interval forms. If ti comes to computation, intervals are potentially concatenated to single result intervals. The missing indication of subintervals reduces traceability features significantly.	+ In Affine Arithmetic each uncertainty is modeled individually and marked by a symbol. Partial deviations indicate portions of uncertainty which can be traced back to its origin by structural or temporal tracing procedures.
Representing uncertainty causes	+ A specific uncertainty is covered by run-wise sequential updating generator values. This represents a specific uncertainty cause including its statistical distribution. - The cause information is just present at model inputs. Worst-case behavior evaluated from detailed result analysis can be just matched to a specific model input valuation.	- Intervals values are driven by a specific uncertainty cause, but are not marked. Thus, intervals propagated through the model can not longer be uniquely identified and matched to an uncertainty cause.	+ Using Affine Arithmetic forms allows identification of represented subintervals to a specific uncertainty cause at any point in time. Within an AAF each partial deviation can be traced back to its cause by following the associated deviation symbol.
Tools for result presentation	+ Traces are temporary ordered sequences of scalar values which can be plotted by standard waverform viewer tools. - The number of traces for a single output is directly correlated with the number of selected simulation runs. For complex system structures and high accuracy this number can become extremely high.	+ An interval representation can be plotted by standard waveform viewing tools. Upper and lower bounds form temporal scalar traces representing worst-case system behavior over time.	- An Affine Arithmetic form is a compound datatype including subintervals which represent different uncertainty causes. Subintervals can be plotted individually. For a representation where deviations are symmetrically located around the center value Affine Arithmetic form have to be converted into a plottable form viewing subintervals in an accumulated style.

	Multi-run approaches	Interval Arithmetic	AAF
Management of the simulation process	- For multi-run simulation the sequential procedure of resetting the model, updating data generators and managing the total number of individual runs has to be managed by the simulation core. + Multi-run procedures may be easily implementable also for third-party simulators. External scripts may use specific simulator API functions managing the multi-run simulation procedure and read data from external files.	+ Uncertainty simulation using IA requires no specific management of the simulation procedure. Range results represented by upper and lower bounds are available after each model computation.	+ AAF based simulation in principle, just replaces scalar types as used in numerical simulation by Affine Arithmetic forms. Thus, no management and sequential execution of the model for a specific point in time is required. - For enhanced analysis processes using AAFs the AAF objects itself, creation of deviation symbols, etc. has to be managed.

2.5 System Analysis Approaches

In this section, I introduce and discuss state of the art system analysis methods. The proposed functions are partially already included in some model-driven development toolkits or even available as individual third-party software applications. However, the techniques selected for this section are somehow related to the analysis of parameter deviations which are potentially caused by uncertainties. I evaluate their applicability in the context of this thesis, highlight advantages/disadvantages and motivate potential enhancements.

2.5.1 Sensitivity Analysis

One of the original definitions of Sensitivity Analysis (SA) is that "Sensitivity analysis studies the relationship between information flowing in and out of a model". [SCS+00] refines this definition by dealing with uncertainties of input variables and model parameters with the goal to increase the confidence of the model and its predictions. Thus, SA is closely linked to uncertainty analysis [SCS+00, p.3]. According to [HG17] the major activity of SA is the identification of the importance of an input signal under consideration of its impact on the model's output. However, the selected context and specific measures for sensitivity assessment are required.

In general, SA is a well proven methodology has as a goal to determine the following selected items [SCS+00, p.6, p.9]:

- The factors mainly influencing the output variability and thus require enhanced verification effort to optimize the system's behavior.
- Parameters which have less importance and can be potentially omitted in following refinement steps.

- Evaluation of calibration input parameter values where low sensitivity measures result in maximum robustness.
- A set of parameters that are correlated and their variability effects interact each other.
- A measure of the quality of the model which is defined as the consistency between the model and the real system implementation concerning input changes.

A possible classification of SA processes given in [SCS+00, HPM17] and other publications is to divide them to local SA, global SA and screening methods.

Local SA
These methods determine the impact of inputs to the outputs by calculating first derivatives with respect to input variables. The approach is also called One-at-a-Time (OAT) because one input is changed while all others have a constant value. Mathematically this is expressed as: $O_i = (X_i/Y_i) \cdot (\partial Y/\partial X_i)$. This is a relative measure definition of Y caused by a change of X_i by a fixed fraction of X_i's central value. The partial derivation values may constitute a comprehensive sensitivity matrix $\mathbf{S}$ of the system. This measure is applicable if the variation around the input central value is small (e. g. $\pm 5\%$). Otherwise, it returns insufficient results if the model's I/O characteristic is highly non-linear. The evaluated OAT measure is erroneous if the system's output is affected by combinations of uncertainties representing correlation effects [HPM17],[HG17],[SCS+00, p.10, p.81].

Global SA
Global SA methods evaluate the impact to an output over the full range of possible input variation. Global methods are often used in combination with a model derived from DoE and the input variation is described under a specific probability density function [SCS+00, p.11, p.101]. Thus, global SA is able to evaluate effects of an input X_i while all other inputs $X_j, j \neq i$ are varied as well. These methods are in general complex and computationally expensive (potential large number of simulations) for reaching an appropriate sampling density [HPM17].

Screening methods
Screening methods are highly applicable for systems having a high number of inputs and thus a high computational complexity. The idea of screening is to identify a subset of dominating inputs. This identification process is highly based on experience knowledge and "trial and error" processes, finding out the often low number of significant inputs [SCS+00, p.10]. A more structured method presented in [HPM17] is the *Morris method*. The evaluation of a multidimensional semi-global trajectory allows efficient identification of influential inputs within a low number of simulation runs [HPM17].

In Figure 2.11 the introduced SA methods are evaluated under the perspective of the system complexity and required simulation runs.

For a conclusion of this subsection, I discuss the main advantages and disadvantages of Sensitivity Analysis as a specific analysis process considering uncertainty:

+ Local SA can be assessed with relatively low computational effort. If the mathematical relation between inputs and outputs is known as a function the derivatives can be evaluated analytically (the reason why 0 simulation runs is included in the gray area in Figure 2.11). If a black box style model has to be analyzed the OAT method is a low effort way to estimate the output reactions in principle if inputs deviate from their nominal value.

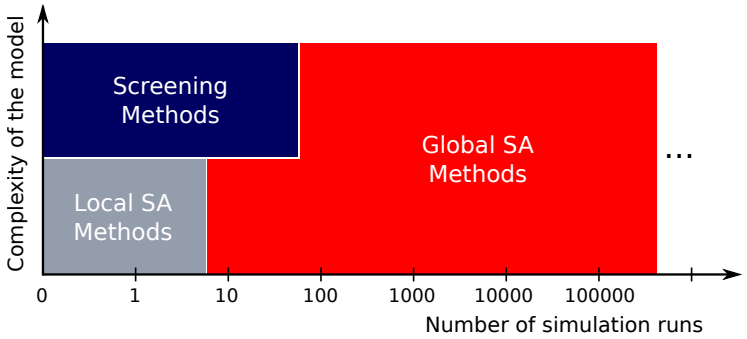


Figure 2.11: Assessment of sensitivity analysis methods under the number of simulation runs and the complexity of the system model [HPM17].

+ Screening converges fast to get a rough impression about the general reaction and the direction of this reaction if a system parameter starts slightly drifting.

+ Global SA delivers good and confident results based on statistical methods and a high number of simulation runs.

+ For standard AA, where only linear deviations are considered, SA defined as first order derivatives, result in corresponding partial deviation values. Thus, local SA is "cost-free" at using AAFs for uncertainty representation.

+ SA is well embedded into industrial devlopment tools. For example: Cadence Virtuoso Analog Design Environment [5], Synopsys Platform Architect [20], and many more.

- If the I/O characteristic is highly non-linear local SA may return very inaccurate results.

- Correlation effects, where uncertainties are not independent from each other, has to be included accurately in the modeling procedure first, before they can be considered in a subsequent SA process.

2.5.2 Stability analysis

If a system is influenced by uncertain system parameters for sure the question of "is it possible that the system gets unstable under the given parameter variations?" arise. This issue is highly related to Sensitivity Analysis. Potential approaches are to extend classical stability analysis as applied in circuit and system theory. Published research articles focus in interval extensions of the Lyapunov stability theory [MMC11, chapter 6],[KPM04] or use Hurwitz polynomials [WH07]. A further interesting challenge in stability analysis is the consideration of time-varying delays within the system structure. The presence of time delays is significant for proper reactions given by feedback loops (e. g. in control systems) and thus may result in catastrophic instabilities and oscillations [KKK11].

2.5.3 Assertion-based Verification

ABV (Assertion-based verification) is a state of the art technology used for design verification through all development phases of a system. It is highly embedded in standard industrial design

flows and tools support comfortable formulation, management, checking and reuse of assertion statements.
Originally ABV was mainly used in software for many decades and first published in 1947. Thus, from there on nearly every programming language features a more or less powerful assertion statement [Jon03]. The main idea is not to use the **assert** primitive to catch invalid arguments of functions as a replacement of robust programming procedures. But, they should be used in the sense of debugging and verification purposes to instrument the piece of code under test [BZ10].
Using assertions also for model-driven hardware development faces several advantages, and it has been publiched in 2007 that already 68% of engineers use **assert** statements for efficient verification of their hardware designs [27].
A significant advantage of ABV is that a created assertion database can be used for verification of the system at the full design process, from specification to operation phase as illustrated in Figure 2.12.

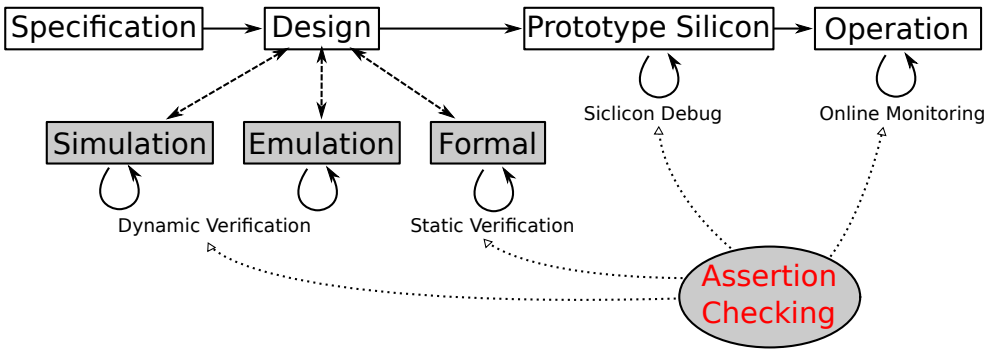


Figure 2.12: Assertion checking for system verification during the full development and operation life cycle [BZ10].

As shown in the Figure 2.12 during the design phase assertion checking may be applied as static or dynamic verification methods. Dynamic means that the model is integrated into a simulation or emulation environment and specified test cases are applied. Expected results are checked by evaluating prior specified assertion formulas. In contrast to dynamic methods, formal algorithms prove the satisfaction of assertion statements on a given system model mathematically (see Section 4.9). Thus, the specification of test cases is not required which is one of the major advantages of formal assertion checking (property checking). However, formal algorithms face an exponentially rising complexity problem for large system structures. Dynamic, simulation-based ABV, which is also used in this thesis, is predominantly used in the industrial design flow. The right side of Figure 2.12 indicates that assertion checking is not limited to the design phase. Based on assertion synthesis algorithms as presented in [BZ10] and Section 4.10, hardware assertion checkers can be integrated into the design for test execution during system operation.

Property specification language (PSL) is the IEEE standard for formulating assertion formulas [PSL10]. For hardware verification the PSL assertion standard got integrated in various hardware modeling and design languages (VHDL IEEE 1076, Verilog IEEE 1364, SystemVerilog IEEE 1800, SystemC IEEE 1666). PSL can be used either for dynamic and static verification. The abstract structure of PSL is organized by four layers (from lower to higher). A Boolean layer contains Boolean expressions as logical operators, bit expressions, clock declarations, etc. The temporal layer is most important to describe hardware behavior. It extends propositional logic by temporal operators which enables the formulation of timing behavior. Thus, PSL in-

cludes LTL (Linear-Time Temporal Logic) and CTL (Computation Tree Logic) operators as **always, next, never, until, etc.** [PSL10, Lam05]. A verification layer is used to define directives for the used verification tool. In PSL beside **assert** also other verification directives as **assume, restrict, cover,** etc. are defined. Last, a modeling layer is used to define the behavior of design inputs (e. g. value ranges, types, etc.) [PSL10].

In the following itemization, I discuss advantages and disadvantages of assertion checking:

+ ABV is easy to integrate into a system design flow and available in various state of the art tools. Also, the design tools support the management of assertions, functional coverage assessment as well as tool functionality for dynamic and static verification processes [14]. As a result, this reduces verification time, increases and focuses the design effort and helps to find errors and bugs in an earlier design stage [39].

+ PSL is an IEEE standardized language. According to [PSL10] PSL has a high expressive power and enables specification of a large class of real-world properties.

+ Properties can be also formulated in a more abstract semi-formal way. A semi-formal property specification uses logical and temporal operators in combination with data and signal descriptions formulated in natural language. In a further step, these semi-formal properties are refined using specific variable and signal names defined in the design under test. Semi-formal properties can be assigned to a specific application, stored in a company's knowledge database and reused over several design projects. In [RSK14] and [RS13] I used a database of malicious circuit properties to check designs for hardware Trojans. In [RHKP15] I use semi-formal property formulations for verifying business processes.

+ According to Figure 2.12 for hardware designs, assertion statements can be synthesized to runtime-verification hardware checkers. These benefits built-in self-test strategies applied during operation of safety-critical applications. According to [BZ10] these synthesis algorithms (translation into FSM checker automats) are well known.

- Dynamic ABV requires the specification of test cases, including test setup, test operation and result checking processes. Thus, assertion formulations may be highly dependent on a specific test case stimuli. If assertions fail two possibilities which have to be further inspected manually arise. First, the lack of proper stimulus or second, evaluated system misbehavior fails an assertion statement [BZ10].

- Primary outputs, and especially for more fine-grained verification also internal signals must be available/accessible [Lam05]. For black-box IP where this is possible, back-tracing of errors to their cause may be a challenging task.

- Assertions represent a "golden rule" manually derived from the system specification [BZ10]. This step may cause errors and/or uncertainties. An engineer has to deal with questions like "How many assertions do I have to write?", "Is a requirement fully covered by the set of defined assertions?", etc.

So, ABV is well published and used for numerical system simulation and verification. We follow up these ideas and extend ABV for input ranges which is presented in Section 4.4.

2.6 Related Work

The topics discussed so far in this state of the art section are highly related to the main contribution of this thesis. The following research fields can be seen as cognate disciplines but potentially interesting for the context of this thesis.

Design of Experiments
The Design of Experiments (DoE) method is related to the theory presented in Subsection 2.2.1. In principle, a metamodel is constructed based on experimental data. The model structure and its parameter set are evaluated from measurements on already realized systems or well-founded simulations. Basically, for the description of input/output relations linear first order models and quadratic polynomials for non-linear effect are used. In publications, DoE is used for sensitivity analysis, worst-case analysis and further model-based verification [Kle08]. If second-order estimates are not considered the DoE model is in principle describing the equal behavior as an AAF model. The main difference of the DoE approach to IA and AAF based models is that DoE constructs a stochastic behavioral model while other approaches operate on deterministic nominal models which expand an exact behavioral description by deviations [Kle08],[SCS+00, p.51].

Checkpointing
Scientific simulations of complex systems may become very computationally expensive and thus long-time running. Checkpointing is a concept where the full state of the simulation is stored periodically. In the case that a long simulation process unpredictably terminated during execution, it can be recovered and continued at the last checkpointed simulation state. However, it is not a trivial task to determine the time interval between continuous checkpoints [XHZ12]. A further application of checkpointing approaches is parallel discrete event simulation. The full model is partitioned into a set of logical processes (LPs). LPs are distributed on multiple simulation cores where event notification messages including the simulation context and a simulation timestamps are shred. If a time-stamp order violation is detected, the simulation has to be rolled back to the last checkpoint where values are correct. This mechanism provides an enormous speed up caused by parallel execution of LPs in contrast to classical single core simulation [QS03] [Fuj99, p.122]. A potential application for checkpoint and return simulation approaches for uncertainty simulation is to use the process in a multi-run perspective. If the impact of a modeled variability affects a selected internal signal, a checkpoint is set. The simulation is sequentially rolled back and further output values are computed resulting in a coverage of the full variability range.

Symbolic simulation
Symbolic simulation is a formal verification procedure. The main idea is to replace specific numeric valuations by symbols. Thus, a comprehensive verification coverage can be reached within a single or very few simulation runs. The origin of symbolic execution of pieces of code is in software testing in the 1970s. The procedure was then adapted for checking models describing abstract (hardware) system behavior. [Her16] presents a complete symbolic simulation of SystemC models by extending a symbolic simulator with a state based search (model checking) for the evaluation of cycles. This approach allows effective formal verification of the specified model against safety properties, deadlocks, etc. However, formal verification has a large potential in general and will significantly affect future verification tools (also for industrial applications) [Dre17, p.155],[16].

Formal methods and probabilistic model checking
Model checking (or property checking) is a formal verification technique based on models of system behavior and properties, specified unambiguously in formal languages. The behavioral model of

the system under verification is often specified using a Finite state machine (FSM). The properties to be checked on the behavioral model are formulated in a Propositional Temporal Logic (PTL). Eligible tools operate on a formalized finite transition representation (M) and check whether such a property (p) holds on this model ($M \models p$) [Kro10, Bie09]. Model checking does not require the execution of the model within a simulation framework. Thus, also no testbench and test case specification is needed. Algorithms analyze the model itself and result in a mathematical proof whether the formulated formal property is satisfied by the model [Kro10, Dre17].

In the perspective of formal model checking in combination with uncertainty analysis, probabilistic model checking is interesting. Similar to the technique preliminary described in the last paragraph probabilistic model checking is a technique for checking qualitative characteristics, but in this case it is a stochastic system model (e. g. Markov decision processes, stochastic multi-player games, etc.) [Dre17]. This has the consequence that possible states, potential transitions including uncertain timing of transition occurrences, etc. can be specified using probability values (e. g. the probability of a system error occurring with the next 5 ms). The set of operators is extended to Probabilistic Propositional Temporal Logic (PPTL) (e. g. s satisfies $P_{\bowtie p}[\psi]$ if the probability of taking a path from s satisfying ψ is in the interval specified by $\bowtie p$) [Dre17]. A popular open source tool used for various scientific publications is the PRISM [35] model checker.

3 Affine Arithmetic Framework with Enhanced Features for Traceability

In this chapter I describe the C++ based AAF framework, called *SESYD* framework (Semi-Symbolic System Discovery), implemented for this thesis. In the following subsections, I discuss functional requirements for range based calculations and enhanced analysis features with a focus on deviation tracing. For satisfying these requirements, I present the *SESYD* approach using an object-oriented software architecture. Properties of the implemented framework are compared with already published AAF libraries (see Section 2.4). However, the *SESYD* framework is a modified, revised re-implementation based on the work form Darius Grabowski [Gra09]. The following subsections discuss, why it's worth to re-design this framework using new data structures and a modular approach especially for enhanced tracing, analysis features and customized expandability.

3.1 Framework Architecture

Based on models and associated simulation methods for considering uncertainty as presented in Chapter 2, I identify the following functional software requirements. These requirements are used as a basis for the implementation of a simulation framework and associated enhanced analysis features:

1. Clear framework structure including a documented Application Programming Interface (API).
2. Expandability using a modular software architecture. Complexity is manageable and represented by a clear set of hierarchically categorized subfeatures.
3. Adaptability of analysis processes for user-specific verification depth and optimized simulation runtime.
4. Clear documentation of all implemented features and models for cross-project reusability.
5. Applicability and ability to integrate various modern (commercial) simulation cores.
6. State of the art implementation language in a way that the framework sources can be compiled for various platforms.

7. Abstract modeling scheme so that methodologies can be applied to any kind/field of simulation application.

In the following sections, I discuss the itemized requirements in detail and describe their impact on the framework design.

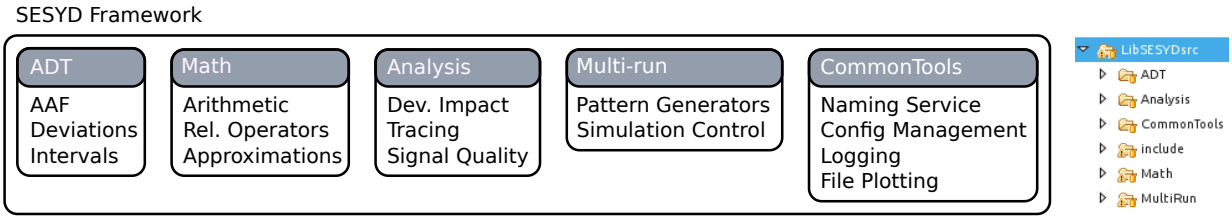


Figure 3.1: *SESYD* framework overview.

For the framework implementation I tried to use a software architecture which can be easily enriched by user-specific modules. Features are partitioned as illustrated in Figure 3.1. Each shown box represents a module including a set of subfeatures. *ADT* (Abstract data type) provides template classes for representing Affine Arithmetic forms, deviations, deviation symbols and intervals used as specific datatypes for representing uncertainty (see Section 2.2). The *Math* module defines arithmetic $(+, -, \cdot, /, \ldots)$ and relational $(<, \leq, >, \geq, ==, \ldots)$ operators required for simulation purposes. This is achieved by C++ operator overloading. In addition to operator redefinitions, the module holds multiple algorithms for approximating non-linear Affine Arithmetic operations (see Subsection 2.2.4). The *Analysis* module is one of the most important implementation contributions described in this thesis. Enhanced algorithms for uncertainty impact analysis, assertion-based verification, tracing, and deviation metric evaluation are included. These system analysis processes, enabled by detailed and straight modeling of uncertainty using AAFs, return the most enhancing result towards an increased system insight into the implemented system model. The *Multi-Run* module extends SystemC simulation core functionalities by multi-run methods (Monte Carlo, Worst-case, etc.). Features of this module are various test pattern generators and core classes for multi-run simulation control. The *Common Tools* module includes auxiliary functionalities and submodules for usability with a good performance (e. g. configuration file management, logging information and plotting simulation results having a datatype specified in *ADT*, etc.). The rightmost part of Figure 3.1 shows the implemented folder structure of modules in the *SESYD* eclipse project workspace.

The functions mainly described and discussed within this thesis are focused on representing uncertainties by Affine Arithmetic forms and associated analysis features. Enhanced analysis methods as tracing, detailed impact analysis, etc. rely on extended uncertainty information included in Affine Arithmetic forms. However, the implemented *SESYD* framework is designed to be expandable for new/other uncertainty representations and their associated analysis algorithms.

A further advantage of the presented *SESYD* framework is a user adaptable setup of analysis features applied during a simulation run. Detailed analysis highly consumes simulation runtime, memory load, and hard disk space. The adaptability of the *SESYD* framework enables specific and on the fly en-/disabling of analysis processes (e. g. tracing of signals, assertion checking, calculation of metrics, etc.). Also for the computation of approximation forms, a similar concept is implemented.

Some modules of the framework are mandatory for the basic operation (e. g. naming service, symbol management, logging, etc.). These are specifically initialized by calling the

void initSesydFramework(void) function. Due to tight functional dependencies the following modules must not be instantiated multiple times (in the code they are defined using the singleton software design pattern): logger, ID generator, deviation symbol manager, tracing module, monitoring module, multi-run management.

The full *SESYD* framework is implemented in the object-oriented programming language C++. The framework architecture and proposed features are significantly influenced by the object-oriented concepts of the C++ programming language. However, this offers several performance and applicability features (see Sections 3.6 and 3.7). A major characteristic of my implemented framework is that "everything" is handled as an object having an unique ID and a name. This means that instantiated functional components and data structures (AAF, deviation, trace, assertion, logger, etc.) are represented by a corresponding C++ object. During the simulator execution objects are located in the user memory space. Corresponding pointers are handed over to object management and analysis modules.

3.1.1 Object Oriented Representation of Affine Arithmetic forms

One of the central data structure in the *SESYD* implementation is the class representing an Affine Arithmetic forms.

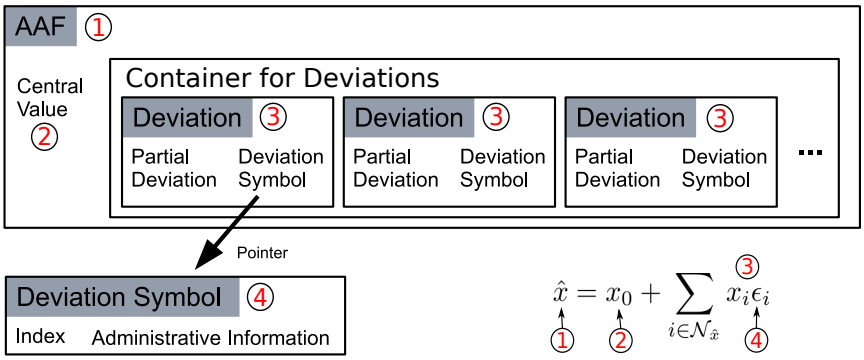


Figure 3.2: *SESYD* framework overview.

Figure 3.2 illustrates the mathematical definition of an AAF (see equation 2.2) and its mapping to software objects implemented in the framework. Thus, an **AAF** instance is in principle a compound object including **Deviations** and **Deviation Symbols**. The central value x_0 (2) is mapped to a single double-type variable. A deviation $x_i\epsilon_i$ (3) includes a scalar partial deviation value and a pointer to a **Deviation Symbol** object(4). A **Deviation Symbol** representing an ϵ_i (4) contains an automatically assigned index variable (see Subsection 3.1.2) and may hold some additional administrative information further describing the modeled uncertainty (e. g. location, level, nature; see Subsection 2.2.2). In an implementation perspective, it is significant that just one pointer (reference) to a **Deviation Symbol** object is included within a **Deviation**. Thus, deviation correlations can be realized. Multiple deviations in various AAFs may point to the same **Deviation Symbol** objects.

An AAF may include multiple deviations (symbol and a scalar partial deviation value) as given in the basic definition (equation 2.2) Thus, **Deviation** instances have to be collected within an appropriate container. During a simulation run the number of **Deviation** in the container is in general not constant. This has the consequence that the container size has to be potentially

adapted. [Gra09] uses for deviation storage a low level C-implementation. As shown in the left part of Figure 3.3, partial deviations and their indices are stored in two expandable array structures. Strictly speaking, within the AAF class two pointers indicating the start address of the arrays are held. If a partial deviation is added (e. g. by a non-linear mathematical operation approximation) and the size of the initially allocated memory space is not longer sufficient, completely new arrays are constructed, content is copied, pointers are overwritten, and the old container is deleted. For partial deviation lookup, the full array structure has to be scanned sequentially. Management functions for pointers, memory spaces and length variables are implemented manually by the developer.
In my framework as shown in the left part of Figure 3.3 I use the standard C ++ 11 **unordered map** STL (Standard Template Library) container type [25]. Within this container partial deviations (strictly speaking deviation objects - see Subsection 3.1.1) are unordered and represented by a pair of a key element and associated data (the deviation object). For internal container management, a bucket is used which is identified by a hash value. This enables a fast (hash-based) access and lookup of **Deviation** objects [25]. In contrast to the approach using array structures, container access and management functions are predefined by the C ++ 11 STL and not a part of the developed code.

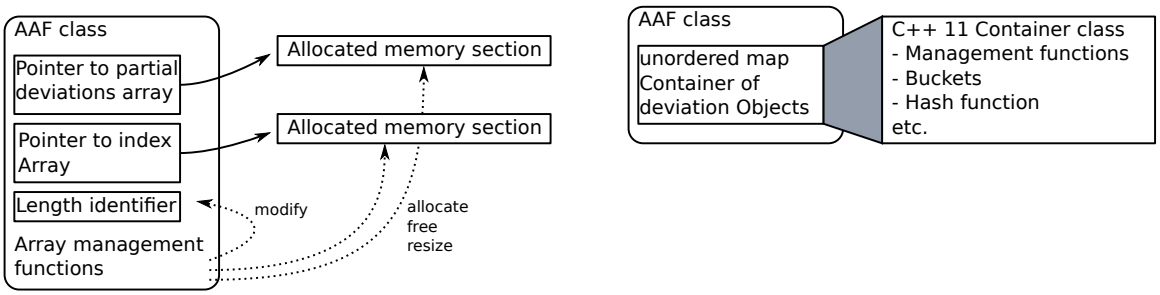


Figure 3.3: Approaches for storing partial deviations in an AAF class.

The efficiency of this **Deviation** container and associated access methods is primarily influencing the effectiveness of the full framework. During development also other container types were tested. Specific performance impacts are discussed in Subsection 3.6.

According to an object oriented software concept the AAF class holds data as shown in Figure 3.2 and associated functions in combination. In C ++ mathematical operators (e. g. $+, -, <, \leq$, etc.) may be overloaded and redefined with corresponding operator symbols but accepting AAF datatype. The following itemization holds a selection of functions and operators implemented in the *SESYD* framework (accepting AAFs, and AAFs in combination with scalar types):

- **Basic mathematical operators:** +,-,*,+=,*=, etc.
- **AAF specific operators:** Radius, Maximum, Minimum, Conversion functions to intervals, etc.
- **Relational operators:** $>, >=, <, <=, \ldots$
- **Approximation functions:** Computation of approximation results, check if a form is an approximation, etc.
- **AAF management functions:** Add a deviation, Rename, Cleanup, GetID, getName, etc.

- **Verification and Debugging:** Dump AAF content, various print functions, write an AAF content into a waveform files, etc.

A detailed list of all implemented functions and operators and their associated descriptions can be found in the doxygen documentation of the AAF class.

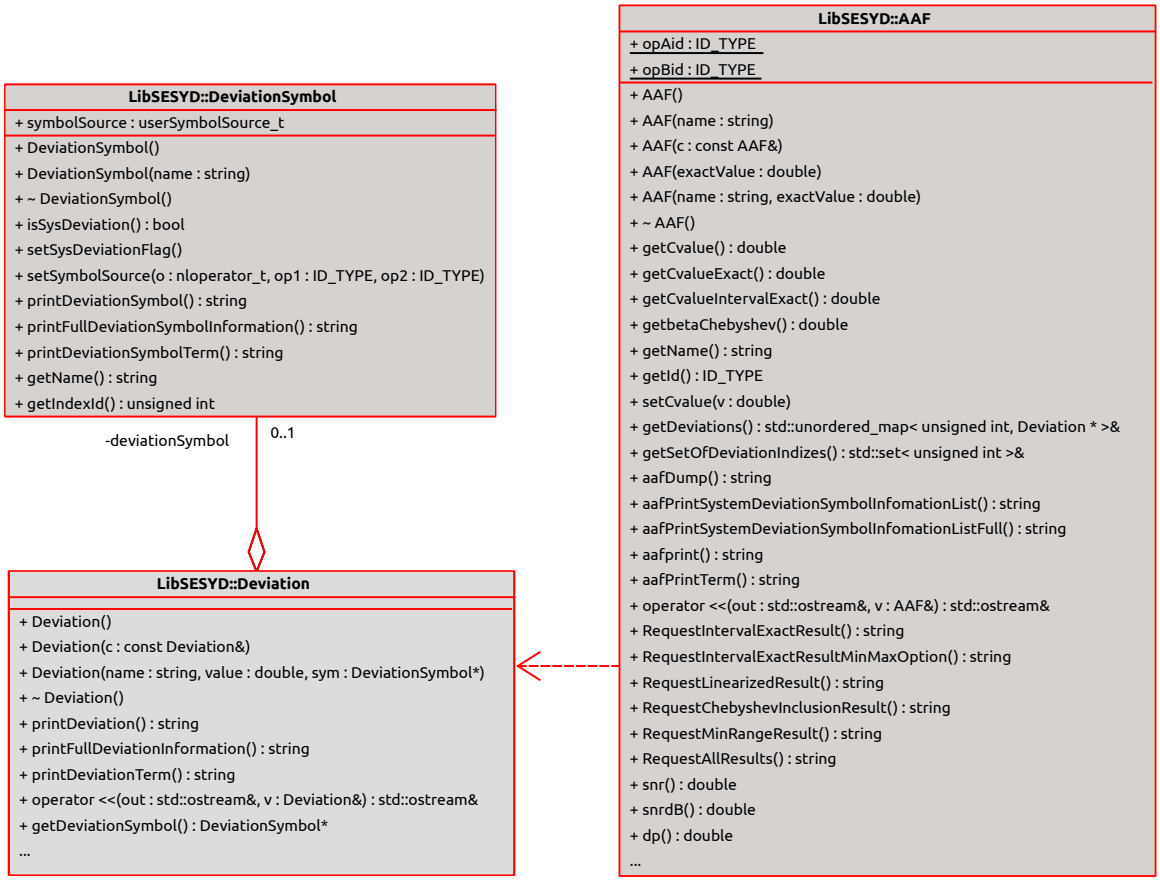


Figure 3.4: Class diagramms of the implemented AAF, Deviation and Deviation Symbol objects (AAF and Deviation are not completely shown. For comprehensive details see the Doxygen code documentation)

Figure 3.4 illustrates the class diagram of **AAF**, **Deviation** and **DeviationSymbol** implementations. In this picture **AAF** and **Deviation** shows just a selected subset of included methods. For further details please check the doxygen code documentation.

In general, representing simulation data by a class structure, with its tight conjunction of data and associated methods is very useful. Thus, Affine forms can be structurally represented, collected in containers, used as a base class, etc. In the work of Grabowski [Gra09] only the AAF type is declared as an object. Partial deviation values are represented as a single scalar value and symbols are not modeled at all. In the *SESYD* framework I followed the concept that "everything" is a software object (AAFs, intervals, monitors, traces, assertions, etc.) So I use class encapsulation of an AAF and all included subparts. Also, deviations and deviation symbols are classes which can hold additional information besides its scalar value (e. g. further values representing approximations (see Subsection 3.2), pointers to other objects, a container of references

to other objects, etc.). In a real-life industrial perspective additional information can be used for enhanced deviation documentation as well as functions for debugging and test procedures can be added. Disadvantages are given by an increased memory footprint caused by the additional overhead. However, this is not a substantial problem for state of the art simulation platforms where several gigabytes of memory are available. If the implemented framework might be migrated to resource constraint in-filed platforms where simulation runs are executed during operation of a device this issue may become critical.

3.1.2 Symbol Management

As described in the last subsection a specific deviation symbol ϵ is represented as an individual software object. AAF instances including an unordered map container of **deviations**. Each deviation holds one references to a **symbol** object. Pointers to deviation symbol instances are returned from the C++ **new** operator. These pointers are stored in the corresponding deviation object and additionally handed over to a symbol management software module.

In the **symbol manager**, pointers to all symbol instances created during a simulation run are registered and collected. In essence the deviation symbol manager holds a vector of pointers to all instantiated deviation symbols (see Figure 3.5). As described in Subsection 2.2.4 a deviation symbols represents a specific deviation cause. So the mentioned vector reflects all uncertainty causes modeled within the system. A specific cause, resulting in a behavioral impact on the system (e. g. temperature, jitter, voltage, etc.), is unique in the physical domain. Thus, the C++ symbol class is implemented using the singleton design pattern. Due to consistency reasons, symbols must not be uncoordinatedly created and destroyed by a user or other objects. The **symbol manager** also manages the creation and destruction of symbols, by offering appropriate request and delete methods.

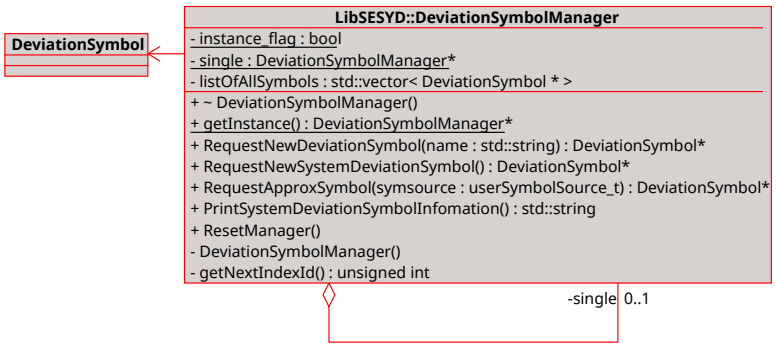


Figure 3.5: Class diagramms of the implemented **DeviationSymbolManager** module.

Figure 3.5 illustrates the class diagram of the deviation symbol manager module. It is implemented using the singleton design pattern because multiple instantiations of managers are not allowed (consitency reasons). The instantiation of the manager is included as a subfunction in the framework's **void initSesydFramework(void)** method.

The central function for the mentioned coordinated creation of new deviation symbols is **RequestNewDeviationSymbol**. If the user wants to add a new deviation to an AAF, the constructor of the deviation object places a symbol request at the manager. Single parameter for this associated request function is a name variable appointing the uncertainty (e. g. chip temperature variation, offset voltage drift, etc.) If the deviation symbol is already existing (name equivalence

check) in the system a reference to the corresponding already existing **DeviationSymbol** object will be returned. Otherwise, as described a new symbol would be instantiated and added to the management vector.

Similar to collecting deviations within an AAF the question of "is the **std::vector** C++ STL container sufficient for collecting deviation symbols" may arise. First, in contrast to deviations the number of symbols, reflecting physical uncertainties, is constant after elaboration of the model. Second, according to [26] vectors have a good performance in adding and removing elements at the ends. Thus, creating new elements during runtime (deviation symbols caused by non-linear mathematical operations) will perform efficiently. Third, for checking if a deviation symbol is already in the container an exhaustive container lookup has to be done. Therefore, vectors are highly efficient in sequential accessing elements (similar to arrays) [26]. As a result of this manual analysis of execution use cases, I argue that **std::vector** is an appropriate and sufficient container for the deviation symbol manager module. During the process of this thesis, I did no evaluation of other container types for the symbol manager.

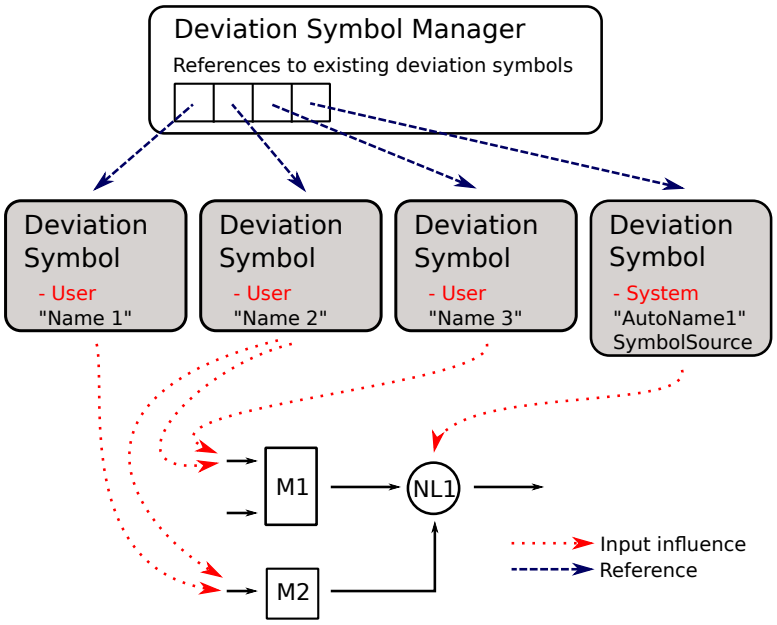


Figure 3.6: Overview over implemented types of deviation symbols and the given relation to modeled uncertainty causes.

Figure 3.6 illustrates the symbol manager, four symbol objects and a small system model (including two modules and a non-linear operation). Blue dashed lines show that references of deviation symbols are registered and stored in the vector container. Red dotted arrows indicate that the modeled deviation symbols represent a physical uncertainty stimulating at inputs or resulting in uncertainty caused by non-linear AAF operations. The second deviation symbol ("Name2") influences the upper and the lower inputs of the system. Thus, corresponding input Affine Arithmetic assertions and deviations are associated with this symbol which models correlated uncertainty. Further illustrated, in this implemented AAF framework two different types of deviation symbols are existing.

- **User deviation symbols** are created by the user (verification engineer or the person creating the model). Each time the **RequestNewDeviationSymbol** method is called as

already mentioned a new user deviation symbol is created (except the symbol is already existing). User deviation symbols have a human readable name and description.

- **System deviation symbols** are added due to the execution of non-linear affine operations [Gra09]. Their creation (call of the request symbol function of the symbol manager) is integrated in the corresponding approximation algorithm (see Section 3.2). However, system deviation symbols have an auto-generated name and a **SymbolSource** variable. For a non-linear operation, a **SymbolSource** variable holds the object identifiers of the operands and an identifier for the applied operation. This is stored in a binary expression tree structure. A defined **SymbolSource** variable is used for evaluation if a user deviation symbol for a dedicated non-linear operation is already existing. Thus, the corresponding check is based on a tree equivalence matching algorithm.

To distinguish user and system deviations a Boolean flag is included in the **deviationSymbol** class (see Figure 3.4). (Note: A potentially more efficient approach could be the creation of a base class, and deriving separate classes for user and system deviation symbols.) The presented approach of having two types of symbols enables a user-friendly uncertainty documentation. Also additional information for location, level, and nature as described in Subsection 2.2.2 can be integrated. For a system deviation, the cause (given by non-linear operations) can be evaluated by manual analysis of **SymbolSource** identifiers.

A known issue of the AAF framework published in [Gra09] addresses the discussed identification if a deviation symbol is already existing or not. As mentioned earlier in [Gra09] a deviation is represented by a single scalar value in combination with an associated array index. After a non-linear operation, consequently a new symbol is created.

In system simulation applications sequential non-linear affine operations as loops of periodically equal operations at each timestep may be included. For example in modeling a finite impulse response (FIR) filter having deviated filter coefficients. For this case, a checking functionality as implemented in the symbol manager guthat guarantees that only one extra deviation symbol (source is always the same) is added independently of the number of executions. In a conservative approach adding a new symbol at each execution the number of included deviation symbols and associated partial deviations will explode. In [Gra09] this is solved by sequentially calling a cleanup method, combining small partial deviation to a single one.

In contrast to the framework implemented in [Gra09] where deviation symbols are not modeled explicitly as objects, for the *SESYD* approach I identify and discuss the following listed advantages:

+ Enhanced traceability of symbols from a specific represented cause to their impact in single Affine Arithmetic forms is provided (see next subsection).

+ Deviation symbols caused by non-linear operations are marked by a specific boolean flag. Thus, for a given AAF (e. g. representing the output of a system) a verification engineer can easily evaluate if an included deviation is caused by a physical uncertainty effect, or caused by approximation of an included non-linear operation within the system (see aso Section 2.3.3).

+ Deviations and associated deviation symbols can be used in an independent way, which has the concequence that customized object to object linking may offer enhanced information for analysis and debugging.

+ Additional information can be added to a symbol specification (location, level, nature, etc. as introduced in [WHR+03] and discussed in Subsection 2.2.2). Representing a deviation symbol as a class allows enhanced uncertainty documentation, including additional information for subsequent analysis.

+ For further enhancment of semi-symbolic simulation methods, new symbols can be defined easily and integrated to the symbol manger's functions (e. g. Introduction of XAAF ω symbols [Rad16]).

- A significant disadvanatge is given by the runtime and memory overhead of the symbol manager. Browsing the containers and keeping track of all registered symbols is slowing simulation. From a verification perspective the additional information about the system's inner behavior is a dominating added value for subsequent analysis and optimization.

3.1.3 Basic AAF Traceability Functionalities

Basic uncertainty traceability information is by definition already included in Affine Arithmetic forms. Deviations and their associated symbols representing a physical effect can be monitored during a simulation run. Figure 3.7 illustrates a system model with two functional blocks and associated affine signals (at one selected point in time) $\hat{a}, \hat{b}, \hat{c}, \hat{z}, \hat{o}$. AAFs enable to trace specific impacts of the two included physical causes, to all instantiated signals by reminding partial deviation values (for ϵ_1 and ϵ_2).

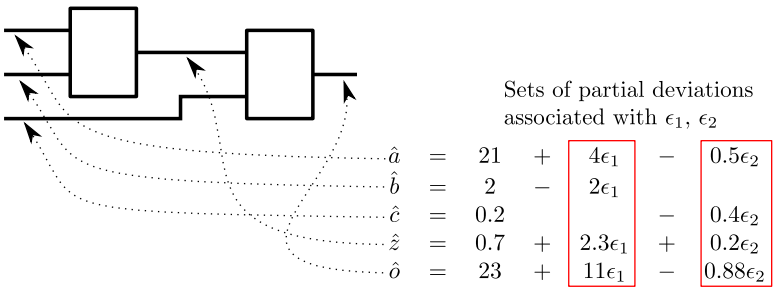


Figure 3.7: Basic deviation tracing is by definition included in the AAF modeling methodology

The described management of symbols and their representation as C++ objects enable efficient basic traceability of uncertainties. The proposed methods are described as basic features given by the implementation of the symbol manager. Enhanced traceability features and associated analysis algorithms are described in Sections 4.5 and 4.6. As previously mentioned deviation symbol information including the index of the deviation symbol will never change during simulation runtime. Naming and user-definable additional information as location, uncertainty effect, etc. provide tracing of deviations in an AAF object. Optimization and cleanup algorithms which delete deviations of an AAF must not delete instances of deviation symbols.

In the following, I define two tracing methods which are featured by the behavior of the symbol management module:

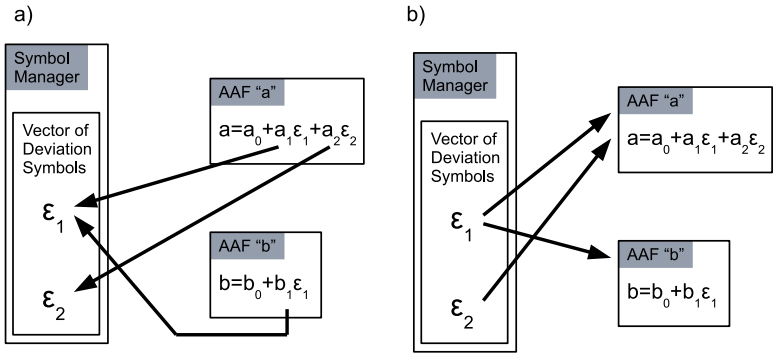


Figure 3.8: a) Forward tracing: Starts at the deviations and links to a deviation symbol. b) Backward tracing: Starts at the deviation symbol and links to all referenced affine forms.

Forward tracing as illustrated in Figure 3.8-a is defined as a pointer link between a deviation and a deviation symbol. Unused symbols are still kept in the symbol manager's vector. This is highly related to the integration of a deviation symbol reference into an AAF (see Figure 3.2). In a higher level abstraction view within this method, tracing is given as the ability to follow partial deviation contributions of a specific AAF to underlying uncertainty causes.

The second tracing method associated with deviation symbol management is the vice versa direction called ***backward tracing*** (see Figure 3.8-b). As described deviation symbols objects may not be constructed by the user, they are requested at the symbol manager which constructs them in a coordinated way. Whenever a symbol is requested a reference to the requesting AAF object is stored in the deviation symbol itself. So a deviation symbol holds references to all AAF objects it is included in (see Figure 3.8-b). This allows tracing ability starting at the deviation symbol representing an uncertainty cause back to its containing AAF objects. Strictly speaking, a deviation symbol holds a list of Affine Arithmetic forms indicating initial (user specified) impacts of uncertainty.

The deviation symbol manager provides a function for writing a full information listing including all deviation symbols into a C++ output stream. This stream containing dumped symbol information can be forwarded to a file, stringbuffer, console output, etc. Listing 3.1 shows a snippet of such a dumping output. The printed symbol has the index 1 and a user-defined name and description is set for deviation documentation. ϵ_1 is a symbol which is associated with a deviation added by the user calling the **addDeviation** method. For forward tracing, the dump function of an AAF prints all included deviations and their name parameter of the included deviation symbols (strictly speaking symbol references). In a backward tracing perspective, it can be seen that the deviation symbol ϵ_1 is included in the AAF objects named **aaf1**, **aaf3** and **aaf7**.

Listing 3.1: Snippet illustrating the list output of all deviation symbols

```
Deviation Symbol Manager:
FULL SYMBOL INFORMATION LIST
******
Symbol e1 -
Name: deviation effect temperature
Description: Ambient temperature deviation of module 1
Is USER deviation
Symbol is referenced by:
- aaf1
- aaf3
- aaf7
******
---8<---
```

Basic traceability as given in Figure 3.7 implemented by AAF naively is also given in the framework presented by [Gra09]. [Gra09] uses two arrays for deviation representation. Thus, identical items within the index arrays at two forms represent deviations having the same cause. This enables forward traceability as described above. Information for backward traceability is not explicitly maintained in Grabowski's framework, whereas the *SESYD* framework allows managed object to object linking. The presented framework architecture and basic traceability features provide information which can be used for enhanced deviation traceability features as presented in Chapter 4.

3.2 User Selectable Approximation of Non-linear Operations

As given in equations 2.15 and 2.20 in AA operations are divided into linear and non-linear ones. In the according state of the art Section 2.3.3 I present how the result of a non-linear operations can be represented by an approximation form (see Figure 2.9). Commonly used and already published approximation schemes are Taylor-, Chebyshev- and Minimal Range approximation. In the framework of Grabowski [Gra09] all of these three methods are implemented for several non-linear operations (including trigonometric, logarithmic and exponential functions). Within the *SESYD* framework these approximations are implemented for the multiplication and the division operator. For enriching the *SESYD* framework with other operations [Gra09] can be taken as a great template. The corresponding algorithms have to be adapted for the *SESYD* software architecture. Differences are given in deviation representation, adding/removing deviations from AAFs, the introduction of system deviation symbols, etc. (see previous subsections).

However, the important part of this thesis is not implementing various types of operations, but to discuss new concepts for how approximations are represented within the *SESYD* approach, how they can be managed, and how they influence analysis possibilities. I will discuss these questions within the following subsections, consequently using the multiplication operation as a running example.

3.2.1 Application Specific Approximation Techniques

The main statement of this section is that the application of a specific approximation algorithm is highly dependent on the analysis and verification task. To motivate this, first, I will identify some requirements on an approximation algorithm and their resulting Affine Arithmetic form:

- As already mentioned, the goal of approximation algorithms is to calculate a valid AAF which represents a potentially non-linear characteristic. In general, the approximated form has to fully enclosure the non-linear exact result (at least by its bounds).
- The result has to be a bounded affine form, also for repeated (looped) execution of the according non-linear operation. This might become a problem because published approximation algorithms, in general, add an extra deviation symbol to a form by every single execution. To overcome this, three methods can be applied: A sequential call of a cleanup method accumulates small partial deviations which are caused by approximation. Hansen's form can be used where no new symbols are added, but approximation deviations are represented by intervals. As used in the *SESYD* framework symbols can be managed, and a cause-identifier is stored. Sequential execution of a specific non-linear operation may have equal operators at every single execution.
- As usual at approximation computing an error measure have to be given.
- An approximation method has to return results which are reasonable respecting the modeled application. This may become important if the approximation is seen in a more general perspective where also piecewise discontinuous functions are considered.

Furthermore, the next itemization lists some selected properties and characteristics of an approximation AAF. Figure 3.9-a shows a non-linear function $\hat{x}_e$ (dashed red line) and its approximation $\hat{x}_a$ (in blue). They are evaluated concerning the given requirements. At the end of the next section, I will give an overview of published approximation schemes using the following properties.

- ***Interval bound values:*** As illustrated in Figure 3.9-b the approximated form has an additional deviation symbol ϵ_2 (blue lines indicate the bounds for $\epsilon_2 = \pm 1$). At the interval bounds, representing worst-case deviations the approximation forms have an over-approximation indicated as o_1 and o_2, where $o_{1,2} = |\hat{x}_a - \hat{x}_e|$ and $\epsilon_1 = \mp 1, \epsilon_2 = \pm 1$
- ***Shift of the central value:*** The central value of the approximated form must be symmetrically to its bound values ($\epsilon_2 = \pm 1$, blue lines). Thus, the central value of $\hat{x}_a$ is in general not equal to the central value of $\hat{x}_e$, $o_3 = |\hat{x}_a - \hat{x}_e|$ where $\epsilon_2 = \epsilon_1 = 0$ (see Figure 3.9-c).
- ***Average, minimum/maximum difference:*** The difference characteristics as average, minimum and maximum can be defined as: $1/2 \cdot \int_{-1}^{1} |\hat{x}_a - \hat{x}_e| d\epsilon_1$, $min/max(|\hat{x}_a - \hat{x}_e|)\big|_{\epsilon_1 = -1 \ldots +1}$
- ***Central value sensitivity:*** Besides these values also sensitivities (for ϵ_1) may differ between the exact and the approximated form. This is shown in Figure 3.9-d. As a property measure I define the quotient of sensitivities at the center value: $\frac{\partial \hat{x}_e(\epsilon_1)}{\partial \epsilon_1} \Big/ \frac{\partial \hat{x}_a(\epsilon_1, \epsilon_2)}{\partial \epsilon_1}\Big|_{\epsilon_1 = 0}$

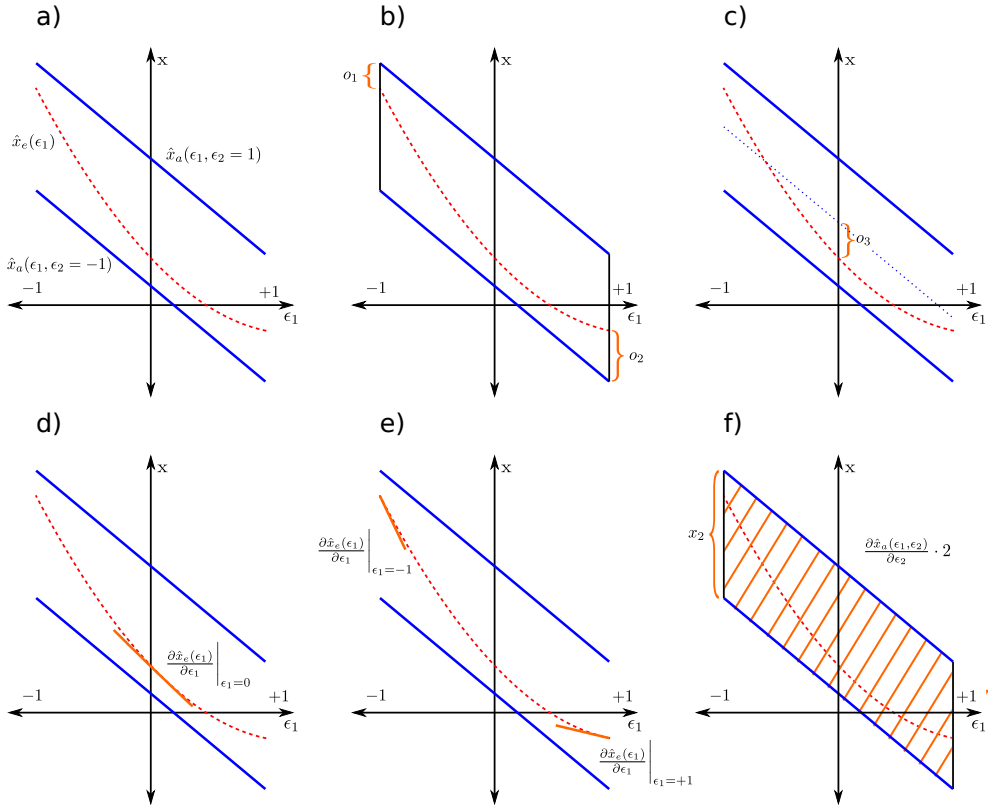


Figure 3.9: Properties of AAF approximations

- ***Bound sensitivity:*** Sensitivities can be also evaluated at the boundaries of the defined AAF range (see Figure 3.9-e): $\frac{\partial \hat{x}_e(\epsilon_1)}{\partial \epsilon_1} \Big/ \frac{\partial \hat{x}_a(\epsilon_1,\epsilon_2)}{\partial \epsilon_1}\Big|_{\epsilon_1=\pm 1}$ A significant situation happens if the according sensitivity quotient is negative. Thus, the reaction of the system caused by slight variation of ϵ_1 is complementary.

- ***Average, Minimum/maximum sensitivity difference:*** Similar to the differences in the value domain average, minimum/maximum can be evaluated for sensitivity: $1/2 \cdot \int_{-1}^{1} \left(\frac{\partial \hat{x}_e(\epsilon_1)}{\partial \epsilon_1} \Big/ \frac{\partial \hat{x}_a(\epsilon_1,\epsilon_2)}{\partial \epsilon_1} \right) d\epsilon_1$, $min/max \left(\frac{\partial \hat{x}_e(\epsilon_1)}{\partial \epsilon_1} \Big/ \frac{\partial \hat{x}_a(\epsilon_1,\epsilon_2)}{\partial \epsilon_1} \right)\Big|_{\epsilon_1=-1\ldots+1}$

- ***Area:*** In Figure 3.9-f the area spanned by the included additional deviation symbol ϵ_2 is marked. This area represents the variability added to the exact solution to strictly enclose the non-linear characteristic. The shaded area can be calculated by $2 \cdot \frac{\partial \hat{x}_a(\epsilon_2)}{\partial \epsilon_2} = 2 \cdot x_2$. In cases of a singularity of the the exact result within the $\epsilon_1 = -1 \ldots 1$ range, this area is divergent towards ∞ and therefore fails the requirement given above.

- ***Computation complexity and memory space:*** Besides the explained mathematical properties these two properties are significant for calculating approximations in a computer. The computation complexity informs about the number of steps that have to be computed for an approximation at a sequential execution. The memory space property gives a measure of the memory area that has to be allocated by a approximation AAF where a non-linear operation is executed sequentially (adding new symbols).

Approximation algorithms have different features regarding the given items above. Hence, for an application, it is an added value if a verification engineer is not limited to sticking at one specific approximation scheme. Different types of verification tasks would require particular constraints on the used approximation forms. As an example I give the following three selected use cases:

- Worst-case analysis has the requirement that potential over-approximation should be as low as possible. Specific sensitivities against variabilities in deviation symbols would be of minor interest (e. g. verification of safety-critical systems).
- On the other hand, for sensitivity analysis applications the over-approximation at the bounds is of small importance. It may be significant that the sensitivity of a limited zone around the central value is as exact as possible.
- If a non-linear operation is executed within a loop structure the given area property can be interesting. Additional deviation symbols covering nonlinearities should be limited to optimize the variability (area) caused by approximation.

Especially for safety-critical mixed signal system applications, correct output bounds are required [SG10]. In this case, the approximation gap inside the specified deviation range is of minor importance. Primary objective of the system analysis is to check whether the bounds of an affine output signal is within a defined tolerance specification or not (worst-case behavior). Based on the presented already published approximation schemes (Section 2.3.3) this property is satisfied by the minimal range scheme.

A further important statement if it comes to AAF approximation is correct linear partial deviation values. They allow qualitative statements about deviation sensitivity at the central value [SF03]. This sensitivity property is a significant disadvantage of minimal range approximation. Sensitivity values (also for user deviations which are in general caused by modeled physical effects) are modified in order to satisfy the minimum area objective (see Figure 2.9). This may falsify the result dramatically if this approximation result is a data basis for system sensitivity analysis. To satisfy this property Chebyshev approximation is the best choice. The sensitivity of the approximation AAF is equal to the exact sensitivity at the center value.

In the framework of Grabowski [Gra09] all approximation computations within a simulation run are calculated by a single selected algorithm type. The algorithm itself is selected before the start of the run. At the *SESYD* framework within an AAF object, multiple approximation forms can be reminded. So an engineer can select in advance which approximation schemes are calculated during a simulation run. For subsequent post-processing one of them is used according to the specified type of verification task. In addition, as an enhanced function, approximations can be enabled/disabled during simulation runtime. This results in a so-called application specific approximation technique for Affine Arithmetic forms.

In an implementation perspective, the AAF class and its included deviations are extended by scalar variables representing different types of approximations. In general an approximation form in this framework is defined as

$$\hat{x}_a = \beta_m + x_1\epsilon_1 + x_1\epsilon_1 + \ldots x_n\epsilon_n + \alpha_m\epsilon_m \tag{3.1}$$

where $\hat{x}_a$ represents an approximation AAF, m is the index for a specific approximation type, β_m is the shifted central value for approximation m (see Figure 3.9-c), $x_{1\ldots n}\epsilon_{1\ldots n}$ are user deviation and

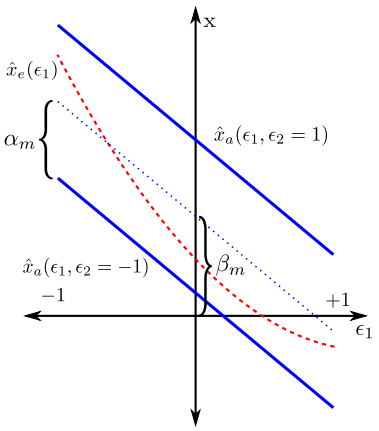


Figure 3.10: Characterization of an approximation using α and β values.

$\alpha_m \epsilon_m$ represents the additional system deviation. The m in the α and β indices can be replaced by a name identifier for the approximation method (e. g. $\alpha_{Chebyshev}$). This representation of an AAF, representing one specific approximation is illustrated in Figure 3.10.

As already mentioned the classes for **AAF** and **Deviation** are extended by scalar variables for alphas and betas. At the time of writing in the *SESYD* framework, Taylor, Chebyshev, minimum range and Interval-exact approximations are implemented. The extended framework structure is illustrated in Figure 3.11. As indicated in yellow a specific approximation is selected by picking corresponding scalar values of AAF and system deviation objects. New approximation methods can be added easily by adding corresponding α and β scalar variables. For future framework versions, the α and β values can be potentially stored in vector containers.

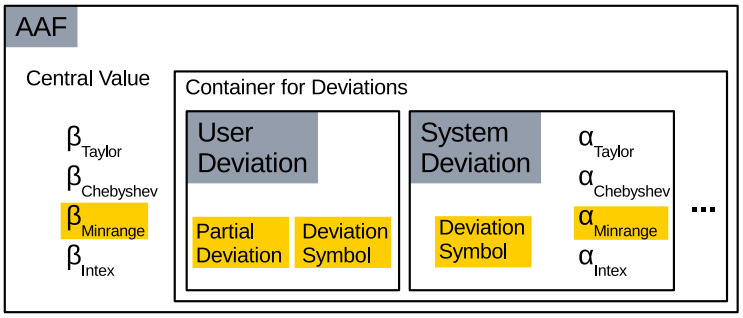


Figure 3.11: Extended classes for **AAF** and **Deviation** including α and β values for Taylor, Chebyshev, minimum range and interval-exact approximations.

In the *SESYD* framework multiple approximations can be computed within a single non-linear operation. For management, an approximation register as illustrated in the class diagram shown in Figure 3.12 is implemented. In this object two static references named **DefaultApproximation** and **ArithmeticApproximation** are included. **ArithmeticApproximation** defines the set of computed approximation types, and **DefaultApproximation** defines which approximation is used for AAF printing, plotting and debugging functions. Strictly speaking, one configuration defines which approximations are calculated during simulation and the other defines which type is used for standard functions. A computation of approximation variables α and β is triggered at all executed operations the associated AAF is involved in. This guarantees that all available approximation forms are valid after each non-linear operation.

In general, the *SESYD* framework has functions for loading framework configurations from a config file on the file system. For config file decoding the GNU libconfig library is used. The configuration file hold path definitions for waveform and result files, switches for debug message levels, deviation tracing and monitoring configurations, approximation configurations, etc.

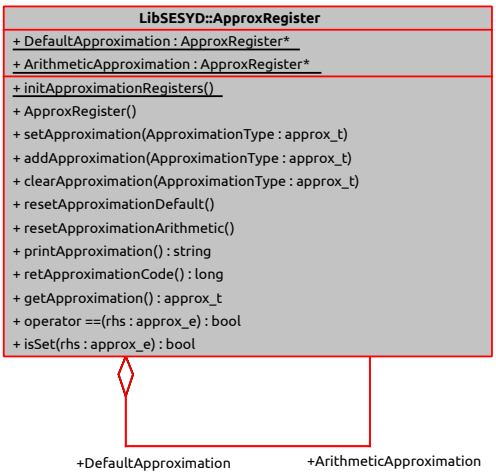


Figure 3.12: Class diagramm of the approximation register used for the definition of computed and default approximation types.

3.2.2 Interval-exact Approximation

Goal and motivation for the interval-exact approximation is to combine advantages of minimal range and Chebyshev approximation. Strictly speaking, having exact interval bound values, and correct sensitivity properties of the center value. The already identified disadvantage of a shifted central value for all approximation schemes adding extra system deviation symbols is solved by the *SESYD* framework architecture (exact central value is stored and shifted ones are handled with β values). A significant drawback of the framework implemented in [Gra09] is that shifted central values are not stored at all. This has the consequence that at sequential multiplications partial user deviations are multiplied by wrong central values. Thus, the mentioned shift (in the value domain) has also an impact to partial deviation values representing system sensitivities. The *SESYD* framework stores and uses the exact cental value correctly in subsequent operations.

For explaining the interval exact approximation algorithm a multiplication of two AAFs will be discussed, but the basic idea of the method can be adapted for any abitrary non-linear operation. An advantage of AA system simulation is that correlated deviation causes can be modeled by using the same ϵ symbols in multiple affine forms [SF03, PLV10]. Such correlations must be taken into consideration at a functional composition of two or more Affine Arithmetic forms. Based on given correlation situations different cases for calculating the interval-exact approximation can occur.

One case of AAF multiplication is that deviations of both affine operators $\hat{a}$ and $\hat{b}$ are completely uncorrelated. Mathematically expressed for $\mathcal{N}_{\hat{a}}$ and $\mathcal{N}_{\hat{b}}$, which are the sets of the operand's deviation symbols indices (see equation 2.2) $\mathcal{N}_{\hat{a}} \cup \mathcal{N}_{\hat{b}} = \{\}$ is valid. An independent consideration of the operands is possible. Each AAF operand can be converted into an equal interval representation [SF03]. These representations can be multiplied by using an algorithm published in [Pop98, Algorithm 3.1]. The result is an exact interval solution (only upper and lower value of the resulting

range are calculated not an AAF) for the operation. The multiplication of two affine forms can be partitioned into two parts. The first one is a pure affine term with an exact central value and linear user deviations. The second part includes all non-linear terms of the multiplication [SF03]. Combined quadratic terms included in the exact mathematical result are replaced by one new system deviation. This contains a α approximation value and a modified central value β in the affine form. At this type of approximation β is set to the middle of the calculated exact result interval and α is set to the radius difference between the exact interval solution and the affine part. This results in an affine representation with no over-approximation and correct calculated linear user deviation parts.

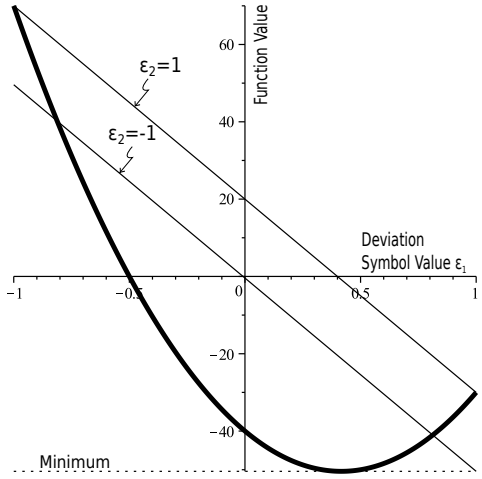


Figure 3.13: Interval exact approximation method

The second case is a multiplication of correlated AAFs. An independent consideration of the operands is not longer possible [SF03]. At having one or more correlated deviation symbols, the mathematically exact result contains a quadratic term. The graph of the result concerning the deviation values ϵ forms a parabola object. In this case, it can happen that minimum or maximum values of the result interval are defined by deviation symbol permutations unequal to their -1 or $+1$ bounds. In the example shown in Figure 3.13 the minimum is located at a deviation symbol value of 0.42 (for n-correlated symbols a combination of ϵ values is possible). In other words, the minimum or maximum resulting value is inside of the $[-1, 1]$ interval for ϵ values. Minimum or maximum values of the parabolic result function are analytically calculated (n-dimensional extremum problem). It can be proven that minimum and maximum values of an exact result generated by the multiplication of correlated AAFs are located at the edges of the object spanned in the n-dimensional ϵ-space. Minima and maxima can be found in one varying ϵ value and all other deviation symbols of the result are permutations of $+1$ or -1 values. This calculation returns exact interval bounds of the operation result. The approximation value α and corrected central value β can be calculated as explained for uncorrelated multiplications.

Multiplications of AAFs which include a mix of correlated and uncorrelated deviation symbols have to consider all $+1/-1$ combinations of uncorrelated symbols. Uncorrelated deviations are always linear factors which extrema are located at -1 or $+1$ ϵ-value combinations. Each $+1/-1$ permutation of uncorrelated symbols will result in a pure correlated form which can be handled as explained previously.

Example:
Figure 3.13 illustrates the result of the interval exact approximation for a multiplication of $2 + 4\epsilon_1$ and $-20 + 15\epsilon_1$. The bold line is the mathematical exact result including quadratic terms $-40 - 50\epsilon_1 + 60{\epsilon_1}^2$. It can be seen in the chart that interval bounds of the result are -50 at $\epsilon_1 = 0.42$ and 70 at $\epsilon_1 = -1$. The result of the interval exact approximation introduces an additional system deviation symbol with an approximation value α of 10.21, and a modified center value β of 9.79. The approximated affine form $\hat{y}$ is $9.79 - 50\epsilon_1 + 10.21\epsilon_2$. The radius of the approximated result AAF is equal to the range spanned by the exact result of the multiplication. As previously mentioned this approximation is optimized for simulations where exact interval bounds and correct user partial deviations of an AAF are required.

For the discussion of advantages and disadvantages of the interval-exact approximation scheme in the following table holds qualitative statements (including also state of the art techniques presented in Section 2.3.3). Basis are the introduced approximation properties defined in Subsection 3.2.1 and Figure 3.9.

Table 3.1: Properties of implemented approximation algorithms

	Taylor	Chebyshev	Minimal Range	Interval-exact
Bounds (Fig. 3.9-b)	- This approximation adds no system deviation symbol. Thus, the approximation form has a potentially large over- and under-approximation if the exact characteristic is highly non-linear.	- At least one interval bound may have an over-approximation caused by conservative enclosure of the exact characteristic.	+ The goal of minimal range approximation is to calculate a minimum enclosure area without over--approximation. Thus, interval bound values are exact.	+ This type of approximation has the goal to calculate a form having exact interval bound.
Cerntral value shift (Fig. 3.9-c)	+ This scheme approximates using a tangent at the center value. Thus, there is no shift of the approximation form's central value.	- The approximation form is based on the first derivation at the central value of the exact function. The central value shift depends on the calculated value for the partial system deviation. The shifted central value has to fulfill the AAF symmetry requirement.	- see Chebyshev approximation	+- Similar to the Chebyshev approximation, but if a minimum/maximum is within the valid values for epsilon, the partial system deviation, and thus the shift of the approximation center value is smaller.
Difference (Fig. 3.9-c)	+- The minimum difference is due to the tangent approximation zero, but the average difference might be dependent on the nonlinearity significantely high	+ The approximation form's central value is always within the enclosure form. Thus, the minimum difference is also zero. Also, the average difference over the full range is lower compared to Taylor approximation.	+- Due to the constraint of exact interval bounds, in general, the partial system deviation is larger compared to Chebyshev. Thus, the average difference is also larger. For the minimum, the same property as for Chebyshev is given.	- If a minimum/-maximum is within the epsilon interval the partial system deviation value is smaller compared to Chebyshev. Thus, the average difference may get disappointingly large.

Table 3.2: Properties of implemented approximation algorithms-part2

	Taylor	Chebyshev	Minimal Range	Interval-exact
Central value sensitivity (Fig. 3.9-d)	+ Taylor approximation is a tangent at the central value representing exact sensitivity. For applications where sensitivities are relevant and behavior can be linearized in a single operational point, Taylor approximation returns less complex forms due to no extra added system deviation symbols.	+ The Chebyshev approximation takes the sensitivity of the exact form as a basis. Thus, the sensitivity is exact at the center value.	- Due to the given approximation constraints, sensitivities are modified and are in general not equal to the sensitivity of the exact form in the center value.	+ This approximation is similar to Chebyshev and has exact sensitivities at the center value.
Bound sensitivities (Fig. 3.9-e)	- The slope (representing sensitivity) of the tangent at the bounds is unequal to slope at the center.	- The Chebyshev approximation takes the sensitivity of the center value which is in general not equal to the sensitivity of the interval bounds.	+- In general, min-range approximation constraints require a sensitivity value at one at the interval bounds. Thus, the approximation sensitivity is equal to sensitivity at the value bounds. If a minimum/maximum is within the epsilon interval, the sensitivity is set to zero.	- The interval-exact approximation takes the sensitivity of the center value which is not equal to the sensitivity at the interval bounds.
Sens. difference (Fig. 3.9-e)	- The average sensitivity difference is highly dependent on the nonlinearity of the initial function. In general the sensitivity of an approximation respecting the non system deviation symbols is constant.	+- The Chebyshev approximation takes the sensitivity from the center value. The average sensitivity difference is low if sensitivities given on the full interval are around the sensitivity at the center value.	- User deviations representing sensitivities are modified to satisfy constraints of the approximation algorithm. Thus, the average sensitivity difference is dependent on the sensitivities inside the based exact function interval.	+- See Chebyshev approximation.

Table 3.3: Properties of implemented approximation algorithms-part3

	Taylor	Chebyshev	Minimal Range	Interval-exact
Area (Fig. 3.9-f)	+- Taylor approximation adds no extra system deviation symbol. Thus, a non-linear characteristic is not enclosed.	+ Compared to minimum range approximation, Chebyshev approximation results in a smaller enclosure area under the price of potential over-approximation/under-approximation.	+- Minrange in general is a minimization result regarding low enclosure area, but due to the second requirement of no over-approximation the area may get for some cases disappointingly large.	+ Interval-exact approximation reduces the value of the additional partial system deviation, with the consequence that a full enclosure over the complete variability interval can not be guaranteed (but with exact interval bounds).

3.2.3 Approximating Behavioral Discontinuities

Within the prior subsections, I described approximation methods focused on basic mathematical operations as (mainly) multiplication, division, exponential, trigonometric, etc. Here a little more generalized approximation approach which is also applicable to operations described in Subsection 2.3.4 (e. g. minimum of two AAFs evaluating an AAF as result) is discussed.

As long as the characteristic to approximate is continuous (but non-linear) an approximation according to the previously defined algorithms, resulting in a full enclosing form, can be computed. The approximation problem gets significantly more complex if discontinuous behavior has to be approximated. Such behavioral discontinuities may be caused by the introduction of compare operators with divides a given joint AAF range into two or more subranges. Strictly speaking, an operator described as $x \oplus \hat{y} : \mathbb{R} \times \mathbb{A} \mapsto \mathbb{A}$.

An elegant and impressive solution for handling discontinuities in the AAF domain is one of the leading research activities by Carna Radojicic introducing the concept of Affine Arithmetic Decision Diagram (AADD) [Rad16]. We describe the basics of the AADD method in the ACC example, Subsection 5.4.

For handling discontinuities within the *SESYD* framework, I compute an AAF approximation based on interval considerations. We describe the algorithm based on the following small running example: Assume two signals as illustrated on the left side of Figure 3.14. The first one is an AAF signal which central value is continuously increasing, $\hat{x}(t)$ (red lines). The second one is a constant value represented by a scalar, $y(t)$ (blue line). As operation for this illustrating example, the maximum of both signals is evaluated. Mathematically expressed as $max(\hat{x}(t), y(t))$, which is a function of the type $\mathbb{A} \times \mathbb{R} \mapsto \mathbb{A}$.

As already mentioned I compute this *max* operation in an interval perspective. Thus, the illustrated time window in Figure 3.14 is partitioned into three sections. In sections 1 and 3 the maximum situation is clearly defined. Whereas in section 1 is caused by the scalar value (in the output form the scalar is the central value, and the set of deviations is empty, $\mathbb{R} \subseteq \mathbb{A}$). The maximum of section 3 is given by the AAF $\hat{x}$ (see right part of Figure 3.14 where the approximation is colored in green). ϵ values in $\hat{x}$ can have any arbitrary values inside their ± 1 specification, but the

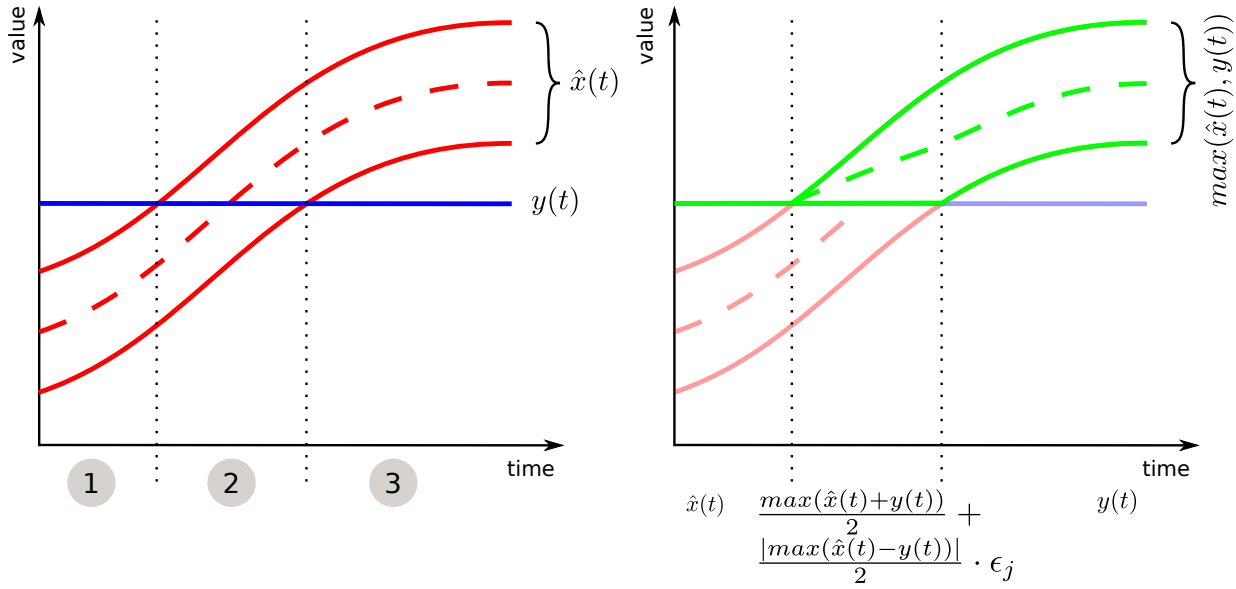


Figure 3.14: Evaluation of an approximation AAF for the maximum of $\hat{x}(t)$ and $y(t)$

scalar y is smaller in any case. In other words, the minimum boundary value of $\hat{x}$ is greater than y. In section 2 the situation is uncertain and somehow unclear. In an interval perspective, the lower bound of the *max* is still given by the scalar value. The upper bound is defined by the upper bound of the AAF $\hat{x}$. Thus, the AAF $\hat{x}$ is split. There are potential epsilon valuations where the maximum is given by the AAF and others where it is defined by the scalar value. AADDs, in this case, consider both possible cases considering previously evaluated epsilon constraints. In the *SESYD* framework an auxiliary AAF for this uncertain section 2 is generated. The form has a symmetric central value according to the evaluated interval bounds, and the size of the range is defined by a partial deviation value associated with a system deviation symbol. This is expressed as $\frac{max(\hat{x}(t)+y(t))}{2} + \frac{|max(\hat{x}(t)-y(t))|}{2} \cdot \epsilon_j$ where $j \notin \mathcal{N}_{\hat{x}}$. For this process the most significant advantage of the *SESYD* framework regarding approximation handling comes into play. At the transfer from section 1 to 2 the AAF $\hat{x}$ is not deleted but stored in the AAF object (modified each point in time). The described auxiliary form representing the approximation is given by corresponding α and β values (see Figure 3.11). The approximation management for section 2 is set that this approximation is presented as the value of the AAF object, while the original form is stored in the background. But the background form is still needed for evaluation of the point in time when to switch from section 2 to 3, where the situation is clear again. It has to be mentioned that if the approximation form is used for subsequent calculations (e. g. within a feedback loop) the transition from section 2 to 3 gets uncertain as well. Thus, this process results in an approximation in value and time domain.

3.3 Interval-based Simulation

In the *SESYD* framework also intervals are implemented (see ADT in Figure 3.1). Therefore, I used the basic endpoint representation and associated basic operations $+, -, \cdot, /$ as presented in Sections 2.2.3 and 2.3.2. Intervals are represented as an object. The corresponding class diagram of a C++ interval object is illustrated in Figure 3.15. As indicated, the class holds beside

overloaded math operators also methods for midpoint-, radius evaluation, bound modification and printing.

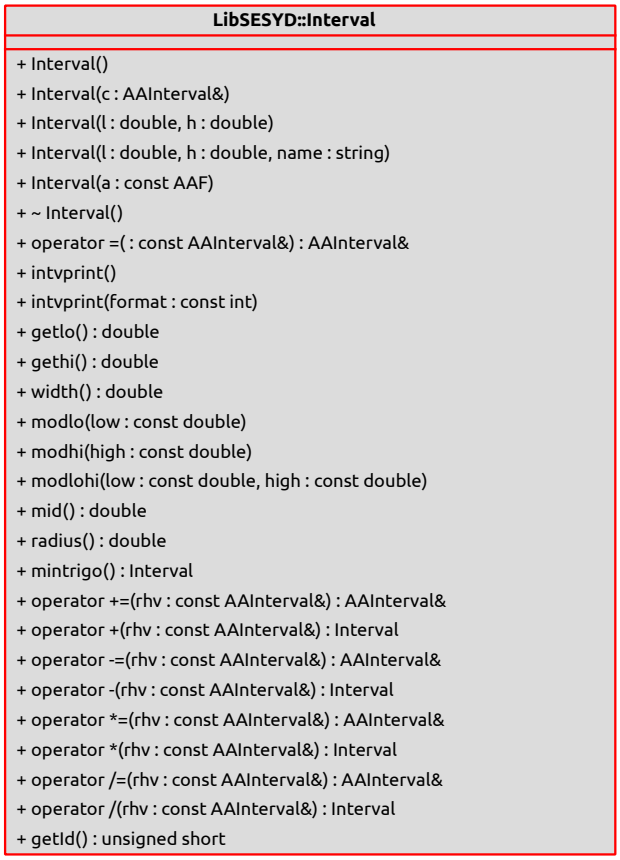


Figure 3.15: Class diagram of the interval ADT

An interval object can be associated with an AAF to create a Hansons form as described in Subsection 2.3.3. This enables modeling of a mixed specification of uncertainties using interval and Affine Arithmetic forms. The implementation of intervals is more or less straightforward coding and can be highly re-implemented using the work of Grabowski [Gra09] as a template. For future applications also the well tested IA framework "Boost" [24] can be interfaced.

3.4 Multi-run Functions

I mainly use the SystemC simulator in combination with the developed *SESYD* framework. Multi-run simulation methods are by default not provided within the SystemC standard [Sys12]. As illustrated in Figure 3.1 I implemented a specific multi-run module integrated in the framework. It enhances SystemC by functions for multi-run simulation as well as result management, tracing and simulation setup.

Conceptual challenges for implementing multi-run simulation in SystemC
The main problem, as depicted by the SystemC simulation flow diagram Figure 3.16, is that the simulator API provides no function to explicitly re-initialize the simulation process. Strictly speaking, if the state of the simulator exceeded to **SC_RUNNING** or **SC_PAUSED** there is no function like **sc_reset()** leading back to **SC_ELABORATION** [Sys12].

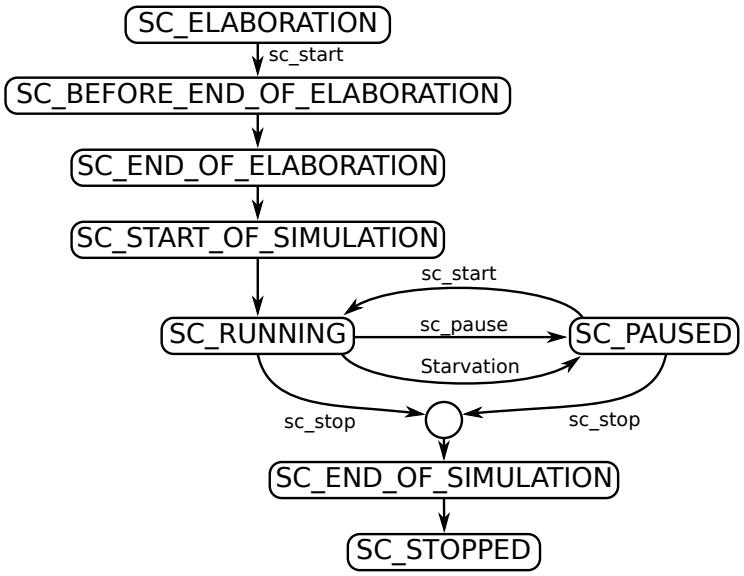


Figure 3.16: SystemC simulation flow (Similified illustration of [Sys12, p. 34]).

At a closer look to the SystemC source code documentation, there is a class named **sc_simcontext** implementing the simulation context for a simulation run. Instantiating multiple simulation context objects for multi-run simulation looks very promising and besides the framework published in [SIVT06] this was also my first approach. Unfortunately, several disadvantages can be highlighted. According to the language reference manual [Sys12] since SystemC version 2.0.1 the class is deprecated. Thus, this approach in combination with state of the art versions (2.3.1 for this work) is not possible. An other drawback, reported in [SIVT06] (which is confirmed by my experiences), is that continuous destroying and recreating the **sc_simcontext** object creates memory leaks. SystemC developers also report in web forums that deleting *sc_simcontext* is not designated. However, I tested the approach but number of executed runs is limited to approximately 8000. Then SystemC crashes with a segmentation fault due to the described generation of memory leaks. A third major drawback is that each recreation of the simulation context is accompanied by a new model elaboration. This behavior increases simulation time is redundant in multi-run simulations and thus not required. The model structure is equal for each multi-run simulation round, just the input parameters are diversified.

A further possible multi-run approach is to call the precompiled SystemC executable (executing a single simulation run of the model) sequentially. Thus, the execution, as well as stimuli definition and collecting results, are fully controlled by external scrips or third-party applications [SIVT06]. However, the sequential load, and unload, of the executable (also including the re-elaboration of the model) adds further additional delays. A significant advantage compiling several executable as a package is that their execution can be selectively distributed to multiple processor cores or machines.

The *SESYD* multi-run implementation approach

My approach for this work is to integrate multi-run simulations into a single executable (including the model, the *SESYD* framework, and the SystemC simulator core). Drawbacks are given by model re-elaboration and sequential loading/unloading the executable should be avoided. By shrinking this runtime overhead, I expect maximum multi-run simulation speed for execution on a single core machine. A further requirement is independent controllability of each simulation

run by management objects which are not destroyed at an application model reset.

A central idea of the multi-run module is having two specific classes. One providing management functionalities for handling simulation rounds, stimulus generation, etc., and one for conducting the simulation core. For this thesis, SystemC is used as a simulation framework, but the generic interface enables also the usage of other third-party simulators.

According to the previously described challenges resetting the application model without resetting the SystemC simulation, core is a challenging task. SystemC provides built-in functions for process control. As described in the Language Reference Manual [Sys12] for each type of processes a handle can be requested. This process handle allows calling control functions as **suspend**, **resume**, **reset**, **kill**, etc. [Sys12]. A dedicated **reset** function, which is a conventional C++ member function may call described SystemC process control functions and re-initializes internal variables. A further possibility is to implement a dedicated reset port in the module's interface. This method requires no calls of process control functions and resetting is fully covered by the system specification. Processes are sensitive to the reset signal, and the according behavior is clearly defined. A detailed description and guideline to make a model ready for multi-run simulations is given later.

Class structure of the *SESYD* multi-run module

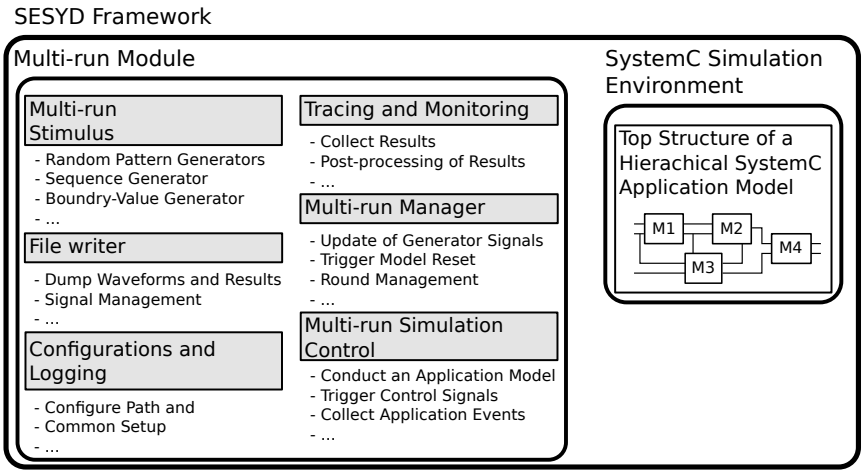


Figure 3.17: Overview over the implemented multi-run module described within this thesis. Classes are highlighted in gray. Most important are the multi-run manager and multi-run simulation control classes fully defining the multi-run behavior and resetting the SystemC application model.

Figure 3.17 illustrated a block diagram of the *SESYD* multi-run module including a short description of associated functions. The module is a collection of classes (marked in gray) which main parts are:

- **Multi-run Stimulus:** This is an extensible set of classes to describe different types of pattern generators.
- **File Writer:** Signal traces are written into *vcd* (value change dump) waveform files. The SystemC simulation environment provides functions for writing SystemC specific signals into files, but not for extended writing behavior used for multi-run simulation. For correct

time information a virtual timestamp, which starts at $t = 0$ after each simulation round, can be requested at the multi-run management class. Registered signal names are extended by "*r", where r is an integer number representing the simulation round.

- **Configurations and Logging:** General simulation configurations and framework settings (path names, enabling debugging, etc.) are read from a configuration file using the *config++* library.

- **Tracing and Monitoring:** For detailed analysis of the application model additional tracing and monitoring functionalities are integrated into the framework. They allow, for example registering a specific data-path through the model for automatic monitoring whether uncertainty values exceed specified bounds.

- **Multi-run Manager:** The multi-run manger implements control functions for the full simulation procedure.

- **Multi-run Simulation Control:** This class interfaces the simulation environment and resets the application model. From a hierarchical perspective it is on the same level as the application model and fully integrated into the SystemC environment. A simulation control object instantiates the application model top structure and two **SC_THREAD** processes. File- and object-name of the top module are configured by macros in the class header file. Thus, the **sc_main** function, which is called after startup, instantiates a new object of the simulation control class. One of the two started threads is a timer which may limit the maximum simulation time of a single round. A new simulation round can be triggered by the described timeout or by notification of a specific user definable set of SystemC events. The second process which is called **round controller** executes the following steps in a loop:

  1. Wait for the timeout or a user specific event to trigger the reset of the model. The first execution is started automatically after elaboration of the SystemC model.
  2. Check conditions whether a next round is triggered or the maximum number of simulation rounds is already reached.
  3. Reset the SystemC application model, including canceling user events and calling reset functions or setting the reset pin as described previously.
  4. Update all stimulus pattern generators to new calculated values. This defines a new parameter set for the following simulation run.
  5. New values of pattern generators, defining the simulation setup of the next round, are written into an *ID-file*. For analysis of the plotted traces this file is very helpful because it contains a complete mapping between specific rounds and the applied parameter set.
  6. Last the described timer process limiting the maximum simulation time per round is resetted.

Stimulus pattern generators

Stimulus generators are used to source the application model with a set of scalar parameter values in each simulation round. Types of generators implemented for this work are random pattern generators, sequencers, boundary value generators and value list generators. In addition to these sampling techniques, basic functions for importance sampling are implemented. The Generators offer the following functions: **getValue**, **update**, **reset** and **getStatus**.

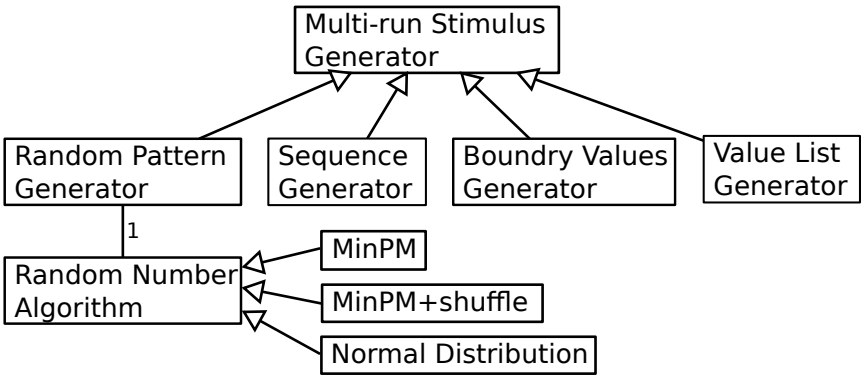


Figure 3.18: Class diagram of implemented pattern generators. Multi-run Stimulus Generator and Random Number Algorithm are designed to be abstract interfaces for the integration of customized pattern generators.

These functions are declared within an abstract C ++ base class. A generic interface is provided to define user-specific generator behavior. The corresponding class diagram is shown in Figure 3.18.

A **random pattern generator** implements a single instance of a random number generation algorithm. This abstract class provides a **rand** function which is overwritten for each specific algorithm implementation. Algorithms may provide characteristics regarding sequence periods and probability distributions. *MinPM*, which stands for "Minimal" random number generator from Park and Miller, calculates uniform random deviate, at a period of approximately $2.1 \cdot 10^9$ [PTVF02]. Authors of [PTVF02] report that this algorithm is quite good but under rare circumstances conflicts with the generation algorithm may happen. Thus, they propose an enhanced algorithm including a shuffling process that the *j-th* value in the sequence is not present at the *j-th* algorithm call. In addition, I implemented a random pattern generator having a normal probability distribution. Therefore, built in C ++ 11 random pattern generator classes are used. For documenting the probability distribution of a pattern generator a so called distribution file can be dumped.

A **sequence generator** returns a scalar number between defined lower and upper boundary values. These limits and a resolution parameter are set at the instantiation of the generator. The returned value chain starts at the specified lower bound and is increased sequentially after each call of the **update** method.

A **boundary value generator** is similar to a sequencer except the definition of a resolution is not needed. Just corner cases represented by lower and upper boundaries are returned.

A **value list generator** is a special kind of a sequencer where generated values are predefined and initially handed over using a vector.

Multi-run round management
For management and behavioral control of the full multi-run simulation the Multi-run Simulation Control class (see Figure 3.17) provides an appropriate set of methods.

Stimulus pattern generators can be registered at the Multi-run Manager object and corresponding references are stored into a list. Thus, the manager has access to generator manipulation functions as **reset**, **update**, etc. The so-called "generator update rule" specifies which and how registered pattern generators are updated automatically at the end of a multi-run round. "update all" means that each of the registered generators, independent of its type are updated. In the case of random

pattern generators, a completely new set of random variables is determined for the next simulation round. For implementing corner case and worst case simulation the update rule can be set to "update exhaustive". At this configuration a generator is sequentially updated until its status is **finished** (e. g. for an increasing sequencer the specified upper value bound is reached). After that, the according generator is reset and the next generator in the list is updated. By implementing this generator behavior, sequencers present in combination all possible combinatorial value at the outputs. To control the behavior of simulation runs, within the **Multi-run Manager** class a second rule called "simulation end rule" is defined. One possible rule-setup is to specify a maximum number of simulation rounds N. Besides this strict limitation also specifying "infinite" is possible. Functions for aborting, pausing and resuming during the execution of the simulation are provided in the corresponding class. A further configuration of the simulation end rule is to calculate the number of simulation rounds given by the registered pattern generators. For the previously described sequencer setup and an "update exhaustive" generator update rule, the number of simulation rounds can be calculated in advance.

Design integration
Making a SystemC design ready for multi-run simulations using this framework the application model has to fulfill some requirements. In general, as reported in Subsection 2.3.1 there is no need for model modification because exact numerical design models can be used. However, the integration in the *SESYD* framework in combination with a SystemC simulation requires the following modifications:

1. In the SystemC main function (**sc_main**), create a new Multi-run Simulation Control object and set specific macros defining the top structure of the application model.

2. Each module must implement the possibility of being resetted. The call of reset functions or driving reset port pins must be accomplished in the full model hierarchy. The Multi-run Simulation Control object just calls the reset function in the top module. In detail, for **SC_THREAD** processes, request a handle and call the process control functions **reset** and **resume**. For **SC_METHOD** processes, add a reset port, add sensitivity and strictly specify the reset behavior.

3. Replace SystemC tracing function calls by their multi-run versions provided by the framework.

4. As described, variations in multi-run simulation methodology are considered by creating an appropriate pattern generator, and repeated execution of single simulation runs. Thus, a major part of making a model ready for multi-run is to replace constant parameter values by the corresponding pattern generators. For automatic round management all generators are registered at the Multi-run Manager class.

5. Before starting the simulation (calling **sc_start**) the simulation behavior has to be configured, by defining the mentioned rule setup.

3.5 Integration in a SystemC/AMS Environment

In an implementation perspective, the *SESYD* framework can be compiled with a standard GCC (GNU Compiler Collection) toolchain (for this thesis I use the version 4.8.1). For logging and configuration file management I use the freely available operating system libraries **log4cpp** and

config++ respectively. Functions declarations are included in the **SESYD.hpp** and integrated in a **LibSESYD** C++ namespace. The framework is initialized by calling the **initSesydFramework** function. I show the usage of basic *SESYD* functions and objects at the following exmple of a filter. For the implemented system I used the SystemC simulator. For detailed information about SystemC see [GLMS10].

The system simulated in this mini-example adds three sinusoidal signals having different frequencies. Then a part of the frequency spectrum is cut out using a high order digital filter. In the following, I describe the *SESYD* significant inline code snippets of the model. First, I designed a waveform generator as a SystemC module, which parameters are name, amplitude frequency and phase:

```
wavegen wavegen_inst1 ("WaveformGenerator1",10,100,0);
wavegen wavegen_inst2 ("WaveformGenerator2",4,700,M_PI/3);
wavegen wavegen_inst3 ("WaveformGenerator3",2,900,M_PI/2);
```

Each of the generated waveform holds an **AAF** object including 2 user deviations called "random noise" (associated with a correlated deviation symbol), and "ModuleDevVariation".

```
LibSESYD::AAF *sample;

d = new Deviation("RandomNoise",0,noise);
moduleDeviceVariation = DeviationSymbolManager::getInstance()->RequestNewDeviationSymbol("ModuleDevVariation");
d2 = new Deviation("ModuleDevVariation",0.2,moduleDeviceVariation);

sample = new AAF("DeviatedSineWave",0);
sample->addDeviation(d);
sample->addDeviation(d2);
```

In the process part of the **wavegen** module the central value of the AAF and the non correlated partial deviation value is modified at each call of the module. Therefore, the methods **setCvalue** and **setDeviationValue** provided by the **AAF** class are used.

```
sample->setCvalue(this->A * sin(2 * M_PI * this->f * sc_time_stamp().to_seconds() + this->phi));
sample->setDeviationValue(sample->getCvalue()*0.1,*d);
```

The accumulated waveform including the central value (black dashed line) and four partial deviation values associated with the symbols $\epsilon_{1,2,3,4}$ are plotted in Figure 3.19.

This AAF signal is then filtered by a digital finite impulse response (FIR) filter. Filter coefficients are computed using the online filter tool *T-filer* [22]. The website allows graphically designing the filter characteristic. The corresponding filter coefficients are computed, inserted into a C-array structure and appropriate filter functions (read and write samples) are generated. These can be directly copied into a SystemC module.

```
void filtering() {
            // Read
            *(MyFilter.history[MyFilter.last_index++]) = (*in.read());

            if (MyFilter.last_index == SAMPLEFILTER_TAP_NUM)
                MyFilter.last_index = 0;

            // Write
            filteredSample->resetAAF();
            int index = MyFilter.last_index;
            for (int i = 0; i < SAMPLEFILTER_TAP_NUM; ++i) {
                index = index != 0 ? index - 1 : SAMPLEFILTER_TAP_NUM - 1;
                (*filteredSample) += ((*(MyFilter.history[index]))
                        * filter_taps[i]);
            }
        out.write(filteredSample);
}
```

The filter for this example is designed to propagate the 100 *Hz* part and attenuate the 700 *Hz* and 900 *Hz* part of the accumulated signal. The evaluated FIR filter has a history length of 103

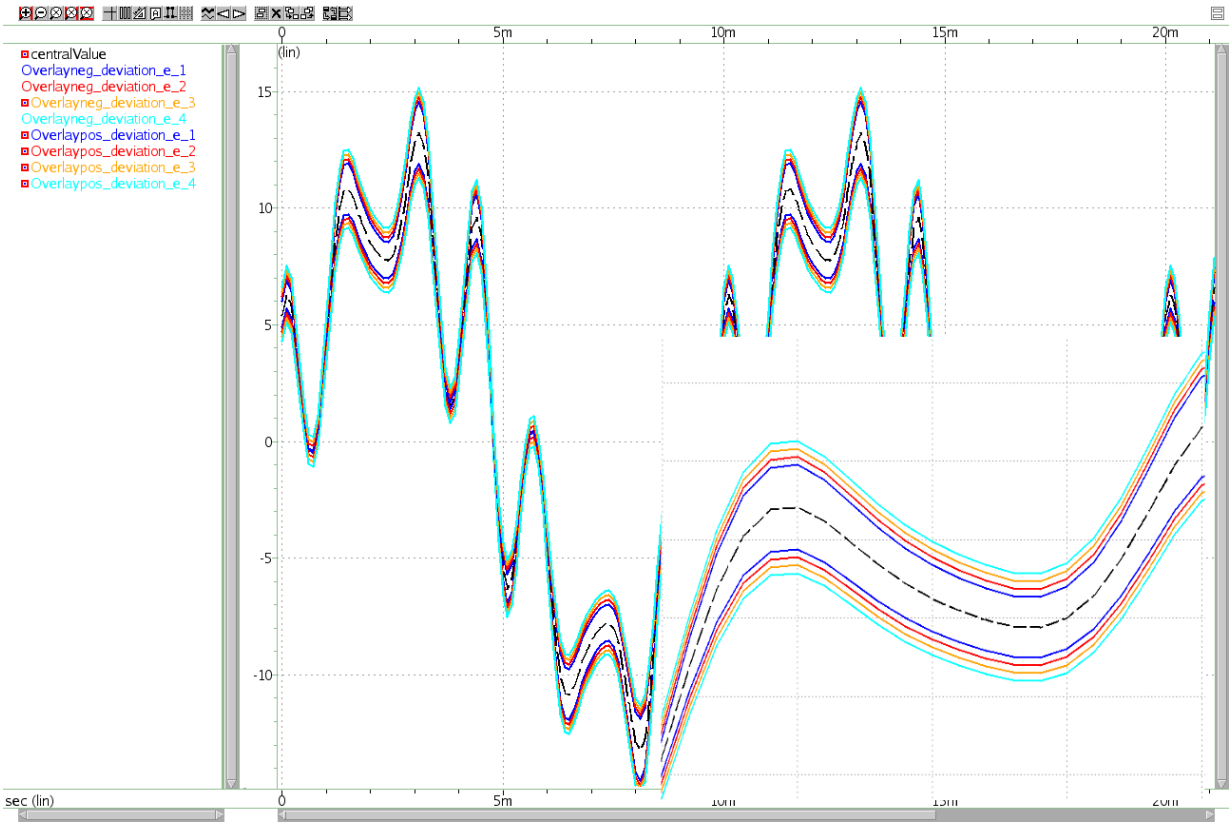


Figure 3.19: Accumulated signal of the three waveform generators. A zoomed section of the signal between 1 *ms* and 3 *ms* is showen in the lower right corner.

taps and its frequency characteristic is illustrated in Figure 3.20

The output of the filter is illustrated in Figure 3.21. The figure shows the output AAF including the central value (dashed line) and all accumulated partial deviation associated with $\epsilon_{1,2,3,4}$. The characteristic of the signal indicates that it takes quite a while until all 103 filter taps are filled and the signal reaches a steady sinusoidal state.

The corresponding frequency spectra of the filter's in- and output signals are evaluated and discussed in the frequency domain analysis Section 4.8.

This section presents an impression of using the *SESYD* framework within a C/C++ based environment as SystemC or SystemC AMS. Implemented classes as described in Section 3.1 offers a set of commonly used methods for handling AAFs.

3.6 Performance and Scalability

To discuss the topic of performance in this section, thee steps are discussed. These steps are organized according to the possible impact on the overall performance of a system simulation using the *SESYD* framework:

1. ***Signal monitoring:*** Access to mass storage on the computer is slow even at the time of solid state drives. As a result, it is inevitable that the recording of simulation data and

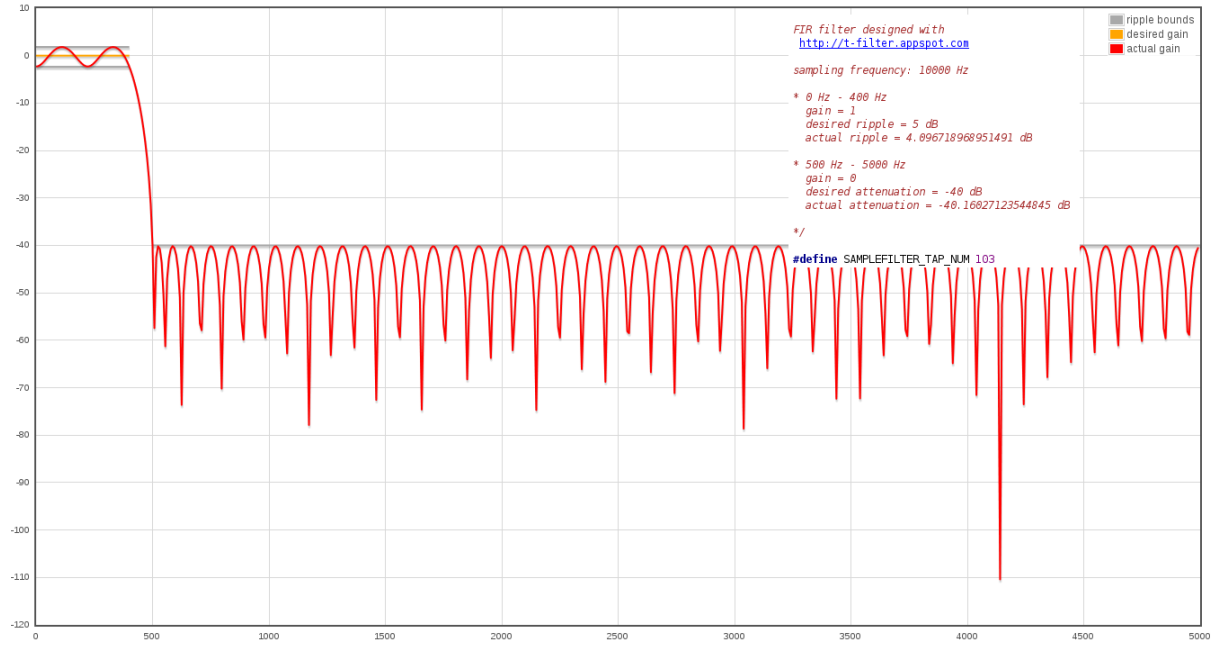


Figure 3.20: FIR filter characteristic

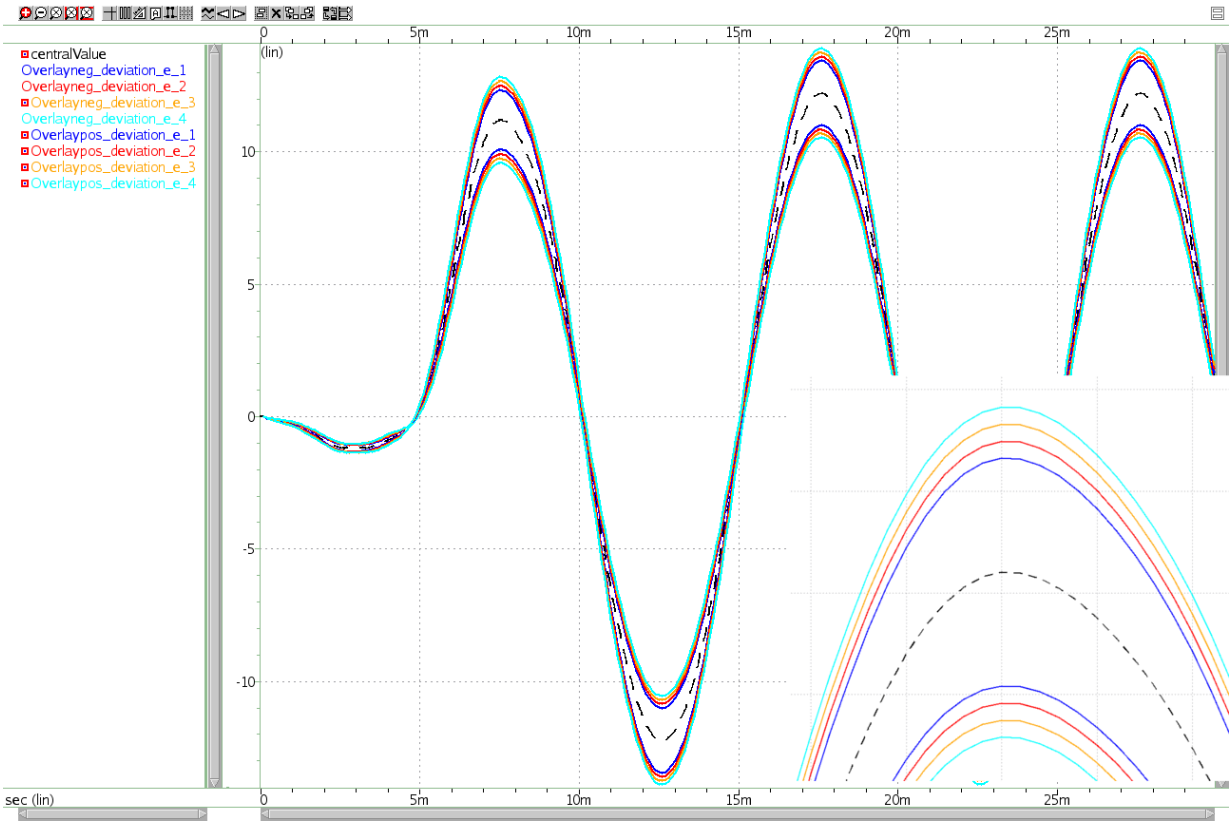


Figure 3.21: The output of the filtered signal. A zoomed section of the first positive maximum in the signal is shown in the lower right corner.

its storage in waveform files can potentially take up a large part of the total simulation runtime. Especially in a highly complex system, this motivates a goal-oriented use of analysis processes as presented in Section 4.2. Planning, estimating the effort and redefining verification processes (focusing on temporal and structural domains) can make a significant contribution to shortening the total verification time (simulation runs plus analysis processes).

2. ***Lookup processes in data structures:*** I briefly discussed this issue in Subsection 3.1.1 already. As mentioned for the storage of references to instantiated objects in AAFs I use appropriate C++ 11 container types. The selection of the container has been optimized during the framework development. The **unordered map** container has shown to be particularly efficient for storing deviations in AAFs. Finding stored items (references to deviation objects) is many times more efficient by using a hash function than using a standard STL **vector** container. Tests have shown that using **unordered map** instead, results in an improvement of a factor of 3-5 (depending on the application and system complexity). The use of arrays and scalar variables for deviations, as implemented in Grabowski's framework [Gra09], is of course even faster. In general, the implementation of Grabowski is well optimized concerning simulation runtime. Unfortunately, *SESYD* framework is slower by a factor of 4-7 (depending on the simulation setup). Nevertheless, the *SESYD* framework outweighs the benefits of an extensible data structure and the associated better system analysis capabilities. In the future, the *SESYD* implementation can also be improved, for example by further optimization of soft- and hardware architectures.

3. ***Algorithmic performance:*** A new approximation scheme for non-linear operations in the implemented framework is the interval-exact approximation (see Subsection 3.2.2) I will discuss the algorithmic performance using this approximation algorithm as an example. A mathematical challenge at the interval-exact algorithm is to find an exact result interval if both operands of the multiplication contain correlated deviation symbols. By evaluating the Hessian Matrix of the n-dimensional extrema problem (each deviation symbol is a dimension with a limited interval of $[-1, 1]$) it can be proven that minima or maxima of the parabola can only be located at the corners of the deviation symbol space. So if G Symbols are correlated, there are $G \cdot 2^{G-1} - 1$ candidates for being minima or maxima of the non-linear result. Each minimum or maxima position calculation requires $6 \cdot (G-1) + 7$ floating-point operations. So the total algorithmic complexity for the calculation of maxima and minima positions at G correlated affine deviation symbols is $(G \cdot 2^{G-1} - 1) \cdot (6 \cdot (G-1) + 7)$. If minima or maxima locations are not inside of the $[-1, 1]$ interval all corners which result from $-1, 1$ permutations of the deviation symbols must be taken into consideration as well. At G correlated symbols these are 2^G corners. The calculation of minima and maxima values requires again $2 \cdot G + 1$ floating-point operations. The given measures are intended to give the impression that the value G(number of correlated deviation symbols) is mainly responsible for the execution time of the algorithm. For example, the execution time for 1000 multiplications approximated with the interval-exact method is given in the following table:

The result of the execution time evaluation is that at increasing number of correlated deviation symbols the execution time is increasing at about a power of 2.4. The test has been executed on a dual-core x64 desktop machine.

Table 3.4: Timing complexity of the discussed interval-exact approximation algorithm

G	Execution Time for 1000 Multiplications
2	104ms
3	250ms
4	530ms
5	1103ms

The *SESYD* framework scales satisfactorily concerning the number of integrated uncertainties. An inconspicuous example of this issue is given in Section 4.8. For the generation of the spectrum, an AAF signal is processed by a FIR filter with a history length of 100 samples. The filter coefficients themselves are AAFs each including an individual deviation symbol, representing 100 different uncorrelated uncertainty causes. The largest system model tested in this thesis is the PLC system described in Subsection 5.3. The system contains 34 SystemC modules, and the AAF simulation takes 31 different uncertainty causes into account. Based on the simulation results and of course the utilized runtime, I expect good scalability and usability for models having increased complexity and comprehensiveness.

Further potential improvements in runtime and scalability would be an adaptation for multi-core processors (both the *SESYD* framework and the used simulation core, in this case, SystemC would have to be adapted). In an application perspective, many calculations could be executed in parallel and distributed to multiple physical cores (e. g. multiplication of AAF matrices, approximation algorithms, etc.). The resulting added value in performance and simulation time would have to be determined in particular (see Section 6.2).

3.7 Framework Expandability

As the first statement in this section, I want to explicitly mention that the implemented *SESYD* framework is open source, freely available and licensed under LGPL. The framework is programmed in C++ (minimum requirement of C++ 11) and uses well-documented STL features. The source code can be compiled for various platforms (e. g. linux, windows - see Section 4.10) and target architectures (e. g. ARM architecture). The code is also documented using Doxygen [37] inline code documentation.

In the implementation, I tried to follow a strategy that the code is well structured and any data is well encapsulated into a corresponding C++ object. These objects can be easily enriched by additional member variables and methods. Also, object-oriented concept as derivation, templates, etc. can be applied using the implemented framework objects. The full framework is partitioned into functional modules including a set of the mentioned objects (see Figure 3.1). New user specific modules can be added under respecting some common functions. These common functions such as logging, configuration file loading, printing, naming service, symbol management etc. are initialized by calling the **initSesyd** method. Interface descriptions for these common modules can be found in the Doxygen documentation. The idea of this architecture is to provide a toolkit where single functional modules can be added to the framework (more or less) independently (similar to packages in a Linux operating system). So, for example, if a user develops a new AAF

approximation algorithm the interface to the approximation management has to be used, and the approximation can be integrated as an extension of the already implemented math module (see Figure 3.1).

There are some open issues in the expressiveness using AAF for modeling deviation effects. To close that potential gap, extensions, strictly speaking, merging of co-existing AAF frameworks can be identified as future tasks. Potential framework extensions are:

- Modeling of enhanced functions. As already mentioned in Section 3.2 the set of mathematical non-linear operations is at the time of thesis writing, reduced to multiplication and division. The framework developed by Grabowski [Gra09], includes the computation of approximations for many other operations. Potential extension of the *SESYD* framework is to use the code of Grabowski as a template to expand the operator set of the *SESYD* framework.
- The theory of AAFs can be extended by a probabilistic arithmetic. Thus, standard deviations are included in AAFs and arithmetic operators have to be extended. We described the basics of such forms in [GR17].
- Already published extensions to the standard AAFs used in this thesis are quadratic AAFs [MT06], modified AA in tensor form [SLMW04], the theory of using zonotopes for uncertainty representation [OBJ05], etc.
- As already reported the concept of Affine Arithmetic Decision Diagrams (AADDs) is very powerful for modeling digital and control flow behavior in systems. Thus extending the *SESYD* framework with the AADD concept is a very ambitious extension [Rad16, RGJR17]
- A potential extension which is related to the last item is to release the constraint that deviation ranges are located symmetrically around its nominal value (e. g. 5 V+2%,−3%). Even more enhanced deviation effects are given by unjoint tolerance windows discrete tolerance probability distributions where the central value might not be enclosed be the corresponding tolerance window.

4 Analysis Techniques Based on Traceability Features

In this chapter, I describe enhanced analysis features based on Affine Arithmetic modeling and simulation methods. They are mainly based on the deviation symbol traceability features facilitated by Affine Arithmetic forms introduced in Subsections 3.1.3 and 3.1.2. Besides the description of the methods itself, I discuss and highlight issues why they are not efficiently implementable using IA or multi-run approaches. The presented analysis techniques are integrated into the analysis module of the *SESYD* framework as illustrated in Figure 3.1. These enhanced processes have a significant added value compared to basic forward and backward tracing as described in Subsection 3.1.3.

4.1 Analysis Techniques facilitated by AAF Simulation

Modeling parameter deviations by AAFs require in general an enhanced effort compared to Interval Arithmetic and multi-run techniques (more complex algorithms, approximations, symbol management, etc.). But, in the perspective of applicable analysis procedures, this approach faces potential possibilities for more detailed, enhanced insight into the system's behavior. As proposed in Section 2.2.4 AAF mainly overcomes the so-called traceability problem. Thus, the following enhanced analysis functions, in detail described in the next subsections, are based on these traceability properties facilitated by AA.

- Ratio-metrics (absolute ratios, correlation measures, quality metrics)
- Assertion driven tracing
- Temporal tracing
- Structural analysis (structural tracing, localization of attenuation and gaining effects, cause and effect analysis, deviation hot-spot detection)
- Sensitivity analysis (sensitivity based tracing)

As mentioned in the introduction of this chapter most of the itemized analysis functions rely on the properties exclusively given by AAFs. For IA and multi-run approaches the effective implementation of the itemized analysis procedures is due to the lack of traceability information not

possible. Besides the required traceability properties of AAF itself, object-oriented implementation characteristics of the *SESYD* framework rises further advantages in implementing enhanced analysis features (e. g. object to object linking by holding pointers to referenced object instances).

Documentation of uncertainties
In general, documentation of design and verification information is essential for a successful design project. AAFs allow documenting deviations of parameters and properties in an uncertainty matrix. In rows potential propagated uncertain values are listed. In columns, the corresponding deviation information given from the formal AAF representation are listed. These are:

- The nominal value x_0, or a range in which nominal values can lie.
- For each basic continuous uncertainty ϵ_i and its according partial deviation value x_i.
- Either a safe bound for higher-order level effects (x_{n+1} of an AAF) or a comment stating this is 0 or an unknown uncertainty ("u") that must be considered as a risk.
- Additional documentation, such as the location, level and nature of the uncertainty (see Subsection 2.2.2).

This uncertainty matrix can be extended by further probabilistic information if enhanced semi-symbolic forms as described in [GR17] are used. Discrete uncertainties lead to different system modes for which, if necessary, rows are repeated if the entries are different. Figure 4.1 gives a template for an uncertainty matrix.

Parameter, input	Nominal value	Uncertainty (per std. unit) associated with varations in					Higher-order effects	Location, level, nature
		∅	P	V	T	···		
<ParName1>	x_0	x_1	x_2	x_3	x_4	···	x_{n+1} ,0 or "u"	internal, external ...

Figure 4.1: Uncerteinty matrix template. ∅ = uncorrelated 1st-order effects; P=Process variations; V = voltage variation; T = temperature variation.

Uncertain signals, as (maybe large) sequences of uncertain AAF samples (see equation 2.3), are a good data basis for analysis tools. For that purpose, we parameterize signal templates, tolerance schemes, or specifications in temporal logic [GR17]. Uncertain signals in such a representation complement the uncertain parameters and properties in an uncertainty matrix. In more complex, hierarchical designs, the information can be integrated into tools and languages for systems engineering, and requirements- and risk-management such as SysML [32] or DOORS [36]. In that way, correlations and risks can be documented for overall system design.

The specification of an uncertainty matrix is in general also possible for uncertainty models using interval Arithmetic and range definitions for multi-run techniques. But correlations of uncertainty with a deviated physical value and the following contribution to uncertain system behavior is not modeled in detail. Deviations are stated as tolerance ranges obfuscating associated causes. Thus, the traceability and detailed sensitivity analysis (as given by the columns in Figure 4.1) are not possible. However, evaluating an uncertainty matrix guides a designer to a comprehensive evaluation of possible variabilities in a design and formally documenting them.

4.2 Objective-driven System Analysis

The layered modeling approach proposed for range based system analysis is in principle illustrated in Figure 1.4. A block oriented highly abstracted application model is stepwise refined towards a physical representation. The depth of refinement depends on the required insight to satisfy specified verification and analysis goals. Similar to the refinement of models a refinement procedure in an analysis perspective is proposed. The objective of analysis may be connected with the context of the system design. Some selected examples are:

- For mobile devices energy efficiency of components and low-power design is essential. Thus modeling and associated analysis procedures focus on identifying potential energy savings within the design (sleep-mode periods, leakage current, etc.).
- Safety critical systems (automotive) have to satisfy hard robustness properties (e. g. for standardization according to ISO 26262). Thus, enhanced verification effort will be invested on cause-impact analysis of deviations and failure and effect analysis.
- Precise analog RF circuits as used in transceiver frontends rely on high accuracy. Analysis objectives may be sensitivity and frequency impact analysis concerning the used analog components and software parts executed in DSP hardware.

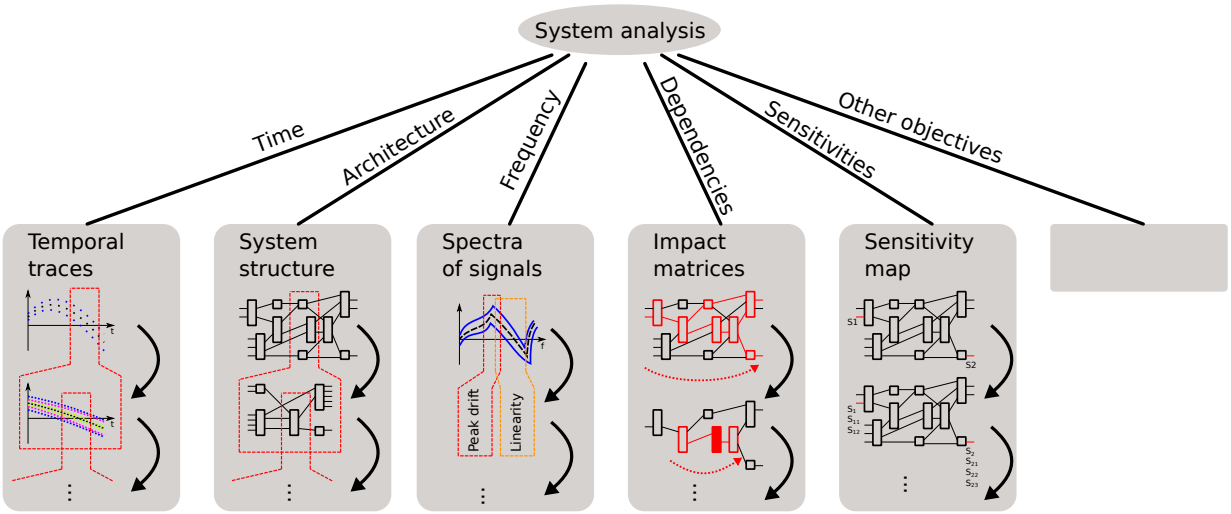


Figure 4.2: Selected initial objecives of AAF analysis, including potential refinement steps.

For the context of this thesis, I propose the introduced analysis refinement for AAF deviation modeling. As initial analysis objectives, I specify the directions as illustrated in Figure 4.2.

- An analysis objective in time, results in temporal traces. For coarse analysis, the temporal trace of a signal is sampled in low resolution which allows coarse identification of interesting periods in simulation time. In subsequent simulation runs the sampling density is increased.
- An analysis goal can be verifying the structural domain of a model. High level models are stepwise refined to evaluate structural details. Analysis goals may target potential structural modifications to increase properties like robustness, safety, etc.

- Frequency domain analysis is a potential objective for systems driven by periodic signals. In refinement steps, characteristic parameters of the spectrum (including variation bounds) may be evaluated.

- For enhanced analysis of variation dependencies, impact matrices are evaluated. In principle, they describe dependencies of signals. Dependency values are represented by specific measures. In refinement steps potential "bad block" unspecified gaining an objected partial deviation are identified. In this thesis this procedure is called structural deviation hot-spot detection.

- In contrast to the described dependency objective a sensitivity map directly holds partial deviation values (In the AAF context sensitivities are represented by corresponding partial deviation values). Within refinement steps potential correlations to other sensitivities or dynamic behavior over time may be identified.

- Besides the itemized objectives also other customer specific analysis directions can be defined. However, in analysis processes objectives are given by a mix of the described objectives (e. g. sensitivity analysis specified in a specific simulation time window where significant events may occur).

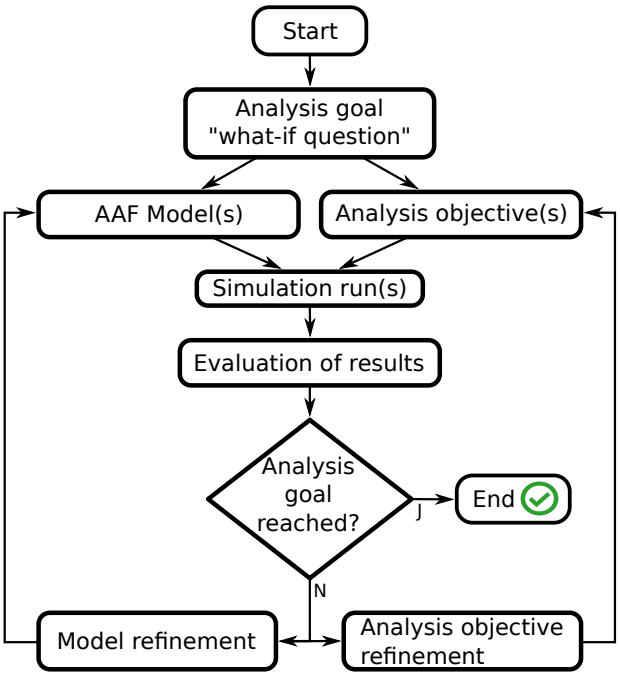


Figure 4.3: Proposed AAF based system analysis flow including model and objective feedback loops.

The definition of analysis objectives will guide an engineer through the course of AAF verification and system analysis (increasing the system insight) in a formal and symbolic way. As illustrated in Figure 4.3 I define a workflow including two feedback loops for model and analysis refinement. A specified analysis goal is representing the interest of the engineer which may be formulated by a "what-if question" (e. g. what is the behavioral impact of the uncertain event XY occurring within the period AB, and which function block has to be optimized to reduce that impact?) Modeling and objective refinements are executed as described above by increasing the level of

detail, decreasing the level of abstraction and focusing the following iterations in a direction towards satisfying the specified analysis goal.

In principle, the proposed objective-driven analysis flow is also applicable for IA and multi-run approaches. But due to the basic traceability features facilitated by Affine Arithmetic assertions specific refinement steps are just enabled or may be executed much more goal-oriented (e. g. Analysis with the objective of enhanced dependency evaluation). In essence, full traceability given by AAF makes the proposed approach efficiently usable and significantly enhances verification including uncertainties.

4.3 Ratio Analysis and Deviation Metrics

As illustrated in Figure 4.2 comparing partial deviations each other is essential for every analysis objective [WHW15]. Just AAFs provide precise information about how a given range is composed by sub-ranges representing a specific uncertainty cause. In principle, measures presented in this section assess the value of deviations relative to other quantities (deviations) in the system model. The calculated measures are presented by an accurate matrix forms and graphical map illustrations. Depending on the verification goal metrics are monitored over a specified period of time or system sub-structures and guide the verification engineer in model and analysis refinement steps.

4.3.1 Absolute and Relative Deviation Analysis

According to equation 2.2 a set of partial deviations is included in an Affine Arithmetic form. They indicate how the total deviation represented by the bounds is portioned. The bounds in general, are given by the radius of the corresponding form (see equation 2.22).

As proposed in Subsection 2.2.4 a partial deviation, strictly speaking, its associated deviation symbol stands for a modeled physical uncertainty. An absolute impact measure is defined by a partial deviation value for each point in time. Static deviations have a constant absolute deviation impact where dynamic deviations have a time dependency.

Thus, more interesting is a relative deviation impact measure. In the basic form, each included partial deviation is rated to the radius of the corresponding AAF. Mathematically these relative deviation impacts $x_{rel,i}(t)$ for a single point in time t is expressed as:

$$x_{rel,i}(t) = \frac{|x_i|}{rad(\hat{x})} \forall i \in \mathcal{N}_{\hat{x}} \quad \text{with} \quad rad(\hat{x}) = \sum_{i \in \mathcal{N}_{\hat{x}}} |x_i| \tag{4.1}$$

Static deviations do not have a constant measure as proposed for an absolute metric. Their relative impact is variable and may get high percentage values if at a specific timestep other dynamic deviations are small. The relative deviation measure has the advantage that dominating

deviation impacts can be easily evaluated for each timestep. To illustrate the results of time-stepwise relative deviation impact calculation a temporal plot can be generated. Absolute and relative uncertainty calculation functions, as well as plotting features, are fully implemented within the analysis module of the *SESYD* framework.

Potential other, application specific measure rate partial deviation values to other characteristic values in the system model (e. g. the maximum occurring partial deviation value within the simulation process).

As discussed the relative and absolute impact measures informs about the partitioning of the deviation range at a given point in time. For impact evaluation over a time window ($t = t_0$ to $t = t_{end}$) (e. g. 0 to the end of simulation $t_{sim,end}$) the corresponding relative impact measures $x_{rel,i}(t)$ get integrated:

$$\gamma_i = \int_{t_0}^{t_{end}} x_{rel,i}(t)dt$$

The result is a global (over the specified time window) measure informing about the accumulated impact of a selected deviation x_i.

A strict requirement for the evaluation of the proposed absolute, relative and accumulated measures is the knowledge of how input deviations affect the analyzed AAF. Strictly speaking, detailed deviation tracing information including partial deviation values is necessary. For IA and multi-run methods where this deviation tracing is not featured the presented metrics can not be defined.

Example: As an example highlighting the proposed absolute, relative and accumulated measures I use the AAF signal $\hat{a}(t) = x_0 + x_1\epsilon_1 + x_2\epsilon_2 + x_3\epsilon_3$ with $x_0 = 1 - (2t \cdot cos(4 + 20t) + 2 \cdot t^4)$, $x_1 = 0.2$, $x_2 = 0.5 \cdot cos(x_0)$ and $x_3 = 0.1 \cdot x_0$. Figure 4.4a shows an AAF signal plot illustrating subranges for partial deviations associated with ϵ_1 (blue), ϵ_2 (green) and ϵ_3 (red), located symmetrically around the central value (dashed black). Figure 4.4b illustrates the trace of partial deviation values over time representing an absolute impact measure as described above. Figure 4.4c shows the relative impact measures evaluated over the plotted time window from 0 to 0.8 *sec.* As described the static partial deviation associated with ϵ_2 is not longer a constant line because its rated to the radius (see equation 4.1). Percentage values complement to 100%. Figure 4.4d shows an accumulated representation of Figure 4.4c. Calculated gamma values $\gamma_{1,2,3}$ are the integrated relative deviation values representing the blue, green and red marked area respectively.

4.3.2 Correlation Analysis

In Subsection 2.2.4 I propose that the ability to describe correlations between deviations is one of the most powerful modeling feature facilitated by Affine Arithmetic forms. In the state of the art Section 2.1 I cite a general definition of correlated deviations and describe how different types of correlation (direct, indirect and reciprocal) can be specified [Dro09, p.117]. For general purposes the Pearson Correlation Coefficient is introduced which is a basis for the following correlation measures defined for Affine Arithmetic forms.

According to the definition of an AAF (see equation 2.2) a deviation symbol ϵ_i represents an uncertainty. If a selected symbol is used in multiple AAFs, these AAFs are correlated in general. Thus, a Boolean deviation measure using the principle of deviation backward tracing (see

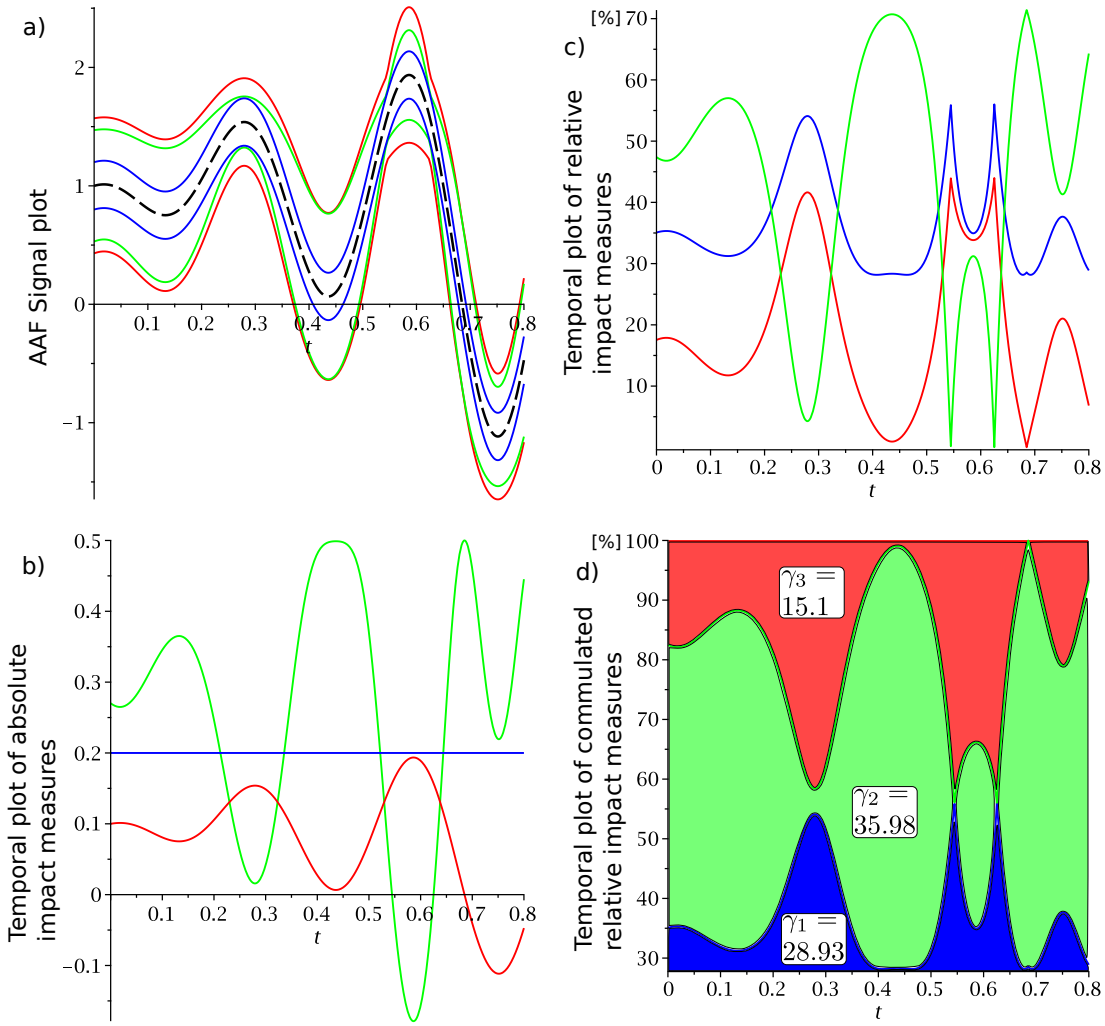


Figure 4.4: Examples for absolute, relative and integrated deviation impact metrics.

Subsection 3.1.3 and Figure 3.8) can be defined. In the *SESYD* framework all symbol objects are collected in a container. For backward tracing a pointer to AAF objects, where the deviation symbols are included in, is stored. In an implementation perspective the so-called **isCorrelated** function checks this stored vector of pointer(s). Mathematically this Boolean correlation measure is described as:

$$isCorrelated(\hat{x},\hat{y}) : \mathbb{A} \times \mathbb{A} \mapsto \mathbb{B}$$

$$isCorrelated(\hat{x},\hat{y}) = \begin{cases} TRUE & \text{if } \mathcal{N}_{\hat{x}} \cap \mathcal{N}_{\hat{y}} \neq \emptyset \\ FALSE & \text{otherwise} \end{cases}$$

A function called **CorrType** is similar to the **isCorrelated** function but returning an element of the set $\{-1, 0, 1\}$, where -1 stands for reciprocal correlated, 0 for uncorrelated and 1 for direct (positive) correlated. **CorrType** has as parameters two partial deviations, and is defined as:

$$corrType(x_i, y_j) : \mathbb{R} \times \mathbb{R} \mapsto \{-1, 0, 1\}$$
$$\text{with } x_i \text{ deviation of } \hat{x} \text{ , } y_j \text{ deviation of } \hat{y}$$
$$corrType(x_i, y_j) = \begin{cases} -1 & \text{if } sig(x_i) \neq sig(y_i) \\ 1 & \text{if } sig(x_i) = sig(y_i) \\ 0 & \text{otherwise} \end{cases}$$

In principle, the result 0 of the **corrType** function is redundant to $CorrType(x_i, y_i)$ with the additional constraint $\mathcal{N}_{\hat{x}} \cap \mathcal{N}_{\hat{y}} \neq \emptyset$.

Further, the **corrType** function is applied for all partial deviations included in the two checked AAFs. The $-1, 1$ values are scaled by the ratio of the partial deviation values and as a result, all these correlation ratios are accumulated. Thus, I get a measure returned by the so-called **corr** function representing a measure for the deviation correlation between two forms. To overcome the problem that this measure returns 0 (representing that two AAFs are completely uncorrelated) if the reciprocal correlation is equal to the direct correlation, I use the absolute values of the partial deviation quotient. The **corr** measure is defined as:

$$corr(\hat{x}, \hat{y}) : \mathbb{A} \times \mathbb{A} \mapsto \mathbb{R}$$
$$corr(\hat{x}, \hat{y}) = \sum_{i \in \{\mathcal{N}_{\hat{x}} \cap \mathcal{N}_{\hat{y}}\}} \left| \frac{y_i}{x_i} \right| \tag{4.2}$$

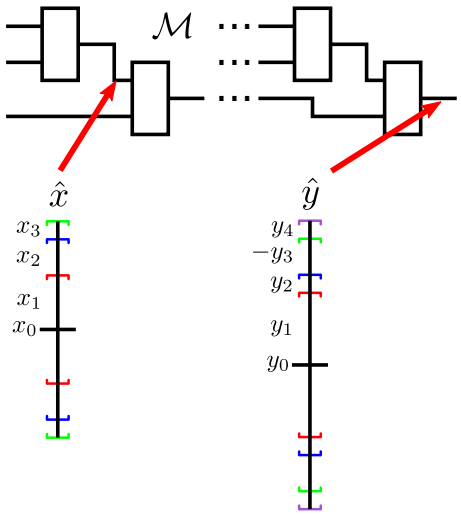
A significant disadvantage of this measure is that the influences of correlated partial deviation values is obfuscated by the accumulation. It would be nice to evaluate which correlated partial deviation is dominating this measure. Thus, I propose a correlation matrix **C** holding this information in its elements. The matrix is defined as:

$$\mathbf{C}(\hat{x}, \hat{y}) = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,j} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ c_{i,1} & c_{i,2} & \cdots & c_{i,j} \end{pmatrix}$$
$$\text{with } i = \{\forall \mathcal{N}_{\hat{x}}\} \text{ , } j = \{\forall \mathcal{N}_{\hat{y}}\}$$
$$c_{i,j} = \begin{cases} \text{“x”} & \text{if } \epsilon_j \text{ is system deviation and } i \in \text{ SymbolSource of } \epsilon_j \\ corrType(x_i, y_j) \cdot \left| \frac{y_j}{x_i} \right| & \text{otherwise} \end{cases}$$

In principle, the (corr) function (equation 4.2) evaluates the absolute sum of the elements placed on the main diagonal. A further interesting information is included in the correlation matrix: According to Subsection 3.1.2 in the *SESYD* framework user and system deviation symbols are introduced. A system deviation symbol results from an approximation, to cover non-linear characteristics. Thus, a partial system deviation is caused by at least two other deviations and is thus however correlated with them (the partial deviation value depends on the approximation

algorithm, see Section 2.3.3). When it comes to the creation of a new system deviation symbol, the *SESYD* framework reminds the symbols of the operators in a SymbolSource structure (see Subsection 3.1.2). Thus, the tracing information is present and can be integrated into the proposed correlation matrix as an element marked as "x".

Example: Figure 4.5 illustrates an example for calculating the correlation measures described above. The two AAFs $\hat{x}$ and $\hat{y}$ are selected signals from a system model $\mathcal{M}$ at a point in time. The symbol ϵ_4 is caused by a non-linear operation between ϵ_1 and ϵ_2 resulting in two "x" elements in the matrix $\mathbf{C}$.



$$\hat{x} = x_0 + x_1\epsilon_1 + x_2\epsilon_2 + x_3\epsilon_3$$
$$\hat{x} = x_0 + 3\epsilon_1 + 2\epsilon_2 + \epsilon_3$$
$$\hat{y} = y_0 + y_1\epsilon_1 + y_2\epsilon_2 - y_3\epsilon_3 + y_4\epsilon_4$$
$$\hat{y} = y_0 + 4\epsilon_1 + 1\epsilon_2 - 2\epsilon_3 + \epsilon_4$$

$$isCorrelated(\hat{x}, \hat{y}) = TRUE$$
$$\mathcal{N}_{\hat{x}} \cap \mathcal{N}_{\hat{y}} = \{1, 2, 3\}$$

$corrType(x_i, y_j)$

	y_1	y_2	y_3	y_4
x_1	1	0	0	0
x_2	0	1	0	0
x_3	0	0	−1	0

$$corr(\hat{x}, \hat{y}) = \sum_{i=\{1,2,3\}} \left|\frac{y_i}{x_i}\right|$$
$$= \left|\tfrac{4}{3}\right| + \left|\tfrac{1}{2}\right| + \left|\tfrac{-2}{1}\right| = 3.83$$

$$\mathbf{C}(\hat{x}, \hat{y}) = \begin{pmatrix} 4/3 & 0 & 0 & x \\ 0 & 1/2 & 0 & x \\ 0 & 0 & -2 & 0 \end{pmatrix}$$

Figure 4.5: An example for calculation the defined correlation measures for two AAFs

4.3.3 Metrics for Deviation Assessment

In the last two subsections, I introduce measures which rate deviation to each other, resulting in absolute and relative metrics as well as correlation statements. The proposed measures within this subsection assess qualitative statements of AAFs and AAF signals.

Signal to Deviation Ratio: First, I define a measure similar to the SNR (Signal to Noise Ratio) known from signal theory. It is defined as the ratio between the signal to the superimposed noise magnitude [Sem05, p. 27]. For AAFs the so-called SDR (Signal to Deviation Ratio) is defined as the ratio between the central value to included partial deviations (absolute values) [Sch13]. The functions are named as *sdr* and sdr_{rad}. Their values can be expressed in a dB scale. The sdr_{rad} computes a signal quality measure considering accumulated deviations (radius) of a form.

$$
\begin{aligned}
sdr(x_i) &= \left|\frac{x_0}{x_i}\right| \\
i &\in \mathcal{N}_{\hat{x}} \\
sdr_{rad}(\hat{x}) &= \left|\frac{x_0}{rad(\hat{x})}\right| \\
sdr_{(rad)}[dB](\hat{x}) &= 10 \cdot log_{10}(sdr_{(rad)}(x_i(\hat{x})))
\end{aligned}
\tag{4.3}
$$

Due to the quotient in the *sdr* functions, there is potential expressivenes problem if the partial deviation x_i gets 0.

Minimum and maximum partial deviatios: As already mentioned the *SESYD* framework provides a snapshot of all signals as a collection of AAF objects for each point in simulation time. Partial deviation information is comprehensively available by iteration over registered AAF objects. Thus, the minimum and maximum partial deviation of a model $\mathcal{M}$ at a selected point in time can be evaluated. They are defined as

$$
\begin{aligned}
mind(\mathcal{M}) &= min(x_{\hat{j},i}) \ , \forall \hat{j} \in \mathcal{M} \text{ and } \forall i \in \mathcal{N}_{\hat{j}} \\
maxd(\mathcal{M}) &= max(x_{\hat{j},i}) \ , \forall \hat{j} \in \mathcal{M} \text{ and } \forall i \in \mathcal{N}_{\hat{j}}
\end{aligned}
$$

More specific, is to evaluate minimum and maximum deviations of a single selected uncertainty (corresponding deviations are associated with the equal ϵ symbol).

$$
\begin{aligned}
mind(\mathcal{M}, i) &= min(x_{\hat{j},i}) \ , \forall \hat{j} \in \mathcal{M} | i \in \mathcal{N}_{\hat{j}} \\
maxd(\mathcal{M}, i) &= max(x_{\hat{j},i}) \ , \forall \hat{j} \in \mathcal{M} | i \in \mathcal{N}_{\hat{j}}
\end{aligned}
$$

In essence, *mind* and *maxd* values are two examples of reference values for rating partial deviation values within AAFs (see relative deviation measures). In an analysis perspective, this is potentially more expressive for the assessment of partial deviations. For future extensions and user-specific analysis procedures also other reference values can be defined (e. g. average of all AAF central values which are chained as a specific computation path).

Figure 4.6 shows a system model $\mathcal{M}$. A special focus is set to partial deviations associated with the symbol ϵ_1. The maximum ϵ_1 related partial deviation holds the AAF $\hat{x}_4$. As described all other ϵ_1 deviations can be rated using $x_{4,1}$ as a reference. Partial deviations can be clustered by using constraints, formulated as assertion statements, defined on rational measures. This is illustrated in blue in Figure 4.6.

Example:
Figure 4.7 shows an example. The system model $\mathcal{M}$, having two functional blocks which are connected by three AAF signals. The proposed measures for *sdr* and sdr_{rad} are computed for each form as well as *mind* and *maxd* for the model.

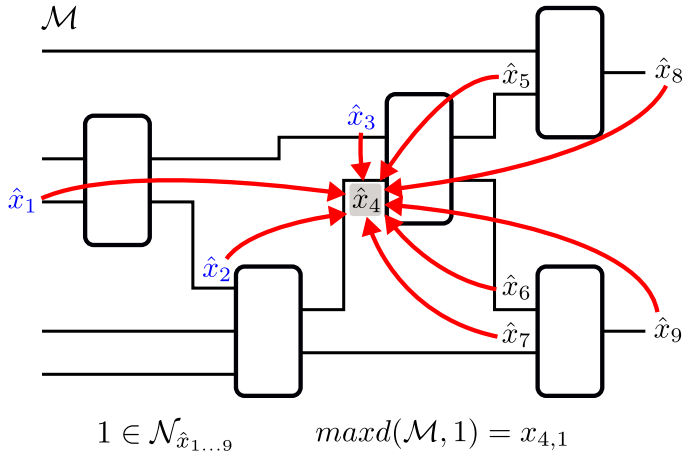


Figure 4.6: Illustration of a model $\mathcal{M}$ where deviations associated with ϵ_1 are rated to the minimum partial deviation associated with ϵ_1 of the full model.

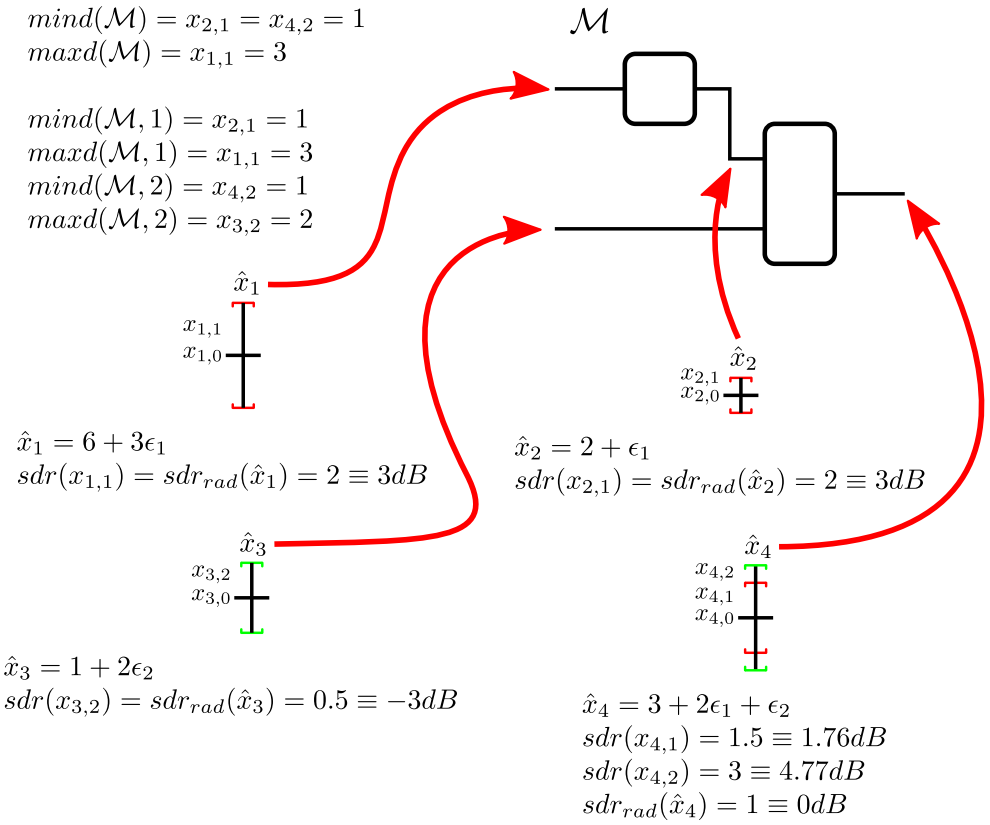


Figure 4.7: System model $\mathcal{M}$ and computed deviation assessment measures.

4.4 Assertion Driven System Analysis

The concept of ABV described in Section 2.5.3 is extebded for handling value ranges (strictly speaking Affine Arithmetic forms). As illustrated in Figure 4.8-a and -b two different approaches can be taken.

The simulation and verification architecture shown in Figure 4.8-a has been published in [RSRG12a].

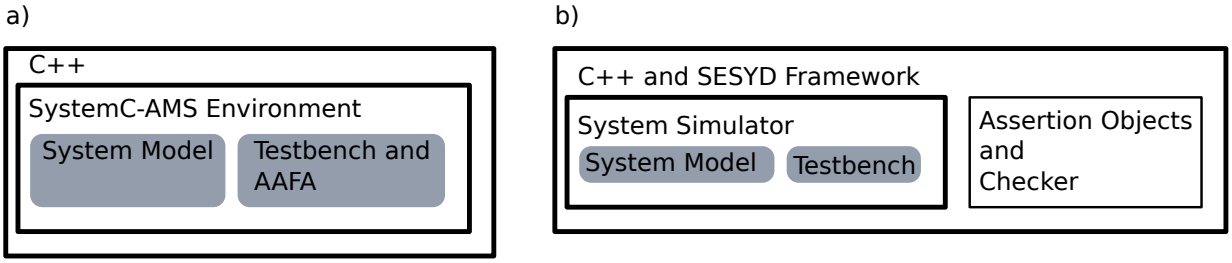


Figure 4.8: Assertion checking approaches [RSRG12a] andimplementation in the *SESYD* framework.

The concept of Affine Arithmethic assertion is introduced which redefines assertion operators accepting Affine Arithmetic forms at their inputs. The system model, the testbench as well as Affine Arithmetic assertions are strictly integrated into the SystemC AMS simulation environment. Thus, formulated assertion statements are compiled to SystemC AMS checker modules [RSRG12a]. Variables included in assertion formulas are directly connected to corresponding signals and operators are represented by predefined SystemC AMS processes. The AAFA toolset contains templates for defining step response, rise/fall time, slew rate and operational range properties. Therefore, standard operators in time and frequency domain are provided. Due to the open framework structure, user-defined operators can be integrated. An assertion statement is computed based on a layered structure similar to the one described for PSL in Subsection 2.5.3: signal layer, Boolean layer, temporal layer, and a verification layer. In [RSRG12a] we demonstrate using AAFA for verifying a IIR (Infinite impulse response) filter example.

As a second approach, which is implemented in the *SESYD* framework, I propose a simulation and verification structure shown in Figure 4.8-b. The main difference is that assertion checking is not integrated within the system simulation environment. At approach a) the system model and the testbench use features of the system simulation tool (MoC, process scheduling, channels, etc.), where the functions of the *SESYD* framework (including assertions and the checker functions) are entirely independent and defined in pure C++. Thus, an assertion statement is not translated to a behaviorally equal set of SystemC AMS modules, but evaluated by a C++ expression parser.

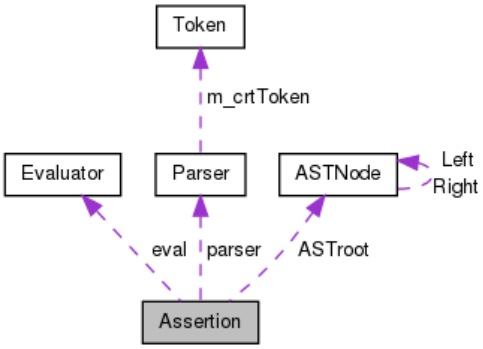


Figure 4.9: Class collaboration diagram for the **Assertion** class.

Features for the proposed assertions checking in the *SESYD* framework are defined in several auxiliary classes. The central one is the **assertion** class. Its interaction is illustrated in the collaboration diagram Figure 4.9. The constructor of an **assertion** object takes the assertion formula as a **string**, which is handed over to a parser object. Functions and characteristics of the implemented parser are discussed in further detail in the next paragraph. The output of a

successful parse process is an AST (Abstract syntax tree) which root node is stored as a pointer in the assertion object. For validation and illustration purposes this AST can be plotted to a ***.dot** file. An **evaluator** object computes the result of the specified assertion. This computation is triggered manually by calling the **evaluate** method of the assertion class.

For assertion parsing, the first approach was to use the well-established free library ***muParser*** [1]. MuParser is proposed for parsing mathematical expressions in an effective and fast way. Unfortunately, the available library is limited in just accepting numeric datatypes [1]. The ability to use other datatypes (as AAF) is not implemented. Thus, I decided to develop a custom assertion parser which is fully extendable in types and operators. As a starting point, I used the tutorial (and following linked articles) presented in [2]. The full parser has a layered architecture, each representing a specific type of operators - numerical, relational, logical and temporal (illustrated in Figure 4.10) The associated grammar of a layer is given on the right side of Figure 4.10. A parse-process starts its evaluation at the top layer, checking first for a temporal operator: **A**, **O**, **P** or **epsilon** if there is no one present. The grammar of the numerical layer (well explained in [2]) handles multiplication/division in precedence to addition/subtraction. In this layer **Number** stands for any numerical constant and **Variable** for a pointer to any data. During the parsing process for each detected element according to the grammar definition corresponding temporary **tokens** (see Figure 4.9) are used. Finally, based on token relations the abstract syntax tree of the expression is synthesized, and its root node is stored in the **assertion** object.

TEMPORAL LAYER	ExpressionTemp -> **A** (ExpressionLog) | **O** (ExpressionLog) | **P** (ExpressionLog) | epsilon
LOGICAL LAYER	ExpressionLog -> ExpressionRel ExpressionLog1 | **(** ExpressionRel ExpressionLog1 **)** ExpressionLog1 -> **&** ExpressionLog | **|** ExpressionLog | epsilon
RELATIONAL LAYER	ExpressionRel -> ExpressionNumeric ExpressionRel1 ExpressionRel1 -> **<** ExpressionNumeric | **>** ExpressionNumeric| **<=** ExpressionNumeric | **>=** ExpressionNumeric | epsilon
NUMERIC LAYER	ExpressionNumeric -> Term Expression1 Expression1 -> **+** Term Expression1 | **-** Term Expression1 | epsilon Term -> Factor Term1 Term1 -> ***** Factor Term1 | **/** Factor Term1 | epsilon Factor -> **(** ExpressionNumeric **)** | **-** Factor | **Number** | **Variable**

Figure 4.10: Layered Architecture of the custom assertion expression parser for the *SESYD* framework. The right side holds the defined grammar for interpreting expressions of the associated layer.

The checker implemented for this thesis accepts the operators given in red in Figure 4.10. Especially the temporal operators **A**, **O** and **P** are significant. Figure 4.11 illustrates them at a state transition example. As indicated, the system changes its state (S_0 to S_3) at (t_0 to t_3). A green tickmark within the circle means that the stated logical expression q is fulfilled at the corresponding state. Below, the **A**, **O** and **P** operators applied on q are evaluated. In this example the **check** function is called at each indicated point in simulation time (t_0 to t_3).

- **A** - Always: The **A** operator at a specific point in time returns **TRUE** if q is **TRUE** at the current point in time and all previous.
- **O** - Once: **O** returns **TRUE** if q is satisfied at the current point in time or at least at one previous simulation point.

- **P** - Previous: The temporal operator **P** returns **TRUE** if q is satisfied at the previous point in time.

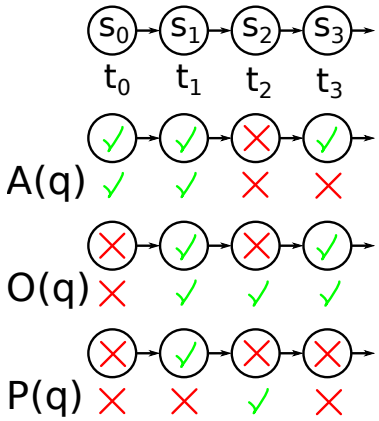


Figure 4.11: *SESYD* Assertion checker temporal operators.

Extension of operators, types, check behavior, etc. are fully customizable, using the already implemented code fragments as templates.

A further instantiated object as shown in the class collaboration diagram Figure 4.9 is the **Evaluator**. It computes the result of a successfully parsed assertion statement by traversing the corresponding AST from bottom to its top. An assertion statement includes abstract names representing data points of the system model. For binding abstract variable names included in the assertion formulas, two variants are possible: For AAFs a naming service lookup can be performed. Numeric types (doubles, integers, etc.) can be bounded manually by pointer assignment. Strictly speaking, a leaf node in the AST holds a constant or a pointer to a data object within the model.

Example:

$$A((Cval > 500)\&(dev1 <= 3))$$

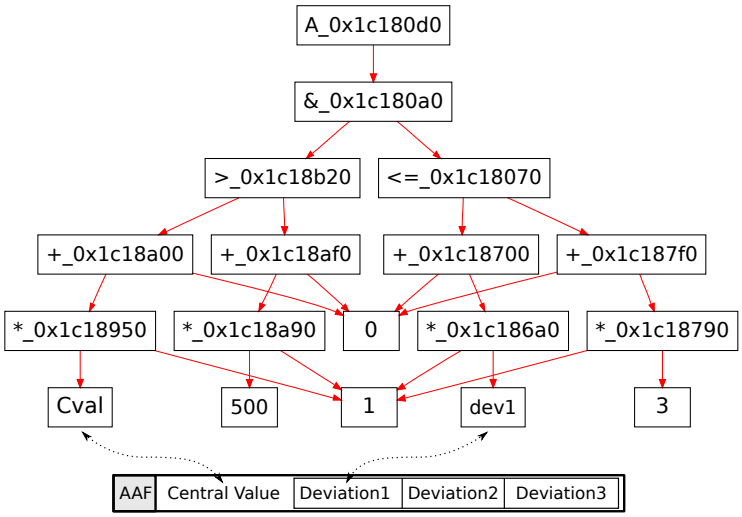


Figure 4.12: Assertion example and the corresponding Abstract syntax tree.

As an example Figure 4.12 illustrates the abstract syntax tree of the assertion formula $A((Cval > 500)\&(dev1 <= 3))$. The assertion specifies that the central value of an AAF must be higher than 500 and its deviation $dev1$ has to be less or equal than 3, at any simulation time (temporal operator **A**). The nodes of the AST hold: operators (and the corresponding address of the node (for dot file representation)), constants (500, 3, 0, 1) and variables which point to the double type variables of the specified AAF. The numerical layer of the parser inserts extra $*1$ and $+0$ nodes to handle parentheses terms and operator precedence correctly (not relevant for this example).

For comparison of the proposed approaches (SystemC AMS and C++ based, see Figure 4.8) the following statements are discussed:

- At the SystemC AMS based approach the assertion formula, strictly speaking, its representation as SystemC AMS modules are tightly bound to the model under verification and its applied Model of Computation. A resulting advantage by the integration of the assertion checker in the simulation environment is a strict synchronization between the assertion check processes and the model computation.
- The SystemC AMS approach strictly requires the SystemC AMS simulator core which decreases flexibility and its integration into other applications.
- At the second presented approach, assertions are defined as C++ objects. The corresponding class includes full information representing the assertion formula as a AST independent of any simulation environment. The check function and other provided methods (management, AST print, etc.) can be called user specifically. Assertion checking synchronized with the execution of the system model is dependent on manual function calls and thus completely defined by the model's verification code.
- C++ assertion representations can be used for any C++ application. Thus, they can be easily managed in data structures, stored in files, handed over to other tools, etc. However, due to loosely binding to a specific simulator tool I use assertions also for objective sensitive triggering of verification processes (e. g. temporal tracing procedures in Section 4.5).
- As reported in Subsection 4.3.3 assertions can be also used for defining constraints for deviation clustering.
- The C++ assertion module integrated into the *SESYD* framework is fully expandable concerning new operators and input datatypes. The chosen grammar and functionalities are not as well tested as commercial or well-established open source assertion frameworks.

4.5 Temporal Tracing of Deviations

New enhanced tracing methodologies expanding basic AAF forward and backward tracing are enabled by the AAF approach. In particular, the analysis of deviation propagation, is significantly interesting. Within this and the next section tracing in temporal and structural domain is introduced and its conceptual implementation in the *SESYD* framework is described. First, I define in general what a "trace" is and introduce associated features.

Definition of an Affine Arithmetic trace:
An AA trace is defined as a sequence of real numbers which represents partial deviation values.

First, I define an auxiliary function called *select* operating on a given Affine Arithmetic form

$$select(\hat{x}(t), \epsilon_i) = \begin{cases} x_i(t), & i \in \mathcal{N}_{\hat{x}} \\ 0, & otherwise \end{cases} \tag{4.4}$$

select as defined in equation 4.4 extracts a partial deviation value with respect to a ϵ symbol. Thus, *select* formally evaluates the corresponding portion of an AAF range associated with a specified uncertainty cause.

Second, the trace sequence itself $\tau(\epsilon_i)$ is defined:

$$\tau(\epsilon_i) =< a_0, a_1, a_2, \ldots, a_k > \tag{4.5}$$

The length of the sequence k may be limited by the defined simulation time or by the analysis setup. Values a_k included in the sequence represent partial deviations in the following described by the usage of *select* functions. As indicated in the equation a trace just includes partial deviations associated with a single uncertainty cause. ϵ_i is a constant function parameter. For the application of the *select* function, there are two remaining possibilities. First, a trace for a selected point in time t, and second for a specific AAF $\hat{x}$ can be defined. These are the two proposed types of traces: temporal and structural.

In this section, I define temporal traces which includes partial deviation values from an AAF at increasing points in time. This is illustrated in Figure 4.13 and can be formally described by the following set of equations:

$$\begin{aligned} \tau_{temporal}(\epsilon_i) =& < a_0, a_1, a_2, \ldots, a_k > \\ a_0 =& \; select(\hat{x}(t_0), \epsilon_i) \\ a_1 =& \; select(\hat{x}(t_1), \epsilon_i) \\ a_2 =& \; select(\hat{x}(t_2), \epsilon_i) \\ & \ldots \\ a_k =& \; select(\hat{x}(t_k), \epsilon_i) \end{aligned} \tag{4.6}$$

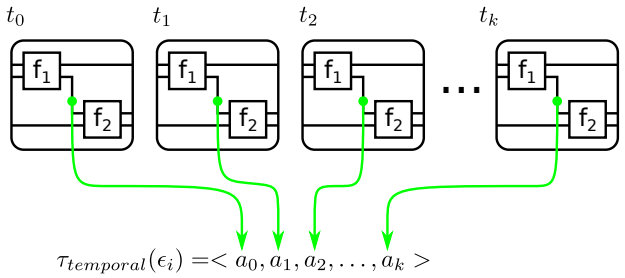


Figure 4.13: Content of a temporal trace are partial definitions of a selected AAF

A temporal trace facilitates a time-specific analysis of a single AAF value within the system. Strictly speaking, a conventional plot of an AAF object, where the central value and accumulated deviation impacts are illustrated, is a special representation of a temporal trace. But features offered by a tracing object implemented in the proposed *SESYD* framework are more enhanced than just recording partial deviation values.

A tracing object within the *SESYD* framework is an autonomous functional (analysis) module independent of the simulation model. Before the start of the temporal tracing procedure, data points are registered, and the tracing behavior is configured. For registering a specific AAF for temporal tracing, at a newly instantiated tracing object the following options can be used:

- In the *SESYD* framework it is possible to hand over a pointer to an AAF object in order to create a trace for each included ϵ symbol in the form, $\tau_{temporal}(\epsilon_i) \forall i \in \mathcal{N}_{\hat{x}}$.
- An accumulated trace of an AAF can be defined including the radius values for a specified AAF (independent of the number of partial deviations).
- A single partial deviation value can be used for tracing, using the *select* function, as defined in equation 4.6.

After a single call of the registering function, the tracing object is locked. Multiple or changing data registration during operation of the tracing object is not provided.

A second configuration option specifies the temporal sampling behavior of the tracing object. One of the following configurations are possible:

- At ***manual sampling*** each call of the **snapshot** function adds a sequence element of the configured data points to the trace.
- A ***constant sample rate*** can be defined. The **snapshot** function is called automatically according to the defined rate in samples per second. The predefined macros **SR_MAX** and **SR_NULL** configured the tracing object to the maximum available sampling rate (depending on the computation period of the model) and deactivates sampling, respectively. For configurations where a tracing object samples the monitored value not at each model computation step $sr \neq SR\_MAX$, a second sequence with equal length holding the corresponding sample timestamps is created.
- For ***focused inspection of importance points*** temporal intervals can be specified associated with a specific sampling rate. Temporal intervals may be specified by giving absolute t_{min} and t_{max} values, or by defining an importance point $t_{important}$ and a temporal radius t_{rad}. Each interval is associated with a specific constant sample rate. Thus, for using this option, a temporal trace holds an ordered set of time intervals and sampling rates. However, this setup also specifies an autonomous start of recording samples.
- A further start option can be defined by using ***assertion statements***. Therefore, an assertion is specified according to Section 4.4 which triggers the start of the temporal tracing procedure.
- For ***memory limited*** simulation environments a static section of tracing memory can be specified within this option. The sampling behavior is defined to sample recent data denser, while historical trace information gets deleted to keep the number of trace elements constant.

As a result a temporal trace may identify points in time when partial deviations exceed unspecified tolerance values or have their minimum or maximum values. The corresponding timestamps can be stored in a set $\{t_{critical1,2,3,...}\}$. These points in time potentially indicate critical states of

the system or event impact where the full behavior may violate system performance or safety requirements. Evaluated points are used for further detailed inspection using a temporal trace at a refined (more specific to this point in time) setup, or for structural evaluation of the deviation cause (analysis refinement - see Figure 4.2).

Results of a temporal trace allow detailed analysis of a deviation cause and its impact on the total radius of an AAF during the specified analysis time. This impact analysis may help designers to evaluate system states where a specific uncertainty cause has a dominant characteristic. For optimization, this may be a potential starting point for decreasing this uncertainty radius and thus majorly improve the system's performance.

A temporal trace sequence can also be integrated over time, similar to the procedure described in Subsection 4.3.1. The temporally limited impact of a partial deviation over the specified trace length is evaluated, indicating the impact of a partial deviation during the analyzed portion of the simulation time.

The first derivation of the sequence of a temporal trace is the sensitivity of the system against the specified uncertainty cause. In this case, the sensitivity is evaluated under special consideration of the configuration of the tracing module (e. g. set sampling frequency). Thus, results may differ from results evaluated from classical AAF sensitivity analysis. The first derivation is reduced to the difference of sequence elements. The focus of the analysis is limited to the period where the temporal trace is specified to execute monitoring.

Example: Figure 4.14 shows a system model $\mathcal{M}$ which is analyzed using proposed temporal tracing objects. The objective of the analysis procedure is to find the maximum positive deviation impact in the AAF signal marked in red, at the configured simulation time window of 0 to 1 s. First, the tracing object **Trace a** is defined. It monitors both integrated partial deviations associated with the symbols ϵ_1 and ϵ_2 respectively. The sampling rate of the trace is set to a rate of 5 samples per second. As a result, we see that the impact of ϵ_2 deviation holds the dominating maximum partial deviation value within the time interval. Thus, I refine the analysis by using tracing object **Trace b**. This reduces the number of samples by constraining the trace activation to periods where the central value is greater or equal to 0.3. Finally, by using **Trace c** I zoom into the area where the maximum of partial deviation is assumed. For numeric evaluation of the maximum value, the sample rate is set to its maximum. As a result, the maximum impact can be found without fully sampling the AAF over the complete 0 to 1 simulation time interval at a high sampling rate (This is illustrated for verification in the red box). For complex systems this process of monitoring and recording all partial deviations can get very time and memory consuming. In contrast, the presented stepwise refined analysis process using temporal traces successes the verification objective in a faster, less complex and more directed (guided) way.

4.6 Structural Deviation Analysis

As a correspondent tracing method to the proposed traces in the temporal domain, structural traces which monitor partial deviations in the system structure are defined. Structural tracing is a path-selective analysis of the system. For its definition, a path $\tilde{p}$ represents an ordered set of signals/segments $s < 1, 2, 3, \dots >$ within the system structure. $\tilde{p}$is specified as $\tilde{p} =< s_1, s_2, s_3, \dots >$;. Signals/segments define the data flow between functional blocks associated with a pointer to an AAF object.

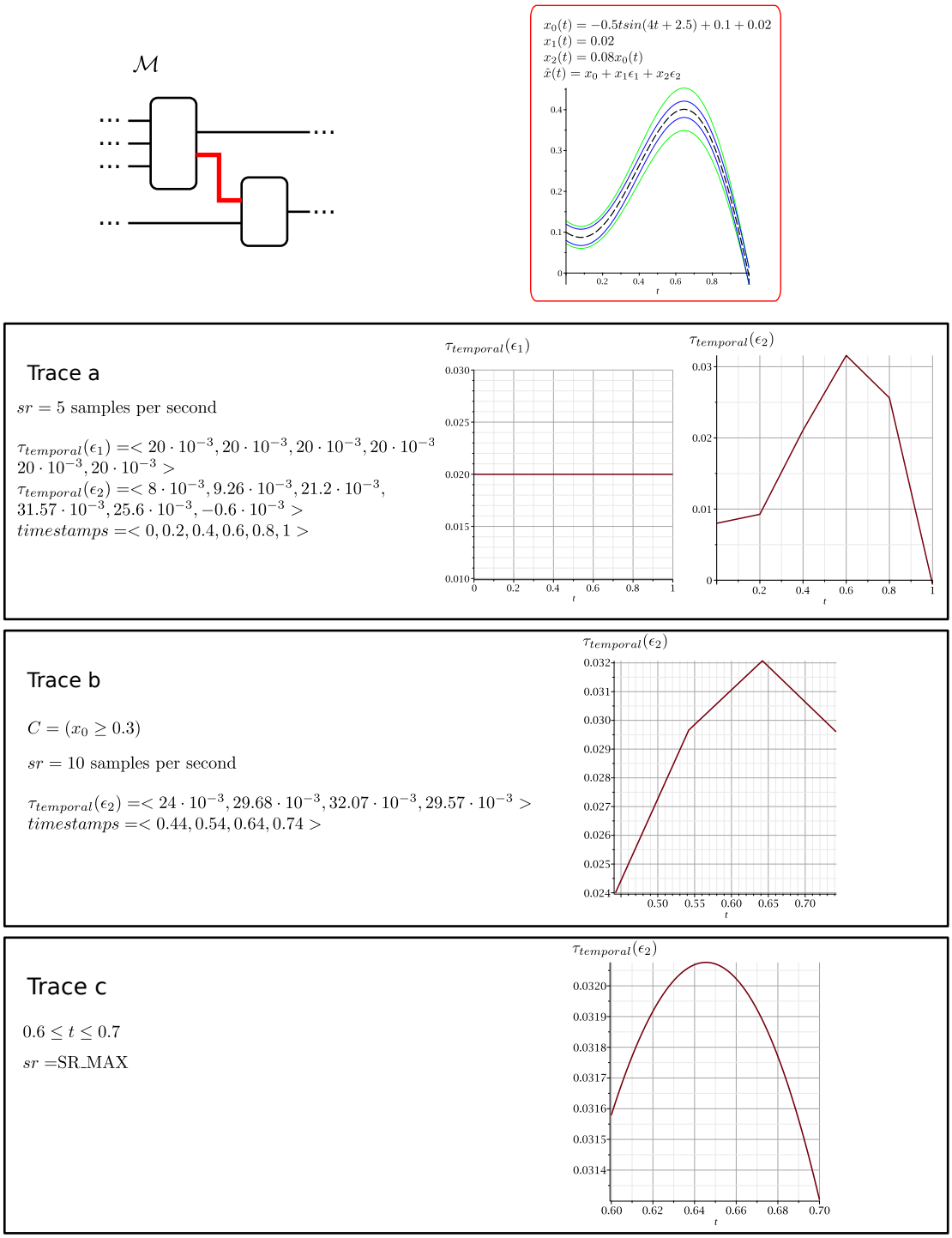
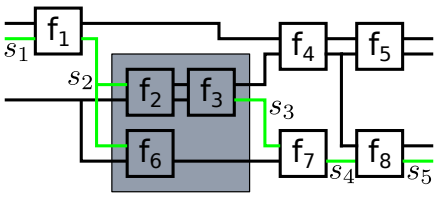


Figure 4.14: Verification of the AAF signal marked in red using a refined sequence of different types of temporal traces.

A path may specified manually or partially guided by the framework:

- For ***manual path definition*** the model is explored by hand. A verification engineer selects signal connections (circuit segments) and adds them to a path structure. This path

is subsequently used for structural tracing. An advantage of manual path specification is the full control of the inspected level of abstraction. Functional blocks can be grouped as illustrated by the gray box in Figure 4.15. Path segments inside specified groups may be omitted which decreases the complexity and focuses following verification steps. Thus, independently of the given model type (white-, gray- or black-box) path definition can be manually abstracted or refined according to the goal of the verification processes. Figure 4.15 shows a defined trace $\tilde{p}$ including segments s_1 to s_5. Functional blocks $f2$, $f3$ and $f6$ are grouped (described as submodels at a refined level of abstraction),because it is assumed that further insight into their connections is not interesting.

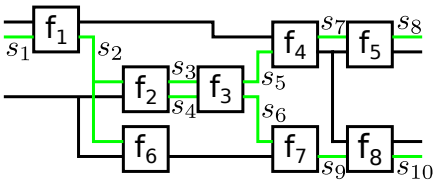


$$\tilde{p} =< s_1, s_2, s_3, s_4, s_5 >$$

Figure 4.15: Manual specification of a path trough a system model.

- ***Guided (semi-automatic) path definition*** is based on analysis of deviation propagation through the system. The path identification process is additionally supported by the module's call sequence information (process scheduling) and static analysis of associated inputs/outputs of functional blocks. For the evaluation of structural information, I used the software package **gSysC** which plots a structural map including all signals and modules within a system [Eib04]. We extended the **gSysC** framework by corresponding macros working with AAF objects. This structural information data includes comprehensive dependency information, and path segments are included into a path structure $\tilde{p}$ in an adequate order. However, some previous simulation steps, used for creating the path information, are required. The impact of a specific deviation cause represented by a deviation symbol and its partial deviation value is analyzed. If the selected deviation impact exceeds a previously defined threshold value at the output of a functional block, the associated output signal(segment) s is added to the path structure. If a deviation has an appreciable impact on multiple outputs the path structure, is forked. As a result, it has to be mentioned that for potential dynamic deviations and their propagation a semi-automatic path definition is associated with a single selected point in simulation time. Figure 4.16 illustrates the results of automatic path extraction in an example system. The analysis of partial deviation values for path extraction starts at the input segment s_1 (specified manually). Output analysis of the block f_1 results that the segment s_2 has to be added to the path set next. The analyzed partial deviation propagates to segments s_8 and s_{10} resulting in four different extracted path specifications $\tilde{p_1}$ to $\tilde{p_4}$.

Similar to temporal traces each signal/segment s_n in a path definition is associated with an Affine Arithmetic form. Strictly speaking, a SystemC signal type holds a pointer to an AAF object representing its affine value. The *select* function is applied to each AAF object associated with the elements of the path definition. A partial deviation value (extracted by *select*) has a specific uncertainty cause ϵ_i and is defined as a constant point in time t. This is illustrated in



$$\tilde{p_1} =< s_1, s_2, s_3, s_5, s_7, s_8 >$$
$$\tilde{p_2} =< s_1, s_2, s_3, s_6, s_9, s_{10} >$$
$$\tilde{p_3} =< s_1, s_2, s_4, s_5, s_7, s_8 >$$
$$\tilde{p_4} =< s_1, s_2, s_4, s_6, s_9, s_{10} >$$

Figure 4.16: Guided specification of a path through the system model.

Figure 4.17). Formally, the sequence elements of a structural trace are defined as:

$$\begin{aligned} \tau_{structural}(\epsilon_i) =< a_0, a_1, \ldots, a_k > \\ a_0 = select(\hat{x_0}(t), \epsilon_i) \\ a_1 = select(\hat{x_1}(t), \epsilon_i) \\ \ldots \\ a_k = select(\hat{x_k}(t), \epsilon_i) \end{aligned} \tag{4.7}$$

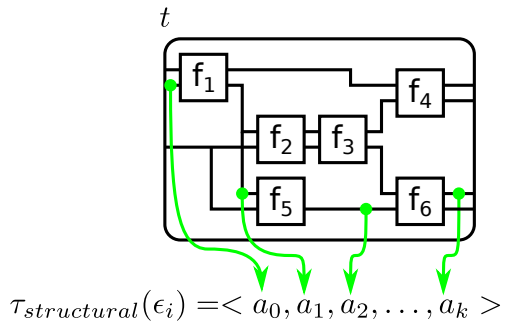


Figure 4.17: Structural trace

Similar to temporal tracing after the simulation process, data monitored by a structural trace is post-processed, and the following conclusions/results may be identified:

A structural trace may indicate data connections and associated chains, where a partial deviation exceeds specified tolerance ranges. The analyzed partial deviation and the point in time are given as parameters to the **select** functions in the trace. Such a check, may also be defined by the violation of a more complex specified assertion statement, as described in Section 4.4. However, a set of critical system signals/segments $s_{critical1,2,3,\ldots}$, representing transferred data between functional modules, are identified. For a verification engineer, this enables detailed analysis of the deviation impact at different portions of a data path through the system architecture.

At an occurring assertion violation as discussed in the previous paragraph, specified paths (guided or manually defined) may be used for tracking the identified critical deviation back to its cause. Evaluated critical segments may also be included in multiple paths. In this case, multiple deviations applied on inputs of the system model may affect the identified critical path. A similar procedure can be used for deviation propagation analysis of a critical segment to the output borders of the system architecture.

As discussed in the last paragraphs at least two threshold values have to be specified (alternatively, two constraints formulated as assertion statements). First, a lower threshold verifies if a specific system segment s is included in the path structure $\tilde{p}$ (semi-automatic procedure). Second, a defined value threshold verifies if a given deviation specification is violated at a point in time t. Strictly speaking, first, it has to be specified if a segment is included into an analysis path and second, it has to be defined whether this path is critical. However, the definition of threshold values for verification in the temporal and structural domain may interact, and the selection of values might become significant for analysis. Thus, I propose a combined, temporal and structural tracing analysis according to a suggested workflow presented in Subsection 4.6.3.

4.6.1 Uncertainties cause-and-effect Analysis

In the prior part of this section, I discuss the basics of structural tracing as trace identification, their representation as a mathematical construct and the evaluation of critical segments within a path.

In this subsection, first, I go one step back and discuss how structural tracing is used for the assessment of a (system) global deviation situation. The goal of this analysis is to evaluate a cause-consequence estimation, by low analysis effort [SCS+00, p.87]. Results are AAFs representing this causes and consequences. For a subsequent structural analysis this identified AAFs are then used as start- and endpoints for path specification.

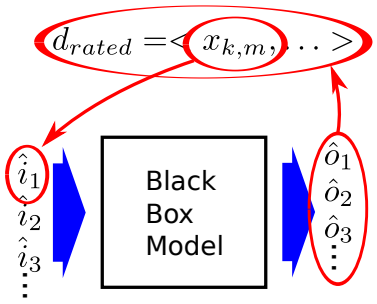


Figure 4.18: Cause-concequence analysis process

The analysis process starts at the inspection of the system's outputs as illustrated in Figure 4.18 $O = \{\hat{o}_1, \hat{o}_2, \hat{o}_3, \ldots, \hat{o}_n\}$. The system model is considered as a black box. The level of abstraction for this process is not significant. Deviations included in output signals are rated by using appropriate deviation measures as introduced in Section 4.3. In addition to the calculated measures, the system specification itself may influence the rating of the output form's partial deviations (e. g. it may happen that a large variation of a selected output is not that significant as a numerically lower deviation at another output, due to the given application). However, this two-step process results in a rating of partial deviations included in output forms of the system, expressed as an ordered sequence $d_{rated} =< x_{k,m}, \ldots >$ where $1 \leq k \leq n$ and $m \in \mathcal{N}_{\hat{o}_k}$ (see Figure 4.18). The highest rated partial deviation is the first potential candidate for analysis refinement (and for optimization). Thus, the associated uncertainty cause ϵ_m which is included in $\hat{o}_k$ is traced to the system's inputs. From a global perspective, this means that I check input AAFs and evaluate the set of AAFs where the identified deviation symbol is included in, see Figure 4.18. This can be expressed as: $I = \{\hat{i}_1, \hat{i}_2, \hat{i}_3, \ldots, \hat{i}_p\}$ where for ϵ_m, $m \in \mathcal{N}_{\hat{o}_k} \wedge m \in \mathcal{N}_{\hat{i}_q}$ is given with $1 \leq q \leq p$.

This cause-consequence (strictly speaking "consequence-cause") tracing information can be presented in tabular form, illustrated in Table 4.1. In this table, partial deviations of output AAFs are presented in rows, and input AAFs in columns. If a selected input (cause) has a consequence in a partial deviation of an output form a red "X" is put in the corresponding cell of the table. The table can be evaluated for a selected set of outputs, or if the process described above is repeated for all included partial deviations in the rating sequence, this results in a complete, rated cause-consequence table.

Table 4.1: Cause consequence table

		$\hat{i}_1$	$\hat{i}_2$	$\hat{i}_3$	...
$\hat{o}_1$	$x_{1,1}$			X	
	$x_{1,2}$				
	$x_{1,3}$		X		
	⋮				
$\hat{o}_2$	$x_{2,1}$		X	X	
	$x_{2,2}$				
	$x_{2,3}$	X			
	⋮				
$\hat{o}_n$					
⋮					

From such a cause-consequence table the following results can be identified enhancing the global overview of the deviation situation in the modeled system or can be used as a basis for a subsequent analysis refinement step.

- Correlations between inputs are represented by multiple "X" marks in a row.
- If a partial deviation is highly rated, associated with a system deviation symbol and thus integrated into the cause-consequence table, the according row has no "X" mark. Further detailed analysis is required with detailed cause tree inspection of the system deviation symbol (see Subsection 3.1.2).
- The presented cause-consequence table is an ordered table. This means that upper rows have more impact considering deviation measures and a manual rating caused by additional constraints given by the system/application specification.
- Output-Input AAF pairs of the table and the associated symbols can be used for automatic or manual structural trace generation. These constructed traces are then used for enhanced local (structural) analysis as described in the next subsection.

4.6.2 Localization of Deviation Causes

As proposed the next step in analysis refinement is a local structural analysis, based on the evaluated cause-consequence results. Information derived from the cause-consequence table (input AAF, output AAF, ϵ-symbol) is used to identify deviation propagation traces through the system (manually or semi-automatic). Within such a path selected segments may violate specified

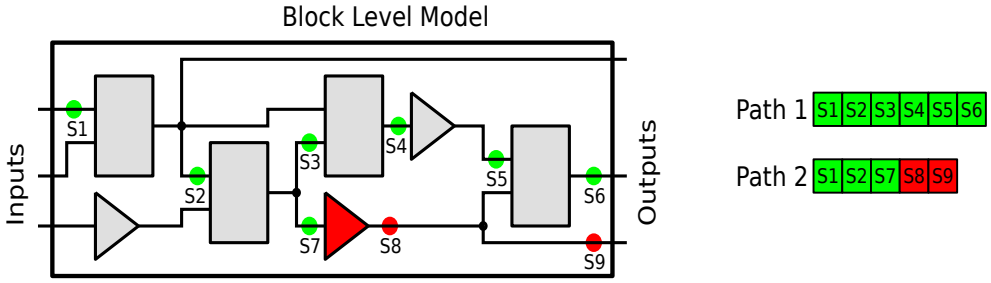


Figure 4.19: Identified traces based on cause-concequence analysis. Segments S8 and S8 violate a deviation specification. But this does not necessarily mean that the amplifier in red is critical.

deviation boundaries (this is decided by a threshold value or an assertion formula). An example is given in Figure 4.19.

In **Path 2** segments S8 and S9 violate a given deviation specification. Segments S represent partial deviation values associated with a selected ϵ_i in a structural trace according to the definition given in equation 4.7. But as clear as the situation may seem, the "bad block" block in the path is not necessarily the amplifier highlighted in red. Only the partial deviation value of segment S7 violates the specification. This motivates a measure for rating the deviation propagation.

The first derivation of a trace which can be defined by the absolute difference of path elements is as a measure qualitatively not expressive. A high deviation change rate from one segment to another does not necessarily mean an attenuation of a deviation (partial deviation can also be negative). However, a more expressive measure is to rate the value of a segment to the previous one in the specified path. Thus, a sequence $\partial\tau_{structural}(\epsilon_i)$ can be defined according to:

$$\partial\tau_{structural}(\epsilon_i) = \left\langle \frac{|a_1|}{|a_0|}, \frac{|a_2|}{|a_1|}, \frac{|a_3|}{|a_2|}, \ldots, \frac{|a_k|}{|a_{k-1}|} \right\rangle \tag{4.8}$$

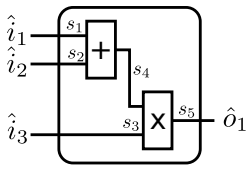
For each element (fraction) within this sequence, representing the deviation propagation from a segment to the next one, the following qualitative statements are defined.

If an element in $\partial\tau_{structural}(\epsilon_i)$
$= 0$: This is the best case that may happen. An included partial deviation is eliminated by the functional block(s) between the analyzed segments
< 1: The partial deviation gets attenuated. Strictly speaking, the variation associated with the symbol (ϵ_i) has at the output of a block(s) a lower value than at the input.
$= 1$: The deviation propagates directly from one segment to the next one, without any attenuation or gaining effect.
> 1: The partial deviation, associated with (ϵ_i) is gained in the functional block(s) between the analyzed blocks. This means, that the output variability concerning the uncertainty represents by (ϵ_i) is increased.
As a consequence, for the evaluation of the worst critical block within a specified path, $max(\partial\tau_{structural}(\epsilon_i))$ has to be evaluated. As specified a critical block includes an internal computation which gains the selected deviation (ϵ symbol for the inspected path) in a way that one or more subsequent path segments violate the deviation specification.

This measure also allows the detection of deviation chaining effects. A chaining effect is defined as a sequence of blocks (included in a path) where no single block is outstandingly gaining a partial deviation. All blocks of the chain, gain the inspected deviation by a similar factor. This finally

leads to a violation of the deviation specification at an arbitrary segment. Strictly speaking, the identification of critical blocks is an anomaly analysis of the deviation propagation sequence. This proposed process of identifying critical blocks is getting significantly more complicated if feedback loops are included in a path (see the ACC example in Subsection 5.4).

Example:
A small example, Figure 4.20 shows a system model with three inputs, one output and two blocks (an adder and a multiplier). The proposed tracing functions and the identification of the critical block is computed for a specific point in time t_0. First, I start with the inspection of the output. It includes two partial deviations, which are in this example rated according to their absolute value. The inspection of according ϵ_1 and ϵ_2 deviation symbols results in the shown cause-consequence table. For each "X"mark in the table, I refined the analysis process by generating a path and a structural trace through the system. Finally, the proposed deviation propagation sequence of the structural traces are computed. The maximum evaluation of the sequences result that the block located between segments s_5 and s_4 (the multiplier) is critical.



$t = t_0$:
$\hat{i}_1 = 3 + 2\epsilon_1$
$\hat{i}_2 = 4 + \epsilon_1 + \epsilon_2$
$\hat{i}_3 = 2$

$\hat{o}_1 = 14 + 6\epsilon_1 + 2\epsilon_2$
$d_{rated} =< x_{1,1}, x_{1,2} >$

		$\hat{i}_1$	$\hat{i}_2$	$\hat{i}_3$
$\hat{o}_1$	$x_{1,1}$	X	X	
	$x_{1,2}$		X	

$\tilde{p_1} =< s_1, s_4, s_5 >$
$\tau_{structural,1}(\epsilon_1) =< 2, 3, 6 >$
$\partial\tau_{structural,1}(\epsilon_1) =< 1.5, 2 >$

$\tilde{p_2} =< s_2, s_4, s_5 >$
$\tau_{structural,2}(\epsilon_1) =< 1, 3, 6 >$
$\partial\tau_{structural,2}(\epsilon_1) =< 3, 2 >$

$\tilde{p_3} =< s_2, s_4, s_5 >$
$\tau_{structural,3}(\epsilon_2) =< 1, 1, 2 >$
$\partial\tau_{structural,3}(\epsilon_2) =< 1, 2 >$

Figure 4.20: Example illustraing structural tracing and the detection of a critical block

4.6.3 Guided Deviation-hot-spot Detection

In this subsection, I describe an approach for combining temporal and structural deviation tracing.

- **Temporal tracing** is an analysis of partial deviation values in the time domain. The objective is to detect critical points in simulation time that potentially correlates with the occurrence of critical events (a point in time where deviation exceeds a specified maximum tolerance).
- **Structural tracing** has the objective to identify critical functional blocks (a block within the system structure which gains a deviation improperly). Thus, it is an analysis process applied along a specified data path through the system structure.

To follow the concept of analysis refinement, integrated into the presented workflow in Figure 4.3, I propose to use both tracing features in a composite way. So-called "deviation-hot-spot detection" is executing temporal and structural tracing in a loop sequentially focusing towards a specific point in time and a part of the system structure.

Figure 4.21 illustrates the hot-spot detection approach at a selected path $\tilde{p_1}$ (highlighted in green) in the block level model. Partial deviation values associated with the deviation symbol ϵ_1 for each

segment of $\tilde{p_1}$ are included in a structural trace $\tau_{structural}(\epsilon_1)$. For illustration on the right side of Figure 4.21, these partial deviation values are plotted as bars. The sequence values of the structural trace are indicated by the red dashed line for one point in simulation time t_0. The structural trace is evaluated for a series of time-slices during the simulation. Thus, each bar in the time domain (illustrates as fading out bar plots) represents a temporal trace $\tau_{temporal}(\epsilon_1)$ for the given segment. The temporal trace is indicated as a blue dashed line for one selected path segment in Figure 4.21. The result is a deviation 3D map in the temporal and structural dimensions for the specified path $\tilde{p_1}$ through the system, and a the deviation symbol ϵ_1. In general critical points in time result in large absolute partial deviation values and critical functional blocks are indicated by large values in the deviation propagation sequence of a structural trace.

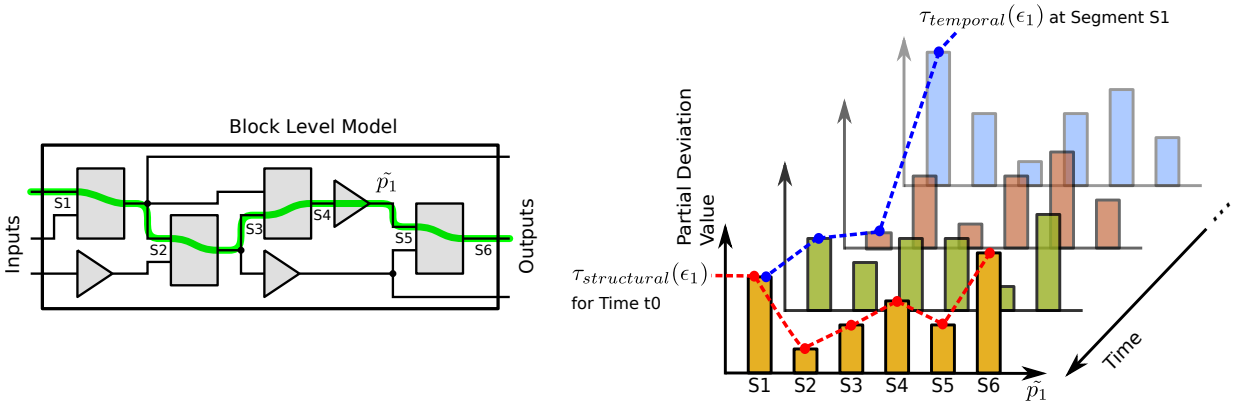


Figure 4.21: Combination of structural and temporal tracing, resulting in a 3D deviation map for an ϵ symbol and a specified path.

For exhaustive analysis, this approach requires high effort. For a full deviation map of a model all deviation propagation paths for all deviations at a high temporal resolution have to be computed and stored for post-processing. This highlights once again the motivation for analysis refinement and objective driven deviation analysis procedures.

Example:

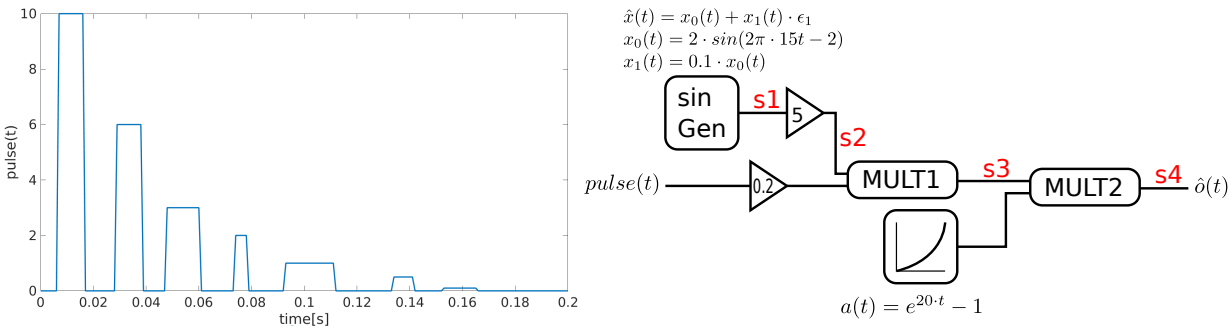


Figure 4.22: Block-level system model of the discussed example and the trace of the pulsed input signal.

In this example a deviation map and the associated hot-spot detection procedure for the model illustrated in Figure 4.22 is shown. The block **sin Gen** generates a sinusoidal AAF signal according to the given equations. There is only one deviation symbol ϵ_1 which propagates through the system on the path $\tilde{p} =< s1, s2, s3, s4 >$. This AAF signal is amplified by a factor of 5

and then multiplied by an input pulse sequence (which is initially multiplied by 0.2). The pulse sequence is illustrated at the left side of Figure 4.22. First, it is multiplied by the sinusoidal AAF (MULT1) and second, due to the decreasing amplitude of input pulses the result is multiplied by an exponentially rising value (MULT2).

It is not directly evident which point in time and which block causes the maximum deviation deviation at the model's output.

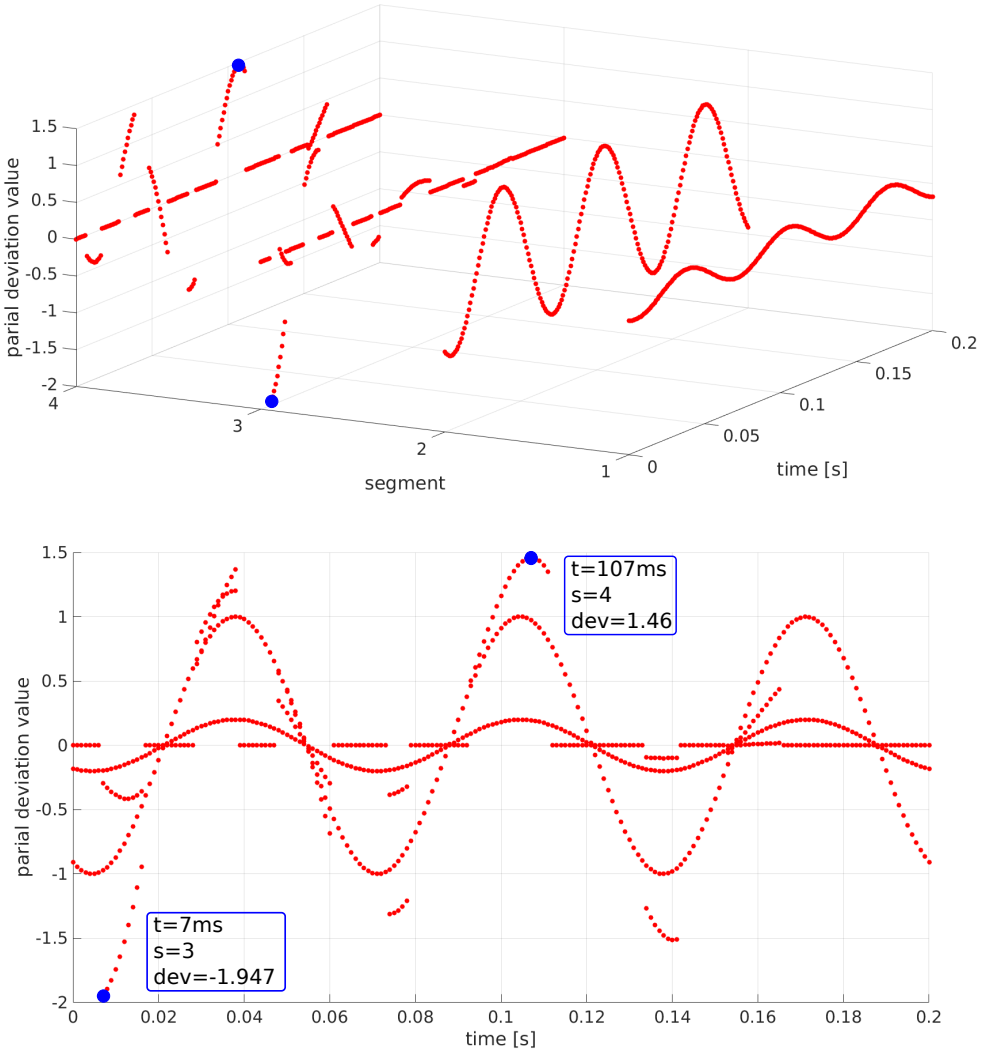


Figure 4.23: Deviation map including markers for two identified points in time and segments within the system structure.

The upper part of Figure 4.23 illustrates the 3D deviation map for the symbol ϵ_1 of the path $\tilde{p}$. For evaluation of the included temporal traces the 3D plot is rotated to a 2D time vs. partial deviation value, viw Figure 4.23. The worst-case absolute partial deviation value (here associated with the symbol ϵ_1) is at a simulation time of 7 ms and occurs on segment $s3$ (see Figure 4.22). The second biggest absolute value occurs at time 107 ms but at segment $s4$ (the output of the system). As a second step, the structural traces at the identified points in time are defined:
for $t = 7\,ms$: $\tau_{structural}(\epsilon_1) =< -0.195, -0.974, -1.947, -0.292 >$
for $t = 107\,ms$: $\tau_{structural}(\epsilon_1) =< 0.195, 0.974, 0.195, 1.46 >$

The according derivation propagation sequences for used for the identification of critical blocks are:
for $t = 7\,ms$: $\partial\tau_{structural}(\epsilon_1) =< 5, 1.997, 0.148 >$
for $t = 7\,ms$: $\partial\tau_{structural}(\epsilon_1) =< 5, 0.2, 7.48 >$

As a result, for simulation time $t = 7\,ms$ the input amplifier is responsible for the worst-case deviation present at segment $s3$. For $t = 107\,ms$ the ouput of multipier **MULT2** has a maximum deviation associated with symbol ϵ_1 at segment $s4$. In a qualitativ perspective the second siuation is worse because this is the deviation impact is present at the system's output.

4.7 Sensitivity Analysis

As already described in Subsections 2.5.1 and 2.3.3 sensitivity (strictly speaking local sensitivity) are given as first partial derivations. In the AAF context this is expressed as $\partial\hat{x}/\partial\epsilon_i$. For AAFs the sensitivity calculation results in the corresponding partial deviation value. Due to the limitation to linear AAFs the sensitivity is constant within the $\epsilon = \pm 1$ range. This has the consequence that non-linear characteristics, have to be approximated. The evaluation of exact sensitivity at the central value is a significant property of approximation algorithms (see Section 3.2).

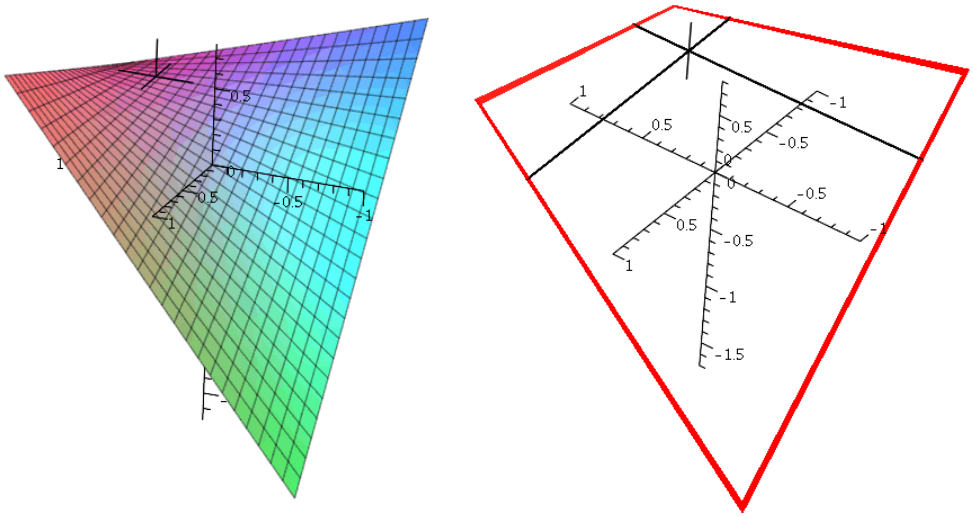


Figure 4.24: Sensitivity analysis of an uncorrelated AAF multiplication

Figure 4.24 illustrates the exact multiplication result of two uncorrelated AAFs. The axis defines ± 1 ranges of two deviation symbols and the colored surface illustrates the result. The right figure shows some characteristic sensitivity results which are potentially interesting for enhanced system analysis (e. g. bound sensitivities (red lines), 0 sensitivity marked by the cross, maximum sensitivity etc.) For some mathematical operations (e. g. AAF multiplication) these sensitivity values can be exactly calculated in advance (see Subsection 3.2.2). For further subsequent calculations, the approximated AAF is used. The architecture of the *SESYD* framework can be easily extended to store such characteristic values in **AAF** or **deviation** classes (see Section 3.1).

4.8 Frequency Domain Analysis

The transformation of AAF signals into a Fourier and Laplace domain is published and well studied by Florian Schupfer [Sch13, SKG+10]. The outcome of the theory behind Fast Fourier Transformation (FFT) for AAF signals is that FFT is a linear transformation and thus partial deviations which superimpose the center value in the time domain are also partial frequency deviations superimpose the Fourier transformed center value signal. Strictly speaking, in the spectrum the transformed center value signal is surrounded by partial frequency deviation subranges representing step-wise accumulated partial deviation bounds where ϵ symbols are ± 1 [Sch13].

I did not integrate the FFT calculation block into the *SESYD* framework yet. For frequency domain analysis the FFT post-processing function of the waveform viewer tool Synopsys Custom WaveView ADV [19] is used. The tool offers FFT calculation of scalar waveforms under various configuration parameters (e. g. windowing function, number of points, sampling rate, etc.). The AAF plotting functions of the *SESYD* framework generate individual waveforms for partially accumulated deviations (see any time domain AAF plots). According to the theory described above they can be used instantly for computing the frequency spectrum of the AAF signal. This allows comprehensive traceability on the first side from one signal spectrum to an other (structural tracing using spectra) and on the other side between time and frequency domain. Analysis refinement as described in Section 4.2 can be also applied using refined parameter settings for FFT post-processing. For enhanced frequency domain analysis also absolute deviation measures as introduced in Subsection 4.3.1 can be FFT post-processed. The resulting information is the frequency characteristic of dynamic partial deviation values. For example, this allows the detection of deviation value oscillations in the system.

Example:

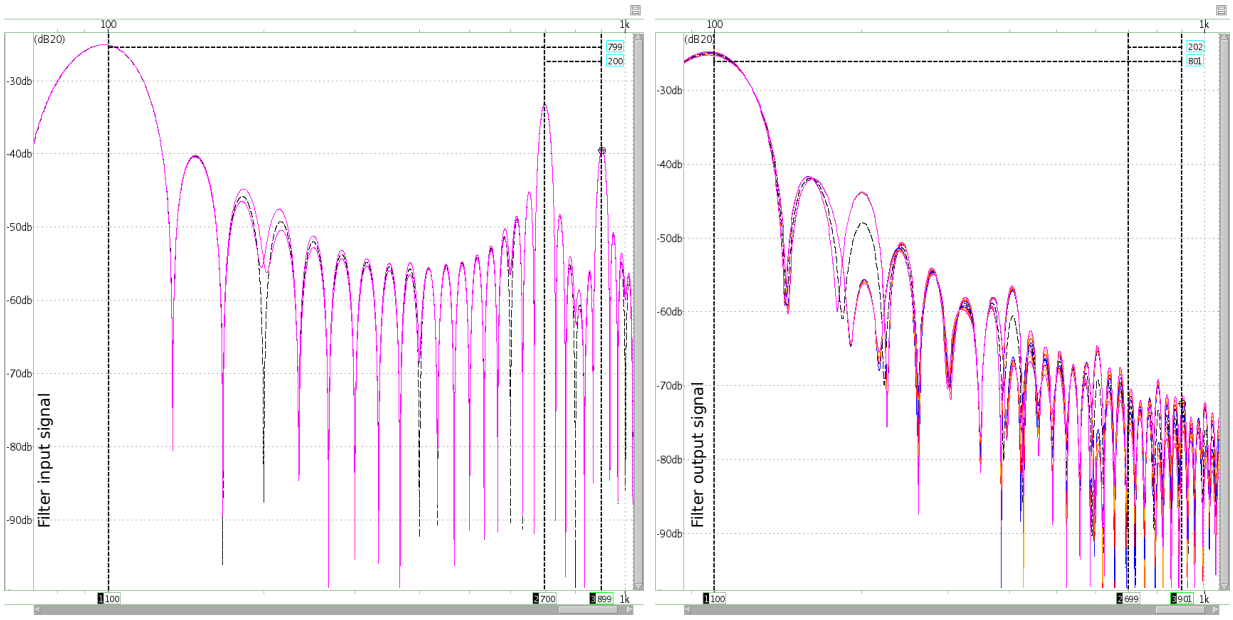


Figure 4.25: Frequency spectra of the input and filtered output signals of the example given in Section 3.5

For demonstration, I use AAF signals from the example described in Section 3.5. Figure 4.25 illustrate the frequency spectra of the filter's input (three added sinusoidal signals with frequencies

of $100, 700, 900\,Hz$) and its output. The spectrum of the input signal is given on the left side of Figure 4.25 and shows that the frequencies of the waveform generators 1, 2 and 3 are partially included in the accumulated signal indicated by spectral peaks at $100, 700, 900\,Hz$ respectively. According to the filter characteristic (see Figure 3.20) these peaks are significantly attenuated in the spectrum of the output AAF (see right side of Figure 4.25). Partial deviations (colored in blue, red, orange and magenta) result in just small deviations in the spectrum of the input signal. Frequency deviations of the output are increased.

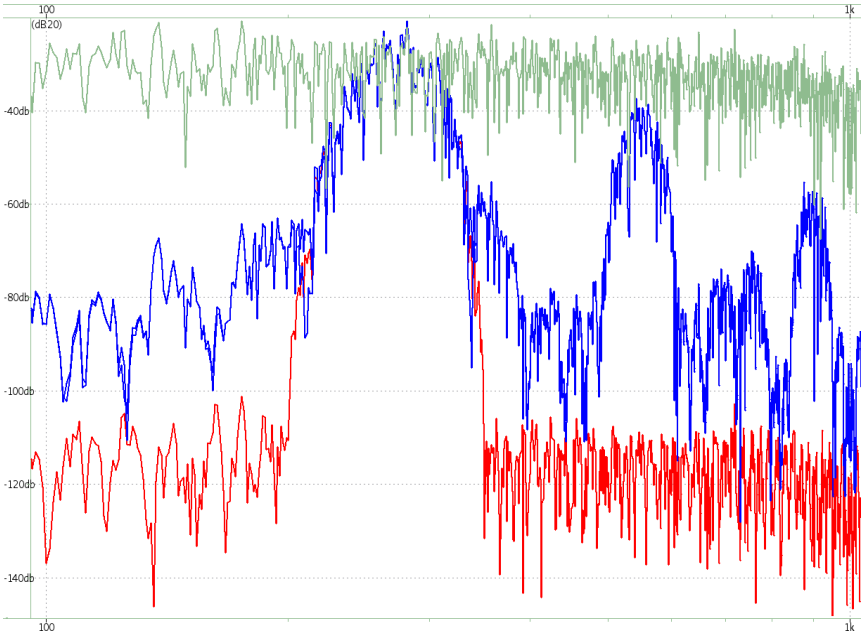


Figure 4.26: Spectrum of a filtered noise signal to generate colored noise.

As a second example, I use a band-pass filter for the generation of colored noise. The input of the filter is a random noise signal, and the spectrum of the filter output is illustrated in Figure 4.26. The filter has a pass frequency of $270\,Hz$, a bandwidth of $150\,Hz$, and a history length of 100 samples. In this case, the filter coefficients are deviated (each filter coefficient is an AAF), which has the result that the frequency spectrum variation is impacted by 100 symbols and associated partial deviation values. Figure 4.26 shows the central value in blue and accumulated (radius) upper and lower bounds in green and red respectively. In this demo, the deviation of filter coefficients is defined that large that for accumulated bounds of each ϵ symbol the band-pass filter characteristic completely disappears.

4.9 Formal System Analysis

Besides simulation-based system verification in the recent years formal, symbolic methods became more and more important caused by integration from research to an industrial context (e. g. Cadence Jasper Gold formal verification platform [13]). I already introduced the basics of formal verification in Section 2.6 and pointed to the specific purpose of probabilistic methods. Here I used basic symbolic model checking for verification of systems with parameter deviations. It has to be emphasized that for this thesis I used formal verification software *NuSMV* as an available tool not digging into details of algorithms and the math behind. The tool is open source, available for

commonly used OS, well documented, easy to use and its efficiency is proven by several scientific publications [3].

In principle for model checking two parts have to be defined according to Figure 4.27:

First, a system model represented by a transition system is defined using a specific NuSMV input language (strictly speaking the model defined in using the input language is automatically translated to an internal transition system for the algorithms). For example, the expressiveness of the NuSMV input language allows the definition of Finite state machines, sequential control flow instructions, etc. as well as constraining variables to subranges of numeric types. That's the point where range based analysis comes into play. Input variations of the model are represented by such subranges, where the initial value is not initially set (see Figure 4.27). In a model checking perspective this means that possible numeric valuations are translated to a set of possible initial states in the transition system. Such specified variations have the characteristic of tolerance intervals. This has the consequence that the association to a specific cause and the definition of subranges as in the AAF methodology is not possible.

Second, a set of properties is defined which are verified against the specified model. Property formulas may include propositional as well as temporal CTL* operators (CTL* is a superset of Computation Tree Logic (CTL) and Linear-Time Temporal Logic (LTL)). Thus, similar (or even equal) formulas used for Assertion-based analysis as presented in Section 4.4 can be used for model checking. Property formulas may particularly express safety and robustness characteristics derived from the system specification (see Figure 4.27). The significant difference (advantage) in contrast to ABV is that model checking requires no test bench, because it's a formal math-based analysis of the system and no simulation based execution [Kro10]. Unfortunately the effort for modelchecking increases sequentially at increasing system complexity. This has the concequence that for complex large systems the runtime of the checking process may become several hours. In [RS16] for formal system verification I propose a effort estimation in form of a cost function for verifcation panning.

As already mentioned the specified formulas are checked on the model, where the result is a (math-based) proof [Kro10, Bie09]. Under the view that variables can be specified by value ranges the property formulas are also checked for these ranges. If a selected property is violated NuSMV prints a representative counterexample for debugging.

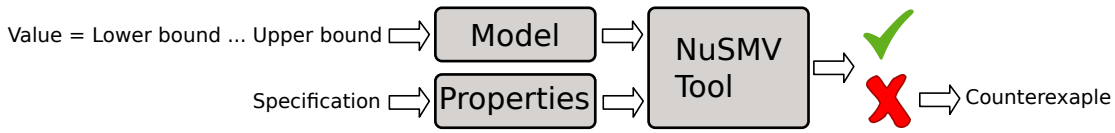


Figure 4.27: The model checking process using the NuSMV tool

4.10 Runtime Verification and Interface to other Simulators

Runtime verification is system analysis and verification during operation. The analysis methods proposed so far are applied during design time of a system. A side activity to developing the simulation based *SESYD* framework I implemented a case study where assertions are translated to particular hardware monitors. The methodology is similar to the Assertion-based verification approach used in [RSRG12a] where assertions are represented by functionally equal SystemC

AMS modules. The process for translating Property specification language (PSL) statements to a monitor structure is well understood [BZ10, SKW08, Eco06]. We translate AAF based definitions into corresponding intervals where the generated hardware monitors check the bounds. In an implementation perspective I used the Hardware description language VHDL and deployed the monitors in a Xilinx Zynq development board [21].

We implemented a mixed software-hardware application which is monitored by various assertion checkers. For evaluation, we stimulated the system's inputs in a way that the specified operational tolerances are violated, which has the consequence that selected assertion violations are reported by the corresponding monitor modules. This enables continuous diagnosis of assertions specified for ABV at design time during the operation of the system. Disadvantages of this approach are similar to drawbacks of Built-In Self-Test (BIST) methods in the context of online system testing (e. g. circuit area overhead, potential distorted timing behavior, enhanced development effort, etc.)

For using the *SESYD* framework within other software tools, I developed an Appropriate ***simulator Application Programming Interface***. As mentioned, the *SESYD* framework is programmed in C++, thus it can be integrated natively (by sourcecode or precompiled) into an C/C++ based application (as SystemC). As a case study I compiled the *SESYD* framework on a windows platform (using Visual Studio) as a Dynamic-Link Library (dll). This library is then integrated into the commercial simulation and prototyping framework LabView [30]. LabView is a block-oriented modeling environment implementing a Timed Data Flow (TDF) model of computation. However, we developed a set of blocks integrated into a toolbox, using the dll interface for calling selected functions of the*SESYD* framework (e. g. creating/adding deviations, symbol management, etc.) (see left side of Figure 4.28). The right side of Figure 4.28 shows a multiplication of two AAFs in LabView code (G programming language). A potential advantage of LabView is the ability of creating simulation prototypes by just dropping some blocks into a modeling pane, connect them, and having a nice looking visualization of results.

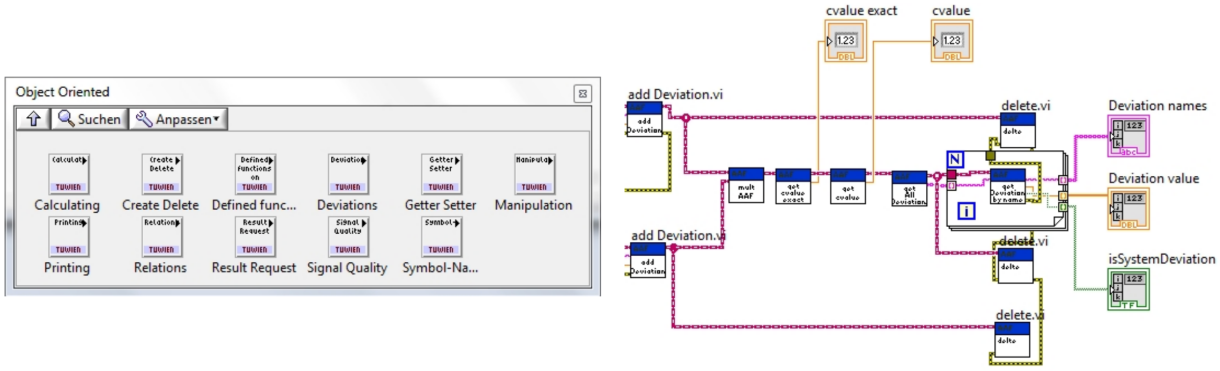


Figure 4.28: *SESYD* framework compiled as windows dll and used in the commercial simulation tool LabView.

5 Demonstration Examples and Results

In the presented examples, I focus to the main contribution of this thesis which is the design of an AA simulation framework including full deviation tracing features and associated enhanced analysis processes. I conclude each example with a comprehensive discussion of results derived from the implementations.

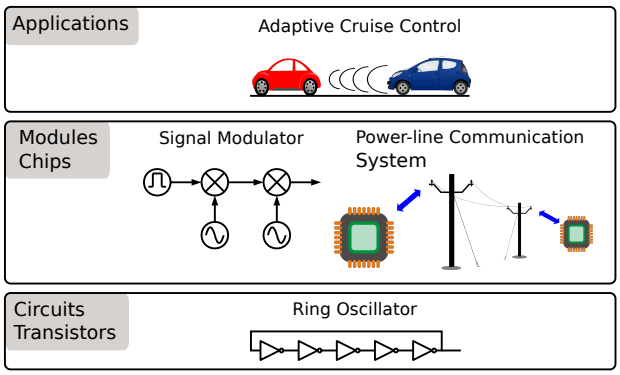


Figure 5.1: Overview of the implemented examples at various Level of Abstraction

The presnted system models are in the context of circuit and cyber-physical system design. According to Figure 5.1 the examples are located in different layers, representing the model's level of abstraction. The ring oscillator demo (Subsection 5.1) models transistor propagation delays impacting the frequency variation of a ring oscillator and is thus located at the lowest level of abstraction. The signal modulator (Subsection 5.2) and the Power-line communication transceiver (Subsection 5.3) examples are defined on a module level combining (abstractly) described behavioral blocks. The last example is at an application level layer modeling the behavior of an Adaptive Cruise Control (ACC) application (Subsection 5.4) implemented in modern cars.

5.1 Inverter Chain as a Ring Oscillator

This example demonstrates the influence of physical manufacturing inaccuracies in semiconductor production. Variations in the shaping of silicon and metal structures, as well as material characteristics, may influence the behavior of implemented transistors. Of course, variations are in a range that the information represented digitally is unaffected. Hence, variations in analog style as propagation delay, threshold voltage, input capacitances etc. are analyzed. They have

an analog cause and can be seen as parasitic MOS PVT deviations. Uncertaintainties in the proposed model are defined as W/L (transistors width to length ratio) mismatch, charge mobility variations, voltage variations etc.

The timing model

The core element of the example is a timing model for a single CMOS inverter as illustrated in Figure 5.2. The figure shows the internal symmetric structure of a PMOS (Q1) and a NMOS (Q2) transistor which logically inverts the Boolean input signals. Hence, for timing evaluation switching delays of the transistors are analyzed in detail.

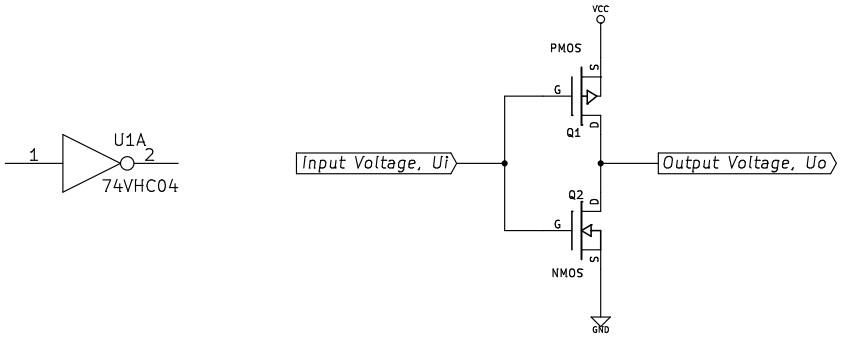


Figure 5.2: CMOS inverter element and the according symmetric inner structure composed by a NMOS and PMOS transistor.

The timing model of the oscillator is evaluated by analysis of switching characteristics of according MOS devices. The proposed model for planar technology transistors is reduced to a simplified behavior omitting physical short channel effects occurring at structural sizes below 100 nm [MMMAY01], resistances of wires and silicon conductivity inside the MOS device.

The propagation delay, which is defined as the time delay until the output has a voltage of $Vcc/2$, is mainly affected by capacitances. At a switch-on event at $t = 0$ as illustrated in Figure 5.4 the logical level changes from **LOW** to **HIGH** which equals a 0 V to Vcc voltage change. Transistor $Q1$ is switched off immediately and for the evaluation of timing values the circuit is reduced to a structure given in Figure 5.3 [FS]. For $t \leq 0$ the load-capacitance is charged to the voltage level Vcc equally to the output voltage Uo. The propagation delay for the switch-on event is defined as the time the output voltage drops from Vcc to $Vcc/2$ (see Figure 5.4).

The output voltage of the inverter follows the output characteristic of the NMOS transistor illustrated in Figure 5.6. As already mentioned, before the switch-on event the output voltage is Vcc and moving from point A to C in according to the characteristic defined by the transistor (see Figure 5.6).

First, I describe the trace between point A and B where the NMOS transistor is in saturation mode. This time interval is defined as a partial delay time named t_{dON1}. Saturation mode for the MOSFET is in general given if the condition $U_{DS} > (Vcc - U_{thN}) \wedge U_{DS} \leq Vcc$ is fulfilled [TS86]. The according drain current i_{D2} for this region is defined as $i_{D2_{sat}} = k_N \cdot (U_{GS2} - U_{thN})^2$, where $U_{GS2} = Vcc$. Relation between current and voltage at the load capacitance C is defined by the differential equation $Q = C \cdot u_{DS2}$ with $Q = -\int i_{D2_{sat}} \mathrm{d}t_{dON1} \Rightarrow t_{dON1} = -C \int \frac{1}{i_{D2_{sat}}} \mathrm{d}u_{DS2}$. u_{DS2} is eqaual with the output voltage u_o of the circuit. Limits of the integral for the evaluation

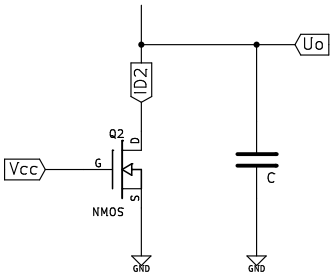


Figure 5.3: Reduced circuit for the turn-on event of the described inverter circuit

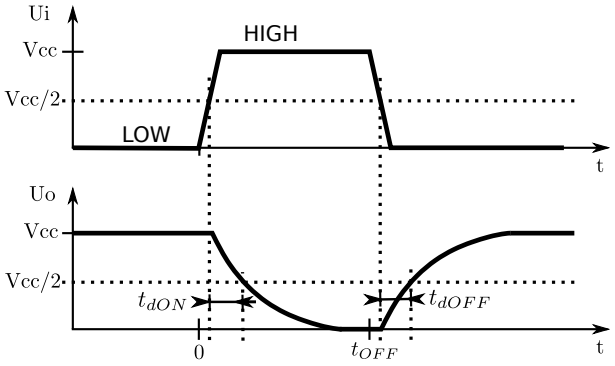


Figure 5.4: Summarized timing trace for a switch-on and switch-off event of a single inverter.

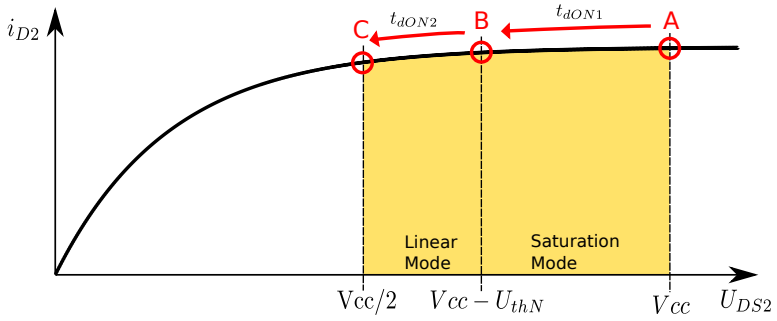


Figure 5.5: The output characteristic of the NMOS transistor. The output voltage of the circuit is moving from point A to C, passing saturation mode and linear mode of the transistor.

of partial time interval t_{dON1} are given by points A and B in Figure 5.6. This results in a partial time interval t_{dON1} where the NMOS transistor $Q2$ is in saturation mode:

$$t_{dON1} = -C \int_{Vcc}^{Vcc-U_{th}} \frac{1}{k_N \cdot (Vcc - U_{thN})^2} \mathrm{d}u_o = \frac{C \cdot U_{thN}}{k_N \cdot (-Vcc + U_{thN})^2} \tag{5.1}$$

Second, the timing interval the output voltage continues dropping down to $Vcc/2$ where the transistor is in linear (triode) mode. According to Figure 5.6 this time interval t_{dON2} is located between marked points B and C. Linear mode of the NMOS transistor is generally defined by the U_{DS} voltage constraints $U_{DS} > 0 \wedge U_{DS} \leq (Vcc - U_{thN})$ [TS86]. The according drain current $i_{D2_{lin}}$ is not longer independent of u_{DS2} and defined as $i_{D2_{lin}} = 2k_N \cdot \left((U_{GS2} - U_{thN}) \cdot u_{DS2} - \frac{1}{2}{u_{DS2}}^2\right)$. The corresponding differential euation for t_{dON2} is given as $t_{dON2} = -C \int \frac{1}{i_{D2_{lin}}} \mathrm{d}u_{DS2}$.

As illustrated in the reduced circuit Figure 5.3, the drain current $i_{D2_{lin}}$, U_{GS2} is substituted by the input voltage level Vcc for $t > 0$, and u_{ds2} is equal to the output voltage u_o. Limits of the integral are given by points B and C in Figure 5.6. The partial timing value t_{dON2} for the NMOS transistor $Q2$ being in linear mode is

$$t_{dON2} = -C \int_{(Vcc-U_{thN})}^{Vcc/2} \frac{1}{2k_N \cdot \left((Vcc - U_{thN}) \cdot u_o - \frac{1}{2}{u_o}^2\right)} \mathrm{d}u_0 = \frac{1}{2} \frac{C \ln\left(\frac{Vcc}{3Vcc - 4U_{thN}}\right)}{k_N \left(-Vcc + U_{thN}\right)} \tag{5.2}$$

For the total propagation delay specified by the switch-on sequence partial delays t_{dON1} and

t_{dON2} are added to the following equation.

$$t_{dON} = t_{dON1} + t_{dON2} = \frac{C \cdot U_{thN}}{k_N \cdot (-Vcc + U_{thN})^2} + \frac{1}{2}\frac{C \ln\left(\frac{Vcc}{3Vcc-4U_{thN}}\right)}{k_N\,(-Vcc + U_{thN})} \tag{5.3}$$

The switch-off event is defined as a logical transition from **HIGH** to **LOW** at the input of the gate. Considering the circuit at transistor level, this is equal to a transition of the output voltage from GND to Vcc (see Figure 5.4). Due to the symmetric structure of the inverter circuit as shown in Figure 5.2 the switch-off event can be analyzed similarly to the switch-on event discussed in the last paragraphs. Differences to the switch-on event are given by the responsibility of $Q1$ for the charge of the load capacitance C. Under consideration of reversed voltage and current counting directions and the negative output characteristic of the PMOS device [TS86] the switch-off time is defined as:

$$t_{dOFF} = \frac{C \cdot |U_{thP}|}{k_P \cdot (-Vcc + U_{thP})^2} + \frac{1}{2}\frac{C \ln\left(\frac{Vcc}{3Vcc-4|U_{thP}|}\right)}{k_P\,(-Vcc + |U_{thP}|)} \tag{5.4}$$

Constants k_N and k_P in equations 5.3 and 5.4 are defined by material as well as structural characteristics of the according P- and NMOS transistor. For MOS transistors k is defined as $k = \frac{\mu C_{Ox}'}{2} \cdot \frac{W}{L}$ where μ defines the charge mobility in the semiconductor material, $C_{Ox}{}'$ the site-related capacitance of the oxide material and W/L the silicon structure gate width to length (called channel aspect ratio). For the according transistor type and under consideration of the site-related capacitances k_N and k_P are given as $k_N = \frac{1}{2} \cdot \frac{\epsilon_{OxN}\mu_N W_N}{T_{OxN} L_N}$ and $k_P = \frac{1}{2} \cdot \frac{\epsilon_{OxP}\mu_P W_P}{T_{OxP} L_P}$ where ϵ_{Ox} are the permitivities of the P and N doped silicon material and T_{Ox} are oxide thickness values of the P- and NMOS transistors [TS86].

As illustrated in equations 5.3 and 5.4 the load capacity C has a significant impact to t_{dON} and t_{dOFF}. For this example the capacity is the sum of the following parts:
Gate input capacitance to the next gate - For this example inverters are structured in a chain, where the output of a single CMOS inverter is driving the next one.

$$C_{inNext} = C_{OxP}{}' W_P L_P + C_{OxN}{}' W_N L_N \tag{5.5}$$

$$C_{OxP}{}' = \frac{\epsilon_{OxP}}{T_{OxP}} \tag{5.6}$$

$$C_{OxN}{}' = \frac{\epsilon_{OxN}}{T_{OxN}} \tag{5.7}$$

Parasitic Depletion Capacitance - Is defined by the bottom and sidewall capacity:

$$C_{DB} = C_{Bottom} + C_{SW} \tag{5.8}$$

$$C_{Bottom} = C_{JN}{}' W_N L_{diffN} + C_{JP}{}' W_P L_{diffP} \tag{5.9}$$

$$C_{SW} = C_{JSWP}{}'(W_P + 2L_{diffP}) + C_{JSWN}{}'(W_N + 2L_{diffN}) \tag{5.10}$$

$C_{JN}{}'$ and $C_{JP}{}'$ are defined as site-rated zero-bias drain-bulk junction capacitance. L_{diffN} and L_{diffP} are diffusion length values for the p and n doped region. $C_{JSW}{}'$ is a length-rated capacitance value at the sidewalls of the inverter.
Parasitic Wire-Capacitance - Wires are metal lines connecting the output of an inverter with the next stage. Due to low depletion influence the wire capacitance C_{wire} is approximately the capacitance formed by the oxide [Mit98].

$$C_{wire} = \frac{\epsilon_{WOx}(W_m L_m)}{T_{WOx}} \tag{5.11}$$

where ϵ_{WOx} is the permitivity of the thick wire oxide, W_m and L_m are the structural dimensions of the wire and T_{WOx} defines the thickness of the wire oxide.

The total load capacitance C for this model is defined by the sum of the described partial capacitance components.

$$C = C_{inNext} + C_{DB} + C_{wire} \tag{5.12}$$

The result is a propagation delay model for an inverter including semiconductor level effects of MOS transistors. Some parameters within the presented equations are evaluated form the BSIM 3 v.2 (Berkeley Short-channel Insulated Gate FET Model) model equations. Manufacturing parameters of the MOS model are set according to the C5 semiconductor process. Values can be evaluated form process datasheets and several published measurements [33]. The C5 process is optimized for 5 V mixed signal applications, but can be adapted for low-voltage and low-power purpose. Structural sizes features a minimum value of 0.5 μm. This enables a relatively cheap production procedure without any extra costs associated with extra mask production for Bipolar-CMOS-DMOS (BCD) processes [33]. However, for this example, I used the set of parameters presented in the following table.

Table 5.1: Parameter set for the presented propagation delay model

Parameter Symbol	Value	Description
V_{CC}	5 V	Inverter supply voltage
U_{thP}	0.92 V	PMOS threshold voltage
U_{thN}	0.65 V	NMOS threshold voltage
$\epsilon_{OxP/N}$	$3.45 \cdot 10^{-11}\, Fm^{-1}$	P/NMOS gate oxide permitivity
μ_P	$135\, cm^2V^{-1}s^{-1}$	PMOS charge mobility
μ_N	$550\, cm^2V^{-1}s^{-1}$	NMOS charge mobility
$T_{OxP/N}$	$10\, nm$	P/N Gate oxide thickness
$C_{JP/N}{}'$	$0.2 \cdot 10^{-3}\, F/m^2$	P/NMOS diffusion capacity
L_{diffP}	$3.5 \cdot 10^{-8}\, m$	PMOS diffusion length
L_{diffN}	$5 \cdot 10^{-8}\, m$	NMOS diffusion length
$C_{JSW}{}'$	$0.5\, nF/m$	P/NMOS sidewall capacity
ϵ_{WOx}	$3.45 \cdot 10^{-11}\, Fm^{-1}$	wire oxide permitivity
L_m	$0.5\, mm$	Metal wire length
W_m	$10\, \mu m$	Metal wire width
T_{WOx}	$100\, nm$	Metal wire oxide thickness

Transistor width and length uncertainty analysis

For analysis purposes, I extended this exact numerical model by considering parameter uncertainties in the transistor's dimensions. Thus, the parameters $L_{P/N}$ and $W_{P/N}$ are represented by the following Affine Arithmetic forms. According to [Mit98] in practice, the PMOS transistor's

(W/L) is a factor of two larger to compensate the lower hole mobility in the channel.

$$W_N = 10 \cdot 10^{-6} + 2 \cdot 10^{-6}\epsilon_1 + 1 \cdot 10^{-6}\epsilon_2 \tag{5.13}$$

$$W_P = 20 \cdot 10^{-6} + 2 \cdot 10^{-6}\epsilon_1 + 1 \cdot 10^{-6}\epsilon_2 \tag{5.14}$$

$$L_{N/P} = 0.6 \cdot 10^{-6} + 0.05 \cdot 10^{-6}\epsilon_1 + 0.06 \cdot 10^{-6}\epsilon_3 \tag{5.15}$$

In principle, the transistor gates have an exact dimension of $10 \times 0.6\,\mu m$. For modeling process variabilities three ϵ symbols are used. ϵ_2 and ϵ_3 models the production accuracy of the planar process in W (e. g. x coordinate) and L (e. g. y coordinate) direction respectively. With ϵ_1 I model common accuracy variability of the process in both directions. Besides the defined deviations model a transistor pair mismatch of the P- and NMOS included in a single inverter stage. In essence, this example shows that considering variability effects associated with physical process variations by AAFs is reduced to the replacement of numerical W and L parameters by Affine Arithmetic forms. The effort for introducing uncertainty is low in contrast to the evaluation of the model itself.

The analysis objective, (see Section 4.2) is to evaluate the bonds of the propagation delay to derive frequency bounds of the subsequently developed ring oscillator and to assess sensitivities and impact of the modeled variability effects.

Results and discussion

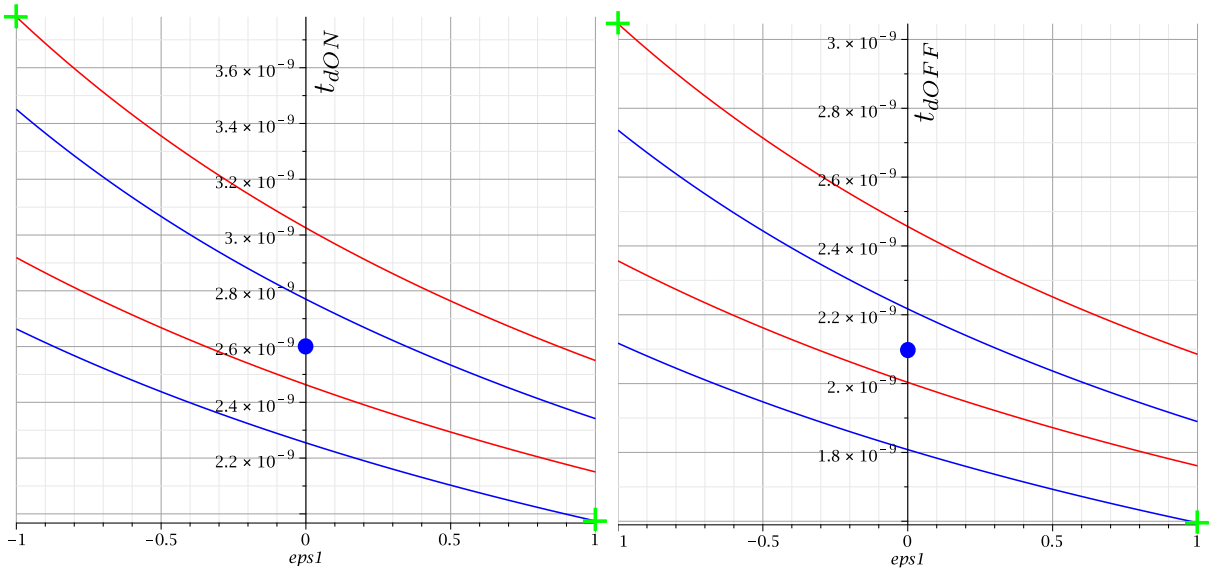


Figure 5.6: Transistor propagation delay under uncertain structural transistor dimensions.

Figure 5.6 shows the switch-on (left side) and switch-off (right side) propagation delay time (on the y-axis). These delays occur at a **LOW** to **HIGH** and vice versa inverter input event. The time values are plotted under variation of the correlated symbol ϵ_1 and the worst-case valuations of ϵ_2 and ϵ_3 (± 1). The exact solution which represents the result of numeric exact computation is for $t_{dON} = 2.6\,ns$ and for $t_{dOFF} = 2.1\,ns$. This corresponds to $\epsilon_{1,2,3} = 0$ and is highlighted in blue in both diagrams. The resulting bounds of the propagation switching delays are for t_{dON}: $1.97\,ns$ to $3.78\,ns$ and for t_{dOFF}: $1.59\,ns$ to $3.05\,ns$. These bounds are highlighted by green crosses.

The minimum propagation delays are caused by worst-case symbol valuations of $\epsilon_1 = 1$, $\epsilon_2 = 1$, $\epsilon_3 = -1$ and the maximum delay by $\epsilon_1 = -1$, $\epsilon_2 = -1$, $\epsilon_3 = 1$ respectively. The lines in the diagram indicate the worst-case traces: red line for $\epsilon_2 = 1$ and $\epsilon_3 = \pm 1$, and blue line for $\epsilon_2 = -1$ and $\epsilon_3 = \pm 1$.

Next, sensitivity analysis is applied to evaluate the impact of single uncertainties to the switch-on/off delay times. Sensitivities are computed by the first derivation under respect of deviation symbols while other deviation symbols are zero. This results in the corresponding sensitivities at the exact central point. The following table holds the snesitivities of t_{dON} and t_{dOFF} at variations of uncertainties represented by $\epsilon_{1,2,3,}$.

Table 5.2: Sensitivity of the propagation delay against variation of uncertainties associated with symbols $\epsilon_{1,2,3,}$

	Sensitivity of t_{dON}	Sensitivity of t_{dOFF}
ϵ_1	$-4.38 \cdot 10^{-10}$	$-3.37 \cdot 10^{-10}$
ϵ_2	$-2.67 \cdot 10^{-10}$	$-2.13 \cdot 10^{-10}$
ϵ_3	$1.15 \cdot 10^{-10}$	$1.08 \cdot 10^{-10}$

The common relations between the transistor's channel dimensions and its speed, reflected by the sensitivity values, are: $W \uparrow \Rightarrow t_d \downarrow$ and $L \downarrow \Rightarrow t_d \downarrow$. In the following enumeration I discuss qualitative statements derived from sensitivity analysis results:

1. For both delay times the uncertainty associated with ϵ_1, which is the correlated one for W and L, has the dominating. If ϵ_1 is positive W increases more, relative to L ($2 : 10/20$ versus $0.05 : 0.6$). Thus the W increase is dominating which results in a dominating negative derivation of propagation delay times.

2. The symbol ϵ_2 is included in W only. If ϵ_2 is positive (W gets larger due to the modeled variation) the transistor switching delay decreases, which correlates to a negative sign of the corresponding sensitivity.

3. The symbol ϵ_3 is included in L only. A positive ϵ_3 value represents an increased channel length (L indicates the physical distance between the source and drain) caused by a variability. This slows down the transistor which correlates with a positive sensitivity value of the propagation delays.

Figure 5.7 illustrates the delay time variation under accumulated variations (radius) of W and L.

Ring oscillator application

A ring oscillator contains an odd number of the described inverter models in a chain. The output of the last inverter is fed back to the input of the first one within the chain. Thus the output, which may be an arbitrary connection signal in the chain, switches continuously between a **HIGH** and **LOW** level. The frequency depends on the propagation delays of the included inverters (t_{dON} and t_{dOFF}) and the number of inverters within the chain (N). If I consider

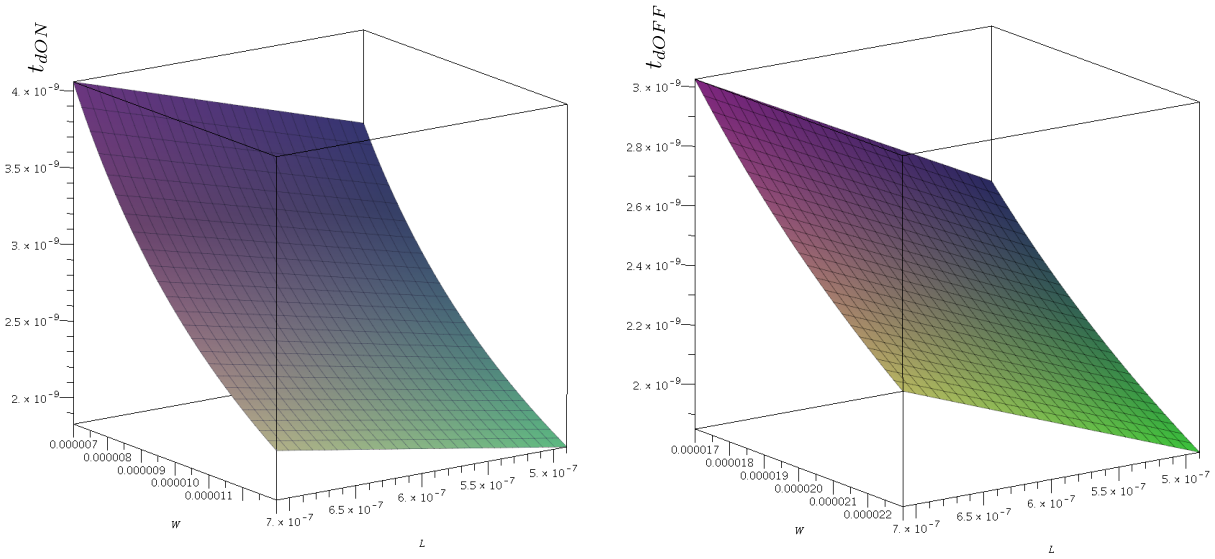


Figure 5.7: Accumulated impact of W and L variatins to the propagation delay times for a switch-on and switch-off event.

ranges for the propagation delay $t_{dON,Min/Max}$ and $t_{dOFF,Min/Max}$ the corresponding range of the oscillators output frequency can be computed by:

$$f_{Max} = 1/T_{Min} = 1/(N \cdot t_{dON,Min} + N \cdot t_{dOFF,Min})$$
$$f_{Min} = 1/T_{Max} = 1/(N \cdot t_{dON,Max} + N \cdot t_{dOFF,Max})$$

The output frequency range using the evaluated propagation delay values of the inverter model is given in Table 5.3. As a result Table 5.3 shows that the variations in the output frequency are highly sensitive to propagation delay variation. Thus, in an application perspective, such a ring oscillator do not have a proper frequency reproducibility. In other words, the output frequency may vary (in the worst case) by a factor of approximately two from one device to another.

Table 5.3: Frequency output range of a N-stage ring oscillator using the deviated inverter model above.

N	T_{Min}	T_{Max}	f_{Min}	f_{Max}
5	17.80 ns	34.15 ns	29.28 MHz	56.18 MHz
19	67.64 ns	129.77 ns	7.71 MHz	14.78 MHz
29	103.24 ns	198.07 ns	5.05 MHz	9.69 MHz
49	174.44 ns	334.67 ns	2.99 MHz	5.73 MHz
99	352.44 ns	676.17 ns	1.48 MHz	2.84 MHz
199	708.44 ns	1.36 μs	735.74 kHz	1.41 MHz
299	1.06 μs	2.04 μs	489.68 kHz	939.46 kHz
399	1.42 μs	2.73 μs	366.95 kHz	704.01 kHz
...				

In essence, this example shows, how an exact numerical model on silicon level is extended for range analysis using Affine Arithmetic. Results, in this case, a high propagation delay impact of

the correlated process uncertainty for W and L, may may enhance the insight, and motivate for subsequent range analysis procedures to maximize the performance of a design.

5.2 Amplitude-Shift Keying (ASK) Modulator

The example described in this section is a two-stage Amplitude-Shift Keying (ASK) modulator. The model is illustrated in Figure 5.8. A digital input data stream described as a rectangular function is multiplied by two sinusoidal carrier signals. Such ASK modulator systems are frequently used for digital communication purposes. A digitally encoded information is modulated on a high-frequency carrier signal (in this example in two stages) for (e. g. wireless transmission).

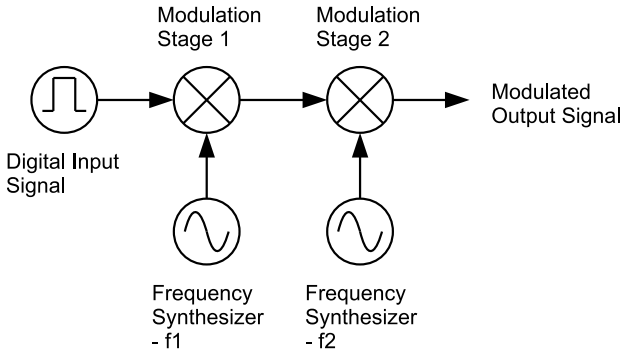


Figure 5.8: Structure of the modeled system. Digital input information is multiplied twice by high-frequency carrier signals. The modulated output signal can be directly used for wireless transmission.

Input signal model

The input signal is a digital data stream illustrated in Figure 5.9. Low and high level of the Boolean input levels will be distorted and superimposed with a deviation signal in the physical domain. These effects are caused by the transmission of the signal, Electromagnetc Compability (EMC) effects, signals influencing each other (crosstalk), etc. In the AAF domain, this is modeled using the deviation symbol ϵ_1. The mathematical representation of the input signal is:

$$\hat{x}(t) = \begin{cases} Hi + a_1\epsilon_1 & \text{if } 1ms \leq t \leq 2ms \\ & \text{and } 4ms \leq t \leq 6ms \\ Lo + a_2\epsilon_1 & \text{otherwise} \end{cases} \tag{5.16}$$

Where Hi is the logical high level of $3.3V$, Lo is the logical low level of $0.2V$, a_1 is the partial deviation value of $0.02V$ at low level and a_2 is the partial deviation value of $0.04V$ at the high level.

Frequency synthesizer model

The two included frequency synthesizers are submodels of the system illustrated in Figure 5.8. In digital electronics design, predefined analog signals are often generated with the help of a lookup

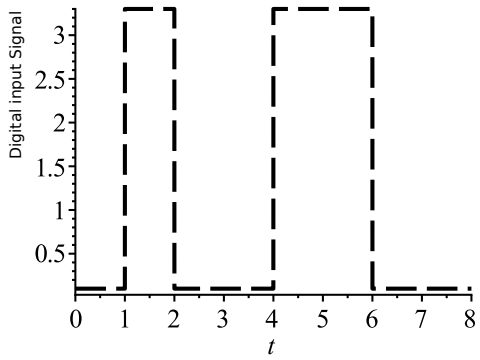


Figure 5.9: Digital input data stream. A rectangular input waveform with two steady levels representing the logical levels of the binary input information.

table. The calculated values (in this case one quarter of a full sine period) are loaded from the table and transferred to an analog output in equidistant timesteps. Due to the limited bit width of the sine values and the continuous reload of sampling values this results in value and time discrete representation of the sine signal (dashed line in Figure 5.10). In this case, the stepwise signal trace describes the physical domain. Through time discretization the sinusoidal signal $s(t)$ is defined as:

$$s(t) = sin(\omega t) \cdot \sum_{n=-\infty}^{\infty} \delta(t - nT) \tag{5.17}$$

Where ω is the angular frequency of the signal, δ is the Dirac impulse and T is the timing period for reloading new samples from the lookup table.

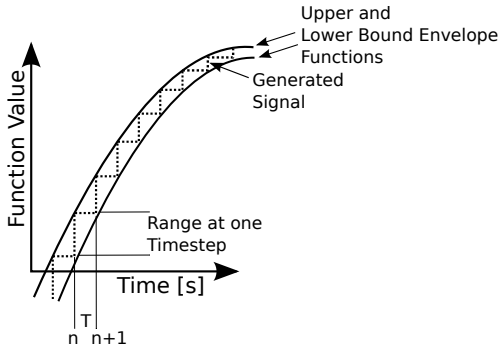


Figure 5.10: The dashed line illustrates the synthesized step wise sine signal. Upper and lower bound envelope functions describes a full inclusion area of the physically generated signal.

A potential model for the defined synthesized signal is to describe the waveform with the help of IA. As shown in Figure 5.10 for each point in time, an upper and lower functional limit is defined. This results in an area enveloping the step wise trace. At the points in time when a new value of the lookup table is loaded the interval is defined as

$$S = [sin(\omega \cdot (n-1)T), sin(\omega \cdot nT)] \tag{5.18}$$

Where ω is the angular frequency of the signal, n is a counting variable indicating the discrete timestep and T is the sample reload period.

In an AAF perspective, the rectangular signal is modeled as an ideal sinusoidal function superimposed with a partial deviation which enclosures the step wise characteristic. In implementations, the discrete generated signal is filtered. This filter between synthesizer and modulation stage transfers the hard-edged rectangular signal into a smooth sinusoidal waveform. However, bound values reflecting partial deviation values are strictly defined at the reload points. In between the signal may float but not exceeding the modeled bounds. The central value is set to the half of the step size, and a deviation is located symmetrically to describe the mentioned bounds (upper and lower sine waveform shown in Figure 5.10).

$$\hat{s}[n] = \frac{sin(\omega(n-1)T) + sin(\omega nT)}{2} + (sin(\omega nT) - sin(\omega(n-1)T)) \cdot \epsilon_i \tag{5.19}$$

Where ω is the angular frequency of the signal, n is a counting variable indicating the discrete timestep, T is the value reload period and ϵ_1 is the unpredictable deviation symbol with a value between -1 and 1.

For this example, which computes values using continuous time, the following AAF signals representing the outputs of the frequency synthesizers, $f_1(t)$ and $f_2(t)$ (see Figure 5.8) are used:

$$\hat{f}_1(t) = A_1 \cdot sin(2\pi f_1 t) + b_1 \cdot cos(2\pi f_1 t)\epsilon_2$$
$$\hat{f}_2(t) = [A_2 \cdot sin(2\pi f_2 t) + b_2 \cdot cos(2\pi f_2 t)\epsilon_2\ ] + b_3\epsilon_3$$

The synthesizer generating $f2$ has an additional constant deviation. A_1 is the first carrier signal amplitude of $3V$, f_1 is the first carrier frequency of $10kHz$, b_1 is the synthesizer-f1 partial deviation value of $0.2V$, A_2 is the second carrier amplitude of $5V$, f_2 is the second carrier frequency of $20kHz$, b_2 is the synthesizer-f2 partial deviation value of $0.1V$ and b_3 is a constant deviation of $0.08V$. ϵ_2 models a deviation which is caused by a step wise generation of the sinusoidal waveforms. The type of signal generation is the same for each generator, so this deviation symbol is present in both modulation signals and reflects a correlation.

The output signal of the system is given by the multiplication of the digital input data sign by the sinusoidal carriers:

$$\hat{y}(t) = \hat{x}(t) \cdot \hat{f}_1(t) \cdot \hat{f}_2(t)$$

The given computation contains an uncorrelated (modulation stage 1) and a correlated (modulation stage 2) multiplication. In this example, for the non-linear multiplication operations, various approximations of the output signal according to Section 3.2 are calculated.

Analysis

The central value (equal to exact simulation without considering any deviations) of the output signal (solid line) and the input signal (dashed line) are shown in Figure 5.11-a. For Analysis of the output signal, main interest is in worst-case signal values at a logical **HIGH** and **LOW** input signal. Thus, the analysis objective is to evaluate the maximum of the output radius in time, and the value of this maximum including all subranges. The point in time for the maximum of the radius can be evaluated using a temporal trace as described in Section 4.5. The maximum deviation of the synthesizer signals is located at their amplitude zero cross (sine central value and cosine partial deviation value). Figure 5.11-b is a zoomed view of the modulated undeviated output signal at a logical level transition ($t = 1\,ms$). As shown in this figure maximum output deviations are calculated at simulation times t_{s1} for a **LOW** input and at t_{s2} for a **HIGH** input.

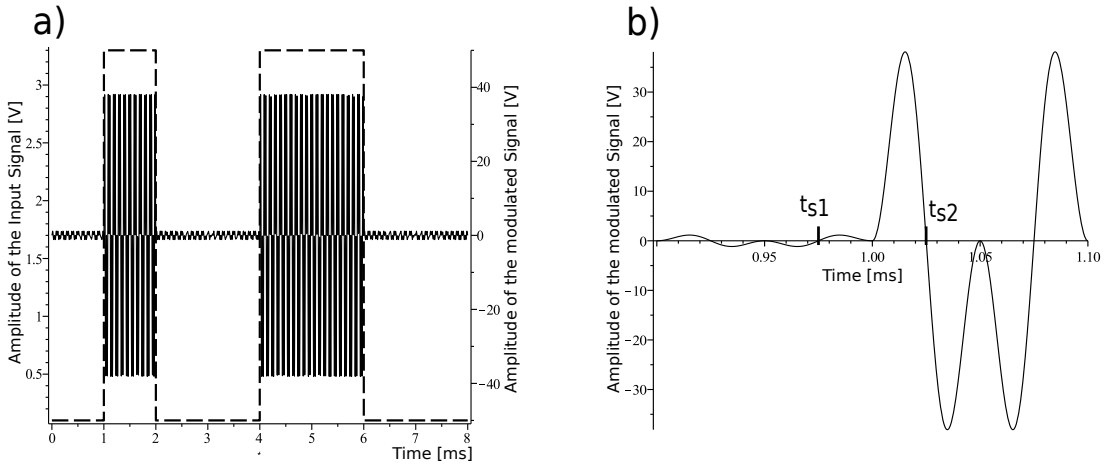


Figure 5.11: a) Rectangular digital input signal (dashed curve) and modulated output signal without deviation calculation. b) Zoomed undeviated output signal at a logical low to high transition at 1 *ms*. Maximum output deviations are calculated at t_{s1} and t_{s2}.

Results and discussion

The essence of this example is to compare the impact of different approximation schemes for the included multiplication. Table 5.4 presents all important output signal parameters of both mixing stages for a **HIGH** and **LOW** logical input state. The first column holds the type of approximation where lin. stands for linearized, Int. Ex. for interval exact and Ch. for Chebyshev approximation (see Section 3.2.1). The second column contains the modified central value β for each approximation. The mathematically exact central value can be found at each linearized approximation. All other methods (marked by a *) have a shifted central value, but due to the system structure according to Figure 3.11 the exact center value will be reminded at any operation. The next two columns hold approximation values for system added deviation symbols due to the non-linear multiplications. ϵ_4 is generated at the first modulation stage and ϵ_5 at the second one (see Figure 3.11). Partial user type deviations a_1, a_2 and a_3 corresponding to the deviation symbols ϵ_1, ϵ_2 and ϵ_3 are stated in the header of each table part. The last two columns of Table 5.4 holds the over-approximation of each calculated method (LB stands for lower bound and UB for upper bound). Over-approximation is defined as the difference between exact interval bounds, which are also stated in each table header and the bounds resulting out of the approximation form.

The linearized (first-order Taylor) approximation contains the exact central value, correct partial deviations and needs no system approximations. The result can be seen as a linearization at an operating point (in this case the central value) as known from the calculation methodology of standard semiconductor circuits. The over-approximation is significant at one interval bound. At the other bound there is an under-approximation. That is why linearization approximation is only applicable for simulations where the correct deviation sensitivities at the central value are required, independent of over- or under-approximation for simulation points unequal to the central value. As it can be seen in Table 5.4 over- and under-approximation values are equal at uncorrelated multiplications and unequal in correlated ones. This has its reason in the existence of quadratic ϵ terms in the symbolic result of a multiplication of correlated forms (see Figure 2.9-a).

The interval exact approximation has the property that interval bounds have no over-approximation. As shown in the first two sub-tables of Table 5.4 for uncorrelated multiplications no

Table 5.4: Resulting affine forms at the modulated output of mixing stage 1 and 2, at a logical high and low input level

Approx. Method	β	ϵ_4 α	ϵ_5 α	Oapprox. LB	Oapprox. UB
Stage 1 - Low Signal, Simulation at t_{s1} Partial Deviations: $a_1 = 0.042427$, $a_2 = 0.084853$ Exact Output Interval=$[0.305472, 0.560031]$					
Lin.	0.424266	0	0	0.008485	-0.008485
Int. Ex.*	0.432751	0	0	0	0
Ch.*	0.428509	0.004243	0	0.008485	0
Stage 1 - High Signal, Simulation at t_{s2} Partial Deviations: $a_1 = 0.084853$, $a_2 = 1.400078$ Exact Output Interval=$[5.532429, 8.502291]$					
Lin.	7.000389	0	0	0.016971	-0.016971
Int. Ex.*	7.017360	0	0	0	0
Ch.*	7.008874	0.008485	0	0.016971	0
Stage 2 - Low Signal, Simulation at t_{s1} Partial Deviations: $a_1 = 0.212133$, $a_2 = 0.636399$, $a_3 = 0.033941$ Exact Output Interval=$[1.350184, 3.124974]$					
Lin.	2.121330	0	0	0.111327	-0.121170
Int. Ex.*	2.237579	0	0.004922	0	0
Ch.*	2.171309	0.021213	0.049979	0.132541	0
Stage 2 - High Signal, Simulation at t_{s2} Partial Deviations: $a_1 = 0.424266$, $a_2 = 10.500584$, $a_3 = 0.560031$ Exact Output Interval=$[24.453335, 47.442782]$					
Lin.	35.00195	0	0	0.936270	-0.955956
Int. Ex.*	35.94806	0	0.009843	0	0
Ch.*	35.45869	0.042427	0.456788	0.978742	0

additional system deviation α is needed. Only if there are quadratic or higher order terms in the mathematically exact result further deviations for the correction of the radius in a way to reach exact interval bounds are required. In comparison to all other approximation methods mentioned in this thesis, the shift of the central value is the largest one.

At the Chebyshev approximation in each multiplication stage a system deviation α is added. For the full inclusion of all results independent of all user deviation signal values an extension of the radius with the approximation value α is always required [Gra09]. At repeated multiplications (as applied in this example) it is required to multiply all deviations (system and user type) by the exact calculated central value, not with the β value of the corresponding approximation. If not done so linear factors of the result representing system's deviation sensitivities will become incorrect. The methodology is to approximate only non-linear terms all other affine representable parts must be calculated exactly. Over-approximations stated for the Chebyshev method in Table 5.4 are only present at one of the interval bounds. This property follows the theoretical description shown in Figure 2.9.

The result of the worst case analysis is that at a logical low level the input signal range of $0.04V$ is

transformed to a worst case modulated output deviation interval of $2.97V$ to $2.986V$ depending on the approximation in the simulation. At a logical **HIGH** input the overlapped input deviation signal reaches an output interval of $22.99V$ to $23.968V$ also depending on the approximation method.

5.3 Power-line Communication (PLC) System

Within this section, I demonstrate the proposed uncertainty tracing and analysis methodologies which are fully integrated into the *SESYD* framework using a Power-line communication (PLC) system. A PLC system transfers payload sensory data between two processing nodes using the available power line wires as a communication medium. Hence, data bit signals are sequentially modulated on the 230/110 V 50/60 Hz power line. For the rest of the section an Austrian power grid having 230 V, 50 Hz is assumed (supply quality standard according to $EN50160$ [DIN11]).

PLC is on the first side a state of the art research field (channel characteristics and modeling, performance increase, optimization, etc. [LGLK14, BKBD06]) and on the other side several applications using PLC are well embedded in the field (in-home communication, data transfer for smart metering services, etc. [LL15, SS16]).

In this example, uncertainties can be divided into deviations of the power line signal and variations in the transceiver modules. Decreases in the power supply quality modeled as frequency and magnitude variations may be caused by alternation of load, unbalanced power generation and consumption on grid branches, electromagnetic influences, switching events, etc. Internal parts of the transceiver chips have commonly known PVT (Process, Voltage and Temperature) uncertainty effects of analog circuits (inaccurate gain, offset drift, temperature impact, voltage supply etc.) Detailed description of variations explicitly considered are described later. However, under the impact of modeled uncertainties and resulting transmission the total system performance is not clear first. The proposed methods guide a designer to increase the system insight and finally to a starting point for optimization.

Block diagram and functional description

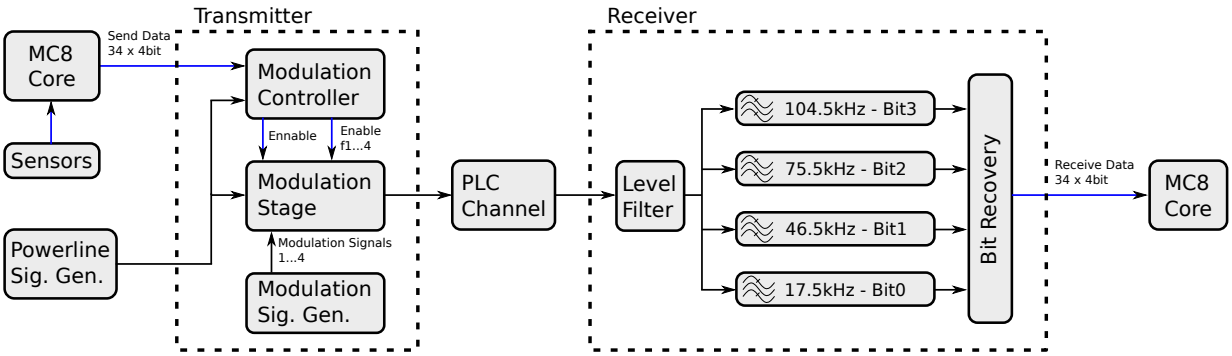


Figure 5.12: Block diagram of the modeled Power-line communication data transmission system. Transmitter and receiver modules are connected to MC8 CPU cores generating and processing application data. A power line signal generator in combination with a simple channel model represents the power distribution grid in a house.

Figure 5.12 illustrates the block diagram of the modeled system. For this example, only the forward data path transmission from the left to the right MC8 processing core is modeled. As data payload a set of eight 16 bit sensor data from a weather station application (temperature, humidity, illumination, air pressure, rainfall, wind speed, wind direction and ozone level) are transmitted. For an application model also including backward transmission, the involved communication equipment just gets mirrored realizing a bidirectional transceiver chip. The power line signal is modeled by a specific signal generator as illustrated in the figure. The positive half cycle (10 ms) of the sinusoidal 230 V, 50 Hz signal is applied to the transmitter module. This time period is divided into 34 transmission slots each 285 μs long and a 152 μs pause (see Figure 5.13). This timing used in the modules of the transmitter and receiver is derived from positive zero-cross of the power line signal. Totally, this results in a transmission time of 152 μs+34·85 μs = 9.842 ms. Thus, the maximum calculated operating power line frequency is 50.803 Hz. During every single slot, four bit of payload data is transferred simultaneously. This results in a unidirectional data transmission rate of 6.64 $kbps$. Inside the transmitter, a modulation controller is responsible for generating control signals according to the slot timing and the applied input data. Sinusoidal modulation signals having frequencies of 17.5 kHz (LSB-Least Significant Bit), 46.5 kHz, 75.5 kHz and 104.5 kHz (MSB-Most Significant Bit) are synthesized in the modulation signal generator. According to applied four bit transmission data packets represented by enable signals corresponding modulation frequency are modulated on the powerline carrier (see Figure 5.13).

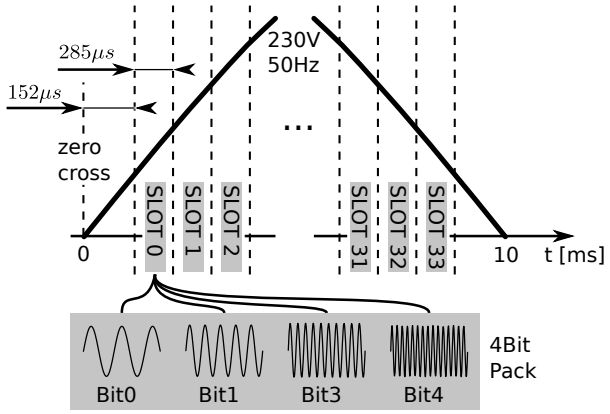


Figure 5.13: Data transmission scheme for the PLC system. The timing is synchronized by the power line zero cross. After a 152 μs pause 34 transmission cycles each including four bit of information are applied.

For the channel characteristic, I defined a transmission characteristic based on Figure 5.14. The diagram shows measured data for an in-house, application published in [BKBD06]. I designed a FIR (finite impulse response) filter, with a length of 101 samples, approximating the shown channel characteristic for the house.

At the receiver side, first a level filter subtracts a 230 $V/50$ Hz function to eliminate large signal amplitudes of the carrier. The following filter bank included as four digital bandpass filters as illustrated in the block diagram (Figure 5.13). Depending on their specific cut-off characteristics their history length are 141 (LSB), 171, 169 and 169 (MSB) samples. Filter coefficients and history length are adjusted to majorily transfer the spectral parts indicating the modulated bit information. The corresponding FFT post-processed spectrum of the signal applied to the filters is plotted in Figure 5.15. The plot shows, that if all four bits are set to *TRUE* each defined frequency component forms a clearly defined peak within the spectrum. The according FIR (finite

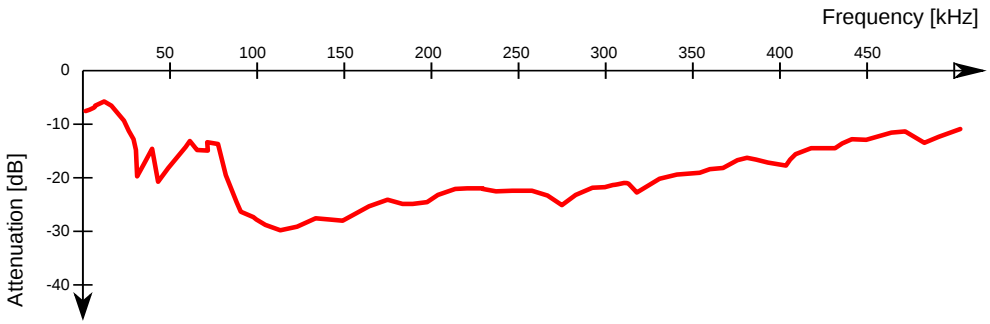


Figure 5.14: PLC channel characteristic of in-house transmissions [BKBD06]. For this example, the illustrated in-house channel characteristic is approximated by a FIR filter.

impulse response) filters are designed with the help of the online tool Tfilter [22] see Section 3.5. Pass-bands are $15\,kHz$ wide and located symmetrically around the expected modulation frequencies. The bit recovery block illustrated in Figure 5.12 implements a recovery algorithm which is equal for all four transferred bits in a pack. The corresponding filtered waveforms are sampled at a clock frequency of $1\,MHz$. If the absolute value of a sample is greater than a defined threshold, a counter is increased. The counter value is evaluated at the end of each slot. If it exceeds a predefined counter threshold value the corresponding bit is recovered to *TRUE*. A tight synchronization between sending and receiving node is not required. Receive and send procedures are triggered by positive zero-cross events of the power line signal.

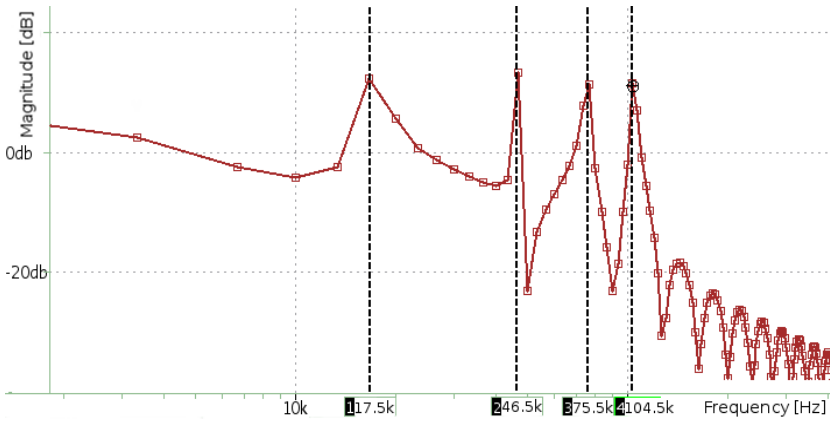


Figure 5.15: FFT view of the received signal after the level filter. The figure illustrates an all **TRUE** four bit pack indicated by clear peaks at the designed frequency components of the corresponding modulation signals.

Parameter setup and uncertainties within the model

As illustrated in Figure 5.16 in total 31 deviation effects are modeled. Their initial influence on the functional blocks in the transmission system is given in red. Variations in the PL generation module, are set to the limits specified in the $EN50160$ [DIN11] standard. The modulation signal generator includes four identical analog signal generators for each used sinusoidal modulation signal. We assume a common synchronization and amplification stage for each generator. Thus, a correlated deviation representing the amplifier's process variation is modeled. In addition, I model an individual offset variation for each single modulation signal generator. At the transmitter side,

the temperature and the supply voltage drift is modeled. The PLC channel used for this example has a varying attenuation factor over the full frequency spectrum. The level filter, which subtracts a $230\,V/50\,Hz$ function to eliminate large signal amplitudes has slight variations in the synthesized subtraction function. Last at the receiver side I modeled deviations in the bandpass filter bank. All filters have a correlated gain variation and each filter have an individual gain variation in the specified filter characteristic. Similar to the transmitter module temperature and supply voltage variations at the receiver module are modeled as well.

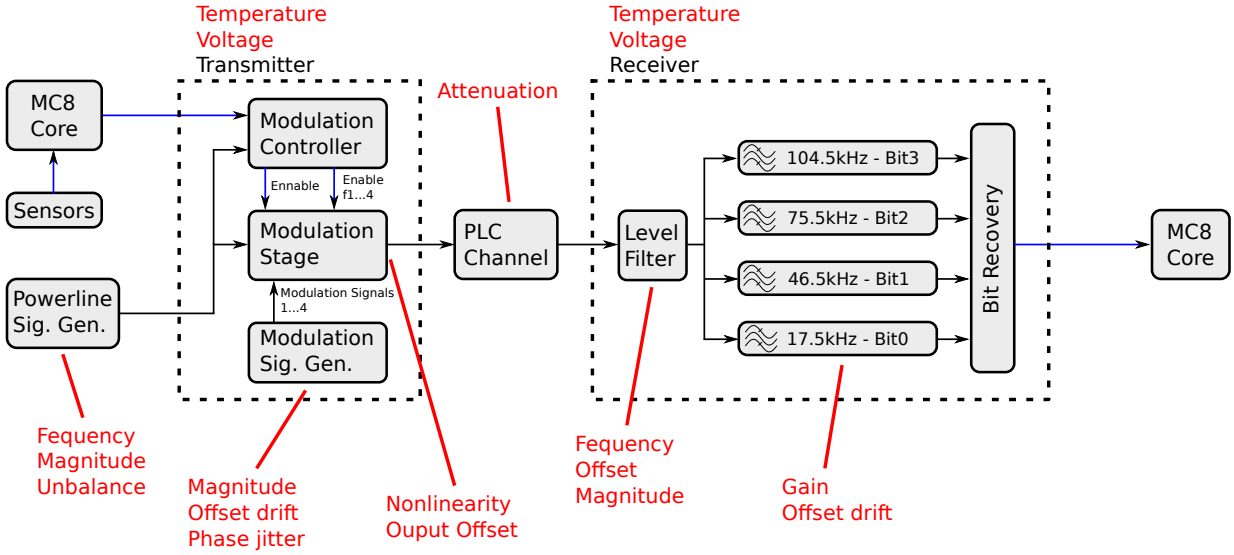


Figure 5.16: Block diagram of the modeled Power-line communication data transmission system and modeled uncertainties given in red.

Figure 5.16 just informs about the uncertainty causes represented as deviation within the model. The following uncertainty matrix, according to the template given in Figure 4.1, is evaluated for detailed documentation of uncertainty impact to parameters in the system. The presented table also includes the configuration of the application parameters (nominal values).

Table 5.5: Uncertainty matrix for the PLC example - detailed caption on the next page

Module	Submodule	Parameter	Nominal Value	Uncertainty cause							Lo, Le, N
				$\emptyset$	T_T	V_T	$P_{Msig}-ampl$	$P_{Msig}-drift$	T_R	V_R	$P_{Filters}$	
Clock signals	Anal. sampling clock	Frequency	$5kHz$	x	x	x	x	x	x	x	x	e
	Digital clock signal	Frequency	$2kHz$	x	x	x	x	x	x	x	x	e
PL signal		Frequency	$50Hz$	$0.5Hz$	x	x	x	x	x	x	x	e, s, 1
		Voltage magnitude	$230\,V$	$0.006 \cdot nv$	x	x	x	x	x	x	x	e, d, 1
		Voltage unbalance	$0\,V$	$2.3\,V$	x	x	x	x	x	x	x	e, s, 1
Transmitter	Modulation signal 1	Magnitude	$5\,V$	x	x	x	$0.02 \cdot nv$	x	x	x	x	i, d
		Offset	$0\,V$	$0.05\,V$	$0.1\,V$	$0.07\,V$	x	x	x	x	x	i, s, P
		Frequency	$17.5kHz$	x	x	x	x	x	x	x	x	i
		Phase	$0rad$	x	$0.005rad$	x	x	$0.04rad$	x	x	x	i, s
	Modulation signal 2	Magnitude	$5\,V$	x	x	x	$0.015 \cdot nv$	x	x	x	x	i, d
		Offset	$0\,V$	$0.05\,V$	$0.15\,V$	$0.07\,V$	x	x	x	x	x	i, s, P
		Frequency	$46.5kHz$	x	x	x	x	x	x	x	x	i
		Phase	$0rad$	x	$0.005rad$	x	x	$0.04rad$	x	x	x	i, s
		Magnitude	$5\,V$	x	x	x	$0.01 \cdot nv$	x	x	x	x	i, d
	Modulation signal 3	Offset	$0\,V$	$0.05\,V$	$0.15\,V$	$0.07\,V$	x	x	x	x	x	i, s, P
		Frequency	$75.5kHz$	x	x	x	x	x	x	x	x	i
		Phase	$0rad$	x	$0.005rad$	x	x	$0.035rad$	x	x	x	i, s
		Magnitude	$5\,V$	x	x	x	$0.01 \cdot nv$	x	x	x	x	i, d
	Modulation signal 4	Offset	$0\,V$	$0.05\,V$	$0.15\,V$	$0.07\,V$	x	x	x	x	x	i, s, P
		Frequency	$104.5kHz$	x	x	x	x	x	x	x	x	i
		Phase	$0rad$	x	$0.002rad$	x	x	$0.03rad$	x	x	x	i, s
	Modulation stage	Nonlinearity stage 1	1	0.01	x	x	x	x	x	x	x	i, s, P
		Nonlinearity stage 2	1	0.01	x	x	x	x	x	x	x	i, s, P
		Nonlinearity stage 3	1	0.01	x	x	x	x	x	x	x	i, s, P
		Nonlinearity stage 4	1	0.01	x	x	x	x	x	x	x	i, s, P
		Output offset	$0\,V$	x	$0.01\,V$	$0.01\,V$	x	x	x	x	x	i, s
Channel		Attenuation	1	$20\,dB$	x	x	x	x	x	x	x	e, s, 1
Receiver	Level Filter	Frequency	$50Hz$	$0.001\,Hz$	x	x	x	x	x	x	x	i, s, 2
		Offset	$0\,V$	$0.02\,V$,$0.05\,V$	x	x	x	x	$0.002\,V$	$0.01\,V$	x	i, s, 2
		Magnitude	$230\,V$	$0.0005 \cdot nv$	x	x	x	x	x	x	x	i, s, 2
	Receiver filter 1	Gain	1	0.02	x	x	x	x	x	x	0.01	i, s, P
		Offset	$0\,V$	$0.03\,V$	x	x	x	x	$0.02\,V$	$0.01\,V$	x	i, s, P
	Receiver filter 2	Gain	1	0.02	x	x	x	x	x	x	0.01	i, s, P
		Offset	$0\,V$	$0.03\,V$	x	x	x	x	$0.02\,V$	$0.01\,V$	x	i, s, P
	Receiver filter 3	Gain	1	0.02	x	x	x	x	x	x	0.01	i, s, P
		Offset	$0\,V$	$0.03\,V$	x	x	x	x	$0.02\,V$	$0.01\,V$	x	i, s, P
	Receiver filter 4	Gain	1	0.02	x	x	x	x	x	x	0.01	i, s, P
		Offset	$0\,V$	$0.03\,V$	x	x	x	x	$0.02\,V$	$0.01\,V$	x	i, s, P

Caption for the table on the previous page:
$\emptyset$ = uncorrelated first-order effects
$T_{T/R}$ = Transmitter/Receiver side temperature
$V_{T/R}$ = Transmitter/Receiver side supply voltage level
$P_{Msig} - Ampl$ = Process variation of the modulation signal amplifier
$P_{Msig} - drift$ = Phase drift of the modulation signal amplifier
Lo, Le, N = Deviation location, level and nature according to Subsection 2.2.2
e/i = external/internal
s/d = static/dynamic deviation
P = The deviation is correlated with a specific modeled Process variation of the according module
1 = Deviation caused by disturbances on the power grid
2 = Deviation caused by parameter variation caused by parameter configurations
nv = Nominal value

Numerical simulation of the PLC system

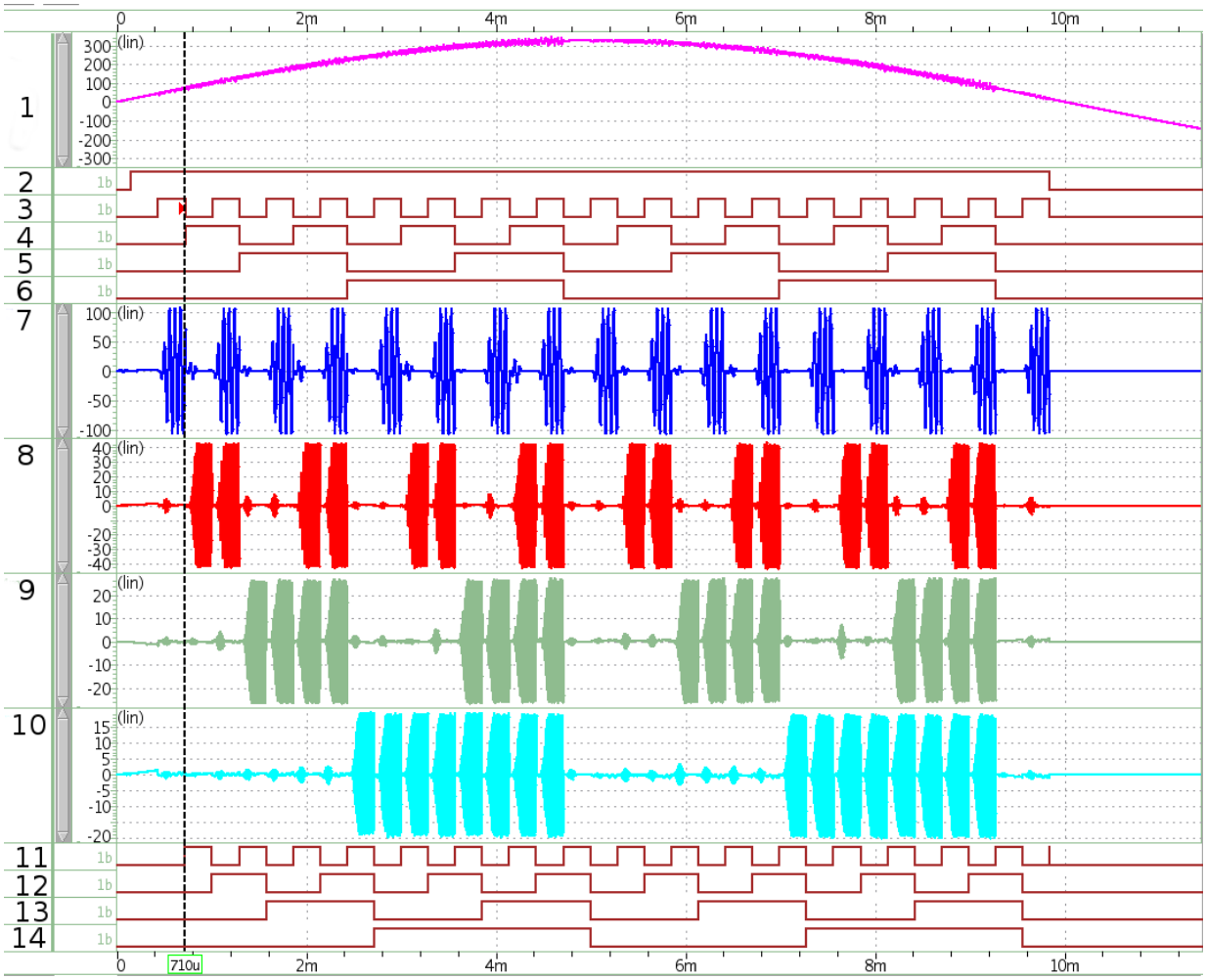


Figure 5.17: Exact numerical simulation run of the PLC transmission system

First, I demonstrate the behavior of the PLC system using not deviated parameters. Therefore, I executed an exact numerical simulation run not considering any deviations. Figure 5.17 illustrates a comprehensive set of input and output signals. The digital processor core MC8 executes for this demo an assembly program that sequentially increases a 4-bit counter value. The frequency for counting up is set that for each slot a new counter value is presented for transmission. Waveform 1 in Figure 5.17 shows the modulated power line signal which is presented at the output of the transmitter block. Waveform 2 illustrates a digital enable signal for the modulation stage which is generated based on the zero-cross events of the PL signal. Waveforms 3-6 illustrate the digital signals representing transmission data (in this case an up counting value). Waveforms 7-10 illustrate the output of the receive filters for every single bit. These signals are forwarded to the bit recovery block. Signals 11-14 illustrate the recovered binary information. Figure 5.17 shows that in the exact case the input information can be recovered without any error ($BER = 0$ - Bit Error Rate) at the receiver side.

System simulation and analysis using multi-run methods

Within this part of the example, I present two selected experiments (out of a series) where I simulate and analyze the given PLC transmission system using the multi-run module implemented in the *SESYD* framework. For simulation control, resetting the model, random pattern generation, processing the results, etc. I used the concepts presented in Section 3.4.

For the first experiment, variations of the power line which are modeled by the following pattern generators are considered:

- Power line frequency: Normal distributed random pattern generator - Mean= $50\,Hz$, Standard deviation = $0.2\,Hz$
- Constant voltage offset: Uniform distributed random pattern generator - Lower bound = $-5\,V$, Upper bound = $5\,V$
- Channel delay characteristic resulting in a phase shift between sender and receiver power line signal: Uniform distributed random pattern generator - Lower bound = $-0.01\,rad$, Upper bound = $0.01\,rad$
- Electromagnetic noise. This generator is updated after each simulation step: Normal distributed random pattern generator - Mean= $0\,V$, Standard deviation=$4\,V$

The deviated power line signal is illustrated in the upper waveform of Figure 5.19. As a goal, we are interested in the number of transmission errors within one slot. Figure 5.18 illustrates a two-dimensional diagram plotting frequency and offset voltage against occurring slot errors. The printed surface forms a trough having a minimum ($BER = 0$) at the $50\,Hz/0\,V$ point. As a result of this multi-run simulation, the itemized power line variations are fully considered and the resulting system responses can be studied in detail. Impact factors to bit transfer error rates can be evaluated in the form of input valuations, but they can not be traced to their initial cause explicitly. Figure 5.19 illustrates the power line half-cycle waveform and the corresponding filtered signal of the LSB. On the left panel of the diagram illustrates the traced multi-run waveforms named by their recorded multi-run simulation round.

In the second experiment (an extension of the first ones) a potential optimization of the receiver is addressed. As described previously, for each slot the bit recovery algorithm is based on counting

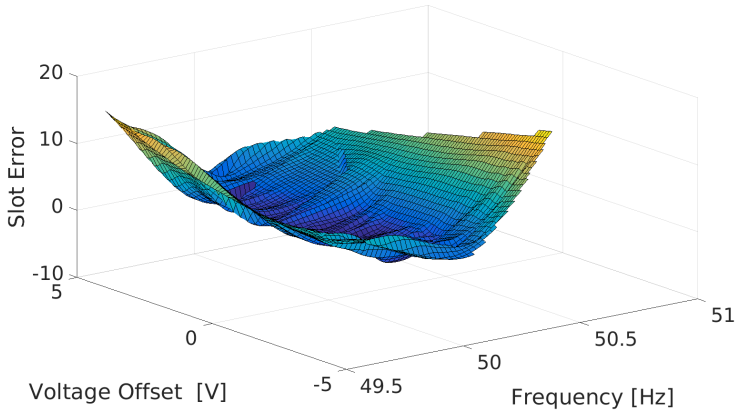


Figure 5.18: Slot errors under the dependence of the power line frequency and a static offset voltage. These parameters have significant impact on the occurrence of erroneous data transmission. Due to the design of sender and receiver nodes the minimum (no errors) is located around the $50\,Hz/0\,V$ point.

samples which absolute values are greater than a defined threshold. As illustrated in the lower waveform of Figure 5.19, showing the filtered LSB signal, due to the occurring deviations the definition of this threshold value may become a tricky task. If it is set too high just a few samples corresponding a correct bit detection exceed this value. This increases the probability of not detecting bits which are set to **TRUE**. On the other side, if the level is set too low disturbances, as occurring if the bit is set **FALSE**, leads to incorrectly triggered counter increases. However, this results in false positive detection of bits. The main challenge of this experiment is to find optimal threshold levels for bit recovery. We consider the four threshold parameters as design variations. Thus, their static values are replaced by sequence generators. Lower and higher bounds are set to ± 3 around their approximated values. So there are eight stimuli generators within this simulation. Setting the corresponding end rule to exhaustive (considering all possible sequencer value combinations) results into 9900 single simulation rounds. For each multi-run round $11\,ms$ are simulated at a sampling frequency of $1\,MHz$. The simulation has been executed on machine with the following specifications: CPU: Intel Core I7-6700K 4.00 GHz 4Core, RAM: 4x 16GB DDR4 2400 RAM, OS: Ubuntu 16.04 LTS. On this computer the required simulation time was $7\,h$ and $35\,min$.

Results of the full simulation are collected in a **.csv* table file. A part of the full table is illustrated in Figure 5.20. Column one holds the simulation run, columns two to seven the given parameter set specified by the stimulus generators and the last column indicates the occurred slot errors (Level bx stand for the described detection threshold level for bit$x$). The first task towards optimization is to filter the table just including runs with no slot errors. 1528 simulation runs indicating diverse parameter sets fulfills that criteria. This table is a starting point for further multi-criteria optimization which is out of scope for this work. However, optimization is based on results provided form the multi-run simulation considering parasitic power line deviations as well as design variations used for optimization.

Simulation and analysis of the PLC system using AAFs

First, I show some general results of the PLC system simulation using AA. Second, I focus int he bit error rate property of the system, and show how it can be elaborated by applying the

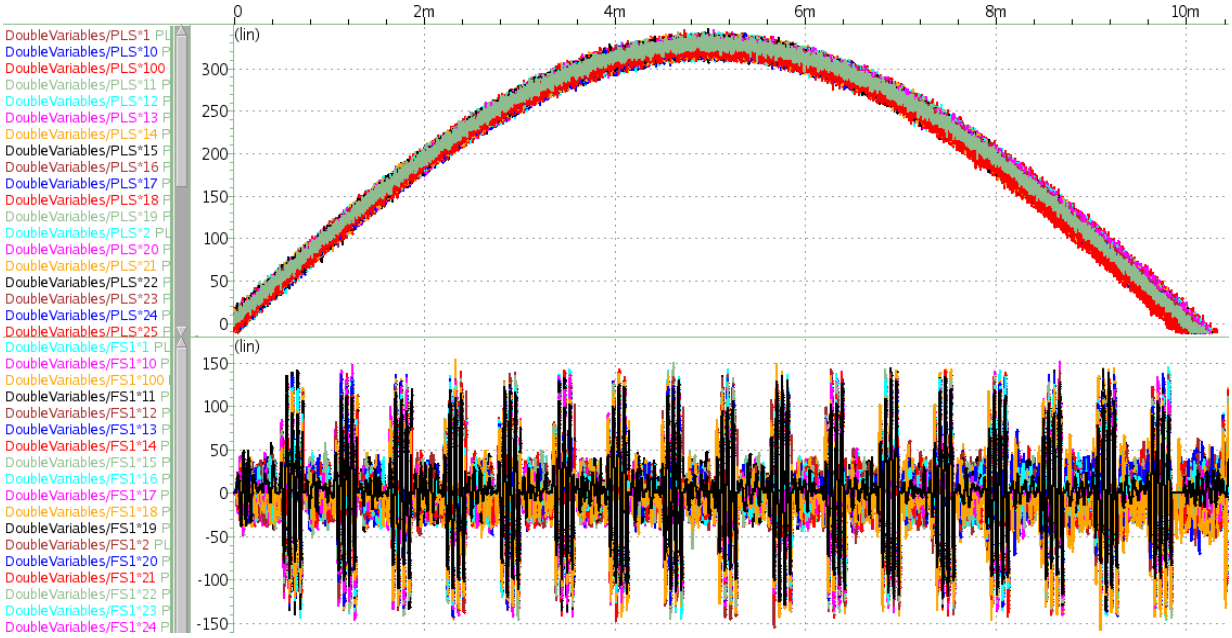


Figure 5.19: The upper waveform illustrates multiple plots of the power line signal including deviations represented by the pattern generators. The lower graph shows the filtered LSB signal at the receiver which is handed over to the bit recovery block. As illustrated the deviations of the power line cause reasonable disturbances in filtered signals.

presented analysis procedures. Uncertainties and the corresponding causes included in the PLC system are given in Table 5.5. There are 31 user deviation symbols and five system deviation symbols instantiated. For system deviation symbols the corresponding AAF object IDs and the associated operations are given as symbol source information. The following table is derived from the output of the symbol management module (the output listing itself holds more details and is thus complex and long).

Figure 5.21 illustrates a part of the transmitted PL signal. The dashed brown line labels the center value, and each subinterval represents an individually modeled uncertainty. As indicated, the first (power line frequency), second (power line magnitude) and fourth (transmitter side temperature) subintervals have a significant impact on the resulting voltage range.

Figure 5.22 illustrates the first 1 *ms* of a transmission cycle (a full PL half cycle). Similar to figure Figure 5.17 transmitted and received bit sequences as well as analog signals are shown. But in this case, blue lines indicate the corresponding bounds given by AAFs under consideration of all 31 modeled uncertainties included in the PLC system.

The objective selected for the subsequent system analysis is the evaluation of deviations responsible for potential transmission errors (causes for bit errors, $BER \neq 0$). As already described in the last paragraphs where I use multi-run methods for analysis, the setup for threshold values for the bit recovery algorithm is critical.

The recovery, block increases an internal counter if the absolute signal value is greater than a configured threshold. Thus, at the end of each slot, the bit is recovered to **TRUE** or **FALSE** explicitly depending on this counter value. If an AAF, as for this analysis, is presented to the bit recovery algorithm a similar situation as described in Subsection 3.2.3, where an AAF is compared to a numeric threshold value (see Figure 5.24), is given.

Run Nr.	Level b1	Level b2	Level b3	Level b4	frequency	Offset Voltage	Slot Error
5	16	8	6	10.8	50.0269	0.673624	0
22	16	9.4	6	8.7	49.9723	-0.693153	0
23	16	9.4	6	9.4	49.9391	-1.52269	0
71	16	12.9	6	8	50.0647	1.61722	0
82	16	13	6	8.7	49.9842	-0.39509	0
104	16.7	8.7	6	10.1	49.9987	-0.0329258	0
123	16.7	10.1	6	9.4	49.9586	-1.03585	0
133	16.7	10.8	6	9.4	49.951	-1.22503	0
164	16.7	12.9	6	10.1	49.9393	-1.51643	0
174	16.7	13	6	10.1	49.9764	-0.590811	0
175	16.7	13	6	10.8	50.0058	0.145279	0
181	17.4	8	6	8	50.0158	0.394565	0
183	17.4	8	6	9.4	49.9873	-0.31744	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 5.20: A part of the full result table for experiment 3. Each row representing a specific simulation run indicates a parameter set resulting in a simulation run with no occurring slot errors. Level b1 to b4 stand for the considered bit recovery threshold values which are continuously modified around a previously approximated value. Optimization algorithms may finally evaluate an optimal threshold parameter set based on this result table.

Table 5.6: Uncertainty and associated symbols in the modeled PLC system. Strictly speaking this table hols the symbol assignments for uncertainties given in the uncertainty matrix Table 5.5. S.d. = System Deviation Symbol, src. = source identifier

ϵ_x	Models uncertainty in	ϵ_x	Models uncertainty in
1	PL signal frequency	19	Receiver filters output gain
2	PL signal unbalance	20	Level filter frequency
3	Relative PL signal magnitude	21	Level filter offset
4	Tansmitter side temperature	22	Level filter magnitude
5	Transmitter side voltage level	23	Level filter offset (P)
6	Modulation signals pahse drift	24	Receiver filter 1 gain
7	Modulation signals amplitude	25	Receiver filter 1 offset
8	Modulation signal 1 offset	26	Receiver filter 2 gain
9	Modulation signal 2 offset	27	Receiver filter 2 offset
10	Modulation signal 3 offset	28	Receiver filter 3 gain
11	Modulation signal 4 offset	29	Receiver filter 3 offset
12	Nonlinearity Modulation stage 1	30	Receiver filter 4 gain
13	Nonlinearity Modulation stage 2	31	Receiver filter 4 offset
14	Nonlinearity Modulation stage 3	32	S.d. - PLC ch. gain mult. - src: ID68 * ID65
15	Nonlinearity Modulation stage 4	33	S.d. - rec. filter 1 gain mult. - src: ID234 * ID225
16	PLC channel attenation	34	S.d. - rec. filter 2 gain mult. - src: ID386 * ID377
17	Receiver side temperature	35	S.d. - rec. filter 3 gain mult. - src: ID568 * ID559
18	Receiverside voltage level	36	S.d. - rec. filter 4 gain mult. - src: ID748 * ID739

The left part of the figure shows different cases of how the threshold level can be located relative to the AAF central value (in blue) and its bounds (in red). However, there are four cases where the bit recovery explicitly defined also under the presence of uncertainty. For all other cases, the bit recovery is uncertain. In contrast to the example presented in Subsection 3.2.3 the evaluation of an output AAF is not required, strictly speaking, it is sufficient to know if the recovery is uncertain or not. For implementing the described analysis in principle, I used two counters. One incrementing its value if the bit recovery is resulting uncertain for a point in simulation

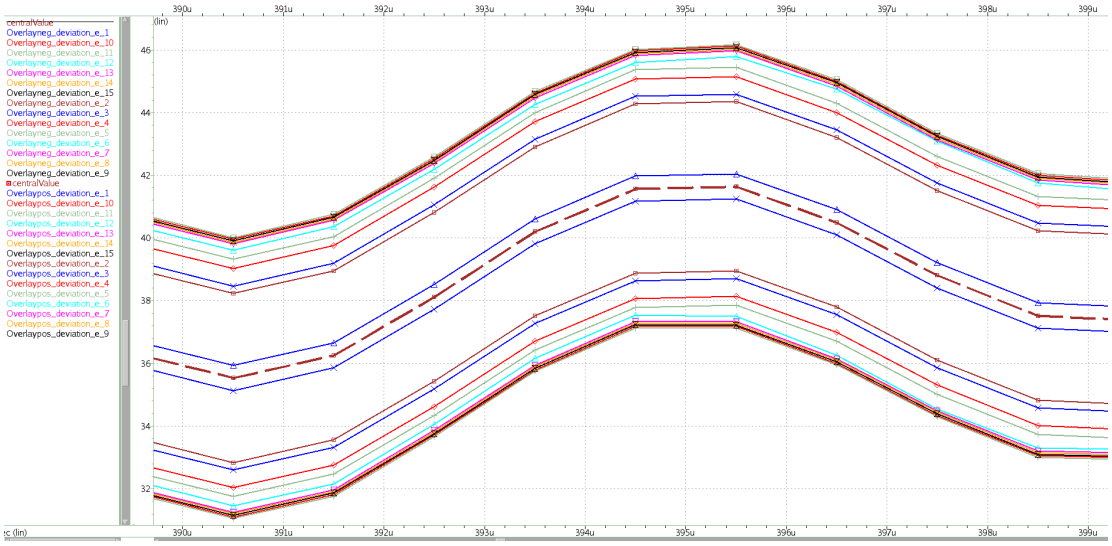


Figure 5.21: The transmitted PL signal and the included partial deviations associated with symbols ϵ_1 to ϵ_{15}

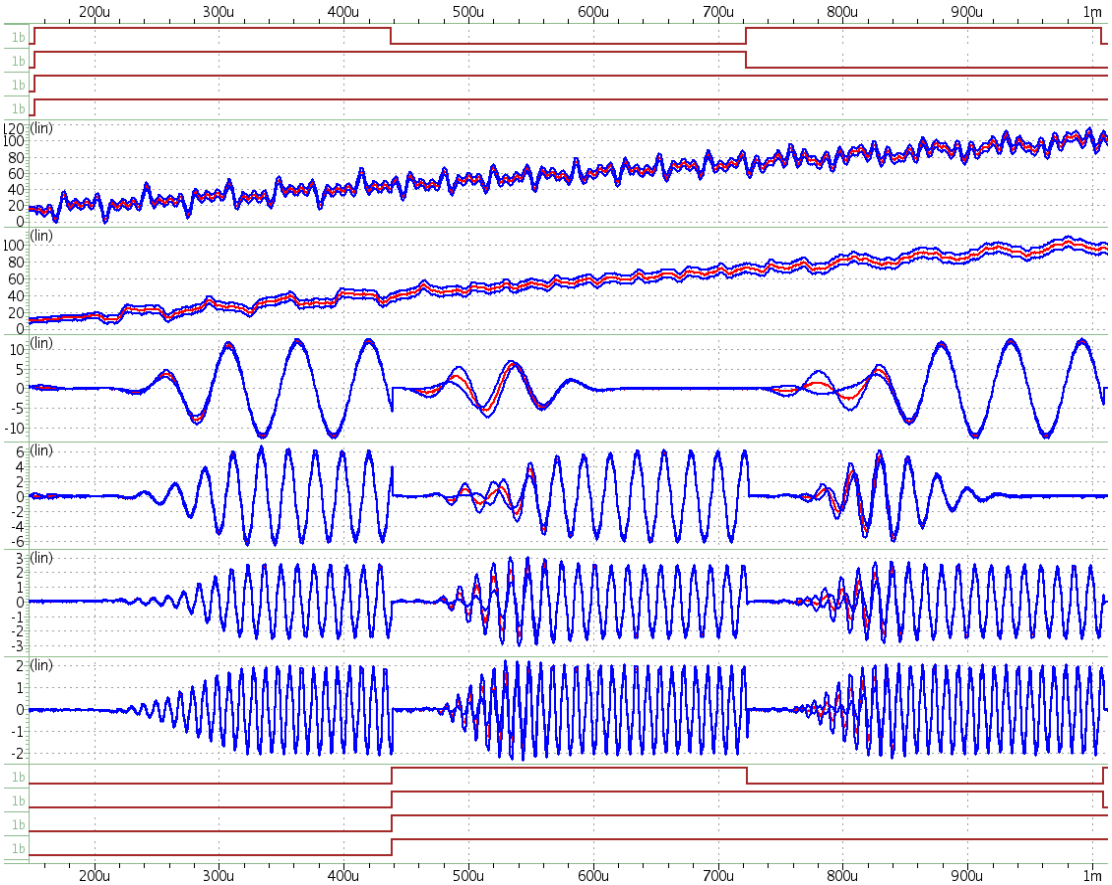


Figure 5.22: Time domain results. Upper and lower signals illustrate the binary transmitted and received values. The analog traces illustrate the transmitter output, receiver input and the four filtered signals used for the bit recovery process.

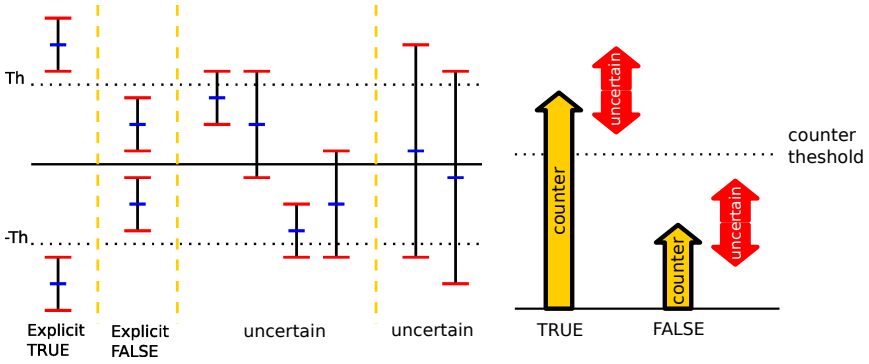


Figure 5.23: Possible cases, if the recovery algorithm is extended for AA.

time and another one increasing if the recovery is explicit. At the end of a slot, the uncertain result counter is added and subtracted from the explicit result counter. If the result of this addition/subtraction is higher or less than the set threshold value, the bit can be recovered to **TRUE** or **FALSE** respectively (see the right illustration of Figure 5.24). However, the point is that the recovery at the end of the slot can still be explicit also under the presence of uncertain bit recovery evaluations during the inspected slot. Strictly speaking, this is the goal of a robust bit recovery concept.

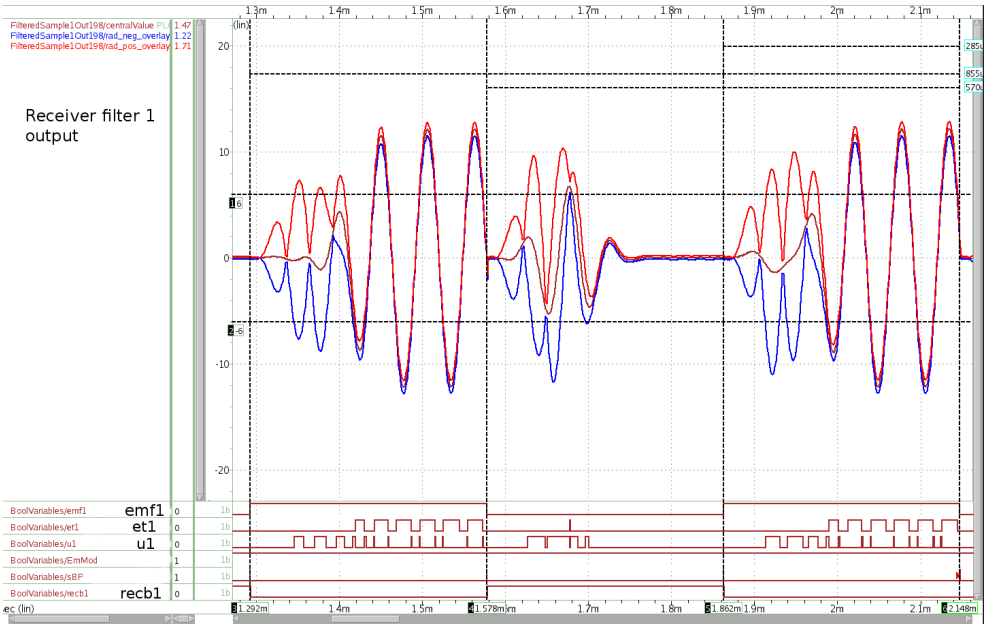


Figure 5.24: Analysis example of three selected time slots recovering a binary signal based on uncertain and explicit recovery results.

An example for the transmission of the LSB in an arbitrary slot is illustrated in Figure 5.24. The vertical cursors delimit the time windows of the shown transmission slots. The horizontal courses at ± 6 mark the threshold value for bit recovery. The central value (brown line) is surrounded by the accumulated bounds of all included partial deviations (radius) (red and blue line). According to the described cases illustrated in Figure 5.24 there are points in time when the recovery situation is explicit, and others where it is uncertain. Corresponding binary signals (representing the situation) are given as $et1$ (explicitly **TRUE**) and $u1$ (uncertain). For each simulation point

where the mentioned binary signals are **TRUE** corresponding counters are incremented. At the end of each slot, the binary input information ($emf1$ - enable modulation signal 1) is tried to be recovered by evaluation of the current counter values (the recovered signal is the $recb1$ signal). As indicated, the first two slots in Figure 5.24 the recovered information is equal to the transmitted one, where at the third slot the recovery fails, representing a bit-transmission error. The cause of the transmission error at this stage of system analysis can be identified as a too large value of the uncertain recovery counter caused by too large accumulated patial deviations.

```
Slot 5
Slot b1: got truecount1 2  - uncertcount b1 27 - range [-25:29] - 1
Slot b2: got truecount2 91 - uncertcount b2 31 - range [60:122] - 1
Slot b3: got truecount3 0  - uncertcount b3 6 - range [-6:6] - 1
Slot b4: got truecount4 109  - uncertcount b4 50 - range [59:159] - 1
Slot 5 send: 1010  received 1010 equal and result explicit: 1

...

Slot 27
Slot b1: got truecount1 97  - uncertcount b1 52 - range [45:149] - 0
Slot b2: got truecount2 91  - uncertcount b2 52 - range [39:143] - 0
Slot b3: got truecount3 17  - uncertcount b3 44 - range [-27:61] - 0
Slot b4: got truecount4 0 - uncertcount b4 46 - range [-46:46] - 1
Data and Explicit Vector 1000 0000
Slot 27 send: 0011  received 0000 equal explicit: 0
```

```
– >8 –
// Bit recovery
#define TH_LEVEL_1 6
#define TH_LEVEL_2 3.5
#define TH_LEVEL_3 1.4
#define TH_LEVEL_4 1.0
#define TH_BITONE_COUNT1 50
#define TH_BITONE_COUNT2 50
#define TH_BITONE_COUNT3 50
#define TH_BITONE_COUNT4 50
– >8 –
```

Figure 5.25: Debug output listing of the BER analysis process

Of course, I did this single bit transmission analysis for all bits and all transmission slots. In Figure 5.25 I give the debug output listing of the analysis for transmission slot 5 and 27 and the configured counter and value thresholds as defined in the model. For example, bit 1 in slot 5 can be explicitly recovered to **FALSE**. The upper range of the recovery counter (in light blue) 29 is small er than the threshold of 50. The recovery is explicit for all bits in the slot (marked by the 1's in green). In slot 27 the situation is worse. Three of the four bits cannot be recovered explicitly. The counter values for explicit recovery looks good but caused by the deviations of the recovery input signal there are too many uncertain recovery states within the slot. For transmission test data again the counting 4bit value as illustrated in Figure 5.17 is used. However, the analysis results that under the specified parameter setup and the given uncertainties 43 of 136 bits cannot be recovered explicitly $\rightarrow BER = 0.316$. This does not necessarily mean that the average bit error rate is that high, but under worst-case deviation considerations for each transmission slot, 31.6% of the transmitted bits are erroneous.

At the next step, this analysis is refined towards the evaluation of more knowledge about the cause of that high worst-case transmission errors. Therefore, I continue with impact analysis using temporal and structural traces.

As discussed in Subsections 4.3.1 and 4.6.3 the integrated impact of partial deviation is calculated. Figure 5.26 illustrates the accumulated impact of $\epsilon_{1,2,3,16}$ uncertainties for the four filter outputs (bit0 to 3 - top to bottom resp.). Corresponding partial deviation values (in this case the absolute values) are integrated over time. $\epsilon_{1,2,3,16}$ are external uncertainties caused by PL variations. The dashed lines are the gain variations of the corresponding filters. As a result, the power line deviations in frequency, magnitude, unbalance and channel attenuation (only these are illustrated in the figure) have a major impact in a global perspective. Thus, for the following analysis, I mainly focus on $\epsilon_{1,2,3,16}$ uncertainties and analysis of the LSB bit output.

Further, temporal tracing is used for the evaluation of the point in time when the critical uncer-

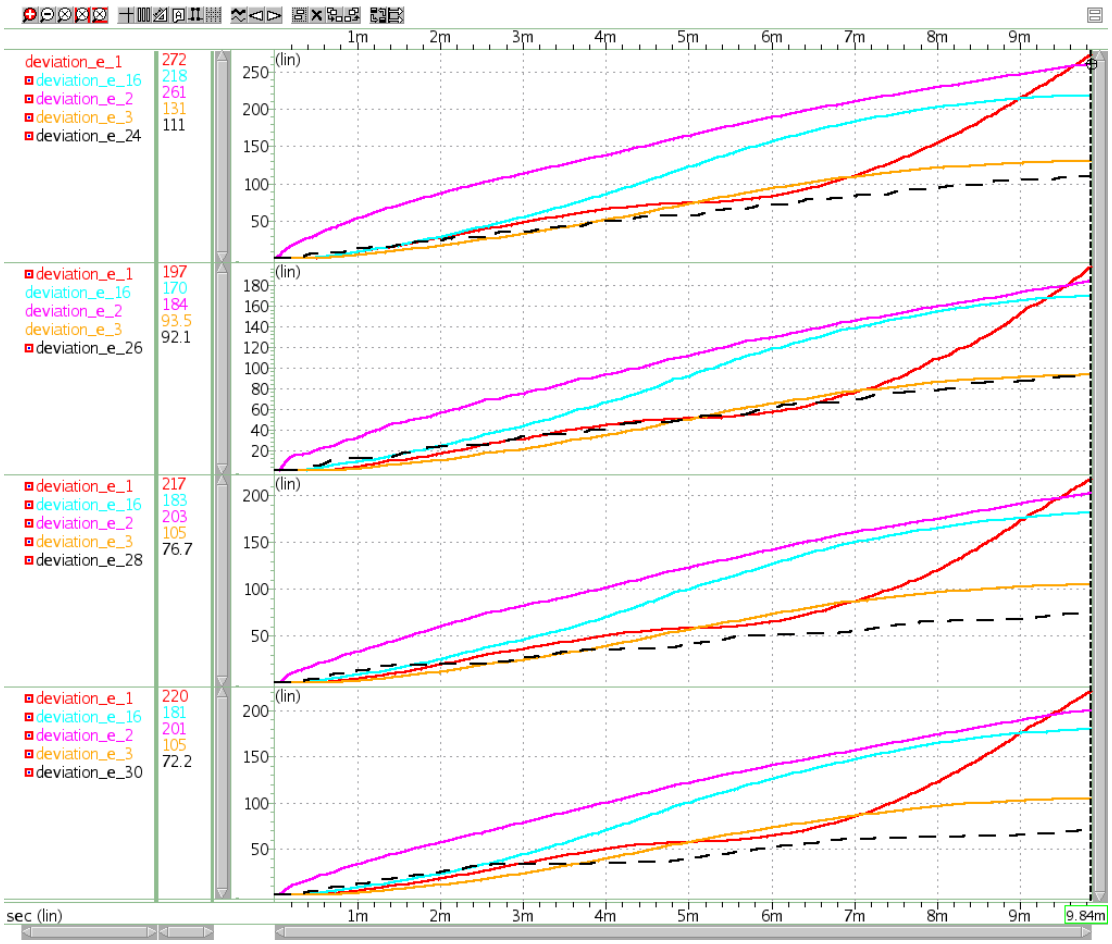


Figure 5.26: Integrated absolute deviations at the receiver filter outputs (shows only 5 having the main impact)

tainties are propagated to the filter output. Figure 5.27 shows the impact of the $\epsilon_{1,2,3,16,24}$ partial deviations in percent. The trend indicates that the impact of the partial deviation associated with ϵ_1 (PL frequency variation) is getting dominant at the end of the transmission cycle.

To verify that, the absolute values for the identified partial deviations are plotted in Figure 5.28. As indicated, in the upper plot, first the pink line forms the peaks in transmitted bit periods, then cyan and finally red gets dominating. Due to the limited time windows (the receiver filter output is just relevant if the LSB is **FALSE**) the situation is more clear at inspecting the output of the level filter (lower plot of Figure 5.28). The partial deviations for PL frequency (red), unbalance (pink), magnitude (orange) and channel attenuation (cyan) $\epsilon_{1,2,3,16}$ are shown. To identify the peak also the sum of the mentioned partial deviation (dark blue line) is illustrated. The dashed lines in the plot illustrate the partial deviation impact of receiver and transmitter side temperature and supply voltage variation. For this analysis, these impacts are not relevant. As a result, I identified not a single critical point in time but a period (trend). The selected partial deviations impact, the total variation and thus the impact to the BER gets worse at the end of the transmission cycle. However, for the following structural analysis, I focus on the last transferred bit pack (cursor in Figure 5.28) at 9 ms to 10 ms.

In Figure 5.29 the path for further structural analysis $\tilde{p} =< s_1, s_2, s_3, s_4, s_5 >$ is illustrated in

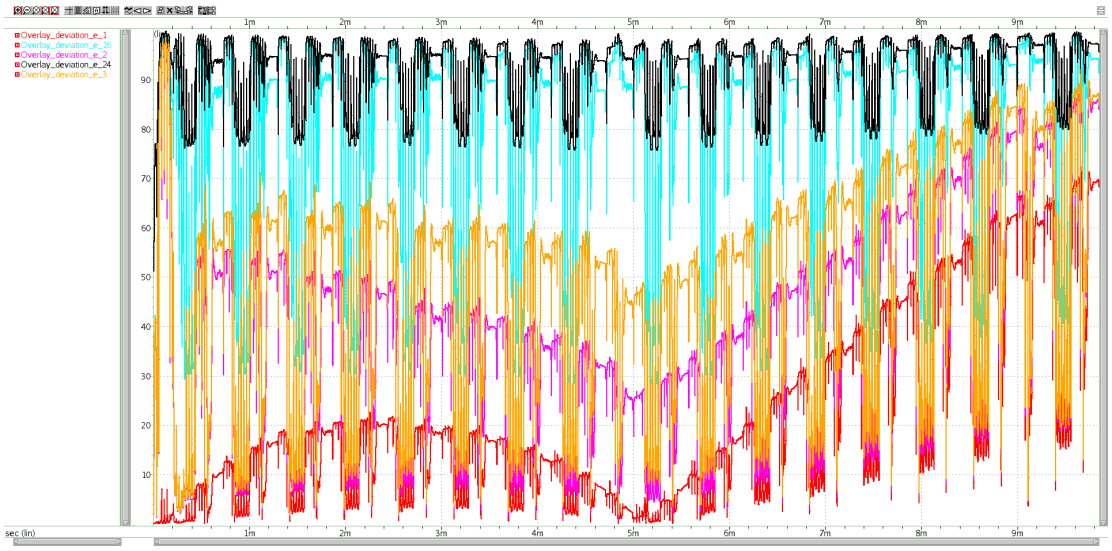


Figure 5.27: Temporal monitor output showing selected deviations and their impact to the radius of the selected AAF

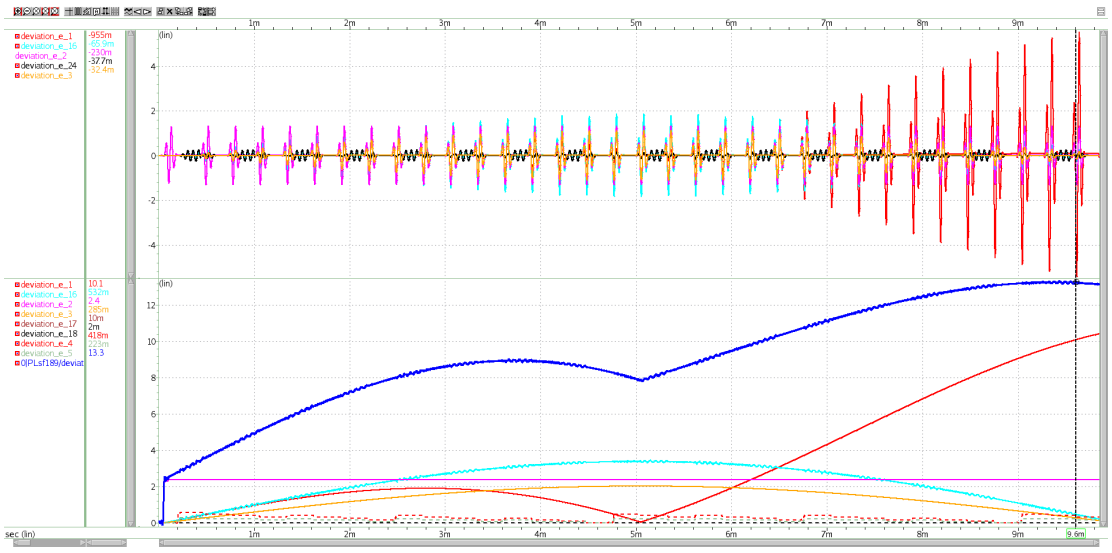


Figure 5.28: Temporal monitor showing the absolute values of selected deviations for verification that the end of a transmission is critical.

green. The path is specified semi-automatically by tracing the propagation of symbols $\epsilon_{1,2,3,16}$. $\epsilon_{1,2,3}$ have their cause in the PL generator block and ϵ_{16} in the PL channel model.

For structural tracing results, the following color code, mapping segments on the specified path to lines in plots, is defined: cyan=s5, pink=s4, orange=s3, black=s2, blue=s1. Figure 5.30 shows a combined temporal and structural view for the selected partial deviations associated with the symbols $\epsilon_{1,2,3,16}$. The time period for the plots is refined to a sampling window of the second half of the transmission 5 *ms* to 10 *ms*. The plot illustrates the structural propagation (color-coded within each plot) of the selected deviations (each individual plot). The partial deviations of the power line gets propagated through the system at nearly no attenuation. Strictly speaking, the pink, orange, black and blue lines are lying upon each other. But this is already clear because the

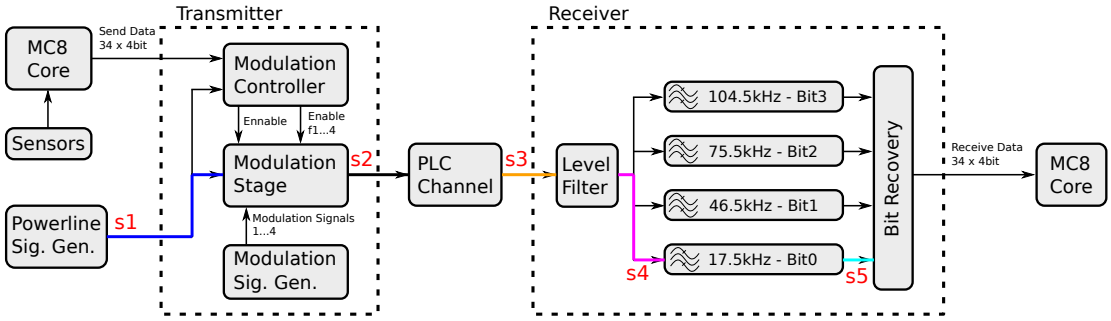


Figure 5.29: The colored trace highlights the path specified for structural analysis

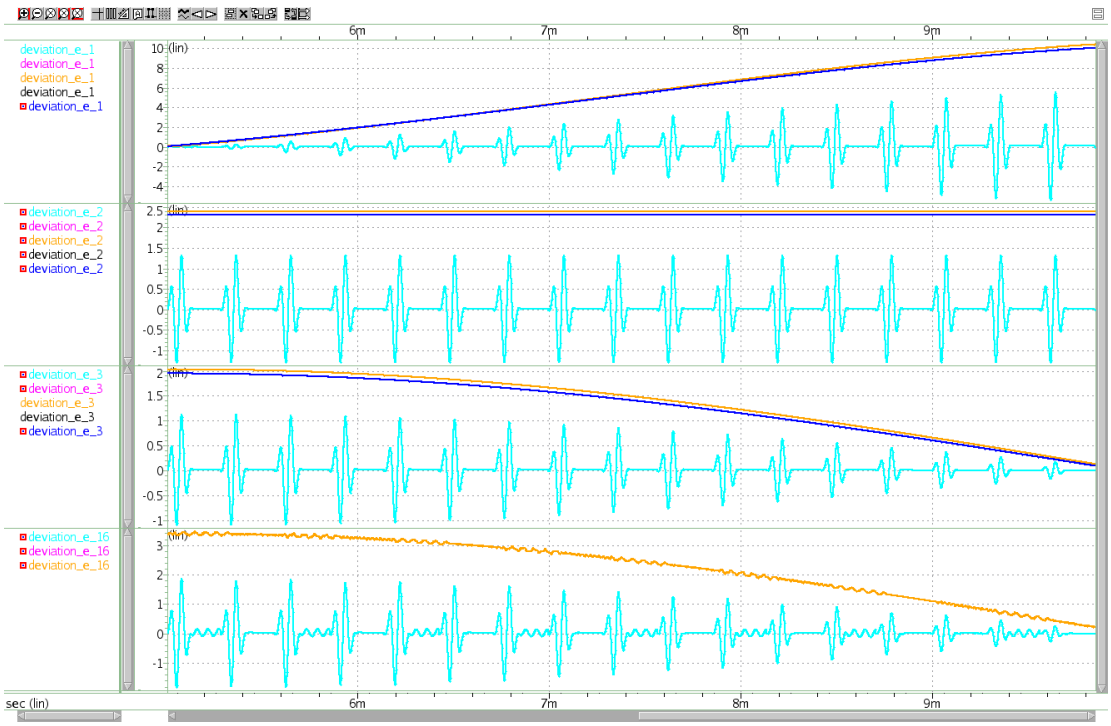


Figure 5.30: Combined temporal and structural trace view illustrating the propagation of selected uncertainties.

50 Hz payload signal of the power line is more or less not damped between the transmitter and receiver block (see frequency characteristic of the channel Figure 5.14). Thus, also PL variations are propagated to the level filter input. The level filter itself just subtracts a constant 230 $V/50\,Hz$ signal, which consequences no attenuation in frequency, unbalance and magnitude variations. As a result, the central value at the output of the level filter is quite small (0 in the exact case) but the variations, especially associated with $\epsilon_{1,2}$ (frequency and unbalance), having a value approximately 2.5 and 10 are directly propagated to the level filter's output. Thus, the pink line (partial deviations on s4=output of the level filter) is approximately equal to the PL deviations at the input (orange line = s3).

To verify this results, which identifies the level filter as the critical block of the model, the SDR (Signal to Deviation Ratio) as proposed in Subsection 4.3.3 and given in equation 4.3 is evaluated. The color-code for the mapping (lines to path segments) is given above. The cyan and pink plot indicating the two signal segments at the output of the level filter has a dominating low

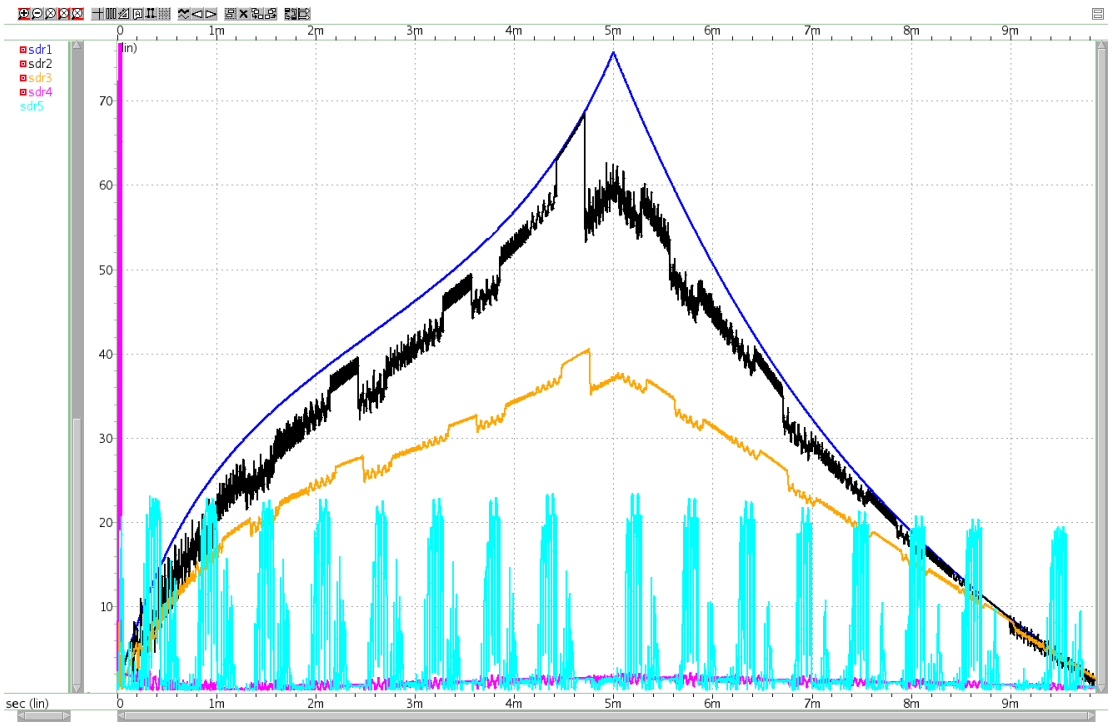


Figure 5.31: SDR (Signal to Deviation Ratio) analyis of the PLC transmission system.

(compared to the others) SDR value. This correlates with the fact that the radius of the AAF is large compared to the AAF's central value.

Discussion of analysis results

As a result of this step by step refined analysis process, I identified the following critical parts of the modeled PLC transmission system:

- In the time domain, due to the specified frequency variation and PL unbalance variation, at the end of the transmission (positive half-cycle of the PL signal) the radius of the PL signal has its maximum. This results in worse bit transmission performance for the slots at the end of a cycle.
- In the structural domain, the identified dominating variations of PL frequency, unbalance, magnitude and the channel attenuation gets directly propagated to the output of the level filter. This has the consequence that the corresponding SDR quality measure is very low. Strictly speaking, the radius of the AAF is large compared to the center value.

All the presented simulation and analysis results are evaluated from 4 simulation runs and 3 analysis refinement steps:

1. Initial AAF based simulation of the system to identify the uncertain behavior of the bit recovery process.

2. Inserting relative monitors at the output of the receiver filters. They show that the integrated impact, and identify dominating uncertainties in the filter's outputs.

3. Specification of an analysis path through the system and inserting temporal and structural monitors for guided deviation hot-spot detection. Results show, that the end of the transmission is critical caused by unattenuated propagation of PL variation via the critical level filter in the receiver.

4. To verify the results also specific SDR monitors are inserted in each segment of the specified analysis path.

This underlines the performance of applied analysis features used for this example.

Each single simulation run (strictly speaking the last simulation run including the maximum number of monitors) took a computation time of approximately $2\,h$ and $8\,min$. The simulation has been executed on machine with the following specifications: CPU: Intel Core I7-6700K 4.00 GHz 4Core, RAM: 4x 16GB DDR4 2400 RAM, OS: Ubuntu 16.04 LTS. This is a significant speedup compared to the multi-run simulation ($7\,h$ and $35\,min$) described above.

Potential first steps towards optimization of the PLC system

As discussed, the critical block in the system is the included level filter. A first optimization is to replace the static behavior by an adaptive filter behavior. The adaptiveness is defined as continuous modification of the $230\,V/50\,Hz$ signal used for subtraction, to reduce the impact of frequency, unbalance and magnitude variations in the filter's output. Strictly speaking, the current PL magnitude, frequency and unbalance parameters are sensed at the input of the filter, and the subtraction signal is correspondingly adapted for the following transmission cycles. In an AAF modeling perspective this means that the following correlations are introduced: $\epsilon_2 = \epsilon_{21} = \epsilon_{23}$ and $\epsilon_3 = \epsilon_{22}$. The frequency variation in the level filter is not considered at all in the specified model. As a first optimization step, the mentioned correlations are modeled. Simulation results show that the bit error rate of the model including the modified daptive filter is 0.082. Compared to the previous results of 0.316, this is a significant performance improvement.

5.4 Adaptive Cruise Control (ACC)

In this example, I present modeling, simulation, and analysis of Adaptive Cruise Control (ACC). ACC is an automotive driver-assistance feature implemented in modern upper-class cars. In modeling perspective ACC is represented as a high-level CPS model including a description of the system's environment.

Automotive feature modeling

ACC has in principle two functional modes as indicated in Figure 5.32. First, it acts as a cruise controller keeping the speed of the car at a constant user-selectable level. This is called Cruise Control (CC) mode. The driver may release the gas pedal completely, and the speed is managed by a controller. The selected cruise speed is held as long as the track in front of the car is clear from

any obstacles (the left scenario in Figure 5.32). In real implementations, this selected target speed may range from 40 to 200 kilometers per hour, depending on the installed sensor equipment and implemented version of the ACC feature. Second, if a vehicle in front appears the distance to it is monitored continuously, and the own speed value is adapted to satisfy a minimum target distance (see Figure 5.32, right scenario). Therefore, the engine torque is reduced, eventually, an other gear is selected, and/or brake input is applied to decelerate the vehicle. This is called Distance Control (DC) mode. Here potential safety considerations come into play. A rear-end collision has to be avoided under full impact of the automatic controller system. In real implementations for critical (emergency) situations manual brake input of the driver is still required [WS15]. Thus, ACC is a driver assistance system and not an entirely dependable "auto pilot". If the car in front accelerates the target distance will be held as long as the speed is not exceeding the configured CC speed.

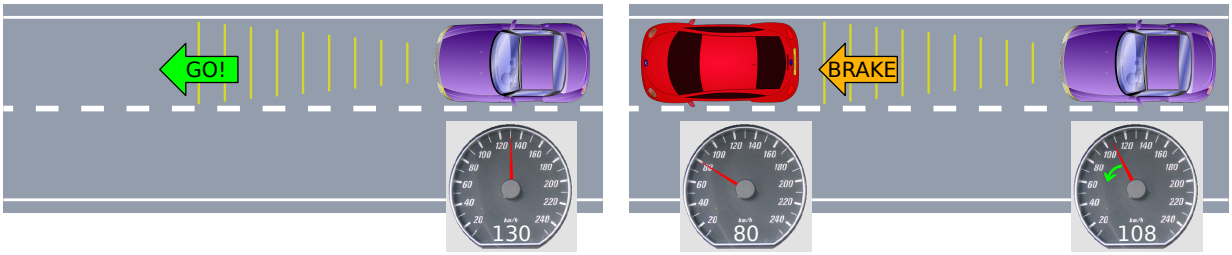


Figure 5.32: Specific ACC scenarios, illustrating Cruise Control (CC) and Distance Control (DC) modes of the feature.

ACC is already well understood as a single feature from an engineering viewpoint [WS15]. I study it from the perspective of a composite feature. Cruise Control (CC) and Distance Control (DC) operate independently while their individual speed requests have to be coordinated. For coordination of these subfeatures features various methodologies based on Mediator-Coordinator patterns, hill-climbing, objective functions, etc. can be found in the literature [LRK17, EDK13]. For this work, I follow the engineering approved coordination scheme strictly selecting the minimum speed request of both subfeatures [WS15]. For the rest of this example car A denotes the own vehicle and car B the vehicle driving in front of A.

In general, I follow a modeling scheme shown in Figure 5.33. The idea is to partition the model into three layers: a physical layer a request layer and a feature(and coordination) layer. The upper layer defines a model of the physical environment describing kinematics of the vehicle. Values as distance, speed, acceleration, and torque are arranged as a chain of values which interact under their corresponding physical dependencies. These dependencies can be specified in three different ways. First, a static characteristic curve (derived from measurements) may be defined (e. g. the influence of an applied engine torque value to the acceleration of the vehicle). Second, values may depend each other by Newton's laws. For example, the acceleration is defined as the first derivation of the speed. Third, dependencies can be described by more complex behavior (e. g. defined as an FSM model). The green circles in the physical model mark potential value conflict where dependencies have to be resolved. For the described ACC model these resolutions are solved by a coordinator function at the feature level. In general, the physical model relies on a strict balance of physical quantities. Interfacing points to the lower feature layer are given by a set of requests and by measurement data. Single features (F) can place their requests directly at the physical chain or as mentioned coordinate their requests by specific coordination functionality (C). Monitored physical values are fed back to the feature's inputs. This results in a closed data path acting as a closed loop controller system.

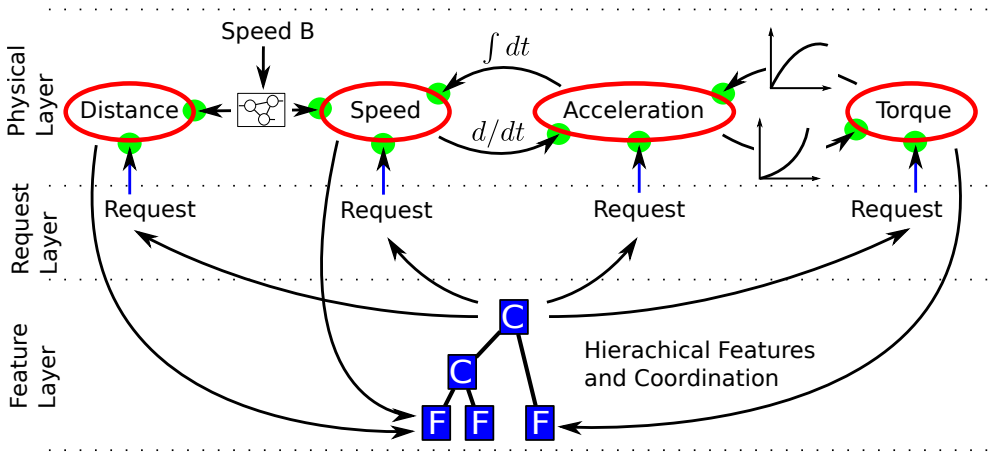


Figure 5.33: A generalized modeling approach for automotive features. The physical model (top layer) is connected to the implemented features via a request layer. Feature requests have to be coordinated that the physical quantities are in balance and corresponding dependencies are resolved.

ACC model and uncertainty documentation

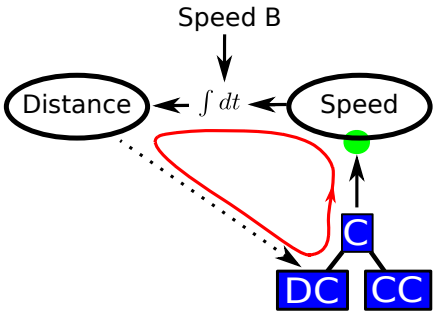


Figure 5.34: Physical and fearure layer of the ACC model deived form the model illustrated in Figure 5.33.

Based on the modeling scheme of features shown in Figure 5.33 I define the model for this ACC example (see Figure 5.34). Physical quantities are reduced to the speed of the cars and the distance between them. The unidirectional dependency is given by Newton's law $d = \int_0^t (v_B - v_A)dt + d_{init}$, where d stands for the distance between car A and car B, v_A and v_B for the corresponding speed values and d_{init} for the initial distance. The dependency between speed and distance has an uncertain characteristic which is defined by the unpredictable speed of car B. In contrast to the prior described examples, this model has a closed loop datapath indicated as a red trace in Figure 5.34. The ACC feature is represented by the CC and DC subfeatures and a coordinator. The coordinated speed value output is a speed request at the physical part of the model.

The block diagram of the implemented ACC model (and then used for simulation and analysis) is illustrated in Figure 5.35. The CC subfeature delivers a constant speed value at its output. DC computes a speed request according to the following controller formula:

$$SpeedRequest_{DC} = (diatance_{sensed} - distance_{target}) \cdot k + speed_{CarB} \tag{5.20}$$

where k is a proportional factor multiplied by the difference of sensed distance and the selected target distance. As described CC and DC may request a speed level according to their individual

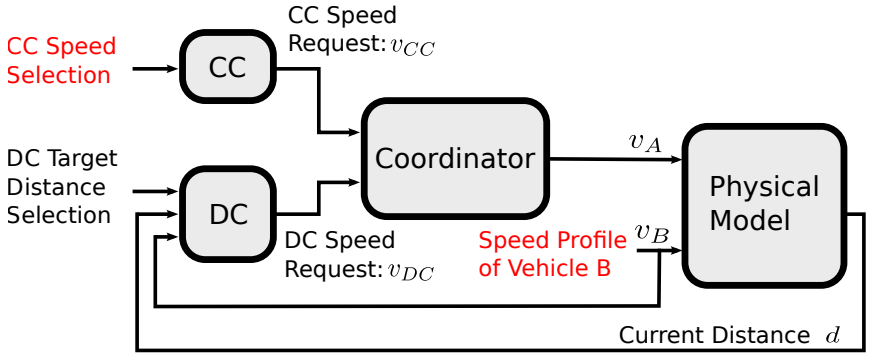


Figure 5.35: Block diagram of the modeled ACC system

independent behavior. The coordinator block forwards the lower speed request of both subfeatures at each point in time. For the driving behavior of car B, I assume a static speed profile as illustrated by the red trace in Figure 5.36. The distance between car A and car B is computed using the physical formula described in the last paragraph. A representative part of the distance output of the model (sensed distance) is fed back to the input, which means that in the following inducted uncertainty will be continuously propagated over time. The ACC system in particular includes besides linear blocks a discontinuity given by the coordinator behavior switching between the system's CC and DC modes.

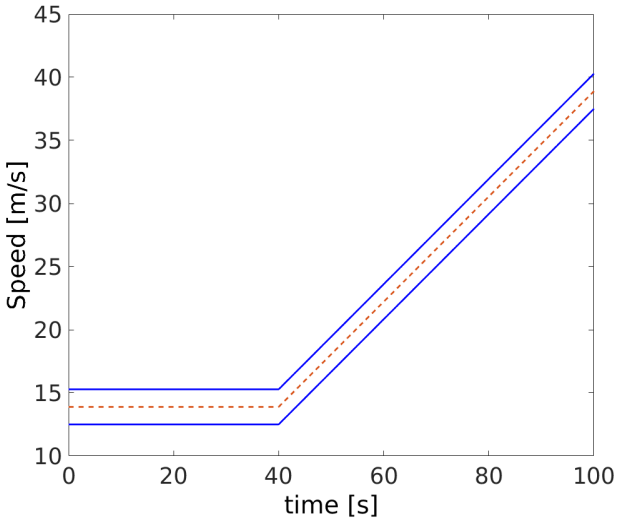


Figure 5.36: Driving profile of vehicle B represented as an AA signal

Much as in [LRK17], only the typical scenario is studied where car A is approaching first $v_{CC} > v_B$ until the selected target distance is reached, while car B does not change its speed according to the left part of Figure 5.36. Car A keeps the distance until car B accelerates (according to the right part of the figure) $v_B > v_{ACC}$. Car A may also increase its speed now, but it must not exceed the selected CC cruise speed.

As a set of parameter deviations representing uncertainty I consider two deviation causes (indicated in red in Figure 5.35). These are a deviating CC speed selection of car A and an unpredictable speed profile offset of car B (see blue boundary value traces in Figure 5.36).

Table 5.7 contains the parameter setup of the model for semi-symbolic simulation, including

deviations and their corresponding values. According to these values, initial AAFs $\hat{v}_A$ and $\hat{v}_B$ can be stated as $\hat{v}_A = v_{CC} + v_{CC,1} \cdot \epsilon_1 = 100 + 2.5 \cdot \epsilon_1$ and $\hat{v}_B = v_{B,init} + v_{B,2} \cdot \epsilon_2 = 50 + 5 \cdot \epsilon_2$. The uncertainty causes represented by the ϵ_1 and ϵ_2 symbols directly influence the coordinator output speed request and the calculated distance between the vehicles. Symbols are propagated through the system and fed back to the input of the Distance Control block. Thus, partial deviations of a time-step also influence future simulation time-steps. The full model is calculated with a time-step period of 10 ms.

Table 5.7: Parameter setup and deviation documentation of the ACC example

Parameter	Description	Value
ACC configuration of car A		
d_{init}	Initial distance between vehicle A and B	$40\,m$
v_{CC}	Cruise control speed setup	$100\,km/h$ $(= 27.78\,m/s)$
$v_{CC,1}$	Deviation of the cruise control speed setup	$\pm 2.5\,km/h$, $(= \pm 0.694\,m/s)$
d_{target}	Distance control target distance setup	$15\,m$
k	Distance controller proportional factor	1
Speed profile of car B		
$v_{B,init}$	Initial speed of vehicle B	$50\,km/h$, $(= 13.9\,m/s)$
$t_{accelerate}$	Time period before vehicle B accelerates	$40\,s$
a_B	Acceleration rate of vehicle B	$0.417\,m/s^2$
$v_{B,2}$	Speed deviation of vehicle B	$\pm 5\,km/h$, $(= \pm 1.39\,m/s)$

Coordinator block

For the coordination of CC and DC, I use a minimum coordinator block, which forwards their lower speed request value as applied at its inputs. In the *SESYD* framework, no minimum operator for AAFs is available yet, $min(\hat{v}_{CC}, \hat{v}_{DC})$, where $\hat{v}_{CC}$ and $\hat{v}_{DC}$ are AAFs.

Figure 5.37 shows the individual CC and DC speed requests when car A is in CC mode and approaching car B. The center value of $\hat{v}_{CC}$ is constant at 100 km/h, superimposed by a 2.5 km/h deviation (symbol ϵ_1). The center value and the specified deviation range are illustrated by the brown dashed and the red lines, respectively. $\hat{v}_{DC}$ is decreasing according to the implemented control equation 5.20. Again, the brown dashed line indicates the center value and the red line the deviation caused by the symbol ϵ_1. Additionally, $\hat{v}_{DC}$ includes the deviation symbol ϵ_2 representing the speed deviation of car B, where the blue lines show the accumulated deviation of both ϵ_1 and ϵ_2. Due to the closed-loop model structure, $\hat{v}_{CC}$ and $\hat{v}_{CC}$ are correlated through including the same symbol ϵ_1.

The rightmost simulation point in Figure 5.37 indicates the first point in time when the ranges of the speed requests from CC and DC overlap, $\hat{v}_{CC} \diamond \hat{v}_{DC} = true$. In principle, starting from this point in time, due to the unpredictable values of the ϵ symbols, I cannot determine with certainty whether the CC or the DC speed request has a lower value. Such a characteristic is called an uncertain minimum.

At a closer look, due to the given symbol correlation between $\hat{v}_{CC}$ and $\hat{v}_{DC}$, it may even happen that the point in time when the minimum of requested speeds becomes uncertain is additionally shifted in time. Hence, in a first step I handle the given ϵ_1-correlation correctly by calculating the AA subtraction of $\hat{v}_{DC}$ and $\hat{v}_{CC}$. The result of $\hat{v}_z = \hat{v}_{DC} - \hat{v}_{CC}$ is plotted in Figure 5.38. The

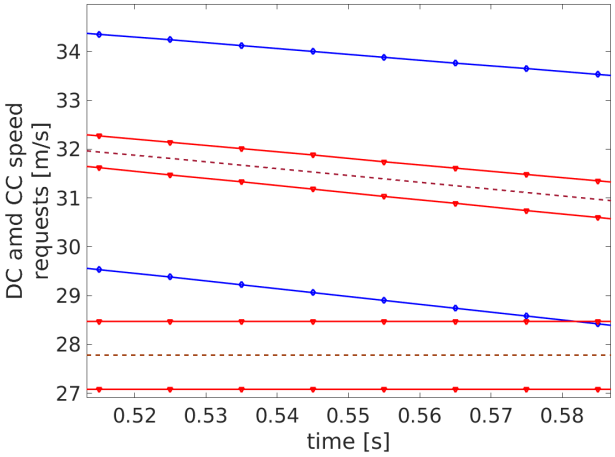


Figure 5.37: CC and DC speed requests and their deviations over time, until their ranges start to overlap

minimum becomes uncertain if the value 0 is within the calculated interval range of $\hat{v}_{DC} - \hat{v}_{CC}$. This can be expressed by using the functions *value* and *range*: $0 \in range(value(\hat{v_z}))$ [Rad16]. The arrows in the figure indicate such a situation. If and when this happens, cannot be uniquely evaluated if the ACC feature is in CC or DC mode, due to the unknown values of ϵ_1 and ϵ_2.

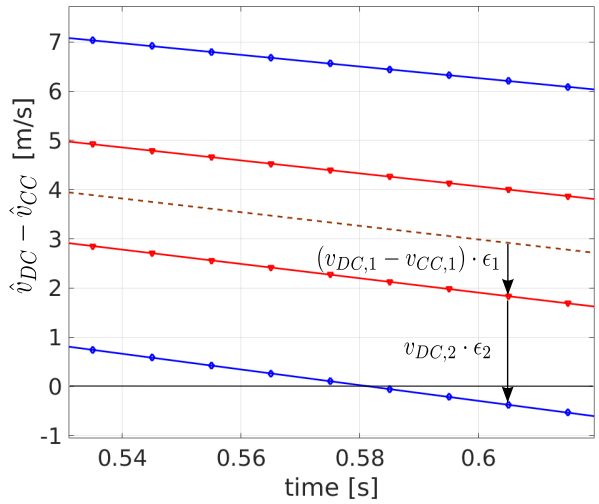


Figure 5.38: Plot of $\hat{v}_{DC} - \hat{v}_{CC}$ and a case where 0 is included within its interval range

Minimum approximation using intervals

As a second step, for points in time when the minimum is uncertain, approximations are calculated. The first minimum approximation is based on Interval Arithmetic calculations, which do not take correlations between CC and DC requests into account. Interval Arithmetic calculations, in general, result in more pessimistic solutions than Affine Arithmetic calculations.

As illustrated in Figure 5.39, I have to distinguish four cases of how the ranges of CC and DC may relate to each other. The ranges on the right plotted in red are the results for each case, to be used for our approximation of the minimum form $min(\hat{v}_{CC}, \hat{v}_{DC})$.

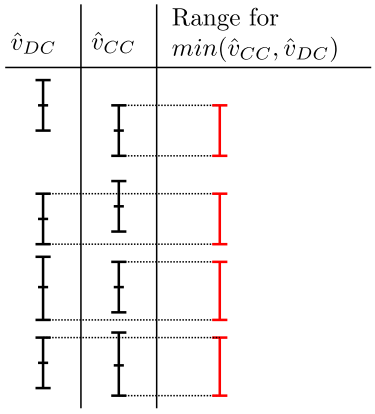


Figure 5.39: Cases of overlapping CC and DC ranges and the corresponding range for the minimum approximation

To create an output AAF for the minimum operator based on that, I handle the approximation representation in the same way as for non-linear operators such as multiplication, division, etc. The affine part of the result is defined either by the CC or the DC request, depending on their central values. However, the output of the minimum operator is an AAF caused by the DC or CC request inputs, respectively. We also add an extra symbol and a partial deviation α_i, which widens this range if the minimum is uncertain (see Subsection 3.2.1) For the resulting AAF, I also calculate an approximation central value β_i, which is located in the middle of the approximated output range.

As a result and to sum up, my approximation for the minimum operator includes a two-step process:

- First I evaluate an AAF for $\hat{v}_{DC} - \hat{v}_{CC}$. If 0 is included within its interval range, we know that the minimum value is uncertain and can be caused by a CC or a DC request, respectively. The system can be in CC or DC mode depending on the unknown value of ϵ symbols.
- As a second step, if the minimum is uncertain, I calculate an approximation form covering the worst case. The approximation interval is evaluated by uncorrelated IA considerations including different cases of overlapping situations. Finally, the approximation is stored within an AAF object setting corresponding α and β values.

Minimum evaluation with Affine Arithmetic Decision Diagrams

Since the minimum approximation with intervals does not utilize any of the information contained in an AAF in addition to the intervals per se, I also use a more precise approximation with AADDs. AADDs are a central concept of the thesis published by Carna Radojicic [Rad16]. Results using AADDs are a more precise reference for the evaluation of the presented interval based approximation implemented in the *SESYD* framework.

As described above, the minimum is uncertain if we cannot determine if CC or DC has a lower value, i.e., if the value 0 is within the calculated interval range of $\hat{v}_{DC} - \hat{v}_{CC}$. The basic idea for handling this uncertainty is to use a binary decision diagram for minimum evaluation. At each

point in time during the simulation when this uncertainty exists, an internal vertex is created in such a diagram with two successor edges, labeled *true* and $false$, depending on whether the condition $(\hat{v}_{DC} - \hat{v}_{CC} \geq 0)$ is fulfilled. The value *true* signifies the CC case, i.e., in this part of the tree the assumption is that the speed request of CC is used. For the value $false$, however, the speed request of DC is taken. Since $\hat{v}_{DC}$ and $\hat{v}_{CC}$ are AAFs, this distinction depends on the values of ϵ_1 and ϵ_2. A solver is used for finding those partitions of epsilon ranges for which the condition is fulfilled or not. Through recursive application for the next point in time of the simulation, a tree structure is created. For each vertex below in the tree, only these value ranges are valid, of course. For each path, there is a terminal vertex, whenever there is no minimum uncertainty any more. It is assigned the AAF of $\hat{v}_{DC}$ or $\hat{v}_{CC}$, respectively.

Simulation and results

In the simulation run, the speed profile of car B as shown in Figure 5.36 is used. Car A is approaching first, while $v_A = v_{CC} > v_B$, until the selected target distance is reached. This is illustrated in the left part of Figures 5.40, 5.41 and 5.42. Then, as long as the speed of car B does not change, $v_A = v_{DC} = v_B$, the DC block holds the given target distance. This time period is cut out in the figures. Once car B accelerates resulting in $v_B > v_{CC}$, car A may also increase its speed, but it must not exceed the selected CC cruise speed (see the right part of Figures 5.40, 5.41 and 5.42). Hence, there is a discrete change involved again, this time from $v_A = v_{DC}$ to $v_A = v_{CC}$.

For calculating the minimum of AAFs, I used the approximations based on interval considerations and AADDs as defined above. The corresponding lines in the figures are shown in magenta and green, respectively. For showing the subintervals for the interval approximations, I use red lines for ϵ_1 and blue lines for accumulated ϵ_1 and ϵ_2 deviations.

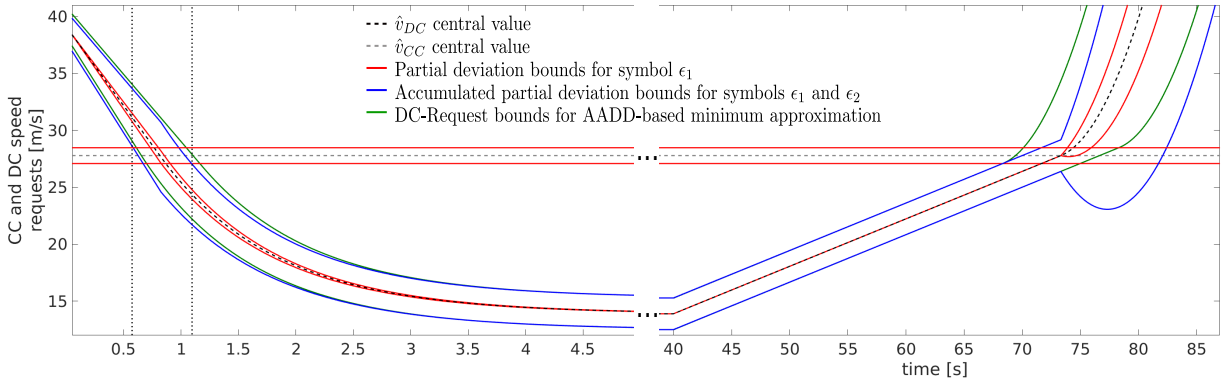


Figure 5.40: CC and DC speed requests

Figure 5.40 illustrates the speed requests of CC and DC, respectively. CC has a constant speed request of 27.78 m/s and a deviation range of 0.694 m/s indicated by the red lines representing the bounds of the range. The DC feature has a decreasing speed characteristic caused by the decreasing distance between the vehicles. The blue lines show the accumulated ϵ_1 and ϵ_2 deviation causes of the DC speed request for AAF using the interval-based approximation for the coordinator block. First, ACC is in CC mode, and the upper bound of the CC request is clearly smaller than the lower bound of the DC request, until the ranges start to overlap, which is marked by the two vertical cursors (strictly speaking, the $\hat{v}_{DC} - \hat{v}_{CC}$ interval range includes 0).

Starting at this point in time, the minimum operation of the coordinator block is approximated. The corresponding output of the coordinator is shown in Figure 5.41. I evaluate two approximations for the minimum based on intervals and AADDs, respectively. The slight differences between the green and the magenta line in the left part of the figure are caused by the behavior of the interval-based approximation, which is still switching between DC and CC modes based on the center value of the AAF.

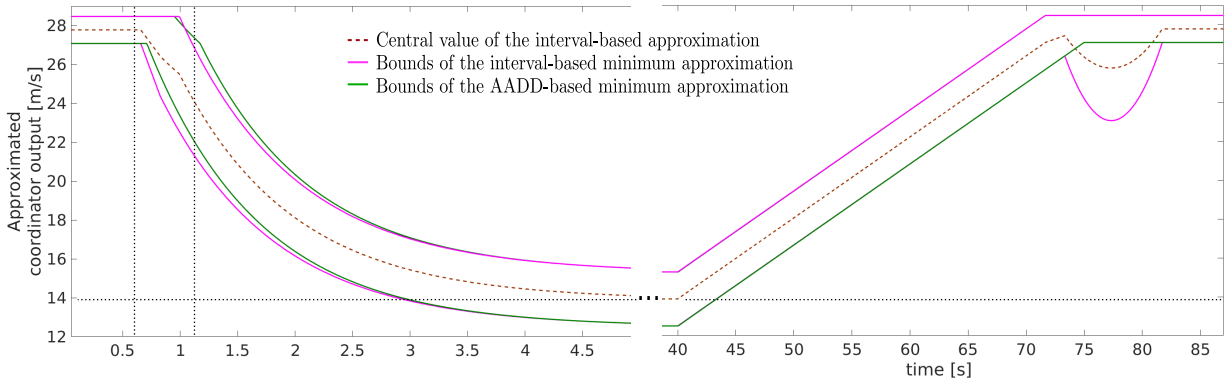


Figure 5.41: Approximated minimum coordinator output

As a result of the coordinator, I plot in Figure 5.41 the boundaries of the approximation forms for interval approximation (in magenta), and for AADD-based approximation (in green). Due to the strict requirement of an AAF that the center value has to be in the middle of the spanned range, for the interval-based approximation a center value is additionally calculated (stored as a β value within an AAF object). It is illustrated by the brown dashed line.

Once the minimum is no longer uncertain (the DC request is for certain smaller than the CC request), ACC is in DC mode. The speed request changes its characteristic to an exponentially shaped curve, asymptotically approaching a value of 13.9 m/s (see the horizontal cursor), which is the speed of car B. The exponential shape is caused by the control behavior of the DC feature (see equation 5.20) and the closed-loop system structure.

The right parts of the Figures illustrate the acceleration scenario of car B. I focus on the effect occurring at the point in time when car B is becoming faster than the CC speed of car A ($t > 68\ s$). Much as in the approach scenario, the DC and CC request ranges start to overlap and the coordinator calculates an approximation for the minimum (more precisely in two different ways). For the interval-based approximation ACC switches between CC and DC mode, depending on the minimum of the center values of their requests. Thus, ACC switches back to CC mode, but DC is still independently calculating speed requests in parallel. Due to the constant output of the coordinator (CC speed), for DC the feedback loop is broken. Thus, the deviation bounds are increasing. The coordinator is still approximating the minimum by taking the interval minimum according to the cases shown in Figure 5.39. This explains the partially parabolic shape of the approximation curve of the minimum values. Hence, for the described scenario where car B is speeding up and exceeding the configured CC speed of car A, the minimum approximation has a pronounced pessimistic range. The approach using AADDs (green line in Figure 5.41) does not have this effect and is, therefore, much more precise.

Figure 5.42 shows the distance between car A and car B over time. The corresponding center value (brown dashed line) starts at the defined initial distance of 40 m. Since there is no initial deviation set for the distance, the first simulation point is an AAF having a radius of 0. In the

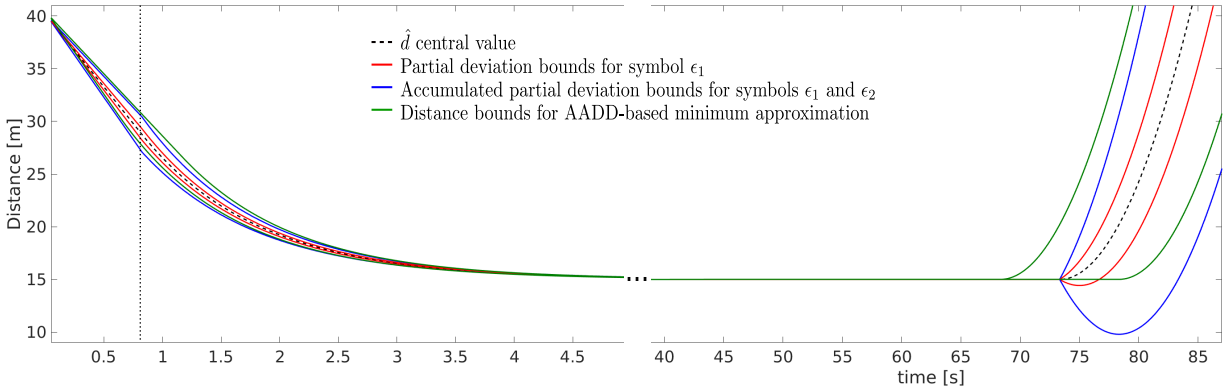


Figure 5.42: Distance between car A and B

following, the deviations of the CC speed (based on the symbol ϵ_1) and the speed of car B (based on the symbol ϵ_2), continuously influence the range of the distance, until ACC is in DC mode and the deviation causes (partial deviation values) are constant. Due to the physical dependency between the speeds of car A and B, also the partial deviation values get integrated over time. The influence of ϵ_1 is illustrated by the partial deviation bounds as shown in red boundary lines, the influence of ϵ_2 by the blue boundary lines, where more precisely the red part has to be taken out, since the blue lines show the accumulated partial deviations.

A closer look at the subinterval ranges for ϵ_1 and ϵ_2 reveals that the partial deviation with the symbol ϵ_2 representing the speed deviation of car B has a higher impact on the deviation range of the distance. This can be seen in Figure 5.42, since for all points in time (except 0) the differences between the red lines and the brown center line is smaller than the differences between the blue and the red lines. If and when ACC is in DC mode (illustrated on the right from the courser in Figure 5.42), the radius of the distance is decreasing. This is caused by the closed-loop system structure including the DC controller block, which continuously attenuates partial deviation values. The asymptotic trend of the distance line approaches the user-specified target distance of 15 m.

The right part of Figure 5.42 illustrates the distance range for the situation where car B exceeds the selected CC speed of car A. Based on the approximation effect for the interval-based approach described in Figure 5.41, there is also an impact on the distance boundaries. The parabolic shape of the curve of the approximated speed output of the coordinator is propagated to the distance. Thus the displayed lines for AAF are also a pessimistic approximation of the distance between car A and B. This effect is strongly decreased if the time span of overlapping CC and DC requests is shorter (e. g. caused by a larger acceleration of car B).

The green lines in Figure 5.42 illustrate the bounds of the minimum using AADDs. Also for the distance, the effect of the uncertain mode switch from DC to CC ($t > 68\,s$) for the interval-based minimum approximation is not present when using AADDs.

The more precise approximation of the minimum operation using AADDs requires an increased simulation runtime, as compared to using the intervals only. The simulation of the presented driving scenario using an AADD-based minimum took about 4 minutes (on a standard PC), compared to 5 seconds using the interval-based approach.

ACC feature analysis

For the analysis of the ACC feature, I created temporal and structural traces as defined in Sections 4.5 and 4.6 First I started with two temporal traces for the partial deviations included in the distance AAF. The initial objective of the analysis is to identify the point in time when the deviation of the distance is a maximum. The temporal traces of the partial deviations included in the distance value is illustrated in Figure 5.43. The red line is the trace for the partial deviation associated with ϵ_1 (deviation of CC speed), the blue line for ϵ_2 (speed deviation of car B).

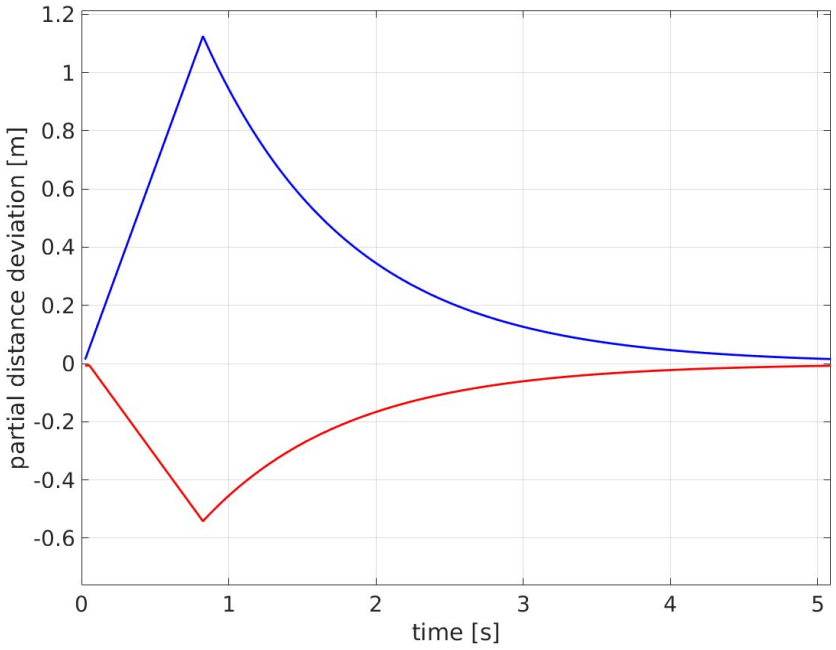


Figure 5.43: Temporal trace of the distance

Based on these trace results, the following qualitative statements for the analyzed ACC feature and the given scenario can be made:

- The partial deviation value of the distance for ϵ_1 is negative due to the dependencies given by the physical model. Consider a positive value of ϵ_1, causing a speed value $v_A > v_{CC}$. This results in a faster approach towards car B, i.e., a smaller distance, as compared to $v_A = v_{CC}$.

- The absolute impact of the deviation associated with ϵ_2 on the distance is higher than the one associated with ϵ_1, by a factor of 2 (equal to the ratio of the given speed deviations).

- The critical system state (in terms of distance deviation) occurs if and when the ACC feature is in CC mode. Due to the lack of any distance control function, deviations get continuously integrated, which also widens the total range (see the left part of Figure 5.43 until ACC switches to DC mode at $t = 0.825\,s$). This is also the case after the mode change from DC back to CC at $t = 73.3\,s$. The slope of the linear part indicates for the initial approach of car A an increase of the deviation of the distance between the vehicles by $2\,m$ within one second.

- The first mode change from CC to DC at $t = 0.825\,s$ is picked as a critical point in time for our simulation and selected for further analysis using a structural trace (analysis refinement).

To refine the analysis procedure I continued the verification process with creating structural traces. For the ACC example, I created two structural traces for ϵ_1 and ϵ_2, respectively, which is specified in the equation below. The point in time for recording the trace is set to $t_s = 0.825\,s$, where ACC switches from CC to DC mode in the given driving scenario. The traces are defined according to the following equation, and include all data values of the system ($\hat{v}_{CC}, \hat{v}_{DC}, \hat{v}_A, \hat{v}_B, \hat{d}$):

$$\begin{aligned}\tau_{structural1,2}(\epsilon_{1,2}) =< & \, select(\hat{v}_{CC}(t_s), \epsilon_{1,2}), \\ & select(\hat{v}_{DC}(t_s), \epsilon_{1,2}), select(\hat{v}_A(t_s), \epsilon_{1,2}), \\ & select(\hat{v}_B(t_s), \epsilon_{1,2}), select(\hat{d}(t_s), \epsilon_{1,2}) >\end{aligned}$$

Evaluating these formulas with the corresponding values results in the following numerical sequences:

$$\begin{aligned}\tau_{structural1}(\epsilon_1) &=< 0.694, 0.534, 0.649, 0, 0.534 > \\ \tau_{structural2}(\epsilon_2) &=< 0, 0.2, 0, 1.39, 1.11 >\end{aligned}$$

Defined traces may include partial deviation values from different physical quantities (speeds and a distance in this case). To facilitate the comparison of the partial deviation values included here, we divide them by the radius (computed by the *rad* operator) of the corresponding AAF (and multiply by 100) to get a percentage value.

$$\begin{aligned}&\tau_{structural1\%}(\epsilon_1) = \\ &\left\langle \frac{69.4}{rad(\hat{v}_{CC})}, \frac{53.4}{rad(\hat{v}_{DC})}, \frac{64.9}{rad(\hat{v}_A)}, \frac{0}{rad(\hat{v}_B)}, \frac{53.4}{rad(\hat{d})} \right\rangle = \\ &< 100\%, 72.75\%, 100\%, 0\%, 32.48\% > \\ &\tau_{structural2\%}(\epsilon_2) = \\ &\left\langle \frac{0}{rad(\hat{v}_{CC})}, \frac{20}{rad(\hat{v}_{DC})}, \frac{0}{rad(\hat{v}_A)}, \frac{139}{rad(\hat{v}_B)}, \frac{111}{rad(\hat{d})} \right\rangle = \\ &< 0\%, 27.25\%, 0\%, 100\%, 67.52\% >\end{aligned}$$

As a result, we can directly interpret these numbers for tracing the development of distance deviations in the system structure. At the given point in time for this structural analysis, ACC is in CC mode, hence the distance contribution of ϵ_1, 32.48%, results from $\hat{v}_{CC}$. The v_B deviation impact of 67.52% on the distance range is only propagated in the physical model, which is based on Newton's laws. Also the speed profile deviation of car B is given by the simulated scenario and cannot, in principle, be improved by the ACC feature. Unfortunately, the major part of the integrating characteristic of the distance (partial deviation associated with ϵ_2) while ACC is in CC mode cannot be optimized. Thus, the only possibility to decrease the integration effect is to reduce the deviation of the CC speed parameter.

For further evaluation of the ACC feature, I abstracted the model to a *qualitative minimalist model* used for formal system analysis as described in Section 4.9 [RLK16]. In general, I followed the modeling scheme presented in Figure 5.33. This includes a physical model, models for the CC and DC features and a feature coordinator. But, in this case, each of this submodels is represented

as a finite state machine. States of the FSM models represent a classification of values (in the form of an interval) for corresponding physical quantities. Thus, I do not longer use scalar values for physical quantities. For example the DC and CC features may request a *low*, *medium* or *high* speed request. The corresponding speed values of the claassification are not relevant for model checking the behavior of the system in general. Transitions between states are given by conditions depending on individual sub-FSM states (e. g. the coordinator FSM changes its state to *low* if CC is in state *high* and DC is in state *low* (minmum behavior)). Also, the distance value between car A and car B is calssified (*CC Guided*,*DC Target*,*Too Close* and *Collision*) representing states physical model FSM. For example, Figure 5.44 illustrates a selected part of the complete model showing the state machines representing the behavior of car B (upper FSM) and the FSM for the distance between the cars. An important benefit of formally checking the ACC feature is that in the perspective of model checking we are not forced to define the driving profile of car B. After an initial speed setup car B may change its states (*low*,*medium*,*high*) individually not defined by a specific speed profile as given in Figure 5.36 above.

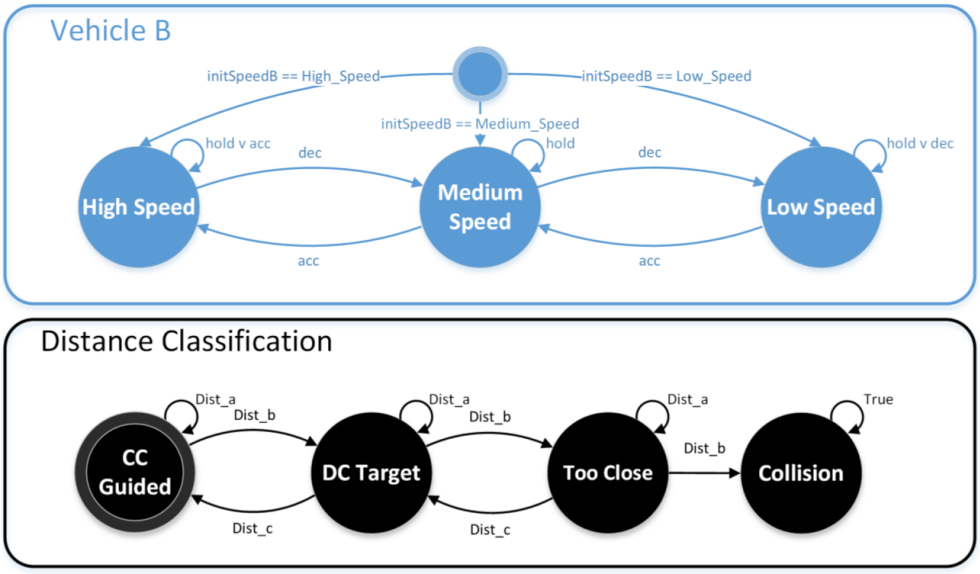


Figure 5.44: FSM models (a selected part of the full ACC model) representing the posiible behaviors of car B and the distance classification.

These states, strictly speaking, the *Collision* state, is used for definition of the following safety property:

$$AG(state\_phy\_dist \neq COLLISION) = \neg EF(state\_phy\_dist = COLLISION) \tag{5.21}$$

In the perspective of formal verification, model checking one of the given formulas results in a state reachability problem which can be in natural language formulated as: Starting from the initial state, there exists no computation path where the *Collision* state of the Distance Classification FSM is reached. For experiments, I used the model checking tool NuSMV [3] and in [RLK16] I show some results of the runs. We discuss the abstraction of the model in detail and address the requirement of the *Too Close* in the FSM representing a model for the distance.

6 Conclusion and Outlook

In this chapter, I conclude this thesis. First, I give a resume of evaluated results and contributions to the research field. This includes advantages of the re-implemented *SESYD* framework and the associated enhanced system analysis processes. Second, I identify and discuss selected future research topics which conduct this thesis and potentially further advance the context of range based system analysis and verification processes.

6.1 Summary and Discussion of Results

In this summary, I reiterate the importance of range based modeling- simulation- and analysis techniques during the design phase. For highly innovative applications, such as those firmly integrated into the areas such as Industry 4.0 and IoT, the requirements in functional density, reliability, flexibility, and robustness are becoming more and more demanding. In general, parameter deviations receive an increasing impact on a system's performance and in worst-case considerations, the system will not operate at all. Besides internal variabilities, also external uncertainties have to be taken into account. cyber-physical systems highly interact with their physical sourrounding and can not rely on predictable static environmental conditions. They have to satisfy minimum performance requirements also at their specified operational bounds.

Thus, the goal of range based processes, as described in this thesis, is to increase the insight into the system during its design phase, with particular attention to potential occurring uncertainty. Such uncertainties (internal or external) entail parameter variations which propagate through the system structure. It is a significant added value for design processes and subsequent system optimization if this propagation of deviations can be tracked and better understood.

This thesis presents a set of concepts (also integrated into a software framework) for modeling, simulation, and analysis of systems with deviant inputs.

Modeling:
The primary approach in this work is to use semi-symbolic techniques (Affine Arithmetic forms) for the representation of uncertainties. This has the consequence that a set of new enhanced analysis features is enabled. I use the original form of AAFs (Affine Arithmetic forms) which do not include any probabilistic information or enhanced structures for representing control flow uncertainty. However, I motivate a new object-oriented approach for mapping AAFs to object data structures (and their implementation in the C ++ programming language). This has the

consequence that the usage of AAFs gets more flexible and extensible. New features, such as user-specific approximations, symbol management, etc. which were not possible in prior AAF framework implementations, are provided.

Simulation:
Concerning simulation procedures, I mainly use already published state of the art algorithms for AAF and IA operations. For multi-run methods, I present a module in the framework which can be interfaced with the SystemC simulation core to enable multi-run functionalities. Multi-run approaches are not provided by the original SystemC implementation till now. However, these simulation methods have to be adapted for being integrated into the mentioned object-oriented framework. Besides arithmetic operations, I developed various new features for plotting and post-processing simulation results with the goal that the designed AAF-, interval-, multi-run variable objects can be presented in a standard waveform viewer.

System analysis:
The developed system analysis features and their application are beside the new framework architecture the main scientific contribution of this thesis. For analysis, I focus on processes which benefit significantly (or are even possible) by representing parameter deviations using AAFs. Strictly speaking, the proposed algorithms rely on the ability to monitor the propagation of deviations through the system (deviation tracing) which is comprehensively enabled by AAFs. To get to the point, appropriate analysis procedures are:

- Ratio analysis and deviation metrics, which allows the identification of correlations as well as an impact assessment of specific uncertainty causes.
- Assertion driven system analysis is used for checking assertion statements, including range based expressions, during the run of a simulation.
- Temporal tracing of deviations is used for enhanced monitoring of deviations in the time domain.
- Structural deviation tracing analyzes deviation impact on specified paths through the system structure.
- Deviation hot-spot detection combines temporal and structural tracing features to identify critical events in the temporal domain, and critical functional blocks in the structural domain.
- Frequency domain analysis is mainly featured by the FFT post-processing functions of the used waveform viewer. However, their application requires the mentioned enhanced simulation result plotting functions.
- Formal system analysis, based on model checking algorithms is used to check potential violations of safety-critical properties caused by parameter deviations.

In general, I propose to use the given analysis features in a planned and refined approach. First, an initial analysis objective is set, and potential verification goals are defined. Second, a proper analysis method is applied and refined in a way that the specified goal orientation is followed step by step until sufficient insight for subsequent optimization is reached.

The developed modeling, simulation and analysis concepts are presented by using four demonstration examples. These examples are models which are defined on different levels of abstraction.

- On a silicon level, a timing model of an inverter stage based on the included transistors is presented. The impact of production uncertainties in transistor dimensions to switching propagation delays is analyzed. The propagation delay model is subsequently used for modeling a ring oscillator. The example shows how silicon level uncertainties propagate to high level consequences in the inverter's switching performance.

- A communication transmitter system is modeled by a two-stage an Amplitude-Shift Keying modulator. For the including non-linear multiplication operations various approximations are calculated. On the first side, this example shows the user specific approximation management implemented in the *SESYD* framework and on the other side, the characteristics of approximation algorithms are highlighted.

- On a module level of abstraction, a power-line communication system is shown. Parameter deviations in the model are given by uncertain power-line characteristics (e. g. frequency, unbalance) and uncertain parameters in transmitter and receiver modules (e. g. temperature, voltage offset). For deviation analysis the propagation of uncertainties is evaluated in temporal and structural domain. Thus, critical events during a transmission cycle and critical functional blocks in the system architecture are identified.

- In a system level perspective an abstract behavioral model of ACC (Adaptive Cruise Control) is presented. ACC is a closed loop controller which has in addition two operational modes. These modes represent behavioral discontinuities which significantly influence the behavioral bounds of the system.

The presented examples underline the added value of the analysis concepts presented in the thesis. Analysis results, which are mainly enabled by enhanced deviation tracing features of the implemented framework, enhance the knowledge about inner systems characteristics and enable goal oriented subsequent design optimization. Thus, this contribution in AAF analysis approaches is a potential further steps towards the integration of AAF modeling, simulation and analysis methodologies into industrial design tools.

6.2 Future Research Work

For future research I highlight selected technological directions for pottential subsequent activities:

Range based in-field simulation
The main idea of this approach is to deploy simulation and analysis applications to computer systems which are distributed in the production field. In contrast to runtime verification approaches as introduced in Section 4.10 at this method, the full simulation core (including analysis functions) and a model of the system is executed on the hardware target. Strictly speaking, a processor integrated within a system runs in addition to the control software a simulation of its application. Possible application fields are: process control, energy and IT applications, ambient electronics, safety-critical applications, etc. In practice, today often very powerful and possibly multi-core processor units (e. g. ARM architecture) are used. Simulation runs for example can be executed during standby phases or on unused processor cores in parallel to the control application software. Goals and advances of this parallel simulation approach are On-the-fly analysis and optimization of the considered system, tuning of the control algorithm by enhanced parameter

estimation and decision-making, detection of potential fault states in advance, increase of the long-time period efficiency, etc.

For the proposed simulation running on the target system also range based (AAF) approaches are promising. This allows more precise forecasts and behavioral estimations if disturbances of input parameters, environmental changes, unforeseeable events, strictly speaking, dynamic deviations, are present. As a result, the efficiency of the system can be potentially increased based on evaluated output ranges. As an initial proof of concept, I compiled the SystemC simulator and the proposed *SESYD* framework for the ARM platform and deployed the simulation to a Raspberry Pi development system. Next steps are to create a specific demo application and show the added value of the approach.

Machine learning guided system analysis
This approach addresses the idea to extend the *SESYD* framework by machine learning algorithms. The created knowledge base is used for analysis processes in a way to identify potentially interesting anomalies in the model's signal outputs and for goal-oriented automatic guidance through the verification process. Training data for appropriate algorithms may be evaluated from sequential simulation-based regression testing or created manually by supervised learning. Generated knowledge databases may also be used in a cross-project way, and can be shared as a cloud knowledge database for the verification of open source system designs. The concept of range based simulation may bring an added value in combination with the proposed learning approach. For example potential causes (e. g. critical AAF symbols) responsible for worst-case behavior can be evaluated first by deviation tracing methods. Second, based on regression runs for the propagation of the deviation cause a corresponding propagation template can be learned. This knowledge may subsequently assist a verification engineer in marking further critical paths through the system.

Blockchain technology for trusted verification
Blockchain technologies will potentially become the revolutionary innovation influencing all kind of applications in the next years [PM17]. The main idea is to use blockchain technology for system verification. In professional system development processes the results of verification are already stored in databases. Maybe these verification data, including the setup, system version, configurations, the results, etc. are added to a blockchain structure. This has the potential benefit that performed verification steps, and their results cannot be manipulated. Admittedly, that's a very vague idea but eventually, these technologies may influence future verification tools.

Literature

[AAC12] Arendt, Paul D. ; Apley, Daniel W. ; Chen, Wei: Quantification of Model Uncertainty: Calibration, Model Discrepancy, and Identifiability. In: *Journal of Mechanical Design* (2012), October, Nr. 100908, S. 1–12

[ABC+11] Arden, Wolfgang ; Brillouet, Michel ; Cogez, Patrick ; Graef, Mart ; Huizing, Bert ; Mahnkopf, Reinhard ; Pelka, Joachim ; Pfeiffer, Jens-Uwe ; Rouzaud, Andre ; Tartagni, Marco ; Van Hoof, Chris ; Wagner, Joachim: Towards a "More-than-Moore" roadmap. In: *Report from the CATRENE Scientific Committee*, 2011

[BGG+09] Barke, E. ; Grabowski, D. ; Graeb, H. ; Hedrich, L. ; Heinen, S. ; Popp, R. ; Steinhorst, S. ; Wang, Yifan: Formal approaches to analog circuit verification. In: *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, 2009. – ISSN 1530–1591, S. 724–729

[Bie09] Chapter Bounded Model Checking In: Biere, Armin: *Handbook of Satisfiability.* Bd. 184. ios press, 2009, S. 457–481

[BKBD06] Bausch, J. ; Kistner, T. ; Babic, M. ; Dostert, K.: Characteristics of Indoor Power Line Channels in the Frequency Range 50 - 500 kHz. In: *2006 IEEE International Symposium on Power Line Communications and Its Applications*, 2006, S. 86–91

[BZ10] Boule, M. ; Zilic, Z.: *Generating Hardware Assertion Checkers - For Hardware Verification, Emulation, Post-Fabrication Debugging and On-line Monitoring.* Bd. 1. SPringer Science+ Business Media, 2010

[Cad10] Cadence Design Systems Inc.: *Cadence PSpice A/D and PSpice advanced analysis.* 2010

[Cel91] Cellier, Francois E.: *Defining Uncertainty - A Conceptual Basis for Uncertainty Management in Model-Based Continuous system modeling.* New York : Springer-Verlag, 1991. – ISBN 0387975020

[CGM07] Cappe, O. ; Godsill, S.J. ; Moulines, E.: An Overview of Existing Methods and Recent Advances in Sequential Monte Carlo. In: *Proceedings of the IEEE* 95 (2007), May, Nr. 5, S. 899–924. – ISSN 0018–9219

[CL07] Chan, Hock P. ; Lai, Tze L.: Efficient importance sampling for Monte Carlo evaluation of exceedance probabilities. In: *Ann. Appl. Probab.* 17 (2007), 04, Nr. 2, S. 440–473

[Cou17] Courtland, R.: Moore's law's next step: 10 nanometers. In: *IEEE Spectrum* 54 (2017), January, Nr. 1, S. 52–53. – ISSN 0018–9235

[CS93] Comba, J. ; Stolfi, J.: Affine atithmetic and its applications to computer graph-

ics. In: *Anais do VI Simposio Brasileiro de Computacao Grafica e Processamento de Imagens (SIBGRAPI)*, 1993

[Die03] Dieschmid, Hans J.: *Höhere Mathematik . Differential- und Integralrechnung.* Manz Verlag, 2003. – ISBN 3–7068–1390–4

[DIN11] DIN: Standard EN 50160 - Voltage Characteristics in Public Distribution Systems. DIN, July 2011. – Scientific Report. – ISSN EN 50160:2011–02

[DLM07] Daumas, Marc ; Lester, David R. ; Muñoz, César A.: Verified Real Number Calculations: A Library for Interval Arithmetic. In: *CoRR* abs/0708.3721 (2007)

[Dor10] Dorey, Fredrick J.: Statistics in Brief: Confidence Intervals. In: *Springer- In Brief* (2010)

[DPAC17] Dutta, T. ; Pahwa, G. ; Agarwal, A. ; Chauhan, Y. S.: Impact of Process Variations on Negative Capacitance FinFET Devices and Circuits. In: *IEEE Electron Device Letters* PP (2017), Nr. 99, S. 1–1. – ISSN 0741–3106

[Dre10] Chapter Introduction In: Drechsler, Rolf: *Advanced Formal Verification.* Bd. 1. Kluwer Academic Publishers, 2010

[Dre17] Drechsler, Rolf: *Formal System Verification: State-of the-Art and Future Trends.* Springer, 2017

[Dro09] Drosg, Manfred: *Dealing with Uncertainties - A Guide to Error Analysis.* 2. Springer Dordecht Heidelberg London New York, 2009

[Eco06] Economakos, George: Behavioral synthesis with SystemC and PSL assertions for interface specification. In: *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on* IEEE, 2006, S. 4–pp

[EDK13] Ertl, D. ; Dominka, S. ; Kaindl, H.: Using a Mediator to Handle Undesired Feature Interaction of Automated Driving. In: *2013 IEEE International Conference on Systems, Man, and Cybernetics*, 2013. – ISSN 1062–922X, S. 4555–4560

[Eib04] Eibl, Christian J. *gSysC - Visualisierung von SystemC-Projekten.* 2004

[FDC13] Farid, Moshgelani ; Dhamin, Al-Khalili ; Come, Rozon: Ultra-Low Leakage Arithmetic Circuits Using Symmetric and Asymmetric FinFETs. In: *Journal of Electrical and Computer Engineering* 2013 (2013), August

[FS] FH-Stralsund: *Die CMOS Logik.* – Lecture notes

[Fuj99] Fujimoto, Richard M.: *Parallel and Distribution Simulation Systems.* 1st. New York, NY, USA : John Wiley & Sons, Inc., 1999. – ISBN 0471183830

[GGB06] Grabowski, D. ; Grimm, C. ; Barke, E.: Semi-symbolic modeling and simulation of circuits and systems. In: *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, 2006, S. 4 pp.–986

[GHW04] Grimm, Ch. ; Heupke, W. ; Waldschmidt, K.: Semi-Symbolic Modeling and Analysis of Noise in Heterogeneous Systems. In: *Froum on Specification and Design Languages (FDL 04)*, 2004

[GLMS10] Groetker, Thorsten ; Liao, Stan ; Martin, Grant ; Swan, Stuart: *System Design with SystemC.* Springer Publishing Company, Incorporated, 2010

[GOB08] Grabowski, D. ; Olbrich, M. ; Barke, E.: Analog circuit simulation using range arithmetics. In: *2008 Asia and South Pacific Design Automation Conference*, 2008. – ISSN 2153–6961, S. 762–767

[GR17] Grimm, Christoph ; Rathmair, Michael: Dealing with Uncertainties in Analog/Mixed-Signal Systems: Invited. In: *Proceedings of the 54th Annual Design Automation Conference 2017.* New York, NY, USA : ACM, 2017 (DAC '17). – ISBN 978–1–4503–4927–7, S. 35:1–35:6

[Gra09] Grabowski, Darius: *Gebietsarithmetische Verfahren zur Simulation analoger*

Schaltungen mit Parameterunsicherheiten, Gottfried Wilhelm Leibniz Universität Hannover, PhD Thesis, 2009

[GY17] Gustafson, John ; Yonemoto, Isaac: Beating Floating Point at its Own Game: Posit Arithmetic. In: *Supercomputing Frontiers and Innovations* 4 (2017), Nr. 2. – ISSN 2313–8734

[HA08] Hung, H. ; Adzic, V.: Monte Carlo Simulation of Device Variations and Mismatch in Analog Integrated Circuits. In: *Proceedingd of National Conference On Ungraduate Research*, 2008

[Her06] Herfort, W.: Skriptum für Mathematik 3 für Elektrotechnik - Wintersemester, TU-Wien, 2006

[Her16] Herdt, Vladimir: *Complete Symbolic Simulation of SystemC Models: Efficient Formal Verification of Finite Non-Terminating Programs*. Springer, 2016

[HG17] He, J. ; Guan, X.: Uncertainty Sensitivity Analysis for Reliability Problems with Parametric Distributions. In: *IEEE Transactions on Reliability* 66 (2017), Sept, Nr. 3, S. 712–721. – ISSN 0018–9529

[HJVE01] Hickey, T. ; Ju, Q. ; Van Emden, M. H.: Interval Arithmetic: From Principles to Implementation. In: *J. ACM* 48 (2001), September, Nr. 5, S. 1038–1068. – ISSN 0004–5411

[HPM17] Hasan, K. N. ; Preece, R. ; Milanovic, J. V.: Priority Ranking of Critical Uncertainties Affecting Small-Disturbance Stability Using Sensitivity Analysis Techniques. In: *IEEE Transactions on Power Systems* 32 (2017), July, Nr. 4, S. 2629–2639. – ISSN 0885–8950

[HWC08] Haase, Joachim ; Wolf, Susann ; Clauss, Christoph: *Monte Carlo Simulation with Modelica*. Frauenhofer-Institute for Integrated Circuits, Design Automation Division, 2008. – 601–604 S

[JHY11] Jie, Han ; Huaiyan, Chen ; Yun, Cao: Uncertainty evaluation using Monte Carlo method with MATLAB. In: *Electronic Measurement Instruments (ICEMI), 2011 10th International Conference on* Bd. 2, 2011, S. 282–286

[Jon03] Jones, C. B.: The early search for tractable ways of reasoning about programs. In: *IEEE Annals of the History of Computing* 25 (2003), April, Nr. 2, S. 26–49. – ISSN 1058–6180

[KD07] Kiureghian, Armen D. ; Ditlevsen, Ove: Aleatory or epistemic? Does it matter? In: *Special Workshop on Risk Acceptance and Risk Communication*, 2007

[KG13] Karimi, Kavian ; Gary, Atkinson: What the Internet of Things (IoT) Needs to Become a Reality. In: *White Paper of Freescale and ARM*, 2013

[KH05] Kang, Yong H. ; Hong, Songcheol: A simple Flash memory cell model for transient circuit simulation. In: *IEEE Electron Device Letters* 26 (2005), Aug, Nr. 8, S. 563–565. – ISSN 0741–3106

[KKK11] Kokil, Priyanka ; Kar, Haranath ; Kandanvli, V: Stability analysis of linear discrete-time systems with interval delay: a delay-partitioning approach. In: *ISRN Applied Mathematics* 2011 (2011)

[Kle08] Kleijnen, J. P. C.: Design Of Experiments: Overview. In: *2008 Winter Simulation Conference*, 2008. – ISSN 0891–7736, S. 479–488

[KO01] Kennedy, Marc C. ; O'Hagan, Anthony: Bayesian calibration of computer models. In: *Journal of the Royal Statistical Society* (2001), Nr. 3, S. 425–464

[Kol93] Kolev, L. V.: *Interval Methods for Circuit Analysis*. Bd. 1. Advanced Series on Circuits and Systems, 1993

[KPM04] Kolev, Lubomir ; Petrakieva, Simona ; Mladenov, Valeri: Interval criterion for stability analysis of discrete-time neural networks with partial state saturation nonlinearities. In: *Neural Network Applications in Electrical Engineering, 2004. NEUREL 2004. 2004 7th Seminar on* IEEE, 2004, S. 11–16

[Kra06] Kraemer, Walter: Generalized Intervals and the Dependency Problem. In: *PAMM* 6 (2006), Nr. 1, S. 683–684. – ISSN 1617–7061

[Kro10] Kropf, Thomas: *Introduction to Formal Hardware Verification.* Bd. 1. Springer-Verlag Berlin Heidelberg, 2010

[Lam05] Lam, William K.: *Hardware Design Verification: Simulation and Formal Method-Based Approaches (Prentice Hall Modern Semiconductor Design Series).* Upper Saddle River, NJ, USA : Prentice Hall PTR, 2005. – ISBN 0131433474

[LGLK14] Liu, S. ; Gou, B. ; Li, H. ; Kavasseri, R.: Power-Line Communication Channel Characteristics Under Transient Model. In: *IEEE Transactions on Power Delivery* 29 (2014), Aug, Nr. 4, S. 1701–1708. – ISSN 0885–8977

[LL15] Li, M. ; Lin, H. J.: Design and Implementation of Smart Home Control Systems Based on Wireless Sensor Networks and Power Line Communications. In: *IEEE Transactions on Industrial Electronics* 62 (2015), July, Nr. 7, S. 4430–4442. – ISSN 0278–0046

[LRK17] Luckeneder, Christoph ; Rathmair, Michael ; Kaindl, Hermann: Investigating and Coordinating Safety-critical Feature Interactions in Automotive Systems Using Simulation. In: *Proceedings of the 50th Hawaii International Conference on System Sciences*, 2017

[LYX+10] Liu, B. ; Yin, J. ; Xiao, Y. ; Cao, L. ; Yu, P. S.: Exploiting Local Data Uncertainty to Boost Global Outlier Detection. In: *2010 IEEE International Conference on Data Mining*, 2010. – ISSN 1550–4786, S. 304–313

[MBDG12] McConaghy, Trent ; Breen, Kristopher ; Dyck, Jeffrey ; Gupta, Amit: *Variation-aware design of custom integrated circuits: a hands-on field guide.* Springer Science & Business Media, 2012

[Mit98] *MIT - CMOS Inverter: Propagation delay.* 1998. – Lecture notes

[MKC09] Moore, Ramon E. ; Kearfott, R. B. ; Cloud, Michael J.: *Introduction to Interval Analysis.* Society for Industrial and Applied Mathematics, 2009

[MMC11] Martynyuk, Anatolii A. ; Martynyuk-Chernienko, Yu A.: *Uncertain dynamical systems: stability and motion control.* CRC Press, 2011

[MMMAY01] Miura-Mattausch, M. ; Mattausch, H.J. ; Arora, N.D. ; Yang, C.Y.: MOSFET modeling gets physical. In: *Circuits and Devices Magazine, IEEE* 17 (2001), Nov, Nr. 6, S. 29–36. – ISSN 8755–3996

[MT06] Messine, Frédéric ; Touhami, Ahmed: A general reliable quadratic form: An extension of affine arithmetic. In: *Reliable Computing* 12 (2006), Nr. 3, S. 171–192

[Neu04] de Neufville, Richard: Uncertainty Management for Engineering Systems Planning and Design. In: *Engineering Systems Sympossium* (2004), March, S. 1–18

[OBJ05] Ostrovsky-Berman, Yaron ; Joskowicz, Leo: Uncertainty envelopes. In: *EuroCG*, 2005, S. 175–178

[OSG11] Ou, Jiong ; Schupfer, Florian ; Grimm, Christoph: System Level Robust Communication System Design using Extended SystemC AMS Building Block Library. In: *Austrochip 2011 - Workshop on Microelectronics*, 2011, S. 39–44

[OSY+16] Okamoto, D. ; Suzuki, Y. ; Yashiki, K. ; Hagihara, Y. ; Tokushima, M. ; Fujikata, J. ; Kurihara, M. ; Tsuchida, J. ; Nedachi, T. ; Inasaka, J. ; Kurata,

K.: A 25-Gb/s; 5 mm2 Chip-Scale Silicon-Photonic Receiver Integrated With 28-nm CMOS Transimpedance Amplifier. In: *Journal of Lightwave Technology* 34 (2016), June, Nr. 12, S. 2988–2995. – ISSN 0733–8724

[PKB07] Perraud, J.-M. ; Kuczera, G. ; Bridgart, R.J.: Towards a Software Architecture to Facilitate Multiple Runs of Simulation Models. In: *MODSIM 2007 International Congress on Modelling and Simulation. Modelling and Simulation Society of Australia and New Zealand*, 2007, S. 846–852

[PLV10] Pennachin, Cassio ; Looks, Moshe ; de Vasconcelos, Joao A.: Robust symbolic regression with affine arithmetic. In: Pelikan, Martin (Publisher) ; Branke, Jürgen (Publisher): *GECCO*, ACM, 2010. – ISBN 978–1–4503–0072–5, S. 917–924

[PM17] Peck, M. E. ; Moore, S. K.: The blossoming of the blockchain. In: *IEEE Spectrum* 54 (2017), October, Nr. 10, S. 24–25. – ISSN 0018–9235

[Pop98] Popova, Evgenija D.: On the Efficiency of Interval Multiplication Algorithms. In: *Real Numbers and Computers* 3 (1998), S. 117–131

[PSL10] IEEE Standard for Property Specification Language (PSL). In: *IEEE Std 1850-2010 (Revision of IEEE Std 1850-2005)* (2010), April, S. 1–182

[PSN04] Portal, J. M. ; Saillet, B. ; Nee, D.: Flash memory cell diagnosis: high level model. In: *Proceedings. 2004 IEEE Computational Systems Bioinformatics Conference*, 2004, S. 100–104

[PTVF02] Press, William H. ; Teukolsky, Saul A. ; Vetterling, William T. ; Flannery, Brian P.: *Numerical Recipes in C++ (2Nd Ed.): The Art of Scientific Computing*. New York, NY, USA : Cambridge University Press, 2002. – ISBN 0–521–75033–4

[QS03] Quaglia, F. ; Santoro, A.: Nonblocking checkpointing for optimistic parallel simulation: description and an implementation. In: *IEEE Transactions on Parallel and Distributed Systems* 14 (2003), June, Nr. 6, S. 593–610. – ISSN 1045–9219

[Qui13] Quinn, Patrick J.: FPGA based silicon innovations exploiting "More than Moore" technology. In: *Conference on Ph.D. Research in Microelectronics and Electronics, PRIME*, 2013

[Rad16] Radojicic, Carna: *Symbolic Simulation of Mixed-Signal Systems with Extended Affine Arithmetic*, PhD Thesis, 2016. – 115 S

[Rat15] Rathmair, M.: Range Based Analysis of Inner Systems Characteristics. In: *Design, Automation and Test in Europe (DATE) 2015, EDAA PhD Forum*, 2015

[Ray08] Raychaudhuri, Samik: Introduction to Monte Carlo simulation. In: Mason, Scott J. (Publisher) ; Hill, Raymond R. (Publisher) ; Mönch, Lars (Publisher) ; Rose, Oliver (Publisher) ; Jefferson, Thomas (Publisher) ; Fowler, John W. (Publisher): *Winter Simulation Conference*, WSC, 2008. – ISBN 978–1–4244–2708–6, S. 91–100

[RGJR17] Radojicic, Carna ; Grimm, Christoph ; Jantsch, Axel ; Rathmair, Michael: Towards Verification of Uncertain Cyber-Physical Systems. In: *Proceedings 3rd International Workshop on Symbolic and Numerical Methods for Reachability Analysis, SNR@ETAPS 2017, Uppsala, Sweden, 22nd April 2017.*, 2017, S. 1–17

[RHKP15] Rathmair, Michael ; Hoch, Ralph ; Kaindl, Hermann ; Popp, Roman: Consistently Formalizing a Business Process and its Properties for Verification: A Case Study. In: *The Practice of Enterprise Modeling - 8th IFIP WG 8.1. Working Conference, PoEM 2015, Valencia, Spain, November 10-12, 2015, Proceedings*, 2015, S. 126–140

[RK15] Rump, Siegfried M. ; Kashiwagi, Masahide: Implementation and improvements of affine arithmetic. In: *Nonlinear Theory and Its Applications, IEICE* 6 (2015),

Nr. 3, S. 341–359

[RL+18] Rathmair, Michael ; Luckeneder, Christoph ; ; Kaindl, Hermann ; Carna, Radojicic: Semi-symbolic Simulation and Analysis of Deviation Propagation of Feature Coordination in Cyber-physical Systems. In: *Proceedings of the 51th Hawaii International Conference on System Sciences*, 2018

[RLK16] Rathmair, M. ; Luckeneder, C. ; Kaindl, H.: Minimalist Qualitative Models for Model Checking Cyber-Physical Feature Coordination. In: *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, 2016. – ISSN 1530–1362, S. 233–240

[RS13] Rathmair, M. ; Schupfer, F.: Hardware Trojan detection by specifying malicious circuit properties. In: *2013 IEEE 4th International Conference on Electronics Information and Emergency Communication*, 2013, S. 317–320

[RS15] Rathmair, M. ; Schupfer, F.: Dealing with Uncertainties in Electronic Systems Simulation. In: *VSS - VIENNA young SCIENTISTS SYMPOSIUM 2015*, 2015, S. 34–35

[RS16] Rathmair, M. ; Schupfer, F.: Metrics for Formal Property Checking Against Undesired Circuit Behavior in Embedded Systems. In: *ITG-Fachbericht Analog 2016*, 2016, S. 64–69

[RSGR14] Rathmair, M. ; Schupfer, F. ; Grimm, C. ; Radojicic, C.: Simulationsgestützte Analyse der inneren Eigenschaften von Mixed-Signal Systemen (german). In: *Proceedings of 16. Workshop - Analog Circuits 2014*, 2014, S. 10–11

[RSK14] Rathmair, M. ; Schupfer, F. ; Krieg, C.: Applied formal methods for hardware Trojan detection. In: *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, 2014, S. 169–172

[RSRG12a] Radojicic, C. ; Schupfer, F. ; Rathmair, M. ; Grimm, C.: Assertion-based verification of signal processing systems with affine arithmetic. In: *Specification and Design Languages (FDL), 2012 Forum on*, 2012. – ISSN 1636–9874, S. 20–26

[RSRG12b] Rathmair, M. ; Schupfer, F. ; Radojicic, C. ; Grimm, C.: Extended framework for system simulation with affine arithmetic. In: *Specification and Design Languages (FDL), 2012 Forum on*, 2012. – ISSN 1636–9874, S. 168–175

[Rub81] Rubinstein, Reuven Y.: *Simulation and the Monte Carlo Method*. New York : John Wiley&Sons, Inc., 1981

[SB15] Santos, K.R.M. d. ; Beck, A.T.: A benchmark study on intelligent sampling techniques in Monte Carlo simulation. In: *Latin American Journal of Solids and Structures* 12 (2015), 08, S. 624 – 648. – ISSN 1679–7825

[Sch13] Schupfer, Florian: *Design Refinement of Communication Systems Applying Range Based Simulations*, Vienna University of Technology, PhD Thesis, 2013

[SCS+00] Saltelli, Andrea ; Chan, Karen ; Scott, E M. [u. a.]: *Sensitivity analysis*. Bd. 1. Wiley New York, 2000

[Sem05] Semmlow, J.L.: *Circuit, Systems, and Signals for bioengineers: A Matlab based introduction*. Elsevier Academic Press, 2005. – ISBN 0–12–08–8493–3

[SF97] Stolfi, Jorge ; Figueiredo, Luiz Henrique D. *Self-Validated Numerical Methods and Applications*. 1997

[SF03] Stolfi, J. ; de Figueired, L.H.: An Introduction to Affine Arithmetic. In: *TEMA Trend. Mat. Apl. Comput.* Bd. 4, 2003, S. 297–312

[SG10] Schupfer, F. ; Grimm, C.: Towards more Dependable Verification of Mixed-Signal Systems. In: *Verification over discrete-continuous boundaries*. Dagstuhl, Germany : Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2010

(Dagstuhl Seminar Proceedings 10271). – ISSN 1862–4405

[SIVT06] Sevillano, Juan F. ; Irizar, Andoni ; Velez, Igone ; Tomasena, K.: Efficient Monte Carlo Simulation Using SystemC. In: *FDL*, ECSI, 2006. – ISBN 978–3–00–019710–9, S. 235–237

[SKG+10] Schupfer, F. ; Kärgel, M. ; Grimm, C. ; Olbrich, M. ; Barke, E.: Towards abstract analysis techniques for range based system simulations. In: *Specification Design Languages (FDL 2010), 2010 Forum on*, 2010, S. 1–6

[SKW08] Straka, Martin ; Kotásek, Zdeněk ; Winter, Jan: The design of hardware checkers for verification and diagnostic purposes. In: *Brno University of Technology-Faculty of Information Technology* 612 (2008), S. 66

[SLMW04] Shou, Huahao ; Lin, Hongwei ; Martin, Ralph ; Wang, Guojin: Modified affine arithmetic in tensor form. In: *The Proceedings of International Symposium on Computing and Information (ISC&I 2004), Zhuhai, Guangdong, China* Bd. 2, 2004, S. 642–646

[SS16] Sharma, Konark ; Saini, Lalit M.: Power-line communications for smart grid: Progress, challenges, opportunities. In: *Renewable and Sustainable Energy Reviews* 67 (2016), S. 704 – 751. – ISSN 1364–0321

[Sys12] IEEE Standard for Standard SystemC Language Reference Manual. In: *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)* (2012), Jan, S. 1–638

[TM13] Chapter From Process Variations to Performance Variations In: Trent McConaghy, Jeffrey Dyck Amit G.: *Variation-Aware Design of Custom Integrated Circuits: A Hands-on Field Guide*. Bd. 1. Springer-Verlag New York, 2013

[TS86] Tietze, U. ; Schenk, Ch.: *Halbleiter-Schaltungstechnik*. 8th. Berlin : Springer, 1986

[TUS16] Taranalli, V. ; Uchikawa, H. ; Siegel, P. H.: On the Capacity of the Beta-Binomial Channel Model for Multi-Level Cell Flash Memories. In: *IEEE Journal on Selected Areas in Communications* 34 (2016), Sept, Nr. 9, S. 2312–2324. – ISSN 0733–8716

[WH07] Wang, Z. H. ; Hu, H. Y.: Robust Stability of Time-Delay Systems with Uncertain Parameters. In: Hu, H. Y. (Publisher) ; Kreuzer, Edwin (Publisher): *Iutam Symposium on Dynamics and Control of Nonlinear Systems with Uncertainty*. Dordrecht : Springer Netherlands, 2007. – ISBN 978–1–4020–6332–9, S. 363–372

[WHR+03] Walker, W.E. ; Harremoes, P. ; Rotmans, J. ; vna der Sluijs, J.P. ; van Asselt, M.B.A. ; Janssen, P. ; von Krauss, M.P. K.: Defining Uncertainty - A Conceptual Basis for Uncertainty Management in Model-Based Decision Support. In: *Journal of Integrated Assessment* (2003), Nr. 1, S. 5–17

[WHW15] Wang, S. ; Han, L. ; Wu, L.: Uncertainty Tracing of Distributed Generations via Complex Affine Arithmetic Based Unbalanced Three-Phase Power Flow. In: *IEEE Transactions on Power Systems* 30 (2015), Nov, Nr. 6, S. 3053–3062. – ISSN 0885–8950

[Wie17] Wiesinger, R:: *Aging and Wear Out Effects in SoCs*. Jan 2017. – TU Wien, SoC Seminar Course (384.159)

[Wil03] Wille, Detlef: *Repetitorium der Linearen Algebra Teil 1*. Bd. 4. Springer, 2003. – ISBN 3–923923–40–6

[WJBK10] Warmink, J.J. ; Janssen, J.A.E.B. ; Booij, M.J. ; Krol, M.S.: Identification and classification of uncertainties in the application of environmental models. In: *Environmental Modelling and Software* 25 (2010), Nr. 12, S. 1518 – 1527. – ISSN 1364–8152

[WJCK16] Welte, A. ; Jaulin, L. ; Ceberio, M. ; Kreinovich, V.: Robust data processing in the presence of uncertainty and outliers: Case of localization problems. In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016, S. 1–7

[Wol09] Wolf, Marco: *A Modeling Language for Measurement Uncertainty Evaluation*, Swiss Federal Institute of Technology, Zuerich, PhD Thesis, 2009

[WS15] Winner, H. ; Schopper, M.: *Adaptive cruise control Handbuch Fahrerassistenzsysteme*. Springer, 2015. – 851–891 S

[XHZ12] Xiao, Xi-sheng ; Huang, Ying-ping ; Zhang, Xi-hui: Optimizing checkpoint for scientific simulations. In: *Journal of Zhejiang University SCIENCE C* 13 (2012), Dec, Nr. 12, S. 891–900. – ISSN 1869–196X

[YWS17] Yu, T. ; Wang, X. ; Shami, A.: Recursive Principal Component Analysis-Based Data Outlier Detection and Sensor Data Aggregation in IoT Systems. In: *IEEE Internet of Things Journal* 4 (2017), Dec, Nr. 6, S. 2207–2216

[ZMR17] Zimpeck, A. L. ; Meinhardt, C. ; Reis, R.: Robustness evaluation of finFET transistors under PVT variability. In: *2017 1st Conference on PhD Research in Microelectronics and Electronics Latin America (PRIME-LA)*, 2017, S. 1–4

Internet References

[1] http://beltoforion.de/article.php?a=muparser. muparser - Fast Math Parser Library - Version 2.2.5, Accessed: July 2017.

[2] http://mariusbancila.ro/blog/2009/02/03/evaluating-expressions-part-1/. Tutorial in developing a parser for mathematical expressions, Accessed: July 2017.

[3] http://nusmv.fbk.eu/. NuSMV: a new symbolic model checker, Accessed: January 2018.

[4] https://blogs.synopsys.com/theeyeshaveit/2012/09/05/silicon-validating-28-nm-interface-and-analog-ip-designs/. Silicon Validating 28-nm Interface and Analog IP Designs, Accessed: September 2012.

[5] https://community.cadence.com/cadence_blogs_8/b/cic/posts/keeping-your-circuit-in-tune. Keeping Your Circuit in Tune: Sensitivity Analysis and Circuit Optimization, Accessed: April 2014.

[6] https://de.mathworks.com/matlabcentral/linkexchange/links/936-b4m-interval-arithmetic-toolbox. b4m - Interval Arithmetic Toolbox, Accessed: November 2017.

[7] https://news.samsung.com/global/qualcomm-and-samsung-collaborate-on-10nm-process-technology-for-the-latest-snapdragon-835-mobile-processor. Qualcomm and Samsung Collaborate on 10nm Process Technology for the Latest Snapdragon 835 Mobile Processor - Samsung Newsroom - November 2017, Accessed: December 2017.

[8] https://semiengineering.com/10nm-versus-7nm/. 10nm versus 7nm, Accessed: April 2016.

[9] https://semiengineering.com/22nm-process-war-begins/. 22nm Process War Begins, Accessed: April 2017.

[10] https://semiengineering.com/moores-law-a-status-report/. Moore s Law: A Status Report, Accessed: April 2017.

[11] https://semiengineering.com/racing-to-10-7nm/. The Race to 10-7nm, Accessed: May 2017.

[12] https://semiengineering.com/uncertainty-grows-for-5nm-3nm/. Uncertainty Grows For 5nm, 3nm, Accessed: December 2016.

[13] https://www.cadence.com/content/cadence-www/global/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform.html?cmp=dmzweb1. Cadence JasperGold Formal Verification Platform (Apps), Accessed: January 2018.

[14] https://www.cadence.com/content/cadence-www/global/en_US/home/tools/system-design-and-verification.html. Cadence Verification Suite, Accessed: July 2017.

[15] https://www.design-reuse.com/articles/28591/mixed-signal-ip-design-challenges-in-28-nm-and-beyond.html. Mixed-Signal IP Design Challenges in 28 nm and Beyond, Accessed: December 2016.
[16] https://www.mentor.com/products/fv/questa-formal/. Questa Formal Verification - Complements simulation-based RTL design verification, Accessed: January 2018.
[17] https://www.modelica.org/. Modelica, Accessed: January 2018.
[18] https://www.sae.org/standards/content/j2748_200610/. VHDL-AMS Statistical Analysis Packages, Accessed: January 2018.
[19] https://www.synopsys.com/verification/ams-verification/waveform-analysis-debug/custom-waveview-adv.html. Synopsys Custom WaveView ADV, Accessed: January 2018.
[20] https://www.synopsys.com/verification/virtual-prototyping/platform-architect.html. Platform Architect MCO, Accessed: January 2018.
[21] https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html. Zynq-7000 All Programmable SoC, Accessed: January 2018.
[22] http://t-filter.engineerjs.com/. TFilter - The free online FIR filter design tool, Accessed: January 2018.
[23] http://www.accellera.org/downloads/standards/systemc. Accellera Systems Initative - SystemC Standard, Accessed: February 2014.
[24] http://www.boost.org/doc/libs/1_65_1/libs/numeric/interval/doc/interval.htm. Boost Interval Arithmetic Library, Accessed: November 2017.
[25] http://www.cplusplus.com/reference/unordered_map/unordered_map/. Unordered Map - C++ reference, Accessed: September 2017.
[26] http://www.cplusplus.com/reference/vector/vector/. Vector- C++ reference, Accessed: November 2017.
[27] http://www.deepchip.com/posts/dvcon07.html. The 2007 DeepChip Verification Census - A Census of 818 Engineers on Design Verification Tool Use, Accessed: July 2017.
[28] http://www.flowcad.de/. FlowCAD, Accessed: January 2018.
[29] http://www.itl.nist.gov/div898/handbook/. NIST/SEMATECH e-Handbook of Statistical Methods, Accessed: Feb 2015.
[30] http://www.ni.com/en-us/shop/labview.html. National Instruments LabView 2017, Accessed: January 2018.
[31] http://www.nongnu.org/libaffa/. Libaffa - C++ Library for Affine Arithmetic, Accessed: August 2017.
[32] http://www.omgsysml.org/. SysML webpage, Accessed: December 2017.
[33] http://www.onsemi.com/PowerSolutions/content.do?id=16693. C5: 0.5 Micrometer Process Technology, Accessed: October 2014.
[34] http://www.oxforddictionaries.com/definition/english/system. Oxford Dictionaries, Accessed: January 2014.
[35] http://www.prismmodelchecker.org/. PRISM - A probabilistic model checker, Accessed: January 2018.
[36] http://www.rcm2.co.uk/products/DOORS.htm. DOORS (Dynamic Object-Oriented Requirements System), Accessed: December 2017.
[37] http://www.stack.nl/~dimitri/doxygen/. Doxygen code documentation tool, Accessed: January 2018.
[38] http://www.systemc-ams.org/. SystemC AMS - Home page of the SystemC AMS study group, Accessed: February 2014.
[39] http://www.techdesignforums.com/practice/guides/guide-

`to-assertion-based-verification/`. Assertion-based verification, Accessed: July 2017.

[40] `http://www.ti.com/solution/software-defined-radio-sdr-diagram`. SDR- Software defined radio solutions from Texas Instruments, block-diagram, design considerations and application notes, Accessed: December 2013.

[41] `http://www.usbdev.ru/cics/icsmi/`. SM2XX Family - Flash Memory Card Controllers, Accessed: December 2017.

Curriculum Vitae

Dipl.-Ing. Michael Rathmair studies at the TU Wien since 2004. He finished the bachelor program for electrical engineering and information technology and the master program in the field of computer technology. His master thesis is about the design of computer systems for smart energy applications. Since 2011 he started his Ph.D. studies, and works as a university- and project assistant at the TU Wien - Insitute of Computer Technology (ICT). His scientific interests focus in the field of enhanced system verification (range-based simulation methods, formal system verification, model-driven testing), and the design of leading-edge IoT and industry 4.0 applications.

List of selected publications:

RATHMAIR, M. ; LUCKENEDER, C. ; KAINDL, H. ; RADOJICIC, C.: Semi-symbolic Simulation and Analysis of Deviation Propagation of Feature Coordination in Cyber-physical Systems. In: *Proceedings of the 51st Hawaii International Conference on System Sciences - HICSS 18*

M. RATHMAIR, C. LUCKENEDER, H. KAINDL: Minimalist Qualitative Models for Model Checking Cyber-physical Feature Coordination. In *Proceedings of the 23rd Asia-Pacific Software Engineering Conference (APSEC 2016)*, S. 8, Hamilton, Neuseeland, 2016.

GRIMM, C. ; RATHMAIR, M.: Dealing with Uncertainties in Analog/Mixed-Signal Systems: Invited. In: *Proceedings of the 54th Annual Design Automation Conference 2017*

M. RATHMAIR, F. SCHUPFER, C. RADOJICIC, C. GRIMM Extended Framework for System Simulation with Affine Arithmetic. In *Proceedings of the 2012 Forum on specification & Design Languages (FDL 2012)*, 2012, S. 161 - 168.

M. RATHMAIR, F. SCHUPFER Metrics for Formal Property Checking Against Undesired Circuit Behavior in Embedded Systems. In: *ITG-Fachbericht ANALOG 2016*, 2016, S. 64 - 69.

M. RATHMAIR, M. MOSBECK, M. MEISEL, S. WILKER Embedded systems design for Industry 4.0, Presentation: *eNDUSTRIE 4.0 Hackathon (invited)*, Sonnenwelt Großschönau.

M. RATHMAIR, F. SCHUPFER, C. KRIEG: Applied Formal Methods for Hardware Trojan Detection. In: *Proceedings of ISCAS2014, International Symposium on Circuits and Systems*, 2014, S. 169 - 172.