# Virtual Prototyping Methodology for Power Automation Cyber-Physical-Systems

THÈSE N$^{O}$ 7649 (2017)

PRÉSENTÉE LE 5 MAI 2017

À LA FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR
LABORATOIRE DE SYSTÈMES MICROÉLECTRONIQUES
PROGRAMME DOCTORAL EN MICROSYSTÈMES ET MICROÉLECTRONIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Juan Sebastian RODRIGUEZ ESTUPIÑAN

acceptée sur proposition du jury:

Dr M. Mattavelli, président du jury
Prof. Y. Leblebici, Dr A. Vachoux, directeurs de thèse
Prof. C. Grimm, rapporteur
Dr Y. Maret, rapporteur
Prof. G. de Micheli, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2017

Be ashamed to die until you have
won some victory for humanity
— Horace Mann

This work is dedicated to my parents, Maria and Jairo (the roots)
and my beautiful niece Alicia (the future)

# Acknowledgements

I would like to start by expressing my deepest and most sincere gratitude to my thesis co-director Dr. Alain Vachoux for his endless support, dedicated guidance, and motivational effort which were crucial in the toughest moments of this research. This thesis work would not have been possible without his advice and patience. I have learned from him many valuable hard and soft skills very useful in my career. During all these years he has been more than a supervisor, he has been my mentor, my friend.

My most sincere gratitude to my thesis director Prof. Yusuf Leblebici for the opportunity to conduct this research in his prestigious laboratory. Likewise, I would like to acknowledge Dr. Christian Ohler, Dr. Andrea Andenna, and Dr. Gian-Luigi Madonna from ABB Switzerland Ltd. for allowing me to conduct this research under a close collaboration with the Integrated Sensor Systems group of ABB Corporate Research in Baden-Dättwil, Switzerland.

I owe a very special and deep gratitude to my thesis supervisor in ABB Dr. Yannick Maret. His very accurate advice and feedback have been key for the success of this thesis work. I strongly appreciate his support, dedication, and diplomacy, which were determining to culminate this work. I cannot forget the enormous support and contribution from Dr. Joris Pascal, he was one of the precursors of this research topic. I appreciate his guidance and positive feedback. I would like also to express my gratefulness to Dr. Calogero Bona from ABB, his explanations and aid were of paramount importance. Additionally, I thank all ABB engineers whom directly or indirectly participated in my research during all these years.

I express my sincere gratitude to all the jury members of my thesis committee, in special Prof. Giovanni de Micheli, Prof. Christoph Grimm, and Dr. Marco Mattavelli, for taking the time to read this dissertation and evaluate my achievements. Let me thank again Prof. Christoph Grimm for his fruitful feedback and precise critics that helped to complement and emphasize the impact of this work.

I warmly thank all my colleagues and friends that were or are currently part of the LSM laboratory: Clemens Nyffeler, Reza Ranjandish, Cosimo Aprile, Jury Sandrini, Kiarash Gharibdoust, Ömer Çoğal, Behnoush Attarimashalkoubeh, Selman Ergünay, Gain Kim, Mustafa Kılıç, Jonathan Narinx, Elmira Shahrabi, Kerem Seyid, Vladan Popovic, Sylvain Hauser, Kadir Akın, Nikola Katic, Giulia Beanato, Alessandro Cevrero, Davide Sacchetto, Hossein Afshari, Armin

Tajalli, among many others. I was very lucky to have the opportunity to share with them important academic moments and/or very special occasions that made my life happier and easier as a doctoral student in EPFL. My gratitude goes to my friend Dr. Clemens Nyffeler, who helped me with the German version of the Abstract of this thesis. I highly appreciate the help from Dr. Armin Tajalli for helping me to understand and solve the stability issues presented in the model of the $2^{nd}$ order $\Sigma\Delta$ modulator. I am very grateful to the Swiss-Italian-Belgian-Iranian friends who made part of my lunch and coffee break crew, those were priceless moments of joy. I also thank my Turkish office mates for the warm and cozy coffee moments and discussions.

I must thank all my friends in Switzerland and abroad who have shared with me precious instants in this exciting journey and have been a great moral support in the toughest moments. I would like to thank my friend Fernando Escobar Pacheco for his legal counseling and recommendations. A special message of gratitude goes to Maria Isabel, who was my inspiration and motivation.

Last but not least, I would like to thank my family, whom has never doubted on my capabilities and has enormously supported me at all time. They have made sure of making me feel their presence despite the distance. I owe this triumph to them: to my mother who taught me dedication and perseverance, to my father who constantly remind me to believe in myself (he helped me with the edition of several plots of this thesis document), to my brother who has supported me from the distance, and to my sister who has been there to help me when nobody else could. I thank my brother Esteban for being an inspiration with his lovely family.

*Lausanne, April 21st 2017* Juan Sebastián Rodríguez Estupiñán

# Abstract

Constant improvements in science, technology and engineering have allowed to build progressively smarter and more complex systems which interrelates almost any technical and scientific domain for a vast amount of industrial applications. However, the overwhelming complexity of these systems, better known as *Cyber-Physical Systems* (CPS), imposes a serious challenge for its design, optimization, testing, manufacturing, and operation. The main cause of the complexity in CPS is the intrinsic heterogeneity of their subsystems and components, which drives the need of analyzing cross-domain interactions among them and their environment.

Although there is a plethora of techniques and methods for CPS design and verification, the only clear consensus to cope with the increasing complexity and heterogeneity of CPS is the use of computational models and simulation, i.e. *virtual prototyping*. The main objective of *virtual prototyping* is to allow the analysis of a system, or part of it, from concerned life-cycle aspects such as the design, optimization, testing, and operation as on a real physical prototype. Nevertheless, the lack of a well-defined and integrated design methodology and its related techniques for multi-domain complex systems, is one of the main hurdles for an effective utilization of *virtual prototyping* in the system development life-cycle. Model-based methodologies and integrated simulation frameworks are required to ensure highly demanded general requirements of CPS such as inherent safety and maximal reliability within a limited developing time and reduced costs.

In this thesis, the author proposes a circular system development model (O-model) which considers all the stages in a typical development process for industrial systems. In particular, the present work shows that the use of *virtual prototyping* at early stages of the system development may reduce the overall design and verification effort by allowing the exploration of the complete system architecture, and uncovering integration issues early on. The modeling techniques of this research are based on VHDL-AMS, yet supporting other modeling languages such as C/C++, SPICE, and Verilog-AMS, together with integrated simulation tools. Contrasting with conventional approaches, it is shown that the proposed methodology is adapted for small-scale CPS design and verification thanks to the modularity and scalability of the modeling approach. The proposed modeling techniques enable seamlessly the CPS design together with the implementation of their subsystems. In particular, the contribution of this work improves the *virtual prototyping* approach that has been successfully used during the development of smart electrical sensors and monitoring equipment for high and medium voltage applications. The design of the measurement and self-calibration circuits of a medium voltage current sensor based on the Rogowski coil transducer is presented as an example.

The proposed small-scale CPS design methodology based on *virtual prototyping*, namely VP-based design methodology, uses important theoretical concepts from layered design, component-based design, and platform-based design. These foundations are the basis to build a modeling methodology that provides a vehicle that can be used to improve system verification towards correct-by-design systems. The main contributions of this research are: the re-definition of the system development lifecycle by using a *virtual prototyping* methodology; the design and implementation of a model library that maximizes the reuse of computational models and their related intellectual property (IP); and a set of VHDL-AMS modeling guidelines established with the purpose of improving the modularity and scalability of virtual prototypes. These elements are key for supporting the introduction of *virtual prototyping* into industrial companies that can thoroughly profit from this approach, but cannot commit a specific team to the creation, support, and maintenance of computational models and its dedicated infrastructure. Thanks to the progressive nature of the proposed methodology, virtual prototypes can indeed be introduced with relatively low initial effort and enhanced over time.

The presented methodology and its infrastructure may grow into a bidirectional communication medium between non-expert system designers (i.e. system architects and virtual integrators) and domain specialists such as mechanical designers, power electrical designers, embedded-electronics designers, and software designers. The proper utilization of system-level modeling and gradual model refinement implemented in a common simulation framework, are among the most important concepts of the VP-based design methodology presented in this thesis. Specifically, the proposed design methodology advocates the reduction of the CPS design complexity by the implementation of a meet-in-the-middle approach for system-level modeling. In this direction, the modeling techniques introduced in this work facilitate the architectural design space exploration, critical cross-domain variable analysis (especially important in the component interfaces), and system-level optimization and verification.

Key words: Cyber-Physical Systems, virtual prototyping, model-based design methodology, system-level modeling and verification, model refinement, modularity and scalability of models, virtual prototypes, power and industrial automation, VHDL-AMS, system development lifecycle, product lifecycle management, Industry 4.0

# Résumé

L’amélioration constante de la science, de la technologie et de l’ingénierie a permis de construire progressivement des systèmes plus intelligents et plus complexes qui interconnectent presque tous les domaines techniques et scientifiques pour une vaste gamme d’applications industrielles. Cependant, la grande complexité de ces systèmes, mieux connus sous le nom de systèmes cyber-physiques (CPS), impose un sérieux défi pour leur conception, leur optimisation, leur vérification, leur fabrication et leur fonctionnement. La principale cause de la complexité dans les CPS est l’hétérogénéité intrinsèque de leurs sous-systèmes et composants, ce qui conduit à la nécessité d’analyser les interactions entre eux et leur environnement.

Bien qu’il existe une pléthore de techniques et de méthodes pour la conception et la vérification des CPS, le seul consensus clair pour faire face à la complexité croissante et l’hétérogénéité de CPS est l’utilisation de modèles informatiques et de la simulation, c’est-à-dire le prototypage virtuel. L’objectif principal du prototypage virtuel est de permettre l’analyse d’un système ou d’une partie de celui-ci lors des différentes étapes de conception, d’optimisation, de test et d’exploitation, de la même manière que sur un prototype physique. Néanmoins, l’absence d’une méthodologie de conception bien définie et intégrée et de ses techniques associées pour les systèmes complexes multi-domaines est le principal obstacle à une utilisation efficace du prototypage virtuel dans le cycle de vie du développement de CPS. Des méthodologies basées sur les modèles et les environnements de simulation intégrés sont nécessaires pour remplir les exigences, telles qu’une sécurité inhérente et une fiabilité maximale, dans un temps de développement limité et pour des coûts réduits.

Dans cette thèse, l’auteur propose un modèle circulaire de développement de systèmes appelé *O-model* qui tient compte de toutes les étapes d’un processus de développement typique de systèmes industriels. En particulier, on montre que l’utilisation du prototypage virtuel aux premiers stades du développement du système peut réduire les efforts de la conception et de vérification de systèmes en permettant l’exploration de l’architecture complète du système, ce qui contribue à découvrir des problèmes d’intégration plus rapidement. Les techniques de modélisation de cette recherche sont basées sur le langage de modélisation VHDL-AMS, mais supportent aussi des langages tels que C/C++, SPICE et Verilog-AMS, ainsi que des outils de simulation intégrés. Contrairement aux approches conventionnelles, il est démontré que la méthodologie proposée est adaptée à la conception et à la vérification de CPS à petite échelle grâce à la modularité et à l’évolutivité de l’approche de modélisation. Les techniques de modélisation proposées permettent de concevoir efficacement la conception de CPS ainsi que la mise en œuvre de leurs sous-systèmes. Particulièrement, la contribution de ce travail améliore l'approche de prototypage virtuel qui a été utilisée avec succès pour le développement de capteurs électriques intelligents et d'équipement de surveillance pour les applications à haute et moyenne tension. Comme exemple d'application principal, on présente la conception des circuits de mesure et d'autocalibrage d'un capteur de courant de moyenne tension basé sur le transducteur enroulement de Rogowski.

La méthodologie de conception de CPS à petite échelle utilise des concepts théoriques importants de la conception stratifiée, de la conception basée sur les composants, et de la conception basée sur les plate-formes. Ces fondements sont la base pour construire une méthodologie de modélisation qui fournit un moyen qui peut être utilisé pour améliorer la vérification de systèmes vers les systèmes corrects par conception. Les principales contributions de cette recherche sont : la redéfinition du cycle de vie du développement de systèmes en utilisant une méthodologie de prototypage virtuel ; la conception et la mise en œuvre d'une bibliothèque de modèles qui maximise la réutilisation des modèles informatiques et de leur propriété intellectuelle connexe ; et un ensemble de directrices de modélisation VHDL-AMS établies dans le but d'améliorer la modularité et l'évolutivité des prototypes virtuels. Ces éléments sont essentiels pour soutenir l'introduction du prototypage virtuel dans les entreprises industrielles qui désirent profiter pleinement de cette approche, mais ne peuvent engager une équipe spécifique pour la création, le support et la maintenance de modèles informatiques et de son infrastructure dédiée. Grâce à la nature progressive de la méthodologie proposée, les prototypes virtuels peuvent en effet être introduits avec un effort initial relativement faible et améliorés au fil du temps.

Dans ce contexte, la méthodologie présentée et son infrastructure peuvent se transformer en un moyen de communication bidirectionnel entre les concepteurs de systèmes non spécialisés (c'est-à-dire les architectes des systèmes et les intégrateurs virtuels) et les spécialistes de domaines tels que les concepteurs mécaniciens, les concepteurs électriques, les concepteurs d'électronique embarquée et les concepteurs de logiciels. L'utilisation appropriée de la modélisation au niveau du système et du raffinement progressif de modèles sont parmi des concepts les plus importants de la méthodologie de conception proposée. Plus précisément, on préconise la réduction de la complexité par la mise en œuvre d'une approche de modélisation dénommée «meet-in-the-middle». Dans cette direction, les techniques de modélisation introduites dans ce travail facilitent l'exploration de l'espace de conception architecturale, l'analyse critique des variables inter-domaines (particulièrement importante dans les interfaces des composants) et l'optimisation et la vérification du système.

Mots clefs : systèmes cyber-physiques, prototypage virtuel, méthodologie de conception basée sur des modèles, modélisation et vérification au niveau du système, raffinement de modèles, modularité et évolutivité de modèles, prototypes virtuels, automatisation industrielle et de puissance, VHDL-AMS, gestion du cycle de vie des produits, Industrie 4.0

# Zusammenfassung

Durch ständige Verbesserungen in Wissenschaft, Technologie und Ingenieurwesen konnten schrittweise intelligentere und komplexere Systeme aufgebaut werden, die nahezu alle technischen und wissenschaftlichen Bereiche für eine Vielzahl von industriellen Anwendungen miteinander verknüpfen. Allerdings stellt die überwältigende Komplexität dieser Systeme, besser bekannt als Cyber-Physische Systeme (CPS), eine ernsthafte Herausforderung für deren Konstruktion, Optimierung, Prüfung, Herstellung und Betrieb dar. Die Hauptursache für die Komplexität von CPS ist die intrinsische Heterogenität ihrer Subsysteme und Komponenten, welche die Analyse domänenübergreifender Wechselwirkungen zwischen ihnen und ihrer Umgebung erforderlich macht.

Obwohl es eine Fülle von Techniken und Methoden für CPS-Design und Verifikation gibt, ist der einzige eindeutige Konsens, mit der zunehmenden Komplexität und Heterogenität von CPS fertig zu werden, die Verwendung von Berechnungsmodellen und Simulation, genannt virtuelles Prototyping. Das Hauptziel des virtuellen Prototyping ist es, die Analyse eines Systems oder eines Teils davon in Lebenszyklusaspekten wie Design, Optimierung, Test und Betrieb wie auf einem realen physikalischen Prototyp zu ermöglichen. Dennoch ist das Fehlen einer wohldefinierten und integrierten Entwurfsmethodologie und der damit verbundenen Techniken für komplexe Systeme mit mehreren Domänen eine der wichtigsten Hürden für eine effektive Nutzung virtueller Prototypen im Systementwicklungszyklus. Modellbasierte Methoden und integrierte Simulations-Frameworks sind erforderlich, um dringend geforderte allgemeine Anforderungen an CPS wie etwa inhärente Sicherheit und maximale Zuverlässigkeit innerhalb einer begrenzten Entwicklungszeit und bei reduzierten Kosten zu gewährleisten.

In dieser Arbeit schlägt der Autor ein zirkuläres Systementwicklungsmodell vor, das alle Stadien eines typischen Entwicklungsprozesses für industrielle Systeme berücksichtigt. Insbesondere zeigt die vorliegende Arbeit, dass der Einsatz virtueller Prototypen in frühen Stadien der Systementwicklung den Gesamtentwurfs- und Verifikationsaufwand verringern kann, indem er die Erforschung der kompletten Systemarchitektur ermöglicht und frühzeitig Probleme bei der Integration aufdeckt. Die Modellierungstechniken basieren auf VHDL-AMS, unterstützen aber kompatible Sprachen wie C/C++, SPICE und Verilog-AMS sowie integrierte Simulationswerkzeuge. Im Gegensatz zu herkömmlichen Ansätzen wird gezeigt, dass die vorgeschlagene Methodik aufgrund der Modularität und Skalierbarkeit des Modellierungsansatzes für die CPS-Konstruktion und -Verifikation im kleinen Masstab geeignet ist. Die vorgeschlagenen Modellierungstechniken ermöglichen eine nahtlose Gestaltung des CPS-Designs und die Implementierung der Subsysteme. Der Beitrag dieser Arbeit verbessert insbesondere den virtuellen Prototyping-Ansatz, der bereits erfolgreich bei der Entwicklung von intelligenten elektrischen Sensoren und Überwachungseinrichtungen für Hoch- und Mittelspannungsanwendungen eingesetzt wurde. Als Beispiel wird der Aufbau der Mess- und Selbstkalibrierungsschaltungen eines Mittelspannungsstromsensors anhand des Rogowski-Spulenkörpers vorgestellt.

Die vorgeschlagene, auf dem virtuellen Prototyping basierende CPS-Designmethodik, verwendet wichtige theoretische Konzepte aus Layered Design, Komponenten-basiertem Design und plattformbasiertem Design. Diese bilden die Grundlage für eine Modellierungsmethodik als ein Instrument mit dem die Systemverivizierung so verbessert werden kann, dass von Design aus korrekte Systeme (correct-by-design) realisiert werden können. Die Hauptbeiträge dieser Forschung sind: die Neudefinition des Systementwicklungs-Lebenszyklus durch eine virtuelle Prototyping-Methodik; Die Konzeption und Implementierung einer Modellbibliothek, die die Wiederverwendung von Berechnungsmodellen und des damit verbundenen intellektuellen Eigentums (intellectual property, IP) maximiert; Und eine Reihe von VHDL-AMS-Modellierungsrichtlinien mit dem Ziel der Verbesserung der Modularität und Skalierbarkeit von virtuellen Prototypen. Diese Elemente sind der Schlüssel, um industrielle Unternehmen bei der Einführung von virtuellem Prototyping zu unterstützen, die von diesem Ansatz zwar profitieren können aber nicht die Möglichkeit haben ein eigenes Team zur Erstellung, Unterstützung und Wartung der Berechnungsmodelle und deren dedizierter Infrastruktur aufzustellen. Dank der progressiven Art der vorgeschlagenen Methodik können virtuelle Prototypen tatsächlich mit relativ niedrigem Anfangsaufwand eingeführt und im Nachhinein fortlaufend verbessert werden.

Die dargestellte Methodik und ihre Infrastruktur können zu einem bidirektionalen Kommunikationsmedium wachsen, anhand dessen weniger nicht-spezialisierte Systemdesigner (d.h. Systemarchitekten und virtuelle Integratoren) mit Domänenspezialisten wie zum Beispiel mechanischen Konstrukteuren, Leistungselektronik-Designern, Embedded-Elektronik-Designern und Software-Designern zusammenwirken können. Die ordnungsgemäße Nutzung der Modellierung auf Systemebene und die schrittweise Verfeinerung der Modelle innerhalb eines gemeinsamen Simulationsrahmens gehören zu den wichtigsten Konzepten der VP-basierten Design-Methodik. Insbesondere fördert die vorgeschlagene Entwurfsmethodologie die Verringerung der CPS-Entwurfskomplexität durch die Implementierung eines meet-in-the-middle-Ansatzes für die Systemebenenmodellierung. In dieser Richtung erleichtern die in dieser Arbeit eingeführten Modellierungsverfahren die architektonische Entwurfsraumerkundung, die domänenübergreifende analyse kritischer Variablen (besonders wichtig in den Komponentenschnittstellen) und die Optimierung und Verifikation auf Systemebene.

Stichwörter: Cyber-Physikalische Systeme, virtuelles Prototyping, modellbasierte Designmethodik, Modellierung und Verifikation auf Systemebene, Modellverfeinerung, Modularität und Skalierbarkeit von Modellen, Produktlebenszyklusmanagement, Industrie 4.0

# Contents

**Acknowledgements** **i**

**Abstract (English/Français/Deutsch)** **iii**

**Table of contents** **xii**

**List of figures** **xiii**

**List of tables** **xvii**

**Acronyms** **xix**

**Glossary** **xxiii**

**1 Introduction** **1**
- 1.1 Context and problem description . . . 1
- 1.2 Main objectives and rationale . . . 3
- 1.3 Focus and application . . . 4
- 1.4 Motivation for system-level modeling . . . 5
- 1.5 Document organization . . . 6

**2 State of the art** **9**
- 2.1 The V-model . . . 9
- 2.2 Layered Design . . . 10
- 2.3 Component-based Design . . . 11
- 2.4 Model-based Design . . . 12
- 2.5 Modeling languages and tools . . . 13
- 2.6 Platform-based Design . . . 16
- 2.7 Contract-based Design . . . 17
- 2.8 Product lifecycle management (PLM) . . . 18
- 2.9 Industry 4.0 . . . 19

**3 The VP-based design methodology** **21**
- 3.1 Design methodology foundations . . . 21
  - 3.1.1 Layered and component-based design . . . 21
  - 3.1.2 Platform-based design . . . 22

3.1.3 System-level modeling . . . 24
3.2 Model refinement . . . 25
3.3 Design methodology elements . . . 28
3.4 Simulation framework . . . 30
3.5 The VP-Modeling Guidelines . . . 31
3.5.1 Mapping in configurations . . . 43
3.6 The VP-Model Library . . . 47
3.6.1 Issues and objectives . . . 48
3.6.2 Model metadata . . . 49
3.7 Conclusions . . . 51

**4 The Rogowski coil sensor system 53**
4.1 System description . . . 53
4.2 The electro-thermal modeling of the Rogowski coil sensor . . . 57
4.2.1 Rogowski coil sensor system assumptions . . . 57
4.2.2 Electrical model of the Rogowski coil . . . 59
4.2.3 Rogowski coil electrical model adaptation . . . 61
4.2.4 Busbar thermal model . . . 63
4.2.5 Thermal coupling model . . . 64
4.2.6 Heat transfer modeling . . . 66
4.2.7 Equation-based model extraction for the Rogowski coil thermal coupling 78
4.2.8 VHDL-AMS electro-thermal Rogowski coil simulation and analysis . . . 80
4.3 The self-calibration unit of the Rogowski coil sensor . . . 83
4.3.1 Self-calibration method . . . 83
4.3.2 The SCU virtual prototype . . . 86
4.4 The current measurement unit of the Rogowski coil sensor . . . 104
4.4.1 Equations for the primary current estimation . . . 105
4.4.2 The CMU virtual prototype . . . 107
4.5 The Virtual prototype of the Rogowski coil sensor system . . . 126
4.6 Conclusions . . . 130

**5 The circular system development model 133**
5.1 Abstract . . . 133
5.2 Introduction . . . 134
5.3 The O-model . . . 135
5.3.1 System definition . . . 137
5.3.2 System-level design . . . 141
5.3.3 Subsystem-level design . . . 147
5.3.4 System verification . . . 151
5.3.5 Physical system validation . . . 153
5.3.6 System manufacturing . . . 154
5.3.7 System commissioning . . . 154
5.3.8 System maintenance . . . 154

5.4 Model maintenance . . . . . 155

**6 Conclusions and future work 161**
6.1 Future work . . . . . 164

**A Appendix A - ABB Rogowski coil specifications 167**
A.1 KEVCR core dimensions . . . . . 167
A.2 KEVCR parameters . . . . . 168
A.3 KEVCR datasheet . . . . . 168
A.4 KECA core dimensions . . . . . 173
A.5 KECA parameters . . . . . 174
A.6 KECA datasheet . . . . . 174

**B Appendix B - Model equations 179**
B.1 Equations of the Rogowski coil electrical model . . . . . 179
B.2 Principles of conductor temperature determination . . . . . 182
B.2.1 Heat balance . . . . . 182
B.2.2 IEEE STANDARD 738 – 2006 . . . . . 184
B.2.3 Equivalent lumped-element circuit model of a Busbar conductor . . . . 186

**C Appendix C - Source code 187**
C.1 Sigma-Delta ADC . . . . . 187
C.1.1 Simple $1^{st}$ Order ΣΔ configuration . . . . . 189
C.1.2 Simple $2^{nd}$ Order ΣΔ configuration . . . . . 190
C.1.3 Sigma-Delta modulator . . . . . 190
C.1.4 $\text{Sinc}^3$ digital filter . . . . . 198
C.2 Instrumentation amplifier (INS_AMP) . . . . . 199
C.3 Non-ideal Operational Amplifier (OPAMP) . . . . . 200
C.4 Polynomial curve fitting algorithm script . . . . . 203

**D Appendix D - Generic parameter mapping 207**

**E Appendix E - Parametric and statistical simulations in VHDL-AMS 211**
E.1 Monte Carlo simulations . . . . . 211
E.1.1 Uniform distribution . . . . . 213
E.1.2 Normal distribution . . . . . 214
E.1.3 Piecewise linear distribution given as PDF . . . . . 215
E.1.4 Bernoulli distribution . . . . . 215
E.2 Parameter sweep simulations . . . . . 216
E.3 Defining custom statistical distributions . . . . . 221

**F Appendix F - VP-Model Library implementation 223**
F.1 Database implementation and deployment . . . . . 223
F.1.1 VP-Model Library actors . . . . . 223

F.1.2 IP security . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 225
F.1.3 File repository setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 226
F.2 VP-Model Library structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 227
F.2.1 Model views . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 228
F.2.2 Test Benches view . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 239
F.2.3 Example Results and Application Notes . . . . . . . . . . . . . . . . . . . . 243
F.2.4 Custom packages view . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 244
F.3 Modeling request . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 246

**Bibliography** **249**

**Curriculum Vitae** **259**

# List of Figures

1.1 V-model diagram . . . . . . . . . . 2

3.1 Platform-based design diagrams. . . . . . . . . . 23
3.2 VP Block Diagram . . . . . . . . . . 33
3.3 Conceptual diagram of a *test bench* and its DUV. . . . . . . . . . 34
3.4 Component declaration and instantiation code fragment example. . . . . . . . . . 35
3.5 Configurations inside configurations source code examples. . . . . . . . . . 36
3.6 Assertion code fragment . . . . . . . . . . 38
3.7 Package and test bench code fragment . . . . . . . . . . 39
3.8 `big_comp` parameter packages source code. . . . . . . . . . 40
3.9 Pacakge from configurations code . . . . . . . . . . 41
3.10 Logical component encapsulation example . . . . . . . . . . 42
3.11 OpAmp example . . . . . . . . . . 45
3.12 Port mapping example . . . . . . . . . . 46
3.13 Port mapping configuration example . . . . . . . . . . 47

4.1 Rogowski Coil Sensors . . . . . . . . . . 54
4.2 Conceptual diagram of the Rogowski coil sensor system . . . . . . . . . . 55
4.3 SCU closed loop measurement principle . . . . . . . . . . 56
4.4 Block diagram of the Electro-Thermal model of the Rogowski coil . . . . . . . . . . 58
4.5 Rogowski Coil symmetric 3Loops model . . . . . . . . . . 59
4.6 Rogowski coil impedance experimental vs. simulation results . . . . . . . . . . 60
4.7 Rogowski Coil asymmetric 1Loop model . . . . . . . . . . 60
4.8 RLC and RL approximated electrical models of the Rogowski coil. . . . . . . . . . . 61
4.9 Fraction of the VHDL-AMS code of the RogoCoil's electrical model. . . . . . . . . . 62
4.10 VHDL-AMS dynamic resistor model. . . . . . . . . . 63
4.11 Equivalent lumped-element circuit model of the Rogowski coil's thermal network 64
4.12 RC 3D Geometry . . . . . . . . . . 65
4.13 Unidimensional radial heat flow . . . . . . . . . . 66
4.14 Rogowski coil geometry . . . . . . . . . . 69
4.15 Cross-sectional view of the 2D axisymmetric RogoCoil model geometry. . . . . . 69
4.16 2D Geometry Meshing . . . . . . . . . . 73
4.17 Mesh quality statistics KEVCR . . . . . . . . . . 74

4.18 3D temperature plots of the KEVCR CHT model. . . . . . . . . . . . . . . . . . . . . 75
4.19 3D air velocity plots of the CHT model. . . . . . . . . . . . . . . . . . . . . . . . . . 76
4.20 CHT and SHT transient simulation. . . . . . . . . . . . . . . . . . . . . . . . . . . . 77
4.21 SHT static simulation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 77
4.22 Estimation of $\tau_{min}$ and $\tau_{max}$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 79
4.23 Temperature (°C) vs. Time (hours), ETRCM transient simulations. . . . . . . . . 81
4.24 ETRCM transient simulation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 82
4.25 Self-calibration method schematic and equations. . . . . . . . . . . . . . . . . . 84
4.26 SCU VP architecture. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87
4.27 Bode diagrams of a $3^{rd}$ order Sallen-Key active LPF. . . . . . . . . . . . . . . . . 88
4.28 RMS-to-DC functional design entity source code. . . . . . . . . . . . . . . . . . . 90
4.29 RMS-to-DC implicit computation architectures. . . . . . . . . . . . . . . . . . . . . 91
4.30 Modular approach for model refinement. . . . . . . . . . . . . . . . . . . . . . . . . 91
4.31 ADC functional modeling. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 92
4.32 ΣΔ ADC architectures. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93
4.33 $\text{Sinc}^3$ digital filter topology code. . . . . . . . . . . . . . . . . . . . . . . . . . . . 94
4.34 RTL $\text{Sinc}^3$ digital filter basic blocks. . . . . . . . . . . . . . . . . . . . . . . . . . . 95
4.35 Frequency response of the $\text{Sinc}^3$ digital filter with $M = 1024$. . . . . . . . . . . . 95
4.36 SCU VP modeling structure. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 96
4.37 SCU VP (Ideal Autocalibration) transient simulation. . . . . . . . . . . . . . . . . 97
4.38 SCU VP (Ideal $2^{nd}$ Order ΣΔ ADC) transient simulation. . . . . . . . . . . . . . . 98
4.39 SCU VP (Simple $1^{st}$ Order ΣΔ ADC) transient simulations. . . . . . . . . . . . . . 100
4.40 Impedance response simulation for the RogoCoil asymmetric 1Loop model. . . 100
4.41 SCU and RogoCoil physical prototypes. . . . . . . . . . . . . . . . . . . . . . . . . 101
4.42 KECA250B1 plots: experimental and virtual $R_{coil}$ and $L_{coil}$ SCU estimation as a function of the primary current. . . . . . . . . . . . . . . . . . . . . . . . . . . . 102
4.43 KEVCR17.5CA1 plots: experimental and virtual $R_{coil}$ and $L_{coil}$ SCU estimation as a function of the primary current. . . . . . . . . . . . . . . . . . . . . . . . . . . 104
4.44 Rogowski coil symmetrical Thevenin equivalent circuit and its load . . . . . . . 107
4.45 Conceptual block diagram of the CMU VP. . . . . . . . . . . . . . . . . . . . . . . 108
4.46 Transient response of the RogoCoil asymmetric 1Loop model. . . . . . . . . . . 109
4.47 CMU VP transient simulations using the reference calculation model. . . . . . . 110
4.48 Conceptual CMU Analog architecture. . . . . . . . . . . . . . . . . . . . . . . . . 111
4.49 Analog Integration block architecture and the `opamp(simple)` simple model. . 112
4.50 Differential amplifiers VHDL-AMS entity declaration. . . . . . . . . . . . . . . . 113
4.51 Conceptual CMU Digital architecture. . . . . . . . . . . . . . . . . . . . . . . . . . 114
4.52 Functional and Software model implementations of the Digital Integration block.116
4.53 Code of the `Hardware` *model implementation* of the Digital Integration block. . 117
4.54 Maximum estimated errors in the output signals of the CMU VP. . . . . . . . . 118
4.55 Analog CMU VP transient simulations at reduced ADC DNR. . . . . . . . . . . . 119
4.56 Input and output voltages of the Analog Integration block. . . . . . . . . . . . . 119
4.57 Effect of the ADC resolution on the current estimation of the Digital CMU VP. . 120

4.58 Input and output signals of the Digital Integration block. . . . . . . . . . . . . . 120
4.59 Analog CMU VP simulation results at two corner ADC frequencies. . . . . . . . 121
4.60 Transient simulations at $F_{ADC}$ = 500 Hz for all the integration methods of the Digital CMU VP. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 122
4.61 Transient simulation at $F_{ADC}$ = 500 Hz for the Digital CMU hardware configuration. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 123
4.62 Transient simulations of the CMU VP showing a DC shift in the primary current estimation. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 124
4.63 Transient simulation of the Analog CMU architecture (Behavioral_refined CMU configuration). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 126
4.64 RCSS VP complete architecture block diagram. . . . . . . . . . . . . . . . . . . 127
4.65 High-level block diagram of the RCSS VP. . . . . . . . . . . . . . . . . . . . . . . 129

5.1 Circular system development model (O-model) . . . . . . . . . . . . . . . . . . . 135
5.2 System definition diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 137
5.3 System-level design diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 142
5.4 Rogowski coil and its analog current measurement unit . . . . . . . . . . . . . . 146
5.5 Subsystem-level design diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . 148
5.6 System verification stage diagram . . . . . . . . . . . . . . . . . . . . . . . . . . 151
5.7 Model maintenance flow . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 156

A.1 KEVCR core schematics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 167
A.2 KECA core schematics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 173

B.1 Rogowski coil technical drawing. . . . . . . . . . . . . . . . . . . . . . . . . . . 179

D.1 Generic mapping principles . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 207
D.2 Generic mapping application example . . . . . . . . . . . . . . . . . . . . . . . 208

E.1 Resistive Voltage Divider . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 212
E.2 `test_stat` architecture of the RVD Test Bench . . . . . . . . . . . . . . . . . . . 213
E.3 Uniform Distribution RVD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 214
E.4 Normal Distribution RVD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 214
E.5 PWL-PDF Distribution RVD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 215
E.6 Bernoulli Distribution RVD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 216
E.7 Sweep function declaration for Monte Carlo Simulation . . . . . . . . . . . . . . 217
E.8 Sweep function declaration for standard parametric simulation . . . . . . . . . 217
E.9 `test_param` architecture of the RVD Test Bench . . . . . . . . . . . . . . . . . . 218
E.10 Vout vs. $R_2$ parameter sweep simulation . . . . . . . . . . . . . . . . . . . . . . 219
E.11 NestedSweep function declaration for MC simulations . . . . . . . . . . . . . . . 219
E.12 `test_nested` architecture of the RVD Test Bench . . . . . . . . . . . . . . . . . 220
E.13 RVD nested sweep surface plot . . . . . . . . . . . . . . . . . . . . . . . . . . . 221
E.14 Exponential distribution function . . . . . . . . . . . . . . . . . . . . . . . . . . 222
E.15 Exponential Distribution RVD . . . . . . . . . . . . . . . . . . . . . . . . . . . . 222

F.1 Relationship graph of the VP-Model Library database. . . . . . . . . . . . . . . . . 224
F.2 IWP Home Page . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 225
F.3 Model_library folder . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 226
F.4 Detailed view . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 228
F.5 Ports portal . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 232
F.6 Execution Capabilities portal . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 235
F.7 Block Diagram portal . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 236
F.8 VP Dependencies portal . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 236
F.9 Verification portal . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 237
F.10 Tool Dependencies portal . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 238
F.11 References portal . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 239
F.12 Test benches view . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 240
F.13 SMASH portal . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 242
F.14 Test bench configurations portal . . . . . . . . . . . . . . . . . . . . . . . . . . . 243
F.15 Custom packages view . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 244

# List of Tables

4.1 KEVCR global parameters. . . . . 71
4.2 Sweep parameter range. . . . . 78
4.3 Curve fitting coefficients for the KEVCR coils. . . . . 80
4.4 Self-calibration unit algorithm. . . . . 86
4.5 Text format for the input file of the transfer function multipurpose filter model. 89
4.6 Parameters of the transfer function multipurpose filter model. . . . . 89
4.7 ΣΔ ADC characteristics. . . . . 96
4.8 Summary of the $L_{coil}$ and $R_{coil}$ estimation errors. . . . . 99
4.9 Design Topics of system-level *virtual prototyping* for the CMU design. . . . . 108
4.10 Main equations of the CMU functional *component models*. . . . . 111
4.11 Digital integration methods of the Digital Integration model. . . . . 115
4.12 CMU VP default parameters and simulation conditions. . . . . 117
4.13 Preliminary strengths and weaknesses of the Analog and Digital CMU architectures. . . . . 125
4.14 RCSS VP packages and main configurations. . . . . 129

A.1 KEVCR parameters. All the values are taken at 25 °C. . . . . 168
A.2 KECA parameters. All the values are taken at 25 °C. . . . . 174

# Acronyms

**0-Series** *zero series.*

**ADC** analog-to-digital converter.

**AMR** anisotropic magneto-resistor.

**AMS** analog and mixed-signal.

**API** application programming interface.

**ASIC** application specific integrated circuit.

**AVG** Average.

**CAD** computer-aided design.

**CAE** computer-aided engineering.

**CHT** conjugated heat transfer.

**CMRR** common mode rejection ratio.

**CMU** current measurement unit.

**CPS** *Cyber-Physical Systems.*

**CRESTF** crest factor.

**CT** current transformer.

**DDF** dynamic dataflow.

**DSP** digital signal processor.

**DUV** design under verification.

**EDA** electronic design automation.

**EMC** electro-magnetic compatibility.

**EMI** electro-magnetic interference.

**ENOB** effective number of bits.

**ETRCM** electro-thermal Rogowski coil model.

**FDTD** finite-difference time-domain.

**FEA** finite element analysis.

**FEM** finite element method.

**FFT** fast Fourier transform.

**FIR** finite-impulse response.

**FIT** finite integration technique.

**FPGA** field-programmable gate array.

**FSM** finite state machine.

**GMI** giant magneto-impedance.

**GUI** graphical user interface.

**HPF** high-pass filter.

**HV** high voltage.

**ID** identifier.

**IED** intelligent electronic device.

**IIR** infinite-impulse response.

**IoT** internet of things.

**IP** intellectual property.

**IWP** *instant web publishing*.

**LPF** low-pass filter.

**LSB** least significant bit.

**MBD** model-based design.

**MC** Monte Carlo.

**MDD** model-driven design.

**MoC** *model of computation.*

**MoM** method of moments.

**MRF** *model registration form.*

**MV** medium voltage.

**O-model** circular system development model.

**OpAmp** operational amplifier.

**P2P** peak-to-peak.

**PBD** platform-based design.

**PDF** probability density function.

**PLM** product lifecycle management.

**PMP** *primary model parameter*.

**PN** process network.

**PSRR** power supply rejection ratio.

**RCSS** Rogowski coil sensor system.

**RMS** root mean square.

**RogoCoil** *Rogowski coil.*

**RVD** resistive voltage divider.

**SCU** self-calibration unit.

**SDF** synchronous dataflow.

**SHT** simplified heat transfer.

**SMP** *secondary model parameter*.

**SNR** signal-to-noise ratio.

**TBC** test bench configuration.

**TLM** transmission-line matrix.

**VCS** version control system.

**VP** *virtual prototype.*

**VSB** Voltage Sensing Block.

# Glossary

***abstraction level***

it refers to the level of details that are used to model a particular entity or system. In other words, the abstraction level indicates how accurate a *computational mockup* models the real entity/system. Since there can be an undetermined amount of abstractions for a model, it is unpractical to exactly distinguish between models with different abstraction levels. Instead, this work considers 3 main level classes, from less to more refined abstractions: *functional models*, *behavioral models*, and *physical models*.

***back-annotation***

this is the process of retrofitting some or all parameter values of a model, after their measurement using the physical implementation of the system/component that is represented by such model.

***behavioral model***

a medium-level abstraction that describes the functionality of a system by taking into account some non-idealities and critical variables that affect the system behavior.

***component model***

it is a model of an individual well-defined entity/component that composes a system. A component model can also be hierarchical in nature, i.e. it can be composed of other more basic component models.

***computational mockup***

a digital representation of a physical entity, system, or phenomenon. It can also be a computational model that executes a certain algorithm.

***conditioned model***

a conditioned model must be understood as the type of model which offers a high degree of trustfulness: it is clear and well-written, it is properly documented, it is accompanied by at least one verification proof (e.g. a test bench) of its functionality, and possibly some behaviors according to its refinement level.

***Cyber-Physical Systems***

a combination of software, hardware, and physical processes which together form a complex multi-domain system with feedback loops where physical processes affect software computations and vice versa.

***design space exploration***

it refers to the activity of discovering and evaluating potential system architecture topologies that can be implemented to perform certain functionality. The design space exploration involves the exploration of different system components, the functional and performance evaluation of the studied architectures, and the selection of the best system architecture according to system requirements and specifications.

***functional model***

a high-level abstraction that mainly describes the functionality of a system.

***functional space***

represents the list of functionalities, specifications, requirements, conditions, and constraints of a specific system or component.

***hierarchical parameter binding***

this is a VHDL-AMS modeling technique which consists on binding all the generic parameters of the components of a hierarchical *component model* or VP to the highest level design entity, normally the *test bench*. This is done by forming a hierarchical structure in which each generic parameter of every component in the design is declared and bound in the subsequent higher level design entity via generic mapping in the component instantiation. This modeling technique is compatible with all types of simulation analysis but is only recommend for models with a low number of generic parameters.

***instant web publishing***

it is a practical method to publish a database on-line using FileMaker server. This method is compatible with the most popular web browsers in Windows, Linux, and MAC computers. All that is required is an internet or an intranet connection to view, edit, sort, or search records in the published database, according to the different access privileges that may have the users.

***instrument transformer***

they are high accuracy class electrical devices used to isolate or transform voltage or current levels. The most common usage of instrument transformers is to operate instruments or metering from high voltage or high current circuits, safely isolating secondary control circuitry from the high voltages or currents. The primary winding of the transformer is connected to the high voltage or high current circuit, and the meter or relay is connected to the secondary circuit.

***model characterization***

the model characterization consists on modifying or including new parameters in the model in order to correct the differences between the simulation and the experimental results.

***model implementation***

it is a specific way to implement the functionalities, behaviors, and interactions of a model. From a VHDL-AMS point of view, a model implementation is a design entity (entity + architecture). A model interface (an entity) can be implemented in different ways by using different architectures.

***model of computation***

it is the definition of the collection of authorized operations used in computation and their respective costs. These set of rules can be used to classify types of models. The complexity of an algorithm in memory space and/or execution time can be measured by assuming a certain model of computation; additionally, it is possible to analyze the computational resources required or to discuss the limitations of algorithms or computers.

***model refinement***

the process of lowering the *abstraction level* of a model, e.g. from purely ideal model descriptions down to physical and detailed descriptions that model the system/component more accurately.

***model registration form***

it is a light copy of the VP-Model Library which contains few model entries as examples, and database writing privileges for contributors. Model registration forms are used for both submitting modeling requests and for collecting all type of *modeling elements* and their respective documentation.

***model scalability***

the scalability of a model refers to the model capability to become more complex, either by using the model in a larger model or by *model refinement*.

***model signature***

the signature of a model is defined by the amount and the type of parameters and ports (i.e. input, output, or bidirectional) in a model. It is similar to the type signature of a function.

***model validation***

comparison of experimental results against simulation results. A model of a component or a system is validated when the simulation results are consistent with the experimental data. The simulation results and the experimental data do not need to exactly match for model validation. Slight corrections can be performed to the model, this is called *model characterization*.

***model verification***

this is the process of establishing the correct functionality, behaviors, performance, and reliability of a model. In other words, it is the process of confirming that the model meets the desired specifications and requirements under a set of assumptions initially given. This can be seen as the virtual testing of the system/component which is represented by a *computational mockup.*

***modeling element***

a modeling element is a piece of instructions and specifications that are required to build models and their virtual verification environments using the VHDL-AMS *simulation framework.* The different types of modeling elements are *component models, virtual prototype*s, *test benches, test cases,* and *simulator setups.*

***modularity***

is the property that makes possible the inclusion of new behaviors in a model by coupling with the preexistent behaviors and functionality. A high modularity implies a minimum effort for including new behaviors.

***physical model***

a low-level abstraction that describes the functionality and the behavior of a system in a very accurate way. These type of models are the closest descriptions to the entities that are modeled. They normally consider the intrinsic physical phenomena and the physical interactions of the entity with the environment.

***platform***

in platform-based design (PBD) theory, a platform is defined as a library of components that can be assembled to generate a design at a certain *abstraction level.* The designs on each platform are represented by platform-specific design models.

***primary model parameter***

this is a constant parameter that is available at the interface of the model with the purpose of modeling certain behavior and/or functionality. This type of parameter does not depend on other internal parameters inside the model. In order to make it accessible from a top level design unit, it must be expressed as a VHDL-AMS generic parameter in the entity declaration of the model.

***refinement level***

*abstraction level.*

***Rogowski coil***

a Rogowski coil is a toroidal coil wrapped around a non-ferromagnetic core. The Rogowski coil can be used as an electrical instrument for measuring alternating current; for that purpose, the

Rogowski coil is located around a primary conductor (also known as Busbar) whose current intensity is intended to be measured. It delivers a voltage proportional to the derivative of the measured current.

***secondary model parameter***

this is a parameter which its value is derived or computed from other primary or secondary model parameters. A secondary model parameter is normally expressed as a VHDL(-AMS) constant inside the model architecture, it is used to model particular functionalities and/or behaviors.

***simulation condition***

this is a particular condition fixed by one or multiple simulator parameter values and/or directives related to the type of simulation, the simulator integration methods, and other related simulator settings affecting the numerical precision and accuracy of the simulation result.

***simulation framework***

the set of simulation tools, modeling languages, and related infrastructure that allows to create, manipulate, and simulate computer-based models.

***simulator setup***

it is a set of specifications that the simulator tool uses for executing a particular type of simulation/s. The simulation results directly depend on these specifications, which are critical for simulation performance and accuracy. These specifications are commonly gathered in a simulator file (e.g. the .pat file in SMASH), some of these specifications are tool-independent, such as the maximum time step (Hmax) for time-domain simulations and the integration method algorithm (e.g. Trapezoidal, Backward Euler, etc.).

***switchgear***

a switchgear is the combination of electrical disconnect switches, fuses or circuit breakers used to control, protect, and isolate electrical equipment. A Switchgear is used either to de-energize equipment for maintenance or to clear faults downstream. This type of equipment is directly linked to the reliability of the electricity supply.

***technology space***

a limited selection of all possible technologies that can be used for a specific application.

***test bench***

it is a verification model that recreates the operation conditions of a particular model that is subject to verification, i.e. the design under verification (DUV). The test bench provides to the DUV its inputs. It also might provide parameter values to the DUV, and/or additional algorithms for post-processing the output signals of the DUV.

***test case***

it is the virtual verification structure that is created for the design under verification (DUV). A *test case* of a DUV is composed by a particular *test bench*, a specific DUV - *test bench* customization environment (i.e. particular parameter values and component interconnections) organized in one or several VHDL-AMS packages and configurations, and the set of simulator directives. A *test case* represents a specific testing scenario. A *test case* can be focused for verifying either a *component model* or a particular configuration of a large and complex VP.

***verification condition***

specific parameter values that represent real design conditions of the system and/or its components. Environmental and multi-domain parameters are also part of the verification conditions when the effect of the physical phenomena is properly included in the model.

***virtual prototype***

a modular and hierarchical computational model that describes the functionalities and behaviors of a complex system including the interaction within its environment. A virtual prototype is often composed of several *component models* at different abstraction levels.

***virtual prototyping***

it is the use of modeling and simulation techniques for the construction, verification, and validation of VPs and all its related models. It makes use of any kind of CAD and/or CAE software to model, simulate, analyze, and verify a system or part of it from concerned life-cycle aspects such as design, optimization, testing, and operation as on a real physical prototype.

***zero series***

this is the first final system/product that is fabricated before mass production and commercialization. The 0-Series is used for the final operational and environmental tests of the system/product.

# 1 Introduction

## 1.1 Context and problem description

Constant improvements in technology, science, and engineering have allowed building increasingly smarter and complex systems in almost any technical and scientific domain. These new type of heterogeneous systems, better known as *Cyber-Physical Systems* (CPS) are, by definition, combinations of software (i.e. algorithms for control and execution), hardware (e.g. embedded electronics, mechanical parts, sensors and actuators), and physical processes (i.e. thermal, chemical, optical, mechanical, electrical, etc.) which together form a system with feedback loops where physical processes affect software computations and vice versa [1]. The implicit bidirectional cyber-physical coupling must be steadfastly controlled, adjustable, accurate, and predictable, offering exceptional opportunities for new services and applications in our society [2]. Furthermore, CPS techniques and tools offer a tremendous opportunity to improve classical systems such as vehicles, buildings, and airplanes, and create a new generation of intelligent and autonomous systems. However, the increasing complexity of the components and the use of more advanced technologies in every domain pose a major challenge to ensure safety and reliability within a limited developing time and reduced cost [3, 4].

In the last decade, ongoing research on CPS has been carried out for multiple industrial applications and using different approaches to cope with the design and verification of complex systems of systems. Most of the efforts in the subject are oriented to large scale CPS, which are those large systems in which the elements of computation, control and networking domains (i.e. the *cyber* elements) are strongly interrelated with the physical processes and human interaction. Some of the most important examples of large CPS are: *critical infrastructures* such as smart grids, smart buildings, security systems for terrorism or extreme weather conditions; and *transportation systems*, such as smart vehicles and aircrafts. Additionally, CPS can also be of smaller scale, such as robotic platforms, wearable and implantable medical sensor systems for healthcare, and smart sensors and actuators for power applications. Nonetheless, all CPS systems share the same challenges and problems emerging from the heterogeneity of both components and interactions regardless the size of the system. The divergent intellectual and technical traditions of the diverse engineering fields make the fast assessment of design alternatives an outstandingly hard task [5].

Traditional system design and development approaches have been developed to deal with complexity, such as the well-known V-model shown in Figure 1.1. The classical V-model, which is based on early system decomposition and later construction, has been used for years in several types of industries, from semiconductor to mechatronic design. Each subsystem is first designed and optimized by different designers/groups or even different companies; later the subsystems are connected together in a heuristic and ad hoc way, this is conventionally done in CPS consortia, for example in automotive and avionics vertical supply chains [6]. The early decomposition encourages to divide the system into smaller subsystems at the earliest stage of the design. The method to cope with the system's complexity consists in designing, verifying, and optimizing independently the subsystems by their respective domain of expertise. For example, for industrial automation systems, the division is often done by areas of expertise such as sensors and actuators, power management, analog and digital electronics, embedded software, communication and networking, etc. Later on, the subsystems are physically fabricated, tested, and later integrated to build the complete system, see Figure 1.1. However, as the systems are getting gradually more complex, this approach is becoming less efficient. A very early system design partition does not contribute to the analysis and understanding of the interactions among the subsystems of different domains; and therefore, producing great difficulty to integrate complex parts and leading to design errors and sub-optimal designs.

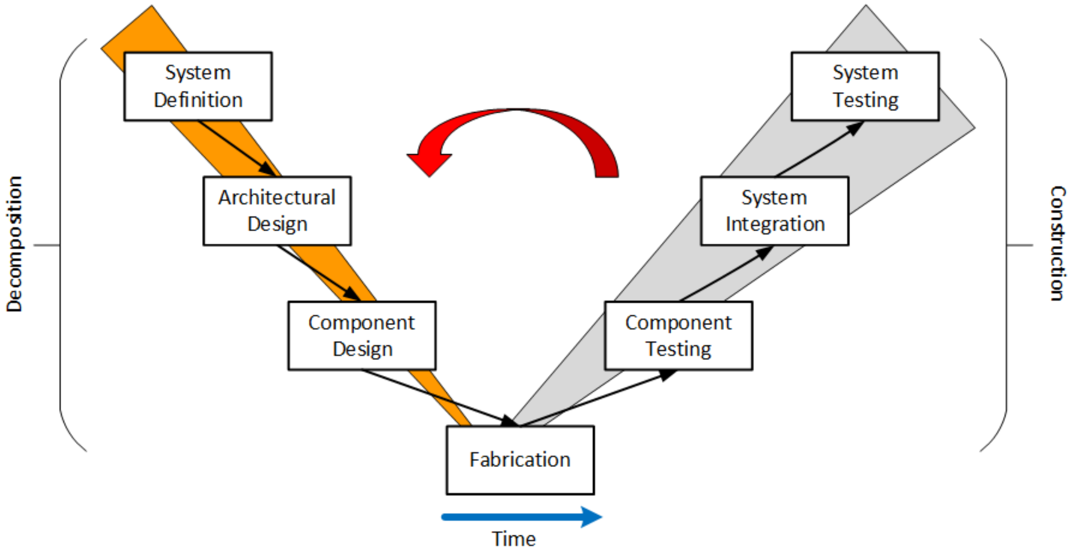


Figure 1.1: Classical V-model diagram.

Multiple non-desired iterations from the system integration stage to the former design stages are required when design errors are detected at advanced stages. The problem can even be worst in terms of safety and costs when the design errors are not detected in the system testing stage, see Figure 1.1. Faulty products could be manufactured and commercialized, this represents high financial losses for the companies.

Among the plethora of techniques and methods that can be applied to CPS design and verification, this work is focused on the use of computational models and simulation. This approach has been called differently in the literature: model-based design (MBD) [7], computer-aided design (CAD) [8], model-driven design (MDD) [9], computer-automated design [10], computer-aided engineering (CAE) [11]. We refer to this strategy as ***virtual prototyping*** or *VP-based* design. The core of the issue is not exclusively about developing new tools for CPS design and system integration, although they are key to make progress in the state of the art in CPS design and verification, it is the comprehension of design principles, system requirements, and capabilities of the modeling languages, what it is required to change the classical system design methodologies and the dynamics of the system integration [6].

Classically, modeling and simulation techniques have evolved independently as they have different specific goals for design and verification in particular domains and applications. For example, digital hardware description modeling techniques used for application specific integrated circuit (ASIC) design [12] are quite different than the modeling techniques used for gas turbine design [13]. This poses a major challenge for a unified system design approach using an integrated framework that properly addresses all the needs of the design of a complex system including its subsystems. Consequently, a complete architectural analysis and virtual verification of the entire system is not undertaken as it is viewed as too time-consuming and impractical. However, since companies involve with CPS development are struggling to guarantee safety and reliability of every time more complex multi-domain systems, it is essential for these companies to perceive an adequate design methodology and tools on their critical path. The design of next generation systems requires a correct understanding of the role of high-level abstractions in the development process and the current capabilities and limitations of simulation tools and modeling techniques [14]. The analysis, optimization, and verification of CPS demand robust and effective *virtual prototyping* methodologies and simulation tools to reduce the development time and costs and improve safety.

## 1.2 Main objectives and rationale

In this thesis, the author presents a conceptual system development model, called the circular system development model (O-model), which considers all the stages of the development of industrial CPS, and in particular, describes how to use a systematic VHDL-AMS *virtual prototyping* methodology for improving the design and verification of small-scale CPS for power automation applications. The proposed methodology, hereafter called the **VP-based design methodology**, consists of a set of modeling and operational elements that are used to allow and improve the *design space exploration* of the complete system, early system integration analysis and early error detection (especially in the component interfaces), verification and optimization at the system level. It is not the same analyzing and optimizing a system as a whole from the earliest stage of the design than interconnecting a set of individually optimized components to compose a system at a later stage of the design.

This work describes and details two main operational elements of the VP-based design methodology that are of great importance to achieve the aforementioned objectives:

**The VP-Model Library:** It is a centralized database for storing, documenting, maintaining, and queering *modeling elements* at different abstraction levels. Its main goal is to provide an environment for maximizing the reuse of models among system and subsystem designers. Although the VP-Model Library supports the inclusion of models written in different languages, its structural organization is mainly focused on supporting VHDL-AMS models.

**The VP-Modeling Guidelines:** They are a set of VHDL-AMS modeling recommendations and techniques that are focused on supporting the creation of complex hierarchical models —*virtual prototype*s (VPs) —and their verification models —*test benches*—. These guidelines are based on the application of a **meet-in-the-middle** approach for conducting **system-level modeling** and simulation. It is shown how these guidelines improve the **modularity** and **scalability** of the models. During this research, multiple VHDL-AMS techniques were explored and evaluated using different design approaches which are suitable for particular design and verification cases. However, the modeling guidelines presented in this document are only addressed to system designers who wants to perform *design space exploration* and **gradual model refinement**.

## 1.3 Focus and application

The concepts and techniques proposed in this research are focused to small-scale CPS in the power automation sector. The interest of this work has been specially dedicated to the design of smart electrical sensors and actuators for measuring and protection of electrical power generation and distribution systems. Adding smart features such as self-calibration, autonomous failure and wearing monitoring to these type of systems contributes to a significant overall improvement of the energy efficiency and cost reduction in power infrastructures. The VP-based methodology has been the result of extensive modeling and simulation activities for industrial products and systems in the power automation sector. Specifically, the author of this thesis has contributed in *virtual prototyping* of complex heterogeneous systems such as the ablation monitoring system for high-voltage generator circuit breakers [15], electrical voltage and current sensors based on electro-optical voltage transducers [16] and Rogowski coil transducers [17] for medium voltage (MV) and high voltage (HV) applications.

Particularly, this thesis describes the *virtual prototyping* of a *Rogowski coil* (RogoCoil) electrical sensor system for current measurement in the MV range. The main purpose of this application example is not the sole design of the sensor system, but the identification of the issues, challenges, and requirements of the proposed *virtual prototyping* approach for small-scale CPS design and verification. In this way, the presented *Rogowski coil* (RogoCoil) sensor system example illustrates a very refined multi-domain modeling of the RogoCoil subsystem using combination of geometrical finite element analysis (FEA) and VHDL-AMS models. Additionally, the design of the other parts of the system and the final VP of the complete RogoCoil sensor system is shown to illustrate the system-level modeling that is advocated in the proposed methodology using higher *abstraction level* models and gradual *model refinement*. This *virtual prototyping* example is used to clarify concepts given in the proposed system development stages for virtual design and verification using the O-model in chapter 5.

## 1.4 Motivation for system-level modeling

In order to properly model the interactions of the system components, it is required to use *component models* with correct interfaces that accurately represents the most critical variables and behaviors among the components and their environment. However, a large number of variables and component interconnections obstruct the design, simulation, maintenance, and re-utilization of both simple and complex *component models* in a large VP. Additionally, the computational burden of simulating large models can be very high, and consequently time-prohibitive. This issue can be faced by rising up the *abstraction level* of the models; and therefore, reducing the complexity of the whole VP. This can be very convenient at the early stages of the design since it enables the system designers to perform functional verification at the system level. These type of simulations are useful for two principal purposes as follows:

**System design:** The design of a system implies obtaining an interconnection of components (system architecture) which performs the desired functionality under a set of constraints and conditions. Indeed, this is defined as the ***design space exploration*** activity.

**System verification:** This is the process of establishing the correctness of the design. Commonly, verification and validation are taken as separate concepts, where verification activities are focused to proof whether the system fulfills the given specifications (e.g. the accuracy and precision level of a sensor), and validation activities are concentrated to corroborate that the behavioral and other non-functional properties of the system meet the requirements and expectations [18] (e.g. the correct operation of a sensor under a certain temperature drift and/or electro-magnetic noise level).

Since the purpose of *virtual prototyping* is to verify the correctness of the functional and non-functional aspects of the system design (i.e. the fulfillment of requirements, specifications, and proper operation in the application context), in this work the verification concept involves the traditional definition of both verification and validation as it is previously mentioned. In fact, functional verification is the first type of verification that must be executed in any design at the system level, its main objective is the proof of the concept. Afterwards, verifications of the system performance, critical behaviors, cross-domain interactions, and more detailed aspects of the design should be gradually performed until obtaining a high coverage. Indeed, low-level model abstractions, which model the behaviors and physical interactions of the system components, are required to obtain a complete set of specifications by performing more detailed verification procedures to cover the critical variables in a design. Unfortunately, modeling and verifying all the dimensions and behaviors of a complex system is an intractable problem due to its intrinsic complexity.

In spite of the limitations that modeling and simulation techniques present for achieving a detailed system verification of all the aspects of a design, *virtual prototyping* approaches currently offer the most practical and affordable way to design and verify complex systems, especially for companies which still relies on classical physical prototyping for the *design space exploration.* This is why one of the main motivations of this research is the amelioration of current *virtual prototyping* approaches using the state-of-the-art *simulation frameworks* for multi-domain system design. The proposed VP-based design methodology encourage the utilization of high-level functional model abstractions; and later, a progressive model refinement for modeling and verifying the critical values of the design. This is done by implementing well-defined verification procedures (e.g. worst-case, induced failures, statistical analysis, etc.), based on different types of simulations, such as transient, frequency, parametric, and Monte Carlo (MC) simulations. One of the main advantages offered by analog and mixed-signal (AMS) description languages is the capability to use event-driven (digital) abstractions to represent continuous-time (analog) interactions, this is especially useful to speed-up the simulation performance in complex VPs.

High-level abstractions can be a very vague term, specially when we consider a large hierarchical VP whose main components can also be hierarchical models. The design of these models is not trivial and their outcome depends on the interconnection context. High-level abstractions are often related to the functionality of the system that is being considered in that moment at certain level of the system architecture hierarchy. For instance, a functional model of an Analog-to-Digital Converter (ADC) can just transform an analog quantity to a digital bit vector signal without modeling some details such as the conversion algorithm, time-delay, bandwidth or signal-to-noise ratio. However, if we consider a greater system such as a robot, which can use a big amount of ADCs associated to its multiple sensors, the ADC functional model is a very low-level abstraction that might not be suitable for functional modeling of the complete system. In this way, high-level abstractions at system, subsystem, and component level must be consistent with the functionality and behaviors that we pretend to model.

## 1.5 Document organization

This thesis document is organized in a total of six chapters and six appendices that include supporting material to the content of the chapters. Chapter 1 explains the context, the main issues, goals, motivations, and the application of this work. Chapter 2 makes a review of some of the most important contributions to the state of the art in modeling and simulation methodologies for CPS design and verification. In particular, it is included the theoretical concepts which have inspired the proposed methodology. Additionally, it is also mentioned formal verification techniques that can be used together with the proposed methodology in a future work. Chapter 3 describes the VP-based design methodology foundations and related theoretical concepts. The elements of the VP-based design methodology are detailed, in particular, the VP-Model Library, and the VP-Modeling Guidelines. Chapter 4 illustrates the RogoCoil sensor system example, showing details about the electro-thermal *virtual prototyping* of the RogoCoil system and the VHDL-AMS system-level modeling of the self-calibration and current measurement units of the RogoCoil sensor system. The VP of the complete RogoCoil sensor system is presented at the end of the chapter. Chapter 5 describes the proposed circular system development model (O-model) making emphasis in the earliest stages of the system development: the system definition, the system-level design, the subsystem-level design, and the final system verification using the VP. Finally, in chapter 6 is presented a brief discussion of the main conclusions of this work and future work.

# 2 State of the art

This chapter contains a review of the main contributions about CPS design methodologies based on modeling and simulation. Most of the ideas contained in this chapter have been used as inspiration for the VP-based design methodology proposed in this thesis. The final goal is to cope with system complexity and improve the system safety and reliability. Since these issues have been historically treated by different types of domain experts, there are a plethora of modeling techniques, languages, and simulation tools with a large number of application examples in many scientific and technical domains. The purpose of this literature review is not to provide a complete historical summary of all the approaches and contributions, but rather list the most relevant techniques and methodologies that are directly and indirectly related to this thesis.

## 2.1 The V-model

The V-model is the most popular design approach that has been applied for system development (both hardware and software), mainly in defense and transportation industries. It allows to conceptually deal with the complexity of the complete development process, from the system conceptualization to the final system testing and validation, see Figure 1.1 on page 2. It was originally developed by the German company iABG[1] for defense applications. This model graphically represents, in a V-shape diagram, the system development in sequential phases along the time. The left-hand of the V is a decomposition process in which the system requirements and specifications are defined and the system is conceptually designed. Traditionally, the design process for complex multi-domain systems is split into several modules or subsystems according to the application domain (i.e. electrical, mechanical, control software, embedded electronics, etc.), this work is carried out in parallel by different teams of specialists. The partitioning of the system by application domains and its natural parallelization have been erroneously considered in the industry as an optimal design approach in which

---

[1] See: http://www.iabg.de/en/corporate-profile/

designers are expected to come up with physical prototypes that must be integrated with other subsystems with zero design errors. However, the premature division of the system at the early stages of the design implies a division of the design space; and therefore, from an epistemic standpoint, we are limited to explore an optimal architecture of the whole system. Moreover, there is no guarantee that the multiple cross-domain variables and interface interactions of the subsystems are correctly taken into account when the system is built and designed in separated parts. A priori, it is not the same to compose a system from modules which have been optimized separately than optimize the complete architecture of the system as a unique design.

The last part of the V-model approach, the right-hand of the V, is dedicated to the system integration, testing, and validation. Following the hundreds of concurrent design processes carried in parallel by multi-domain and cross-organizational design teams [19], CPS integration becomes a challenge in the industry. System safety and reliability procedures demand rigorous and time taking testing processes to verify and validate the correct operation of the system. Any design error detected in this phase implies a re-design and re-implementation of the affected modules, consequently, the development time and cost can be very high. By applying the traditional V-model approach for complex multi-domain system design and development, the industries struggle to meet the schedule and budget [20, 21].

## 2.2 Layered Design

As it has been described in [19], this approach copes with the complexity by focusing on those aspects that are pertinent to support the design activities at the corresponding abstraction level, i.e. layered design decompose the complexity of the systems “vertically”. This approach is especially useful when the details of the lower abstraction layers can be included when the design is performed at a higher level of abstraction. Although the layered design is well understood and standard in many application domains, it is essential to have clear connections between the different abstraction layers for an effective implementation. Some of the most popular examples are the AUTOSAR standard [22] for the automotive industry and the ARINC standard in the avionics application domain [23].

The AUTOSAR layer structure allows the designer to completely separate the logical architecture of a specific application. For instance, a number of automotive logical functions in areas such as powertrain, safety, multimedia/telematics, and body/comfort, can be decouple from the specific hardware components thanks to the so-called *virtual functional bus*, which allows to interconnect those functions represented as a set of components, irrespective of the actual physical hardware, i.e. the available ECU and network topology [22]. This exemplifies the important role of the abstraction layers: they allow designers to completely focus on the functionality and logical operation of the application by using abstractions, while at the same time imposing minimal (or even no) constraint on the design space of possible hardware architectures [19].

The problem of layered design approach remains on the dilemma of completely supporting hardware independent abstractions, whilst at the same time requiring to perform analysis of properties which implicitly depend on it. For example, suppose that we need to verify the correct functionality of the system considering hardware delay times or wrong value calculations (failure situation). Including the desired non-ideal effects or failure situations as abstract behaviors which can be produced by using one or more sets of hardware components and architectures can be a practical solution to this dilemma. These vertical assumptions can be checked by contract implementation, see section 2.7.

## 2.3 Component-based Design

This approach aims to reduce the complexity by assembling strongly encapsulated design entities called “components” provided with well-defined interface specifications [19], i.e. ports and parameters. Contrarily to layered design approach, the component-based design reduces the complexity of the system “horizontally”, this is achieved by dividing the complete design of the system into smaller subsystems that can be connected together with the lowest possible effort. The efficiency of the approach relies on two key features that are often mutually exclusive, on one side it is desired to count with modular interfaces that allow connecting multiple components together in a practical way, on the other side it is desired to maximize the potential for re-use of any component in different deployment contexts. According to [19], the first condition has been classically achieved by using components with “small interfaces”, i.e. interfaces which are both small in terms of number of interface variables or ports, as well as “logically small”, i.e. when protocols governing the invocation of component services have compact specifications not requiring deep levels of synchronization. On the other hand, the second condition is inherently expressible in terms of interface specifications, where re-use can be maximized by finding the weakest assumptions on the environment enough to set the guarantees on a given component implementation. However, we believe that component re-usability is not only about sufficient interface specifications with the environment, we can always make assumptions to use specific component models. The major issue for component re-usability is the lack of mechanisms to establish a proper and pragmatical IP trustworthiness [24], where the actual designer need to gain trust in the validity of the results given by a foreign or native model for a specific task.

Component-based design for embedded systems and CPS faces three main challenges. Firstly, it is required to provide rich enough interface specifications to cover all phases of the design cycle. Classical component interface models are static entities that only receives and provides specific information at a certain design level. Purely functional characterized components can only be used for limited functional verification so that a complete virtual integration and testing is hardly accomplished. Secondly, there is a natural trade-off between generality and efficiency in component implementations. On one side we require component implementations which allow us to verify the functionalities of the system, often under normal operation conditions, on the other side we want to verify the robustness and the safety of the system, i.e to verify the functionalities under critical operation conditions. Classically, a component implementation able to support simulations under any possible condition is not efficient, and therefore, its utilization in a bigger complex system will be even more complicated and restricted. Finally, if the complete system consists of a large number of components, a manual component interconnection can be overwhelming and tedious. Automatic synthesis of system architectures from high-level functional descriptions has been proposed for automotive CPS [25]. Although this methodology solves the component interconnection problem in big complex systems, it relies on a well-defined and reliable library of components and architectures to support the automatic synthesis only at the functional level of abstraction. There are no clear links between the system integrator view and the domain specialist view.

## 2.4 Model-based Design

Model-based design (MBD), also called model-driven design (MDD), has been generally undertaken as a crucial strategy to cope with complex system design due to its capabilities to support early requirement verification and virtual system integration [19]. Computer modeling and simulation has been used for a long time in almost every scientific and technical domain. However, its scope has been limited to conceptual verification rather than a true design technique, perhaps with the exception of one application domain, electronic hardware design. The first scientific report of computer-automated product design dates from 1963 [26], who created a computer program able to determine suitable logic circuits satisfying certain hardware constraints while at the same time evaluating the ability of the logics to perform character recognition. Model-based approaches have been applied for years in the digital hardware domain, where we can certainly assure that this approach has had a great success since the introduction of automatic logic synthesis [6]. However, the same digital automation principles cannot be applied easily to other application domains that are analogue in nature, such as electrical or mechanical systems.

There are some methodology proposals aiming to modify the V-model approach by using MBD to introduce different variations of the system development phases in order to support early requirement verification and virtual system integration [21], [27]. However, the fundamental issues related to MBD still remain a barrier for an effective methodology implementation. It turns out that there are almost as many design languages and simulation tools as applications; and therefore, the model compatibility issues and platform-dependencies are problems that have been making more difficult the system integration task, especially between cross-domain applications. Nevertheless, around the plethora of simulation frameworks (i.e. the set of software tools, modeling languages, and methodologies), there are interesting simulation tools and high-level modeling languages that allow performing system verification and virtual system integration from subsystems at a particular level of abstraction, rather than "physically" integrate the system after their implementation at the right-side of the V-model. In this way, such virtual integration permits discovering potential integration problems early on, at the initial phases of the design.

One of these interesting concepts for CPS design is functional modeling and verification [25]. The architectural design space exploration can be facilitated by rising the model abstractions using very high-level functional descriptions in a “Functional Basis Language”. The idea is to express what the system does without any details of the implementation or deep knowledge about the architecture of the subsystems. In a functional model, specialized knowledge is irrelevant and discouraged. The challenge of the approach is to be able to automatically select architectures from functional descriptions, for this purpose it is required to have a reliable and well-defined library of components and architectures previously created. For instance, this approach would allow a software engineer, with marginal knowledge in mechanical engineering principles, to design and engine control system. However, in order to successfully implement this type of design methodologies in current CPS industries, it is required to properly define the path between functional system level models with domain-specific models which are closer to implementation and are more reliable for verification. Additionally, the compatibility issues of modeling languages and simulation tools must be harmonized to cope with heterogeneous system models and permit an effective virtual system integration.

## 2.5 Modeling languages and tools

MBD has inspired a bunch of modeling languages and software tools, they have evolved from previous existing programming languages such as C, C++, SML, and also from hardware description languages and circuit simulator netlist languages such as VHDL, Verilog, SPICE among others. The current state-of-the-art of modeling languages and tools includes a rich amount of options that have been developed by different vendors and application domains. A good language classification review for complex heterogeneous system design for micro-electromechanical systems (MEMS) applications is given by [21]. In this review, the author highlights the digital modeling languages and their analog extensions such as VHDL-AMS, Verilog-AMS [28] and SystemC-AMS [29] as the basis of complex heterogeneous system design. Nevertheless, the main weakness of these languages is the slowness of low-level simulations. This issue is overcome by using circuit simulators such as FAST-SPICE [30]. However, since there is no standardized SPICE language, it is not expected a great language support from this type of circuit simulators. We can add to this review an important simulation modeling technique that is widely used in MEMS and other multiphysics applications, it is the Finite Element Analysis (FEA) simulators such as COMSOL[2] and ANSYS[3]; which can be used to simulate complicated geometries that cannot be modeled using equation-based models based on mixed-signal modeling languages. Co-design techniques using both approaches have been reported in [17] for high-voltage electrical transducers, more sophisticated approaches involving modal decomposition techniques to perform automatically lumped parameter extraction from finite-element models are reported in [31] for MEMS applications. Commercial software tools for system level design able to support multiple hardware description languages

---

[2] https://www.comsol.com/

[3] http://www.ansys.com/

and co-simulation frameworks has been gaining popularity in the industrial automation domain [32], SMASH[4] of Dolphin integration and System Vision[5] of Mentor Graphics are able to support VPs including models written in different languages such as VHDL-AMS, Verilog-AMS, SystemVerilog and SPICE; additionally, it also support software descriptions written in C and C++. Therefore, these are suitable tools for a broad range of applications such as MEMS, embedded systems and small scale CPS.

On the other hand, a good summary of the modeling languages and tools used for CPS design for mechatronics applications is given in [19]. We can mention some of the current commercial and open modeling languages and tools such as SysML [33] and AADL [34] for system level modeling, Catia/Dymola[6] and Modelica [35] for physical system modeling, Matlab[7] and Simulink [36] for control-law design and general purpose modeling, and UML[8] [37], ANSYS Scade[9] and Simplorer[10] for MBD embedded software development and multi-domain system modeling. The state-of-the-art in MBD includes automatic code-generation, simulation coupled with requirement monitoring, co-simulation of heterogeneous models such as UML and Matlab/Simulink, model-based analysis verification of compliance of requirements and specification models, model-based test-generation, rapid prototyping and virtual integration.

Modeling languages play fundamental roles in MBD and virtual integration, these roles can be divided into three categories [38]:

- *Unified (or universal) modeling languages*, which are general purpose languages focused on offering designers the advantage of remaining in a single language framework, independent of the application domain.
- *Interchange languages*, which permit model sharing across analysis tools (hybrid system analysis). Interchange languages are optimized for providing specific quantitative analysis capabilities in design flows by facilitating tool integration. One can argue that the interchangeability property of a language can be maximized if the language is made an industrial standard. Therefore, the models will be platform-independent and several tools can process the model for different purposes.
- Domain-specific modeling languages (DSMLs) specify a design platform, including the concepts, relationships, and well-formedness constraints linked to the application domain they address. They are optimized to be focused, i.e. the modeling language should offer the simplest possible formulation that is sufficient for the modeling task.

Unfortunately, there is no single “Universal” modeling language that fills all roles. It would

---

[4] http://www.dolphin.fr/index.php/eda_solutions/products/smash/overview

[5] https://www.mentor.com/products/sm/system_integration_simulation_analysis/systemvision/

[6] http://www.3ds.com/products-services/catia/

[7] http://www.mathworks.com/products/matlab/

[8] http://www.uml.org/

[9] http://www.esterel-technologies.com/products/scade-suite/

[10] http://www.ansys.com/it-IT/Products/Systems/ANSYS-Simplorer

be very useful to count with one language that allows making both system level modeling and domain specific modeling in an optimal way, however difficult and cost prohibitive to achieve. Since DSMLs and tools focused on specific applications has been evolved separately, abstraction layers and platforms have been made using different languages and tools historically. In MBD today non-functional aspects such as performance, timing, power or safety analysis are typically addressed in dedicated specialized tools using tool-specific models, with the entailed risk of incoherency between the corresponding models, which generally interact [19]. One of the most popular initiatives developed to mitigate this problem is the so-called Functional Mockup Interface (FMI) standard. The FMI is a tool independent standard for the exchange of dynamic models and for co-simulation. The development of FMI was initiated and organized by Daimler AG within the ITEA2 project MODELISAR. The primary goal is to support the exchange of simulation models between suppliers and OEMs even if a large variety of different tools are used. The FMI was developed in a close collaboration between vendors of Modelica tools (such as AMESim, Dymola, SimulationX) and non-Modelica tools (such as SIMPACK, Silver, Exite), as well as research institutes [39]. The goal of the Model Exchange interface is to numerically solve a system of differential, algebraic and discrete equations to allow discrete and continuous model co-simulation in different tools.

Another major initiative developed to counteract the incompatibility risks of using diverse proprietary modeling tools with several DSML is metamodeling. The use of metamodels enables to encompass multiple views of design entities, enabling co-modeling and co-analysis of heterogeneous models. Metamodels specify the set of valid models that can be determined with a particular modeling language and behavior in a specific domain. Metamodeling in the semantic context is an approach to consistently abstracting away *model of computation* (MoC) specificities while enhancing MoC similarities in the semantics metamodel. The results obtained by metamodeling allow to analyze and design complex systems without abandoning the properties of the components' MoCs. This metamodeling notion allows to evaluate and compare different MoCs, use mathematical frameworks to prove design properties, and encourage platform-based design (PBD), see section 2.6. Likewise, metamodeling forms the basis of several actor-based design [40], [41] environments such as Ptolemy II [42] and Metropolis [38], [43]. It is worth mentioning other relevant projects that make use of metamodeling concepts for developing heterogeneous design frameworks such as MARTE UML[11] [44] for real-time system analysis, the SPEEDS[12] project and its HRC metamodel [45] in the field of embedded systems for avionics applications, and the CESAR project[13] and its improved common metamodel (CMM) which implements the design-by-contract paradigm enabling modeling of complex heterogeneous systems and formal specification of requirements. This metamodel forms an integral part of the metamodel-based interoperability concepts of the reference technology platform developed under [46]. Similarly, the Vanderbilt university group has evolved a simulation framework and methodology, the OpenMETA tool suite and particularly the CyPhyML integration model language, for the Defense Advanced Research

---

[11] http://www.omg.org/omgmarte/

[12] http://www-verimag.imag.fr/SPEEDS.html

[13] http://www.cesarproject.eu/

Program Agency (DARPA)'s Adaptive Vehicle Make (AVM) program. The OpenMETA tool suite uses semantic constructs for model integration across engineering disciplines and facilitates trade analysis across those disciplines (such as electrical, mechanical, thermal, fluid, and cyber) [47].

## 2.6 Platform-based Design

PBD is a powerful concept that has been developed for decreasing the time-to-market and manufacturing costs in the semiconductor industry. PBD has been exploited for several years in the design of personal computers and similar electronic devices. PBD was introduced in the late 1980s to capture a design process that could encompass horizontal and vertical decompositions by multiple viewpoints [19], i.e. from component-based design and virtual integration to layered and model-based design. By combining these ideas together, PBD can support both the supply chain and the multi-layer optimization. The principles at the basis of PBD consist of starting at the highest level of abstraction, hiding unnecessary details of an implementation, summarizing the important parameters of the implementation in an abstract model, limiting the design space exploration to a set of available components, and carrying out the design as a sequence of refinement steps that go from the initial specification towards the final implementation using platforms at various levels of abstraction [48], [49].

In PBD the design progresses in precisely defined abstraction layers. Each of those layers is determined by a design platform. A platform is defined as a library of components that can be assembled to generate a design at a certain level of abstraction [6]. In this sense, a design platform represents a family of designs that satisfies a set of platform-specific constraints. The designs on each platform are represented by platform-specific design models. A complete design is obtained by creating platform instances via composing platform components and by mapping the platforms in the design flow onto subsequent abstraction layers. In other words, the PBD design methodology tries to map the system behaviors with the system architectures by reusing all abstraction levels; so that, it is obtained a meet-in-the-middle structured methodology that limits the design space exploration. The details and examples of the PBD methodology can be found in [6], where the idea of using PBD to cope with the challenges faced by the industry in heterogeneous system level design was initially proposed. PBD offers key concepts to develop a practical design methodology for CPS design in a general way [19]. However, it does not dictate a clear insight of how to move across platform layers and their respective abstractions. Further research has been carried out to improve PBD by guaranteeing a formal verification of the requirements to map functionalities with components and architectures. This is called contract-based design.

## 2.7 Contract-based Design

The contract-based design is an emerging paradigm for the design of complex systems which is based on the notion of components. A component is defined as a hierarchical entity that represents a logical unit of design [19]. Components are connected together by communicating and matching their values of determined ports and parameters. Each component can have associated multiple implementations, i.e. a set of ports, internal and external parameters (constant and variables), and a set of behaviors that implement the design unit. Likewise, each component is associated with a contract. A contract specifies the input-output behavior of a component by defining what the component guarantees, provided that its environment obeys some given assumptions. The ultimate goal of the contract-based design is to allow for compositional reasoning, stepwise refinement, and a principled reuse of components that are already pre-designed, or designed independently [50]. A contract for a component can be observed as a pair of assertions, which express its assumptions and guarantees.

The notion of contracts has been initially originated in the context of formal verification theories, specifically compositional assume-guarantee reasoning [51], which has been used for a long time, mostly for software verification in embedded systems. In a contract framework, design and verification complexity are reduced by decomposing system-level task into more manageable subproblems at the component level, under a set of assumptions. System properties can then be inferred or proved based on component properties [52]. Rigorous contract theories have been developed over the years, including assume-guarantee (A/G) contracts [53], interface theories [54], and generic construction of contracts [55]. A concrete example of how a system architecture can be verified by means of contracts is presented in [50], where a formal contract framework for verification based on the aforementioned works is presented. This is one of the first attempts to implement a contract verification framework with a concrete language and tool support for such verification. This framework exploits system architecture and component contracts decomposition to automatically generate a set of proof obligations, which once verified, allow concluding the correctness of the top-level system properties. The proof system reduces the correctness of contracts refinements to entailment of temporal logic formulas and is supported by a verification engine based on automated SMT techniques. However, their concrete adoption in CPS design is still in its infancy, a major challenge being the absence of a comprehensive modeling formalism for CPS, due to their complexity and heterogeneity [19], [56].

Contracts provide formal support to the complete system design flow in a hierarchical and modular way, combined with PBD methodology it can be used to address the complexity and heterogeneity of CPS. The foundations of the system design flow and some specific examples supporting the deployment of the PBD methodology with contracts is presented in [52]. In PBD, the design process is performed as a sequence of refinement steps from the most abstract representation of the design (top-level requirements) to its most concrete representation (physical implementation). Contracts provide guarantees on the correctness of each refinement step, and this becomes essential for the safety and reliability of the design.

It is desired to have an automatic and efficient way to prove the validity of the contracts statements. The assume-guarantee (A/G) contract framework introduced by Benveniste et al [53], [56], and later improvements including circular reasoning [45], introduce mathematical formalisms which are centered around behaviors and can be implemented in any kind of models encountered in system design. Nuzzo et al [52], proposes contracts as mechanisms to formally prove the following three conditions:

1. *Consistency:* a set of requirements is consistent,
2. *Compatibility:* an aggregation of components is compatible.
3. *Refinement by aggregation:* an aggregation of components refines a specification.

However, it is still not clearly defined how to properly write down a set of all possible requirements and constraints of a model in a mathematical way to define a formal contract. Additionally, virtual verification in tools based on simulation cannot be used to formally certify the satisfaction of a contract since the simulation cannot cover the entire state space, but rather to monitor and detect possible violations.

## 2.8 Product lifecycle management (PLM)

According to Stark and associates [57][14], product lifecycle management (PLM) is the business activity of managing, in the most effective way, a company's products all the way across their lifecycles; i.e. from the very first idea of a product until it is retired and disposed of. In principle, PLM applies to any type of product and industrial sector, it not only manages a simple product of a company, it manages all of the production processes and product portfolio of the company in an integrated way. At the highest level, the aim of PLM is to increase the product revenues, reduce product-related costs, maximize the value of the product portfolio, and maximize the value of the current and future products for both customers and shareholders. Therefore, although these issues are beyond the scope of this thesis, the PLM paradigm is directly related to the proposed O-model presented in chapter 5; especially, those concepts associated with the product manufacturing, commissioning, and maintenance process.

The new PLM paradigm emerged at the beginning of the $21^{st}$ century, and it has been evolving ever since, as a result of the vast scientific and technological advances in many different domains such as Microelectronics, Nanotechnology, Software, Biotechnology, and Mechatronics. Nowadays, manufacturing management is being transformed by a shift from the mass production paradigm to a new on demand, business-oriented, formally-defined, lifecycle, holistic, digital, customer-driven, and knowledge-based proactive production paradigm [58, 57]. This is not a matter of developing new technologies, but rather how to use the available technologies in the best way in order to capture the knowledge, to include a context into

---

[14] http://www.johnstark.com/

the knowledge, to share and re-use the knowledge in a smart way, and to coordinate the work of all manufacturing stakeholders during the entire product and factory lifecycle processes.

We can briefly illustrate the two main characteristics of the PLM paradigm as follows [57]:

- In the PLM paradigm, the activities of managing the products of a company must be defined and documented in cross-functional business processes across the product lifecycle. In the previous management paradigm, each department of the company defined its own activities independently from the functions of the other departments. Frequently, these activities were not formally documented.
- In the PLM paradigm, a cross-functional product data management (PDM) system manages product data across the product lifecycle. In the previous management paradigm, each department managed its own data independently from another department. For example, detailed information from the marketing and after-sales department were not available for the engineering department in charge of the design.

The most significant technology innovations that are currently boosting this paradigm shift towards more autonomous and efficient processes of the product lifecycle are the CPS, the internet of things (IoT) [59], and the rise of big data on cloud computing [60]. These are in fact the technology drivers of the so call Industry 4.0 revolution.

## 2.9 Industry 4.0

Industry 4.0 refers to the current trend of automation and data exchange towards the creation of a **"smart factory"**. The previously mentioned technology drivers of the Industry 4.0 promise a significant economic potential [61] in many applications mainly related to the development of large CPS. Some authors refer to this new concept as the *"$4^{th}$ industrial revolution"* [62, 63]. However, albeit this term is currently a top priority for many technology companies, research centers, and universities (mainly in the German-speaking area), a general accepted understanding of this term does not exist [64]. This is why it becomes difficult, and sometimes vague, to talk about the scope and characteristics that need to be implemented in a 4.0 industry.

A smart factory consists of an intelligent network of manufacturing systems equipped with sensors, actuators, and autonomous systems (smart technologies) forming a modular structure. These CPS monitor physical processes by making a virtual copy of the physical world (i.e. digitalizing the production chain) to take decentralized decisions faster and efficiently. Over the IoT, the CPS network communicates and cooperates with each part of the system and with humans in real time, both internal and cross-organizational services are offered and used by participants of the value chain [64]. In this definition, we clearly observe the importance of *virtual prototyping* far beyond the product design and verification, i.e. for product manufacturing, operation, and maintenance.

# 3 The VP-based design methodology

The proposed system design methodology based on VHDL-AMS VPs is described and explained in this chapter. The main purpose of this methodology is to support the virtual design and verification stages of the proposed O-model (described in chapter 5) for the development of small-scale CPS for power applications. This chapter presents the principal theoretical concepts and foundations of the methodology, together with the elements that compose it. The different elements are described, making emphasis in two operational elements of the VP-based design methodology: the VP-Model Library and the VP-Modeling Guidelines.

## 3.1 Design methodology foundations

The VP-based design methodology is a model-based design (MBD) methodology largely inspired by most of the concepts stated by layered, component-based, and platform-based design approaches for performing system-level modeling. This section discusses those important concepts taken from the state of the art.

### 3.1.1 Layered and component-based design

The most important concepts of layered and component-based design have been taking into account for developing the VP-based design approach. That is, the proposed methodology follows a systematic process to allow a modular architecture exploration by interconnecting components in the same hierarchical level (*component integration*), or in different hierarchical levels (*component refinement by aggregation*). Both the interface specifications and the component implementations can evolve to support various abstraction levels in a VP, from purely functional component abstractions to more refined physical implementations. In particular, the VP-based design methodology proposes solutions for the first two challenges previously mentioned in section 2.3, i.e. the need of rich interface model specifications, and the trade-off between model accuracy and simulation efficiency.

Since a component can be represented by more than one model interface with multiple component implementations (hereafter called *model implementations*), it is a challenge to guarantee the maintainability of a system-level VP over time. This issue is addressed in the proposed methodology by two particular ways: the use of a model library infrastructure for documenting related models (see section 3.6), and the application of model interface mappings in configurations (see subsection 3.5.1). The VHDL-AMS modeling language allows to define multiple *model implementations* (architectures) for the same component interface (entity); however, different component interfaces require the creation of different *component models*.

The proposed modeling methodology is based on indirect component instantiation via VHDL-AMS configuration declarations. This approach maximizes the design flexibility by enabling the designer to implement more than one design alternative in the same VP. A hierarchical VP that represents a complete system can be simulated using several configurations that allow selecting a list of particular *model implementations*, *abstraction levels*, and parameter values to represent different simulation scenarios (*test cases*). Contrarily to traditional component-based design [19], the VP-based methodology does not necessarily rely on "small interfaces" and is not limited by small logic components for building complex systems. Functional system-level modeling and gradual *model refinement* are used for building the system hierarchy. As long as model interfaces are properly defined in terms of ports and parameters, and the modeled system is properly decomposed into well-defined component models, the complexity of the component implementation can be managed at the system-level. Similarly, the efficient re-use of components can only be possible when the operation and simulation conditions of the component are correctly documented and verified. The VP-based design methodology relies on a model library to ensure the maximum re-usability of components and systems, see section 3.6.

### 3.1.2 Platform-based design

The system design approach of the VP-based design methodology is based on the **meet-in-the-middle** approach proposed by the PBD theory, which is a mix of the conventional top-down and bottom-up approaches for system design. This process intends to map the *functional space* (i.e. the set of constraints, requirements, and functionalities that the system must obey), to the *technology space* (i.e. the available technologies that we are able to use to build the system). By performing architectural space exploration, it is possible to find more than one architecture that can meet the *functional space* conditions; therefore, the mapping process can be seen as an optimization problem, where a set of performance metrics and quality factors are optimized over a space constrained by both system requirements and component feasibility constraints. Mapping is the mechanism that allows moving from a level of abstraction to a lower one using the available components within a library [52].

This idea can be better understood in Figure 3.1, where it is graphically represented the PBD approach for the design of a MV current sensor system based on the RogoCoil technology, this system is presented in section 4.4 on page 104. In this example, the PBD approach is illustrated from the earliest stage of a design, i.e. from the initial problem, which consists in measuring current in MV tension range with a particular set of requirements such as minimum accuracy level, safety, and low-costs (*functional space*). On the other hand, we have a set of available technologies that can be used for this specific problem (*technology space*). A decision making process have to be performed in order to select the best option before proceeding with the system design. The details of this process are shown in subsection 5.3.1.1 on page 138, where RogoCoil technology shows great features and advantages for current sensor implementation. At this stage, the functional and technology spaces have been mapped to solve the problem, i.e. a solution has been found. However, at this *platform* level, the mapping only represents an idea of how the problem can be solved. In order to truly get an implementable solution, the same mapping process is performed progressively, with more refined *platforms*, until obtaining the specifications of all the basic blocks of the system and their interconnection, i.e. the system implementation.

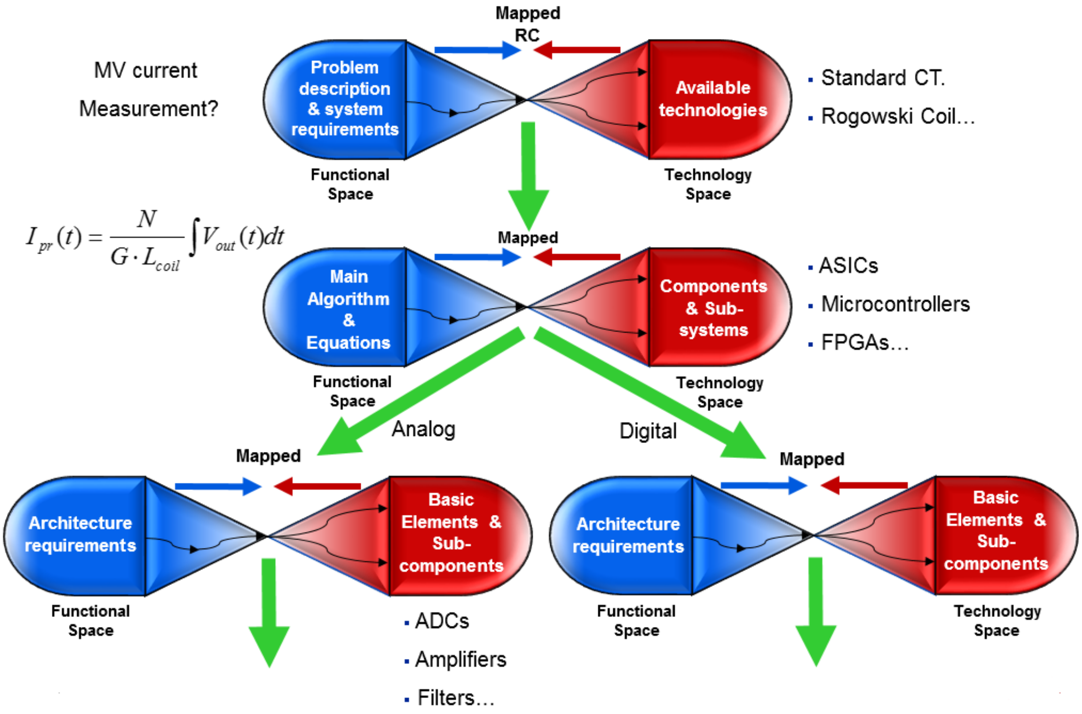


Figure 3.1: PBD diagrams for the RogoCoil current sensor system example presented in section 4.4. The design flow moves from higher to lower level platforms (i.e. more refined components) each time that the *technology space* is mapped to the *functional space*.

The *functional space* in the second *platform* includes the equation for electrical current estimation using the RogoCoil; the current is obtained by integrating the output voltage of the RogoCoil and multiplying by a certain factor that depends on coil characteristics, environmental conditions, and signal conditioning. So, there are various forms to solve this equation as there are several ways to obtain and process the required values to compute the integration. For example, the integration can be done by analog signal processing in hardware; and later, the multiplication can be done in software by using a microcontroller after digital conversion. Another option is to convert directly the analog output signal of the RogoCoil and perform all the calculation in digital, this can be done by software in a microcontroller, or by hardware in a field-programmable gate array (FPGA). Each solution implies a particular functional algorithm that can be mapped to a specific architecture (digital or analog architectures). We can observe that these two architectures can be considered at the same time, but they are designed and optimized independently. This allows system designers to evaluate diverse options and choose the best architecture taking into account the system requirements and architecture performances.

We can observe that this process continues on lower level *platform* levels, by considering both the architecture requirements and functionalities in the *functional space*, and the available components that can be interconnected together to perform the desired functionality meeting the established requirements. Contrarily to the PBD theory, the VP-based design methodology does not treat the system design on isolated *platforms*. The approach proposed in this research advocates for using, as far as possible, a unified *platform* (the VP-Model Library) for system and subsystem design. This is restricted by modeling language limitations that are related to specific modeling needs. For example, if the design requires the analysis of multiphysics interactions in the system geometry, FEA models are more convenient than VHDL-AMS models. Model migration techniques shall be applied in order to reduce the need of different software platforms for subsystem-level design, a particular example is presented in section 4.2 for obtaining a VHDL-AMS model from a geometrical FEA model. The model migration task is part of the Model maintenance process that has been proposed in order to support the *virtual prototyping* activity using the VP-based Methodology, see section 5.4 on page 155.

### 3.1.3 System-level modeling

The author of this thesis has analyzed and extrapolated the meet-in-the-middle approach from the PBD theory, in order to be used at any level of the system design, independently of the tools, languages, or technologies considered. Although this approach is useful to establish the conceptual view of an MBD methodology, it does not provide a clear procedure to map the technology and functional spaces. This is a challenging issue due to the difficulty to generalize a procedure that efficiently works for any type of system and requirements.

Instead, the VP-based design methodology uses the meet-in-the-middle concept to propose a method that starts the design by functional system-level modeling; and posteriorly, applies gradual *model refinement* to cover the desired verification needs. Functional system modeling consists in providing the most basic (ideal) descriptions for verifying the most basic functions and proof the system concept. However, the main problem is knowing how to scale up/down the functional models in order to perform more detailed non-functional system verifications, far beyond the apparently trivial functional models. There seems to be no compelling reason to argue that this issue is mainly caused by the lack of *modularity* of the modeling approach.

The functional model-based methodology proposed by Wan et al [25], establishes the advantages of rising to a very high-level the modeling abstractions of CPS in the automotive industry, see section 2.4. One of this advantages is the possibility to propose a system architecture without a specialized knowledge of all the components and subsystems of the CPS. Nevertheless, an effective industrial implementation of this approach, specially the automatic architecture synthesis, is only possible by counting with a reliable and well-defined model library of components and architectures. Whereas in a functional CPS model the specialized knowledge is irrelevant and discouraged, this knowledge is essential for building the *component models* and their specific *model implementations*. This is why the VP-based design methodology is focused on two main aspects of this paradigm: the implementation of a platform-independent model library dedicated to *component models* and VPs, and the description of modeling techniques for enhancing functional and non-functional system-level modeling by improving the scalability of the models and exploiting their *modularity*.

## 3.2 Model refinement

The *refinement level* or *abstraction level* of a model is a critical metric that determines the level of details, the status, and capabilities of a model to represent a real entity/system. In other words, the *abstraction level* indicates how accurate the real entity/system is modeled. Nonetheless, the main issue for precisely classifying a model according to its *abstraction level* is the undetermined amount of abstractions that a model can have. For example, there are different types of analog-to-digital converters (ADCs), e.g. direct-conversion [65], successive approximation [66], $\Sigma$-$\Delta$ ADCs [67], among many others. Consequently, an ADC model can be created using different representations meant to model any type of ADC or each of the aforementioned types. However, each of these representations can be created using several abstractions to model specific details of the functionality and the behavior of the ADC; for instance, the ADC dynamic range, bandwidth, or signal-to-noise ratio (SNR). Each particular abstraction leads to a specific *model implementation*.

Three main *abstraction levels* can be considered to classify *component models* in a non-rigorous way:

**1. Functional models:** They are the models with the highest level of abstraction. They are meant to represent the functionality of the entity/system that is modeled in an ideal fashion. For example, a *functional model* of an operational amplifier (OpAmp) can consist of a simple equation that relates the magnitude of the output voltage ($V_o$) with respect to the magnitude of the input voltage ($V_i$) by the OpAmp's gain ($G$), i.e. $V_o = G \cdot V_i$. Likewise, large CPS can also have very high-level functional models that describe the system operation in a simplistic and idealistic way. For instance, a *functional model* of a car, a train, or a plane, can be given in terms of the traveling speed, cost of the fuel, and amount of passengers that can be transported. These *component models* could be convenient for building a model of the transport infrastructure of a country for optimization purposes, but they are not very useful to design the components by themselves.

**2. Behavioral models:** They are medium-level abstractions that describe the functionality of a system by taking into account some non-idealities and critical variables that affect the system behavior. Considering the same case of the OpAmp example, a behavioral model abstraction can take into account non-ideal behaviors caused by the characteristics of the elements used to build the OpAmp and their interconnection. For instance, the input and output impedances, offset voltages, bias currents, common mode rejection ratio (CMRR), power supply rejection ratio (PSRR), transconductance, slew rate, gain-bandwidth product, etc. All these behaviors have the possibility to affect the model functionality according to different simulation conditions as in a real component.

**3. Physical models:** They are low-level abstractions that describe the functionality and behaviors of a system in a very accurate way, i.e. they are able to reproduce the physical interactions very close to the interactions of the real component/system. They can consider the intrinsic and extrinsic physical phenomena (i.e. thermal, electrical, optical, and mechanical processes) and some multiphysics such as electro-thermal, thermo-mechanical, or electro-optical interactions. Taking into account the OpAmp example once again, a physical model of this device could consider the physical behavior of the transistors, the substrate of the chip, the package of the OpAmp and the interaction with the environment. A transistor-level model of the OpAmp can be considered a low-level physical model if the functionality and behaviors of the device are reproduced by a physical model of the transistor, which can be an approximation of the physical phenomena within a transistor. For example, parameters such as the oxide thickness, the substrate doping concentrations, or the carrier mobility can be taken into account for calculating voltages and currents by equation-based models.

From the given *refinement level* classification, it can be deduced that it is not possible to define clear boundaries to include a model in one of the aforementioned *abstraction levels*. For example, if we add the output voltage supply limitation to the previously described ideal functional OpAmp model given by the equation: $V_o = G \cdot V_i$, could we classify this model as behavioral instead as functional? The answer is no, this OpAmp model is still no refined enough to model OpAmp behaviors that are relevant for the design of a larger system in which the OpAmp could be used. Therefore, the *abstraction level* update from functional to behavioral, or from behavioral to physical, is subjective and scarcely relevant for the virtual design and verification process.

The important concept that this thesis considers is the *model refinement* process, which is defined as the process of lowering the *abstraction level* of a model, e.g. from purely ideal model descriptions down to physical and more detailed descriptions. Precisely, a gradual *model refinement* process consists of incrementally adding non-idealities and behaviors to a functional model in order to verify by simulation the effects on the desired functionality. This process is done considering first the most critical behaviors, and subsequently the less relevant behaviors for the design. Since the designers often do not know whether specific non-idealities (e.g. environmental or cross-domain interactions) might significantly affect the behavior of the model, is highly desired to refine the models to the lowest possible level. However, the complexity of CPS and their subsystems makes inadequate to use very low-level physical models for quantitative design. The VP-based design methodology proposes to cope with this issue by using two approaches:

**1. High-level abstractions:** Cross-domain and physical interactions between components or between the system and the environment can be modeled by high-level abstractions that only considers the cause and the effect, an example is shown in subsection 4.2.7 on page 78 for modeling the effect of temperature variations on the main electrical variables of a RogoCoil sensor dynamically. Highly parameterized *component models* can consider the important properties whilst modeling only a few critical variables in a low-level fashion.

**2. Mixing *abstraction levels* in VPs:** The main issue with highly refined models (low-level abstractions) is their poor simulation performance, particularly for hierarchical system-level models (VPs) that require to use a large amount of this *component models*. However, mixing *component models* at different *abstraction levels* in the VPs can improve the simulation performance. This is especially useful for system design and functional verification, where the analysis can be focused on critical *component models* leaving the rest on a higher abstraction level. In the example given in subsection 4.3.2 in page 86 for the self-calibration unit (SCU) design of the RogoCoil sensor system, it can be observed that the VP is analyzed using some *model implementations* more refined than others, e.g. the RogoCoil model and the second-order Σ-Δ ADC model. In this way, a VP cannot be simply observed as a structural model at certain *abstraction level*, it must be seen as a hierarchical model that mixes several *component models* at different *abstraction levels* according to the verification needs. Thus, maximizing simulation performance and overall cost/benefit ratio of modeling and simulation activities.

One of the important consequences of the gradual *model refinement* is the possibility to refine a model by the simple aggregation of *component models*, these embedded *component models* shall be initially very-high level basic models. Afterwards, the refinement of the new hierarchical model can be done by *model refinement* of each *component model* included in the hierarchy. For example, suppose the case of a high-level low-pass filter model implemented by a specific transfer function. The same functionality performed by this transfer function can be implemented by adding passive and active electrical elements, such as resistors, capacitors, and OpAmps, forming a certain circuit topology. Thus, a more refined model is obtained. In this case, the gradual model refinement continues with the refinement of the OpAmp model.

## 3.3 Design methodology elements

The VP-based design methodology proposed in this work consists of a set of **modeling** and **operational** elements. The *modeling elements* are pieces of instructions and specifications that are required to build and set models and their virtual verification environments. On the other hand, the operational elements are the tools and media used to create and manipulate the *modeling elements*. The list of elements is described as follows:

**Modeling elements:**

**1. Component model:** It is a model of an individual predefined entity composing a system. A *component model* can represent an entity by one or multiple *model implementations*, each one at particular *abstraction level*. A *component model* can be hierarchical, i.e. constituted of more basic (less refined) *component models*.

**2. Virtual prototype:** This is the model of the complete system that is the subject of the design. A VP is normally a hierarchical and modular model that describes the architecture of a complex system by a small amount of interconnected main *component models*. A VP is built to model the system functionalities, behaviors, and environmental interactions; it may mix *component models* with different *abstraction levels*.

**3. Test bench:** It is a verification model that recreates the operation conditions of a particular model that is subject to verification, i.e. the design under verification (DUV). The *test bench* provides to the DUV its inputs. It also might provide parameter values to the DUV, and/or additional algorithms for post-processing the output signals of the DUV.

**4. Test case:** It is the virtual verification structure that is created for DUV verification. A *test case* of a DUV is composed by a particular *test bench*, a specific (DUV - *test bench*) customization environment organized in one or several VHDL-AMS packages and configurations, and the set of simulator directives related to the verification case. A *test case* represents a specific testing scenario. A *test case* can be focused for verifying either a *component model* or a particular configuration of a large and complex VP.

**5. Simulator setup:** It is a set of specifications that the simulator tool uses for executing a particular type of simulation/s. The simulation results directly depend on these specifications, which are critical for simulation performance and accuracy. These specifications are commonly gathered in a simulator file (e.g. the .pat file in SMASH), some of these specifications are tool-independent, such as the maximum time step (Hmax) for time-domain simulations and the integration method algorithm (e.g. Trapezoidal, Backward Euler, etc.).

**Operational elements:**

**1. Simulation framework:** It is a set of simulation tools, modeling languages, and related infrastructure that allows to create, manipulate, and simulate the aforementioned *modeling elements*. The selected *simulation framework* details are explained in section 3.4.

**2. VP-Model Library:** It is a centralized infrastructure for storing, documenting, maintaining, and queering *modeling elements* at different abstraction levels. Its main purpose is to provide an environment for maximizing the reuse of models, see details in section 3.6.

**3. VP-Modeling Guidelines:** They are a set of recommendations and techniques that are aimed to support the creation of scalable VPs and their *test cases* using a modular modeling infrastructure. These guidelines are specifically focused on ameliorating the manipulation of large VPs in order to improve the system-level modeling in VHDL-AMS.

Considering the previously mentioned elements of the VP-based design methodology, the important questions to answer are: how the modeling elements should be created? and how the operational elements interact among each other to create and manipulate the modeling elements?. In order to answer the first question, it is important mentioning that the VP-based design methodology is mainly focused on system-level modeling, i.e. focused on hierarchical models that are built by composition of more basic *component models*. Therefore, it is out of the scope of this research to describe a detailed procedure to elaborate basic *component models*. Actually, the available particular forms for creating basic *component models* are given by the different types of MoCs, such as process networks (PNs), dynamic dataflow (DDF), synchronous dataflow (SDF), continuous time, discrete events, synchronous/reactive, finite state machines (FSMs), hybrid systems, among many others [68]. In fact, different components and/or physical phenomena are modeled more cleanly with different MoCs because of their relative expressiveness and efficiency [69].

In this way, the VP-Modeling Guidelines treat the issue of properly building VPs and *test benches* for system-level modeling and verification. On the other hand, the matter of elaboration *test cases* is directly related to the type of verifications that are desired to be executed and the specific organization of the simulation project. Therefore, no systematic procedure can be generalized. The particular examples of the RogoCoil sensor system *virtual prototyping* presented in sections 4.3, 4.4, and 4.5 use several *test cases* for a predefined simulation project organization. Regarding the *simulator setups*, their elaboration exclusively depends on the capabilities offered by the simulation tool. Details and recommendations are given by each tool vendor in their reference manuals. This is why a deep analysis on *simulator setups* is not included in this work.

Finally, in order to answer the last question, we should consider the operational elements of the proposed methodology as partially independent elements that must interact in an effective way. In principle, any *simulation framework* allowing to represent the type of small-scale CPS of our interest, could be used. However, this would imply the use of different modeling guidelines. Although the discussed theories and concepts in the design methodology foundations are independent of the selected *simulation framework*, the modeling techniques directly depend on the used modeling language and simulation tools. For instance, the inheritance feature offered by object-oriented languages such as Modelica [35] or SystemC-AMS [70, 71], can be very useful for efficient system-level modeling due to the ability to relate different model interfaces with different *model implementations*. The implementation of this type of features in the proposed design methodology requires a change or an update of the modeling guidelines.

On the other hand, the proposed VP-Model Library is completely independent of the modeling guidelines and partially independent of the *simulation framework*. Since a model library shall contain the required *component models* to build VPs, the data structure and the presentation of the model's meta-information must be properly designed according to the modeling language in which the model is written. Thus, elaborate a model library to store models written in any language is a great challenge. The proposed VP-Model Library has been created to mainly support VHDL-AMS models and compatible modeling languages of the selected *simulation framework*. In the following three sections of this chapter, each operational element of the VP-based design methodology is presented in more details, making emphasis on the contributions of this work: the VP-Model Library and the VP-Modeling Guidelines.

## 3.4 Simulation framework

A *simulation framework* is here defined as the set of simulation software tools and modeling languages that allow to create, manipulate, and simulate *computational mockups*. In this section, it is explained both the selected modeling language and simulation tools used for the proposed methodology in the specific application field.

Analog and digital electronic components are the most common subsystems in the design of CPS for industrial and power automation applications. One of the best options for electronic system design and modeling is VHDL-AMS, which allows both multi-domain system-level modeling and specialized mixed-signal electronic design. VHDL-AMS is a powerful hardware description language which supports the description of both behavior and structure. The behavioral features of VHDL-AMS allow expressing the operation of a system at various levels of abstraction: from highly abstract ideal models to very detailed physical models. Designers normally proceed using a top-down methodology, first performing conceptual studies using ideal functional models, and then continuously refining the models until the whole design is completed in sufficient detail before physical implementation. Additionally, VHDL-AMS enables to model other domains such as thermal, mechanical, hydraulic among other systems, by introducing static and dynamic behavioral semantics together with differential and algebraic acausal equations [72]. Due to its multi-abstraction capabilities, VHDL-AMS can be used throughout all the design and verification of the system development life cycle, from architectural exploration and conceptualization, down to single component modeling [73].

The VHDL-AMS features and advantages previously mentioned, together with the strong importance of electronics in small-scale CPS design for power automation applications, are the main reasons to chose this language for system-level modeling and simulation. Although the proposed VP-based design methodology relies on VHDL-AMS as the main language for virtual integration and system simulation, the VHDL-AMS language characteristics and the simulation tool capabilities allow us to use other programming and hardware description languages for building more complex models by reusing available IP models. Commercial software tools such as SMASH from Dolphin Integration, support the simulation of VHDL-AMS hierarchical models that can integrate models written in other languages such as SystemC, Verilog-AMS, C/C++, or SPICE. This multi-language feature is possible by using VHDL-AMS foreign attributes, language wrappers and/or specific tool application programming interfaces (APIs). In this way, the *simulation framework* capabilities permit to elaborate multi-language VPs of complex systems using *component models* at different *abstraction levels*. For instance, a VP could consist of equation-based mechanical models implemented in VHDL-AMS, software algorithms implemented in C/C++, and synthesizable digital blocks written in VHDL or Verilog, i.e. hardware and software in the loop. Additionally, VHDL-AMS offers timed semantics that can reproduce accurate results for both analog and digital interactions, this feature is a practical solution for dealing with the lack of timing in the core abstractions of computing that software modules suffer [1]. FEA tools can be used separately to complement 3D geometrical modeling needs and multiphysics interactions as is shown in section 4.2.6 on page 66.

Since the VP-based design methodology relies on the capability to simulate the complete system using a mix of components at different *refinement levels*, it is highly important to simulate using the same or compatible tools. Therefore, the chosen *simulation framework* consists of electronic design automation (EDA) software tools such as SMASH or System Vision that permit system-level simulations based on VHDL-AMS and compatible modeling languages.

Currently, these EDA tool vendors offer the possibility to build structural models by using tool-dependent model schematics. Although model schematics can be a very practical feature for the visualization and elaboration of models, this jeopardizes the model sharing and its re-utilization capabilities. Current model schematics reduce the flexibility and scope of VPs and *component models*. This is why the approach proposed in this research is mainly focused in *computational mockups* represented by its source code written in a standard modeling language. Therefore, tools for bidirectional and automatic schematic-to-source-code conversion are highly desired for working with this *simulation framework* and the proposed methodology. At the moment of this research, there were no fully functional tools for this purpose.

## 3.5 The VP-Modeling Guidelines

The VP-Modeling Guidelines are a series of modeling recommendations and coding techniques in VHDL-AMS that have been obtained from the *virtual prototyping* activity for power sensors and actuators, such as the RogoCoil sensor system example presented in section 4.5. These modeling practices foster well-organized hierarchical architectures which reduce systematically the complexity in large VPs, facilitating the detection of simulation problems and generic parameter handling. Although several VHDL-AMS forms for building and organizing hierarchical models have been explored in this research, this document does not pretend to be exhaustive. The modeling practices here proposed are centered on the mechanisms that VHDL-AMS packages and configurations offer to facilitate the *design space exploration* and gradual *model refinement*.

In order to support the meet-in-the-middle approach, one of the most important issues that these guidelines address is about how to properly write scalable hierarchical models and *test benches* for system-level verification. The scalability of a *component model* at certain *abstraction level* must be considered in the two possible directions:

- By using the model in a larger and more complex model, i.e. a bottom-up *model scalability*.
- By *model refinement*. This is the case of architectural refinement by aggregation of sub-models, i.e. a top-down *model scalability*.

For the sake of discussion, suppose that the objective is to perform a *design space exploration* of a complex hierarchical system graphically represented by the VP shown in Figure 3.2. This is a fictitious hierarchical model that do not represent any particular system, its goal is to show a typical *component model* interconnection, a system architecture topology, and its VP modeling infrastructure. The example VP architecture is made of void *component models* that do not execute any particular algorithm. They are only interconnected to mimic a real system.

The component `CMP1:big_comp` (green) is a structural model which is composed of components `A` (red) and `B` (yellow). Likewise, the component `CMP2:complex_comp` (orange) is a structural model which is composed of simple model components `D:single_unit` and `H:single_unit`. The component `C:small_comp` (brown) is a simple model similar to the `comp` model, which only reports the values of the applied generic parameters (`F1`, `F2`, and `F3`). The complete VP (called `vp_top`) consists of the `CMP1`, `CMP2`, and `C` components. The `vp_top` model is instantiated as the DUV in the highest level design entity called `vp_top_tb(bench)`, which is the *test bench* of the complete design. The four independent boxes at the right part of the figure represent the VHDL-AMS packages that are used in the example VP. The four ellipses at the bottom represent the VHDL-AMS configurations of the `vp_top_tb(bench)`, i.e. the set of test bench configurations (TBCs).

We can observe from the VP structure shown in Figure 3.2, that the top-level of the virtual design is the *test bench*. A VHDL-AMS *test bench* is a design entity[1] which is used to verify the behavior of the DUV using one or more utility models. Usually, the *test bench* contains the model stimuli (e.g. analog and digital sources), constant parameters of the model, and/or the test scenario algorithm, i.e. processes, procedures, and/or functions to execute a particular

---

[1] i.e. entity + architecture

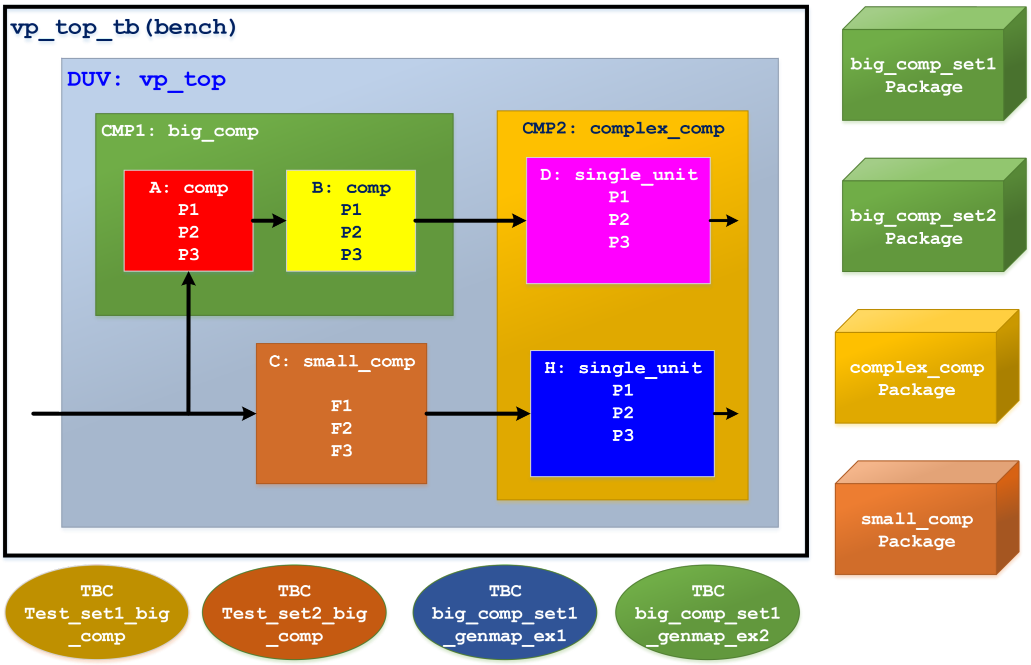


Figure 3.2: Graphical representation of the hierarchical VP model example. This is a high-level view of a complex system, a representation of the typical *component model* interconnection and the VP modeling infrastructure.

algorithm for the model verification. In this way, the *test bench* becomes the highest level structure of the model[2] in which the *design space exploration* and *model verification* take place. Although the given VP example only consists of one single *test bench*, the DUV can be declared and simulated in more than one *test bench* when multiple external conditions need to be verified, for example when it is desired to apply different types of input sources. However, since it is also possible to parametrize and configure multiple *model implementations* for the utility models in the *test bench*, a single *test bench* is sufficient in most of the cases. Additionally, considering that a VP is a model of the whole system including its environment, it is not expected to have a large amount of utility *component model* out of the DUV. Therefore, different testing scenarios can be simulated by a set of *test cases*, in which is only required a unique a simple *test bench* with a set of configurations and packages that are in charge of defining the specific model architectures and parameter values.

Following, it is described the procedure and the recommended modeling guidelines to build the foregoing virtual structure, starting from the highest level functional abstraction:

**1) Design Under Verification:** Regardless the size of the model, the first condition for creating a suitable *test bench* is to include (encapsulate) in a design entity the complete model to be verified, i.e the DUV, see Figure 3.3. In this way, the main conceptual algorithm that

---

[2] without considering the VHDL-AMS configurations or simulator directives to setup parameter values.

represents the functionality of the complete system can be modeled in the first functional *model implementation* of the VP. The objective is to clarify the required inputs, parameters, and the desired (ideal) output results. This model is very useful to serve as a reference for upcoming more refined *model implementations* of the VP. The first functional verification must lead to identifying the main components of the system.

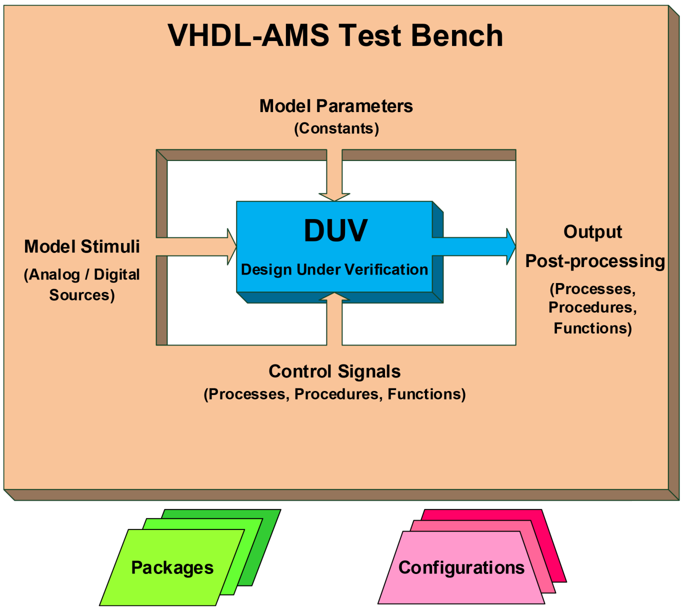


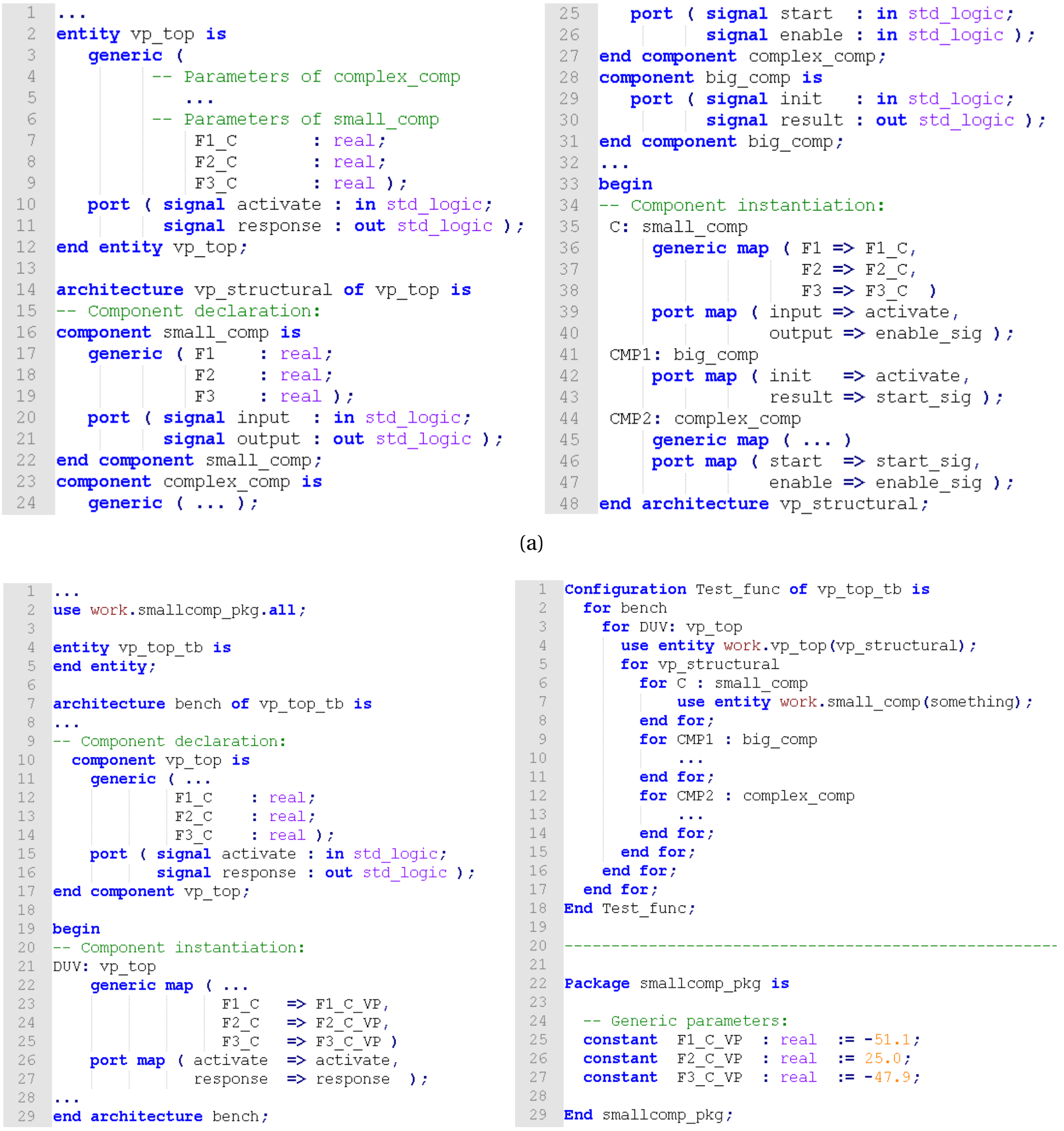
Figure 3.3: Conceptual diagram of a *test bench* and its DUV. The internal *model implementations* and the parameters of the DUV are defined from packages and configurations.

**2) VP hierarchical construction:** The main conceptual architecture of the whole system shall be created by modeling the functional behavior of the main components of the system. The main components of the VP example shown in Figure 3.2, are the `CMP1`, `CMP2`, and `C` components. Therefore, there is no need to initially think about lower level *component models* like the `comp` or `single_unit` models. These models are gradually aggregated after successful VP verification at the current *refinement level*. The recommended way to build the VP hierarchy is by component declaration, component instantiation, and configuration declarations, as it is done in the code of the VP example shown in Figure 3.4. In this case, the components `small_comp`, `complex_comp`, and `big_comp` are declared and instantiated in the architecture `vp_structural` of the `vp_top` entity, see Figure 3.4(a). This design entity is the DUV that is also declared and instantiated as a component in `vp_top_tb`, see Figure 3.4(b). Finally, one or more configurations (a TBC in this case) shall be used for binding the components to the desired design entities as it is shown in the `Test_func` TBC shown in Figure 3.4(c).

It is not recommended to use VHDL-AMS direct instantiation for structural models since this is less flexible and restrictive than the component declaration method. Likewise, designers should not rely on the default configuration mechanism[3], which is highly error prone.

---

[3] At compilation time, when a component declaration exactly match with an entity declaration previously



```vhdl
...
entity vp_top is
   generic (
         -- Parameters of complex_comp
            ...
         -- Parameters of small_comp
            F1_C       : real;
            F2_C       : real;
            F3_C       : real );
   port ( signal activate : in std_logic;
          signal response : out std_logic );
end entity vp_top;

architecture vp_structural of vp_top is
-- Component declaration:
component small_comp is
   generic ( F1    : real;
             F2    : real;
             F3    : real );
   port ( signal input  : in std_logic;
          signal output : out std_logic );
end component small_comp;
component complex_comp is
   generic ( ... );
   port ( signal start  : in std_logic;
          signal enable : in std_logic );
end component complex_comp;
component big_comp is
   port ( signal init   : in std_logic;
          signal result : out std_logic );
end component big_comp;
...
begin
-- Component instantiation:
 C: small_comp
     generic map ( F1 => F1_C,
                   F2 => F2_C,
                   F3 => F3_C  )
     port map ( input => activate,
                output => enable_sig );
 CMP1: big_comp
     port map ( init   => activate,
                result => start_sig );
 CMP2: complex_comp
     generic map ( ... )
     port map ( start  => start_sig,
                enable => enable_sig );
end architecture vp_structural;
```

(a)

```vhdl
...
use work.smallcomp_pkg.all;

entity vp_top_tb is
end entity;

architecture bench of vp_top_tb is
...
-- Component declaration:
  component vp_top is
    generic ( ...
              F1_C     : real;
              F2_C     : real;
              F3_C     : real );
    port ( signal activate : in std_logic;
           signal response : out std_logic );
end component vp_top;

begin
-- Component instantiation:
DUV: vp_top
    generic map ( ...
                  F1_C   => F1_C_VP,
                  F2_C   => F2_C_VP,
                  F3_C   => F3_C_VP )
    port map ( activate  => activate,
               response  => response  );
...
end architecture bench;
```

(b)

```vhdl
Configuration Test_func of vp_top_tb is
  for bench
    for DUV: vp_top
      use entity work.vp_top(vp_structural);
      for vp_structural
        for C : small_comp
            use entity work.small_comp(something);
        end for;
        for CMP1 : big_comp
            ...
        end for;
        for CMP2 : complex_comp
            ...
        end for;
      end for;
    end for;
  end for;
End Test_func;

-----------------------------------------------------

Package smallcomp_pkg is

  -- Generic parameters:
  constant  F1_C_VP  : real  := -51.1;
  constant  F2_C_VP  : real  := 25.0;
  constant  F3_C_VP  : real  := -47.9;

End smallcomp_pkg;
```

(c)

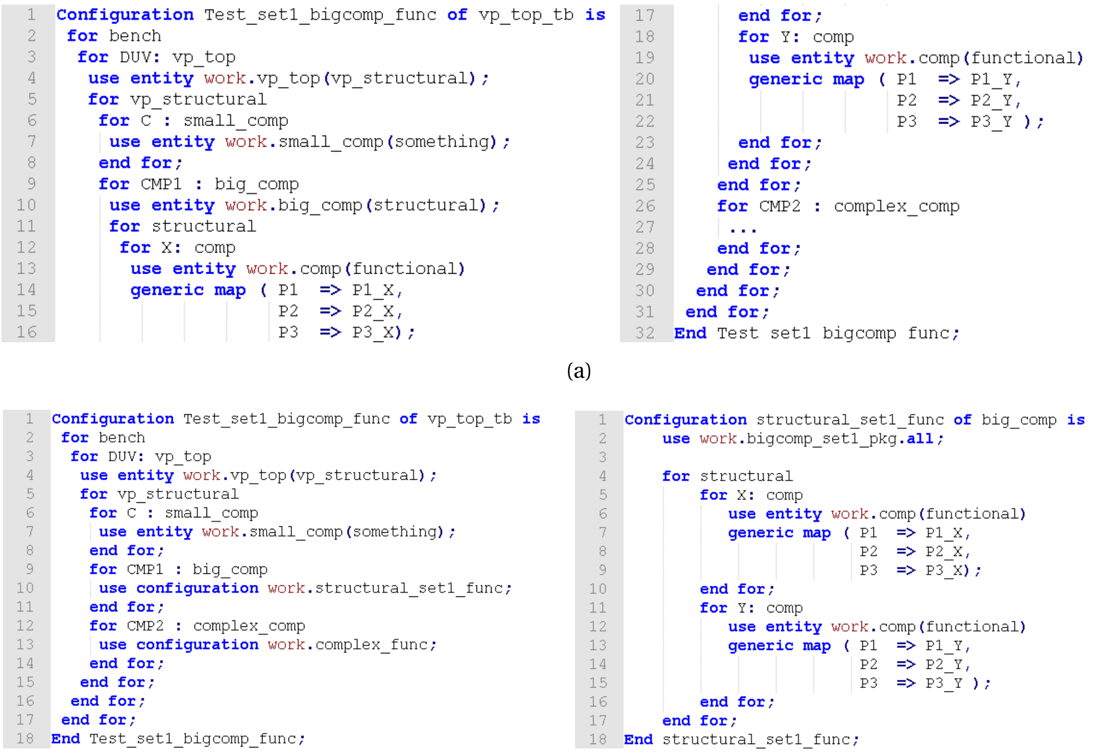
Figure 3.4: Component declaration and instantiation code fragment example (the three dots (**...**) means omitted code). (a) `vp_top` hierarchical design entity code fragment. (b) `vp_top_tb` *test bench* code fragment. (c) `Test_func` TBC and `smallcomp_pkg` package code.

**3) VP modeling infrastructure:** The refinement process of a hierarchical *component model*, can result in a significant amount of sub-components and architectures that belong to particular main *component models* in the VP. Therefore, the most practical way to deal with such model structure that will be gradually becoming more complex is to define model configura-

---

compiled, it is not necessary to specify the binding between such component and its entity in the configuration of that structural model. In this case, the configuration is binding by default the component with the corresponding entity by using its more recently analyzed architecture.

tions for the top-level entities that contain a large number of components, i.e. the *test bench*, the DUV, and some big main *component models*.



```
Configuration Test_set1_bigcomp_func of vp_top_tb is
 for bench
  for DUV: vp_top
   use entity work.vp_top(vp_structural);
   for vp_structural
    for C : small_comp
     use entity work.small_comp(something);
    end for;
    for CMP1 : big_comp
     use entity work.big_comp(structural);
     for structural
      for X: comp
       use entity work.comp(functional)
       generic map ( P1  => P1_X,
                     P2  => P2_X,
                     P3  => P3_X);
      end for;
      for Y: comp
       use entity work.comp(functional)
       generic map ( P1  => P1_Y,
                     P2  => P2_Y,
                     P3  => P3_Y );
      end for;
     end for;
    end for;
    for CMP2 : complex_comp
     ...
    end for;
   end for;
  end for;
 end for;
End Test set1 bigcomp func;
```

(a)

```
Configuration Test_set1_bigcomp_func of vp_top_tb is
 for bench
  for DUV: vp_top
   use entity work.vp_top(vp_structural);
   for vp_structural
    for C : small_comp
     use entity work.small_comp(something);
    end for;
    for CMP1 : big_comp
     use configuration work.structural_set1_func;
    end for;
    for CMP2 : complex_comp
     use configuration work.complex_func;
    end for;
   end for;
  end for;
 end for;
End Test_set1_bigcomp_func;
```

(b)

```
Configuration structural_set1_func of big_comp is
    use work.bigcomp_set1_pkg.all;

    for structural
        for X: comp
           use entity work.comp(functional)
           generic map ( P1  => P1_X,
                         P2  => P2_X,
                         P3  => P3_X);
        end for;
        for Y: comp
           use entity work.comp(functional)
           generic map ( P1  => P1_Y,
                         P2  => P2_Y,
                         P3  => P3_Y );
        end for;
    end for;
End structural_set1_func;
```

(c)

Figure 3.5: Example of the use of main component configurations inside of a higher level configuration (the three dots (**...**) means omitted code). (a) Large TBC before using configurations inside configurations technique. (b) Compact TBC after using configurations inside configurations technique. (c) Configuration for the `big_comp` main *component model*.

In principle, the complete design could be managed using a single top-level configuration, the test bench configuration (TBC). However, as the VP is becoming bigger and more complex, the TBC will turn clumsy and difficult to manage. This is why the recommended approach is to use a TBC to manage the first high-level VP implementations and progressively transfer the configuration code to a DUV configuration or to a lower level configuration of main *component model*. Hence, specific configurations of any *component model* used in the VP can be called from the highest level configurations, i.e. from TBCs or DUV configurations. One file per configuration declaration is highly recommended for representing different *test cases*. These files must be logically organized together with the source code of the complete design. Consider the source code examples in Figure 3.5. Figure 3.5(a) shows how initially looks the TBC termed `Test_set1_bigcomp_func`, observe that the component `big_comp` of the DUV is bound to a hierarchical design entity that possesses more *component models* which need to be bound using a configuration. This TBC can be simplified as is shown in Figure 3.5(b) by using a separate configuration for the `big_comp` component, see Figure 3.5(c).

On the other hand, it is not recommended to rely on configuration specifications[4] since the goal of this guideline is to be able to build modular models, i.e. without the need to edit the concerned *component model* architectures. In order to improve the bottom-up scalability of the VP, the complete set TBCs must be converted in DUV configurations. Consequently, the DUV can be used as a component of a larger model and all its *test cases* previously created can be called from a higher level configuration to create new *test cases*.

**4) Parameter organization and classification:** In general, the parameters of a model can be classified into two types: primary and secondary parameters. A *primary model parameter* (PMP) is a constant parameter that does not depend on other internal or external parameters. It is available at the interface of the model with the purpose of modeling certain behavior and/or functionality. In order to make a PMP accessible from a top-level design unit, it must be expressed as a VHDL-AMS generic parameter in the entity declaration of the model. On the other hand, a *secondary model parameter* (SMP) is a parameter whose value is derived or computed from other primary or secondary model parameters. An SMP is normally expressed as a VHDL-AMS constant inside the model architecture, but can also be computed externally in *test benches* or in packages and transmitted to the model by its generic parameters as it is done for the PMPs. An SMP is also used to model particular functionalities and/or behaviors. There are many ways to organize the parameters of the model as PMPs or SMPs. For instance, in a top-down design of an analog filter, we could consider PMPs to define specifications (i.e. what to achieve?), e.g. the gain and the bandwidth of the filter. Likewise, we could use SMPs to define the implementation information (i.e. how to meet the specifications?), e.g. the resistance and capacitance values of a specific implementation of the filter. Contrarily, in a bottom-up design of the same filter, the PMPs might be the resistance and capacitance values, and the SMPs could be the gain and bandwidth of the filter. In any case, the PMPs and SMPs definition depends on the purpose of the design.

**5) Parameter value verification:** It is a good practice to use VHDL-AMS assertions for verifying the correctness of the generic and constant parameter values. This practice enhances the reliability and robustness of the models. The VHDL-AMS `assert` statement can be used as a concurrent statement in entity or architecture declarations, or as a sequential statement in processes, functions, and procedures. The VHDL-AMS `report` and `severity` clauses[5] are also used together with the `assert` statement. If the `severity` clause is omitted, the default level depends on the simulator settings. It is recommended to use `error` or `failure` severity levels for verifying generic and constant parameter values since the simulation stops immediately after the assertion evaluation[6], allowing the user of the model to quickly notice the problem without waiting for the simulation result analysis. If the report message is omitted, a default message is displayed, but it is not recommended to rely on default output messages as they can be difficult to understand in a model with many verifications done by assertions. In order

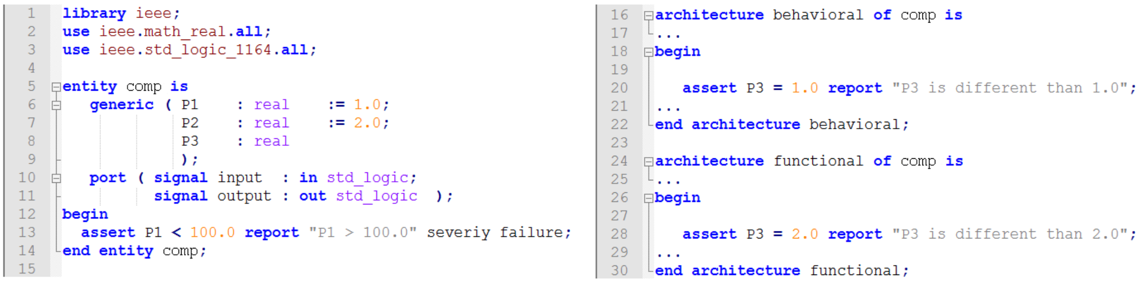
---

[4] In this less flexible VHDL-AMS approach, the components are configured within the architecture that instances them, rather than using a separate configuration declaration design unit.

[5] The report clause can be used as an independent sequential statement in processes, functions, and procedures.

[6] This feature depends on how the simulator is configured.

to illustrate the generic parameter verification, consider the example shown in Figure 3.6. We can observe that the `comp` model has two architectures: `behavioral` and `functional`. In lines 20 and 28, it is shown that an assertion is used to verify a particular value that the generic parameter `P3` must have in each architecture. Similarly, in line 13 on the same figure, an assertion is included in the `comp` model entity declaration to verify that the generic parameter `P1` is lower than `100.0`. This verification is always done disregarding the selected *model implementation* of `comp`.



```
 1  library ieee;
 2  use ieee.math_real.all;
 3  use ieee.std_logic_1164.all;
 4
 5  entity comp is
 6     generic ( P1    : real    := 1.0;
 7               P2    : real    := 2.0;
 8               P3    : real
 9               );
10     port ( signal input  : in std_logic;
11            signal output : out std_logic  );
12  begin
13    assert P1 < 100.0 report "P1 > 100.0" severiy failure;
14  end entity comp;
15
```

```
16  architecture behavioral of comp is
17  ...
18  begin
19
20     assert P3 = 1.0 report "P3 is different than 1.0";
21  ...
22  end architecture behavioral;
23
24  architecture functional of comp is
25  ...
26  begin
27
28     assert P3 = 2.0 report "P3 is different than 2.0";
29  ...
30  end architecture functional;
```

Figure 3.6: Example code: Assertions for verifying parameter values.

**6) Parameter grouping in packages and usage:** As the VP grows in size and complexity the number of parameters gathered by the DUV can be very large. This is why is recommended to create a VHDL-AMS package for grouping all the PMPs declarations of each main component (or subsystem) in the VP. In this way, it is easier to manage properly organized parameters from individual files.

In fact, there are different forms to use those packages in a VP. The recommended approach considers the two following cases that might be applied one after the other as the complexity of the VP grows:

**6.1) From the *test bench*:** The packages can be called from the *test bench* by using the `use` clause at the beginning of the *test bench* architecture declaration when the VP presents the following conditions[7]:

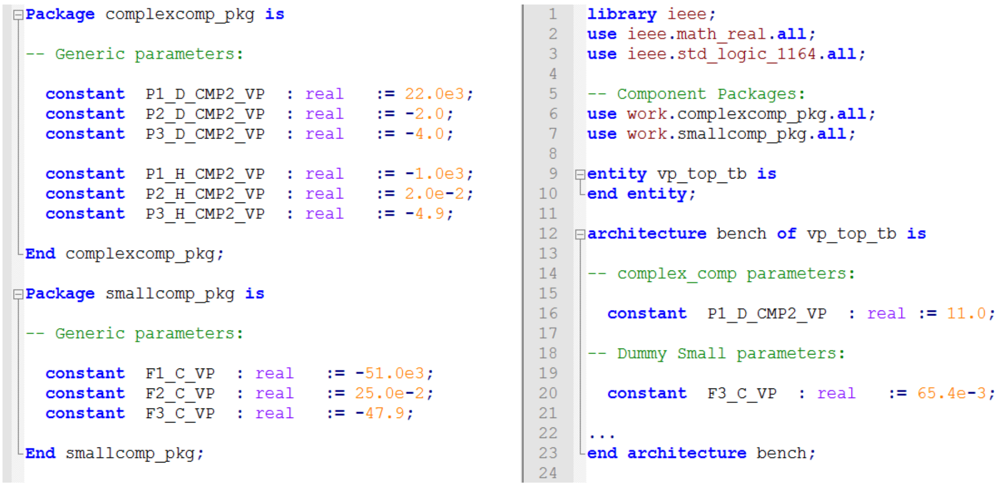
- Each generic parameter of every component in the design is declared and bound in the subsequent higher level design entity via generic mapping in the component instantiation until reaching the *test bench*. This approach is known as *hierarchical parameter binding*.
- There are no parameters in any package that exclusively belong to a particular architecture or a verification scenario (*test case*).
- The number of packages and *test cases* is low.

For example, in Figure 3.7 we can observe that all the generic parameters of the `complex_comp` and the `small_comp` components of the example VP are declared in two different

---

[7] Typically for early high-level VPs that contain relatively small amount of generic parameters per *component model*

packages, see Figure 3.7(a) (each package must be declared in an individual file). These packages are used on the *test bench*, see lines 6 and 7 of the code in Figure 3.7(b). Since VHDL-AMS allows to overwrite the value of a parameter declared in a package in the architecture in which the package is being used (in this case the *test bench*), the PMP values set in the packages can be re-assigned in the *test bench*. Figure 3.7(b) shows that only the parameters `P1_D_CMP2_VP` and `F3_C_VP` are overwritten in the *test bench*, see lines 16 and 20 of the code shown in Figure 3.7(b). All the other numerical values of the constant parameters are given from their respective packages.



```
Package complexcomp_pkg is

-- Generic parameters:

  constant  P1_D_CMP2_VP  : real    := 22.0e3;
  constant  P2_D_CMP2_VP  : real    := -2.0;
  constant  P3_D_CMP2_VP  : real    := -4.0;

  constant  P1_H_CMP2_VP  : real    := -1.0e3;
  constant  P2_H_CMP2_VP  : real    := 2.0e-2;
  constant  P3_H_CMP2_VP  : real    := -4.9;

End complexcomp_pkg;

Package smallcomp_pkg is

-- Generic parameters:

  constant  F1_C_VP  : real    := -51.0e3;
  constant  F2_C_VP  : real    := 25.0e-2;
  constant  F3_C_VP  : real    := -47.9;

End smallcomp_pkg;
```

(a)

```
 1  library ieee;
 2  use ieee.math_real.all;
 3  use ieee.std_logic_1164.all;
 4
 5  -- Component Packages:
 6  use work.complexcomp_pkg.all;
 7  use work.smallcomp_pkg.all;
 8
 9  entity vp_top_tb is
10  end entity;
11
12  architecture bench of vp_top_tb is
13
14  -- complex_comp parameters:
15
16    constant  P1_D_CMP2_VP  : real := 11.0;
17
18  -- Dummy Small parameters:
19
20    constant  F3_C_VP  : real   := 65.4e-3;
21
22  ...
23  end architecture bench;
24
```

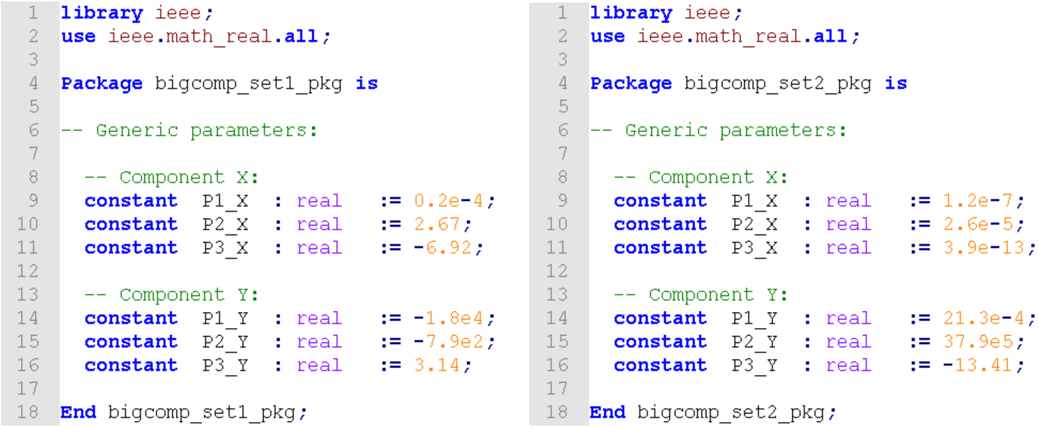
(b)

Figure 3.7: Example code: (a) `complexcomp_pkg` and `smallcomp_pkg` packages. (b) Utilization of parameter packages in the *test bench*.

The parameter overwriting technique can be very useful for performing statistical and parametric simulations using the proposed approach explained in Appendix E. Note that in order to correctly overwrite a constant parameter, the constant declaration in the *test bench* must have the same name than its corresponding constant declared in the package.

**6.2) From configurations:** When the complexity of the VP increases either by the interconnection of more *component models* or by *model refinement*, it is not recommended to use the conventional *hierarchical parameter binding* approach since the manipulation of the hierarchical chain of generic parameters and ports becomes unmanageable. The *hierarchical parameter binding* approach is also not recommended when a *component model* use parameters values that are particular to a specific architecture or to a specific *test case*. For example, the electrical model of the RogoCoil transducer described in section 4.2.2 can be used to represent multiple RogoCoil transducer types that use particular dimensions and electrical specifications values, see appendices A.2 and A.5. These set of PMPs values are organized in different packages that can be selected according to the configuration specified in particular *test cases*.

The recommended approach consists in using generic parameter mapping of the components from configurations, this is done in order to set the generic parameter values of the entities bound to the components. Contrarily to the *hierarchical parameter binding* approach, the generic parameter mapping from configurations does not require to use generic parameters in component declarations. Consequently, setting the parameter packages from configurations is the technique that gives more flexibility and scalability in complex designs than any other VHDL-AMS approach. A large number of components and generic parameters can be handled from packages and set by the proposed configurations, i.e. the TBCs, the DUV configurations, or any other subsystem configuration. In order to illustrate this approach let us consider the two sets of `big_comp` packages shown in Figure 3.8.



```
library ieee;
use ieee.math_real.all;

Package bigcomp_set1_pkg is

-- Generic parameters:

  -- Component X:
  constant  P1_X  : real    := 0.2e-4;
  constant  P2_X  : real    := 2.67;
  constant  P3_X  : real    := -6.92;

  -- Component Y:
  constant  P1_Y  : real    := -1.8e4;
  constant  P2_Y  : real    := -7.9e2;
  constant  P3_Y  : real    := 3.14;

End bigcomp_set1_pkg;
```

(a) `bigcomp_set1_pkg`

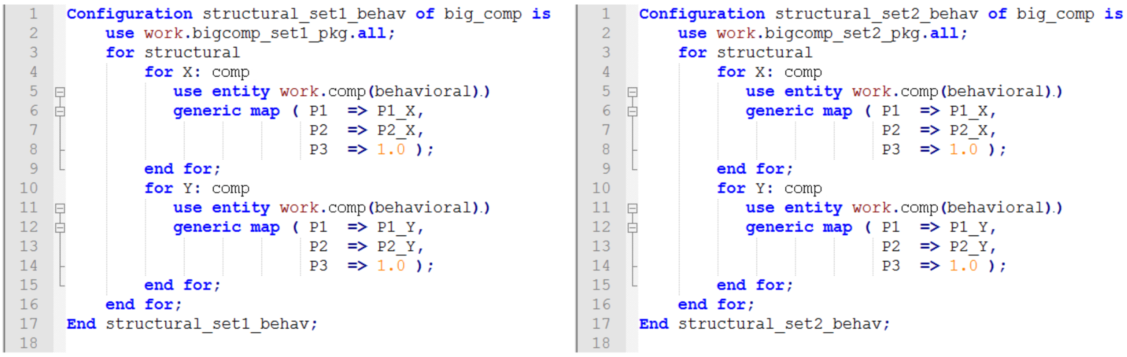
```
library ieee;
use ieee.math_real.all;

Package bigcomp_set2_pkg is

-- Generic parameters:

  -- Component X:
  constant  P1_X  : real    := 1.2e-7;
  constant  P2_X  : real    := 2.6e-5;
  constant  P3_X  : real    := 3.9e-13;

  -- Component Y:
  constant  P1_Y  : real    := 21.3e-4;
  constant  P2_Y  : real    := 37.9e5;
  constant  P3_Y  : real    := -13.41;

End bigcomp_set2_pkg;
```

(b) `bigcomp_set2_pkg`

Figure 3.8: `big_comp` parameter packages of the hierarchical VP example.

Figure 3.9 shows the two example configurations for the `big_comp` entity. The packages `bigcomp_set1_pkg` and `bigcomp_set2_pkg` are made visible by the `use` clause, see line 2 in both Figures 3.9(a) and 3.9(b). Therefore, the parameters `P1_X`, `P2_X`, `P1_Y` and `P2_Y` declared inside the packages are visible by the configuration.

In this way, a large number of parameters can be easily switched by using particular configurations. Using component declarations without generic parameter declarations can be adopted at any point of the VP design. This is why if an initial *hierarchical parameter binding* VP structure is in place, the parameter management can be easily migrated to configurations without any particular change. The parameter mapping in configurations has the priority over the parameter mapping in the component instances, see the details in subsection 3.5.1. In fact, a VP parameter management using a mix of both approaches can be implemented when the VP is built from the bottom-up, i.e. using packages for the main components of the VP (subsystems) which might be built by *hierarchical parameter binding*, then assigning these packages directly from the TBCs or DUV configurations. So that, there is no need to declare generic parameters either for the components of the DUV or for the DUV component in the *test bench*. An example of this modeling approach is shown in section 4.4 for the current measurement unit (CMU)



```
Configuration structural_set1_behav of big_comp is
    use work.bigcomp_set1_pkg.all;
    for structural
        for X: comp
           use entity work.comp(behavioral).)
           generic map ( P1  => P1_X,
                         P2  => P2_X,
                         P3  => 1.0 );
        end for;
        for Y: comp
           use entity work.comp(behavioral).)
           generic map ( P1  => P1_Y,
                         P2  => P2_Y,
                         P3  => 1.0 );
        end for;
    end for;
End structural_set1_behav;
```

(a)

```
Configuration structural_set2_behav of big_comp is
    use work.bigcomp_set2_pkg.all;
    for structural
        for X: comp
           use entity work.comp(behavioral).)
           generic map ( P1  => P1_X,
                         P2  => P2_X,
                         P3  => 1.0 );
        end for;
        for Y: comp
           use entity work.comp(behavioral).)
           generic map ( P1  => P1_Y,
                         P2  => P2_Y,
                         P3  => 1.0 );
        end for;
    end for;
End structural_set2_behav;
```

(b)

Figure 3.9: Example code: Configurations of the `big_comp` entity.

of the RogoCoil sensor system.

It is, however, important to foresee early in advance an adequate organization of the VP and its *test bench* for managing different *test cases* in a practical way. It is not practical to elaborate a modeling infrastructure with a large number of configuration units to set individual design entities and their parameters. A proper identification of the subsystems is key for the functional system-level modeling and gradual *model refinement* approach.

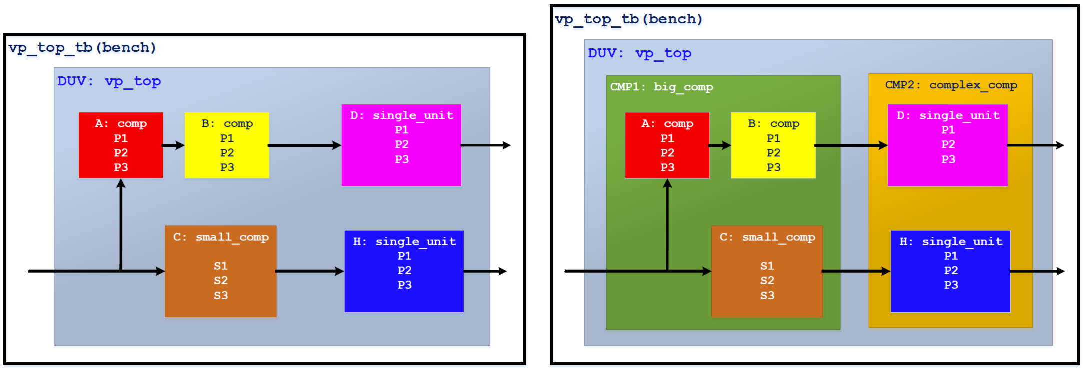
Finally, note that components without generic parameters are not suitable for parametric and statistical analysis (e.g. Monte-Carlo simulations) since a VHDL-AMS configuration only allows to bind parameters but does not provide any mechanism for supplying different parameter values in simulation. Although these types of simulations can be done in components without generic parameters by implementing the statistical and parametric packages (see Appendix E) inside the component architectures, this alternative just translates the complexity of the parameter management from the highest level to lower levels in the hierarchy. Therefore, it is up to the model designer to choose the best modeling approach according to the verification needs and the expected size of the complete design.

**7) Logical component encapsulation:** One of the main challenges of a VHDL-AMS modeling approach for complex system design, is to cope with large structural models in which many components can be instantiated at the same level. It does not matter at which level of the hierarchy the models are instantiated, a hierarchical model is not easy to manipulate if the component interconnection is managed by a huge amount of lines of code. This problem can be mitigated by using a graphical software tool which allows seeing a schematic view of the model. However, at the time that this research has been done, there is no commercial tool to properly use schematics for VHDL-AMS models[8] and at the same time take the advantage of the flexibility that VHDL-AMS configurations provide for complex VP modeling.

---

[8] System Vision from Mentor Graphics and SLED from Dolphin Integration, are software tools that allow building schematics for multi-domain complex systems based on VHDL-AMS and Spice models. However, none of those tools properly support the use of VHDL-AMS configurations.

Alternatively, it is recommended to minimize the number of components per architecture at any level by forcing a logical grouping (encapsulation) of a set of components inside other structural design entities that must be contained inside the original structural architecture. In order to illustrate this modeling guideline, suppose that we are designing a model that contains five main components, as it is shown in Figure 3.10(a). We can reduce the number of components of the DUV to only 2 main components by encapsulating the initial five components in two groups. Suppose that the components `A`, `B`, and `C` are part of the analog front-end of the system, and the components `D` and `H` are digital processing units of the system. Therefore, it makes sense to create two new structural components `CMP1` and `CMP2` as it is shown in Figure 3.10(b). This logical division can be applied in any of the structural architectures of the design. If the VP is being refined (i.e. a top-down design approach), it is easy and natural to make these divisions. However, a logical component encapsulation is less evident when the system is designed from the bottom-up.



(a) (b)

Figure 3.10: Logical component encapsulation example. (a) Flat structural model. (b) Logically encapsulated structural model.

Highly encapsulated VPs and *component models* bring important advantages for hierarchical model elaboration and utilization; it is important to highlight the following advantages:

- The interconnection of the components can be manipulated easily by dealing with smaller structural architectures.
- It is easier to understand how to add or suppress components in the design since specific locations in the hierarchy can be found more easily. The *modularity* of the DUV is improved.
- This technique facilitates the *design space exploration* and verification of large VPs since different component interconnections can be simulated by simply selecting different *model implementations* from the main configurations of the design, i.e. from the TBCs or the DUV configurations.

Nevertheless, it is not recommended to apply the logical component encapsulation, and later a *hierarchical parameter binding*. The additional hierarchical levels created in a highly encapsulated model makes more difficult to make parameter binding from the top to the bottom architectures.

### 3.5.1 Mapping in configurations

The use of VHDL-AMS configuration design units to structure and organize large and complex VPs and their *test cases*, demands a deep understanding of the operation principle and the modeling and simulation capabilities of the language. The discussion presented in this subsection is centered on the idea of decreasing the complexity and increasing the efficiency of model interfaces dedicated to CPS design and verification. A VHDL-AMS model interface consists of a set of *ports* and *parameters*. This research does not pretend to explain the details of the VHDL-AMS configurations per se. Instead, it gives key modeling rules for using configurations, and it reports VHDL-AMS operation principles of port and parameter mapping that were not reported in the literature at the moment of this research. The principles were deduced by studying different configuration cases that the language and the *simulation framework* allow building, but they are not conventionally used.

By definition, a configuration is a VHDL-AMS design unit that defines how component instances in a given block are bound to design entities. The configurations describe how the *model implementations* are used in the system architecture represented by a VP and its *test bench* to form a complete design. It is important to highlight the design flexibility offered by configurations, which is essential to cope with complex designs with a large number of components.

As it is mentioned in the VP hierarchical construction guideline (modeling guideline 2) in section 3.5, it is not recommended to rely on default configurations, i.e. a default model interface mapping when the component and its bound entity have exactly the same *model signature*. A clear component binding to the corresponding entity must be always defined in a configuration using at least the following VHDL-AMS instruction:

```
use entity work.entity_name(architecture_name);
```

where `entity_name` is the name of the entity to be bound, and `architecture_name` is the name of the desired architecture of such design entity. Therefore, the component name and the entity name do not necessarily need to be the same as we should always indicate the binding in the configuration. However, it is important to know how the VHDL-AMS interface binding (mapping) works. In configurations, the VHDL-AMS interface mapping mechanism between components and entities obeys the following principles:

**I)** If there is a name match between a parameter/port in the component declaration and the used entity, VHDL-AMS binds by default those parameters/ports.

**II)** A generic/port mapping in a configuration is a direct access to the parameters/ports of a component so that it nullify the generic/port mapping done in the component instantiation.

Appendix D illustrate these principles for generic parameter mapping considering models built using a *hierarchical parameter binding*. In the following subsection, it is presented a more interesting alternative case that illustrates the mapping in configuration principles and the definition of specific case rules. These rules can be used together with the modeling approach suggested by the VP-Modeling Guidelines, but also supports VPs built by *hierarchical parameter binding*.

#### 3.5.1.1 Port mapping

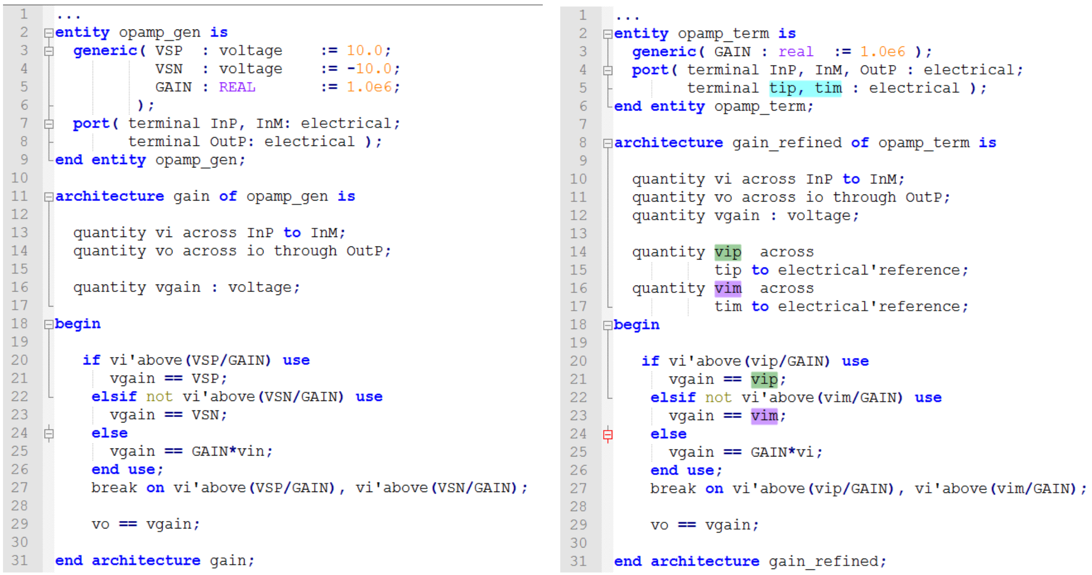
One of the advantages of using component instances in structural architectures is to be able to simulate different *model implementations* of the same subsystem/device without the additional effort to modify the component interconnection and/or the *test bench*. In this way, the typical case for a component instance is to explore different architectures of the same component, which has a fixed interconnection determined in the structural architecture. By rule of thumb, when different models of the same device are being designed, the same entity and different architectures should be used if the *model implementations* have the same *model signature*. Otherwise, the different models of the device must use different entities[9].

In the majority of the cases, the ports of the component are often the same among the possible design entities that can be bound in configurations. Therefore, only one entity is sufficient. However, can the designers use a single component declaration to include in a hierarchical model different *model implementations* of the same device with more than one entity? The traditional approach implies that this is not possible since different components instantiated in different structural architectures are used for this purpose. However, the author of this thesis has established a method to systematically use a single component declaration to instantiate not only different architectures but also different entities.

This case can occur in particular modeling scenarios, typically when a model refinement is done progressively over time. The type of the ports may change or more ports may be added to the model. For instance, when the values given by generic parameters need to be transformed in variable quantities as input ports. Let us consider a practical example of an OpAmp. Figure 3.11(a) shows a simple implementation of an OpAmp in which the positive `VSP` and negative `VSN` supply voltages are given as generic parameters in the entity. On the other hand, in Figure 3.11(b), the negative and positive supply voltages are not given anymore as generic parameter constants, but instead, they are determined by the dynamic terminal ports `tip` and `tim`. In the `opamp_term(gain_refined)` design entity, the positive and negative supply voltages are now given by the quantities `vip` and `vim` respectively.

---

[9] This rule is consistent with the VP-Model Library structural organization.



```
...
entity opamp_gen is
  generic( VSP  : voltage    := 10.0;
           VSN  : voltage    := -10.0;
           GAIN : REAL       := 1.0e6;
         );
  port( terminal InP, InM: electrical;
        terminal OutP: electrical );
end entity opamp_gen;

architecture gain of opamp_gen is

  quantity vi across InP to InM;
  quantity vo across io through OutP;

  quantity vgain : voltage;

begin

   if vi'above(VSP/GAIN) use
     vgain == VSP;
    elsif not vi'above(VSN/GAIN) use
     vgain == VSN;
    else
     vgain == GAIN*vin;
    end use;
    break on vi'above(VSP/GAIN), vi'above(VSN/GAIN);

    vo == vgain;

end architecture gain;
```

(a) OPAMP gain architecture

```
...
entity opamp_term is
  generic( GAIN : real  := 1.0e6 );
  port( terminal InP, InM, OutP : electrical;
        terminal tip, tim : electrical );
end entity opamp_term;

architecture gain_refined of opamp_term is

  quantity vi across InP to InM;
  quantity vo across io through OutP;
  quantity vgain : voltage;

  quantity vip  across
           tip to electrical'reference;
  quantity vim  across
           tim to electrical'reference;
begin

   if vi'above(vip/GAIN) use
     vgain == vip;
    elsif not vi'above(vim/GAIN) use
     vgain == vim;
    else
     vgain == GAIN*vi;
    end use;
    break on vi'above(vip/GAIN), vi'above(vim/GAIN);

    vo == vgain;

end architecture gain_refined;
```

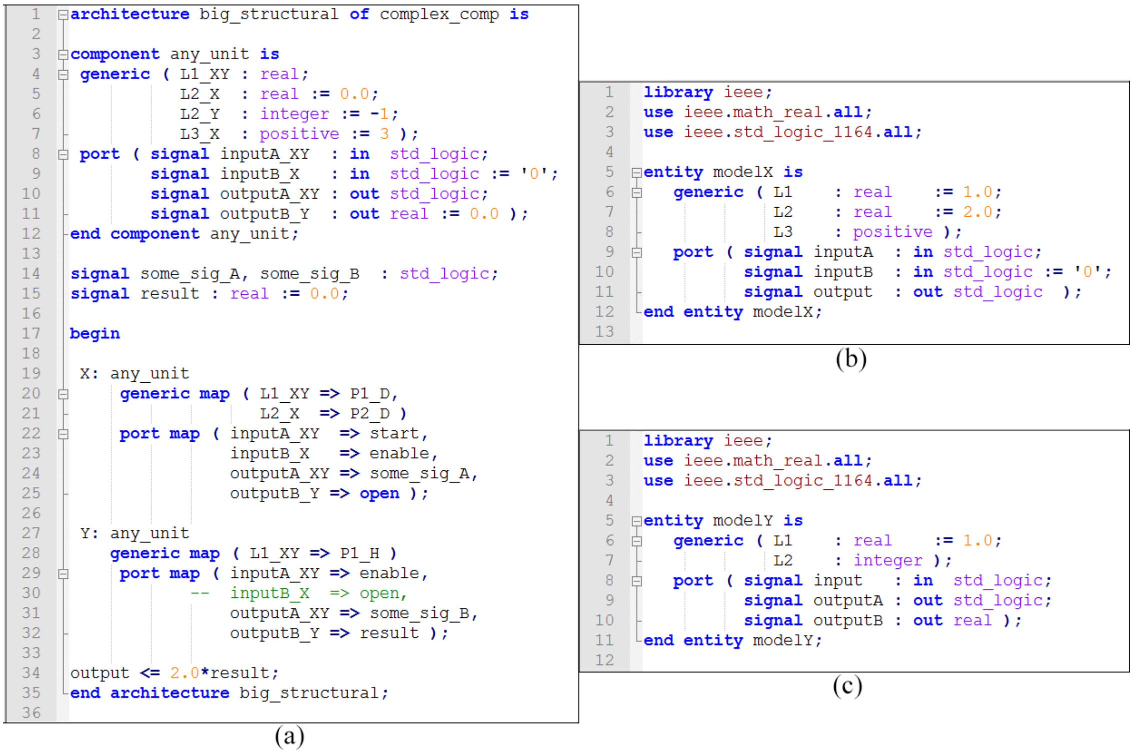
(b) OPAMP gain_refined architecture

Figure 3.11: Model refinement example, the OpAmp positive and negative supply voltage is given as (a) generic parameters or as (b) terminal ports.

As it is observed in this example, a component can be declared to instantiate both `opamp_gen` and `opamp_term` entities and their architectures. In general, the component declaration must comply with the following rules:

1. Do not declare generic parameters for the components unless the model is built using *hierarchical parameter binding*. In the latter case, the generic parameters of the related component must be declared equal to the union of all generic parameters of all related entities. In this case, all the parameters that could be unused in any component instance must have default parameter values in the component declaration.
2. The ports of the component must be equal to the union of all ports of all related entities.
3. All the input ports that could be unused in any component instance must have default values in the component declaration.
4. The unused ports in any component instance can be bound by using the keyword `open`, or they can be simply omitted. Likewise, it is recommended to use default values for the related signals or quantities of unused output ports.

In order to illustrate in a general fashion the rules mentioned above, let us consider the non-specific example shown in Figure 3.12. Suppose that `modelX` and `modelY` shown in Figure 3.12(b) and 3.12(c) respectively, are two different models of the same device with different amount and types of generic parameters and ports. Suppose that we want to use a single component to instantiate both `modelX` and `modelY` entities, the component (`any_unit`) is declared in the `big_structural` architecture of the `complex_comp` entity that belongs to the example VP previously introduced, see line 3 in Figure 3.12(a). This component has been created following the aforementioned rules.



```
 1 architecture big_structural of complex_comp is
 2
 3 component any_unit is
 4  generic ( L1_XY : real;
 5            L2_X  : real := 0.0;
 6            L2_Y  : integer := -1;
 7            L3_X  : positive := 3 );
 8  port ( signal inputA_XY  : in  std_logic;
 9         signal inputB_X   : in  std_logic := '0';
10         signal outputA_XY : out std_logic;
11         signal outputB_Y  : out real := 0.0 );
12 end component any_unit;
13
14 signal some_sig_A, some_sig_B  : std_logic;
15 signal result : real := 0.0;
16
17 begin
18
19  X: any_unit
20      generic map ( L1_XY => P1_D,
21                    L2_X  => P2_D )
22      port map ( inputA_XY  => start,
23                 inputB_X   => enable,
24                 outputA_XY => some_sig_A,
25                 outputB_Y => open );
26
27  Y: any_unit
28     generic map ( L1_XY => P1_H )
29      port map ( inputA_XY => enable,
30             --  inputB_X  => open,
31                 outputA_XY => some_sig_B,
32                 outputB_Y => result );
33
34 output <= 2.0*result;
35 end architecture big_structural;
36
```

(a)

```
library ieee;
use ieee.math_real.all;
use ieee.std_logic_1164.all;

entity modelX is
    generic ( L1      : real      := 1.0;
              L2      : real      := 2.0;
              L3      : positive );
    port ( signal inputA  : in std_logic;
           signal inputB  : in std_logic := '0';
           signal output  : out std_logic   );
end entity modelX;

```

(b)

```
library ieee;
use ieee.math_real.all;
use ieee.std_logic_1164.all;

entity modelY is
    generic ( L1      : real      := 1.0;
              L2      : integer );
    port ( signal input    : in  std_logic;
           signal outputA  : out std_logic;
           signal outputB  : out real );
end entity modelY;

```

(c)

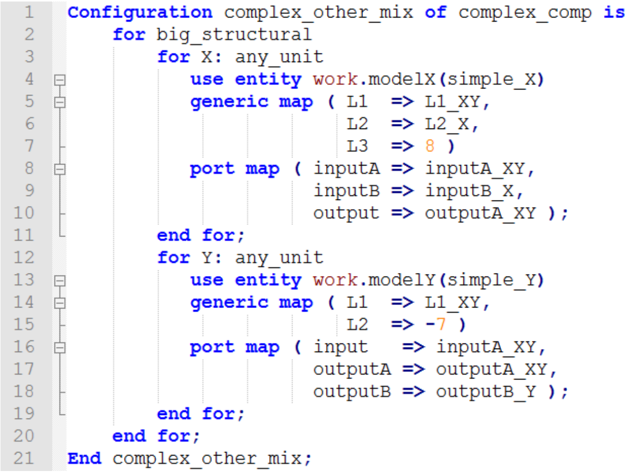
Figure 3.12: (a) `big_structural` architecture of the `complex_comp` entity. (b) `modelX` entity declaration. (c) `modelY` entity declaration.

Since it is required to bind the real generic parameters `P1_D`, `P2_D`, and `P1_H` of the `complex_comp` entity with the generic parameters `L1` and `L2` of `modelX`, and the parameter `L1` of `modelY` respectively, the component `any_unit` declares the union of all generic parameters of the `modelX` and `modelY`. If it is desired to set the generic parameters of the bound entity via generic mapping in a configuration, there is no need to define generic parameters in the component declaration. The parameter `L1_XY` of the `any_unit` component is the only one instantiated in both instances `X` and `Y` of `any_unit`, see lines 20 and 28 in Figure 3.12(a). For all the other parameters, it is mandatory to fix a default generic parameter in the component declaration[10]. The values of the generic parameters are finally set in the `complex_other_mix` configuration shown in Figure 3.13.

Likewise, the declared ports of the `any_unit` component are equal to the union of all ports

---

[10] Since the name of the component does not match with any of the bound entities, the default parameter values must be declared in the component declaration; otherwise, the default parameter values can be declared in the entity declaration.



```
1  Configuration complex_other_mix of complex_comp is
2      for big_structural
3          for X: any_unit
4             use entity work.modelX(simple_X)
5             generic map ( L1   => L1_XY,
6                           L2   => L2_X,
7                           L3   => 8 )
8             port map ( inputA => inputA_XY,
9                        inputB => inputB_X,
10                       output => outputA_XY );
11         end for;
12         for Y: any_unit
13            use entity work.modelY(simple_Y)
14            generic map ( L1   => L1_XY,
15                          L2   => -7 )
16            port map ( input    => inputA_XY,
17                       outputA => outputA_XY,
18                       outputB => outputB_Y );
19         end for;
20     end for;
21 End complex_other_mix;
```

Figure 3.13: Configuration of the `complex_comp` entity.

of `modelX` and `modelY` entities. `modelX` has two inputs and one output port, whereas `modelY` has one input and two output ports, see Figures 3.12(b) and 3.12(c). In Figure 3.12(a), we can observe that the port `inputB_X` does not need to be used in the component `Y` instantiation, see line 30. Therefore, this port can be omitted in the port mapping, or equivalently, this port can be bound by using the keyword `open`. Finally, in line 25, we can see that the output port `outputB_Y` is not used in the component `X` instance. Since the `result` signal, which is the one associated with the `outputB_Y` port, is used for the signal assignment shown in line 34, the `result` signal has a default value in its declaration as is shown in line 15. This is mandatory in the case that only `modelX` is instantiated.

To conclude, it can be observed the definitive generic and port mapping of the `any_unit` component instances in the configuration shown in Figure 3.13. In lines 5, 6, and 14 we observe that those generic parameters are bound to the respective generic parameters of the component; and therefore, the hierarchical binding is conserved. All the other parameters are numerically set in this configuration as it can be seen in lines 7 and 15. The port mapping is done straightforward; the respective ports of the bound entities are bound to the respective ports of the `any_unit` component.

## 3.6 The VP-Model Library

This section describes the objectives, the concepts, and the issues behind the classification and organization of model meta-information (or metadata). The ideas presented in this section represent the foundations of the proposed library of models of the VP-based modeling methodology, namely the VP-Model Library. The development, deployment, and structure of the VP-Model Library is presented in details in Appendix F. It is strongly advised to the reader to check this Appendix after reading this section, especially the VP-Model Library structure in Appendix F.2, which gives a complete explanation of what is the type of model meta-information stored and displayed by the VP-Model Library developed in this research.

As it has been previously mentioned, the VP-Model Library is a centralized database for storing, documenting, maintaining, and querying *modeling elements* at different abstraction levels. Its main goal is to provide an environment for maximizing the reuse of models among system and subsystem designers. Although the VP-Model Library supports the inclusion of models written in different languages, its structural organization is principally focused on supporting VHDL-AMS models. This structure facilitates the system design knowledge preservation and transfer by providing a method to store, document and share the IP related to the *virtual prototyping* activity. The proposed model library is mainly focused on supporting models for electrical power applications and embedded electronic design.

### 3.6.1 Issues and objectives

The ability to maximize the reuse of local or foreign IP is crucial to speed up the design and verification processes of CPS in any domain. Reducing time and costs in system development are the main advantages of the IP reuse when it is done in an effective way. This IP is represented here by *computational mockups* and their related verification models. Despite the benefits of the approach, there are several issues which currently complicate this process:

- Unawareness of existing models and their applications.
- Insufficient knowledge of modeling languages and simulation tools.
- Difficult model adoption due to the lack of proper documentation of the existing models.
- Lack of confidence for existent models. It is especially critical with foreign and complex designs.
- There are no tools for carrying out extensive and precise queries for existing models.
- Most of the *model implementations* elaborated for particular designs are application specific, which cannot be used directly for general purpose applications.

The VP-Model Library offers a solution to some of the problems mentioned above, particularly it provides an environment to find well-documented models and verification examples. This is a library that provides a clear view of the models, in addition to examples of their utilization in larger hierarchical models, and robust verification procedures implemented in *test benches*. The VP-Model Library contributes to facilitate the understanding of foreign models and increase the model trustworthiness that is required to maximize the reuse of models.

Although there are tool-specific library managers provided by different companies such as Cadence, Mentor Graphics, or Dolphin Integration, these cannot cope properly with the management of the models out of the software tool and they not offer an adequate solution for a sufficient comprehension of the models, their interrelations, and their usage, especially for complex designs.

The VP-Model Library is mainly dedicated to facilitating the *design space exploration* activity and gradual *model refinement* for system-level design. For this purpose, the models need to be well-documented and properly organized to support fast model queries. The user interface must be friendly and easy to understand. Additionally, the models contained in the library must be presented using an adequate summary to quickly perceive how the model is built and how it can be used.

In order to properly judge if a model is suitable for a specific design, the information of the model structure and its functionality must be clear. Likewise, since it is not always possible to count with tool-independent models, the VP-Model Library shall provide clear information about the tools that the models are expected to be compatible with. Moreover, the VP-Model Library must allow to categorize and link models sharing detailed criteria. In this way, the selection process among several potential models becomes more complete, and the choice of the best candidate becomes easier. The models stored in the library must be properly related to their verification models and files (i.e. *test benches* and *test cases*).

The VP-Model Library shall only provide *conditioned models*. These models can be described at different *abstraction levels*, and they can be used as references and/or building blocks of new designs. The criteria of VHDL-AMS *conditioned models* is given as follows:

- The source code is clear, well-written, and organized, the different design units can be easily identified.
- The model ports and parameters are correctly described and documented.
- All the functionalities and behaviors that are modeled in the *model implementations* are clearly described and documented.
- It is provided at least one *test bench* for verifying the different functionalities and behaviors that are modeled. The validity range of the variables and parameters of the model must be identified as better as possible; effective *test benches*/ *test cases* are vital to achieving this purpose, they must be properly documented.
- The *simulator setups* are correctly described and documented. Although this is not mandatory for simple models, it is desirable to describe *simulator setup* limitations if they have been identified in the verification procedure of complex models.

### 3.6.2 Model metadata

A *computational mockup* can be implemented in different ways according to its construction: for instance, it can be implemented in terms of a source code written in a determined modeling language, or by graphical representations such as schematic diagrams (e.g. RTL, Gate-level models) or by 2D/3D CAD geometrical models. Nonetheless, disregarding the type of *computational mockup* construction, the model by itself give few and ineffective information about the model scope, its semantics, its functionalities, and behaviors. Only model experts could extract some important information hidden in a model by only looking its source code or schematics. This becomes an impossible task the more complex and larger the model is. This is why the model meta-information or metadata come to be highly desired for effective model re-utilization.

One of the main issues for administrating model metadata is the lack of a consistent and proper definition of the information that must be included for a correct description of the model. In addition, the incapability to embed the model metadata within the model makes the issue more problematic. Some software tools such as COMSOL or LabVIEW for modeling and system design, offer a text field to freely add the description of the model. However, this is not much more different than adding comments to a source code in order to provide a description and explanation of the model. These strategies are not good enough to clearly understand important details of the model as the model complexity grows.

Previous attempts to classify the model metadata are reported in [24, 74], where two categories are defined: the **structural meta-information**, which describes *how the model is built*; and the **semantical meta-information**, which describe *how the model can be used*. The proposed structure of the VP-Model Library has been designed in order to answer these two questions. However, trying to classify important model meta-information such as the description of the port interface, the software tool compatibility, or the types of supported simulations, in structural or semantic meta-information does not bring any benefit to the design of a model library, instead, it creates confusion. For instance, the refinement levels proposed by the SAE J2546 standard [75] mixes several concepts such as different model interfaces, simulation types, non-ideal behaviors, and model capabilities; making this classification rather fuzzy and unpractical. Moreover, most of the important meta-information of a model have characteristics that can be classified in both structural and semantical kinds. For example, the verification meta-information of a *test bench* model can provide information about both how the model is built and how it can be used. By logical sequence, the model user first should understand how the model is built and how it computes the desired functionality and behaviors; afterwards, the user should analyze how the model can be implemented for a specific application.

Contrarily, the VP-Model Library has been focused on gathering all the essential aspects that are required to clearly and accurately describe a *component model* or a VP of any size and at any *refinement level*. The VP-Model Library is organized in three main database views (metadata interfaces): the **Model**, the **Test Bench**, and the **Custom Package** views, the detailed description of the database views is presented in **Appendix F.2**. The database views group the metadata of each model entry and their relationships with other model entries, with their verification models (*test benches*), and with other model dependencies, e.g. custom packages and configurations.

The VP-Model Library contains *component models* from very basic building blocks to more complex hierarchical system models, i.e. VPs. Each model entry in the library can have one or more *model implementations* at different refinement levels. This is true always that the model interface remains identical, i.e. if the port interface is different between two *model implementations*, this requires two different model entries in the library, even if these implementations model the same device/system. Additionally, each model entry can be linked to one or more *test benches* entries and/or Custom packages. Although a *test bench* can be composed of several models, it is normally designed to test only one specific model (the DUV).

## 3.7 Conclusions

The main theoretical premise behind VP-based design advocates that its continuous execution improves the design and verification of small-scale CPS by a sustained system-level modeling boosted by model reuse. This is why this work proposes mechanisms for the improvement of these two concepts. On the one hand, the VP-Modeling Guidelines facilitate the system-level modeling in VHDL-AMS and compatible languages. On the other hand, the VP-Model Library provides a way to maximize the reuse of models making the *virtual prototyping* task easier and faster.

The VP-based design methodology is based on a meet-in-the-middle approach for the execution of system-level modeling and verification. This approach is the basis of the functional system-level modeling and a gradual *model refinement* promoted in this work. In particular, the VP-Model Guidelines are focused on improving both the *modularity* and the scalability of VPs and their *component models*. The proper use of VHDL-AMS packages and configurations allow to improve the flexibility in VPs and to reduce the complexity in the manipulation and interconnection of models with large interfaces and parameters.

The ability to store, document, maintain, and query models in an effective way are the main features that the VP-Model Library must attain in order to maximize the reuse of models. It is important to clarify that most of the times, the models (their functions and algorithms), are designed for application specific purposes rather than general purpose applications. The most effective models for simulation are build based on a priori knowledge of their applications. Therefore, a complete and direct re-usability of models cannot be always 100% guaranteed. However, it is possible to reuse most of the IP (i.e. complete or partial models and functions) that has been previously designed for distinct applications. The VP-Model Library is proposed as a critical part of the VP-based design methodology, it defines a specific model metadata structure and library functionalities that contribute to achieving the desired model re-usability.

# 4 The Rogowski coil sensor system

The fundamental application project that has been treated during this doctoral research is presented in this chapter. The main goal of this work has been the identification of the issues, challenges, requirements, and opportunities of the VP-based design methodology. Especially, the connection between two co-related parts of the system design, i.e. high-level *design space exploration* at the system level and detailed low-level modeling of critical cross-domain interactions at the component level.

This chapter starts by describing the RogoCoil transducer and the conceptual overview of the complete RogoCoil sensor system that requires being designed. Later on, is presented a multi-domain (electro-thermal) *virtual prototyping* analysis for detailed subsystem design and verification of the RogoCoil sensor using 3D geometrical FEA and VHDL-AMS. Finally, the functional system level modeling and gradual *model refinement* concepts are used in the construction of the VP of the complete system and the VPs of the main signal processing subsystems, i.e. the self-calibration unit (SCU) and the current measurement unit (CMU). Although the RogoCoil sensor system has not been designed using the VP-based design methodology from the beginning, most of the VHDL-AMS modeling techniques proposed in the VP-Modeling Guidelines are illustrated in the presented examples.

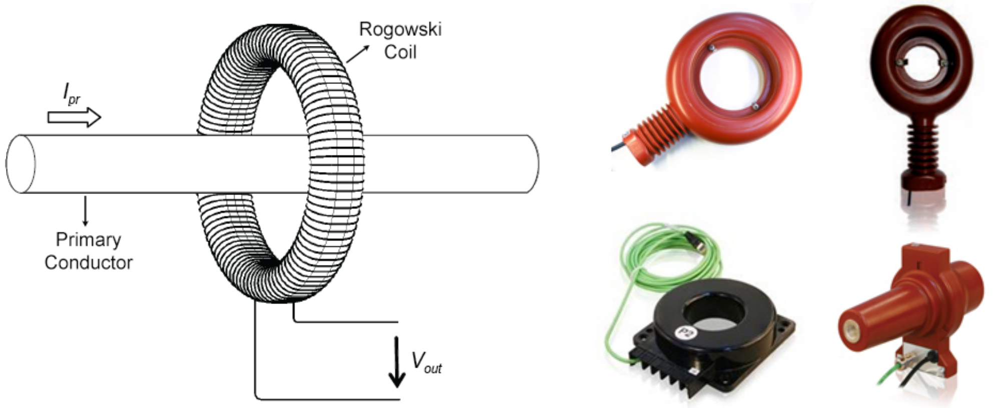
## 4.1 System description

A RogoCoil transducer is a toroidal coil wrapped around an air (non-ferromagnetic) core. The RogoCoil can be used as an electrical instrument for measuring alternating current; for that purpose, the RogoCoil is located around a primary conductor (also known as Busbar) whose current intensity is intended to be measured, see Figure 4.1(a). It the delivers a voltage ($V_{out}$)

proportional to the derivative of the AC primary current ($I_{pr}$) as follows:

$$V_{out} = M \cdot \frac{dI_{pr}}{dt} \tag{4.1}$$

where $M$ is mutual inductance of the RogoCoil.



(a) Rogowski coil current transducer (b) MV Rogowski coil current sensors

Figure 4.1: (a) A Rogowski coil transducer around a primary conductor. (b) Indoor RogoCoil current sensors for MV applications (ABB portfolio).

This type of sensor has many advantages and features which make it very attractive for alternating current measurement. The main RogoCoil advantages are listed as follows [76, 77]:

- High linearity and large dynamic range thanks to its non-ferromagnetic core.
- Effective galvanic isolation from high current and tensions (safety).
- Large overload withstand without damage.
- Current measurement in a wide range without saturation.
- Easy to use and easy to install.
- Low cost.
- Wide bandwidth, in a range from 0.1 Hz to 1 GHz.
- Excellent transient response.
- No power consumption from the main circuit.

The main problem of the RogoCoil sensor is its accuracy, which depends on multiple factors such as manufacturing tolerances, the sensor position with respect to the primary conductor, temperature drifts and the degradation of the sensor by aging. Therefore, methods to improve the accuracy of this sensor are highly desired. During this research, different self-calibration [78] techniques have been explored based on the RogoCoil electrical properties such as its resistance and inductance [79]. An estimation of the winding resistance ($R_{coil}$) and inductance ($L_{coil}$), allows to continuously estimate the RogoCoil sensitivity ($S$) which is directly related to its mutual inductance ($M$).

The conceptual overview of the complete RogoCoil sensor system is presented in Figure 4.2, which is a high-level representation of the complete system and the main signals at the subsystem interfaces. As it can be observed, the RogoCoil sensor system consists of 3 principal subsystems: the RogoCoil, the SCU, and the CMU. The RogoCoil receives a MV current signal (around 50 A to 8 kA) in the primary and produces a low voltage (tens of mV) analog signal which is processed by the electronics of the sensor system, called the intelligent electronic device (IED). And adequate signal processing algorithm executed by a hardware/software implementation in the IED, must improve the accuracy of commercial RogoCoils of the ABB portfolio, see some commercial product datasheet in Appendices A.3 and A.6 on pages 168 and 174 respectively.

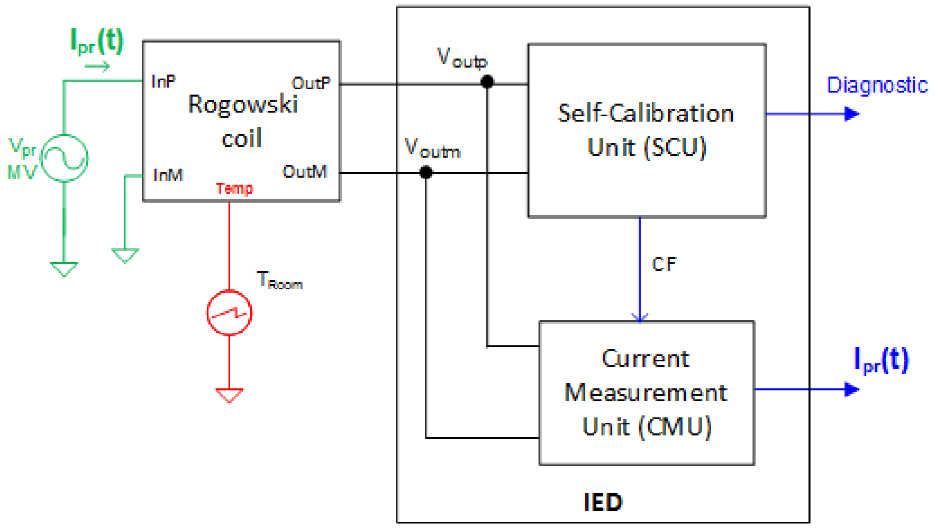


Figure 4.2: Conceptual diagram of the Rogowski coil sensor system. The different signal domains are: MV electrical signal (green), temperature signal (red), analog electronic signal (black), digital electronic signal (blue).

Whereas the CMU continuously measures the primary current ($I_{pr}(t)$) from the output voltage of the RogoCoil, the SCU continuously calculates and communicates to the CMU the RogoCoil correction factor ($CF$) that is required for the primary current measurement. Therefore, the $CF$ is not anymore a fixed value measured once in the factory and typed in the IED during installation. It becomes a dynamic value that can change during the system operation by temperature drifts, mechanical stress, or aging.

In this way, the accuracy of the current measurement mainly depends on the accuracy of the $CF$ estimation. It is important to note that the signals of the analog and digital electronics shall also be corrected in terms of gain errors, ensuring an entire instrumentation chain amplitude accuracy. The phase error is known to be in the case of the RogoCoil negligible as long as the passive components (i.e. resistors and capacitors) connected to the sensor are constant [80]. Indeed, the RogoCoil itself can be considered as a perfect derivation device in the frequency domain used for electrical measurement in power systems (50Hz - 1kHz) [77].

An important feature of this system is the signal exchange between the IED and the RogoCoil sensor is bidirectional, i.e. the SCU is not only passively reading the rated signal delivered by the sensor, but also it generates a reference (calibration) signal and sends it to the sensor. The SCU analyzes the response in order to determine the necessary electrical quantities (sensor's inductance and winding resistance), see Figure 4.3.

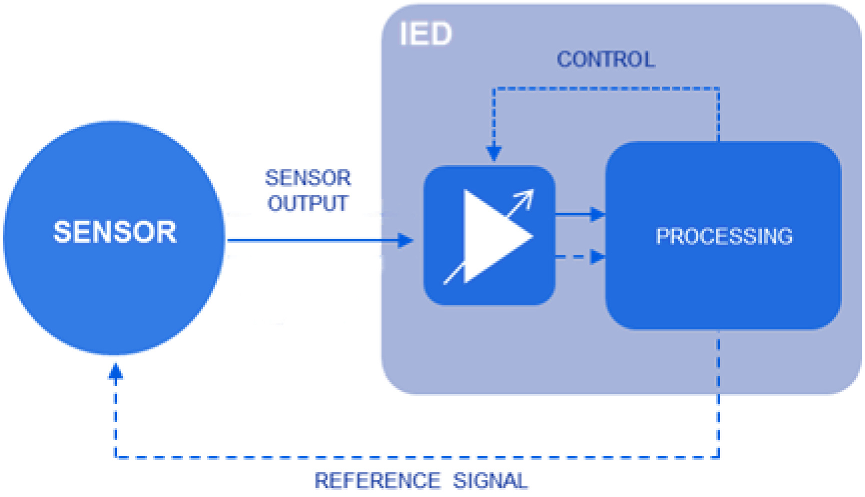


Figure 4.3: SCU closed loop measurement principle [80].

Some of the benefits of using embedded electronics to build smart self-calibrating RogoCoil sensors for MV power applications are listed as follows [80]:

**No in-factory calibration:** Once a RogoCoil is plugged to the IED, the SCU can calculate automatically the correction factor that is currently calculated manually and written in ABB's RogoCoil current sensors after its fabrication.

**Shortening lead time of *switchgears*:** A lead time reduction of *switchgears* is possible with self-calibration since it allows to reach the same accuracy class (IEC 60044-8 class 0.5) with all types of current sensors. The change of rated current (e.g. 80A, 250A, 500A, 1600A) does not affect the accuracy using a IED with a SCU since it will simply adapt its input gain stage without affecting the accuracy performance.

**Cheap material and components:** An automatic calculation of the correction factor allows to lower manufacturing costs of RogoCoils by using cheap commercial electronic components for the IED instrumentation. The poor accuracy (tolerance and drifts) is compensated by self-calibration.

**Simple installation:** That is, plug and play.

**Maintenance free:** No external calibration is needed neither during installation nor during lifetime (>20 years).

**Service:** The direct estimation of the coil inductance and coil resistance allows to calculate directly the coil temperature, this could be used to deliver data for installation diagnosis.

**Accuracy improvement:** The SCU allows us to guarantee the accuracy class over the full temperature range and during the entire lifetime of the sensor and its analog instrumentation electronics.

## 4.2 The electro-thermal modeling of the Rogowski coil sensor

As it was previously mentioned, one of the main factors that can cause variations in the accuracy of the sensor is the temperature of the RogoCoil, which mostly depends on the principal heat sources of the system, i.e. the room temperature and the Busbar self-heating. This is why we are interested in modeling the thermal and electrical effects concurrently. This section describes the theoretical assumptions and approximations done for modeling the electrical and thermal behaviors of the RogoCoil. Indeed, these multi-domain behaviors form an unidirectional coupling in which the thermal interactions affect the electrical variables of the system. It is shown in this section, how each of the 3 main parts of the RogoCoil sensor can be modeled independently and how they can be put together to build a system in a modular way. The analysis is focused on the heat transfer FEA modeling and the subsequent equation-based model extraction to implement a highly parametrized electro-thermal model of the RogoCoil in VHDL-AMS.

### 4.2.1 Rogowski coil sensor system assumptions

In order to model properly the thermal behavior of the RogoCoil sensor system, it is required to adequately describe the heat transfer physics of the coil and its surroundings. For that purpose, the following basic assumptions are defined in order to support the selection of the physics that rule the system behavior:

- The RogoCoil sensor system consists of a cylindrical primary conductor (Busbar) and a RogoCoil protected by a thick layer of epoxy material that is separated from the Busbar by few millimeters.

- The RogoCoil is designed for indoor applications. Therefore, the heat transfer of the RogoCoil with the surroundings is mainly caused by natural convection, i.e. there are no air flow sources around the RogoCoil system.

- The complete RogoCoil sensor system is contained in an air box at atmospheric pressure, in which no other heat sources are present.

Changing those basic assumptions will lead into changing either, some parameter values (case 1) or, defining a new physics for the system behavior (case 2). The first case is trivial, no major changes to the model are required; for example, the situation of inserting a heat source at a certain region of the RogoCoil sensor system surroundings. The second case is a bit more complex; for instance, the situation of inserting an air cooling system will require using forced convection physics (air velocity different than zero) instead of natural convection.

Taking into account the aforementioned basic assumptions, the electro-thermal model of the RogoCoil sensor system can be modeled by 3 parts as shown in Figure 4.4: the symmetric 3Loops-2Layers electrical model of the RogoCoil explained in subsection 4.2.2, the thermal model of the Busbar explained in subsection 4.2.4, and the thermal coupling between the Busbar and the RogoCoil explained in subsection 4.2.5. Both the Busbar thermal model and the thermal coupling model form the **Thermal Network** of the RogoCoil.

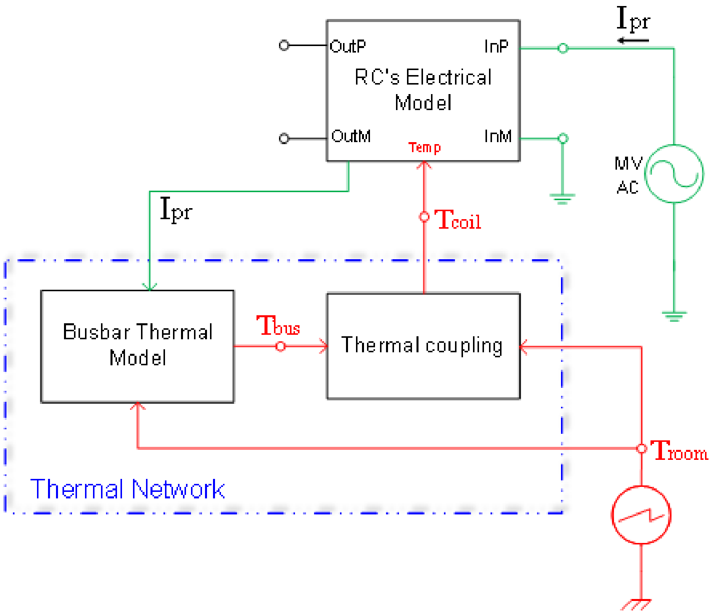


Figure 4.4: Block diagram of the electro-thermal Rogowski coil model (ETRCM). The model is an abstraction of the RogoCoil electrical behavior and the thermal behavior represented by the thermal network modules.

### 4.2.2 Electrical model of the Rogowski coil

The most abstract *functional model* of the RogoCoil, given by Equation 4.1, relates the output voltage of the transducer with the input current in the primary. This model could be adequate for functional verifications in a VP of a large system. However, it does not provide relevant information for the design of the SCU and the RogoCoil transducer itself. This is why it has been studied a series of lumped-element circuit models of the RogoCoil based in previous studies [81, 79, 82], to obtain the critical electrical variables for the sensor system design. The most refined RogoCoil model that was obtained in this research, called the *symmetric 3Loops model*, is shown in Figure 4.5. This is a parametric lumped-element electrical model with a distributed architecture developed in [79] and characterized in [82], see Figure 4.5.

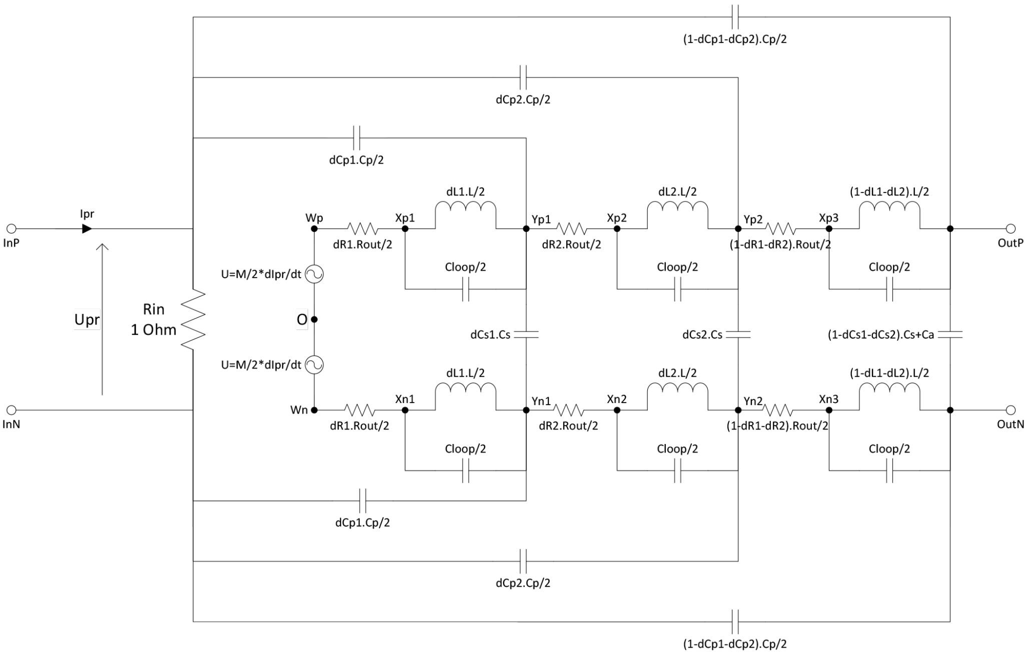
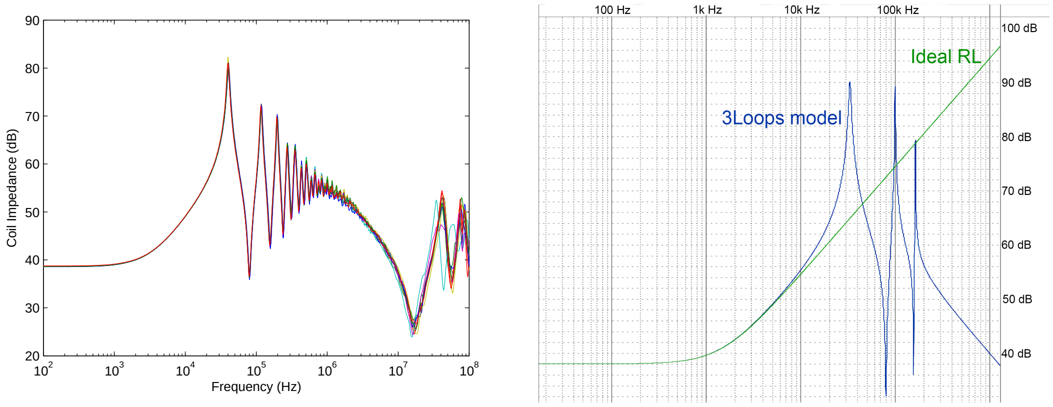


Figure 4.5: RogoCoil symmetric 3Loops model.

The RogoCoil symmetric 3Loops model consists of a simple resistance at the primary which simulates the Busbar conductor, and elementary RLC circuit distributed cells in the secondary. The symmetric 3Loops model is an approximation of the model presented in [81], where each RLC cell represents a loop of the RogoCoil. The value of the passive circuit elements of each cell determines a resonance frequency as shown in Figure 4.6. Consequently, the 3Loops model is only a valid approximation until the third resonance frequency as shown in Figure 4.6(b).

The values of the passive and active elements of the circuit are calculated in terms of geometrical and material PMPs particular to each type of RogoCoil. Additionally, it turns out that the KEVCR and KECA coils evaluated in this work are double winded. Therefore, the final refinement of the presented symmetric 3Loops model is called the symmetric 3Loops-2Layers



(a) Experimental: KEVCR impedance

(b) Simulation: KEVCR impedance

Figure 4.6: RogoCoil impedance experimental vs. simulation results: (a) Impedance measurements using an Agilent 4395A network analyzer for 10 samples of the KEVCR coil type, see Appendix A.3. (b) Frequency-domain simulation of the **3Loops-2Layers** model and the **Ideal RL** model (i.e. a resistor in series with and inductor) of the RogoCoil.

model, which has exactly the same circuit topology presented in Figure 4.5, but the equations to calculate the circuit elements are corrected by considering the mechanical and magnetic effects of the double layer. The analysis and the equations are presented in Appendix B.1.

Before being used for simulation, the symmetric 3Loops-2Layers model has been characterized according to the measurements of impedance obtained with an impedance analyzer, see the example in Figure 4.6(a). This study only considers two type of RogoCoil cores, see the KEVCR and KECA coil parameters in Appendices A.2 and A.5 on pages 168 and 174 respectively. The model is valid for few tens of kHz. However, for the self-calibration algorithm explained in section 4.3, we are only interested in operating below the first resonance frequency.

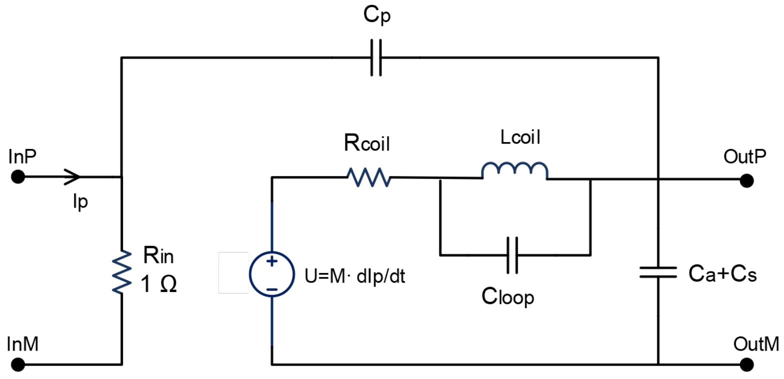
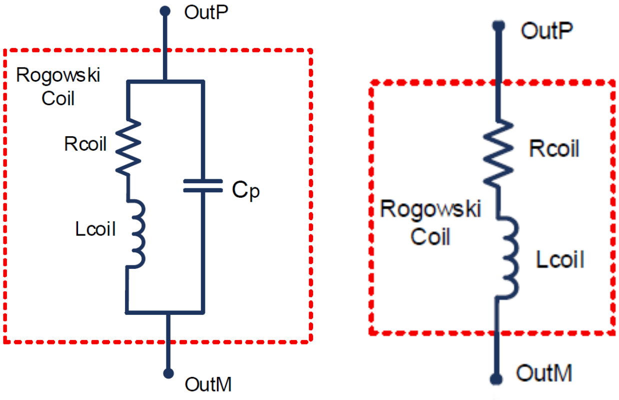


Figure 4.7: RogoCoil asymmetric 1Loop model.

We can observe from the secondary of the 3Loops circuit model, that this model can be simplified by only considering one RLC cell as it is shown in Figure 4.7. In this case, the asymmetric 1Loop model only is valid until the first resonance frequency. Moreover, we can further simplify this model considering the RogoCoil impedance behavior shown in Figure 4.6(b), where below the first resonance frequency, the RogoCoil can be approximated by a simple RLC or RL circuit. In fact, disregarding the primary Busbar, a RogoCoil can be seen as a simple inductor with a determined internal resistance and equivalent capacitance, see Figure 4.8(a). Considering that the equivalent capacitance is very low, at frequencies far below the first resonance frequency, the RogoCoil behaves as an ideal RL circuit, see Figure 4.8(b). The ideal RL approximation is taken into account for obtaining the self-calibration method presented in section 4.3.1.

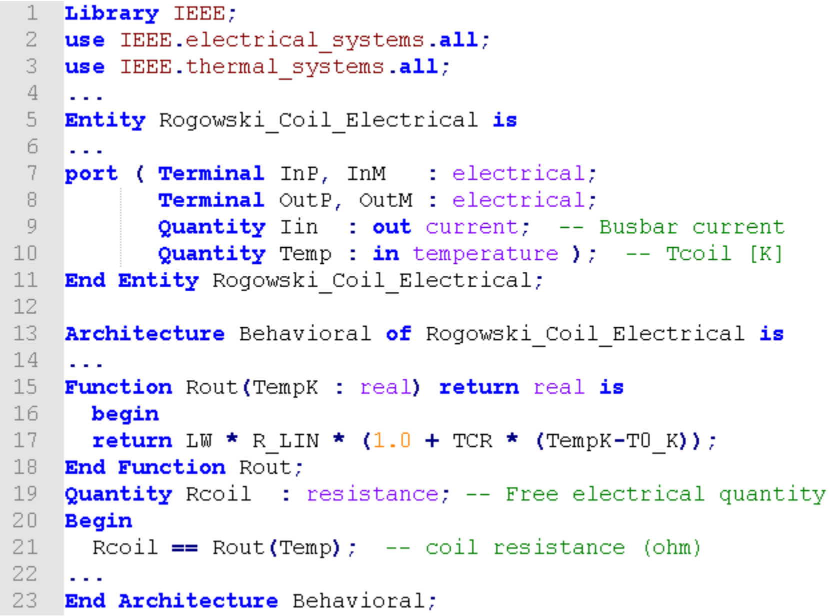


(a) Simple RLC circuit model. (b) Ideal RL circuit model.

Figure 4.8: RogoCoil approximated electrical models for frequencies far below the first resonance frequency.

### 4.2.3 Rogowski coil electrical model adaptation

As it can be observed in the equations presented in Appendix B.1, all the electrical variables of the symmetric 3Loops-2Layers RogoCoil model (i.e. the coil resistance, inductance, and capacitance), depend on the coil temperature. However, in previous studies of the KEVCR and KECA coils [79, 82], the temperature and the coil electrical variables have been considered static parameters in the RogoCoil model. This means that the temperature, the coil resistance ($R_{coil}$), and the coil inductance ($L_{coil}$) are fixed values in transient simulations. The author of this research has built a VHDL-AMS electrical model of the RogoCoil which supports electrical element circuit variables that depend on temperature changes dynamically. Thus, the model can simulate how the main electrical variables (i.e. the equivalent internal resistance and inductance) of the RogoCoil change according to temperature variations over the time. As it can be observed in the symmetrical 3Loops-2Layers model equations B.5 and B.7a on page 181, both $M$ and $C_l$ are depending on the temperature. Likewise, $R_{coil}$ and $L_{coil}$ are also temperature-dependent variables which are calculated dynamically by functions that depend on the temperature of the coil ($T_{coil}$), see the source code fraction for the calculation of $R_{coil}$ in Figure 4.9 as example.



```
 1 Library IEEE;
 2 use IEEE.electrical_systems.all;
 3 use IEEE.thermal_systems.all;
 4 ...
 5 Entity Rogowski_Coil_Electrical is
 6 ...
 7 port ( Terminal InP, InM   : electrical;
 8        Terminal OutP, OutM : electrical;
 9        Quantity Iin  : out current;  -- Busbar current
10        Quantity Temp : in temperature );  -- Tcoil [K]
11 End Entity Rogowski_Coil_Electrical;
12
13 Architecture Behavioral of Rogowski_Coil_Electrical is
14 ...
15 Function Rout(TempK : real) return real is
16   begin
17   return LW * R_LIN * (1.0 + TCR * (TempK-T0_K));
18 End Function Rout;
19 Quantity Rcoil  : resistance; -- Free electrical quantity
20 Begin
21   Rcoil == Rout(Temp);  -- coil resistance (ohm)
22 ...
23 End Architecture Behavioral;
```

Figure 4.9: Fraction of the VHDL-AMS code of the RogoCoil's electrical model. `LW` is the wire length in meters, `R_LIN` is the wire resistance per meter, `TCR` is the temperature coefficient of resistance of the wire and `T0_K` is the reference temperature in Kelvin.

Figure 4.4 shows the proposed RogoCoil model adaptation, called the electro-thermal Rogowski coil model (ETRCM). The electrical primary input ports of the RogoCoil (`InP` and `InM`) are modeled by two electrical terminals connected by a resistor of 1 Ω. In this way, the magnitude of the current through the Busbar ($I_{pr}$) can be controlled by a simple AC voltage source. By magnetic induction, $I_{pr}$ causes an AC voltage of the same frequency but only of few tens of mili-Volts between the electrical output of the RogoCoil (`OutP` and `OutM`). Additionally, $I_{pr}$, which is used to calculate the Joule heating of the Busbar, is injected to the thermal model of the Busbar by using an output quantity port, see the line 9 in Figure 4.9.

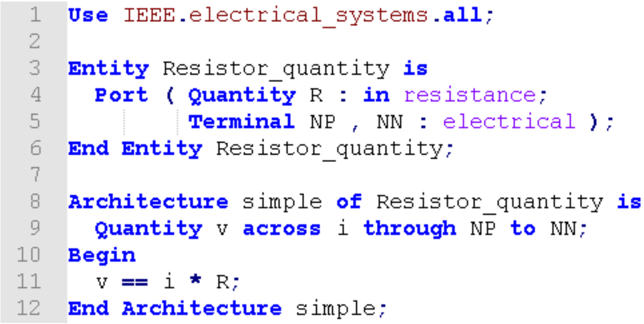
The coil resistance, the mutual inductance, and the winding capacitance of the coil are temperature dependent as it can be observed in Equations B.4, B.5, and B.7a respectively. Typically, the temperature, like other cross-domain variables, is included in electrical circuit models as a constant parameter. However, this approach is not convenient for modeling properly the multiphysics interaction between electrical and thermal quantities. The author of this research propose to use VHDL-AMS quantities for both thermal and electrical variables; consequently, the model allows to estimate dynamically the effect of the temperature into the RogoCoil internal impedances and the coil mutual inductance.

One can consider two different options in VHDL-AMS for including dynamic effects of the temperature in the electrical model of the RogoCoil as follows:

**Conservative:** This option consists in using thermal terminals. The VHDL-AMS terminals guarantee the energy conservation in its related quantities. Therefore, a bidirectional coupling between the electrical and thermal quantities can be simulated. However, we do not need to consider the RogoCoil self-heating, since the current inside the RogoCoil winding is low enough, in the order of few mA.

**Acausal:** This option consists in including the temperature using an input quantity port as is shown in the code shown in Figure 4.9. In this way, the temperature of the RogoCoil is calculated in the thermal network and simply passed to the electrical model of the RogoCoil in order to calculate the aforementioned temperature dependent electrical variables.

Figure 4.9 shows how $R_{coil}$ is calculated through the function `Rout`, which changes with the temperature given by the quantity port `Temp`. Likewise, the dynamic value of the internal lumped impedances can be transferred via quantity ports; a VHDL-AMS code example is given for a simple resistor model shown in Figure 4.10.



```
Use IEEE.electrical_systems.all;

Entity Resistor_quantity is
  Port ( Quantity R : in resistance;
         Terminal NP , NN : electrical );
End Entity Resistor_quantity;

Architecture simple of Resistor_quantity is
  Quantity v across i through NP to NN;
Begin
  v == i * R;
End Architecture simple;
```

Figure 4.10: VHDL-AMS dynamic resistor model.

### 4.2.4 Busbar thermal model

Calculating the temperature of a power conductor has been reported in detail in the literature, see [83] and [84]. The temperature of the conductor ($T$) can be obtained by the conductor's heat balance differential equation as follows:

$$m_c \cdot C_p \cdot \frac{dT}{dt} = Q_J + Q_M - Q_R - Q_C \tag{4.2}$$

where $m_c$ is the conductor's mass per unit length, $C_p$ is the specific heat capacity at constant pressure, $Q_J$ is the Joule heating, $Q_M$ is the heat generated by magnetic losses (i.e. skin and spiral effects), $Q_R$ and $Q_C$ are the energy loss by radiation and convection respectively.

The Busbar thermal model can be specified by means of an equivalent lumped-element circuit analogy or by using directly the differential equations in the model. As the VHDL-AMS language allows to directly express the equations in the model, is highly recommended to use the equation-based approach. The equivalent lumped-element circuit approach introduces unnecessary circuit bias which might lead to inaccuracies in simulation. The approximation proposed by the IEEE Standard 738-2006 [84] is here used to implement the thermal model of the Busbar. The detailed physical conditions, assumptions, and the resulted expressions used in Equation 4.2 are given in Appendix B.2.

### 4.2.5 Thermal coupling model

This model describes the thermal coupling between the Busbar and the RogoCoil, i.e. it represents the heat transfer from the Busbar surface to the environment passing through the RogoCoil geometry. Therefore, the effective temperature of the RogoCoil ($T_{coil}$), is calculated by the thermal coupling model considering only the two principal heat sources, i.e. the Busbar temperature ($T_{bus}$) and the room temperature ($T_{room}$). This problem is classically simplified by considering the thermal coupling as a thermal circuit with 2 thermal resistors connected in series: the first thermal resistor ($R_{CRc}$) symbolizes the thermal resistance between the Busbar and the RogoCoil; the second thermal resistor ($R_{RcA}$) symbolizes the thermal resistance between the RogoCoil and the ambient, see Figure 4.11.

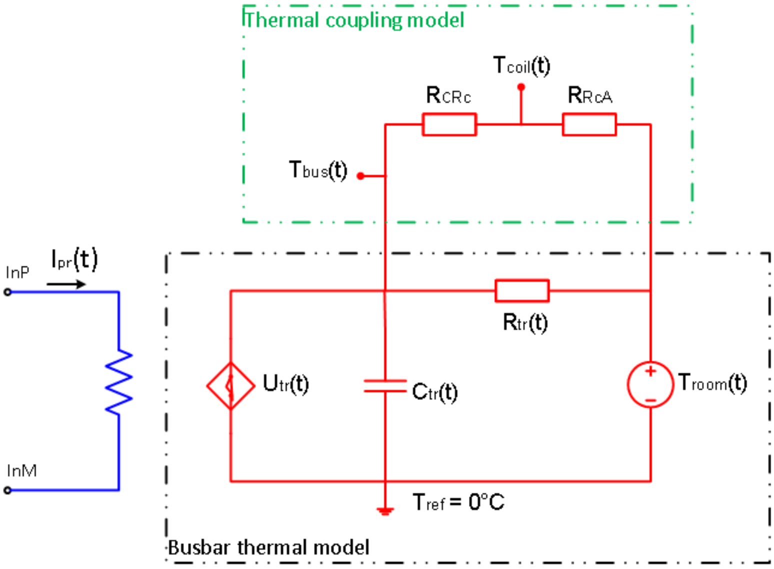


Figure 4.11: Equivalent lumped-element circuit model of the RogoCoil's thermal network. The expressions for the elements of the equivalent Busbar thermal circuit are given in Appendix B.2.3 on page 186.

By specifically defining $T_{coil}$ as the temperature of the RogoCoil core, $R_{CRc}$ represents all the thermal resistances generated by the layers from the Busbar surface and the coil's core. Likewise, $R_{RcA}$ represents all the thermal resistances of the layers from the coil's core to the environment. Under this approach, $T_{coil}$ can be determined as follows:

$$T_{coil} = \frac{R_{CRc} \cdot T_{room} + R_{RcA} \cdot T_{bus}}{R_{CRc} + R_{RcA}} = \frac{\varphi_R \cdot T_{room} + T_{bus}}{\varphi_R + 1} \tag{4.3}$$

where $\varphi_R$ is the ratio between the thermal resistances $R_{CRc}/R_{RcA}$. The temperature can be given in ($^{o}$C) or in (K) and the thermal resistance in ($^{o}$C/W) or in (K/W) accordingly.

The problem of the simplification shown in Equation 4.3 lies in the difficulty to calculate accurately the values of the two considered thermal resistances. In fact, is not required to calculate each thermal resistance independently, but the ratio between them is the important value. However, this simplification does not help to determine such ratio. The challenge at this point is to be able to obtain a parametric expression for the thermal coupling, i.e. a $T_{coil}$ equation as a function of $T_{bus}$, $T_{room}$, and other geometric and material parameters of the RogoCoil. One of the main problems for obtaining a direct parametric equation for the RogoCoil temperature relies on the geometry of the system. The heat transfer equations need to be solved in 3 dimensions according to the RogoCoil sensor system geometry, both in steady-state and in time-domain.

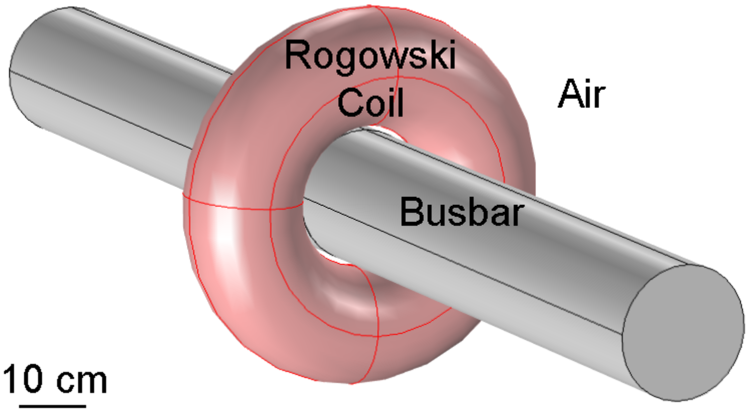


Figure 4.12: Basic 3D geometry of the RogoCoil sensor system. Here is shown the cylindrical Busbar conductor and the toroidal RogoCoil around the Busbar. The RogoCoil core is protected by an epoxy layer here shown in red.

Figure 4.12 shows the basic 3D geometry of the RogoCoil sensor system. Modeling this problem directly in VHDL-AMS is very complicated and unpractical due to the complicated geometry of the system. However, approximations can be done in order to achieve this purpose. One of these approximations consists in reducing the whole system to a one-dimensional heat flow radial system, which is explained in [85]. In this case, is required to assume that heat flux is only present in the radial direction, see Figure 4.13.

The issues with the radial unidimensional reduction are basically two: Firstly, in order to have a heat flux only available in the radial direction, is required that the length $L$ is much greater than the larger radius $r_4$, see Figure 4.13. This means that the height of the RogoCoil (both core and exterior) must be much greater than the radius of the coil, which is not the case for both KEVCR and KECA coils. This approximation is only valid for the Busbar. Secondly, the approximation only takes into account static equations for heat transfer by conduction in solids, yet it is neither considering time dependency nor heat transfer by convection. Natural

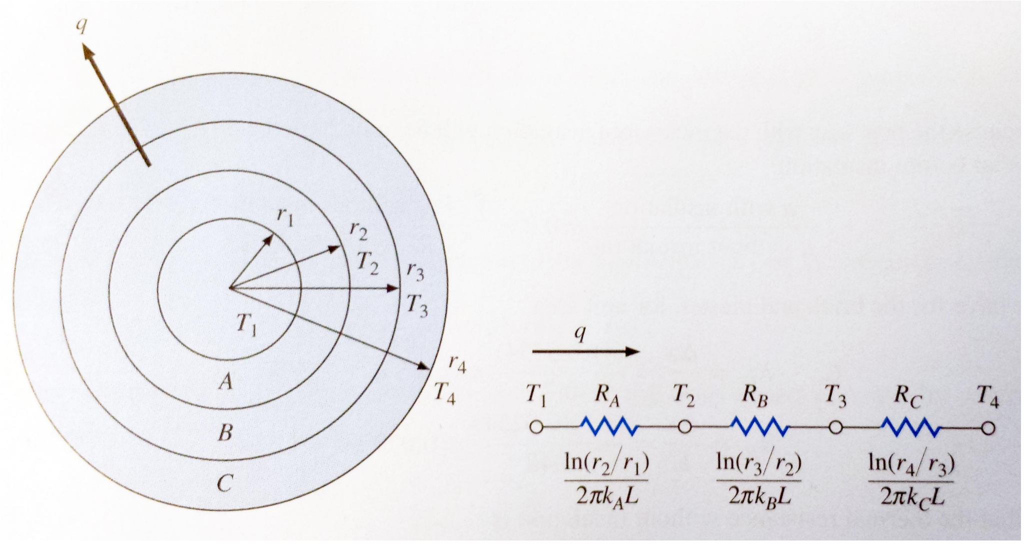

q
$r_1$
$r_2$
$T_2$
$r_3$
$T_3$
$T_1$
$r_4$
$T_4$
A
B
C
q
$T_1$
$R_A$
$T_2$
$R_B$
$T_3$
$R_C$
$T_4$
$\frac{\ln(r_2/r_1)}{2\pi k_A L}$
$\frac{\ln(r_3/r_2)}{2\pi k_B L}$
$\frac{\ln(r_4/r_3)}{2\pi k_C L}$

Figure 4.13: Unidimensional heat flow through multiple cylindrical sections with the electrical circuit analogy. Where $q$ is the heat flux, $k$ is the material thermal conductance and $L$ is the length of the cylinder. The image is taken from [85].

convection is important in air gaps, the larger the gap, the faster the temperature decrease. We are specially interested in observing the thermal effect of the air gap between the Busbar and the RogoCoil epoxy surface ($A_{gap}$).

Considering the limitations of the lumped-element thermal circuit and the unidimensional radial heat flow approximations, the author of this thesis has used a more accurate modeling approach to model the heat transfer of the RogoCoil sensor using a different *simulation framework*: a geometrical 3D model implemented in FEA software, COMSOL Multiphysics®. Heat transfer problems among other multiphysics interactions related to energy and mass transfer are commonly modeled and solved using FEA software. In this study, FEA modeling is used for two purposes: Firstly, to obtain a parametric equation-based model of the thermal coupling block that can be implemented in VHDL-AMS. Secondly, to serve as a validation reference for the complete VHDL-AMS electro-thermal model of the RogoCoil sensor. This is especially important since accurate measurements of $T_{coil}$ are complicated to obtain by experimental measurements.

### 4.2.6 Heat transfer modeling

Building a geometrical heat transfer FEA model in commercial software tools such as COMSOL Multiphysics®requires several steps. First, define the proper outer boundary of the system. Second, choose the correct physics. Third, build the geometry. Fourth, define and assign materials and other parameters. Fifth, define initial values, domain, and internal boundary conditions. And finally, mesh the geometry. Finally, the model can be simulated and analyzed. By following these steps, the heat transfer model of the RogoCoil is here explained.

#### 4.2.6.1 Outer boundary conditions

The first question to answer before starting to build this type of models is: where is the outer boundary of the system? At some distance, is needed to consider our thermal system closed. In general, the description of the system must be chosen in such a way that it resolves the fine structure of the model only to the degree of interest. Therefore, as is mentioned in section 4.2.1, is assumed that the RogoCoil sensor is contained in an air box at atmospheric pressure. The three fundamental boundary conditions to take into account at this level are:

1. Prescribed Temperature (Dirichlet condition):

   $$T = T_0 \tag{4.4}$$

   In this condition, the temperature of the boundary wall ($T$) is fixed at certain constant value $T_0$.

2. Prescribed Normal Flux (Neumann condition):

   $$\mathbf{n} \cdot \mathbf{q} = q_0 \tag{4.5}$$

   In this condition, the heat flux ($\mathbf{q}$) is normal at the boundary and is fixed at certain constant value $q_0$.

3. Mix of both Temperature and Normal Flux (Robin-Cauchy condition):

   $$\mathbf{n} \cdot \mathbf{q} = h(T_{ext} - T) \tag{4.6}$$

   where $\mathbf{n} \cdot \mathbf{q}$ is the heat flux normal at the boundary wall, $h$ is the heat transfer coefficient and $T_{ext}$ is the external temperature.

The selected boundary condition for the RogoCoil sensor model is the Robin-Cauchy condition due to it represents a mix of the two first conditions. In this way, Equation 4.6 is used with $h = 15\left(\frac{W}{m^2K}\right)$, this is a typical value of the heat transfer coefficient for free convection in air.

#### 4.2.6.2 Physics

The next step consists in determining the type of heat transfer processes which appear in the RogoCoil sensor and its environment. The heat transfer processes are classified in conduction, convection, radiation, advection, phase change, and multiphysics [86]. Taking into account the modeling assumptions, the dominant heat transfer processes are conduction in solids (for the RogoCoil and the Busbar) and free natural convection in the surrounding air. Hence, a complete thermal model of the RogoCoil sensor includes the following equations:

- Heat transfer equation in solids:

$$\rho C_p \frac{\partial T}{\partial t} + \rho C_p \mathbf{u} \cdot \nabla T = \nabla T \cdot (k \nabla T) + Q \tag{4.7}$$

- Heat convection-diffusion equation:

$$\rho C_p \frac{\partial T}{\partial t} - \nabla T \cdot (k \nabla T) = Q - \rho C_p \mathbf{u} \nabla T \tag{4.8}$$

- Navier-Stokes for free convection - Non-Isothermal flow (It contains the fully compressible formulation of the continuity and momentum equations):

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{4.9a}$$

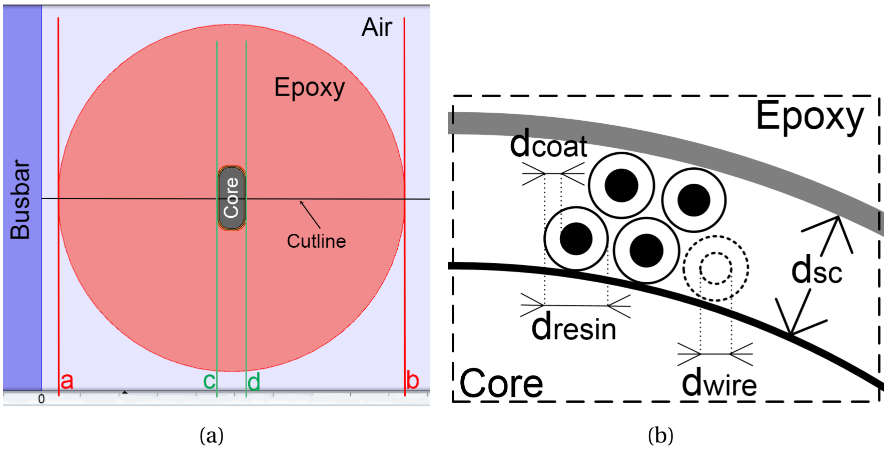
$$\rho \frac{\partial \mathbf{u}}{\partial t} + \rho \mathbf{u} \cdot \nabla \mathbf{u} = \nabla \cdot \left[ -p\mathbf{I} + \mu \left( \nabla \mathbf{u} + (\nabla \mathbf{u})^T \right) - \frac{2}{3} \mu (\nabla \mathbf{u}) \mathbf{I} \right] + \Delta \rho(T) g \tag{4.9b}$$

where $\rho$ is the fluid density (kg/m$^3$), $C_p$ is the specific heat capacity at constant pressure (J/(kg·K)), $T$ is the absolute temperature (K), $\mathbf{u}$ is the velocity vector (m/s), $k$ is the thermal conductivity (W/(m·K)), $\mu$ is the dynamic viscosity (Pa·s), $g$ is the gravity force (m/s$^2$) and $Q$ contains all the heat sources (W/m$^3$) other than viscous heating. A conjugated heat transfer (CHT) model solves the system through a bidirectional coupling of equations 4.8, 4.9a, and 4.9b. This model can only be solved as time-dependent model since there is no steady-state due to the air flow is continuously fluctuating around the RogoCoil and the Busbar, as the Navier-Stokes equation dictates. However, by ignoring the natural convection flow, i.e. zero air velocity ($\mathbf{u} = 0$), the system can be solved by steady-state simulations only using the convection-diffusion equation for modeling the heat transfer in the air. This approximation is called the simplified heat transfer (SHT) model.

#### 4.2.6.3 Geometry

The next step consists in creating the geometry of the system. The high symmetry of the geometry and the heat transfer process (i.e. in a radial direction from the Busbar), allow us to simplify the geometry of the model as is shown in Figure 4.14(a). Consequently, the number of elements is reduced and the performance of the simulation is improved. By revolving the 2D axisymmetric geometry around its symmetry axis (leftmost vertical line), the 3D geometry of the RogoCoil shown in Figure 4.12 on page 65 is reconstructed.

In a more detailed view around the RogoCoil core, we can observe how the RogoCoil sensor is wired in Figure 4.14(b). The RogoCoil has a double winding of copper wire which is insulated by a thin coating of resin. In Figure 4.15 it can be observed how the double winding of copper wire is approximated. Instead of recreating geometrically the real winding around the RogoCoil core, the wire and its surroundings are modeled as layers. The author of this thesis has made 2 layers of copper with a thickness equal to the copper wire diameter ($d_{wire}$). Despite the fact



(a) (b)

Figure 4.14: (a) 2D Axisymmetric geometry of the RogoCoil sensor system (cross-sectional view zoom). (b) Cross-sectional view of the physical Rogowski coil winding (not scaled).

that the thin resin coating of the copper wire does not appear in this geometry approximation, its effect is properly considered by using a boundary condition at the double layer interfaces called '**thin thermally resistive layer**', more details are given in section 4.2.6.5 on page 71.

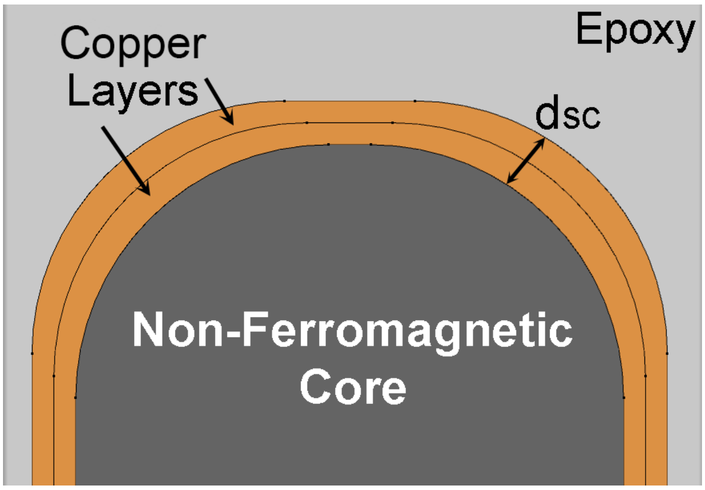


Figure 4.15: Cross-sectional view of the 2D axisymmetric RogoCoil model geometry, zoom of the core region in Figure 4.14(a).

### 4.2.6.4 Materials and Parameters

The constant parameter values and the equations used for the material definition in the heat transfer model are taken from COMSOL material browser and from standard material libraries such as [87]. A total of 6 materials are used to build the model, their principal properties are described as follows:

1. Air:[1] This is the material applied to the surroundings of the RogoCoil sensor system as is shown in Figure 4.14(a).
    - Ratio of specific heats ($\gamma$): $\gamma = 1.4$
    - Dynamic viscosity ($\mu$): $\mu[Pa \cdot s] = -8.38 * 10^{-7} + 8.357 * 10^{-8} * T - 7.694 * 10^{-11} * T^2 + 4.64 * 10^{-14} * T^3 - 1.06585 * 10^{-17} * T^4$
    - Heat Capacity at constant pressure ($C_p$): $C_p[\frac{J}{kg \cdot K}] = 1047.6 - 0.37 * T + 9.45 * 10^{-4} * T^2 - 6.024 * 10^{-7} * T^3 + 1.286 * 10^{-10} * T^4$
    - Density ($\rho$): $\rho[\frac{kg}{m^3}] = p * 0.02897/(8.314 * T)$
    - Thermal conductivity ($k$): $k[\frac{W}{m \cdot K}] = -0.00227583 + 1.15 * 10^{-4} * T - 7.9 * 10^{-8} * T^2 + 4.117 * 10^{-11} * T^3 - 7.4 * 10^{-15} * T^4$
2. Aluminum: This is the material selected for the Busbar conductor.
    - Density ($\rho$): $\rho[\frac{kg}{m^3}] = 2700$
    - Thermal conductivity ($k$): $k[\frac{W}{m \cdot K}] = 160$
    - Heat Capacity at constant pressure ($C_p$): $C_p[\frac{J}{kg \cdot K}] = 900$
3. Alumina, alpha $Al_2O_3$ 99.5%: This is the material used in the non-ferromagnetic *RC* core.
    - Density ($\rho$): $\rho[\frac{kg}{m^3}] = 3900$
    - Thermal conductivity ($k$): $k[\frac{W}{m \cdot K}] = 30$
    - Heat Capacity at constant pressure ($C_p$): $C_p[\frac{J}{kg \cdot K}] = 880$
4. Copper: This is the material used in the wire layers around the RogoCoil core.
    - Density ($\rho$): $\rho[\frac{kg}{m^3}] = 8700$
    - Thermal conductivity ($k$): $k[\frac{W}{m \cdot K}] = 400$
    - Heat Capacity at constant pressure ($C_p$): $C_p[\frac{J}{kg \cdot K}] = 385$
5. Wire resin: This is the material used at the boundaries of the wire layers around the RogoCoil core.
    - Thermal conductivity ($k$): $k[\frac{W}{m \cdot K}] = 0.45$
6. Epoxy cure resin: This is the material used as insulator around the *RC*.
    - Density ($\rho$): $\rho[\frac{kg}{m^3}] = 0.00133$
    - Thermal conductivity ($k$): $k[\frac{W}{m \cdot K}] = 0.541$
    - Heat Capacity at constant pressure ($C_p$): $C_p[\frac{J}{kg \cdot K}] = 1315$

---

[1] In the equations $T$ is the absolute temperature and $p$ is the atmospheric pressure. All the numbers in the equations have been approximated

Table 4.1 show the geometrical and material parameters used in the 3D heat transfer model for the KEVCR coil. It is important to mention that all the parameters which are represented by constant values can be swept in order to study the effect of different values in $T_{coil}$. This study only considers different scenarios for `Tbus`, `Troom`, and `Agap` variations.

| Name | Value/Expression | Description |
|---|---|---|
| `D1` | 0.1[m] | Primary conductor (Busbar) diameter |
| `L1` | 1[m] | Primary conductor length |
| `D` | 204.9[mm] | Outer core diameter |
| `d` | 191.9[mm] | Inner core diameter |
| `h_core` | 16[mm] | Core height |
| `R` | 3[mm] | Core curvature |
| `dsc` | 0.52[mm] | Distance from shield to core |
| `Wcore` | `(D-d)/2` [m] | Core width |
| `Ccore` | `d/2+Wcore/2` [m] | Core center |
| `Repoxy` | 45[mm] | Radious of the Coil epoxy |
| `Agap` | (49.2-`Repoxy`)[mm] | Air gap between the Busbar and the RC external layer |
| `dwire` | 0.224[mm] | Copper wire diameter |
| `Tbus` | 100[$^o$C] | Temperature of the Busbar surface |
| `Troom` | 25[$^o$C] | Room temperature |
| `h_thermal` | 15[W/($m^2 \cdot K$)] | Convective heat transfer coefficient |
| `dcoat` | `(dsc/2-dwire)/2` | Effective thickness of the wire coating |
| `F_fact` | 1.908 | Filling factor of the winding |

Table 4.1: KEVCR global parameters.

#### 4.2.6.5 Domain and internal boundary conditions

The next step for setting up the 3D heat transfer model of the RogoCoil sensor consists in the proper application of the heat transfer equations to the different domains and boundaries in the geometry. It is also required to set up initial values and check any other default properties. First of all, it must be assigned the correct physics (i.e. the heat transfer equations for fluids and solids) to the different domains in the system. Secondly, the initial conditions are:

- Velocity of the air = 0 [m/s] in any direction.
- Pressure = 1 [atm].
- Initial temperature = `Troom` in every domain.

As it was mentioned previously on page 67, the Robin-Cauchy condition (Equation 4.6) is used as the outer boundary condition. Thermal insulation is the condition at the upper and lower boundaries of the Busbar, i.e. $-\mathbf{n} \cdot (-k\nabla T) = 0$; the left boundary of the Busbar uses the condition called Axial Symmetry; whilst the right boundary of the Busbar, which represents the Busbar interface with the air, uses a constant value as boundary condition by using the `Tbus` parameter. Although the Joule heating effect can also be modeled in the 3D heat transfer FEA model, there is no need to include this effect for obtaining the equation-based thermal coupling model in VHDL-AMS. The Joule heating will be considered in the complete electro-thermal model as is described in section 4.2.4 on page 63.

The final temperature and the time response of the RogoCoil sensor are the variables of interest in the two versions of the 3D heat transfer model, i.e. the CHT and the SHT models. Both `Tbus` and `Troom` are swept from -5[$^o$C] to 200[$^o$C] and from -10[$^o$C] to 85[$^o$C] respectively, for steady-state and time-dependent simulations. Moreover, only for the CHT model, the boundary condition around the air boundaries is '**No Slip**', which means that the velocity of the air is zero at the boundaries.

An important boundary condition to establish is the thin thermally resistive layer condition at the double layer interfaces, which is used to model the effect of the resin coating of the copper wire. In general terms, the thickness ($d_s$) and thermal conductivity ($k_s$) of a resistive material located at the boundary are used to define the thermal resistance ($R_s$) as follows:

$$R_s = \frac{d_s}{k_s} \tag{4.10}$$

Likewise, the heat flux across this thin thermally resistive layer is defined by:

$$-\mathbf{n}_d \cdot (-k_d \nabla T_d) = -k_s \frac{T_u - T_d}{d_s} \tag{4.11a}$$

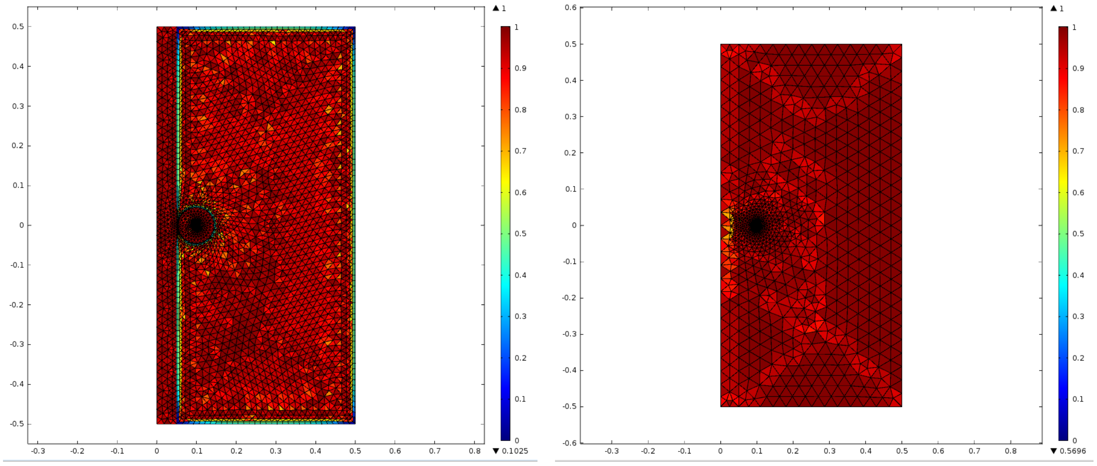
$$-\mathbf{n}_u \cdot (-k_u \nabla T_u) = -k_s \frac{T_d - T_u}{d_s} \tag{4.11b}$$

where the $u$ and $d$ subscripts refer to the upside and the downside of the boundary, respectively.

Therefore, the thermal resistance at the external boundaries of the double layer of copper is given by Equation 4.10, where $d_s$ is equal to `dcoat` given in Table 4.1, and $k_s$ is the thermal conductivity of the insulating resin coat. Similarly, the internal boundary of the double layer also uses the thin thermally resistive layer condition. However, the thickness of this layer ($d_{coat\_int}$) is not exactly two times `dcoat`, since the second winding of copper wire is not perfectly aligned above the first winding, see Figure 4.14(b). Therefore, $d_{coat\_int}$ is defined as ($F_{fact} \cdot d_{coat}$), where $F_{fact}$ is the filling factor of the double winding, see Table 4.1. By using this modeling approach is easier to generate a high-quality mesh and to avoid typical FEA model problems when trying to mesh a geometry with relatively very small distances.

#### 4.2.6.6 Meshing

The last step before simulation is the geometry meshing. Figure 4.16 shows the geometry meshing for both the CHT and the SHT models. Although the geometry of the CHT model is exactly the same as the SHT model, the meshing result is different in both models. Both heat transfer models are meshed using a **physics-controlled mesh** approach, which means that the internal meshing algorithm of the simulation tool puts more effort in the meshing process at the boundaries where different physics are interacting. As is explained in section 4.2.6.2, the CHT model, the Equations 4.7, 4.8, 4.9a, and 4.9b are taken into account to solve the system, i.e. there is a bidirectional multiphysics coupling between the heat convection-diffusion equation and the Navier-Stokes for free convection equation. On the other hand, the SHT model only considers the Equations 4.7 and 4.8, which means that it does not consider the air flow caused by the Busbar heat transfer to the air.



(a) CHT model meshing.

(b) SHT model meshing.

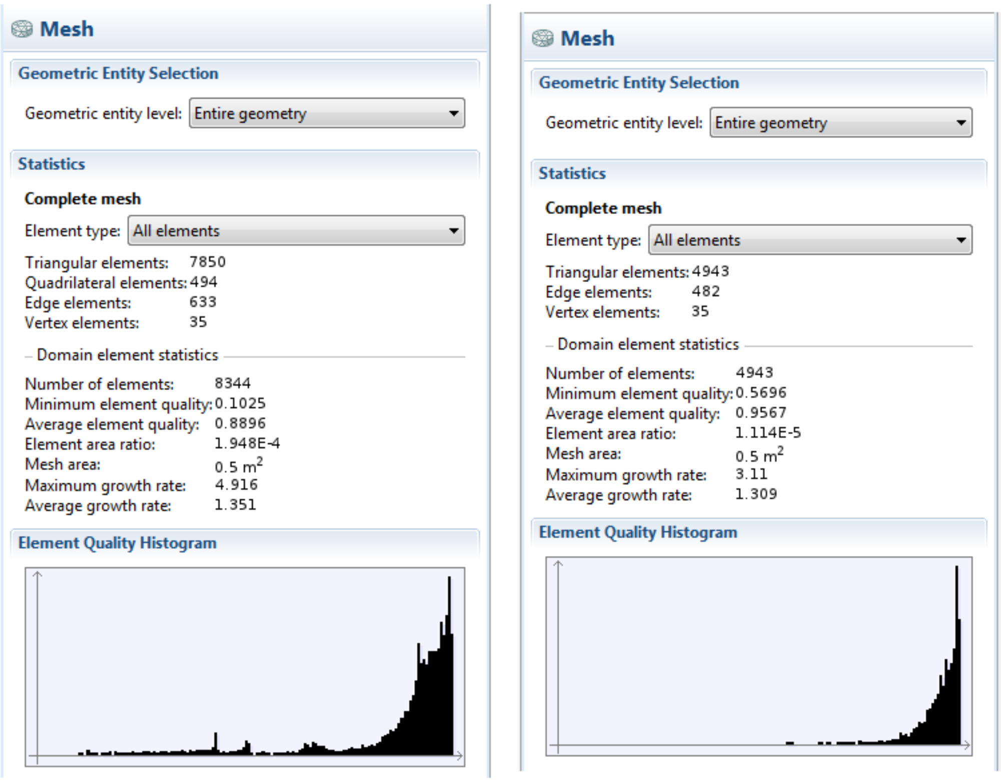
Figure 4.16: 2D Axisymmetric Geometry Meshing. The element quality is represented from 0 (blue) to 1 (red) in the color scale. (a) conjugated heat transfer (CHT) model. (b) simplified heat transfer (SHT) model.

Consequently, the CHT model has quadrilateral elements at the air boundaries while the SHT model does not, see Figure 4.17. This condition makes the CHT model meshing more element-populated than the SHT model meshing.

Figure 4.17 shows statistical information about the mesh element quality for both models. The CHT model meshing uses a total of 8344 elements choosing a `Fine` mesh size[2], whilst the simpler heat transfer model meshing has a total of 4943 elements choosing a `Finer` mesh size. So, the difference in the number of elements does not lie in the element size but in the meshing sequence type.

---

[2] The mesh size specifies how the predefined mesh algorithm generates meshes. Fine and Finer are just two of the predefined mesh sizes in COMSOL. If the physics interface is the controlling interface for the mesh, these settings are used when automatically generating the mesh each time it is solved.



**Mesh**

**Geometric Entity Selection**

Geometric entity level: Entire geometry

**Statistics**

**Complete mesh**

Element type: All elements

Triangular elements: 7850
Quadrilateral elements: 494
Edge elements: 633
Vertex elements: 35

Domain element statistics

Number of elements: 8344
Minimum element quality: 0.1025
Average element quality: 0.8896
Element area ratio: 1.948E-4
Mesh area: 0.5 $m^2$
Maximum growth rate: 4.916
Average growth rate: 1.351

**Element Quality Histogram**

(a)

**Mesh**

**Geometric Entity Selection**

Geometric entity level: Entire geometry

**Statistics**

**Complete mesh**

Element type: All elements

Triangular elements: 4943
Edge elements: 482
Vertex elements: 35

Domain element statistics

Number of elements: 4943
Minimum element quality: 0.5696
Average element quality: 0.9567
Element area ratio: 1.114E-5
Mesh area: 0.5 $m^2$
Maximum growth rate: 3.11
Average growth rate: 1.309

**Element Quality Histogram**

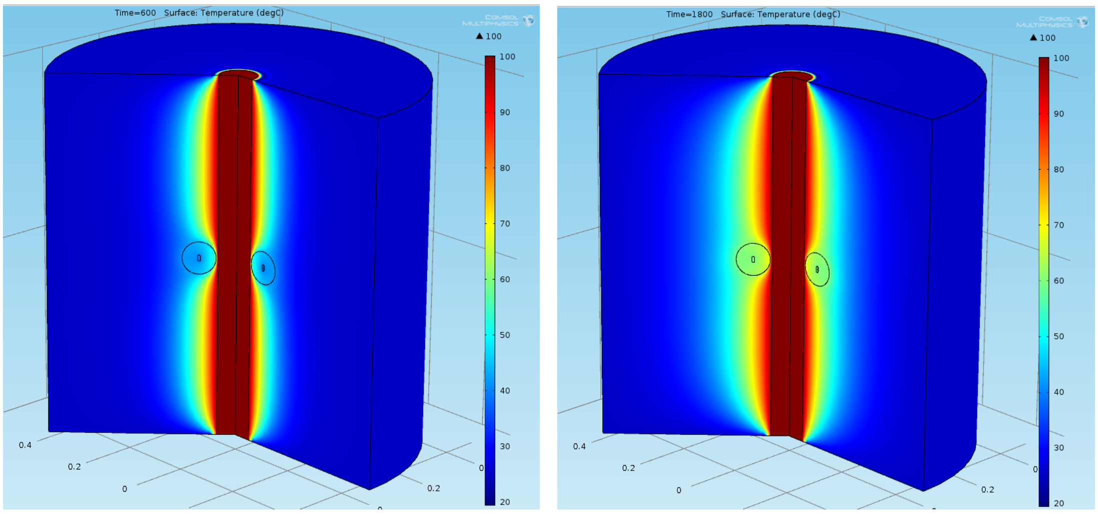
(b)

Figure 4.17: Mesh quality statistics of the KEVCR model. (a) CHT model. (b) SHT model.

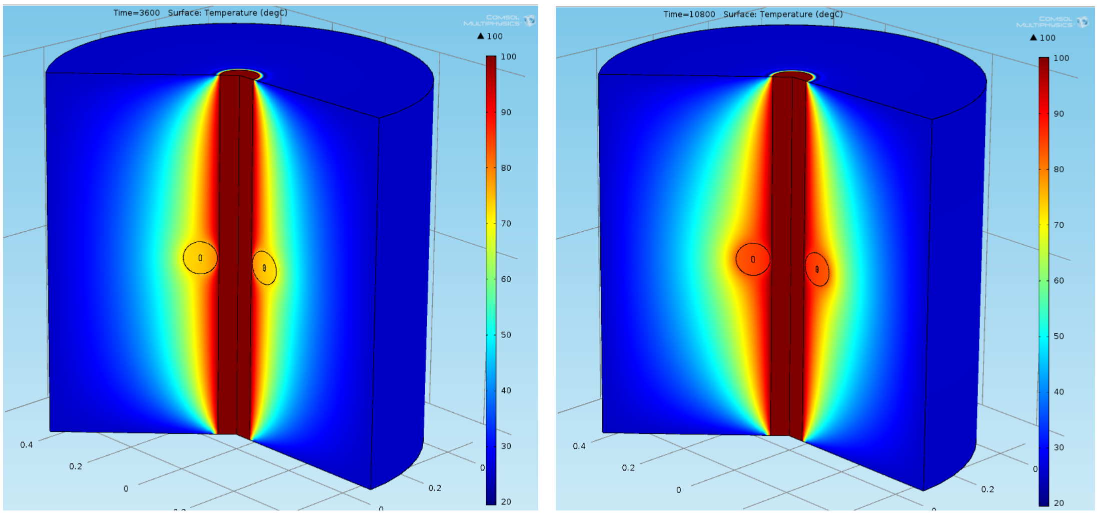
Figure 4.17 displays a histogram plot of the mesh element quality for all element types in the entire geometry. The x-axis represents the element quality, and the y-axis represents the number of elements of similar quality. The absolute value of the mesh element quality is always between 0 and 1, where 0 represents a degenerated element and 1 represents the best possible element. An element is considered of good quality when all its edges have the same length, this is why the quadrilateral elements in the CHT model meshing are of less quality than the triangular elements. In general, the higher the element quality the lower the effort for the solver in simulation and the better the accuracy of the model results. Both models have a high average element quality, which suggests a reliable model for simulation.

#### 4.2.6.7 FEA simulation results

For the sake of simplicity, the results presented in this section are only performed for the KEVCR coil. Same conclusions are obtained using the parameter values of the KECA coil. In order to simulate the thermal behavior of the RogoCoil sensor without taking into account electrical influences, the temperature of the Busbar is here assumed as a constant value. The initial analysis considered is the dynamic heat transfer behavior and the final steady-state temperature of the RogoCoil sensor using the most refined model, i.e. the CHT model.


Time=600 Surface: Temperature (degC)
▲ 100
100
90
80
70
60
50
40
30
20
0.4
0.2
0
0.2
Time=1800 Surface: Temperature (degC)
▲ 100
100
90
80
70
60
50
40
30
20
0.4
0.2
0
0.2

(a) 10 minutes. (b) 30 minutes.

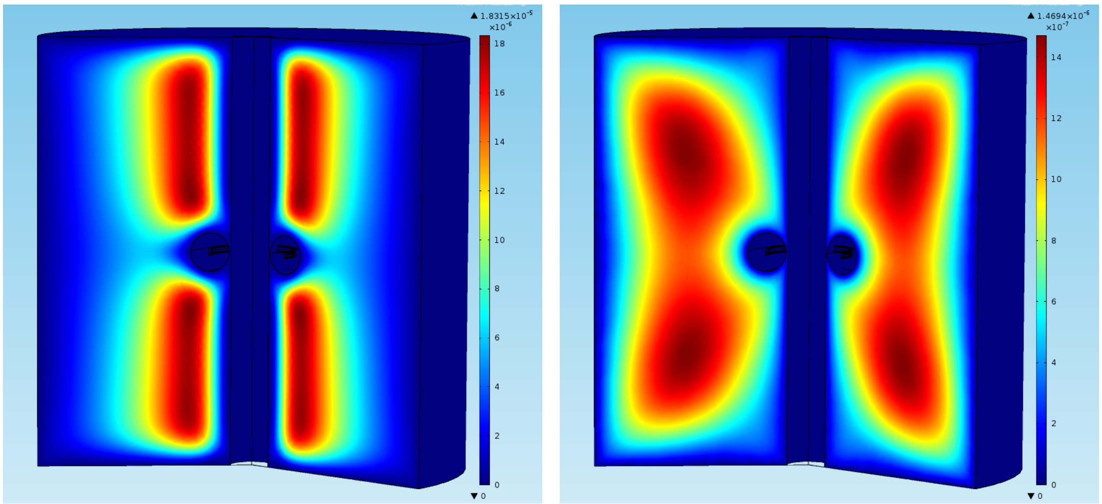

Time=3600 Surface: Temperature (degC)
▲ 100
100
90
80
70
60
50
40
30
20
0.4
0.2
0
0.2
Time=10800 Surface: Temperature (degC)
▲ 100
100
90
80
70
60
50
40
30
20
0.4
0.2
0
0.2

(c) 1 hour. (d) 3 hours.

Figure 4.18: Time-domain simulation: 3D temperature plots of the KEVCR CHT model. Conditions: $\texttt{Tbus} = 100^{o}C$, $\texttt{Troom} = 25^{o}C$ and $\texttt{Agap} = 4.2mm$.

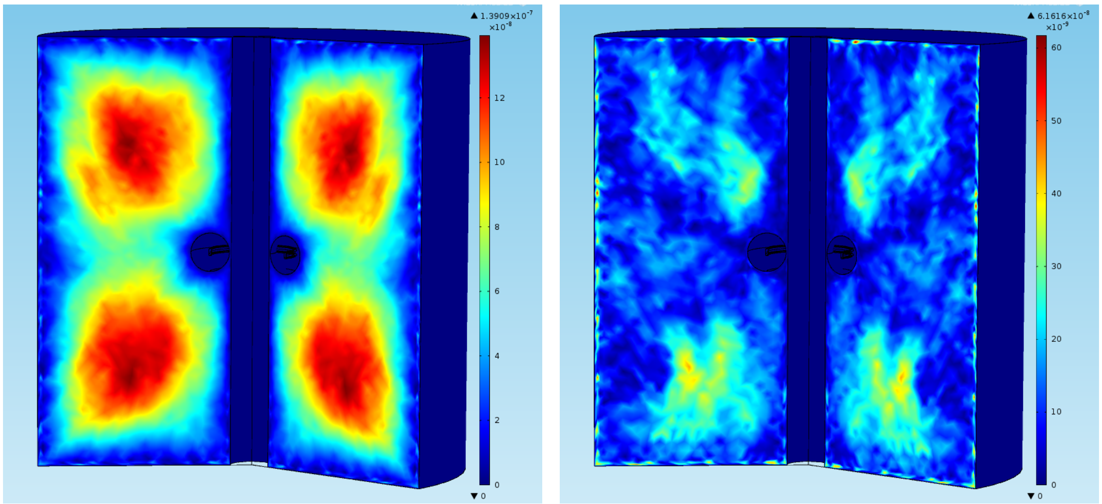
The CHT model considers both energy and mass transfer (i.e heat and air flow) in the RogoCoil system. Figure 4.18 shows a series of 3D plots taken at different system times. It can be observed the evolution of the temperature gradient around the Busbar and the RogoCoil. At 10 minutes, see Figure 4.18(a), is shown how the RogoCoil obstructs a uniform temperature increasing in the surrounding air caused by the heat transfer from the hotter Busbar to the colder environment. In Figure 4.18(d) the temperature gradient is practically in steady-state. Although the temperature gradient takes a little longer than 3 hours to be stable, the air flow around the Busbar is always fluctuating due to the energy transfer by natural convection, similar to a boiling water system.

Figure 4.19 also shows 4 instances of the air velocity magnitude around the RogoCoil sensor system at different system times. The velocity of the air around the Busbar is higher near to the hottest air regions and is decreasing with the time. After a couple of hours is observed that the high-velocity spots start to decrease approaching to a more uniform air velocity distribution. However, the air flow is constantly fluctuating around the Busbar and never arriving at a steady-state. In Figure 4.19(d), it can be observed small turbulent regions at the air box boundaries. Although the air flow is considered as laminar flow in this simulation, the system can be turbulent in nature. However, a turbulent air flow is out of the scope of interest due to the real system is supposed to be closed inside a substation.



(a) 2 minutes.

(b) 30 minutes.

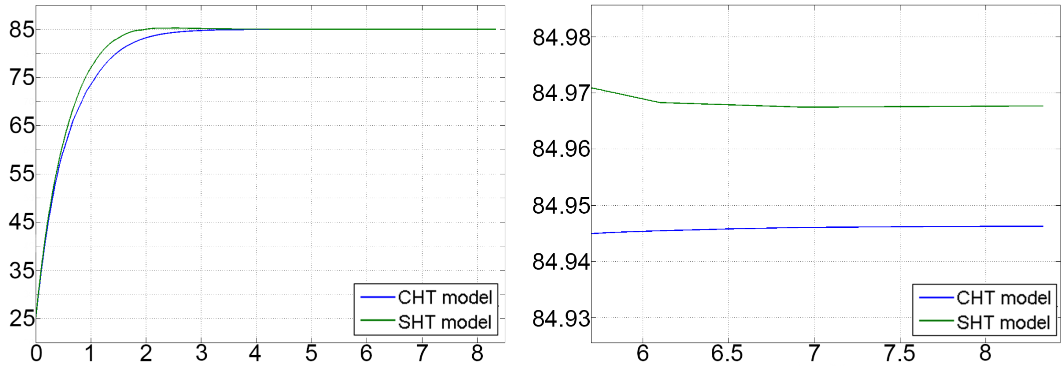


(c) 2 hours.

(d) 3 hours.

Figure 4.19: Time-domain simulation: 3D air velocity plots of the CHT model. Conditions: `Tbus` = $100^oC$, `Troom` = $25^oC$ and `Agap` = $4.2mm$.

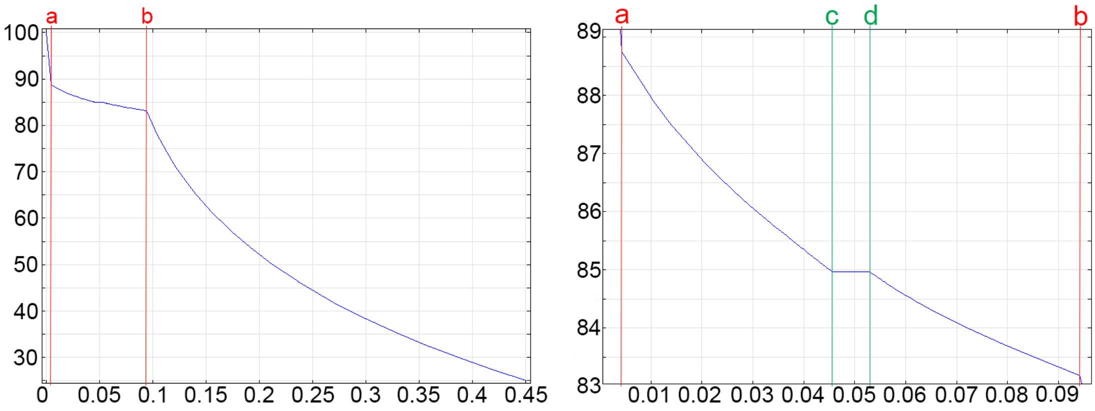
As the temperature gradient reaches the stability, we are motivated to simplify the model by ignoring the air velocity field, i.e. by using the SHT model. In Figure 4.20 we can observe a transient simulation of both the CHT and SHT models. At steady-state, the difference between the temperature predictions of the two models is in the order of tens of mili-Celsius as it can be observed in Figure 4.20(b). Since there is no energy lost by the air movement in the SHT model, $T_{coil}$ is a little higher, the temperature response is a bit faster and slightly under-damped than the CHT model. However, the temperature settling time in both models is about the same.



(a)

(b)

Figure 4.20: $T_{coil}$ (°C) vs. Time (hours), CHT and SHT transient simulation. $T_{bus}$ = 100 °C, $T_{room} = 25$ °C, $A_{gap} = 4.2mm$.



(a)

(b)

Figure 4.21: Temperature (°C) vs. Distance (m). SHT static simulation throughout the cutline shown in Figure 4.14(a) on page 69. (a) Entire region. (b) Zoom in the RogoCoil region.

In order to explain how and where $T_{coil}$ is defined, let us consider Figure 4.21. This is a steady-state simulation of the SHT model in which is plotted the temperature throughout a cutline from the Busbar surface (distance = 0 m) to the extreme of the air box that contains the system. The cutline is taken at the RogoCoil geometrical center as is shown in Figure 4.14(a) on page 69. The distances **a**, **b**, **c** and **d** in Figure 4.21 correspond to the highlighted distances in Figure

4.14(a). It can be observed that the temperature decrement is significantly faster in the air region than inside the RogoCoil, i.e. the region between **a** and **b**. In a closer look in Figure 4.21(b), we can see that the temperature remains practically constant throughout the RogoCoil core, i.e. the region between **c** and **d**. Actually, the temperature inside the RogoCoil core also decreases with the distance to the Busbar surface. However, the temperature difference between the inner and the outer extreme of the RogoCoil core is less than few tens of mili-°C. Therefore, $T_{coil}$ can be safely defined as the temperature at any point inside **c** and **d**.

### 4.2.7 Equation-based model extraction for the Rogowski coil thermal coupling

From the transient behavior of both CHT and SHT models, the time-dependent equation of $T_{coil}$ can be estimated as follows:

$$T_{coil}(t) = T_0 + (T_{SS} - T_0)\left(1 - e^{-\frac{t+\Delta t}{\tau}}\right) \tag{4.12}$$

where $T_0$ is the effective initial temperature of the RogoCoil, which in our case is equal to `Troom` plus an additional temperature ($T_{INI}$), which can be used to chose the initial temperature of the RogoCoil. This is useful to simulate the system starting from a specific steady-state condition. $T_{SS}$ is the final steady-state temperature of the RogoCoil, and $\tau$ is the time constant of the thermal behavior of the RogoCoil sensor system. If the user wants to simulate the system from a specific steady-state condition, the initial temperatures for `Tbus`, `Tcoil`, and `Troom` must be defined accordingly, in such case $\Delta t$ must be greater than $5\tau$, otherwise $\Delta t = 0$. A boolean parameter (`SSC`) has been created for selecting the appropriate time constant.

The SHT model can be used for static parametric simulations to obtain $T_{SS}$ at different temperature and geometrical conditions. Afterwards, a polynomial curve fitting algorithm is performed to obtain the parameters of Equation 4.12, see Appendix C.4 on page 203.

| Name | Min | Max | Step |
|---|---|---|---|
| Tbus | -5 $^oC$ | 200 $^oC$ | 5 $^oC$ |
| Troom | -10 $^oC$ | 90 $^oC$ | 5 $^oC$ |
| Agap | 0 $mm$ | 5.2 $mm$ | 0.2 & 1 $mm$ |

Table 4.2: Sweep parameter range.

Considering `Tbus`, `Troom`, and `Agap` as the parameters of interest, both $T_{SS}$ and $\tau$ in Equation 4.12 are variables which depend on these 3 parameters. However, $\tau$ can only be obtained by time-dependent simulations, which makes its estimation by parametric simulation highly time-consuming and impractical. This is why a worst case scenario has been studied to estimate the maximum and minimum $\tau$ of the system within the parameter range.

For a constant `Agap` value, it can be observed that the RogoCoil system arrives faster to the

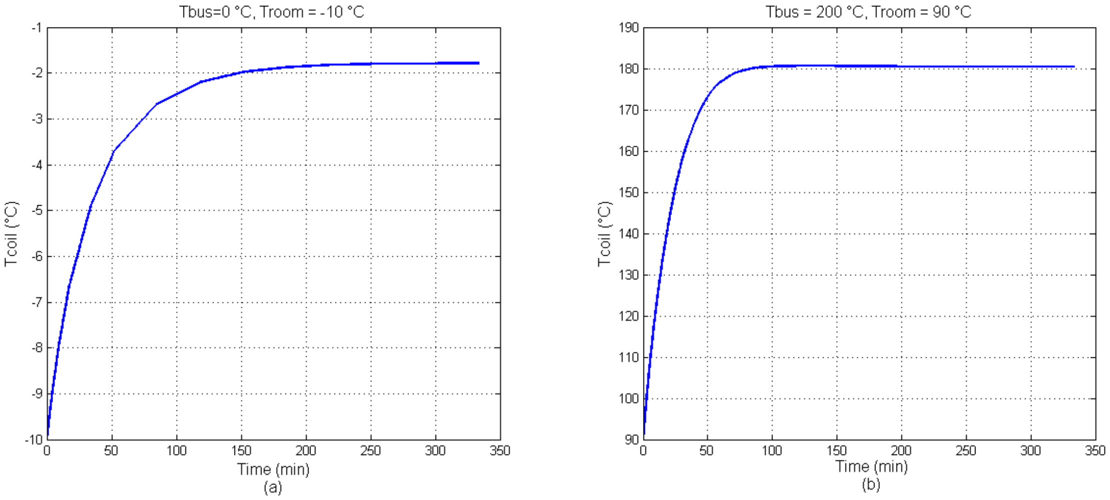


Figure 4.22: SHT model simulation: RogoCoil temperature (`Tcoil`) $[^oC]$ vs. Time [min]. Conditions: `Agap` = 2.2 $mm$ . $\tau$ value: (a) $\tau_{max} = 36.778\ min$. (b) $\tau_{min} = 26.896\ min$.

steady-state condition when the maximum values of `Tbus` and `Troom` in the temperature range are used. Conversely, the RogoCoil system presents its slowest response when the lowest `Tbus` and `Troom` values are used[3], see Figure 4.22. Consequently, we have performed a curve fitting of the equation 4.12 using the extreme hot (`Tbus` = $200^oC$, `Troom` = $90^oC$) and cold (`Tbus` = $0^oC$, `Troom` = $-10^oC$) conditions in the `Agap` parameter range. Similarly, an equation for $\tau$ can also be acquired by polynomial curve fitting. However, $\tau$ can only be obtained by transient simulations. Consequently, its estimation by parametric simulation is highly time-consuming and impractical. A good approximation for the time behavior of the thermal coupling model can be made by considering an average ($x = avg$) or a worst case scenario, i.e. to simulate the maximum ($x = max$) and minimum ($x = min$) $\tau_x$ of the system within the sweep parameter range shown in Table 4.2. After a series of simulations, it has been observed that $\tau$ is highly dependent on $A_{gap}$ and slightly dependent on $T_{bus}$ and $T_{room}$. Thus, the temperature of the RogoCoil is better described as follows:

$$T_{coil}(t, T_{bus}, T_{room}, A_{gap}) = T_0(T_{room}) + (T_{SS}(T_{bus}, T_{room}, A_{gap}) - T_0(T_{room}))\left(1 - e^{-\frac{t}{\tau_x(A_{gap})}}\right) \tag{4.13}$$

---

[3] This behavior can be understood considering that the temperature is the kinetic energy of the molecules. This is consistent for a constant thermal conductivity of the material in the temperature range of `Tbus` and `Troom`.

where:

$$T_0(T_{room}) = T_{room} + T_{INI} \tag{4.14a}$$

$$\tau_x(A_{gap}) = S_1 \cdot A_{gap}^3 + S_2 \cdot A_{gap}^2 + S_3 \cdot A_{gap} + S_4 \tag{4.14b}$$

$$T_{SS}(T_{bus}, T_{room}, A_{gap}) = P_0 + P_1 \cdot T_{bus} + P_2 \cdot T_{room} + P_3 \cdot A_{gap} \tag{4.14c}$$

where $P_i$ and $S_j$ are the curve fitting coefficients which depend on the RogoCoil coil used, see the values obtained for the KEVCR coil in Table 4.3.

| | $\mathbf{S_1}$ | $\mathbf{S_2}$ | $\mathbf{S_3}$ | $\mathbf{S_4}$ |
|---|---|---|---|---|
| $\tau_{\mathbf{min}}$ | 0.1005 | -1.083 | 5.306 | 14.48 |
| $\tau_{\mathbf{max}}$ | 0.2655 | -2.864 | 12.72 | 20.61 |
| $\tau_{\mathbf{avg}}$ | 0.183 | -1.973 | 9.011 | 17.55 |
| | $\mathbf{P_0}$ | $\mathbf{P_1}$ | $\mathbf{P_2}$ | $\mathbf{P_3}$ |
| $\mathbf{T_{SS}}$ | 3.49 | 0.84 | 0.15 | -1.51e3 |

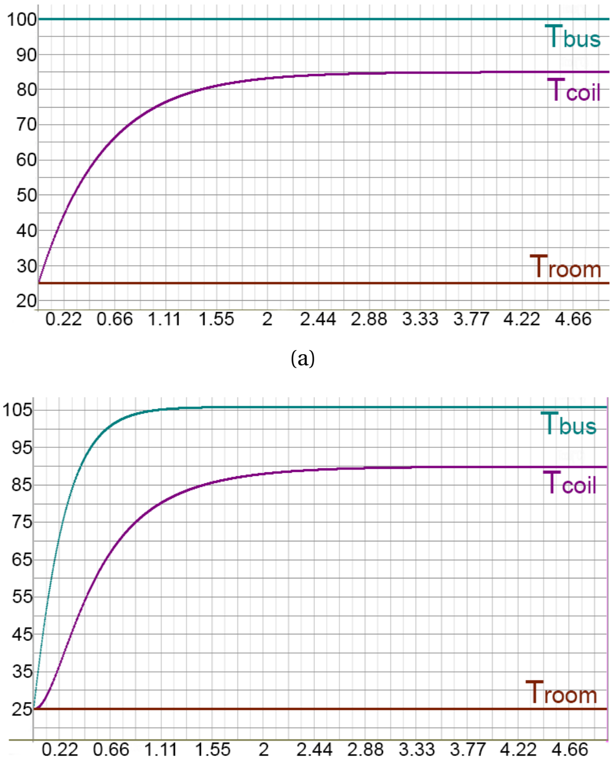
Table 4.3: Curve fitting coefficients for the KEVCR coils.

### 4.2.8 VHDL-AMS electro-thermal Rogowski coil simulation and analysis

By parametric static and transient simulations of the geometrical SHT model, we can get an equation-based model of the thermal coupling by using the equations previously described in subsection 4.2.7. Afterwards, the ETRCM shown in Figure 4.4 on page 58 can be completed in VHDL-AMS. It is worth mentioning that this model only fits a particular selection of RC dimensions and materials.

The first step consists in simulating the ETRCM model using the same conditions as the SHT model, see Figure 4.23(a). We can observe a similar response in terms of settling time and final temperature as in Figure 4.20. This validation can also be done for different parameters in the range given in Table 4.2. In a second step, it is simulated the ETRCM using the IEEE Standard model of the Busbar [84], see Fig. 4.23(b). In this simulation, we assume the same initial temperature for the complete system (25 °C). We can observe how the Busbar increases its temperature thanks to an AC input current; likewise, the RC sensor follows this temperature increment with a lower speed than the Busbar. This can be intuitively understood as the heat transfer goes from the Busbar to the RC sensor.

Finally, we are interested in simulating critical $T_{room}$ variations in the RC sensor system. Fig. 4.24 shows how the key thermal and electrical variables of the system react to a strong $T_{room}$ fluctuation, an exponential heating up and cooling down between 25 and 80 °C. The initial condition for this simulation is the operational steady-state temperature of the RC sensor system shown in Fig. 4.23(b). This is a very fast but feasible temperature profile. The heating



(b)

Figure 4.23: Temperature (°C) vs. Time (hours), ETRCM transient simulations. $A_{gap}$ = 4.2 mm, $T_{room}$ = 25 °C, $\tau_{avg}$ = 34.14 min. (a) At Constant $T_{bus}$. (b) Using the Standard Busbar model: Input current of 2 $kA_{peak}$ @ 50 Hz.

up process takes more than 30 minutes to achieve the maximum temperature. The cooling down process takes less than 50 minutes to come back to the initial temperature since the falling time constant ($\gamma_f$) is half of the rising time constant ($\gamma_r$). We can see how $T_{bus}$ and $T_{coil}$ are affected by the $T_{room}$ fluctuation, the thermal inertia of the Busbar and the RC are clearly evident from the different marginal changes in their temperature profile with respect to $T_{room}$. We observe slower temperature changes in both the Busbar and the RC, and a small delay for the maximum $T_{bus}$ compared to the maximum $T_{room}$, i.e. around 41.66 minutes.

Fig. 4.24 also shows the effect of the $T_{room}$ fluctuation in the RC resistance ($R_{coil}$) and the RC inductance ($L_{coil}$). The code shown in Figures 4.9 and 4.10 in subsection 4.2.3, explain how the temperature effect on $R_{coil}$ is modeled. Similarly, the temperature dependence of $L_{coil}$ is modeled from the same idea but using a different temperature coefficient. Consequently, both electrical variables have the same waveform as $T_{coil}$.

In order to quantify the effect of the $T_{room}$ variation on the electrical variables, we can define the relative variable change as follows:

$$\Delta M = \frac{(M_{max} - M_{min})}{M_{min}} * 100 \tag{4.15}$$

where $M$ is the specific electrical variable. In Fig. 4.24, the relative $R_{coil}$ change ($\Delta R_{coil}$) is about 14%, whereas the relative $L_{coil}$ change ($\Delta L_{coil}$) is only about 0.0992%. This significant difference is mainly due to we have used in simulation a temperature coefficient of inductance (`TCL`) two orders of magnitude lower than the temperature coefficient of resistance of copper (`TCR` = 0.0039 $K^{-1}$).

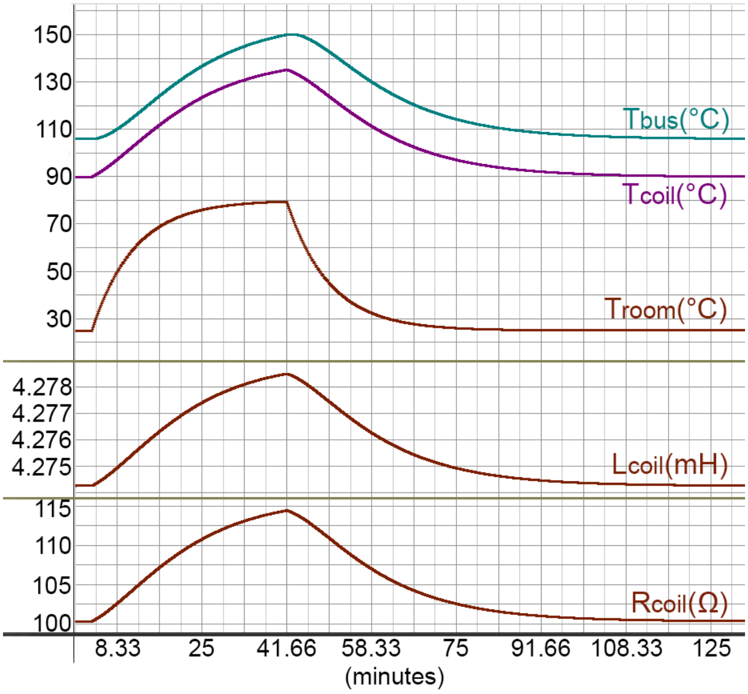


Figure 4.24: ETRCM transient simulation. Vertical axes: Temperature (upper graph), Inductance (middle graph), Resistance (bottom graph). Exponential time constants for $T_{room}$: $\gamma_r$ = 500 sec (rising), $\gamma_f$ = 1000 sec (falling). Initial temperature condition: $T_{bus}$ = 105.9 °C, $T_{coil}$ = 88.9 °C and $T_{room}$ = 25 °C.

This type of simulation can be used to test the system under different room temperature conditions. Although the ETRCM can give results for divergent $T_{room}$ profiles, is important to consider realistic fluctuations in terms of variation speed and waveform. This is especially important to validate this model by using experimental measurements. Moreover, non-commercial RogoCoil applications, in which very low temperatures might appear in the system (such as in [88]), could benefit from this co-design methodology between VHDL-AMS and FEA. However, is important to know the validity range of the thermal and electrical models. Further modifications to the physics and other assumptions in the model can be done for studies under extreme temperature conditions.

## 4.3 The self-calibration unit of the Rogowski coil sensor

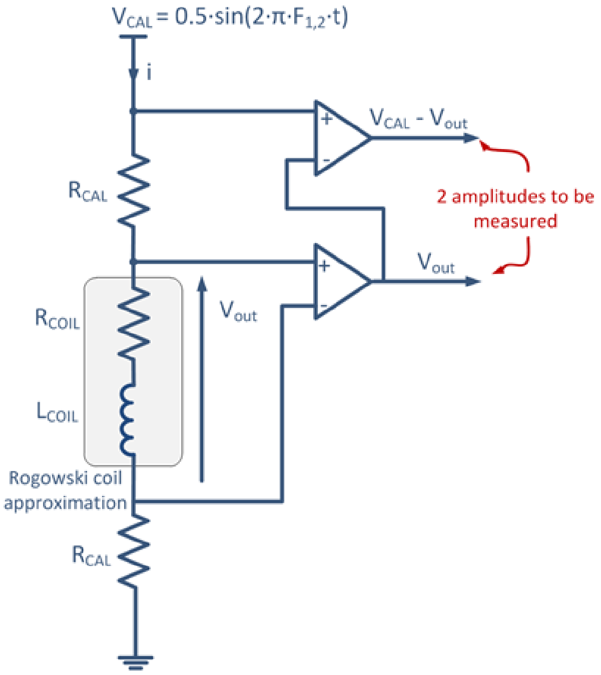
As it is explained in section 4.1, the main task of the self-calibration unit (SCU) is to calculate autonomously and continuously the RogoCoil correction factor ($CF$) that is required for the primary current calculation performed by the CMU, see section 4.4. During this research, multiple VPs were implemented to study the different self-calibration techniques. In particular, this section describes one of the self-calibration methods that has been proposed by ABB as a result of the ongoing research on the RogoCoil transducer.

Although the design approach applied for the SCU did not follow the recommended meet-in-the-middle approach from the earlier stage of the design, its VP and their *component models* have been designed using the VHDL-AMS techniques described in section 3.5. In fact, the *virtual prototyping* of the SCU has been key to decide the most convenient VHDL-AMS modeling approach and the recommended VP modeling structure that are generalized in the VP-Modeling Guidelines. The SCU VP presented in subsection 4.3.2 follows a hierarchical model structure using VHDL-AMS packages and configurations, which facilitates the gradual *model refinement* process. The presented simulations show a simple functional verification analysis of the SCU estimation accuracy for $L_{coil}$ and $R_{coil}$ values. Additionally, a comparison between experimental and simulation results for a particular *test case* is also included in this section.

### 4.3.1 Self-calibration method

For the sake of simplicity, the self-calibration method assumes that the RogoCoil acts as a resistor ($R_{coil}$) connected in series with an inductor ($L_{coil}$) as we can observe in Figure 4.8(b) on page 61. This approximation is valid when the calibration signal, which is injected to the RogoCoil secondary outputs, is sufficiently below the first resonance frequency of the RogoCoil. In fact, we can neglect the effect of parasitic capacitances (i.e. cable, inter-loop capacitances, etc.) at low frequencies.

As is conceptually shown in Figure 4.3 on page 56, the self-calibration method uses a closed-loop configuration in which a high-frequency calibration signal is generated in the IED and injected to the RogoCoil secondary ports. The self-calibration method consists in the direct estimation of the RogoCoil sensor impedance by measuring its winding resistance ($R_{coil}$) as well as its inductance ($L_{coil}$). A low voltage AC signal with a predetermined frequency and amplitude is applied to one of the RogoCoil terminals through a calibration resistor ($R_{CAL}$), that is connected in series, a second calibration resistor is used to connect the other RogoCoil terminal to ground, as is depicted in Figure 4.25(a). Only 2 voltage measurements ($V_{CAL} - V_{out}$ and $V_{out}$) at 2 different frequencies are required for the impedance estimation; the phase information is conserved. In fact, since $V_{CAL}$ is an input whose value can be known a priori, only the measurement of $V_{out}$ should be performed. However, this would require the implementation of an AC voltage source highly accurate in both its root mean square (RMS) voltage and frequency, which require the use of more expensive electronics. Therefore, in order to relax the calibration source implementation, it was chosen to include $V_{CAL}$ voltage in the measurement and ensure an accurate frequency calibration source using commercial electronic components.


V<sub>CAL</sub> = 0.5·sin(2·π·F<sub>1,2</sub>·t)
i
V<sub>CAL</sub> - V<sub>out</sub>
R<sub>CAL</sub>
2 amplitudes to be measured
V<sub>out</sub>
R<sub>COIL</sub>
V<sub>out</sub>
L<sub>COIL</sub>
Rogowski coil approximation
R<sub>CAL</sub>
+
-
+
-

(a)

$$i_{1,2} = \frac{V_{CAL_{1,2}} - V_{OUT_{1,2}}}{2R_{CAL}} \qquad \text{(I)}$$

$$z_{1,2} = \frac{V_{OUT_{1,2}}}{i_{1,2}} \qquad \text{(II)}$$

$$\sqrt{L_{coil}^2 \cdot \omega_1^2 + R_{coil}^2} = |z_1| \qquad \text{(III)}$$

$$\sqrt{L_{coil}^2 \cdot \omega_2^2 + R_{coil}^2} = |z_2| \qquad \text{(IV)}$$

$$L_{coil} = \sqrt{\frac{z_1^2 - z_2^2}{\omega_1^2 - \omega_2^2}} \qquad \text{(V)}$$

(b)

Figure 4.25: (a) Analog signal processing scheme to estimate the resistance and the inductance of RogoCoil sensors. (b) Equations for calculating the RogoCoil inductance $L_{coil}$ by measuring two different voltages at frequencies $F_1$ and $F_2$.

In the schematic shown in Figure 4.25(a), we can observe that the current $i$ flowing through the two calibration resistors $R_{CAL}$ and through the RogoCoil sensor can be measured using the two voltages $V_{CAL}$ and $V_{out}$ as Equation I establishes, see Figure 4.25(b). It is important to note that thanks to a relatively high $R_{CAL}$ ohmic value (10 kΩ) and low sensor output voltage levels (150 mV at rated current for both KEVCR and KECA sensors), the power dissipation in $R_{CAL}$ is negligible and will not introduce any significant resistor drift. Even in the case of peak values in the primary current, the sensor output can be controlled by using protection diodes. The maximal power dissipation in the calibration resistors must be less than 1 W. Although this verification can be performed using *virtual prototyping*, these details are not taking into account since they are not critical for the system operation.

It turns out that the RogoCoil sensitivity depends directly on the $L_{coil}$ value, therefore the goal of the SCU is to calculate as accurate as possible $L_{coil}$ using Equation V in Figure 4.25(b), where $z_i$ is the RogoCoil impedance at frequency $F_i$ and $\omega_i = 2\pi F_i$. The frequency of the calibration signal must change from a high value ($F_1$) to a lower value ($F_2$). In order to calculate $R_{coil}$, which is used to calculate the measurement gain in the CMU, see section 4.4.1 on page 105, we can perform the measurement at DC (i.e. $F_2 = 0$). In this case $z_2 = R_{coil}$.

In previous IED designs that do not contain a SCU, the equivalent input resistance of the IED was chosen in the MΩ range, i.e. much higher than $R_{coil}$ [80]. This decision was taken in order to make the measurement insensitive to changes in $R_{coil}$ due to temperature drifts or manufacturing tolerances. In this case, since $R_{coil}$ is measured, it is possible to compensate the voltage loss incurred by the voltage divider circuit in Figure 4.25(a). Likewise, $R_{CAL}$ cannot be chosen in the MΩ range because it would bring the calibration signal to a low amplitude level that makes its accurate measurement more difficult using standard commercial electronic components. A low RogoCoil output voltage ($V_{out}$) would require very long averaging time when it is sampled with a typical 16 effective number of bits (ENOB) ΣΔ ADC.

The calibration frequency is chosen depending on the RogoCoil sensor properties. We will obtain better results when this calibration frequency is low. i.e. where the impedance of the sensor is mainly resistive. On the contrary, this calibration frequency shall not be too high where the impedance of the sensor is strongly capacitive, see Figure 4.6 on page 60. The calibration resistance ($R_{CAL}$ = 10 kΩ) and the calibration high frequency ($F_1$ = 3 kHz) were chosen by Pascal et al [80] considering different assumptions and facts tested in a physical prototype, called the '**concept demonstrator**'.

On one hand, at 3 kHz the influence of parasitic capacitances (cable, inter-loop, etc.) are negligible. On the other hand, this frequency allows to obtain an RL impedance value high enough to measure the output voltage of the RC sensor $V_{out}$ with the selected $R_{CAL}$ value, see Figure 4.25(a). An exception to this choice is made for the sensor types which exhibit a $R_{coil}$ smaller than 400 Ω[4], in these cases, the selected calibration frequency is 20 kHz in order to obtain a higher $V_{out}$ signal that is attenuated by the small $R_{coil}$ value. However, the simulations performed using a more refined (1Loop) RogoCoil model shown subsection 4.3.2.2 on page 101, demonstrate the closer the calibration signal to the first resonance frequency, the lower the estimation accuracy of the $L_{coil}$.

Regarding the selection of $R_{CAL}$, a value of 10 kΩ is unique for the entire RogoCoil sensor portfolio that must be supported by this self-calibration method, where the $R_{coil}$ values range from 90 Ω to 5kΩ. The author of this thesis has found in simulation a fundamental limitation of the self-calibration method related to the RogoCoil sensors that present a high winding resistance value ($R_{coil} \approx 5k\Omega$) and low inductance value ($L_{coil} \approx$ 0.2 mH), i.e. the KEVCD B family sensors.

Observing the RogoCoil sensor as a simple RL circuit in series, the RogoCoil output voltage $V_{out}$ is composed by the voltage drop in $R_{coil}$ and the voltage drop in $L_{coil}$. Smaller impedance values will induce greater estimation errors of both $R_{coil}$ and $L_{coil}$ since $R_{CAL}$ and the amplification gain of the signal conditioning are fixed values. Now, let us consider the case when only one of the RogoCoil impedance values is small, let us suppose a small $R_{coil}$ value, i.e. $R_{coil} \ll j\omega L_{coil}$. In this case, $V_{out}$ is mainly caused by $L_{coil}$ contribution, therefore, $L_{coil}$ estimation will be more accurate than $R_{coil}$ due to noise and ADC resolution limitations. However,

---

[4] This is the case for both KEVCR and KECA sensors of the ABB portfolio.

this does not represent a big issue. In fact, by using $R_{coil} = 0$, $L_{coil} = 100$ mH in simulation, the estimation of both values is accurate. On the other hand, the situation is totally different when $L_{coil}$ is small, i.e. $R_{coil} \gg j\omega L_{coil}$. In this case, the contribution to the $V_{out}$ signal is mainly caused by $R_{coil}$. Therefore, $z_1$ (taken at frequency $F_1$) will be closer to $z_2$ (taken in DC). Simulating with $R_{coil} = 10$k and $L_{coil} = 1$ mH, the calculation fails since the estimated value of $z_1$ is almost the same than $z_2$. Consequently, simulating with a higher value of $R_{coil}$ produce $z_1 = z_2$; and therefore, $L_{coil} = 0$, i.e. an error of 100% simulating using the Ideal RL approximation of the RogoCoil.

The algorithm presented in Table 4.4 shows the main tasks that the SCU must execute when running continuously. It is important mentioning that the $R_{coil}$ and $L_{coil}$ calculation is directly used to recognize the type of RogoCoil sensor that is connected to the IED. Since the difference among the impedance values of different type of sensors is large enough, the RogoCoil identification task carried out by the SCU is not critical. Therefore, it has been excluded from the self-calibration algorithm used for the VP implementation, see Table 4.4.

```
While calibration is active loop
    * Measure $V_{CAL} - V_{out}$ at frequency $F_2$
    * Measure $V_{out}$ at frequency $F_2$
    * Calculate $R_{coil} = z_2$ (only if $F_2 = 0$)
    * Measure $V_{CAL} - V_{out}$ at frequency $F_1$
    * Measure $V_{out}$ at frequency $F_1$
    * Calculate the impedance $z_1$
    * Calculate $L_{coil}$
    * Calculate the correction factor (CF)
    * Send the CF to the measurement circuit (MEAS)
End loop
```

Table 4.4: Self-calibration unit algorithm.

### 4.3.2 The SCU virtual prototype

The VP architecture shown in Figure 4.26 depicts the implementation of the SCU algorithm shown in Table 4.4. Since the $CF$ estimation is only relevant for the current estimation carried out by the CMU, and it directly depends on $L_{coil}$ and $R_{coil}$ values, the $CF$ calculation is not taken into account in this VP example. Instead, this VP focuses on the modeling and functional verification of the $L_{coil}$ and $R_{coil}$ estimation accuracy subject to *component model* refinement under specific simulation conditions.

Figure 4.26 is an example of a mapping process between the *technology space*, represented by a set of standard hardware/software blocks including analog and digital components, and

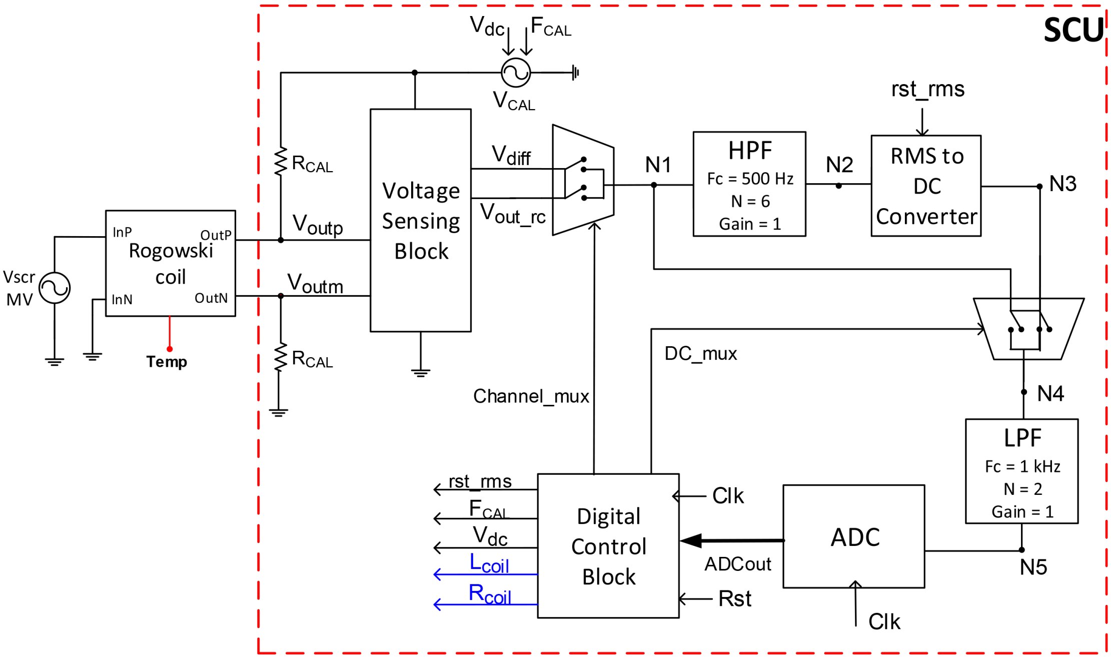


Figure 4.26: SCU architecture interconnected with the RogoCoil electrical model. This subsystem-level VP implements the algorithm presented in Table 4.4. The *CF* calculation is not taken into account in this VP example.

the *functional space*, represented by the particular SCU algorithm. We can observe that the secondary of the RogoCoil (`OutP` and `OutN` ports) are connected to the calibration resistors as is explained in Figure 4.25(a) on page 84. The `Voltage Sensing Block (VSB)` represents the circuitry responsible for measuring the voltages $V_{out\_rc} = V_{outp} - V_{outm}$ and $V_{diff} = V_{cal} - V_{out}$ as the SCU algorithm requires. Figure 4.25(a) shows a particular implementation of the VSB; however, this block can be implemented in multiple forms executing the same functionality. This is why the VSB is treated in the VP as a generic *component model* that can be later refined using different *model implementations*.

$V_{diff}$ and $V_{out\_rc}$ signals are connected to a 2-to-1 multiplexer controlled by the digital signal `Channel_mux`. The SCU signal conditioning chain is formed by two circuit paths that are selected according to the calibration frequency that is being applied. When the high frequency $F_1$ is applied, the VSB output voltage signals are passing through the path formed by the nodes `N1`, *N2*, `N3` and `N4`. It means that the signal is treated by a high-pass filter (HPF) that removes the DC component of the signals and an RMS-to-DC converter to get the desired RMS voltage values to use them in the equations shown in Figure 4.25(b). On the other hand, when it is applied a DC calibration voltage ($F_2 = 0$), the measured voltage is bypassed from node `N1` to `N4` by the 2-to-1 multiplexer controlled by the digital signal `DC_mux`. Finally, the analog signal is converted to digital by the ADC as it is shown in the Figure. The low-pass filter (LPF) between node `N4` and `N5` is an anti-aliasing filter that eliminates undesired high-frequency components caused by the multiplexing and other possible noise sources.

`Digital Control Block` is responsible for sending the control signals to the multiplexers, and the DC and frequency values of to the calibration voltage source. The `Digital Control Block` can be easily implemented in a microcontroller or in an FPGA. The *model implementation* used in the SCU VP is a non-synthesizable FSM that executes the SCU algorithm by measuring each of the required voltages in a different state and controlling the state transitions according to the expected delays of the components of the analog signal processing chain. It is important mentioning that both $V_{diff}$ and $V_{out\_rc}$ signals will have a 50 Hz component when a current is applied to the primary. Consequently, it is required to filter this component before the analog-to-digital conversion. We have considered the three following options to implement such filter:

1. By decreasing the cutoff frequency and/or increasing the filter order of the LPF located before the ADC.
2. By implementing a digital LPF after analog-to-digital conversion.
3. By using a ΣΔ ADC architecture topology that uses a proper decimation filter.

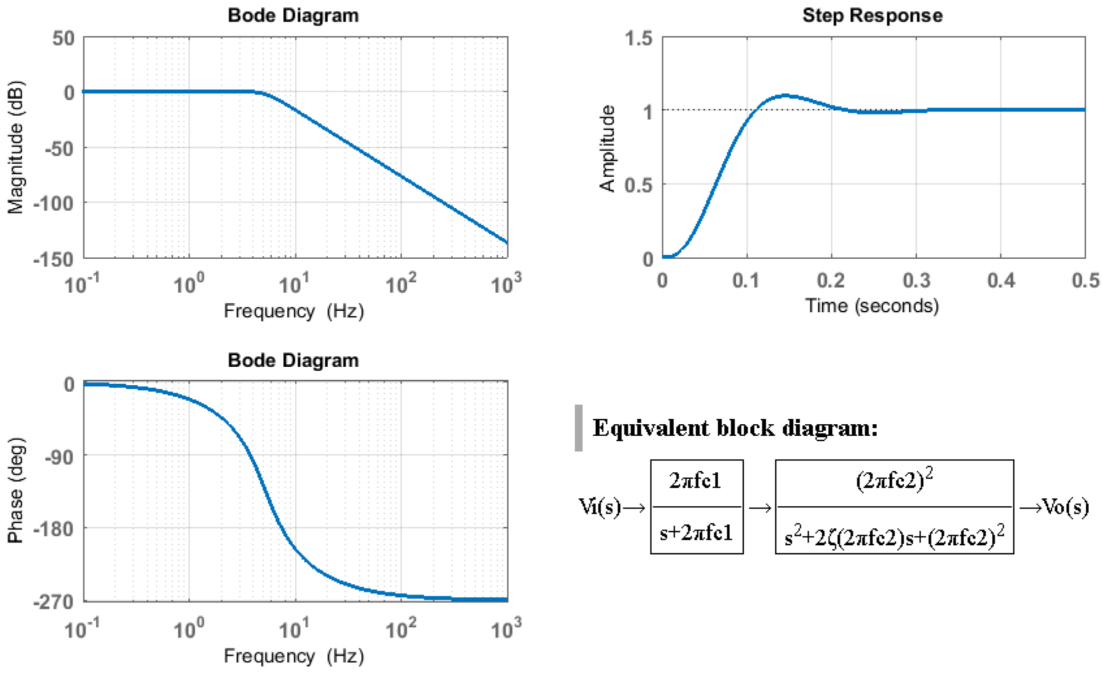


Figure 4.27: Bode diagrams, step response, and equivalent block diagram of a $3^{rd}$ order Sallen-Key active LPF topology, where: fc1=5.47 Hz, fc2=5.22 Hz, $\zeta$=0.485.

The first option implies the elaboration of a more expensive and high order filter. The current anti-aliasing analog LPF can be easily implemented by an RC filter. A cheap analog filter capable to attenuate the 50 Hz component can also be implemented, for example using a $3^{rd}$ order Sallen-Key active LPF we can get an attenuation of about 60 dB at 50 Hz, see Figure 4.27. However, this filter has a very slow time response, we can observe a settling time of about 500 ms in it step response plot. The filter time response issue can be solved by increasing the filter order. This can be done in a more practical way considering the second option, i.e. using a high order digital filter. For instance, implementing a high-order finite-impulse response (FIR) filter in software or in hardware. All these options were explored using the VP. Lastly, the third option, which is the most practical option for the physical implementation has been chosen for modeling, i.e. using the digital decimation filter of a ΣΔ ADC architecture.

Following, three main SCU VP *component models* and their *model implementations* are described in more detail.

- **Transfer function multipurpose filter:** This model allows to simulate any type of analog filter by using its transfer function representation. The model needs as input the numerator and denominator coefficients of the filter's transfer function. The coefficients can be stored in any external text file using the format shown in Table 4.5. The model can use multiple filter descriptions appended in the same text file.

```
Filter <#>. <Filter_Type> filter. Cutoff freq = <#>. Order = <N>.
H(s) = A(s)/B(s) = {a(1)s^N+a(2)s^(N-1)+...+a(N+1)} / {b(1)s^N+b(2)s^(N-1)+...+b(N+1)}:
<N>
<a(1) a(2) ... a(N) a(N+1)>
<b(1) b(2) ... b(N) b(N+1)>
```

Table 4.5: Text format for the input file of the transfer function multipurpose filter model.

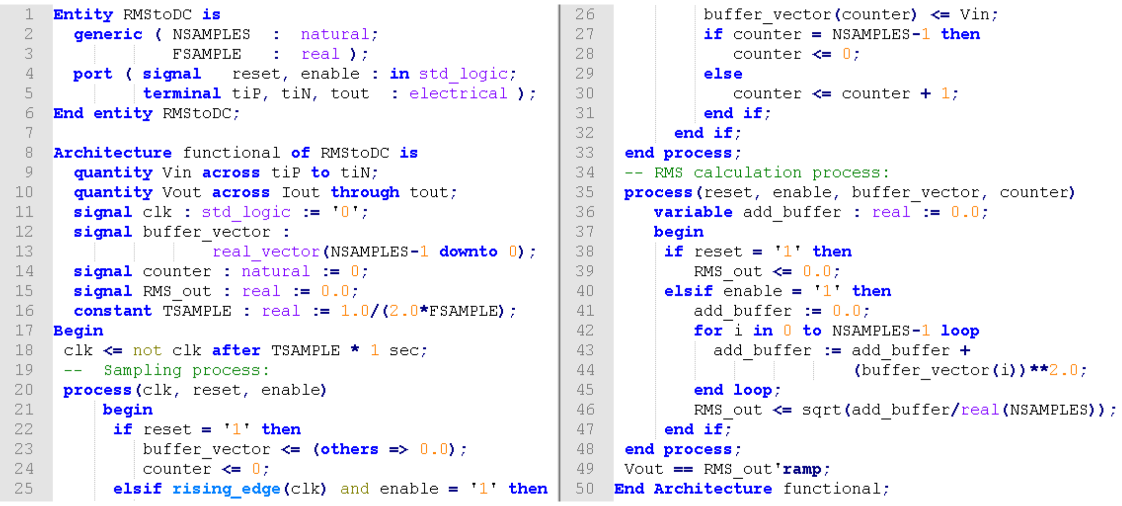
A filter description consists of 5 lines separated by an initial dashed line: The first text line indicates the filter number (`FNUM` parameter) (starting at 0); the type of the filter: low-pass, high-pass, band-pass or stop-band; the cutoff frequency and the order of the filter. The second text line shows how the coefficients are interpreted for elaborating the transfer function. The third text line is the filter order. The fourth text line contains the coefficients of the numerator, and the fifth text line contains the coefficients of the denominator of the transfer function.

| Name | Type | Description |
|---|---|---|
| FNUM | Natural | Filter number in the list |
| GAIN | Real | Filter gain |
| OUT_OFFSET | Voltage | Output offset voltage |
| FILE_ADD | String | Address of the filter list text file. |

Table 4.6: Parameters of the transfer function multipurpose filter model.

The high-level *model implementation* of the filter uses a local VHDL-AMS procedure and a function for reading the coefficients and the order of the filter indicated by the `FNUM` parameter. The text file can be stored in any folder. A full or relative text file path can be given in the `FILE_ADD` parameter. The filter is implemented using the VHDL-AMS `'LTF` attribute.

- **RMS-to-DC converter:** As its name suggest, this is a model of a component that calculates the RMS value of the analog input signal and returns this value as a DC output voltage signal. This model should operate in a wide range of input frequencies, sufficient to cover DC and high-frequency calibration signals (20 kHz or higher). Additionally, the conversion should work for different types of waveforms not limited to sine waves. Three *model implementations* were obtained as a result of gradual *model refinement*. The first *model implementation*, shown in Figure 4.28, samples the input signal at a clock frequency given by the `FSAMPLE` PMP and stores the samples in an internal buffer of size `NSAMPLES`. The RMS value is calculated by using the formula: $\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$, where $n$ is the internal buffer size and $x_i$ is the i$^{th}$ sample. This *model implementation* suffers from an inherent latency time window given by: $t_{ltw} = \frac{n}{f_s}$, where $f_s$ is the sampling frequency.

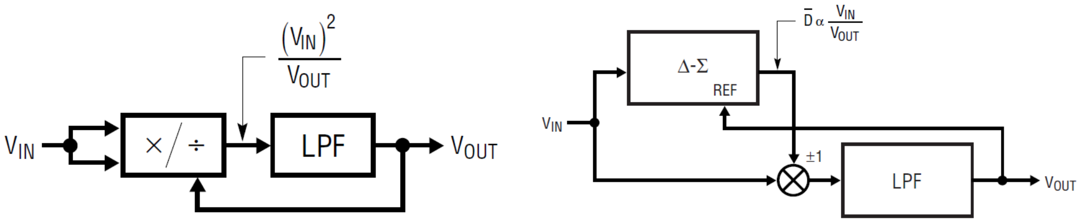


```vhdl
Entity RMStoDC is
  generic ( NSAMPLES   :  natural;
            FSAMPLE    :  real );
  port ( signal   reset, enable : in std_logic;
         terminal tiP, tiN, tout  : electrical );
End entity RMStoDC;

Architecture functional of RMStoDC is
  quantity Vin across tiP to tiN;
  quantity Vout across Iout through tout;
  signal clk : std_logic := '0';
  signal buffer_vector :
                real_vector(NSAMPLES-1 downto 0);
  signal counter : natural := 0;
  signal RMS_out : real := 0.0;
  constant TSAMPLE : real := 1.0/(2.0*FSAMPLE);
Begin
 clk <= not clk after TSAMPLE * 1 sec;
 --  Sampling process:
 process(clk, reset, enable)
     begin
      if reset = '1' then
         buffer_vector <= (others => 0.0);
         counter <= 0;
      elsif rising_edge(clk) and enable = '1' then
         buffer_vector(counter) <= Vin;
         if counter = NSAMPLES-1 then
            counter <= 0;
         else
            counter <= counter + 1;
         end if;
      end if;
 end process;
 -- RMS calculation process:
 process(reset, enable, buffer_vector, counter)
    variable add_buffer : real := 0.0;
    begin
     if reset = '1' then
        RMS_out <= 0.0;
     elsif enable = '1' then
        add_buffer := 0.0;
        for i in 0 to NSAMPLES-1 loop
          add_buffer := add_buffer +
                        (buffer_vector(i))**2.0;
        end loop;
        RMS_out <= sqrt(add_buffer/real(NSAMPLES));
     end if;
 end process;
 Vout == RMS_out'ramp;
End Architecture functional;
```

Figure 4.28: RMS-to-DC converter VHDL-AMS source code: functional model implementation.

Some analog RMS calculation techniques have been evaluated in order to refine the RMS-to-DC converter model. The chosen method was the implicit RMS computation, which uses a multiplier and a feedback loop to perform the square root function implicitly, see Figure 4.29(a). The signal averaging is done by an LPF at the output of the circuit. A high-level equation-based model which implement the architecture shown in Figure 4.29(a) was implemented as a second RMS-to-DC *model implementation.* Some advantages of the implicit RMS computation over other methods are fewer components, greater dynamic range, and generally lower cost [89].

A further model refinement can be observed in Figure 4.29(b), which is the topology used by the commercial low-cost RMS-to-DC converter used in the concept demonstrator prototype [90]. It basically consists of a 2$^{nd}$ order $\Sigma\Delta$ modulator that acts as the divider, and an LPF to



(a)

(b)

Figure 4.29: RMS-to-DC implicit computation architectures: (a) Conceptual architecture of an RMS-to-DC Converter with Implicit Computation. (b) Topology of the LTC1967 RMS-to-DC converter. The images are taken from [90].

perform the averaging of the RMS function. Additional non-ideal behaviors can be included in the RMS-to-DC converter model using the approach shown in Figure 4.30, where the non-linearity effect and voltage offsets are modeled by including input and output stages. These effects can be modeled at both high and low *abstraction levels*.

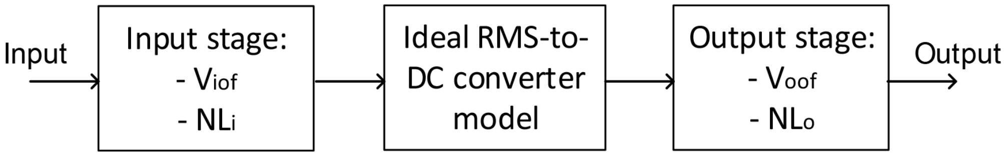
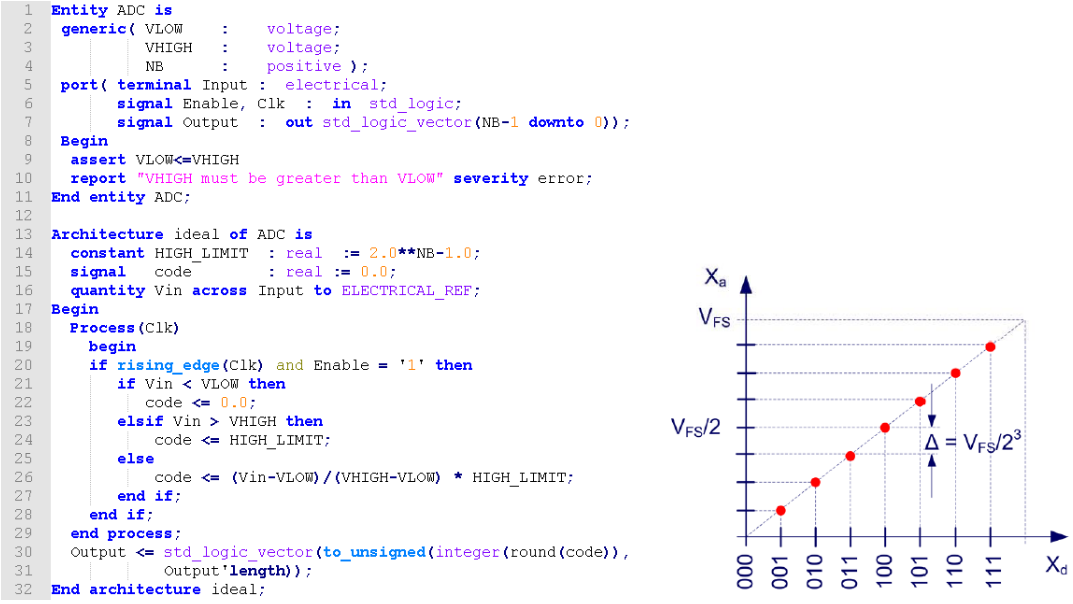


Figure 4.30: Modular approach for model refinement: modeling non-ideal behaviors in the RMS-to-DC converter; where $V_{iof}$ is the input offset voltage, $NL_i$ is the input non-linearity, $V_{oof}$ is the output offset voltage, and $NL_o$ is the output non-linearity.

- **Analog-to-Digital converter:** The first *model implementation* to consider is a functional ideal abstraction which encodes the analog input voltage using an unsigned binary representation given by the expression: $Bin_{code} = \frac{V_{in}-V_{LOW}}{V_{HIGH}-V_{LOW}} \cdot (2^{NB}-1)$; where $NB$ is the bit resolution, $V_{in}$ is the analog input voltage, $V_{LOW}$ and $V_{HIGH}$ are the lowest and highest input voltage respectively, see Figure 4.31(a).

As it was previously described, we are interested in refining the ADC model by implementing a ΣΔ ADC architecture. In fact, this type of ADC is one of the best solutions for analog-to-digital conversion of DC and low-frequency signals thanks to the elimination of most of the high-frequency noise by the anti-aliasing and decimation LPFs. Figure 4.32 shows the architectures of the first and second order ΣΔ ADC *model implementations*. The ΣΔ modulator, which is the principal component of this ADC, uses a signal oversampling technique to convert the mean of the analog input voltage into the mean of an analog pulse frequency. The result is a stream of digital pulses coming out from the quantizer (1bit ADC operating at a high-frequency clock). By dividing the pulse count during a known interval, we can obtain an accurate digital representation of the mean analog voltage during such interval. This functionality is implemented by the feedback loop shown in the $1^{st}$ order ΣΔ modulator in Figure 4.32(a). A



```
Entity ADC is
 generic( VLOW     :    voltage;
          VHIGH    :    voltage;
          NB       :    positive );
 port( terminal Input :  electrical;
       signal Enable, Clk  :  in  std_logic;
       signal Output  :  out std_logic_vector(NB-1 downto 0));
 Begin
  assert VLOW<=VHIGH
  report "VHIGH must be greater than VLOW" severity error;
End entity ADC;

Architecture ideal of ADC is
  constant HIGH_LIMIT  : real   := 2.0**NB-1.0;
  signal   code        : real := 0.0;
  quantity Vin across Input to ELECTRICAL_REF;
Begin
  Process(Clk)
    begin
    if rising_edge(Clk) and Enable = '1' then
       if Vin < VLOW then
          code <= 0.0;
       elsif Vin > VHIGH then
           code <= HIGH_LIMIT;
       else
           code <= (Vin-VLOW)/(VHIGH-VLOW) * HIGH_LIMIT;
       end if;
    end if;
  end process;
  Output <= std_logic_vector(to_unsigned(integer(round(code)),
            Output'length));
End architecture ideal;
```

(a)

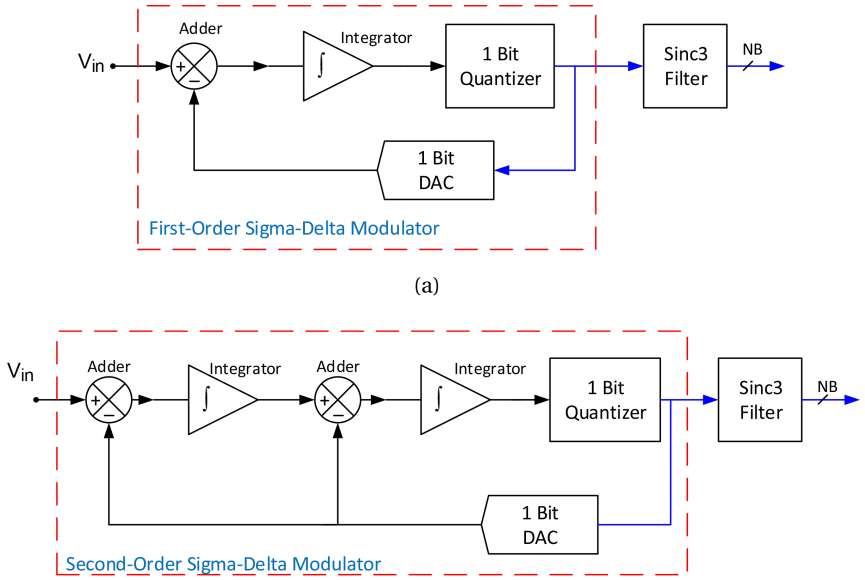
(b)

Figure 4.31: ADC functional modeling: (a) VHDL-AMS code of the ADC functional ideal *model implementation*. `NB` is the bit resolution, `VLOW` and `VHIGH` are the lowest and highest input voltage of the ADC respectively. (b) Full-scale coding representation graph using a 3bits resolution ADC, $V_{fs}$ =`VHIGH-VLOW`.

way to increase the ADC resolution [91], i.e. the effective number of bits (ENOB) of the ΣΔ modulator, is to use a higher-order loop filter by adding another integrator and feedback path as is shown in Figure 4.32(b). However, the stability a ΣΔ modulator becomes a more critical when a higher order is implemented. The second order ΣΔ modulator coefficients used in this work, which are PMPs of the `Adder` *component model* described in Appendix C.1.3.4, have been set using the criteria described in [92].

All the *model implementations* of the components of the ΣΔ modulator, given in Appendix C.1.3, are *functional model* abstractions that reproduce the desired behavior of the components for the correct operation of the modulator. These models can be gradually refined by component aggregation, or by including non-idealities caused by the internal sub-components such as resistors, capacitors, and OpAmps. However, this approach will make the ADC model too complex for simulation at the system level using the SCU VP. Instead, the signal distortions and non-linearity effects caused by the non-idealities of the modulator components can be modeled with input and output circuit stages, similar as it was done with the RMS-to-DC converter, see Figure 4.30. In this way, the ΣΔ modulator model will be less complex; and therefore, it will have a better simulation performance for system-level verification.

In order to retrieve the mean voltage data from the digital signal stream at the modulator's output, it is used a Sinc$^K$ filter of third order, which is in charge of reducing the signal's sampling rate (decimating), and averaging the signal (low-pass filtering) in order to remove all the high-frequency components (e.g. quantization noise, aliases). The total quantization



(a)

(b)

Figure 4.32: First and second order ΣΔ ADC architectures. See the complete source code in Appendix C.1.

energy is very high in the signal coming from the ΣΔ modulator, because the number of bits per sample is extremely low. It is designated to the decimator to filter undesirable noise in the spectrum over the Nyquist band so that the noise is not aliased into the baseband by the decimation process. At it is presented in [93], one of the most popular filter architectures for ΣΔ conversion entails the combination of a Sinc$^K$ filter and a FIR or infinite-impulse response (IIR) digital LPF. The ADC model only includes the *model implementation* of the Sinc$^K$ filter; an additional filtering stage, which can be implemented in software using a programmable digital signal processor (DSP) or a microcontroller, it is left to the block that receives the output signal of the ADC.

A Sinc filter is an idealized filter that removes all frequency components above a given cutoff frequency, without affecting lower frequencies, and has linear phase response. The filter's impulse response is a Sinc function in the time domain, and its frequency response is a rectangular function [94]. Being $K$ the filter order, the Sinc$^K$ filter transfer function is given as follows [93]:

$$H_z = \left(\frac{1}{M}\cdot\frac{1-z^{-M}}{1-z^{-1}}\right)^K \tag{4.16}$$

where $M$ is the decimation ratio of the down-sampling process, which is an integer or rational factor[5]. The hardware implementation of this type of filters is very easy since they do not

---

[5] $M > 1$ is used for decimation, whereas $M < 1$ is used for interpolation. Decimation factors are normally a

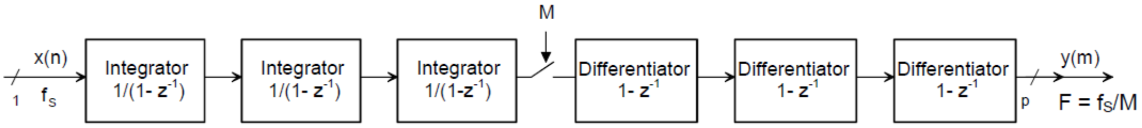


Figure 4.33: $\text{Sinc}^3$ digital filter topology. The integrators operate at the modulator sampling frequency $f_S$, whereas the differentiators operate at the low clock frequency $f_S/M$. The image is taken from [93].

require the use of digital multipliers. They can be more effectively implemented by cascading $K$ stages of integrators followed by $K$ stages of cascaded differentiators as it is shown in Figure 4.33. The order $K$ of the $\text{Sinc}^K$ filter shall be at least 1 plus the order of the $\Sigma\Delta$ modulator ($D$) in order to prevent excessive aliasing of the out-of-band noise from the modulator from entering the base-band (i.e. $K \geq 1 + D$). Since the maximum $\Sigma\Delta$ modulator order in the ADC model is 2, $K$ is fixed at 3 for both first and second order $\Sigma\Delta$ ADC as is shown in Figure 4.32. Similarly, the output bit vector size from the $\text{Sinc}^K$ filter is larger than the input by a factor $V$, which is a function of $M$ and $K$ as follows:

$$V = K \cdot log_2(M) \tag{4.17}$$

By including an additional filter order, $V$ is increased by $log_2(M)$. For example, if the input is 1 bit, the output from a Sinc filter ($M = 128$) will be increased by a 7-bit word. A second-order filter ($\text{Sinc}^2$) will add another 7 bits, i.e. its output will be 15-bit and so on. It means that the internal bus of the $\text{Sinc}^K$ filter, the integrators, and the differentiators, require a bus width that is one bit wider than the filter's DC gain given by $G_{DC} = M^K$ [93].

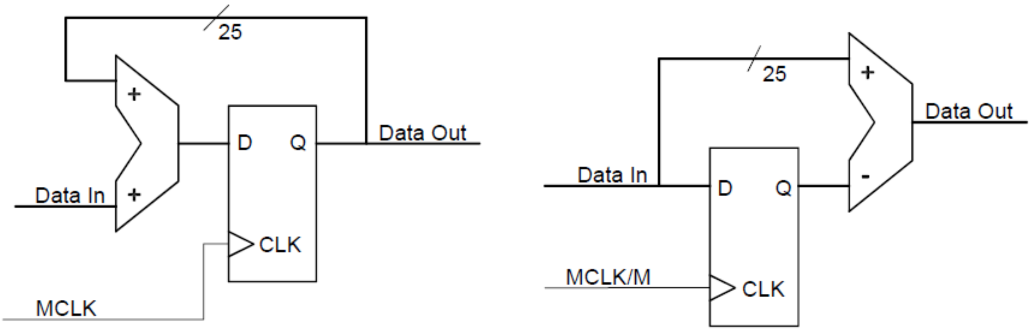
Figure 4.34 shows the RTL *model implementations* of the single integrator and differentiator blocks of the $\text{Sinc}^3$ filter. Any $\text{Sinc}^K$ filter can be built by cascading these basic blocks. The complete $\text{Sinc}^3$ filter model is given in Appendix C.1.4, which is a fully synthesizable VHDL model. Considering that the ADC should only operate for converting DC analog signals, the oversampling frequency value is not so important. However, the $\Sigma\Delta$ ADC precision is directly proportional to $M$, since the greater the $M$ the greater the ENOB. Ideally, the $\Sigma\Delta$ modulator signal-to-noise ratio (SNR) and the ENOB are obtained by the following Equations [93, 95]:

$$SNR_{ideal} = 6.02 \cdot N + 1.76 - 20 \cdot log\left(\frac{\pi^D}{\sqrt{2 \cdot D + 1}}\right) + (20 \cdot D + 10) \cdot log(M) \tag{4.18a}$$

$$ENOB_{ideal} = \frac{SNR_{ideal} - 1.76}{6.02} \tag{4.18b}$$

---

power of 2 numbers due to the easiness of digital implementation.



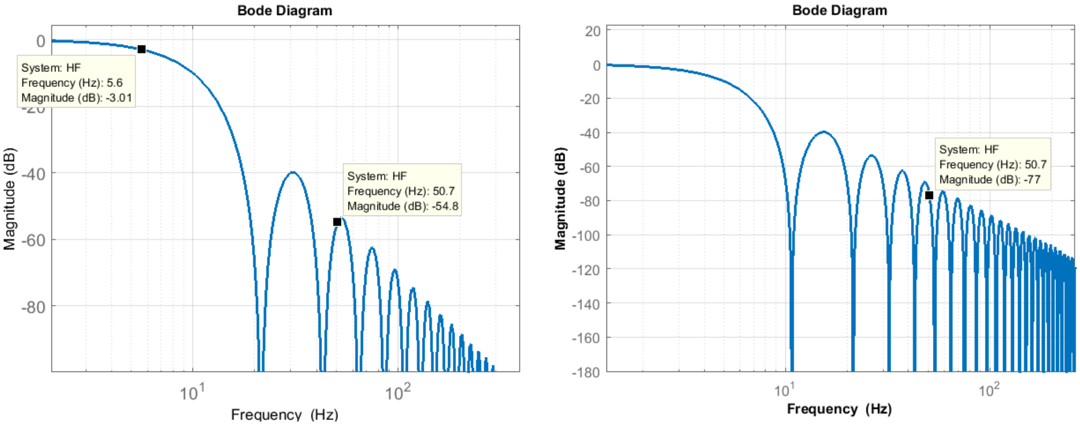
(a) Integrator RTL model.

(b) Differentiator RTL model.

Figure 4.34: RTL *model implementations* of the single integrator and differentiator blocks of the Sinc$^3$ digital filter. Where `MCLK` is the ΣΔ modulator clock and `M` is the decimation ratio. The images are taken from [93].

where $N$ is the quantizer bit resolution. Therefore, in order to use a high decimation ratio with a feasible bit vector size $V$ and a reasonable cutoff frequency for the decimation filter, the modulator frequency must be selected in the order of kHz. Table 4.7 shows the results for the first and second order modulators and the Sinc$^3$ filter.

Since one of the filter's requirements is to eliminate the 50 Hz signal component caused by the primary current, the output data rate of the ΣΔ ADC (given by $f_S/M$) can be set as low as possible, or preferably, it can be used to place a particular notch frequency in the digital filter frequency response using Equation 4.16, see Figures 4.35.



(a)

(b)

Figure 4.35: Frequency response of the Sinc$^3$ digital filter with $M = 1024$. (a) $f_S = 22$ kHz, $f_{cutoff} = 5.6$ Hz. (b) $f_S = 11$ kHz, $f_{cutoff} = 2.9$ Hz.

| Decimation Ratio (M) | 1$^{st}$ Order ΣΔ modulator | | 2$^{nd}$ Order ΣΔ modulator | | Sinc$^3$ filter | |
|---|---|---|---|---|---|---|
| | SNR (dB) | ENOB (bits) | SNR (dB) | ENOB (bits) | $G_{DC}$ (bits) | BusW (bits) |
| 4 | 20.67 | 3.1 | 24.99 | 3.9 | 6 | 7 |
| 16 | 38.73 | 6.1 | 55.09 | 8.9 | 12 | 13 |
| 64 | 56.79 | 9.1 | 85.19 | 13.9 | 18 | 19 |
| 256 | 74.85 | 12.1 | 115.30 | 18.9 | 24 | 25 |
| 1024 | 92.92 | 15.1 | 145.40 | 23.9 | 30 | 31 |

Table 4.7: ΣΔ characteristics for different decimation ratios. Where `SNR` and `ENOB` are the theoretical quantities given by Equations 4.18; $G_{DC}$ is the DC gain of the Sinc$^3$ filter given in bits; and `BusW` is the size of the output bus vector of the Sinc$^3$ filter given by $BusW = 1 + V$.

#### 4.3.2.1 Functional verification analysis

This subsection presents a simple analysis of the SCU functionality with different *model implementations* at particular *refinement levels*. The SCU VP modeling infrastructure can be observed in Figure 4.36. This is a hierarchical VHDL-AMS VP built by using a mix of the modeling techniques given in section 3.5. This VP uses 6 main parameter packages that are set from the *test bench*, and 4 TBCs for setting 4 *test cases*. The RogoCoil model can be

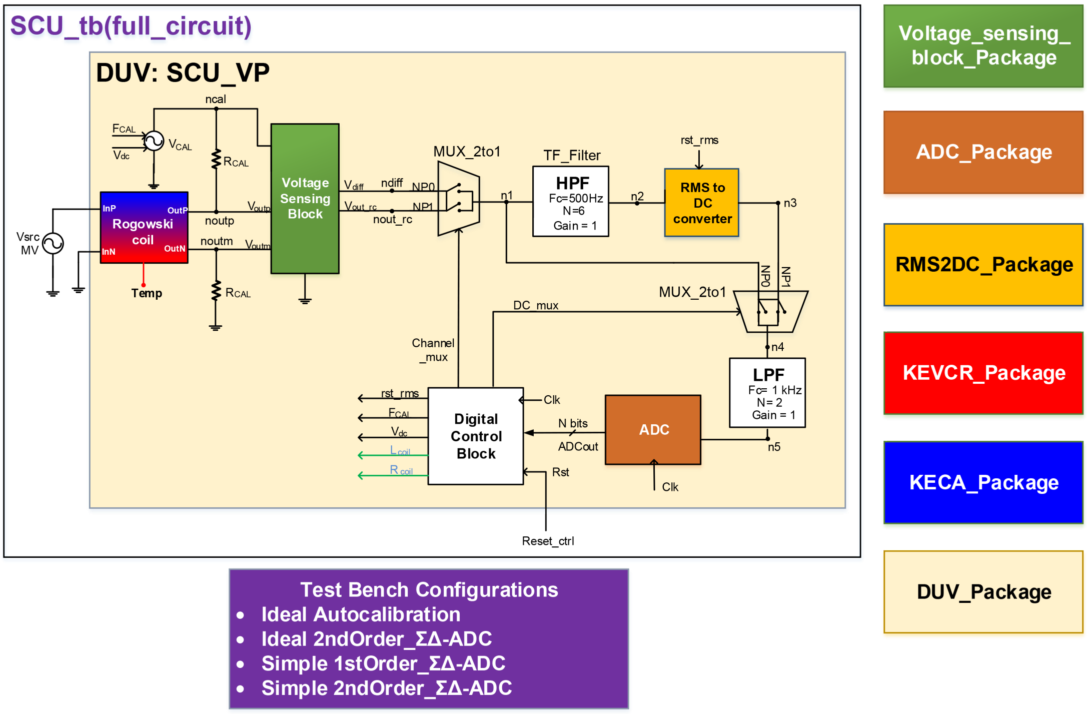


Figure 4.36: SCU VP modeling structure. It contains 2 different packages for the RogoCoil model and 1 package for each main *component model*. The PMPs of the other components of the DUV are given in the `DUV_package` and from the *test bench*. The VP uses 4 principal TBCs.

instantiated with one of the two available RogoCoil parameter packages, i.e. the `KEVCR_-`

`Package` or the `KECA_Package`. The `Voltage Sensing Block`, the `RMS-to-DC converter`, and the `ADC` models have their own parameter packages. The PMPs of the other components of the DUV are declared in the `DUV_package` or in the *test bench*.

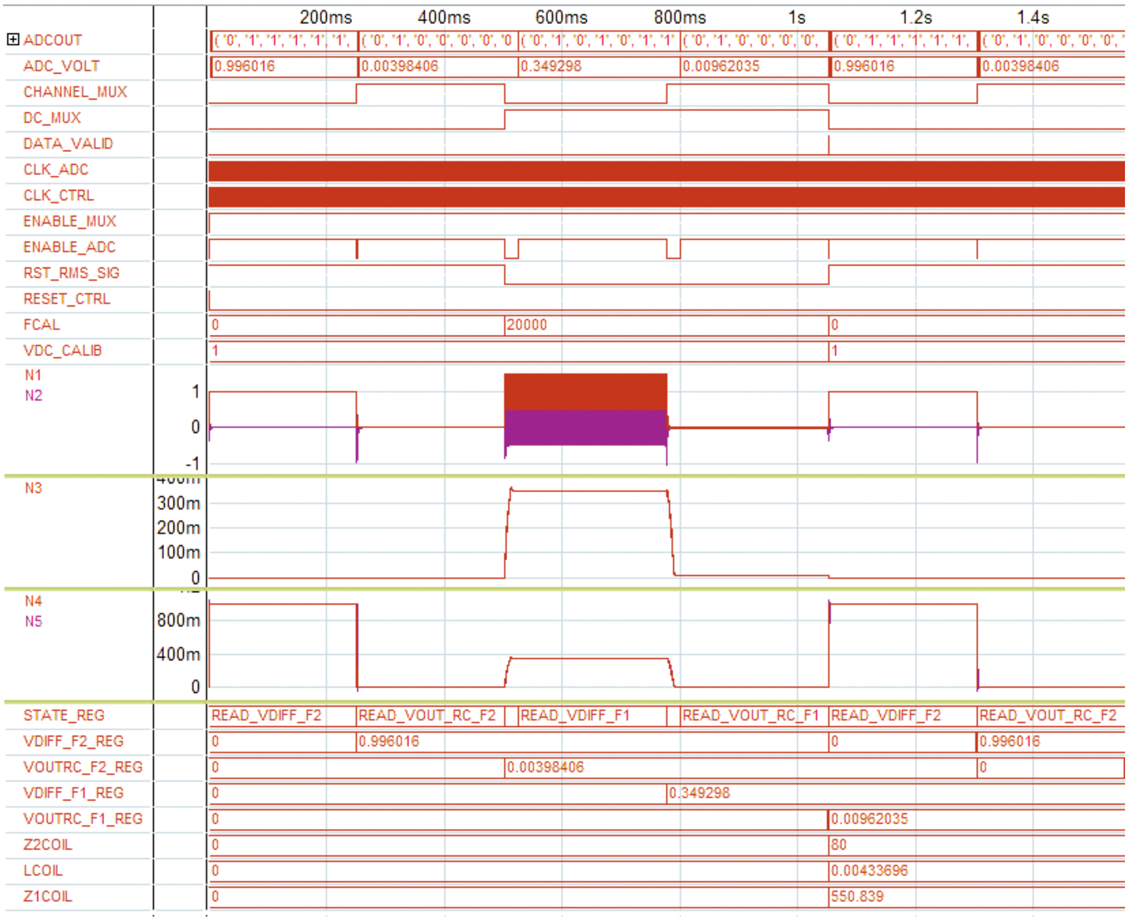


Figure 4.37: SCU VP (Ideal Autocalibration) transient simulation. *Verification conditions*: $R_{coil} = 80\Omega$; $L_{coil} = 4.3$ mH; sinusoidal calibration signal with: $V_{DC} = 1$ V, $V_{AMP} = 500$ mV, $F_1 = 20$ kHz. *Simulation conditions*: simulation tool = SMASH 6.4; integration method = Backward-Euler; time simulated = 2.5 s; maximum time step: $H_{max} = 1\ \mu$s. The digital signals `Z2COIL` and `LCOIL` are the estimated $R_{coil}$ and $L_{coil}$ respectively.

The Ideal Autocalibration TBC is used to perform the first simulation shown in Figure 4.37. This figure shows the most germane analog and digital signals of the SCU VP for one self-calibration loop. The $R_{coil}$ and $L_{coil}$ estimation are performed after $V_{diff}$ and $V_{out\_rc}$ voltages are measured at two different frequencies ($F_1$ and $F_2$) and stored in the registers `VDIFF_F2_REG`, `VOUTRC_F2_REG`, `VDIFF_F1_REG`, and `VOUTRC_F1_REG`, in the same order as is shown in the figure. The names of the signals correspond to the node name shown in Figure 4.26. In this verification case, no primary current is taken into account; therefore, the RogoCoil *model implementation* used is the `ideal_RL` model presented in Figure 4.8(b) on page 61. All the other *component models* are also the highest level abstractions of each component. Only the most relevant verification and simulation conditions are mentioned in the figure caption. It is important mentioning that all the simulations presented in this subsection are performed using a constant control clock frequency and temperature values, i.e. $F_{CTRL} = 4$ kHz and $T = 25$ °C respectively. Likewise, the Backward-Euler method is the used simulator integration method, since it permits a good simulation accuracy. This method is used in spite of the default trapezoidal method, which generates small oscillations in the RMS-to-DC conversion affecting significantly the accuracy of the result.

We can observe in Figure 4.37 that there is no error in the $R_{coil}$ estimation ($\Delta E\_R_{coil} = 0$), but the error in the $L_{coil}$ estimation is around 0.86% ($\Delta E\_L_{coil} = 0.86\%$). The time taken for performing the simulation shown in Figure 4.37 is about 1 min 17 s using a PC with a CPU Intel i7-3770 @ 3.4 GHz and 16 GB RAM[6]. Reducing the maximum simulation time step by a half (i.e. $H_{max} = 0.5\ \mu$s), makes the simulation time increase almost twice (2 min 6 s), but the error of the $L_{coil}$ estimation is also reduced by twice ($\Delta E\_L_{coil} = 0.45\%$) approximately. Not always the maximum time step will affect proportionally the simulation time and the accuracy of the result, it strongly depends on the particular *refinement levels* of the *component models*. Using a $H_{max} = 10$ ns, $\Delta E\_L_{coil}$ is reduced almost to zero, but the simulation time is increased to about 32 min for one self-calibration loop, which takes about 1.05 s for the `Ideal Autocalibration` configuration. Therefore, is key to find a good trade-off between the simulation time and the desired simulation accuracy of the VP using the highest *abstraction level* components. This initial ideal simulation case can be used to set a reference to help the comparison of different results obtained by performing a gradual *model refinement* process in future simulations.

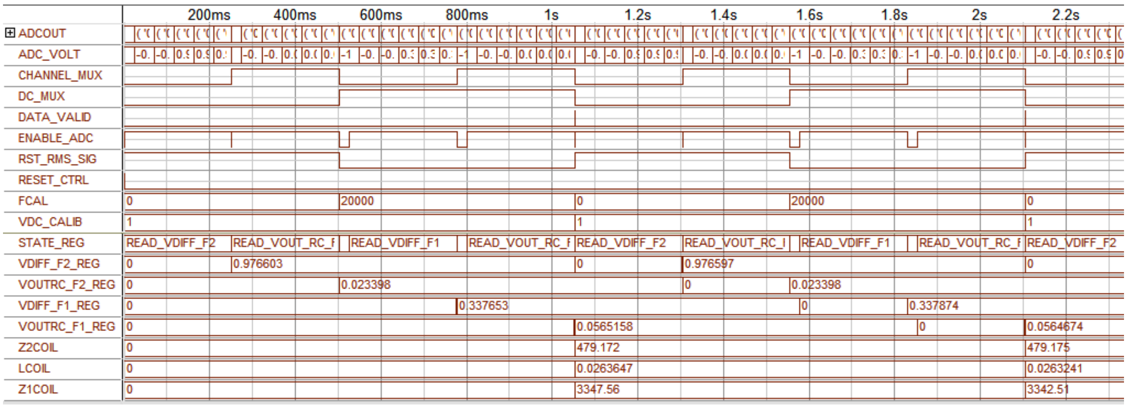


Figure 4.38: SCU VP (Ideal $2^{nd}$ Order ΣΔ ADC) transient simulation. *Verification conditions*: $R_{coil} = 479.15\Omega$; $L_{coil} = 26.15$ mH; ADC relevant parameters: $V_{REF} = 1$ V, $V_{DELTA} = 4$ V, $F_S = 22$ kHz, $M = 1024$; sinusoidal calibration signal with: $V_{DC} = 1$ V, $V_{AMP} = 500$ mV, $F_1 = 20$ kHz. *Simulation conditions*: simulation tool = SMASH 6.4; integration method = Backward-Euler; maximum time step: $H_{max} = 0.5\ \mu$s.

---

[6] The same computer has been used for all the simulations presented in this thesis.

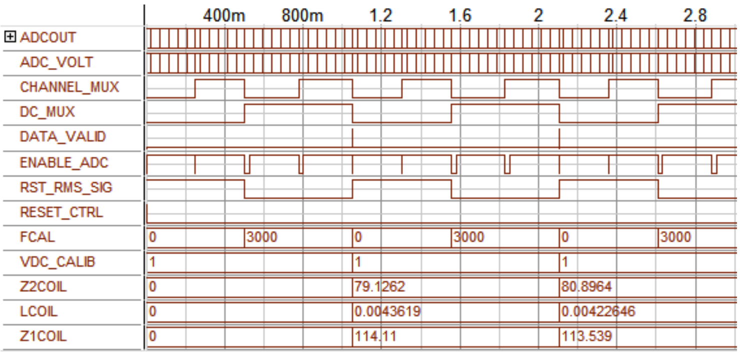
For the next simulation, see Figure 4.38, the ideal ADC model is replaced by a more refined *model implementation*, the $2^{nd}$ Order ΣΔ ADC model. The first two calibration results are shown in the figure. In this case, we obtain a small increase in the $L_{coil}$ estimation error ($\Delta E\_L_{coil}$ = 0.743%), but also a smaller increase in the $R_{coil}$ estimation error ($\Delta E\_R_{coil}$ = $4.9 \times 10^{-3}$%), which is caused by small numerical fluctuations of the ΣΔ modulation and the subsequent decimation down sampling. By simulating with a lower calibration frequency ($F_1$ = 3 kHz), there is a significant decrease in $\Delta E\_L_{coil}$ and almost no change in $\Delta E\_R_{coil}$, see Table 4.8. Since the ADC receives a DC value, the estimation error difference between the high and low calibration frequencies is caused by the RMS-to-DC converter *functional model* shown in Figure 4.28. Since the sampling frequency is a constant value (`FSAMPLE` = 100 kHz), the RMS to DC conversion is more accurate for a calibration signal at 3 kHz than at 20 kHz. This difference is reduced by using the RMS-to-DC implicit computation *model implementation* shown in Figure 4.29(a). Similarly, this SCU VP configuration was simulated with the $1^{st}$ Order ΣΔ ADC *model implementation*; no significant differences were obtained. Since the $2^{nd}$ Order ΣΔ ADC model is slightly more complex than the $1^{st}$ Order ΣΔ ADC model, the latter model can be used to obtain a better simulation performance when applying a gradual *model refinement* in other *component models* of the VP.

| **$F_1$ (kHz)** | **Ideal Autocalibration** | | **Ideal $2^{nd}$ Order ΣΔ** | | **Simple $1^{st}$ Order ΣΔ ADC** | |
|---|---|---|---|---|---|---|
| | **$\Delta E\_L_{coil}$** | **$\Delta E\_R_{coil}$** | **$\Delta E\_L_{coil}$(avg)** | **$\Delta E\_R_{coil}$(avg)** | **$\Delta E\_L_{coil}$(avg)** | **$\Delta E\_R_{coil}$(avg)** |
| 3 | 0.44% | 0 | 0.462% | 4.4e-3% | 0.487% | 0.094% |
| 20 | 0.45% | 0 | 0.743% | 4.9e-3% | 3.12% | 0.094% |

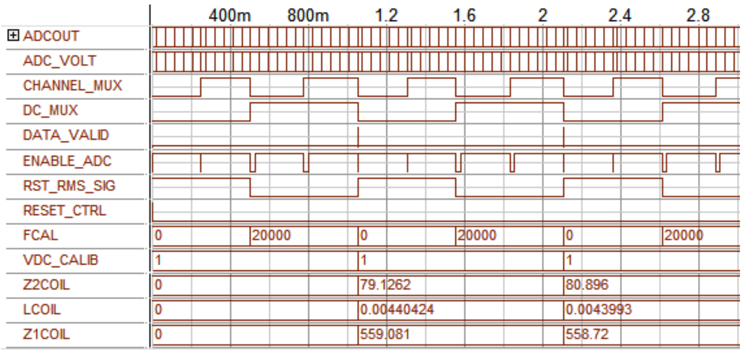
Table 4.8: Summary of the $L_{coil}$ and $R_{coil}$ estimation errors at two calibration signal frequencies ($F_1$) for three different TBCs. *Simulation conditions*: simulation tool = SMASH 6.4; integration method = Backward-Euler; maximum time step: $H_{max}$ = 0.5 $\mu$s. The average (avg) was taken for 10 consecutive estimated values.

Finally, the last TBC to analyze is the Simple $1^{st}$ Order ΣΔ ADC. This configuration uses the RMS-to-DC implicit computation and the $1^{st}$ Order ΣΔ ADC *model implementations*. In this analysis, we are interested in simulating the SCU system with a primary input current ($I_{pr}$). For this purpose, the RogoCoil asymmetric 1Loop model presented in Figure 4.7 on page 60 is used. Figure 4.39 shows the estimations for 2 self-calibration loops at two calibration signal frequencies. The average estimation errors for $L_{coil}$ and $R_{coil}$ are shown in Table 4.8. In this case, there is a significant increment in the $R_{coil}$ estimation error ($\Delta E\_R_{coil}$ = 0.094 %), which is caused by the presence of the 50 Hz signal component in the RMS-to-DC conversion algorithm and the subsequent ΣΔ modulation. Since $\Delta E\_R_{coil}$ is still very small, we can conclude that the digital low-pass Sinc$^3$ filter attenuates successfully the 50 Hz signal for this particular *test case*.

On the other hand, considering the $L_{coil}$ estimation error at $F_1$ = 3 kHz ($\Delta E\_L_{coil}$ = 0.487 %), the increase is not significant in comparison to the result of the previous configuration. However, at $F_1$ = 20 kHz, the $L_{coil}$ estimation error rocketed up ($\Delta E\_L_{coil}$ = 3.12 %), the reason can be observed in Figure 4.40. The approximation made for the self-calibration method requires that the calibration signal uses a frequency far below the first resonance



(a) $F_1$ = 3 kHz



(b) $F_1$ = 20 kHz

Figure 4.39: SCU VP (Simple $1^{st}$ Order ΣΔ ADC) transient simulations. *Verification conditions*: RogoCoil core = KEVCR; primary AC current: $I_{pr}$ = 10 $A_{peak}$, $F_{pr}$ = 50 Hz; ADC relevant parameters: $V_{REF}$ = 1 V, $V_{DELTA}$ = 4 V, $F_S$ = 22 kHz, $M$ = 1024; sinusoidal calibration signal with: $V_{DC}$ = 1 V, $V_{AMP}$ = 500 mV. *Simulation conditions*: simulation tool = SMASH 6.4; integration method = Backward-Euler; maximum time step: $H_{max}$ = 0.5 $\mu$s.

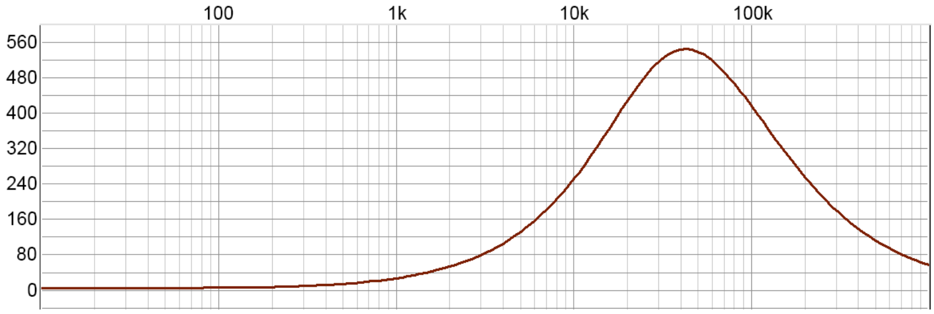


Figure 4.40: Impedance (dB) vs. Frequency (Hz). Impedance response simulation for the RogoCoil asymmetric 1Loop *model implementation*. The primary of the RogoCoil model is short-circuited for this simulation.

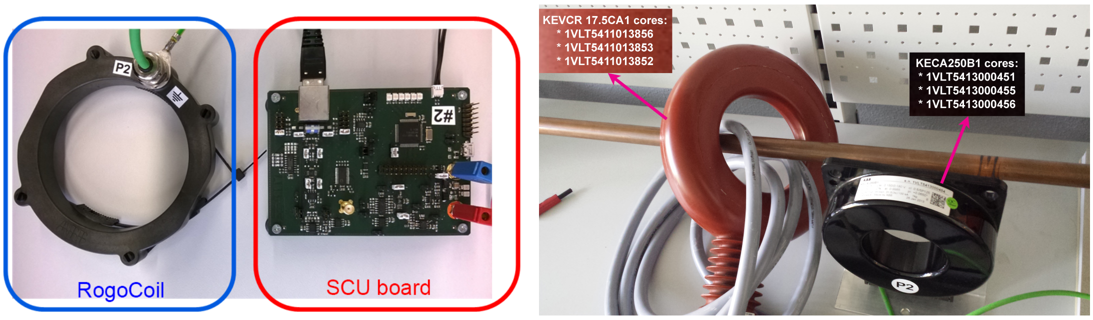
frequency of the RogoCoil. We can observe that the first resonance frequency is near to 40 kHz; so that, a calibration signal at 20 kHz produces a $L_{coil}$ estimation error that is out of the system specifications for RogoCoils with resonance frequencies of 40 kHz or below. Since $L_{coil}$

is proportional to the correction factor estimation, see subsection 4.4.1 on page 105, $\Delta E\_L_{coil}$ affects directly the accuracy of the primary current measurement, which is required to meet class 0.5 (IEC 60044-8 Standard [96]).

#### 4.3.2.2 Simulation and experimental results

Previous virtual and experimental studies of the RogoCoil SCU [80, 82] did not verify the behavior of the SCU when a primary current is applied. This is an essential verification procedure for testing the SCU operation under normal and critical conditions. This is why the availability of VPs allowing to make this type of verifications before physical prototyping offers great benefits for an efficient system design.

With the intention of validating the SCU VP, a comparison between simulation and experimental results using the physical prototype and the setup shown in Figure 4.41 was carried out by the author of this research. A series of $R_{coil}$ and $L_{coil}$ estimations of the SCU were measured on three batch samples for two RogoCoil core types, see Figure 4.41(b). Since the primary current generator available for the experiment can only produce a 50 Hz AC current up to 11 $\mathrm{A}_{peak}$ per wire (18 wires per cable) without saturation[7], 5-6 loops of the primary cable were used to emulate a Busbar as is shown in the experimental setup of Figure 4.41(b). The measurements for 6 consecutive self-calibration cycles were taken for each AC primary current step sweeping from 0 to 1.17 $\mathrm{kA}_{peak}$ at 50 Hz. The temperature of the primary cable rises considerably at high current; so that, the $R_{coil}$ and $L_{coil}$ measurements for larger self-calibration consecutive cycles cannot be taken without affecting the RogoCoil inner temperature. Although the thermal network model of the RogoCoil can be used to model this experimental scenario, the electro-thermal model of the RogoCoil also needs to be validated separately. In this case,



(a) (b)

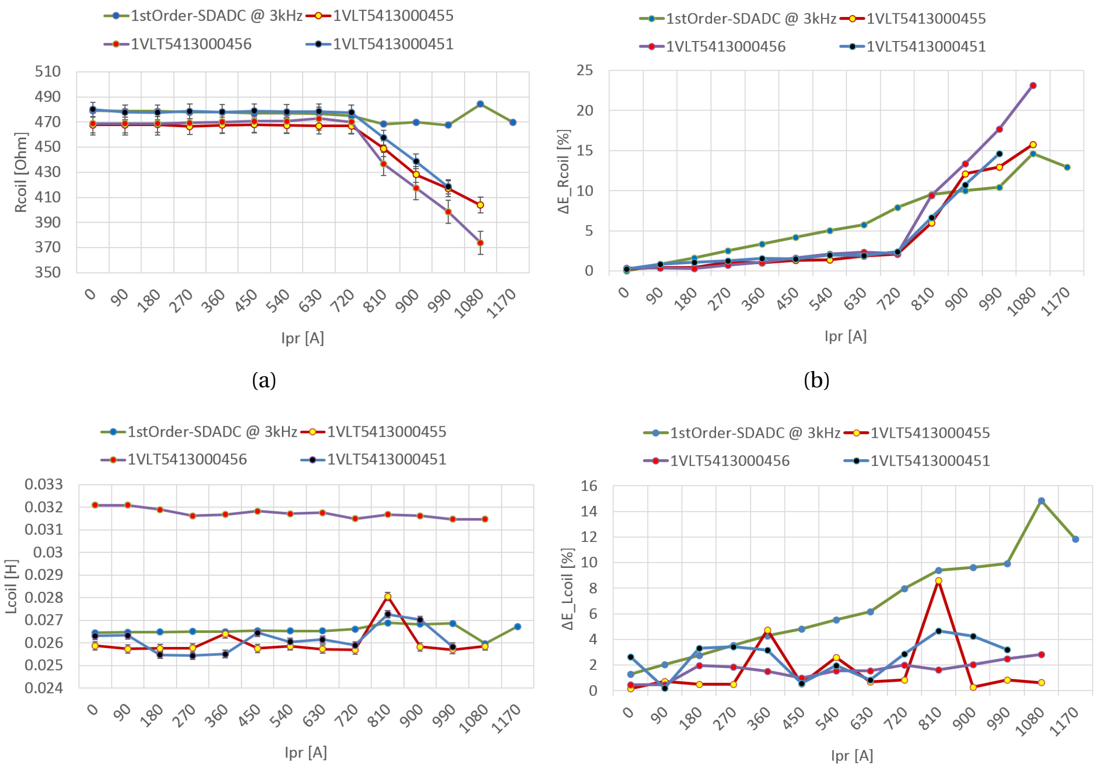
Figure 4.41: (a) SCU electronics physical prototype (red) connected with an RJ45 cable to a RogoCoil (blue). (b) KEVCR17.5CA1 and KECA250B1 ABB RogoCoils. In this experimental setup 5-6 loops of cable (18 wires per cable) are used to generate the equivalent magnetic field of a primary AC current up to 1.17 $\mathrm{kA}_{peak}$ at 50 Hz. Three batch samples per RogoCoil core type were used for the experimental measurements.

---

[7] At saturation, the AC current lose its sinusoidal waveform and becomes heavily distorted.

the system was periodically cold down between current measurements to avoid temperature influence on the electrical variables of the RogoCoil. The complete set of measurements were carried out in two days. On the other hand, the SCU VP were used to reproduce the same experimental verification scenario. The custom VHDL-AMS parametric package presented in Appendix E.2 was used to make the primary current sweep. The complete set of simulations were performed in about 4 hours with a maximum time step of 0.5 $\mu$s in an Intel i7-3770 CPU @ 3.4 GHz 16 GB RAM.

Figure 4.42 shows the plots of the $R_{coil}$ and $L_{coil}$ estimated values as a function of $I_{pr}$ using the KECA250B1 core type. The Simple $1^{st}$ Order $\Sigma\Delta$ ADC TBC, previously presented in subsection 4.3.2.1, was used for comparison between the virtual and experimental results. We can observe in Figure 4.42(a) that $R_{coil}$ remains stable until a current amplitude of about 720 A, afterwards $R_{coil}$ decreases proportionally to $I_{pr}$. Although the same behavior is not observed in simulation, it can be noticed a slight fluctuation on the $R_{coil}$ estimation after the same $I_{pr}$ amplitude in simulation. This observation is consistent with a significant increment in



(a) (b)

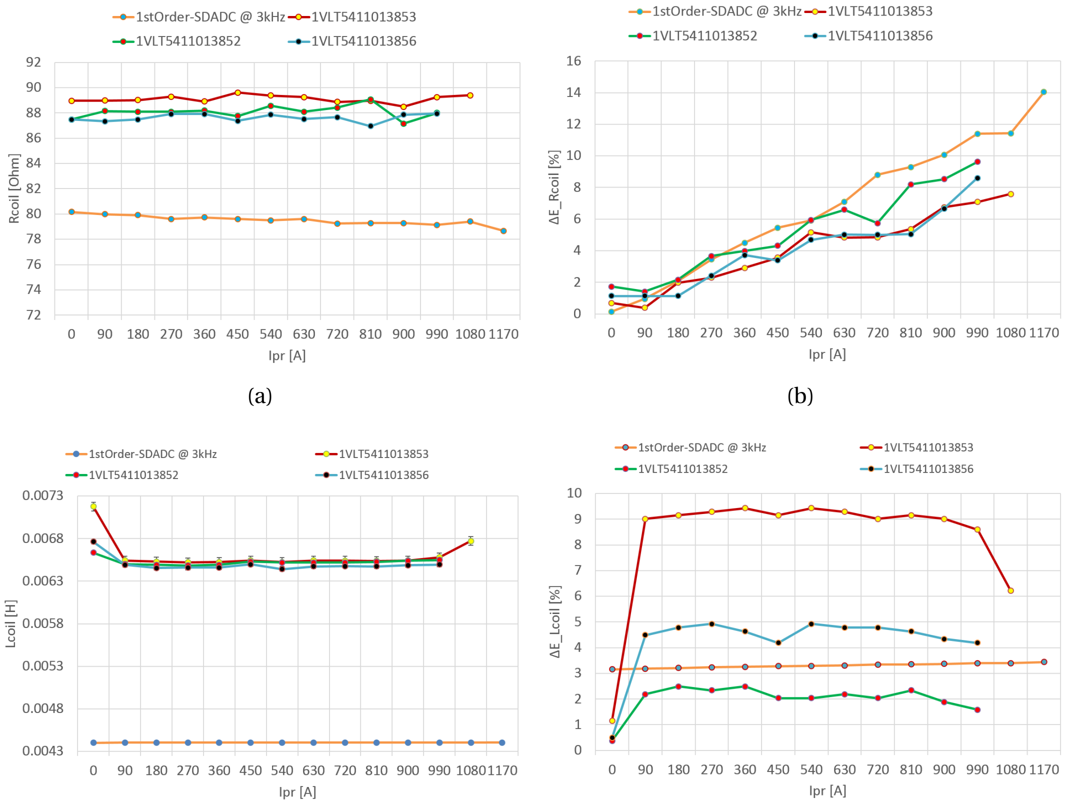
(c) (d)

Figure 4.42: KECA250B1 plots: experimental and virtual $R_{coil}$ and $L_{coil}$ SCU estimation as a function of the primary current amplitude ($I_{pr}$). (a) $R_{coil}$ average estimated value. (b) Maximum $R_{coil}$ relative error ($\Delta E\_R_{coil}(max)$). (c) $L_{coil}$ average estimated value. (d) Maximum $L_{coil}$ relative error ($\Delta E\_L_{coil}(max)$).

$\Delta E\_R_{coil}(max)$ also after the same $I_{pr}$ amplitude in Figure 4.42(b). In fact, the minimum estimated $R_{coil}$ values in simulation follow the same trend as the experimental values. However, the SCU VP also calculate increased $R_{coil}$ values at high primary current; and therefore, since the plotted $R_{coil}$ values in Figure 4.42(a) are average values $R_{coil}$ seems to remain stable. The exact reason for the unexpected $R_{coil}$ decrease in the experimental results is unknown, further analysis is required with both virtual and physical prototypes. A proportional small increment of both $\Delta E\_R_{coil}(max)$ and $\Delta E\_L_{coil}(max)$ is expected as a function of $I_{pr}$, this is caused by the 50 Hz signal component that is attenuated by the low-pass filtering in the ADC. The high $R_{coil}$ and $L_{coil}$ estimation values obtained in simulation suggest that a stronger 50 Hz signal attenuation and a better ADC accuracy is needed. Indeed, the physical prototype uses a $2^{nd}$ Order $\Sigma\Delta$ ADC architecture with an additional LPF in the signal conditioning chain, this is why higher relative maximum errors are obtained in simulation.

Regarding the $L_{coil}$ estimation shown in Figure 4.42(c), we can see that there is no significant difference between the experimental and the simulation values. The only exception is observed in the RogoCoil sample with reference `1VLT5413000456`, which presents a higher average $L_{coil}$ value. This is a manufacturing RogoCoil error of this particular coil sample, which presents a $L_{coil}$ around 23% higher than the nominal value. On the other hand, the experimental results for $\Delta E\_L_{coil}(max)$ do not unveil any clear trend as is shown in Figure 4.42(d). Contrarily, the simulation results show a linear increment of $\Delta E\_L_{coil}(max)$ as a function of $I_{pr}$.

Figure 4.43 presents the same plots of the $R_{coil}$ and $L_{coil}$ estimated values as a function of $I_{pr}$ using the KEVCR17.5CA1 core type. In this case, we do not observe any significant influence of $I_{pr}$ in the $R_{coil}$ estimation, see Figure 4.43(a). On the other hand, despite the apparent stability of the $L_{coil}$ estimation in both experimental and simulation cases, the experimental results show a noteworthy difference between the measurements with a certain primary current value and without a primary current, see Figure 4.43(c). Unfortunately, this estimation ‘jump’ is not observed in simulation; so that, the SCU VP is limited in its capability for helping to understand this particular behavior. The estimated $R_{coil}$ value in simulation is around 9% lower than the experimental measurements, whereas the estimated $L_{coil}$ value in simulation is around 32% lower. This kind of preliminary differences are normal between experimental and simulation results, they can be easily corrected by a model characterization process. These simulation results can fit the experimental measurements by a minimal modification of the material or geometrical parameter values of the RogoCoil core, for instance, a small increment of the resistivity of the coil cable or varying few hundreds of micrometer the core curvature. The characterized $R_{coil}$ and $L_{coil}$ values must be typical, lying inside the manufacturing tolerance range for this type of RogoCoil core, as it was done with the KECA250B1 core. Considering the maximum relative estimation errors in Figures 4.43(b) and 4.43(d), it is observed that both experimental and simulation results are much more similar and stable than in the KECA250B1 case. This allows to conclude that the KECA250B1 has some physical and/or geometrical condition that is not considered in the RogoCoil model; therefore, the RogoCoil behavior cannot be predicted accurately.



(a) (b)

(c) (d)

Figure 4.43: KEVCR17.5CA1 plots: experimental and virtual $R_{coil}$ and $L_{coil}$ SCU estimation as a function of the primary current amplitude ($I_{pr}$). (a) $R_{coil}$ average estimated value. (b) Maximum $R_{coil}$ relative error ($\Delta E\_R_{coil}(max)$). (c) $L_{coil}$ average estimated value. (d) Maximum $L_{coil}$ relative error ($\Delta E\_L_{coil}(max)$).

## 4.4 The current measurement unit of the Rogowski coil sensor

After the RogoCoil transducer, the current measurement unit (CMU) is indeed the most important part of the RogoCoil sensor system. As it can be observed in Figure 4.2 on page 55, the CMU continuously measures the primary current ($I_{pr}$) from the output voltage of the RogoCoil. Originally, the CMU can measure the primary current without using the SCU. In this case, the calibration factor ($CF$) is a fixed value measured once during the RogoCoil fabrication and typed in the IED during installation. However, since the $CF$ can change during the system operation by temperature drifts, mechanical stress, or aging, the continuous $CF$ calculation made by the SCU improves the accuracy of the $I_{pr}$ measurement over the entire lifetime of the system.

The first following subsection (4.4.1) details the equations of the primary current estimation which are used for the implementation of the complete VP of the RogoCoil sensor system, presented in section 4.5. Here, it is explained the actual value received by the CMU from the

SCU and how it is related to the estimated $L_{coil}$ and $R_{coil}$ values. The last subsection (4.4.2) presents a *virtual prototyping* example of the design of the CMU without considering the SCU implementation in the sensor system. This particular study case was made in order to illustrate the application of the meet-in-the-middle approach for system-level modeling at the former stage of the design, i.e. when the concept has not been previously analyzed. This study shows a simplified example of *design space exploration* and gradual *model refinement* using a VP built following the VP-Modeling Guidelines presented in section 3.5. The fast virtual assessment of two design options represented by two different architectures can be attained using this VP.

### 4.4.1 Equations for the primary current estimation

As it has been previously mentioned, the main function of the SCU is to compute the RogoCoil correction factor ($CF$) and transmit it to the CMU. Indeed, the $CF$ is the gain that the CMU requires to calculate accurately the current on the RogoCoil primary. However, the effect of the signal conditioning gain directly affects the primary current estimation, and therefore must be included in the $CF$ calculation. For this purpose, consider the self-calibration configuration shown in Figure 4.25(a) on page 84.

The output voltage of the RogoCoil sensor $V_{out}$ is defined as follows:

$$V_{out} = G \cdot M \cdot \frac{dI_{pr}}{dt} \tag{4.19}$$

where $G$ is the signal conditioning gain, $M$ is the mutual inductance of the RC, and $I_{pr}$ is the primary current. The mutual inductance $M$ is directly related to the RogoCoil sensor sensitivity $S$ as follows:

$$M = \frac{L_{coil}}{N} = \frac{S}{2\pi F_{pr}} \tag{4.20}$$

where $N$ is the number of loops of the RogoCoil and $F_{pr}$ is the frequency of the primary current (50 Hz / 60 Hz in power applications). Using equations 4.19 and 4.20, we can obtain the $I_{pr}$ as follows:

$$I_{pr}(t) = \frac{1}{G \cdot M} \int V_{out}(t)dt = \frac{2\pi F_{pr}}{G \cdot S} \int V_{out}(t)dt = \frac{N}{G \cdot L_{coil}} \int V_{out}(t)dt \tag{4.21}$$

On the other hand, the $CF$ is defined by the following equation:

$$CF = \frac{S}{S_{th}} = \frac{2\pi F_{pr} \cdot L_{coil}}{N \cdot S_{th}} \tag{4.22}$$

where $S_{th}$ is the experimental sensitivity of the RogoCoil which is defined by: $S_{th} = U_{sr}/I_{prn}$, where $U_{sr}$ is the output rated voltage of the RC (e.g. 150 mV for the KECA250B1 coil) and $I_{prn}$ is the primary rated current of the RC (e.g. 250 A for the KECA250B1 coil).

The $CF$ value is normally measured and written in RogoCoil after fabrication using Equation 4.22, which determines how $CF$ is related to $S$. However, the electrical abstractions used for modeling the RogoCoil, do not take into account the experimental RogoCoil sensitivity effect. Therefore, $S_{th} = 1$ and we can use directly the expression in function of $L_{coil}$ and $N$ in order to calculate $I_{pr}$, see Equation 4.21.

In this way, the only remaining term to define is the gain of the signal conditioning chain ($G$). If the RogoCoil is directly connected to the CMU, without implementing the self-calibration circuit described in Figure 4.25(a), $G$ will be composed exclusively by the gain of the signal conditioning of the CMU components, i.e. analog amplifiers, integrators, filters, etc. This is the case for the *virtual prototyping* example presented in subsection 4.4.2. However, for the complete RogoCoil self-calibrated sensor system, $G$ can be defined by the product of two parts of the signal conditioning chain ($G = G_1 \cdot G_2$). These parts are described as follows:

**1. Attenuation of the calibration resistors:** Since the calibration resistance value $R_{CAL}$ (10 kΩ) is much smaller than the standard input impedance of the previous IED versions (around 2 MΩ), the RogoCoil rated signal will then suffer from an attenuation that must be compensated. The RogoCoil can be represented as a floating differential voltage source with an internal $R_{coil}$ output resistor as depicted in Figure 4.44. Therefore, the attenuation of the calibration resistors ($G_1$) is given as follows:

$$G_1 = \frac{R_{CAL}}{R_{CAL} + R_{coil}/2} \tag{4.23}$$

**2. Front-end analog gain:** The two voltage drops in the order of mili-Volts ($V_{CAL} - V_{out}$ and $V_{out}$), shown in Figure 4.25(a) on page 84, can be measured by analog amplifiers using a determined gain ($G_2$). In order to simplify the calculations done in the complete VP of the RogoCoil sensor system, $V_{out}$ is measured with a unitary gain, i.e. $G_2 = 1$. Different $G_2$ values can be easily included in the `Voltage Sensing Block` model of the SCU, see Figure 4.26 on page 87. In this case, the correction factor calculated by the digital control unit in the SCU must be modified accordingly.

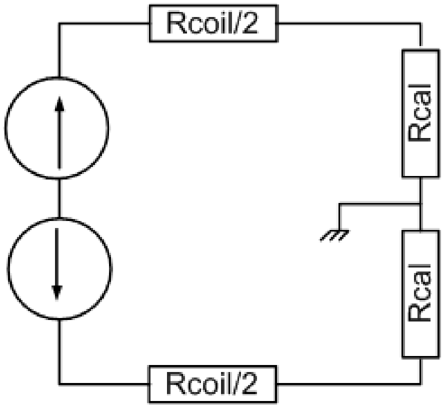


Figure 4.44: Rogowski coil symmetrical Thevenin equivalent circuit and its load.

Considering Equation 4.21 and $G = G_1$, the corrected correction factor ($CF_c$), which is the actual value calculated by the SCU and transmitted to the CMU, is given by the following expression:

$$CF_c = \frac{N}{G \cdot L_{coil}} = \frac{N(R_{coil}/2 + R_{CAL})}{R_{CAL} \cdot L_{coil}} \tag{4.24}$$

$CF_c$ takes into account both the RogoCoil sensitivity and the attenuation of the calibration signal conditioning. Therefore, the primary current equation can be re-written as follows:

$$I_{pr}(t) = CF_c \int V_{out}(t)dt = \frac{N(R_{coil}/2 + R_{CAL})}{R_{CAL} \cdot L_{coil}} \int V_{out}(t)dt \tag{4.25}$$

### 4.4.2 The CMU virtual prototype

Table 4.9 unveils some of the most relevant design enquiries that have arisen during the CMU design at the former design stage. Specifically, this *virtual prototyping* analysis provides a fast assessment focused on the first 5 design topics. It is assumed that some *component models*, such as the RogoCoil, the ADC, and the OpAmp models, are previously available. In this initial virtual assessment, the goal is not to achieve a detailed design of the system, it is rather to explore and compare two different architectures for accurate primary current estimation. Early error detection and the quantification of some critical behavioral effects are the most important results obtained during this study.

The CMU VP presented in this subsection only takes into account the RogoCoil and the CMU subsystems as is shown in Figure 4.45. This is a simple high-level block diagram of the RogoCoil and the CMU interconnection. The DUV is declared in the *test bench* called

| Design topic | Design enquiry |
|---|---|
| Functional verification | Does the current measurement equation/algorithm work? |
| Design space exploration | Analog or Digital current calculation approach? / Software or Hardware solution? / What integration method? |
| Performance estimation | Does the CMU proposed design meet the requirements? (Accuracy and current measurement range) |
| Identification of critical design parameters | RogoCoil voltage measurement gain; ADC frequency, resolution, dynamic range, etc; digital processing clock frequency. |
| Early design error detection | Does the current calculation depend on the input signal initial condition (Phase)? |
| Evaluation of key non-idealities of the system components | Important design variables and parameters that put at risk the correct operation of the system: e.g. offset voltages, bandwidth, tolerances, noise, etc. |
| Functional safety | Verification of critical operation/environmental conditions and/or hardware failures: e.g. harmonics in the primary current, temperature fluctuations, current breakdowns, etc. |

Table 4.9: Design topics of system-level *virtual prototyping* for the CMU design. The CMU VP is used to solve these type of design enquiries classified in general design topics normally treated in this order.

`RC_MEAS_TB`, which provides the primary current ($I_{pr}$) and a reset signal for the CMU. As it has been described in section 4.4.1, the output voltage of the RogoCoil is described by Equation 4.19. Knowing that the coil mutual inductance ($M$) is directly proportional to $L_{coil}$ and inversely proportional to the number of coil turns ($N$), see Equation 4.20, the CMU must calculate continuously $I_{pr}(t)$ by implementing the Equation 4.21. This is called the **reference calculation** and is given by the output digital signal: `Ibus_ideal` = $\frac{N}{G \cdot L_{coil}} \int V_{out}(t)dt$, where

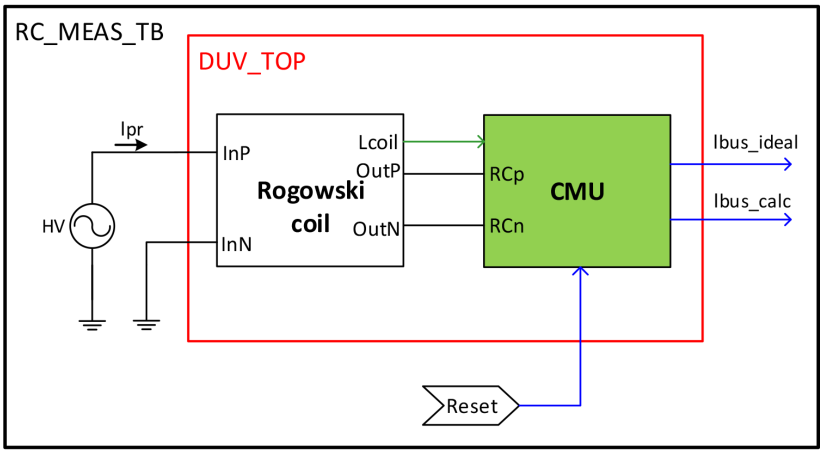


Figure 4.45: Conceptual block diagram of the CMU VP. The DUV is declared in the *test bench* called `RC_MEAS_TB`, which provides the primary current $I_{pr}$ and a reset signal for the CMU.

$N$ is a known PMP and $L_{coil}$ is also assumed a known value. Note that no calibration circuit is connected to the RogoCoil secondary output terminals; therefore, $G$ only depends on the gain for the RogoCoil output voltage ($V_{out}$) given in the CMU.

Assuming that the RogoCoil is an available *component model*, any of the RogoCoil *model implementations* previously described can be used. A pertinent model for starting the functional verification may be a high-level functional RogoCoil model which implements Equation 4.1 on page 54; however, we can directly use the available RogoCoil asymmetric 1Loop model presented in Figure 4.7 on page 60, see its typical transient response in Figure 4.46. Since this model calculates its impedance value from the geometrical and material parameters according to the specific RogoCoil core, the $L_{coil}$ value used in the CMU is given by the RogoCoil model using a quantity port as is depicted in Figure 4.45.

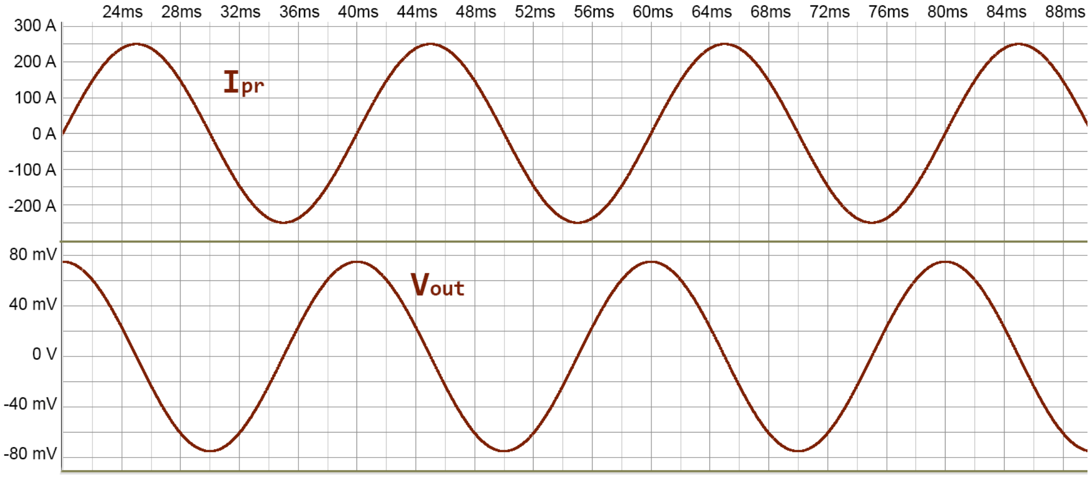
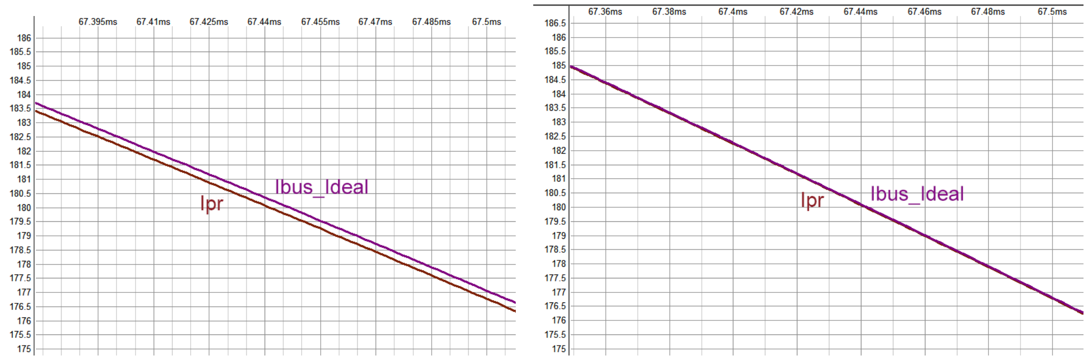


Figure 4.46: Transient response of the RogoCoil asymmetric 1Loop model. RogoCoil core = KEVCR (see the PMPs in Table A.1). $I_{pr} = 250\ \mathrm{A}_{peak}$ @ 50 Hz. $V_{out}$ is a perfect derivate of the input primary current waveform of about 75 $\mathrm{mV}_{peak}$ at the same frequency.

During the first analysis, an initial functional verification, the *simulator setup* parameters are fixed in order to get a good simulation performance with a proper model estimation accuracy over the variable that we want to verify, in this case, the measured primary current. The estimation accuracy is defined by simulating the CMU VP with the reference calculation model, where the output signal `Ibus_ideal` should be exactly equal to the $I_{pr}$ input signal (i.e. a successful functional verification result). Figure 4.47(a) shows an observable error between `Ibus_ideal` and $I_{pr}$, which can be quantified using different criteria such as the Average (AVG), RMS, peak-to-peak (P2P), or crest factor (CRESTF) errors for sinusoidal signals. Since these estimation errors are much higher for the KECA core than for the KEVCR core under the same *simulation conditions*, we can conclude that the KECA parameters are not fine tuned for the RogoCoil asymmetric 1Loop model with this specific verification and simulation conditions. Considering that this type of simulation errors are always present disregarding the model used (**systematic errors**), the initial reference calculation model becomes useful for making relative



(a) KECA (b) KEVCR

Figure 4.47: CMU VP transient simulations using the reference calculation model (Zoom). AC input signal: $I_{pr} = 250\ \mathrm{A}_{peak}$ @ 50 Hz. The output (`Ibus_ideal`) looks exactly the same as $I_{pr}$ in Figure 4.46 without zoom.

comparisons of the detriment in the model estimation accuracy caused by *model refinement*, changes in *verification conditions*, and changes in *simulation conditions*. In fact, since system designers are mainly interested in errors caused by changes in the *verification conditions*, it is necessary to check that the error caused by a certain *simulation condition* remain constant for a proper quantitative comparison under different *verification conditions*.

Let us continue with the *design space exploration* study. As it was presented in Figure 3.1 on page 23, there are two potential architectures that are considered for implementing the desired functionality given by the `Ibus_ideal` expression: the **Analog CMU** and the **Digital CMU** architectures.

#### 4.4.2.1 The Analog CMU architecture

The Analog CMU architecture shown in Figure 4.48 consist of four main blocks: an initial amplification stage implemented by an instrumentation amplifier (`INS_AMP`), an `Analog Integration` block, an ADC, and a final `Digital Calculation` block that multiplies the digital signal (previously integrated in the analog domain) by the correction factor ($CF$), defined only for the analysis of the CMU VP as $CF = \frac{N}{G \cdot L_{coil}}$. In this way, the `Ibus_calc` output signal represents the measured primary current. Note that the `Reference Calculation` block, which is not part of the CMU system architecture is included in the same *model implementation* to obtain in parallel the `Ibus_ideal` signal for comparison purposes.

Figure 4.48 shows the different *model implementations* that are available for each *component model*, they all include one basic *functional model* abstraction and at least one more refined *model implementation* that allows evaluating critical behaviors of the system. For example, the ADC model can be configured using one of the three *model implementations* previously explained in section 4.3.2, i.e. the **functional** ideal ADC model shown in Figure 4.31 on

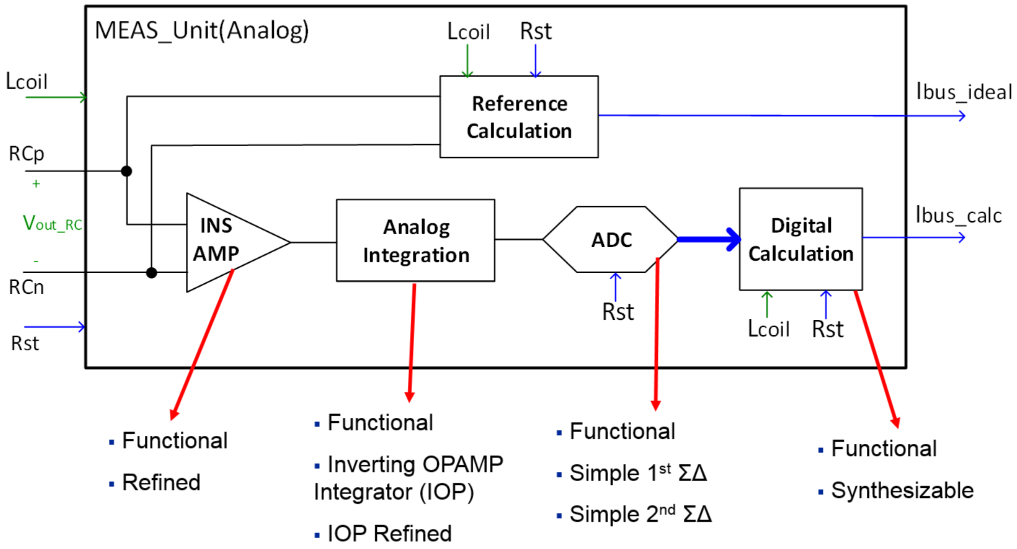


Figure 4.48: Conceptual CMU Analog architecture. The `Reference Calculation` block is not part of the real system architecture. The red arrows indicate the available *model implementations* for each *component model.*

page 92, and the first and second order ΣΔ ADC *model implementations* shown in Figure 4.32 and detailed in Appendix C.1. Following the meet-in-the-middle approach, the virtual analysis implies a gradual refinement of the CMU VP in order to identify the effect of critical design parameters in the accuracy of the primary current estimation. This is achieved by verifying the correct functionality and performance of the CMU in simulation. Thus, some basic configurations[8] using specific combinations of the available *model implementations* are presented as follows. The goal is not to explain every possible VP configuration, it is rather to show how to perform a quick system-level analysis using the proposed VP-based methodology.

**1. Ideal CMU configuration:** This configuration sets the most basic functional *model implementations* for each *component model.* Table 4.10 shows the main equations of those functional *component models*: where `GAIN_VRC` is the gain of the INS_AMP component, `vo`

| Component model | Main equation |
|---|---|
| **INS_AMP** | `vo == GAIN_VRC * vi;` |
| **Analog Integration** | `Vout == GAIN_INTEG * Vin'integ;` |
| **Digital Calculation** | `Ibus_calc <= N/(GAIN_INS*Lcoil) * ADCout;` |

Table 4.10: Main equations of the CMU functional *component models*. The ideal ADC model shown in Figure 4.31 is used in the ideal CMU configuration together with these models.

and `vi` are the output and input voltage of the INS_AMP amplifier respectively; `GAIN_INTEG`

---

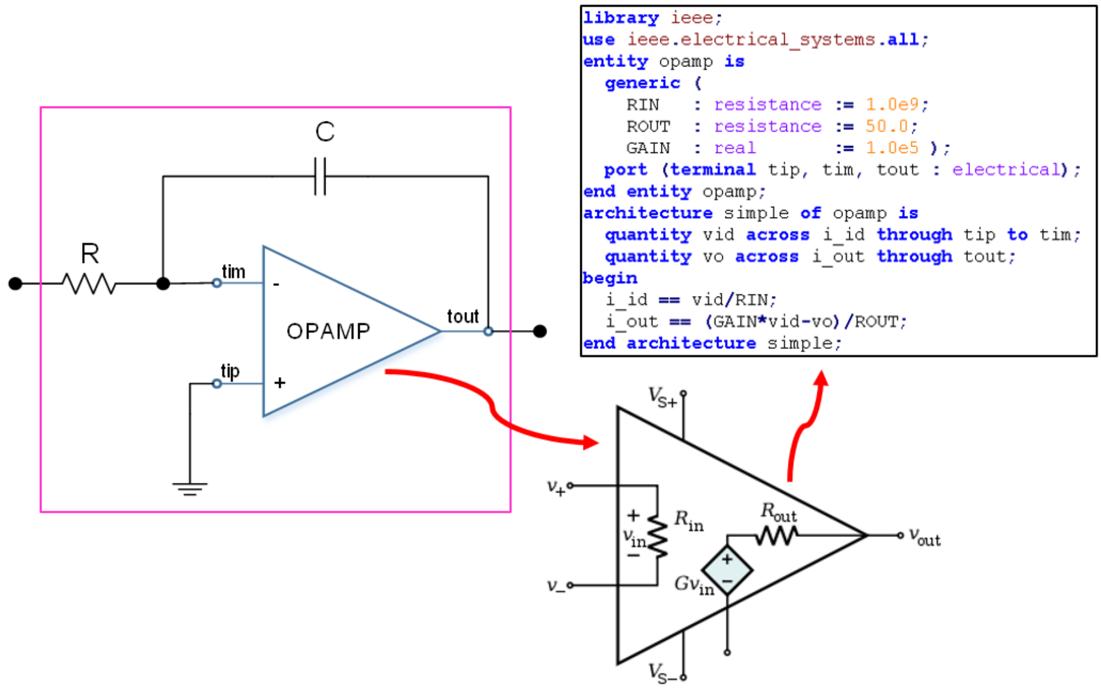
[8] The name used for a particular configuration is merely indicative. Unfortunately, it cannot fully indicate the complete set of implemented design entities in a large hierarchical model, the name would be too large.

is the gain of the Analog Integration block, `Vout` and `Vin` are the output and input voltage of the Analog Integration block respectively; `ADCout` is the digital output value[9] of the ADC, `GAIN_INS` is the total gain of the instrumentation chain circuitry (in this case `GAIN_INS = GAIN_VRC * GAIN_INTEG`).

**2. Behavioral CMU configuration:** Only the Analog Integration block is refined in this configuration. All the other *component models* use the same functional *model implementations* of the Ideal CMU configuration. Figure 4.49 exhibits the Analog Integration block architecture, which is the well-known inverting OpAmp integrator circuit using a resistor ($R$) and a capacitor ($C$) in the feedback loop. The output voltage of this integrator ($V_{integ}$) is given as follows:

$$V_{integ} = \frac{-1}{R \cdot C} \cdot \int V_{in} dt \tag{4.26}$$

where $V_{in}$ is the input voltage of the integrator. Since $R$ and $C$ particular values are not important for the integration functionality, they are defined as SMPs depending on the `GAIN_INTEG` PMP, see modeling guideline number 4 presented in section 3.5.



```
library ieee;
use ieee.electrical_systems.all;
entity opamp is
  generic (
    RIN   : resistance := 1.0e9;
    ROUT  : resistance := 50.0;
    GAIN  : real       := 1.0e5 );
  port (terminal tip, tim, tout : electrical);
end entity opamp;
architecture simple of opamp is
  quantity vid across i_id through tip to tim;
  quantity vo across i_out through tout;
begin
  i_id == vid/RIN;
  i_out == (GAIN*vid-vo)/ROUT;
end architecture simple;
```

Figure 4.49: Analog Integration block architecture and the `opamp(simple)` simple model.

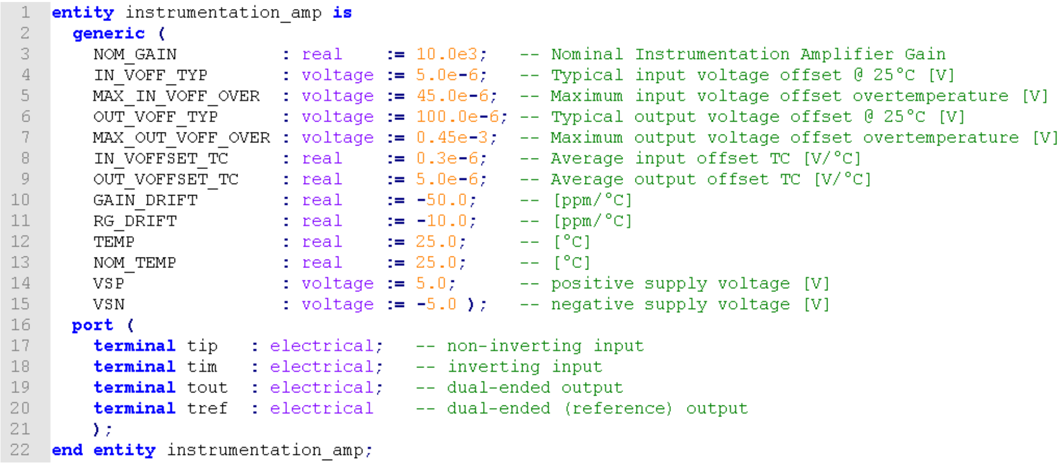
Likewise, the OpAmp simple *model implementation* shown in Figure 4.49, models the OpAmp as a voltage dependent current source with finite gain, input resistance, and output resistance.

---

[9] This is a real number that has been converted from the standard logic vector output of the ADC.

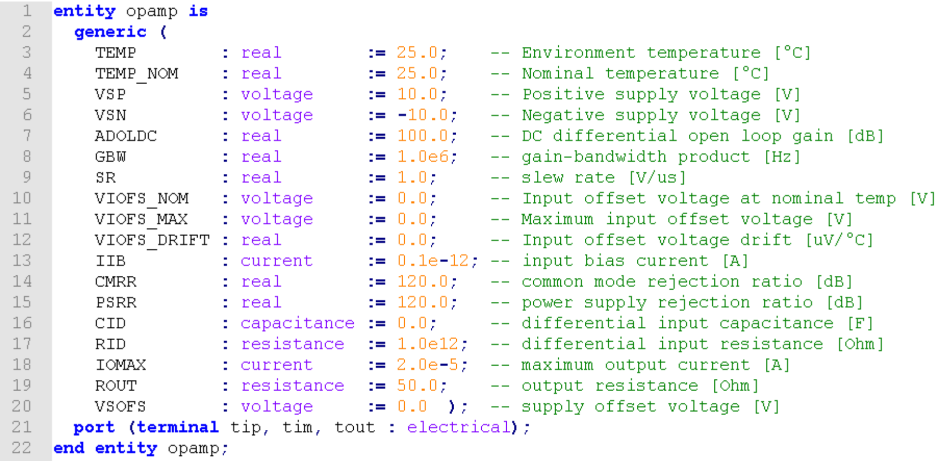
This model is adequate to be implemented in feedback loops but does not model the OpAmp voltage saturation determined by the supply voltages.

**3. Behavioral_refined CMU configuration:** In this configuration, the interest is focused on verifying the effect of some of the most important non-ideal behaviors of the differential amplifiers, i.e. in the `INS_AMP` and the `OPAMP` *component models*. Figure 4.50 shows the generic parameters of the more refined *model implementation* for these components. The `INS_AMP` model considers non-ideal behaviors like saturation voltages, input and output offset voltages, and the effect caused by temperature drifts in the offset voltages and the input resistance, see Figure 4.50(a). On the other hand, a more refined OpAmp *model implementation* (denominated `opamp(nonideal)`), considers non-ideal effects commonly given in datasheets of



```
entity instrumentation_amp is
  generic (
    NOM_GAIN          : real    := 10.0e3;   -- Nominal Instrumentation Amplifier Gain
    IN_VOFF_TYP       : voltage := 5.0e-6;   -- Typical input voltage offset @ 25°C [V]
    MAX_IN_VOFF_OVER  : voltage := 45.0e-6;  -- Maximum input voltage offset overtemperature [V]
    OUT_VOFF_TYP      : voltage := 100.0e-6; -- Typical output voltage offset @ 25°C [V]
    MAX_OUT_VOFF_OVER : voltage := 0.45e-3;  -- Maximum output voltage offset overtemperature [V]
    IN_VOFFSET_TC     : real    := 0.3e-6;   -- Average input offset TC [V/°C]
    OUT_VOFFSET_TC    : real    := 5.0e-6;   -- Average output offset TC [V/°C]
    GAIN_DRIFT        : real    := -50.0;    -- [ppm/°C]
    RG_DRIFT          : real    := -10.0;    -- [ppm/°C]
    TEMP              : real    := 25.0;     -- [°C]
    NOM_TEMP          : real    := 25.0;     -- [°C]
    VSP               : voltage := 5.0;      -- positive supply voltage [V]
    VSN               : voltage := -5.0 );   -- negative supply voltage [V]
  port (
    terminal tip   : electrical;   -- non-inverting input
    terminal tim   : electrical;   -- inverting input
    terminal tout  : electrical;   -- dual-ended output
    terminal tref  : electrical    -- dual-ended (reference) output
    );
end entity instrumentation_amp;
```

(a) `INS_AMP`



```
entity opamp is
  generic (
    TEMP         : real        := 25.0;     -- Environment temperature [°C]
    TEMP_NOM     : real        := 25.0;     -- Nominal temperature [°C]
    VSP          : voltage     := 10.0;     -- Positive supply voltage [V]
    VSN          : voltage     := -10.0;    -- Negative supply voltage [V]
    ADOLDC       : real        := 100.0;    -- DC differential open loop gain [dB]
    GBW          : real        := 1.0e6;    -- gain-bandwidth product [Hz]
    SR           : real        := 1.0;      -- slew rate [V/us]
    VIOFS_NOM    : voltage     := 0.0;      -- Input offset voltage at nominal temp [V]
    VIOFS_MAX    : voltage     := 0.0;      -- Maximum input offset voltage [V]
    VIOFS_DRIFT  : real        := 0.0;      -- Input offset voltage drift [uV/°C]
    IIB          : current     := 0.1e-12;  -- input bias current [A]
    CMRR         : real        := 120.0;    -- common mode rejection ratio [dB]
    PSRR         : real        := 120.0;    -- power supply rejection ratio [dB]
    CID          : capacitance := 0.0;      -- differential input capacitance [F]
    RID          : resistance  := 1.0e12;   -- differential input resistance [Ohm]
    IOMAX        : current     := 2.0e-5;   -- maximum output current [A]
    ROUT         : resistance  := 50.0;     -- output resistance [Ohm]
    VSOFS        : voltage     := 0.0  );   -- supply offset voltage [V]
  port (terminal tip, tim, tout : electrical);
end entity opamp;
```

(b) `OPAMP`

Figure 4.50: Differential amplifiers VHDL-AMS entity declaration. The list of generic parameters show the behaviors included in the most refined *model implementation* of these *component models*. See the complete design entities code in Appendices C.2 and C.3. The default parameter values are typical in commercial amplifiers.

commercial components such as gain-bandwidth product, slew-rate, common mode rejection ratio, power supply rejection ratio, among others, see Figure 4.50(b).

Similarly, other CMU configurations can be considered to identify critical design parameters of the system components by simulating with more refined *model implementations* for the ADC and the Digital Calculation block. Note that is not necessary to refine at the same time the ADC and the Digital Calculation models for studying the critical behaviors of the differential amplifiers in the first components of the instrumentation chain. Likewise, if we would like to improve the simulation performance for studying non-ideal behaviors in the ADC or the Digital Calculation models, we can use the *functional models* of the INS_AMP and the Analog Integration block.

#### 4.4.2.2 The Digital CMU architecture

Continuing with the *design space exploration* of the CMU, the other considered approach for measuring the primary current is to convert to digital the output voltage of the instrumentation amplifier. Consequently, the integration and the multiplication by the *CF* is performed in the digital domain, see the conceptual architecture for this case in Figure 4.51.

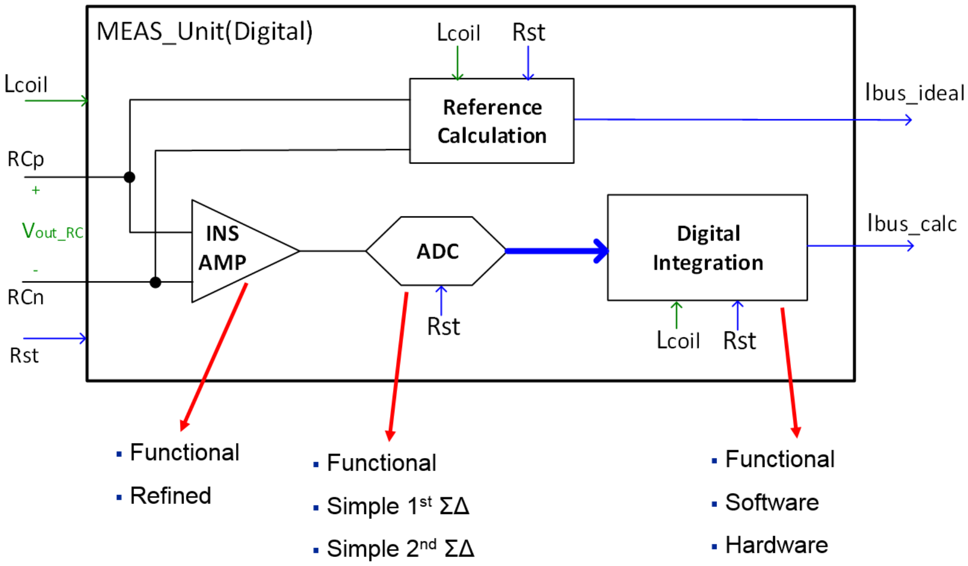


Figure 4.51: Conceptual CMU Digital architecture. The `Reference Calculation` block is not part of the real system architecture. The red arrows indicate the available *model implementations* for each *component model.*

The considered *model implementations* for the INS_AMP and the ADC components are the same as the Analog CMU architecture. For the digital CMU analysis and comparison with the Analog counterpart, there were considered three CMU configurations, each one setting a particular *model implementation* of the **Digital Integration** block together with the *functional* *models* of the INS_AMP and the ADC components. These CMU configurations are described as follows:

**1. Functional CMU configuration:** This configuration uses the **functional** *model implementation* of the Digital Integration block, which implements in VHDL-AMS an algorithm that can integrate the input digital signal using the three different methods as it can be observed in Table 4.11. Here we observe the approximation taken in each method for the continuous Laplace transfer function $(1/s)$ as a function of the sample time $T$ and the discrete $z$-domain variable. The resultant difference equation for each method is also given.

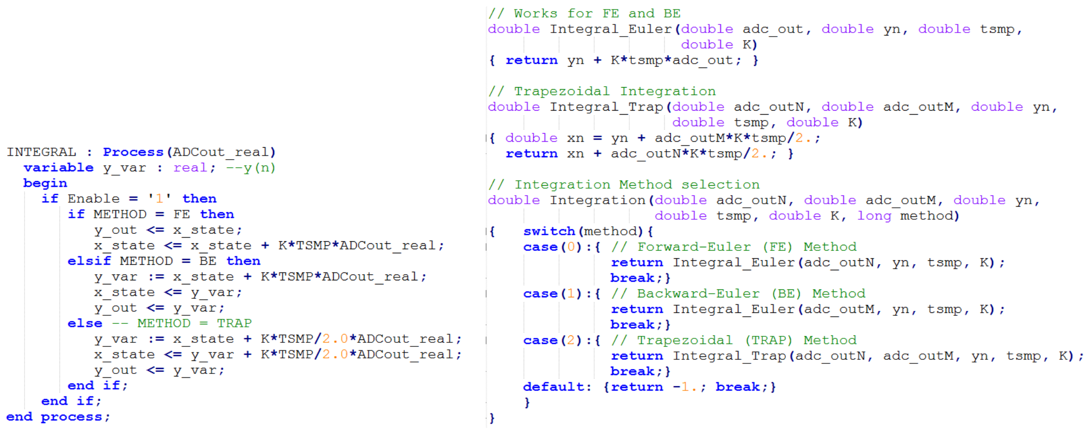
| Integration method | Equation |
|---|---|
| **Forward Euler (FE)** | $\frac{1}{s} \approx \frac{T}{z-1} \Rightarrow y(n) = y(n-1) + K \cdot T \cdot u(n-1)$ |
| **Backward Euler (BE)** | $\frac{1}{s} \approx \frac{T \cdot z}{z-1} \Rightarrow y(n) = y(n-1) + K \cdot T \cdot u(n)$ |
| **Trapezoidal (TRAP)** | $\frac{1}{s} \approx \frac{T}{2} \cdot \frac{z+1}{z-1} \Rightarrow y(n) = y(n-1) + K\frac{T}{2}[u(n) + u(n-1)]$ |

Table 4.11: Digital integration methods of the Digital Integration model. The `functional` and the `software` *model implementations* of this block use these integration methods, where $u$ is the input, $y$ is the output, $K$ is the integrator gain value, and $T$ is the sample time.

**2. Software CMU configuration:** This configuration uses the **software** *model implementation* of the Digital Integration block. This implements the same algorithm of the functional *model implementation* but in C++. Figure 4.52(a) shows the VHDL-AMS process used to implement the integration methods of the `functional` architecture of the Digital Integration block. Likewise, Figure 4.52(b) shows the C++ functions used to implement the same integration methods in the software architecture of the Digital Integration block. In this case, the `FOREIGN` attribute allows calling C/C++ functions in VHDL-AMS. The C++ function '`Integration`' is called from a VHDL-AMS process as follows:

```
INTEGRAL : Process(ADCout_real)
  begin
  y_out <= FOREIGN_Integration(ADCout_real, ADCout_realnm1, y_out, TSMP,
    K, METHOD);
end process;
```

where `ADCout_real` is the ADC output at time $t$, the `ADCout_realnm1` is the ADC output at time $t-1$, `y_out` is the output of the integrator, `TSMP` is the sample time, `K` is the gain value, and `METHOD` is the integration method ('FE', 'BE', or 'TRAP').



```vhdl
INTEGRAL : Process(ADCout_real)
  variable y_var : real; --y(n)
  begin
    if Enable = '1' then
       if METHOD = FE then
          y_out <= x_state;
          x_state <= x_state + K*TSMP*ADCout_real;
       elsif METHOD = BE then
          y_var := x_state + K*TSMP*ADCout_real;
          x_state <= y_var;
          y_out <= y_var;
       else -- METHOD = TRAP
          y_var := x_state + K*TSMP/2.0*ADCout_real;
          x_state <= y_var + K*TSMP/2.0*ADCout_real;
          y_out <= y_var;
       end if;
    end if;
end process;
```

(a) VHDL-AMS

```cpp
// Works for FE and BE
double Integral_Euler(double adc_out, double yn, double tsmp,
                      double K)
{ return yn + K*tsmp*adc_out; }

// Trapezoidal Integration
double Integral_Trap(double adc_outN, double adc_outM, double yn,
                     double tsmp, double K)
{ double xn = yn + adc_outM*K*tsmp/2.;
  return xn + adc_outN*K*tsmp/2.; }

// Integration Method selection
double Integration(double adc_outN, double adc_outM, double yn,
                   double tsmp, double K, long method)
{   switch(method){
    case(0):{ // Forward-Euler (FE) Method
              return Integral_Euler(adc_outN, yn, tsmp, K);
              break;}
    case(1):{ // Backward-Euler (BE) Method
              return Integral_Euler(adc_outM, yn, tsmp, K);
              break;}
    case(2):{ // Trapezoidal (TRAP) Method
              return Integral_Trap(adc_outN, adc_outM, yn, tsmp, K);
              break;}
    default: {return -1.; break;}
    }
}
```

(b) C++

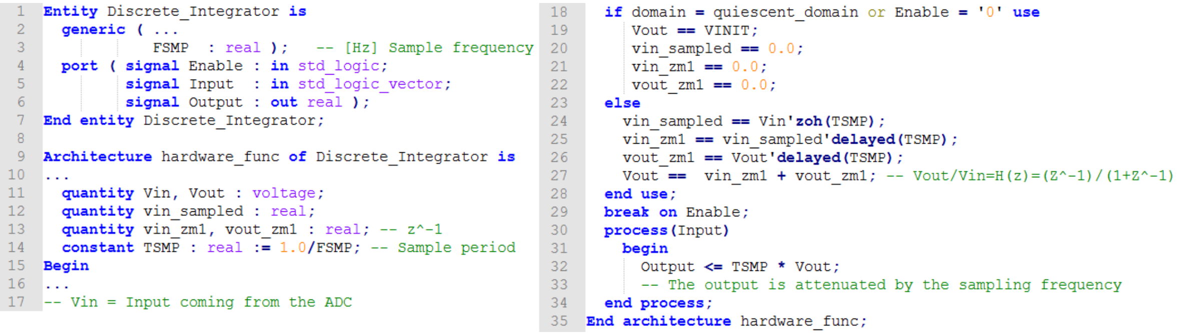
Figure 4.52: Source code fragment of the `functional` and `software` *model implementations* of the Digital Integration block. (a) VHDL-AMS process, part of the `functional` architecture. (b) C++ functions, part of the `software` architecture.

Although the `software` is not a refinement of the `functional` *model implementation*, they were used to compare simulation accuracy and performance in the study presented in subsection 4.4.2.3. This simple example demonstrates the versatility of the VHDL-AMS approach and the potential to integrate hardware and software abstractions in the same model.

**3. Hardware CMU configuration:** This configuration uses a more refined *model implementation* of the Digital Integration block, a first order IIR digital integrator described by the forward rectangular discrete transfer function shown in Equation 4.27.

$$y(n) = y(n-1) + T \cdot u(n-1) \Rightarrow H(z) = T\left(\frac{z^{-1}}{1-z^{-1}}\right) \tag{4.27}$$

Figure 4.53 exhibits a fraction of the VHDL-AMS code used to implement the forward rectangular discrete transfer function. In fact, this is an implementation of the FE method described in Table 4.11. This *model implementation* can be refined by making it fully synthesizable in VHDL. In that case, the algorithm will only use standard logic vectors, accumulators (registers) and a multiplier. For a first analysis and comparison of the different integration methods, the VHDL-AMS `Hardware` model abstraction is sufficient.



```
Entity Discrete_Integrator is
  generic ( ...
            FSMP  : real );   -- [Hz] Sample frequency
  port ( signal Enable : in std_logic;
         signal Input  : in std_logic_vector;
         signal Output : out real );
End entity Discrete_Integrator;

Architecture hardware_func of Discrete_Integrator is
...
  quantity Vin, Vout : voltage;
  quantity vin_sampled : real;
  quantity vin_zm1, vout_zm1 : real; -- z^-1
  constant TSMP : real := 1.0/FSMP; -- Sample period
Begin
...
-- Vin = Input coming from the ADC
  if domain = quiescent_domain or Enable = '0' use
     Vout == VINIT;
     vin_sampled == 0.0;
     vin_zm1 == 0.0;
     vout_zm1 == 0.0;
  else
    vin_sampled == Vin'zoh(TSMP);
    vin_zm1 == vin_sampled'delayed(TSMP);
    vout_zm1 == Vout'delayed(TSMP);
    Vout ==  vin_zm1 + vout_zm1; -- Vout/Vin=H(z)=(Z^-1)/(1+Z^-1)
  end use;
  break on Enable;
  process(Input)
    begin
      Output <= TSMP * Vout;
      -- The output is attenuated by the sampling frequency
  end process;
End architecture hardware_func;
```

Figure 4.53: Code fragment of the `Hardware` *model implementation* of the Digital Integration block. This is the VHDL-AMS implementation of Equation 4.27.

#### 4.4.2.3 Comparative study: Analog vs. Digital

Following the design topics mentioned in Table 4.9 on page 108, this study consists of evaluating the effect of some of the PMPs from the main components of the CMU. This is carried out by comparing the important waveform properties between the `Ibus_calc` and the `Ibus_ideal` signals for all the presented configurations of both the Analog and Digital architectures of the CMU. The idea is to find critical PMPs values of the main components. This is done by assigning relaxed PMPs values that can easily perform the required functionality in the ideal CMU configuration, where the ADC resolution ($NB$), ADC clock frequency ($F_{ADC}$), ADC Full-scale voltage ($V_{fs}$), INS_AMP gain ($G_{INS\_AMP}$), and integrator gain ($G_{INTEG}$) are among the important parameters to take into account for a reliable current estimation. Table 4.12 shows the default simulation conditions and parameter values for the seed *test case*, which is a stable set of PMPs values and simulator settings that are chosen as the starting point after functional verification simulations.

| Primary parameters | Simulation conditions |
|---|---|
| RogoCoil core = KEVCR.<br>$I_{pr}$ =250 A @ 50 Hz, $\theta$ =0°.<br>$F_{ADC}$=500 kHz.<br>$NB$=19 bits.<br>$V_{fs}$ =0.8 V [-0.4, 0.4].<br>$G_{INS\_AMP}$ = 5.<br>$G_{INTEG}$ = −2.<br>Reset time = 10 ms. | Transient Simulation from 0 to 200 ms.<br>Minimum time step: $H_{min}$ =1 fs.<br>Maximum time step: $H_{max}$ =2 us.<br>Integration method = Trapezoidal.<br>Post-processing `MEAS` directive from 10 ms<br>to 190 ms: (`AVG, RMS, MIN, MAX, P2P, CRESTF`). |

Table 4.12: CMU VP default PMPs and simulation conditions. The most critical parameters for the current estimation are the $NB$ and $F_{ADC}$. The comparative study presented in this subsection is focused on the variation of these parameters. Simulation tool: SMASH 6.5.

Once the reset signal (`Rst`) is deactivated at 10 ms, the SMASH post-processing `.MEAS` directive starts to calculate the average (`AVG`) value, the root mean square (RMS) value, the minimum

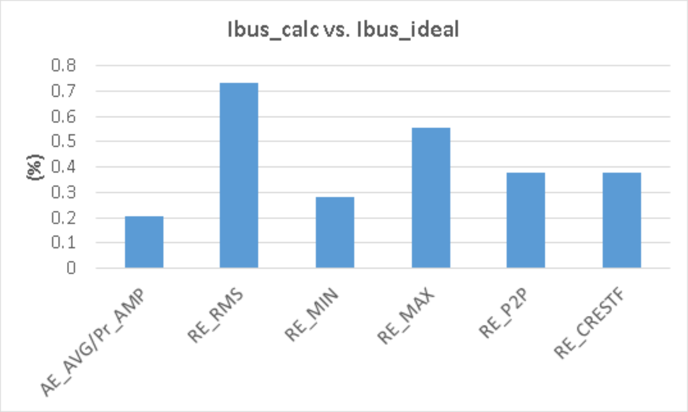


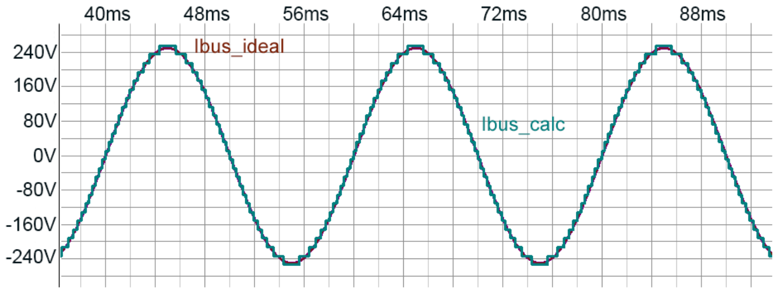
Figure 4.54: Maximum estimated errors between the `Ibus_calc` and the `Ibus_ideal` output signals of the CMU VP using the default conditions shown in Table 4.12; where `AE_AVG` is the absolute error of the average value, `Pr_AMP` is the amplitude of the primary, and `RE_...` stands for the relative error of the respective value. All the maximum errors are produced by the configurations of the Digital CMU architecture.

(`MIN`) value, the maximum (`MAX`) value, the peak-to-peak (`P2P`) value, and the crest factor (`CRESTF`) of the output signals of the CMU. During the comparative study, changing the PMPs values by using different configurations allow us to observe their effect on the estimated `Ibus_calc`. Figure 4.54 shows the default maximum error values (after simulating all the considered CMU configurations), which are calculated using the main quantitative output waveform characteristics of the CMU given by the `.MEAS` directive. All the maximum errors are produced by the configurations of the digital CMU architecture. However, it is required a deeper understanding of the factors that affect the current estimation in each architecture in order to obtain strong evidence for conclusions. This is why the analysis starts by varying some critical parameters in both the Analog and Digital CMU architectures.

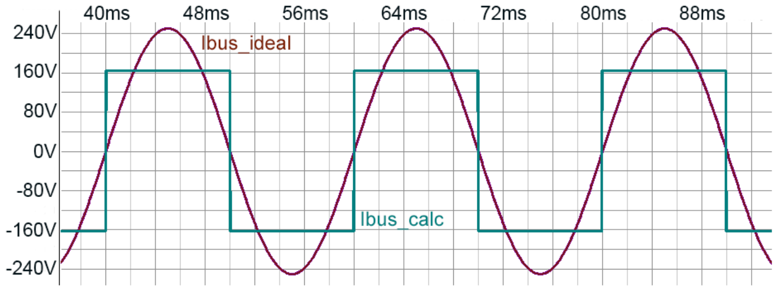
Figure 4.55 exhibits the effect of decreasing the ADC dynamic range (DNR), i.e low $NB$. The Analog CMU VP is simulated by using $NB = 12$ bits, see Figure 4.55(a), where the quantization noise of the ADC is observed but the CMU current estimation works as is expected. On the other hand, the current estimation fails at a reduced ADC precision of $NB = 8$ bits as it is shown in Figure 4.55(b). Strangely, there are only 2 values in the whole current range.

The reason for not being able to reconstruct the current at 8 bits is understood observing the output voltage of the Analog Integration block (`Vout_INT`), see Figure 4.56. A reduced $NB$ causes a least significant bit (LSB) of 3.125 mV; so that, the ADC can only distinguish 2 different values for an input signal of 4.8 mV$_{p2p}$. Although $G_{INTEG}$ is by default -2, the analog sinusoidal signal integration suffers an intrinsic attenuation proportional to $2\pi f_{pr}$. Therefore, limited ADC resolution issues can be solved by increasing the integrator gain or reducing the full-scale voltage of the ADC.

Performing the same analysis with the Digital CMU architecture produces different results. In



(a) NB=12 Bits / DNR = 72 dB



(b) NB=8 Bits / DNR = 48 dB

| **Resolution (NB)** | **12 bits** | **8 bits** |
|---|---|---|
| `RE_RMS` (%) | 0.43 | 9.89 |
| `RE_P2P` (%) | 2.07 | 34.43 |
| `RE_CRESTF` (%) | 1.62 | 27.24 |

Figure 4.55: Analog CMU VP transient simulations at reduced ADC dynamic range (DNR). These results are valid for all the Analog CMU configurations. A comparison between the most important relative errors can be observed in the included table.

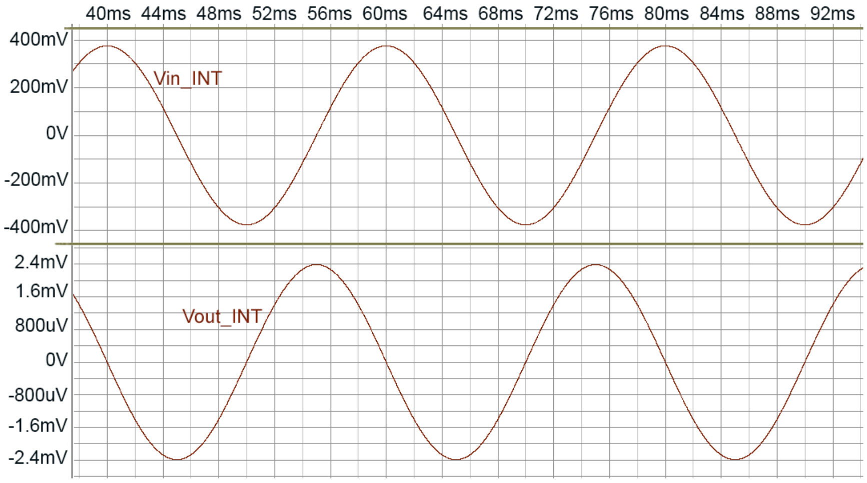
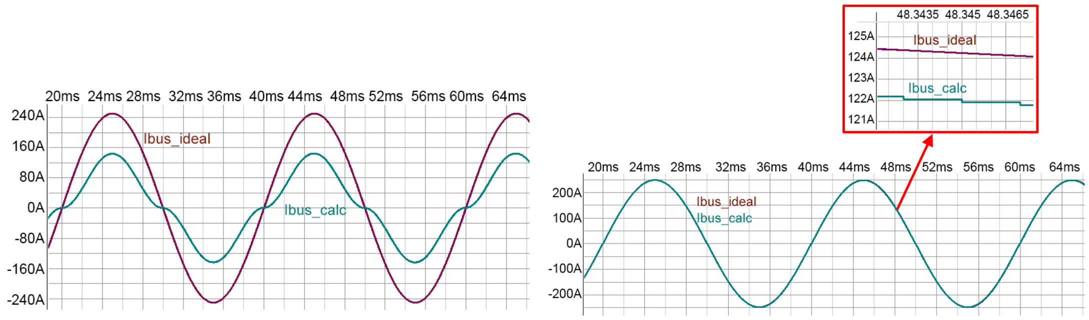


Figure 4.56: Input (`Vin_INT`) and output (`Vout_INT`) voltages of the Analog Integration block. The analog sinusoidal signal integration suffers an intrinsic attenuation proportional to $2\pi f_{pr}$.

fact, using $NB = 12$ bits the obtained current estimation is a highly attenuated and distorted signal, see Figure 4.57(a). The situation is even worst at $NB = 8$ bits. Comparing to the simulation result using the default $NB = 19$ bits, see Figure 4.57(b), we observe a successful primary current estimation with no apparent issues. However, a zoom around 48.34 ms allows to see that these signals are not exactly equal; consequently, we can conclude that the signal distortion caused by the ADC resolution is always present. The relative errors for these two cases are shown in Figure 4.57



(a) NB=12 Bits / DNR = 72 dB

(b) NB=19 Bits / DNR = 114 dB

| **Resolution (NB)** | **19 bits** | **12 bits** |
|---|---|---|
| RE_RMS (%) | 0.63 | 49.28 |
| RE_P2P (%) | 0.31 | 41.96 |
| RE_CRESTF (%) | 0.32 | 14.43 |

Figure 4.57: Effect of the ADC resolution on the current estimation of the Digital CMU VP architecture (`functional` and `software` configurations). The current estimation of the Digital CMU architecture is more sensitive to lower values of $NB$ than its Analog counterpart.

We can further analyze the origin of the signal attenuation and distortion by observing the input (`ADCout`) and the integration result (`Yout`) signals of the Digital Integration block, see Figure 4.58. Albeit there is no particular difference in the `ADCout` signal for $NB = 19$ or $NB = 12$ bits, a lower ADC resolution causes that the digital integration algorithm (in all 3 methods)

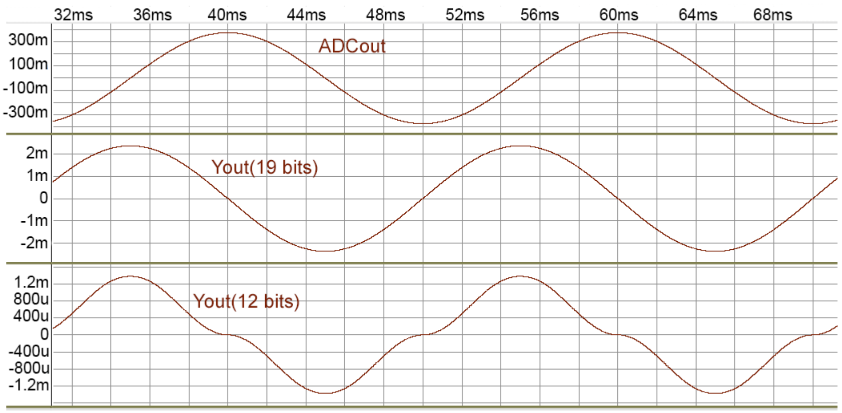
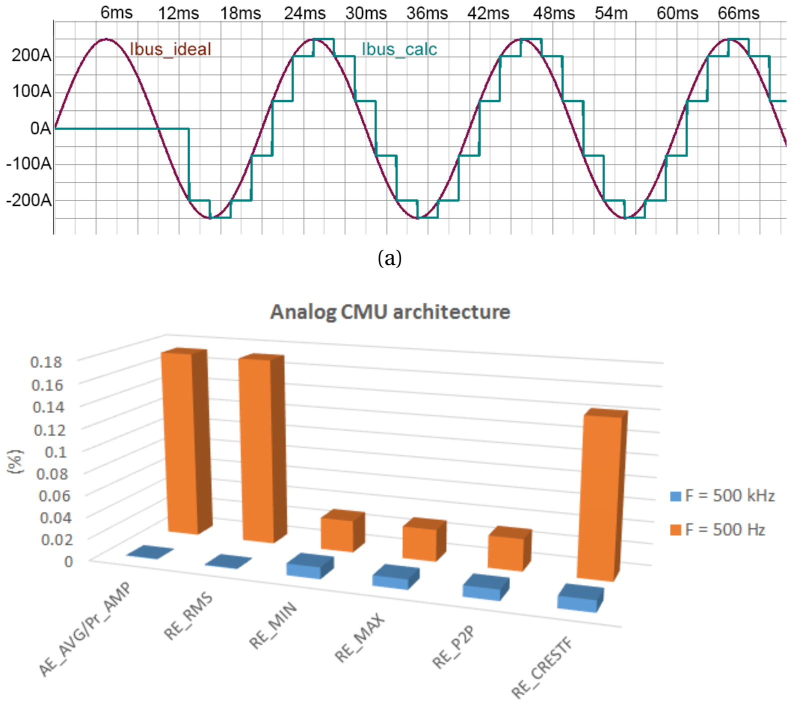


Figure 4.58: Input (`ADCout`) and output (`Yout`) signals of the Digital Integration block. The analog sinusoidal signal integration suffers an intrinsic attenuation proportional to $2\pi f_{pr}$.

presents a significant distortion near zero and a lower amplitude (about 2.4 mV for the correct `Yout` signal in this specific verification case). Consequently, the RMS, P2P, and CRESTF relative errors are largely affected.

Following the comparative architecture study of the CMU, we can make a corner case analysis using the ADC clock frequency ($F_{ADC}$). In fact, the default value has been set at a very high frequency; thus, we are now interested in observing the effect of a low $F_{ADC}$ in the current estimation. Considering the Analog CMU architecture, a large quantization noise in `Ibus_calc` is observed at $F_{ADC}$=500 Hz, see Figure 4.59(a). However, none of the quantitative error estimators evidences the bad situation. In fact, `AE_AVG/Pr_AMP` and `RE_RMS` are the greatest obtained errors with a value around 0.17%, which is lower than the errors obtained for the digital CMU configurations with the default PMPs values, see Figure 4.54. However, comparing to the errors obtained at $F_{ADC}$=500 kHz, `AE_AVG/Pr_AMP`, `RE_RMS` and `RE_CRESTF` are 3 orders of magnitude greater for the low-frequency case. This is mainly caused due to the quantization noise directly affects the symmetry of the signal after the reset signal is activated, i.e. at 10 ms. Therefore, the average and RMS measured values are greatly affected. A fast Fourier transform (FFT) analysis will give more detailed information about the quantization noise in `Ibus_calc`.

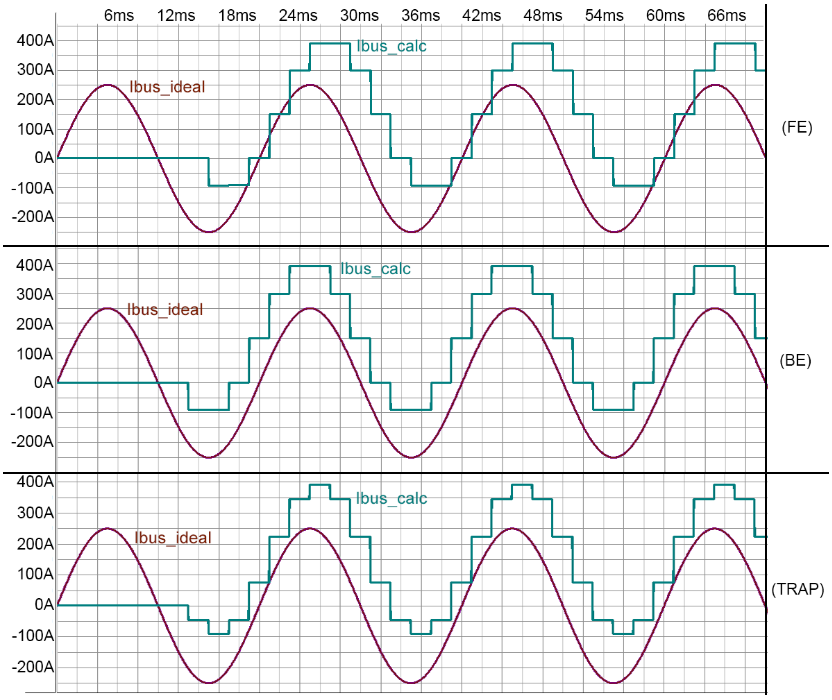


(a)

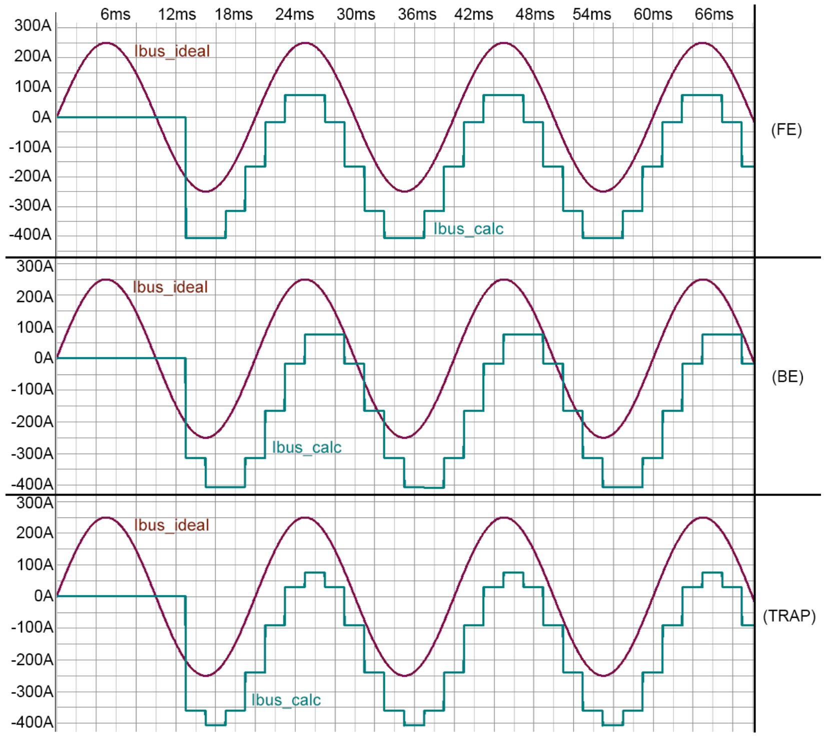
(b)

Figure 4.59: Analog CMU VP simulation results at two corner ADC frequencies. These results are valid for all the Analog CMU configurations. (a) Transient simulation at $F_{ADC}$ = 500 Hz. (b) Bar diagram of the relative errors of the CMU output signals at two corner ADC frequencies.

On the other hand, Figure 4.60 presents the effect of the low $F_{ADC}$ value on the `functional` and `software` CMU configurations of the Digital architecture for all the considered integra-



(a) Functional CMU configuration



(b) Software CMU configuration

Figure 4.60: Transient simulations at $F_{ADC}$ = 500 Hz for all the integration methods of the Digital CMU VP. It can be observed that the `Ibus_calc` signal suffers a DC estimation error that is approximately equal in magnitude for both the `functional` and `software` CMU configurations. The `RE_P2P` for all cases is 3.285%.

tion methods. We observe that in addition to the quantization error, the estimated current has a DC component that is positive for the `functional` CMU configuration in Figure 4.60(a), and negative for the software CMU configuration in Figure 4.60(b). By performing these simulations with higher values of $F_{ADC}$ we can observe that the DC error is inversely proportional to the quantization error. For ADC frequencies higher than 30 kHz the DC error is practically imperceptible.

The positive or negative DC error difference between the two Digital CMU configurations is a matter of algorithm implementation that is not relevant for the analysis. In fact, it can be observed in the simulations shown in Figure 4.60 that the DC error looks like constant. So, the important question is what is the origin of such DC shift?. In order to determine the reasons, let us consider the same simulation using the Hardware CMU configuration shown in Figure 4.61. It can be observed that in this case there is no DC error, only the quantization error equivalent to the FE integration method as expected. Further simulation analysis with different $I_{pr}$ initial conditions and reset times[10] allowed, on the one hand, to find the cause of the DC shift and, on the other hand, to detect a design error that was initially ignored and has to be corrected in both the Analog and Digital architectures. The evaluation of different architectures and implementation algorithms in the same VP is quite useful to understand unexpected results and discover design errors early in advance.

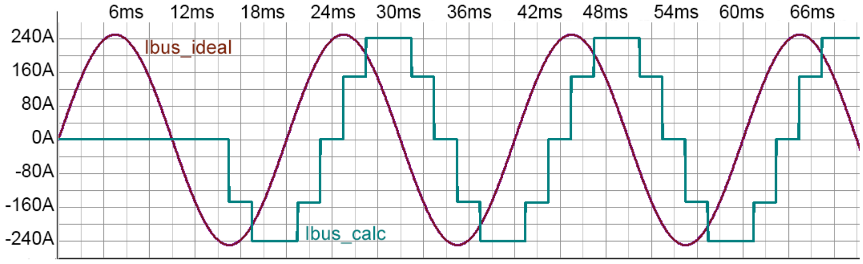


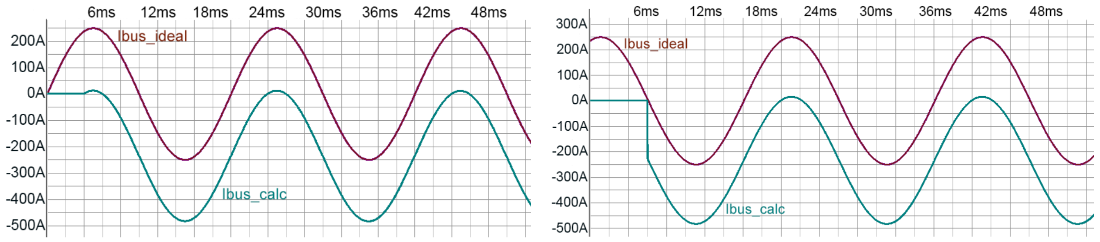
Figure 4.61: Transient simulation at $F_{ADC}$ = 500 Hz for the Digital CMU VP architecture (Hardware CMU configuration).

As it is shown in Figure 4.62(a), the DC error amplitude of the `Ibus_calc` signal varies with the Reset signal deactivation time for the `functional` and `software` CMU configurations of the Digital architecture. Contrarily, the equations of the Digital Integration block in the hardware CMU configuration are active from time 0 in simulation, instead of right after the Reset signal deactivation, as it must be if the block is made synthesizable. Similarly, the Analog Integration block in the Analog CMU architecture is active from time 0 in simulation. In these two later cases, no `Ibus_calc` DC error depending on the Reset signal deactivation time is observed.

On the other hand, Figure 4.62(b) reveals an `Ibus_calc` DC error in the Analog CMU architecture when the $I_{pr}$ signal has an initial phase ($\theta$ =70°=$\frac{7\pi}{18}$ rad) where the magnitude of the primary current is different than zero, i.e. $\theta = n\pi \ \forall \ n = 0, 1, 2, \ldots$ This error can also be obtained using the hardware CMU configuration of the Digital CMU architecture. Thus, it can

---

[10] The simulations were performed using all the presented CMU configurations.



(a) $\theta$ =0°, Reset time = 4 ms.

(b) $\theta$ =70°, Reset time = 6 ms.

Figure 4.62: CMU VP transient simulations showing a DC shift in the primary current estimation. Table 4.12 shows the default simulation conditions. (a) Digital CMU architecture (`functional` and `software` CMU configurations). (b) Analog CMU architecture (all configurations).

be deduced that the DC error is caused by the delay of the input signal on the integration block. We can see in the simulations shown in Figure 4.60 that the ADC at low frequency causes an `Ibus_calc` estimation delay larger than 2 ms. If this delay matches to the time in which the primary signal is zero, there is no DC error in `Ibus_calc`.

In order to understand the design issue that causes the DC error on the `Ibus_calc` signal, we need to re-examine the mathematical expressions used for the primary current estimation. The Equation 4.21 on page 105 tell us that $I_{pr}$ can be obtained by integrating the output voltage of the RogoCoil ($V_{out}$). However, this is a truncated integral from time 0 to time $t$, instead of an indefinite integral as it was initially assumed. Therefore, the correct expression is as follows:

$$I_{pr}(t) = \frac{N}{G \cdot L_{coil}} \int_0^t V_{out}(\tau)d\tau = CF \int_0^t V_{out}(\tau)d\tau \tag{4.28}$$

Considering Equation 4.19 on page 105, and assuming a sinusoidal primary current of amplitude $A$ and phase $\theta$, the resulted $V_{out}$ waveform is the following:

$$I_{pr}(t) = A\sin(\omega t + \theta) \Rightarrow \tag{4.29a}$$

$$V_{out}(t) = G \cdot M \cdot \frac{dI_{pr}}{dt} = G \cdot M \cdot A\omega\cos(\omega t + \theta) \tag{4.29b}$$

$$V_{out}(t) = B\cos(\omega t + \theta) \tag{4.29c}$$

By using Equation 4.29c in Equation 4.28, the estimated $I_{pr}$ waveform is obtained as follows:

$$I_{pr}(t) = CF \int_0^t B\cos(\omega\tau + \theta)d\tau \tag{4.30a}$$

$$I_{pr}(t) = CF \cdot \frac{B}{\omega}\sin(\omega\tau + \theta)|_0^t = A \cdot \frac{N}{L_{coil}} \cdot M\left[\sin(\omega t + \theta) - \sin(\theta)\right] \tag{4.30b}$$

$$I_{pr}(t) = A\sin(\omega t + \theta) - A\sin(\theta) \tag{4.30c}$$

Thus, the term $A\sin(\theta)$ in Equation 4.30c evidences the DC error on the `Ibus_calc` signal that we have been observing. This error is proportional to the amplitude of the primary current and occurs when $\sin(\theta)$ is different than zero, i.e. $\theta = n\pi \ \forall \ n = 0, 1, 2, \ldots$ A delay in the input signal of the integration block is equivalent to have a sinusoidal input signal with a certain phase $\theta$. This issue can be solved by different forms depending on the CMU architecture. For example, an HPF can be implemented after the Analog Integration block in order to remove any DC component caused by $\theta$ in the Analog CMU architecture. On the other hand, in the Digital CMU architecture, the effective Reset (or Enable) signal can be triggered by a $V_{out}$ zero crossing detector. In this case, the CMU ensures that the digital integration algorithm will start when $\sin(\theta)$ is zero. Nonetheless, both CMU architectures can use the $A\sin(\theta)$ term on the `Ibus_calc` signal to calculate the initial phase of the primary current on a three-phase system, this could be useful when unbalanced loads cause line currents with a phase angle different than 120°. In this case, the DC error must be removed by digital processing after calculating its maximum and minimum values.

The conclusions of the simulation results performed so far are listed in Table 4.13, where most of the points indicate that the Analog CMU architecture seems to have an advantage over the Digital solution. Of course, a complete comparative study must evaluate more critical parameters of the main components of the system using more refined *behavioral models*. Although high-level *functional models* are useful for obtaining initial insights of the *design space exploration* assessment, these models cannot be used for a final system verification due to they do not take into account some non-idealities of the components that can be critical for the correct operation of the system. For example, if we only consider the first two Analog CMU configurations for executing further verification of the system behavior, we are omitting a critical non-ideality of the differential amplifiers, which will become a design error if it is not properly addressed. Figure 4.63 shows that an output offset voltage of just 1 $\mu$V (which is actually too optimistic for current commercial components), produces a DC voltage at the output of the Analog Integration block (`Vout_INT` signal) of about -100 mV (without considering the offset voltage of the OpAmp). This cause a huge DC error on the `Ibus_calc` signal after the Digital Calculation block is activated at 10 ms. Consequently,

| **Analog** | **Digital** |
|---|---|
| ☺ Better accuracy at default parameter values. | ☹ Lower accuracy at default parameter values. |
| ☺ Low `RE_P2P` at low ADC frequency (0.003%) | ☹ High `RE_P2P` at low ADC frequency (3.285%) |
| ☺ Better accuracy at low ADC resolution (12 bits). | ☹ Huge signal distortion at low ADC resolution (12 bits). The accuracy cannot be improved by the integrator gain. |
| ☹ Initial phase detection implies additional circuitry. | ☺ The initial phase detection can be handled entirely in Digital, no additional circuitry is required. |

Table 4.13: Preliminary strengths and weaknesses of the Analog and Digital CMU architectures.

the implementation of an HPF (analog or digital) for the Analog CMU architecture becomes mandatory. On the other hand, the Digital CMU architecture is not affected by the `INS_AMP` output offset voltage value in this verification case, since the ADC resolution ($0.8V/2^{19} = 1.526\mu$V) is greater than the offset voltage. Consequently, the Digital CMU architecture brings certain advantages for the possible analog signal deviations and noise despite its apparent lower accuracy in comparison to the Analog CMU architecture.

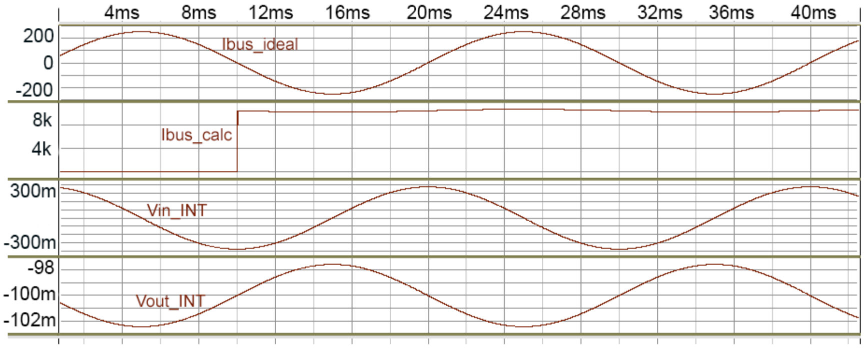


Figure 4.63: Transient simulation of the Analog CMU architecture (Behavioral_refined CMU configuration). `INS_AMP` output offset voltage equal to 1 $\mu$V.

In order to continue with a further CMU verification analysis, i.e. towards the design enquiries related to the functional safety topic mentioned in Table 4.9 on page 108, a more refined *model implementation* of the ADC (e.g. the $\Sigma\Delta$ ADC topology) is recommended to be included in a CMU configuration, together with the implementation of the solution for the initial phase issue.

## 4.5 The Virtual prototype of the Rogowski coil sensor system

During the three previous sections of this chapter, we have observed how the *virtual prototyping* activity is conducted on the three main subsystems of the RogoCoil sensor system introduced in Figure 4.2 on page 55. The apparent higher tractability of a system by a separate analysis of its subsystems makes the design prone to multiple issues such as interface design errors, increased complexity for the virtual analysis, and sub-optimal system design. This is why the virtual design and verification approach proposed in chapter 5 encourage an initial virtual analysis of the complete system before dividing the system into smaller parts for a more detailed analysis. The VP-based design methodology plays an important role in this initial system-level analysis. Despite the RogoCoil main subsystems were analyzed and designed individually, this section exemplifies how the modeling techniques advocated by the VP-Modeling Guidelines helps to improve a large VP of a complete system. This is essential for system-level modeling of CPS.

The natural elaboration of the complete Rogowski coil sensor system (RCSS) VP starts by including the models previously developed for each subsystem, for instance, including the

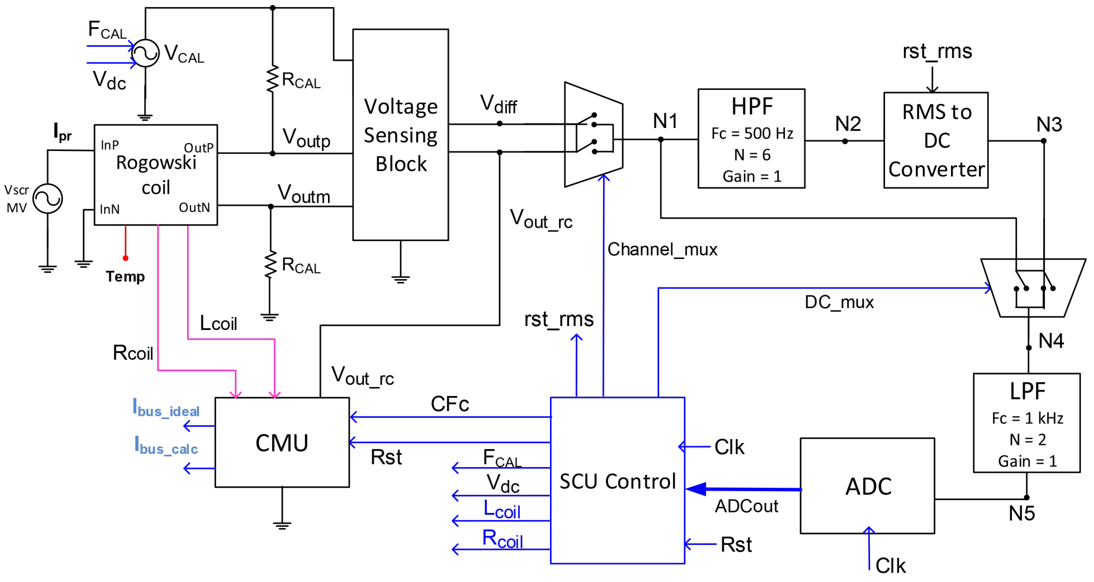


Figure 4.64: RCSS VP complete architecture block diagram. This is a large hierarchical VP in terms of component parameters and interconnections. The execution capabilities of the RCSS VP are mainly transient and parametrical simulations.

CMU model in the SCU VP described in Figure 4.26 on page 87. The complete system shown in Figure 4.64 reveals in detail how the CMU shall be connected to the components of the SCU. The SCU Control, which is in charge of the $R_{coil}$ and $L_{coil}$ estimation, calculates the correction factor `CFc` according to Equation 4.24 explained in subsection 4.4.1. The `Ibus_ideal` output signal of the CMU is calculated by a reference calculation block which uses directly the $R_{coil}$ and $L_{coil}$ values given by the RogoCoil model to obtain the primary current as it is shown in Equation 4.25 on page 107. On the other hand, the `Ibus_calc` signal is the result of the implementation of the same equation but using the estimated `CFc` from the SCU. In this case, the output voltage of the RogoCoil ($V_{out\_rc} = V_{outp} - V_{outm}$) is measured by the Voltage Sensing Block and transferred to both the SCU and the CMU as it is shown in Figure 4.64. Despite the straightforward construction of the RCSS VP using the available *component models*, there are two principal issues for its elaboration:

1. The $V_{out\_rc}$ contains a high-frequency and a DC signal components produced by a periodical application of the calibration signal for the self-calibration method. Consequently, the CMU VP design carried out in subsection 4.4.2 cannot be directly applied since it was designed without considering the calibration circuit.

2. The large number of generic parameters and *model implementations* of the *component models* make difficult the elaboration and the manipulation of the complete VP modeling infrastructure.

The first issue is a direct result of a separated subsystem design at the early stages of the development. This is a very common problem in CPS design which is not always easy to solve. In this case, both the DC and the high-frequency components can be easily filtered in the CMU by including a band-pass filter in the signal processing chain before the $V_{out\_rc}$ integration. We can also remove the calibration signal effect in $V_{out\_rc}$ by removing the high and low-frequency signal components separately. The low-pass filtering can be performed by a proper selection of an anti-aliasing filter and a decimation factor in a $\Sigma\Delta$ ADC architecture as it is explained in subsection 4.3.2; so that, it is only required to include and additional digital or analog HPF before the signal integration. This supplementary requirement will make obsolete the initial phase detection mechanism proposed in the individual analysis of the CMU in presented subsection 4.4.2.3. This is why a former virtual analysis of the complete system becomes handy and very convenient. The RCSS VP can perfectly address verification studies for checking the compatibility of different subsystems; therefore, the modularity of the VP and its *component models* is an important quality that must be exploited. However, the modularity of the models goes far beyond to the capability of connecting other blocks in a signal processing chain in the same domain, it is also related to the capability of introducing additional modules for verifying complementary behaviors and interactions such as cross-domain interactions, power consumption, and safety indicators among others. The presented electro-thermal RogoCoil modeling is an example of how to exploit such modularity using a VHDL-AMS approach. The thermal network modules can be added or omitted from the RogoCoil electrical model in a large VP according to the verification needs.

The second issue is one of the main limitations of *virtual prototyping* for CPS in any application field. In particular, the construction of large VHDL-AMS *test benches* and multiple *test cases* for *design space exploration* and critical parameter verification becomes unpractical for the construction from the bottom-up of VPs of this size. At this point of the system design, the VHDL-AMS modeling techniques recommended in the VP-Modeling Guidelines can be used in order to obtain a VP that possesses both high-level *functional models* for their main components and the lower level *model implementations* of their subcomponents that have been currently modeled by the system designer, or are available in the VP-Model Library. In this way, the VP-Modeling Guidelines allows reducing the complexity of the RCSS VP by obtaining a well-organized modeling infrastructure that facilitates the management of generic parameters, *model implementations*, and gradual *model refinement*.

We can start by applying a logical component encapsulation (VP-Modeling Guideline 7) as is shown in Figure 4.65. This is a hierarchical VP composed by only four main blocks interconnected by analog (conservative) signals, digital (discrete-event) signals, and abstract (signal-flow) variables[11], see the signal classification associated with the ports of the *component models* in Appendix F.2.1.2. Subsequently, the DUV, which is conformed by these four blocks, is included in a *test bench* that provides the primary current. The RCSS VP mo-

---

[11] The abstract variables do not model real information flow from the physical system. They are used to exchange information between the models only for analysis purposes, such as the comparison between the result of the system and a reference value.

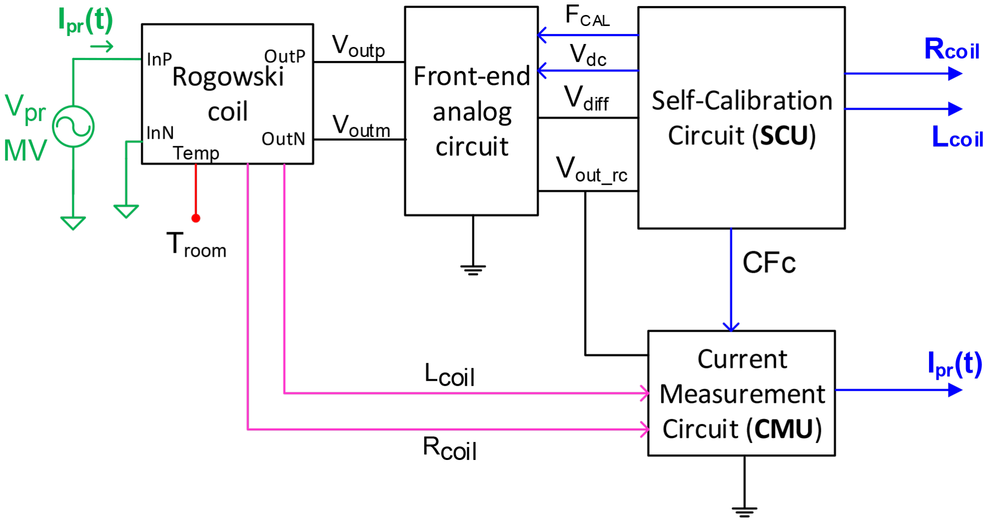


Figure 4.65: High-level block diagram of the RCSS VP after a logical encapsulation of the *component models* of the complete system given in Figure 4.64. The front-end analog circuit block contains the following subcomponents previously included in the SCU: the calibration resistances ($R_{CAL}$), the calibration voltage source ($V_{CAL}$), and the Voltage Sensing Block. The signal interconnection color code stands for: MV analog signals (green), low voltage analog signals (black), digital signals (blue), thermal signals (red), abstract variables (magenta).

deling infrastructure has been built following the VP-Modeling Guideline 6.2; it consists of six parameter packages set from the RCSS VP top-level configurations as is shown in Table

| Package | Description | Components | Config |
|---|---|---|---|
| KEVCR_pkg | KEVCR geometric, material, and characterization parameters | Rogowski coil | DUV |
| KECA_pkg | KECA geometric, material, and characterization parameters | Rogowski coil | DUV |
| Front-end Circuit_pkg | Component parameters of the Front-end analog circuit | Voltage Sensing Block, Calibration source, Calibration resistors | DUV |
| SCU_pkg | SCU PMPs and share functions | Filters, Multiplexers, RMS-to-DC converter, ADC, digital control block | SCU |
| CMU_pkg | CMU PMPs and share functions | INS_AMP, ADC, Digital/Analog Integration block | CMU |
| RCSS_TB_pkg | External DUV verification parameters: primary signal, harmonics, temperature conditions, Start/Stop control signals | Primary current source, Temperature source, etc. | TBC |

Table 4.14: RCSS VP packages and main configurations (**Config**) of the VHDL-AMS modeling infrastructure. The SCU and CMU configurations are called from the DUV configurations, which in turn are called from the TBCs.

4.14. Individual configurations are used for setting the *component models* and the parameter packages of the SCU and the CMU. Likewise, these configurations are called from the DUV configurations, which are in charge of setting the parameter package that defines the type of RogoCoil used in simulation (KEVCR or KECA coils), and the parameter package of the Front-end analog circuit components. Finally, the highest level TBCs call the DUV configurations and set *test bench* utility *component models* and their parameters for defining the DUV stimuli. The primary current and/or temperature signals can be easily defined in the RCSS VP *test bench* by equations. More complicated behaviors for specific verification cases are recommended to be modeled in individual design entities and instantiated in the *test bench.* Each particular configuration combination together with their *simulator setup* forms a *test case.*

The restructured RCSS VP is now a simplified hierarchical version of the VP depicted in Figure 4.64 and is ready for gradual *model refinement* and virtual verification at the system level such as the analysis presented in subsection 4.4.2.3. The effective execution of virtual verification studies such as the ones shown in Table 4.9 on page 108, strongly depends on the VP manipulation capability offered by the VP modeling infrastructure.

## 4.6 Conclusions

The *virtual prototyping* studies presented in this chapter using the RogoCoil sensor system as the main example, illustrate the challenges, requirements, and potential of the VP-based methodology. Although the detailed subsystem and component modeling is essential for understanding the system interactions and behaviors, the complete system design can benefit from *virtual prototyping* at the system level. This work exemplifies how the VP-based methodology elements improve the modularity and scalability of VPs and their *component models,* which is key for successful *virtual prototyping* at the system level. Likewise, the gradual *model refinement* concept is illustrated as an essential process for *design space exploration* and system verification.

Each of the modeling and simulation studies presented through the sections of this chapter infers particular conclusions for the RogoCoil sensor system design.

**Electro-thermal modeling of the RogoCoil:** It is shown how to exploit the multi-domain capabilities of VHDL-AMS together with geometrical FEA to create high-level parametric abstractions. The proposed electro-thermal model can estimate dynamically the internal temperature of the coil and its dependency on geometrical, electrical and thermal parameters of the system. Furthermore, the model is able to simulate electrical variable fluctuations caused by room temperature drifts. This is very important for a better understanding of the implication of the temperature effects in the signal processing electronics of the sensor.

**The SCU:** The SCU VP allows to observe the limitation of the SCU approximation for obtaining an acceptable $L_{coil}$ estimation accuracy using a calibration frequency closer to the first resonance frequency. Moreover, this VP has proven to be effective for detecting and understanding an important limitation for estimating $L_{coil}$ for coils with a high $R_{coil}$ value and low $L_{coil}$ value. This situation was initially thought that can be solved by a large RogoCoil output voltage amplification. However, as is shown in subsection 4.3.1, it is demonstrated that this limitation cannot be corrected by signal amplification due to is not possible to amplify only the voltage drop caused by $L_{coil}$. As long as $R_{coil}$ is big, this issue cannot be corrected by reducing $R_{CAL}$ value neither. This issue was later confirmed by experimental measurements with one of the ABB RogoCoil types (the KEVCD B family sensors) that present this characteristic, i.e. $R_{coil} \approx 5$ kΩ and $L_{coil} \approx 0.2$ mH.

**The CMU:** Contrarily to the SCU VP, the CMU VP was elaborated following the VP-Modeling Guidelines and reusing important *component models* previously designed for the SCU. A significant reduction of the modeling effort and the improvement of the VP manipulation were observed. The CMU VP shows a particular case of *design space exploration* by using a single *test bench* and several *test cases* built with a hierarchical structure using configurations. This example illustrates how to create a virtual environment to make system-level functional verification, performance estimation, and critical design parameters identification. One of the benefits of the approach is observed in the detection of a design error caused by the initial condition of the input signal. Although it is not presented in the given example, the VP-based methodology suggests that the CMU VP can be used for more detailed system verification studies by refining the *component model* in a gradual and modular way.

**The RogoCoil sensor system:** Particular integration and modeling issues of the RCSS are presented and briefly discussed. The VP-Modeling Guidelines are used to obtain an improved VP of the complete system which facilitates its maintenance and manipulation. Some of the benefits of the VP modularity and limitations of the VP bottom-up scalability are discussed.

# 5 The circular system development model

In this chapter, the author of this thesis proposes the circular system development model (O-model), which uses some concepts of classical software and hardware development models such as the Waterfall and the V-model [97]. The development of the *Rogowski coil* sensor system introduced in chapter 4 is presented here in order to illustrate this approach. Additionally, it is explained where and how the proposed VP-based design methodology presented in chapter 3 can be applied in the system development lifecycle.

## 5.1 Abstract

The permanent increasing on the complexity of new multi-domain systems of systems (CPS) makes technology-leader companies struggle to guarantee general requirements of their products such as high precision, high reliability, intrinsic safety, low product development costs, and low power consumption among others. To comply with a list of mutually exclusive requirements is necessary to update/re-define classical methods and approaches for system development.

This chapter describes the O-model, which implements the VP-based design methodology as the vehicle to perform *design space exploration* and system verification at the former stages of the design. Although the O-model can be applied in general to CPS development of any scale, the *simulation framework* and proposed modeling techniques are focused on the design of small-scale CPS in the power automation domain, such as smart electrical sensors and circuit breakers for high and medium voltage applications. The *Rogowski coil* sensor system presented, modeled, and simulated in chapter 4, is used as the specific example to illustrate the *virtual prototyping* tasks of the O-model. Therefore, in spite of the fact that the O-model considers all the stages of an industrial system development life-cycle, this research is only focused on the first stages of the system development process, i.e. the design and verification stages. The utilization of *virtual prototyping* at the early stages of the system development contributes to the reduction of the design and verification effort by allowing architectural design space exploration, early error detection, and complete system optimization.

## 5.2 Introduction

In order to overcome the issues presented by the traditional system design and development approaches such as the V-model, shown in Figure 1.1 on page 2, it is proposed the O-model, which uses the VP-based design methodology as the backbone of the approach. The VP-based design methodology can be used to reduce the design and verification effort by allowing 'virtual integration' and 'virtual testing', especially at the earlier stages of the design. The idea is to include a system-level design procedure for exploring and analyzing the complete system architecture before separating the design and verification in subsystems. The virtual integration feature is achieved by functional system-level modeling and simulation, which allows performing architectural design space exploration and complete system optimization. It is not the same to optimize a system as a whole than optimize individually the components of the system. The virtual testing feature is possible by functional verification of VPs and *model refinement* techniques.

Ideally, iterative modeling and simulation should be applied until obtaining complete and clear specifications of each component of the system. However, there are several limitations that make very difficult or almost impossible to model and simulate all the behaviors and interactions of the components of a CPS. Firstly, there is no universal *simulation framework* capable of addressing all the specific system design needs for any application [38]. Secondly, very refined and large VPs have poor simulation performance, making its simulation not feasible in a reasonable time. Finally, a significant modeling effort and domain expertise are required for modeling detailed subsystems and components. Therefore, the *virtual prototyping* activity is also divided into three stages. In the first stage a VHDL-AMS *simulation framework* is used for system-level design; in the second stage other domain-specific *simulation frameworks* and/or physical prototyping can be used for detailed subsystem design; lastly, a virtual system integration and verification stage is proposed before the elaboration of the *zero series* (0-Series) system.

One of the most important concepts for a successful implementation of the VP-based design methodology in any industrial design flow is the capability to reuse *component models* or complete VPs for the design of new systems and products. Therefore, the *virtual prototyping* activity proposed in the O-model relies on a rich library of models that can provide reliable and fully documented models. Thus, this work also describes the required tasks and outcome of a model maintenance process which allows reaching this goal using the proposed operational elements of the VP-based design methodology. The final objective of the approach is to decrease product development time and costs.

## 5.3 The O-model

The O-model consist of 8 main stages for the product development lifecycle, which are depicted in the diagram shown in Figure 5.1. The diagram shows the relationship between contiguous stages which are connected in order. The complete lifecycle of any complex system that is developed, installed, and made operative is shown in the O-model diagram; it starts in the system definition stage and finishes in the system maintenance stage. All the information obtained from a system previously developed can be used in a next system development lifecycle for an updated version of the current system or a completely new system. Although computer-based models can be used at any stage of the development process, it is shown in Figure 5.1 where the *virtual prototyping* activity is more useful in the system lifecycle. VPs are very useful at the early design and verification stages, even though they can also support the system development after physical prototype fabrication or product manufacturing processes.

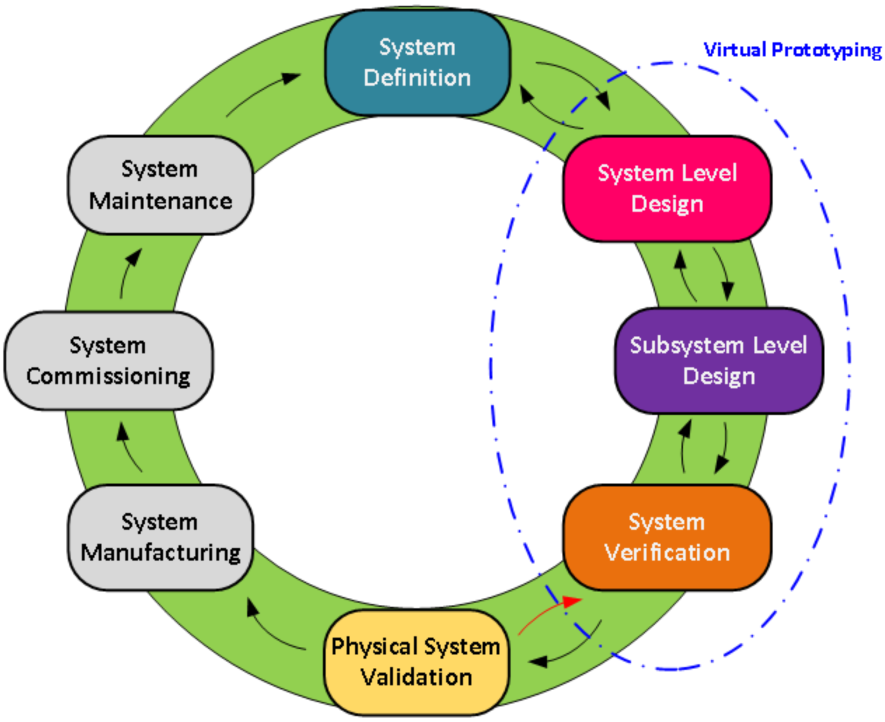


Figure 5.1: Circular system development model (O-model).

A common mistake is to consider the system design as a linear process with a predefined series of steps. This perception oversimplifies the system development and hides the complexity of the design. It is important clarifying that particular issues presented at any stage could make necessary to come back to any previous stage in order to correct the issues. These iterations are only shown between the first contiguous stages in Figure 5.1 since the goal of the proposed approach is to only get iterations between the stages directly related with *virtual prototyping* as early as possible in the development process.

At early stages of the design of complex systems, it is common that the problems with one stage are never solved completely during that stage, and in fact, many problems regarding a particular stage arise after the stage is signed off. The most typical examples are the design issues that are only discovered after the firsts physical prototypes are fabricated and tested. Wrong assumptions, unknown component interactions, and incomplete system specifications are among the most frequent reasons to find design errors in the initial physical prototypes of a complex system. On the other hand, adequate model abstractions and well-designed *test cases* can help to foresee issues before a physical elaboration of the system and their subsystems.

The three later stages shown in the O-model diagram: **system manufacturing**, **system commissioning**, and **system maintenance**, have been classically not considered in traditional product design models such as the V-model, see Figure 1.1 on page 2. Although scientific and technological improvements directly related to the tasks included in these stages are not addressed this research, the main goals and outputs of these stages are of great importance for the development lifecycle of a system seen as a product. Consequently, the concepts and recommendations given by the product lifecycle management (PLM) study are complementary. As it is described in section 2.8 on page 18, PLM claims to take care of the management of the product from its inception until its disposal. In this context, the proposed O-model is a realization of the ideas promoted in PLM.

PLM considers five phases for the product lifecycle [57]: imagination, definition, realization, support/utilization, retire/disposal. However, this is a very simplistic view of the complete lifecycle that does not consider the intrinsic iterations between these main phases. Other similar linear stages from different points of view have been also considered for the product lifecycle; for instance, from the market-oriented point of view, from the manufacturing point of view, or from the environmental point of view. The O-model is one of the first attempts, from the product design point of view, to have a detailed unified product lifecycle. In line with the PLM paradigm, cross-functional effective methods and good documentation exchange, among the O-model stages and their internal processes, shall be encouraged in order to facilitate the development of the product and stimulate the incremental innovation of the product.

The different inputs, outputs, and main tasks of each stage of the O-model have been studied and identified; however, this work only details the first stages of the system development, i.e. the stages included in the *virtual prototyping* phase. Even though it is not explicitly mentioned either in the explanation or in the diagrams of this section, before passing from any development stage to the next one, is required to make a proper documentation of the activities and results obtained in that specific development stage. Clear and organized documentation is key for the success of the complete development process.

### 5.3.1 System definition

Every system development cycle starts by a process called system definition, which consists of the first analysis of the problem that requires a solution or a new idea with a potential market opportunity, i.e. a product that can be commercialized. Disregarding of the type of innovation that could arise, i.e. disruptive or incremental innovation, it is assumed that the considered solution involves the development of a small-scale CPS for power automation applications. Figure 5.2 shows the principal input data of this process, they are explained as follows:

- The **problem definition** requires a clear information of the issue that needs to be solved, the description of the environment and the conditions. A well-defined problem contains the essential actions and functionalities that the required system must perform.
- A **market analysis** is a quantitative and qualitative assessment of the potential market in which the system will be commercialized [98]. It looks into the size of the market both in volume and in value, the various customer segments and buying patterns, the competitors, and the economic environment in terms of barriers to entry and regulation.
- At this level, all the **available technologies** that are related to the central functionality of the system should be considered. Commonly, in the development of new systems, we must consider emergent technologies that offer new solutions and/or advantages against conventional technologies. A comparison matrix between the main advantages and disadvantages of all possible technology solutions must be given. Since most of this information might be unclear or incomplete, we can let this analysis to be performed in the next stages of the development process in order to choose the best option.

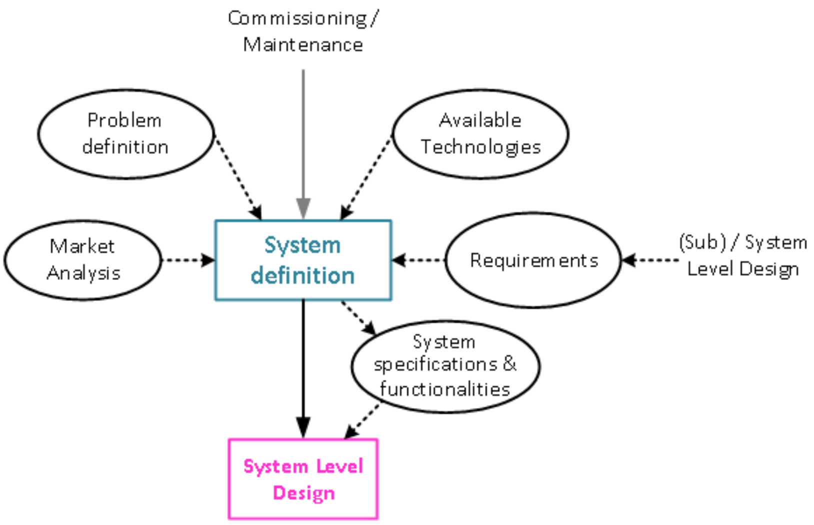


Figure 5.2: System definition diagram. The dotted arrows indicate data flow (input and output data). The solid arrows indicate process transitions.

- The **requirements** of the system ought to be understood as the set of qualitative conditions that the system must comply in order to be accepted. The most common general requirements for industrial CPS design are:

    - Low costs.
    - Low power consumption.
    - High functional reliability.
    - High precision.
    - Intrinsic safety.
    - Long operation life.
    - Strong electro-magnetic compatibility (EMC).
    - Industrial temperature range.

  Typically, the more general requirements, the larger the design effort and the stronger the required know-how within a company. Some combinations of general requirements —which are mutually exclusive —drastically increase the design effort; for instance, low cost and high reliability, or low power consumption and high precision.

Likewise, the information obtained in the commissioning or maintenance stages of a previous system development lifecycle can be used as an input to the system definition process. In this stage, the objective is to evaluate all the input data to generate the ideas for solving the required problem. The sought solution implies the development of a system able to execute the required functionalities under a set of requirements and constraints.

The output of the system definition is a set of system specifications and functionalities as we can observe in Figure 5.2. The system specifications at this level are the result of a preliminary analysis of the algorithm and/or the identification of the main subsystems, probably a first idea of the system architecture and its potential to solve the system requirements. However, since the functionality and behaviors have not been explored and verified, the set of system specifications are not well-defined at this stage, i.e. there might be unclear or missing specifications.

#### 5.3.1.1 The RogoCoil sensor system example

**Problem definition:** In the last years, the increasing emergence of decentralized power supply leads to a destabilization of the power grid network, producing undesired grid losses, and difficulting the handling of power quality. This issue can be addressed by monitoring and controlling the network conditions on relevant points in the MV grid. This raises the need of developing smart voltage and current sensors able to provide a high measurement accuracy (i.e. accuracy class 0.5 or better [99]), remote signal controlling and monitoring, simple installation in substations, and reduced costs.

**Available technologies** There are a wide range of technologies for DC and AC current measurement [77]: Hall sensors [100], anisotropic magneto-resistor (AMR) current sensors [101], current shunt ammeter [102], current comparators [103], CMOS current sensor [104], magneto-optical current sensor [105], SQUID current sensors [106], giant magneto-impedance (GMI) current sensors [107], current transformers (CTs) [77], and RogoCoil transducers [76]. Taking into account the characteristics of the application field, electrical MV power grid, we must only consider AC current sensor technologies which offers the best advantages for measurements at low frequencies (i.e 50/60 Hz) in the range of tens to kiloamps. Therefore, we shall only consider for our *technology space* CTs, RogoCoils, and magneto-optical current sensors.

Magneto-optical current sensors are based on the Faraday effect [108], either in bulk material or in an optical fiber. The Magneto-optical point sensors use a piece of glass or a crystal rod placed in the neighborhood of the electrical conductor. These devices are robust, cheap, and sensitive. They belong to the class of 'extrinsic fiber sensors', i.e. sensors which use optical fibers for transmission, not for sensing [109]. On the other hand, in intrinsic fiber sensors, the magneto-optical material encloses the electrical conductor, and thus these sensors are not sensitive to external currents and magnetic fields [77]. Back light propagation is used in magneto-optical sensors to compensate birefringence. In this approach, the light wave is reflected on the far end and its polarization state is rotated by 90°. Then, it is coupled back into the fiber [110]. A sensor of this type, made of low-birefringent flint fiber with a very low photoelastic constant, has been reported to achieve 0.1% accuracy class around 1 kA [111].

Magneto-optical current sensors have several advantages which are very attractive for power distribution applications [77]:

- Effective isolation from high potentials.
- Immunity against electromagnetic interferences.
- High dynamic range, no saturation effects.
- High bandwidth.
- Compact and lightweight design.

These features offer a significant cost reduction in comparison to conventional HV CTs. However, since the Faraday effect for most materials is extremely small, the magnetic field strength must be considerably high to be detectable. Therefore, magneto-optical current sensors are ideally suited for HV applications rather than MV applications.

The standard technology that has been used during decades for electrical measuring and protection in MV and HV applications, is the current transformer (CT) technology. The CT is a type of *instrument transformer* that is designed to produce an alternating current in its secondary winding which is proportional to the current being measured in its primary. Unlike a conventional voltage transformer, the CT consists of only one or very few turns at its primary winding. This primary winding can be of either a single flat turn, a coil of heavy duty wire wrapped around the core or just a conductor or busbar placed through a central hole. The secondary winding instead, has a larger number of turns wounded on a laminated core of ferromagnetic material. This core has a large cross-sectional area so that the magnetic flux density created is low using much smaller cross-sectional area wire for the turns, it depends upon how much the current must be reduced, independent of the connected load. There are three basic types of CTs as follows [112]:

**Wound** The CT primary winding is physically connected in series with the conductor that carries the measured current flowing in the circuit. The magnitude of the secondary current is dependent on the turns ratio of the transformer.

**Toroidal** This type does not contain a primary winding. Instead, the line that carries the current flowing in the network is threaded through a window or hole in the toroidal transformer.

**Bar** This type of CTs uses the actual cable or busbar of the main circuit as the primary winding, which is equivalent to a single turn. They are fully insulated from the high operating voltage of the system and are usually bolted to the current carrying device.

CTs are very popular devices, they are very simple and robust, they do not require external power, they have high galvanic insulation, they are cheap and they have long lifetime [77]. However, CTs must be periodically calibrated.

On the other hand, as it is presented in section 4.1, the RogoCoil current transducer, which possesses all the previously mentioned benefits of the CTs, is a slightly different device than toroidal CTs. The main difference is that the RogoCoil does not have a ferromagnetic core; and therefore, this type of sensor has many advantages over conventional CTs [76, 113]: it has excellent linearity and an extremely large dynamic range thanks to its air-core, it is open-ended, easy to install in substation and overhead conductors, and much more economic than standard CTs.

**Requirements** The required smart current sensor for MV applications must comply with the following general requirements:

- Low materials and manufacturing costs.
- High functional reliability for measurement purposes; even for use in harsh environmental conditions such as temperature drifts, condensation, EMC.
- Intrinsic safety, i.e. isolated from the main circuit electrically.

- Long operational life.

Additionally, some of the specific requirements of the sensor are:

- No in-factory calibration.
- Simple installation in indoor locations and in original substation equipment.
- Cost-effective retrofitting in old substation facilities.
- Intelligent electronic device (IED) support for all the RogoCoils of the ABB's portfolio.
- Plug-and-play automatic RogoCoil transducer detection.

**System specifications and functionalities** Taking into account the previously mentioned facts of each input of the system definition stage, the RogoCoil current transducer gathers the best characteristics and advantages for the system implementation. In this way, considering the state-of-the-art achievements using RogoCoil for AC current sensing [76, 114, 115], the RogoCoil current sensor system should comply with the following specifications:

- Accuracy class 0.1% over AC currents from 10 A to 10 kA at 50/60 Hz.
- Implemented with commercial, low-cost electronic components.
- Accuracy class guarantee in a temperature range from -5 °C to 40 °C.
- Sensor lifetime (> 20 years), electronics (> 10 years).

### 5.3.2 System-level design

The system-level design stage allows an early exploration of the system architecture and its functionality so that it allows to clarify the system specifications considering a limited set of available technologies (*technology space*). This is possible by system-level modeling and simulation using the proposed VP-based design methodology and the selected VHDL-AMS *simulation framework*, see chapter 3. In this way, the system-level design stage avoids an early division of the design process of the complete system; therefore, it opens the possibility to study different system architectures, to understand critical variables and interactions between the main subsystems[1] of the design, and to optimize the complete system as a whole.

Figure 5.3 shows the inputs, outputs, and main processes executed at this stage. The main activity in the system-level design stage is to identify, define, interconnect, and analyze the main

---

[1] A subsystem must be understood as an aggregation of components. A complete system is often composed of two or more subsystems.

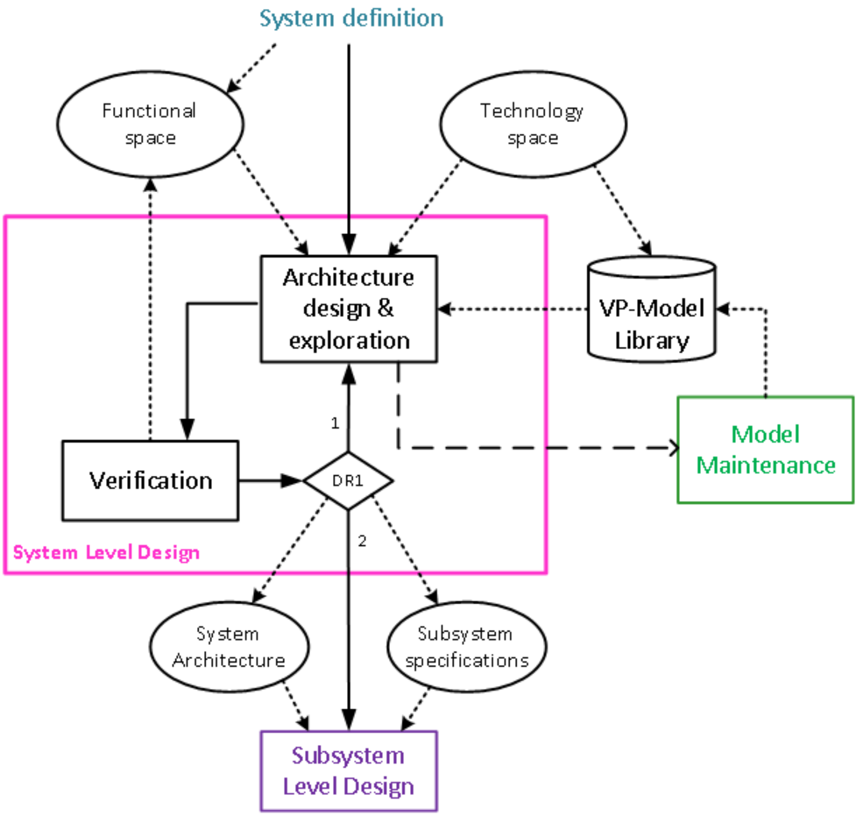


Figure 5.3: System-level design diagram. The dashed arrow indicate a modeling request to an external process.

subsystems and some components of the design by building an adequate system architecture. This process is called *design space exploration* (Architecture design & exploration process in Figure 5.3). The result of this process is a *virtual prototype* (VP), i.e. a model of the complete system. The theoretical concepts behind this design process are inspired in PBD theory, where we try to match the *functional space* with the *technology space*. The *functional space* represents the list of functionalities, specifications, requirements, conditions, and constraints of the system; this is the information that we obtain from the system definition stage. On the other hand, the *technology space* is a selection of all possible technologies that can be used to build the system and its subsystems. If low cost is an important general requirement for the system, some expensive existing technologies, which might be used to build the system, would not be considered in the *technology space*. Contrarily, new technologies in development, which could be subject to investigation in the current design project or in a parallel project, can be considered in the *technology space*.

To illustrate a typical example of architecture design and technology selection, let us suppose that we need to implement the signal processing algorithm that performs the integral of an analog signal as it is done in section 4.4 on page 104 for the current measurement unit of a RogoCoil sensor. We have considered the following three possible solutions:

1. using passive and active lumped element circuit components such as resistors, capacitors, and operational amplifiers (analog implementation);
2. using an ADC and an FPGA (digital hardware implementation);
3. using a microcontroller (digital software implementation).

All the previous options use different type of technologies (components) to implement the same functionality, but with different performances, accuracy, and precision. By effectuating system-level modeling and simulation, we can compare the three different architectures and verify if they meet the system specifications. Although we can get more than one implementation that meets the system specifications, it is very useful to run a verification procedure able to find out an adequate implementation based on the simulation results. Likewise, it is possible to explore the effect of different parameters of the system, as it was done in the *virtual prototyping* example shown in section 4.4.2.

The suggested approach for performing the architecture design and exploration by using *virtual prototyping*, is a meet-in-the-middle approach, i.e. a gradual modeling of the system architecture from the highest level of abstraction to lower levels supported by a library of components, namely **VP-Model Library**, see Figure 5.3. The VP-Model Library is an online database which can provide *component models* at different abstraction levels, see its implementation in Appendix F. In a meet-in-the-middle approach, the designer tries to find an appropriate architecture with its main subsystem specifications that match the *functional space* with the *technology space*. In other words, the goal is to study the best architectures that meet the requested functionalities and system specifications by using a selection of the available components (technologies).

The VP-Model Library database allows the system designers to query modeling elements that can be used to build the desired system architecture faster than building it from scratch, see section 3.6. The fact of reusing models, *test benches*, and other modeling elements, reduces the overall design and modeling effort and helps to speed-up the design process. In order to explain this concept easily, it is here assumed that we count with a reliable and rich library of models, which provides well-documented and trusted models. In fact, this is one of the most important goals of the VP-Model Library in the long term.

As is shown in Figure 5.3, the architecture design and exploration is followed by the *model verification* process, in which the goal is to verify the modeled functionalities and behaviors of the system by simulating its VP. It is important to mention that the models provided by the VP-Model Library have been previously verified independently. Therefore, the designers count with the verification models (e.g. *test benches*, configurations, packages) that permit to check the *component models* separated from the VP. One of the important verification tasks is to create the required verification models for simulating the VP. According to the VP characteristics and the simulation needs, a VP and its verification models can support different types of simulations (e.g. time-domain, frequency-domain, parametric, etc.) and verification techniques (statistical, worst-case simulations, functional coverage, etc.). Although several types of *model implementations* could be required to achieve different simulation objectives, at the system-level design stage, it is crucial to verify the correct functionality and the performance of the complete system architecture. The modeling and simulation of the RogoCoil VP presented in chapter 4, show examples of different *model implementations* for parametric and worst-case simulations in time-domain.

The system's architecture design and verification is an iterative process that can iterate multiple times before obtaining the required information to pass into the next stage of the system development flow, i.e. the subsystem-level design. The decision rhombus called '`DR1`' in Figure 5.3 shows two paths for the system design flow. Path number 1 represents an iteration of the architecture design and exploration process after its verification. Path number 2 represents the finalization of the system-level design stage. The two possible scenarios after VP verification are:

1. **Incorrect or unexpected verification results:** they could suggest an erroneous architecture or the use of an incorrect *component model* for the specific application. In a new experimental VP environment, a *component model* could work in conditions out of its range of operation. These type of issues are easy to identify in simulation, especially when comparing the simulation results of the *component models* alone and connected within the system architecture. In any case, architecture redesign and verification are required until it is obtained satisfactory simulation results.

2. **Correct verification results:** positive verification results suggest that the functionalities, algorithms, and the modeled interactions and behaviors are correct under the set of assumptions and conditions given in simulation. However, this not necessarily means that the design is finished and complete. In fact, the first verification tasks are done for high-level functional models that do not take into account most of the non-idealities of the real subsystems and components. Therefore, design iterations are needed in order to verify the VP each time with more refined *component models* or more refined architectures.

VP verification results could also make necessary to change the system specifications initially defined in the system specification stage, see functional space in Figure 5.3. For example, suppose that after an iterative architectural exploration and a worst-case verification analysis of the RogoCoil current sensor VP, the best results showed that the studied self-calibration architecture only meets an accuracy of 0.1% in a current range from 100 A to 5 kA by using low-cost electronic components. In spite of this limitation, the system could still be used for measurement applications within a smaller current range. On the other hand, the self-calibration architecture can still be used for measurement purposes out of this current range as long as it improves or equals the accuracy obtained with other methods or technologies, normally within 0.5% accuracy class. In both cases, the design process should continue with slightly changed system specifications.

An iterative *model refinement* is a key concept for the system-level design. This is supported by the VP-Model Library, which contains *model implementations* at different abstraction levels. Each *model implementation* has the function of modeling the functionalities and behaviors of a certain device/component/subsystem considering a particular set of ports, variables, parameters, and algorithms. The refinement and verification process in a VP shall be done one component at a time, i.e. a more refined *model implementation* is verified independently, later on, it is integrated into the VP, and the VP verification is re-executed. A good indicator of the detriment of the of simulation performance in a large VP can be obtained by component refinement and gradual integration into the VP, i.e. a *component model* at a time. One of the main functions of the VP-Model Library is to minimize the modeling effort by providing properly verified models with its respective *test benches* and documentation.

Nonetheless, what if the VP-Model Library does not contain a required model or a more refined *model implementation*? In this case, the system designer must create the required model according to the design needs. This is a necessary task that non-expert system designers could perform when high-level functional models are needed. However, modeling and verifying subsystems and components are not trivial tasks at lower levels, especially when refined behavioral or physical models are required. The elaboration of refined *component models* and their verification requires both domain-specific knowledge of the modeled subsystem/-component and modeling expertise. Therefore, a model request (see dashed arrow in Figure 5.3) from the architecture design process to an external **Model maintenance** process can be done if a required model is not available in the VP-Model Library. The Model maintenance is an independent process which can run in parallel with the system development flow. The main objective of the Model maintenance process is to create, verify, and update the modeling elements of the VP-Model Library, see section 5.4.

Even if correct verification results are achieved, we can see that *model refinement* and verification iterations are encouraged in this approach. So, the challenging question is: for how long must we continue performing this iterative process? Well, there is no specific rule allowing to know when to stop doing *model refinement*. In principle, system designers are limited by the refinement level of the models available in the VP-Model Library. However, it is not the purpose in the system-level design stage, to model every single component of the system in detail. Only the most critical design variables and the main interactions between subsystems must be modeled at this stage. Highly refined models of complex systems present several complications for modeling and simulation. Therefore, system designers must balance the VP size and the refinement level of the VP models. The VP of the RogoCoil current sensor system introduced in chapter 4 presents clear examples of gradual *model refinement* for system-level simulation, from high-level functional models to behavioral *component models*. Adequate models for system-level simulation include the critical behaviors and non-idealities that determine the accuracy of the sensor; from slightly refined models such as the output voltage deviation errors of the voltage sensing block, until more accurate models such as or the $2^{nd}$ order $\Sigma\Delta$ ADC.

The main outputs of a successful system-level design stage are the following:

**System architecture:** a clear system architecture must be defined and selected after the architectural exploration and verification. If multiple architectural solutions were considered at the beginning of the stage, the architectural exploration and verification must perform a comparative study that shows the best architecture for the design, see the RogoCoil current measurement unit example in section 4.4 on page 104. Although the models that represent physical interconnections and interactions between subsystems and components could be implemented in an abstract fashion in the VP, the technologies used for the main subsystems and some of its components must be properly identified as a result of the design exploration. For example, let us consider the analog architecture of the current measurement unit shown in Figure 5.4. The main unit blocks that compose the system are presented as high-level abstractions that model the specific functionalities that are required. However, this representation of the system architecture is consistent with the different components that can be used to realize the physical system. For instance, the **Analog Integration** block can be realized using active and passive lumped element circuit components such as OpAmps, resistors, and capacitors; the **Digital Calculation** block can be realized by using an FPGA or a microprocessor. More refined *model implementations* of these subsystems are done in the subsystem-level design stage, in which is obtained the netlist and further details about the specific components that are going to be used to realize the system. In this way, the VP will represent more accurately the real system. Likewise, it is easier to model the effect of cross-domain variables such as the temperature or mechanical interactions. In the example shown in Figure 5.4, we can observe that the effect of the room temperature ($T0$) is only taken into account in the RogoCoil sensor block. This is correct if the electronics of the current measurement unit is thermally isolated from the RogoCoil sensor, and/or the cross-temperature dependence of the electrical variables in the current measurement unit is not significant compared to the RogoCoil sensor. In this case, the RogoCoil thermal model can be implemented inside the RogoCoil subsystem block. Contrarily, if all the subsystems and components can be affected by the room temperature or by thermal interactions between the system components, a good modeling option is to use a separate 'system thermal network' abstract model, which represents the critical heat transfer interactions separated from the electrical variables of the system as it is done in the electro-thermal model of the RogoCoil sensor system shown in section 4.2.

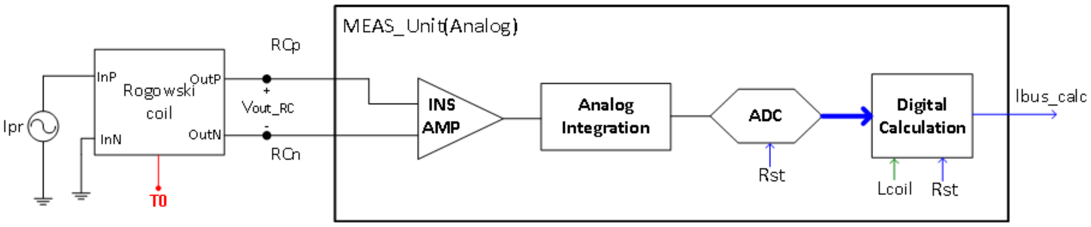


Figure 5.4: Block diagram example of the Rogowski coil and its analog current measurement unit architecture.

**Subsystem specifications:** an important part of the system architecture definition, is the proper identification of the principal subsystems and the critical variables of the design, the interactions between the subsystem interfaces, and the main subsystem specifications in terms of the modeled and simulated critical variables. Therefore, the architecture exploration and verification process must yield a clarified view of the system specifications obtained in the system definition stage. Since the main subsystems have been identified, the system specifications and functionalities must be distributed along the subsystems of the design. As a result, we can obtain the list of subsystem specifications for each subsystem. Considering the RogoCoil sensor system example developed in chapter 4, suppose that the result of the virtual system-level exploration clarifies and verifies that the general system architecture shown in Figure 4.65 on page 129, as equal as the SCU architecture shown in Figure 4.26 on page 87. Additionally, the analysis performed for the two different architectural implementations (analog and digital) of the current measurement unit, see section 4.4, give us structured arguments to prefer the analog implementation shown in Figure 5.4. The modeled system and subsystem architectures meets the system specifications (modified during the system-level design), and defines the most important subsystem specifications as follows:

- The calibration resistor values and tolerances
- The calibration signal amplitude, DC, and frequency values.
- HPF and LPF main cutoff frequencies, settling time, stop-band attenuation and band-pass gain.
- RMS-to-DC converter time delay, bandwidth, accuracy, and non-linearity.
- ADC architecture type, resolution, precision, dynamic range, SNR, ENOB, etc.
- The clock frequency of the self-calibration control.
- Main specifications of the components of the CMU such as the ADC, filters and analog integration architecture/method (analog or digital).

### 5.3.3 Subsystem-level design

The O-model approach proposes an initial system-level design stage for architectural exploration at reduced complexity, followed by a subsystem-level design stage in which the main goal is to let expert designers complete the design of each subsystem and its components in a parallel way. The specifications of the subsystems and their components are not well-defined in the system-level design stage. Therefore, the design of the subsystems and its related components must be completed by using the tools and techniques that are optimized for specific-domain design. Although the VHDL-AMS *simulation framework* and compatible modeling languages can be used to model and simulate multi-domain interactions by equation-base models with conservative or acausal port interfaces, detailed interactions, and behaviors in mechanical, thermal, optical, or electro-magnetic domains cannot be directly modeled by the selected VHDL-AMS *simulation framework* for the system-level design.

Figure 5.5 shows the general design flow that shall be performed by each principal subsystem independently. Each subsystem-level design procedure receives the output data of the system-level design stage, i.e. an incomplete set of subsystem specifications and its main functionalities (**subsystem functional space**), and the information about where and how the subsystem is going to be used as part of a defined **system architecture**. The main critical variables, a basic subsystem architecture, and some specific behaviors of the subsystem components are commonly modeled in the system-level design stage.

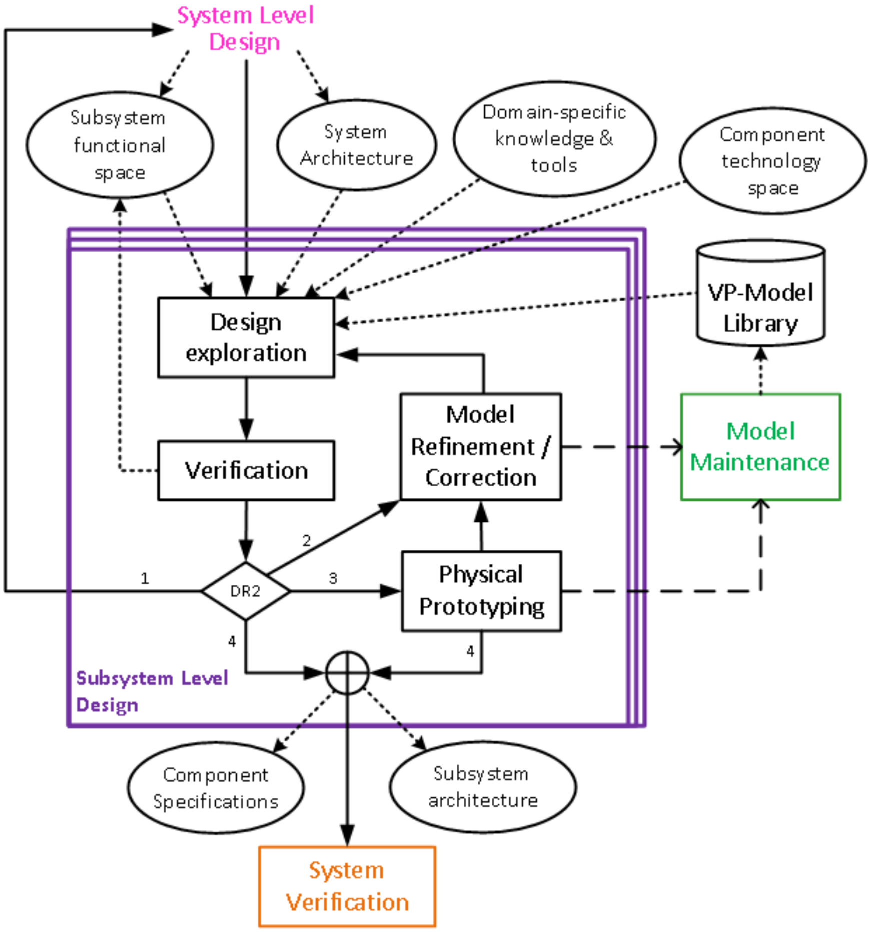


Figure 5.5: Subsystem-level design diagram. The dashed arrow indicate a modeling request to an external process.

As it is shown in Figure 5.5, two additional inputs are considered: the ***technology space* of the components** of the subsystem, and the **specific knowledge and tools** that are applied in a particular domain. For example, methods and tools that are used for geometrical modeling and simulation of mechanical, thermal, optical, and multi-physics effects; particular design methods and application tools that are used for electro-magnetic interference (EMI) protection of embedded electronics and the EMC for equipment operating in industrial environments; or particular methods and software tools that are used for analog electronics design and digital electronics design and synthesis [116, 117].

Since simulation tools, modeling languages, and modeling techniques have evolved independently, specific subsystem design needs are better addressed by particular *simulation frameworks* and techniques that have been optimized for specific applications. For example, mechanical interactions in actuators and sensors can be modeled by finite element method (FEM) models, which can describe physical and geometrical interactions that cannot be directly described in equation-based models in VHDL-AMS. Likewise, the analysis and control of conducted and radiated emissions from electrical and electronics systems are carried out by electro-magnetic models and specific software tools [118]. These tools combine several types of solvers (e.g. time-domain, frequency-domain, integral equation, multilayer) that are based on different methods such as transmission-line matrix (TLM), finite integration technique (FIT), finite-difference time-domain (FDTD), method of moments (MoM), among others [119]. These methods are chosen according to the type of problem that is treated, e.g. cross-talk calculations, printed circuit boards, EMI problems, packaging problems, etc.

Similarly, the VHDL-AMS *simulation framework* has evolved from hardware description languages for digital electronic design, such as VHDL and Verilog, and integrates the functionalities of analog circuit simulator such as SPICE. Therefore, this *simulation framework* is adequate for most of the details of subsystem-level design in analog/digital electronics and electrical power circuits. For example, detailed transistor-level and lumped-element circuit models for performing time and frequency domain simulations can be described and simulated using the VHDL-AMS *simulation framework*.

The subsystem-level design can also be described as an iterative process which involves design exploration, verification, and *model refinement* for all subsystem domains, see Figure 5.5. The design exploration can also consider architectural evaluation of the subsystem. Nevertheless, for a detailed component modeling, this step mainly consist on the exploration of behavioral and *physical models*. The following cases can arise after *model verification*, see the decision rhombus `DR2` in Figure 5.5:

**Path 1:** The verification results with detailed models could reveal issues caused by different reasons, e.g. wrong assumptions for the component abstractions, component behaviors and/or interactions that were not considered in the system-level design. In this case, the subsystem specifications might require a modification. However, a change in the subsystem specifications could affect the way in which the subsystem interacts with the other components of the system; and therefore, a system-level redesign could be necessary.

**Path 2:** Correct or incorrect/unexpected verification results might need *model refinement* or model correction respectively. In both cases, another design iteration is required. Since other *simulation frameworks* can be used, the model refinement/correction process is an internal process carried by the designers of the subsystem. However, for the final system verification stage, it is required that some of the additional behaviors and interactions modeled and analyzed in the subsystem-level stage can be added to the *component model* used in the system-level VP. Therefore, a modeling request to the **model maintenance** process can be made in order to include those behaviors in a *model implementation* stored in the VP-Model Library, see details in Appendix F.3. Although this process is very useful to improve the quality and autonomy of the *component models* and the richness of the VP-Model Library, migrating all the additional behaviors modeled in the subsystem-level design stage is not always possible and practical.

**Path 3:** Considering that not all the component behaviors and interactions can be properly and/or practically modeled and simulated within a reasonable effort, there are cases in which it is more practical to elaborate a physical prototype and take experimental measurements than creating an accurate virtual prototype. For example, in the work reported in [79], and later improved in [82], different types of distributed lumped-element electrical models of the RogoCoil were compared and characterized with experimental measurements for time and frequency domain simulations. Despite they are very detailed parameterized models that take into account physical and geometrical parameters of the RogoCoil, they only can reproduce the real RogoCoil behavior approximately. Similar results are reported in [81, 120]. At this point, the effort to improve the modeling results are too high. Instead, experimental measurements on the physical device can be used to characterize the RogoCoil models in order to minimize the differences between the experimental measurements and simulation results. Under this philosophy, virtual and physical prototyping can be used as complementary techniques rather than competing approaches. Elaborating a small physical prototype for studying a subsystem or a part of it, could be faster and less complex than creating a detailed model. So that, the measurements on the physical prototype can be used to understand, verify, and directly clarify the specifications of the subsystem and its components. Moreover, these measurements and additional testing performed with the physical prototype can also be used to create, correct, validate, and refine subsystem models. This is why in Figure 5.5, we can see that the results of the physical prototyping process can be used for an additional subsystem design and verification iteration when a *virtual prototyping* activity has been previously done, or they can go directly to the output of the subsystem-level design stage when the component specifications and subsystem architecture are clear enough. Nonetheless, the experimental results obtained from physical prototype testing can be used to model additional behaviors and interactions that are not included in the most refined *model implementation* used in the system-level VP. Therefore, in a similar way to the modeling request done in the *model refinement*/correction process in path 2, a modeling request to the external model maintenance process is highly desirable in this case. This is very useful for the final system verification stage, and in general, for future re-utilization of the models of the library.

**Path 4:** This path represents the finalization of the subsystem-level design stage. As it can be observed in Figure 5.5, satisfactory verification results can be obtained from the virtual iterative verification process, from measurements taken in a required physical testing process (physical prototyping), or from both sides. In this case, the combination of these results shall clarify all the subsystem specifications in terms of a distributed set of **component specifica-**

**tions**. It means, that the output of this stage is a set of realistic component specifications that are described together with the **subsystem architecture**, i.e. the detailed list of components and their specific interconnection (subsystem implementation). For example, transistor-level schematics, circuit netlist, embedded C/C++ code, synthesized VHDL code, among other detailed outputs.

### 5.3.4 System verification

The system verification stage is the final verification of the complete design before building the first physical prototype of the complete system. This stage gathers all the information coming from the previous system and the subsystem-level design stages, see Figure 5.6. The main goals in this stage are: firstly, elaborate a final VP of the complete system including well-refined parameterized models of the subsystems and its components; and secondly, perform the last set of system verification tasks using the final VP.

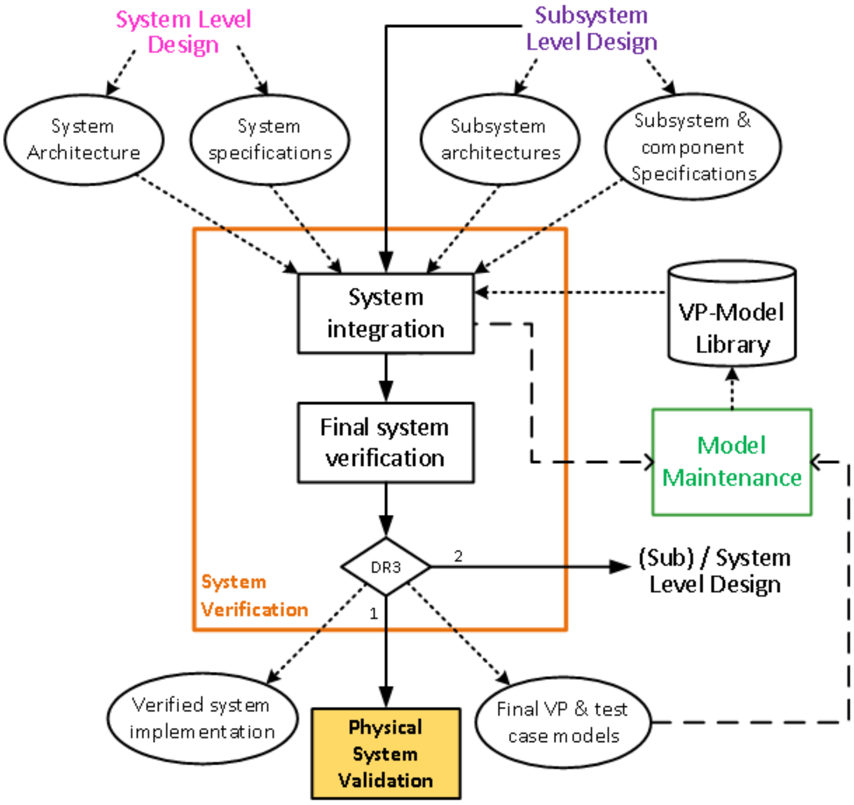


Figure 5.6: System verification stage diagram.

The inputs of this stage are shown in Figure 5.6: the VP-Model Library with updated *model implementations*; clarified specifications of the system, subsystem, and components; and the selected system and subsystem architectures represented by the system VP and its related modeling elements. The virtual **system integration** process complements and finishes the meet-in-the-middle design approach started by the architecture design and exploration at the beginning of the virtual design process. Whereas at the system-level design stage the overall approach looks like top-down, i.e. from high to low level abstractions, the design approach in the system verification stage is more bottom-up like. New validated and parameterized *component models* available in the VP-Model Library, shall be the result of modeling requests done in the subsystem-level design stage to the external Model maintenance process. These new *model implementations* are abstractions that could model all, or at least, the most relevant component functionalities, non-ideal behaviors, and physical interactions to be simulated in the system VP. The VHDL-AMS electro-thermal model of the RogoCoil sensor system presented in section 4.2 on page 57, is an specific example of a refined *model implementation* that has been updated in the VP-Model Library after a detailed study that would normally carried out on a subsystem-level design stage.

Ideally, if all modeling requests done in the previous stages are updated in the VP-Model Library properly satisfying the modeling needs, the system integration should be a fast process since the designers count with a previously created VP and updated *component models* in the library. At this point, the system integration tasks are highly dependent on the Model maintenance tasks. The system integration could make new modeling requests to the Model maintenance process. The system integration process only finishes when all the modeling requests have been satisfied and the new models are integrated into the VP.

The **Final system verification** process consists of creating the *test benches*, packages, configurations, and additional modeling elements to define the final *test cases* that will verify the system design. At this stage, additional verification procedures should be implemented in order to test the new behaviors that have been included in the *component models* of the VP. Adequate system verification procedures will allow clarifying the multi-domain interactions at the *component models* interfaces and the effect of the most critical environmental variables.

The decision rhombus `DR3` in Figure 5.6 shows two possible paths after the final system verification. If the verification results reveal issues among the interactions of the components, or if the system specifications cannot be met by using the final VP configuration(s), all the issues need to be solved before proceeding into the next development stage. The verification results must show useful information to detect the origin of the problem. Returning to the previous design stages might be necessary. On the other hand, successful verification results will show that the achieved system implementation (i.e. a fully determined set of components and their interconnection) meets the system specifications and no issues are found after performing different verification procedures. The more rigorous and complete are the system verification procedures, the lower the probability of getting a design issue in the physical elaboration of the system. The final VP and their verification models can be submitted to the Model maintenance procedure to be stored in the VP-Model Library.

### 5.3.5 Physical system validation

This stage comprises the physical fabrication, integration, and testing of the system. The information resulted from the three previous *virtual prototyping* stages shall give sufficient and precise information to fabricate and integrate the physical system. At this stage, the system can be integrated by following the sequence proposed in the right part of the V-model (i.e. the **construction** part), see Figure 1.1 on page 2. It means that the subsystems must be fabricated and tested separately; afterwards, all the parts must be integrated until obtaining the complete system. Finally, several tests shall be performed in order to determine that the system meets correctly all the requirements, specifications, and functionalities.

The three stages of the *virtual prototyping* phase contribute to the reduction of the number of physical prototypes that are commonly required when the design and verification tasks are done using a classical approach, i.e. using a ‘paper-driven’ design [20] and physical exploration. Although physical prototype testing could start in the subsystem-level design stage, the idea in the physical system validation stage is to fully complete this task by fabricating and integrating a physical prototype of the complete system. The unique way to validate the correctness of the results given by the VP and its *component models* is to verify the designed behaviors and functionalities building the physical system. According to the size and complexity of the system, the physical validation must be performed initially to the subsystems, later on, the most critical subsystem interconnections (e.g. sensor and front-end electronics), and finally to the complete system.

Additional work towards the design of the final physical prototype, namely the 0-Series product, involves tasks such as the packaging of the system, communication protocols, the physical appearance, the hardware and software user interfaces, among others. In fact, the 0-Series product is more than a simple prototype, is the first final system that is fabricated before mass production and commercialization. The 0-Series is used for the final operational and environmental tests of the system. Those tests are of vital importance to the complete system validation. Some 0-Series system validation might involve the following tests:

- Functional system testing at normal operation conditions.
- Functional system testing at critical operation conditions (i.e. at operational limits and out of range).
- System performance testing. This involves the test of the main system specifications, especially the specifications that makes the product attractive, such as the response speed, accuracy, linearity, and robustness.
- Environmental testing such as EMC, temperature, noise, humidity, radioactivity, or other harsh conditions.
- Safety tests, this evaluates the behavior of the system during probable dangerous scenarios such as very high tensions, fire, or short circuit.

Failing to pass the complete set of testings would require coming back to previous design stages to solve the issue(s) according to the origin of the nature of the failure(s). The theoretical premise under discussion is that the proposed O-model, together with the VP-based design methodology, reduce the probability of presenting design errors at the physical validation stage, especially in the 0-Series validation tests. This is true, if and only if the VP and its *component models* are correct abstractions of the real system and components under real conditions. Unfortunately, proving this hypothesis is out of the scope of this thesis. This will require the implementation of this methodology within a company, and to apply it to several development projects. This process could take several years.

### 5.3.6 System manufacturing

The manufacturing stage consists of the mass/unitary fabrication of the product that will be commercialized. At this stage, all the safety verifications and correct functional validation of the 0-Series system have been done successfully. Additionally, the final system packaging, hardware and software user interfaces, user manual, system datasheet, among other final system terminations must be concluded at this stage. According to the physical system validation results, the final product might differ from the 0-Series system if additional corrections are required.

The manufacturing process can be done by the same company that has designed the product or can be outsourced to one or more companies. Supply chain management, vertical and horizontal integration strategies and theories can be used for optimizing (at an operational level), not only the manufacturing process but also the design of complex systems. This is possible by correctly identifying the companies' know-how and organizing the supply chain structure. This is clearly out of the scope of this research.

### 5.3.7 System commissioning

In this phase, the product is installed and brought into working condition in the client's location. The person/teams executing the commissioning task must make the product/system fully or partially operational according to the client needs in a specific location. The commissioning can be done completely by the client, by the manufacturer company, by a third-party company (distributor), or by a mix of the aforementioned.

### 5.3.8 System maintenance

The maintenance stage consists of the complete or partial revision of the system/product in order to preserve it and guarantees its correct operability. Typical maintenance procedures are carried out periodically by the client, by the manufacturer company (contracted service), by a third-party company (distributor), or by a mix of the aforementioned. However, in the design of power system products, it is highly desired to minimize the human intervention in order to reduce operational costs. Therefore, the power systems are designed to operate for decades. Since the system is made of several types of components that can have different operational lifetimes, it is a common task in the system maintenance stage to replace certain components of the system. For example, standard electronic components based on CMOS technology have an operational lifetime of about 10 years [121], whereas *instrument transformers*, RogoCoils, or similar electrical devices can have an operational lifetime of more than 30-50 years [122]. Therefore, small redesigns may be required when a component that needs to be replaced is not produced anymore. The availability of VPs and its related verification models would be very useful to make these small redesigns and other bug fixing tasks in the system maintenance stage.

The information obtained from the commissioning and maintenance activities has a high commercial importance for the company and also a great technical and scientific importance for developing new products/systems. This information can be used in the next system definition life cycle and be used as a source of sustainable innovation for the products of the company. For instance, suppose the case of a client who wants to acquire a specific product (e.g. a system for electrical monitoring or protection) to be used in a specific location with non-conventional conditions. In the commissioning/maintenance stage, the engineers observe that the product present an undesirable behavior. Further analysis would show that the reason is caused by subtle environmental conditions which were not taken into account for the product design, these will change the system requirements and specifications for a new product and could also lead to research in new technologies.

## 5.4 Model maintenance

One of the essential requirements of the proposed *virtual prototyping* based approach employed in the design and verification stages of the O-model, is the availability of good quality, clear, and updated *component models*, VPs, *test benches*, *test cases*, and their related documentation. This implies a strong modeling and simulation activity using the VHDL-AMS *simulation framework*. *Virtual prototyping* is a key and useful mechanism for the system development flow, but is not the final goal of the development life-cycle. This is why this research proposes an external process outside of the system development flow, that can be used to support the design and verification stages of the O-model.

The model maintenance is an independent process, that can run in between development life-cycles or in parallel with the design and verification stages of the O-model. The main purpose of the model maintenance is to create, verify, and update the modeling elements of the VP-Model Library. Figure 5.7 shows the five different scenarios that generate inputs to the model maintenance process. They are explained as follows:

**System-level design request:** As it has been described previously in section 5.3.2, the architecture design and exploration process in the system-level design stage might need a

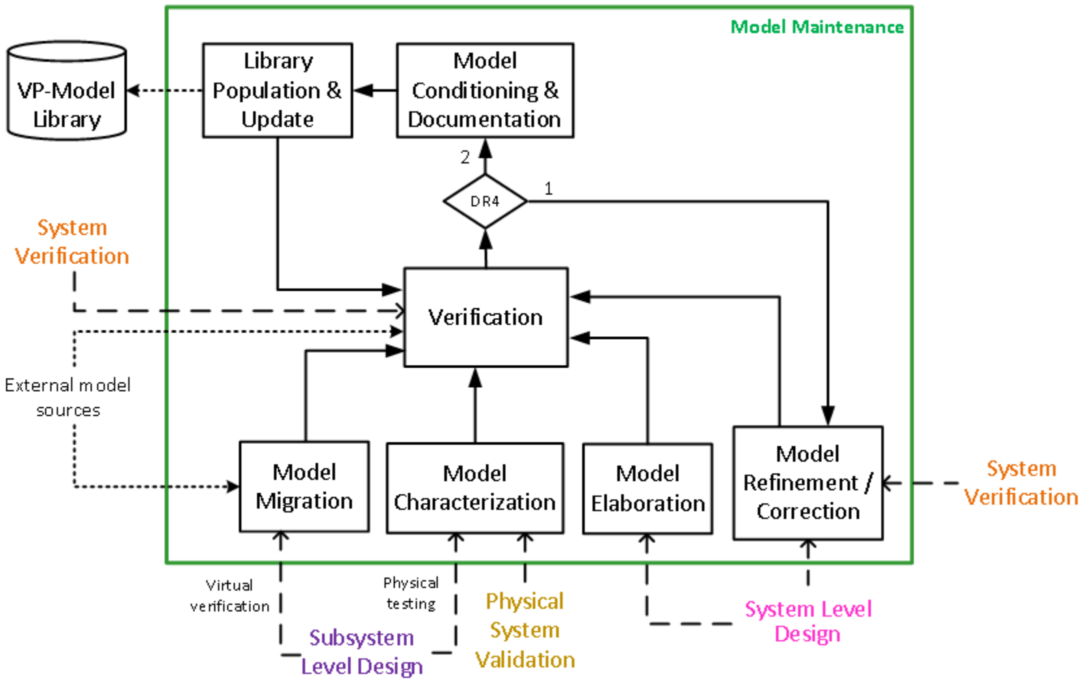


Figure 5.7: Model maintenance flow. The flow goes from the bottom to the upper part.

specific *model implementation* that is not yet available in the VP-Model Library. This is the typical case when a more refined *model implementation* of a *component model* is required for the iterative design and verification procedure. In this case, if the elaboration of the *model implementation* requires a specific level of domain expertise that the system designer does not have, and/or a significant degree of modeling effort, a modeling request can be submitted to the model maintenance process. The modeling request consists of the detailed explanation of the interface ports, parameters, behaviors, and functionalities that are required in the solicited *model implementation*. For that purpose, the system designer must provide this information using a *model registration form* (MRF), see details and examples in Appendix F.3 on page 246.

**Subsystem-level design requests:** The path 2 and 3 in Figure 5.5 described in section 5.3.3, explain two different cases in which a modeling request can be generated from the subsystem-level design stage. The first case is when **virtual verification** is used (path 2), in this case, the result of the *model refinement*/correction process in the subsystem-level design stage could be used to implement a more refined *model implementation* using the VHDL-AMS *simulation framework*. The electro-thermal model of the RogoCoil sensor system explained in section 4.2 on page 57, is a specific example of this case; where a 3D geometrical FEM model of the RogoCoil sensor system made in COMSOL Multiphysics®, is used to simulate the heat transfer in the RogoCoil system, and later on, for obtaining an equation-based electro-thermal model of the RogoCoil sensor system in VHDL-AMS, see section 4.2.7 on page 78. On the other hand, when **physical testing** is used (path 3), the experimental measurements obtained with the physical prototype can also be used to implement a more refined *model implementation* using the VHDL-AMS *simulation framework*. For example, the experimental measurements taken for several RogoCoil samples reported in [79], were used to characterize the VHDL-AMS asymmetric and symmetric electrical models of the RogoCoil by including correction coefficients for the RogoCoil parasitic capacitances, inductances, and resistances in each loop.

**External model sources:** Since the model maintenance is an independent process, the modeling and simulation activity does not necessarily come from a modeling request from the O-model design and verification stages. External model sources can also be used to include models in the VP-Model Library, for instance, there are available electrical power systems libraries created in both VHDL-AMS [123] and Modelica [124]. In the first case, the request can go directly to the verification process, in the latter case, it is required to execute a Model Migration process to obtain the equivalent VHDL-AMS models.

**System verification request:** As it is explained in section 5.3.4, the virtual system integration subprocess carried out before the final system verification, see Figure 5.6, can only finish when all the modeling requests made in previous stages have been satisfied and the new models are available in the VP-Model Library. Additionally, the system integration subprocess might produce new modeling request in case of issues in the system integration tasks. Since both the model maintenance process and the system verification stage use the same *simulation framework*, these issues can be handled by the model correction subprocess, or they can be processed directly by the verification subprocess, see Figure 5.7. On the other hand, at the end of the system verification stage, the final VP and its related *test case* models can be submitted to be included in the VP-Model Library. In this case, the information goes directly to the verification subprocess in order to have a second instant checking of the complete simulation project by external designers.

**Physical system validation request:** In principle, the physical testing and the virtual verification procedures must be as similar as possible in order to compare directly the experimental data with the simulation results. The ultimate status of the *component models* stored in the VP-Model Library is the **validated** status, i.e. when a physical prototype has been built to reproduce the results predicted by the model verification, see section 4.3.2.2 on page 101. The final goal of the VP-Model Library is to provide parameterized model abstractions properly validated with physical measurements, this increases the truthfulness of the models stored in the library. Since VPs are approximations of the real systems and its components, it is not expected that the simulation results and the experimental measurements perfectly match each other. Contrarily, in a successful *model validation*, it is sufficient if the simulation results follow the same trend of the experimental measurements with a constant error over the whole operational range and with the same system conditions. In other words, the model must be accurate but not necessarily precise. Model precision can be later corrected by a *model characterization* subprocess.

According to the type of the modeling request at the input of the model maintenance process, one of the following modeling subprocesses is executed:

**Model migration:** When the input consists of one or several *computational mockups* elaborated in a different *simulation framework* which is not directly compatible with the selected VHDL-AMS *simulation framework*, it is required to translate the model semantics from the original modeling language and/or specific software tool to VHDL-AMS, i.e. migrate the model from one platform to the other. For example, there is a large collection of free and commercial libraries written in Modelica (https://www.modelica.org/libraries) that can be used for power systems [124] among many other applications. The IP contained in these libraries can be directly migrated to VHDL-AMS. Although the model exchange between Modelica and VHDL-AMS is possible in almost all cases, a straightforward code-based transformation could not deliver the expected quality in some cases. Principles for model transformation between these two languages are given in [125]; likewise, techniques for automatic generation of models in VHDL-AMS and Modelica for power system applications have been subject of investigation in recent years [126].

**Model elaboration:** This process starts after a model elaboration request come from the system-level design stage. In this case, it is required to elaborate a new model that implements specific behaviors and/or interactions that have not been modeled yet, i.e. there is no previous related model available.

**Model characterization:** This process starts when a *model characterization* request appears typically after physical testing in the subsystem-level design stage or in the physical system validation. The *model characterization* consists in modifying or including new parameters in the model in order to correct the differences between the simulation and the experimental results. Although the characterization of a model can only consist of model *back-annotation*, it is important clarifying that a *model characterization* request shall be done only when new parameters must be included or when the effects of such parameters need to be modified according to the experimental results. If only the values of the parameters are needed to be updated, this *back-annotation* process can be done directly by the system designers.

**Model refinement:** A *model refinement* process starts after the request is submitted at the system design level stage. The *model refinement* process shall yield a more precise and realistic *model implementation* taking into account the less refined *model implementations* that are available in the VP-Model Library. The examples about how to perform a gradual *model refinement* from ideal *functional models* towards *behavioral models* are given in subsections 4.3.2 and 4.4.2. These models can continuously be refined until getting closer to physical implementation.

**Model correction:** This process starts after a model correction request is generated from the system design level stage or from the system verification stage. This case can happen when a model obtained from the VP-Model Library is not behaving as it is expected, or when its simulation cannot be achieved when the model is integrated into a VP. This problem can appear in the architectural exploration subprocess at the system-level design stage, or in the virtual system integration subprocess in the system verification stage. This sort of problems might be caused by trivial reasons such as software tool version incompatibility, or by more awkward reasons such as solver limitations. Since all the candidate models must be properly verified before inclusion to the library, it is expected that most of the model correction requests are caused by simulation errors. However, since *component models* from the library can be used in different conditions and model architectures, design errors are also expected. This is why is very important to submit the model correction request together with the complete VP and all its verification models.

The output of all the previously described process must be a VHDL-AMS. If the model is a hierarchical VP, it must be built following the VP-Modeling Guidelines given in section 3.5. As it is shown in Figure 5.7, all the modeling subprocesses converge in the verification subprocess in Figure 5.7. The verification implies the elaboration of the required *test benches* and *test cases*. In addition to verifying the correct functionality of the model, one of the main goals of a verification procedure is to determine the operational performance and the validity range of the model under verification, i.e. the minimum and maximum values of the model parameters, inputs, and outputs in which a correct response is guaranteed. However, similarly to testing procedures in physical devices, verifying all the degrees of freedom of a *component model* is not possible in a reasonable time [127].

After the verification process there are two possibilities, see decision rhombus `DR4` in Figure 5.7. If the model does not comply with the required functionalities and performances, a model correction process is needed until obtaining the desired results, see path 1. Otherwise, we can proceed to the **model conditioning and documentation** process, see path 2. This process consists in preparing the previously verified model, with all its related verification models, to be included in the VP-Model Library. This preparation consist on the two following main tasks:

**Conditioning:** it consists of organizing, commenting, and writing in a clear fashion the source code of the models so that it can be easily understood by system designers. The models must be written following the modeling guidelines defined in the VP-based design methodology given in section 3.3 on page 28.

**Documentation:** The documentation should contain all the details about the theoretical concepts used for modeling the functionalities and behaviors described in the *component model*. Commonly, the model documentation requires the elaboration of block diagrams, state diagrams, schematics, or any other type of graphical representation that explains the model. The VP-Model Library consists of three main database views which contain a rich selection of data fields that have been properly categorized for a complete documentation of *component models*, VPs, *test benches* and custom packages, see section F.2 on page 227. The minimum information that a verified model documentation must contain is: a proper description of the model, the explanation of how is performed and what is the scope of the verification procedure, and the explanation of the organization of the verification model, i.e. *test benches* and *test cases*.

Finally, the **Library Population and Update** takes place after the model conditioning and documentation process is completed. The Library Population and Update is a task that only can be done by library administrators, see Appendix F.1.1. Since the simulation tools used in the selected *simulation framework* are being updated periodically, the models stored in the library need to be also checked periodically in new simulation tool versions. Otherwise, the correct operation of the models cannot be guaranteed in the long term. This periodical process is represented by the arrow from the Library Update to the Verification subprocess, see Figure 5.7. Additional details about the VP-Model Library, and how must be populated and updated are given in Appendix F.

To conclude, it is worth mentioning that the model maintenance process provides support for the *virtual prototyping* activity associated with the system development flow in the proposed O-model. This is crucial for the correct operation of the design and verification stages.

# 6 Conclusions and future work

The *virtual prototyping* results and data obtained during this research are evidence suggesting that the early system design and verification powered by the VP-based design methodology may contribute to reduce the development cost and speed up the development process of small-scale CPS for power and industrial applications. The specific way that the VP-based design methodology contributes to attaining this goal is by allowing the creation of modular high-level abstractions of complete systems (namely virtual prototypes); so that, the overall design and verification complexity are reduced. This is possible by applying a meet-in-the-middle approach for system-level modeling and gradual model refinement. Specifically, this work proposes mechanisms to enhance *component models*, VPs, and their verification models by improving their modularity and scalability, which are the key features that allow coping with complexity in virtual designs.

An important and challenging problem that has been considered in this work, is the question of how to implement the proposed VP-based design methodology in current industrial companies that can benefit from this approach, but cannot commit a specific team to the creation, support, and maintenance of computational models and its dedicated infrastructure. The proposed O-model provides a solution to this question by giving a detailed procedure that use the VP-based design methodology to include virtual system-level design and virtual system verification stages in the system development flow. The success of the proposed approach depends on the following main factors:

1. **Pragmatic and versatile *virtual prototyping*:** The experience collected during this applied research work, mainly focused on commercial industrial applications, allows to conclude that the *virtual prototyping* activity effort shall be minimum compared to the physical system design and verification effort. In other words, building and manipulating VPs must be an easy process, disregarding the level of reliability and robustness that we can achieve with a model. Failing to accomplish this condition might reduce the interest of companies in adopting *virtual prototyping* for CPS design, especially those companies that do not possess dedicated

teams/specialists working on virtual environments and/or simulation tools. The VP-Modeling Guidelines have been conceived not only with the intention of improving the modularity and the scalability of the models, but also to ease the manipulation and maintenance of large VHDL-AMS designs.

Similarly, the ability of the selected *simulation framework* to integrate other standard modeling and hardware description languages also play a pivotal role at the inception of new *virtual prototyping* methodologies. Particular cultures and practices adopted by system design engineers and domain experts are difficult to change within a company. Moreover, the multi-domain nature of the CPS design makes necessary to use more than one modeling language for optimal modeling and simulation. This is why new *virtual prototyping* methodologies should support multi-language *model implementations*. The author of this thesis has successfully implemented SPICE, VERILOG-A, and C++ *component models* in VHDL-AMS hierarchical models. In spite of the fact that multi-language models were not properly supported by the commercial simulation tools of the selected *simulation framework* at the time of this research. This problematic has been well-understood by tool vendors; so that, an improvement of the support of multi-language models has been observed and is expected to growth in the future.

**2. Rich model library:** An important factor that reduces the modeling effort for system designers is to count with a rich library of models that can provide reliable and fully documented modeling elements. Only by complying with this condition the reuse of models among system and subsystem designers can be maximized. However, this is a demanding task that requires permanent execution together with the system development flow. This is why this work proposes to guarantee the availability of good quality and updated *conditioned models* by implementing a *Model maintenance* process independently from the development processes.

Notwithstanding, it is important to mention that an independent Model maintenance process does not necessarily mean that these tasks must be executed by an external team/company. Since it is highly desirable to count with VPs and *component models*, both properly verified and validated against experimental measurements, *virtual prototyping* is complementary to physical prototyping and their activities must be coupled efficiently within the company. Performing the Model maintenance processes utterly apart from the system development model is highly inefficient, reduces the verification effectiveness of *component models*, and makes the final system verification difficult and prone to errors. The proposed subsystem-level design flow is a general solution to this problematic. Specific solutions depend on particular physical prototyping and integration tasks executed in the company.

**3. Model and simulation stability:** The experience gathered during this work evidences that one of the main issues that hamper the use of VPs, especially by non-experienced users, is the stability of the models and their simulation results. The lack of stability is here categorized according to the origin as follows:

- **Convergence issues:** There are several factors that can make models not converge in both initial (quiescent) or subsequent (time/frequency) simulation stages. The simulation tools rarely offer hints of the non-convergence reason. The great dilemma is understanding whether the issue is caused by an inappropriate model construction, by wrong simulation settings, or by a mix of both. For instance, the insufficient number of time steps, or computing variables with a difference of several orders of magnitude, are among the main causes of non-convergence in transient simulations. Convergence problems are common in the process of building VPs, they may occur by interconnecting *component models* that have been verified and are operating correctly alone, but not together. The gradual modeling and verification approach advocated by the VP-based design methodology helps to cope with this type of troubles. Furthermore, the convergence issues become more problematic when are caused by simulation tool updates.This is an obstacle to the Model maintenance process that makes more difficult the reuse of models. Consequently, it is required further research on systematic convergence checking procedures for model maintenance.

- **Model configuration:** It has been observed numerical instability in VP simulations caused by the use of different configurations, as it is shown in subsection 4.4.2.3. Although these numerical fluctuations are normally very small, they can affect significantly the accuracy of the result when the estimated value is also small. Even though numerical errors can be corrected via simulator settings, e.g. by using a fixed time step instead of a variable time step in transient simulations, VP designers and users need to be aware of this type of issues, particularly when they are caused by default simulator settings.

  This is why the VP-Model Library includes data fields to properly include and document *simulator setups* details and limitations. This type of model meta-information is not normally considered in available model libraries. This practice improves the model trustworthiness and helps to the model maintenance, especially when new versions of the simulation tool are used. Unfortunately, few simulator setup specifications are tool independent. Some of them are used by default in the simulation tool, and their manipulation requires tool expertise and simulation experience. Consequently, it is not rare that the same model produces different results when it is simulated by different simulation tools.

**4. Verification capability:** Taking into account that the final purpose of *virtual prototyping* is to run verification[1] tasks for the complete system and its components, the benefit of *virtual prototyping* can be measured by the ratio between its verification capability and the modeling effort. It is desired to obtain VPs and *component models* with low modeling effort and high verification capability. The verification capability is composed of two aspects: the verification coverage and the potential of performing different types of verifications. The first is related to the extent to which the obtained simulation results confirm the initial hypothesis in a specific type of verification, for example, functional verification. The second is related to the capability to test multiple behaviors of the system and its components, and diverse interactions of the components of the system among them and their environment.

It is well-known that numeric simulation, the type of simulations supported by the models of

---

[1] Remember that by **verification**, this work considers what is normally corroborated by both verification and validation procedures, i.e. functional and non-functional aspects of the design.

this research, cannot supply a high verification coverage due to the exponential amount of computations that are required to test all the possible states and inputs that a system/component may have. Even for small components of low/medium complexity might be impossible to obtain a high coverage in a reasonable time. Formal and semi-formal verification methods, which are still in its *'early dawning'*, such as contract-based design [56] or symbolic simulation [128], propose solutions to increase the verification coverage using mathematical proofs in computational models.

On the other hand, considering that the VP-based design methodology is limited by the capabilities of the selected *simulation framework* for performing numeric simulation, this research only addresses the capability of the models to execute multiple types of verifications in a structured and progressive way. Certainly, by enhancing the modularity and scalability of VPs and their *component models*, it is possible to execute a gradual verification process which is particularly useful for the system design at the early stages of the system development. The selected VHDL-AMS *simulation framework* supports continuous-time (analog), discrete-event (digital), and mixed-signal model abstractions to execute different types of simulations such as operating point, time-domain, frequency-domain, parametric and statistical simulations. These features together with the proposed modeling approach, allow conducting multiple analysis and verification tasks such as functional verification, *design space exploration*, performance estimation, identification of critical design parameters, evaluation of key non-idealities, functional safety, among others.

## 6.1 Future work

This work can be broadened in many different directions according to industrial and academic interests. The author of this thesis proposes the following directions:

**Last stages of the system development life-cycle:** The O-model processes after the Physical system validation need to be formalized in order to obtain a complete circular system development model. This research must contemplate the concepts coming from PLM and Industry 4.0 initiatives. Specifically, the utilization of *virtual prototyping* for manufacturing and commercialization to potential customers. Although the VPs created for manufacturing, marketing, and design/verification are in principle different in purpose and nature, it is highly interesting to investigate possible relationships for unification.

In order to use a high-level VP abstraction for these purposes, the VP must be implemented in different *simulation frameworks* using appropriate ontologies in a strong graphical interface. Thus, non-expert users shall be able to manipulate the system architecture to get and display useful information.

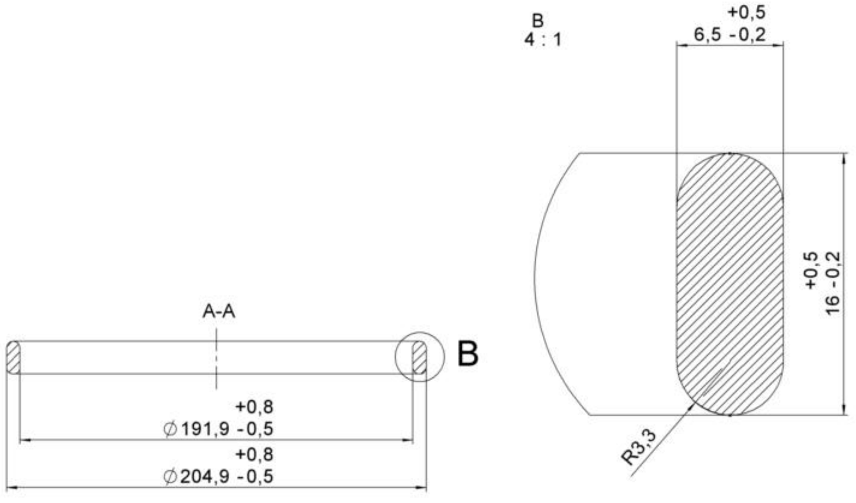
**Automatic reduction of 3D geometrical FEA models:** The heat transfer modeling and analysis of the RogoCoil sensor, presented in section 4.2.6, allow us to see the high modeling effort and expertise that is required for obtaining simple equation-based models that can be implemented in VHDL-AMS. Fostering this type of modeling techniques within a team of system designers, not experts in multiphysics FEA modeling, will require a more practical approach. Model reduction strategies [129] claim to make possible the obtention of computationally cheaper algorithms that however still accurately capture the most important features of the phenomena being modeled. It is interesting to explore these techniques in order to achieve a general automatic reduction of 3D geometrical FEA models. This would benefit enormously the VP-based design methodology, and in general, the CPS modeling and verification.

Model reduction strategies can be classified according to two main approaches: *"reduce-then-model"* and *"discretize-then-reduce"* [130]: "In the former approach the continuous equations representing the underlying physics are first reduced, e.g. by symmetry assumptions that allow us to consider 1D or 2D equations instead of the full 3D equations, before a computational model is derived. In the latter approach, a computational model is obtained by discretizing the continuous equations and only then a reduced model is sought. Some subtopics include spatial dimensionality reduction and multi-scale modeling frameworks in the "reduce-then-model" category; state space and parameter space reduction —with a special accent on reduced basis and proper orthogonal decomposition—in the "discretize-then-reduce" category ".

**Informal contract verification framework:** Since VPs and their *component models* are hierarchically organized to represent the system at different levels of abstraction, the proposed modeling methodology is compatible to the organization of the assume-guarantee (A/G) contract framework [53, 56]. At each level of abstraction, the component has a set of behaviors, requirements, and restrictions (complementary viewpoints) associated with different design concerns (e.g. safety, performance, reliability). They clearly can be expressed by different formalism and analyzed by different tools. The proposal consists in developing a systematic approach to include a priori known complementary viewpoints into the *component models* using any of the *simulation framework* modeling languages; so that, the verification capabilities can be improved by quantifying those design aspects by simulation. This idea is inspired by the analysis presented in [52], where it is stated that most of the requirements and constraints written as contracts for CPS design, even the most complex operations and relations (e.g. on temporal logic and hybrid automata contracts) can be reduced to basic verification tasks. As long as we can quantify those contracts, they can be gradually included in a VP.

# A Appendix A - ABB Rogowski coil specifications

## A.1 KEVCR core dimensions



(a) (b)

Figure A.1: KEVCR core schematics: (a) KEVCR core complete cross section. (b) KEVCR core detailed cross section.

## A.2 KEVCR parameters

| Name | Value/Expression | Description |
|---|---|---|
| $D1$ | 0.1[m] | Primary conductor (busbar) diameter |
| $D$ | 204.9[mm] | Outer core diameter |
| $d$ | 191.9[mm] | Inner core diameter |
| $h$ | 16[mm] | Core height |
| $R$ | 3[mm] | Core curvature |
| $N$ | 4460 | Number of loops |
| $W_{ff}$ | 1.908 | Filling factor |
| $epsr$ | 4.8 | Relative dielectric permittivity of the core |
| $TC_{epsr}$ | 0.022[1/K] | Temperature coefficient of EPSR |
| $d_{wire}$ | 0.224[mm] | Copper wire diameter |
| $d_{resin}$ | 0.255[mm] | Cooper and resin diameter |
| $d_{sc}$ | 0.52[mm] | Distance from shield to core |
| $R_l$ | 0.4338[$\Omega/m$] | Linear coil resistance |
| $C_a$ | 380[pF] | Output capacitance (including cable) |
| $dR_1$ | 0.33 | Loop coefficient |
| $dR_2$ | 0.33 | Loop coefficient |
| $dL_1$ | 0.18 | Loop coefficient |
| $dL_2$ | 0.45 | Loop coefficient |
| $dCs_1$ | 0.6 | Loop coefficient |
| $dCs_2$ | 0.15 | Loop coefficient |
| $dCp_1$ | 0.33 | Loop coefficient |
| $dCp_2$ | 0.33 | Loop coefficient |

Table A.1: KEVCR parameters. All the values are taken at 25 °C.

## A.3 KEVCR datasheet

**Medium Voltage Product**

# KEVCR 24 AC2, 24 OC2 Indoor current sensor

| | | |
|---|---|---|
| Highest voltage for equipment, $U_m$ | kV | 24 |
| Rated continuous thermal current, $I_{cth}$ | A | AC2 - 1250<br>OC2 - 630 |
| Rated transformation ratio, $K_{ra}$ | | 250 A / 0.15 V at 50 Hz<br>250 A / 0.18 V at 60 Hz |
| Accuracy class | | 1/5P30 |
| Length of cable | m | 1.6 |

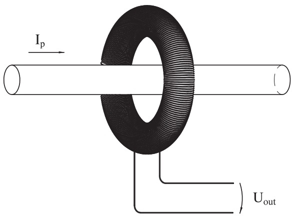
## Sensor principles

A new solution for measuring currents needed for protection and monitoring in medium voltage power systems, is sensor. Sensors based on alternative principles have been introduced as successors to instrument transformers in order to obtain the size reduction, performance improvement, and better standardization. These principles are far from new, but not until now, with the introduction of versatile electronic relays, it has been possible to make use of the sensors advantageous properties.

## Current sensor

The measurement of currents in KEVCR sensors is based on the Rogowski coil principle. A Rogowski coil is a toroidal coil without an iron core placed around the primary conductor in the same way as the secondary winding in a current transformer. However, the output signal from a Rogowski coil is not a current, but a voltage:

$$u_{out} = M \frac{di_p}{dt}$$



In all cases, a signal reproducing the actual primary current waveform is obtained by integration of the transmitted signal.

## Protection and control IEDs (Intelligent Electronic Devices)

The functions of a traditional relay, as well as new additional functions, are included in a protection and control IED. The information transmitted from the sensors to the IED is, during fault conditions, more accurate than the corresponding secondary information from an instrument transformer, hence giving the possibility for a versatile relay function. However, the IED must be able to operate at a sensor's low input signal level with sufficient accuracy, and the signal from the Rogowski coil must be integrated. Modern IEDs (e.g. ABB's Feeder terminals in the RE-series) are designed for sensor use, and they are also equipped with built-in integrators for Rogowski coil sensor inputs.

## Sensor application

The sensor is suitable for the application with Circuit Breaker and Relay as integrated solution.

Power and productivity
for a better world™

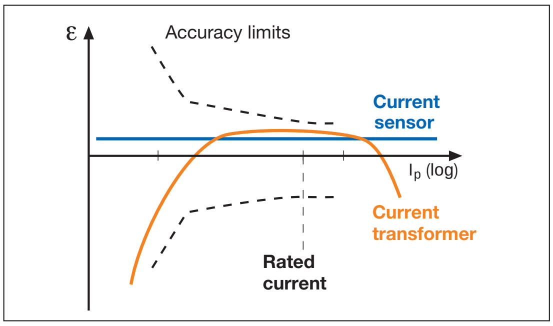
# Appendix A. Appendix A - ABB Rogowski coil specifications

## Differences between Sensor and Instrument Transformer

There are noticeable differences between Sensors and traditional Instrument Transformers:

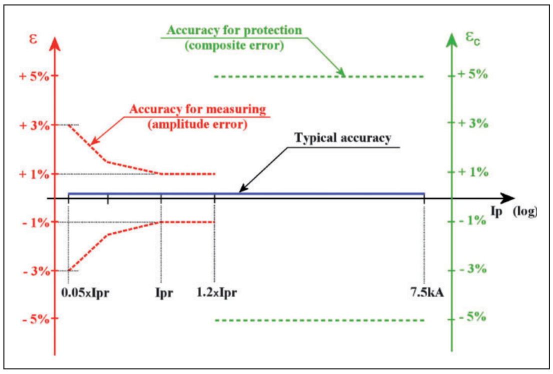
**Linearity**

Due to the absence of ferromagnetic core the sensor is linear up to the highest currents. Measurement and protection can be realized with one single secondary winding with double ratings. In addition, one single standard sensor can be used for a range of rating currents.



For this sensor type, the variation of amplitude error under constant ambient temperature and same application within current range from 5% Ipr ( 12.5 A ) up to 30*Ipr ( 7500 A ) is smaller then 0.2% , that more than fulfils the accuracy class requirements within its whole range.

*Example : Rated current 250 A accuracy class 1 + protection purposes up to 7500 A accuracy class 5P30. The accuracy limits are according to the picture bellow.*



## Compactness

As sensing elements are noticeably small, and the same elements are used for both measurement and protection, the current sensors can be easily integrated into other equipment.

## Correction factor

The amplitude error of a current sensor is in practice constant and independent of the primary current. Hence, it can be corrected in the IED by using a correction factor, measured separately for every sensor.

## Secondary cables

The accuracy classes of the sensor are given at the ends of its secondary cables. The cables are intended to be connected directly to the IED, and subsequently no burden calculation for the secondary wiring is needed. Therefore, every sensor is accuracy tested when equipped with its own cable.

# Technical data

## Standard

IEC 60044-8 (2002-07)
Instrument transformers
Part 8: Electronic current transformers

## Technical parameters of Current Sensor type KEVCR 24 _C2

Current Sensor type KEVCR 24 _C2 : Rogowski coil + 1.6 m cable

## Highest voltage for equipment and test voltages

| | |
|---|---|
| Highest voltage for equipment, $U_m$: | 24 kV |
| Power frequency voltage withstand test on primary terminals : | 50 kV |
| Power frequency voltage withstand test on secondary terminals : | 0.5 kV |
| Impulse voltage withstand test on secondary terminals : | 1 kV |

## Current sensor, rated values

| | | |
|---|---|---|
| Rated frequency, $f_r$: | 50/60 Hz | |
| Rated accuracy: | class 1/5P30 | |
| Rated burden, $R_{br}$: | > 4 MΩ | |
| Rated continuous thermal current, $I_{cth}$: | 630 A | type OC2 |
| | 1250 A | type AC2 |
| Rated short-time thermal current, $I_{th}$: | 21 kA, 3s | |
| Rated dynamic current, $I_{dyn}$: | 63 kA | |
| Rated primary current, $I_{pr}$: | 250 A | |
| Rated transformation, $K_{ra}$: | 250 A/0.150 V at 50 Hz | |
| | 250 A/0.180 V at 60 Hz | |

## Temperature category

| | |
|---|---|
| Operation: | -5°C / + 40°C |
| Transport and storage: | -40°C / + 70°C |

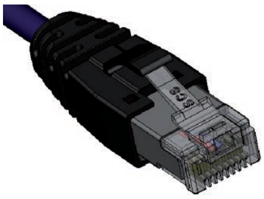
## Protection and control IEDs

Sensor could be connected to a protection and control IED-unit from ABB:

- IED types: RE_601

## Cable

| | |
|---|---|
| Connector: | RJ 45 shielded |
| Length: | 1.6 m |



## Dimensions and weight

| | | |
|---|---|---|
| outline drawing number: | type OC2 | 1VL5300617R0101 |
| | type AC2 | 1VL5300617R0102 |
| Weight: | type OC2 | 5.7 kg |
| | type AC2 | 6.7 kg |

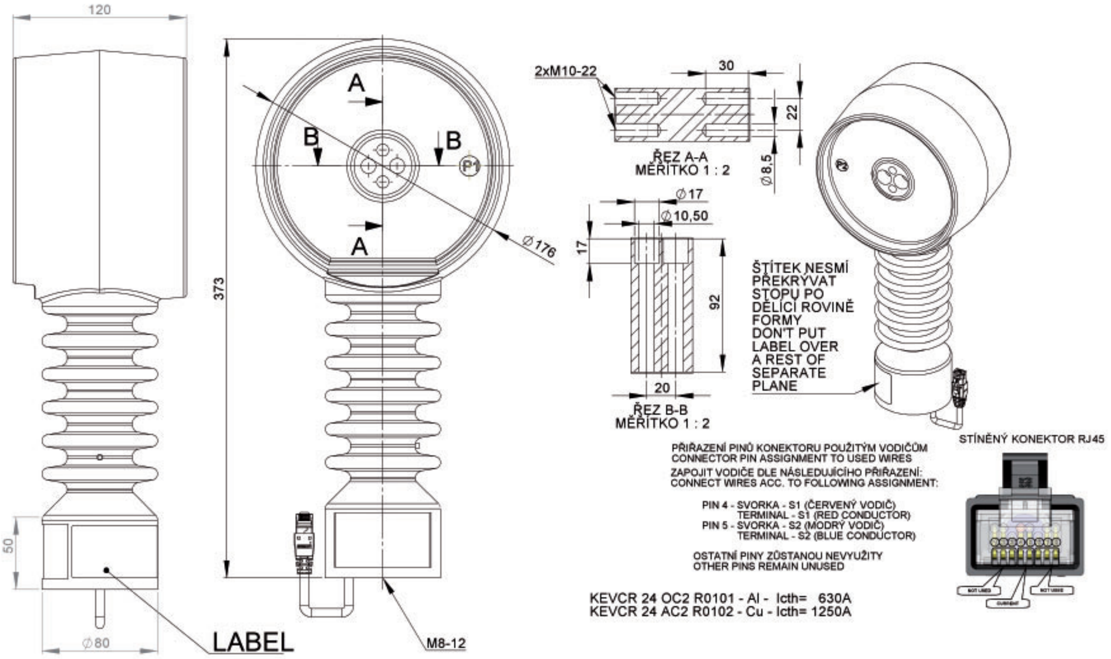
## Ordering data

| | |
|---|---|
| KEVCR 24 AC2: | 1VL5400050V0101 |
| KEVCR 24 OC2: | 1VL5400051V0101 |

# Appendix A. Appendix A - ABB Rogowski coil specifications

**KEVCR 24**

| | |
|---|---|
| Outline drawing number : | OC2 - 1VL5300617R0101<br>AC2 - 1VL5300617R0102 |
| Weigth : | OC2 - 5.7 kg<br>AC2 - 6.7 kg |
| Cable length : | 1.6 m |

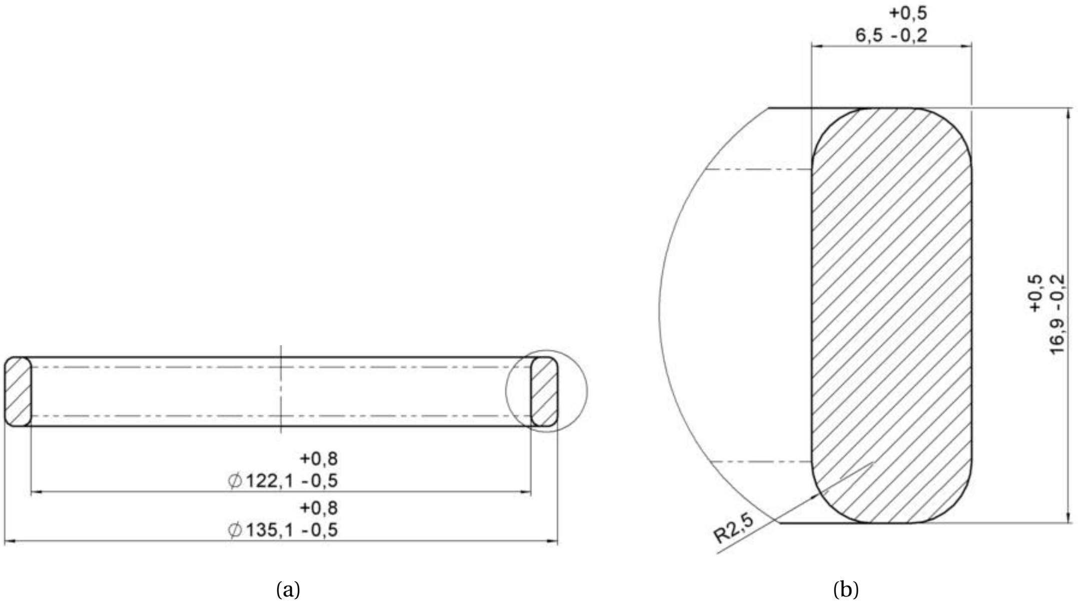

120
373
A
B
B
P1
A
⌀176
50
⌀80
LABEL
M8-12
2xM10-22
30
22
ŘEZ A-A
MĚŘÍTKO 1 : 2
⌀8,5
⌀17
⌀10,50
17
92
20
ŘEZ B-B
MĚŘÍTKO 1 : 2
ŠTÍTEK NESMÍ PŘEKRÝVAT STOPU PO DĚLICÍ ROVINĚ FORMY
DON'T PUT LABEL OVER A REST OF SEPARATE PLANE
STÍNĚNÝ KONEKTOR RJ45
PŘIŘAZENÍ PINŮ KONEKTORU POUŽITÝM VODIČŮM
CONNECTOR PIN ASSIGNMENT TO USED WIRES
ZAPOJIT VODIČE DLE NÁSLEDUJÍCÍHO PŘIŘAZENÍ:
CONNECT WIRES ACC. TO FOLLOWING ASSIGNMENT:
PIN 4 - SVORKA - S1 (ČERVENÝ VODIČ)
TERMINAL - S1 (RED CONDUCTOR)
PIN 5 - SVORKA - S2 (MODRÝ VODIČ)
TERMINAL - S2 (BLUE CONDUCTOR)
OSTATNÍ PINY ZŮSTANOU NEVYUŽITY
OTHER PINS REMAIN UNUSED
NOT USED
CURRENT
NOT USED
KEVCR 24 OC2 R0101 - Al - Icth= 630A
KEVCR 24 AC2 R0102 - Cu - Icth= 1250A

1VLC000585-Rev.-, en 2009.04.17

**ABB s.r.o.**
Videnska 117
619 00 Brno, Czech republic
E-mail: info.ejf@cz.abb.com
Phone.: +420 547 152 602
+420 547 152 604
+420 547 152 631
Fax: +420 547 152 626

The data and ilustrations in this catalogue are not binding. We reserve the right to make changes of the content, in the course of technical development of the product.

## A.4 KECA core dimensions



(a) (b)

Figure A.2: KECA core schematics: (a) KECA core complete cross section. (b) KECA core detailed cross section.

## A.5 KECA parameters

| Name | Value/Expression | Description |
|---|---|---|
| $D1$ | 0.06[m] | Primary conductor (busbar) diameter |
| $D$ | 135.1[mm] | Outer core diameter |
| $d$ | 122.1[mm] | Inner core diameter |
| $h$ | 16.9[mm] | Core height |
| $R$ | 2.5[mm] | Core curvature |
| $N$ | 5440 | Number of loops |
| $W_{ff}$ | 1.908 | Filling factor |
| $epsr$ | 4.8 | Relative dielectric permittivity of the core |
| $TC_{epsr}$ | 0.022[1/K] | Temperature coefficient of EPSR |
| $d_{wire}$ | 0.112[mm] | Copper wire diameter |
| $d_{resin}$ | 0.135[mm] | Cooper and resin diameter |
| $d_{sc}$ | 0.278[mm] | Distance from shield to core |
| $R_l$ | 1.735 $[\Omega/m]$ | Linear coil resistance |
| $C_a$ | 380[pF] | Output capacitance (including cable) |
| $dR_1$ | 0.33 | Loop coefficient |
| $dR_2$ | 0.33 | Loop coefficient |
| $dL_1$ | 0.2 | Loop coefficient |
| $dL_2$ | 0.45 | Loop coefficient |
| $dCs_1$ | 0.45 | Loop coefficient |
| $dCs_2$ | 0.2 | Loop coefficient |
| $dCp_1$ | 0.33 | Loop coefficient |
| $dCp_2$ | 0.33 | Loop coefficient |

Table A.2: KECA parameters. All the values are taken at 25 °C.

## A.6 KECA datasheet

**Medium Voltage Product**

# KECA 250 B1
# Indoor current sensor

| Parameters for Application | Unit | Value |
|---|---|---|
| Rated primary current of application | A | up to 2000 |

| Sensor Parameters | Unit | Value |
|---|---|---|
| Highest voltage for equipment, $U_m$ | kV | 0.72 |
| Rated power frequency withstand voltage | kV | 3 |
| Rated primary current, $I_{pr}$ | A | 250 |
| Rated continuous thermal current, $I_{cth}$ | A | 2000 |
| Rated transformation ratio, $K_{ra}$ for current measurement | - | 250 A /<br>150 mV at 50 Hz<br>180 mV at 60 Hz |
| Current accuracy class | - | 0.5/5P125 |
| Length of cable | m | 5.0 |

## Sensor principles

Electronic Instrument Transformers (Sensors) offer an alternative way of making the current and voltage measurements needed for the protection and monitoring of medium voltage power systems. Sensors based on alternative principles have been introduced as successors to conventional instrument transformers in order to significantly reduce size, increase safety, and to provide greater rating standardization and a wider functionality range. These well known principles can only be fully utilized in combination with versatile electronic relays.

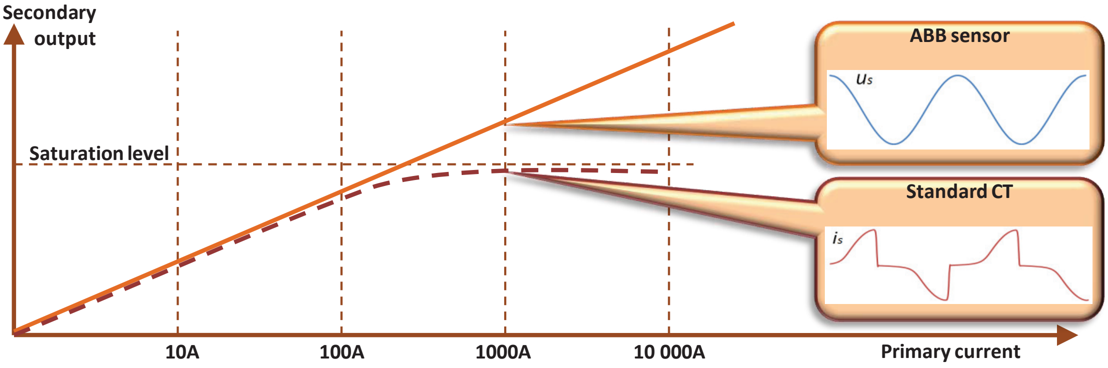
## Sensor characteristics

Construction of ABB's current sensors is done without the use of a ferromagnetic core.

This fact results in several important benefits for the user and the application. The main benefit is that the behavior of the sensor is not influenced by non-linearity and width of hysteresis curve, which results in a highly accurate and linear response over a wide dynamic range of measured quantities.

A linear and highly accurate sensor characteristic in the full operating range enables the combination of metering and protection classes in one winding.

With KECA 250 B1 sensor measuring **class 0.5** is reached for continuous current measurement in the extended accuracy range from 5% of the rated primary current $I_{pr}$ **not only up to 120% of $I_{pr}$** (as being common for conventional current transformers), **but even up to the rated continuous thermal current $I_{cth}$**. For dynamic current measurement (protection purposes) the ABB sensor KECA 250 B1 fulfills requirements of protection class **5P up to an impressive value reaching the rated short-time thermal current $I_{th}$**. That provides the possibility to designate the corresponding accuracy class as **5P125**, proving excellent linearity and accuracy measurements.

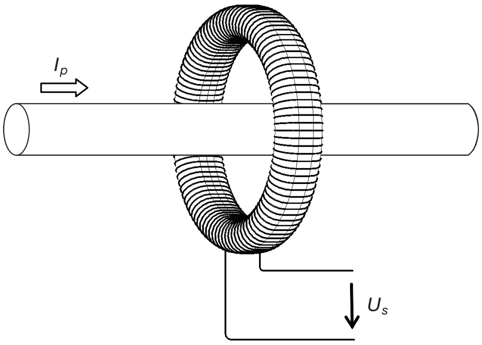


Power and productivity
for a better world™

# Appendix A. Appendix A - ABB Rogowski coil specifications

### Current sensor

Current measurement in KECA 250 B1 sensors is based on the Rogowski coil principle. A Rogowski coil is a toroidal coil, without an iron core, placed around the primary conductor in the same way as the secondary winding in a current transformer. However, the output signal from a Rogowski coil is not a current, but a voltage:



$$u_s(t) = M \frac{di_P(t)}{dt}$$

In all cases, a signal that represents the actual primary current waveform is easily obtained by integrating the transmitted output signal.

### Protection and control IEDs (Intelligent Electronic Devices)

Protection and control IEDs incorporate the functions of a traditional relay, as well as allow new additional functions. The information transmitted from the sensors to the IED is very accurate, providing the possibility of versatile relay functionality.
However, the IED must be able to operate with sufficient accuracy at a sensor’s low input signal level, and the signal from the Rogowski coil must be integrated. Modern IEDs (such as ABB’s 601 series relays) are designed for such sensor use, and they are also equipped with built-in integrators for Rogowski coil sensor inputs.
Modern digital apparatuses (microprocessor based relays) allow protection and measurement functions to be combined. They fully support current sensing realized by the single sensor with double the accuracy class designation (e.g.: current sensing with combined accuracy class 0.5/5P125).

### Sensor applications

The current sensor type KECA 250 B1 is intended for use in current measurement in low voltage or medium voltage switchgear.
In case of medium voltage switchgear the current sensor should be installed over a bushing insulator, insulated cable or any other type of insulated conductor.

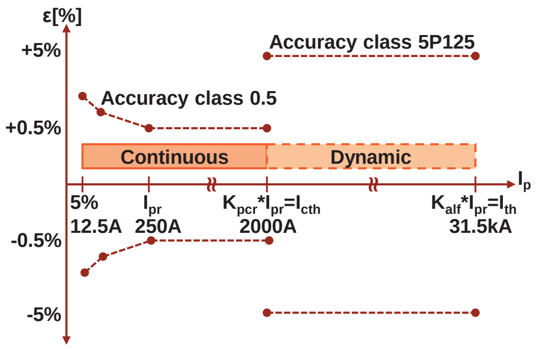
### Differences between Sensors and Instrument Transformers

There are some noticeable differences between Sensors and conventional Instrument Transformers:

### Linearity

Due to the absence of a ferromagnetic core the sensor has a linear response over a very wide primary current range, far exceeding the typical CT range. Thus, current sensing for both measurement and protection purposes could be realized with single secondary winding with a double rating.
In addition, one standard sensor can be used for a broad range of rated currents and is also capable of precisely transferring signals containing frequencies different from rated ones.
For this type of sensor, the variation of amplitude and phase error or composite error in a current range from 5% of rated primary current $I_{pr}$ up to the rated short-time thermal current $I_{th}$ is within the limits specified by IEC 60044-8.

*Example of current measurement range with rated current 250 A and accuracy class 0.5/5P125:*
*Metering accuracy class 0.5 is, according to the IEC 60044-8 standard, guaranteed from 5% of $I_{pr}$ up to $K_{pcr}$ x $I_{pr}$ where $K_{pcr}$ is rated extended primary current factor and $I_{pr}$ is rated primary current. Factor $K_{pcr}$ is in the case of conventional CTs usually just 1.2, but in the case of the KECA 250 B1 sensor the $K_{pcr}$ factor is several times higher and equals 8.*
*Protection accuracy 5P125 is guaranteed, for the advanced KECA 250 B1 sensor, from the current equal to $K_{pcr}$ x $I_{pr}$ up to the current corresponding to $K_{alf}$ x $I_{pr}$ value, where $K_{alf}$ is, according to IEC 60044-8, the accuracy limit factor. For this type of sensor the value of $K_{pcr}$ x $I_{pr}$ is equal to the rated continuous thermal current $I_{cth}$ (2000 A) and the value of $K_{alf}$ x $I_{pr}$ is equal to the rated short-time thermal current $I_{th}$ (31.5 kA).*
*The accuracy limits are described on the graph below.*



### Compactness

Since the sensing elements are particularly small, and the same elements are used for both measurement and protection, the current sensors can be easily integrated into other equipment.

### Rated parameters

Because the sensors are highly linear within a very wide range of currents, the same single sensor can be used for the various rated currents associated with each specific application up to the specified maximum voltage for equipment. There is no need to specify other parameters such as burden, safety factor, etc. since they are standard over the defined range. To achieve the correct function of the protection and control IED, the selected rated current, as well as the rated transformation ratio, must be properly set into the IED.

### Energy savings concept

As there is no iron core, no necessity for high burden values and thus a possibility for low current losses and only one secondary winding needed, KECA 250 B1 sensors exhibit extremely low energy consumption that is just a fraction of that transferred to heat in conventional CTs. This fact contributes to huge energy savings during its entire operating life, supporting the world-wide effort to reduce energy consumption.

### Correction factors

The amplitude and phase error of a current sensor is, in practice, constant and independent of the primary current. Due to this fact it is an inherent and constant property of each sensor and it is not considered as unpredictable and influenced error. Hence, it can be easily corrected in the IED by using appropriate correction factors, stated separately for every sensor.

Values of the correction factors for the amplitude and phase error of a current sensor are mentioned on the sensor label (for more information please refer to Instructions for installation, use and maintenance) and should be uploaded without any modification into the IED before the sensors are put into operation (please check available correction in the IED manual). To achieve required accuracy classes it is recommended to use all correction factors (Cfs): amplitude correction factor (al) and phase error correction factor (pl) of a current sensor.

### Example of a sensor label

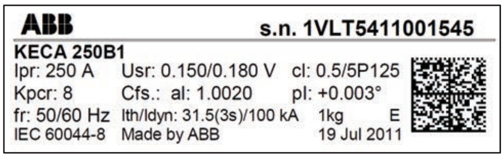
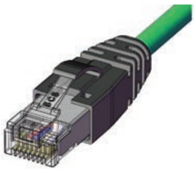


ABB s.n. 1VLT5411001545
KECA 250B1
Ipr: 250 A Usr: 0.150/0.180 V cl: 0.5/5P125
Kpcr: 8 Cfs.: al: 1.0020 pl: +0.003°
fr: 50/60 Hz Ith/Idyn: 31.5(3s)/100 kA 1kg E
IEC 60044-8 Made by ABB 19 Jul 2011

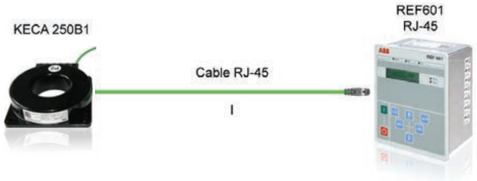
### Secondary cables

The sensor is equipped with a cable for connection with the IED. The cable connector is type RJ-45. The sensor accuracy classes are verified up to the RJ-45 connector, i.e. considering also its secondary cable. This cable is intended to be connected directly to the IED, and subsequently neither burden calculation nor secondary wiring is needed. Every sensor is therefore accuracy tested when equipped with its own cable and connector.

### Connector RJ-45



*Example: Direct connection of connectors between the sensor and new IED family*



### Standards

Current sensors: IEC 60044-8 (2002-07)
Instrument transformers –
Part 8: Electronic current transformers

## Highest voltage for equipment and test voltages

- Highest voltage for equipment, $U_m$: 0.72 kV
- Power frequency voltage withstand test on primary terminals: 3 kV

## Current sensor, rated values

- Rated primary current, $I_{pr}$: 250 A
- Rated transformation ratio, $K_{ra}$: 250 A/0.150 V at 50 Hz<br>250 A/0.180 V at 60 Hz
- Rated secondary output, $U_{sr}$: 3 mV/Hz i.e. 150 mV at 50 Hz or 180 mV at 60 Hz
- Rated continuous thermal current, $I_{cth}$: 2000 A
- Rated short-time thermal current, $I_{th}$: 31.5 kA/3 s
- Rated dynamic current, $I_{dyn}$: 100 kA
- Rated frequency, $f_r$: 50/60 Hz
- Rated extended primary current factor, $K_{pcr}$: 8
- Accuracy limit factor, $K_{alf}$: 125
- Accuracy class: 0.5/5P125
- Rated burden, $R_{br}$: 10 MΩ

## Temperature category

- Operation: - 5°C / + 40°C
- Transport and storage: -40°C / + 70°C

## Cable

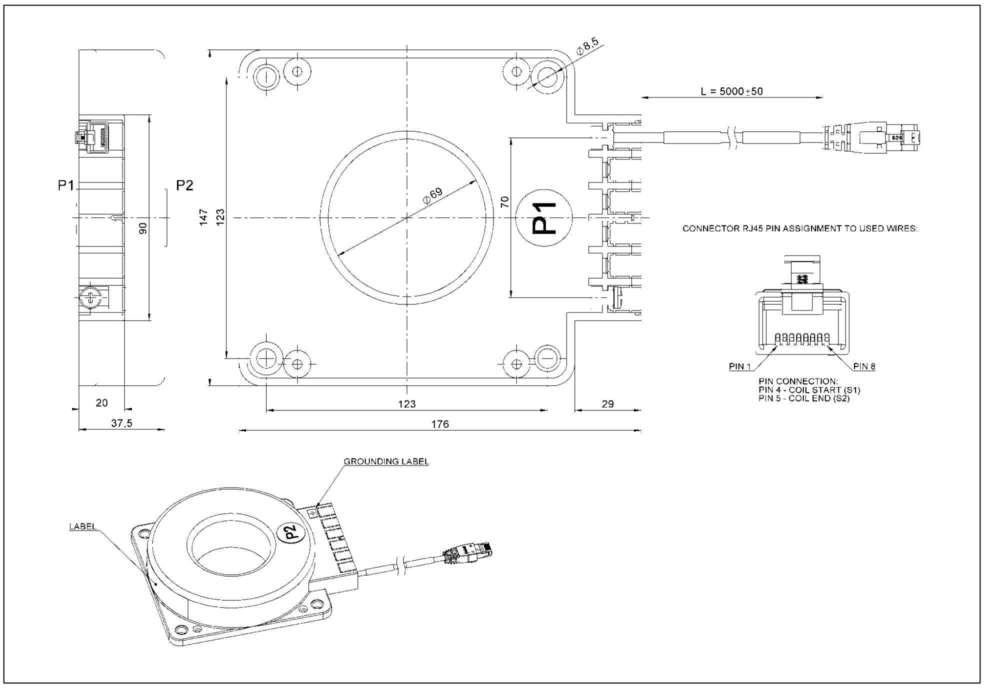
- Length: 5.0 m
- Connector: RJ-45 (CAT-6)

## Ordering data

- KECA 250 B1 1VL5400052V0101

## Dimensions and weight

- Outline drawing number: 1VL5300632R0101
- Weight: 1 kg



1VLC000584 - Rev.2, en, 2011.08

**ABB s.r.o.**
PPMV Brno
Videnska 117
619 00 Brno, Czech Republic
Tel.: +420 547 152 082
+420 547 152 602
Fax: +420 547 152 626
E-mail: info.ejf@cz.abb.com

**www.abb.com**

The data and ilustrations in this catalogue are not binding. We reserve the right to make changes of the content, in the course of technical development of the product.

Power and productivity
for a better world™

# B Appendix B - Model equations

## B.1 Equations of the Rogowski coil electrical model

This section details the equations of the symmetric 3Loops-2Layers electrical model of the RogoCoil. Given the circuit model shown in Figure 4.5 on page 59, and the RogoCoil PMPs shown in Tables A.1 or A.2 in Appendix A, the values of the passive and active elements of the circuit are calculated using the equations here described. Let us consider the frontal and cross-sectional views of the RogoCoil as shown in Figure B.1.

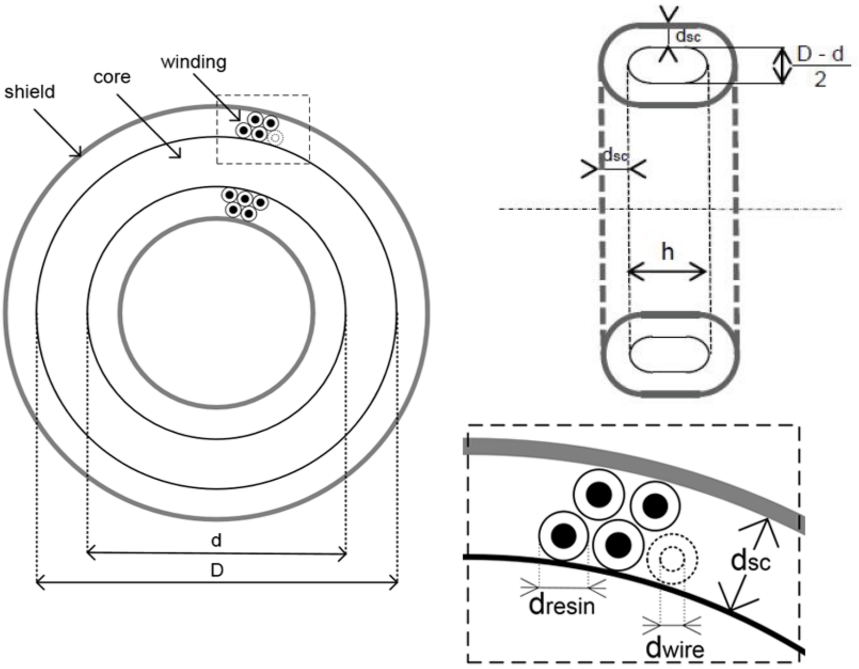


Figure B.1: Rogowski coil technical drawing (not to scale), frontal and cross-sectional views, zoom on winding. `D` represents the outer coil diameter, `d` the inner coil diameter, and `h` the coil height.

The geometrical PMPs `D`, `d`, and `h` require slight corrections caused by mechanical and magnetic effects provoked by the thickness of the winding and the shapes delimited by the cooper cable. The corrections are shown as follows:

Mechanical corrections:

$$D_{ef} = D + 2 \cdot d_{resin} \tag{B.1a}$$

$$d_{ef} = d - 2 \cdot d_{resin} \tag{B.1b}$$

$$h_{ef} = h + 2 \cdot d_{resin} \tag{B.1c}$$

Magnetic corrections:

$$D_{eff} = D + d_{resin} \tag{B.2a}$$

$$d_{eff} = d - d_{resin} \tag{B.2b}$$

$$h_{eff} = h + d_{resin} \tag{B.2c}$$

The SMPs defined in Equations B.1 and B.2 are used for the definition of the following geometrical SMPs:

$$A = h_{eff} \cdot \left( \frac{D_{eff} - d_{eff}}{2} \right) + (D_{eff} - d_{eff})^2 \cdot \left( \frac{\pi - 4}{16} \right) \tag{B.3a}$$

$$l_c = \pi \cdot \left( \frac{D + d}{2} \right) \tag{B.3b}$$

$$l_l = 2 \cdot h_{ef} + (D_{ef} - d_{ef}) + (2\pi - 8) \cdot R \tag{B.3c}$$

$$l_w = l_l \cdot N + (D_{ef} - d_{ef}) \tag{B.3d}$$

where $A$ is the coil area, $l_c$ is the coil length, $l_l$ is the length of one loop of the coil, R is the core curvature, $l_w$ is the wire length, and $N$ is the total number of loops of the coil. It is important mentioning that the KEVCR and KECA coils are double winded, meaning that the return wire (whose length is included in the term $l_l \cdot N$), is also looped around the core. In this way, the coil resistance $R_{coil}$ and the coil mutual inductance $M$ are defined in terms of the geometrical and material parameters of the coil winding by Equations B.4 and B.5 respectively.

$$R_{coil} = l_w \cdot R_l \cdot (1 + TC_R \cdot (T_{coil} - T_0)) \tag{B.4}$$

$$\begin{aligned} M = (1 - TC_L(T_{coil} - T_0)) \cdot \frac{\mu_0 N}{2\pi} \Bigg[ & (h_{ef} - 2R) Ln\left(\frac{D_{ef}}{d_{ef}}\right) + 2R \cdot Ln\left(\frac{D_{ef} - 2R}{d_{ef} + 2R}\right) \\ & + \pi \frac{D_{ef} + d_{ef}}{2} - \left(\pi + 2\arcsin\left(\frac{R}{R - \frac{D_{ef}}{2}}\right)\right) \sqrt{\frac{D_{ef}}{2}\left(\frac{D_{ef}}{2} - 2R\right)} \\ & - \left(\pi + 2\arcsin\left(\frac{R}{R + \frac{d_{ef}}{2}}\right)\right) \sqrt{\frac{d_{ef}}{2}\left(\frac{d_{ef}}{2} + 2R\right)} \Bigg] \end{aligned} \tag{B.5}$$

Equation B.4 shows that $R_{coil}$ is directly proportional to the wire length $l_w$, the linear wire resistance $R_l$, and the temperature coefficient of resistance of the cooper wire $TC_R$. In this Equation $T_{coil}$ represents the temperature of the coil and $T_0$ its reference temperature (25 °C).

In Equation B.5 we can recognize the magnetic constant $\mu_0$ and the temperature coefficient of inductance $TC_L$ which is used to model the change of the coil inductance based on a linear thermal expansion of the material. The logarithmic and hyperbolic terms correspond to the oval shape of the coil loops and the wire geometry. Consequently, the coil inductance is given by Equation B.6 as follows:

$$L_{coil} = N \cdot M \tag{B.6}$$

Likewise, the parasitic capacitances of the electrical model are also given in terms of geometrical and material parameters as follows:

$$C_l = \frac{2\pi\epsilon_0 epsr \cdot l_c}{Ln\left(\sqrt{\frac{\left(\frac{D-d}{2}+2d_{wire}\right)(h+2d_{wire})}{\left(\frac{D-d}{2}\right)h}}\right)} \cdot (1 - TC_{eprs} \cdot (T_{coil} - T_0))$$

$$+ \frac{2\pi\epsilon_0 l_c}{Ln\left(\sqrt{\frac{\left(\frac{D-d}{2}+2(d_{wire}+d_{resin})\right)(h+2(d_{wire}+d_{resin}))}{\left(\frac{D-d}{2}+2d_{wire}\right)(h+2d_{wire})}}\right)} \tag{B.7a}$$

$$C_{loop} = \frac{2\pi\epsilon_0 l_l}{Ln\left(\frac{l_c}{\frac{N}{2}d_{wire}} + \sqrt{\left(\frac{l_c}{\frac{N}{2}d_{wire}}\right)^2 - 1}\right)} \tag{B.7b}$$

$$C_{sh} = \frac{2\pi\epsilon_0 l_c}{Ln\left(\sqrt{\frac{\left(\frac{D-d}{2}+2d_{sc}\right)(h+2d_{sc})}{\left(\frac{D-d}{2}+2(d_{wire}+d_{resin})\right)(h+2(d_{wire}+d_{resin}))}}\right)} \tag{B.7c}$$

$$C_p = \frac{4\pi \cdot \epsilon_0 \cdot h_{ef}}{Ln\left(\frac{2 \cdot d_{ef}}{D1}\right)} \tag{B.7d}$$

$$C_s = C_l + C_{sh} \tag{B.7e}$$

where $C_l$ is the winding capacitance, $C_{loop}$ is the single loop capacitance, $C_{sh}$ is the shielding capacitance, $C_p$ is the primary capacitance, and $C_s$ is the secondary capacitance of the coil.

Finally, is important to mention that the corrective terms and the parasitic coefficients ($dR$, $dL$, $dCp$, and $dCs$) of the model, are characterization values obtained for fitting the experimental impedance measurements of the KEVCR and KECA coils, see the respective values in Tables A.1 and A.2 on pages 168 and 174 respectively.

## B.2 Principles of conductor temperature determination

### B.2.1 Heat balance

The conductor temperature is based on the heat balance at the conductor [83], which is influenced by:

- Joule heat $Q_J$, due to the current.
- Solar radiation $Q_S$
- Magnetic Losses $Q_M$
- Energy Loss by convection $Q_C$
- Energy Loss by radiation $Q_R$

From these values, the heat balance is described by:

$$m_c \cdot C_p \cdot \frac{dT}{dt} = Q_J + Q_S + Q_M - Q_R - Q_C \tag{B.8}$$

where $m_c$ is the conductor's mass per unit length, $C_p$ is the specific heat capacity of the conductor at constant pressure, and $T$ is the conductor's temperature. The components $Q_J$ and $Q_M$, which are function of the current, can be expressed as follows:

$$Q_J + Q_M = I^2 \cdot R_{AC}(T) \tag{B.9}$$

where $I$ is the effective current in the conductor (in Amperes) and $R_{AC}(T)$ is the AC resistance at temperature $T$ in $\Omega$/m. The AC resistance increases as a result of the skin and spiral effects. The skin effect occurs due to the higher inductance of the internal layers of wires in the conductor, causing a larger internal current flux density. As the voltage drop in all wires is the same, a larger portion of the current flows through the outer conductor layers causing an increase of the effective resistance. On the other hand, the spiral effect significantly influences composite conductors with odd number of layers, in particular where there is only one aluminum layer. $R_{AC}(T)$ is calculated as follows:

$$R_{AC}(T) = \begin{cases} R_{DC}(T) \cdot \left(1 + \frac{\chi^4}{3}\right) \text{ for } \chi \leq 1 \\ R_{DC}(T) \cdot \left(0.25 + \chi + \frac{3}{64}\right) \text{ for } \chi \geq 1 \end{cases} \tag{B.10}$$

with $\chi$ being the parameter related with the skin and spiral effects as follows:

$$\chi = 0.5r\sqrt{\pi \cdot f \cdot \kappa \cdot \mu_0 \cdot \mu_r} \tag{B.11}$$

where $r$ is the conductor radius in *mm*, $f$ the frequency in *Hz*, $\kappa$ the conductivity in m/$\Omega \cdot mm^2$, $\mu_0$ is the vacuum permeability constant and $\mu_r$ the relative permeability of the material. The DC resistance $R_{DC}(T)$ depends on the conductor temperature according to:

$$R_{DC}(T) = R_{20} \cdot [1 + \alpha \cdot (T - 20)] \tag{B.12}$$

where $R_{20}$ is the DC resistance at 20 °C in $\Omega$/m and $\alpha$ is the temperature coefficient of resistance.

The solar radiation thermal contribution is here discarded due to the RogoCoil sensor is designed for indoor applications, i.e. $Q_S = 0$.

The following equation is taken for the energy loss by radiation:

$$Q_R = k_s \cdot k_e \cdot D \cdot \pi \left(T^4 - T_{room}^4\right) [W/m] \tag{B.13}$$

where $D$ is the conductor diameter, $T_{room}$ is the room temperature, $k_s$ is the Stefan-Boltzmann constant (5.67e-8 W/$m^2K^4$) and $k_e$ is the emission coefficient.

The energy loss by convection can be calculated from:

$$Q_C = \pi \cdot \lambda \cdot Nu \cdot (T - T_{room}) \, [W/m] \tag{B.14}$$

where $\lambda$ is the thermal conductivity of air in W/$(K \cdot m)$ and $Nu$ is the Nusselt number, which depends on Reynolds number ($Re$) using the following approximation:

$$Nu \cong 0.65 \cdot Re^{0.2} + 0.23 \cdot Re^{0.61} \tag{B.15a}$$

$$Re = V \cdot D \cdot \frac{\gamma}{\eta} \tag{B.15b}$$

where $V$ is the wind velocity in m/s, $\gamma$ is the specific mass of air in kg/$m^3$, and $\eta$ is the dynamic viscosity in $N \cdot s/m^2$. All these values are dependent on temperature and air pressure. At sea level, the following equations were obtained by curve fitting on experimental data [83]:

$$\lambda = 7.327 \cdot 10^{-5} \cdot T_{room} + 0.02428 \, [W/(K \cdot m)] \tag{B.16a}$$

$$\gamma = \frac{354.7}{T_{room} + 274.5} \left[kg/m^3\right] \tag{B.16b}$$

$$\eta = 4.7 \cdot 10^{-4} \cdot T_{room} + 0.175 \left[(N \cdot s)/m^2\right] \tag{B.16c}$$

The previously mentioned parameters can also be replaced by other conservative or extreme values according to known conditions. However, the described deterministic approach usually leads to conservative values for the current carrying capacity (also called Ampacity), since conditions having low probability of occurrence are assumed as acting simultaneously. Measurements of the conductor temperatures have systematically shown lower temperatures in steady-state than the ones determined by this deterministic approach [83]. Nevertheless, the model can be corrected by including a correction factor in the Joule heating calculation, see Equation B.9.

### B.2.2 IEEE STANDARD 738 – 2006

In this revision all environmental and parameter conditions were extensively revised. This model is basically a simplified version obtained from many other methods such as the previously presented in Appendix B.2.1. All of them based in the same principle, the heat balance equation. Accurate results can be obtained by this model always than the real conditions are closer to the assumed ones. The temperature of a power conductor is continuously varying in response to changes in electrical current and environmental conditions. In this standard, however, environment parameters (wind speed, room temperature, etc.) are assumed to remain constant.

A step transition of the electrical current will produce an exponential transient temperature change in the conductor. Immediately prior to the current step change (t = $0^-$), the conductor is assumed to be in thermal equilibrium ($\frac{dT}{dt} = 0$), i.e. the sum of heat generation by Ohmic losses and solar heating equals the heat loss by convection and radiation. Immediately after the current step change (t = $0^+$), the conductor temperature is unchanged (as are the conductor resistance and the heat loss rate due to convection and radiation), but the rate of heat generation due to Ohmic losses has increased. Therefore, at time t = $0^+$, the temperature of the conductor begins to increase at a rate given by the heat balance equation B.8.

After a period of time, the increased conductor temperature yields higher heat losses due to convection, radiation, and somewhat higher Ohmic heat generation due to the increased conductor resistance. The conductor temperature continues increasing with time, but does so at a lower rate until achieving its final steady-state temperature. The accuracy in the iterative transient calculation requires a simulation time step sufficiently small with respect to the thermal time constant. It is always prudent to rerun the calculation with a smaller time step to check whether the calculated values change [84].

Contrarily to the expressions given in Appendix B.2.1, this standard use the following heuristic expressions which leads to more accurate simulation results. The Radiated heat loss is defined by:

$$Q_R = 0.0178 \cdot D \cdot \epsilon \left( \left( \frac{T + 273.15}{100} \right)^4 - \left( \frac{T_{room} + 273.15}{100} \right)^4 \right) [W/m] \tag{B.17}$$

where $D$ is the conductor diameter in $mm$, $\epsilon$ is the emissivity coefficient (0.23 to 0.91), and $T$ and $T_{room}$ are given in °C. With zero wind speed, as we can suppose for indoor applications, natural convection occurs. The energy loss by convection is calculated as follows:

$$Q_C = 0.0205 \cdot \rho_f^{0.5} \cdot D^{0.75} \cdot (T - T_{room})^{1.25} \, [W/m] \tag{B.18}$$

where $\rho_f$ is the air density in kg/$m^3$. As it is mentioned in [84], it has been argued that at low wind speeds, the convection cooling rate should be calculated by using a vector sum of the wind speed and a natural wind speed. However, it is recommended that only the larger of the forced and natural convection heat loss rates be used at low wind speeds instead of their vector sum as this is conservative. For indoor applications, we do not need to taken into account the forced convention in the thermal conductor model since there is no significant wind flow forcing the thermal convection.

Finally, the Joule heating thermal contribution in Equation B.8 is depending on the conductor electrical resistance. The electrical resistance of a bare stranded conductor varies with frequency, average current density and temperature. However, it also strongly depends of the conductor construction. This is why, the best tabulated values of electrical resistance for using in our model are the ones given by the conductor manufacturer under standard conditions. This values should include the frequency-dependent skin effect and other magnetic effects affecting the conductor. In this standard, electrical resistance $R_{AC}(T)$ is calculated solely as a function of the conductor temperature; however, the resistance values entered may be function of frequency and current density. The conductor resistance at any other temperature

$T$, is found by linear interpolation according to Equation B.19.

$$R_{AC}(T) = \left( \frac{R(T_{high}) - R(T_{low})}{T_{high} - T_{low}} \right) \cdot (T - T_{low}) + R(T_{low}) \, [\Omega/m] \tag{B.19}$$

where $R(T_{high})$ and $R(T_{low})$ are the electrical resistances at high and low temperature respectively, usually $T_{low} = 25$ °Cand $T_{high} = 75$ °C. This method of resistance calculation allows to calculate the high and low temperatures resistance values by whatever means is appropriate. The error obtained by Equation B.19 between $T_{high}$ and $T_{low}$ is negligible. For higher temperatures, although the resistance calculation could be non-conservative for rating calculations, the resistance error will be low enough to be used, always that the conductor operates under its maximum operational temperature.

### B.2.3 Equivalent lumped-element circuit model of a Busbar conductor

Following the equations and nomenclature given in Appendices B.2.1 and B.2.2, the expressions for the lumped elements of the equivalent Busbar thermal circuit are given as follows:

$$U_{tr}(t) = I^2(t) \cdot R_{AC} + Q_S \tag{B.20a}$$

$$C_{tr}(t) = m_c \cdot C_p \tag{B.20b}$$

$$R_{tr}(t) = \frac{T_{bus}(t) - T_{room}(t)}{Q_C + Q_R} \tag{B.20c}$$

# C Appendix C - Source code

## C.1 Sigma-Delta ADC

```vhdl
Library IEEE;
use IEEE.electrical_systems.all;
use IEEE.std_logic_1164.all;
use IEEE.math_real.all;
use IEEE.numeric_std.all;
Entity ADC is
    generic(
    -- Ideal Architecture:
       VLOW    :    voltage   := -1.0;  -- Low threshold voltage [V]
       VHIGH   :    voltage   := 1.0;   -- High threshold voltage [V]
       NB      :    positive  := 12;    -- ADC Bit resolution (ideal) /
    Bus Width (Sigma-Delta)
    -- Delta-Sigma:
       MFCLK         : real     := 1.0e5;  --[Hz] Sampling frequency of the
     ADC(Modulator)
       M_DSIG        : positive := 32;     --[] Decimation Ratio;
       K1_ADD1       : real     := 1.0;    --[] First Adder Gain Factor
       K2_ADD1       : real     := -1.0;   --[] First Adder Gain Factor
       GINT_INT1     : real     := 1.0;    --[] First Integrator Gain
       K1_ADD2       : real     := 1.0;    --[] Second Adder Gain Factor
       K2_ADD2       : real     := -1.0;   --[] Second Adder Gain Factor
       GINT_INT2     : real     := 1.0;    --[] Second Integrator Gain
       TR_QUANT      : real     := 0.0;    --[Sec] Output transition (rise/
    fall) time of the Quantizer
       VREF_DSIG     : voltage := 0.0 );  --[V] Reference voltage
    port( terminal Input      :   electrical;
          signal Enable, Clk  :   in  std_logic;
          signal Output       :   out std_logic_vector(NB-1 downto 0) );
Begin
  assert VLOW<=VHIGH report "VHIGH must be greater than VLOW" severity
    error;
    assert VREF_DSIG<=VHIGH report "VHIGH must be greater than VREF_DSIG" severity error;
    assert VLOW<=VREF_DSIG report "VREF_DSIG must be greater than VLOW" severity error;
End entity ADC;

Architecture Sigma_Delta of ADC is
  constant DELTA_DSIG   : voltage := VHIGH - VLOW; --[V] Quantizer step
  component Delta_Sigma_Mod is
    generic ( MFCLK        : real;
              K1_ADD1      : real;
              K2_ADD1      : real;
              GINT_INT1    : real;
              K1_ADD2      : real;
              K2_ADD2      : real;
              GINT_INT2    : real;
              TR_QUANT     : real;
              VREF         : voltage;
              DELTA        : voltage );
    port ( signal Enable, MClk  : in  std_logic;
           signal Sout          : inout std_logic;
           terminal Tin, Tout   : electrical );
  end component Delta_Sigma_Mod;
  component Sinc3 is
    generic ( N : positive);
    port( RESN, MOUT, MCLK, CNR : in std_logic;
          CN5 : out std_logic_vector(N-1 downto 0) );
  end component Sinc3;
  signal DSIG_Sout : std_logic;
  signal DClk      : std_logic := '0'; -- Decimator clock
  signal cnt : integer := 0;
  terminal Tdsout : electrical;
Begin
   DSMOD: Delta_Sigma_Mod
           generic map (  MFCLK       =>  MFCLK,
                          K1_ADD1     =>  K1_ADD1,
                          K2_ADD1     =>  K2_ADD1,
                          GINT_INT1   =>  GINT_INT1,
                          K1_ADD2     =>  K1_ADD2,
                          K2_ADD2     =>  K2_ADD2,
                          GINT_INT2   =>  GINT_INT2,
                          TR_QUANT    =>  TR_QUANT,
                          VREF        =>  VREF_DSIG,
                          DELTA       =>  DELTA_DSIG )
           port map ( Enable => Enable,
                      MClk  => Clk,
                      Sout  => DSIG_Sout,
                      Tin   => Input,
                      Tout  => Tdsout );
   FILTER: Sinc3
    generic map ( N => NB)
    port map ( RESN  => Enable,
```

```vhdl
                  MOUT   => DSIG_Sout,
                  MCLK   => Clk,
                  CNR    => DClk,
                  CN5    => Output  );
  -- Decimator Clock:
    Process(Clk, Enable)
      begin
      if Enable = '0' then
         cnt <= 0;
         DClk <= '0';
      else
         if cnt = M_DSIG - 1 then
            cnt <= 0;
            DClk <= not DClk;
         else
            cnt <= cnt + 1;
         end if;
      end if;
    end process;
-- Parameter display:
   process
   begin
     report LF & "---------------------------------" &
            LF & "    ADC" &
            LF & "---------------------------------" &
            LF &" "&
            LF & "### VLOW" & " = " & real'image(VLOW) & " " & "V" &
            LF & "### VHIGH" & " = " & real'image(VHIGH) & " " & "V" &
            LF & "### NB" & " = " & integer'image(NB) &
            LF & "### MFCLK" & " = " & real'image(MFCLK) & " " & "Hz" &
            LF & "### M_DSIG" & " = " & integer'image(M_DSIG) &
            LF & "---------------------------------";
      wait;
   end process;
End architecture Sigma_Delta;
```

### C.1.1 Simple $1^{st}$ Order ΣΔ configuration

```vhdl
Library IEEE;
use WORK.all;
Configuration Simple_1stOrderSDADC of ADC is
    for Sigma_Delta
        for all: Delta_Sigma_Mod
            use entity work.Delta_Sigma_Mod(struct_first_order);
            for struct_first_order
                for all: adder
                    use entity work.adder(simple);
                end for;
                for all: Integrator
```

```
                    use entity work.Integrator(ideal_discrete_time);
                end for;
                for all: Quantizer_behav
                    use entity work.Quantizer_behav(ideal);
                end for;
                for all: DAC_behav
                    use entity work.DAC_behav(ideal);
                end for;
            end for;
        end for;
    end for;
End Simple_1stOrderSDADC;
```

### C.1.2 Simple $2^{nd}$ Order $\Sigma\Delta$ configuration

```
Library IEEE;
use WORK.all;
Configuration Simple_2ndOrderSDADC of ADC is
    for Sigma_Delta
        for all: Delta_Sigma_Mod
            use entity work.Delta_Sigma_Mod(struct_second_order);
            for struct_second_order
                for all: adder
                    use entity work.adder(simple);
                end for;
                for all: Integrator
                    use entity work.Integrator(ideal_discrete_time);
                end for;
                for all: Quantizer_behav
                    use entity work.Quantizer_behav(ideal);
                end for;
                for all: DAC_behav
                    use entity work.DAC_behav(ideal);
                end for;
            end for;
        end for;
    end for;
End Simple_2ndOrderSDADC;
```

### C.1.3 Sigma-Delta modulator

#### C.1.3.1 First Order modulator

```
Library IEEE;
use IEEE.electrical_systems.all;
use IEEE.std_logic_1164.all;
```

```vhdl
use work.all;
Entity Delta_Sigma_Mod is
  generic ( MFCLK     : real    := 1.0e5;  --[Hz] Sampling frequency of the Modulator
            K1_ADD1   : real    := 1.0;    --[] First Adder Gain Factor
            K2_ADD1   : real    := -1.0;   --[] First Adder Gain Factor
            GINT_INT1 : real    := 1.0;    --[] First Integrator Gain
            K1_ADD2   : real    := 1.0;    --[] Second Adder Gain Factor
            K2_ADD2   : real    := -1.0;   --[] Second Adder Gain Factor
            GINT_INT2 : real    := 1.0;    --[] Second Integrator Gain
            TR_QUANT  : real    := 0.0;    --[Sec] Output transition (rise/fall) time of the Quantizer
            VREF      : voltage := 0.0;    --[V] Reference voltage
            DELTA     : voltage := 5.0 );  --[V] Quantizer step
  port ( signal Enable, MClk  : in   std_logic;
         signal Sout          : inout std_logic;
         terminal Tin, Tout   : electrical );
End entity Delta_Sigma_Mod;

Architecture struct_first_order of Delta_Sigma_Mod is
  Component adder is
     generic ( K1 : real;
               K2 : real );
     port ( terminal inp1 : electrical;
            terminal inp2 : electrical;
            terminal aout : electrical );
  End component adder;
  Component Integrator is
     generic ( GINT  : real;
               VINIT : voltage;
               FSMP  : real );
     port    ( signal Enable : in std_logic;
               terminal Tin, Tout : electrical );
  End component Integrator;
  Component Quantizer_behav is
    generic ( TR           : real;
              VREF         : voltage;
              DELTA        : voltage );
    port (  signal Enable, Clk  : in  std_logic;
            signal Sout         : out std_logic;
            terminal Tin, Tout  : electrical );
  End component Quantizer_behav;
  Component DAC_behav is
    generic ( VREF         : voltage;
              DELTA        : voltage  );
    port (  signal Enable, Sin  : in std_logic;
            terminal Tout    : electrical );
  End component DAC_behav;
  terminal Tfeed, T1, T2 : electrical;
 Begin
   ADD: adder
    generic map (  K1   =>  K1_ADD1,
                   K2   =>   K2_ADD1 )
    port map ( inp1  => Tin,
               inp2  => Tfeed,
               aout  => T1 );
   INT: Integrator
    generic map ( GINT  =>  GINT_INT1,
                  VINIT => VREF,
                  FSMP => MFCLK )
    port map ( Enable => Enable,
               Tin  => T1,
               Tout  => T2 );
   QUANT: Quantizer_behav
    generic map (  TR     =>  TR_QUANT,
                   VREF   =>  VREF,
                   DELTA  =>  DELTA )
    port map ( Enable  => Enable,
               Clk     => MClk,
               Sout    => Sout,
               Tin     => T2,
               Tout    => Tout );
   DAC: DAC_behav
    generic map (  VREF   =>  VREF,
                   DELTA  =>  DELTA )
    port map ( Enable  => Enable,
               Sin     => Sout,
               Tout    => Tfeed );
-- Parameter display:
   process
   begin
    report LF & "---------------------------------" &
           LF & "First Order Sigma-Delta Modulator" &
           LF & "---------------------------------" &
           LF &" "&
           LF & "### MFCLK" & " = " & real'image(MFCLK) & " " & "Hz" &
           LF & "### K1_ADD1" & " = " & real'image(K1_ADD1) &
           LF & "### K2_ADD1" & " = " & real'image(K2_ADD1) &
           LF & "### GINT_INT1" & " = " & real'image(GINT_INT1) &
      LF & "### TR_QUANT" & " = " & real'image(TR_QUANT) & " " & "Sec" &
           LF & "### VREF" & " = " & real'image(VREF) & " " & "V" &
           LF & "### DELTA" & " = " & real'image(DELTA) & " " & "V" &
           LF & "---------------------------------";
     wait;
   end process;
End struct_first_order;
```

### C.1.3.2 Second Order modulator

```
Library IEEE;
use IEEE.electrical_systems.all;
```

```vhdl
use IEEE.std_logic_1164.all;
use work.all;
Entity Delta_Sigma_Mod is
  generic ( MFCLK     : real    := 1.0e5; --[Hz] Sampling frequency of the Modulator
            K1_ADD1   : real    := 1.0;    --[] First Adder Gain Factor
            K2_ADD1   : real    := -1.0;   --[] First Adder Gain Factor
            GINT_INT1 : real    := 1.0;    --[] First Integrator Gain
            K1_ADD2   : real    := 1.0;    --[] Second Adder Gain Factor
            K2_ADD2   : real    := -1.0;   --[] Second Adder Gain Factor
            GINT_INT2 : real    := 1.0;    --[] Second Integrator Gain
            TR_QUANT  : real    := 0.0;    --[Sec] Output transition (rise/fall) time of the Quantizer
            VREF      : voltage := 0.0;    --[V] Reference voltage
            DELTA     : voltage := 5.0     --[V] Quantizer step
          );
  port ( signal Enable, MClk  : in  std_logic;
         signal Sout          : inout std_logic;
         terminal Tin, Tout   : electrical );
End entity Delta_Sigma_Mod;

Architecture struct_second_order of Delta_Sigma_Mod is
  Component adder is
     generic ( K1 : real;
               K2 : real );
     port ( terminal inp1 : electrical;
            terminal inp2 : electrical;
            terminal aout : electrical );
  End component adder;
  Component Integrator is
     generic ( GINT  : real;
               VINIT : voltage;
               FSMP  : real );
     port    ( signal Enable : in std_logic;
               terminal Tin, Tout : electrical );
  End component Integrator;
  Component Quantizer_behav is
    generic ( TR          : real;
              VREF        : voltage;
              DELTA       : voltage );
    port (  signal Enable, Clk  : in  std_logic;
            signal Sout         : out std_logic;
            terminal Tin, Tout  : electrical );
  End component Quantizer_behav;
  Component DAC_behav is
    generic ( VREF        : voltage;
              DELTA       : voltage  );
    port (  signal Enable, Sin  : in std_logic;
            terminal Tout    : electrical );
  End component DAC_behav;
  terminal Tfeed, T1, T2, T3, T4 : electrical;
 Begin
```

```
   ADD1: adder
    generic map (  K1   =>  K1_ADD1,
                   K2   =>  K2_ADD1 )
    port map ( inp1  => Tin,
               inp2  => Tfeed,
               aout  => T1 );
   INT1: Integrator
    generic map ( GINT  =>  GINT_INT1,
                  VINIT => VREF,
                  FSMP => MFCLK )
    port map ( Enable => Enable,
               Tin   => T1,
               Tout  => T2 );
   ADD2: adder
    generic map (  K1   =>  K1_ADD2,
                   K2   =>  K2_ADD2 )
    port map ( inp1  => T2,
               inp2  => Tfeed,
               aout  => T3 );
   INT2: Integrator
    generic map ( GINT  =>  GINT_INT2,
                  VINIT => VREF,
                  FSMP => MFCLK )
    port map ( Enable => Enable,
               Tin   => T3,
               Tout  => T4 );
   QUANT: Quantizer_behav
    generic map (  TR     =>  TR_QUANT,
                   VREF   =>  VREF,
                   DELTA  =>  DELTA )
    port map ( Enable   => Enable,
               Clk      => MClk,
               Sout     => Sout,
               Tin      => T4,
               Tout     => Tout );
   DAC: DAC_behav
    generic map (  VREF   =>  VREF,
                   DELTA  =>  DELTA )
    port map ( Enable   => Enable,
               Sin      => Sout,
               Tout     => Tfeed );
-- Parameter display:
   process
   begin
    report LF & "---------------------------------" &
        LF & "Second Order Sigma-Delta Modulator" &
        LF & "---------------------------------" &
        LF &" "&
        LF & "### MFCLK" & " = " & real'image(MFCLK) & " " & "Hz" &
        LF & "### K1_ADD1" & " = " & real'image(K1_ADD1) &
        LF & "### K2_ADD1" & " = " & real'image(K2_ADD1) &
        LF & "### GINT_INT1" & " = " & real'image(GINT_INT1) &
        LF & "### K1_ADD2" & " = " & real'image(K1_ADD2) &
        LF & "### K2_ADD2" & " = " & real'image(K2_ADD2) &
        LF & "### GINT_INT2" & " = " & real'image(GINT_INT2) &
       LF & "### TR_QUANT" & " = " & real'image(TR_QUANT) & " " & "Sec" &
        LF & "### VREF" & " = " & real'image(VREF) & " " & "V" &
        LF & "### DELTA" & " = " & real'image(DELTA) & " " & "V" &
        LF & "---------------------------------";
    wait;
   end process;
End struct_second_order;
```

#### C.1.3.3 Integrator

```
Library IEEE;
  use IEEE.electrical_systems.all;
  use IEEE.std_logic_1164.all;
  use IEEE.math_real.all;
Entity Integrator is
   generic ( GINT  : real    := 1.0;  --[] Integrator gain
             VINIT : voltage := 0.0;  --[V] Initial output voltage. Make
    it equal to VREF for Delta-Sigma correct initial/disable state
             FSMP  : real    := 1.0 ); -- [Hz] Sample frequency
   port    ( signal Enable : in std_logic;
             terminal Tin, Tout : electrical );
End entity Integrator;

Architecture ideal_discrete_time of Integrator is
  quantity Vin across Tin;
  quantity Vout across Iout through Tout;
  quantity vin_sampled : real; -- discrete sample of input quantity
  quantity vin_zm1, vout_zm1 : real; -- z^-1
  constant TSMP : real := 1.0/FSMP; -- Sample period
  constant N0 : real := 0.0; -- Z0 numerator coefficient
  constant N1 : real := 1.0; -- Z^-1 numerator coefficient
  constant D0 : real := 1.0; -- Z0 denominator coefficient
  constant D1 : real := -1.0; -- Z^-1 denominator coefficient
Begin
   if domain = quiescent_domain or Enable = '0' use
      Vout == VINIT;
      vin_sampled == 0.0;
      vin_zm1 == 0.0;
      vout_zm1 == 0.0;
   else
     vin_sampled == Vin'zoh(TSMP);
     vin_zm1 == vin_sampled'delayed(TSMP);
     vout_zm1 == Vout'delayed(TSMP);
     Vout == vin_sampled*N0/D0 + N1*vin_zm1/D0 - D1*vout_zm1/D0;
   end use;
   break on Enable;
```

```
End architecture ideal_discrete_time;
```

### C.1.3.4 Adder

```
Library IEEE;
  use IEEE.electrical_systems.all;
Entity adder is
  generic ( K1 : real := 1.0;     -- First input gain factor
            K2 : real := 1.0 );  -- Second input gain factor
     port ( terminal inp1 : electrical;      -- First Input
            terminal inp2 : electrical;      -- Second Input
            terminal aout : electrical);     -- Output
End entity adder;

Architecture simple of adder is
   quantity vout across iout through aout;
   quantity v1 across inp1;
   quantity v2 across inp2;
 Begin
  vout == K1* v1 + K2* v2;
End simple;
```

### C.1.3.5 Quantizer

```
Library IEEE;
  use IEEE.electrical_systems.all;
  use IEEE.std_logic_1164.all;
Entity Quantizer_behav is
  generic ( TR      : real    := 0.0;    --[Sec] Output transition (rise/
    fall) time
            VREF    : voltage := 2.5;    --[V] Reference voltage
            DELTA   : voltage := 5.0 ); --[V] Quantizer step
  port (    signal Enable, Clk  : in  std_logic;
            signal Sout         : out std_logic;
            terminal Tin, Tout  : electrical );
  begin
   assert DELTA >= 0.0
     report "Quantizer step must be positive" severity error;
End entity Quantizer_behav;

Architecture ideal of Quantizer_behav is
  quantity vin across Tin;
  quantity vout across iout through Tout;
  signal level   : real := VREF;   -- Quantizer output
 Begin
  process(Clk)
```

```
    begin
    if Enable = '0' then
      Sout <= 'Z';
      level <= VREF;
    elsif rising_edge(Clk) then
      if vin >= VREF then
         Sout <= '1';
         level <= VREF + DELTA/2.0;
      else
         Sout <= '0';
         level <= VREF - DELTA/2.0;
      end if;
    end if;
  end process;
  vout == level'ramp(TR);
end ideal;
```

#### C.1.3.6 single-bit DAC

```
Library IEEE;
  use IEEE.electrical_systems.all;
  use IEEE.std_logic_1164.all;
Entity DAC_behav is
  generic ( VREF   : voltage  := 2.5;     --[V] Reference voltage
            DELTA  : voltage  := 5.0 );   --[V] Quantizer step

  port (    signal Enable, Sin  : in std_logic;
            terminal Tout    : electrical );
  begin
   assert DELTA >= 0.0  report "Quantizer step must be positive" severity
     error;
End entity DAC_behav;

Architecture ideal of DAC_behav is
  quantity vout across iout through Tout;
  constant VHIGH : real := VREF + DELTA/2.0;
  constant VLOW  : real := VREF - DELTA/2.0;
 Begin
   if Enable = '0' use
      vout == VREF;
   elsif Sin = '1' use
      vout == VHIGH;
   else
      vout == VLOW;
   end use;
   break on Enable, Sin;
end ideal;
```

### C.1.4 Sinc$^3$ digital filter

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

Entity Sinc3 is
  generic ( N : positive);
  port( RESN, MOUT, MCLK, CNR : in std_logic;
        CN5 : out std_logic_vector(N-1 downto 0));
End Sinc3;

Architecture RTL of Sinc3 is
  signal DN0, DN1, DN3, DN5 : std_logic_vector(N-1 downto 0);
  signal CN1, CN2, CN3, CN4 : std_logic_vector(N-1 downto 0);
  signal DELTA1 : std_logic_vector(N-1 downto 0);
Begin
  process(MCLK, RESn)
    begin
    if RESn = '0' then
       DELTA1 <= (others => '0');
    elsif MCLK'event and MCLK = '1' then
       if MOUT = '1' then
          DELTA1 <= DELTA1 + 1;
       end if;
    end if;
  end process;
  process(RESN, MCLK)
    begin
    if RESN = '0' then
       CN1 <= (others => '0');
       CN2 <= (others => '0');
    elsif MCLK'event and MCLK = '1' then
       CN1 <= CN1 + DELTA1;
       CN2 <= CN2 + CN1;
    end if;
  end process;
  process(RESN, CNR)
    begin
    if RESN = '0' then
       DN0 <= (others => '0');
       DN1 <= (others => '0');
       DN3 <= (others => '0');
       DN5 <= (others => '0');
    elsif CNR'event and CNR = '1' then
       DN0 <= CN2;
       DN1 <= DN0;
       DN3 <= CN3;
       DN5 <= CN4;
    end if;
  end process;
  CN3 <= DN0 - DN1;
  CN4 <= CN3 - DN3;
  CN5 <= CN4 - DN5;
end RTL;
```

## C.2 Instrumentation amplifier (INS_AMP)

```
library ieee;
use ieee.electrical_systems.all;

entity instrumentation_amp is
  generic (
    NOM_GAIN    : real   := 10.0e3;  -- Nominal Instrumentation Amplifier Gain
    IN_VOFF_TYP : voltage  := 5.0e-6;  -- Typical input voltage offset @ 25℃[V]
    MAX_IN_VOFF_OVER  : voltage := 45.0e-6;  -- Maximum input voltage offset overtemperature [V]
    OUT_VOFF_TYP : voltage := 100.0e-6; -- Typical output voltage offset @ 25℃[V]
    MAX_OUT_VOFF_OVER : voltage := 0.45e-3;  -- Maximum output voltage offset overtemperature [V]
    IN_VOFFSET_TC  :  real   := 0.3e-6;   -- Average input offset TC [V/℃]
    OUT_VOFFSET_TC    :  real   := 5.0e-6;  -- Average output offset TC [V/℃]
    GAIN_DRIFT    :  real     := -50.0;  -- [ppm/℃]
    RG_DRIFT      :  real     := -10.0;  -- [ppm/℃]
    TEMP          :  real     := 25.0;   -- [℃]
    NOM_TEMP      :  real     := 25.0;   -- [℃]
    VSP           :  voltage  := 5.0;    -- positive supply voltage [V]
    VSN           :  voltage  := -5.0);  -- negative supply voltage [V]
  port (terminal tip, tim, tout, tref : electrical);
end entity instrumentation_amp;

architecture gain of instrumentation_amp is

    function offset_voltage(VOFF_TYP:real; VOFF_OVER:real; TC:real; TEMP:real; NOM_TEMP:real) return real is
      --------------------------------------------------
      --calculate the equivalent offset voltage at current temperature in [V]
      --the voltage offset cannot be greater than VOFF_OVER
      --------------------------------------------------
      variable VOFF_cal  : real := 0.0;
    begin
        VOFF_cal := VOFF_TYP + TC*(TEMP - NOM_TEMP);
        if VOFF_cal < 0.0 then
           if abs(VOFF_cal) > VOFF_OVER then
              return -VOFF_OVER;
           else
              return VOFF_cal;
           end if;
        else
           if VOFF_cal > VOFF_OVER then
```

```
              return VOFF_OVER;
           else
              return VOFF_cal;
           end if;
        end if;
  end function offset_voltage;

  constant GAIN : real  :=  NOM_GAIN*(1.0+(GAIN_DRIFT + RG_DRIFT)/1.0e6*(TEMP - NOM_TEMP));
  constant VOSI : real  :=  offset_voltage(IN_VOFF_TYP, MAX_IN_VOFF_OVER, IN_VOFFSET_TC, TEMP, NOM_TEMP);
  constant VOSO : real  :=  offset_voltage(OUT_VOFF_TYP, MAX_OUT_VOFF_OVER, OUT_VOFFSET_TC, TEMP, NOM_TEMP);
  constant VOFF_RTI : voltage :=  VOSI + VOSO/GAIN;
  quantity vi across tip to tim;
  quantity vo across io through tout to tref;
  quantity vgain : voltage;
  quantity vin : voltage;

begin

   vin == vi + VOFF_RTI;

   if vin'above(VSP/GAIN) use
      vgain == VSP;
   elsif not vin'above(VSN/GAIN) use
      vgain == VSN;
   else
      vgain == GAIN*vin;
   end use;
   break on vin'above(VSP/GAIN), vin'above(VSN/GAIN);

   vo == vgain;

end architecture gain;
```

## C.3 Non-ideal Operational Amplifier (OPAMP)

```
library ieee;
use ieee.math_real.all;
use ieee.electrical_systems.all;

entity opamp is
  generic (
    TEMP        : real     := 25.0;     -- Environment temperature [℃]
    TEMP_NOM    : real     := 25.0;     -- Nominal temperature [℃]
    VSP         : voltage  := 10.0;     -- Positive supply voltage [V]
    VSN         : voltage  := -10.0;    -- Negative supply voltage [V]
    ADOLDC      : real     := 100.0;    -- DC differential open loop gain [dB]
    GBW         : real     := 1.0e6;    -- gain-bandwidth product [Hz]
    SR          : real     := 1.0;      -- slew rate [V/us]
    VIOFS_NOM   : voltage  := 0.0;      -- Input offset voltage at nominal temperature [V]
    VIOFS_MAX   : voltage  := 0.0;      -- Maximum input offset voltage [V]
    VIOFS_DRIFT : real     := 0.0;      -- Input offset voltage drift [uV/℃]
    IIB        : current  := 0.1e-12;   -- input bias current [A]
    CMRR       : real     := 120.0;     -- common mode rejection ratio [dB]
    PSRR       : real     := 120.0;     -- power supply rejection ratio [dB]
    CID        : capacitance := 0.0;    -- differential input capacitance [F]
    RID        : resistance  := 1.0e12; -- differential input resistance [Ohm]
    IOMAX      : current     := 2.0e-5; -- maximum output current [A]
    ROUT       : resistance  := 50.0;   -- output resistance [Ohm]
    VSOFS      : voltage     := 0.0;    -- supply offset voltage [V]
    DEBUG      : boolean     := false); -- dump parameter values
  port (
    terminal tip   : electrical;   -- non-inverting input
    terminal tim   : electrical;   -- inverting input
    terminal tout  : electrical);  -- single-ended output
end entity opamp;

architecture nonideal of opamp is

    function offset_voltage(VOFF_NOM:real; VOFF_MAX:real; TC:real; TEMP:real; TEMP_NOM:real) return real is
    -----------------------------------------------
    --calculate the equivalent offset voltage at the current temperature in [V]
    --the voltage offset cannot be greater than VOFF_MAX
    -----------------------------------------------
    variable VOFF_cal  : real := 0.0;
    begin
       VOFF_cal := VOFF_NOM + TC*1.0e-6*(TEMP - TEMP_NOM);
       if VOFF_cal > VOFF_MAX then
          return VOFF_MAX;
       elsif VOFF_cal < -VOFF_MAX then
          return -VOFF_MAX;
       else
          return VOFF_cal;
       end if;
    end function offset_voltage;

   -- Internal parameters:
   constant  VIOFS : real :=  offset_voltage(VIOFS_NOM, VIOFS_MAX,
   VIOFS_DRIFT, TEMP, TEMP_NOM); -- Effective input offset voltage [V]
   constant ADOLDCL  : real := 10.0**(ADOLDC/20.0);  -- differential open loop gain [V/V]
   constant ACML    : real := ADOLDCL/(10.0**(CMRR/20.0)); -- common mode gain [V/V]
   constant APS     : real := ADOLDCL/(10.0**(PSRR/20.0)); -- Power Supply Voltage gain for both positive and negative power supplies:  APS = Add = Ass [V/V]
   constant FP  : real := GBW/ADOLDCL;  -- dominant pole frequency [Hz]
   constant WP  : real := MATH_2_PI*FP; -- dominant pole frequency [rad/s]
   constant CP  : capacitance := IOMAX/(1.0e6*SR); -- dominant pole capacitance [F]
   constant RP  : resistance := 1.0/(WP*CP); -- dominant pole resistance [Ohm]
   constant GM0 : real := ADOLDCL/RP;   -- transconductance [S]
   constant VILIMP : voltage := VIOFS + IOMAX/GM0; -- positive slew rate limit voltage [V]
   constant VILIMM : voltage := VIOFS - IOMAX/GM0; -- negative slew rate limit voltage [V]

   -- Terminals and branch quantities:
   quantity vip  across        -- positive terminal voltage
            iibp through       -- positive input bias current
            tip;               -- to electrical'reference

   quantity vim  across        -- negative terminal voltage
            iibm through       -- negative input bias current
            tim;               -- to electrical'reference

   quantity vid across         -- input differential voltage
            icid,              -- differential input capacitance current
            irid through       -- differential input resistance current
            tip to tim;

   terminal np : electrical;   -- internal stage node

   quantity vg across          -- amplified voltage
            ig,                -- transconductance current
            ip,                -- filtered current
            ilim through       -- limitation current
            np;                -- to electrical'reference

   quantity vout across        -- single-ended output voltage
            iout through       -- output current
            tout;              -- to electrical'reference

   -- Free quantities:
   quantity vide   : voltage; -- effective inout differential voltage
   quantity vicm   : voltage; -- common mode input voltage
   quantity igcm   : current; -- common mode transconductance current
   quantity igpps  : current; -- positive power supply transconductance current
   quantity ignps  : current; -- negative power supply transconductance current
   quantity vosatp : voltage; -- positive output voltage limit
   quantity vosatm : voltage; -- negative output voltage limit

begin

   -- Input stage:
   vicm   == (vip + vim)/2.0;
   vide   == vid + VIOFS;
   iibp   == IIB/2.0;
   iibm   == IIB/2.0;
   icid   == CID*vide'dot;
   irid   == vide/RID;
```

```

    -- Transconductance, slew rate and compensation stage:
    igcm  == (ACML/RP)*vicm;
    igpps == (APS/RP)*VSP;
    ignps == (APS/RP)*VSN;
    if vide'above(VILIMP) use
       ig == -IOMAX;
    elsif not vide'above(VILIMM) use
       ig == IOMAX;
    else
       ig == -(GM0*vide + igcm + igpps + ignps);
    end use;
    break on vide'above(VILIMP), vide'above(VILIMM);

    -- Frequency-domain behavior:
    if domain = quiescent_domain use
       -- initial conditions for quantities:
       vg == ADOLDCL*vide;
    else
       ip == CP*vg'dot + vg/RP;
    end use;

    -- Output stage and limitation:
    vout == vg + ROUT*iout;
    vosatp == VSP - VSOFS;
    vosatm == VSN + VSOFS;
    if vout'above(vosatp) use
       ilim == GM0*(vout - vosatp);
    elsif  not vout'above(vosatm) use
       ilim == GM0*(vout - vosatm);
    else
       ilim == 0.0;
    end use;
    break on vout'above(vosatp), vout'above(vosatm);

end architecture nonideal;
```

## C.4 Polynomial curve fitting algorithm script

This algorithm has been written in Matlab for multivariable polynomial curve fitting of the $T_{SS}$ parameter in Equation 4.13 on page 79. In order to speed up the simulation time, the static parametric FEA simulation is performed by only sweeping `Troom` and `Tbus` parameters. Therefore, for a correct use of this algorithm it is required to have the `Troom` and `Tbus` vectors as it is indicated in Table 4.2 together with the $T_{SS}$ (`Tcoil`) vector result from COMSOL for each value of the `Agap` parameter in the range, as it is also described in Table 4.2 on page 78.

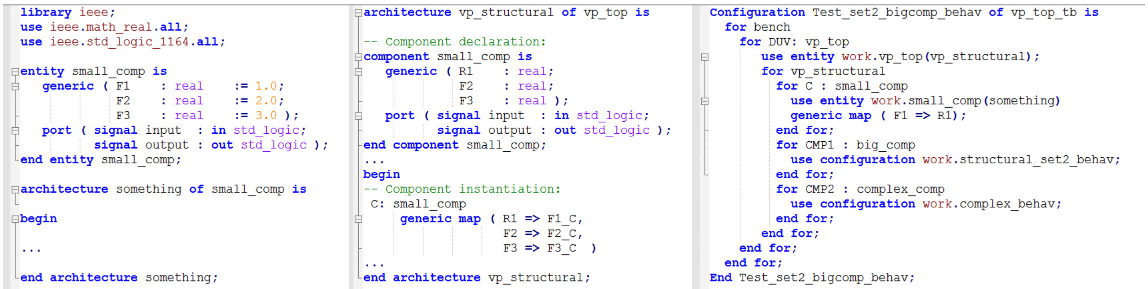
```
vect1 = ones(840,1);
% Param_Matrix Headers = [Repoxy, Troom, Tbus, Tcoil]
Param_Matrix = [44e-3*vect1, Troom, Tbus, Tcoil_Re44K; 45e-3*vect1, Troom
    , Tbus,
Tcoil_Re45K; 46e-3*vect1, Troom, Tbus, Tcoil_Re46K; 47e-3*vect1, Troom,
    Tbus,
Tcoil_Re47K; 48e-3*vect1, Troom, Tbus, Tcoil_Re48K; 49e-3*vect1, Troom,
    Tbus,
Tcoil_Re49K; 50e-3*vect1, Troom, Tbus, Tcoil_Re50K];

% Kelvin to Celcius scaling
Troom_C = Param_Matrix(:,2) - 273.15;
Tbus_C = Param_Matrix(:,3) - 273.15;
Tss_C = Param_Matrix(:,4) - 273.15;
Agap = 49.2e-3 - Param_Matrix(:,1);

for i = 1:size(Agap,1)
    if Agap(i) < 0
        Agap(i) = 0;
    end
end

% Decrease the data space
Tbus_imp = 0;
Troom_imp = 0;
Tss_imp = 0;
Agap_imp = 0;
j=1;

for i = 1:size(Tss_C,1)
    if Tbus_C(i) >= Troom_C(i)
        Tbus_imp(j) = Tbus_C(i);
        Troom_imp(j) = Troom_C(i);
        Tss_imp(j) = Tss_C(i);
        Agap_imp(j) = Agap(i);
        j=j+1;
    end
end

Tbus_imp = Tbus_imp';
Troom_imp = Troom_imp';
Tss_imp = Tss_imp';
Agap_imp = Agap_imp';

% Curve fitting by optimization rutine:
[coeff, model] = fitcurvepoly(Tbus_imp, Troom_imp, Agap_imp, Tss_imp);
[sse, FittedCurve] = model(coeff);
Rsquare = 1 - sse/(sum((Tss_imp-mean(Tss_imp)).^2));

Tss_fit = coeff(1) + coeff(2)*Tbus_imp + coeff(3)*Troom_imp + coeff(4)*
    Agap_imp;
```

The `fitcurvepoly` function uses the **fminsearch** optimization rutine of Matlab as follows:

```
function [estimates, model] = fitcurvepoly(x1, x2, x3, y)
% Call fminsearch with a random starting point.
start_point = rand(1, 4);
model = @poly111;
estimates = fminsearch(model, start_point);
% poly111 accepts curve parameters as inputs, and outputs sse,
% the sum of squares error for P0 + P1*x1 + P2*x2 + P3*x3 - y,
% and the FittedCurve. FMINSEARCH only needs sse, but we want
% to plot the FittedCurve at the end.
    function [sse, FittedCurve] = poly111(params)
        P0 = params(1);
        P1 = params(2);
        P2 = params(3);
        P3 = params(4);
        FittedCurve = P0 + P1*x1 + P2*x2 + P3*x3;
        ErrorVector = FittedCurve - y;
        sse = sum(ErrorVector .^ 2);
    end
end
```

# D Appendix D - Generic parameter mapping

In order to exemplify how the principles for mapping in configurations operate, see subsection 3.5.1, let us consider Figure D.1(b), where the component `small_comp` is declared and instantiated inside the `vp_structural` architecture of `vp_top`. The signature of this component corresponds with the `small_comp` entity declaration shown in Figure D.1(a) with the exception of the first generic parameter: `R1` for the component, `F1` for the entity. The `small_comp` component binding with the corresponding entity is done in the configuration shown in Figure D.1(c). As the principle (**I**) indicates, the parameters `F2` and `F3` of the component are bound by default with the corresponding parameters in the entity due to their names are equal. However, there is no default binding for the first generic parameter since the names in the entity and in the component are different. A compilation error will be generated if no generic mapping is given in such case. A typical error by trying to solve this problem is defining a generic mapping only for the parameter that needs to be bound as is shown in Figure D.1(c). In this case, there are no compilation errors and designers can erroneously think that the parameter is correctly bound as it is intended, i.e. to set the values of the parameters `F1_C`, `F2_C` and `F3_C` as is indicated in the component instantiation in Figure D.1(b).



```
library ieee;
use ieee.math_real.all;
use ieee.std_logic_1164.all;

entity small_comp is
   generic ( F1    : real    := 1.0;
             F2    : real    := 2.0;
             F3    : real    := 3.0 );
   port ( signal input  : in std_logic;
          signal output : out std_logic );
end entity small_comp;

architecture something of small_comp is

begin

...

end architecture something;
```

(a)

```
architecture vp_structural of vp_top is

-- Component declaration:
component small_comp is
   generic ( R1    : real;
             F2    : real;
             F3    : real );
   port ( signal input  : in std_logic;
          signal output : out std_logic );
end component small_comp;
...
begin
-- Component instantiation:
 C: small_comp
     generic map ( R1 => F1_C,
                   F2 => F2_C,
                   F3 => F3_C  )
...
end architecture vp_structural;
```

(b)

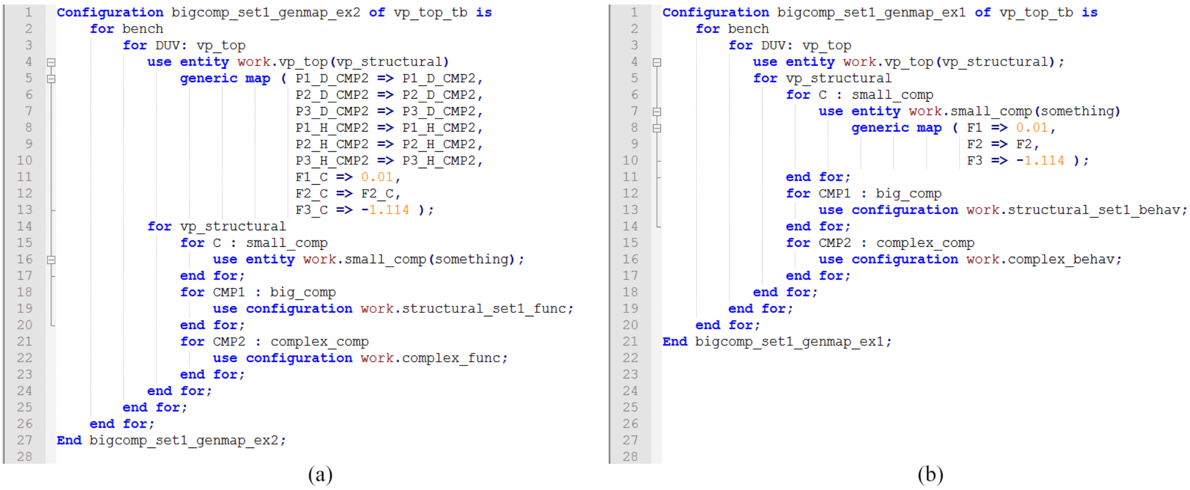
```
Configuration Test_set2_bigcomp_behav of vp_top_tb is
  for bench
    for DUV: vp_top
      use entity work.vp_top(vp_structural);
      for vp_structural
        for C : small_comp
          use entity work.small_comp(something)
          generic map ( F1 => R1);
        end for;
        for CMP1 : big_comp
          use configuration work.structural_set2_behav;
        end for;
        for CMP2 : complex_comp
          use configuration work.complex_behav;
        end for;
      end for;
    end for;
  end for;
End Test_set2_bigcomp_behav;
```

(c)

Figure D.1: Generic parameter mapping in configurations. This code is based on the example VP given in Figure 3.2 on page 33.

As principle **II** states, the generic mapping done in the component instantiation is overwritten and only the bindings indicated in the generic mapping in the configuration will be valid. Consequently, any hierarchical chain of the component is broken. In this case, the generic parameter `F1` of the entity is bound to the generic parameter `R1` of the component which in turn is bound to the parameter `F1_C` in the component declaration. On the other hand, since there are no generic mappings for the parameters `F2` and `F3` in the configuration, those parameters will have the default value given in the entity declaration, i.e. `2.0` and `3.0` respectively. If the parameters did not have a default value in the entity declaration, an error would arise at compilation time. Taking into account the principle **II**, defining default values in the component declaration, i.e. for the parameters `F2` and `F3` in Figure D.1(b), does not have any effect as long as a generic mapping of the component from a configuration exists[1].

From the VHDL-AMS binding mechanism highlighted in principle **II**, a modeling solution can be derived for setting the value of few generic parameters via configuration, in a structural model that has been built by using a *hierarchical parameter binding* approach. Let us consider the example VP configurations shown in Figure D.2.



```
Configuration bigcomp_set1_genmap_ex2 of vp_top_tb is
    for bench
        for DUV: vp_top
           use entity work.vp_top(vp_structural)
               generic map ( P1_D_CMP2 => P1_D_CMP2,
                             P2_D_CMP2 => P2_D_CMP2,
                             P3_D_CMP2 => P3_D_CMP2,
                             P1_H_CMP2 => P1_H_CMP2,
                             P2_H_CMP2 => P2_H_CMP2,
                             P3_H_CMP2 => P3_H_CMP2,
                             F1_C => 0.01,
                             F2_C => F2_C,
                             F3_C => -1.114 );
           for vp_structural
               for C : small_comp
                   use entity work.small_comp(something);
               end for;
               for CMP1 : big_comp
                   use configuration work.structural_set1_func;
               end for;
               for CMP2 : complex_comp
                   use configuration work.complex_func;
               end for;
           end for;
        end for;
    end for;
End bigcomp_set1_genmap_ex2;
```

(a)

```
Configuration bigcomp_set1_genmap_ex1 of vp_top_tb is
    for bench
        for DUV: vp_top
           use entity work.vp_top(vp_structural);
           for vp_structural
               for C : small_comp
                   use entity work.small_comp(something)
                       generic map ( F1 => 0.01,
                                     F2 => F2,
                                     F3 => -1.114 );
               end for;
               for CMP1 : big_comp
                   use configuration work.structural_set1_behav;
               end for;
               for CMP2 : complex_comp
                   use configuration work.complex_behav;
               end for;
           end for;
        end for;
    end for;
End bigcomp_set1_genmap_ex1;
```

(b)

Figure D.2: Generic parameter mapping in configurations, application example.

Suppose that it is only required to set the values of the parameters `F1` and `F3` of the `small_comp` component to `0.01` and `-1.114` respectively. These parameters can be set from the TBC by the two options shown in Figure D.2(a) and D.2(b). Observe that both examples produce exactly the same result in the `small_comp` component. However, the code of the first example (D.2(a)) is larger than the second example (D.2(b)) due to is mandatory to specify the generic mapping of all parameters of the design entity bound to the component in the configuration. Otherwise, the hierarchical bound is broken and the parameters without generic mapping will use the default parameters of the entity. This little code size difference is trivial in this example but can be significant in a real application with much more parameters and hierarchical levels. Consequently, in order to have more compact configurations with less code, it is recommended to make the generic mapping in configurations at the lowest level of the hierarchy in the VP. In fact, as long as configurations are used for setting the generic parameters of the DUV, the hierarchical parameter binding of the model can be ignored.

VHDL-AMS modeling language allows to set the values of the generic parameters of the complete design from the configurations by using the following options:

1. Numeric values.
2. Constant declared in packages.

---

[1] i.e. the default values will never be used and/or compilation errors cannot be solved.

3. Constant declared in the architecture of the top design entity of the configuration.
4. Binding to the generic parameters of the component.

The first option is clear and straightforward. For the second option, the unique condition is that the package which contains the constant is made visible in the configuration by using the `use` clause.

The third option refers to the fact that only the constants declared at the highest level architecture seen by the configuration can be used in such configuration. However, these constants can only be used in the components declared at the highest level architecture seen by the configuration. Using again the example shown in Figure D.2, the aforementioned restriction means that the constants declared in the *test bench* architecture (named `bench`) can only be used to set the values of the generic parameters of the DUV component via configurations; so that, the numeric values in `F1_C` and `F3_C` can be replaced for constants declared in the *test bench* in Figure D.2(a). Conversely, the *test bench* variables cannot be used in the case shown in Figure D.2(b) since the component `small_comp` is out of the scope of the configuration. At the time of this research, it has been found that this VHDL-AMS mechanism is not well supported by the tools of the *simulation framework*. ModelSim from Mentor Graphics supports this mechanism but SMASH (6.5) does not. Taking into account that this mechanism is not clearly supported by tools and the scope of the test bench constants is limited for hierarchical models, this option is not recommended.

Finally, the fourth option is related to the typical case of binding the generic parameters of the entity to the generic parameter of the component. In this way, the value of the generic parameter will be determined by the generic mapping in the component instantiation. Consequently, this option is useful when it is desired to set the generic parameter values by *hierarchical parameter binding*. If default parameter binding in configurations is desired, the names of the generic parameters of the component declaration must be the same than the corresponding parameter names of the entity that is bound, otherwise, the syntax to follow for generic parameter binding is as follows:

```
use entity work.entity_name(architecture_name)
generic map ( P_name_entity => P_name_component,
                 ...  );
```

where `P_name_entity` is the parameter name in the entity and `P_name_component` is the corresponding parameter name in the component.

# E Appendix E - Parametric and statistical simulations in VHDL-AMS

For the design of complex heterogeneous systems, it is often desired to study the effect of parameter variations on the behavior of components and complete systems. Parametric and statistical simulations, such as MC simulations, are normally used to analyze the performance and reliability of those components and systems. These type of simulations become very useful to estimate results and trends when they cannot be obtained by using deterministic formulas. Furthermore, applying distribution functions on the parameters of simulation models helps to determine where design on process optimizations become necessary to get the system behavior in an acceptable range. The ability to simulate tolerances and process variations of the system components allows performing statistical studies for system verification and validation.

Commonly, the mechanisms to perform parametric and statistical simulations are highly dependent on the simulator tool. In this appendix, by using a simple example, we present how to make standard Monte Carlo and parameter sweep simulations by using a tool independent VHDL-AMS approach. This is possible by using the **SAE J2748** statistical packages [131] and a custom **Parametric & Statistical** package developed by the author of this thesis; this package allows to make parametric sweep simulations and defining new statistical distributions based on the SAE statistical packages.

The SAE packages support the statistical modeling of design parameters subject to tolerances for models described using the VHDL-AMS language. The impact of tolerances in a design that uses the SAE packages can be analyzed by a MC simulation, which consists of multiple simulation runs of the design, each run with a different set of parameter values according to their statistical distributions. The MC simulation yields an estimate of the behavior of the design subject to parameter variations. Additionally, the packages can be used to perform worst case analysis of a design, i.e. an analysis that shows if the system performance remains in a specified range even in presence of uncertainties [18].

An interesting application example of the SAE J2748 statistical packages can be found in [132], where the variance-reducing Monte Carlo method is presented as a technique to reduce the simulation runs of the standard MC method when small probabilities have to be determined.

## E.1 Monte Carlo simulations

In order to illustrate the usage of some important statistical functions of the SAE J2748 packages, let us consider a simple resistive voltage divider (RVD) example, see Figure E.1. The

RVD is fed with a DC voltage of 10 V and uses two resistors with a nominal resistance of 1 kΩ. Thus, the output voltage `Vout` is equal to 5 V if there is no fluctuation in the circuit. Now, suppose that we want to simulate the resistance tolerance effect into the output voltage. For that purpose, we perform an Operating Point MC simulation consisting of 500 runs. In order to simplify our analysis, we only include the tolerance effect in `R2`, `R1` takes the nominal value in all simulations.

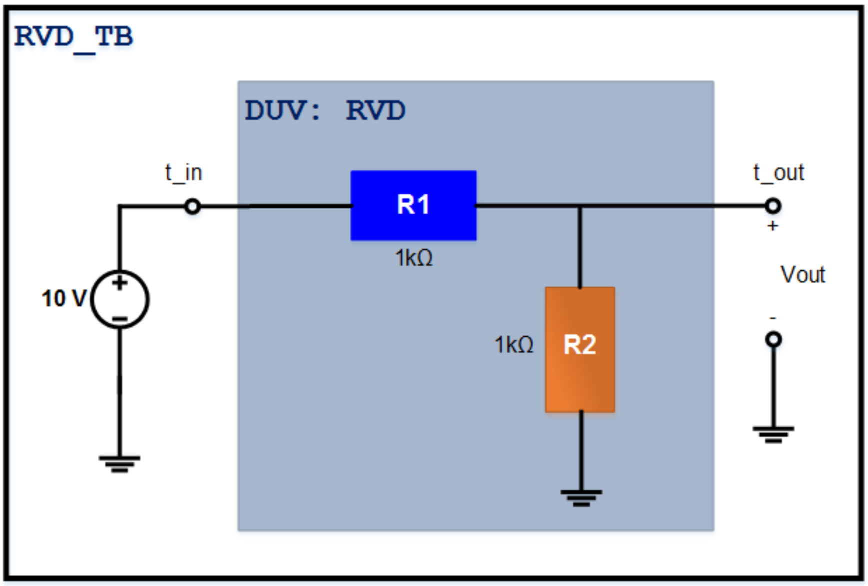


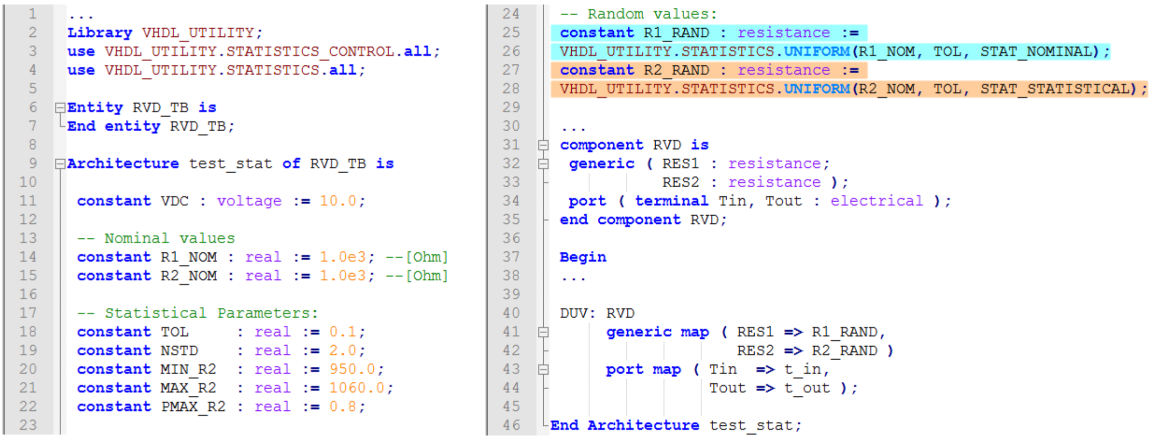
Figure E.1: Block diagram of the Resistive Voltage divider test bench.

The DUV is a structural architecture which consists of two instantiations of an ideal resistor component. The most important part of the test bench code is shown in Figure E.2. We can observe in lines 2, 3, and 4 how the `STATISTICS` and `STATISTIC_CONTROL` packages of the SAE J2748 are used. As it is explained in [131] (chapter 4), the SAE functions can be used to assign statistical distributions to constants inside a model or outside of a model, i.e. by using a generic map instantiation. In order to get a maximum flexibility for the manipulation of the models, as a general rule, we recommend setting the statistical distributions outside of the models, i.e. from the *test bench* or from the main packages of the design as is explained in the modeling guidelines, section 3.3.

Defining the statistical distributions inside the models works for a few number of parameters, but may become unmanageable for models with many parameters. Moreover, to simulate the design with different distributions would require manipulating the architectures of the design in a more complicated way. Therefore, the SAE package must be used at the top level design unit, as it is shown in Figure E.2.

Let us concentrate our attention on the code shown from line 25 to 28 in Figure E.2. In order to set the nominal value to the resistor `R1`, it is enough to write the numerical value in line 25 as it is done for the constant `R1_NOM` in line 14. However, we can also write as follows:

```
constant R1_RAND : resistance := VHDL_UTILITY.STATISTICS.UNIFORM(R1_NOM,
                                  TOL, STAT_NOMINAL);
```



```
 1  ...
 2  Library VHDL_UTILITY;
 3  use VHDL_UTILITY.STATISTICS_CONTROL.all;
 4  use VHDL_UTILITY.STATISTICS.all;
 5
 6  Entity RVD_TB is
 7  End entity RVD_TB;
 8
 9  Architecture test_stat of RVD_TB is
10
11   constant VDC : voltage := 10.0;
12
13   -- Nominal values
14   constant R1_NOM : real := 1.0e3; --[Ohm]
15   constant R2_NOM : real := 1.0e3; --[Ohm]
16
17   -- Statistical Parameters:
18   constant TOL     : real := 0.1;
19   constant NSTD    : real := 2.0;
20   constant MIN_R2  : real := 950.0;
21   constant MAX_R2  : real := 1060.0;
22   constant PMAX_R2 : real := 0.8;
23
```

```
24   -- Random values:
25   constant R1_RAND : resistance :=
26   VHDL_UTILITY.STATISTICS.UNIFORM(R1_NOM, TOL, STAT_NOMINAL);
27   constant R2_RAND : resistance :=
28   VHDL_UTILITY.STATISTICS.UNIFORM(R2_NOM, TOL, STAT_STATISTICAL);
29
30   ...
31   component RVD is
32    generic ( RES1 : resistance;
33              RES2 : resistance );
34    port ( terminal Tin, Tout : electrical );
35   end component RVD;
36
37   Begin
38   ...
39
40   DUV: RVD
41        generic map ( RES1 => R1_RAND,
42                      RES2 => R2_RAND )
43        port map ( Tin  => t_in,
44                   Tout => t_out );
45
46  End Architecture test_stat;
```

Figure E.2: `test_stat` architecture of the RVD Test Bench.

where the first argument `R1_NOM` is the nominal resistance value, the second argument `TOL` is the tolerance value, and the third argument is a `STAT_CONTROL` deferred constant value, see the complete definition in [131]. There are 3 possible values for this argument: if the constant has a value of `STAT_NOMINAL`, all regular distribution functions (in this case the UNIFORM function) will return nominal values. If the constant has a value of `STAT_STATISTICAL`, the distribution functions will return statistical values. If the constant has a value of `STAT_INTEGRATED`, the simulator will control whether nominal or statistical values will be returned by the distribution functions. In the specific case of the SMASH 6.5 simulator, the `STATISTIC_CONTROL` package is only partially integrated; therefore, the `STAT_INTEGRATED` value must not be used. The default value is `STAT_NOMINAL`.

In this example, we are using the `UNIFORM` distribution function of the SAE package, nonetheless, we can use any of the functions of the SAE package. As long as this argument is set to the value `STAT_NOMINAL`, the value of `R1_RAND` will be the value of `R1_NOM`, i.e. 1 kΩ. Since it is easy to switch between nominal and random values for the constant parameters, it is a good practice to set all the potential statistical parameters of the design in this way from the *test bench*. Finally, random values for the resistor `R2` are assigned by setting a distribution function at the constant `R2_RAND` in lines 27-28 of Figure E.2. Some illustrative examples are given in the following sub-sections.

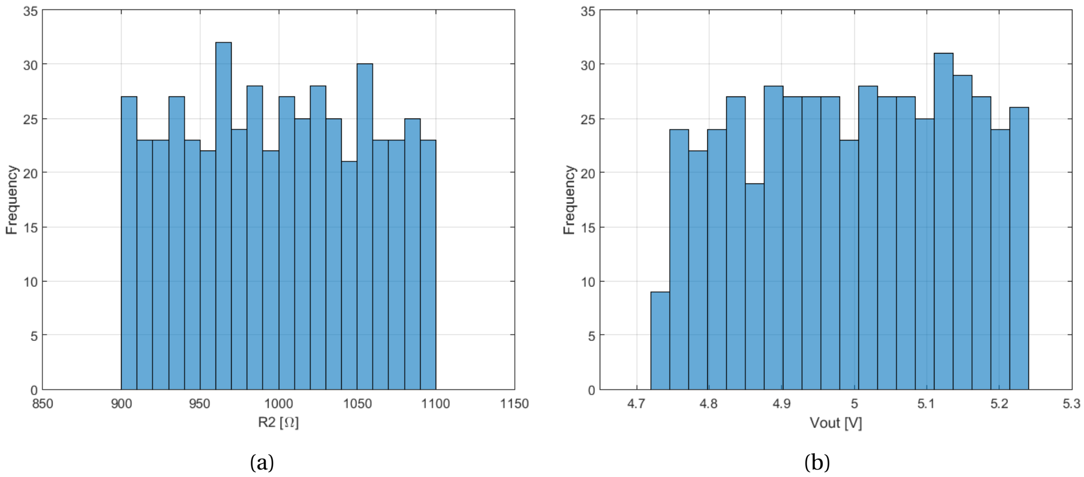
### E.1.1 Uniform distribution

```
constant R2_RAND : resistance := VHDL_UTILITY.STATISTICS.UNIFORM(R2_NOM, TOL,
                                              STAT_STATISTICAL);
```

If `R2_RAND` is defined as it is specified above, the `R2` resistor value will be uniformly distributed between 900 Ω and 1100 Ω (i.e. 1 kΩ±10%) given the `TOL` value declared at line 18 in Figure E.2. We can observe in Figure E.3 that the histograms of both the `R2` resistor value and the `Vout` voltage denote approximately a uniform distribution after 500 runs of MC simulation. The higher the number of runs the closer is the histogram graph to a uniform distribution[1].

---

[1] The same affirmation is true for all the probability distributions presented in this appendix.



(a)

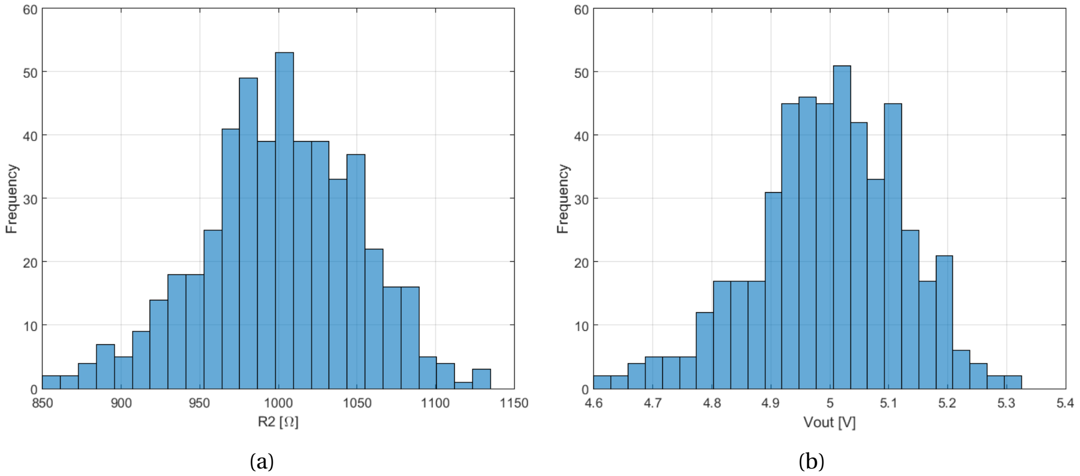
(b)

Figure E.3: Uniform distribution histograms. (a) R2 resistor value. (b) RVD output voltage.

### E.1.2 Normal distribution

```
constant R2_RAND : resistance := VHDL_UTILITY.STATISTICS.NORMAL(R2_NOM, TOL,
                                    False, NSTD, STAT_STATISTICAL);
```

By using the definition above for setting the value of `R2_RAND` constant, the `R2` resistor value will be normally distributed with a mean equal to the nominal value ($\mu = 1000$) and a standard deviation equal to: `R2_NOM * TOL/NSTD` ($\sigma = 50.0$). This normal distribution is not truncated at maximum and minimum values, which are given by the `NSTD` parameter. `NSTD` defines how many standard deviations are between minimum and nominal value, and between nominal and maximum value. Figures E.4(a) and E.4(b) demonstrate that both `R2` and `Vout` values are normally distributed as it is defined in the `R2_RAND` constant declaration.



(a)

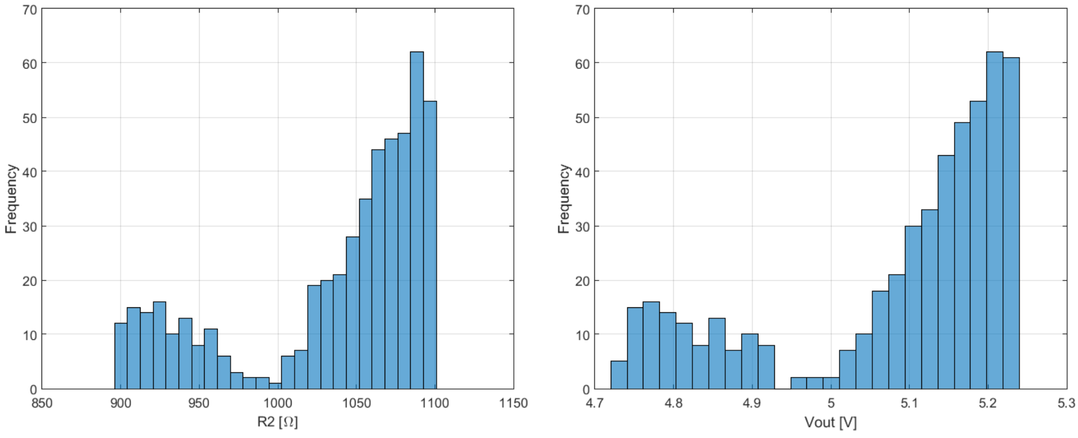
(b)

Figure E.4: Normal distribution histograms. (a) R2 resistor value. (b) RVD output voltage.

### E.1.3 Piecewise linear distribution given as PDF

```
constant R2_RAND : resistance := VHDL_UTILITY.STATISTICS.PWL_PDF(R2_NOM, TOL,
        ((-1.0,0.8),(0.0,0.01),(1.0,2.9)), STAT_STATISTICAL);
```

Some statistical processes might not behave as a standard probability density function (PDF) with a specific equation. If we can observe experimentally the statistical behavior of a random variable, we can use a piecewise linear distribution to approximate the behavior of the random variable. The SAE package offers two useful ways to define a distribution, the piecewise linear distribution given as a cumulative density function (`PWL_CDF`) or a piecewise linear distribution given as a PDF (`PWL_PDF`), see the `R2_RAND` declaration above. In this example, the parameters `R2_NOM` and `TOL` are defined with the same tolerance range as in the previous examples (1 kΩ±10%). Additionally, it is required to add a real (x,y) pair table, which the abscissa values (x) must be defined on a normalized scale, where -1, 0, 1 correspond respectively to the minimum, nominal, and maximum values. The ordinate values (y) are weight coefficients (greater than zero) proportional to the PDF value of the abscissa points. Figure E.5 shows the respective results for the `R2` and `Vout` distributions.



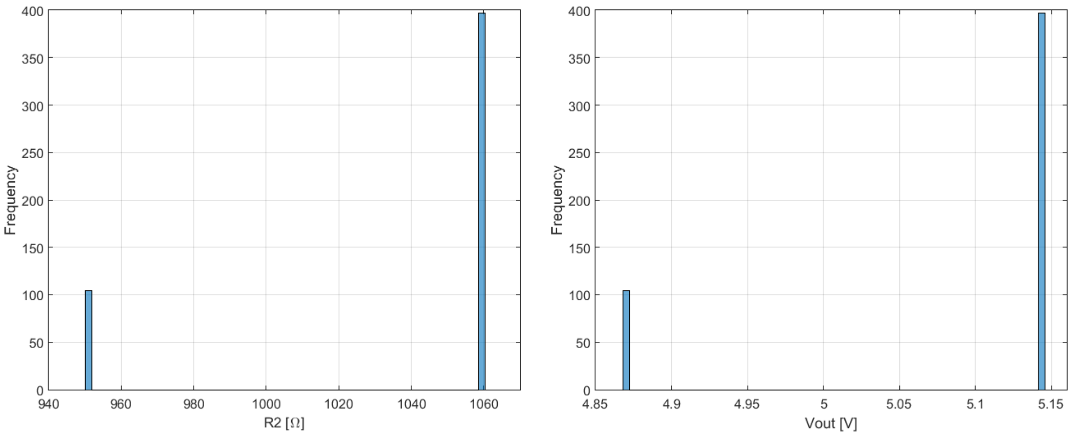
(a) (b)

Figure E.5: Histograms of the Piecewise linear distribution given as PDF. (a) `R2` resistor value. (b) RVD output voltage.

### E.1.4 Bernoulli distribution

```
constant R2_RAND : resistance := VHDL_UTILITY.STATISTICS.BERNOULLI(R2_NOM,
        MIN_R2, MAX_R2, PMAX_R2, STAT_STATISTICAL);
```

An interesting random variable case that the SAE package allows easily simulating is the Bernoulli distribution, which is a discrete distribution that has two possible outputs, one with probability `P`, and the other with probability `1-P`. The `R2_RAND` declaration above allows simulating the resistor value `R2` from two possible values: `MIN_R2` and `MAX_R2`. The probability of returning the value `MAX_R2` is `PMAX_R2`, whereas the probability of returning `MIN_R2` is `1 - PMAX_R2`. Given the values of `MIN_R2`, `MAX_R2`, and `PMAX_R2` in lines 20, 21, and 22 respectively in Figure E.2 on page 213, the MC simulation result is given in Figure E.6.



(a) (b)

Figure E.6: Bernoulli distribution histograms. (a) `R2` resistor value. (b) RVD output voltage.

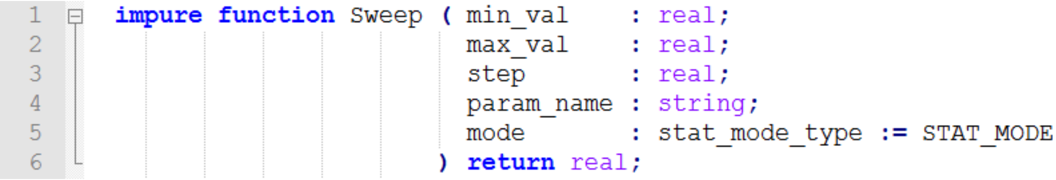
Although the Bernoulli distribution is not an adequate PDF for modeling resistor values, it is very useful for modeling the random behavior of boolean parameters. Likewise, the SAE package offers additional discrete distribution functions such as the `DISCRETE_CDF` and the `DISCRETE_PDF`, see more details in [131].

## E.2 Parameter sweep simulations

Another type of simulations that are often exclusively dependent on the simulator features are the parametric simulations. Graphical and code-based simulator tools from the main vendors such as Dolphin Integration, Mentor Graphics, and Cadence, allow VHDL-AMS implementations, but they do not use the same mechanism to define parameters for implementing parametric simulations. This is one of the main issues to implement platform-independent models able to be simulated in any software simulator.

The author of this thesis has developed a platform-independent VHDL-AMS-based mechanism to execute single and multiple parameter sweep simulations using the same approach of the SAE packages. It is important mentioning that the simulator must offer the possibility to run multiple simulations of the standard simulation types, i.e. operating point (DC), time-domain (transient), and frequency-domain (AC) simulations. In general, the simulators offer two ways to perform multiple simulations: using a MC command (e.g. **.MC** in SMASH) for running MC simulations or using a parametric sweep command (e.g. **.STEP** in SMASH) for running parameter sweep simulations. Our approach works with the two options with some restrictions and specifications that are detailed in this appendix. For this purpose, the custom package `PARAMETRIC_STATISTICAL_PKG` has been created to group the parameter sweep functions and other custom statistical distributions based on the SAE packages [131].

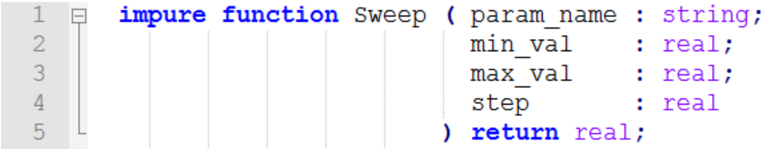
Figure E.7 shows the declaration of the impure function `Sweep`, which is meant to be used with the MC command of the simulator. This function assigns a parameter value starting from the minimum (`min_val`) until the maximum (`max_val`) argument by increasing the amount given by the `step` argument. The function receives a string argument `param_name` to distinguish among two or more parameters that can be swept at the same time. The `Sweep` function has been implemented by using an external text file for reading and storing the parameter value of the next simulation run; it employs one file per each parameter swept. The function uses the `STAT_CONTROL` deferred constant from the `STATISTICS_CONTROL` package to control the initialization of the parameter file[2]. The simulator always performs the first simulation using the `STAT_NOMINAL` value, in which is created the corresponding parameter file and returns the `min_val` for the first simulation.



```
1  impure function Sweep ( min_val    : real;
2                          max_val    : real;
3                          step       : real;
4                          param_name : string;
5                          mode       : stat_mode_type := STAT_MODE
6                        ) return real;
```

Figure E.7: Sweep function declaration to be used with MC simulations.

Similarly, by using function overloading, the `Sweep` function can also be used with the standard parametric sweep command provided in all simulators. In Figure E.8 we can observe that this function requires the same arguments that the `Sweep` function for MC simulations with the exception of the `mode` argument. Since the `mode` argument can be omitted in the instantiation of the `Sweep` function for MC simulations, the order of the arguments is different in the `Sweep` function for standard parametric simulations with the purpose of making a stronger difference between both functions. The difference relies on the fact that the `Sweep` function for standard parametric simulations requires the parameter name (`param_name`) as the first argument, whereas the same argument for the `Sweep` function for MC simulations is given in fourth place.



```
impure function Sweep ( param_name : string;
                        min_val    : real;
                        max_val    : real;
                        step       : real
                      ) return real;
```

Figure E.8: Sweep function declaration for standard parametric simulations.

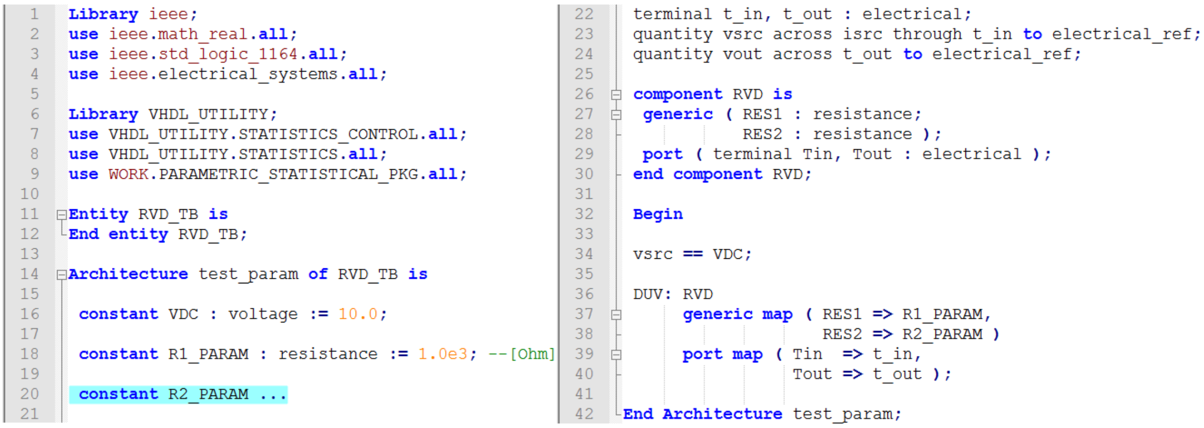
The `Sweep` function for standard parametric simulations does not need to use any function from the SAE package and it is completely independent of the simulator. However, the price to pay for tool non-dependency is a relative smaller simulation efficiency in comparison to the previous `Sweep` function and to the own mechanism of the simulator to perform parametric simulations. This is mainly caused by the additional burden to open files to write and read control variables and parameter values. Before using the `Sweep` function for standard parametric simulations, the user must make an external initialization of the control file (*First_run.dat*). This text file contains just 2 lines of data: the first line holds a boolean value (TRUE or FALSE) of the variable in charged of making the initialization of the parameter file and setting the `min_val` as the first value for simulation; the second line holds an integer number which specifies the number of parameters to be swept by one or more `Sweep` function calls. It is important clarifying that one `Sweep` function call only sweeps one parameter. For multiple-parameter sweeping more calls to this function must be done from the test bench or any other design entity in which the `PARAMETRIC_STATISTICAL_PKG` package is used.

In order to better explain the operation of the aforementioned `Sweep` functions, let us use the

---

[2] In line 5 of Figure E.7, the `STAT_MODE` constant contains the current value of the `STATISTICS_CONTROL` constant.

same example previously introduced in section E.1, the RVD test bench, see Figure E.1 on page 212.



```
 1  Library ieee;
 2  use ieee.math_real.all;
 3  use ieee.std_logic_1164.all;
 4  use ieee.electrical_systems.all;
 5
 6  Library VHDL_UTILITY;
 7  use VHDL_UTILITY.STATISTICS_CONTROL.all;
 8  use VHDL_UTILITY.STATISTICS.all;
 9  use WORK.PARAMETRIC_STATISTICAL_PKG.all;
10
11  Entity RVD_TB is
12  End entity RVD_TB;
13
14  Architecture test_param of RVD_TB is
15
16   constant VDC : voltage := 10.0;
17
18   constant R1_PARAM : resistance := 1.0e3; --[Ohm]
19
20   constant R2_PARAM ...
21
22   terminal t_in, t_out : electrical;
23   quantity vsrc across isrc through t_in to electrical_ref;
24   quantity vout across t_out to electrical_ref;
25
26   component RVD is
27    generic ( RES1 : resistance;
28              RES2 : resistance );
29    port ( terminal Tin, Tout : electrical );
30   end component RVD;
31
32   Begin
33
34   vsrc == VDC;
35
36   DUV: RVD
37        generic map ( RES1 => R1_PARAM,
38                      RES2 => R2_PARAM )
39        port map ( Tin  => t_in,
40                   Tout => t_out );
41
42  End Architecture test_param;
```

Figure E.9: `test_param` architecture of the RVD Test Bench.

Let us consider the test bench code shown in Figure E.9, the custom parametric and statistical package is included in line 9. In this case, we want to keep constant the `R1` resistance value as is shown in line 18 whilst sweeping the `R2` resistance value from 50 Ω to 2000 Ω in steps of 50 Ω. For this purpose, the `R2_PARAM` constant highlighted in line 20 must be declared as follows:

**For MC simulations:**

```
constant R2_PARAM : resistance := WORK.PARAMETRIC_STATISTICAL_PKG.Sweep(50.0,
                                   2.0e3, 50.0, "R2");
```

**For standard parametric simulations:**

```
constant R2_PARAM : resistance := WORK.PARAMETRIC_STATISTICAL_PKG.Sweep("R2",
                                   50.0, 2.0e3, 50.0);
```

The result for both cases is the same, see Figure E.10. In order to sweep the `R2` resistance over the complete range, it is required to run 40 operating point simulations. For the MC simulations case, the first simulation is always using the nominal value, i.e. `min_val`.

Therefore, the number of MC simulations to indicate in the simulator is always equal to the maximum number of requested simulations (in the case of sweeping additional parameters at the same time) minus one, in our case 39. If more simulations are run than the required, the initial value of the parameter will be again set, and therefore, the sweep will be executed cyclically. For the standard parametric simulations case, there is no need to calculate the required number of simulations to execute due to the minimum, maximum, and step values of the parametric simulation can be directly indicated in the simulator graphical interface (or using a directive). It is only required to indicate in the simulator the same values of the parameter to be swept with the largest amount of requested simulations. Similarly to the MC simulation case, the parameter sweep will be executed cyclically if the number of requested simulations are exceeded for the indicated range.

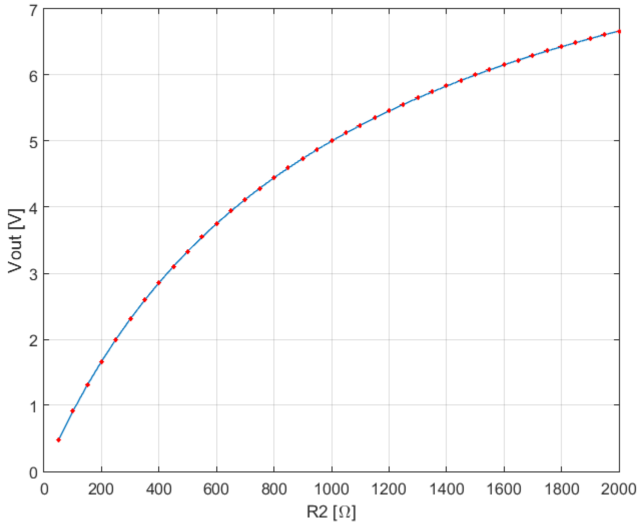
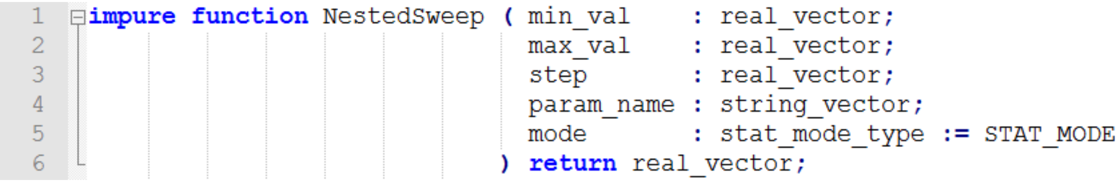


Figure E.10: RVD output voltage (`Vout`) vs. `R2` resistor value. Operating point parametric sweep simulation result.

Although the previously defined `Sweep` functions allow multiple parameter sweeps, it can be easily noticed that these functions do not support nested parameter sweeping. The nested parameter sweeping requires the simulation every single possible value of one parameter with every single possible value of other parameters that are being swept. The algorithm is implemented in the `NestedSweep` function included in the `PARAMETRIC_STATISTICAL_PKG` package, see the function declaration in Figure E.11.



```
impure function NestedSweep ( min_val    : real_vector;
                              max_val    : real_vector;
                              step       : real_vector;
                              param_name : string_vector;
                              mode       : stat_mode_type := STAT_MODE
                            ) return real_vector;
```

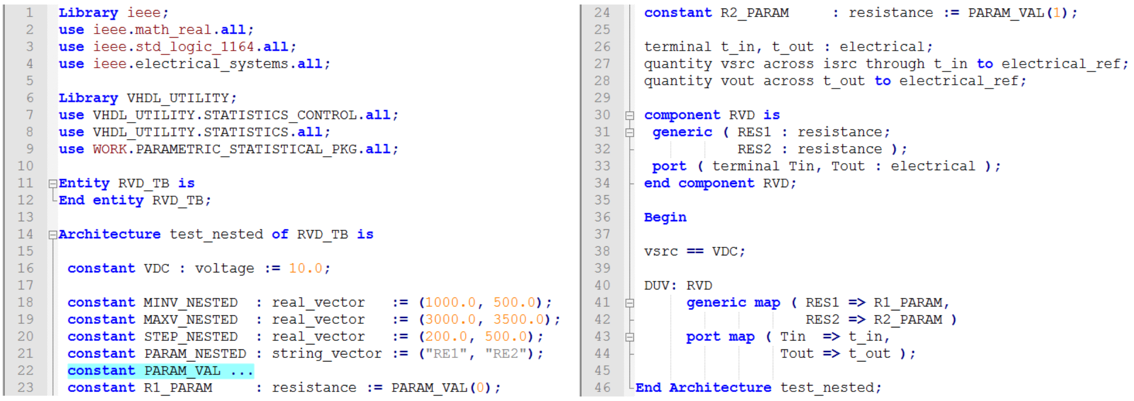
Figure E.11: NestedSweep function declaration for MC simulations.

The `NestedSweep` function receives as arguments the `min_val`, `max_val`, and `step` real vectors. The `min_val` vector contains all the minimum (initial) values of the arguments to be swept, the `max_val` vector contains all the maximum (stop) values of the arguments to be swept, and the `step` vector contains all the step values of the arguments to be swept. The order of the real parameter values in these vectors must correspond to the order of the parameters entered in the `param_name` vector, which is a string vector that contains the names of the parameters to be swept. Since the elements of a composite type must be constrained in VHDL, the size of the subtype `STRING` must be constrained. `STRING` is an array of characters in VHDL; therefore, the string vector is defined as follows:

```
type STRING_VECTOR is array (NATURAL range <>) of STRING(1 to 3);
```

Consequently, the parameter names must be exactly of 3 characters. Additionally, the function creates a text control file (called *NestedCounter.dat*), which contains the number of simulations to perform for each parameter, they are stored as integer numbers in the file, one number per line. The number of MC simulations to run can be easily calculated by multiplying all the numbers of the *NestedCounter.dat* file minus one: (C1*C2*...Cn - 1) where Ci is the integer value in each row of the *NestedCounter.dat* file.

In order to explain how to use the `NestedSweep` function, let us consider the same RVD example on Figure E.1 on page 212. In this case, we want to perform a nested sweep simulation on `R1` and `R2` resistance values. `R1` will be swept from 1 kΩ to 3 kΩ in steps of 200 Ω, whilst `R2` will be swept from 500 Ω to 3.5 kΩ in steps of 500 Ω. The *test bench* code shown in Figure E.12 contains the constant declarations of the arguments required by the `NestedSweep` function in this example: the string vector `PARAM_NESTED` and the 3 real vectors `MINV_NESTED`, `MAXV_NESTED`, and `STEP_NESTED`, see lines 18 to 21.



```
 1  Library ieee;
 2  use ieee.math_real.all;
 3  use ieee.std_logic_1164.all;
 4  use ieee.electrical_systems.all;
 5
 6  Library VHDL_UTILITY;
 7  use VHDL_UTILITY.STATISTICS_CONTROL.all;
 8  use VHDL_UTILITY.STATISTICS.all;
 9  use WORK.PARAMETRIC_STATISTICAL_PKG.all;
10
11  Entity RVD_TB is
12  End entity RVD_TB;
13
14  Architecture test_nested of RVD_TB is
15
16   constant VDC : voltage := 10.0;
17
18   constant MINV_NESTED  : real_vector   := (1000.0, 500.0);
19   constant MAXV_NESTED  : real_vector   := (3000.0, 3500.0);
20   constant STEP_NESTED  : real_vector   := (200.0, 500.0);
21   constant PARAM_NESTED : string_vector := ("RE1", "RE2");
22   constant PARAM_VAL ...
23   constant R1_PARAM     : resistance := PARAM_VAL(0);
24   constant R2_PARAM     : resistance := PARAM_VAL(1);
25
26   terminal t_in, t_out : electrical;
27   quantity vsrc across isrc through t_in to electrical_ref;
28   quantity vout across t_out to electrical_ref;
29
30   component RVD is
31    generic ( RES1 : resistance;
32              RES2 : resistance );
33    port ( terminal Tin, Tout : electrical );
34   end component RVD;
35
36   Begin
37
38   vsrc == VDC;
39
40   DUV: RVD
41        generic map ( RES1 => R1_PARAM,
42                      RES2 => R2_PARAM )
43        port map ( Tin  => t_in,
44                   Tout => t_out );
45
46  End Architecture test_nested;
```

Figure E.12: `test_nested` architecture of the RVD Test Bench.

The `NestedSweep` function returns a real vector which contains the swept values of the parameters given by the `PARAM_NESTED` vector in the same order. The declaration of the constant which contains those values (`PARAM_VAL`) is highlighted in line 22, the `NestedSweep` function must be used in this line as follows:

```
constant PARAM_VAL : real_vector :=
WORK.PARAMETRIC_STATISTICAL_PKG.NestedSweep(MINV_NESTED, MAXV_NESTED,
STEP_NESTED, PARAM_NESTED);
```

Lines 23 and 24 show how the swept of `R1` and `R2` resistances can be obtained from the `PARAM_VAL` vector. It is worth mentioning that these vector constants can be directly used in the generic map of the RVD component or in any other instruction, without creating the additional constants `R1_PARAM` and `R2_PARAM`. These constants are used only for illustrative purpose.

Finally, the integer values contained in the *NestedCounter.dat* file are 11 and 7, meaning that the amount of required MC simulation runs is 76. After post-processing, the simulation results can be observed in the surface plot shown in Figure E.13.

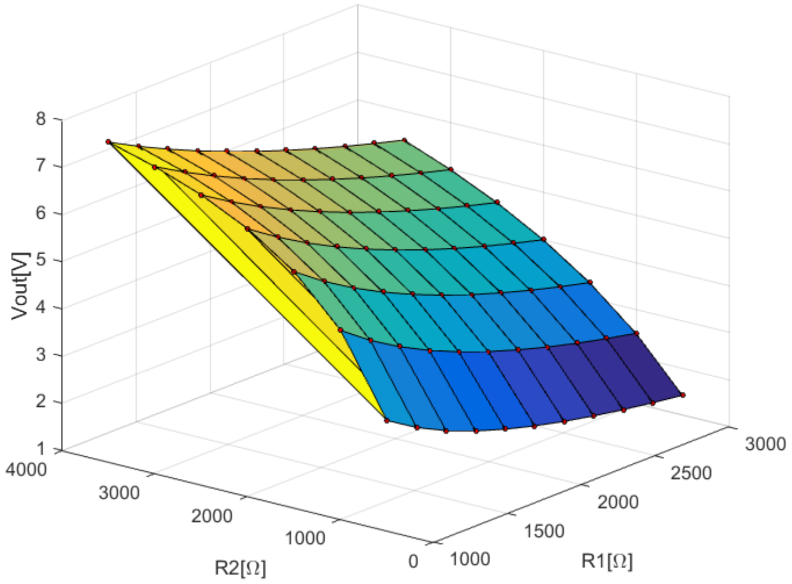


Figure E.13: Surface plot of the output voltage `Vout` and the swept values `R1` and `R2` in the RVD circuit example. Operating point nested parametric sweep simulation result.

## E.3 Defining custom statistical distributions

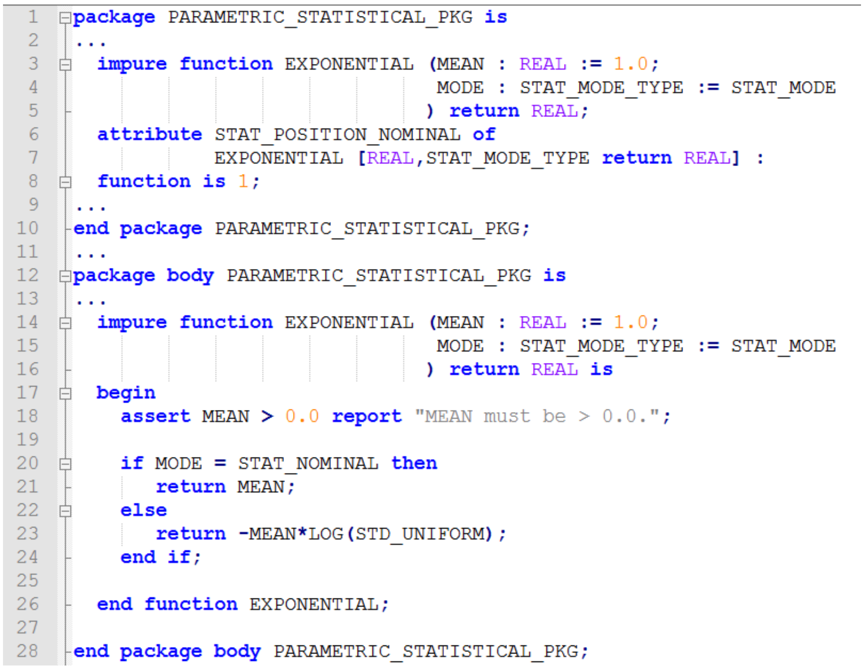
One of the most useful features of the SAE packages is the option to define new statistical distributions by using the standard distribution functions as building blocks. The standard distribution functions return a random value on a normalized scale according to the statistical distribution implemented by the function. The SAE J27748 provides 8 standard distribution functions, they are fully described in [131].

The user custom statistical distributions should not be added to the SAE packages, the `PARAMETRIC_STATISTICAL_PKG` package has been created for this purpose. In Figure E.14 we observe an example of how an exponential distribution function can be created. The `MEAN` value of the distribution is returned if the `MODE` flag is set to return a nominal value by the deferred control constant `STAT_NOMINAL`. The implementation of the exponential function uses the standard distribution function `STD_UNIFORM` from the `STATISTICS` package.

Using the RVD circuit example used for the regular distribution functions of the SAE package, under the same conditions provided in the test bench code in Figure E.2 on page 213, the MC simulation results for `R2` resistor value and the `Vout` RVD output voltage are depicted in Figure E.15. The random variable `R2_RAND` which is used to call the `EXPONENTIAL` function is declared as follows:

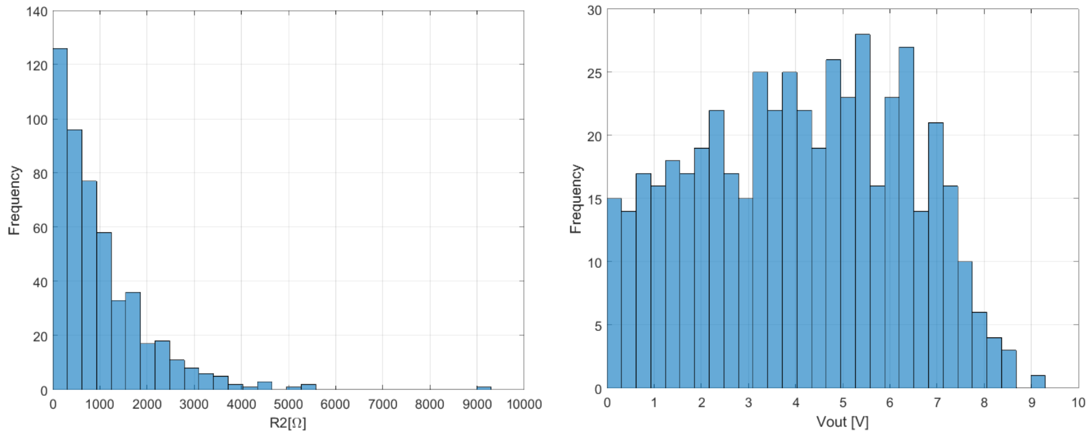
```
constant R2_RAND : resistance :=
WORK.PARAMETRIC_STATISTICAL_PKG.EXPONENTIAL(R2_NOM);
```

In this case, we can observe clearly that `Vout` does not present the same exponential behavior as `R2`. New statistical distributions can be defined by the combination of several types of basic statistical distributions and the interaction of additional variables in the system.



```
package PARAMETRIC_STATISTICAL_PKG is
...
  impure function EXPONENTIAL (MEAN : REAL := 1.0;
                               MODE : STAT_MODE_TYPE := STAT_MODE
                              ) return REAL;
  attribute STAT_POSITION_NOMINAL of
            EXPONENTIAL [REAL,STAT_MODE_TYPE return REAL] :
  function is 1;
...
end package PARAMETRIC_STATISTICAL_PKG;
...
package body PARAMETRIC_STATISTICAL_PKG is
...
  impure function EXPONENTIAL (MEAN : REAL := 1.0;
                               MODE : STAT_MODE_TYPE := STAT_MODE
                              ) return REAL is
  begin
    assert MEAN > 0.0 report "MEAN must be > 0.0.";

    if MODE = STAT_NOMINAL then
       return MEAN;
    else
       return -MEAN*LOG(STD_UNIFORM);
    end if;

  end function EXPONENTIAL;

end package body PARAMETRIC_STATISTICAL_PKG;
```

Figure E.14: Exponential distribution function. Custom statistical distribution.



(a)

(b)

Figure E.15: Exponential distribution histograms. (a) R2 resistor value. (b) RVD output voltage.

# F Appendix F - VP-Model Library implementation

This appendix describes the implementation, deployment, and the detailed infrastructure of the proposed VP-Model Library database.

## F.1 Database implementation and deployment

The prototype of the VP-Model Library has been created in a commercial database software (Filemaker Pro 12[1]) to demonstrate, explain, and investigate the library scope, benefits, and limitations. One of the main challenges for the implementation of the VP-Model Library is the construction of a rather complex relational database, see Figure F.1. The VP-Model Library database consists of a set of tables that are dedicated to storing model metadata according to the structure shown in Figure F.1. Each table uses an identifier (ID) record that allows to create equality (=) or Cartesian product (X) relationships for correct navigation and data displaying functionality. The related data among two or more tables is displayed through scroll-down window portals and select buttons in the database views, see section F.2.

The prototype of the VP-Model Library has been deployed on a single Windows machine using Filemaker Server 12 so that the access to the database and its file repository is on-line by using any Internet browser. Therefore, the VP-Model Library is independent of any *simulation framework* software.

Filemaker Server allows administrating the database remotely by using a simple graphical user interface (GUI). The method to publish the database on-line is called *instant web publishing* (IWP), which allow to publish several databases using the same database homepage, see Figure F.2 on page 225. Full access database administrators have their own password-protected account which allows them to enable/disable and update database files from any computer.

### F.1.1 VP-Model Library actors

The VP-Model Library need to be used and manipulated by three main actors in order to support the VP-based design methodology implementation for CPS development, presented in chapter 5 on page 133. Therefore, three fundamental types of privilege sets are available in the VP-Model Library. Each privilege set is defined by a set of permissions that can be used to create password protected access accounts to the VP-Model Library. The three main actors and their privilege sets are described as follows:

---

[1] http://www.filemaker.com/

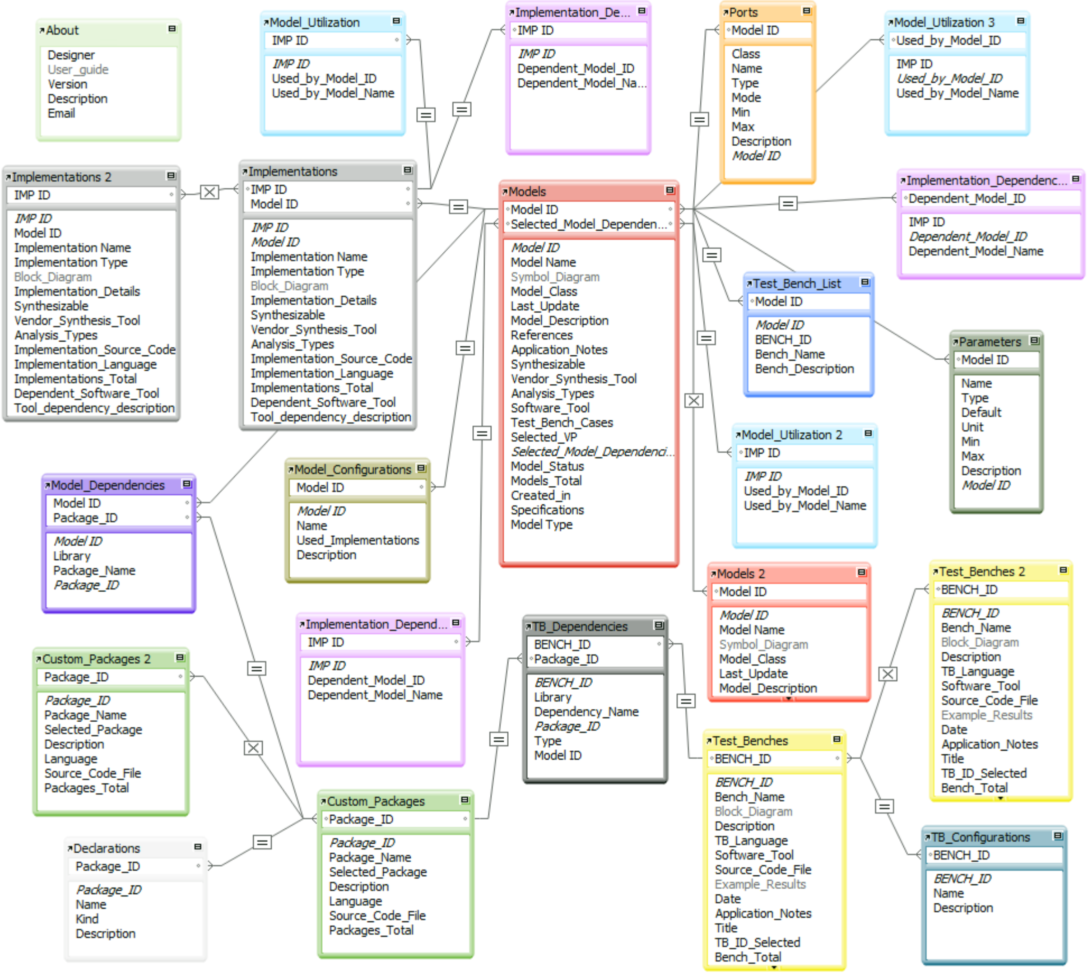


Figure F.1: Relationship graph of the VP-Model Library database. Each table of the database is represented by a different color; duplicated tables share the same color.

**Administrators:** The administrators are responsible for the VP-Model Library operation, update, and maintenance. This applies not only to the database files but also for the file repository structure and its content. Only the administrators can modify the library content. Therefore, administrators possess a **full-access** privilege set, meaning that they have unrestricted access to any content in the database. Two types of administrators can be distinguished: **infrastructure administrators**, which are responsible for the library structural maintenance and design; and the **content administrators**, which are responsible for populating and maintaining the library content as is explained in section 5.4 on page 155.

**Contributors:** A contributor is a person (active system or subsystem-level designer) who elaborates, verifies, documents, and/or requests VHDL-AMS *modeling elements* at any *abstraction level*. In order to avoid misuse of the library services and errors in the data entry, the contributors cannot modify directly the VP-Model Library database, they have the same access privileges than VP-Model Library users. Instead, the contributors send their model contributions and/or model requests by using an MRF. An MRF is a light copy of the VP-Model Library which contains few model entries as examples, and database **write privileges** for contributors, i.e. contributors can include new model entries with their related metadata using an MRF dedicated for each contributor, see Figure F.2. MRFs are used for both submitting modeling requests and for collecting all type of *modeling elements* and their respective documentation.

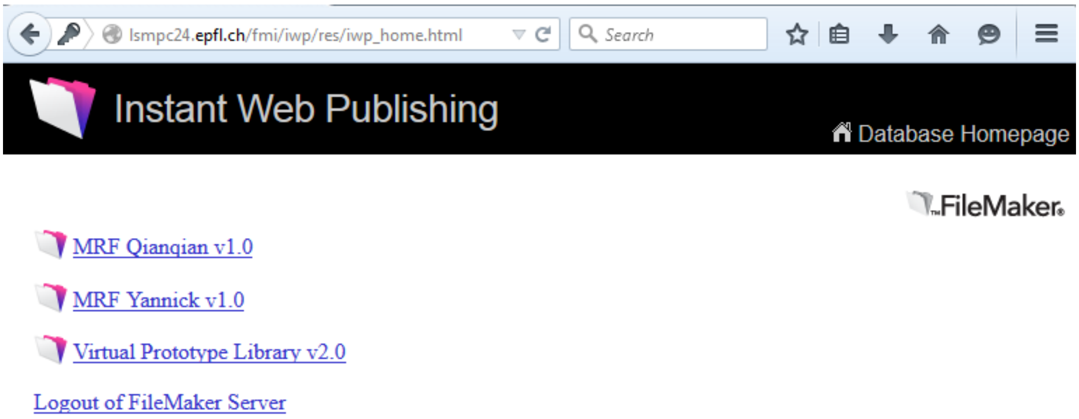

lsmpc24.epfl.ch/fmi/iwp/res/iwp_home.html
Search
Instant Web Publishing
Database Homepage
FileMaker
MRF Qianqian v1.0
MRF Yannick v1.0
Virtual Prototype Library v2.0
Logout of FileMaker Server

Figure F.2: Instant Web Publishing home page.

**Users:** A VP-Model Library user has **read-only** access privileges to the database and its file repository. The users can browse, query, sort records, and download model documentation from the VP-Model Library views. Users cannot create, modify or delete any information in the database. They only have permissions to modify certain fields used for navigation purposes.

### F.1.2 IP security

The VP-Model Library database offers the capability to reuse and share models and their related IP not only among internal company designers and managers but also outside the company. This is why IP protection is a fundamental requirement for the library utilization.

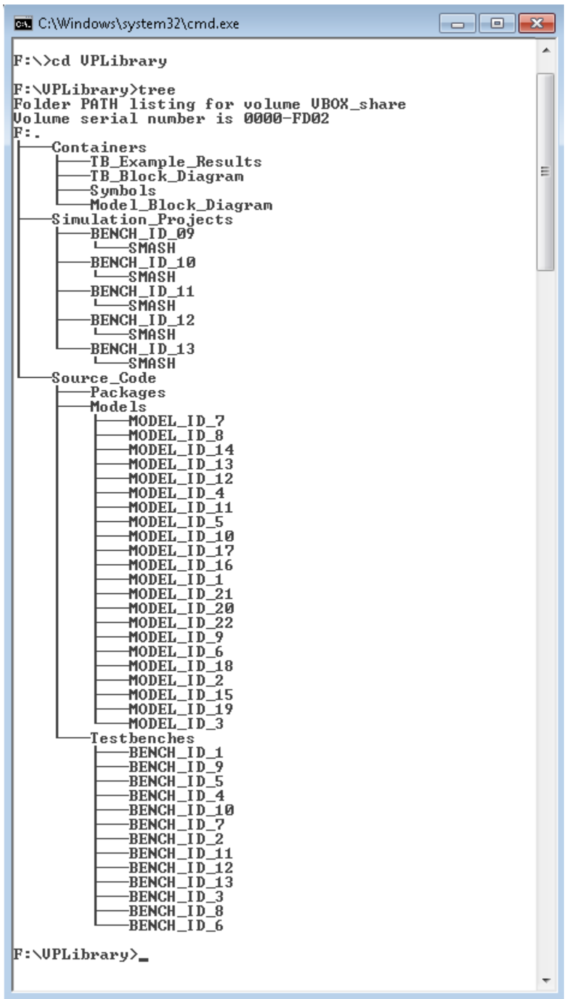
VP-Model Library administrators can create new privilege sets based on the three main types given in the previous subsection. Different access accounts, always password protected, can be created by modifying their privilege sets. For example, suppose that the company has two types of users of the VP-Model Library: external and internal users. The internal users have read access to all the models contained in the library. But, suppose that the company wants to limit the access to some models of the library to external users. In this case, a new account can be created which uses a modified privilege set based on the **Users** main set described in subsection F.1.1.

Since the VP-Model Library is an on-line database, it is vulnerable to hacker attacks as any other on-line infrastructure. Therefore, if the security threat is a top priority for the company, strong server authentication and encryption protections are recommended. However, this is out of the scope of this research.

### F.1.3 File repository setup

In order to better understand the structure of the file repository of the model library explained in this subsection, it is recommended to revise the VP-Model Library structure described in section F.2 on page 227 before reading this subsection.

All the database model metafiles such as symbols, source codes, documentation and project files are stored in a Linux-based Apache (Red Hat) HTTP server. This folder is password protected, and it uses the same login data of the VP-Model Library. The organization of the VP-Model Library folder is shown in Figure F.3. The structure consists of three main folders: `Containers`, `Simulation_Projects`, and `Source_Code`.



```
F:\>cd VPLibrary

F:\VPLibrary>tree
Folder PATH listing for volume VBOX_share
Volume serial number is 0000-FD02
F:.
├───Containers
│   ├───TB_Example_Results
│   ├───TB_Block_Diagram
│   ├───Symbols
│   └───Model_Block_Diagram
├───Simulation_Projects
│   ├───BENCH_ID_09
│   │   └───SMASH
│   ├───BENCH_ID_10
│   │   └───SMASH
│   ├───BENCH_ID_11
│   │   └───SMASH
│   ├───BENCH_ID_12
│   │   └───SMASH
│   └───BENCH_ID_13
│       └───SMASH
└───Source_Code
    ├───Packages
    ├───Models
    │   ├───MODEL_ID_7
    │   ├───MODEL_ID_8
    │   ├───MODEL_ID_14
    │   ├───MODEL_ID_13
    │   ├───MODEL_ID_12
    │   ├───MODEL_ID_4
    │   ├───MODEL_ID_11
    │   ├───MODEL_ID_5
    │   ├───MODEL_ID_10
    │   ├───MODEL_ID_17
    │   ├───MODEL_ID_16
    │   ├───MODEL_ID_1
    │   ├───MODEL_ID_21
    │   ├───MODEL_ID_20
    │   ├───MODEL_ID_22
    │   ├───MODEL_ID_9
    │   ├───MODEL_ID_6
    │   ├───MODEL_ID_18
    │   ├───MODEL_ID_2
    │   ├───MODEL_ID_15
    │   ├───MODEL_ID_19
    │   └───MODEL_ID_3
    └───Testbenches
        ├───BENCH_ID_1
        ├───BENCH_ID_9
        ├───BENCH_ID_5
        ├───BENCH_ID_4
        ├───BENCH_ID_10
        ├───BENCH_ID_7
        ├───BENCH_ID_2
        ├───BENCH_ID_11
        ├───BENCH_ID_12
        ├───BENCH_ID_13
        ├───BENCH_ID_3
        ├───BENCH_ID_8
        └───BENCH_ID_6

F:\VPLibrary>_
```

Figure F.3: Model_library folder organization example.

Firstly, the `Containers` folder is dedicated to storing the symbols, the block diagrams and any other image of the VP-Model Library. The `Containers` folder has 4 subfolders as it is shown in Figure F.3. The *Symbols* folder contains all the Symbol diagrams of the models in the library database. The image must be stored using the `MODEL_ID` number of the respective model record. The `Model_Block_Diagram` folder contains the block diagram images of the model implementations (architectures) in the database. The image must be stored using the Implementation number `IMP_ID` of the respective implementation (architecture). The `TB_Block_Diagram` folder contains the images of the block diagram of the associated *test bench*. The image must be stored using the `Bench_ID` number of the respective *test bench*. Finally, The `TB_Example_Results` folder contains the images of relevant simulation results of the associated *test bench*. The images must be stored using the `Bench_ID` number of the respective *test bench*.

Secondly, for storing simulation project files, the database administrators must create a folder inside the `Simulation_Projects` folder by following this naming rule for the new folder and its subfolders:

`/BENCH_ID_N/Software_tool/Bench_name/`

where `N` is the respective `Bench_ID` number, `Software_tool` is the name of the software tool used for the test bench simulation and `Bench_name` is the name of the respective *test bench*.

Finally, the source code files of the library models are stored in the `Source_Code` folder. For the different code files of the library, it has been chosen a practical organization which consists of 3 main sub-folders shown in Figure F.3. Each of these folders represents a library view: In the folder `Packages`, the package files are stored using the unique `PACKAGE_ID` number at the beginning of the file name for every single package file. The folder `Testbenches` is dedicated to storing *test bench* files. There is a folder for every single *test bench* record in the database, which is named by using the unique `BENCH_ID` identifier. Inside these folders must be stored the respective *test bench* file and any other associated file. Similarly, the `Models` folder is dedicated to storing source code files for every single model record in the database. A folder using the unique `MODEL_ID` identifier as name, see Figure F.3, is used to store de source code of all the model implementations related to the respective model record. The source code of the models inside these folders is characterized by a unique Implementation ID number (`IMP_ID`) which is binding the file which its respective location in the database.

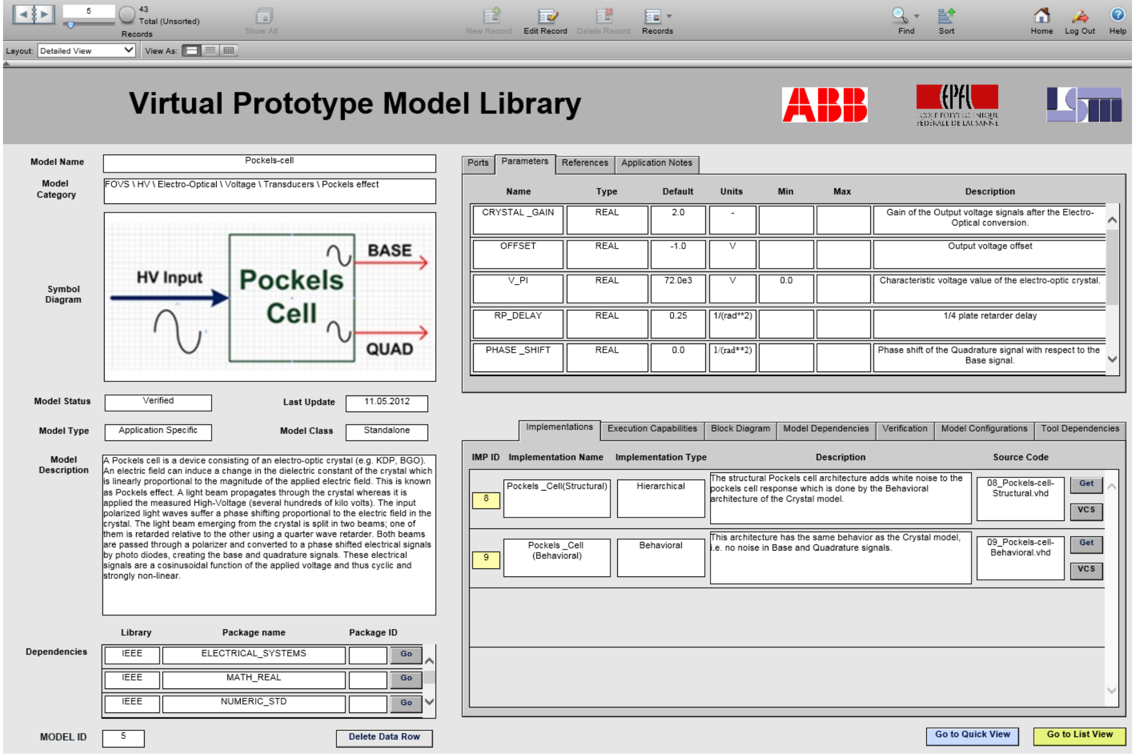
### F.1.3.1 Version Control System

The history of all the documents related to the models (i.e. source code, documentation, pictures, etc.) is managed by the version control system (VCS) software (GIT - http://git-scm.com/). It is highly recommended for VP-Model Library users to have their own version controlling application in for their simulation projects using the same or similar tools. The version tracking system is independent of the database software. The VP-Model Library provides a link to the latest version of the source code files. It is also desired to count with a link to the history of the source code files from the VP-Model Library; however, since the file history is managed externally by GIT software, this was not fully implemented.

## F.2 VP-Model Library structure

The metadata of the *component models*, VPs, and their verification models and files (i.e. *test benches*, packages, and configurations), are stored in tables following the structure displayed in Figure F.1 on page 224. The metadata is accessed via database views, which are data interfaces holding alphanumeric fields, graphical fields, portal and tab windows for displaying related metadata. The views of the VP-Model Library are explained in the three following subsections.

## F.2.1 Model views



5 / 43 Total (Unsorted) Records

Layout: Detailed View

View As:

Virtual Prototype Model Library

**Model Name:** Pockels-cell

**Model Category:** FOVS \ HV \ Electro-Optical \ Voltage \ Transducers \ Pockels effect

**Symbol Diagram**

**Model Status:** Verified

**Last Update:** 11.05.2012

**Model Type:** Application Specific

**Model Class:** Standalone

**Model Description:** A Pockels cell is a device consisting of an electro-optic crystal (e.g. KDP, BGO). An electric field can induce a change in the dielectric constant of the crystal which is linearly proportional to the magnitude of the applied electric field. This is known as Pockels effect. A light beam propagates through the crystal whereas it is applied the measured High-Voltage (several hundreds of kilo volts). The input polarized light waves suffer a phase shifting proportional to the electric field in the crystal. The light beam emerging from the crystal is split in two beams; one of them is retarded relative to the other using a quarter wave retarder. Both beams are passed through a polarizer and converted to a phase shifted electrical signals by photo diodes, creating the base and quadrature signals. These electrical signals are a cosinusoidal function of the applied voltage and thus cyclic and strongly non-linear.

**Dependencies**

| Library | Package name | Package ID |
|---|---|---|
| IEEE | ELECTRICAL_SYSTEMS | |
| IEEE | MATH_REAL | |
| IEEE | NUMERIC_STD | |

**MODEL ID:** 5

| Name | Type | Default | Units | Min | Max | Description |
|---|---|---|---|---|---|---|
| CRYSTAL _GAIN | REAL | 2.0 | - | | | Gain of the Output voltage signals after the Electro-Optical conversion. |
| OFFSET | REAL | -1.0 | V | | | Output voltage offset |
| V_PI | REAL | 72.0e3 | V | 0.0 | | Characteristic voltage value of the electro-optic crystal. |
| RP_DELAY | REAL | 0.25 | 1/(rad**2) | | | 1/4 plate retarder delay |
| PHASE _SHIFT | REAL | 0.0 | 1/(rad**2) | | | Phase shift of the Quadrature signal with respect to the Base signal. |

| IMP ID | Implementation Name | Implementation Type | Description | Source Code |
|---|---|---|---|---|
| 8 | Pockels _Cell(Structural) | Hierarchical | The structural Pockels cell architecture adds white noise to the pockels cell response which is done by the Behavioral architecture of the Crystal model. | 08_Pockels-cell-Structural.vhd |
| 9 | Pockels _Cell (Behavioral) | Behavioral | This architecture has the same behavior as the Crystal model, i.e. no noise in Base and Quadrature signals. | 09_Pockels-cell-Behavioral.vhd |

Figure F.4: Model detailed view example (Pockels-cell).

The model records and its related metadata are displayed in the Model views. The VP-Model Library is implemented with three model views: **list view**, **quick view**, and **detailed view**. The list and quick views are dedicated to showing summarized data, e.g. the name and description of the model, in order to facilitate a fast search of models. The detailed view shows the complete model information, see Figure F.4. Each model record can be dedicated to a simple *component model* or to a more complex VP; it is important mentioning that the VP-Model Library can link the metadata of several *component models* records used in a more complex hierarchical *component model* or VP.

From the VHDL-AMS perspective, a model record in these views defines a design entity with a well-defined interface, i.e. an entity declaration with a set of well-defined generic parameters and ports (*model signature*). Similar *model implementations* that model the same component or system, but using different port interfaces (disregarding the model parameters), need to be included in separated model records. For example, an ADC can be modeled at particular *refinement levels* by using different system architectures such as successive approximation, sigma-delta, direct conversion, etc. Therefore, the ADC model can consist of several *model implementations* related to the same model record if they share the same port interface; otherwise, another model record is required for storing the different *model implementations*. If the entity of different *model implementations* only differs in the number and type of generic parameters, they can be included in the same model record. In this case, the *model signature* will contain the union of the generic parameters of all *model implementations*.

The example given in Figure F.4 shows a model with two implementations. Each model record has a unique positive integer identifier number called `MODEL_ID`, see left-bottom part of the figure. Likewise, each *model implementation* has a unique positive integer identifier number called `IMP_ID` as it is shown in the figure. The description of the fields and portals of the detailed view is given as follows:

1. **Model Name:** Complete and self-explicative name of the Model. Since a model can have several implementations, the Model Name must be a general name which encloses all the possible implementations of that model. An appropriate generic name, related to the function of such model, is the principal criterion to search for models.

2. **Symbol Diagram:** This is an abstract diagram of the model that shows the port interface of the model represented as a *'black-box'*. All the interface ports must be clearly depicted in the symbol.

3. **Model Category:** It refers to the classification of the model inside the library structure; it represent a list of keywords of the model. This field is important for model searching together with the Model Name field. In order to facilitate the searching process of the model, the Model Category field contains one or more keywords related to the domain, functionalities, algorithms, and type of the device/system that is modeled. The order of the keywords is not relevant. For instance, for a specific ADC model that implements the successive approximation algorithm, the keywords could be: \ Electrical \ Mixed-signal \ Converters \ Successive approximation.

4. **Model Status:** This field indicates the status of the model according to its level of development. A series of accumulative levels classify the model in a certain status, meaning that it complies with the characteristics of that level and all the previous levels from the most basic status defined in the Model Library. The model status levels available in MRFs and in the VP-Model Library are listed as follows:

    (a) **Requested:** It means that the model does not exist yet, but the desired function and interface can be described and requested using an MRF. In this way, the MRFs can be used as a bidirectional communication channel between system/subsystem designers and the Model maintenance process, see section 5.4 on page 155. This status is only used in MRFs for Contributors or Administrators.

    (b) **In development:** This status is meant for unfinished models or finished but not verified models. Since the VP-Model Library only contains *conditioned models*, models with this status only can appear in MRFs. If an incomplete/problematic model needs to be shared (e.g. for analysis and correction purposes), the model can be documented in an MRF. If the model is finished but the verification is executed by another person, the model can be documented and shared using an MRF.

    (c) **Verified:** In this status, at least the main functionality of the model has been verified in simulation. In a simple model, a complete functional verification is done by properly identifying the operational range of the interface ports and the parameters of the model. The main cause of simulation failure/errors when building complex hierarchical models by aggregation of simpler models, is the simulation out of the operational range of the model variables. This is why it is highly desirable to perform a robust functional verification of the model before performing further verifications towards the device/system validation. A verified model must include one or more *test benches* properly documented and stored in the VP-Model Library, see more details in subsection F.2.2 on page 239.

   (d) **Validated:** In this status, the model has a high development process, it has been both verified extensively and validated against experimental data. Normally, these are highly refined models that pass through a *model validation* process using physical prototypes for experimental measurements. The model accuracy and precision can be estimated when the simulation and experimental conditions very close.

5. **Last update:** Date of the last update of the model. This date must correspond to the current version of the model.

6. **Model Type:** This field indicates the type of the model from the more simple and general to the more complex and specific type of model. The classification considers three types:

   (a) **Utility:** These type of models describe very basic (ideal) behaviors which are essential for the operation of *test benches* or other hierarchical models; these models are **building blocks**. For instance, passive circuit elements such as resistors, inductors or capacitors, which can be used to create more complex models. Likewise, these models can also be functional blocks for producing excitation signals or measuring output signals of a design under verification (DUV) in a *test bench*; for example, signal sources or standard signal processing units. The sole condition of a model to be classified as `Utility` is that it has a high probability to be reused for different applications. Since the utility models possess a simple functionality and they are needed to build *test benches* for another type of models, they do not require a dedicated *test bench* to verify them exclusively.

   (b) **Application Specific:** As it name suggest, these type of models have been created to be used in a specific application. The application range of these models is less general than the utility models. Some models can have more potential to be reused than others; therefore, they are less `Application Specific`. In principle, every single model in the library has a potential to be completely or partially reused. Frequently, the size and the complexity of the model is directly proportional to the specificity of the model.

   (c) **General Application:** This type of models can be used in different applications for several purposes. General purpose models can describe more complex behaviors and functionalities than the Utility models; for example, digital or analog components such as converters, amplifiers, adders, multipliers, integrators, etc. These models can have any implementation type, i.e. they can be functional, behavioral, physical, or hierarchical models.

   (d) **Virtual Prototype:** These are hierarchical models composed of several *component models* which execute multiple specific applications for producing a particular functionality. These models can be very complex in terms of interconnections and number of parameters (not necessarily in the number of components), they represent the complete system to be designed that is often an important part of a CPS or a complete CPS.

7. **Model Class:** This field indicates the class of the model based on two class criteria:

   (a) **Dependent:** These models are part of a bigger model/architecture that carries out a specific task or represents certain behavior. Dependent models do not make sense if they are used alone or without their correspondent co-models. For example, the Thermal Coupling model of the electro-thermal Rogowski coil model (ETRCM) (see Figure 4.4 on page 58) cannot be used alone or connected with a different set of models. Dependent models are always Application specific model types, but Application specific models are not necessarily Dependent models.

   (b) **Standalone:** These are independent models that can be used in an autonomous way according to its model type, i.e. they can be used for several applications or they can be only used for specific applications but in different ways. For example, the Busbar Thermal model that is part of the ETRCM is an Application specific and Standalone model that can be used in other architectures or configurations within its application context, i.e. a Standalone model is not automatically tied to any architecture and can be used alone.

8. **Model Description:** This field is dedicated to explaining what the model is about and what are its applications in general terms. A self-explicative summary of the model with its main characteristics should be included. The detailed description of the *model implementations* should not be included in this field, it must be included in the respective description fields in the Implementations portal, see subsection F.2.1.4 on page 234.

#### F.2.1.1 Dependencies

The Dependencies are the set of specific libraries and/or packages that the model requires for a proper operation. The related fields are explained as follows:

1. **Library:** This is the specific name of the VHDL-AMS design library used by the model (at the top level). All the list of standard and non-standard libraries of the model must be included. For custom packages contained in the VP-Model Library, the name displayed in this field is `VP`, it must be properly documented in the Custom Packages view[2], see section F.2.4 on page 244.
2. **Package name:** As its name suggests, this is the name of the package contained in the specified library.
3. **Package ID:** For packages contained in the VP-Model Library database, this is the ID number of the package record. The button 'Go' is a link to the respective package information. Standard libraries such as the IEEE (in VHDL and VHDL-AMS) are not contained in the database.

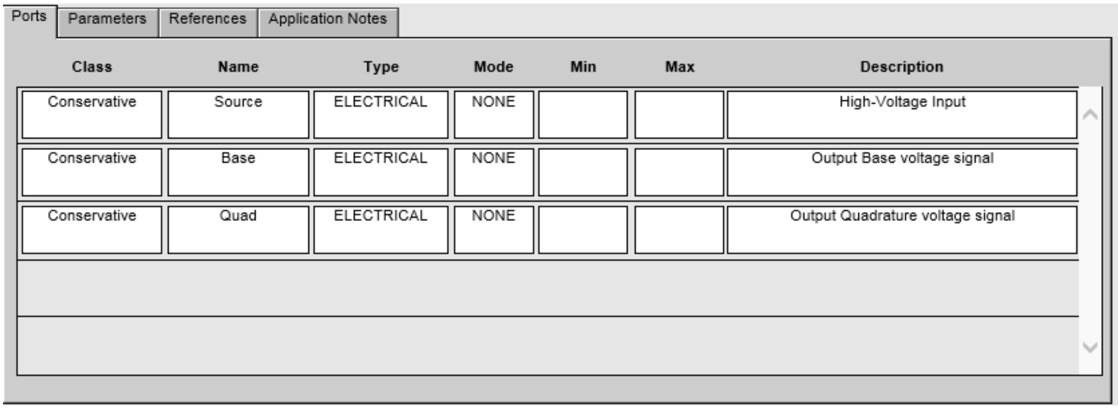
#### F.2.1.2 Ports

The Ports portal contains the list of the port interface of the model. It is important to mention that the field structure of the port interface has been designed to comply VHDL-AMS port specifications. Although the port properties do not always have the same meaning among the different description languages, the port classification proposed here can be used to classify any *computational mockup* interface in any description language supported by the *simulation framework.*

1. **Class:** This refers to the sort/class of port. The following classes are defined based on VHDL-AMS terminology:

---

[2] Custom library names should not use the name **WORK**. The design library WORK is a logical name defined in the IEEE VHDL-AMS standard that refers to a physical design library (some folder or directory) that is hosting the binary codes of compiled design units. Custom design libraries should have more meaningful names.



| Class | Name | Type | Mode | Min | Max | Description |
|---|---|---|---|---|---|---|
| Conservative | Source | ELECTRICAL | NONE | | | High-Voltage Input |
| Conservative | Base | ELECTRICAL | NONE | | | Output Base voltage signal |
| Conservative | Quad | ELECTRICAL | NONE | | | Output Quadrature voltage signal |

Figure F.5: Ports portal corresponding to the Pockels-cell example of Figure F.4.

(a) **Signal-flow:** is the class of port used to represent an analog value such as a voltage or a current level which is a continuous function of time. This class is directly related to quantity ports in VHDL-AMS.

(b) **Conservative:** This port class represents the energy-conservative connection nodes. A key feature of a conservative port is the two values associated with that node: the across and the through values. Any node in a conservative system (e.g. electrical, mechanical, or thermal system), is modeled by two quantities: the potential (also known as the across value) and the flow (also known as the through value). When a component is connected to a conservative port, it can either affect or be affected, by either the potential and/or the flow through the port. In this way, the conservation of the energy is modeled.

(c) **Discrete-event:** This class is normally used to represent digital signals of different nature; e.g. the input clock signal of a synchronous digital block. This class is directly related to the signal ports in VHDL and VHDL-AMS.

2. **Name:** Name of the port as is written in the source code.

3. **Type:** This field keeps the information of the port type (nature) that is normally (always in VHDL-AMS) giving more specific information about the port class. For example, in electrical circuits, an electrical port can be modeled as a conservative terminal port of nature `ELECTRICAL`. It means that to this node is associated an across (voltage) and through (current) values (branch quantities). Likewise, multi-domain conservative and signal-flow ports of different natures (e.g. electrical, thermal, hydraulical) can also be modeled. For Discrete-event ports, the Type is directly related to the values that the digital signal can have; e.g. `BIT`, `STD_LOGIC` or other types. For any other kind of port in VHDL-AMS or any other description language, this field should contain the information of the basic types, e.g. `REAL`, `INTEGER`, `BOOLEAN`, etc.

4. **Mode:** The mode of a specific port can be one of the following: Input, Output, Bidirectional or None. For instance, discrete-event or signal-flow ports can be instantiated as Input, Output, or Bidirectional according to their usage, e.g. a VHDL-AMS digital signal of type `STD_LOGIC`. On the other hand, since conservative ports can affect, or be affected, by the model that is connected, they do not have any predefined direction in reality (None or Bidirectional); yet they can be used to represent the inputs or outputs in

a model more accurately. For instance, the analog input voltage of an ADC can be modeled by a VHDL-AMS terminal port of type electrical. For any conservative VHDL-AMS port, the mode is ‘None’; contrarily, the mode for conservative Verilog-A(MS) ports is ‘Bidirectional’, since the conservative behavior is modeled by bidirectional (inout) ports. Whatever the case may be, the same conservative behavior is represented by None or Bidirectional modes.

5. **Min:** Minimum value(s) of the port. It is desired that the model verifies these values automatically by using VHDL-AMS assertions. The minimum values for the two related quantities in conservative ports must be specified. Not used for discrete-event class ports.
6. **Max:** Maximum value(s) of the port. It is desired that the model verifies these values automatically by using VHDL-AMS assertions. The maximum values for the two related quantities in conservative ports must be specified. Not used for discrete-event class ports.
7. **Description:** The meaning of the port and perhaps, a clarifying expected utilization of the port, are part of a proper description of a port.

#### F.2.1.3 Parameters

The Parameters portal shown in Figure F.4 contains the list of the PMPs. The following is the list of fields dedicated to documenting the model parameters:

1. **Name:** Name of the parameter as is written in the source code.
2. **Type:** Standard data type of the parameter, e.g. `REAL`, `INTEGER`, `POSITIVE`, `BOOLEAN`, etc.
3. **Default:** Default value of the parameter, if any.
4. **Units:** Units of the parameter value. Non-dimensional values are indicated with a dash.
5. **Min:** Minimum value of the parameter, if any. This value can be verified by using VHDL-AMS assertions in the entity declaration or in the model architecture.
6. **Max:** Maximum value of the parameter, if any. This value can be verified by using VHDL-AMS assertions in the entity declaration or in the model architecture.
7. **Description:** Description of the parameter. A complete description explains the meaning of the parameter, its purpose, and how it is used in the model. Parameters can be used not only for modeling functionalities, and behaviors of a real system/component, but they can also be used for controlling particular features of the model abstractions and their structures.

   For instance, suppose that a model has three parameters `A`, `B` and `C`. Let’s assume that parameter `A` acts as a flag, i.e., its default value is ‘infinite’ (i.e., real’high). If `A` is defined with a different value than the default one, the value of the parameter `C` is computed internally from the values of *A* and `B`, while if `A` has its default value, the value of parameter `C` is the default one. Another example could be a configuration parameter that can (statically) select different sets of equations and/or processes (e.g. by using generate statements).

#### F.2.1.4 Implementations

The Implementations portal shown in Figure F.4 contains the list of architectures or implementations which are associated with the model. Since a model record can have several implementations, the Implementations portal has the following fields for the model documentation:

1. **IMP ID:** This ID number is a unique identifier for a particular implementation of the model. This means that a specific *model implementation* is only available for a unique model record, it cannot be shared. However, a model record can have more than one implementation. Likewise, a complete model record (i.e. a model interface with one or more implementations) can be used in other hierarchical models. Therefore, a specific *model implementation* can evolve independently from other *model implementations* that could belong to the same model record. The lack of association between implementations of the same model tends to cause redundancy in the VP-Model Library. Although it is highly desirable to eliminate such redundancy, it can be difficult and unpractical in the long run. Instead, the VP-Model Library offers mechanisms such as unique IDs and bidirectional relationships between interdependent models to deal with redundancy. This number is automatically generated in the database.

2. **Implementation Name:** Name of the model implementation. This name is useful for the identification of specific implementations and their differentiation. This name is practical to distinguish among any implementation in similar models. For example, the VP-Model Library could contain different models of ADCs, where each of those models could have multiple and similar implementations. Based on VHDL-AMS syntax, it is used an “entity_name(architecture_name)” approach for naming, but this is not strictly required as long as the Model Name and the Implementation Name are clear.

3. **Implementation Type:** The VP-Model Library proposes four different Implementation Types according to the *abstraction level* of the model, see section 3.2:
   (a) **Functional:** This is the simplest implementation which describes the main ideal operation of the model.
   (b) **Behavioral:** This implementation type includes the main critical features of the design. It can include first, and possibly second order effects that allow evaluating the system performances, e.g. throughput, bandwidth, temperature, power consumption and so on.
   (c) **Physical:** Implementations of this type are low-level models which include detailed information about physical effects (both external and internal effects), e.g. quantum effects, thermal effects, mechanical effects.
   (d) **Hierarchical:** This implementation describes a hierarchical design which can be formed of one or more sub-models of any of any Implementation Type. This category can include any model record in the VP-Model Library.

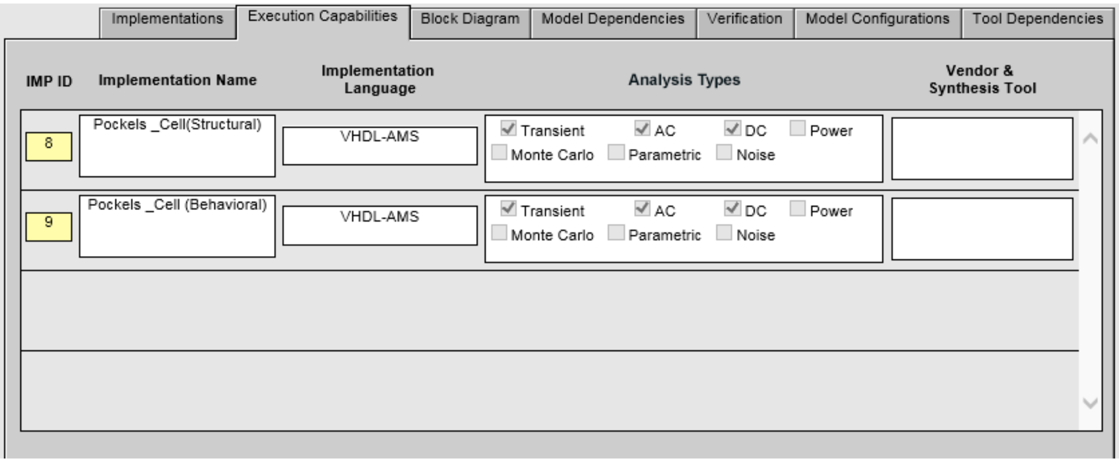
4. **Description:** Detailed description of the *model implementation*, i.e. how the design operates and small details about its operation, functionalities, and behaviors. In other words, it should present an overview of the general behavior of the model, with the main characteristics and features.

5. **Source Code:** This field contains the name of the source code file of the particular *model implementation*. The exact name of the file (including its extension) must be provided.

The button ‘Get’ offers a link to the last version of the file. Since the source code version tracking task is managed independently throughout a GIT repository as it is described in subsection F.1.3.1. The ‘VCS’ button provides a link to the history of the respective file.

### F.2.1.5 Execution Capabilities

Since a model can be used for different types of executions depending on its implementations; the Execution Capabilities portal, see Figure F.6, contains the same list of Model implementations specified in the Implementations portal, organized by the `IMP ID`, the `Implementation Name`, and the following information about the *model implementations*:



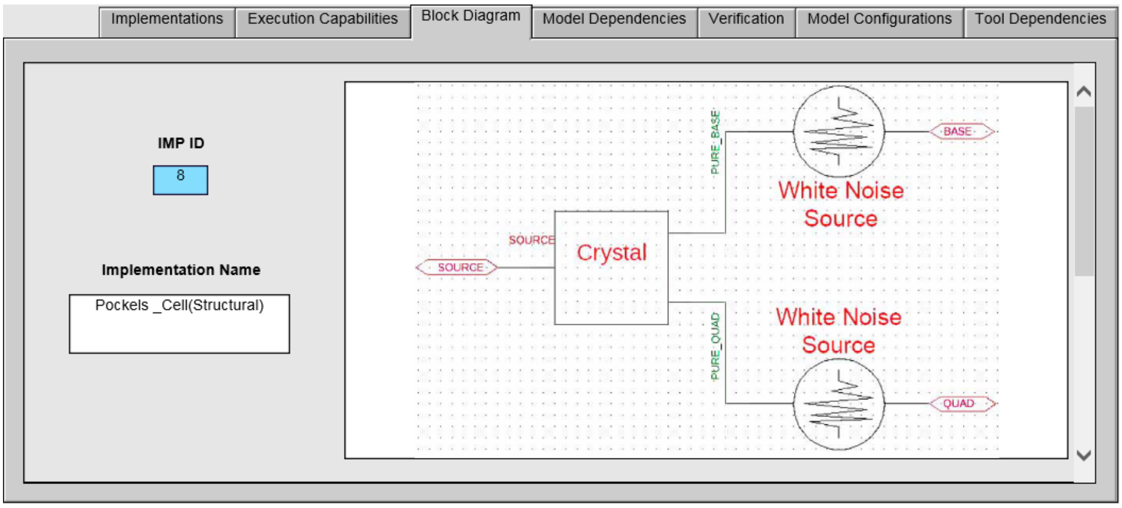
| IMP ID | Implementation Name | Implementation Language | Analysis Types | Vendor & Synthesis Tool |
|---|---|---|---|---|
| 8 | Pockels _Cell(Structural) | VHDL-AMS | ☑ Transient ☑ AC ☑ DC ☐ Power<br>☐ Monte Carlo ☐ Parametric ☐ Noise | |
| 9 | Pockels _Cell (Behavioral) | VHDL-AMS | ☑ Transient ☑ AC ☑ DC ☐ Power<br>☐ Monte Carlo ☐ Parametric ☐ Noise | |

Figure F.6: Execution Capabilities portal corresponding to the Pockels-cell example of Figure F.4.

1. **Implementation Language:** Since VHDL-AMS supports *model implementations* of different languages (e.g. C/C++, SPICE, or Verilog-AMS), this field must contain the modeling language of the specific implementation.
2. **Analysis Types:** It describes the specific type of simulations that the model supports. Any special restriction or configuration used for correct simulation must be provided in the Application Notes text field.
3. **Vendor & Synthesis tool:** If the *model implementation* is synthesizable and has been elaborated using a particular semi-custom technology, this field shall include the information related to the vendor and the software used for synthesis.

### F.2.1.6 Block Diagram

For hierarchical *model implementations*, The Block Diagram portal must show the respective block diagram schematic for each implementation. This portal, see Figure F.7, contains the same list of implementations shown in the implementations portal, organized by the `IMP ID`, the `Implementation Name`, and the block diagram area. It is important mentioning that the portal only shows one entry, for seeing more entries use the scroll bar at the right of the portal window.



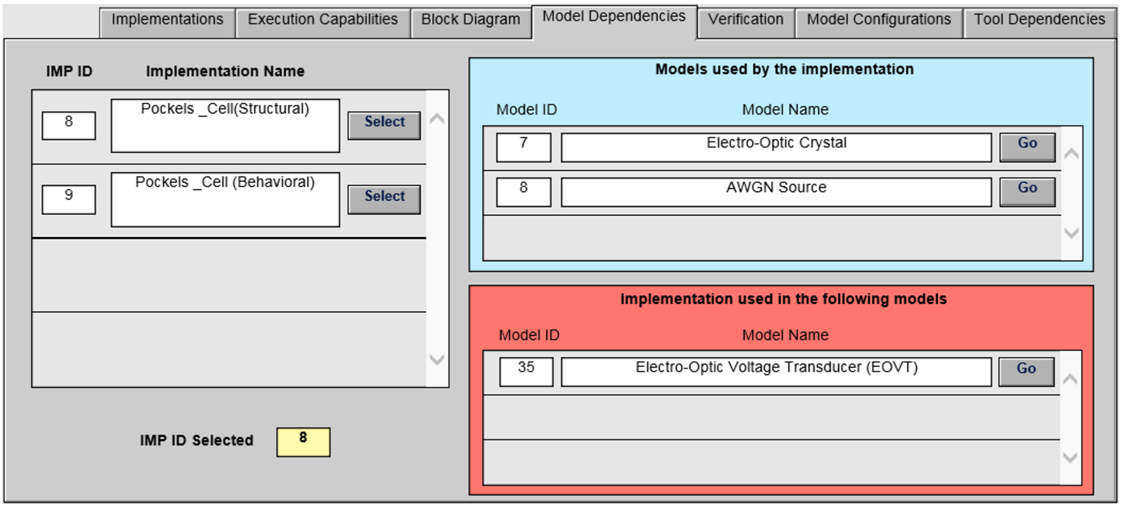
IMP ID: 8

Implementation Name: Pockels _Cell(Structural)

Figure F.7: Block Diagram portal corresponding to the Pockels-cell example of Figure F.4.

### F.2.1.7 Model Dependencies

The Model Dependencies portal indicates all the dependencies that certain implementation has with other models stored in the VP-Model Library. In the right side of the portal, see Figure F.8, we can see two windows, cyan (upper) and magenta (lower): The upper window called **"Models used by the implementation"**, indicates which models are used in the selected implementation, this is true for hierarchical models. On the other hand, the lower window called **"Implementation used in the following models"**, indicates if the selected implementation is used in other models in the library. Keeping track of these dependencies is important for documentation and evolution of complex hierarchical models. Furthermore, this portal helps



| IMP ID | Implementation Name |
|---|---|
| 8 | Pockels _Cell(Structural) |
| 9 | Pockels _Cell (Behavioral) |

IMP ID Selected: 8

**Models used by the implementation**

| Model ID | Model Name |
|---|---|
| 7 | Electro-Optic Crystal |
| 8 | AWGN Source |

**Implementation used in the following models**

| Model ID | Model Name |
|---|---|
| 35 | Electro-Optic Voltage Transducer (EOVT) |

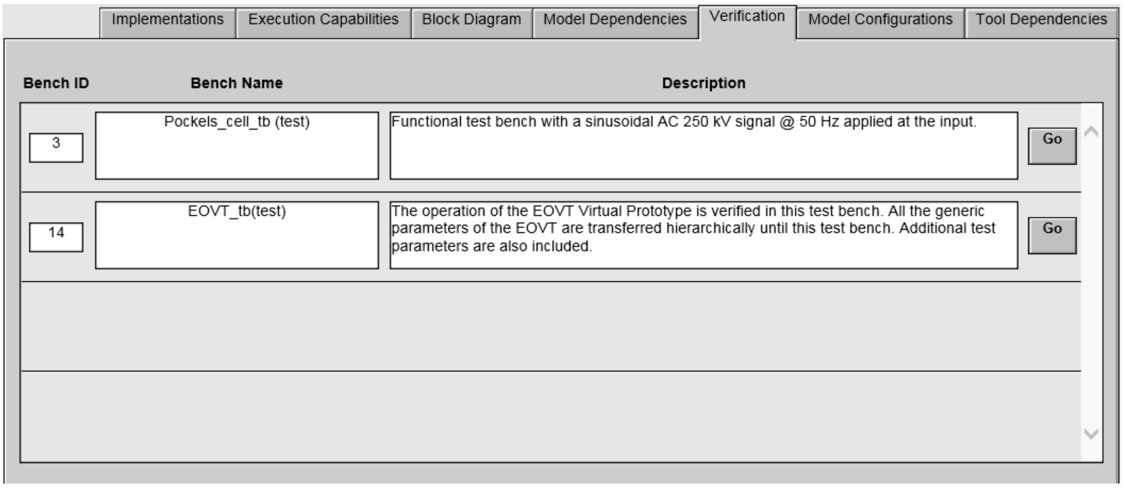
Figure F.8: Model Dependencies portal corresponding to the Pockels-cell example of Figure F.4.

to distinguish among the possible slight variants of a model, which can lead to redundancy. The Model Dependencies portal has been designed for indicating where and how the models are used so that it helps to deal with redundancy.

#### F.2.1.8 Verification

This portal is dedicated to indicating the list of *test benches* in which the current model record is used. The fields in Figure F.9 are explained as follows:

1. **Bench_ID:** This is the ID number of the *test bench* in which the model is tested. The model can be present in one or more *test benches* of the VP-Model Library. All of them must be correctly stored and documented using the Test Benches view, see section F.2.2.
2. **Bench Name:** Name of the *test bench.* This name is the same as the name provided in the Bench Name field in the Test Benches view.
3. **Description:** This is a small description of the *test bench.* Specifically, a small description of what are the type of verifications made on the *test bench.* A broader description of the *test bench* must be included in the corresponding field in the Test Benches view.



| Bench ID | Bench Name | Description |
|---|---|---|
| 3 | Pockels_cell_tb (test) | Functional test bench with a sinusoidal AC 250 kV signal @ 50 Hz applied at the input. |
| 14 | EOVT_tb(test) | The operation of the EOVT Virtual Prototype is verified in this test bench. All the generic parameters of the EOVT are transferred hierarchically until this test bench. Additional test parameters are also included. |

Figure F.9: Verification portal corresponding to the Pockels-cell example of Figure F.4.

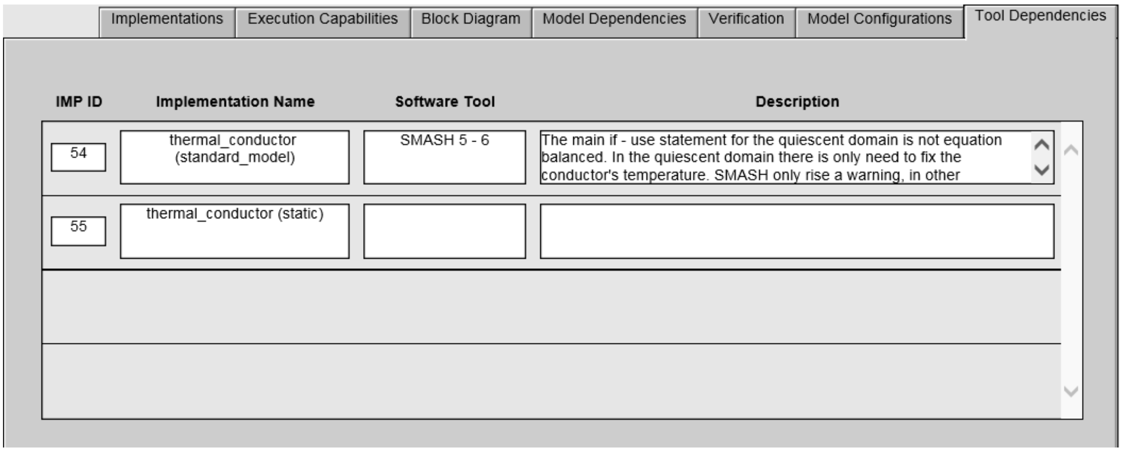
The button 'Go' is a link to the respective *test bench* record in the Test benches view.

#### F.2.1.9 Model Configurations

This portal contains the list of VHDL-AMS configurations that the model might have. Large hierarchical models can have multiple configurations to describe several possible model architectures at different *abstraction levels.* For example, an ADC model could have several implementations to describe several types of ADCs at more than one *abstraction level.* This portal includes the following fields:

1. **Name:** This field contains the name of the configuration. Meaningful names are highly desired instead of short abbreviations.
2. **Description:** This field contains a brief description of the configuration mentioning the particular *model implementations*, packages, and other details that constitute the configuration.
3. **File:** This field contains the name of the particular source code file of the configuration. The exact name of the file (including its extension) shall be provided.

#### F.2.1.10 Tool Dependencies



| IMP ID | Implementation Name | Software Tool | Description |
|---|---|---|---|
| 54 | thermal_conductor (standard_model) | SMASH 5 - 6 | The main if - use statement for the quiescent domain is not equation balanced. In the quiescent domain there is only need to fix the conductor's temperature. SMASH only rise a warning, in other |
| 55 | thermal_conductor (static) | | |

Figure F.10: Tool Dependencies portal corresponding to the Busbar thermal model of the ETRCM example, see Figure 4.4 on page 58.

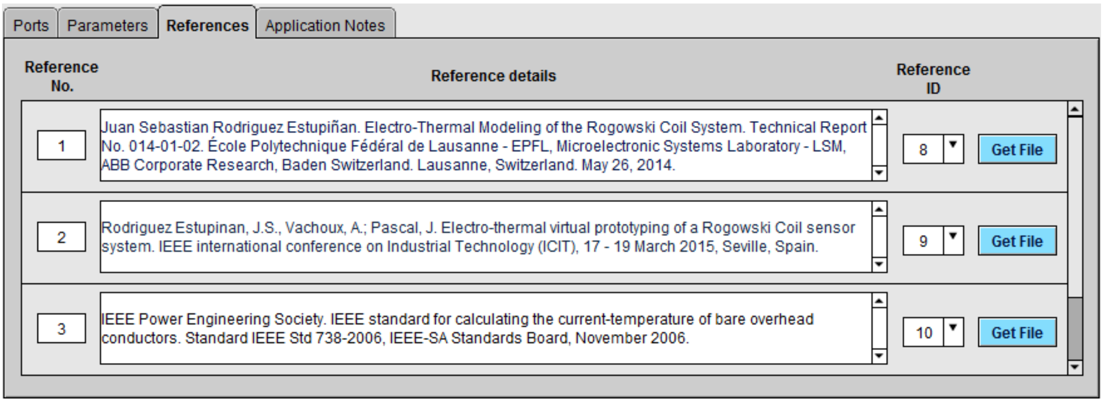
This portal is dedicated to describing the restrictions and/or behaviors of *model implementations* that depend on particular simulation tools. Ideally, the VP-Model Library should only contain software independent models. Unfortunately, this cannot be 100% ensured since different simulators support most of the VHDL-AMS standard in different ways, and they can offer multiple specific features that are tool dependent. Therefore, the VP-Model Library allows documenting these simulation limitations in each *model implementation* when they have been previously identified. This portal contains the list of *model implementations* organized by the `IMP ID` number and the `Implementation Name` similarly to the Implementations portal. Additionally, it includes the following fields:

1. **Software Tool:** This field contains the software tool name (with version) in which the *model implementation* has been designed and/or simulated.
2. **Description:** This field is dedicated to describing the identified restrictions and/or behaviors that are particular to the *model implementation* using the selected Software Tool. For example, the *model implementation* called `thermal_conductor(standard_model)` shown in Figure F.10, uses an unbalanced **if - use** VHDL-AMS statement for the quiescent domain that is only supported in SMASH software tool. Similarly, if the *model implementation* uses quantity tolerances, this restriction shall also be reported in this field since other software tools such as Mentor Graphics simulators do not support this VHDL-AMS feature.

#### F.2.1.11 References

This portal contains a list of pertinent bibliographic references that are related to the model record, see Figure F.11. These documents are stored in the VP-Model Library repository and can be accessed by the library users. The fields in this portal are explained as follows:

1. **Reference No.:** This is an automatically generated number that simply indicates the order of a particular reference within the model record.
2. **Reference details:** This field contains all the details about the bibliographic reference document such as authors, document title, publisher, journal, report number, dates, etc.
3. **Reference ID:** Since multiple model records can use the same reference, a unique document is required to be stored in the File repository of the VP-Model Library. The `Reference ID` number is the unique identifier of that document. By pressing the button **"Get File"** it is possible to download the reference document if the user counts with the proper permissions.



| Reference No. | Reference details | Reference ID |
|---|---|---|
| 1 | Juan Sebastian Rodriguez Estupiñan. Electro-Thermal Modeling of the Rogowski Coil System. Technical Report No. 014-01-02. École Polytechnique Fédéral de Lausanne - EPFL, Microelectronic Systems Laboratory - LSM, ABB Corporate Research, Baden Switzerland. Lausanne, Switzerland. May 26, 2014. | 8 |
| 2 | Rodriguez Estupinan, J.S., Vachoux, A.; Pascal, J. Electro-thermal virtual prototyping of a Rogowski Coil sensor system. IEEE international conference on Industrial Technology (ICIT), 17 - 19 March 2015, Seville, Spain. | 9 |
| 3 | IEEE Power Engineering Society. IEEE standard for calculating the current-temperature of bare overhead conductors. Standard IEEE Std 738-2006, IEEE-SA Standards Board, November 2006. | 10 |

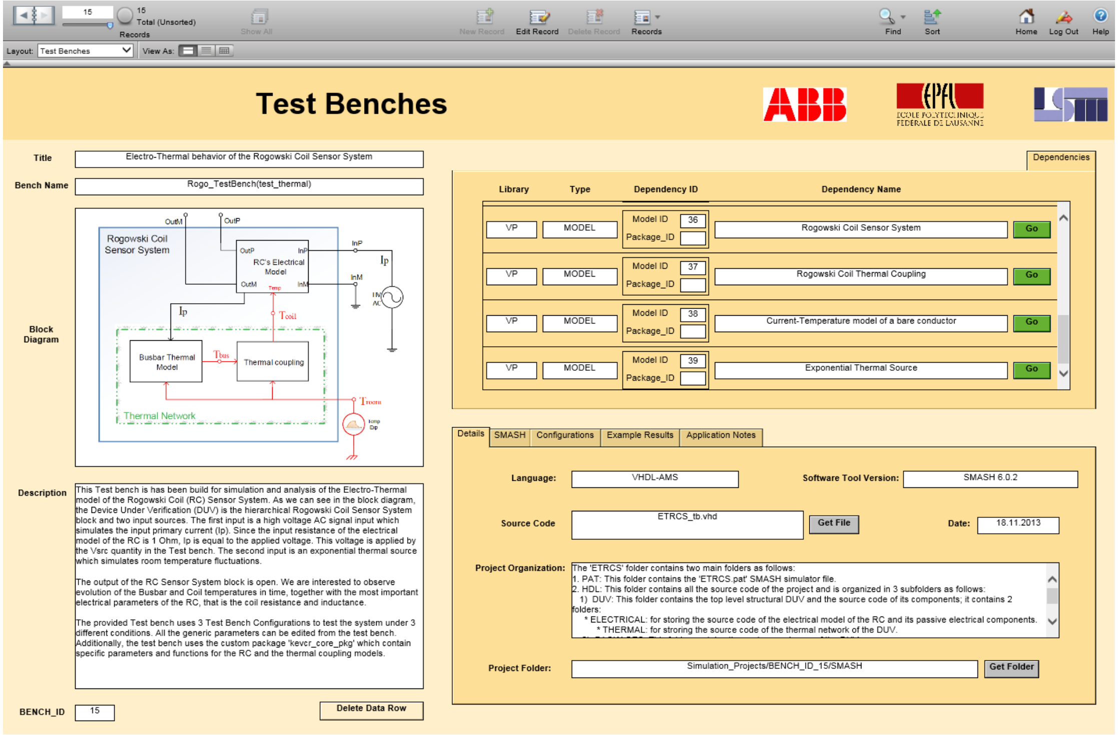
Figure F.11: References portal corresponding to the Busbar thermal model of the ETRCM example, see Figure 4.4 on page 58.

#### F.2.1.12 Application Notes

This portal, which is located beside the Reference portal, contains a single text field dedicated to store and display any other useful information for the users of the model regarding the model utilization. For example, this field can be used in case of model validation and characterization procedure using experimental data. Any important details about the model operation and the conditions for its utilization can be included here. This improves the model reliability and trustworthiness for verification.

### F.2.2 Test Benches view

The VP-Model Library dedicates an independent view for *test benches*, which are the essential modeling elements for verification in the proposed VP-based design methodology. The Test Benches view contains the documentation and links to all models and files that are related to



**Test Benches**

**Title** Electro-Thermal behavior of the Rogowski Coil Sensor System

**Bench Name** Rogo_TestBench(test_thermal)

**Block Diagram**

**Description** This Test bench is has been build for simulation and analysis of the Electro-Thermal model of the Rogowski Coil (RC) Sensor System. As we can see in the block diagram, the Device Under Verification (DUV) is the hierarchical Rogowski Coil Sensor System block and two input sources. The first input is a high voltage AC signal input which simulates the input primary current (Ip). Since the input resistance of the electrical model of the RC is 1 Ohm, Ip is equal to the applied voltage. This voltage is applied by the Vsrc quantity in the Test bench. The second input is an exponential thermal source which simulates room temperature fluctuations.

The output of the RC Sensor System block is open. We are interested to observe evolution of the Busbar and Coil temperatures in time, together with the most important electrical parameters of the RC, that is the coil resistance and inductance.

The provided Test bench uses 3 Test Bench Configurations to test the system under 3 different conditions. All the generic parameters can be edited from the test bench. Additionally, the test bench uses the custom package 'kevcr_core_pkg' which contain specific parameters and functions for the RC and the thermal coupling models.

**BENCH_ID** 15

| Library | Type | Dependency ID | Dependency Name |
|---|---|---|---|
| VP | MODEL | Model ID 36<br>Package_ID | Rogowski Coil Sensor System |
| VP | MODEL | Model ID 37<br>Package_ID | Rogowski Coil Thermal Coupling |
| VP | MODEL | Model ID 38<br>Package_ID | Current-Temperature model of a bare conductor |
| VP | MODEL | Model ID 39<br>Package_ID | Exponential Thermal Source |

Details | SMASH | Configurations | Example Results | Application Notes

**Language:** VHDL-AMS

**Software Tool Version:** SMASH 6.0.2

**Source Code** ETRCS_tb.vhd

**Date:** 18.11.2013

**Project Organization:** The 'ETRCS' folder contains two main folders as follows:
1. PAT: This folder contains the 'ETRCS.pat' SMASH simulator file.
2. HDL: This folder contains all the source code of the project and is organized in 3 subfolders as follows:
  1) DUV: This folder contains the top level structural DUV and the source code of its components; it contains 2 folders:
    * ELECTRICAL: for storing the source code of the electrical model of the RC and its passive electrical components.
      * THERMAL: for stroring the source code of the thermal network of the DUV.

**Project Folder:** Simulation_Projects/BENCH_ID_15/SMASH

Figure F.12: Test benches view. The electro-thermal behavior of the RogoCoil sensor system *test bench* example.

a *test bench* record. *Component models*, custom packages, simulation results, configurations and project files are organized and displayed in the Test Benches view in the VP-Model Library database.

Since the criteria of VHDL-AMS conditioned models (see subsection 3.6.1) requires to have at least one *test bench* for verifying each model record in the library[3], the Test Benches view shown in Figure F.12 becomes of high importance. Properly designed and documented *test benches* are the final examples that demonstrate the capabilities and potentials of *component models* and VPs. A model can be verified by simulation using different analysis types and verification procedures such as nominal analyses (DC, transient, small signal, and stability analysis), noise analysis, power analysis, stress analysis, perturbation or sensitivity analyses, worst-case or corner-case analyses, failure modes and effects analyses (FMEA), sneak circuit analysis (SCA), parametric and statistical analyses (Monte Carlo, nested parameter sweep). For each analysis type, the model can give different results (e.g., current, voltage, temperature, force, power, failure) obtainable either dynamically during the simulation or only after completion. Each result can be in a different form like a flag, a message, a scalar, a vector, a waveform, or a relation/equation (e.g. a non-time series forming a table of the interaction of two or more variables).

The metadata fields of the Test Benches view contains the following information:

---

[3] The exception is the utility models, which do not necessarily need a dedicated *test bench*.

1. **Title:** Complete and self-explicative name of the *test bench*.
2. **Bench Name:** Abbreviated name of the *test bench* useful for the *test bench* identification. Similar to the model `Implementation Name`, we can use a VHDL-AMS approach of the form "testbench_name(architecture_name)" for naming.
3. **Block Diagram:** This field shows an abstract diagram of the *test bench* model. All the main *component models* included inside and outside the DUV are shown as black boxes in the diagram as is shown in Figure F.12.
4. **Description:** General description of the test bench. It is briefly described the construction, the purpose, the manipulation, and the assumptions of the *test bench* case.
5. **Bench_ID:** Unique *test bench* record identifier.

The Test Benches view contains the following portals:

### F.2.2.1 Dependencies

This portal contains the list of *component models* and packages that are part of the selected *test bench* record. The fields are explained as follows:

1. **Library:** This is the specific name of the VHDL-AMS library used on the *test bench* (at the top level). All the list of standard and non-standard VHDL-AMS libraries must be included. For custom packages and models contained in the VP-Model Library, the `Library` name is **'VP'**. The same clarification for naming custom libraries given in subsection F.2.1.1 on page 231 also applies.
2. **Type:** There are 2 types of dependencies in the *test bench*: the `PACKAGE` type, which indicates a custom or standard package contained in the VP-Model Library; and the `MODEL` type, which indicates a *component model* or VP stored in the VP-Model Library.
3. **Dependency ID:** If the type of dependency is `MODEL`, the `Model ID` number is available in the respective field. Otherwise, if the dependency is a Custom Package included in the VP-Model Library, the `Package_ID` number is available in the respective field. If the dependency is a standard package, no number is included in these fields.
4. **Dependency Name:** Name of the dependency. In the case of models, if only one specific implementation of a model is used in the *test bench*, the `Dependency Name` corresponds to the `Implementation Name`. Otherwise, the `Dependency Name` is the `Model Name`. In the case of custom packages, the `Dependency Name` corresponds to the `Package Name`. The button **'Go'** is a link to the respective model or custom package record.

### F.2.2.2 Details

The Details portal shown in the lower-right side in Figure F.12 contains the following specific information related to the *test bench*:

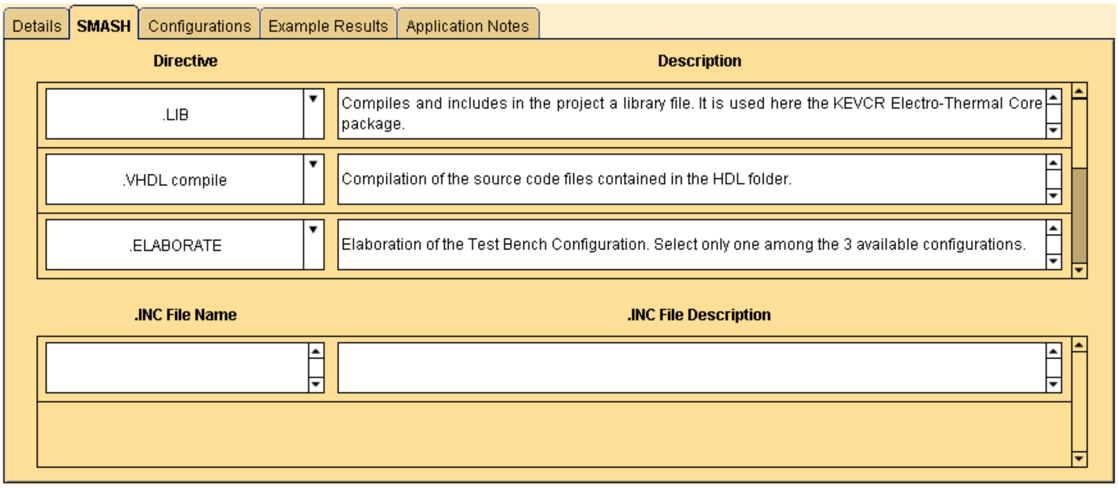
1. **Language:** It is the language of the *test bench* top level file.

2. **Software Tool Version:** This field contains the software tool name and the version in which the *test bench* has been implemented.
3. **Source Code File:** This field contains a link to the test bench source code. To get the source code click on the button **'Get file'**.
4. **Date:** Date of the last update of the *test bench*.
5. **Project Organization:** This field contains a detailed explanation of the structure of the simulation project folders and their contents, i.e. simulator files, .inc files, and source code files.
6. **Project Folder:** This is a link to the simulation project folder (which is archived in a .zip file). By clicking the button **'Get Folder'**, a web browser window must show the main project folder in which one or more software tool subfolders can exist when the simulation project is available in different simulation tools.

### F.2.2.3 SMASH

During this work, the SMASH simulator was the preferred simulation tool; therefore, a SMASH portal has been included in the Test Benches view in order to explain how the main simulator files (the .pat and .inc files) are designed and how they should be manipulated for simulation. The portal fields showed in Figure F.13 are explained as follows:

1. **Directive:** These fields contain the list of principal SMASH directives included in the .pat file.
2. **Description:** This is a brief description of each main SMASH directive included in the .pat file. It is important to include details about how to manipulate these directives to obtain the desired simulation results.
3. **.INC File Name:** Name of the available include (.inc) files. In order to decrease the size of a large .pat SMASH file, several .inc files can be used to defined and organize multiple



| Directive | Description |
|---|---|
| .LIB | Compiles and includes in the project a library file. It is used here the KEVCR Electro-Thermal Core package. |
| .VHDL compile | Compilation of the source code files contained in the HDL folder. |
| .ELABORATE | Elaboration of the Test Bench Configuration. Select only one among the 3 available configurations. |

Figure F.13: SMASH portal corresponding to the *test bench* example in Figure F.12.

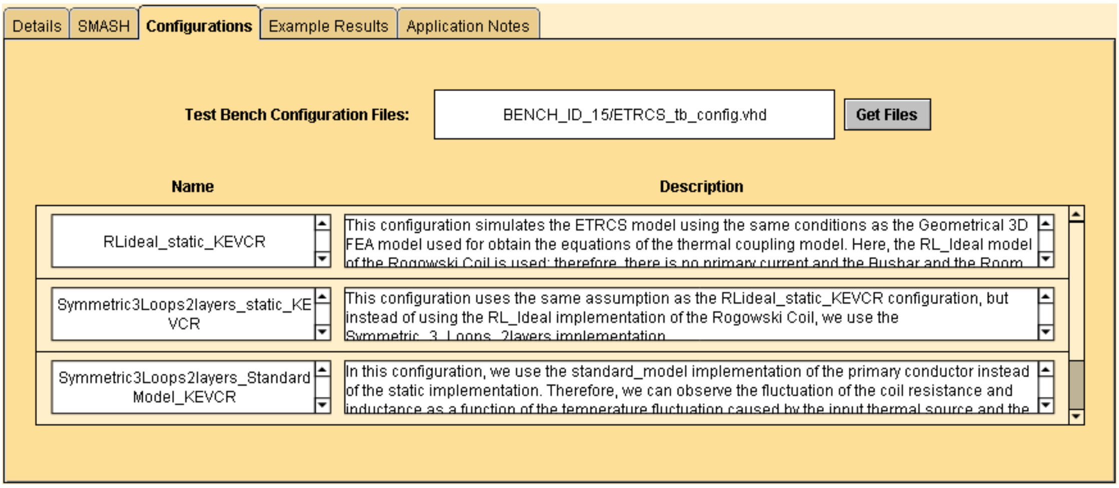
*test cases* by using a different set of SMASH directives. An adequate selection of .inc files is called from the .pat file in order to set a particular *test case*.

4. **.INC File Description:** This is the description of the specific .inc file. An explanation of the include file purpose and its key SMASH directives must be given.

#### F.2.2.4 Configurations

This portal contains the list of all the related test bench configurations (TBCs) used by the *test bench*. The fields included in this portal, see Figure F.14, are explained as follows:

1. **TBC Files:** This is a link to the file or files containing the source code of the TBCs. The button **"Get Files"** allows the access to the source code in a web browser window.
2. **Name:** This field contains the complete name of the configuration. The use of meaningful names is advocated disregarding the length.
3. **Description:** This field contains a clear description of the TBC including its *test case* purpose. Enough details of the TBC must be included in order to distinguish from other TBCs.



**Test Bench Configuration Files:** BENCH_ID_15/ETRCS_tb_config.vhd

| Name | Description |
|---|---|
| RLideal_static_KEVCR | This configuration simulates the ETRCS model using the same conditions as the Geometrical 3D FEA model used for obtain the equations of the thermal coupling model. Here, the RL_Ideal model of the Rogowski Coil is used; therefore, there is no primary current and the Busbar and the Room |
| Symmetric3Loops2layers_static_KEVCR | This configuration uses the same assumption as the RLideal_static_KEVCR configuration, but instead of using the RL_Ideal implementation of the Rogowski Coil, we use the Symmetric_3_Loops_2layers implementation. |
| Symmetric3Loops2layers_StandardModel_KEVCR | In this configuration, we use the standard_model implementation of the primary conductor instead of the static implementation. Therefore, we can observe the fluctuation of the coil resistance and inductance as a function of the temperature fluctuation caused by the input thermal source and the |

Figure F.14: Test bench configurations portal corresponding to the *test bench* example in Figure F.12.

### F.2.3 Example Results and Application Notes

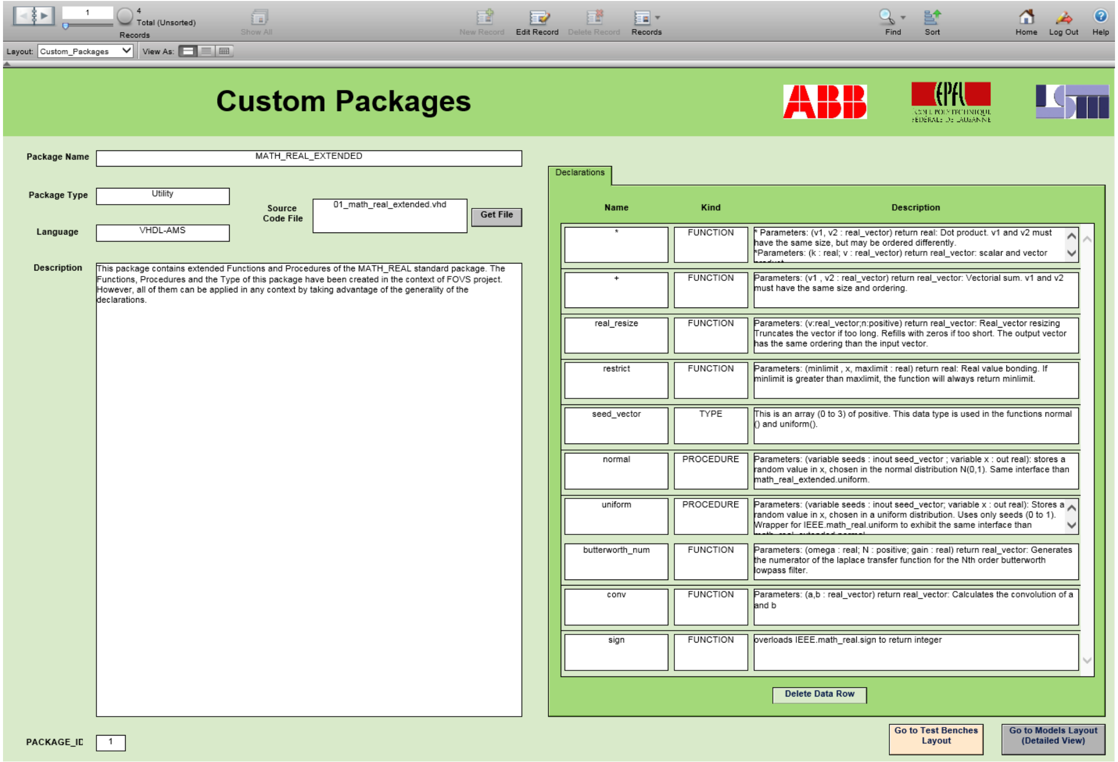
The last two portals of the Test Benches view are the Example Results and the Application Notes. The first portal is dedicated to store and display images of the *test bench* simulation results that allow to better understand the expected verification results of a given example. This field is optimized for storing a single image file or a .pdf file with multiple simulation images.

On the other hand, the Application Notes portal is a single text field devoted to extending the information about the *test bench* utilization. Any additional information that can be useful to understand and manipulate the *test bench* must be included in this field. For example, if instead of using configurations, the model designer uses other technique to switch among different *test case* within the same test bench, the explanation about how to manipulate the *test bench* must be included here. Simulation issues and limitations can also be reported in this field.

### F.2.4 Custom packages view

Commonly, projects which involve the development of large VPs require the elaboration of custom packages that contain functions, constants, classes, and/or any other custom types which are used in different blocks throughout the design. Since these custom packages can be easily adapted for other projects, the VP-Model Library dedicates the Custom Packages view for documenting and displaying custom packages and all its related declarations. Each custom package record has a unique identifier number, the `PACKAGE_ID`.

A good strategy in VHDL-AMS *virtual prototyping* consists of developing custom packages gathered in project libraries for those functions, constants, classes, types and many other declarations which are applied to several blocks in the design. The advantage of placing them in a package is that they do not clutter up other parts of the model, and they can be shared within and among models without having to rewrite them.



Layout: Custom_Packages

Custom Packages

Package Name: MATH_REAL_EXTENDED

Package Type: Utility

Language: VHDL-AMS

Source Code File: 01_math_real_extended.vhd

Description: This package contains extended Functions and Procedures of the MATH_REAL standard package. The Functions, Procedures and the Type of this package have been created in the context of FOVS project. However, all of them can be applied in any context by taking advantage of the generality of the declarations.

PACKAGE_ID: 1

| Name | Kind | Description |
|---|---|---|
| * | FUNCTION | * Parameters: (v1, v2 : real_vector) return real: Dot product. v1 and v2 must have the same size, but may be ordered differently.<br>*Parameters: (k : real; v : real_vector) return real_vector: scalar and vector |
| + | FUNCTION | Parameters: (v1 , v2 : real_vector) return real_vector: Vectorial sum. v1 and v2 must have the same size and ordering. |
| real_resize | FUNCTION | Parameters: (v:real_vector;n:positive) return real_vector: Real_vector resizing Truncates the vector if too long. Refills with zeros if too short. The output vector has the same ordering than the input vector. |
| restrict | FUNCTION | Parameters: (minlimit , x, maxlimit : real) return real: Real value bonding. If minlimit is greater than maxlimit, the function will always return minlimit. |
| seed_vector | TYPE | This is an array (0 to 3) of positive. This data type is used in the functions normal () and uniform(). |
| normal | PROCEDURE | Parameters: (variable seeds : inout seed_vector ; variable x : out real): stores a random value in x, chosen in the normal distribution N(0,1). Same interface than math_real_extended.uniform. |
| uniform | PROCEDURE | Parameters: (variable seeds : inout seed_vector; variable x : out real): Stores a random value in x, chosen in a uniform distribution. Uses only seeds (0 to 1). Wrapper for IEEE.math_real.uniform to exhibit the same interface than |
| butterworth_num | FUNCTION | Parameters: (omega : real; N : positive; gain : real) return real_vector: Generates the numerator of the laplace transfer function for the Nth order butterworth lowpass filter. |
| conv | FUNCTION | Parameters: (a,b : real_vector) return real_vector: Calculates the convolution of a and b |
| sign | FUNCTION | overloads IEEE.math_real.sign to return integer |

Figure F.15: Custom packages view. `MATH_REAL_EXTENDED` package example.

This practice can save a lot of time in future projects, in which a package implementation can be done quickly and effectively. However, the success of the package adoption will strongly depend on the practicality of its declarations and the quality of the documentation. The fields of this view, shown in Figure F.15, are dedicated to facilitating the package documentation in a direct and organized way as follows:

1. **Package Name:** This is the name of the package as it is used in the source code.
2. **Package Type:** There are two types of packages:
    (a) **Utility:** This type of packages offer general purpose declarations such as functions, types, or procedures that can be applied in many different *virtual prototyping* projects. For example, extended mathematic custom packages can offer functions for custom type operations.
    (b) **Application specific:** This type of packages are particularly designed for specific applications; so that, its direct re-utilization in different projects is limited. They are normally parameter packages with mainly constant declarations. However, they can be used as examples for future modeling projects.
3. **Language:** This is the language in which the package is written.
4. **Source Code File:** This field contains the complete name (including the extension) of the source code file of the package. To get the source code click on the button **'Get file'**.
5. **Description:** A clear and complete description of the package must be included in this field.

#### F.2.4.1 Declarations

The Custom Packages view only contains one portal named 'Declarations', see Figure F.15. This portal list all the declarations that the custom package contains. The fields are explained as follows:

1. **Name:** This is the name of the listed declaration as it is given in the source file.
2. **Kind:** Following the VHDL-AMS syntax, there can be multiple kinds/types of declarations as follows:
    (a) **Constant:** This is a constant value, its type and value must be described in the Description field. By defining a constant in a package we can avoid scattering literal values throughout the models. If we need to update the value we only need to change it in the package declaration, this is much easier and reliable than trying to find all instances of a literal value throughout the models.
    (b) **Variable:** This is a shared variable value, its type must be described in the Description field. The initialization expression is optional. If we omit it, the default initial value assumed by the variable when it is created depends on the type. For scalar types, the default initial value is the leftmost value of the type. For example, for integers, it is the smallest representable integer. As normal variables can only be accessed by one Process, Function or Procedure, a variable in a package only make sense if it can be shared in multiple places, i.e. if it is a shared variable.

(c) **Signal:** This is a standard VHDL-AMS signal declaration. Typically, a Signal is declared in an Architecture and is known and able to be evaluated and driven without redeclaration within any Process, Block, Procedure or Function in an Architecture. The scope of such signals does not extend into any Component defined by the Architecture, nor to any VHDL-AMS Entity outside the Architecture at all. A signal's value may be passed to any other Entity only through connection with a Port Signal of that Entity. Signals that are declared in Packages are identical in basic nature to any other Signal anywhere in the design, they conserve all its typical characteristics. The difference is that a signal declared in a package can potentially be evaluated and/or driven by every concurrent or sequential Process in every Architecture in the entire design. The use of Signals declared in Packages is a valuable and powerful *test bench* technique which can solve tough problems, but this is not recommended for synthesis. It is unknown if the synthesis engines will consider those signals since they are not included in any Entity or Architecture. This technique must be only used by carefully studying the impact of those signals in simulation.

(d) **Procedure:** This is a common VHDL-AMS subprogram (or function) which receives input, output or inout parameters and execute a series of sequential statements to produce a result. This function can modify one or more output or global parameters. Similar to Process statements, the Procedure can be written in a form of concurrent call statements. A proper explanation must be included in the Description field to understand the purpose and the operation of the Procedure

(e) **Function:** Contrarily to the Procedure, the VHDL-AMS Function can only take constant or input parameters to return a determined value of a specific type. The functions can be pure or impure. An adequate explanation of the Function must be given in the Description field, including the input parameters and the expected value to be returned.

(f) **Type:** This is a specific VHDL-AMS custom data type, it must be properly explained in the Description field.

(g) **Subtype:** When a model contains objects that should only take on a restricted range of the complete set of values of a Type, we can represent such objects by declaring a Subtype. A clear description of the Subtype must be included in the Description field.

(h) **Alias:** An Alias is simply an alternate name for something, we can make our code simpler by defining Alias to complex structures. If we have a model that includes a data object, such as a constant, a variable, a signal, a quantity, a terminal or, a file, we can declare an Alias for the object. We can also declare Aliases for other named items that do not represent stored data, such as types, natures, subprograms, packages, entities and so on. In fact, the only kinds of items for which we cannot declare Aliases are labels, loop parameters and generate parameters. A clear description of the Alias must be included in the Description field.

3. **Description:** A clear and complete description of the declaration must be included here.

## F.3 Modeling request

A modeling request is a specific petition of a *modeling element* update of the VP-Model Library. The modeling requests are the inputs of the model maintenance process that supports the *virtual prototyping* activity using the VP-based design methodology. The different types of modeling requests are explained in section 5.4. The success of the Model maintenance flow depicted in Figure 5.7 on page 156, largely depends on the mechanism that the VP-Model Library actors (see section F.3) have to communicate with them. The result of a modeling request is an update (i.e. modification, correction, addition) of the VP-Model Library content.

As part of the VP-Model Library implementation, the proposed vehicle for executing modeling requests is denominated the ***model registration form*** **(MRF)**, which is basically a copy of the VP-Model Library structure that can be modified[4] by contributors and administrators. The idea is to assign to each library contributor an MRF that can be used for both submitting modeling requests and for collecting all type of *modeling elements* and their respective documentation.

If the modeling request is about the refinement of an existent model in the library, using an MRF provides the advantage of linking directly the available model with the requested *model implementation*. In this way, it is faster and easier both to describe and understand the refinement direction in terms of new behaviors and interactions, such as modeling additional non-idealities, cross-domain variables, parameters, or more new architectures (refinement by aggregation of components).

---

[4] i.e. writing privileges for all the fields of the model's detailed view.

# Bibliography

[1] E. A. Lee. Cyber Physical Systems: Design Challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369, May 2008.

[2] R. Poovendran, K. Sampigethaya, S. K. S. Gupta, I. Lee, K. V. Prasad, D. Corman, and J. Paunicka. Special Issue on Cyber - Physical Systems [Scanning the Issue]. *Proceedings of the IEEE*, 100(1):6–12, 2012.

[3] Radhakisan Baheti and Helen Gill. Cyber-physical Systems. *The Impact of Control Technology*, 2011.

[4] J. Sztipanovits, X. Koutsoukos, G. Karsai, N. Kottenstette, P. Antsaklis, V. Gupta, B. Goodwine, J. Baras, and Shige Wang. Toward a Science of Cyber-Physical System Integration. *Proceedings of the IEEE*, 100(1):29–44, 2012.

[5] B. Balaji, M. A. Al Faruque, N. Dutt, R. Gupta, and Y. Agarwal. Models, abstractions, and architectures: The missing links in cyber-physical systems. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2015.

[6] A. Sangiovanni-Vincentelli. Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design. *Proceedings of the IEEE*, 95(3):467–506, 2007.

[7] J. C. Jensen, D. H. Chang, and E. A. Lee. A model-based design methodology for cyber-physical systems. In *2011 7th International Wireless Communications and Mobile Computing Conference*, pages 1666–1671, July 2011.

[8] W. D. Li, W. F. Lu, J. Y. H. Fuh, and Y. S. Wong. Collaborative computer-aided design—research and development status. *Computer-Aided Design*, 37(9):931–940, August 2005.

[9] Daniel Cocks, Michael Dickerson, David Oliver, Joseph Skipper, and MDSD Working Group and AP233 Teamr. Model Driven Design. *INSIGHT*, 7(2):5–8, July 2004.

[10] Robin T. Bye, Ottar L. Osen, and Birger Skogeng Pedersen. A computer-automated design tool for intelligent virtual prototyping of offshore cranes. In *ECMS 2015*, Albena (Varna), Bulgaria, May 2015. Springer.

[11] Zhiyi Pan, Xin Wang, Rumin Teng, and Xuyang Cao. Computer-aided design-while-engineering technology in top-down modeling of mechanical product. *Computers in Industry*, 75:151–161, January 2016.

[12] David C. Ku and Giovanni DeMicheli. *High Level Synthesis of ASICs under Timing and Synchronization Constraints*. Springer Science & Business Media, March 2013.

[13] Valter Bellucci, Bruno Schuermans, Dariusz Nowak, Peter Flohr, and Christian Oliver Paschereit. Thermoacoustic Modeling of a Gas Turbine Combustor Equipped With Acoustic Dampers. *Journal of Turbomachinery*, 127(2):372–379, May 2005.

[14] Jay Lee, Behrad Bagheri, and Hung-An Kao. A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23, January 2015.

[15] Q. Ha, Y. Maret, J. S. R. Estupiñan, and A. Vachoux. VHDL-AMS virtual prototyping of a generator circuit breaker ablation monitoring system. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1854–1857, May 2015.

[16] Olivier Steiger, Sergio V. Marchese, Joris Pascal, Klaus Bohnert, and Stephan Wildermuth. Signal processing for electro-optic voltage sensor. pages 1–4. IEEE, November 2013.

[17] J. S. R. Estupiñán, A. Vachoux, and J. Pascal. Electro-thermal virtual prototyping of a Rogowski Coil sensor system. In *2015 IEEE International Conference on Industrial Technology (ICIT)*, pages 1451–1456, March 2015.

[18] C. Grimm and C. Radojicic. Verification and validation of AMS systems: Towards coverage of uncertainties. In *2015 IEEE 20th International Mixed-Signals Testing Workshop (IMSTW)*, pages 1–6, June 2015.

[19] Alberto Sangiovanni-Vincentelli, Werner Damm, and Roberto Passerone. Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems*. *European Journal of Control*, 18(3):217–238, 2012.

[20] John Vargas. Reduce Project Schedules and Increase Quality Using Model Driven Development for Design, Verification and Test. In *2013 NDIA Ground Vehicle Systems Engineering and Technology Symposium*, page 7, Troy, Michigan, USA, August 2013.

[21] Y. Hervé and A. Legendre. Chapter 11 - Functional Virtual Prototyping for Heterogeneous Systems. In *Design Technology for Heterogeneous Embedded Systems*, pages 223–253. Springer Science, 1 edition, 2012.

[22] AUTOSAR Partnership. *AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures.* SAE International, 2004.

[23] P. J. Prisaznuk. ARINC 653 role in Integrated Modular Avionics (IMA). In *2008 IEEE/AIAA 27th Digital Avionics Systems Conference*, pages 1.E.5–1–1.E.5–10, October 2008.

[24] T. Mahne, A. Vachoux, and Y. Leblebici. Fostering the reuse and collaborative development of models in the AMS SoC design process. In *Research in Microelectronics and Electronics Conference, 2007. PRIME 2007. Ph.D.*, pages 285–288, July 2007.

[25] J. Wan, A. Canedo, and M. A. Al Faruque. Functional Model-Based Design Methodology for Automotive Cyber-Physical Systems. *IEEE Systems Journal*, PP(99):1–12, 2015.

[26] L. A. Kamentsky and C. N. Liu. Computer-Automated Design of Multifont Print Recognition Logic. *IBM Journal of Research and Development*, 7(1):2–13, January 1963.

[27] Herbert Palm, Jorg Holzmann, Robert Klein, Stefan-Alexander Schneider, and Dieter Gerling. A Novel Approach on Virtual Systems Prototyping Based on a Validated, Hierarchical, Modular Library. page 10, Nuremberg, Germany, February 2013.

[28] F. Pecheux, C. Lallement, and A. Vachoux. VHDL-AMS and Verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(2):204–225, 2005.

[29] Alain Vachoux, Christoph Grimm, and Karsten Einwich. SystemC-AMS Requirements, Design Objectives and Rationale. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1*, DATE '03, pages 10388–, Washington, DC, USA, 2003. IEEE Computer Society.

[30] Michał Rewieński. A Perspective on Fast-SPICE Simulation Technology. In Peng Li, Luís Miguel Silveira, and Peter Feldmann, editors, *Simulation and Verification of Electronic and Biological Systems*, pages 23–42. Springer Netherlands, 2011. DOI: 10.1007/978-94-007-0149-6_2.

[31] F. Bennini, J. Mehner, and W. Dötzel. System level simulations of mems based on reduced order finite element models. *International Journal of Computational Engineering Science*, 04(02):385–388, June 2003.

[32] F. Mendoza, J. Pascal, P. Nenninger, and J. Becker. Framework for dynamic verification of multi-domain virtual platforms in industrial automation. In *IEEE 10th International Conference on Industrial Informatics*, pages 935–940, July 2012.

[33] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann, October 2014.

[34] Peter H. Feiler and David P. Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley, September 2012.

[35] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. John Wiley & Sons, August 2010.

[36] Ahmad A. Mahfouz, Mohammed M. K., and Farhan A. Salem. Modeling, Simulation and Dynamics Analysis Issues of Electric Motor, for Mechatronics Applications, Using Different Approaches and Verification by MATLAB/Simulink. *International Journal of Intelligent Systems and Applications*, 5(5):39–57, April 2013.

[37] Marian Petre. UML in Practice. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 722–731, Piscataway, NJ, USA, 2013. IEEE Press.

[38] A. Sangiovanni-Vincentelli, Guang Yang, S.K. Shukla, D.A. Mathaikutty, and J. Sztipanovits. Metamodeling: An Emerging Representation Paradigm for System-Level Design. *IEEE Design Test of Computers*, 26(3):54–69, 2009.

[39] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauss, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. pages 105–114, June 2011.

[40] P. Derler, E.A. Lee, and A.-S. Vincentelli. Modeling Cyber-Physical Systems. *Proceedings of the IEEE*, 100(1):13–28, 2012.

[41] I. Akkaya, P. Derler, S. Emoto, and E. A. Lee. Systems Engineering for Industrial Cyber Physical Systems Using Aspects. *Proceedings of the IEEE*, 104(5):997–1012, May 2016.

[42] J. Eker, J. W. Janneck, E. A. Lee, Jie Liu, Xiaojun Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Yuhong Xiong. Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, January 2003.

[43] Gabriela Nicolescu and Pieter J. Mosterman. *Model-Based Design for Embedded Systems*. CRC Press, November 2009.

[44] Aymen Louati, Kamel Barkaoui, and Chadlia Jerad. Temporal Properties Verification of Real-Time Systems Using UML/MARTE/OCL-RT. In Thouraya Bouabana-Tebibel and Stuart H. Rubin, editors, *Formalisms for Reuse and Systems Integration*, number 346 in Advances in Intelligent Systems and Computing, pages 133–147. Springer International Publishing, 2015. DOI: 10.1007/978-3-319-16577-6_6.

[45] Susanne Graf, Roberto Passerone, and Sophie Quinton. Contract-Based Reasoning for Component Systems with Rich Interactions. In Alberto Sangiovanni-Vincentelli, Haibo Zeng, Marco Di Natale, and Peter Marwedel, editors, *Embedded Systems Development*, number 20 in Embedded Systems, pages 139–154. Springer New York, 2014. DOI: 10.1007/978-1-4614-3879-3_8.

[46] Ajitha Rajan and Thomas Wahl. *CESAR: Cost-efficient Methods and Processes for Safety-relevant Embedded Systems*. Springer, 2013.

[47] Kevin Lynch, Randall Ramsey, George Ball, Matt Schmit, and Kyle Collins. Ontology-Driven Metamodel Validation in Cyber-Physical Systems. In Shahram Latifi, editor, *Information Technology: New Generations*, number 448 in Advances in Intelligent Systems and Computing, pages 1255–1258. Springer International Publishing, 2016. DOI: 10.1007/978-3-319-32467-8_109.

[48] Alberto Sangiovanni-Vincentelli, Luca Carloni, Fernando De Bernardinis, and Marco Sgroi. Benefits and Challenges for Platform-based Design. In *Proceedings of the 41st Annual Design Automation Conference*, DAC ’04, pages 409–414, New York, NY, USA, 2004. ACM.

[49] K. Keutzer, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1523–1543, December 2000.

[50] A. Cimatti and S. Tonetta. A Property-Based Proof System for Contract-Based Design. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, pages 21–28, September 2012.

[51] E. M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.

[52] P. Nuzzo, A. L. Sangiovanni-Vincentelli, D. Bresolin, L. Geretti, and T. Villa. A Platform-Based Design Methodology With Contracts and Related Tools for the Design of Cyber-Physical Systems. *Proceedings of the IEEE*, 103(11):2104–2132, November 2015.

[53] Albert Benveniste, Benoît Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. Multiple Viewpoint Contract-Based Specification and Design. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul de Roever, editors, *Formal Methods for Components and Objects*, number 5382 in Lecture Notes in Computer Science, pages 200–225. Springer Berlin Heidelberg, October 2007. DOI: 10.1007/978-3-540-92188-2_9.

[54] Luca de Alfaro and Thomas A. Henzinger. Interface Automata. In *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-9, pages 109–120, New York, NY, USA, 2001. ACM.

[55] Sebastian S. Bauer, Alexandre David, Rolf Hennicker, Kim Guldstrand Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wąsowski. Moving from Specifications to Contracts in Component-Based Design. In Juan de Lara and Andrea Zisman, editors, *Fundamental Approaches to Software Engineering*, number 7212 in Lecture Notes in Computer Science, pages 43–58. Springer Berlin Heidelberg, March 2012. DOI: 10.1007/978-3-642-28872-2_3.

[56] Albert Benveniste, Benoit Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Thomas Henzinger, and Kim G. Larsen. Contracts for System Design. report, INRIA, November 2012.

[57] John Stark. *Product Lifecycle Management (Volume 1)*. Decision Engineering. Springer International Publishing, third edition, 2015. DOI: 10.1007/978-3-319-17440-2_1.

[58] Filipe Ferreira, José Faria, Américo Azevedo, and Ana Luisa Marques. Product lifecycle management in knowledge intensive collaborative environments: An application to automotive industry. *International Journal of Information Management*, 37(1, Part A):1474–1487, February 2017.

[59] Felix Wortmann and Kristina Flüchter. Internet of Things: Technology and Value Added. *Business & Information Systems Engineering; Berkeley*, 57(3):221–224, June 2015.

[60] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. The rise of “big data” on cloud computing: Review and open research issues. *Information Systems*, 47:98–115, January 2015.

[61] Jay Lee, Behrad Bagheri, and Hung-An Kao. A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23, January 2015.

[62] Heiner Lasi, Peter Fettke, Hans-georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & Information Systems Engineering; Berkeley*, 6(4):239–242, August 2014.

[63] Henning Kagermann. *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0: Securing the Future of German Manufacturing Industry ; Final Report of the Industrie 4.0 Working Group*. Forschungsunion, Germany, April 2013. Google-Books-ID: AsfOoAEACAAJ.

[64] M. Hermann, T. Pentek, and B. Otto. Design Principles for Industrie 4.0 Scenarios. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pages 3928–3937, January 2016.

[65] S. Weaver, B. Hershberg, and U. K. Moon. Digitally Synthesized Stochastic Flash ADC Using Only Standard Digital Cells. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(1):84–91, January 2014.

[66] Tatsuji Matsuura. Recent progress on CMOS successive approximation ADCs. *IEEJ Transactions on Electrical and Electronic Engineering*, 11(5):535–548, September 2016.

[67] O. Jimenez, O. Lucia, I. Urriza, L. A. Barragan, and D. Navarro. Design and Evaluation of a Low-Cost High-Performance #x2013; ADC for Embedded Control Systems in Induction Heating Appliances. *IEEE Transactions on Industrial Electronics*, 61(5):2601–2611, May 2014.

[68] Achi Brandt. Multiscale Scientific Computation: Review 2001. In Timothy J. Barth, Tony Chan, and Robert Haimes, editors, *Multiscale and Multiresolution Methods*, number 20 in Lecture Notes in Computational Science and Engineering, pages 3–95. Springer Berlin Heidelberg, 2002. DOI: 10.1007/978-3-642-56205-1_1.

[69] Antoon Goderis, Christopher Brooks, Ilkay Altintas, Edward A. Lee, and Carole Goble. Heterogeneous composition of models of computation. *Future Generation Computer Systems*, 25(5):552–560, May 2009.

[70] A. Vachoux, C. Grimm, and K. Einwich. Towards analog and mixed-signal SOC design with systemC-AMS. In *Proceedings. DELTA 2004. Second IEEE International Workshop on Electronic Design, Test and Applications*, pages 97–102, January 2004.

[71] Accellera. SystemC AMS 2.0 Standard for Mixed-Signal Design of Electronic Systems, 2013.

[72] E. Christen and K. Bakalar. VHDL-AMS-a hardware description language for analog and mixed-signal applications. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 46(10):1263–1272, October 1999.

[73] J Verries and A Sahraoui. Case Study On SYSML and VHDL-AMS for Designing and Validating Systems. In *Proceedings of the World Congress on Engineering and Computer Science*, volume 1, San Francisco, USA, 2013.

[74] Torsten Mähne. *Efficient Modelling and Simulation Methodology for the Design of Heterogeneous Mixed-Signal Systems on Chip.* Doctoral Thesis, Swiss Federal Institute of Technology Lausanne, Lausanne, Switzerland, 2011.

[75] SAE Electronic Design Automation Standards Committee. Model Specification Process Standard J2546_200202, February 2002.

[76] M. H. Samimi, A. Mahari, M. A. Farahnakian, and H. Mohseni. The Rogowski Coil Principles and Applications: A Review. *IEEE Sensors Journal*, 15(2):651–658, February 2015.

[77] Pavel Ripka. Electric current sensors: a review. *Measurement Science and Technology*, 21(11):112001, 2010.

[78] G. Meijer, M. Pertijs, and K. Makinwa. *Smart Sensor Systems: Emerging Technologies and Applications.* John Wiley and Sons, Delft University of Technology, the Netherlands, May 2014.

[79] Raphael Segas. Self calibrating current sensor system: virtual prototyping and experimental test. Internship Report 9ADB005157-004, ABB Corporate Research Switzerland, Baden-Dattwil, March 2013.

[80] Joris Pascal and Beat Kramer. SCALES Technology Demonstrator Design Report. Technical 9ADB005157-016, ABB Corporate Research Switzerland, Baden-Dättwil, November 2014.

[81] V. Dubickas and Hans Edin. High-Frequency Model of the Rogowski Coil With a Small Number of Turns. *IEEE Transactions on Instrumentation and Measurement*, 56(6):2284–2288, 2007.

[82] Mats Forssell. Rogowski Coil Self-calibration Frequency Method. Internship Report 9ADB005157-011, ABB Corporate Research Switzerland, Baden-Dättwil, August 2013.

[83] Peter Nefzger, Ulf Kaintzyk, and Joao Felix Nolasco. *Overhead Power Lines: Planning, Design, Construction.* Springer, April 2003.

[84] IEEE Power Engineering Society. IEEE Standard for Calculating the Current-Temperature of Bare Overhead Conductors. Standard IEEE Std 738-2006, IEEE-SA Standards Board, November 2006.

[85] J.P. Holman. *Heat Transfer.* Series in Mechanical Engineering. Mc. Graw Hill, 10th edition, 2010.

[86] Adrian Bejan and Allan D. Kraus. *Heat Transfer Handbook.* John Wiley & Sons, June 2003.

[87] MatWeb, LLC. Online Material Properties Database. http://www.matweb.com, 2013.

[88] Ph. Moreau, A. Le-Luyer, P. Malard, P. Pastor, F. Saint-Laurent, P. Spuig, J. Lister, M. Toussaint, P. Marmillod, D. Testa, S. Peruzzo, J. Knaster, G. Vayakis, S. Hughes, and K. M. Patel. Prototyping and testing of the Continuous External Rogowski ITER magnetic sensor. *Fusion Engineering and Design*, 88(6–8):1165–1169, October 2013.

[89] Charles Kitchin and Lew Counts. RMS to DC Conversion Application Guide. Available in: http://www.analog.com/media/en/training-seminars/design-handbooks/RMStoDC_cover-Section-I.pdf?doc=AD8436.pdf, 1986.

[90] Linear Technology Corporation. LTC1967 - Precision Extended Bandwidth, RMS-to-DC Converter. Available in: http://cds.linear.com/docs/en/datasheet/1967f.pdf, 2004.

[91] Richard Schreier and Gabor C. Temes. *Understanding Delta-Sigma Data Converters.* John Wiley & Sons Inc, 2017.

[92] B. E. Boser and B. A. Wooley. The design of sigma-delta modulation analog-to-digital converters. *IEEE Journal of Solid-State Circuits*, 23(6):1298–1308, December 1988.

[93] Miroslav Oljaca and Tom Hendrick. Combining the ADS1202 with an FPGA Digital Filter for Current Measurement in Motor Control Applications. Application Report SBAA094, Texas Instruments, June 2003.

[94] N. S. Alagha and P. Kabal. Generalized raised-cosine filters. *IEEE Transactions on Communications*, 47(7):989–997, July 1999.

[95] James Barnes. Data Converters. Colorado State University, Dept of Electrical and Computer Engineering. Available at: https://www.engr.colostate.edu/ECE423/lectures_-pdf/2014/notes_lec03.pdf, 2014.

[96] International Electrotechnical Commission. IEC 60044-8 Standard, 2002.

[97] S Balaji and M Sundararajan Murugaiyan. Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management*, 2(1):26–30, 2012.

[98] The Business Plan Shop web page. How to do a market analysis for a business plan. https://www.thebusinessplanshop.com/blog/en/entry/market_analysis_for_business_plan, 2016.

[99] International Electrotechnical Commission. IEC 61869-2 Standard, September 2012.

[100] Hirokazu Goto and Takashi Kato. Hall-effect current detector, April 2004. US Patent: U.S. Classification 324/117.00H, 324/117.00R; International Classification G01R15/20; Cooperative Classification G01R15/202; European Classification G01R15/20B.

[101] P. P. Freitas, R. Ferreira, S. Cardoso, and F. Cardoso. Magnetoresistive sensors. *Journal of Physics: Condensed Matter*, 19(16):165221, 2007.

[102] Wayne C. Goeke. Active shunt ammeter apparatus and method, March 2016. US Patent: International Classification H03F3/45, G01R19/00, H03F1/34, G01R1/20, G01R1/30; Cooperative Classification H03F2203/45138, H03F3/45475, G01R19/0092, G01R1/203, G01R19/0023, H03F2200/261, H03F1/34.

[103] B. Djokic. A new calibration system for small AC voltages at power frequencies. In *29th Conference on Precision Electromagnetic Measurements (CPEM 2014)*, pages 770–771, August 2014.

[104] E. Martí-Arbona, D. Mandal, B. Bakkaloglu, and S. Kiaei. A High-Voltage-Compliant Current-to-Digital Sensor for DC-DC Converters in Standard CMOS Technology. *IEEE Transactions on Power Electronics*, 32(3):2180–2188, March 2017.

[105] Yan Shen, Yunhe Lu, Zhao Liu, Xueliang Yu, Guoqing Zhang, and Wenbin Yu. Performance of magneto-optical glass in optical current transducer application. *Journal of Magnetism and Magnetic Materials*, 389:180–185, September 2015.

[106] D. Drung, J. H. Storm, and J. Beyer. SQUID Current Sensor With Differential Output. *IEEE Transactions on Applied Superconductivity*, 23(3):1100204–1100204, June 2013.

[107] Hua-Xin Peng, Faxiang Qin, and Manh-Huong Phan. Giant Magnetoimpedance Sensors and Their Applications. In *Ferromagnetic Microwire Composites*, Engineering Materials and Processes, pages 99–117. Springer International Publishing, 2016. DOI: 10.1007/978-3-319-29276-2_8.

[108] Y. S Didosyan, H Hauser, and J Nicolics. Magneto-optical current sensors of high bandwidth. *Sensors and Actuators A: Physical*, 81(1–3):263–267, April 2000.

[109] B. Yi, B. C. B. Chu, and K. S. Chiang. Magneto-optical electric-current sensor with enhanced sensitivity. *Measurement Science and Technology*, 13(7):N61, 2002.

[110] Petr Drexler and Pavel Fiala. Utilization of Faraday Mirror in Fiber Optic Current Sensors. *Radioengineering*, 17(4):101–107, December 2008.

[111] Kiyoshi Kurosawa, Kazunori Yamashita, Tomohiro Sowa, and Yasuhisa Yamada. Flexible Fiber Faraday Effect Current Sensor Using Flint Glass Fiber and Reflection Scheme. *IEICE TRANSACTIONS on Electronics*, E83-C(3):326–330, March 2000.

[112] Team Electronics Tutorials. Current Transformer Basics and Current Transformer Theory. http://www.electronics-tutorials.ws/transformer/current-transformer.html, September 2013.

[113] A. Marinescu. A calibration laboratory for Rogowski Coil used in energy systems and power electronics. In *2010 12th International Conference on Optimization of Electrical and Electronic Equipment*, pages 913–919, May 2010.

[114] E. Hemmati and S. M. Shahrtash. Digital Compensation of Rogowski Coil's Output Voltage. *IEEE Transactions on Instrumentation and Measurement*, 62(1):71–82, January 2013.

[115] J. P. Dupraz, A. Fanget, W. Grieshaber, and G. F. Montillet. Rogowski Coil: Exceptional Current Measurement Tool For Almost Any Application. In *2007 IEEE Power Engineering Society General Meeting*, pages 1–8, June 2007.

[116] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1st edition, 1994.

[117] M. Cirstea. Modeling and design of digital electronic systems. In *2016 International Conference on Development and Application Systems (DAS)*, pages 189–194, May 2016.

[118] CST Computer Simulation Technology AG. CST Microwave Studio - Workflow and Solver Overview Manual, 2016.

[119] David B. Davidson. *Computational Electromagnetics for RF and Microwave Engineering*. Cambridge University Press, October 2010.

[120] E. Hemmati and S. M. Shahrtash. Investigation on Rogowski coil performance for structuring its design methodology. *IET Science, Measurement Technology*, 7(6):306–314, November 2013.

[121] J. H. Stathis. Reliability limits for the gate insulator in CMOS technology. *IBM Journal of Research and Development*, 46(2.3):265–286, March 2002.

[122] M. Wang, A. J. Vandermaar, and K. D. Srivastava. Review of condition assessment of power transformers in service. *IEEE Electrical Insulation Magazine*, 18(6):12–25, November 2002.

[123] F. Legrand, H. Levi, N. Couture, and J. J. Charlot. VHDL-AMS modeling and library building for Power Electrical Engineering. In *The 8th IEEE International Workshop on Advanced Motion Control, 2004. AMC '04*, pages 111–116, March 2004.

[124] Franke Rüdiger and Wiesmann Hansjürg. Flexible modeling of electrical power systems – the Modelica PowerSystems library. In *Proceedings of the 10th International Modelica Conference*, pages 515–522, Lund, Sweden, March 2014.

[125] O. Enge-Rosenblatt, J. Haase, and C. Clauss. Important characteristics of VHDL-AMS and modelica with respect to model exchange. In *1st International Workshop on Equation-Based Object-Oriented Languages and Tools, EOOLT 2007*, pages 89–98, 2007.

[126] Asma Merdassi, Laurent Gerbaud, and Seddik Bacha. Automatic Generation of Average Models for Power Electronics Systems in VHDL-AMS and Modelica Modelling Languages. *ResearchGate*, 1(3):176–186, May 2010.

[127] Prakash Rashinkar, Peter Paterson, and Leena Singh. *System-on-a-Chip Verification: Methodology and Techniques*. Springer Science & Business Media, May 2007.

[128] C. Radojicic and C. Grimm. Instrumentation of the Control Flow of SystemC AMS - Models for Symbolic Simulation. In *ANALOG 2016; 15. ITG/GMM-Symposium*, pages 1–6, September 2016.

[129] David J. Lucia, Philip S. Beran, and Walter A. Silva. Reduced-order modeling: new approaches for computational physics. *Progress in Aerospace Sciences*, 40(1–2):51–117, February 2004.

[130] Alfio Quarteroni and Gianluigi Rozza. *Reduced order methods for modeling and computational reduction*, volume 9. Springer, 2014.

[131] SAE Electronic Design Automation Standards Committee. VHDL-AMS Statistical Analysis Packages. Standard J2748_200610, October 2006.

[132] J. Haase and C. Sohrmann. VHDL-AMS Statistical Analysis for marginal probabilities. In *2009 IEEE Behavioral Modeling and Simulation Workshop*, pages 114–119, September 2009.

## Juan Sebastián Rodriguez Estupiñán

Address: Chemin des Croix-Rouges 6
1007 Lausanne, Switzerland
Mobile: +41 78 616 44 81
E-mail: juansebastian.rodriguez22@gmail.com

### Strengths

- Multicultural and multidisciplinary experience.
- Result-oriented and self-disciplined Electrical and Electronics Engineer.
- Project Management and Leadership skills.

## QUALIFICATIONS PROFILE

- More than 6 years of experience in programmable logic design, simulation and implementation in FPGAs and microcontrollers.
- Comprehensive knowledge in embedded systems, analog and digital electronic design.
- Good knowledge in control engineering, power electronics, power management.
- Strong communication skills, mobility, reliability, initiative and teamwork skills.

## EDUCATION

**Ecole Polytechnique Fédérale de Lausanne (EPFL). Lausanne, Switzerland**

**2013 – 2017** Doctorate of Philosophy in Microsystems and Microelectronics (EDMI).
**2010 – 2012** Master of Science in Electrical and Electronics Engineering
Major: Microelectronics. Minor: Management of Technology and Entrepreneurship (MTE).

**Universidad de Los Andes. Bogota, Colombia**

**2003 – 2008** Bachelor of Science in Electronics Engineering (Graduation with Cum Laude).
**2003 – 2008** Bachelor of Science in Electrical Engineering (Graduation with Cum Laude).

## PROFESSIONAL EXPERIENCE & PROJECTS

**03/2013 – 01/2017. Ecole Polytechnique Fédérale de Lausanne & ABB, Lausanne, Switzerland: Doctoral Assistant**

- My research consists on the development of a design and verification methodology based on virtual prototyping for industrial Cyber-Physical Systems (CPS). In particular, my work shows that the use of virtual prototyping at early stages of complex system development reduces the overall design and verification effort by allowing the exploration of the complete system architecture and uncovering integration issues early on, thus reducing the developing time and costs. The main contributions of my research are: the re-definition of the system development life-cycle by using system level virtual prototypes; the design and implementation of a model library that maximizes the reuse of computational models and their related IP; and a set of VHDL-AMS modeling guidelines established with the purpose of improving the modularity and scalability of virtual prototypes. These techniques have been successfully used during the design of smart electrical sensors and monitoring equipment for high and medium voltage applications. This work has been carried out under a special collaboration between EPFL - LSM and the Integrated Sensors Group of ABB Corporate Research at Baden-Dättwil, Switzerland.
- Teaching assistant of Hardware Systems Modeling I & II Master level courses (4 years of experience in VHDL-AMS modelling, simulation and verification of digital and analog electronics).

**02/2012 – 08/2012 ABB Switzerland Ltd (Integrated Sensors Group), Baden, Switzerland: R&D Intern**

- Designed virtual prototypes of innovative industrial electrical sensors for High and Medium Voltage applications.
- Developed a robust and efficient model of an Electro-Optic Voltage Transducer (EOVT) which involves mixed-signal components and digital signal processing for accurate high voltage measuring. A fully synthesizable signal processing algorithm was developed for direct implementation in a FPGA system; so that, the current sensor system is improved and optimized.
- Designed a self-calibration and self-diagnostic system for enhancing Medium Voltage non-conventional sensors. The objective is to compensate the different possible drifts of simple and non-expensive sensors by adding more complex electronics associated with embedded signal processing. A robust, multi-physics and multi-language model was developed for temperature and noise analysis for different circuit topologies. The results obtained demonstrate that the proposed self-calibration methodology is feasible and patentable. Current commercial devices can be improved using this technique.

**02/2011 – 07/2011 Microelectronic Systems Laboratory (LSM) EPFL, Lausanne, Switzerland: R&D Intern**

- Designed PCBs for chip system testing in the context of 3D chip staking project of the Microelectronic Systems Laboratory. The new system works with the JTAG boundary scan standard, improving both the naked and packaged experimental chip testing.

**01/2010 – 09/2010 Czech Technical University in Prague (CVUT), Prague , Czech Republic: R&D Hardware Engineer**

- Designed an interface electronic board to carry out faster and reliable readouts of pixel detectors (Medipix & Timepix). This embedded system is able to store, process and transmit data at high frequency using an FPGA-DSP system, removing important limitations of current interfaces and providing enhanced operability.

**07/2009 – 12/2009 Colegio de Estudios Superiores de Administración (CESA), Bogota, Colombia: Assistant lecturer**

- Taught Integral calculus lectures focused for management students. Excellent results for the youngest docent in this college.

**02/2009 – 12/2009 Universidad de Los Andes, Bogota, Colombia: Research and Teaching Assistant**

- Active researcher of the Center for Microelectronics of Los Andes University. Co-advisor of several undergraduate projects related with Multiphysics design and modeling. FPGA and Embedded systems.
- Taught lab sessions and lectures on Materials and Semiconductor Devices. (Special assignation granted by the titular professor)
- Designed, implemented and developed an electrical engineering lab on Circuit Foundations.

**07/2008 – 01/2009 European Organization for Nuclear Research (CERN), Geneva, Switzerland: R&D Intern**

- Designed and implemented a multi-channel high-speed optical system based on a FPGA's network which emulates the data flow from the CMS-Preshower silicon strip detectors, for functional testing and commissioning of the detector acquisition system.
- Carried out clean room tests of Preshower hybrid micromodules to be installed in the CMS detector of the Large Hadron Collider (LHC). Carried out pin-to-pin length measurements of the Preshower's support in the installation disks of the CMS end caps.

## TECHNICAL SKILLS

- **VLSI :** Software tools for Full & Semi-Custom design and verification: (Cadence, Mentor Graphics, Dolphin Integration, Synopsis)
- **Multiphysics modeling:** COMSOL Multiphysics, ANSYS (Electromechanical & Electrothermal device modeling).
- **FPGA programing & Verification:** ISE (Xilinx); Quartus II, NIOS IDE, SOPC Builder (Altera); ISP Lever (Lattice Semiconductor); ModelSim.
- **PCB design & IDEs:** Altium Designer, OrCAD, Eagle, Code Composer Studio.
- **Industrial Automation:** LabVIEW(Core I certified), LTSpice, Siemens PLC SIMATIC 300 Family, serial communication protocols.
- **Programming Languages:** C/C++, VHDL-AMS, Verilog, Python, Visual Basic, GRAFCET, SML, Spice, Assembler, MATLAB.
- **Database software:** Filemaker pro, Filemaker server, Microsoft Access.
- **Applications:** MATLAB, Autocad, Microsoft Office software, Adobe Photoshop, GIT, LaTeX.
- **OS:** MAC OS, Windows & Linux.

## HONORS AND AWARDS

**2011 Ecole Polytechnique Fédérale de Lausanne (EPFL):** Scholarship granted by the Social Commission of EPFL.

**2008 Los Andes University (Bogota, Colombia):** Graduation with Cum Laude in both Electrical and Electronics Engineering.

**2008 High Energy Physics Latin-American-European Network - HELEN:** Six-month fellowship at CERN.

**2007 Publicar S.A. (Bogota, Colombia):** High GPA Scholarship sponsored by this company.

**2001 Liceo San Basilio Magno High School (Bogota, Colombia):** Ranked First among 42 Fellow Graduated Students.

## PUBLICATIONS

- **Rodriguez Estupinan, J.S.**, Vachoux, A.; Pascal, J. Electro-thermal virtual prototyping of a Rogowski Coil sensor system. IEEE international conference on Industrial Technology (ICIT), 17 - 19 March 2015, Seville, Spain.
- Qianqian Ha; Maret, Y.; **Rodriguez Estupinan, J.S.**; Vachoux, A. VHDL-AMS virtual prototyping of a generator circuit breaker ablation monitoring system. IEEE international conference on Circuit and Systems (ISCAS), 24-27 May 2015, Lisbon, Portugal.
- **Rodriguez Estupinan, J.S.**, Vachoux, A.; Pascal, J.Electro-thermal modeling of a Rogowski coil sensor system. IEEE international Symposium on VLSI Design, Automation and Test (VLSI-DAT), 27-29 April 2015, Hsinchu, Taiwan.
- **J.S. Rodriguez**, S. Bonilla and A. Ávila. "Mediciones de Rompimiento Electrostático en Separaciones Micrométricas" (Electrostatic Breakdown Measurements in Micro-gaps), *Revista de Ingeniería*, R.29 Sección Técnica, May 2009. Electronic-version ISSN 2011-0049. Print-version ISSN 0121-4993.
- G. Antchev, D. Barney, W. Bialas, R.S. Bonilla Osorio, K.-F. Chen, C.-M. Kuo, R.-S. Lu, V. Patras, S. Reynaud, **J.S. Rodriguez Estupiñan**, P. Vichoudis "Commissioning and performance of the Preshower off-detector readout electronics in the CMS experiment", Published in Proceedings of the Topical Workshop on Electronics for Particle Physics TWEPP-09. 21-25 September 2009, Paris, France.
- **J.S. Rodriguez**, A. Avila, D.F. Reyes. "Modelaje y simulación de un biosensor para detección bioquímica extra/intracelular" (Modeling and Simulation of a Silicon Nanowire-Based Sensor for Extra/Intracellular Biochemical Detection). Published in Proceedings of the 6th Ibero-American Congress on Sensors – IBERSENSOR. 24-26 November 2008, Sao Paulo, Brazil.

## LANGUAGES

| Spanish | English | French | German |
|---|---|---|---|
| Native language | Fluent (C1) | Intermediate (B1-B2) | Beginner (A1) |

## EXTRA-CURRICULAR ACTIVITIES

- Music: Salsa dancer, Guitar and Percussion player.
- Sports: Football, Volleyball, Squash, Ski, Rafting.
- Member of the Colombian Association of Researchers in Switzerland (ACIS)

## Personal Situation

32 years old, Swiss residence permit (type B), Married, No children.