Evaluation of system-on-chip devices for embedded real-time simulators of electrical systems

Daniel Tormo Borreda

EVALUATION OF SYSTEM-ON-CHIP DEVICES FOR EMBEDDED REAL-TIME SIMULATORS OF ELECTRICAL SYSTEMS

DANIEL TORMO BORREDÀ

Université / / Paris Seine

A dissertation for becoming Doctor of Philosophy in
Electrical and Electronic Engineering

11th July 2018 – version 1.1

Année 2018

Université Paris Seine

THÈSE

Présentée pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ PARIS SEINE

École doctorale: Sciences et Ingénierie

Spécialité: Génie Électrique et Électronique

Soutenue publiquement le 11 Juillet 2018 par

DANIEL TORMO BORREDÀ

EVALUATION OF SYSTEM-ON-CHIP DEVICES FOR EMBEDDED REAL-TIME SIMULATORS OF ELECTRICAL SYSTEMS

Université / / Paris Seine

Laboratoire de *Systèmes et Applications des Technologies de l'Information et de l'Énergie* (SATIE) - CNRS UMR8029

Devant le Jury composé par

Président :	Prof. Serge PIERFEDERICI	Univ. de Lorraine
Rapporteurs :	Prof. Guillaume GATEAU	Univ. de Toulouse
	Prof. Mickaël HILAIRET	Univ. de Franche-Comté
Examinateur :	Prof. Ramón BLASCO-GIMÉNEZ	Univ. Politècnica de València
Directeur de Thèse :	Prof. Eric MONMASSON	Univ. Paris Seine
Co-encadrant de Thèse :	Ph.D. Lahoucine IDKHAJINE	Univ. Paris Seine

Daniel Tormo Borredà: *Evaluation of System-on-Chip devices for Embedded Real-Time Simulators of Electrical Systems,* A dissertation for becoming Doctor of Philosophy in Electrical and Electronic Engineering, © 11th July 2018

Per als meus pares i germans. Per als meus nebots. Per a tots aquells que en major o menor mesura m'han ajudat a ser la persona que sóc. Esta tesi no haguera sigut possible sense tots vosaltres.

A tots, gràcies de tot cor.

I recordeu: sigueu curiosos. La curiositat és la recerca constant de coneixement.

— D. Tormo

To my parents and brothers. To my niece and nephews. To all those who to a greater or lesser extent have helped me become the person I am. This thesis would not have been possible without all of you.

Thank you very much indeed.

And remember: be curious, because curiosity is the constant pursuit of knowledge.

— D. Tormo

ABSTRACT

This Doctoral Thesis is a detailed study of how suitable *System-on-Chip* (SoC) devices are for implementing *Embedded Real-Time Simulators* (eRTS) of electromechanical and power electronic systems. This emerging class of *Real-Time Simulators* (RTS) are not only expected for *Hardware-in-the-Loop* (HIL) validations of systems; but they also have to be embedded within the controller to play several roles like observers, parameter estimation, diagnostic, health monitoring, fault-tolerant and *sensorless* control, etc.

The design of these *Intellectual Properties* (IP) must rigorously consider a set of constraints at different development stages: (i) during the modeling of the system to be real-time simulated; (ii) during the digital realization of the IP; and also (iii) during its final implementation in the digital platform. Thus, the conducted work of this Thesis focuses specially on this last stage and its aim is to evaluate the time/resource performances of recent SoC devices and study how suitable they are for implementing eRTSs. These kind of digital platforms combine powerful general purpose processors, a *Field-Programmable Gate Array* (FPGA) and other peripherals which make them very convenient for controlling and monitoring a complete system.

One of the limitations of these devices is that control engineers are not particularly familiarized with FPGA programming, which needs extensive expertise in order to code these highly sophisticated algorithms using *Hardware Description Languages* (HDL). Notwithstanding, there exist *High-Level Synthesis* (HLS) tools which allow to program these devices using more generic programming languages such as C, C++ or SystemC. Moreover, by inserting directives and constraints to the source code, these tools can produce different hardware implementations (e.g. full-combinatorial design, pipelined design, parallel or factorized design, partition or arrange data for a better utilisation of memory resources, etc.).

This dissertation is based on the implementation of two representative applications that are well known in our laboratory: a *Doubly-fed Induction Generator* (DFIG) commonly used as wind turbines; and a *Modular Multi-level Converter* (MMC) that can be arranged in different configurations and utilized for many different energy conversion purposes. Since the DFIG has low/medium system dynamics (electrical and mechanical ones), both a *full-software* implementation using solely the ARM processor and a *full-hardware* implementation using HLS to program the FPGA will be evaluated with different design optimizations and data formats (64/32-bit floating-point and 32-bit fixed-point). Moreover, it will also be investigated whether a system of these characteristics is interesting to be run as a *hardware accelerator*. Different data transfer options between the *Processor System* (PS) and the *Programmable Logic* (PL) have been studied as well for this matter. Conversely, because of its harsh dynamics (switching dynamics), the MMC will be implemented only with a *full-hardware* approach using HLS tools, as well.

For the experimental validation of this Thesis work, a complete MMC test bench has been built from scratch in order to compare the real-world results with its SoC eRTS implementation.

Keywords – Embedded Real-Time Simulator, System-on-Chip, High-Level Synthesis, Modular Multi-level Converter, Field-Programmable Gate Array

RÉSUMÉ

L'objectif de ce travail de Thèse est d'évaluer les capacités de composants numérique de type *Système-sur-Puce* (SoC en anglais) pour l'implantation de *Simulateurs Temps Réel Embarqués* (eRTS en anglais) de systèmes électromécaniques et d'électronique de puissance. En effet, l'utilisation de ces simulateurs n'est pas seulement limitée aux validations *matériel dans la boucle* (en anglais *Hardware-in-the-Loop* ou HIL) du système mais doivent également être embarqués avec le contrôleur afin d'assurer plusieurs fonctionnalités additionnelles comme l'observation, l'estimation, commande sans capteur (ou *sensorless*), le diagnostic ou la surveillance de la santé, commande tolérante aux défauts, etc.

La réalisation de ces simulateurs doit néanmoins considérer plusieurs contraintes à plusieurs niveaux de développement : durant la modélisation de la partie du système à simuler en temps-réel, durant la réalisation numérique et enfin durant l'implantation sur le composant numérique utilisé. Ainsi, le travail réalisé durant cette Thèse s'est focalisé sur ce dernier niveau et l'objectif était d'évaluer les capacités temps/ressources des composants de type SoC pour l'implantation de modules eRTS. Ce type de plateformes intègrent dans un même composant de puissants processeurs, un circuit logique programmable (*Field-Programmable Gate Array* ou FPGA), et d'autres périphériques, ce qui offre plusieurs opportunités d'implantation.

Afin de pallier les limitations liées au codage VHDL de la partie FPGA, il existe des outils *High-Level Synthesis* (HLS) qui permettent de programmer ces dispositifs en utilisant des langages à haut niveau d'abstraction comme C, C++ ou SystemC. De plus, en incluant des directives et contraintes au code source, ces outils peuvent produire des implémentations matérielles différentes (architecture totalement combinatoire, « pipeline », architecture parallélisées ou factorisées, arranger les données et leurs formats pour une meilleure utilisation des ressources de mémoire, etc.).

Dans le but d'évaluer ces différentes implantations, deux cas d'études ont été choisis : le premier se compose d'un *Générateur Asynchrone à Double Alimentation* (GADA) et le second d'un Convertisseur Modulaire Multiniveau (ou *Modular Multi-level Converter* - MMC). Vu que la GADA a une dynamique basse/moyenne (dynamiques électriques et mécaniques), deux versions d'implantations ont été évaluées : (i) une implantation *full-software* en utilisant seulement les processeurs ARM; et (ii) une implantation *full-hardware* en utilisant l'outil HLS pour programmer la partie FPGA. Ces deux versions ont été évaluées avec différentes optimisations du compilateur et trois formats de données: 64/32-bit en virgule flottante, et 32-bit en virgule flottante. L'approche mixe *software/hardware* a également été évaluée à travers la caractérisation des transferts de données entre le processeur et l'IP eRTS implantée dans la partie FPGA. Quant au convertisseur MMC, sa complexité et sa forte dynamique (dynamique de commutation) impose une implantation exclusivement *full-hardware*. Celle-ci a également été réalisée à base d'outils HLS.

Enfin pour la validation expérimentale de ce travail de Thèse, une maquette à base de convertisseur MMC a été construite dans le but de comparer des mesures du système réel avec les résultats fournis par l'IP eRTS.

Mots clés – Simulateurs en temps réel embarqués, Système-sur-puce, Synthèse de haut niveau, Convertisseur Modulaire Multiniveaux, Field-Programmable Gate Array

RESUM

L'objectiu d'aquest treball de Tesi és avaluar les capacitats de dispositius digitals de tipus *System-on-Chip* (SoC) per a la implantació de *Simuladors en Temps Real Embarcats* (*Embedded Real-Time Simulators* o eRTS en anglès) de sistemes electromecànics i d'electrònica de potència. Aquesta classe emergent de *Simuladors en Temps Real* (RTS) no estan pensats només per a realitzar validacions de controladors amb *Hardware-in-the-Loop* (HIL); aquests deuen estar embarcats amb el controlador per a executar diferents tasques tals com observadors, estimació de paràmetres, diagnòstic, monitorització de l'estat de salut del sistema, control amb tolerància a fallades o *sensorless*, etc.

El disseny d'aquestes *Propietats Intel·lectuals* (IP) deu considerar rigorosament certes restriccions en cadascuna de les etapes de desenvolupament: (i) durant el modelat del sistema a ser simulat en temps real; (ii) durant la discretització de l'IP; i també (iii) durant la fase final d'implementació en la plataforma digital escollida. Per això, el present treball s'enfoca específicament en aquesta última etapa i el seu objectiu és avaluar el rendiment considerant temps d'execució i recursos utilitzats dels dispositius SoC i estudiar quant apropiats són per a implementar eRTSs. Aquests tipus de plataformes combinen potents processadors de propòsit general, *Field-Programmable Gate Arrays* (FPGA), juntament amb altres perifèrics que els fan molt adequats per a controlar i monitoritzar un sistema complet.

Una de les limitacions d'aquests dispositius és que els enginyers de control no estan particularment familiaritzats amb la programació d'FPGAs, els quals requereixen una gran experiència per a programar els algoritmes altament sofisticats utilitzant *Llenguatges de Descripció Hardware* (HDL). No obstant això, existixen ferramentes de Síntesi d'Alt Nivell (*High-Level Synthesis* en anglès o HLS) que permeten programar aquests dispositius utilitzant llenguatges de programació més genèrics tals com C, C++ o SystemC. A més a més, inserint directives i restriccions al codi font, aquestes ferramentes poden produir diferents implementacions hardware (e.g. disseny completament combinacional, disseny segmentat, disseny paral·lel o factoritzat, partint o reordenant les dades per a una millor utilització dels recursos de memòria, etc.).

Aquesta dissertació està basada en la implementació de dos aplicacions representatives que són ben conegudes al nostre laboratori: un *Generador Inductiu Doblement Alimentat* (*Doubly-fed Induction Generator* en anglès o DFIG) utilitzat normalment en turbines eòliques; i un Convertidor Modular Multinivell (*Modular Multi-level Converter* o MMC) que pot ser assemblat en diferents configuracions i utilitzat en diverses aplicacions de conversió d'energia. Ja que el DFIG té dinàmiques baixes/mitjanes (elèctriques i mecàniques), dos implementacions una completament software usant només el processador ARM i una completament hardware utilitzant HLS per a programar l'FPGA seran avaluades utilitzant diferents optimitzacions i formats de dades (64/32-bit en coma flotant i 32-bit en coma fixa). A més a més, serà investigat si un sistema d'aquestes característiques és adequat d'ésser implementat com a accelerador hardware. Diferents modes de transferència de dades entre el *Sistema Processador* (PS) i la *Lògica Programable* (PL) han sigut estudiades per a aquest propòsit. Contràriament, a causa de les dinàmiques exigents de l'MMC (dinàmiques de commutació), aquest només serà implementat completament en hardware utilitzant també ferramentes HLS.

Per a la validació experimental d'aquest treball de Tesi, una maqueta completa d'un MMC ha sigut construïda des de zero per tal de comparar resultats reals amb el seu eRTS implementat en un SoC.

Keywords – Simuladors en Temps Real Embarcats, System-on-Chip, Síntesi d'alt nivell, Convertidor Modular Multinivell, Field-Programmable Gate Array

PUBLICATIONS

The following publications have been issued during the development of this dissertation:

Published papers

[1] Daniel Tormo, Lahoucine Idkhajine, Eric Monmasson, Ramón Blasco-Gimenez, *Evaluation of SoC-based embedded real-time simulators for electromechanical systems*, presented at: IECON 2016 - 42nd Annual Conference of IEEE Industrial Electronics Society, 23-26 October, Florence, Italy. DOI: 10.1109/IECON.2016.7793185

[2] Daniel Tormo, Lahoucine Idkhajine, Eric Monmasson, Ramón Blasco-Gimenez, *Embedded real-time simulator implementations of electromechanical systems using system-on-chip devices*, presented at: ELECTRIMACS 2017, 4-6 July, Toulouse, France.

[3] Daniel Tormo, Ricardo Vidal-Albalate, Lahoucine Idkhajine, Eric Monmasson, Ramón Blasco-Gimenez, *Study of system-on-chip devices to implement embedded real-time simulators of modular multi-level converters using high-level synthesis tools*, presented at: 2018 IEEE International Conference on Industrial Technology (ICIT), 20-22 February, Lyon, France. DOI: 10.1109/ICIT.2018.8352393

[4] Daniel Tormo, Ricardo Vidal-Albalate, Lahoucine Idkhajine, Eric Monmasson, Ramón Blasco-Gimenez, *Modular multi-level converter hardware-in-the-loop simulation on low-cost system-on-chip devices*, will be presented at: IECON 2018 - 44th Annual Conference of IEEE Industrial Electronics Society, 21-23 October, Washington DC, USA.

Publication pending

[5] Daniel Tormo, Ricardo Vidal-Albalate, Lahoucine Idkhajine, Eric Monmasson, Ramón Blasco-Gimenez, *Embedded real-time simulators for electromechanical and power electronic systems using system-on-chip devices*, to be published in a special issue of Transactions of IMACS - Mathematics and Computers in Simulation (MATCOM) from Elsevier.

Great minds discuss ideas;
Average minds discuss events;
Small minds discuss people.

— Eleanor Roosvelt

ACKNOWLEDGMENTS

I would like to give special thanks to Professor Eric Monmasson, Professor Ramón Blasco Giménez and Dr. Lahoucine Idkhajine for guiding me in the longest and darkest hours of this PhD experience. It was indeed a hell of an experience, in the good sense.

Special mention as well to all the administrative staff of the Université Paris Seine (formerly Université de Cergy-Pontoise): to Mme. Aude Brebant, to Mme. Marie-Hélène Moreau and specifically to Don Abasse Boukary; for his kindness, his always smiling face, and because he always had a solution for no matter what problem I had. Thanks a lot indeed Abasse!

Other people who I want to show my gratitude is to Mr. Miguel Albero Gil, Mr. Raúl Morillas Pérez, Dr. Ricardo Vidal Albalate, and Professor Ramón Blasco Giménez. The experimental work accomplished in Chapter 5 that was carried out at the *Institut d'Automàtica i Informàtica Industrial* (AI2) at the *Universitat Politècnica de València* (UPV), Spain, would not have been possible without their help and effort.

Thanks as well to the amazing colleagues I met in the laboratory: to Wided Zine who was always bringing sweets from Tunisia and rushing to get the RER A; to Gianluca Vicidomini and Sarah Ciaglia for the amazing trips to Normandie, Bretagne and the amazing week in Salerno; to Amina Mseddi who brought sweets and food as well and for our conversations letting us learn more about our different cultures; to Hanen Arfaoui for her discussions about life and for her mama's harissa; and to Fabio and Marco, for their Italian coffee and the coffee maker they let me!

And last but not least to Mathilde Hay, parce que sans toi, ces trois années de thèse à Paris n'auraient pas été aussi spéciales et amusantes. Merci de tout cœur pour tout ce temps qu'on a passé ensemble, pour ta patience, pour ne pas t'énerver contre moi pour arriver trop tard à diner chez toi, pour ces escapades touristiques qui m'aidaient à reprendre des forces pour continuer, et parce que si je parle français aussi bien c'est grâce à toi Coucou ! Il y a pas de mots pour exprimer les remerciements que je devrais te faire. Merci beaucoup ma chère Mathilde.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTE DES SYMBOLES

ADC	Analog-to-Digital Converter
AMBA	Advanced Microcontroller Bus Architecture
AXI	Advanced Extensible Interface
CPLD	Complex Programmable Logic Device
DAC	Digital-to-Analog Converter
DDR	Double Data Rate memory

DFIG	Doubly-Fed Induction Generator
DMA	Direct Memory Access
ERTS	Embedded Real-Time Simulator
FPGA	Field-Programmable Gate-Array
HB	Half-Bridge
HDL	Hardware Description Languages
HLS	High-level Synthesis
I/O	Input/Output
IP	Intellectual Property
MMC	Modular Multi-level Converter
MSPS	Mega Sample Per Second
OCM	On-chip Memory
OPAMP	Operational Amplifier
PCB	Printed Circuit Board
PL	Programmable Logic
PS	Processor System
PWM	Pulse-Width Modulation
QoS	Quality-of-Service
RT	Real-Time
RTL	Register Transfer Level
SM	Submodule
SoC	System-on-Chip
VHDL	VHSIC Hardware Description Language
XADC	Xilinx Analog-to-Digital Converter

Part I

GENERAL INTRODUCTION AND STATE OF THE ART

1

GENERAL INTRODUCTION

In the course of the last two decades, real-time digital simulation has been under the focus of important research in almost any area of engineering. Its aim is mainly to provide control engineers with another testing scenario right after the offline simulation verification –but much closer to reality than this one, before its trial and deployment on the real plant. Actually, a real-time simulation differs from a standard offline simulation in that the system behavior is simulated in natural time by implementing its dynamical model on hardware. As a consequence, the design of a *Real-Time Simulator* (RTS) includes additionnal constraints to the development cycle which can be classified in three levels: (i) the modeling of the system to be controlled, (ii) its digital realization, and (iii) its digital implementation.

These RTS have been used mainly as *Hardware-in-the-Loop* (HIL) testing of controllers, where the plant model is executed at the same pace as the real system providing a similar dynamic response. Even though the focus of this thesis is very much related with them, its aim goes a bit further, where these RTS will be embedded within the controller to provide additional functionalities. These *Embedded Real-Time Simulators* (eRTS) could play several roles like observators, estimators, diagnosis and health-monitoring, or fault-tolerant and *sensorless* control to mention some of them.

Nevertheless, when implementing such model-based eRTS, the main concern is how to balance between the accuracy of the model, the system dynamics, the simulation time-step (implicitly the execution time) and the needed mathematical computations to be executed in the chosen digital platform. The main challenge when designing these simulator *Intellectual Properties*[1] (IPs) is to cope with their complexity having in mind that, in the case of embedded systems, the available hardware resources are limited. Moreover, this challenge is increased by the requirement of very short simulation time-steps when simulating power electronics [2].

To bring technological solutions to these implementation issues, for some years there has existed a new kinds of devices called *System-on-Chip* (SoC) which integrate in the same device CPU-based processor cores along with a *Field-Programmable Gate Array* (FPGA) fabric and other peripherals such as ADCs, DACs, DMAs, and hardware interfaces which provide these platforms with great connectivity capabilities.

Alongside these devices, the design tools have evolved as well aiming to reduce development time of the increasing complexity of nowadays controllers, providing professional tools that ease the utilisation and exploitation of SoC devices.

1 An IP core is a block of logic or data that is used in making an FPGA or *Application-Specific Integrated Circuit* (ASIC) for a product. As essential elements of design re-use, IP cores are part of the growing *Electronic Design Automation* (EDA) industry trend towards repeated use of previously designed components. Ideally, an IP core should be entirely portable –that is, able to easily be inserted into any vendor technology or design methodology. *Universal Asynchronous Receiver/Transmitter* (UART), *Central Processing Units* (CPU), Ethernet controllers, *Universal Serial Bus* (USB) and PCI interfaces are all examples of IP cores [1].

1.1 THESIS OBJECTIVES AND AUTHOR CONTRIBUTIONS

The goal of this dissertation is then to evaluate and analyze the capabilities of these kinds of devices and the software tools that come with them for the implementation of accurate and reliable eRTS of electrical systems. In this work, the *Zynq-7000 All Programmable SoC* from *Xilinx Inc.* has been selected and its choice will be motivated later on.

To this purpose, two case applications have been chosen. *Offshore Wind farm* are composed of several subsystems. Among them, electrical machines (wind generators) and power electronic systems (power converters) are significantly different in terms of requirements, and quite representative of what SoC devices can be used for.

Regarding the electrical machine, the decision of choosing a *Doubly-fed Induction Generator* (DFIG) has been taken because it is a quite popular generator used as wind turbines and it has a representative model complexity compared to other electromechanical systems [3]. The specific application is a DFIG working as an islanded generator and feeding a three-phase RL load [4, 5, 6, 7].

Concerning power electronics systems, some might think that it would have been more appropriate to use the classical two-level power converter that is usually connected to the DFIG in order to test the device in a complete power electronics application. However, this work has already been done; it can be found extensively in the literature, and it does not have the amount of complexity to evaluate the full capabilities of a SoC device [8, 9, 10]. Therefore, in this work it has been decided to use a *Modular Multilevel Converter* (MMC) [11]. The aim of choosing it is however to use a scalable power converter, with a much more complex structure in order to exploit the device capabilities at maximum, coping with the eRTS requirements of speed, computational power, and degree of parallelism [12, 2, 13, 14].

In more detail, the idea is to have one –or several– IP blocks which estimate: e.g. stator and rotor currents and voltages, phase output voltages, arm and load currents, *Half-Bridge* (HB) capacitor voltages, etc.; based on other measured magnitudes. One of the most interesting applications these IP blocks could be utilized for is in the context of fault-tolerant embedded controllers, where these estimated variables would be used in case of a sensor fault. Furthermore, they could also be adopted as estimators, observers, or also applied for diagnosis and health-monitoring [8, 15], without forgetting HIL applications for testing the control before its deployment on the real plant [16, 17, 18, 19, 8], and *sensorless* control [20, 21, 22, 23].

Accordingly, several IP blocks will be developed. On one hand, by the creation of several DFIG models using two different discretization methods and exploring whether is more suitable to execute the coded algorithms either in software using the *Processing System* (PS), in hardware using its *Programmable Logic*[2] (PL), or a collaboration between both. And on the other hand, by the conception of various MMC models to estimate different converter variables, modifying the number of cells per arm to exploit all the computational power of the device.

Moreover, a deep exploration of the *Vivado HLS* software tool will be performed as well aiming to reduce the execution time at maximum, using different data formats (*fixed-* and *floating-point*) and data widths (64- and 32-bit) for the internal and external variables. The device performance will be compared in terms of execution time, resources utilisation and precision of the calculations.

2 A *Programmable Logic Device* (PLD) is an electronic component used to build reconfigurable digital circuits. Unlike a logic gate, which has a fixed function, a PLD has an undefined function at the time of manufacture. Before the PLD can be used in a circuit it must be programmed, that is, reconfigured [24].

In this study, a total of three boards mounting two different *Zynq* versions will be utilized. The *ZedBoard*[3] and the *MicroZed*[4] development boards manufactured by *Avnet Inc.*, both mounting a *Zynq-7020* classified as mid-range with an *Artix-7* equivalent FPGA fabric, will be used for the DFIG study and for the experimental tests respectively. For the MMC case evaluation however, the *ZC706 Evaluation Kit* produced by *Xilinx Inc.* will be utilized. The latter mounts a high-end *Zynq-7045* having a *Kintex-7* equivalent FPGA fabric with considerably more hardware resources.

To conclude, an MMC test rig has been developed and used to prove a real-world implementation of eRTS. First, by using one IP to perform HIL validation of the controller before its deployment on the real plant; and second, by implementing a capacitor voltage estimator for fault-tolerant control.

This thesis work is mainly focused on the digital implementation issues of eRTSs. In fact, the proposed work can be seen as an extension of Dr. Dagbagi's thesis [25] where the focus was limited to *full-hardware* FPGA-based implementation of eRTSs. However, the implementation constraints have been extended to consider SoC devices, which allow *full-hardware*, *full-software* and mixed hardware/software solutions.

1.2 THESIS OUTLINE

The rest of this thesis dissertation is organized as follows: Chapter 2 presents the State of the Art and introduces the main problems linked to the development of eRTSs. Then it focuses on the digital implementation by presenting the SoC technology and the programming tools linked to them. Chapter 3 is dedicated to eRTSs for electromechanical systems, hence the DFIG case. Chapter 4 is devoted to the eRTS for power electronic systems, therefore the MMC case. In Chapter 5 an exhaustive experimental validation of the MMC implementation is performed. Finally, Chapters 6 and 7 are left for general conclusions and future work respectively. At the end of this document Appendix A can be found with the description of the test bench, and Appendix B with the parameters of the state space model utilised in the experimental tests.

3 Official website *zedboard.org*.
4 Official website *microzed.org*

2

STATE OF THE ART

2.1 INTRODUCTION

This chapter discusses the main problems linked to the development of eRTS of electrical systems. They can be located at the main development stages of the eRTSs and can be classified as follows: (i) Selection of the system's mathematical model, (ii) digital realization of this model and (iii) its digital implementation.

1. *Modeling*. A model of the plant has to be found, with a good approach to emulate the system's behavior using appropriate equations. Regarding power systems dynamics, they can be divided into two groups:
   - Electromechanical and electromagnetic systems, which have relatively slow dynamic response
   - And power electronics systems, which have much more fast dynamics
2. *Digital realization*. In this step have to be chosen: a –constant– time-step, a numerical solver (explicit or implicit), and an appropriate data format (fixed- or floating-point), aiming always at the highest level of precision considering the computational power available.
3. *Implementation*. The last step is where the simulation platform to be used has to be decided. There exist many different possibilities ranging from powerful multi-core processors, FPGAs, or SoC devices to mention the most commonly used nowadays. The methodology when programming each of these platforms differ significantly and so do the software tools utilised. Moreover, the programming languages are also a big issue specially when FPGAs are involved.

2.2 WHAT IS AN EMBEDDED REAL-TIME SIMULATOR?

The *real-time* concept has been extensively used by the industry but worn and utilised inappropriately most of the time by the general public. According to [26] "*A real-time system can be described as one which controls an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that time*". By using this description one could think that every controller is implemented in real-time. In regard to the subject we are on, this other more appropriate and specific definition is preferred: "*Real-time is an adjective describing a system in which results and feedback follow input with no noticeable delay from the system's point of view*". Accordingly, the timing requirements will rely upon the dynamics of the system under study.

A system is said to be in real-time if the total correctness of an operation depends not only on its logical fidelity, but also on the time in which it is performed. According to [27],

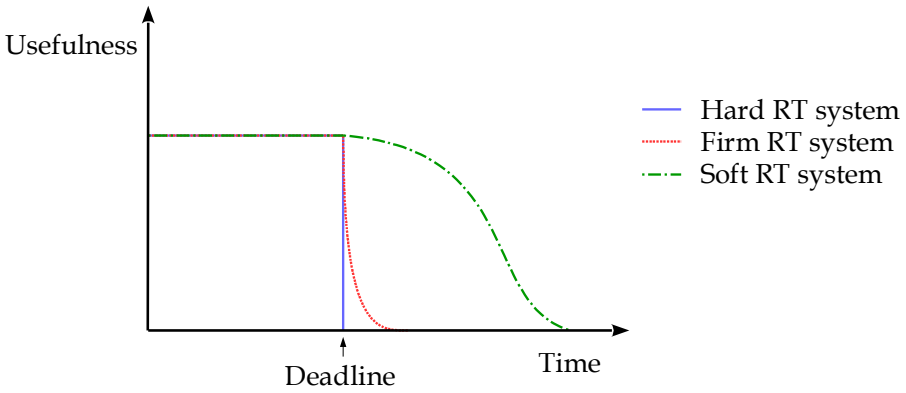


Figure 2.1: Real-Time systems classification

real-time systems, as well as their deadlines, are classified by the consequence of missing a deadline:

- *Hard*: any missed deadline is a total system failure
- *Firm*: infrequent deadline misses are tolerable, but may degrade the system's *Quality of Service*[1] (QoS)
- *Soft*: The usefulness of a result degrades after its deadline, thereby degrading the system's QoS

Figure 2.1 shows a representation of this classification.

Moving forward on the definition, a *Real-Time Simulator* (RTS) can be seen as a digital entity that mimics a real-world system or a part of it, which is executed at a rate that matches the one of the real process. As it will be explained later, this digital entity is executed at a constant time-step. Any missed deadline due to implementation issues is not tolerated, causing a simulation failure. Hence, RTSs can be classified as *hard real-time systems*.

When applied to electrical systems, RTS are mostly implemented in the context of *Hardware-in-the-Loop* (HIL) testing of the system controller in order to validate it in every operating condition, which would not be possible using only experimental setups [18, 19, 16, 17]. However, an RTS can also be embedded within the controller of the system in what is known as an *Intellectual Property*[2] (IP) to play several roles like observators, estimators, model-based diagnostics, fault-tolerant and sensorless control, health monitoring, etc. It is in this context where such IP modules are called *Embedded Real-Time Simulators* (eRTS) [25, 8, 6, 28, 7, 14].

Apart from the execution rate, another constraint when designing an eRTS is to cope with its complexity having in mind that, in the case of embedded systems, the available hardware resources of the digital platform are limited.

Moreover, this challenge is increased by the requirement of very short simulation time-steps when simulating power electronics [15]. This is the reason why at each development stage,

1 Description or measurement of the overall performance of a service.

2 An IP core is a block of logic or data that is used in making an FPGA or *Application-Specific Integrated Circuit* (ASIC) for a product. As essential elements of design re-use, IP cores are part of the growing *Electronic Design Automation* (EDA) industry trend towards repeated use of previously designed components. Ideally, an IP core should be entirely portable - that is, able to easily be inserted into any vendor technology or design methodology. *Universal Asynchronous Receiver/Transmitter* (UART), *Central Processing Units* (CPU), Ethernet controllers, *Universal Serial Bus* (USB) and PCI interfaces are all examples of IP cores [1].

designer must rigorously consider a set of constraints linked to: (i) the chosen model of the system part to be RT simulated; (ii) its digital realization and (iii) its implementation on the targeted digital platform.

In the following, these development constraints will be discussed. Since the outline of this thesis is more about the implementation issues, an important focus will be given to the description of the available SoC FPGA platforms and their value-added to this context.

2.3 ERTS DEVELOPMENT (I) : SYSTEM MODELING

To face today's society and industry demands, the electrical systems have been the focus of intensive research starting from power generation, storage, until its consumption. As a consequence, a wide range of increasingly complex machine drives, power electronic converters and their controllers are now available in the market. This trend makes their design very challenging since the time-to-market and the growing development cost must also be considered. All these reasons make the real-time digital simulation of these electrical systems mandatory in modern design cycles.

The first thing that influences the quality of a real-time simulation is the chosen model. It has to be properly formulated in order to represent the static and dynamic behavior of the system accurately. However, it has to be understood that a mathematical model of a system is a simplified representation of the reality. This simplification has to be done mainly because of two reasons: (i) models can become extremely complicated and hence, their development time can be significantly long; and (ii), these models have to be executed in a digital device, so execution time and resources must be considered. Therefore, neglecting physical phenomena that occur in the system (e.g. saturations, core losses, saliencies, power converter switching, skin effects, etc.) has to be assumed in most of the cases. This becomes even more important when considering RTS where the execution time of the mathematical model is a critical constraint.

An electrical system can be described as a set of interconnected elements that have different characteristics and dynamics. Naming them and starting from slow to fast dynamics: thermal dynamics, electromechanical dynamics, electrical dynamics, electromagnetic dynamics, and power electronics dynamics. Due to their wide gap between them, these phenomena can be grouped in two: *fast dynamic systems* considering only the power electronics systems, and *slow dynamic systems* grouping the rest [25]. In this dissertation, an example representing each of the groups will be studied.

2.4 ERTS DEVELOPMENT (II) : DIGITAL REALIZATION

Besides its mathematical modeling, the real-time simulation of an electrical system is also conditioned by the digital realization of the chosen models. This consists in selecting the appropriate numerical solver, the right simulation time-step, and the numerical representation of the processed data. The choice has to consider: the *complexity* (number of operations to implement), *accuracy* (contrasted to the original continuous-time model); *numerical stability* (the system may become unstable due to proximity to singularities of various kinds or by growth of round-off errors or small fluctuations in initial data); and respect of *real-time operation* (computation time shorter than the chosen time-step).

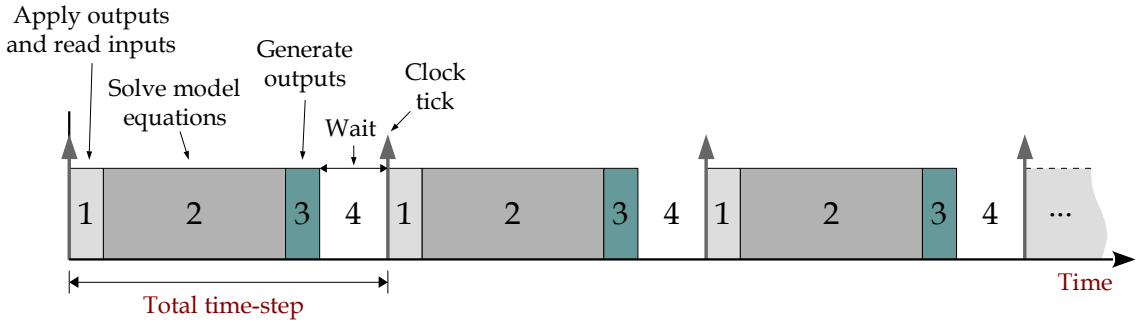


Figure 2.2: Real-time execution tasks

2.4.1 *Numerical solver*

The numerical solvers use a set of integration algorithms that approximate, based on Taylor series, the continuous-time differential equations of the plant. The approximation order has a huge impact, but if only the first-order approximation methods are considered, two categories can be differentiated: *explicit* and *implicit* methods. The explicit methods calculate the *state variables*[3] of the current time-step based only on previous values. Conversely, implicit methods compute the state variables based on current and previous values. More details and quantitative comparisons have been discussed in [25].

2.4.2 *Time-step selection*

The right choice of the real-time simulation time-step is crucial, specially when explicit methods are in use. They might become unstable if the simulation time-step is too short. Implicit methods do not present this problem as far as all the continuous-time real poles are negative [29].

As previously introduced, it is assumed a *discrete-time* and *constant* time step. This means that during a discrete-time simulation, time moves forward in steps of equal duration. This is known as *fixed* time-step simulation [30]. However, other solving techniques exist which use *variable* time-steps. Such techniques are used for solving high frequency dynamics and non-linear systems, but they are not suitable in the context of RTSs because of important implementation issues [31]. In addition, the required time to compute the model at a given time-step must be shorter than the wall-clock duration of the chosen time-step. Therefore, if the simulator operations are not all achieved within the required time-step, the RTS commits an *overrun* and the simulation is considered as non-valid [26].

As shown in Figure 2.2, for each time-step, the simulator must execute the same series of tasks: 1) apply previous iteration outputs and read current iteration inputs, 2) solve model equations, 3) generate outputs and 4) wait for the start of the next iteration. So in order to guarantee a real-time execution, the amount of time needed to perform these operations should be less than the total time-step.

The four most important selection criteria when defining the time-step duration are the following:

3 A *state variable* is one of the set of variables that are used to describe the mathematical "state" of a dynamical system.

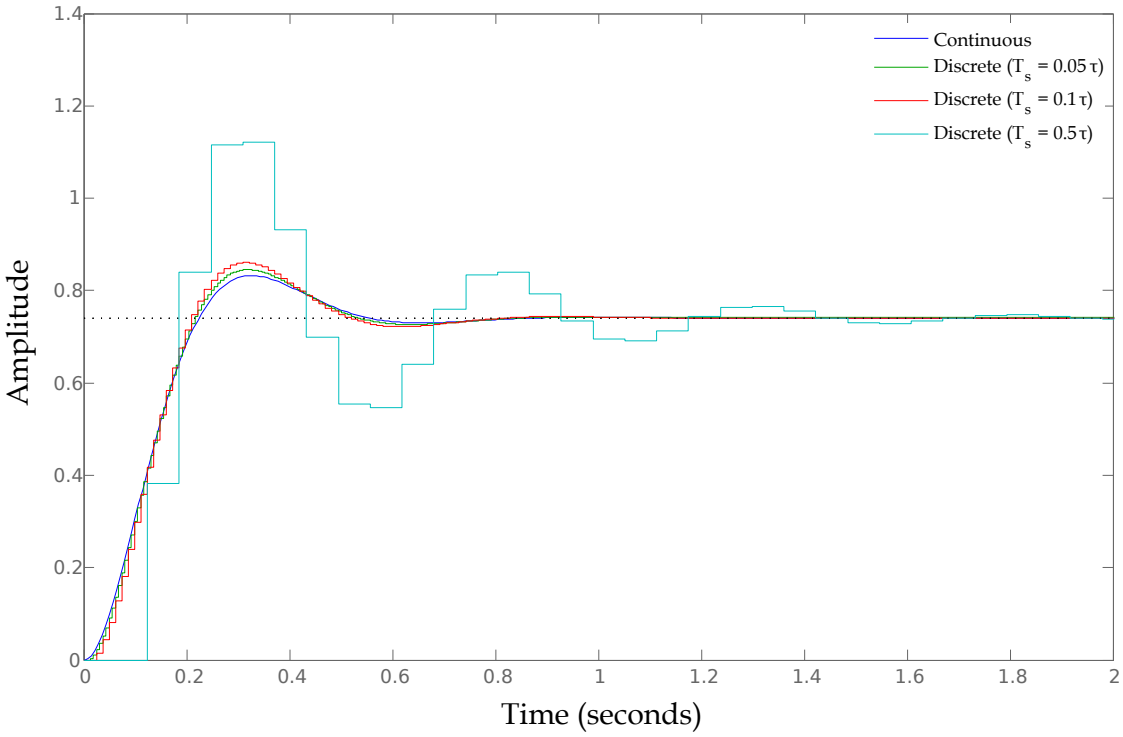


Figure 2.3: Effect of time-step selection when discretizing

- *System dynamics*. According to [26, 32], when an electromagnetic element is of concern, a time-step between 5% to 10% of the smallest time constant has to be chosen. To give an idea, most mechanical systems –which have slow dynamics– require time-steps from $1ms$ to $10ms$, whereas electromagnetic transient simulation with frequency content up to $10kHz$, typically require a simulation time-step around $10\mu s$. Furthermore, if the objective is to accurately simulate fast-switching power electronic devices, the timing requirements can be in the order of $250ns$ without interpolation, or $10\mu s$ if an interpolation technique is utilized [32, 33]. In Figure 2.3 is shown how different time-steps affect the discrete-time system's similarity with the continuous-time version.

- *Interfacing errors*. When considering switching elements which are interfaced with a controller, the power converter RTS can respond to its control signals only at each simulation time-step [34]. As a result, switching events that occur between two steps cannot be detected and therefore errors are inevitably introduced to the simulation results. To avoid this problem, the time-step should be at least 100 times shorter than the switching period [35]. Figure 2.4 shows how the PWM sampled signal fidelity decreases as the sampling period increases. Note that $T_{s_3} = 2T_{s_2} = 4T_{s_1}$.

- *Numerical stability*. Whatever the numerical solver is, even if it is unconditionally stable[4], when the time-step decreases, the poles of the discrete-time model move closer to the stability limit (i.e., $|z| = 1$). Hence, poles become very sensitive to variations caused by limited precision data quantification [36]. These variations may then lead to instability

4 According to [29], A-stable discretization methods are *unconditionally stable* if all continuous-time poles are stable, i.e. they are placed in the negative part of the s-plane. All implicit methods have this characteristic.

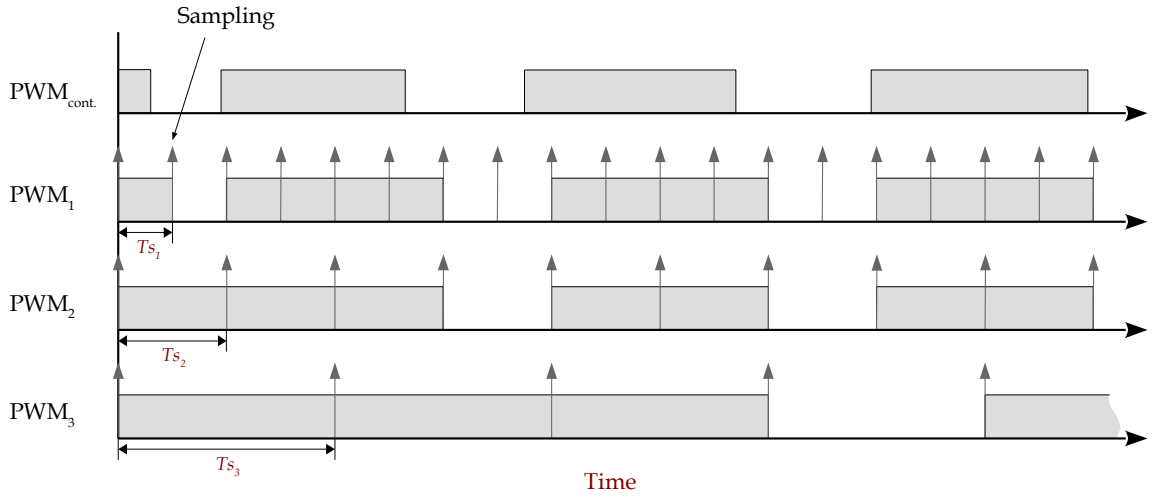


Figure 2.4: Interfacing errors

especially when fixed-point format is used. Hence, care should be taken when choosing the amount of fractional bits and its representation. Moreover, the growth of round-off errors or small fluctuations in initial data could also affect the stability of the system [37].

- *Real-time operation*. Logically, when considering a real-time simulation, the time-step must be short enough to reproduce system's behavior, but at the same time, sufficiently long to allow processing of the model equations (see Figure 2.2). Accordingly, to guarantee real-time operation, the processing time must be shorter than the chosen time-step [26].

2.4.3 *Data representation*

Regarding the numerical data representation, fixed- or floating-point can be adopted. The first uses between 3 to 4 times less hardware and is 3 to 4 times faster [38]. However, the price to pay is accuracy. With floating-point representation the range of the data is much more broader but the price to pay is complexity of floating-point operations and thus, both area utilisation and computation time increases.

2.5 ERTS DEVELOPMENT (III) : DIGITAL IMPLEMENTATION

Once the importance of selecting the right model, the appropriate numerical solver, a suitable time-step, and the right data format has been explained, the final step that influences the performance of a real-time simulation is its digital implementation. The main challenge from the application's point of view is to find an appropriate platform able to handle the complexity of the simulated system and ensure the required timing performances, i.e. providing a simulation time-step small enough to properly simulate the highest system dynamics.

Along this line, many performance digital platforms are available in the market. They integrate powerful multi-core processors where the number of CPUs and other components can be arranged on demand by the manufacturer regarding system's scale, complexity, number of I/Os, timing requirements, cost, etc. The most widely known companies providing commer-

cial real-time simulators are *RTDS Technologies*[5] and *Opal-RT*[6]. The first are more focused on power systems applications whereas the second excel on power electronic based applications.

From the designer's point of view, these platforms are provided with a set of software tools that allow creating the system's model, setting up the simulation, controlling and modifying the system parameters during simulation, acquire data, and analyze its results. Two examples can be *RTDS Simulator* from *RTDS Technologies*, and *HYPERSIM* from *Opal-RT*. Besides these tools, which are specific to each manufacturer, they often provide as well toolboxes that allow using their hardware in traditional simulation tools like *MATLAB/Simulink* from *Mathworks*[7].

As far as digital implementation is concerned, using FPGAs for real-time simulation has been a big trend in the last few years [18, 19, 35, 39, 40]. This approach has many advantages. First, computation time is almost independent of the size of the system because of the parallel nature of FPGAs. Second, overruns cannot occur when the model is implemented in hardware once timing constraints are met during the design process. And finally, the simulation time-step achieved can be in the order of few hundred of nanoseconds. However, there are still limitations on model size since the number of gates and their interconnections are limited.

Hence, to boost timing performances these commercial platforms are also equiped with FPGA-based boards, which have been successfully integrated because of the following reasons:

- *Parallel computational capability*. Works [35, 39, 40, 9] show how these kinds of devices are used in the real-time simulation of electrical systems with ultra-fast transients such as high-frequency IGBT-based power converters. Taking advantage of the inherent parallelism of FPGAs, the computation time is reduced setting small time-steps in the order of $1\mu s$ or less.

- *Additional hardware resources*. These devices include distributed memories, *Digital Signal Processor* (DSP) blocks, high speed I/Os, *Analog-to-Digital Converters* (ADC), *Digital-to-Analog Converters* (DAC), logic blocks, etc., which give the possibility to implement complex models like power converters with large number of switches [41], or using the combination of multiple FPGA boards to simulate large-scale systems [42].

- *New design tools*. The traditional way of programming these kinds of devices was using *Hardware Description Languages* (HDL). The most widely utilised are VHDL, Verilog and SystemC. However, the use of these languages can be seen as a disadvantage since control and software engineers might not be particularly familiar with them. To overcome this issue, two kinds of software tools are available in the market. Most commercial simulation platforms allow today to program the FPGAs by using graphical tools such as *Xilinx System Generator* from *Xilinx*, *DSP Builder* toolbox from *Intel*[8], or the *LabVIEW FPGA Module* from *National Instruments*. The second group are *High-Level Synthesis* (HLS) tools let the designers to work at a higher abstraction level by specifying the behavior of an algorithm using C, C++ or SystemC. Additional information regarding this last set of tools can be found in Section 2.7.

As introduced earlier, these high-performance platforms are most of the time dedicated to HIL testing during the design of digital controllers [18, 19, 16, 17]. However, their cost is not always taken into account. Additionally to the reduction of the development time and the

5 Official website *rtds.com*
6 Official website *opal-rt.com*
7 Official website *mathworks.com*
8 *Intel* recently bought *Altera Corporation*, an American manufacturer of programmable logic devices who released its first *Programmable Logic Device* (PLD) in 1984.

verification of the controller in every situation, the other primary objective of a HIL test is to achieve a high level of correspondence with the real plant. This implies utilizing detailed and complex algorithms running at very short time-steps. The consequence is the use of powerful but costly digital platforms [41, 43]. Nevertheless, The use of such expensive devices is not acceptable in the case of embedded applications where eRTSs are not only developed for HIL testing but also embedded within digital controllers [8, 25, 7, 14].

Consequently, the constraints linked to the real-time simulation discussed in the former Section are strengthened by those of embedded systems such as cost, power consumption, reliability, size, and flexibility. To bring solutions to this new field, SoC devices including FPGA fabric are surely one of the most promising technologies. The following Section tries to prove it by describing these devices and giving an overview of the design opportunities that they offer thanks to their versatile architecture and improved design tools.

2.6 SYSTEM-ON-CHIP DEVICES

A wide and diversified range of powerful platforms are available nowadays in the market. Among them, recent SoC devices are surely the most promising since they bring a new design paradigm thanks to their heterogeneity and computational power. Heterogeneity because they include in the same chip general-purpose processors and an FPGA fabric associated to several peripherals like ADCs or DACs; and computational power thanks to these potent hard-processors plus the many capabilities the PL offers, which can increase massively the computing speed of the whole system.

Considering all these properties, one can easily see that SoC-based eRTS offer much more flexibility compared to the pure hardware FPGA-based eRTS approach studied in [25]. To provide clear examples about it, with this new technology it is possible to go with a *full-software* implementation (everything runs in the processor or processors), a *full-hardware* one (all is computed in the FPGA fabric), or a *hardware-software co-design* where only the heavy calculations are performed in hardware (also known as a *hardware accelerator*).

2.6.1 *General overview*

To cite some of these new SoC devices, the first version of *Microsemi*'s *SmartFusion SoCs* features an *ARM Cortex-M3* processor clocked at $100MHz$, an FPGA fabric, and other programmable analog circuitry [44]. The processor is the actual industry-leading 32-bit processor for highly deterministic real-time applications [45]. Its FPGA part is based on their *ProASIC3* architecture, ranging between 60.000 to 500.000 system gates with up to $350MHz$ system performance, embedded SRAMs and FIFOs and up to 128 FPGA I/Os supporting LVDS, PCI, PCI-X and LVTTL/LVCMOS standards [46]. The programmable analog has high-performance analog signal condition blocks with voltage, current and temperature monitors, 12-bit ADC and DAC converters, up to ten $15ns$ high-speed comparators, and up to 32 analog inputs and 3 analog outputs. They offer as well the *SmartFusion2* (diagram shown in Figure 2.5) with lower power consumption, higher security, and better reliability for safety critical and mission critical systems [47].

Intel –formerly *Altera*– offers a full-range SoC FPGA product portfolio spanning high-end, mid-range, and low-end applications [48]. Their high-end *Stratix 10 SoC* mount the powerful 64-bit quad-core *ARM Cortex-A53* processor [49], whereas the *Arria 10*, *Arria V* and *Cyclone V SoCs* feature a 32-bit dual-core *ARM Cortex-A9* processor [50], a significantly more powerful processor than the one present in *Microsemi*'s products. These multiprocessors come as well

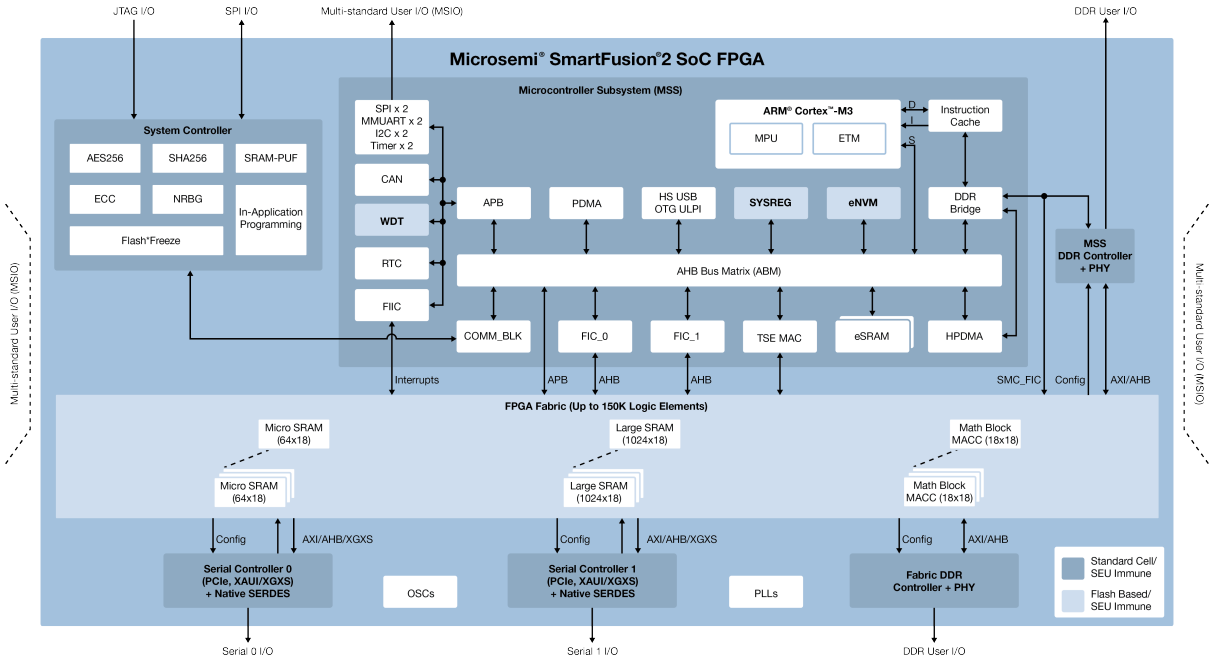


Figure 2.5: Microsemi's *SmartFusion2 SoC FPGA* [credit: *Microsemi Corp.*]

with two embedded hardware accelerators per core, which will be described afterwards: the *NEON Single Instruction, Multiple Data* (SIMD), capable of executing the same mathematical operation on several registers in parallel [51]; and the VFPv3 (in Cortex-A9) and VFPv4 (in Cortex-A53), two *Floating-point Units* (FPU), which can compute operations with very few clock cycles in either 32- and 64-bit precision [52, 53]. These two units, initially developed and broadly used in media processing, can be configured and used to execute intensive operations –like matrix multiplications and matrix inversions– reducing the execution time significantly compared to sequential operations in common processors or DSPs. Moreover, they are composed as well of configurable *L1* caches up to 64*kB* of memory with a system coherency support using the *Accelerator Coherency Port* (ACP), and hardened floating-point DSP blocks capable of processing rates up to 1.5 TFLOPS[9] and power efficiency up to 40 GFLOPS/Watt. However, *Intel*'s SoC does not come with a built-in ADC. The *Cyclone V SoC* diagram is shown in Figure 2.6.

The last example of SoC devices available in the market, which is included in this thesis work, is the *Zynq All Programmable SoC* produced by *Xilinx*, which main diagram is shown in Figure 2.7. They offer a broad portfolio ranging from single-core 32-bit *ARM Cortex-A9* plus an *Artix* FPGA fabric, passing through a double-core version with the FPGA hardware available in both technologies *Artix* and *Kintex*. Recently, they released the *Zynq UltraScale+* featuring dual- and quad-core *ARM Cortex-A53* (64-bit architecture), another version including as well a dual-core *ARM Cortex-R5* deterministic real-time microcontroller, plus the highest-end version which also comes with a *GPU*[10] *Mali-400 MP2* supporting the H.264-H.265 video codec. Wistfully, *Xilinx* has not included in any of the *Zynq UltraScale+* versions the double 12-bit ADC, 1*MSPS*[11] ADCs present in the older *Zynq-7000* devices [54].

9 In computing, *floating-point operations per second* (FLOPS) is a measure of computer performance, useful in fields of scientific computations that require floating-point calculations.

10 A *Graphics Processing Unit* (GPU) is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device.

11 *Millions of Samples Per Second* (MSPS).

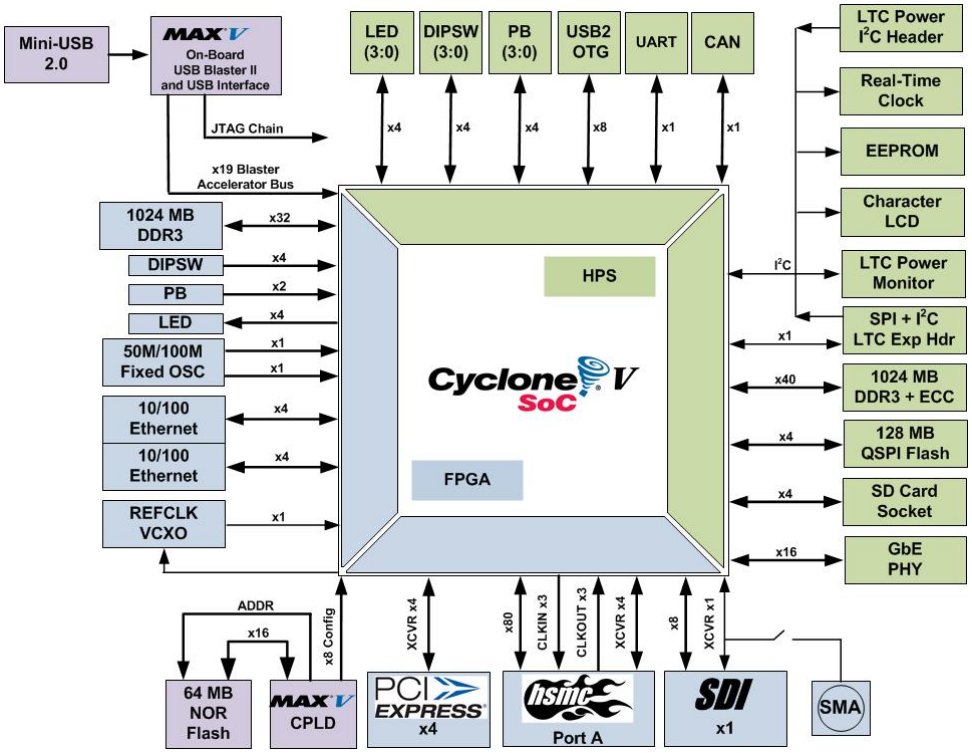


Figure 2.6: Intel's *Cyclone V SoC* [credit: *Intel Corp.*]

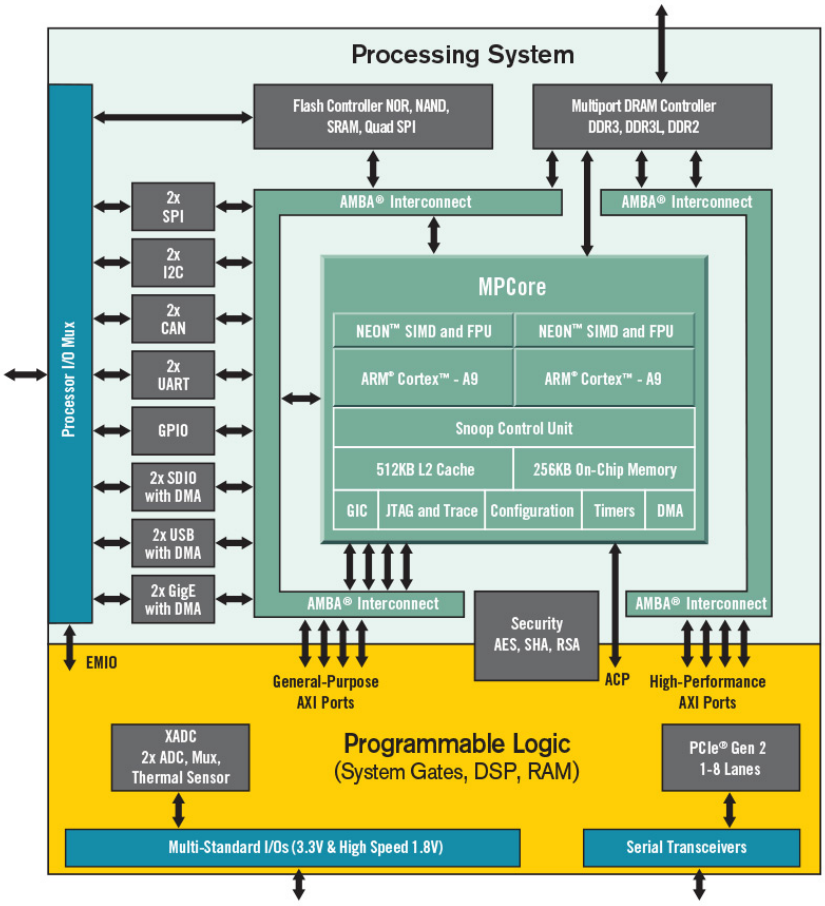


Figure 2.7: Xilinx's *Zynq-7000 All Programmable SoC* [credit: *Xilinx Inc.*]

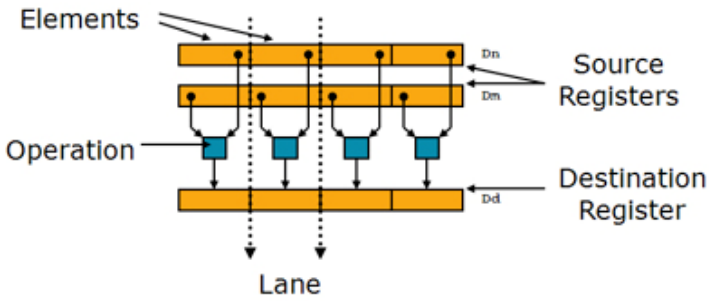


Figure 2.8: NEON operations [credit: *Xilinx Inc.*]

2.6.2 *ARM Cortex-A9 hardware accelerators*

When implementing an RTS, the processing accuracy and execution time are the most critical parameters [8, 55, 22]. Indeed, the choice between using floating- or fixed-point data format is directly related not only with the accuracy of the calculations, but also with the time needed to execute them. As previously introduced, the *ARM Cortex-A9* present in the *Intel* and *Xilinx's* products, have two powerful FPUs [52], one per core, which are capable of computing operations with very few clock cycles either in 32- or 64-bit precision –see Table 2.1. Additionally, these powerful processors come as well with two NEON SIMD units which can execute the same mathematical operation on several registers in parallel [51], decreasing significantly the algorithm execution time as it will be demonstrated later on. These two units can be configured and used to execute intensive operations reducing the execution time significantly.

2.6.2.1 *The NEON SIMD*

NEON technology is a 128-bit SIMD architecture extension of the *ARM Cortex-A9* series processors, designed initially to provide flexible and powerful acceleration for consumer multimedia applications [51]. It has 32 registers, 64-bits wide (dual view as 16 registers, 128-bits wide), which are shared with the FPU.

It can perform *packed SIMD* processing –see Figure 2.8:

- Registers are considered as vectors of elements of the same type
- Data types can be: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, and single precision floating-point
- Instructions perform the same operation in all lanes

2.6.2.2 *The VFPv3 FPU*

This FPU provides hardware support for floating-point operations in half-, single- and double-precision floating-point arithmetic [52]. The FPU fully supports the basic instructions shown in Table 2.1 along with their corresponding latency[12] and throughput[13] for either 32- and 64-bit floating-point formats.

12 Number of clock cycles to process an operation.
13 Number of clock cycles before a new input data value is accepted.

Table 2.1: FPU instruction throughput and latency cycles

UAL	Single-Precision		Double-Precision	
	Throughput	Latency	Throughput	Latency
VADD	1	4	1	4
VSUB				
VMUL	1	5	2	6
VMLA	1	8	2	9
VDIV	10	15	20	25
VSQRT	13	17	28	32

2.6.3 *PS-PL interfacing*

Regarding the interface between PS, PL, and the additional peripherals of the devices presented above, all of them use *ARM*'s *Advanced Microcontroller Bus Architecture* (AMBA) open-standard, on-chip interconnect specification. However, each manufacturer uses a different version. *Microsemi* uses the AMBA 2, a 1999 protocol called *Advanced High-performance Bus* (AHB), a single clock-edge protocol. *Intel* chose the AMBA 3 specification from 2003, which included the *Advanced eXtensible Interface* (AXI3), a high-performance, high-clock frequency system designs that uses separate address/control and data phases, supports unaligned data transfers and burst based transactions with only start address issued. *Xilinx* opted for the AMBA 4 specification released in 2011, which included the AXI4 defining three new different protocols: *AXI4-Master*, *AXI4-Lite*, and *AXI4-Stream*. The *AXI4-Master* is an improved version of the AXI3 protocol, allowing bigger burst transactions and QoS signals among other enhancements. The *AXI4-Lite* is a simpler version of the *AXI4-Master* that uses less hardware resources and is intended to be used mainly for control and configuration purposes. Both are memory mapped protocols, where the read/write transactions contain the destination addresses. Finally, the *AXI4-Stream* is the simplest and fastest interconnection, where there are no addresses involved, uses one single channel and hence the data is traveling only in one direction. In [56] can be found further details about the AMBA AXI protocol.

Among the three devices presented above, *Xilinx's* Zynq include other useful peripherals like ADCs, external *Direct Memory Access* (DMA) controllers, and I/O peripherals and interfaces. All of them are interconnected using the ARM *Advanced Microcontroller Bus Architecture* (AMBA), which in the 4th specification of the protocol –the one implemented in the *Zynq-7000*– include the *AXI4* point-to-point channels for communicating addresses, data, and response transactions between master and slave clients [56]. These interfaces, seen in Figure 2.9, can be divided into three groups:

- Four *AXI High-Performance* slave ports (HP0-HP3):
  - Configurable 32-bit or 64-bit data width
  - Access to OCM and DDR only
  - *AXI FIFO Interface* (AFI) are 1kb FIFOs to smooth large data transfers
  - Automatic conversion and synchronization to processing system clock domain
- Four *AXI General-Purpose* ports:
  - Two masters from PS to PL (GP0 and GP1)

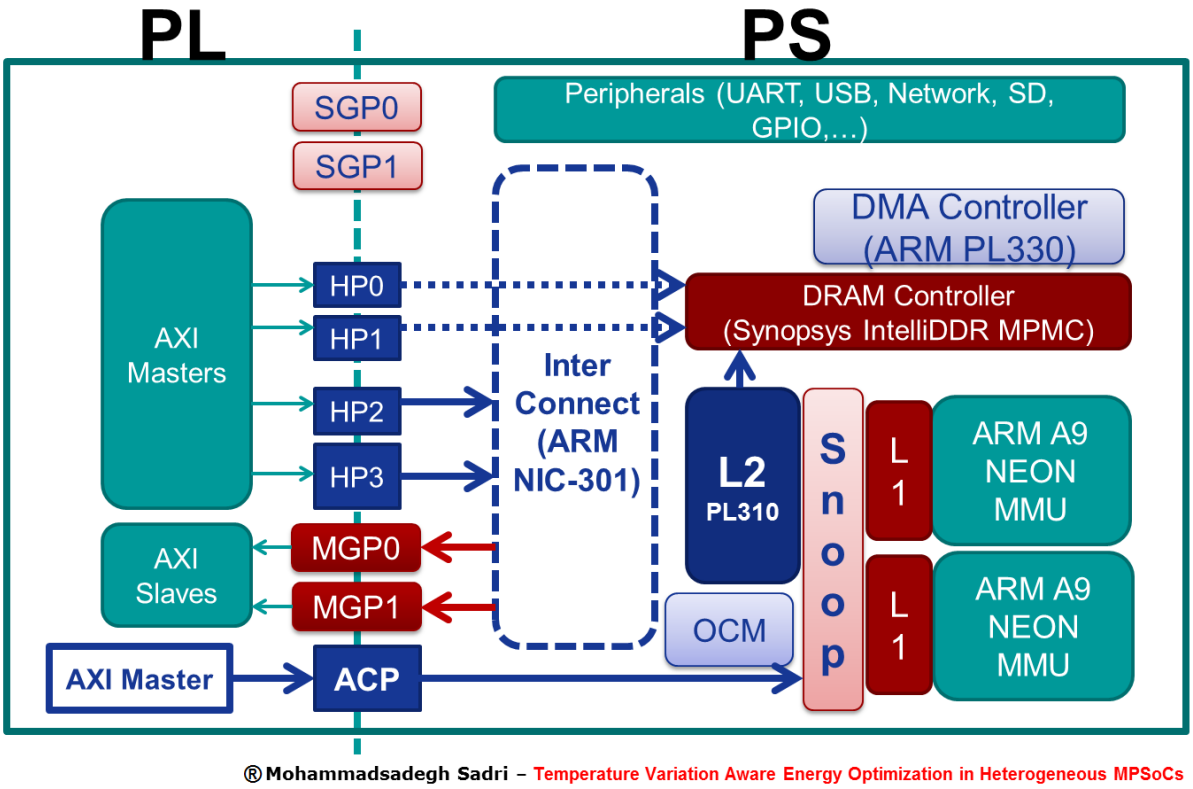


Figure 2.9: *Zynq* interfaces [credit: *Mohammadsadegh Sadri*]

  - Two slaves from PL to PS (SGP0 and SGP1)
  - 32-bit data width
  - Automatic conversion and synchronization to processing system clock domain

- One 64-bit *Accelerator Coherency Port* (ACP):
  - *AXI* slave interface performing OCM and DDR coherent accesses

All the system components can be accessed using physical addresses except for the CPU cores and their *L1* instruction caches. The memory map of the *Zynq-7000* defined in [57] indicates the address range of each logic block.

2.7 DESIGN TOOLS AND METHODOLOGY

In addition to this technological evolution, the design tools have also been subject of important progress. The design issues of these powerful platforms are cumbersome. Hence, manufacturers have invested notable efforts in making the development cycle shorter for a faster time-to-market, relieving programming efforts by automating the configuration of the device, and increasing the level of abstraction easing the utilisation of the FPGA fabric and peripherals.

Some of these tools let the system designer to graphically interconnect the different IPs present in the FPGA fabric with the PS and other subsystems, allowing to configure them using *Graphical User Interfaces* (GUI) instead of setting up control and configuration registers one by one. Once the design process has finished, these tools *synthesize*[14] the design, perform

14 In electronics, *logic synthesis* is a process by which an abstract form of desired circuit behavior, typically at *Register Transfer Level* (RTL), is turned into a design implementation in terms of logic gates, typically by a computer

the *place and route*[15], and generate a *bitstream*[16] containing the whole hardware code and device configuration. Every manufacturer has their own set of tools that allow users to program their devices, but there exist as well third-party products like *MATLAB/Simulink* and its *HDL Coder* for example. It can be a good choice if the designer is already familiar with it and wants to develop a not very demanding application using SoC devices. To cite other notable companies that produce *Electronic Design Automation* (EDA) tools: *Synopsys*, *Cadence Design Systems*, or *Mentor Graphics* are among the most important.

Similarly, *High-Level Synthesis* (HLS) tools ease the utilisation of SoCs –and FPGAs too– to software designers, allowing them to use the benefits of hardware acceleration as processing speed and energy saving, without the mandatory need of building up extensive hardware expertise. Some examples of these programs are *Catapult HLS* from *Mentor Graphics*, *VivadoHLS* from *Xilinx*, *Bambu* developed at *Politecnico di Milano*, or *Kiwi* from the *University of Cambridge* [60]. They let designers work at a higher abstraction level by specifying the behavior of an algorithm using C, C++ or SystemC rather than inferring its hardware definition using HDLs such as VHDL or Verilog. Indeed, these software tools allow to express the algorithm in a behavioral way instead of using tedious *Register-Transfer Level*[17] (RTL) descriptions [62], reducing considerably the development time.

All in all, these software tools have reached a significant grade of maturity in recent times, besides, the size and price of FPGAs improve year after year. Therefore, the use of such mature platforms have an added value for implementing eRTSs of electrical systems since they have proven their efficiency in many applications such as medical imaging [63], convolutional neural networks [64], image processing [65], encryption algorithms [66], big data processing [67], computer vision [68], or machine learning [69] with successful improvements in energy consumption and performance. Notwithstanding, the hardware code these tools produce is still far from what can be achieved manually using HDL languages in terms of performance and FPGA resources [10]. Nevertheless, when the complexity of the applications increase, the use of these tools becomes convenient if not mandatory because it impacts considerably in the development time and cost in man-hours.

Furthermore, some of these HLS platforms allow the use of *test bench*[18] files coded as well using C, C++ or SystemC. It eases significantly the debugging and verification of the proper functioning of the whole algorithm, avoiding the necessity of creating complicated HDL test bench files commonly used in hardware designs. Furthermore, these tools also admit the use of *pragmas*, *directives* and/or *constraints* to infer different hardware configurations from the same source code in order to compel the design meeting the required specifications [70]. As an illustration example, in [6, 7, 14] authors evaluate different word-sized implementations by

program called a *synthesis tool*. Common examples of this process include synthesis of HDLs, including VHDL and Verilog. Some synthesis tools generate bitstreams for programmable logic devices such as PALs or FPGAs, while others target the creation of ASICs. Logic synthesis is one aspect of *Electronic Design Automation* (EDA) [58].

15 *Place and route* is a stage in the design of printed circuit boards, integrated circuits, and field-programmable gate arrays. As implied by the name, it is composed of two steps, placement and routing. The first step, placement, involves deciding where to place all electronic components, circuitry, and logic elements in a generally limited amount of space. This is followed by routing, which decides the exact design of all the wires needed to connect the placed components. This step must implement all the desired connections while following the rules and limitations of the manufacturing process [59].

16 Sequence of bits carrying up the configuration data to be loaded into the FPGA through a serial bit stream.

17 In digital circuit design it refers to a design abstraction which models a synchronous digital circuit in terms of flow of digital signals (data) between hardware registers, and the logical operations performed on those signals [61].

18 *Test bench* files are pieces of code that are used during FPGA or ASIC simulation. This kind of simulation allows the programmer to evaluate the FPGA or ASIC design and ensure that it does what is expected to. A test bench provides the stimulus that drives the IP block and then collects the output results to verify their correctness.

changing the data types to be inferred by the tool simply by changing the *typedef*[19] alias used in the whole design, whereas modifying the RTL code would require days or even weeks to be accomplished. A good study of different HLS tools available in academia and the market can be found in [60].

2.8 CHAPTER CONCLUSIONS

The objective of this chapter was to give an overview of the challenges linked to the development of eRTSs of electrical systems. After introducing what these systems are and what they can be used for, all the relevant constraints linked to the modeling of the plant, its digital realization, and its digital implementation were profusely exposed.

Since it is at the central point of this dissertation, recent SoC devices including FPGA fabric were discussed by describing device architectures and features and their associated design tools. Regarding the latter, HLS tools were depicted as they will become an important pillar in this thesis work for programming the device PL.

According to the previously exposed, the *Zynq-7000 All Programmable SoC* has been chosen because of its powerful dual-core *ARM Cortex-A9* processor, its versatile programmable logic, and the simultaneous dual-channel $1MSPS$ ADC. Furthermore, this choice has been motivated by the complete *Vivado Design Suite* that allows managing all the device configuration and block interconnections graphically, the *Vivado HLS* tool which converts C/C++/SystemC into RTL to directly program the FPGA fabric, and also because of the large and active community behind it.

The following chapters are dedicated to the evaluation of SoC devices to implement eRTSs of electrical systems.

19 The *typedef* keyword allows the C/C++ programmer to create new names for types such as *int* (integer), *float* (single-precision floating-point) or *double* (double-precision floating-point). *Typedefs* can be used both to provide more clarity to your code and to make it easier to make changes to the underlying data types that you use.

Part II

CASE APPLICATIONS

3

ERTS FOR ELECTROMECHANICAL SYSTEMS: THE DFIG CASE

3.1 INTRODUCTION

The objective of this chapter is to evaluate the capabilities of the *Zynq-7000* through the development of an eRTS of a DFIG. The choice of this system was motivated because it is a quite popular machine utilized in wind generators and it has a representative complexity regarding electromechanical systems.

The study explores various implementations that will give a clear picture of how this SoC can be used for performing advanced control strategies using eRTS. These are the different implementations that will be carried out:

- Two numerical solvers with contrasted computational load will be analyzed in order to find which is more appropriate for developing eRTSs:
  - Euler's Forward discretization
  - the bilinear or Tustin discretization
- Once a discretization method has been chosen, three implementation approaches will be carried out:
  - a *full-software* ARM-based implementation
  - a *full-hardware* one using solely the FPGA fabric
  - a *hardware-software co-design*, also known as *hardware accelerator*
- And these proposed solutions will be realized using three different data formats, depending on where they will be implemented:
  - 64-bit floating-point (ARM only)
  - 32-bit floating-point (ARM and FPGA)
  - 32-bit fixed-point (FPGA only)

The aim is then to evaluate and compare the device performance in terms of execution time, resources utilisation and precision of the calculations. Notwithstanding, this approach can be extrapolated to any other application involving eRTS, HIL and modern control techniques which require performing extensive computations on SoC devices. Papers [6] and [7] were issued out of this work.

3.2 THE METHODOLOGY

The methodology followed for this evaluation can be briefly explained as follows. First, the equations of the machine will be obtained using the *dq* reference frame [71]. Second, an

evaluation of two discretization methods will be performed. Then, according to results, the selected discretized model will be implemented using various approaches and a conscientious examination of the different solutions will be made.

As introduced previously, when implementing an embedded RTS in a heterogeneous system like the *Zynq*, and considering that the DFIG model is coded in C/C++, one can think of different approaches:

- A *full-software* implementation using solely the ARM processor. The mathematical model of the machine is a function that is called from within the main program. Even though the *ARM Cortex-A9* is a 32-bit processor, it has a FPU capable of executing operations in 64-bit efficiently. Hence, it is worth studying the differences when using 32-bit and 64-bit variables in terms of accuracy and execution time. This option is evaluated in Section 3.5

- A *full-hardware* implementation using Vivado HLS to program the FPGA fabric using the same C/C++ model coded for the software version. In this case, two different implementations are proposed for a further comparison with the software version: a 32-bit floating-point version and a 32-bit fixed-point version. This implementation is covered in Section 3.6

- A *hardware-software co-design*. If a whole control system is considered, some subsystems can run in different parts of the *Zynq* trying to alleviate some PS computational load transferring it to the PL. These are known as *hardware accelerators* and for some applications they can help reducing the total algorithm execution time. When this option is going to be implemented, a shared memory space has to be used in order to send data to and receive it from the PL. This approach is covered in Section 3.7 and evaluates two different implementations, one using BRAM and the other using the *On-Chip Memory* (OCM)

All these different versions will be compared in terms of accuracy of the results with a *Simulink* off-line implementation of the continuous model equations, setting a $100ns$ fixed time-step, using the *Runge-Kutta ODE4* solver and double-precision floating-point variables. This solver was chosen to produce the reference values because of its high accuracy. The *Simulink* model can be seen in Figure 3.1. The input signals to the block are in the *abc* frame but they are converted to the *dq* frame before performing the model computations. In the next Section, Figure 3.8 shows the block diagram continuous equations of the model based in the *dq* reference frame.

The model will be tested in two different scenarios: (i) one changing the speed of the shaft $\pm 30\%$ around the synchronous value; and (ii) another modifying the load to pass from consuming 50% to 85% of the nominal power at constant speed. The variation of the input signals on each of these experiments can be seen in Figures 3.2 and 3.3, along with the speed of the shaft –which is an internal variable. Currents and voltages are shown in *per-unit system*[1] whereas angular speed and torque are in $^{rad}/_{s}$ and Nm respectively.

1 A *per-unit system* is the expression of system quantities as fractions of a defined base unit quantity. Calculations are simplified because quantities expressed as *per-unit* do not change when they are referred from one side of a transformer to the other. This can be a pronounced advantage in power system analysis where large numbers of transformers may be encountered. Moreover, similar types of apparatus will have the impedances lying within a narrow numerical range when expressed as a *per-unit* fraction of the equipment rating, even if the unit size varies widely. The main idea of a *per-unit* system is to absorb large difference in absolute values into base relationships. Thus, representations of elements in the system with *per-unit* values become more uniform. More information can be found in [72].

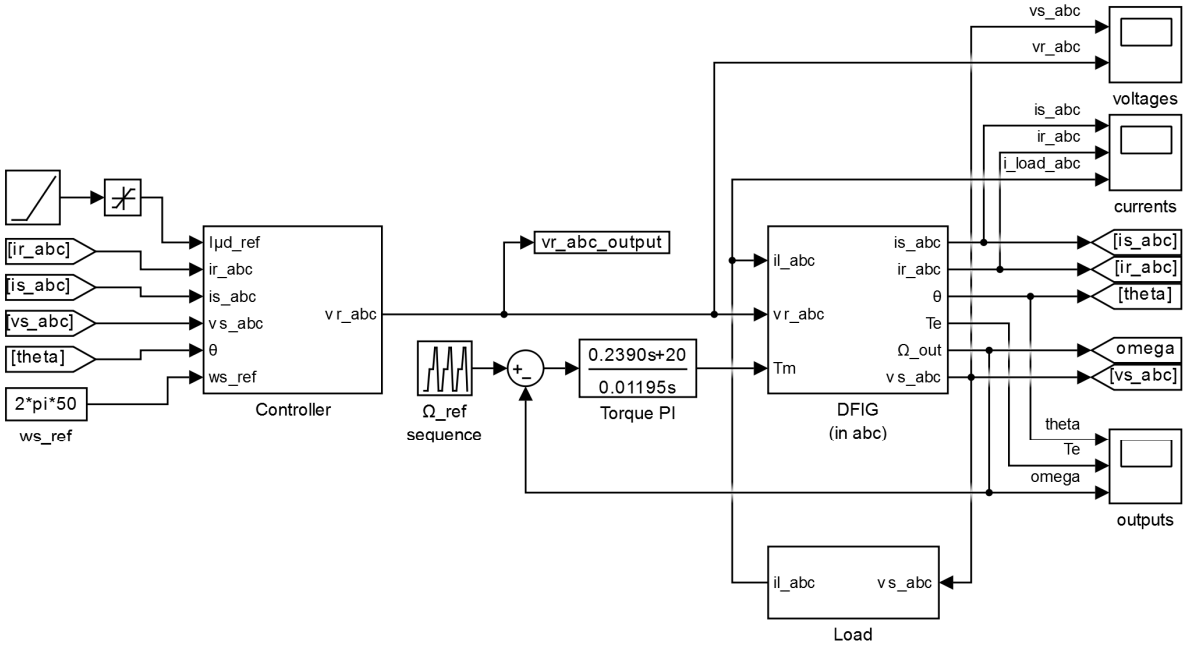


Figure 3.1: DFIG as a standalone generator implemented in *Simulink*

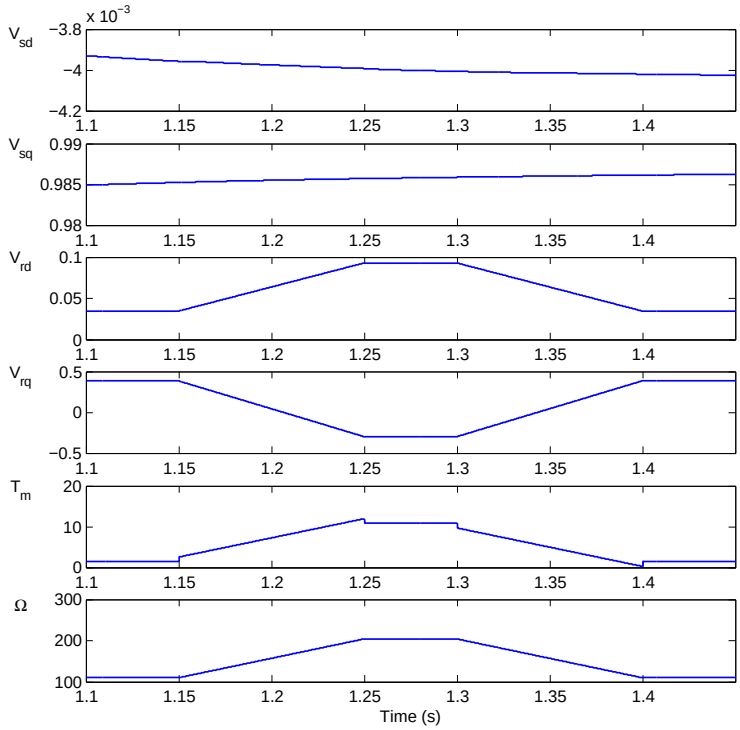


Figure 3.2: Speed change test inputs

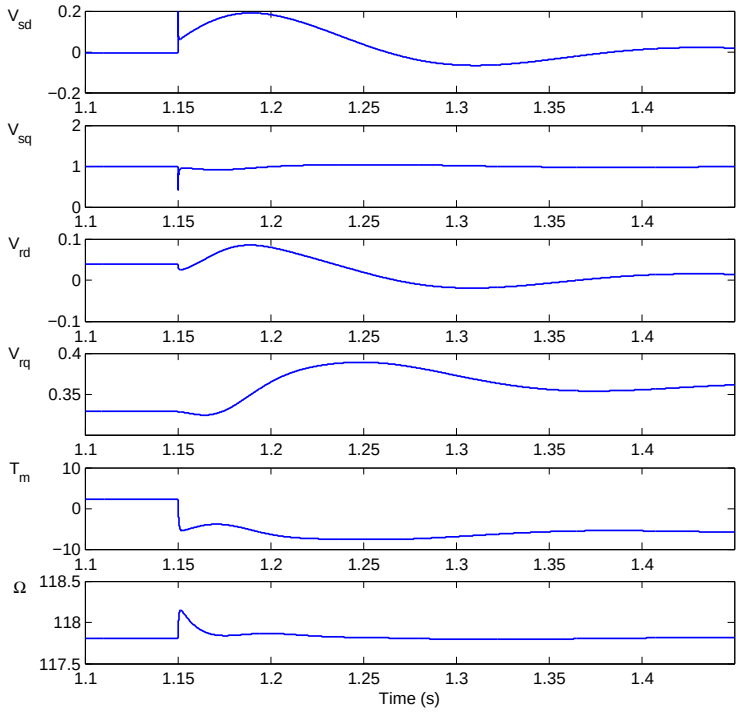


Figure 3.3: Load change test inputs

It is recommendable to set the time-step of the reference data equal to that of the shortest implementation in order to have iteration results produced simultaneously. Moreover, it would be very appropriate for the other implementations to have a time-step multiple of that same value so as to have results synchronized with them as well. This is illustrated in Figure 3.4, where it can be seen the continuous signal from a real system, the fastest implementation being executed at $100ns$ –same as the golden data, a second one computed at $200ns$, and the slowest one at $400ns$, all of them used in the *Zero-Order Hold*[2] (ZOH) context. In this way, every 4th iteration, all implementations will generate a result, thus allowing to properly compare the results with all the other solvers.

The error when comparing the golden data with the *Zynq* implementations will be calculated using equation 3.1.

$$e_{avg} = \frac{\Sigma \left| x_{ref} - x_{imp} \right|}{N} \tag{3.1}$$

where x_{ref} is the reference signal obtained in *Simulink*, x_{imp} is the actual implementation result, and N the total number of samples.

Regarding the *full-software* implementation, the discretized models will be coded in a function using C/C++. When being executed by the ARM, this function will be called from the main routine and the results logged for their latter verification. The execution time needed to compute each model function will be measured as well in order to set the corresponding sampling time parameter in the discretized model equations, configure the software interrupt to ensure synchronized calls to the function, and to then compare the amount of time needed to perform the calculations with the other implementations.

2 The ZOH is the simplest mathematical model to reconstruct a continuous signal. It basically consists in generating an analog value and holding it for one sample interval. It is also the method used by the analog-to-digital converters (ADC), where the value of the continuous-time signal is measured and the value kept until the next sampling.

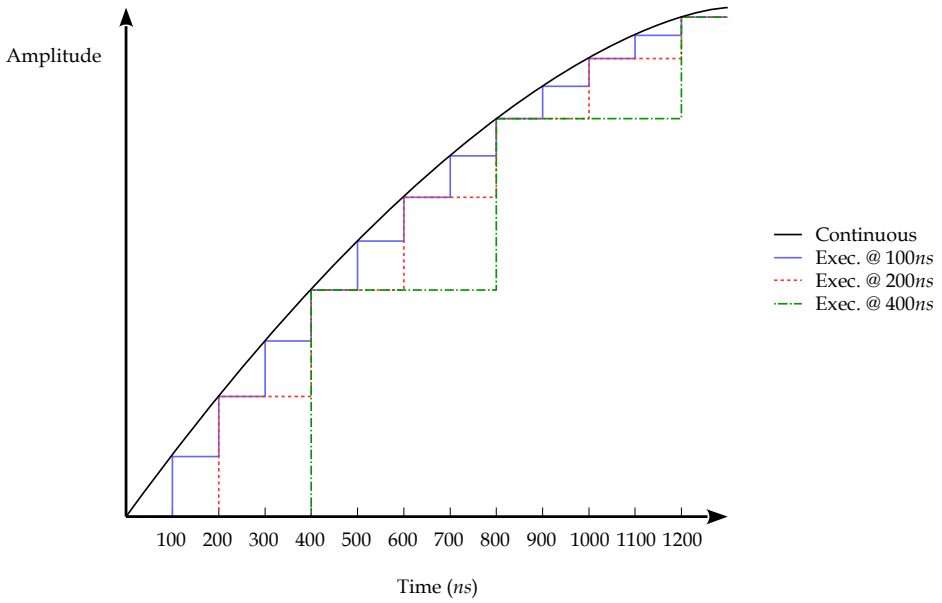


Figure 3.4: Different time-step synchronization

Conversely, the common way of programming the FPGA fabric is using *Hardware Description Languages* (HDL). The most widely used HDLs are VHDL, Verilog and SystemC. Thereafter, this code is *synthesized* into *Register Transfer Level* (RTL), a type of architecture specific for digital logic circuits based on gates, registers and other more manufacturer-specific hardware. It is also important to point out the important amount of time needed to code and debug the increasingly sophisticated algorithms used nowadays with them.

Due to these drawbacks and because of the fact that not all control engineers are familiarized with these languages and platforms, an interesting alternative is to use *High-Level Synthesis* (HLS) tools like *Vivado HLS* provided by *Xilinx* [73]. This software converts algorithmic description written in C, C++ or SystemC into RTL. From the same source code, it can generate code which is device specific and therefore, this source code can be re-utilized when targeting other platforms. It is supplied with extensive libraries ranging from arbitrary precision data types, to data streaming, math, linear algebra, or data signal processing among others. Furthermore, it is possible to use *pragmas*, *directives* and/or *constraints* to infer different hardware architectures from the same source code in order to obtain an appropriate solution for the specific implementation regarding latency, initiation interval and resources utilization [70]. This tool is therefore very powerful for conducting an in depth exploration of the architectural space.

3.3 CASE APPLICATION DESCRIPTION: THE DFIG

The chosen application is a DFIG working as an islanded generator feeding a three-phase RL load [4]. This asynchronous generator is widely used in wind-generation applications because it allows operating with variable speed of the shaft but maintaining a fixed voltage and frequency in the stator terminals [3, 4, 74, 5]. This is accomplished by controlling the rotor voltage using power converters. This feature gives the machine an advantage compared to others regarding the high-power generators, because for a varying speed of ±30% around the synchronous speed, the power converters are considered to be around one-third of the nominal output power of the wind turbine [3, 4]. This adds a surplus to this kind of generators

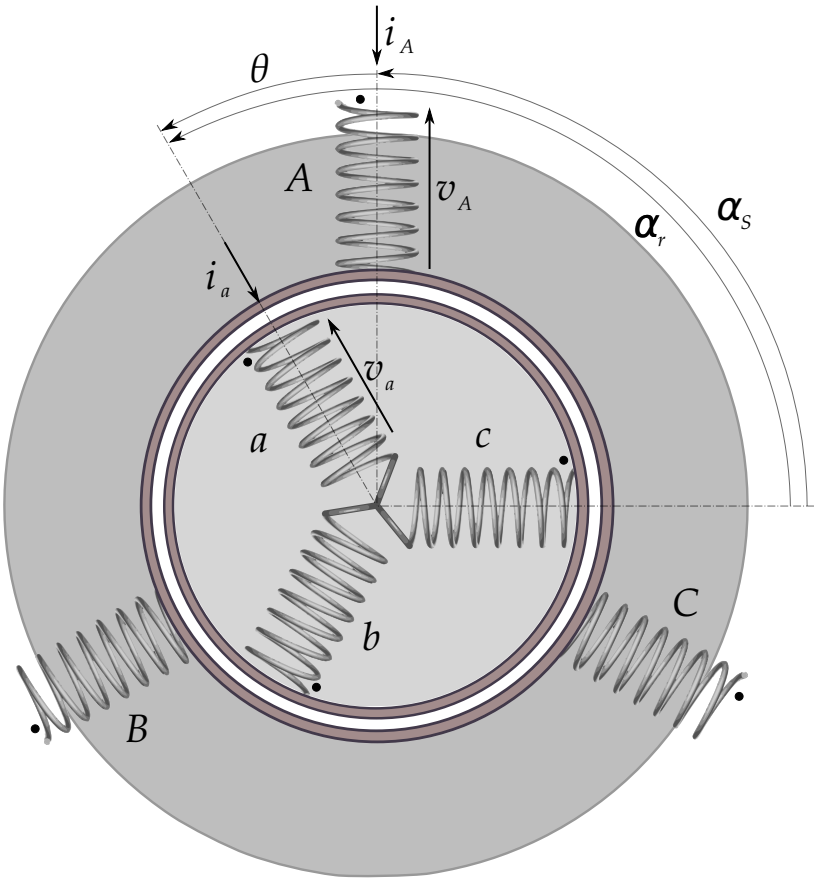


Figure 3.5: DFIG diagram

due to the reduction in size and losses of the converter. Moreover, this machine can produce and consume reactive power avoiding the use of batteries and capacitors connected to the stator. The main drawback of the DFIG are the brushes and slip rings which make the machine bulkier than the one with squirrel cage rotor, less reliable and suitable of constant maintenance. Figure 3.5 shows the electrical diagram of the machine, having the stator quantities in uppercase letters whereas their rotor analogues in lowercase.

A controller has been also included to guarantee constant frequency and voltage despite of load and shaft's speed variations [5]. Hence, the considered system can be divided into three different parts: the load, the DFIG, and the controller. In order to optimize the use of the *Zynq*, the best distribution would be to use one ARM core to implement the controller, the other core for the DFIG model, and use the FPGA fabric to test different loads and power converters, which require very short simulation time-steps for adequate real-time simulation of the power electronic switching components. A schematic of the actual system under study is shown in Figure 3.6. It is composed of a DFIG, a power converter, a three-phase RL load and the controller. However, in this study only the DFIG model will be evaluated.

3.3.1 *DFIG dynamic equations in the dq reference frame*

Some assumptions need to be made in order to simplify the machine dynamic equations:

- Saturation is neglected since the generator is operating in the linear region. This means that self and mutual inductances are independent of the winding current level

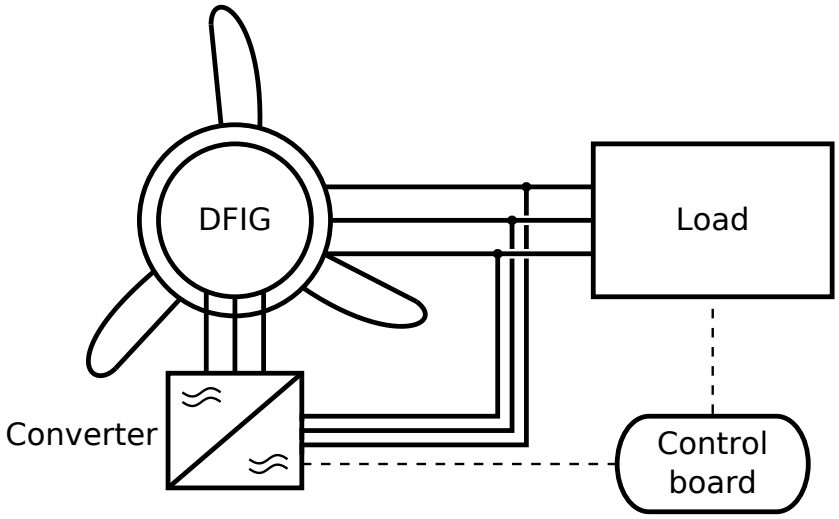


Figure 3.6: DFIG connected to an isolated load

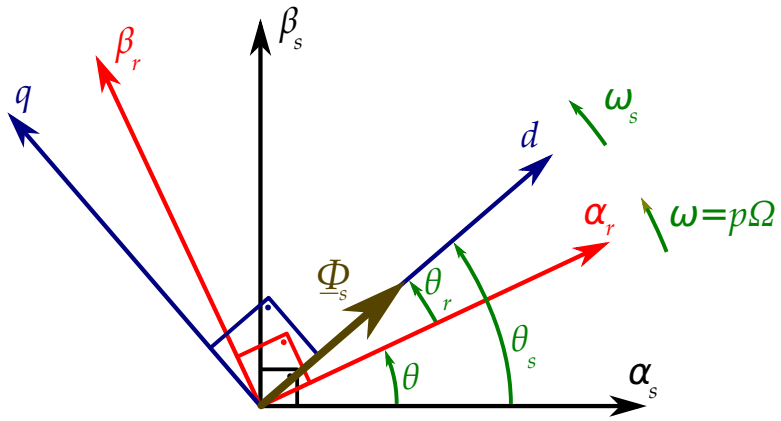


Figure 3.7: dq reference frame

- A smooth, uniform air-gap is assumed, so rotor and stator slotting is neglected
- The hysteresis and Foucault losses are also neglected
- The temperature effect in the windings is not considered

According to these simplifications and to the convention shown in Figure 3.7, the dynamic behavior of a DFIG can be described by equations 3.2.

$$\begin{aligned} \underline{v}_s &= R_s \underline{i}_s + \frac{d}{dt}\underline{\varphi}_s + j\omega_s \underline{\varphi}_s \\ \underline{v}_r &= R_r \underline{i}_r + \frac{d}{dt}\underline{\varphi}_r + j\omega_r \underline{\varphi}_r \\ \underline{\varphi}_s &= L_s \underline{i}_s + M_{sr} \underline{i}_r \\ \underline{\varphi}_r &= M_{sr} \underline{i}_s + L_r \underline{i}_r \end{aligned} \tag{3.2}$$

where the s and r subscript mean stator and rotor quantities respectively, φ is the flux, L is the self inductance, M is the mutual inductance between rotor and stator windings, and ω is the angular speed of the electrical field which is related to the mechanical angular speed of the rotor Ω by equation (3.3), where p is the number of pairs of poles.

$$\omega_r = \omega_s - p\Omega \tag{3.3}$$

For the equations, it will be adopted the convention in the dq reference frame [71]. Substituting, operating, and arranging the equations in a matrix representation, 3.4 is obtained.

$$\begin{bmatrix} \frac{d}{dt}i_{sd} \\ \frac{d}{dt}i_{sq} \\ \frac{d}{dt}i_{rd} \\ \frac{d}{dt}i_{rq} \end{bmatrix} = \frac{1}{\sigma}\mathrm{A}\begin{bmatrix} i_{sd} \\ i_{sq} \\ i_{rd} \\ i_{rq} \end{bmatrix} + \frac{1}{\sigma}\mathrm{B}\begin{bmatrix} v_{sd} \\ v_{sq} \\ v_{rd} \\ v_{rq} \end{bmatrix} \tag{3.4}$$

A, B and σ being:

$$\mathrm{A} = \begin{bmatrix} -\frac{R_s}{L_s} & \omega_s - \omega_r\frac{M_{sr}^2}{L_sL_r} & \frac{M_{sr}R_r}{L_sL_r} & p\Omega\frac{M_{sr}}{L_s} \\ \omega_r\frac{M_{sr}^2}{L_sL_r} - \omega_s & -\frac{R_s}{L_s} & -p\Omega\frac{M_{sr}}{L_s} & \frac{M_{sr}R_r}{L_sL_r} \\ \frac{M_{sr}R_s}{L_sL_r} & -p\Omega\frac{M_{sr}}{L_r} & -\frac{R_r}{L_r} & \omega_r - \omega_s\frac{M_{sr}^2}{L_sL_r} \\ p\Omega\frac{M_{sr}}{L_r} & \frac{M_{sr}R_s}{L_sL_r} & \omega_s\frac{M_{sr}^2}{L_sL_r} - \omega_r & -\frac{R_r}{L_r} \end{bmatrix}$$

$$\mathrm{B} = \begin{bmatrix} \frac{1}{L_s} & 0 & -\frac{M_{sr}}{L_sL_r} & 0 \\ 0 & \frac{1}{L_s} & 0 & -\frac{M_{sr}}{L_sL_r} \\ -\frac{M_{sr}}{L_sL_r} & 0 & \frac{1}{L_r} & 0 \\ 0 & -\frac{M_{sr}}{L_sL_r} & 0 & \frac{1}{L_r} \end{bmatrix}$$

$$\sigma = \frac{L_rL_s - M_{sr}^2}{L_rL_s}$$

Regarding the mechanical part of the DFIG model, equation 3.5 describes its behavior.

$$J\frac{d}{dt}\Omega = T_m - T_e - B\Omega \tag{3.5}$$

considering the angular speed of the rotor Ω, J the inertia, B the friction coefficient, T_m the load mechanical torque, and T_e the electromagnetic torque of the machine. Rearranging this formula in matrix form:

$$\begin{bmatrix} \frac{d}{dt}\Omega \end{bmatrix} = \begin{bmatrix} -\frac{B}{J} \end{bmatrix}[\Omega] + \begin{bmatrix} \frac{1}{J} & -\frac{1}{J} \end{bmatrix}\begin{bmatrix} T_m \\ T_e \end{bmatrix} \tag{3.6}$$

T_m is the input torque of the mechanical load and T_e is calculated using equations (3.4) and (3.7).

$$T_e = M_{sr}\left(i_{rd}i_{sq} - i_{rq}i_{sd}\right) \tag{3.7}$$

These equations were implemented in *Simulink* using continuous blocks as seen in Figure 3.8.

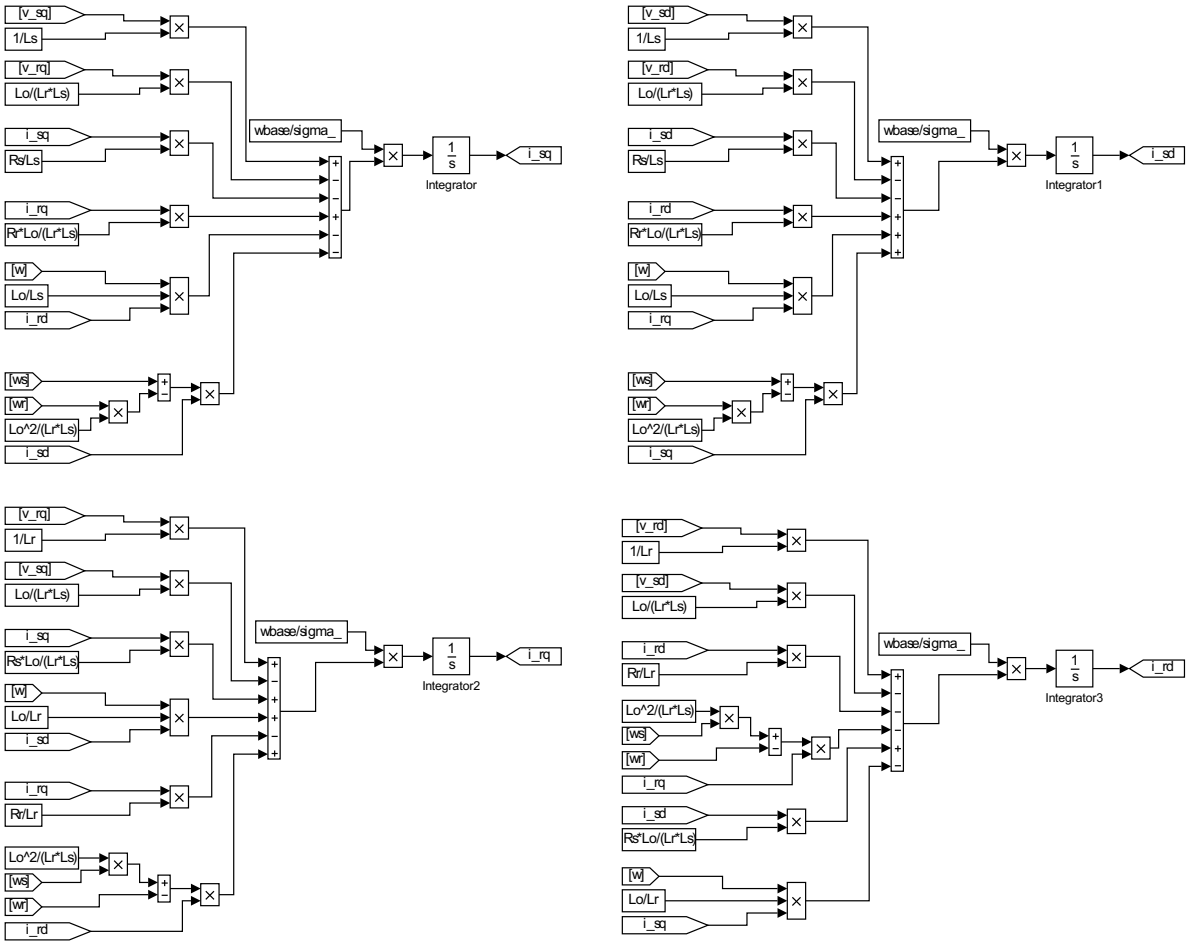


Figure 3.8: DFIG continuous model implementation in *Simulink*

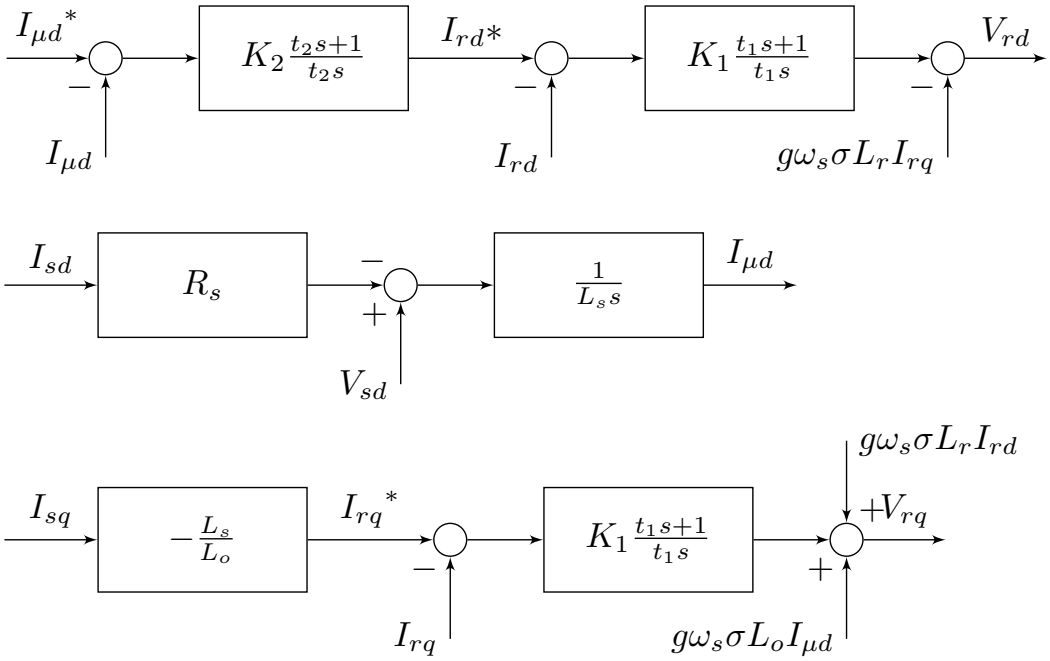


Figure 3.9: Control diagram

3.3.2 *The controller*

An indirect decoupled vector control using stator flux oriented techniques is adopted to adjust the DFIG stator output voltages [5]. Its magnitude is indirectly controlled through the regulation of the stator flux. Unlike the utility grid case, in standalone applications the stator flux is no longer imposed by the grid and can be adjusted by action on the *d* component of the rotor current. Among the main advantages of this indirect vector approach are the low harmonic distortion and the sinusoidal grid voltage even in the presence of non-linear loads [5, 3]. The block diagram of the controller is shown in Figure 3.9.

The proper functioning of the controller shown in Figure 3.10, was verified using *MATLAB/Simulink*, where the voltages and currents in the RL load are maintained while the speed of the shaft varies ±30% around the nominal value ($\frac{2\pi f}{p} = 157,1^{rad/s}$).

3.4 DISCRETIZATION METHODS COMPARISON

In order to simulate the dynamics of a continuous system in a digital environment, a discrete model of the system must be calculated. All systems subjected to a simulation in state-space representation [75] are written as:

$$\dot{x}(t) = Ax(t) + Bu(t)$$

The *Laplace* transform[76] applied to the system equation leads to:

$$sx(s) = Ax(s) + Bu(s)$$

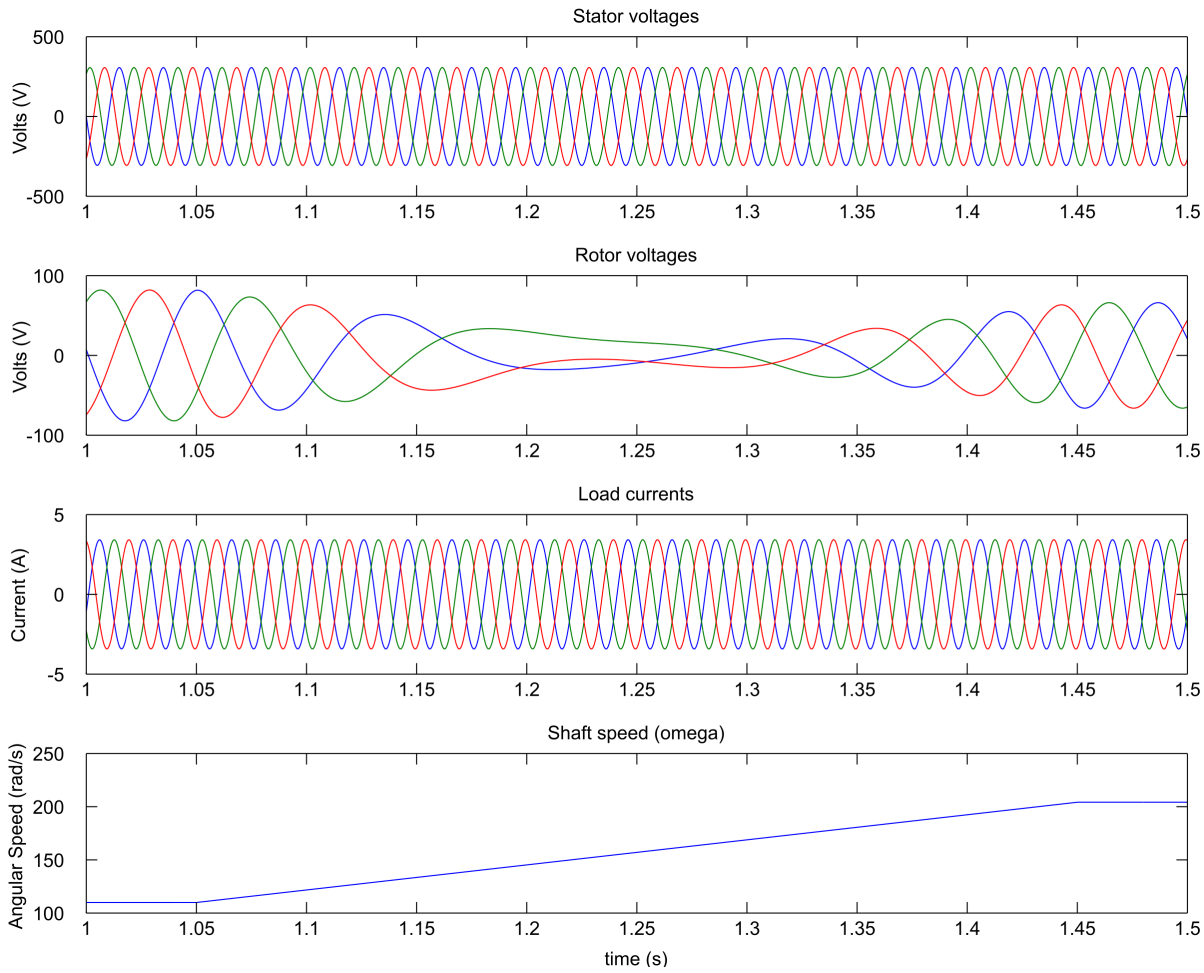


Figure 3.10: System signals

3.4.1 *Euler method*

The first and foremost basic numerical integration approximation technique is the Euler's Forward discretization used in the ZOH context. It is based on replacing the *Laplace* operator s with $(z-1)/T_s$, where z refers to the z-transform operator and T_s is the numerical integration time-step. Accordingly, the discretization using this method leads to 3.8.

$$x_k = (\mathrm{I} + T_s\mathrm{A})\, x_{k-1} + T_s\mathrm{B}u_{k-1} \tag{3.8}$$

3.4.2 *Tustin method*

The bilinear or Tustin discretization is an integration method that is used mainly in the *First-Order Hold*[3] (FOH) context. It assumes that the input vector u varies linearly between kT_s and $(k+1)\,T_s$. Consequently, and substituting the s of the *Laplace* transformation by $2\,(z-1)/T_s\,(z+1)$ equation 3.9 is obtained.

$$x_k = \mathrm{M}\left[\mathrm{I} + \frac{T_s}{2}\mathrm{A}\right] x_{k-1} + \mathrm{MB}T_s\,(u_{k-1} + u_k) \tag{3.9}$$

where $\mathrm{M} = \left[\mathrm{I} - \frac{T_s}{2}\mathrm{A}\right]^{-1}$.

In this case, like A is changing, $\left[\mathrm{I} - \frac{T_s}{2}\mathrm{A}\right]^{-1}$ and $\left[\mathrm{I} + \frac{T_s}{2}\mathrm{A}\right]$ need to be calculated at every time-step which increases significantly the computational load. However, the use of this method will make sense if the error compared to the Euler implementation (using a smaller time-step due to its moderate computational load) is lower.

3.4.3 *C-code implementation*

If equations (3.4) and (3.6) are to be implemented in C-code using the discretization methods explained in Sections 3.4.1 and 3.4.2, the following constraints must be considered: First of all, a sample time must be specified depending on the system's time constants. Nonetheless, like the main aim of this work is to test the maximum computational power of the device, the sample time will be set to the minimum possible. And secondly, the inputs and outputs of the system and the parameters must be specified. The model under study has five input variables (v_{sd}, v_{sq}, v_{rd}, v_{rq} and T_m), five output variables (i_{sd}, i_{sq}, i_{rd}, i_{rq} and T_e), and ten parameters of the model: the electrical parameters of the machine (R_s, L_s, R_r, L_r, M_{sr}), the mechanical parameters (B, J and the number of pair poles p), the grid frequency w_s, and the sample time T_s used in the discretization. A representation of the resulting IP block can be seen in Figure 3.11.

The total amount of computations needed by the two solvers can be seen in Table 3.1. Some simplifications were made in the code in order to alleviate the number of operations and ease the compiler interpretation to produce quality assembly code: precomputing parameters that do not change during the simulation, changing divisions by multiplications by the inverse value when possible, arranging the input data in a proper way for a sequential reading, etc.

3 The FOH is a mathematical model which consists of reconstructing an analog signal by a linear interpolation its actual value and the previous sample.

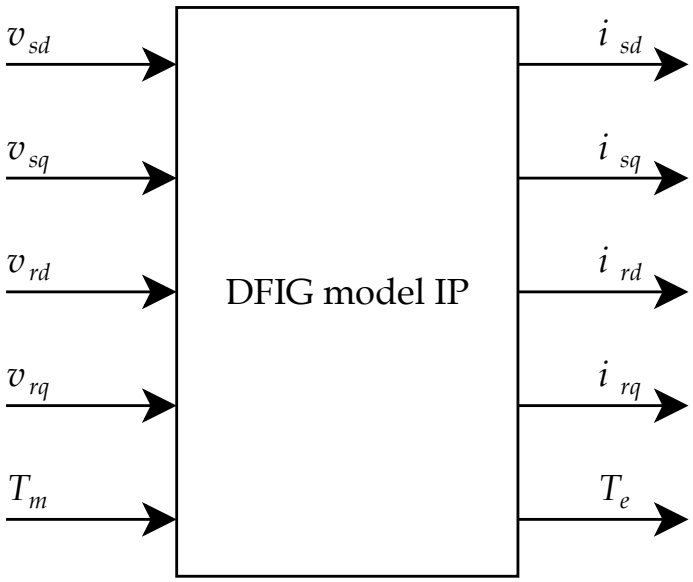


Figure 3.11: DFIG IP block

Table 3.1: Number of operations to be executed

	ADD	SUB	MUL	DIV
Euler	25	6	40	2
Tustin, of which:	152	46	324	2
- Matrix inversion	43	40	212	1
- Matrix multiplication	64	0	64	0

Once the number of operations has been obtained, the next step is to compare the error between the two methods and the golden data generated using *Simulink*. If the computationally lighter Euler model, which will use shorter time-step, produce equal or less error than its Tustin counterpart, it will therefore be more convenient to use it because of lesser hardware utilisation and lower power consumption.

3.4.4 *Full-software implementation of both discretizations*

In this subsection, equations (3.8) and (3.9) are programmed in the PS using standard C-code. The SIMD and FPU units available on the ARMv7 architecture will be utilized to boost the algorithm performance.

Two different versions of each discretization method were coded to be executed by the ARM: one using 32-bit and another one using 64-bit floating-point formats. The results regarding execution time are shown in Figure 3.12. The difference between the two versions of the code is only the size of the digital words. The different optimization options utilised in the GCC compiler, explained in detail in [77], are the following:

- -O0: no optimization. It turns off automatic vectorization regardless of additional compiler options
- -O3: with no additional commands. Automatically turns on *-ftree-vectorize*
- -O3a: adding *-mfpu=vfpv3 -ffast-math -mcpu=cortex-a9*
- -O3b: adding *-mfpu=neon -ffast-math -mcpu=cortex-a9*
- -O3c: adding only *-mcpu=cortex-a9*

To briefly explain what these compiler options are, the *Xilinx ARM GNU Toolchain* use these below by default:

- *-mfloat-abi=softfp*: This option sets the overall strategy for floating-point code compilation. It allows the generation of code using specific hardware floating-point instructions, but still uses the soft-float calling conventions
- *-mfpu=neon-fp16*: It configures the type of FPU hardware accelerator. In this case, the NEON for half-precision floating-point format is enabled

Then, depending on the level of specification (-O0 to -O3), further optimizations can be included in the compiler:

- *-ftree-verctorize:* enables NEON *automatic vectorization*[4]
- *-mfpu=vfpv3:* select the VFPUv3 as FPU
- *-mfpu=neon:* select the NEON as FPU
- *-ffast-math:* some floating-point operations are not vectorized by default due to possible loss of precision. Using this option forces vectorization of floating-point operations
- *-mcpu=cortex-a9:* the specific processor model is passed to the compiler

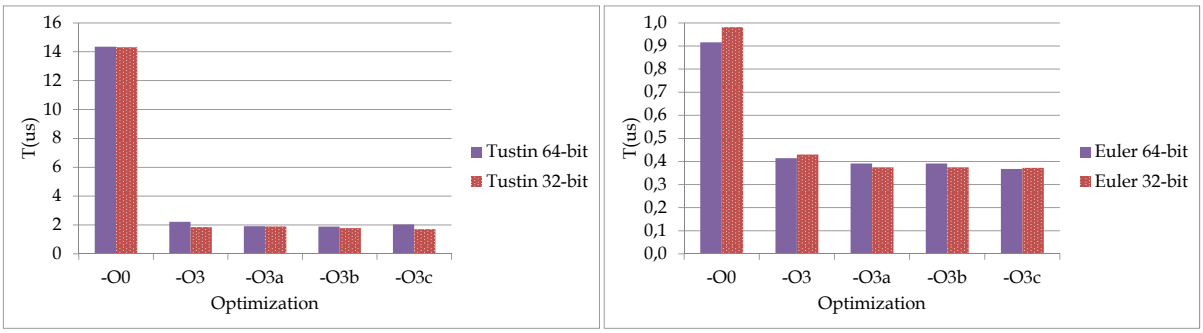


Figure 3.12: Algorithm execution time comparison on the ARM

Table 3.2: Best execution times (per iteration) on the ARM

CPU @ 667MHz	Optimization	Time (μs)
Tustin 64-bit	-O3b	1,891
Tustin 32-bit	-O3c	1,713
Euler 64-bit	-O3c	0,372
Euler 32-bit	-O3c	0,367

It can be seen in Figure 3.12 that the execution time depends highly on the algorithm implementation (and the quality of the code) but it does not vary significantly whether 32- or 64-bit variables are used. The best results obtained and the corresponding optimization option are shown in Table 3.2. Note that the clock frequency of the ARM processor present in the *Zynq-7020* is 667MHz.

3.4.5 *Full-hardware implementation of both discretizations*

Only minor changes were made to the ARM version of the C-code in order to properly adapt it to Vivado HLS. Regardless of the input and output format arrangements, the main structure of the code was kept unmodified. A test bench program was also created to let the tool automatically execute the C algorithm validation (*Pre-synthesis*[5]), and the RTL verification (*Post-synthesis*[6]).

Table 3.3 shows the latency for a 100MHz clock and the resources utilisation for the *Zynq-7020*.

3.4.6 *Discretization results and conclusion*

The reference values were obtained using a *Simulink* implementation of the continuous model equations, setting for this evaluation a 2μs fixed time-step, the *Runge-Kutta ODE4* solver and double-precision variables. The simulation was run for 1.5s and then the results compared with the *Zynq* implementations with the same time-step just to ease the evaluation of the

4 Automatic vectorization is a special case of automatic parallelization, where a computer program is converted from a scalar implementation, which processes a single pair of operands at a time, to a vector implementation, which processes one operation on multiple pairs of operands at once (see Figure 2.8 in Chapter 1)

5 A *Pre-synthesized* design is merely an RTL schematic. It enables you to see the netlist after the RTL elaboration.

6 The *Post-synthesis* is the technology schematic. It takes the synthesized input as netlist. Using a synthesized netlist as input provides information about clocks and clock-related logic in the design.

Table 3.3: Latency and area utilisation on the *Zynq-7020*

	Latency	BRAM	DSP	FF	LUT
Tustin 64b	253	12 (4%)	185 (84%)	30.959 (29%)	32.855 (61%)
Tustin 32b	216	2 (~0%)	63 (28%)	12.583 (11%)	14.870 (27%)
Euler 64b	103	12 (4%)	100 (45%)	6.660 (6%)	11.007 (20%)
Euler 32b	86	6 (2%)	32 (14%)	3.107 (2%)	5.683 (10%)

Table 3.4: 64-bit floating-point average errors (software version)

	Speed change test	Load change test
i_{sd}	$7.302e^{-8}$	$1.739e^{-6}$
i_{sq}	$6.947e^{-8}$	$2.064e^{-6}$
i_{rd}	$7.328e^{-8}$	$1.599e^{-6}$
i_{rq}	$4.724e^{-8}$	$1.884e^{-6}$
T_e	$1.451e^{-6}$	$4.289e^{-5}$

results. The average relative error for the 32-bit Euler version varied between 1.1% and 0.3% on the different case scenarios, whilst for the Tustin varied between 0.23% and 0.1%. Concerning the 64-bit version, the Euler method gave values between 0.06% and 0.016%, and the Tustin between 0.014% and 0.0016%.

These very small average absolute errors and the little difference between an Euler and Tustin discretization can be explained due to the dynamics of the machine. Most mechanical systems require time-steps from $1ms$ to $10ms$. So it is not surprising that when executing the code at a range between 500 and 5.000 times faster, any of the two methods give pretty good results.

Accordingly, it was decided to use the Euler method for its significantly lesser resources utilisation.

3.5 FULL-SOFTWARE EULER IMPLEMENTATION

This is the easiest and most straightforward implementation. It is supposed that all the system submodules are implemented in the PS and hence, all the system variables are stored in the same memory space which facilitates and reduces the implementation time considerably.

As stated in Section 3.4.4, compiler optimization options were used to infer automatic vectorization which implies arranging data and operations in a proper way to be executed by the NEON SIMD [51] and the VFPv3 FPU [52]. In some cases (see Figure 3.12) it reduced the execution time by 80% while maintaining the same precision in the results [6].

3.5.1 *64-bit floating-point full-software implementation*

When using double-precision floating-point variables, the time needed by the DFIG Euler function to return the values was $372ns$. Then, in order to evaluate the results obtained by this version, a comparison with the *Simulink* continuous-time model equations was performed. Error results of both case scenarios obtained using equation 3.1 are shown in Table 3.4. The values are in *per-unit*.

Table 3.5: 32-bit floating-point average errors (software version)

	Speed change test	Load change test
i_{sd}	$1.136e^{-3}$	$1.986e^{-3}$
i_{sq}	$7.959e^{-4}$	$2.057e^{-3}$
i_{rd}	$1.222e^{-3}$	$1.896e^{-3}$
i_{rq}	$7.554e^{-4}$	$1.925e^{-3}$
T_e	$1.674e^{-2}$	$4.485e^{-2}$

3.5.2 *32-bit floating-point full-software implementation*

On the other hand, when using single-precision floating-point variables, the execution time needed by the IP was not significantly smaller: $367ns$. This gives us an idea about how optimized are these CPUs to perform 64-bit operations, even though their architecture is based on 32-bit. Similarly to the previous implementation, the results when comparing this version with the *Simulink* model using equation 3.1 are shown in Table 3.5.

3.5.3 *Full-software implementation conclusions*

Comparing the errors of both software implementations and taking into account that the difference in execution time is barely noticeable, it is obvious that using double-precision floating-point variables is more convenient.

3.6 FULL-HARDWARE EULER IMPLEMENTATION

The design and development of this version –which includes managing the variables and the communications between the different system blocks– is significantly more time consuming, even if HLS tools are utilised. However, this option can be a good candidate –or sometimes the only option– if the objective is to obtain the minimum execution time.

During the development of this model it was found that the fastest way of realizing all the calculations was when using a very slow clock in order to let Vivado HLS implement a *full-combinatorial*[7] solution, i.e. an IP having zero latency. By default, Vivado HLS uses a $100MHz$ clock to drive the FPGA fabric. However, in this specific application it was better to set the oscillator to $5MHz$ so as to allow the internal logic to generate a result with zero latency. For the 32-bit floating-point version for example, the time needed by the logic to propagate the digital signals from input to output was $172ns$. Conversely, when utilizing a $100MHz$ clock, the logic required 76 clock cycles (i.e. $760ns$) to output a result.

It is also important to mention that not using input and output arrays in the IP avoids the use of memory interfaces allowing the data to be accessed in one clock cycle, thus reducing the total latency of the IP.

7 In a digital design, when the logic has enough time to propagate the signals between input and output without requiring to add intermediate registers, it is said that the the paths are implemented in *full-combinatorial*. Thus, the logical operations are performed with zero latency.

Table 3.6: 32-bit floating-point absolute average errors (hardware version)

	Speed change test	Load change test
i_{sd}	$4.957e^{-4}$	$1.848e^{-3}$
i_{sq}	$5.937e^{-4}$	$1.837e^{-3}$
i_{rd}	$5.788e^{-4}$	$1.745e^{-3}$
i_{rq}	$4.595e^{-4}$	$1.705e^{-3}$
T_e	$5.877e^{-4}$	$2.132e^{-3}$

3.6.1 *32-bit floating-point full-hardware implementation*

A 64-bit floating-point version of the DFIG model was quickly peeked as well using Vivado HLS. However, the amount of DSP slices available in the *Zynq-7020* was not enough to obtain a solution comparable to the software version in terms of execution time. However, a 32-bit floating-point implementation gave results which could compete with the software version. As previously announced, the best execution time was achieved using a slow clock frequency ($5MHz$) in which the longest data path required $172ns$ to output a result according to Vivado HLS. The FPGA resources used can be seen in Table 5.1. However, empiric evaluations showed that the time estimation is not what the IP really needs to stabilize the combinatorial output after performing the *place and route*. After the implementation on the *Zynq*, the timing analysis summary said that the longest data path was $256ns$. Notwithstanding, the model was tested empirically in the *ZedBoard* and looking at the waveform it was seen that the outputs were stabilized between 70 and $80ns$. Even though the results were correct, care should be taken when considering utilizing smaller timings as there is a data path that exceeds the $200ns$.

Regarding the accuracy of the results in the Speed and Load tests, they do not differ significantly from the 32-bit floating-point software version. See Table 3.6 for more details.

3.6.2 *32-bit fixed-point full-hardware implementation*

The aim to study a fixed-point version was because the logic involved in the mathematical operations is much simpler than its floating-point counterpart, and hence, the execution time tends to be shorter and the area utilisation reduces significantly. Like the model uses *per-unit* variables, it was easy to define the format of the fixed-point word for voltages and currents: $32Q30$, which uses 1 sign bit, 1 integer bit, and 30 fractional bits. For the torque and angular speed the format used was $32Q23$, using 1 sign bit, 8 integer bits, and 23 fractional bits. This allows the current and voltage signals to be between the range $]2, -2]$, whereas torque and angular speed can vary between $]256, -256]$.

Regarding the HLS implementation, the same procedure as with the floating-point version was followed. A slow clock was configured in order to let Vivado HLS synthesize a *full-combinatorial* IP. Once the synthesis was finished, Vivado HLS estimations showed that the fixed-point version executes all the necessary operations in $110ns$ using the resources shown in Table 5.1. Again, these results did not match after performing the *place and route*, which showed that the longest path required only $76ns$. Moreover, after programming and running the model in the real SoC platform, the waveform captured by the *Integrated Logic Analyzer*[8] (ILA) showed that the output results were valid and stable after $20ns$ letting the DFIG model

8 An *Integrated Logic Analyzer* is a logic analyzer IP core that can be used to monitor the internal signals of a digital design.

Table 3.7: 32-bit fixed-point absolute average errors (hardware version)

	Speed change test	Load change test
i_{sd}	$4.391e^{-4}$	$5.657e^{-4}$
i_{sq}	$8.225e^{-4}$	$4.351e^{-4}$
i_{rd}	$4.706e^{-4}$	$6.015e^{-4}$
i_{rq}	$7.842e^{-4}$	$3.971e^{-4}$
T_e	$8.680e^{-2}$	$5.165e^{-3}$

Table 3.8: FPGA hardware resources utilisation (32-bit)

Resource type	BRAM	DSP48E	FF	LUT
Available	280	220	106400	53200
Float. ver.	0	179 (81%)	10832 (10%)	24505 (46%)
Fix. ver.	0	128 (58%)	353 (~0%)	1441 (2%)

to be executed in an extraordinary short amount time. But once again, experimental results should be considered with care as according to the timing summary, which takes into account worst-case scenario (highest temperature and lowest voltage), there exists a critical path which needs *76ns* to output a valid value.

The average absolute errors compared with the *Simulink* 64-bit floating-point version can be seen in Table 3.7.

3.6.3 *Full-hardware implementation conclusions*

The amount of time required to develop and implement the fixed-point version in the FPGA was much longer than for the floating-point version even when using HLS tools. That without considering converting the model to the *per-unit* system, which is a methodology broadly used when working with electrical power systems [72]. So taking into account that the errors achieved by both hardware solutions are very low, it is not rare to suggest a floating-point implementation if the execution time is not a critical requirement.

3.7 HARDWARE-SOFTWARE CO-DESIGN

If a cooperation between PS and PL is chosen, always aiming to reduce the computation time and make the response of the controller faster, the process becomes much more complicated. The data needs to be accessible either from the PS as from the PL. This is automatically managed when using the software version or easier to handle in the *full-hardware* one, but it is not the case when both fabric and processor get involved for two reasons: (i) the data needs to be in a common area where the two technologies could access it, and (ii) both systems need to be perfectly synchronized to perform data reads and writes in the right moment. In this case, two data transfer solutions are evaluated.

The *Zynq* system has several ways of transferring data between PS and PL. As introduced in Section 2.6.3, it uses the *AXI4* data bus standard [56] which specifies three different interfaces:

- The *AXI4-Lite*, address-driven, easy to configure and mainly used to send scalar values

- The *AXI4-Master* or *AXI4-Full*, also address-driven, able to send data bursts but more complicated to configure
- And the *AXI4-Stream*, not address-driven, single-direction, easier to implement and used mainly to process data continuously and at a very high data rates

Considering the characteristics of the system where it is required to write and read back only five different variables, the *AXI4-Stream* was discarded for this application. Either the *AXI4-Master* or the *AXI4-Lite* can be a possible option.

Regarding the input and output data, it has to be stored in an area accessible from either the PL as the PS. There exist three possible solutions in order to accomplish this. One method could be to use the DDR. The PS is directly connected to it using dedicated channels and the PL can have access to it through an *AXI slave* port. The second option is to use the $256KB$ OCM available inside the PS. Again, the PL can access to it via the *AXI slave* port. The last possibility is to use the BRAM available in the FPGA fabric. The main difference between the three is that in the first two cases the hardware IP has to behave as a master writing the data to the memory space without any intervention of the PS. Conversely, when using the BRAM, it has to be the PS the one managing the data transfers. The OCM is used by both processors to communicate to each other. When compared to DDR memory, OCM provides very high performance and low latency access from both processors. Hence, OCM has the lowest latency for the PS while the BRAM provides the lowest for the PL. DDR is suitable for large volumes of data which neither the OCM nor BRAM can manage. Hence, the DDR was dismissed. A simplified diagram of the two implementations can be seen in Figure 3.13.

Moreover, if the PL has a master role there are two channels to accomplish data transfers. One is using the *High-Performance* (HP) port. The *Accelerator Coherency Port* (ACP) emerges as another possible solution by enabling hardware accelerators to issue coherent accesses to the memory space. This eliminates the need of flushing the data compared to the HP option, which in the end reduces the total execution time.

In order to compare the use of a hardware accelerator with the *full-software* and *full-hardware* versions, the total time has to include the time the IP needs to access the data, the time it needs to perform the calculations –same as the *full-hardware* implementation, and finally the time it needs to write the data in the shared memory space to make it available for the PS.

Regarding the data to transfer at every cycle, the IP needs four voltages (v_{sd} , v_{sq}, v_{rd}, and v_{rq}) and the mechanical –load– torque (T_m) to compute the four currents (i_{sd}, i_{sq}, i_{rd}, and i_{rq}) and the electromagnetic torque (T_e). Therefore, using 32-bit variables this translates to 20 bytes to read and 20 to write.

3.7.1 *Hardware accelerator using OCM*

Accessing the OCM from the PL was made through the ACP port. It has a 64-bit wide *AXI* interface which allows the PL to access the OCM using an *AXI4-Master* interface while maintaining memory coherency. Additionally, the DFIG IP has an *AXI4-Lite* interface used to set the model parameters and control the IP.

Using a clock for the *AXI* connections of $250MHz$, the time needed by the ARM when using the OCM was $869ns$ and $885ns$ to write and read data respectively.

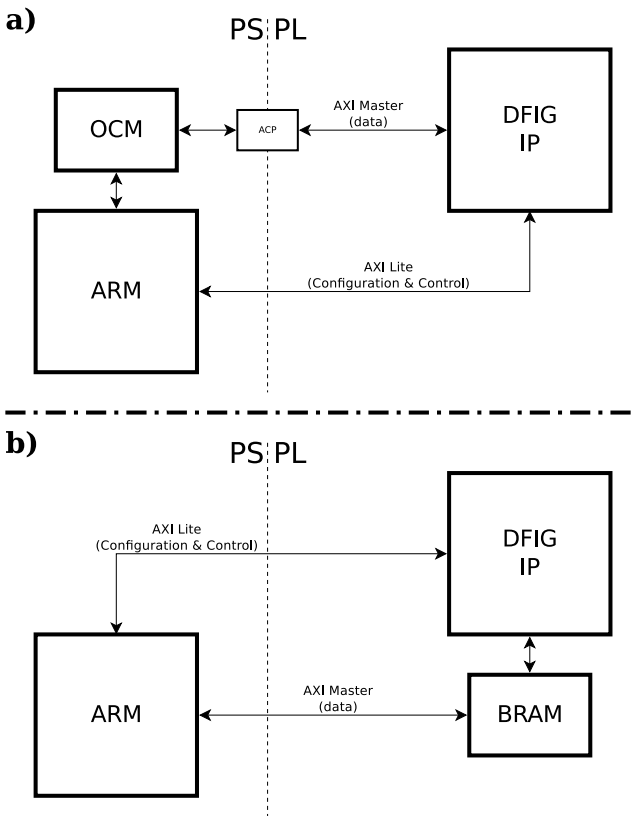


Figure 3.13: PS-PL Interconnections. a) Using OCM. b) Using BRAM.

3.7.2 *Hardware accelerator using BRAM*

In the case where the BRAM was used as the shared memory, the data transfer times were $444ns$ to write and $562ns$ to read data using an *AXI4-Master* interface through a *General-Purpose* (GP) port driven by a $250MHz$ clock.

3.7.3 *Hardware-software co-design conclusions*

It has to be taken into account that the time needed to transfer small amounts of data is not optimized and hence, for this specific case scenario, this solution lacks of interest.

3.8 CHAPTER CONCLUSIONS

The objective of this Chapter was to evaluate how suitable is the *Zynq* SoC platform to be used in low-cost embedded RTS of electromechanical systems. This study was achieved through the implementation of a DFIG coded in C/C++ in either the PS as well as in the PL.

A first study of two different discretization methods was made in order to see whether it was more interesting of using a simpler, lighter and faster solver (Euler's Forward method), or a more precise and with more computational burden one (bilineal or Tustin method). Results showed that for this specific case, where the system dynamics of the DFIG are between 500 and 5.000 times slower, the accuracy of the Euler version was sufficient. Hence, the Euler model was chosen to be used for the rest of the study.

A total of six different versions, two in the processor, two in the FPGA fabric, and two using hardware accelerators were implemented and then compared in terms of execution

time and accuracy. The last two options were achieved making an evaluation of the data transfer speeds to find whether it is interesting or not to have the DFIG model running as a hardware accelerator.

A *Simulink* continuous-time model using 100*ns* of simulation time step and a *Runge-Kutta ODE4* solver was taken as a reference to compare the precision of the results in two different scenarios: (i) a $\pm$30% progressive change in the speed of the shaft; and (ii) a change in the load from 50% to 85% of the nominal power of the generator.

Results showed that in terms of accuracy, having a model running with a time-step four times shorter but half the precision did not obtain better results than the 64-bit floating-point software implementation. Furthermore, using a 32-bit floating-point software implementation did not reduce significantly the execution time compared with its 64-bit counterpart. Hence, if the system is not execution-time critical, a software, double-precision version is recommended. If however, the best performance in terms of rapidity of the calculations is needed, the fixed-point version implemented in the PL is the most convenient solution.

Regarding the full-hardware implementation, the execution time estimations made by Vivado HLS after synthesis were not coherent with the post *place and route* timing analysis, showing in the fixed-point implementation more than 30% of time reduction. Even though an execution time of 76*ns* for a DFIG model may seem too short considering the real system response, this study shows the power of this platform to compute complex-averaged algorithms. It allows the possibility to improve and/or extend this model with additional functionalities like including non-linearities, losses, saturation, aging, etc. Besides, the re-utilization of hardware resources using HLS directives in order to reduce area might be as well considered.

If the focus is put on implementing a hardware accelerator in the PL, the time needed to transfer small amounts of data (40 bytes of total data transfer) is not optimized. This option is therefore not interesting for this specific application. However, for control algorithms running at a relatively large period, data transfer time delay has a smaller impact on overall performance. Thus, PL hardware acceleration might be interesting as well if the purpose is to alleviate computational load from the PS.

4

ERTS FOR POWER ELECTRONIC SYSTEMS: THE MMC CASE

4.1 INTRODUCTION

As previously stated, the aim of this Section is to use a scalable power converter with a complex structure in order to exploit the device capabilities at maximum coping with the eRTS requirements. Hence, an MMC was chosen as the application to be studied [11]. The idea is to have one –or several– IP blocks which estimate, e.g. phase output voltages, arm and load currents, *Half-Bridge* (HB) capacitor voltages, etc., based on other measured magnitudes. Then, these IP blocks could be utilized for fault-tolerant embedded controllers, where the estimated variables would be used in case of a sensor fault. Furthermore, they could also be adopted as well as estimators, observers, or also applied for diagnosis and health-monitoring [8, 15, 6, 7, 14], without forgetting HIL applications for testing the control before its deployment on the real plant [16, 17, 18, 19, 8]., and *sensorless* control [20, 21, 22, 23].

MMCs offer significant benefits compared to other types of *Voltage-Sourced Converters* (VSC) [78, 12, 13]. The modular structure provides the flexibility to scale the voltage and power level by adding more *sub-modules* (SM). Hence, when the number of them is sufficiently high, the produced AC voltage has very low distortion which eliminates most of the filtering requirements. Furthermore, the losses decrease significantly compared to other VSC due to the low switching frequency of each SM, hence, of each IGBT [13, 79].

However, the high number of devices in MMCs increase the computational burden in electromagnetic transient-type simulations. Detailed MMC models must include the representation of hundreds, sometimes thousands of switches and small numerical integration time steps are required to accurately represent the multiple simultaneous switching events. The significant computational load needed by these models require developing other simpler, more efficient models [80, 81, 41, 43].

In this paper, the *Xilinx*'s *Zynq-7045 SoC* has been chosen because of its powerful dual-core *ARM Cortex-A9* processor and its versatile and large programmable logic (*Kintex-7* family equivalent) [54]. This platform is a superior version of the one used in Chapter 2 (*Zynq-7020*) because the requirements in terms of area and computational power the digital simulation of MMC needs are greater than the ones to simulate electromagnetic systems. The additional hardware resources will be needed because the goal is to keep short the execution time while increasing the number of SM [14].

4.2 THE METHODOLOGY

Conversely to the DFIG case, due to the huge amount of computations when the number of HBs increase and the short time-step required to properly emulate the switching behavior, a *full-software* version will not be considered for this study. Nevertheless, thanks to the inherent parallel structure of the MMC and making use of the appropriate formulation, realizing one

–or several– IP blocks to perform the mathematical computations of the different converter arms simultaneously stands out to be a potentially good solution.

Hence, an IP block for its use in a *full-hardware* design or as a *hardware accelerator* will be developed. This IP will be evaluated when changing the number of SM from 6 to 700 per phase, and also using different data word sizes and formats. Therefore, 64- and 32-bit floating-point and 32-bit fixed-point data formats for variables and parameters will be contrasted. Then, these different versions of the eRTS will be valued in terms of precision, computational power and area utilization.

4.3 CASE APPLICATION DESCRIPTION: THE MMC

The basic structure of an MMC is shown in Figure 4.1. It is formed by N Half-Bridge (HB) SM per arm capable of producing a line-to-neutral voltage waveform of $N+1$ levels [11]. An inductor L_{arm} is added on each arm to limit current harmonics and the short circuit current in the event of a DC fault. Each SM includes a capacitor and two IGBTs with antiparallel diodes as shown in the enlarged part of Figure 4.1.

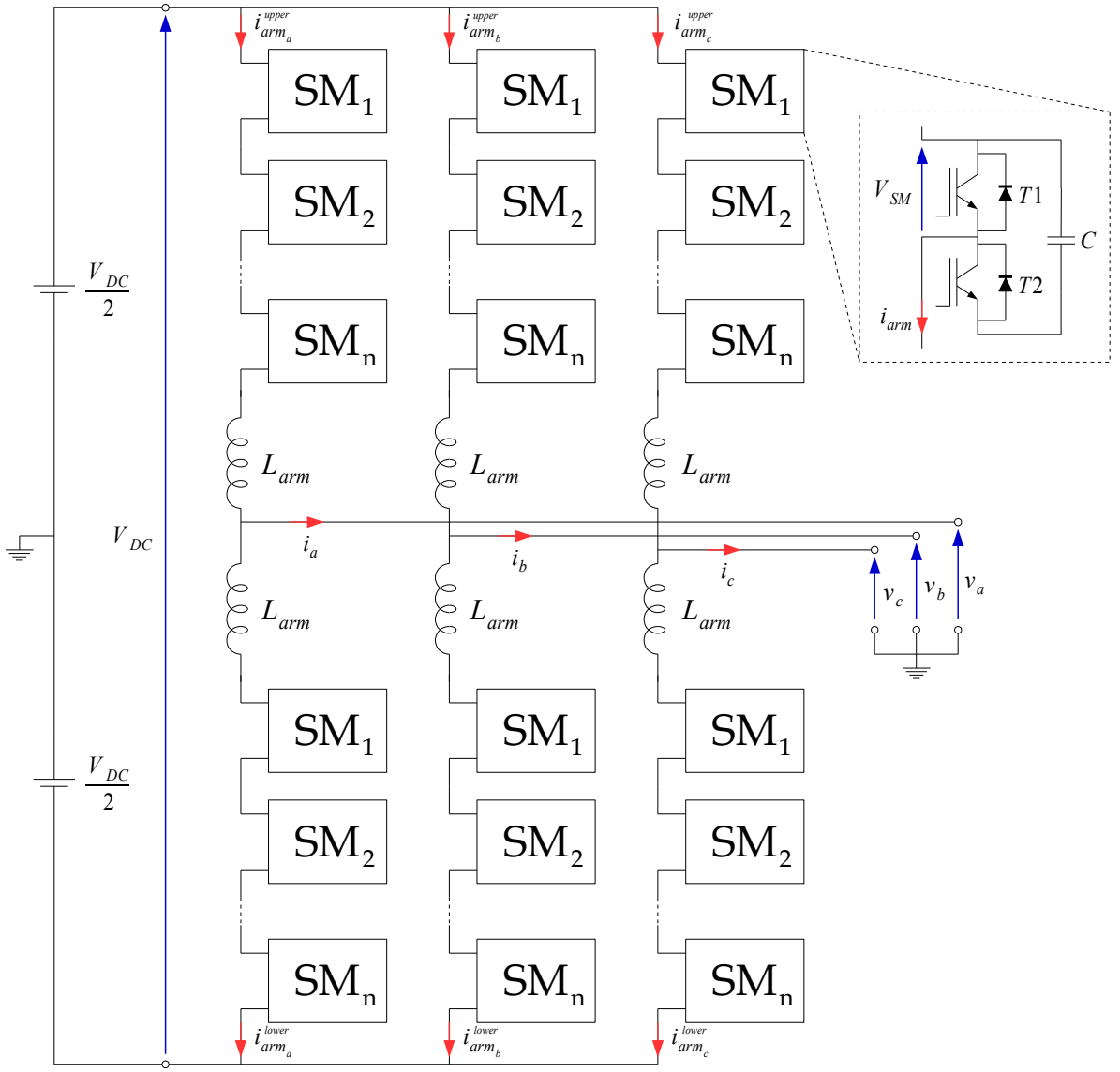


Figure 4.1: Structure of the MMC

The two IGBTs of each SM can be controlled through gate signals which allow two different states. In the ON-state $T2$ is fired and $T1$ is blocked and accordingly, the SM voltage V_{SM} is equal to the capacitor voltage. Depending on the arm current i_{arm} direction, the capacitor will be charged through the IGBT as shown in Figure 4.2a, or discharged through the diode as shown in Figure 4.2b (equations 4.1 and 4.2 respectively). In the OFF-state $T1$ is fired and $T2$ is blocked, resulting in $V_{SM} = 0$. The capacitor voltage remains constant whatever the direction of i_{arm} is, but in one case the current flows through the IGBT as shown in Figure 4.2c, and in the other through the diode as shown in Figure 4.2d (equations 4.3 and 4.4 respectively).

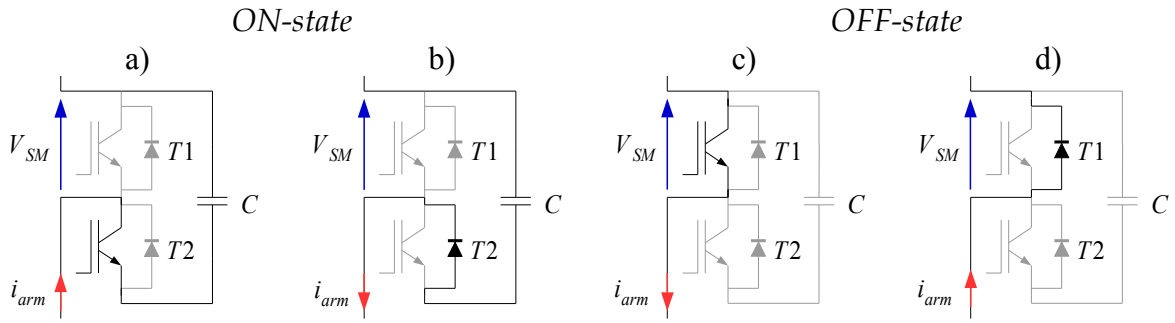


Figure 4.2: HB functioning modes

The corresponding output voltage of these four modes can be calculated using the following equations:

$$\text{a) } V_{SM} = R_{on}^{IGBT} i_{arm} + V_{cap} \tag{4.1}$$
$$\text{b) } V_{SM} = R_{on}^{diode} i_{arm} + V_{cap} \tag{4.2}$$
$$\text{c) } V_{SM} = R_{on}^{IGBT} i_{arm} \tag{4.3}$$
$$\text{d) } V_{SM} = R_{on}^{diode} i_{arm} \tag{4.4}$$

After explaining how each SM works and coming back to Figure 4.1, the output AC voltages of each converter phase can be computed using 4.5.

$$v_{abc} = \frac{V_{arm}^{l} - V_{arm}^{u}}{2} + \frac{R_{eq}}{2} i_{abc} + \frac{L_{arm}}{2} \frac{d}{dt} i_{abc} \tag{4.5}$$

where V_{arm}^{u} and V_{arm}^{l} are the voltages to be inserted in the upper and lower arm (equations 4.6 and 4.7 respectively), R_{eq} the equivalent resistor considering the ON resistance of the MOSFETs and anti-parallel diodes plus the parasitic resistance of the arm inductors, and i_{abc} are the phase AC output currents (equation 4.8). Hence, the output current i_{abc} can be controlled by means of V_{arm}^{u} and V_{arm}^{l} when the MMC is connected to an AC grid of voltage v_{abc}.

$$V_{arm}^{u} = \frac{V_{DC}}{2} - v_{abc}^{ref} - v_{circ_{abc}}^{ref} \tag{4.6}$$
$$V_{arm}^{l} = \frac{V_{DC}}{2} + v_{abc}^{ref} - v_{circ_{abc}}^{ref} \tag{4.7}$$
$$i_{abc} = i_{arm_{abc}}^{u} - i_{arm_{abc}}^{l} \tag{4.8}$$

From the above equations, V_{DC} is the pole-to-pole DC voltage, v^{ref}_{abc} is the required output phase voltage, and $v^{ref}_{circ_{abc}}$ the voltage linked to the circulating currents which is computed using equation 4.9 according to [82].

$$v^{ref}_{circ_{abc}} = R_a \left(i^{ref}_{circ_{abc}} - i_{circ_{abc}} \right) - \hat{R} i^{ref}_{circ_{abc}} \tag{4.9}$$

where R_a is the referred as the "active resistance" –which is in fact the gain parameter of the P regulator that controls the circulating current, and $\hat{R}$ is an estimate of R_{eq}, and $i^{ref}_{circ_{abc}}$ is the circulating current reference which is calculated using equation 4.10. More information about this implementation can be found in [83].

$$i^{ref}_{circ_{abc}} = \frac{i^{u}_{abc} + i^{l}_{abc}}{2} \tag{4.10}$$

In a perfectly balanced three-phase MMC, each arm would provide half of the AC output current plus a circulating current whose value would correspond to one third of the total DC current. Notwithstanding, capacitor voltage variations lead to additional circulating currents that increase not only the RMS value of the arm currents, but the capacitor voltage oscillations and the overall losses as well.

4.4 MMC NUMERICAL MODELS

4.4.1 *MMC model classification*

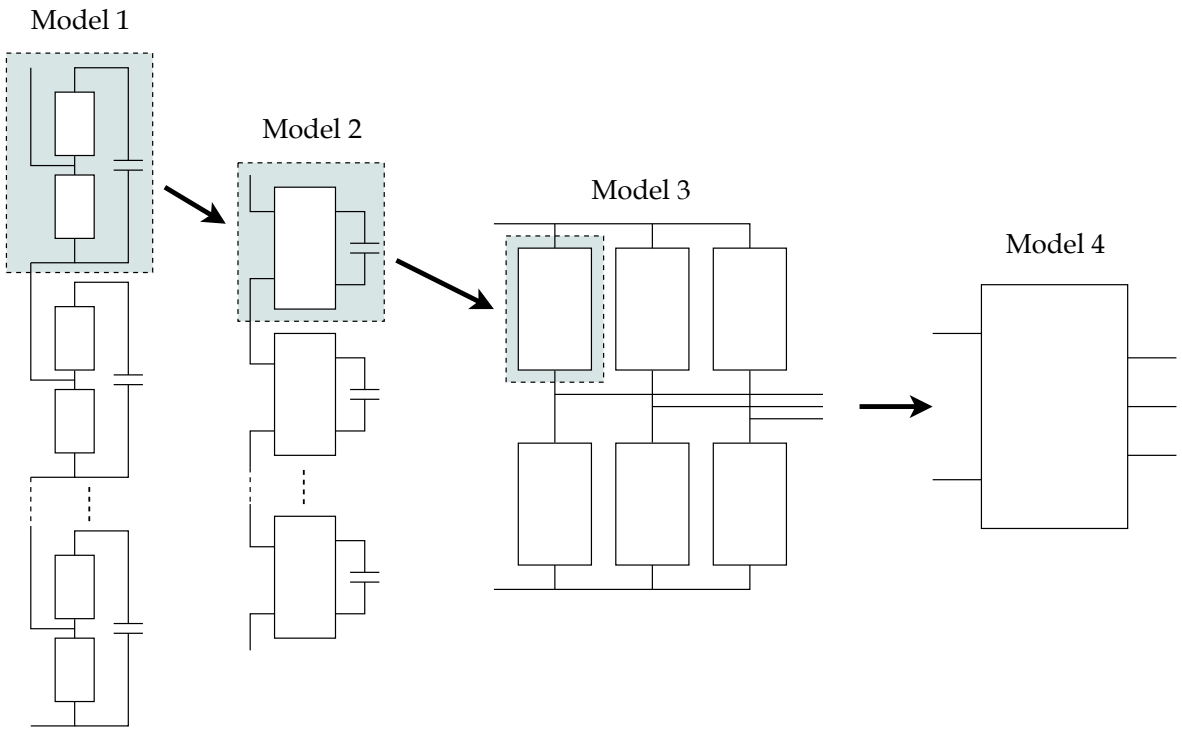


Figure 4.3: Model evolution with decreasing complexity

[2] defines four levels of models according to the type of study and required accuracy. A graphical depiction of these models can be seen in Figure 4.3. These four models are:

- **Model 1:** The *Detailed IGBT-based Model* is the most detailed and computationally costly representation of power electronics. This model uses an ideal switch, two non-linear (series and anti-parallel) diodes, and two snubber circuits [81].

- **Model 2:** The *Equivalent Circuit-based Model* replace the power switches by ON/OFF state resistors. R_{on} is in the range of $10^{-3}\Omega$ whereas R_{off} is in the range of $10^{6}\Omega$. This approach reduces the amount of internal electrical nodes allowing the creation of a Norton equivalent for each MMC arm [80].

- **Model 3:** In the *MMC Arm Switching Function* and according to [84], each arm is reduced to a single cell with a capacitance equal to $\frac{C}{N}$ and a voltage equal to $\sum v_{caps}$, where C is the cell capacitance, N the number of SMs, and $\sum v_{caps}$ the summation of all HB capacitor voltages. Thus, it assumes all capacitor voltages are perfectly balanced, having all the same voltage value. If S_n is the logical function denoting ON and OFF-STATE cells (i.e. 1 and 0 respectively), then, the arm voltage v_{arm} is calculated using the following equation:

$$v_{arm} = \sum S_n v_{caps} + N R_{on} i_{arm}$$

  where i_{arm} is the arm current and R_{on} the linear conductivity losses ON for each SM. The equivalent current $i_{caps_{tot}}$ that flows through the capacitor is:

$$i_{caps_{tot}} = S_n i_{arm}$$

- **Model 4:** The *Average Value Model* presented in [81] does not explicitly represents the IGBTs and their diodes. Conversely, the behavior of the MMC is modeled using controlled voltage and current sources. Three ideal AC voltage sources are used to model the AC-side. The DC-side is modeled by means of a DC current source whose value is derived using the principle of power balance. This model also assumes that the internal variables of the MMC are fully controlled, i.e. all SM capacitor voltages are perfectly balanced and circulating currents in each phase are suppressed.

Models 1 and 2 offer great representation of the power electronics. However, the computational burden is cumbersome if the focus is to implement a reliable and fast MMC model into SoC devices, specially when the number of SM is high. Conversely, models 3 and 4 have very few computation requirements, but their main limitation is their inability to represent the internal converter dynamics, which are essential when developing eRTSs. The model used in this dissertation does not apply to any of these cases and it is explained in the next Subsection.

4.4.2 *The simplified model*

The model proposed in [85] can be placed between models 2 and 3. It is based on the *Equivalent Circuit-based Model* and aims to reduce the computational requirements of the detailed models. It neglects the R_{off} resistance due to its high value when compared to R_{on}, which reduces the computation burden, but conversely to Model 3, it stores the capacitor voltages and represents the circulating currents.

As shown in Figure 4.4a, this simplified model consists of a variable voltage source V_{arm}, a variable capacitor C_{eq}, a variable resistor R_{eq}, and a reactor L_{arm} on each phase branch. Hence, all the SMs in each arm are reduced to these components, regardless of the number of

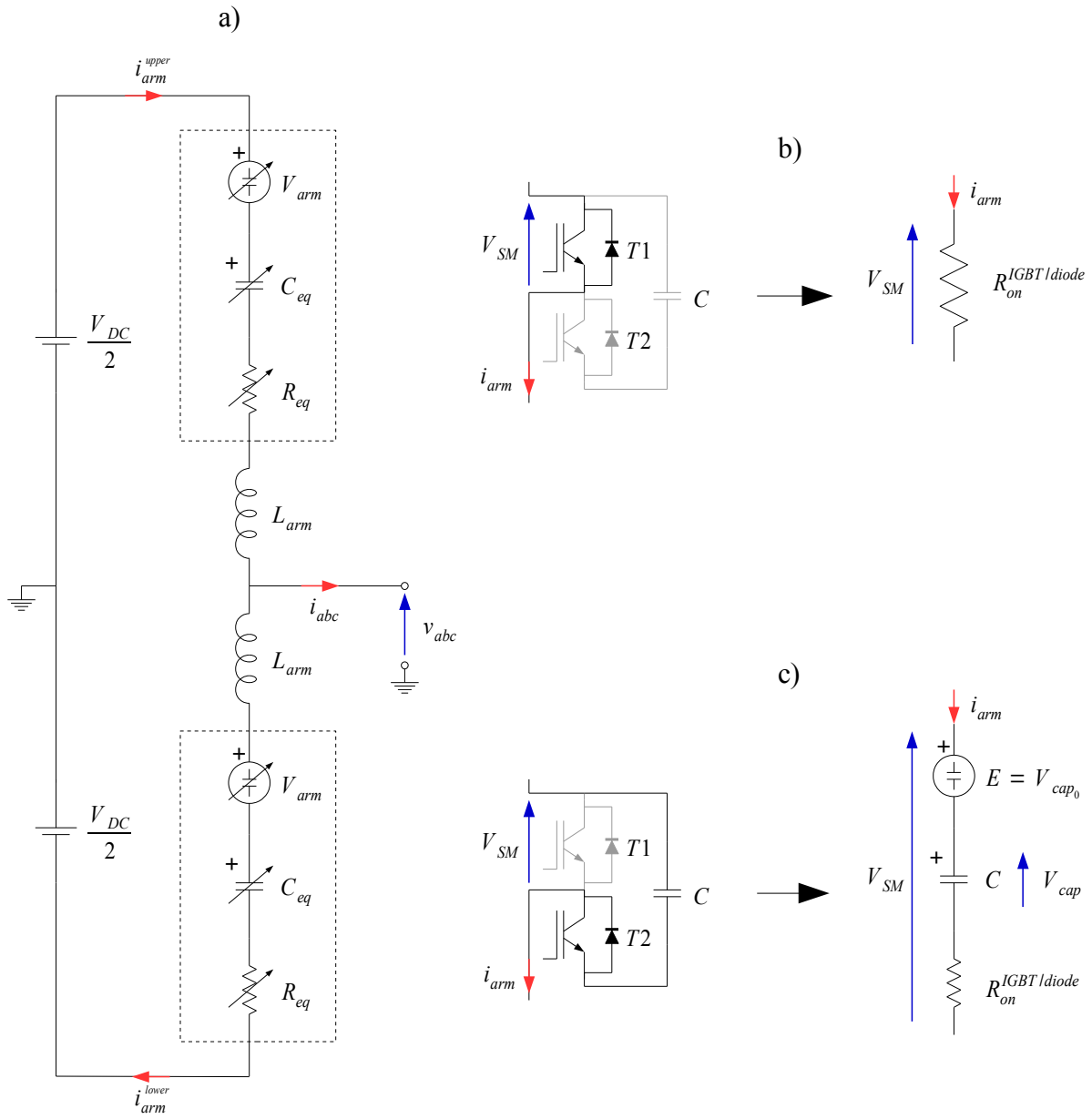


Figure 4.4: a) MMC arm circuit, b) OFF-state SM, c) ON-state SM

levels considered. This model is based on the Thevenin equivalent circuit with the following considerations:

- The IGBTs and diodes are modeled as a two-state resistor: R_{on} and R_{off}.
- The OFF-state resistance R_{off} of diodes and IGBTs are considered to be infinite.
- The OFF-state SMs (see Figure 4.4b) are replaced by an equivalent resistor whose value is R_{on}^{IGBT}if i_{arm} is positive, or R_{on}^{diode} if i_{arm} is negative. R_{on}^{IGBT} and R_{on}^{diode} are the IGBT and diode conduction resistance respectively.
- The ON-state SMs (see Figure 4.4c) are replaced by an equivalent capacitor C with zero voltage, a voltage source E with the same voltage of the capacitor V_{cap_0} at the instant of time when the SM is switched on, and a resistor. The resistor is equal to R_{on}^{diode} if i_{arm} is positive, and R_{on}^{IGBT} if i_{arm} is negative.

Hence, the model parameters V_{arm} and C_{eq} can be calculated using 5.4 and 4.12.

$$V_{arm} = \sum_{i=1}^{N_{on}} V_{cap_0} \tag{4.11}$$

$$C_{eq} = \frac{C}{N_{on}} \tag{4.12}$$

But as seen in Figures 4.4b and 4.4c, the equivalent resistor R_{eq} however depends on the sense of the arm current i_{arm} and the state of each SM:

- If $i_{arm} > 0$:

$$R_{eq} = N_{on}R_{on}^{diode} + N_{off}R_{on}^{IGBT} \tag{4.13}$$

- If $i_{arm} < 0$:

$$R_{eq} = N_{off}R_{on}^{diode} + N_{on}R_{on}^{IGBT} \tag{4.14}$$

being N_{on} and N_{off} the number of ON and OFF SMs of each arm respectively, and taking into account that the total number of cells N is equal to:

$$N = N_{on} + N_{off}$$

4.5 DISCRETIZATION

The capacitor charge in continuous time can be expressed by means of equation 4.15.

$$i_{arm}(t) = C\frac{d}{dt}V_{cap}(t) \tag{4.15}$$

Applying the integral to both parts of the equal and operating leads to:

$$V_{cap}(t) = V_{cap_0} + \frac{1}{C}\int_0^{T_s} i_{arm}(t)\,dt \tag{4.16}$$

Then, realizing the integral equation 4.17 is obtained.

$$V_{cap}(t) \approx V_{cap_0} + \frac{T_s}{C}\frac{i_{arm}(t) + i_{arm}(t - T_s)}{2} \tag{4.17}$$

being T_s the sampling time, V_{cap_0} the capacitor voltage at instant $t - T_s$, $i_{arm}(t)$ the measured current at instant t, and $i_{arm}(t - T_s)$ the measured current at instant $t - T_s$. This equation can be directly implemented in software, but for a more appropriate notation, equation 4.17 can be rewritten as:

$$V_{cap}(k) = V_{cap}(k-1) + \frac{T_s}{C}\frac{i_{arm}(k) + i_{arm}(k-1)}{2} \tag{4.18}$$

Then, in order to calculate the phase voltage equation 4.19 is used.

$$v_{abc}(t) = \frac{V^l_{arm}(t) - V^u_{arm}(t)}{2} + \frac{R_{eq}}{2} i_{abc}(t) + \frac{L_{arm}}{2}\frac{d}{dt} i_{abc}(t) \tag{4.19}$$

But this time the Euler's Forward method is applied leading to equation:

$$v_{abc}(k) = \frac{V^l_{arm}(k-1) - V^u_{arm}(k-1)}{2} + \frac{R_{eq}}{2} i_{abc}(k-1) + \frac{L_{arm}}{2\Delta T_s}\left(i_{abc}(k) - i_{abc}(k-1)\right)$$

4.6 HARDWARE IMPLEMENTATION

As within the DFIG case, the MMC model will be coded using Vivado HLS in order to generate an IP block containing the considered inputs and outputs of the system. Several approaches were studied considering the characteristics of the converter aiming to reduce at maximum the execution time for an adequate emulation of the power electronics switching.

Focusing on the objective of parallelizing the calculations, the three most obvious possibilities were to divide the system into 6 IP blocks (one per arm), 3 IP blocks (one per phase), or one single IP block for emulating the whole converter. Authors discovered that the best results were found when the model was divided into three identical functions, i.e. one per phase, mainly because a compromise between parallelization of computations and reduction in input and output signals. These three IPs could be called simultaneously in order to compute each phase at the same time reducing the total execution time by three if compared to a sequential realization as it would be done using a general purpose processor.

Vivado HLS was very successful in reusing hardware, managing memory accesses and pipelining the calculations reducing the latency significantly. It is relevant to mention another work found in the literature comparing a hard-coded HDL IP with their HLS counterparts for its intended use in power electronics systems [10]. However, they evaluate a generic *hardware solver* –which simply performs matrix multiplications– changing the number of states to be computed rather than a real model of the power converter. Moreover, they do not compare different variable formats but a custom-made floating-point format.

Needless to say that the decision of format and word size of variables and parameters impacts directly in the precision of the results, the execution time, and particularly the FPGA area utilized. The difference between fixed-point and floating-point is that the firsts have

smaller footprints and latencies, whereas the lasts have larger dynamic ranges and better accuracy. In this Section three different formats are therefore compared: 64- and 32-bit floating-point, and 32-bit fixed-point. It is worth saying that when using C-code it is quite straightforward to change from one format to another. Making use of the type definition functionality *typedef* when declaring variables, and then changing it to either *double* (64-bit floating-point), *float* (32-bit floating-point), or *ap_fixed<M,N>* (a fixed-point declaration where M is the word width and N the number of integer bits), all variables and parameters of the system can be changed instantaneously. This is a powerful feature of HLS tools because it is not necessary to handle all the logic involved in floating-point format when using HDL languages, which is an arduous task and quite error-prone.

4.6.1 *Description of the IP*

When coding in Vivado HLS, the function that performs the calculations has to be defined where the inputs and outputs of the system are defined as function parameters. They can be scalar, vectors, or even arrays with several dimensions. In this case, the voltage reference to be generated by each phase v^{ref}_{abc}, the upper and lower arm currents i^{u}_{abc} and i^{l}_{abc}, the triangular waveform used to be compared with the v^{ref}_{abc} to generate the PWM signals of all the SMs, the internal voltage controlling the circulating current u^{ref}_{circ}, and the output voltage of each phase v_{abc} are all scalar values. The capacitor voltages of every cell $v_{caps}[N]$ are defined as vectors in order to easily change the number of SM setting the appropriate parameter, without the need of modifying the internal code of the function. All these inputs and outputs are depicted in Fig. 4.5.

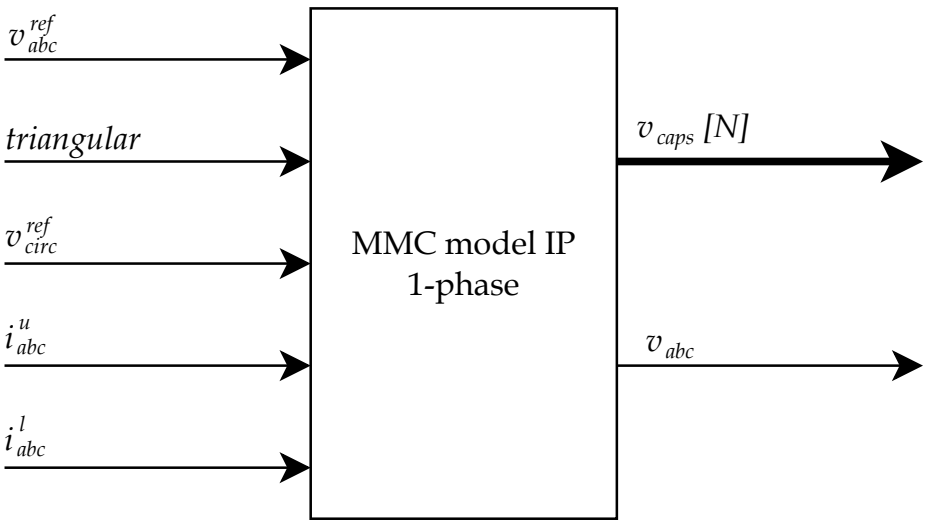


Figure 4.5: An IP block representing one phase of the MMC

In order to reduce the block inputs to decrease the data transfers and improve the total execution time, the capacitor balance algorithm was included inside the IP This way, the capacitor voltage values and the SM states do not need to be driven out so as to be computed in another place, hence improving the total IP performance. Moreover, the PWM duty cycle comparison and the number of cells required was also calculated in it for the same reason. All calculations performed by this IP block –or C function actually– can be divided into the following steps:

- **Step 1:** Capacitor voltages are updated by using 4.20, where $S_i^{u,l}$ is a binary function containing the ON-OFF switch states.

$$V_{cap_i}^{u,l}(k) = V_{cap_i}^{u,l}(k-1) + S_i^{u,l}(k-1)\frac{T_s}{C}\frac{i_{arm}^{u,l}(k) + i_{arm}^{u,l}(k-1)}{2} \tag{4.20}$$

- **Step 2:** The number of cells $N_{on}^{u,l}(k)$ needed to produce the required voltage at instant k is calculated.

– *The following steps are only executed if the number of cells change in a specific arm, i.e.* $N_{on}^{u,l}(k) \neq N_{on}^{u,l}(k-1)$ *where* $N_{on}^{u,l}(k)$ *is the number of SMs to be inserted in the upper and lower arms.*

- **Step 3:** The capacitor balancing algorithm chooses, according to the direction of the current i_{arm} and the voltage of the SMs V_{cap_i}, the most suitable one to be switched ON or OFF according to the required output voltage v_{abc}^{ref}.

- **Step 4:** The equivalent capacity C_{eq}, the equivalent resistor R_{eq}, and the output voltage v_{abc}^{ref} are calculated using equations 4.12, 4.13 or 4.14 depending on the current direction, and 4.5.

- **Step 5:** The variables registering the values of the previous iteration of switch gates $S_i^{u,l}(k-1)$, capacitor voltages $V_{cap_i}^{u,l}(k-1)$, and currents $i_{arm}^{u,l}(k-1)$ are updated.

Once the MMC function is coded, Vivado HLS generates automatically all the control signals, internal data channels and operations to be performed, create inputs and output ports, and performs scheduling and binding process generating an IP block ready to be utilised in the *Vivado Design Suite*.

4.7 RESULTS

In this Section, the three different implementations are compared in terms of resources usage, execution time, and precision against a PSCAD simulation which uses 64-bit floating-point data format.

4.7.1 *Resources usage*

It is convenient to remember that the *Zynq-7045* FPGA has 900 DSP units, 437.200 Flip-Flops, 1.090 BRAM blocks, and 218.600 Look-up Tables. The amount of resources allowed to handle up to 700 HB-SMs per phase except for the fixed-point case, where the amount of DSPs exceeded the available units when the number of SMs surpassed 200. It has to be highlighted that these evaluations are per phase. Therefore, attention has to be taken when any of the resources approach 30% because the IP has to be triplicated in order to be able to emulate a three-phase converter. Other possibilities would be either to execute the function three times to calculate all the phases, hence triplicating the execution time; or by selecting the highest range *Zynq* (the one used in this Section is the second highest); or by limiting the number of DSPs –or any other resources– utilized making use of HLS directives at the expense of execution time. However, in order to have a fair comparison between the fixed- and floating-point implementations, no resource allocation directives or any other *pragmas* were used, thus letting Vivado HLS perform all the synthesis with the default settings.

As said previously, the model was structured in order to easily change the number of cells and the type of data used for inputs, outputs, internal variables and parameters. Three different types of data formats were evaluated changing the number of cells per phase from 6 to 700. Figure 4.6 shows the resources usage of all implementations. The results are shown in percentage of the total amount of every hardware resource available in the *Zynq-7045*.

It can be seen in these graphs that in the worst case, i.e. 700 HB per phase, both floating-point versions did not consume more than 30% of any of the hardware resources. Conversely, in the fixed-point version the maximum amount of resources for a single phase exceed the available DSPs once reached 200 HB cells per phase. But as said previously, if the purpose is to simulate a three-phase converter, the usage of any resource should not exceed the 30%. These limits are reached for LUTs around 700 cells for both of the floating-point versions, and 60 cells for the fixed-point version because of exhaustion of DSP units.

4.7.2 *Execution time*

Several tests were previously performed trying to find which clock was the most appropriate for most of the cases. It was found that using a $100MHz$ clock gave the shortest execution time on average. However, author suggests to vary this parameter to explore possible better implementations once the number of cells has been defined. Figure 4.7 shows the results of all the implementations for that specific clock frequency, and in Table 4.1 the detailed results.

Table 4.1: Execution time (in μs)

no. cells	64-bit float.	32-bit float.	32-bit fixed
6	1,25	0,93	0,24
8	1,38	1,03	0,28
10	1,69	1,38	0,57
20	2,48	2,12	0,57
30	3,29	2,88	1,05
60	6,26	5,12	1,56
120	11,74	9,76	3,05
200	19,15	15,97	6,06
400	37,65	31,47	10,08
700	65,50	54,72	35,09

Regarding execution time, it can be seen that the difference between the 64-bit floating-point and the 32-bit fixed-point versions is up to 5 times shorter for six cells per phase. Furthermore, it was possible to get the fixed-point version executed at $216ns$ when using a $166.7MHz$ clock. However, this gap between fixed- and floating-point implementations reduces when the number of cells increase. On the other hand, both floating-point versions do not differ significantly. Therefore, author suggests to utilize the 32-bit fixed-point version using *per-unit* values with 2 integer bits –plus one for the sign whenever needed– and the rest as decimal values. If huge dynamic ranges need to be handled though, the 64-bit floating-point version is the best choice if the available resources are not a constraint.

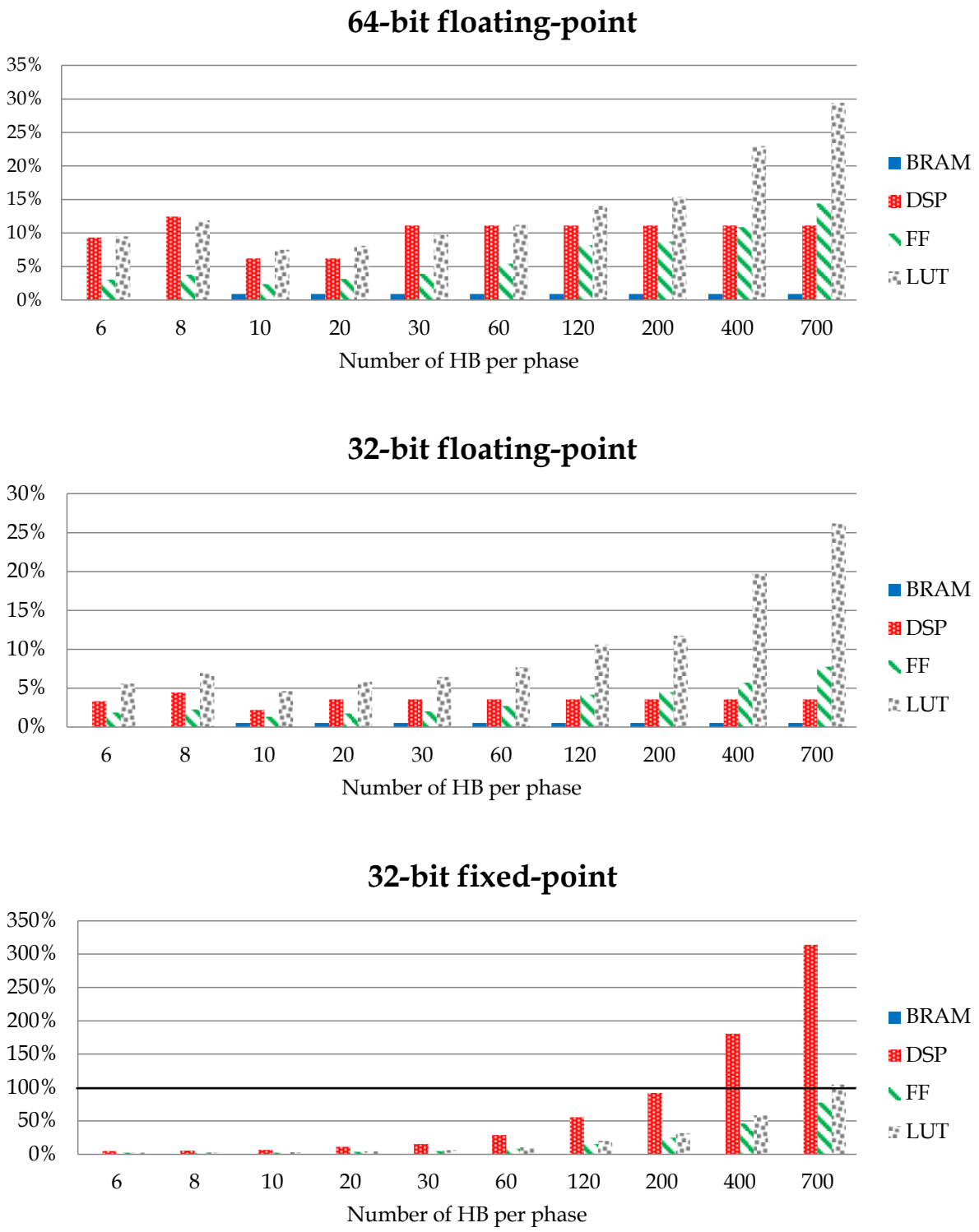


Figure 4.6: FPGA resources usage when using no directives or *pragmas*

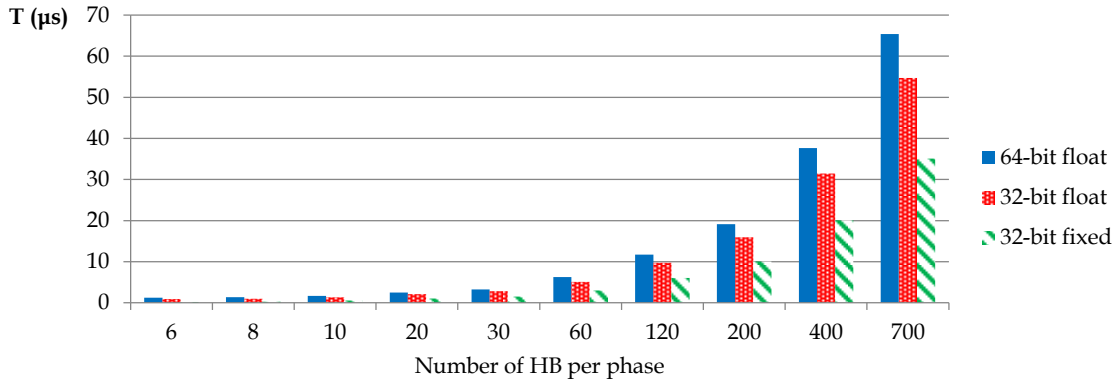


Figure 4.7: Execution time comparison

4.7.3 *Precision*

The correct functionality of the model was verified by feeding the IP block with offline simulation data generated using a detailed model in PSCAD. The number of cells chosen for this comparison was eight per phase (hence four per arm) and the simulation was run for half a second using a time step of $5\mu s$. Even though a shorter time step could be chosen according to Table 4.1, if the IP would be used in a real application, the management of input and output variables should be considered. This kind of study has been carried out in Subsection 3.7 and in [7]. The converter's output voltage v_a of the PSCAD simulation was compared with the HLS implementations for the three data formats. The equation used for obtaining the absolute error is shown in Table 4.2 is 4.21.

$$error_{abs} = \frac{\sum_{i=1}^{n} \left(v_{a_{PSCAD}}(i) - v_{a_{zynq}}(i) \right)}{n} \times 100 \tag{4.21}$$

being n the number of simulation steps.

Table 4.2: Precision results for v_a

	64-bit float	32-bit float	32-bit fixed
Relative error	0.404%	0.437%	3.218%

4.8 CHAPTER CONCLUSIONS

In this section was presented an evaluation of the *Zynq-7045* to perform eRTS of MMCs. An improved average model which keeps record of the SMs capacitor voltages was utilised. An IP block containing the power converter equations of a single phase was programmed where the circuit parameters, the number of HBs and the format of the variables can be changed easily.

There were compared three different data formats: 64- and 32-bit floating-point, and 32-bit fixed-point. Moreover, the number of HB cells per phase was changed by 6, 8, 10, 20, 30, 60, 120, 200, 400, and 700 SMs. They were analyzed in terms of resources usage, execution time, and precision compared with a PSCAD simulation.

With regard to the area utilised, both floating-point versions reached 30% of the resources when computing 700 HB per phase. However, the fixed-point version ran out of DSP units at 200 HB per phase.

Regarding the execution time, bigger differences were observed when the number of cells was low. For the case of 6 HBs per phase, the 32-bit version is below $1\mu s$ whereas the 64-bit is at $1.25\mu s$. But for the same number of cells, the fixed-point version was 5 times faster than its 64-bit floating-point counterpart.

The error evaluation was performed comparing the results for an IP of 8 HBs per phase with a $0.5s$ PSCAD simulation using $5\mu s$ as time-step. The absolute error achieved was around 0.4% for the two floating-point versions, whereas for the fixed-point version was a bit above 3%.

It has to be highlighted that when using HLS tools, changing the width and format of the data was quite straightforward. The tool dealt with all the hardware manipulations automatically. The choice will be made depending on the number of cells of the converter, the precision of the results required, and the FPGA area available in order to best emulate the behavior of the real system.

Part III

EXPERIMENTAL VALIDATION

5

APPLICATION OF AN ERTS IN AN EXPERIMENTAL PROTOTYPE

5.1 INTRODUCTION AND OBJECTIVES

The purpose of this Chapter is to explain the development and experimental validation of an eRTS for Modular Multi-level power electronic Converters. As previously mentioned, particular attention has been paid to the use of HLS for eRTS SoC implementation. For this purpose, a complete low-power test rig was designed, built and put in operation –see Appendix A for a detailed description of the test bench. The developed eRTS ecosystem was validated in a single phase MMC based on 3 HBs per arm. Three test cases have been considered:

1. HLS implementation of both MMC eRTS models and their control
2. *Hardware-in-the-Loop* simulation: whereas the FPGA fabric simulates the power converter and generates the corresponding feedback signals, the control is carried out in the ARM
3. eRTS for fault-tolerant control, where the eRTS is used for cell voltage estimation. In case of cell voltage sensor failure, it is shown that the developed eRTS can provide adequate feedback signals for the converter to operate safely after the fault. The experimental results, their applications and limitations will be thoroughly discussed.

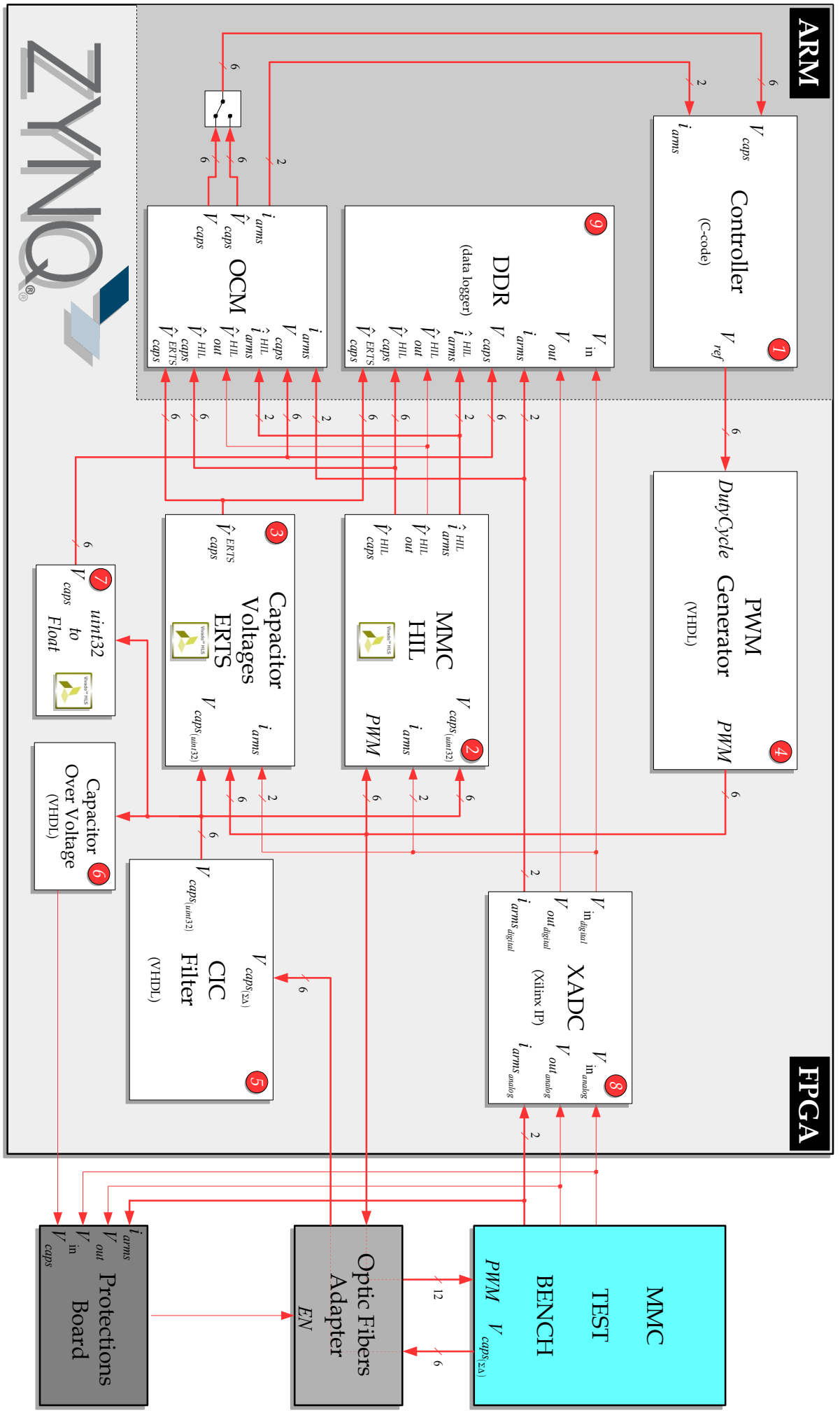
5.2 SOFTWARE/HARDWARE CO-DESIGN DESCRIPTION

A single-phase MMC converter based on 3 HBs per arm was used for eRTS experimental validation. This means three sub-modules forming the upper arm, and the remaining three forming the lower arm.

The different subsystems necessary to control the test bench were classified into three groups, according to their increasing difficulty, in either (i) VHDL, (ii) HLS, or (iii) in software. Hence:

- (i) VHDL: the basic blocks with simple tasks that require the fastest execution time possible and will not be modified in the future
- (ii) HLS: the blocks with average complexity with only a few number of parameters and requiring the fastest possible execution time with great parallelization possibilities
- (iii) Software: the parts of the system which perform complicated tasks and calculations, with very few parallelization chances, and that will definitely change during development cycle

Accordingly, among the different ways of implementing all the blocks needed for a system of these characteristics –diagram shown in Figure 5.1, it was decided to partition the system as follows:


ARM
FPGA
ZYNQ
Controller
(C-code)
PWM Generator
(VHDL)
DDR
(data logger)
OCM
MMC HIL
Capacitor Voltages ERTS
uint32 to Float
Capacitor Over Voltage
(VHDL)
CIC Filter
(VHDL)
XADC
(Xilinx IP)
Optic Fibers Adapter
Protections Board
MMC TEST BENCH

1. *Master Controller* → ARM. The master controller is in charge of booting the system up, configuring all the peripherals and IP blocks needed, managing the interrupts to ensure the synchronization of all of them, verifying that every single block execute their tasks when they are required, perform part of the data logging, and last but not least executing the MMC control functions.

2. *MMC HIL* → FPGA. Considering versatility, scalability, and of course the most important parameter which is execution time, it was the best option to implement it in the fabric. Moreover, it needs to read the PWM signals coming from the *PWM Generator* as fast as possible. Therefore, it has to be directly connected by hardware to this block to read its values with the smallest possible delay.

3. *Capacitor Voltage eRTS* → FPGA, because of the same reasons of the previous block.

4. *PWM Generator* → FPGA. Its main function is to generate a PWM signal with enough resolution so the control commands are applied in the right moment. Hence, if the control is being executed at $100\mu s$, the comparator of the PWM has to run at least 100 times faster to have a minimum resolution of $1\mu s$ [35]. However, like the switching frequency could change and in order to make this block reusable in other projects, it was decided to make it fully configurable and implement it in PL using VHDL for achieving the fastest performance.

5. *CIC Filter* → FPGA. It converts the frequency-modulated capacitor voltages coming from the HBs to an integer value. It has been also coded in VHDL mainly because this filter was designed specifically for it and because it uses very little FPGA fabric [86].

6. *Capacitor Overvoltage Alarm* → FPGA. This simple block has to constantly check the capacitor voltages coming from the *CIC Filter* and issue an error if a capacitor has reached a predefined voltage. Consequently, the best option is to place it in the PL.

7. *CIC conversion to float* → FPGA. This block is in charge of reading the integer value produced by the *CIC Filter* and converting it to a 32-bit floating-point value in order to avoid the PS from doing it, thus alleviating some computational burden. This is what is known as hardware accelerator.

8. *XADC* → FPGA. This *Xilinx*'s Analog-to-Digital Converter IP is actually just for configuring the XADC peripheral.

9. *Data Logger* → ARM. This is more of a functionality than an actual functional block. All the system signals need to be registered for its later study. The variables read by the XADC (branch currents, and input and output voltages) will be logged using the CPU. However, the measured capacitor voltages, the *MMC HIL* results, and the capacitor voltages estimated by the eRTS will be placed automatically on the DDR memory by implementing *AXI4-Master* interfaces in the corresponding HLS IPs without any intervention of the PS.

The FPGA resource usage of the whole system can be seen in Table 5.1 and graphically in Figure 5.2. The *AXI Interconnect* (Zynq-to-IPs) and the *AXI SmartConnect* (IPs-to-Zynq) have been as well added for relevance. The latency of the HLS generated blocks considering *AXI4-Master* transactions is shown in Table 5.2. The complete block design is displayed in the following Section.

The next thing to be defined were the interfaces to operate the IPs and transfer data between them aiming always to achieve the fastest transfer time, which in the end means reducing the

Table 5.1: FPGA hardware resources utilisation

Resource type	BRAM	DSP	FF	LUT
Available	140	220	106.400	53.200
Zynq Processing System	0	0	44	230
MMC HIL	1	29	6.095	6.583
Capacitor voltage eRTS	1	60	12.003	9.454
PWM generator	0	0	2.302	1.581
CIC filter	0	0	5.957	2.782
CIC conversion to float	1	5	3.444	2.458
Capacitor overvoltage	0	0	473	395
XADC	0	0	235	160
AXI Interconnect	0	0	7.710	5.953
AXI SmartConnect	0	0	6.281	7.383
Complete design	3	94	44.654	36.417

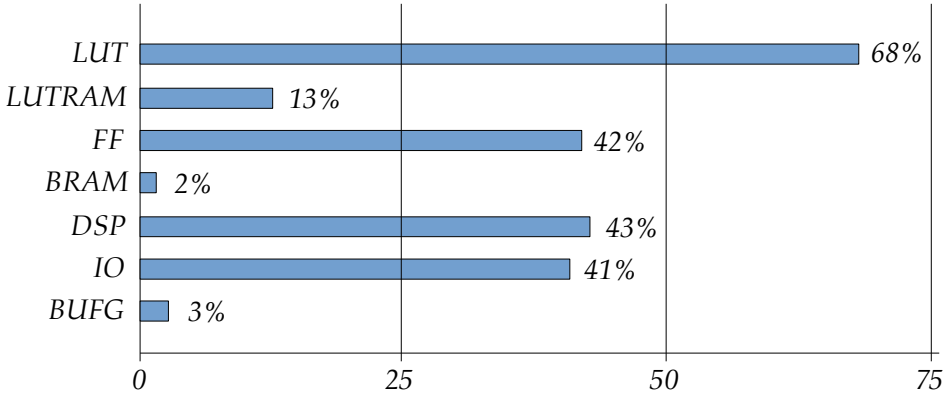


Figure 5.2: FPGA hardware resources utilisation (%)

Table 5.2: HLS IPs execution times using a $100MHz$ clock

	Latency	Execution time (in ns)
MMC HIL	88	880
Capacitor voltage eRTS	71	710
CIC conversion to float	32	320

overall execution time. Considering the amount of data to be transferred in this specific application, there are three kinds of interconnections that are used: *AXI4-Master, AXI4-Lite*, and simple wires (no protocol used). *AXI4-Stream* is not an option because it implies processing a steady flow of high speed data continuously removing memory addressing to reduce streaming buses overhead. So mainly connections between blocks implemented on the fabric will be done by simple wires, and then the operation of each IP and the data transfers between FPGA and ARM will be done using *AXI* interfaces.

Following, each part of the control system will be described.

5.2.1 *Zynq block design*

In Figure 5.3 the Vivado block design of the complete system is displayed. There are all the blocks needed to control the MMC test bench plus the HIL and the eRTS IPs. It can be seen as well the *AXI* and *non-AXI* interconnections between blocks. Though, some of they are used just for debugging and could be removed before its final commissioning, reducing area utilisation and improving data rate due to reduction of data transfers on the *AXI Interconnect*. The *AXI* clock is running at $100MHz$ like all the other IPs.

When adding a new IP that uses an *AXI* connection, Vivado assigns a memory address automatically. Then, that address has to be used in the SDK in order to configure and access this peripheral. Apart from the interconnections between blocks, it is needed to define and assign the system inputs and outputs to the physical pins of the device. This is done using the *Xilinx Design Constraints* (XDC) file.

Once the block design is ready, an HDL wrapper has to be generated and the *bitstream* produced, which might take quite a while for such a big design. So when the *bitstream* has been successfully created, it is suggested to have a look to the timing report, which might cause design malfunctioning if the constraints have not been met due to clock uncertainties, power supply oscillations, exceeded temperature range, etc. Some problems were experienced related to this when the *AXI* clock was set to $250MHz$ trying to speed up data transmissions. In this case though, the *bitstream* was generated without any error. However, the timing report was showing Negative Slack, which essentially says that there are some paths which might not meet timing in harsh conditions. Xilinx strongly recommends to modify the design to remove the Negative Slack. In the end it was decided then to reduce the *AXI* clock to $100MHz$ in order to avoid any problem.

5.2.2 *IP descriptions and configurations*

In this section, all the IPs utilized in the test bench are presented and explained.

5.2.2.1 *Zynq Processing System*

The *MicroZed System-on-Module* (SOM) consists of a *Zynq-7020* which processor runs at $667MHz$ and its $1GB$ DDR3 at $533MHz$ [57]. Its Vivado block design IP is shown in Figure 5.4.

It was activated the UART1 and configured to work at $115.2kbps$, 8 data bits and 1 stop bits. It is used mostly to print out in console the state of the simulation and to output some values in order to verify the proper functioning of the system.

The single clock driving all the FPGA IP modules and the *AXI interconnect* was set to $100MHz$. It was tested for higher speeds but the implementation failed to meet timing requirements, so it was left to this value in order to avoid Negative Slack. The *General Purpose Port 0* (GP0) was enabled to manage all the *AXI4-based* peripherals present in the fabric.

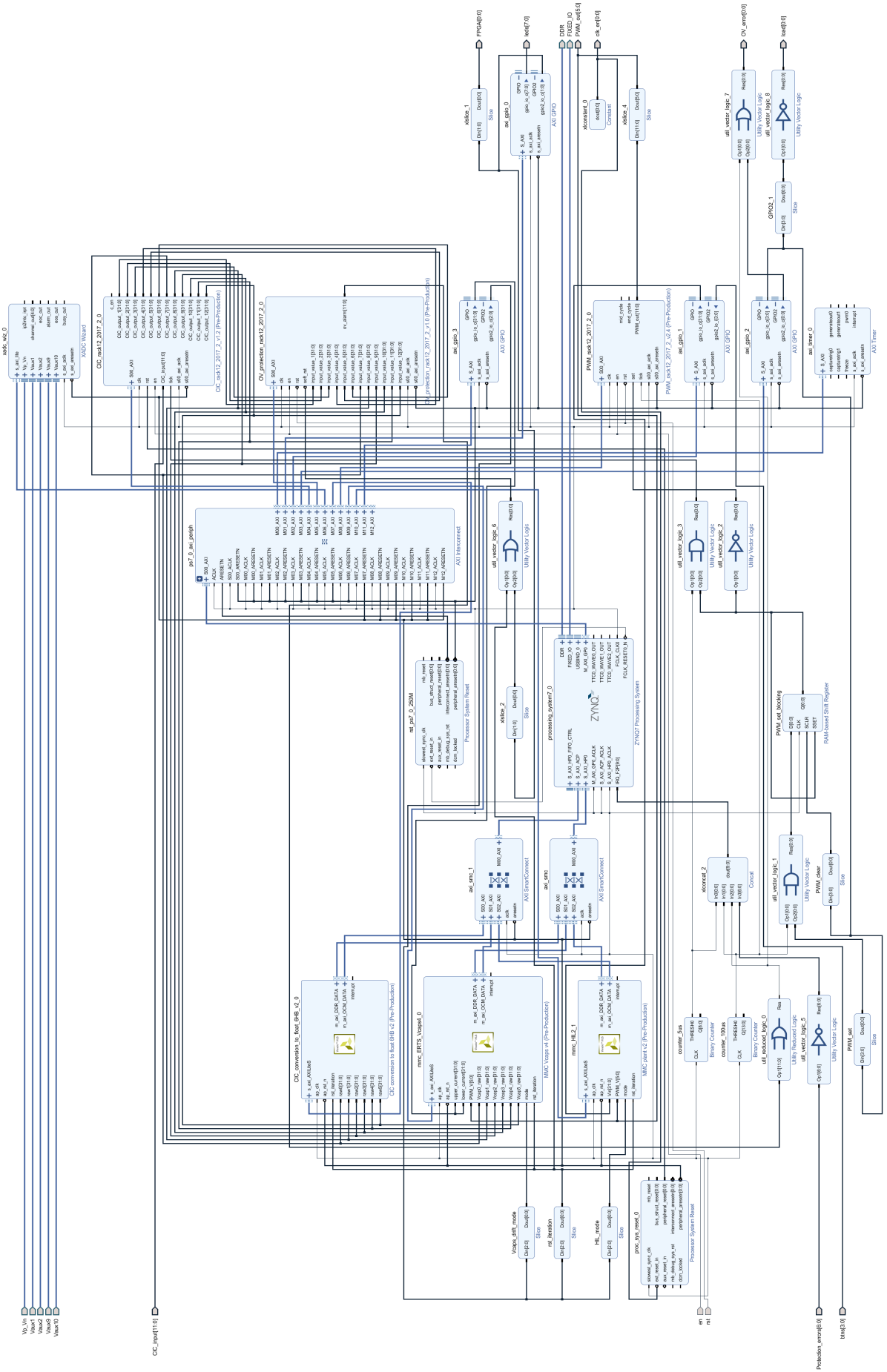


Figure 5.3: Vivado Block Design

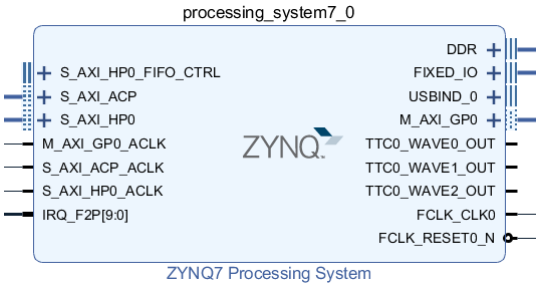


Figure 5.4: Zynq Processing System IP

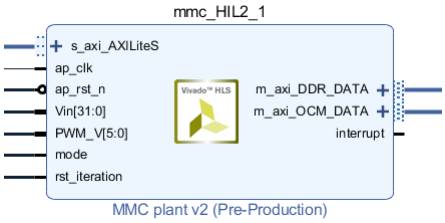


Figure 5.5: MMC HIL model IP

The 16-bit shared interrupt port, the *IRQ_F2P*, was as well enabled and connected to several hardware IPs. There are ten interrupt signals connected to it notwithstanding that not all of them are critical for the system to work properly. The only two which are essential are the $100\mu s$ and the $5\mu s$ interrupts. The $5\mu s$ interrupt is the main interrupt. In its routine it retrieves and logs the ADC measurements, and controls the HIL and eRTS IPs. Then, the $100\mu s$ interrupt is used to synchronize the control with the PWM generation. The other interrupts are used just for letting know the ARM that an error has occurred because they are managed inside the FPGA without its direct intervention. This is the case of the *Capacitor Overvoltage*, the emergency button, and the three overvoltages and overcurrents coming from the protection card.

It was also enabled the *Accelerator Coherency Port* (ACP) to allow the *CIC conversion to float*, the *MMC HIL* and the *Capacitor voltage eRTS* IPs to register their results autonomously in the OCM memory using *AXI4-Master* ports. In this way, the CPU is able to read these values directly from the OCM keeping coherency with the *L1* caches, which is in the end a lot quicker than reading them from the DDR through the *High-Performance Port* (HP) [87]. Nevertheless, the HP port was enabled as well and connected to a second *AXI4-Master* port present in all three IPs in order to store all the iteration results by increasing the writing address automatically on each iteration, again without any intervention of the ARM.

5.2.2.2 *MMC HIL*

Besides the *Capacitor voltage eRTS*, the other IP –shown in Figure 5.5– is as well a device under test and the other main objective of this dissertation. Roughly speaking, it is an IP module that emulates the MMC behavior in real-time and has many different applications: from *sensorless* and fault-tolerant control, to *Hardware-in-the-Loop* (HIL) testing, or diagnostic and fault detection among others. Due to its complexity and to allow easier modification of the converter parameters, it was coded using HLS instead of VHDL. The application involving this IP is explained in Section 5.3.3.

This block reads the DC input voltage fed to the converter and the PWM signals and computes all the HB capacitor voltages, the currents of the upper and lower arms, and the output voltage in the middle connection point of the phase leg.

In order to reduce the data transfer time at minimum, the PWM input signals were configured to use *ap_none* instead of any *AXI* or BRAM protocol. This way, the block was getting the PWM signals directly from the *PWM Generator*. Regarding the input DC voltage, it was used as well an *ap_none* port which value was fed by the PS through an *AXI4-GPIO*. Besides the *AXI4* interfaces, the other inputs present in the IP shown in 5.5 are used to change the functioning mode of the IP from *charging mode* (no load is present) to *normal operation mode* (a 44Ω resistor is connected to the middle point of the leg), and to reset the internal variables if a new simulation is performed by using *rst_iteration*. As explained earlier, the results are automatically registered into two reserved spaces, one in the OCM for a faster access from the PS, and another one into the DDR memory for data logging purposes using two *AXI4-Master* interfaces: *m_axi_OCM_DATA* and *m_axi_DDR_DATA* respectively.

Regarding the *pragmas* used in this IP, only a few were employed:

- `ARRAY_PARTITION complete dim=1`: in order to avoid the use of BRAMs for the A, B, and C matrices
- `INTERFACE s_axilite`: applied to the `return` port in order to make the IP controllable through *AXI4-Lite*. Vivado HLS automatically generates the drivers with all the functions necessary for this purpose
- `INTERFACE m_axi offset=slave bundle=DDR_DATA`: for the DDR and OCM output ports, using different *bundles* to force implementing two *AXI4-Master* ports. This way, one will be connected to the DDR memory through the HP port for data logging, and the other to the OCM through the ACP to maintain cache coherency with the PS and achieve the lowest latency access
- `INTERFACE ap_none register`: used in the *Vin*, *PWM*, *mode* and *rst_iteration* ports to infer simple wire connections adding an input register

This block performs 13 multiplications, 2 divisions, 18 additions, and 1 subtraction in 88 cycles which correspond to $880ns$ with a running clock of $100MHz$. Regarding area utilisation, 1 BRAMs, 29 DSPs, 6.095 FF, and 6.583 LUT were needed, which correspond to 1%, 13%, 6% and 12% of the total hardware resources of the *Zynq-7020*.

The time constant, capacitance values, L_{arm}, R_{on}, and load value are hardcoded in the IP in order to reduce computations and improve execution time.

5.2.2.3 *Capacitor voltage eRTS*

This, together with the *MMC HIL* described in the former Subsection, is the other device under test and the other main objective of this dissertation. This IP is used to estimate the capacitor voltages just from the currents and PWM signals. The raw –integer– capacitor voltages provided by the *CIC filter* IP are utilised to estimate the current offset and compensate the capacitor voltage drift. This IP is used in a real-case scenario of fault-tolerant control in Section 5.3.5. The controller utilizes the capacitor voltages estimated by this block when losing a capacitor voltage measurement in order to continue operating the converter.

Figure 5.6 shows the resulting IP where voltages, currents and PWMs are fed into the block using *ap_none* ports. The capacitor voltages and the PWM signals are connected directly to the *CIC filter* and *PWM Generator* IP blocks respectively, thus retrieving the values without

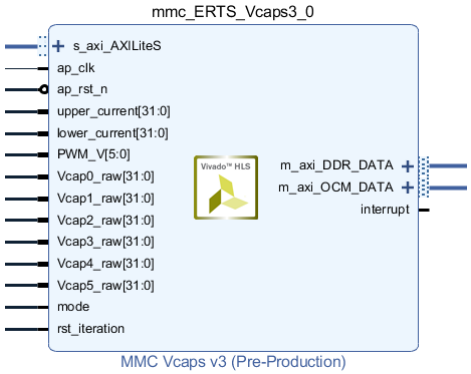


Figure 5.6: Capacitor voltage eRTS IP

intervention of the PS. The currents however are sent to the IP by the processor after being read from the ADC using two generic *AXI4-GPIO* ports. Like in the *MMC HIL* case, the results are automatically registered into the OCM and DDR memories.

These are the *pragmas* used in this IP:

- `INTERFACE s_axilite`: applied to the *return* port in order to make the IP controllable through *AXI4-Lite*, and also to the *a* and *b* parameters of the ramp function used to calibrate and convert the integer values generated by the *CIC filter* into floating-point values.
- `INTERFACE m_axi offset=slave bundle=DDR_DATA`: for the DDR and OCM output ports, using different *bundles* to force implementing two *AXI4-Master* ports similarly to the previous IP
- `INTERFACE ap_none register`: used in the *upper_current*, *lower_current*, *PWM*, *VcapX_raw*, *mode* and *rst_iteration* ports to infer simple wire connections adding an input register

This block performs 36 multiplications, 36 additions, and 24 subtraction in 71 cycles. Clocking the entity with a $100MHz$ oscillator, the latency needed to compute all the six voltages is 71 ($710ns$). It uses 1 BRAMs, 60 DSPs, 12.003 FF, and 9.454 LUT, which corresponds to 1%, 27%, 11% and 18% of the total resources of the *Zynq-7020*.

5.2.2.4 *PWM generator*

This block coded in VHDL receives the duty cycle to be applied to each cell by an *AXI4-Lite* interface and then generates the PWM signals at the desired switching frequency. It has four configuration registers per HB that are modified as well via *AXI4-Lite* which correspond to: the phase angle delay, the frequency, the duty cycle and the minimum pulse width. This single block is able to manage a complete rack at once, i.e. 12 HB cells.

The control of this IP is made through four scalar signals: *en*, *rst*, *set*, and *tick*:

- *en*: The enable signal. If this signal is low, the PWM outputs are fixed to one, which equals to switch the capacitors in (i.e. the output voltage of each SM is V_{cap}) to avoid a short-circuit of the input DC voltage if any.
- *rst*: The reset signal. It is active high, so if this signal is asserted, all the configuration registers are set to their predefined values.

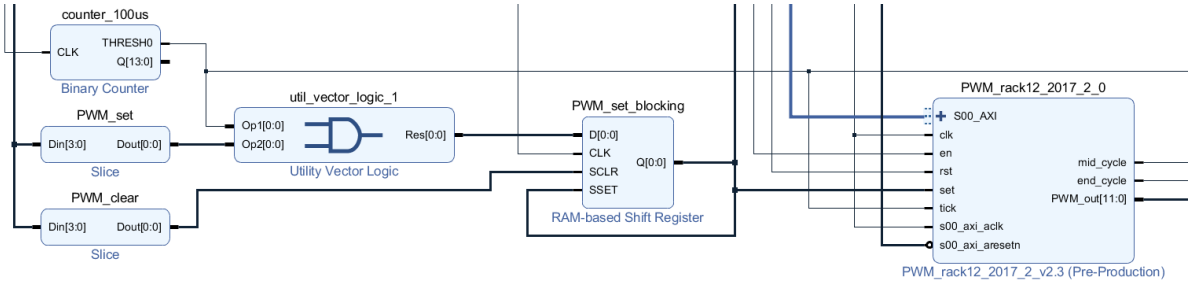


Figure 5.7: PWM Generator IP

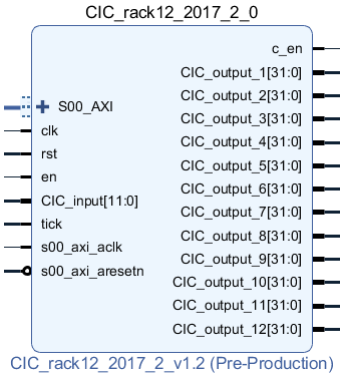


Figure 5.8: CIC filter IP

- *set*: By setting this signal to logical 1, the counters (there is one per SM) start counting and do not stop until either the enable or reset signals are activated. It is used to synchronize the PWM signals with the MMC control function.
- *tick*: This signal is used to make valid the duty cycle values stored in the registers. So it is supposed to send a pulse to this signal once the ARM has written the control references for the next control period. The values are updated in the registers but will not be applied until the counter reaches either zero or the maximum count, i.e. the bottom and the top of the triangular waveform respectively.

The operation of this block is quite straightforward. In the initialization, phase and frequency registers are configured. Then, the *en* and *set* signals are both asserted by the ARM using an *AXI4-GPIO* port. In this way and thanks to the additional hardware seen in Figure 5.7, the *set* signal gets synchronized with the control executed every $100\mu s$.

5.2.2.5 *CIC filter*

As explained in Appendix A, the VHDL block shown in Figure 5.8 performs a *Cascaded Integrator-Comb* (CIC) filtering of the frequency-modulated signals coming from the $\Sigma\Delta$ ADCs present in the HBs. A CIC is *Finite Impulse Response* (FIR) filter whose principal advantage is that it does not need multipliers. It is solely composed of adders and registers. This implies an important benefit in terms of FPGA area usage. Moreover, it can be used either for decimation as for interpolation [86].

It is characterized by the next parameters:

- N: number of stages
- M: delay of the comb stage

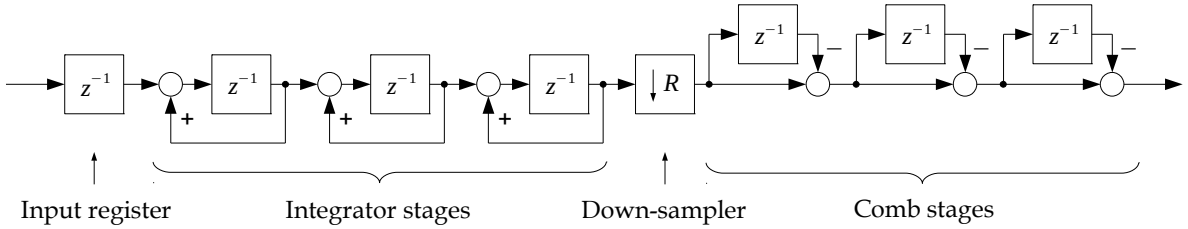


Figure 5.9: CIC filter structure for decimation ($N = 3$, $M = 1$)

- R: decimation/interpolation factor

According to these parameters, the filter needs a minimum register size of B_{out} to provide valid results, and will have a gain G determined by the next equations:

$$\begin{aligned} B_{out} &= N \log_2 (RM) + B_{in} \\ G &= (RM)^N \end{aligned}$$

Being B_{out} the size of the number of internal and external bits, and B_{in} the size of the number of bits of the input signal. So according to these equations, the maximum value the output filter can produce is G. Hence, dividing the result of the filter by G will give a value between 0 and 1.

The filter implementation of this block used in this Thesis work is a 3 stages decimation. Its structure is shown in Figure 5.9.

Although it can control up to 12 sub-modules, it only has one configuration register where the R parameter –the number of clock pulses the filter will be integrating– is set fixing the periodicity of a valid output.

This block is controlled by three inputs: *en*, *rst*, and *tick*.

- *en*: When this signal is set high, the filter starts integrating. It will provide a valid result once it has reached R clock cycles.
- *rst*: Active high, this input sets the R register to zero and *c_en* to logical 1.
- *tick*: By sending a pulse to this input, the results of every filter are transferred to their corresponding *CIC_output_X[31:0]* output registers and are kept constant until the next rising edge arrives.
- *CIC_input[11:0]*: These are the 12 frequency-modulated binary signals coming from the ΣΔ ADCs present on each HB. The ΣΔ module operates at $10MHz$ bit rate.
- *c_en*: Conversion finished or new data is available. This signal outputs a pulse every R clock cycles once the *en* signal has gone high.
- *CIC_output_x[31:0]*: A 32-bit integer value corresponding to the number of high pulses the filter has integrated. This *CIC Filter* has been configured with three stages, which means that the maximum value it can have is G. This means that if every clock cycle the filter finds *CIC_input[x]* high, the output will be equal to G. Conversely, if the filter finds every clock cycle *CIC_input[x]* low, its output will be 0. Consequently, if the number of pulses counted by the filter is even, its output value will be $G/2$.

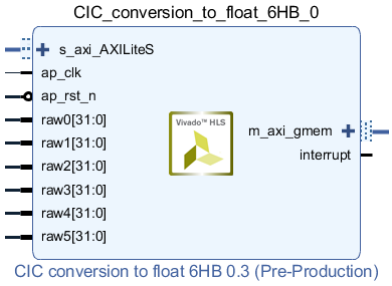


Figure 5.10: CIC converter to float IP

It has to be said that if the R value is relatively low, the results will have a high noise component. This IP is actually a low-pass filter and the R parameter is inversely proportional to the cutting frequency. Reasonably good results (i.e. with not so much noise) were found when the integration parameter was bigger than 200.

5.2.2.6 *CIC conversion to float*

This block, displayed in Figure 5.10 and also generated using HLS, reads the capacitor voltages in raw (integer), converts them to floating-point, and stores them in the DDR memory without any intervention of the processor.

The conversion from the integer value to floating-point format is done using equation 5.1, where a and b are the *ramp* and *offset* values of the ramp function respectively, and R is the CIC factor. The ramp function parameters are calibrated for every HB and configured in the initialization using an *AXI4-Lite* connection. Then, the block converts from integer to floating-point each time there is a value produced by the *CIC Filter* and stores it automatically in the DDR memory in the address previously specified through *AXI4-Lite* as well.

$$V_{cap_{float}} = \frac{V_{cap_{int}} a}{R^3} - b \tag{5.1}$$

The directives used in this IP are the following:

- `INTERFACE s_axilite`: similarly to the *Capacitor voltage eRTS*, this directive was applied to the *return* port in order to make the IP controllable through *AXI4-Lite*, and also to the a and b parameters of the ramp function used to calibrate and convert the integer values generated by the *CIC filter* into floating-point values.
- `INTERFACE m_axi offset=slave bundle=DDR_DATA`: for the DDR and OCM output ports, using different *bundles* as with the other HLS IPs to force implementing two *AXI4-Master* ports
- `INTERFACE ap_none register`: used in the *VcapX_raw* and *rst_iteration* ports to infer simple wire connections with an input register

Regarding resources and execution time, this IP uses 1 BRAMs, 5 DSP units, 3.444 FFs, and 2.458 LUTs which correspond to 1%, 2%, 3% and 5% respectively. With respect to the latency, it needs 32 clock cycles to generate valid results, thus 320ns.

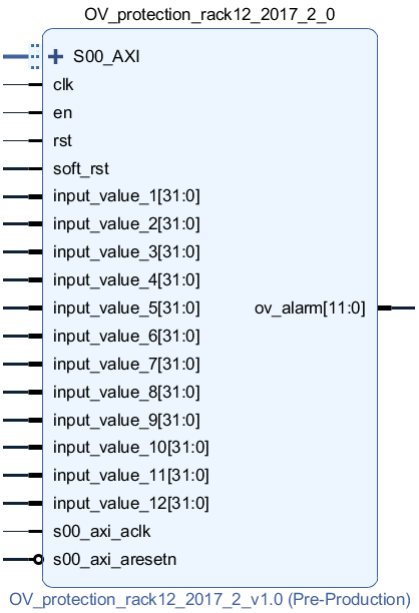


Figure 5.11: Capacitor overvoltage alarm IP

5.2.2.7 *Capacitor Overvoltage alarm*

This block coded in VHDL and shown in Figure 5.11 is a simple comparator between the *CIC Filter* output and a predefined value. There are 12 registers, one for each SM, which are modified using the *AXI4-Lite* protocol. The value has to be entered in integer because it is much faster to compare an integer value than a floating-point one. Accordingly, this voltage has to be converted to integer using equation 5.2.

This block has three control signals: *en*, *rst*, and *soft_rst*.

- *en*: Active high, this signal allows the comparison to be performed each rising edge of the clock signal.
- *rst*: Also active high, this signal resets the comparison registers to its maximum value, *G*.
- *soft_rst*: This signal sets the alarms to zero, so if any of the input values has surpassed the comparison value, it resets the output to compare it again.

$$V_{cap_{int}} = \frac{V_{cap_{float}} + b}{a} G \tag{5.2}$$

5.2.2.8 *XADC*

The *Zynq* ADC has been configured using the XADC wizard to read two voltages and two currents. It has a double channel, 1*MSPS* ADC which is capable of performing simultaneous samplings on two channels. Accordingly, channels 1 and 2 of ADC A, and channels 9 and 10 of ADC B were enabled and configured to perform measurements simultaneously of the input and output voltages (channel 1 and channel 9 respectively), and upper and lower branch currents (channel 2 and 10 respectively). The results of the measured signals are retrieved through an *AXI4-Lite* interface and stored in the DDR memory, serving for the control of the converter and as data logging.

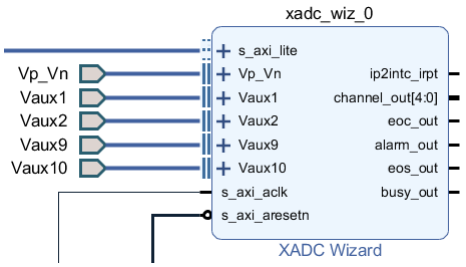


Figure 5.12: XADC IP

Several functioning modes were tested, like performing measurements using an external pin to be completely synchronized with the capacitor voltage measurements, but the conversion time was oddly non-constant. Moreover, it only performed one measurement on two channels at a time and hence, it was decided to be activated using the *AXI4-Lite* interface. Measurements were not exactly synchronized with the capacitor voltages but the limitations of this ADC forced this situation. The maximum delay measured experimentally was not bigger than $1\mu s$ in the worst case.

The configuration in the XADC wizard is as follows:

- *DCLK frequency*: $100MHz$
- *ADC conversion rate*: $1MSPS$
- *Interface options*: *AXI4-Lite*
- *Timing mode*: Continuous mode
- *Startup channel selection*: Channel sequencer
- *Sequencer mode*: One pass
- *Channel averaging*: none
- *ADC calibration*: ADC Offset Calibration and ADC offset and Gain calibration checked
- *Supply sensor calibration*: Sensor Offset Calibration and Sensor Offset and Gain Calibration checked
- *External multiplexer setup*: disabled
- *Alarms*: all disabled
- *Channels enabled*: 1, 2, 9 and 10

5.2.3 *The C code*

In order to make these blocks work on real-time, it was necessary to know, develop, interconnect and configure appropriately each of the modules that form the system. Moreover, it was mandatory to make them perform their tasks in the right moment and in a predefined amount of time. Hence, to develop good hardware is as important as to code good software to manage it correctly.

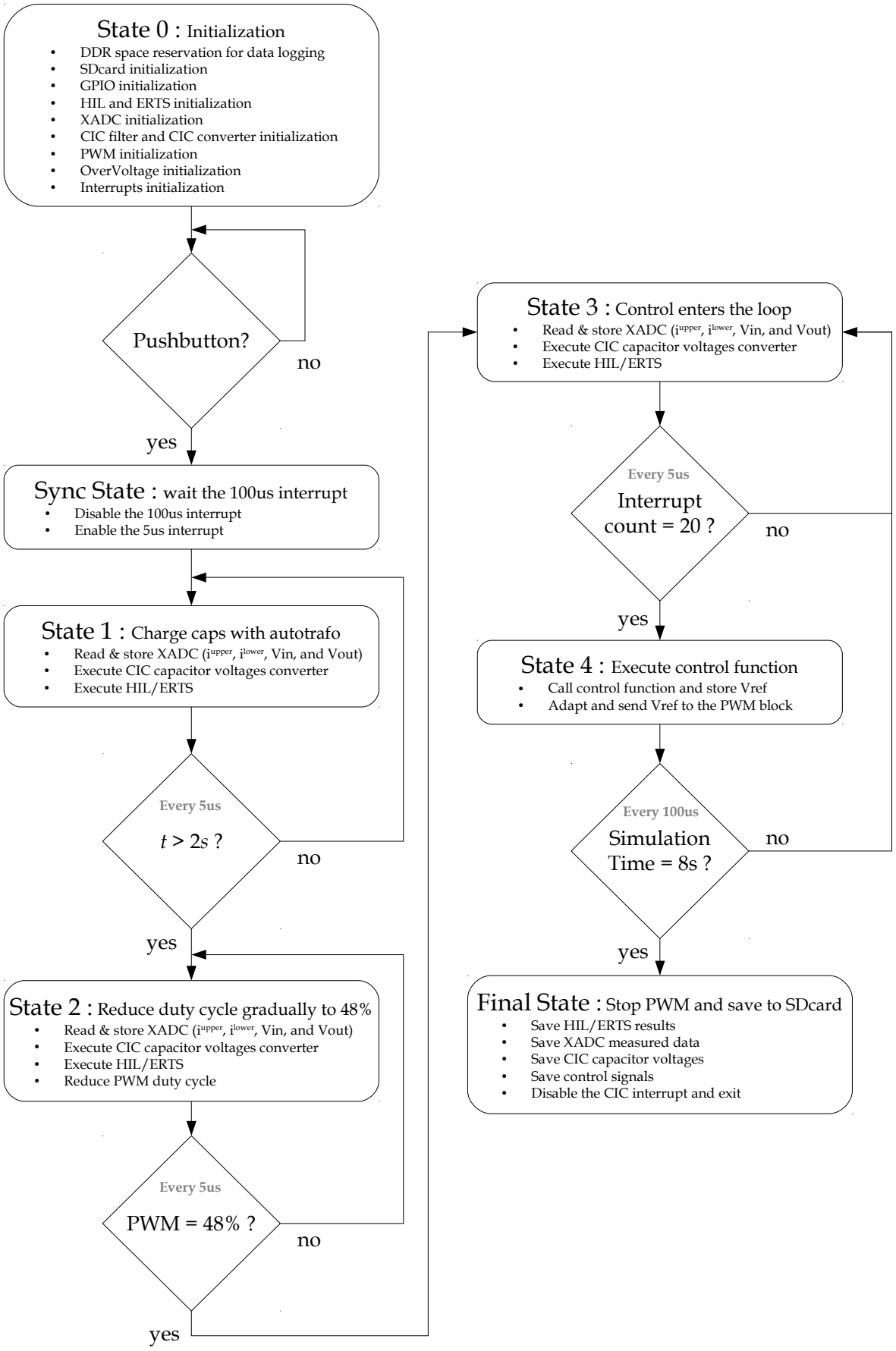


Figure 5.13: State Machine diagram including software initialization, MMC capacitor charging, MMC normal operation and data retrieval

5.2.3.1 *Structure*

A diagram of the code structure is shown in Figure 5.13. There can be seen the 7 stages of the *Finite State Machine* (FSM) which are explained following:

1. *State 0*: The initialization stage. The software reserves enough DDR space to register all the signals that want to be registered, executes the initialization of all the IPs and then waits the push-button to be pressed to start.
   This version of *MicroZed* board has $1GB$ of memory. If the simulation is set to 8 seconds and the eRTS execution time to $5\mu s$, this means that for every variable that has to be stored in the DDR it must be reserved a vector of 1.6 million seats of 32-bits, which corresponds that each signal will need $6.1MB$. Therefore, if measured signals V_{in}, i_s, i_d, V_{out} and V_{caps} [6] need to be stored, plus the HIL and eRTS estimated signals $\hat{i}_s$, $\hat{i}_d$, $\hat{V}_{out}$, $\hat{V}_{caps}^{HIL}$ [6] and $\hat{V}_{caps}^{ERTS}$ [6], this makes a total of nearly $153MB$, plus the control signals which are sampled at a lower rate (20 times slower) adds a bit less than $2MB$. So there is enough room for running quite long simulations without utilizing external storage.

2. *Sync State*: This is a temporary state that has to be included in order to synchronize the control function with the PWM.
   There are two main interrupts configured in the ARM: one is the PWM interrupt connected to a $100\mu s$ counter, and the other one is main interrupt which is executed every $5\mu s$. In order to synchronize the execution of the control with the PWM, both being called at $100\mu s$, once the push-button has been pressed, the PWM interrupt is activated by software. Hence, when the next $100\mu s$ pulse comes the PWM counter starts, then it enters the PWM interrupt routine, disables it, enables the $5\mu s$ CIC interrupt and exits. Straightaway, it enters into the CIC interrupt routine (*State 1*) and the simulation begins. What is accomplished with this is to have the execution of the control, which is not executed until *State 3*, to be synchronized with the PWM counter, which is necessary for the system for several reasons. One is to avoid possible instabilities of the control commands due to a big delay between its calculation and when are they actually applied, and the second is to have results coherent with the PSCAD simulation in order to compare them, which executes the steps following a similar procedure.

3. *State 1*: The simulation starts. When the system stays in this state, the PWM signals are kept to 100%, i.e. all the capacitors are switched in so when the voltage is increased using the autotransformer, the total input voltage is divided among the six SMs. In this state and all the following ones except the final state, the software executes always and every $5\mu s$ these two tasks: Read and store in the DDR the ADC measurements (i_s, i_d, V_{in} and V_{out}), execute the HIL and/or the eRTS using the ADC read value V_{in}, and exit. Once a significant amount of input voltage has been reached ($311V$) and the capacitors are charged to more or less $50V$, the software moves to the next state. This exact moment occurs at $2s$ as shown in Figure 5.14.

4. *State 2*: Once the capacitors had been charged to V_{in}/N, being N the number of cells in one leg, the duty cycle is decreased in order to raise their voltages. The maximum input voltage is $311V$, that is the reason why if the capacitors need to be charged to $100V$, the duty cycle has to be set to 48%. This ramp has been programmed to last 1.5 seconds in order to not have an excessive amount of current passing through switches and inductances. This stage is observed in Figure 5.14 from $2s$ to $3.5s$. Once the duty cycle has reached the specified value, it moves to the next state.

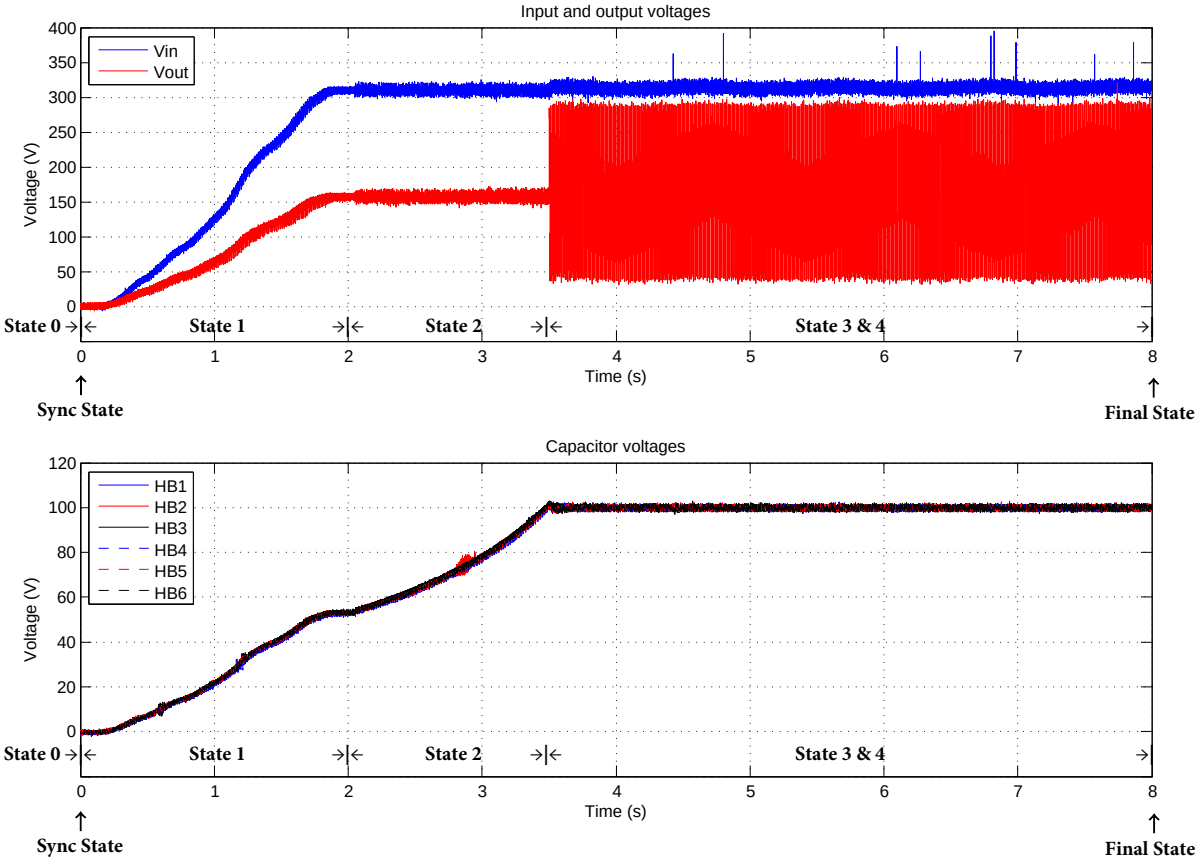


Figure 5.14: Evolution of the input, output and capacitor voltages through the execution states

5. *State 3*: As stated before, it executes all tasks of *State 1* and increments a counter so when it reaches 20 (which corresponds to 100μs) it passes to the next state.

6. *State 4*: The control enters the loop. In this state, the control function gets executed, then it stores the control commands in the DDR, adjusts the voltage references to the corresponding duty cycle, and sends the values to the PWM block through *AXI4-Lite*. When it finishes, it verifies that the simulation has not reached 8 seconds, and if not it moves back to *State 3*. Otherwise, it switches to the *Final State*.

7. *Final State*: In this state only is left to disable the CIC interrupt and dump all the simulation data stored in the DDR to the SDcard for its later analysis.

Figure 5.14 shows how the input, output and capacitor voltages evolve along all the explained states.

5.2.3.2 *Configuration*

The three different groups of blocks have very diverse ways of being configured. The configuration of the VHDL blocks is done by writing on some registers. Like those parameters might change depending on the test to be performed, it was not possible of fixing them directly on fabric. So the best solution was to wrap these blocks using *Vivado IP Packager* which creates an IP block from the VHDL code letting to set these registers using *AXI4-Lite* connections. In this way, the new IP is easily configured by software using an *AXI Interconnect*. All the wrapping

is done automatically by Vivado which implements the *AXI4-Lite* protocol and creates a C driver to write to and read from those VHDL registers.

For example, the PWM driver defines the registers present in the VHDL code using names easy to understand, then creates the basic functions to read to and write from those registers:

```
#define PWM_RACK12_mWriteReg(BaseAddress, RegOffset, Data) \ Xil_Out32((BaseAddress) + (
    RegOffset), (u32)(Data))
#define PWM_RACK12_mReadReg(BaseAddress, RegOffset) \ Xil_In32((BaseAddress) + (
    RegOffset))
```

Moreover, Vivado allows to create user specific functions to be included in the driver. For example, the next function takes the switching frequency and the clock and calculates the period of the PWM triangular waveforms:

```
u32 PWM_rack12_SetFrequency(u32 PWM_rack12_address, u32 PWM_rack12_reg, u32 Freq, u32
    PWM_rack12_clk){
        u32 N_Period = 0;
        N_Period = PWM_rack12_clk/(2*Freq);
        PWM_RACK12_mWriteReg(PWM_rack12_address, PWM_rack12_reg, N_Period);
    return N_Period;
}
```

Regarding the HLS generated IPs, the configuration is much more complicated. Using Vivado HLS basically means coding a C function which inputs and outputs are the arguments passed to it. Then, every of these arguments have to be assigned to a type of interface. In this design three different ports have been used: *ap_none*, *s_axilite*, and *m_axi*.

- *ap_none*: By applying this directive to a function argument, simple wires are inferred. These are the fastest interfaces and the most appropriate to interconnect VHDL IPs.
- *s_axilite*: Implements an *AXI4-Lite* interface to access the function arguments. Mainly used to configure and operate the IP. Vivado HLS creates all the logic necessary to implement the protocol and the driver functions to initialize and manage the IP block.
- *m_axi*: Implements an *AXI4-Master* interface performing all the data transactions autonomously. This port is used to write the data results directly to the OCM and DDR memory without any intervention of the CPU.

Taking as an example the *CIC conversion to float* block which has the three types of interface, the address Vivado has given to this IP in the block design can be found in the *xparameters.h* file and then it can be assigned to an easy-to-remember definition:

```
#define CIC_CONV_DEVICE_ID      XPAR_CIC_CONVERSION_TO_FLOAT_6HB_0_DEVICE_ID
```

Following, the next type definition needs to be created in order to handle the IP block:

```
XCic_conversion_to_float_6hb CICconv;
```

Next, the block has to be initialized by using this function created automatically by Vivado HLS:

```
int XCic_conversion_to_float_6hb_Initialize(XCic_conversion_to_float_6hb *InstancePtr, u
    16 DeviceId);
```

where the first argument and second arguments are the ones just created: `&CICconv` and `CIC_CONV_DEVICE_ID` respectively.

Table 5.3: MMC HIL Execution time per state

Operation	Time in μs
State 1 (HIL)	0,9
State 2 (PWM)	2,9
State 3 (HIL)	0,9
State 4 (control)	5,3

Now the IP is fully configured and ready to be used. Its operation is done through *AXI4-Lite* –thanks to have applied to the `return` variable an *s_axilite* interface directive, automatically inferring the following driver functions:

```
void XCic_conversion_to_float_6hb_Start(XCic_conversion_to_float_6hb *InstancePtr);
u32 XCic_conversion_to_float_6hb_IsDone(XCic_conversion_to_float_6hb *InstancePtr);
u32 XCic_conversion_to_float_6hb_IsIdle(XCic_conversion_to_float_6hb *InstancePtr);
u32 XCic_conversion_to_float_6hb_IsReady(XCic_conversion_to_float_6hb *InstancePtr);
void XCic_conversion_to_float_6hb_EnableAutoRestart(XCic_conversion_to_float_6hb *
    InstancePtr);
void XCic_conversion_to_float_6hb_DisableAutoRestart(XCic_conversion_to_float_6hb *
    InstancePtr);
```

And finally, each input parameter that has been decided to use an *AXI4-Lite* interface can be set using the next two functions (this is the case for input parameter `b0`):

```
void XCic_conversion_to_float_6hb_Set_b0(XCic_conversion_to_float_6hb *InstancePtr, u32
    Data);
u32 XCic_conversion_to_float_6hb_Get_a_b0(XCic_conversion_to_float_6hb *InstancePtr);
```

The last interface type this IP uses is the *AXI-Master*. As explained, it was decided to use it in order to alleviate the CPU from computing the integer to floating-point conversion of the *CIC Filter* results. Basically, what is needed to use this hardware accelerator is once all the ramp parameters and the DDR address have been configured, it has to be called the function `XCic_conversion_to_float_6hb_Start(&CICconv)`. 32 clock cycles after (320*ns* actually), the results will appear on the OCM without doing anything more. Then, the ARM just needs to read and use them when needed.

5.2.3.3 *Real-time implementation*

First of all, the execution periods of the control, the HIL and eRTS blocks had to be defined. As the $5kHz$ PWM is configured for double update [88], the control system period is set to $100\mu s$ ($10kHz$). A sampling time of $5\mu s$ for the *MMC HIL* was selected as a trade-off between the simulation accuracy and ease of implementation. Table 5.3 shows the time needed to perform all the tasks of every state.

However, the timing analysis was performed using *debug* mode, so these results might improve when choosing in the compiler the *release* mode.

5.3 EXPERIMENTAL RESULTS

This section presents the experimental results. The different Sub-sections are the following, they represent all the verification steps done during the design process:

1. *PSCAD MMC model:* first, a detailed model was developed and used as reference for the validation of the C model that will be used as eRTS in the SoC device
2. *HLS MMC model implementation:* the C model validated in PSCAD is moved to Vivado HLS. A C simulation is run in the same software tool in order to verify its proper functioning. A controller to manage branch energies, branch currents and to balance the capacitor voltages is coded and tested as well
3. *Control verification using the MMC IP as HIL:* the MMC model IP is generated and exported to Vivado. A block design is created and the controller is tested in real-time using the MMC model as a HIL platform
4. *Control verification with experimental prototype:* the test rig is put in operation controlled with the recently validated regulator
5. *eRTS for cell voltage estimation and fault-tolerant control:* a capacitor voltage eRTS is developed including a current offset estimator to compensate voltage drifts. Finally, it is tested in a scenario where the measurements of the capacitor voltages cannot be retrieved due to a induced fault

5.3.1 *PSCAD MMC model*

An MMC detailed model was developed in PSCAD and used as a reference model for the HLS MMC implementation. From an RTS developer's point of view one of the main important features of PSCAD is that it allows to program entities using FORTRAN and C. Therefore, this tool is very convenient in order to compare an accurate model using electrical elements with a coded implementation.

5.3.1.1 *PSCAD detailed model*

Figure 5.15 shows the PSCAD detailed model used as reference. It is composed of generic transistors and configured without the snubber circuit[1], a 10^{12} OFF resistor, a zero forward voltage drop, and an ON resistor equal to the one the test rig MOSFET's have, $33m\Omega$. The anti-parallel diodes are set to the same configuration parameters. Finally, the capacitors are set to $940\mu F$ and the branch inductances to $5mH$, same as the ones installed in the test rig.

5.3.1.2 *PSCAD C model*

The function that emulates the PSCAD detailed model is quite simple. It takes as inputs the total DC voltage V_{in} and the PWM signals, and from these it determines the currents through the upper and lower arms $i^{u,l}$, the capacitor voltages V_{caps_n} and the output voltage at the middle of the leg V_{out}.

The capacitor voltages are updated at each iteration using equation 5.3, where T_s is the time-step, and S_n is a logic function which depends on the corresponding PWM signal.

$$V^{u,l}_{caps_n}(k) = V^{u,l}_{caps_n}(k-1) + S_n(k-1)\frac{T_s}{C} i^{u,l}(k-1) \tag{5.3}$$

Then, using these results, the equivalent arm voltages are calculated using equation 5.4, where $N^{u,l}$ is the number of SMs per arm.

1 A snubber is a device used to suppress a phenomenon such as voltage transients in electrical systems, pressure transients in fluid systems or excess force or rapid movement in mechanical systems.

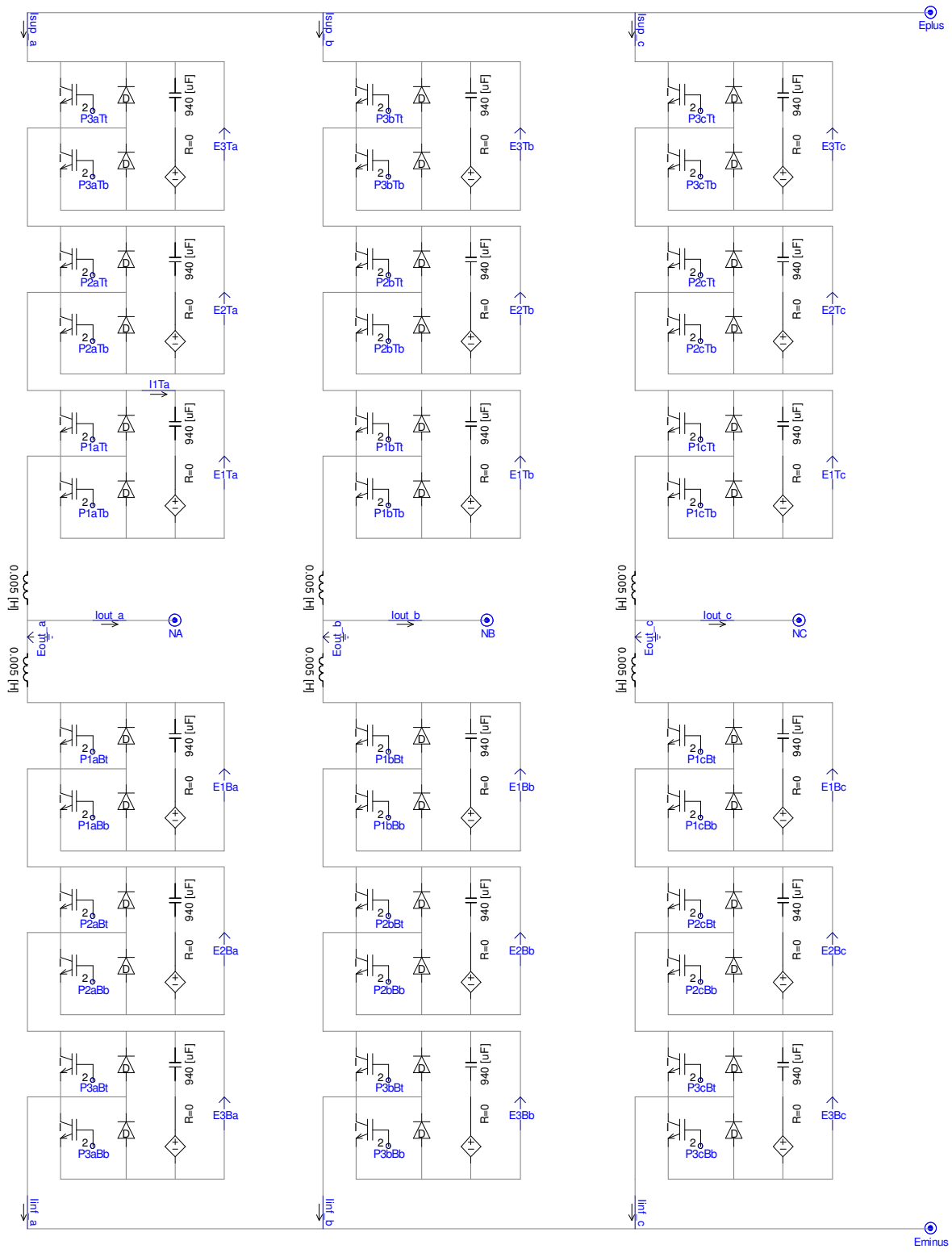


Figure 5.15: PSCAD detailed model of the MMC

$$V_{arm}^{u,l}(k) = \sum_{n=1}^{N^{u,l}} S_n(k)\, V_{caps_n}^{u,l}(k) \tag{5.4}$$

The method utilised to emulate the electrical circuit of the MMC is based on a state-space model discretized using the *c2d* MATLAB command in the ZOH context. Thus, all systems subjected to a simulation in state-space representation [75] are written as:

$$\begin{aligned} \dot{x}(t) &= \mathrm{A}x(t) + \mathrm{B}u(t) \\ y(t) &= \mathrm{C}x(t) + \mathrm{D}u(t) \end{aligned}$$

Hence, if the the total DC voltage V_{in} and the arm voltages $V_{arm}^{u,l}$ are taken as inputs, arranging the system equations following the form presented above leads to 5.5 and 5.6.

$$\begin{bmatrix} i^u(k) \\ i^l(k) \end{bmatrix} = \mathrm{A}\begin{bmatrix} i^u(k-1) \\ i^l(k-1) \end{bmatrix} + \mathrm{B}\begin{bmatrix} V_{in}(k) \\ V_{arm}^u(k) \\ V_{arm}^l(k) \end{bmatrix} \tag{5.5}$$

$$\begin{bmatrix} i^u(k) \\ i^l(k) \\ V_{out}(k) \end{bmatrix} = \mathrm{C}\begin{bmatrix} i^u(k) \\ i^l(k) \end{bmatrix} \tag{5.6}$$

where A, B, and C are:

$$\mathrm{A} = \begin{bmatrix} -\frac{R_{eq}+R_{load}}{L_{arm}} & \frac{R_{load}}{L_{arm}} \\ \frac{R_{load}}{L_{arm}} & -\frac{R_{eq}+R_{load}}{L_{arm}} \end{bmatrix}$$

$$\mathrm{B} = \begin{bmatrix} 1/L_{arm} & -1/L_{arm} & 0 \\ 0 & 0 & -1/L_{arm} \end{bmatrix}$$

$$\mathrm{C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ R_{load} & -R_{load} \end{bmatrix}$$

and L_{arm} the arm inductance, R_{load} the charge connected to the middle point, and R_{eq} the equivalent value considering the ON resistance of the MOSFETs and anti-parallel diodes plus the parasitic resistance of the arm inductors.

In order to alleviate computational burden, like both MOSFETs and anti-parallel diodes have similar ON resistances, the R_{eq} present in the A matrix has been set to the constant value:

$$R_{eq} = N^{u,l} R_{on} + R_L$$

where $N^{u,l}$ is the number of HB cells per branch, R_{on} an average value of the ON resistance of the power electronics, and R_L the parasitic resistance of the arm inductors. By doing this, the A matrix remains constant and therefore there is no need to calculate it every time an HB changes its state.

Now, applying the *c2d* MATLAB command for a $5\mu s$ time step, the parameters used for the model are issued and utilised in the C model function. These parameters can be seen in Appendix B.

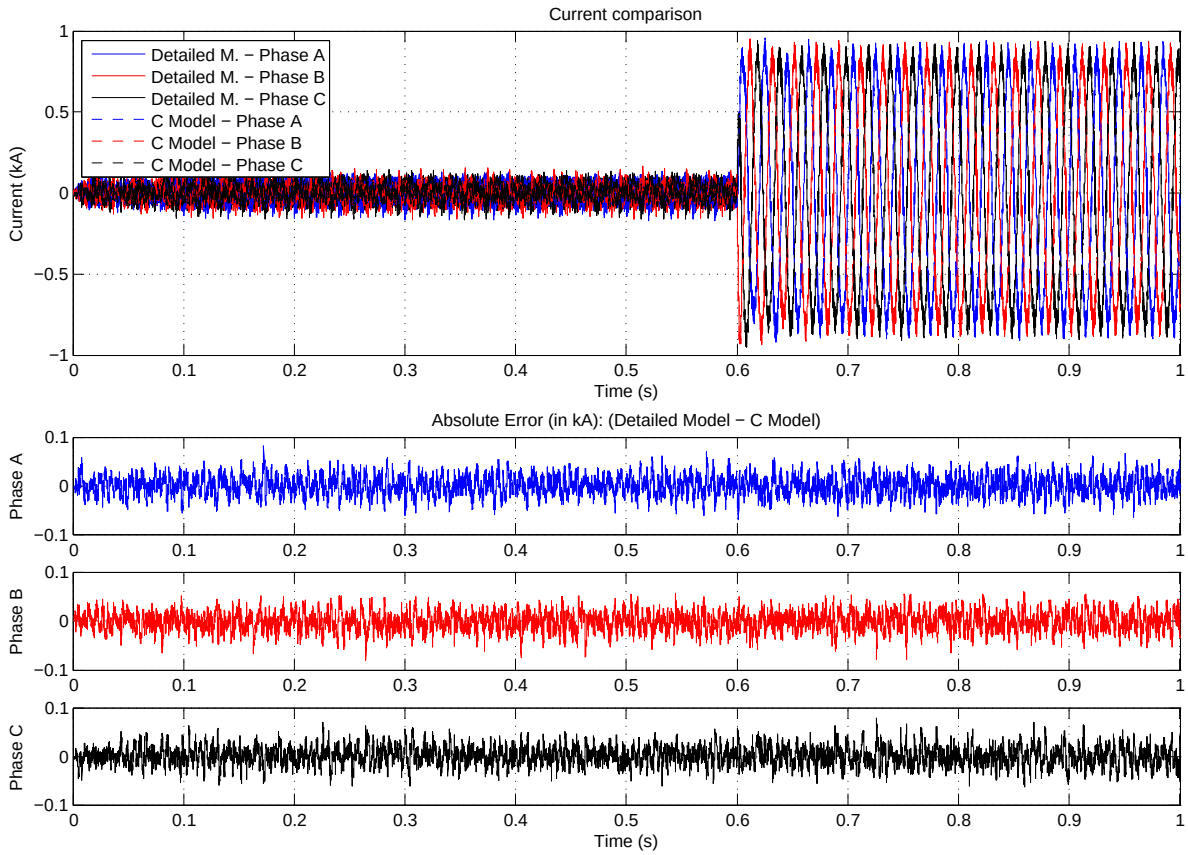


Figure 5.16: MMC model validation: Output currents

5.3.1.3 *PSCAD detailed model versus C model*

The simulation was run at $5\mu s$ time-step for $1s$ changing the active output power at $0.6s$. The reference of the capacitor voltages were set to $1.5kV$ instead of the $100V$ that will be used for the experimental test. However, this will not infer in the proper validation of the model.

Figures 5.16, 5.17, 5.18 and 5.19 display a $1s$ simulation of the output currents, output voltages, upper and lower branch capacitor voltages of both models respectively. At the bottom of each graph is presented the error between the detailed PSCAD model and the C model of each magnitude demonstrating that the error never exceeds 10%, regardless of the required power.

Figure 5.16 shows how the output current of the three-phase converter increases at $0.6s$ to provide the required output power. The error plots show that the error does not increase which imply that the C model follows the PSCAD detailed model dynamics correctly. The output voltages of Figure 5.17 do not show any disturbance throughout the whole simulation and as in the current comparison, the errors are kept stable.

Regarding the capacitor voltages shown in Figures 5.18 and 5.19, an increase in the ripple is clearly seen after augmenting the converter's output power. Here, however, the errors increase slightly after $0.6s$, but conversely to the previous signals, they are much smaller and never exceeding the 3%.

Figures 5.20 and 5.21 show an enlarged view of the previous graphs right in the moment when the output power increases. It can be seen that the C and PSCAD detailed models evolve following the same dynamics and that the error of both signals do not increase and are always swinging around zero.

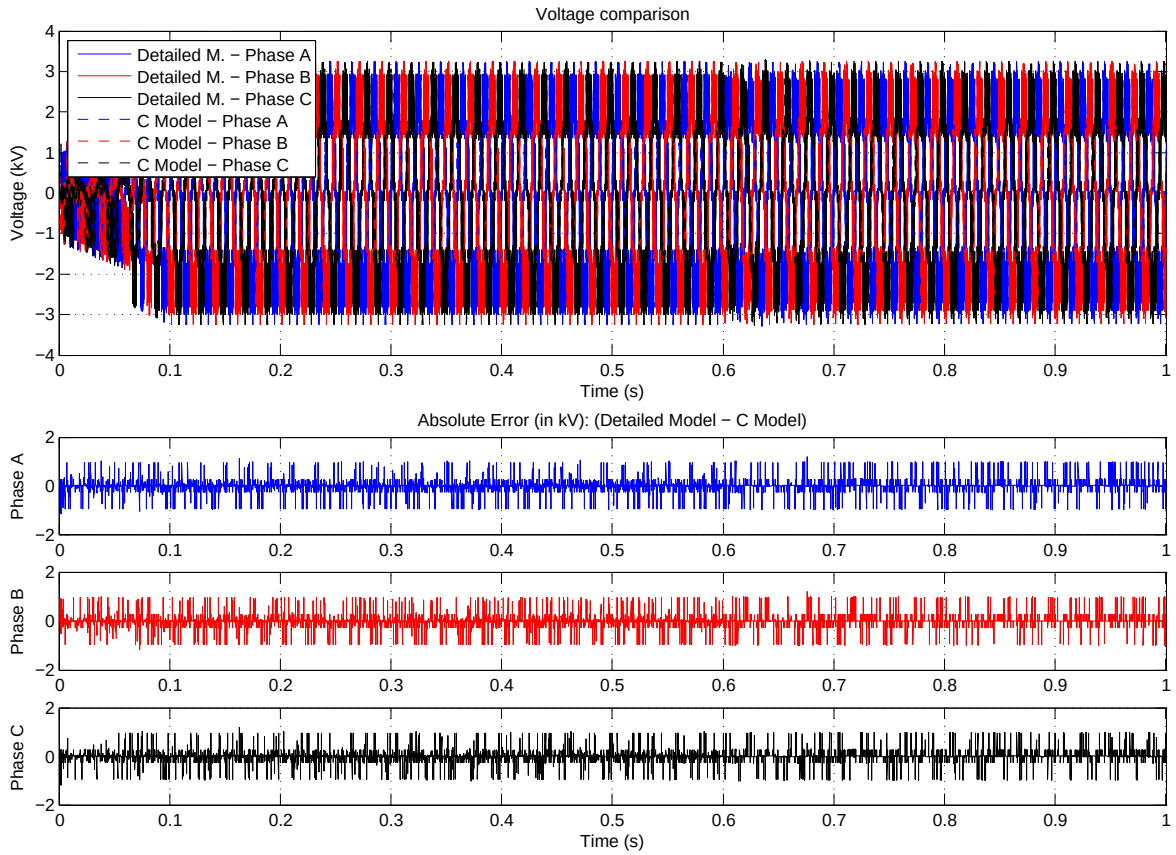


Figure 5.17: MMC model validation: Phase output voltages

Figures 5.22 and 5.23 show the capacitor voltages ripple enlarged. It can be seen that the ripple of the detailed model is followed closely by the C model.

The errors in some of the signals that were close to 10% could be explained because of the solver PSCAD uses for the detailed model computations. PSCAD does not use a strict fixed step-time. Once a switching state is detected, the solver comes back to the previous iteration and starts calculating at a smaller rate until it finds the switching moment. This increases the accuracy of the detailed model significantly and thus, the differences found in the developed C model might be caused in great manner by this issue. Nonetheless, it can be said that the dynamics and results of both models are very close, hence confirming the C model as valid.

5.3.2 *HLS MMC model implementation*

The MMC C model developed in PSCAD was ported to Vivado HLS in order to implement a hardware IP that will run in the FPGA fabric of the *Zynq*.

For a preliminary verification of the proper operation of the IP, a C simulation was run in Vivado HLS using the same control signals generated by PSCAD. The V_{in} and PWM signals fed to the PSCAD C model were stored in a data file. Then, in Vivado HLS, they were read from that same file and fed into the C model function. The results were stored in another file for a latter verification with the results of the C model implemented in PSCAD. Running a simulation of $1s$ and then comparing the model outputs, the average error between the two implementations was below $10e^{-7}$.

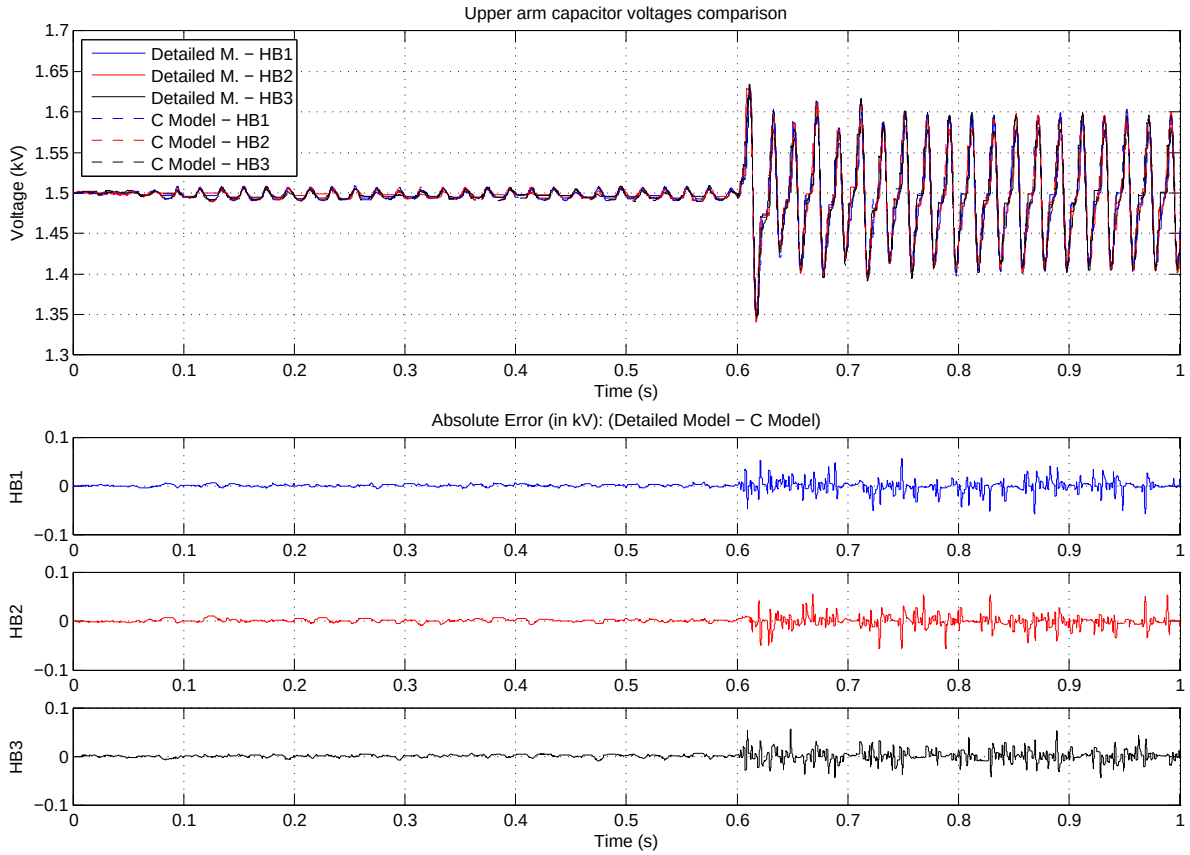


Figure 5.18: MMC model validation: Upper capacitor voltages

5.3.2.1 *MMC control*

The controller used for this application is based on the *Modular Multi-level DC-DC Converter for HVDC grids* presented in [89]. Regarding the control function, it takes as inputs the currents and capacitor voltages and outputs the reference voltages to be applied to each HB cell. Hence, if there are 3 SMs in the upper arm, and 3 SMs in the lower arm, the function will require 6 input voltages, 2 input currents (one per arm), and will output 6 reference voltages (one per cell).

It is intended for the controller to be executed every $100\mu s$, using a double-update method being able to change the duty cycle in the highest and lowest value of the triangular waveform [88].

Focusing on a single phase, this regulator can be divided into three sections: (i) branch energy control, (ii) branch current control, and (iii) capacitor balancing. Figure 5.24 shows the first loop where the energy of each arm is compensated by the currents $i_e^{u,l}$.

The lower arm is controlled to create a DC voltage plus an AC voltage (V_{DC} and V_{AC} respectively) [89]. V_{in} is the input DC voltage, $V^u_{caps_n}$ and $V^l_{caps_n}$ are the upper and lower capacitor voltages, ref^u and ref^l the reference values calculated using equation 5.7, where $V^{u,l}_{caps_{ref}}$ are the voltages at which the capacitors want to be kept.

$$ref^{u,l} = N^{u,l}\left(V^{u,l}_{caps_{ref}}\right)^2 \tag{5.7}$$

The control diagram utilised to control the arm currents $i_c^{u,l}$ is shown in Figure 5.25, where f is the frequency of the desired output voltage ($100Hz$), t is the time in seconds, $N^{u,l}$ are

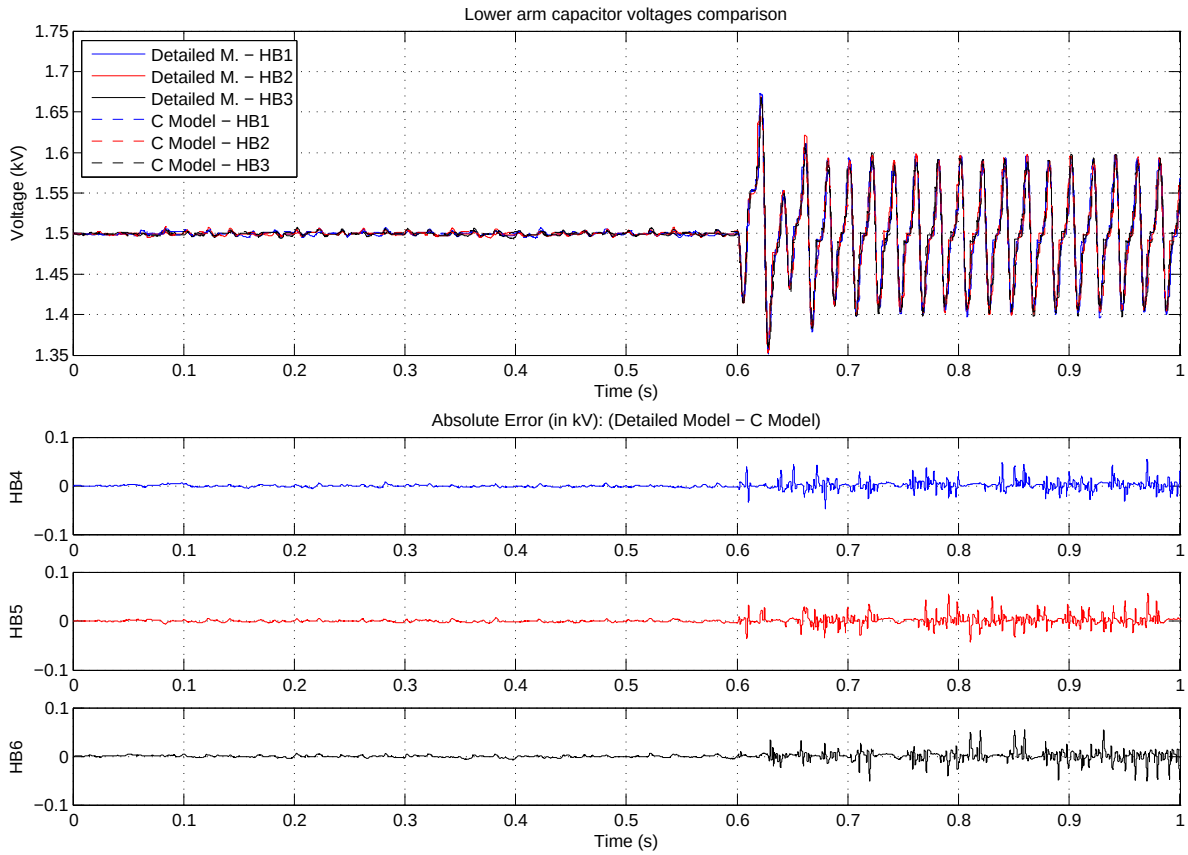


Figure 5.19: MMC model validation: Lower capacitor voltages

the number of SMs in the upper and lower arms, and $i^{u,l}$ are the measured upper and lower currents.

Figure 5.26 shows the upper and lower capacitor balancing controllers, which are replicated depending on the number of HB cells n. $V^{u,l}_{caps_n}$ are each measured capacitor voltage and $V^{u,l}_{ref_n}$ the voltage to be generated by each SM. This last value will be fed into the *PWM generator* to modulate the SM according to the required output voltage.

5.3.3 *Control verification using the MMC IP as HIL*

The MMC hardware IP was generated and loaded into the Vivado repository. A block design was created adding and connecting the IP with the PS and all other required peripherals. A simplified diagram of the block design is shown in Figure 5.27. It can be seen that the input DC voltage V_{in} is retrieved from the SDcard, where there is a data file with the PSCAD simulation values. This data is loaded and then fed into the HIL block as if it would come from an ADC. The PWM signals are derived from the *PWM generator* IP, which has been coded in VHDL, and basically translates the reference voltages calculated by the controller to pulse-width modulated signals. Then, the HIL outputs are calculated and registered automatically into the OCM and DDR memory spaces. At the end of the simulation, the data logged in the DDR is transferred into the SDcard for analysis purposes.

Once the hardware IP is configured and included in the Vivado block design, the next step is to open the SDK, add the libraries, create the necessary variables to manage the IP and initialize it:

```
// Libraries
```

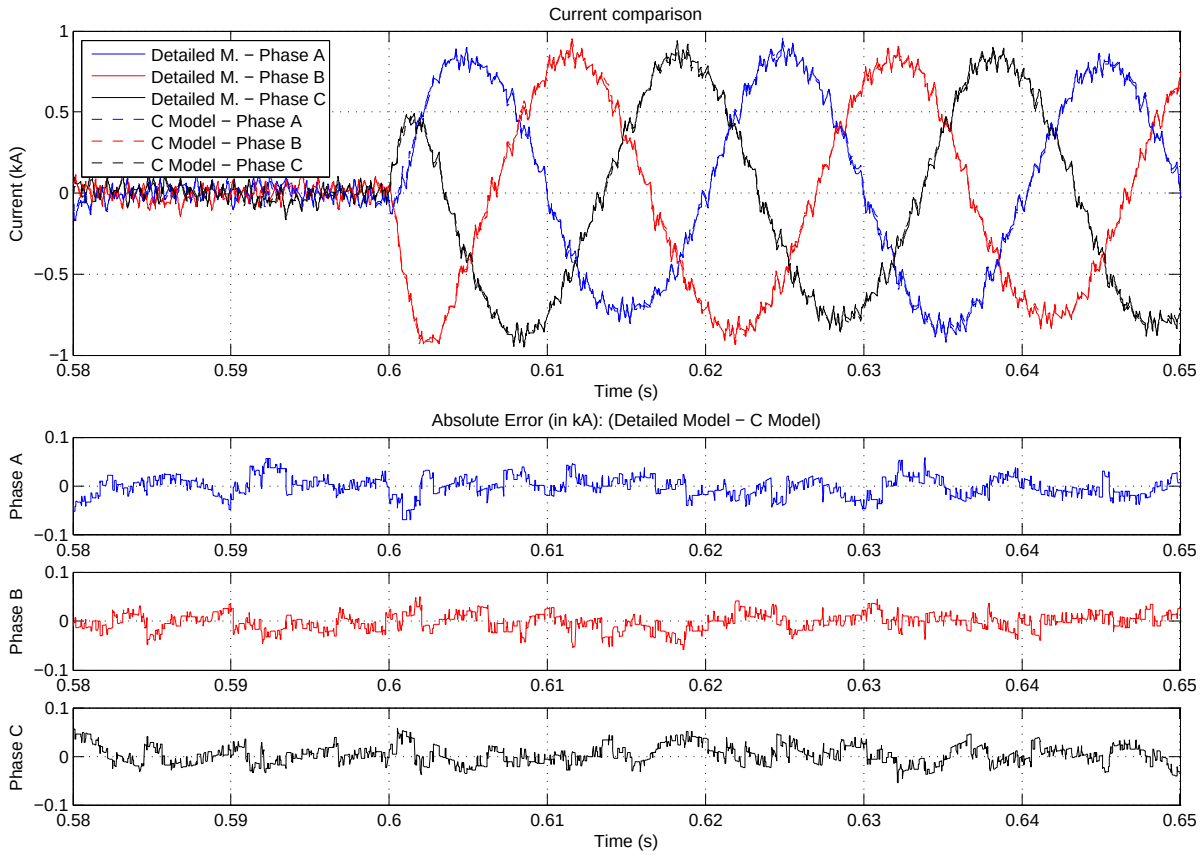


Figure 5.20: MMC model validation: Phase output currents - Detail

```
#include "xparameters.h"
#include "xgpio.h"
#include "xmmc_hil2.h"

// Variables and parameters
#define MMC_HIL_DEVICE_ID       XPAR_MMC_HIL2_1_DEVICE_ID
#define MMC_HIL_OCM_ADDRESS 0x00000018
float *MMC_HIL_outputs = new float[sim_length*9];
unsigned int MMC_HIL_outputs_DDR_addr = *(unsigned int*)&MMC_HIL_outputs;
XMmc_hil2 MMC_HIL;

// Initialization
XMmc_hil2_Initialize(&MMC_HIL, MMC_HIL_DEVICE_ID);
XMmc_hil2_Set_OCM(&MMC_HIL, MMC_HIL_OCM_ADDRESS);
XMmc_hil2_Set_DDR(&MMC_HIL, MMC_HIL_outputs_DDR_addr);
```

At this point, the IP is ready for its utilisation. It is only left to set the input DC voltage using a generic *AXI4-GPIO* port and start the IP computations. Nothing else is required. As soon as the IP finish calculations, they will be automatically saved into the OCM for its use by the processor, and to the DDR memory for data logging purposes. Therefore, the only two commands needed to manage this IP are the following:

```
XGpio_DiscreteWrite(&Gpio_ERTS_Vin, VIN_CHANNEL, *(unsigned int*)&XADC_Vin[count_CIC_int
    ]);
XMmc_hil2_Start(&MMC_HIL);
```

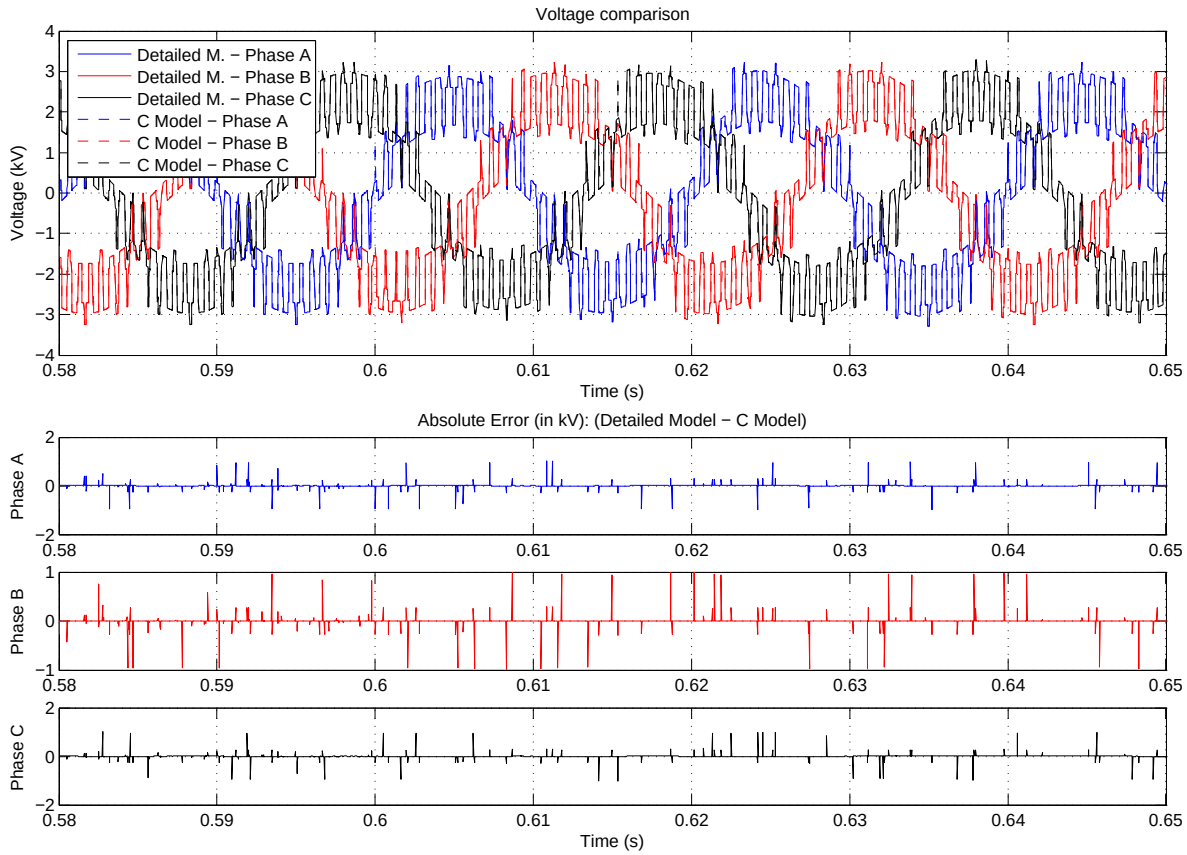


Figure 5.21: MMC model validation: Phase output voltages - Detail

An interrupt was configured to halt every $5\mu s$ and start the IP calculations, ensuring a real-time execution. Figures 5.28 and 5.29 show a power up test, where the next stages can be seen:

1. from $0s$ to $1.5s$ the input voltage is linearly raised up to $311V$ (Figure 5.28 up), keeping the duty cycle of the PMW signals at 100%, thus the capacitors start charging (Figure 5.29)

2. from $1.5s$ to $3s$ the duty cycle of all the cells is reduced gradually from 100% to 48% increasing the voltage of the capacitors til $100V$. Notice the increase of the arm currents in Figure 5.28

3. from $3s$ to $4.5s$ the duty cycle is kept to 48% letting the capacitor voltages diverge as seen in Figures 5.29 and 5.30.

4. at $4.5s$ and until $5.5s$ the controller is activated without the integral part. Hence, the voltages are kept balanced but not exactly at $100V$ as seen in Figure 5.30. Figure 5.28 shows how the output voltage changes from $155V$ DC to $125V$ AC plus $155V$ DC at $100Hz$

5. at $5.5s$ and until $6.5s$ the integral part of the regulators is activated driving all the capacitor voltages to $100V$

6. at $6.5s$ a 44Ω load is introduced (i.e. the HIL IP switches to *normal operation mode*). Figure 5.28 down shows how the currents increase significantly. Observe as well in figures 5.29 and 5.30 how the capacitor ripple increase

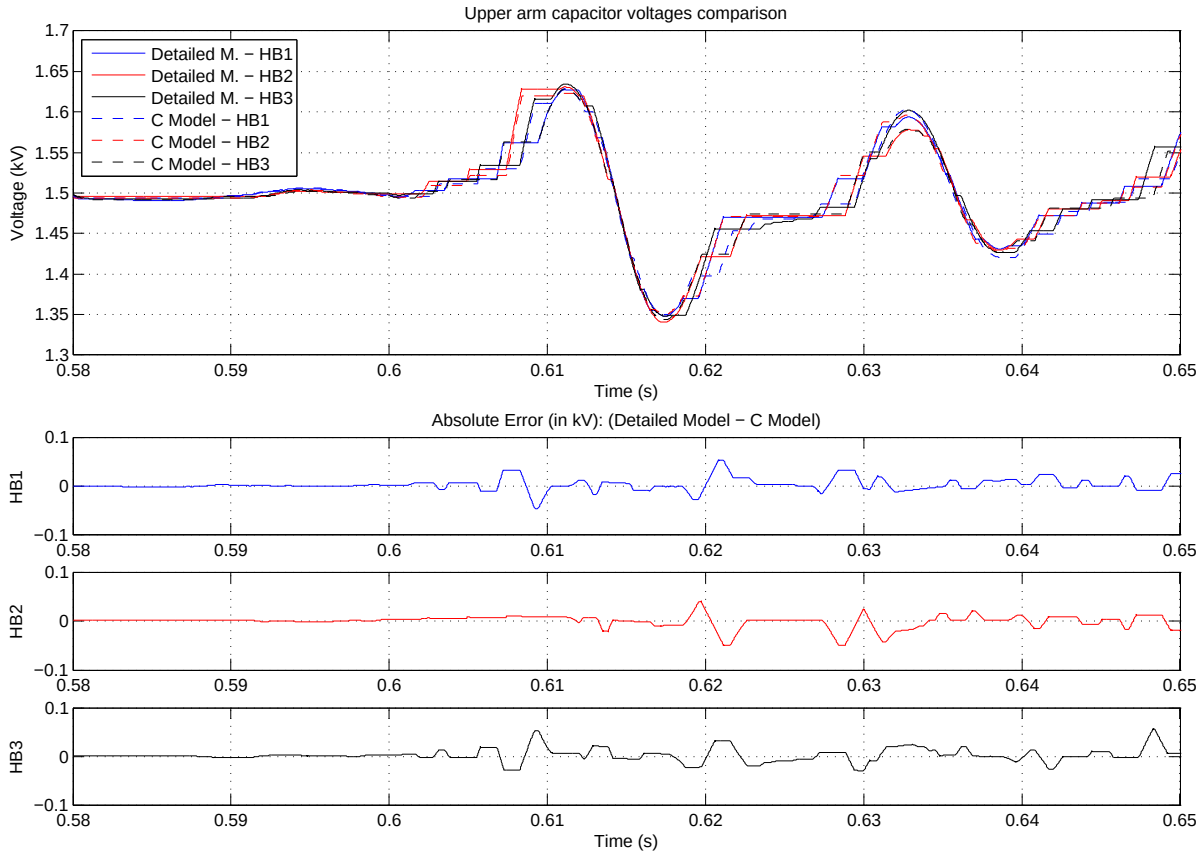


Figure 5.22: MMC model validation: Upper capacitor voltages - Detail

7. at 7.5s the load is removed, the capacitor ripple decrease and the controller sets their voltages back to 100V

Figure 5.31 shows at 4.5s the exact moment when the control starts operating, showing the typical 7 levels waveform produced by the MMC configuration.

Figure 5.32 presents the change at 6.5s from no-load to a 44Ω load. The smoothing of the output voltage when the load is connected is caused by the arm inductances L_{arm}. The voltage produced by the SMs is a PWM signal, but when the load current is increased, all the noise caused by the PWM infers an important voltage drop in the inductances. This latter, added to the voltage created by the HBs results in a sinusoidal-like waveform. Hence, these inductors, apart from filtering the output current, also filter the output voltage. In the same figure can be observed as well how the upper branch is providing much more current than the lower arm. Figure 5.33 shows the opposite case, where at 7.5s the load resistor is removed, thus disappearing the voltage filtering and appearing again the 7-levels common in this kind of converters.

According to all the presented results, it can be said that the control performs appropriately and is ready for its test on the real test rig.

5.3.4 *Control verification with experimental prototype*

Figures 5.34 and 5.35 display a 5.5s charging test on the real system. Next, the following stages are explained:

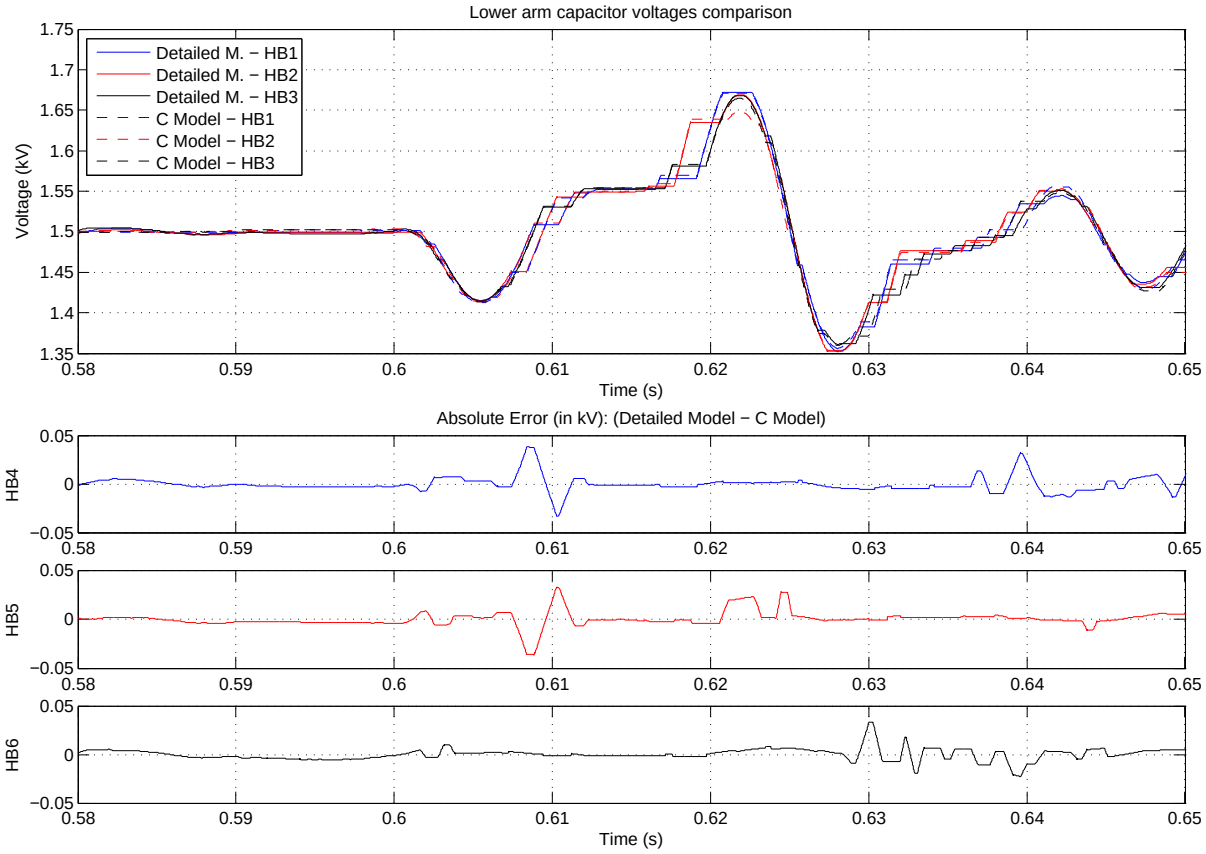


Figure 5.23: MMC model validation: Lower capacitor voltages - Detail

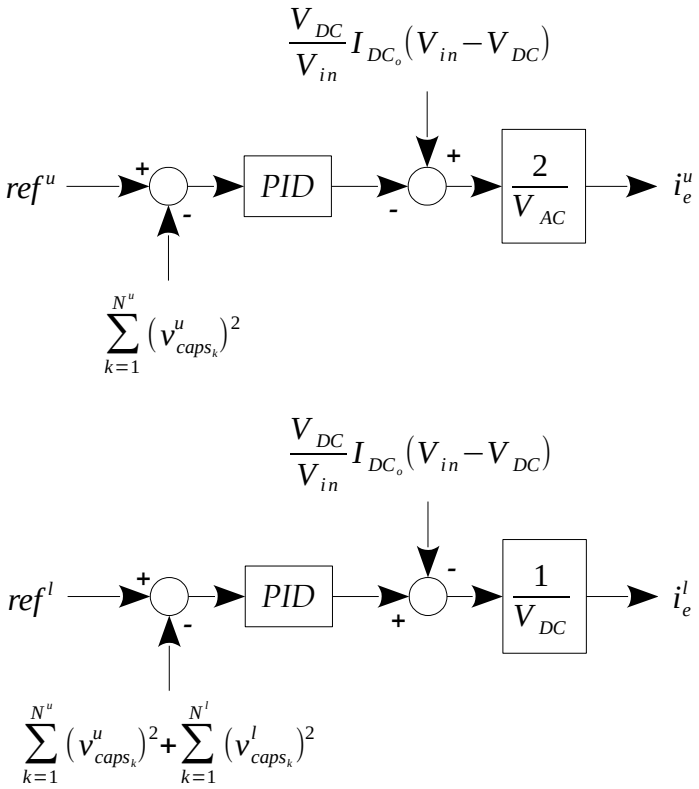


Figure 5.24: Energy controllers

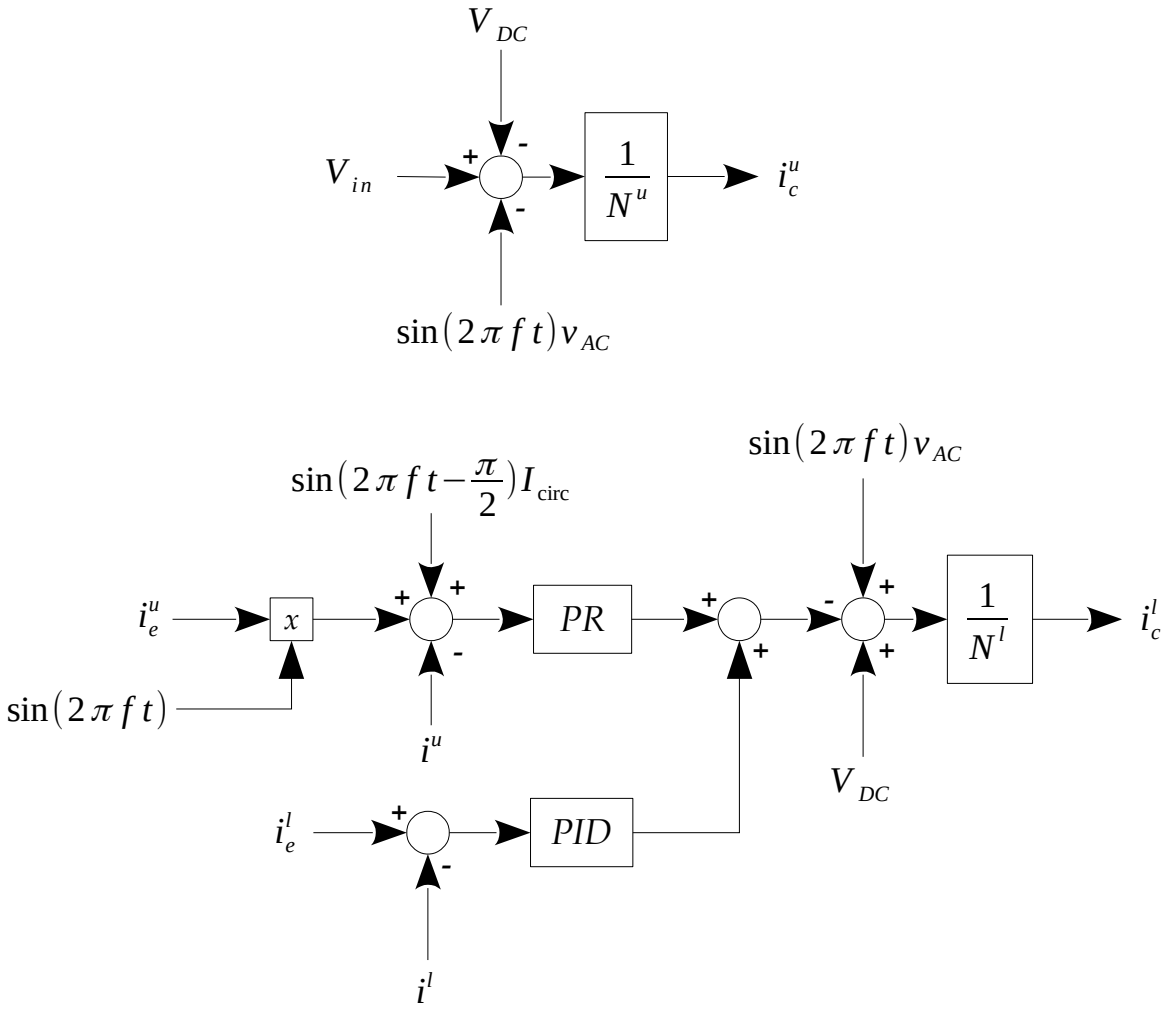


Figure 5.25: Current controllers

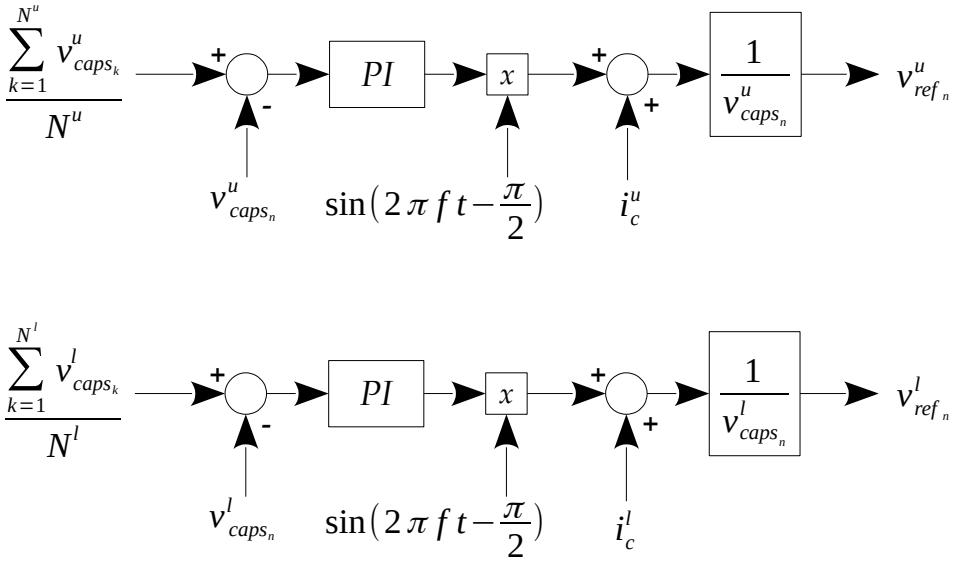


Figure 5.26: Capacitor balancing controllers

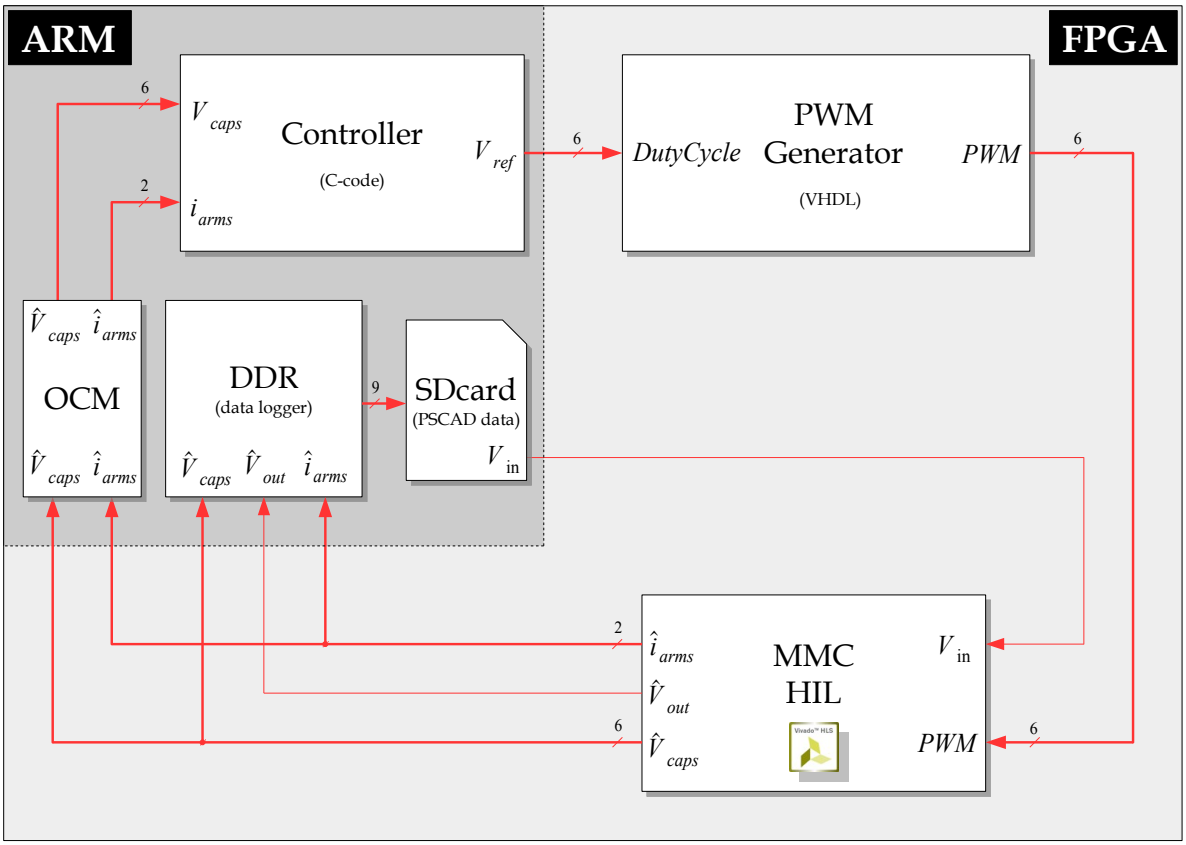


Figure 5.27: MMC IP working as HIL

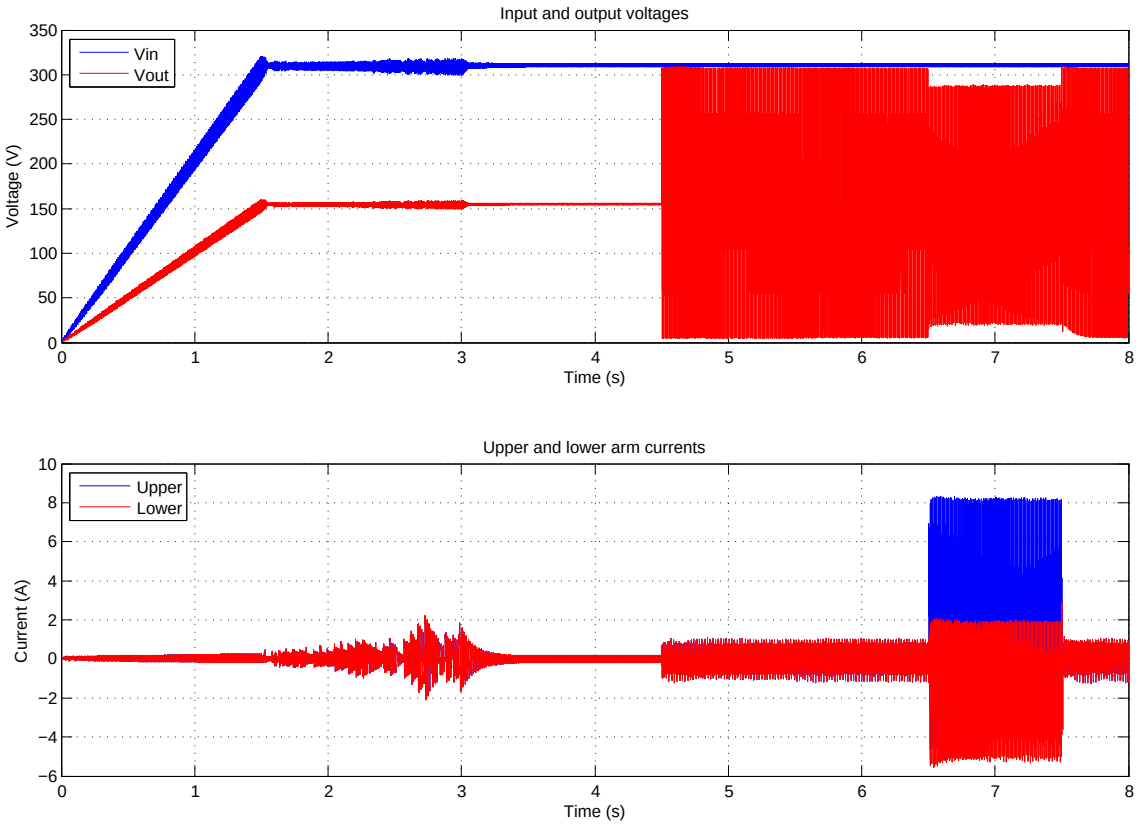


Figure 5.28: Input voltage and HIL estimated output voltage and currents - Complete simulation

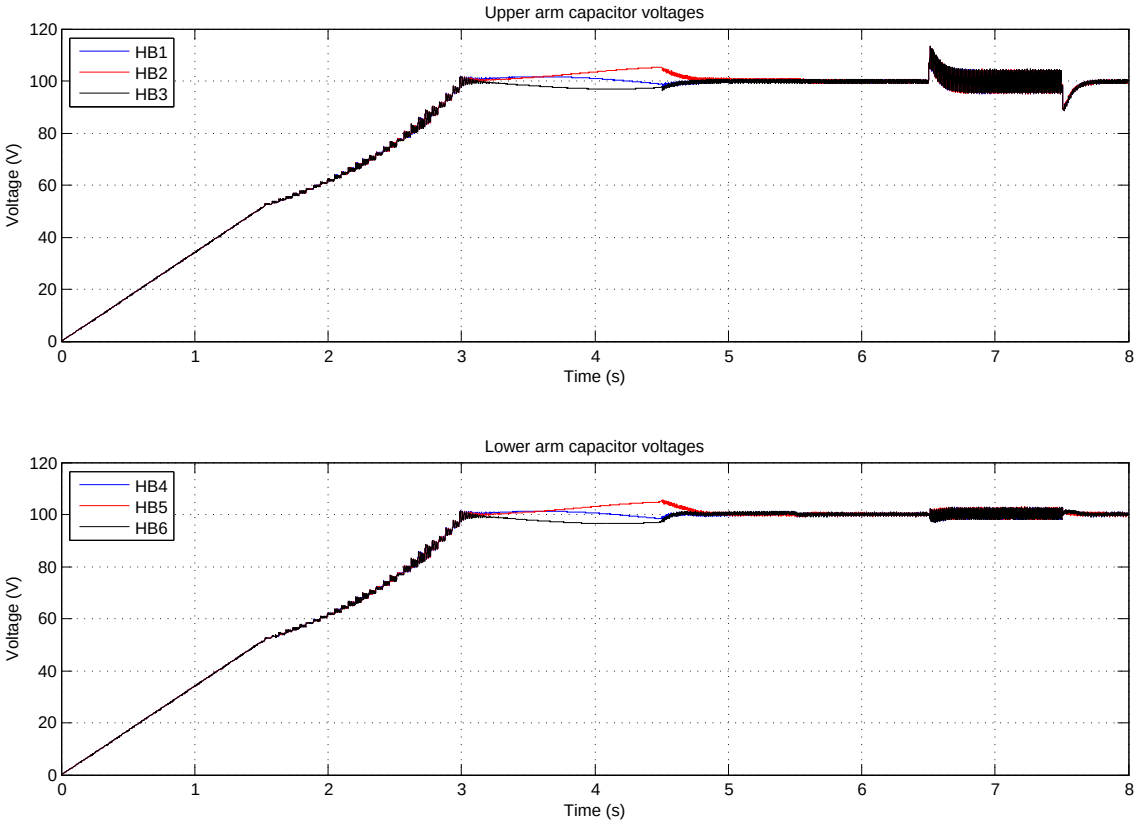


Figure 5.29: HIL estimated capacitor voltages - Complete simulation

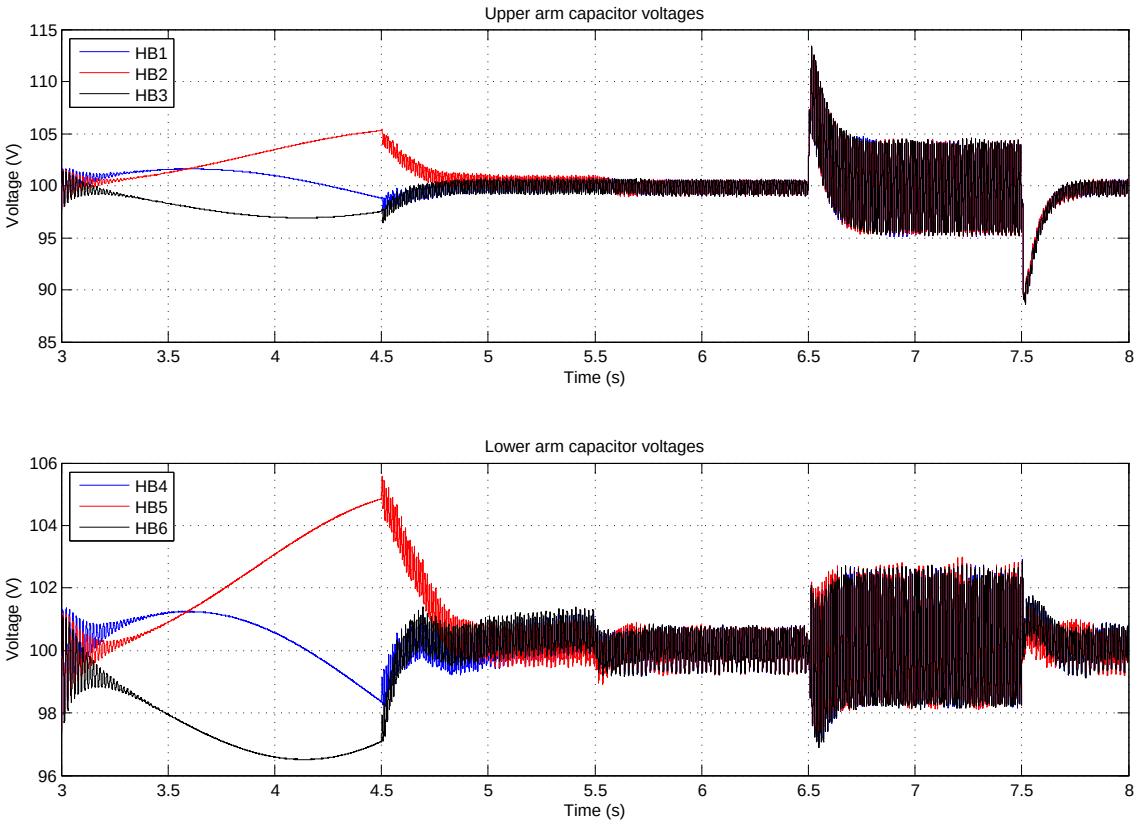


Figure 5.30: HIL estimated capacitor voltages - Capacitor oscillations

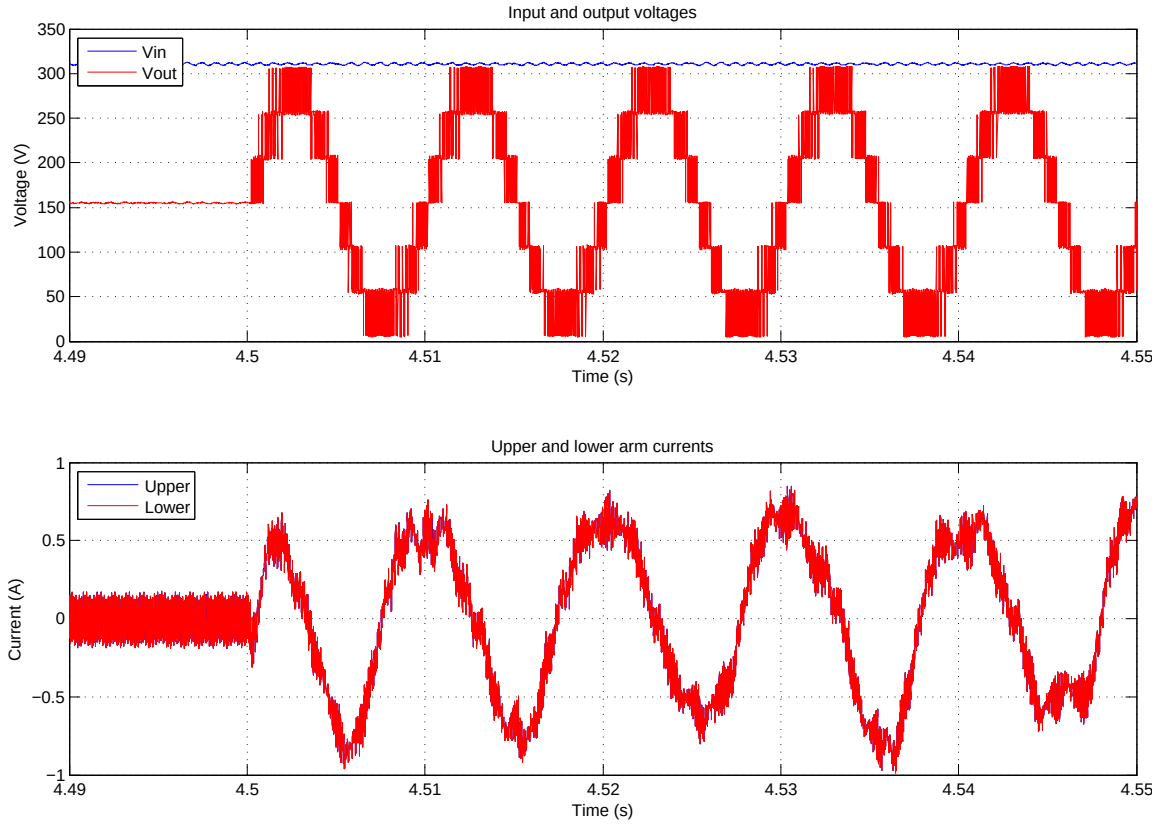


Figure 5.31: Input voltage and HIL output voltage and currents. At 4.5*s* the control starts operating

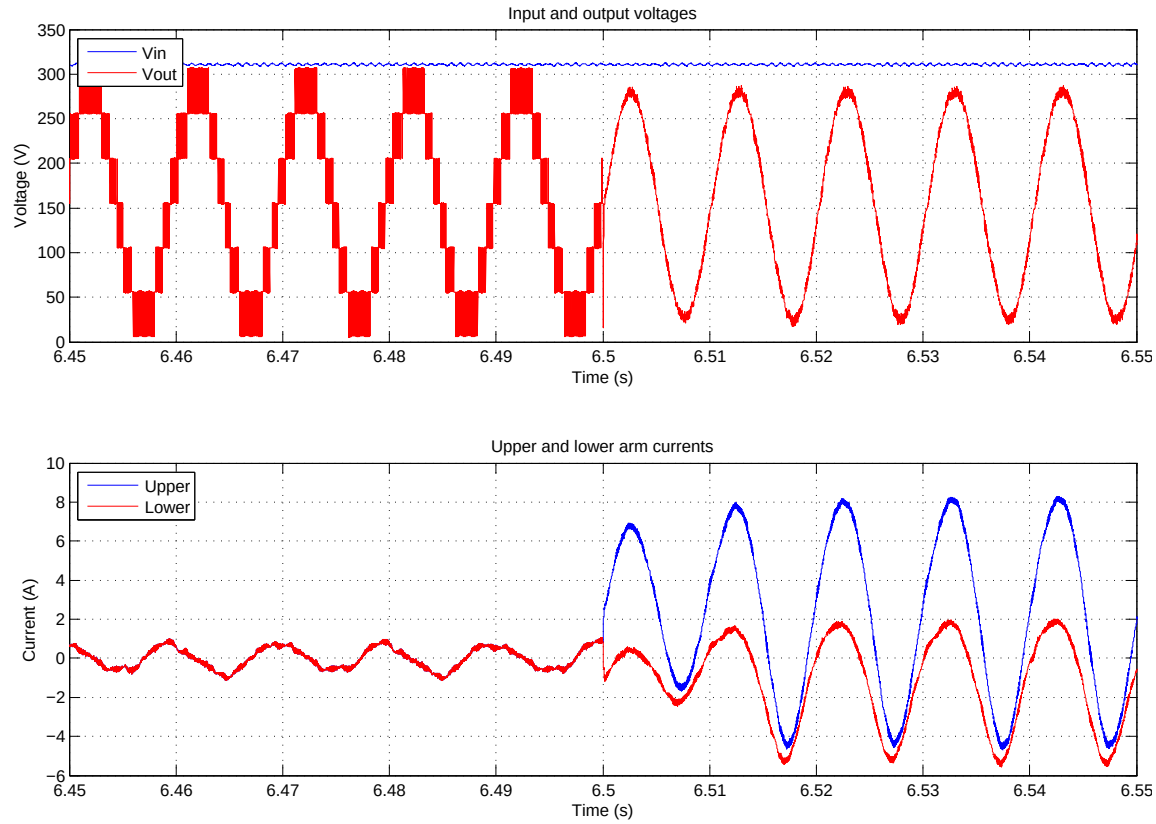


Figure 5.32: Input voltage and HIL estimated output voltage and currents. Load switched in at 6.5*s*

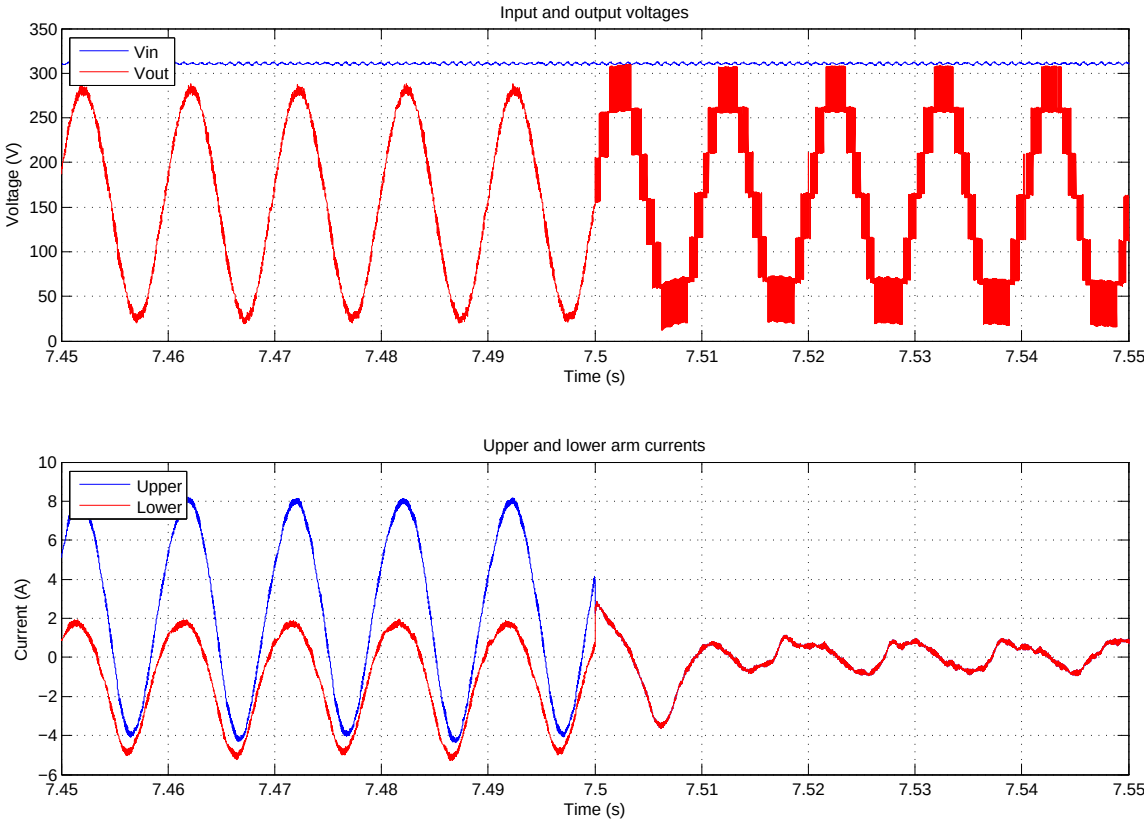


Figure 5.33: Input voltage and HIL estimated output voltage and currents. Load switched out at 7.5*s*

1. from 0*s* to 2*s* the input voltage is increased manually using an autotransformer. Figure 5.34 shows how the voltage reaches 311*V* at 1.6*s* approximately and in Figure 5.35 can be observed how the charge of the capacitors follow the same ramp. The duty cycle of the PWM signals is kept at 100% for the whole period

2. from 2*s* to 3.5*s* the duty cycle is reduced gradually down to 48%. The capacitors reach roughly 100*V* at the end as displayed in Figure 5.35

3. at 3.5*s* the control starts to operate setting the output voltage to 125*V* AC around 155*V* DC and at 100*Hz* as specified by the controller (see Figure 5.36), and keeping the capacitor voltages at 100*V* (see Figure 5.35). Figure 5.37 shows how all the capacitors are oscillating at this same frequency

At this point, the controller has been validated, first with the offline simulation, then in real-time using the MMC IP as HIL, and finally in the experimental test bench. However, due to a lack of time, the load test was not possible to be implemented.

5.3.5 *eRTS for cell voltage estimation and fault-tolerant control*

One of the most interesting applications of eRTS is when used in the context of fault-tolerant control. The eRTS will operate alongside the real system so when a measurement cannot be read or is faulty, the control switches to use the signals provided by the eRTS block.

In this Subsection, this scenario has been tested assuming a simultaneous fault in all cell capacitor voltage sensors, as a worst case scenario. Right after the fault has been detected, the controller switches from reading the input signals from the *CIC filter* OCM memory space to

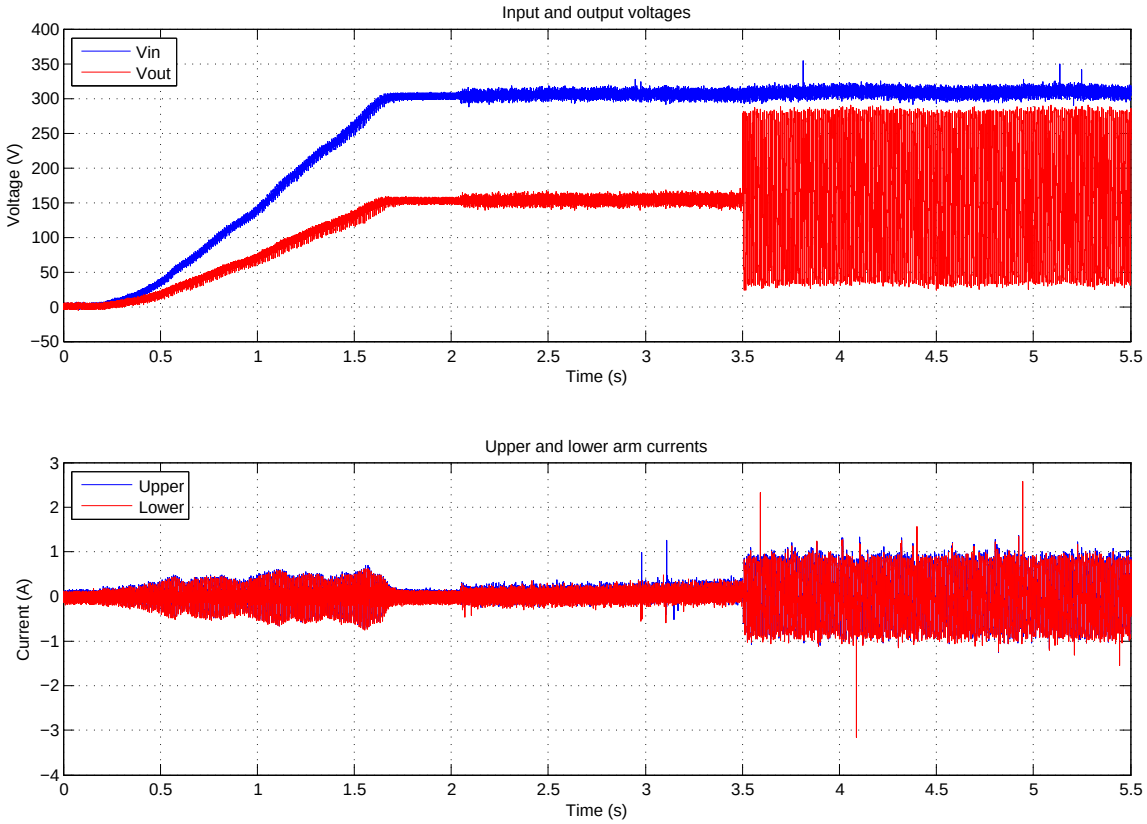


Figure 5.34: Test bench input and output voltages and output currents (from 0*s* to 5.5*s*)

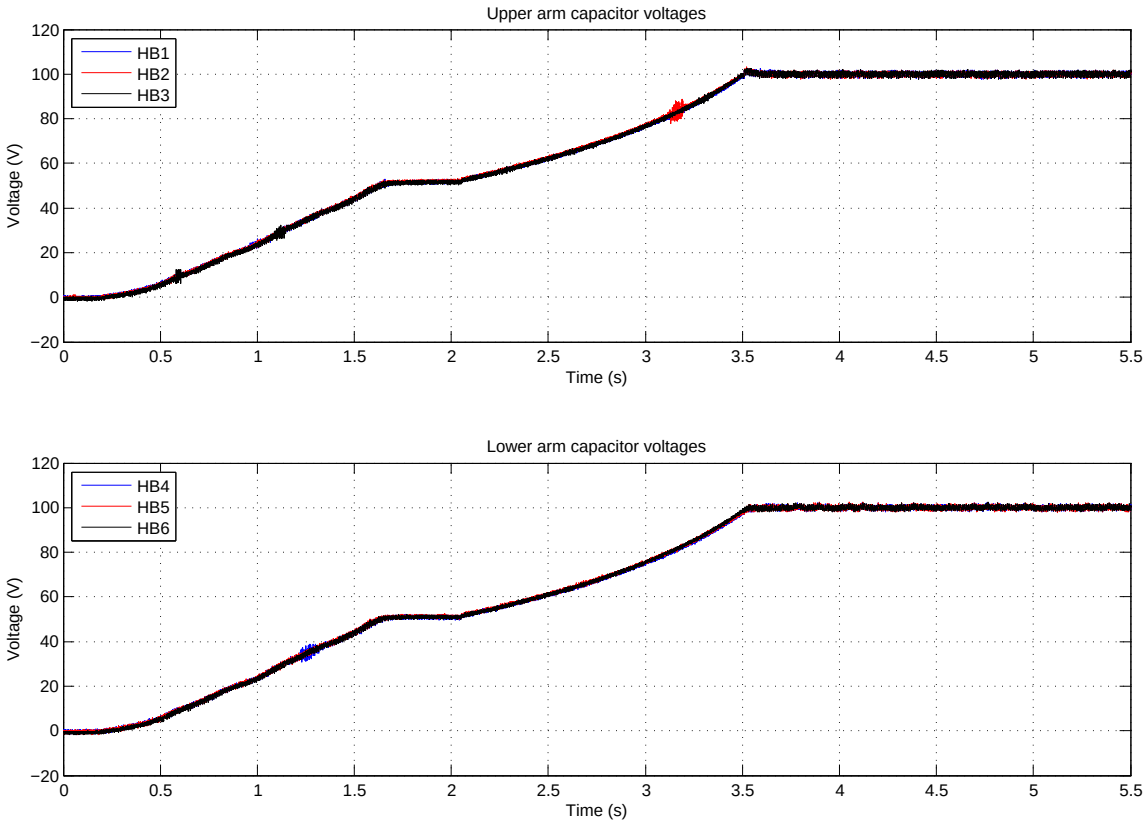


Figure 5.35: Test bench capacitor voltages (from 0*s* to 5.5*s*)

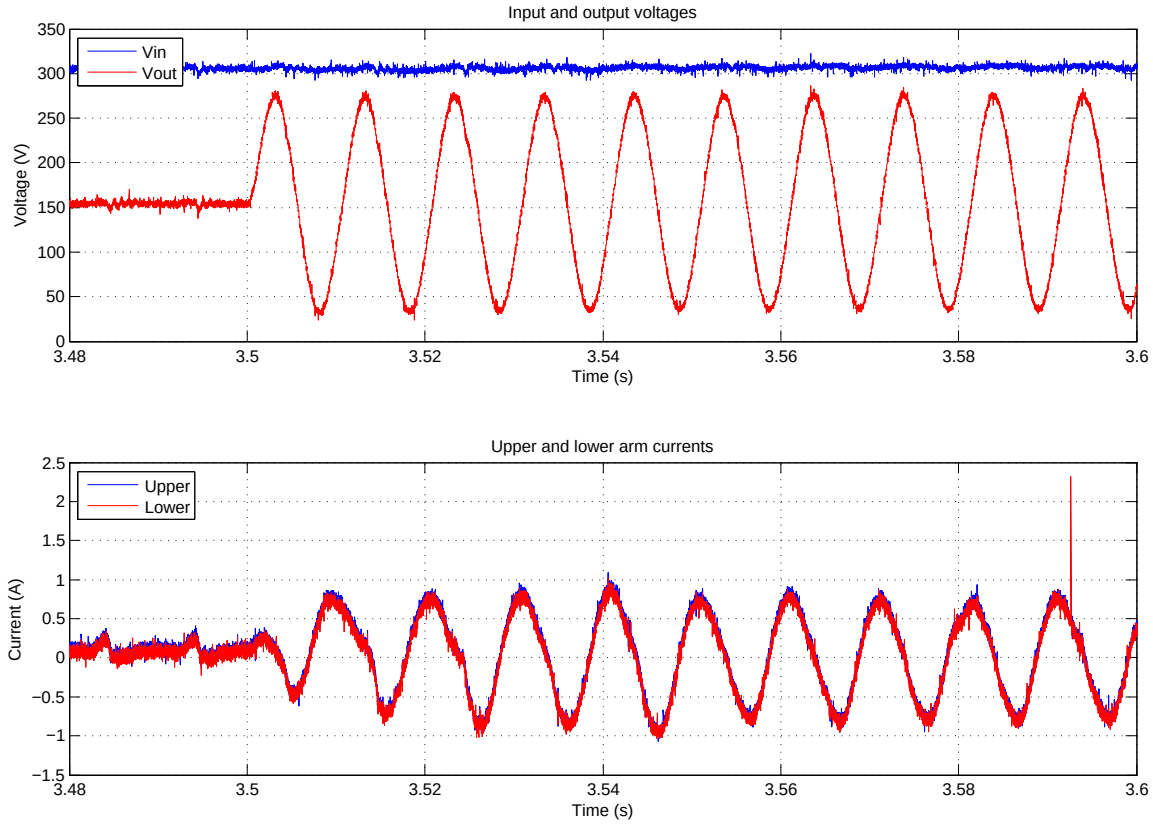


Figure 5.36: Test bench input and output voltages and output currents(from 3.48*s* to 3.6*s*)

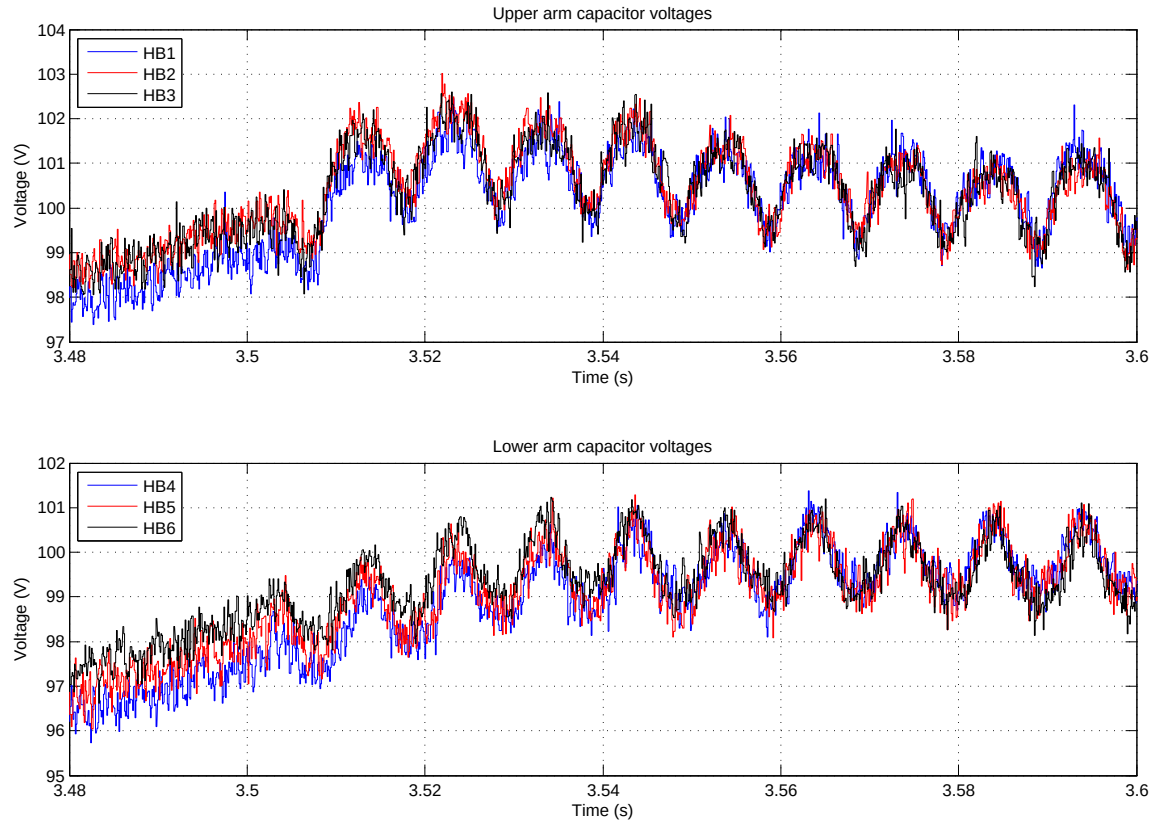


Figure 5.37: Test bench capacitor voltages (from 3.48*s* to 3.6*s*)

reading them from the eRTS OCM memory space. Figure 5.38 shows a diagram of how the different blocks and system data paths are structured.

5.3.5.1 *Current offset observer*

The purpose of the observer shown in Figure 5.39 is to estimate the capacitor current offset and then use it to compensate automatically the drift in the capacitor voltage estimation. i_c is the current through the capacitor which is calculated using $i_c = S_n i^{u,l}$, where S_n is a logical function derived from the PWM signals. V_c and $\hat{V'}_c$ are the measured and estimated capacitor voltages respectively. Then, the estimated offset is fed through a low-pass filter (LPF) –in order to smooth the noise present in the measured signals, and finally is used to estimate the capacitor voltage.

5.3.5.2 *Capacitor voltage estimation*

Equation 5.8 is used to implement the capacitor voltage estimation shown in Figure 5.39. It receives the current offset estimated by the observer and then subtracts it to the current flowing through the cell in order to compensate the voltage drift caused by measurement errors of the current sensors.

$$\hat{V}_c(k) = \hat{V}_c(k-1) + S_n \frac{T_s}{C}\left(i^{u,l}(k-1) - i^{u,l}_{offset_{LPF}}(k-1)\right) \tag{5.8}$$

5.3.5.3 *eRTS for cell voltage estimation*

The two diagrams presented in Figure 5.39 are replicated 6 times –one per SM– and coded in a single IP using Vivado HLS. Hence, the only two commands needed to manage the IP are the following:

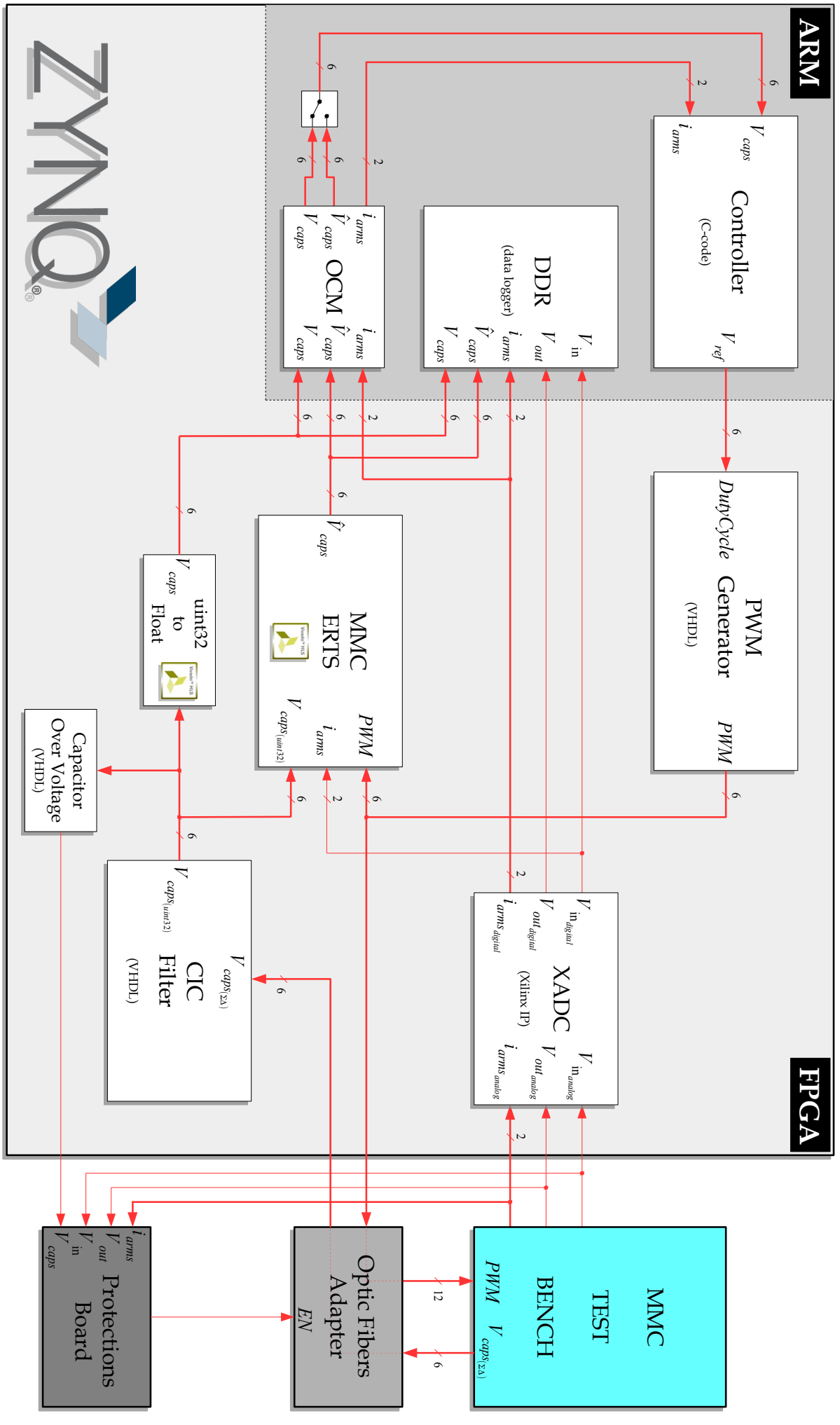
```
XGpio_DiscreteWrite(&Gpio_currents, UPPER_CURRENT_CHANNEL, *(unsigned int*)&XADC_
    currents[count_CIC_int*2]);
XGpio_DiscreteWrite(&Gpio_currents, LOWER_CURRENT_CHANNEL, *(unsigned int*)&XADC_
    currents[count_CIC_int*2+1]);
XMmc_erts_vcaps3_Start(&MMC_ERTS_Vcaps3);
```

In case of failure in the voltage measurement V_c, it is possible to switch the IP to *fault mode*, which causes the IP to freeze the current offset observer (upper diagram of Figure 5.39) from integrating the current and the voltage errorx, keeping i_{offset} steady. Hence, the ZOH maintains its last value –which is an averaged value thanks to the PI– and continues estimating the capacitor voltages $\hat{V}_c$ based on that averaged i_{offset} value and the new current measurements i_C.

Figures 5.40 and 5.41 show the measured and eRTS estimated cell voltages for the top and bottom cells, respectively. The measured voltage is in blue, whereas the eRTS estimated voltage is shown in red.

The eRTS estimated voltage follows very closely the $100Hz$ ripple caused by the MMC circulating currents. Note that the absolute ripple is about $2V$ around the $100V$ of each cell (2%). The estimation error is much smaller than this figure, being consistently below $0.5V$.

This result is remarkable as this experiment considers nominal values of cell capacitance, does not take into account PWM interlock delays and, moreover, the eRTS samples the PWM signals at $5\mu s$ intervals.


ARM
FPGA
ZYNQ
Controller
(C-code)
DDR
(data logger)
OCM
PWM
Generator
(VHDL)
MMC
ERTS
uint32
to
Float
Capacitor
Over Voltage
(VHDL)
CIC
Filter
(VHDL)
XADC
(Xilinx IP)
MMC
TEST
BENCH
Optic Fibers
Adapter
Protections
Board

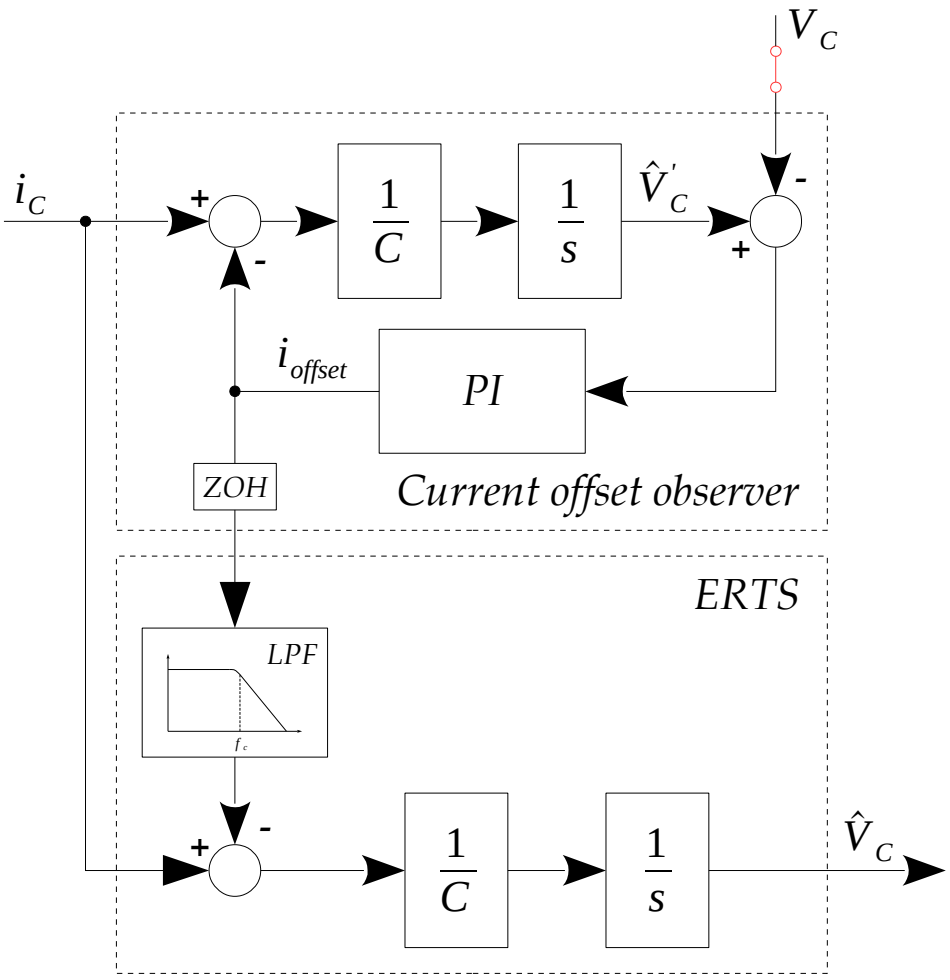


Figure 5.39: Current offset observer and Capacitor voltage eRTS diagram

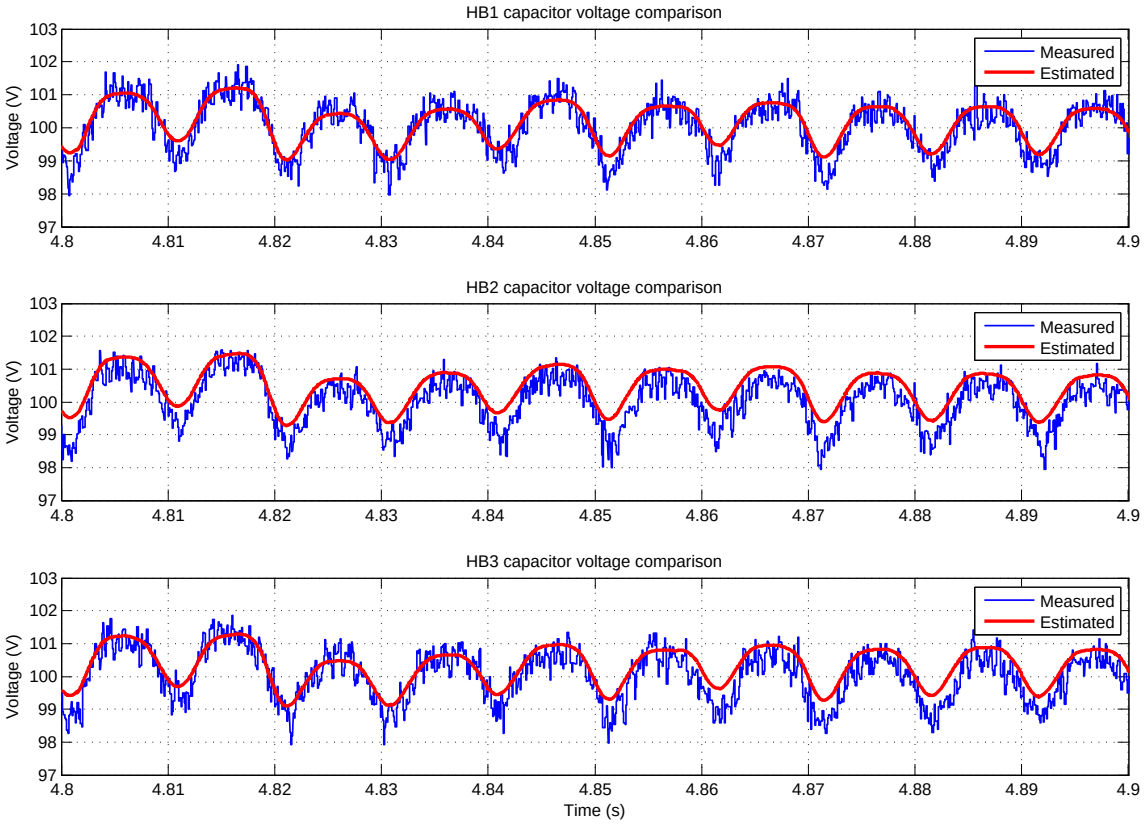


Figure 5.40: eRTS upper cell voltage estimation during normal operation

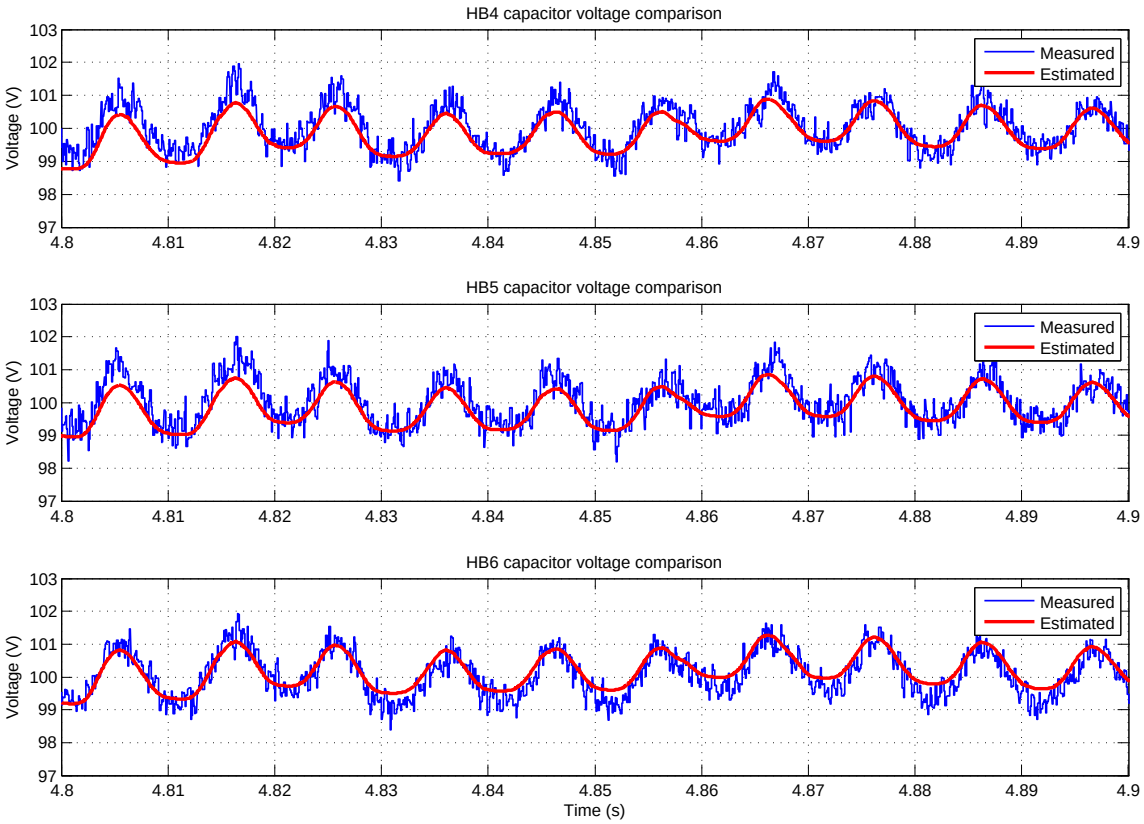


Figure 5.41: eRTS lower cell voltage estimation during normal operation

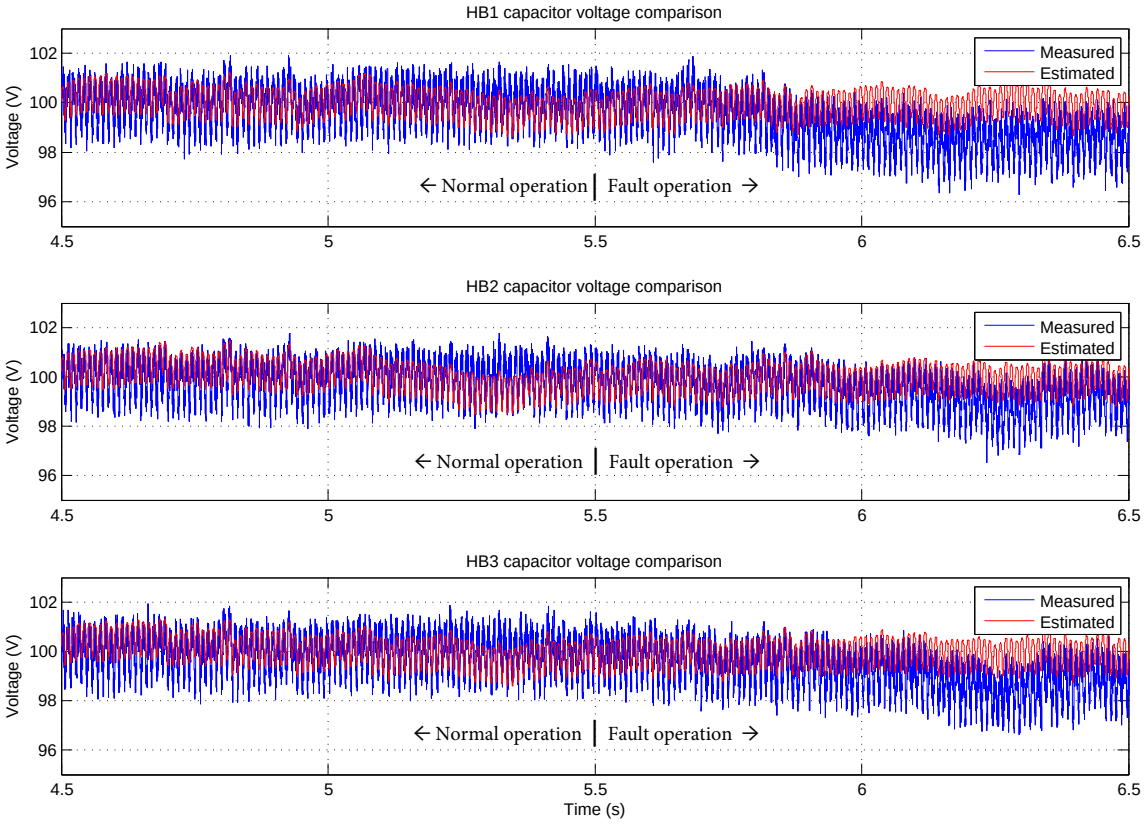


Figure 5.42: eRTS upper cell voltage estimation during change from normal operation mode to fault mode at 5.5*s*

5.3.5.4 *Fault-tolerant control using eRTS voltage measurements*

This section shows the capability of the eRTS estimation to provide adequate feedback signals for the controller when there is a fault in one or more of the cell voltage signals.

Figures 5.42 and 5.43 show the capacitor voltages of all the SMs before and after the failure of all the cell voltage sensors. This is an extreme case to show the performance of the eRTS estimate in such an unlikely situation.

The fault in all voltage feedback sensors is emulated by just switching the control feedback signals from those actually measured by the CIC to those provided by the eRTS at 5.5*s*. In an actual system, a fault detection algorithm would be used. However, as a proof of concept, here the feedback signals are just switched by a software command at 5.5*s*.

Looking at the error evolution in Figures 5.44 and 5.45, it is worth noting that the system operates reliably after the fault, however, there is a noticeable drift in the actual voltage. This is caused by the non-considered effects listed before (nominal parameters, converter non-linearities and PWM resolution). However, it is clearly seen that the controller drives the eRTS voltage estimate to their 100*V* reference.

Figures 5.46 and 5.47 show a detailed view of the estimated cell capacitor voltages exactly when the fault occurs. As in the previous graphs, the measured value is plotted in blue whereas the estimation is in red. It is worth noting that there is no abrupt change and the dynamics are followed as if it would not have occurred any fault.

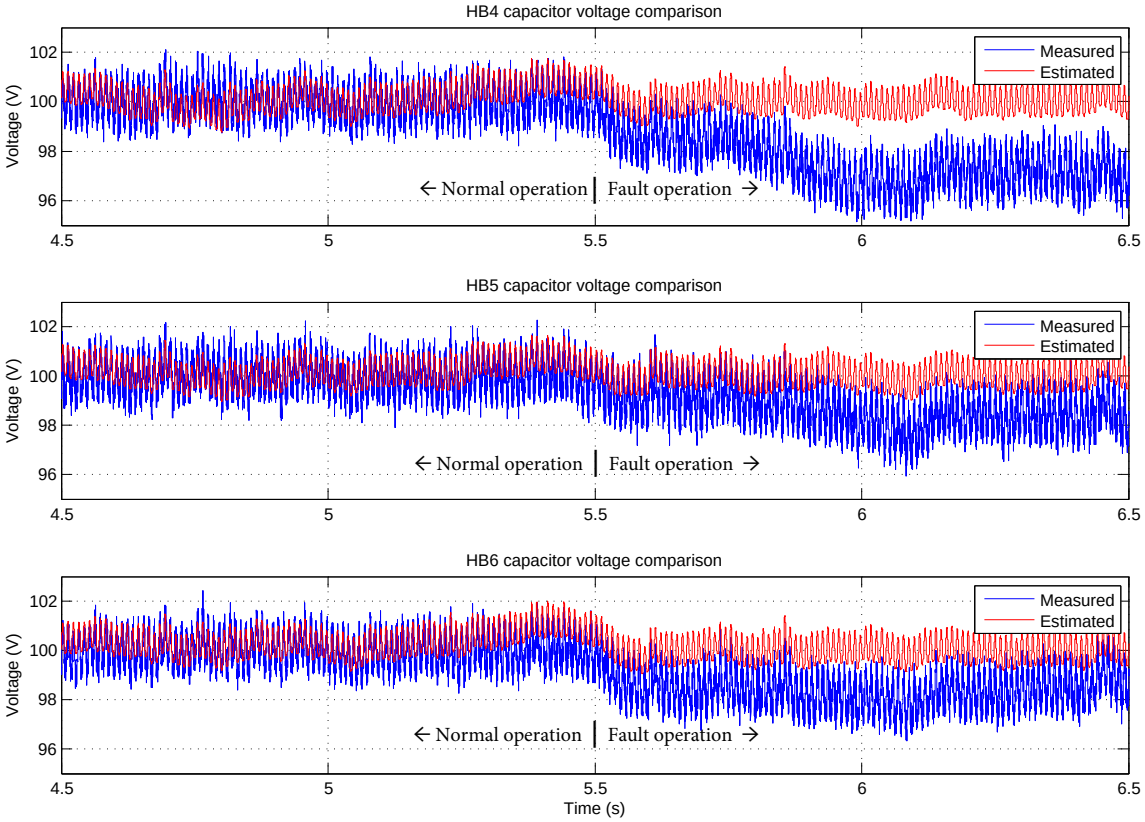


Figure 5.43: eRTS lower cell voltage estimation during change from normal operation mode to fault mode at 5.5*s*

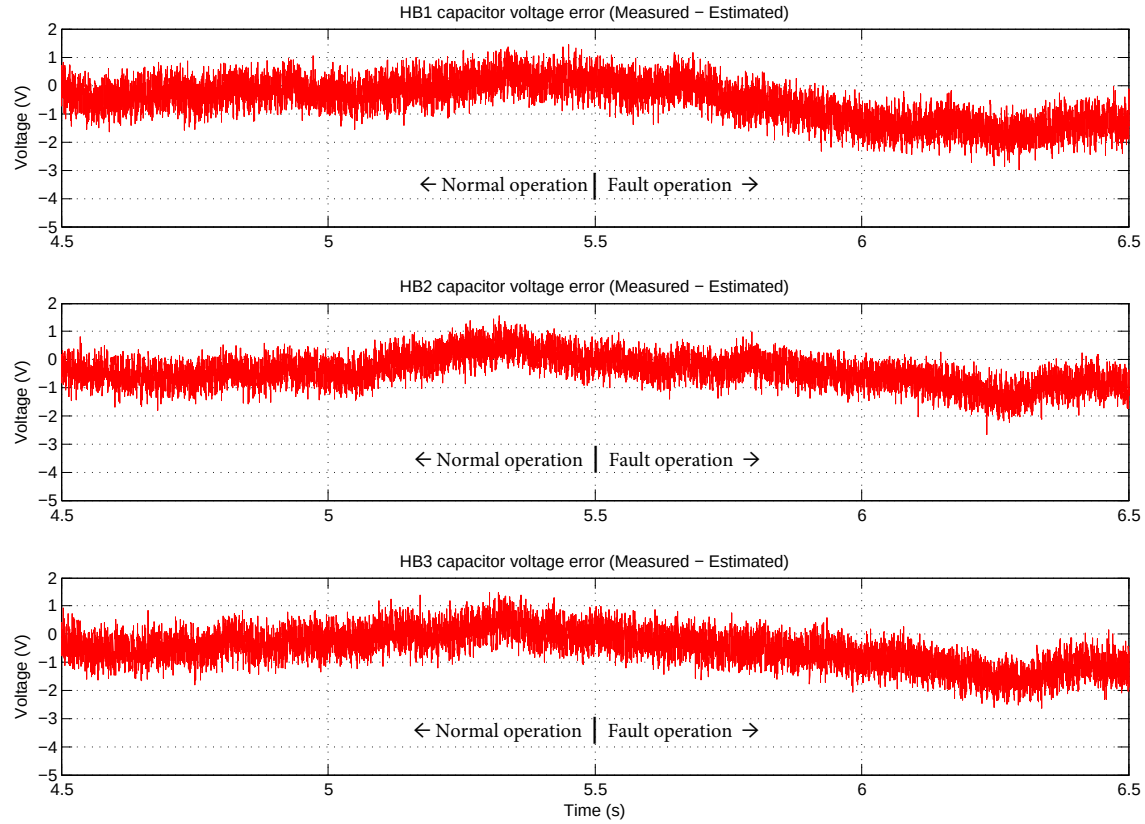


Figure 5.44: Upper capacitor voltage error

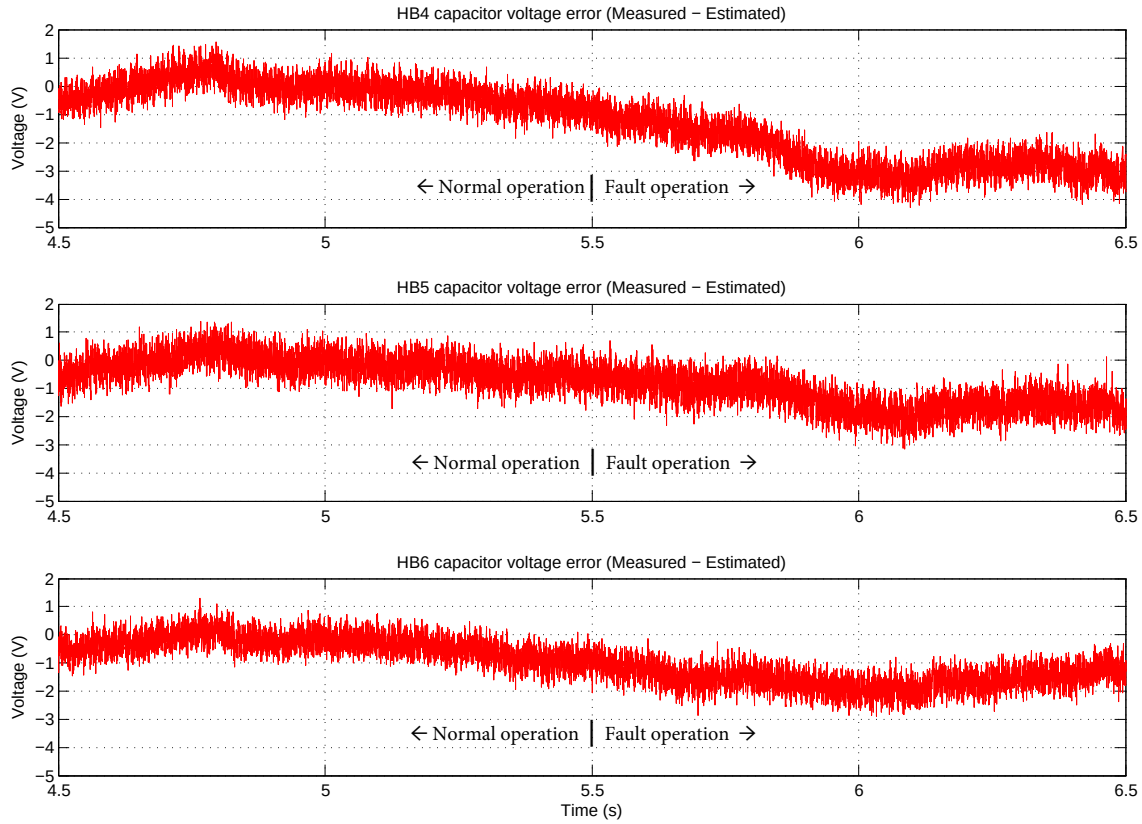


Figure 5.45: Lower capacitor voltage error

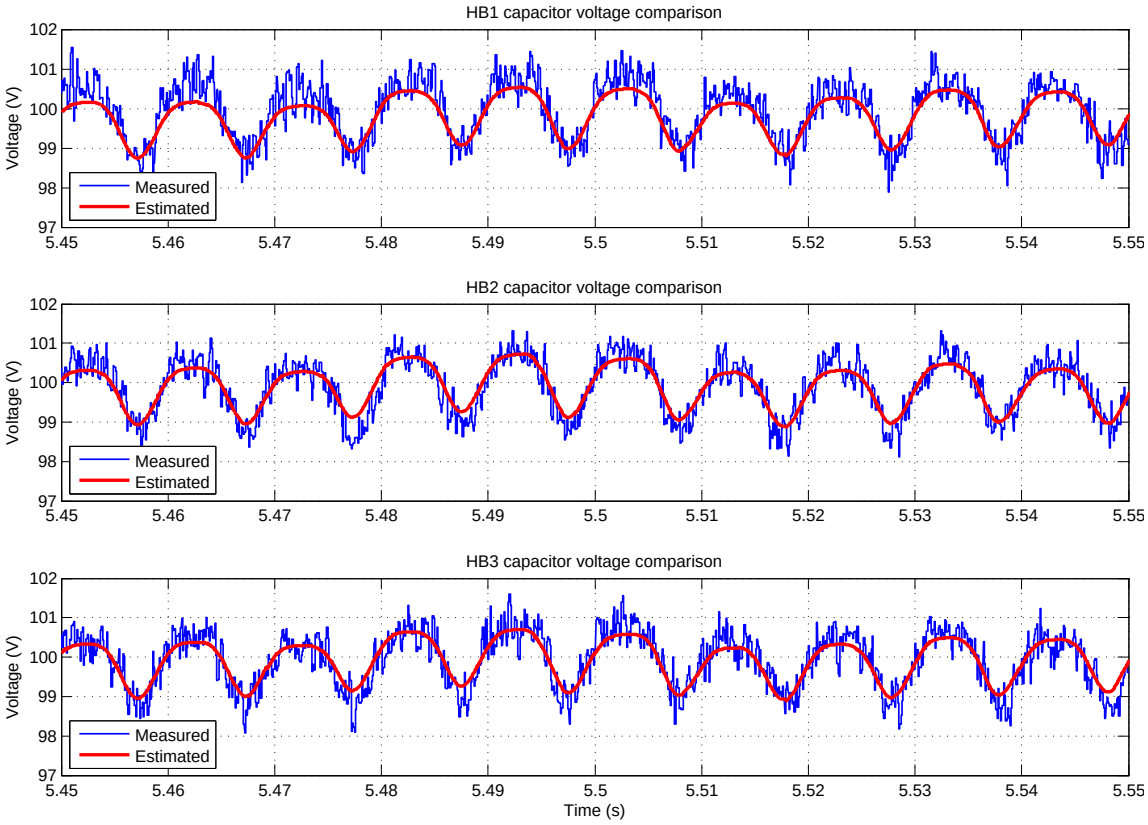


Figure 5.46: eRTS upper cell voltage estimation during change from normal operation mode to fault mode at 5.5*s* - Detail

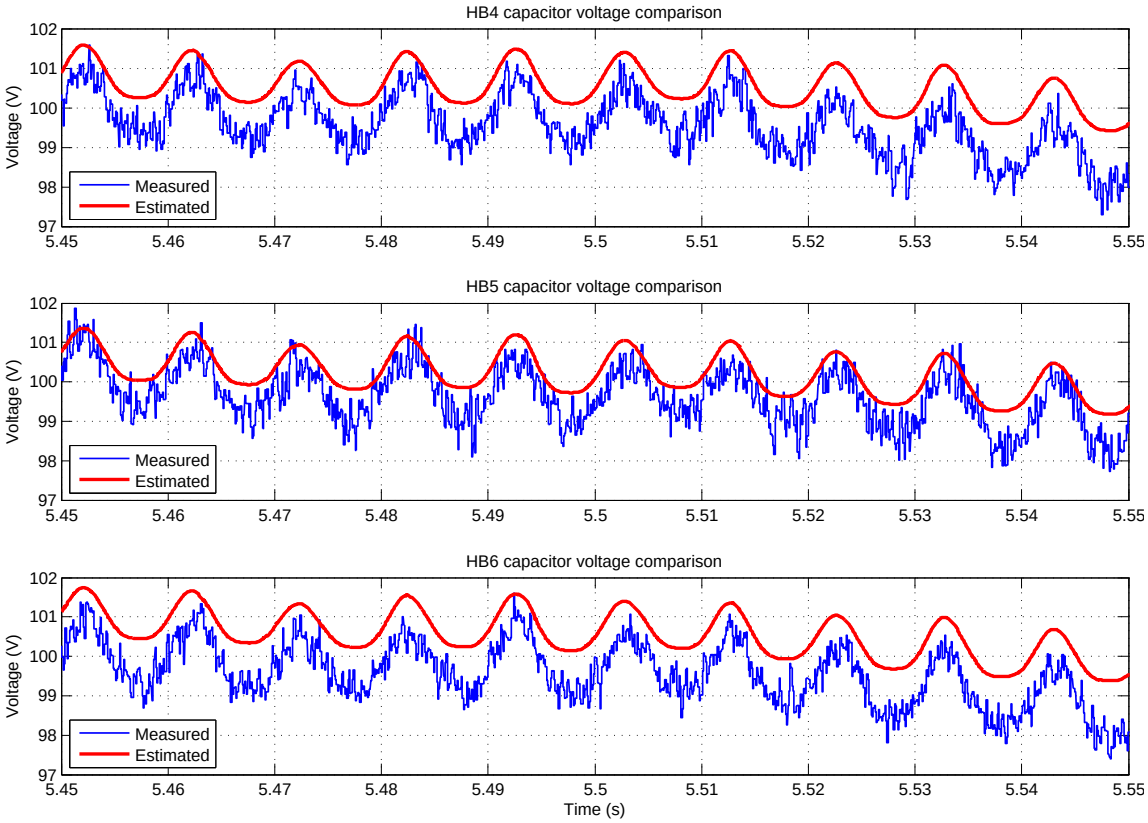


Figure 5.47: eRTS lower cell voltage estimation during change from normal operation mode to fault mode at 5.5*s* - Detail

5.3.5.5 *Output voltage reconstruction from estimated capacitor voltages*

The eRTS estimated voltages can be used for output voltage reconstruction. Similarly as it has been done in the *MMC HIL* IP, the output voltage can be determined by using the estimated capacitor voltages, the measured currents and input voltage. Figures 5.48 and 5.49 show a comparison between the captured data using a *Yokogawa DL850 ScopeCorder* sampling the signals at $1\mu s$, and the estimated output voltage derived from the *Capacitor voltage eRTS* and computed using MATLAB as a proof of concept.

In the zoomed Figure 5.48 small differences are observed when the pulses are too short. This might be caused because of two reasons: (i) the eRTS had no sufficient resolution to capture them (remember that the reconstruction of the signal has been performed using the data logged signals from the eRTS); and/or (ii) the dead-band –or interlock– implemented in hardware inside the CPLD present on each HB, that prevents both switches to be turned on at the same time, which causes rejecting pulses which are smaller than $2\mu s$. Nevertheless, the reconstruction of the signal is considerably good, which can have interesting applications if the output voltage measurement fails.

The shape of the signal is perfectly followed as seen in 5.49 even when the eRTS is switched in at 5.5*s* with only a slight voltage difference when reaching the highest voltage magnitude.

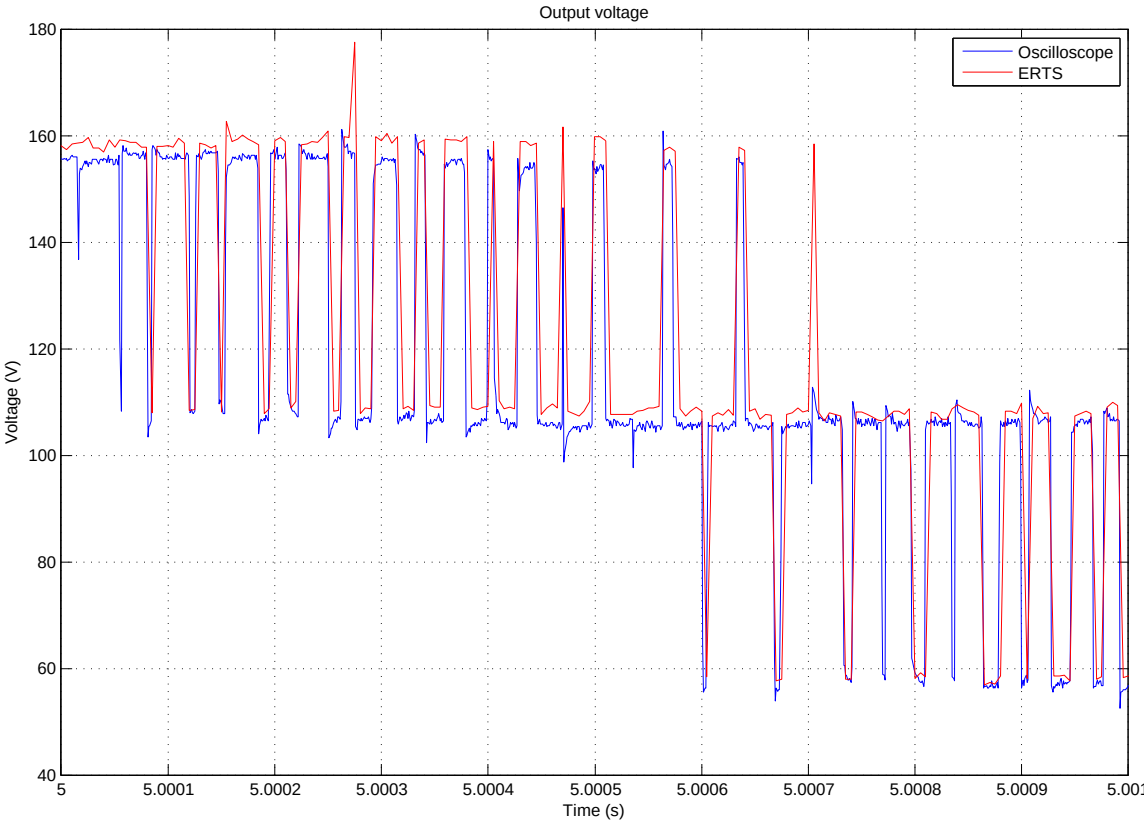


Figure 5.48: V_{out} reconstruction

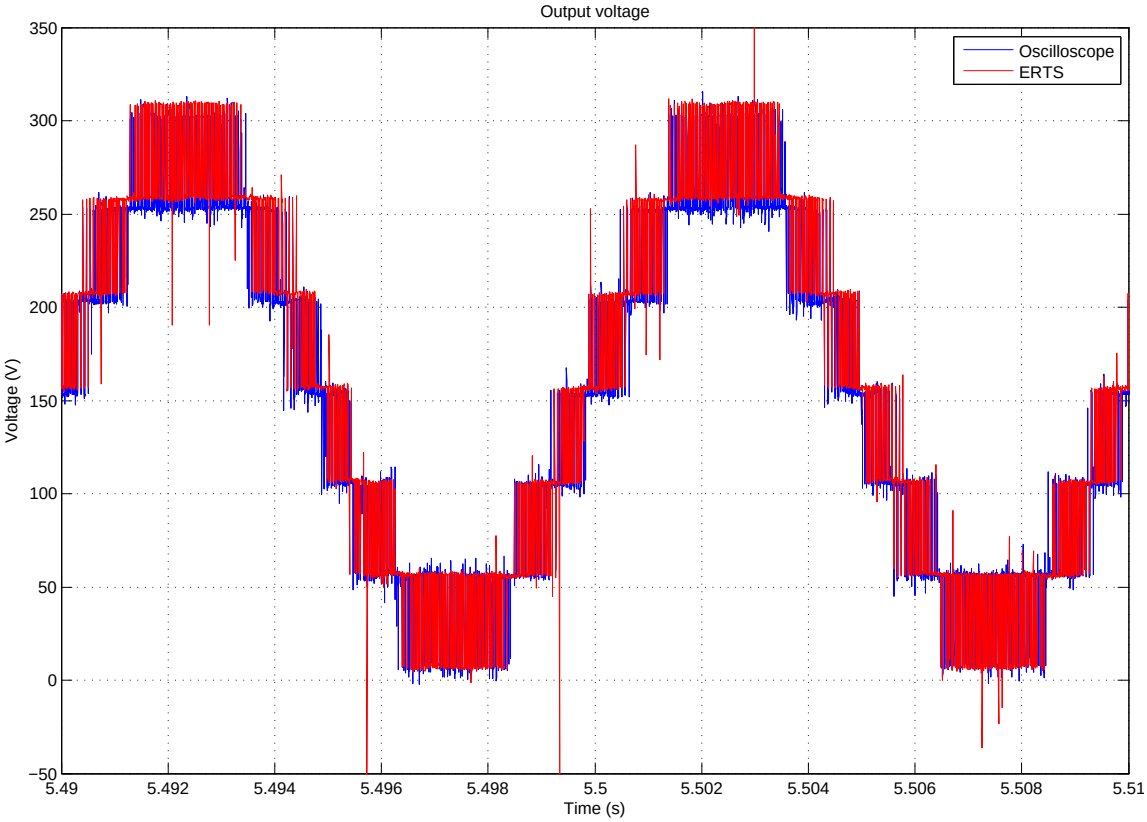


Figure 5.49: V_{out} reconstruction switching the Capacitor voltage eRTS in at 5.5*s*

5.4 CHAPTER CONCLUSIONS

In this Chapter the experimental validation of the use of SoC devices for implementing eRTS has been shown. Firstly, the hardware/software co-design utilised to put in operation the low-power converter was detailed. Finally, all the steps that were followed to develop, test, and validate the different converter models were explained and results presented.

Given the many possibilities a SoC provides to implement such complex control systems, a partition methodology was suggested depending on the tasks to be performed by each subsystem. Three classifications were outlined: (i) the basic blocks executing simple tasks and requiring the fastest execution time possible (e.g. *PWM generator*, *CIC filter*, *Capacitor overvoltage alarm*, etc.) were implemented in hardware using VHDL; (ii) the blocks with average complexity, with only a few number of parameters and requiring as well fast execution time with possibility of parallelization (e.g. *MMC HIL*, *Capacitor voltage eRTS*, *CIC conversion to float*, etc.) were developed using HLS tools and implemented in hardware as well; and finally (iii) the parts of the system which perform complicated tasks and will definitely change during the development cycle (e.g. controllers, communications, data logging on external devices, etc.) were implemented in software and executed in the PS.

Apart from the whole ecosystem built to put them in operation, two model IPs were developed using HLS tools. The first one was a complete MMC model which estimates output voltage, branch currents, and capacitor voltages based on the PWM signals and the input DC voltage. This IP was used as HIL in order to validate the converter's controller in real-time before its deployment on the real test rig. The regulator was validated and tested in the experimental prototype satisfactorily.

The second IP was a capacitor voltage eRTS including a current offset estimator. This time, the entity was used in a real application of fault-tolerant control corroborating its validity when the measured capacitor voltages are not received in the control platform, thus using the estimated ones instead.

To finalize, a proof of concept of the output voltage reconstruction using these estimated voltages was outlined, demonstrating another possible utilisation of eRTSs.

Regarding execution time of the IPs under study, they were capable of providing the calculation results and write them in memory below the microsecond. However, in order to be in the safe path, simplify and ensure the start-up of the whole system, a conservative $5\mu s$ execution time was considered. Nonetheless, very satisfactory results were obtained.

Even though the device used in this experimental setup is considered as low/mid-range, the FPGA fabric resources utilised were all below 50% but the LUT, which area usage raised up to 68% considering both HIL and eRTS IPs being implemented in the same single design. This can give an idea of how suitable these systems are to implement complicated control systems as the one presented in this dissertation.

Part IV

GENERAL CONCLUSIONS AND PERSPECTIVES

6

GENERAL CONCLUSIONS

The objective of this dissertation was to evaluate and analyze how suitable are SoC devices to implement eRTSs of electrical systems.

After a general introduction outlined in Chapter 1, in Chapter 2 an overview of the State of the Art of eRTSs was given, pointing to the most important challenges related to their development and implementation, and picturing where this Thesis work is framed. Several SoC platforms were briefly introduced plus the development tools that come with them, and the reasons to choose the *Zynq-7000* SoC and Vivado HLS to be used in this dissertation were stated.

Chapters 3 and 4 were devoted to the evaluation of two case scenarios related with *offshore wind farm* generation: one focusing on electromechanical systems and the other on power electronic systems. The first example was a DFIG and the study comprehended two discretization methods: Euler's Forward method and the bilineal or Tustin method. Results showed that the results of the Euler version were accurate enough considering the low/medium dynamics of these kinds of systems. Next, an implementation study was carried out to see if it was more convenient to run the model in the PS or the PL. Moreover, a data transfer study was also realized to find if it was interesting to have this IP running as a hardware accelerator. Furthermore, different data types were evaluated as well: 64/32-bit floating-point and 32-bit fixed-point data formats. Results showed that the best results were found when running the model in the ARM using 64-bit floating-point variables which only needed $372ns$ to perform all the model computations. The second example was an implementation of an MMC. In this case however, only a PL evaluation was envisaged due to the harsh dynamics of power electronic systems. A broad study was carried out considering the same three data formats used in the previous case, and changing the number of SM per phase from 6 to 700. They were analyzed in terms of resources usage, execution time and precision of the results compared to a PSCAD simulation. Results showed that even though the fixed-point version was much faster, the maximum amount resources (DSPs) were used up at 200 HB per phase, without using any directive or *pragma*. Conversely, both floating-point versions were capable of managing 700 SMs per phase without even reaching 30% of resources also without using any directive. The absolute errors compared to a 64-bit PSCAD version were below 0.4% for both floating-point versions, whereas the fixed-point version got a bit over 3%.

In Chapter 5 was outlined all the hardware/software ecosystem necessary to set up the MMC low-power test bench. Given the many possibilities a SoC provides in terms of implementation, a partition methodology was suggested depending on the tasks to be performed by each subsystem. Three classifications were outlined: (i) The basic blocks executing simple tasks and requiring the fastest execution time possible were implemented in hardware using VHDL; (ii) the blocks with average complexity, with only a few number of parameters and requiring as well fast execution time with possibility of parallelization were developed using HLS tools and implemented in hardware as well; and finally (iii) the parts of the system

which perform complicated tasks and will definitely change during the development cycle were implemented in software and executed in the PS. Regarding execution time of the IPs under study, they were capable of providing the calculation results and write them in memory below the microsecond. However, in order to be in the safe path, simplify and ensure the start-up of the whole system, a conservative $5\mu s$ execution time was considered. Nonetheless, very satisfactory results were obtained. Then, two eRTSs were introduced: one utilised as HIL for validating the converter controller; and the other to be used in an application of fault-tolerant control. The *MMC HIL* was successfully implemented and used to verify the proper functioning of the controller, ratifying its validity in a later test using the experimental prototype. Finally, a *Capacitor voltage eRTS* used to estimate and compensate the capacitor voltage drift was implemented and tested with the real system as well. There, a failure in retrieving the voltage measurements was induced changing the eRTS IP to *fault mode* and modifying the control inputs to retrieve the capacitor voltages from the ones estimated by the IP. The IP performed significantly well having a maximum error of 4% in the worst case after 1 second of operation.

7

PERSPECTIVES

In this final Chapter, author wants to outline some ideas of how this work could be continued and/or improved.

7.1 MINIMISE THE ERTS EXECUTION TIME

Author is confident that the whole system could run at $1\mu s$ –or even less– by slowing down the FPGA clock in order to allow execute operations with less latency –or even zero latency as outlined in Section 3.6, increasing the *AXI* interface clock to perform data transfer faster –the maximum clock frequency is $250MHz$, automating tasks on hardware to free the PS from doing them, and by using the second processor to divide the computational burden of the controller by two.

7.2 IMPROVE ERTS MODEL

In Subsection 5.3.5 a *Capacitor voltage eRTS* was developed and put in operation. The $470\mu F/400V$ aluminum electrolytic capacitors by *EPCOS* have a capacitance tolerance of 20%. The capacitor voltage estimation could be significantly improved if the capacitance of each pair of capacitors (remember that there are two per SM connected in parallel) is identified and the values used in the eRTS model.

Moreover, the model could include the converter non-linearities (dead-time) and increase the PWM granularity.

7.3 INCREASE THE NUMBER OF SM TO BE CONTROLLED

At this moment, the test bench is only managing 6 SMs in total, even though it is prepared and physically configured to command 12 by doing very few modifications, or 24 with a little more of work. This way, it could be tested one single leg with 12 HB per arm, or it could also be implemented all the three phases by using 4 SM per phase, hence 2 per arm. Regarding FPGA resources and considering the conservative $5\mu s$ chosen, the system could be able to perform all the computations necessary to double the number of cells without major problems.

However, if the system wants to be increased in order to manage 24 HBs, more work would need to be done, modifying not only the IPs, but also the whole hardware/software co-design and probably the data logger would have to be improved by either reduce the amount of data registered, or by lowering the frequency at which it is registered. Furthermore, managing 24 SMs would lead to have available a more-close-to-reality three-phase converter, handling 8 SMs per phase –hence 4 per arm, being capable of producing a 9-level output voltage per phase.

7.4 IMPLEMENT NEW CONVERTER TOPOLOGIES

The double-T DC-DC MMC topology for high-voltage DC applications presented in [89] could be tested in our facilities using the same development procedure followed in this Thesis work and explained in Chapter 5. Firstly, by the development of a HIL IP in order to verify that developed controller performs well in real-time, and secondly by assembling and testing this characteristic converter topology in our low-power test rig.

The only new thing that would eventually take more time to be developed is the management of the output branch of the converter, which is formed by Full-Bridge (FB) SMs instead of the Half-Bridge SMs that form the input series and the derivation branches. This would require not only the development and implementation of the FB model, but also to decide how to manage the PWM outputs considering that these SMs have 4 possible states instead of the 2 of the HB.

7.5 TEST NEW CONTROL STRATEGIES

The controller used in this Thesis work was based on the converter introduced in the previous Subsection [89], which goal was to generate an AC voltage plus a DC voltage at the middle point of connection. Other more demanding control strategies like the heuristic model predictive modulation explained in [90] could be tested in our platform. Also, the improvement of the capacitor balancing algorithm by using the one-dimensional cell inversion presented [91] could reduce significantly the capacitor voltage ripple. Moreover, an optimization of the switching losses and the capacitor voltage ripple using model predictive control outlined in [92] could be as well implemented to evaluate even further the computational capabilities of the controller platform.

7.6 ERTS CURRENT ESTIMATOR FOR FAULT-TOLERANT CONTROL

This eRTS was as well one of the objectives of this Dissertation. However, due to a lack of time it was not possible to be included in the experimental results.

This could be used in a similar way as the *Capacitor voltage eRTS* presented in 5.3.5, where a faulty current sensor would be replaced by the *eRTS current estimator* in order to continue operating the converter.

Moreover, it could also include a fast fault-detection and isolation algorithms to run in FPGA for both voltage and current eRTSs.

Part V

APPENDIX

A

EXPERIMENTAL TEST BENCH

A.1 INTRODUCTION

There were developed HB modules which included the power switches, the capacitors, and all the electronics necessary to manage them using optical fibers, employing two to drive the switches and one to transmit the capacitor voltage back to the control card. Moreover, several additional cards were built to handle these signals, perform current and voltage measurements, and include hardware-controlled protection to avoid the system to be damaged in the event of software and/or hardware failures. Figures A.1 and A.2 show the diagram of the complete system and the interconnections between all the blocks that form the experimental setup.

MOSFETS have been utilised instead of IGBTs as the forward voltage drop of IGBTs in low-power applications is proportionally too high. At low voltages, MOSFETS exhibit a forward voltage drop proportionally comparable to that of high power IGBT based MML power converters.

The values of the passive elements present in the test rig are the following: Both L_{arm} are $5mH$ inductances produced by *Jesiva Transformers*[1], an Spanish company that produces inductances and converters on demand; C, the SM capacitors, are all two $470\mu F/400V$ capacitors by *EPCOS AG* connected in parallel, hence $940\mu F/400V$; C_2 is a $5\mu F/1.3kV$ ceramic capacitor by *EPCOS AG* as well used for noise filtering in the DC bus; an LC_1 filter was added and tuned at the $100Hz$ of the circulating current using a $5mH$ inductor by *Jesiva Transformers* and a $470\mu F/400V$ capacitor by *EPCOS AG*; finally, the load is a simple electric heater with a measured resistance of 44Ω. This choice was made because it was found that the linearity of the power resistors was not as good as this one.

With respect to the control board, it was decided to use *Avnet*'s *MicroZed I/O Carrier Card* [93], featuring a *Zynq-7020* produced by *Xilinx Inc*. It has twelve PMOD connectors where it can easily be connected expansion modules like the fiber optics interface and the overcurrent and overvoltage protection card that are described later in this Chapter.

Concerning the time constraints, even though it would be feasible to get the ERTS working at $1\mu s$, it was decided to be more conservative and set the ERTS execution time to $5\mu s$ using a single clock of $100MHz$ to drive all the FPGA IPs. The PWM frequency was set to $5kHz$, hence $200\mu s$, but using a double-update method being able to recalculate and change the duty cycle in the highest and lowest value of the triangular waveform [88]. Therefore, the control will be executed at double that frequency: $100\mu s$.

Figure A.3 shows a picture of the experimental prototype built at the AI2 facilities.

Following, the complete system used for validating the ERTS is explained in detail.

1 Official website *jesiva.com*

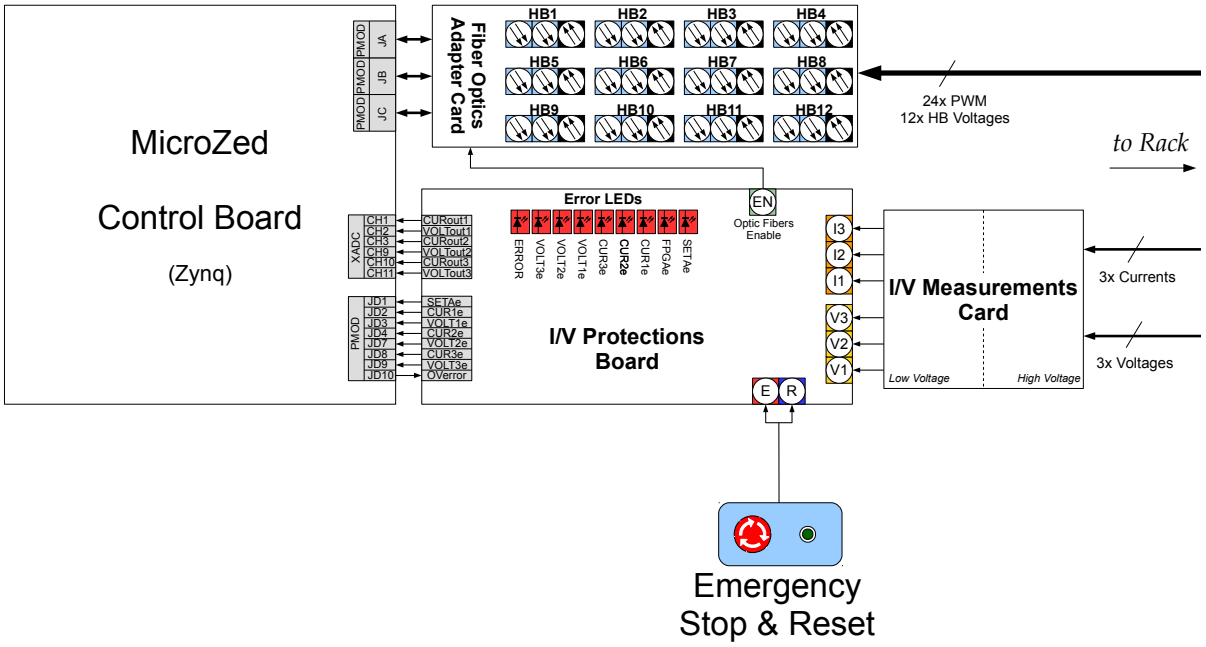


Figure A.1: Test bench main diagram - Control system

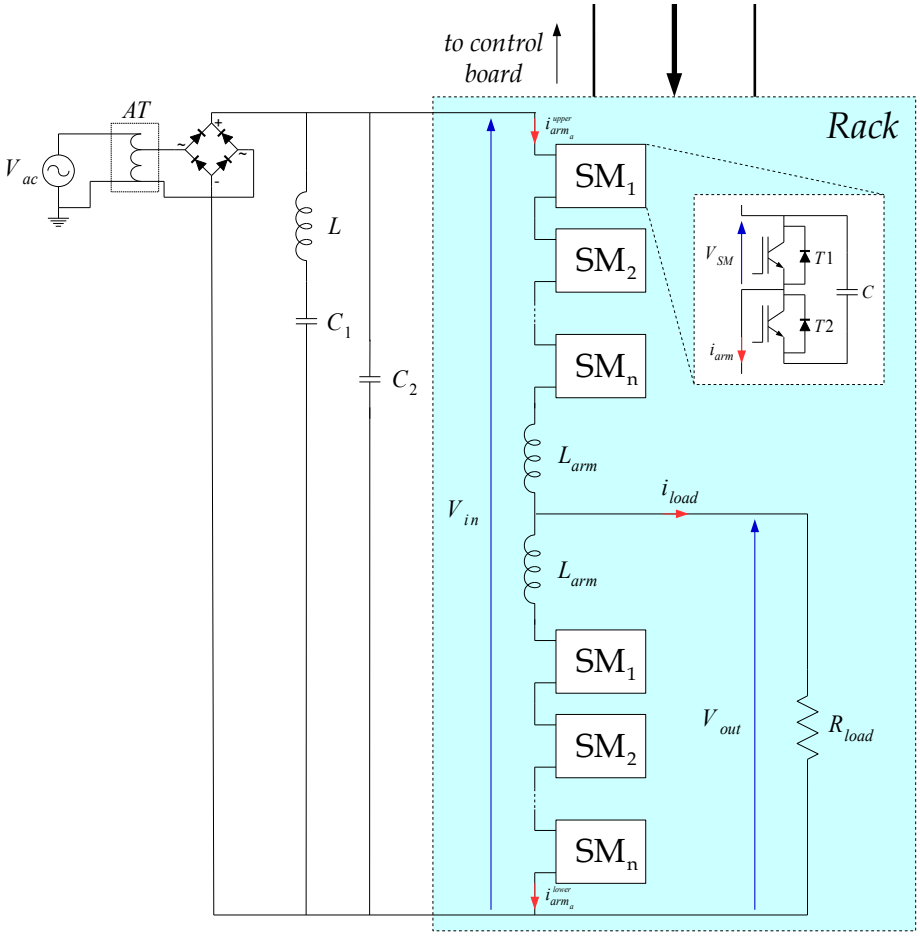


Figure A.2: Test bench main diagram - Power converter

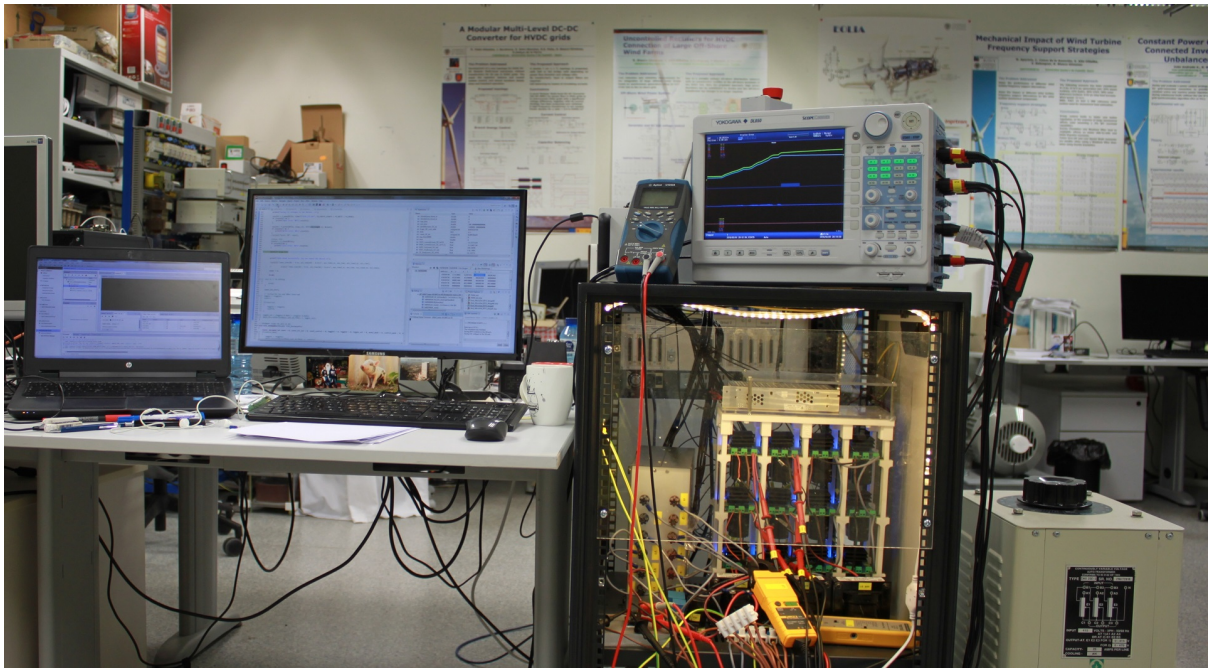


Figure A.3: Picture of the experimental prototype

A.2 TEST BENCH DESCRIPTION

A.2.1 *The Half-Bridge sub-module*

The MMC developed in the AI2 laboratory is composed of twelve HBs mounted in a 3D printed structure forming a rack. In one side of this block (see Figure A.4) there are all the power connections accessible in order to ease the interconnections between sub-modules to create different converter topologies. In the opposite side (see Figure A.5) there are the optical fiber transceivers used to control each HB plus the $12V$ connector for powering the onboard electronics, which employ a single $12V$ power supply to feed all of them.

The sub-module shown in Figure A.6 mounts two MOSFET *TK62N60W* produced by *Toshiba*, which includes in the same package an anti-parallel diode [94]. It has all the electronics necessary to drive the switches using fiber optics signals in order to isolate the converter from the main control, which is also developed in-house.

It has two $470\mu F$ capacitors connected in parallel rated at $400V$, thus forming a $940\mu F$ total capacitor. The voltage measurement is performed using the *ACPL-7970* optically isolated *Sigma-Delta* ($\Sigma\Delta$) modulator produced by *Broadcom* [95]. It is a 1-bit, second-order amplifier that converts an analog input signal into a high-speed data stream through integration of galvanic isolation based on optical coupling technology. It is fed by a $10MHz$ clock. It has been included also a hardware dead-band protection –or interlock– that avoids both switches to be connected simultaneously.

Figure A.7 displays a diagram of the complete rack and its fiber optics interface.

A.2.2 *Control System*

The control system is formed by four individual boards, all interconnected from one another:

- Control board

Figure A.4: Rack picture - Front

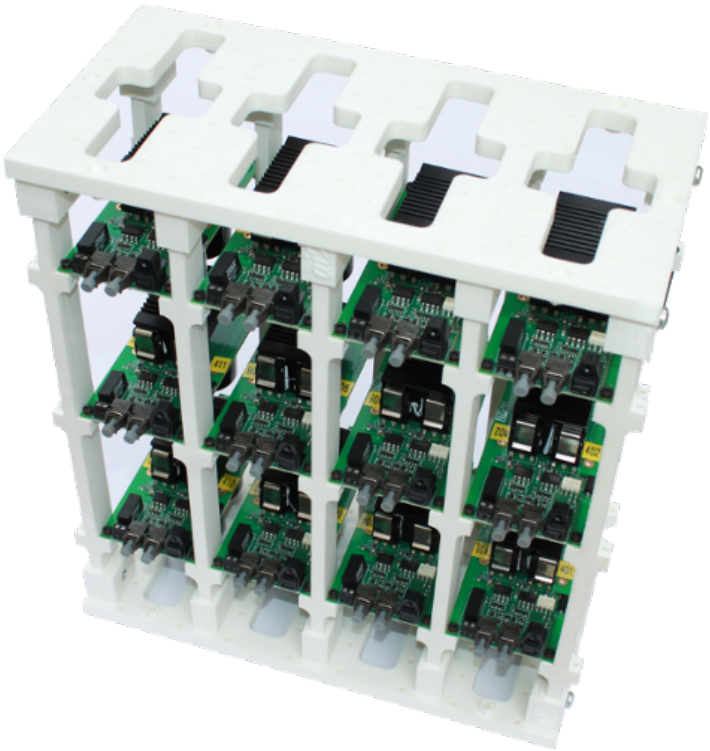


Figure A.5: Rack picture - Back

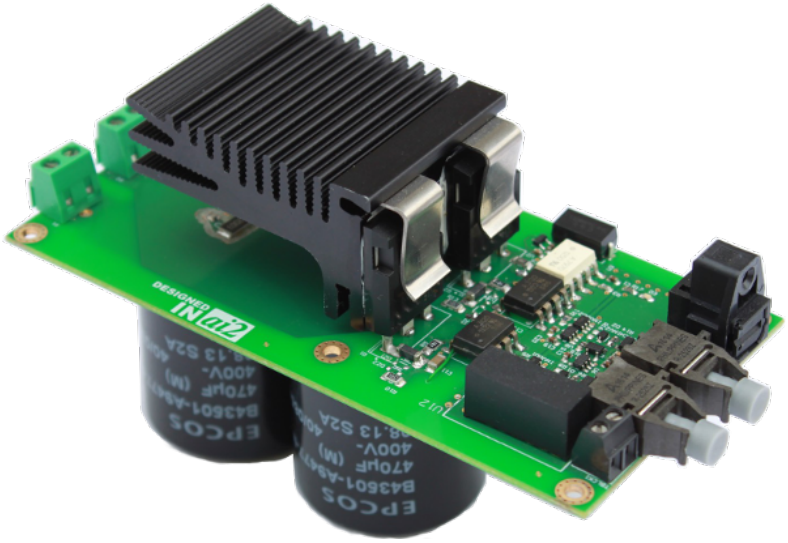


Figure A.6: Half-Bridge sub-module picture

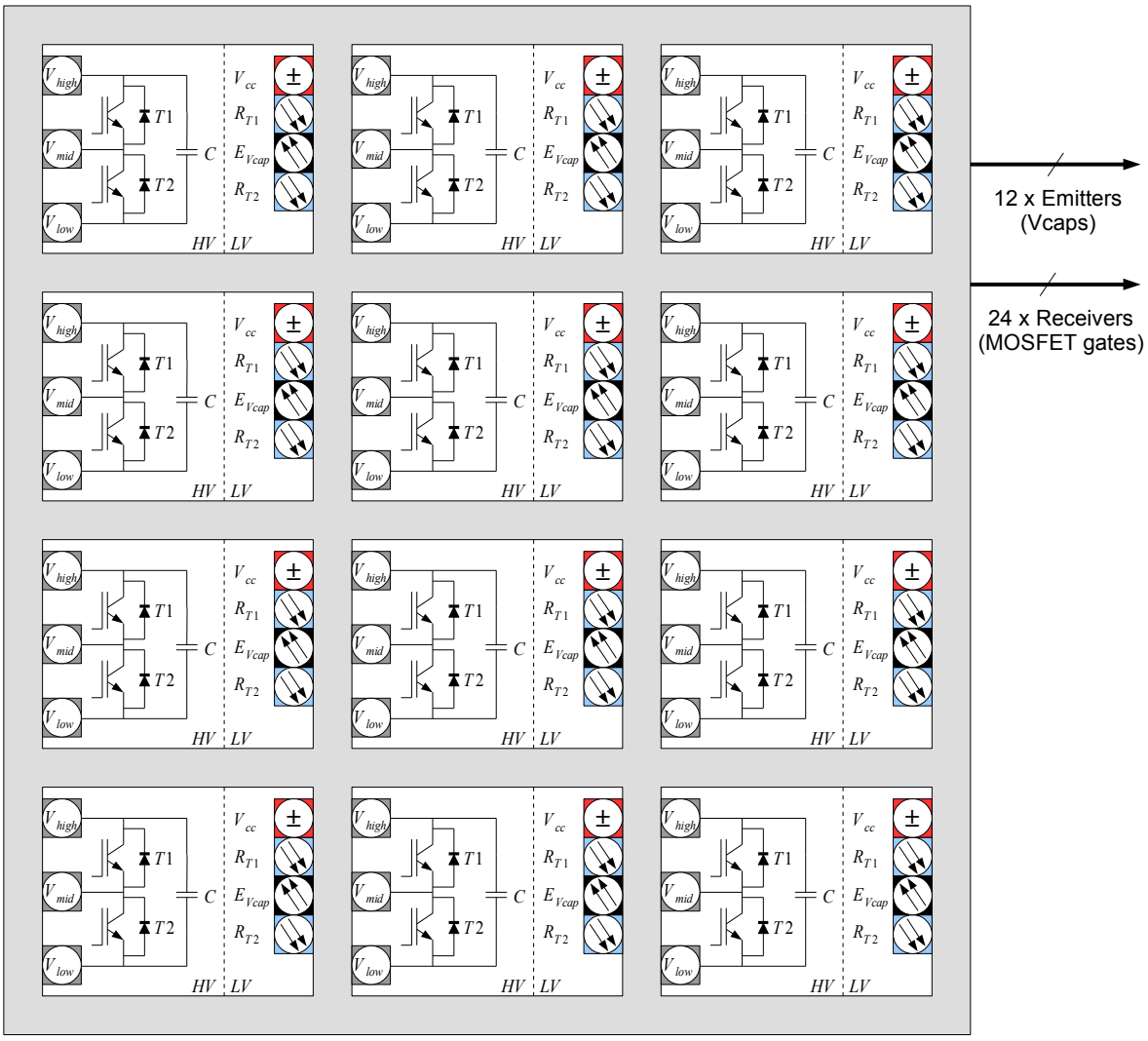


Figure A.7: Half-Bridge rack diagram

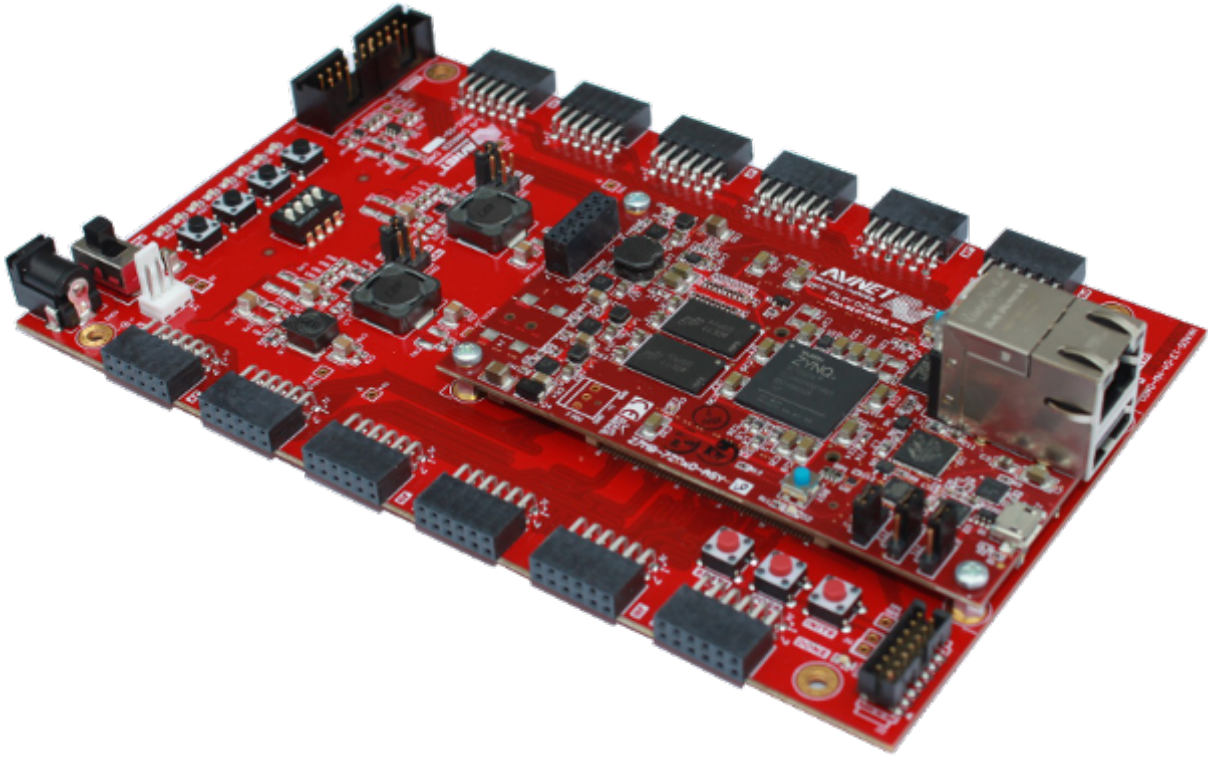


Figure A.8: Control board picture

- Fiber optics interface
- Protection board
- Sensor box

Following, a detailed description of all these subsystems.

A.2.2.1 *Control board*

In order to control the MMC, it was decided to use *Avnet*'s[2] *MicroZed I/O Carrier Card* [93], featuring a *Zynq-7020* produced by *Xilinx Inc*. As shown in Figure A.8, it has twelve PMOD connectors where it can easily be connected expansion modules like the fiber optics interface and the overcurrent and overvoltage protection card.

A.2.2.2 *Fiber optics interface*

As shown in Figure A.9, the fiber optics interface card uses three of the carrier card PMODs to access 24 I/O of the *Zynq*: 12 are used as outputs to drive the HBs, and the remaining 12 as inputs to read the capacitor voltages of every sub-module. The 12 outputs are inverted and connected to 24 optical fiber emitters to drive each of the 24 MOSFETs. Therefore, each SM receives two complementary PWM signals. This was done in order to add some redundancy, so in case one of the optical fiber links fails, it would still be possible to partially control the HB.

Regarding the capacitor voltage measurement, the 12 optical fiber receivers read the frequency-modulated signal generated by the $\Sigma\Delta$ ADC. Then, a configurable *Cascaded Integrator-Comb*

2 Official website *avnet.com*

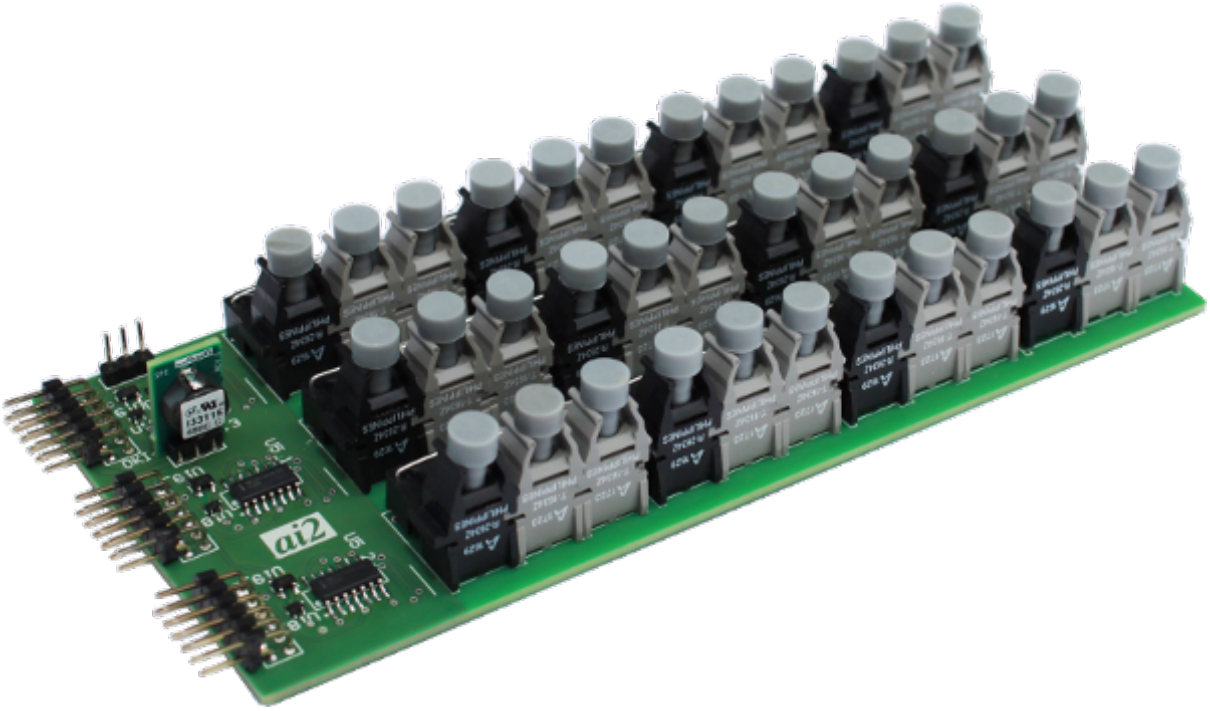


Figure A.9: Fiber optics interface

(CIC) filter [86] implemented in the fabric decodes this signal providing an accurate voltage using very little hardware. More information about the filter can be found in 5.2.2.5.

This PCB includes as well an enable pin (EN) connected to the protection board which disables the switch signals in case an overvoltage or overcurrent is detected.

A.2.2.3 *Protection board*

This card pictured in Figure A.10 is placed in between of the converter and the *Zynq* being able of cutting the MOSFET's signals if an error is detected. Its main purpose is to supervise voltages and currents and block the optical fiber signals if any has surpassed the predefined maximum value. All the signals are monitored using only hardware devices, i.e. several *operational amplifiers* (OPAMP) and a *Complex Programmable Logic Device* (CPLD). This means it is not software dependent and hence, code and bug errors are avoided and the response time of the alarms are treated as fast as possible.

In total, this PCB monitors three currents and three voltages, plus the capacitor voltages which are pre-checked inside the *Zynq*'s PL. It counts as well with an emergency button plus a reset in order to re-activate the system. Additionally, the board has nine status LEDs that help the user to know what signals are causing the system's halt. In Figure A.11 is displayed a diagram of the board showing all its inputs and outputs.

Entering more in detail, the OPAMPs are configured as differential amplifiers to reduce the range of the input signals in a first stage, and as comparators to verify maximum values that could damage the converter in a second stage. The purpose of the CPLD is to manage the logic signals coming from the OPAMPs, light up the corresponding error LED, cut the fiber optics signals, and warn the *Zynq* that an error has occurred.

The three currents and three voltages that are monitored on-board are converted from the $\pm 10V$ coming from the sensor box to a 0-to-1 volts range. Then, each of these six signals is driven into two comparators each, which check they do not surpass a minimum and maximum limit value configured using two multi-turn potentiometers. Hence, if any of these

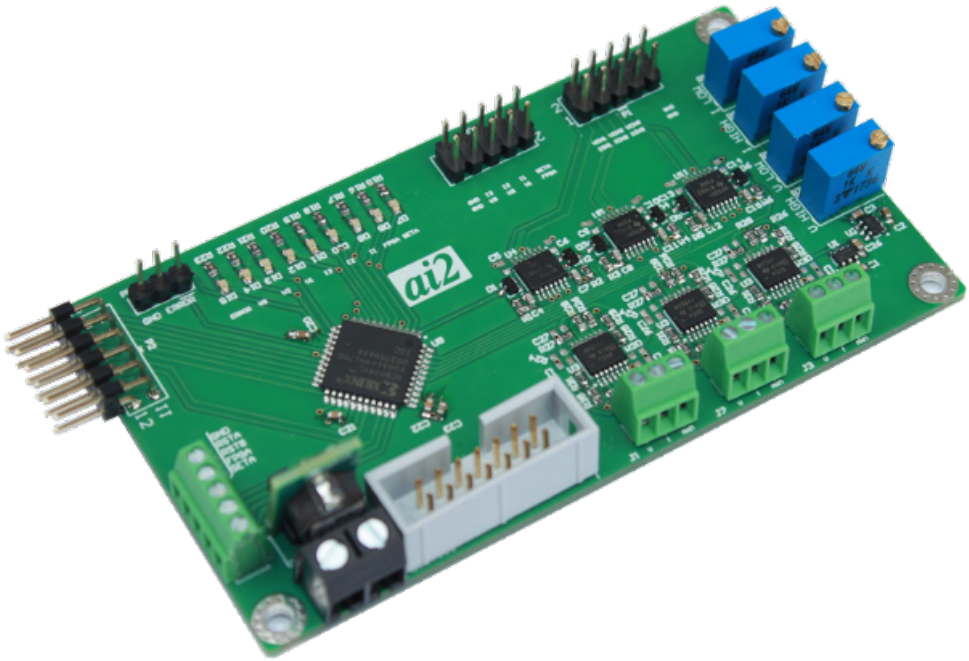


Figure A.10: Protection board

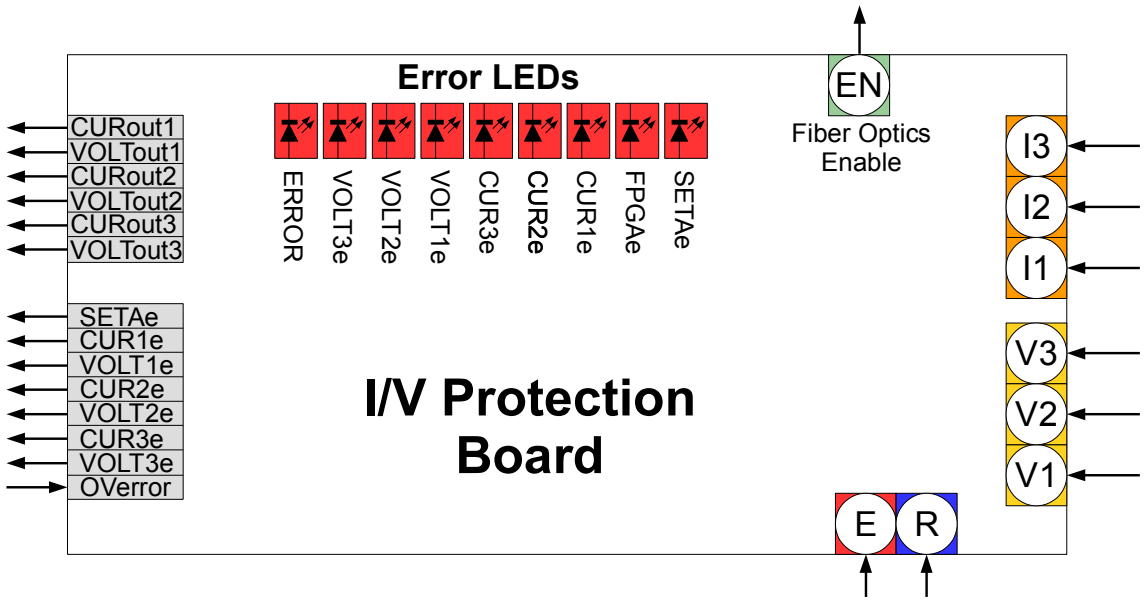


Figure A.11: Protection board diagram

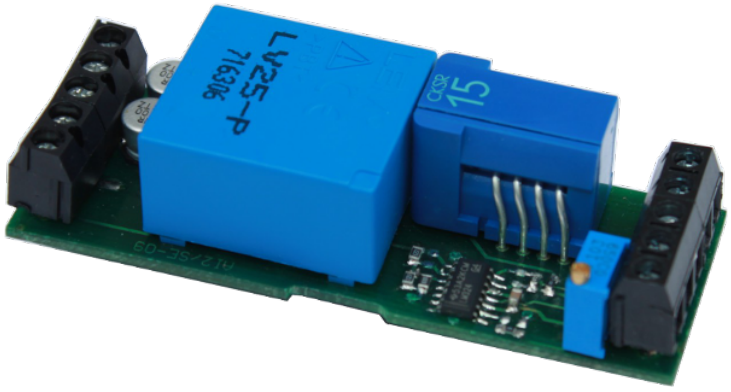


Figure A.12: Current and voltage sensor

values is reached, the output of the comparators change their logic value and the CPLD acts accordingly cutting the fiber optics signals.

Moreover, these six 0-to-1 volts signals are also connected to the *Zynq*'s ADC through PMODs JE and JF, so it can perform simultaneous reads on channels 1 and 9, channels 2 and 10, and channels 3 and 11.

A.2.2.4 *Sensor box*

This sensor box, which includes three sub-modules like the one shown in Figure A.12, converts measurements of three currents and three voltages to $\pm 10V$ signals. It uses the LTA 50-P/SP1 Hall effect transducer to measure DC and AC currents, and the LV25-P voltage transducer for voltage measurements, both produced by *LEM International SA*. All the electronics are installed inside a metallic box in order to avoid electromagnetic interferences and placed next to the MMC. Inputs and outputs are galvanically isolated. The maximum currents and voltages that can be measured are $\pm 31A$ and $\pm 500V$ respectively.

B

MMC STATE SPACE MODEL PARAMETERS

These are the parameters used in the *MMC HIL* state space model described in Subsection 5.3.1.2.

Matrices used for the *charging mode*: no-load present ($1M\Omega$ load actually) used for charging the SM capacitors:

$$A = \begin{bmatrix} 0.49995 & 0.49995 \\ 0.49995 & 0.49995 \end{bmatrix}$$

$$B = \begin{bmatrix} 4.5477e^{-4} & -4.5477e^{-4} & -4.5427e^{-4} \\ 4.5427e^{-4} & -4.5427e^{-4} & -4.5477e^{-4} \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1e^{6} & -1e^{6} \end{bmatrix}$$

Matrices used for the *normal operation mode* (44Ω resistive load):

$$A = \begin{bmatrix} 9.6146e^{-1} & 3.8438e^{-2} \\ 3.8438e^{-2} & 9.6146e^{-1} \end{bmatrix}$$

$$B = \begin{bmatrix} 8.9134e^{-4} & -8.9134e^{-4} & -1.7705e^{-5} \\ 1.7705e^{-5} & -1.7705e^{-5} & -8.9134e^{-4} \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 44 & -44 \end{bmatrix}$$

BIBLIOGRAPHY

[1] Quora, What is an ip intellectual property core in VLSI (apr 2018). doi:https://www.quora.com/What-is-an-IP-intellectual-property-core-in-VLSI.

[2] H. Saad, S. Dennetiere, J. Mahseredjian, P. Delarue, X. Guillaud, J. Peralta, S. Nguefeu, Modular multilevel converter models for electromagnetic transients, IEEE Transactions on Power Delivery 29 (3) (2014) 1481 – 1489. doi:10.1109/TPWRD.2013.2285633.

[3] R. Pena, J. Clare, G. M. Asher, Doubly-fed induction generator using back-to-back pwm converters and its application to variable-speed wind-energy generation, in: IEE Proceedings - Electric Power Applications, IEEE, 1996, pp. 1350–2352.

[4] R. Pena, J. Clare, G. Asher, A doubly fed induction generator using back-to-back pwm converters supplying an isolated load from a variable speed wind turbine, in: IEEE Proceedings - Electric Power Applications, Vol. 143, IEEE, 1996, pp. 380–387.

[5] M. E. Achkar, R. Mbayed, G. Salloum, N. Patin, S. L. Ballois, E. Monmasson, Modeling and control of a stand alone cascaded doubly fed induction generator supplying an isolated load, in: IEEE (Ed.), Power Electronics and Applications (EPE'15 ECCE-Europe), 17th European Conference on, 2015.

[6] D. Tormo, L. Idkhajine, E. Monmasson, R. Blasco-Gimenez, Evaluation of SoC-based embedded real-time simulators for electromechanical systems, in: IEEE (Ed.), The 42nd Annual Conference of IEEE Industrial Electronics Society, 2016.

[7] D. Tormo, L. Idkhajine, E. Monmasson, R. Blasco-Gimenez, Embedded real-time simulator implementations of electromechanical systems using system-on-chip devices, in: ELECTRIMACS, 2017.

[8] M. Dagbagi, A. Hemdani, L. Idkhajine, M. Naouar, E. Monmasson, I. Slama-Belkhodja, ADC-based embedded real-time simulator of a power converter implemented in a low-cost FPGA: Application to a fault-tolerant control of a grid-connected voltage-source rectifier, IEEE Transactions on Industrial Electronics 63 (2) (2016) 1179–1190.

[9] C. Liu, X. Guo, F. Gao, E. Breaz, P. Damien, F. Gechter, FPGA based real-time simulation of high frequency soft-switching circuit using time-domain analysis, in: Industrial Electronics Society , IECON - 42nd Annual Conference of the IEEE, 2016. doi:10.1109/IECON.2016.7793227.

[10] F. Montano, T. Ould-Bachir, J.-P. David, An evaluation of a high-level synthesis approach to the FPGA-based sub-microsecond real-time simulation of power converters, IEEE Transactions on Industrial Electronicsdoi:10.1109/TIE.2017.2716880.

[11] A. Lesnicar, R. Marquardt, An innovative modular multilevel converter topology suitable for a wide power range, in: Power Tech Conference Proceedings, IEEE Bologna, 2003. doi:10.1109/PTC.2003.1304403.

[12] H. Akagi, Classification, terminology, and application of the modular multilevel cascade converter, IEEE Transactions on Power Electronics 26 (11) (2011) 3119–3130.

[13] K. Sharifabadi, L. Harnefors, H.-P. Nee, S. Norrga, R. Teodorescu, Design, Control and Application of Modular Multilevel Converters for HVDC Transmission Systems, Wiley, 2016.

[14] D. Tormo, R. Vidal-Albalate, L. Idkhajine, E. Monmasson, R. Blasco-Gimenez, Study of system-on-chip devices to implement embedded real-time simulators of modular multilevel converters using high-level synthesis tools, in: IEEE International Conference on Industrial Technology, 2018. doi:https://www.youpak.com/watch?v=8jL0vy2YaDg.

[15] J. Sawma, F. Khatounian, E. Monmasson, R. Ghosn, L. Idkhajine, Evaluation of the new generation of system-on-chip platforms for controlling electrical systems, Industrial Technology (ICIT), 2015 IEEE International Conference on (2015) 1570–1575.

[16] B. Lu, X. Wu, H. Figueroa, A. Monti, A low-cost real-time hardware-in-the-loop testing approach of power electronics controls, IEEE Transactions on Industrial Electronics 54 (2) (2007) 919–931.

[17] W. Li, G. Joos, J. Belanger, Real-time simulation of a wind turbine generator coupled with a battery supercapacitor energy storage system, IEEE Transactions on Industrial Electronics 57 (4) (2010) 1137–1145.

[18] A. Hasanzadeh, C. Edrington, N. Stroupe, T. Bevis, Real-time emulation of a high-speed microturbine permanent-magnet synchronous generator using multiplatform hardware-in-the-loop realization, IEEE Transactions on Industrial Electronics 61 (6) (2013) 3109–3118.

[19] A. Schmitt, J. Richter, U. Jurkewitz, M. Braun, FPGA-based real-time simulation of nonlinear permanent magnet synchronous machines for power hardware-in-the-loop emulation systems, in: Industrial Electronics Society, IECON -40th Annual Conference of the IEEE, IEEE, 2014, pp. 3763–3769.

[20] V. D. Colli, R. D. Stefano, F. Marignetti, A system-on-chip sensorless control for a permanent-magnet synchronous motor, IEEE Transactions on Industrial Electronics 57 (11) (2010) 3822–3829. doi:10.1109/TIE.2009.2039459.

[21] I. Bahri, A. Maalouf, L. Idkhajine, E. Monmasson, FPGA-based implementation of sensorless AC drive controllers for embedded electrical systems, in: Sensorless Control for Electrical Drives (SLED), 2011 Symposium on. doi:10.1109/SLED.2011.6051539.

[22] I. Bahri, L. Idkhajine, E. Monmasson, M. E. A. Benkhelifa, Hardware/software codesign guidelines for system on chip FPGA-based sensorless AC drive applications, Industrial Informatics, IEEE Transactions on 9 (4) (2013) 2165–2176. doi:10.1109/TII.2013.2245908.

[23] E. Monmasson, I. Bahri, L. Idkhajine, A. Maalouf, W. M. Naouar, Recent advancements in FPGA-based controllers for AC drives applications, in: Optimization of Electrical and Electronic Equipment (OPTIM), 2012 13th International Conference on, 2012. doi:10.1109/OPTIM.2012.6231993.

[24] Wikipedia, Programmable logic device (apr 2018).
URL https://en.wikipedia.org/wiki/Programmable_logic_device

[25] M. Dagbagi, FPGA-based embedded real time simulation of electrical systems, Ph.D. thesis, SATIE Laboratory, Université de Cergy-Pontoise, France (oct 2015).

[26] J. W. Liu, Real-Time Systems, Upper Saddle River, NJ: Prentice-Hall, 2000.

[27] Wikipedia, Real-time computing (apr 2018).
URL https://en.wikipedia.org/wiki/Real-time_computing

[28] M. Ricco, M. Gheorghe, L. Mathe, R. Teodorescu, System-on-chip implementation of embedded real-time simulator for modular multilevel converters, in: Energy Conversion Congress and Exposition (ECCE), IEEE, 2017. doi:10.1109/ECCE.2017.8095968.

[29] J. R. Marti, J. Lin, Suppression of numerical oscillations in the emtp, IEEE Transactions on Power Systems 4 (2) (1989) 739–747. doi:10.1109/59.193849.

[30] H. Dommel, Digital computer solution of electromagnetic transients in single-and multiphase networks, IEEE Transactions on Power Apparatus and Systems PAS-88 (1969) 388–399.

[31] J. Sanchez-Gasca, R. D'Aquila, W. Price, J. Paserba, Variable time step, implicit integration for extended term power system dynamic simulation, in: Proceedings of the Power Industry Computer Application Conference, Salt Lake City, UT, USA, 1995, pp. 183–189.

[32] L. Snider, J. Belanger, G. Nanjundaiah, Today's power system simulation challenge: High-performance, scalable, upgradable, affordable COTS-based real-time digital simulators, in: PEDES conference, 2010.

[33] J. Belanger, P. Venne, J. Paquin, The what, where and why of real-time simulation, IEEE PES General Meeting (2010) 25–29.

[34] V. Dinavahi, Real-time digital simulation of switching power circuits, Ph.D. thesis, University of Toronto (2000).

[35] M. Matar, D. Iravani, FPGA implementation of the power electronic converter model for real-time simulation of electromagnetic transients, IEEE Transactions on Power Delivery 25 (2) (2010) 852–860. doi:10.1109/TPWRD.2009.2033603.

[36] G. Li, M. Gevers, Roundoff noise minimisation using delta operator realisations, IEEE Transaction On Signal Processing 42 (2). doi:10.1109/78.193204.

[37] G. Engeln-Müllges, F. Uhlig, Numerical Algorithms with C, Springer, 1996.

[38] J. Liu, V. Dinavahi, A real-time nonlinear hysteretic power transformer transient model on FPGA, IEEE Transactions on Industrial Electronics 61 (7) (2014) 3587–3597. doi:10.1109/TIE.2013.2279377.

[39] A. Myaing, V. Dinavahi, FPGA-based real-time emulation of power electronic systems with detailed representation of device characteristics, IEEE Transactions on Industrial Electronics 58 (1) (2011) 358–368. doi:10.1109/TIE.2010.2044738.

[40] M. Dagbagi, L. Idkhajine, E. Monmasson, I. Slama-Belkhodja, FPGA implementation of power electronic converter real-time model, in: Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM), International Symposium on, Sorrento, Italy, 2012. doi:10.1109/SPEEDAM.2012.6264543.

[41] T. Ould-Bachir, H. Saad, S. Dennetiere, J. Mahseredjian, CPU-FPGA-based real-time simulation of a two-terminal MMC-HVDC system, IEEE Transactions on Power Delivery 32 (2) (2017) 647 – 655. doi:10.1109/TPWRD.2015.2508381.

[42] Y. Chen, V. Dinavahi, Multi-FPGA digital hardware design for detailed large-scale real-time electromagnetic transient simulation of power systems, IET Generation, Transmission & Distribution 7 (5) (2013) 451–463. doi:10.1049/iet-gtd.2012.0374.

[43] H. Saad, T. Ould-Bachir, J. Mahseredjian, C. Dufour, S. Dennetiere, S. Nguefeu, Real-time simulation of MMCs using CPU and FPGA, IEEE Transactions on Power Electronics 30 (1) (2015) 259 – 267. doi:10.1109/TPEL.2013.2282600.

[44] Microsemi Corp., DS0112: SmartFusion Customizable System-on-Chip (cSoC) (mar 2015).

[45] ARM Holdings, ARM Cortex-M3 Technical Reference Manual (2010).

[46] Microsemi Corp., DS0097: ProASIC3 Flash Family FPGAs (mar 2016).

[47] Microsemi Corp., DS0128: IGLOO2 and SmartFusion2 Datasheet (nov 2017).

[48] Intel Inc., Intel User-customizable SoC FPGAs (2018).

[49] ARM Holdings, ARM Cortex-A53 MPCore Processor Technical Reference Manual (feb 2014).

[50] ARM Holdings, ARM Cortex-A9 MPCore Technical Reference Manual (jan 2016).

[51] Xilinx, Inc., Cortex-A9 NEON Media Processing Engine - Rev: r4p1 (2012).

[52] ARM Holdings, Cortex-A9 Floating-Point Unit (2012).

[53] ARM Holdings, ARM Cortex-A5 Floating-Point Unit (apr 2015).

[54] Xilinx Inc., DS190: Zynq-7000 All Programmable SoC Overview (January 2018).

[55] J. Sawma, F. Khatounian, R. Ghosn, L. Idkhajine, E. Monmasson, Quasi-continuous real-time simulation of an RLE load with a current MPC regulation, Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), 2015 Third International Conference on (2015) 289–294.

[56] A. H. plc, AMBA AXI and ACE Protocol Specification (2013).
URL http://infocenter.arm.com

[57] Xilinx Inc., UG585: Zynq-7000 Technical Reference Manual (2016).

[58] Wikipedia, Logic synthesis (apr 2018).
URL https://en.wikipedia.org/wiki/Logic_synthesis

[59] Wikipedia, Place and route (apr 2018).
URL https://en.wikipedia.org/wiki/Place_and_route

[60] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, K. Bertels, A survey and evalutaion of FPGA high-level synthesis tools, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35 (10) (2015) 1591 – 1604. doi:10.1109/TCAD.2015.2513673.

[61] Wikipedia, Register-transfer level (apr 2018).
URL https://en.wikipedia.org/wiki/Register-transfer_level

[62] E. Monmasson, M. N. Cirstea, FPGA design methodology for industrial control systems - a review, IEEE Transactions on Industrial Electronics 54 (4) (2007) 1824–1842.

[63] D. J. Pagliari, M. R. Casu, L. P. Cartoni, Acceleration of microwave imaging algorithms for breast cancer detection via high-level synthesis, in: Computer Design (ICCD), 33rd IEEE International Conference on, 2015. doi:10.1109/ICCD.2015.7357152.

[64] C. Zhang, P. L. Li, G. Guangyu, Y. Guan, B. Xiao, J. Cong, Optimizing FPGA-based accelerator design for deep convolutional neural network, in: 3rd ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2015.
URL http://cadlab.cs.ucla.edu/~cong/slides/fpga2015_chen.pdf

[65] A. Cortes, I. Velez, A. Irizar, High level synthesis using Vivado HLS for Zynq SoC: Image processing case studies, in: Design of Circuits and Integrated Systems (DCIS), Conference on, 2016. doi:10.1109/DCIS.2016.7845376.

[66] H. Jacinto, L. Daoud, N. Rafla, High level synthesis using vivado hls for optimizations of sha-3, in: IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), 2017.

[67] H. Boeui, K. Han-Yee, K. Minsu, X. Lei, S. Weidong, S. Taeweon, FASTEN: An FPGA-based secure system for big data processing, IEEE Design & Test PP (99). doi:10.1109/MDAT.2017.2741464.

[68] B. Anirudh, V. Vivek, R. Abhishek, D. Sumam, Accelerating real-time computer vision applications using HW/SW co-design, in: Computer, Communications and Electronics (Comptelix), International Conference on, 2017. doi:10.1109/COMPTELIX.2017.8004013.

[69] B. C. Schafer, K. Wakabayashi, Machine learning predictive modelling high-level synthesis design space exploration, IET Computers & Digital Techniquesdoi:10.1049/iet-cdt.2011.0115.

[70] Xilinx Inc., UG1270: Vivado HLS Optimization Methodology Guide (dec 2017).

[71] R. Park, Two-reaction theory of synchronous machines generalized method of analysis, Transactions of the American Institute of Electrical Engineers 48 (3) (1929) 716–727.

[72] O. I. Elgerd, Electric Energy Systems Theory: An Introduction, Tata McGraw-Hill, 1971.

[73] Xilinx Inc., UG902: High-Level Synthesis with Vivado HLS (2017).

[74] S. Muller, M. Deicke, R. D. Doncker, Doubly fed induction generator systems for wind turbines, IEEE Industry Applications Magazine 8 (3) (2002) 26–33.

[75] W. L. Brogan, Modern Control Theory, 1st Edition, Quantum Publishers, 1974.

[76] E. W. Weisstein, Laplace transform, Mathworld.
URL http://mathworld.wolfram.com/LaplaceTransform.html

[77] J. Wu, NEON/VFPU hardware acceleration (2012).

[78] B. Andersen, L. Xu, P. Horton, P. Cartwright, Topologies for VSC transmission, Power Engineering Journal 16 (3) (2002) 142 – 150. doi:10.1049/pe:20020307.

[79] U. Gnanarathna, S. Chaudhary, A. Gole, R. Teodorescu, Modular multi-level converter based HVDC system for grid connection of offshore wind power plant, in: AC and DC Power Transmission, 2010. ACDC. 9th IET International Conference on, 2010. doi:10.1049/cp.2010.0984.

[80] U. Gnanarathna, A. Gole, R. Jayasinghe, Efficient modeling of modular multilevel HVDC converters (mmc) on electromagnetic transient simulation programs, IEEE Transactions on Power Delivery 26 (1) (2011) 316 – 324. doi:10.1109/TPWRD.2010.2060737.

[81] J. Peralta, H. Saad, S. Dennetiere, J. Mahseredjian, S. Nguefeu, Detailed and averaged models for a 401-level MMC-HVDC system, IEEE Transactions on Power Delivery 27 (3) (2012) 1501 – 1508. doi:10.1109/PESMG.2013.6672356.

[82] L. Angquist, A. Antonopoulos, D. Siemaszko, K. Ilves, M. Vasiladiotis, H. Nee, Open-loop control of modular multilevel converters using estimation of stored energy, IEEE Transactions on Industry Applications 47 (6) (2011) 2516–2524.

[83] L. Harnefors, A. Antonopoulos, S. Norrga, L. Angquist, H. Nee, Dynamic analysis of modular multilevel converters, IEEE Transactions in Industrial Electronics 60 (7) (2013) 2526–2537. doi:http://dx.doi.org/10.1109/TIE.2012.2194974.

[84] S. Norrga, L. Angquist, K. Ilves, L. Harnefors, H.-P. Nee, Frequency-domain modeling of modular multilevel converters, in: IECON - 38th Annual Conference on IEEE Industrial Electronics Society, 2012. doi:10.1109/IECON.2012.6389570.

[85] R. Vidal-Albalate, E. Belenguer, H. Beltran, R. Blasco-Gimenez, Efficient model for modular multi-level converter simulation, Mathematics and Computers in Simulation (MATCOM) 130 (2016) 167 – 180. doi:https://doi.org/10.1016/j.matcom.2015.10.001.

[86] E. Hogenauer, An economical class of digital filters for decimation and interpolation 29 (2) (1981) 155 – 162. doi:10.1109/TASSP.1981.1163535.
URL http://ieeexplore.ieee.org/document/1163535/

[87] M. Sadri, C. Weis, N. Wehn, L. Benini, Energy and performance exploration of accelerator coherency port using xilinx Zynq, in: FPGAWorld'13 Copenhagen and Stockholm, 2013.
URL http://www.googoolia.com/downloads/papers/sadri_fpgaworld_ver2.pdf

[88] N. Hoffmann, F. W. Fuchs, J. Dannehl, Models and effects of different updating and sampling concepts to the control of grid-connected pwm converters — a study based on discrete time domain analysis, in: Power Electronics and Applications (EPE 2011), Proceedings of the 2011-14th European Conference on, 2011.

[89] R. Vidal-Albalate, J. Barahona, D. Soto-Sanchez, E. Belenguer, R. Peña, R. Blasco-Gimenez, A modular multi-level DC-DC converter for HVDC grids, in: Industrial Electronics Society , IECON - 42nd Annual Conference of the IEEE, 2016. doi:10.1109/IECON.2016.7793043.

[90] C. Townsend, D. Tormo, R. Baraciarte, Y. Yu, H. Z. de la Parra, G. Demetriades, V. Agelidis, Heuristic model predictive modulation for high-power cascaded multi-level converters, IEEE Transactions on Industrial Electronics.

[91] C. D. Townsend, D. Tormo, H. Z. D. L. Parra, One dimensional cell inversion: A modulation strategy for hybrid cascaded converters, in: Energy Conversion Congress and Exposition (ECCE), IEEE, 2014.

[92] C. Townsend, T. Summers, J. Vodden, A. Watson, R. Betz, J. Clare, Optimization of switching losses and capacitor voltage ripple using model predictive control of a cascaded h-bridge multilevel statcom, IEEE Transactions on Power Electronics 28 (7) (2013) 3077–3087.

[93] Avnet Inc., MicroZed I/O Carrier Card (Aug 2014).
URL https://www.avnet.com/opasdata/d120001/medias/docus/58/AES-MBCC-IO-G-IOCC_HW_UG_1p3.pdf

[94] Toshiba Corporation, TK62N60W Power MOSFET (2012).
URL https://toshiba.semicon-storage.com/ap-en/product/mosfet/detail.TK62N60W.html

[95] Broadcom Corporation, ACPL-7970 Optically Isolated Sigma-Delta Modulator (2012).
URL https://www.broadcom.com/products/optocouplers/industrial-plastic/isolation-amplifiers-modulators/sigma-delta-modulators/acpl-79

COLOPHON

This document was typeset using `classicthesis` style developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. It is available for LaTeX and LyX at

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send the author a real postcard; the collection of postcards received so far is featured at

http://postcards.miede.de/

Final Version as of September 12, 2018 (`classicthesis` version 1.1).

DECLARATION

This Doctoral Thesis was financed by public French funds and carried out at the *Systèmes et Applications des Technologies de l'Information et de l'Énergie* (SATIE) laboratory of the *Département de Génie Electrique et Informatique Industrielle* (GEII) of the *Université Paris Seine*, Cergy-Pontoise, France.

Cette Thèse Doctorale à été financée par l'État Français et tenue dans le laboratoire SATIE (*Systèmes et Applications des Technologies de l'Information et de l'Énergie*) du *Département de Génie Electrique et Informatique Industrielle* (GEII) de l'*Université Paris Seine*, Cergy-Pontoise, France.

Aquesta Tesi Doctoral ha estat finançada per l'Estat Francés i portada a terme en el laboratori SATIE (*Systèmes et Applications des Technologies de l'Information et de l'Énergie*) del *Département de Génie Electrique et Informatique Industrielle* (GEII) de l'*Université Paris Seine*, Cergy-Pontoise, France.

Paris, France, 11th July 2018

Daniel Tormo Borredà