POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Architectures and methodology for the design of real-time power converter simulators on FPGAs

FEDERICO MONTANO
Département de génie électrique

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie électrique

Août 2021

Cette thèse intitulée :

Architectures and methodology for the design of real-time power converter simulators on FPGAs

présentée par **Federico MONTANO**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de:

DEDICATION

To all the people that have been next to me
in this long s-path. . .

ACKNOWLEDGEMENTS

This doctoral thesis represents for me the achievement and culmination of a long academic career that would not have been possible without the help of all the people who supported me along this long winding road.

First of all, I would like to thank my directors, Professor Jean Pierre David for having confidence in me and guiding me through this process and Professor Tarek Ould-Bachir for his constant support, excellent ideas and challenging discussions. Also to Professor Yvon Savaria who guided me at the beginning of my academic journey when I arrived at Polytechnique.

I would also like to thank my close family, my son Nicolas and my wife Beatriz Elena for their unconditional support and for putting up with me every day. To my extended family, my father who is no longer here, my mother, my sisters, my brothers and my countless nieces and nephews who have always been there for me.

I would also like to thank the members of the jury for taking the time to evaluate this work.

I would also like thanks to OPAL-RT Technologies and MITACS for their financial support.

Finally, I would like to thank my friends and lab partners Shiva, Roberto and Hemin who made this journey more enjoyable.

RÉSUMÉ

Cette thèse porte sur une méthodologie pour la mise en œuvre sur FPGA de simulateurs en temps réel des convertisseurs de puissance. Un simulateur en temps réel donne un aperçu du comportement des convertisseurs. La complexité du simulateur peut changer dépendamment de la topologie du convertisseur à simuler. Le simulateur permet d'évaluer les spécifications, de vérifier le comportement des convertisseurs modélisés au cours des différentes phases de développement et de simplifier leur intégration. L'utilisation de simulateurs en temps réel facilite, par exemple, le test d'un convertisseur de puissance et de son contrôleur dans une configuration avec matériel dans la boucle (*hardware-in-the-loop* – HIL). Il permet de réduire les risques et les coûts associés à l'utilisation d'un banc d'essai réel.

La mise en œuvre de simulateurs en temps réel sur FPGA est une tâche complexe réservée aux experts. De plus, le cycle de développement sur des FPGA est relativement long. Actuellement, il existe des outils de conception à haut niveau d'abstraction tel que Vivado HLS de Xilinx qui permettent de générer du matériel à partir d'un module programmé en C/C++/SystemC et d'utiliser des directives pour guider le processus de synthèse. HLS fournit une approche logicielle qui réduit le temps de vérification fonctionnelle et facilite la génération de matériel. Toutefois, selon l'application, ces outils n'atteignent pas toujours des performances équivalentes à celles d'un expert utilisant une description RTL (*register transfer level*). L'une des principales limitations de l'approche HLS est qu'elle ne fournit pas de mécanismes permettant d'instancier efficacement des modules personnalisés préfabriqués.

Cette recherche propose l'utilisation des approches combinées des architectures de superposition (*overlay architectures* — OA) et de la conception insensible à la latence (*latency insensitive design* — LID) comme méthodologie pour la mise en œuvre de simulateurs en temps réel basés sur les FPGA. Ce travail a été développé en trois étapes principales. Premièrement, nous avons réalisé une étude sur l'utilisation d'un outil HLS (Vivado HLS) pour la mise en œuvre de simulateurs en temps réel pour les convertisseurs de puissance. Deuxièmement, nous proposons la conception d'un réseau d'interconnexion multi-niveaux (multi-stage interconnection network — MIN) en utilisant le paradigme de conception insensible à la latence — LID. Ce MIN peut être reprogrammé et paramétré. Il offre des capacités de contention de données pour éviter les conditions de blocage. Le MIN a une conception et un algorithme de routage simples, qui permettent un haut débit à des fréquences élevées (>500 MHz) tout en gardant une consommation de ressources faible. Il est utilisé comme mécanisme d'interconnexion pour le routage des données entre les principaux modules du système. Finalement, nous avons généralisé l'approche LID en ajoutant des unités de contrôle distribuées aux modules fonctionnels (opérateurs arithmétiques et mémoires) pour permettre des transferts synchronisés par les données et nous avons relié les modules pour former une architecture de superposition d'un solveur linéaire paramétrable et reprogrammable. Une nouvelle architecture peut être facilement générée en modifiant certains paramètres, par exemple, le nombre et le type d'opérateurs. Une architecture déjà synthétisée peut être reprogrammée pour simuler différents types de convertisseurs. Trois architectures ont été générées et testées sur différents circuits. Elles opèrent à des fréquences d'horloge allant jusqu'à 300 MHz et permettent d'atteindre des pas de calcul inférieurs à la microseconde — un requis fort pour la simulation en temps réel des convertisseurs de puissance modernes.

La génération automatique de l'implémentation matérielle obtenue avec la méthodologie de conception proposée dans cette recherche facilite l'utilisation des FPGA aux spécialistes des applications et aux développeurs de logiciels pour la mise en œuvre de simulateurs de convertisseurs de puissance pour les applications HIL.

ABSTRACT

This thesis focuses on a methodology for the implementation of real-time FPGA-based simulators for power converters. A real-time simulator provides a preview of the behavior of converters. Its complexity may change depending on the converters' topology. The real-time simulator allows evaluating the specifications, verifying the behavior of modeled systems during the various phases of development, and simplifying their integration. The use of real-time simulators facilitates, for example, the testing of a power converter and its controller in a hardware-in-the-loop (HIL) configuration. It helps to reduce the risks and costs associated with the use of a physical test bench.

The implementation of real-time simulators on FPGAs is a complex task reserved for experts. Moreover, the development cycle for FPGAs is relatively long. Currently, there are high level synthesis tools such as Vivado HLS from Xilinx that allow the generation of hardware from a module programmed in C/C++/SystemC and the use of directives to guide the synthesis process. HLS provides a software approach that reduces the time of functional verification and facilitates the generation of hardware. However, depending on the application, these tools do not always reach a performance equivalent to that achieved by an expert using RTL description. One of the main limitations of the HLS approach is that it doesn't provide a mechanism to effectively instantiate custom-made modules.

This research proposes the use of the combined approaches of overlay architectures (OA) and latency insensitive design (LID) as a methodology for the implementation of real-time simulators based on FPGA. This work has been developed in three main stages. Firstly, we performed a study of the use of an HLS tool (Vivado HLS) for implementing real-time simulators for power converters. Secondly, we propose the design of a multi-stage interconnection network (MIN) using the latency insensitive design paradigm — LID. This MIN can be reprogrammed and parametrized. It provides data contention capabilities to avoid blocking conditions. The MIN has a simple design and routing algorithm, which enables high throughput at high frequencies (>500 MHz) while maintaining low resource consumption. It is used as an interconnection mechanism for data routing between the main modules of the target simulation system. Finally, we generalized the LID approach by adding distributed control units to the functional modules (arithmetic operators and memories) to allow synchronized data transfers and interconnected the modules to form a parametrizable and reprogrammable linear solver overlay architecture. New architectures can be easily generated by modifying certain parameters, for example, the number and type of operators. An architecture already synthesized can be reprogrammed to simulate different types of converters. Three architectures have been generated and tested on different circuits. They support clock frequencies up to 300 MHz, and generate computation time-steps lower than one microsecond which is a requirement for real-time simulation of modern power converters.

The automatic hardware implementation obtained with the design methodology proposed in this research facilitates the use of the FPGA by application specialists and software developers for the implementation of power converter simulators for HIL applications.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

ADC	Associated Discrete Circuit
CP	Computing Power
DAE	Differential Algebraic Equations
DP	Dot Product
DSP	Digital Signal Processor
EMT	Electro-Magnetic Transient
FDP	Fused Datapath
FLOPS	FLoating-point OPerations per Second
FP	Floating-Point
FPGA	Field Programmable Gate Array
FXP	Fixed-Point
HIL	Hardware-In-the-Loop
HLS	High Level Synthesis
HS	Hardware Solver
LID	Latency-Insensitive Design
LUT	Look-Up Table
MANA	Modified-Augmented Nodal Analysis
MIN	Multi-stage Interconnection Network
MVM	Matrix-Vector Multiplication
NoC	Network on Chip
NTT	Network Tearing Technique
OA	Overlay Architecture
PEC	Power Electronic Circuit
RAM	Random Access Memory
RSM	Resistive Switch Model
RTL	Register-Transfer Level
RTS	Real-Time Simulator
SAF	Self-Alignment Format
SNP	Switching Network Partitioning
SPS	SimPowerSystem
THD	Total Harmonic Distortion
VHDL	Very high speed integrated circuit Hardware Description Language
XBAR	Crossbar

CHAPTER 1 INTRODUCTION

The complexity associated with modern electrical networks used in industries such as generation, transport and distribution of electricity, electrical vehicles, and aerospace requires multiple iterations during their development cycle to warrant proper functioning. For many decades, simulation has been used in research and industry as an efficient method to reduce the cost and risk related to the development, test, and deployment of new systems. The use of a simulator allows foreseeing the system behavior before its physical implementation. The physical system (typically a plant) is modeled using a set of equations that describe its behavior under the operation range. The computing power and time required to perform the associated calculations depend on the complexity of the model.

The simulation can be performed in offline or real-time modes. An offline simulation means that the computing power is used to solve the system's equations and obtain its state and outputs regardless of the time required to perform all the calculations. Offline simulation offers flexibility and is normally used during the initial stages of the development process. However, the lack of time synchronicity makes offline simulation unusable during the later stages of the development process, when the response time must be considered. In contrast, real-time simulation performs all the model calculations within the time-step duration and produces outputs in a synchronized way under a hard real-time constraint.

Real-time simulation eases the test and the verification of real systems during the multiple phases of the development process. It helps shortening the development iteration loop, to reduce costs and implementation time. For this reason, real-time simulation is the preferred technique to apply in the development of devices and systems rather than hardware prototyping. Furthermore, real-time simulation enables testing the system under development at the limits of its normal operation, avoiding the risks associated with doing such tests with physical devices. Typically, real-time simulators are used for testing the controller of a plant in a hardware-in-the-loop (HIL) configuration. In such an approach, the real controller is connected to the simulator and cannot see any difference between the simulator and the real real system.

Historically, real-time simulators were designed using PC clusters to accommodate for complex models. However, real-time simulation implies that all calculations pertaining to a time-point are completed within the duration of the simulation time-step. In certain applications such a modern power electronics, this condition introduces heavy constraints on the round-trip time due to the latency associated with the communication links in CPU-based simulation. For example, a power converter requires that the simulator solves the system equations within a 1 μs time when the switching frequencies are over tens of kHz. Hence, real-time simulation manufacturers started considering the use of FPGAs as a hardware platform.

The digital design of FPGA-based real-time simulators is mainly performed at low level, following a register-transfer level (RTL) approach and using hardware description languages (HDL) such as VHDL and Verilog. Graphical tools such as Xilinx System Generator are also used. This comes at a cost, due to the complexity associated with the RTL approach, as time to market increases. Additionally, an FPGA specialist is usually required to perform the design. An update to the digital design can take many days before implementation. Hence, there is clearly a need for a high-level solution for the design and implementation of real-time simulators on FPGA.

Nowadays, there are commercial and academic tools that use the high-level synthesis (HLS) design for FPGA development. These tools allow the use of languages like C/C++ as entry points, then a specialized compiler translates the C/C++ description into a hardware module. Following this approach, the usage of FPGA can be facilitated to software developers or application specialists. However, it has been shown that the hardware produced with such tools can hardly compete with hand-crafted designs for real-time simulation purposes.

1.1 Problem Definition

Modern simulators used in the industry are platforms combining general processors (CPUs) and FPGAs. In applications requiring short calculation time, the FPGAs are used to complement or replace the CPUs when the latter are unable to meet the timing requirements on their own. FPGA-based real-time simulators require high computing power, good flexibility, efficient interconnection systems, and short response times.

The architecture of FPGAs offers high parallelism capacity and structural flexibility. These features make the FPGAs suitable for the development of high-speed hardware solvers for real-time simulation of electrical networks. Hardware development on FPGAs isn't necessarily a task that can be easily performed by a power application engineer. Frequently, the support of a hardware specialist for the low-level design, or the use of high level approaches can ease the task. Two main aspects are affected by this condition: i) If the design is performed at low level (RTL-HDL), the time and complexity attached to the development increase the total cost; ii) If commercial HLS tools are used for the design of real-time simulators, at their current state, such tools can fail to achieve the required performance. There are other design alternatives, such as latency insensitive design (LID) and overlay architectures (OAs), which are known to facilitate modularity and reprogramming. Although OAs are already in use in the industry, none have considered LID in the implementation of hardware real-time simulators. These conditions lead to the interest of this research to establish a methodology for implementing real-time simulators over FPGAs using LID and OA approaches to generate architectures that are easy to configure and reprogram.

In this research, we show that:

1. The ever-increasing resources and performance of modern FPGAs make them the privileged hardware platform for the implementation of real-time simulators.

2. Solving the system of equations of power systems implies matrix operations that offer parallel processing opportunities.

3. The combination of LID and OA allows a fast implementation of FPGA-based real-time simulators that are accurate by design. This methodology leads to an automatic generation of hardware implementations.

1.2 Research Objectives

The main goal of this research to provide a design methodology for FPGA-based real-time simulators. This methodology allows the automatic generation of hardware solvers aimed at the real-time simulation of power converters on FPGAs. Thus, facilitating the use of FPGAs to application specialists for the implementation of power converters simulators for HIL applications and reducing design time. In order to achieve our goal we:

1. Propose and develop predefined hardware solver architectures that can be parameterized and re-configurable. These architectures use an OA design that can be reprogrammed after synthesis.

2. Introduce the latency insensitive design approach to the development of FPGA-based real-time simulators. This approach facilitates the integration of modules.

3. Propose and develop a cost-effective interconnection system that allows data transfer among the different modules forming the simulator at high throughput and low cost.

4. Propose a methodology to wrap the specialized operators already built by hardware specialists to allow their reuse.

1.3 Thesis Outline

This chapter presented an introduction to the real-time simulation, the problems associated with their development, and the overview of the objectives for this research. The following chapters show a layout for the rest of this work.

- Chapter 2: This chapter introduces the basic concepts related to real-time simulation. The most used mathematical formulations and switch models are presented. The offline and online simulation are differentiated, and a context of utilisation of real-time simulators is presented ;
- Chapter 3: This chapter reviews the works more related to this research. It explains the different techniques for the implementation of real-time simulations for power converters over FPGAs, as well as the implementation approaches and their limitations. Finally, it presents the state-of-art of real-time simulation ;
- Chapter 4: This chapter describes the process followed to generate and assemble the main chapters that form this thesis ;
- Chapter 5: This chapter presents a comparative evaluation between custom design and the use of a high-level synthesis tool. Both techniques were used as design methods for the hardware solvers implementation. A detailed study for the performance, latency, and resource consumption over multiple circuits of different sizes is presented. This is the first paper issued from this work [11]. It has been published in IEEE Transactions on Industrial Electronics in 2018 ;
- Chapter 6: This chapter presents the system interconnection mechanism conceived as a re-configurable (low latency/high throughput) multistage interconnection network (MIN). The latency insensitive design approach is first introduced in this chapter. This topic is our second paper [12]. It has been presented at IEEE-CCECE in 2019 ;
- Chapter 7: This chapter presents the complete methodology used for the implementation of a re-configurable and re-programmable overlay architecture following a LID approach to implement hardware solvers, its advantages, and limitations. The real-time simulation of power converters is presented as an application case. This topic is our third paper [13]. It has been published in MDPI-Electronics in 2020 ;
- Chapter 8: This chapter presents a general discussion of the approaches and results obtained in this work ;

- Chapter 9: Conclusion and recommendations: This last chapter summarizes the results and presents the main teachings obtained from this thesis. This chapter ends by suggesting that further research must be done to create a mathematical model to predict the achievable time step for each circuit to be simulated on any of the implemented architectures.

CHAPTER 2 BACKGROUND

2.1 Introduction

Real-time simulation is an industry practice used for designing and implementing power systems. It is widespread in industrial sectors such as the generation, transmission and distribution of electric power, electric and hybrid vehicles, and in the aerospace field. Real-time simulation allows systems to be tested before their construction, thereby reducing risk, costs and development time.

The evolution of real time simulators started with the Transient Network Analyzer (TNA), where the models were recreated at a small scale with a configuration of analog components. TNAs have been used to perform simulations with Hardware-In-the-Loop (HIL) to test control equipment and protections [14]. Then, computer evolution allowed the use of CPU-based architectures to replace progressively the TNAs on real-time simulation also increasing their flexibility [15, 16]. Finally, FPGAs were integrated into these simulators. Initially as efficient interfacing mechanisms, then as high performance parallel computing units in applications that require small time-steps, thus increasing the application range of real-time simulators [6, 17].

In this chapter we present briefly some concepts, methods, models and tools associated with the usage and design of digital real-time simulators of electrical networks, particularly for those implemented on FPGAs.

2.2 Electric Circuit Equations

Electric network simulation tools meet different needs, including the study of switchgear, analysis of harmonics on the network, electricity generation, power distribution on the network, failure conditions, machine behavior, and their controls, etc. [14]. This work mainly focuses on the simulation of the network's transient in the time domain, better known as the electromagnetic transient simulation (EMT) [18]. The system to be simulated is described by a mathematical model which reflects its behavior for the studied conditions. The network model is generally of a discrete nature and involves Differential Algebraic Equations (DAE) [19]. An EMT simulator must assemble, discretize and solve the DAEs. Network equations are composed using either state-space or nodal analysis methods.

2.2.1 State-Space Representation Method

The representation of states for a linear differential equation system is given by:

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu} \\ \mathbf{y} = \mathbf{Cx} + \mathbf{Du} \end{cases}, \tag{2.1}$$

where $\mathbf{x}$, $\mathbf{u}$, and $\mathbf{y}$ are the state, input, and output vectors respectively. $\mathbf{A}$ is the dynamic matrix, $\mathbf{B}$ the control matrix, $\mathbf{C}$ the observation matrix and $\mathbf{D}$ the direct action matrix. With certain reservations, a power circuit can be described by a representation of states. This is the approach adopted by software such as SimPowerSystems [20] from Hydro Québec or PLECS [21] from PLEXIM.

If the time step is many folds smaller than the time constants of the system, a low order integration method such as the implicit Backward Euler (BE) method can be used to discretize the state-space equations, which yields:

$$\begin{cases} \mathbf{x_{n+1}} & = & \mathbf{A}_d\mathbf{x_n} + \mathbf{B}_d\mathbf{u_{n+1}} \\ \mathbf{y_{n+1}} & = & \mathbf{C}_d\mathbf{x_n} + \mathbf{D}_d\mathbf{u_{n+1}} \end{cases}, \tag{2.2}$$

with

$$\begin{cases} \mathbf{A}_d & = & (\mathbf{I} - \boldsymbol{\Delta}t\mathbf{A})^{-1} \\ \mathbf{B}_d & = & (\mathbf{I} - \boldsymbol{\Delta}t\mathbf{A})^{-1}\boldsymbol{\Delta}t\mathbf{B} \\ \mathbf{C}_d & = & \mathbf{C}(\mathbf{I} - \boldsymbol{\Delta}t\mathbf{A})^{-1} \\ \mathbf{D}_d & = & \mathbf{D} + \mathbf{C}(\mathbf{I} - \boldsymbol{\Delta}t\mathbf{A})^{-1}\boldsymbol{\Delta}t\mathbf{B} \end{cases}. \tag{2.3}$$

$\mathbf{A}_d$, $\mathbf{B}_d$, $\mathbf{C}_d$ and $\mathbf{D}_d$ are the discretized matrices of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ respectively. $\mathbf{x_n}$, $\mathbf{y_n}$, and $\mathbf{u_n}$ are respectively the sampled version of vectors $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{u}$, and $\mathbf{I}$ is the identity matrix. It was shown in [22] that rewriting Eq. (2.2) in a matrix-vector multiplication form as follows:

$$\begin{bmatrix} \mathbf{x_{n+1}} \\ \mathbf{y_n} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_d & \mathbf{B}_d \\ \mathbf{C}_d & \mathbf{D}_d \end{bmatrix} \begin{bmatrix} \mathbf{x_n} \\ \mathbf{u_n} \end{bmatrix}, \tag{2.4}$$

the parallelism associated with processing on FPGAs helps to warrant small-time steps for solving the system.

2.2.2 Nodal Method

The (classical) nodal analysis method gives the system equations as:

$$\mathbf{Yv} = \mathbf{i}, \tag{2.5}$$

where $\mathbf{v}$ is the vector of node voltages (unknown), $\mathbf{i}$ is the vector of node current injections and $\mathbf{Y}$ is the admittance matrix. In power circuits, it is customary to replace electrical components (L, C, etc.) by their Norton equivalent, as obtained by a discretization rule (the trapezoid rules and BE are most often used) [23]. The Norton equivalent of a two-terminal device connected to nodes a and b is given by:

$$g_{ab}(v_a^{n+1} - v_b^{n+1}) = i_{ab}^{n+1} + j^{n+1}, \tag{2.6}$$

where $(v_a^{n+1} - v_b^{n+1})$ is the voltage drop between nodes a and b, i_{ab}^{n+1} is the current flowing through the device from node a to node b, g_{ab} is the equivalent conductance of the device, and j^{n+1} is the history term. Using BE, the equivalent conductance of an inductance L is $g_L = \Delta t/L$, and $g_C = C/\Delta t$ for a capacitance C. The history term j^{n+1} is $-i_{ab}^n$ for L, and $g_c(v_a^n - v_b^n)$ for C. $\Delta t = t^{n+1} - t^n$ is the simulation time step.

2.2.3 The Modified and Augmented Nodal Analysis Method

The classical nodal method does not support independent voltage sources, more so when the independent source has none of its terminals connected to the ground. The Modified and Augmented Nodal Analysis (MANA) method overcomes these (and many other) shortcomings [24]. The system equations are augmented and yield:

$$\mathbf{Ax} = \mathbf{b}, \tag{2.7}$$

where $\mathbf{A}$ is the modified and augmented nodal matrix, $\mathbf{x}$ is a vector containing unknown node voltages and certain branch currents, and $\mathbf{b}$ is a vector composed of known sources and history terms. More specifically, Eq. (2.7) reads as follows:

$$\begin{bmatrix} \mathbf{Y} & \mathbf{V_c} & \mathbf{D_c} & \mathbf{S_c} \\ \mathbf{V_r} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{D_r} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{S_r} & \mathbf{0} & \mathbf{0} & \mathbf{S_d} \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mathbf{i_V} \\ \mathbf{i_D} \\ \mathbf{i_S} \end{bmatrix} = \begin{bmatrix} \mathbf{i} \\ \mathbf{v_b} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \tag{2.8}$$

where $\mathbf{Y}$, $\mathbf{v}$, and $\mathbf{i}$ represent the admittance matrix, the unknown node voltages and the current injections from the classical nodal analysis. Subscripts $\mathbf{r}$, $\mathbf{c}$, and $\mathbf{d}$ refer to row, column and diagonal respectively. $\mathbf{V_r}$ is the voltage source connection matrix. $\mathbf{D_r}$ holds dependent branch equations (ideal transformers). Sub-matrices $\mathbf{S}$ contains the equations of the ideal switches with $\mathbf{S_d}$ representing the state of switches. The column matrices are the transposed of row matrices, and the rest of sub-matrices are very often null. This representation fixes the rank of the matrix giving the possibility of easily reformulating the equations for any change on the switch statuses.

To facilitate the implementation over FPGAs, the MANA equations can also be represented by matrix-vector multiplication as done for the state-space representation. Defining $\mathbf{b_n} = \mathbf{K_c}[\mathbf{u_n},\ \mathbf{j_n}]^T$, where $\mathbf{K_c}$ is the incidence matrix, $\mathbf{u_n}$ is a vector containing the independent sources of the network, $\mathbf{j_n}$ is the vector of historic for reactive (L/C) components and switches. Considering $\mathbf{A^s}$ to be the matrix for a given combination of switch statuses. Eq. (2.7) becomes

$$\mathbf{x}_n = (\mathbf{A}^s)^{-1}\mathbf{K}_c \begin{bmatrix} \mathbf{u}_n \\ \mathbf{j}_n \end{bmatrix}, \tag{2.9}$$

we can have:

$$\begin{bmatrix} \mathbf{j}_{n+1} \\ \mathbf{y}_n \end{bmatrix} = \begin{bmatrix} \mathbf{H^s}_{ju} & \mathbf{H^s}_{jj} \\ \mathbf{H^s}_{yu} & \mathbf{H^s}_{yj} \end{bmatrix} \begin{bmatrix} \mathbf{u}_n \\ \mathbf{j}_n \end{bmatrix}, \tag{2.10}$$

where the matrices $\mathbf{H^s}_{ju}$, $\mathbf{H^s}_{jj}$, $\mathbf{H^s}_{yu}$, and $\mathbf{H^s}_{yj}$ are obtained from Eq. (2.9).

2.2.4 Switch Models

It is common to find power semiconductors such as diodes, IGBTs, GTOs or MOSFETs in power converters. These components can be treated as ideal switches. Two currently used models were born from this idea:i) the resistive switch model (RSM) and the associated discrete circuit (ADC) switch model.

The RSM represents the switch by a high resistor R_{off} when OFF and by a low one R_{on} when ON, as shown in Fig. 2.1:. Replacing the switch by changing the status of these switches alters the topology of the network. Solving each topology involves a new factorization of the circuit matrix [25]. For real-time simulation, it is common to pre-compute the inverse of the matrix for the different combinations of switch statuses. In these conditions, the memory requirements increase as 2^n (where n is the number of switches) limiting the size of the circuit to be simulated. To tackle this limitation, often the associated discrete circuit (ADC) switch model is used. The ADC switch model [26] proposes to replace the switch by an

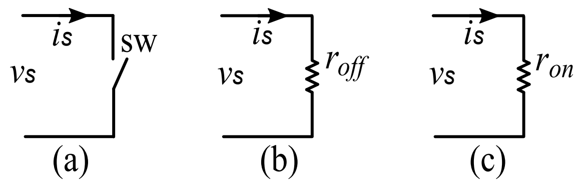


Figure 2.1: Resistive switch model (a) Ideal switch; (b) OFF state; (c) ON state

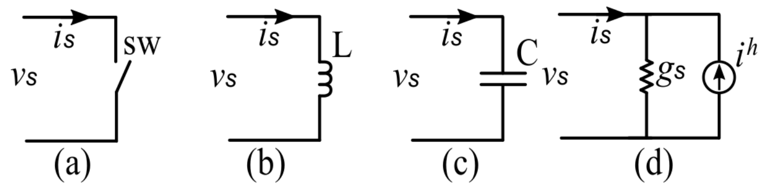


Figure 2.2: ADC Switch model (a) Ideal switch; (b) ON state; (c) OFF state; (d) Discrete model

inductance when it is in the ON state, and by a capacitance when it is in the OFF state as shown in Fig. 2.2:. The inductance and capacitance are replaced by their Norton equivalent circuits obtained by the discretization of the L/C equations. The conductance of the Norton equivalent associated with the inductance (ON) is given by:

$$\mathbf{gs_{ON}} = \mathbf{g_L} = \frac{\mathbf{\Delta t}}{\mathbf{L}}, \tag{2.11}$$

The conductance of the Norton equivalent associated with the capacitance (OFF) is given by:

$$\mathbf{gs_{OFF}} = \mathbf{g_C} = \frac{\mathbf{C}}{\mathbf{\Delta t}}, \tag{2.12}$$

Where $\mathbf{\Delta t}$ is the fixed simulation time step. By making sure that $\mathbf{\Delta t} = \sqrt{\mathbf{LC}}$ we get $\mathbf{gs_{ON}} = \mathbf{gs_{OFF}}$. In this case, the circuit equations become constant regardless of the condition (ON-OFF) of the switches, so it suffices to precompute the inverse of a single matrix. As $\mathbf{A}$ is invariant when using the ADC model, Eq. 2.9 becomes :

$$\begin{bmatrix} \mathbf{y_n} \\ \mathbf{i^{\hat{s}}_{n+1}} \\ \mathbf{i^h_{n+1}} \end{bmatrix} = \mathbf{H} \begin{bmatrix} \mathbf{u_n} \\ \mathbf{i^s_n} \\ \mathbf{i^h_n} \end{bmatrix}, \tag{2.13}$$

where $\mathbf{H}$ is a matrix obtained by algebraic manipulation of $\mathbf{A}^{-1}$ and $\mathbf{K_c}$. $\mathbf{y_n}$ represents the wanted outputs of the system, $\mathbf{i}^{\hat{\mathbf{s}}}_{\mathbf{n+1}}$ represents the historic of switches for both conditions (ON and OFF) computed, whilst $\mathbf{i}^{\mathbf{s}}_{\mathbf{n+1}}$ contains only one of two historic for each switch based on its actual status. This system can be solved by a processing architecture composed by a linear solver and a switch logic treatment module. The details of design, implementation, and performance of this solver are presented in chapter 7. It is important to notice that the ADC model introduces faulty oscillations during switching times, what requires an execution with small steps of calculation to keep the precision of the results [27].

2.3 Types of Simulations

A digital simulation is the representation for the functioning of a system by using the computing power to solve its model equations. It is used to facilitate different tasks, from the evaluation of electrical components characteristics to the analysis and design of complex electrical networks. The simulation methods and characteristics considered in this work will try to reproduce the real wave forms at different points of the circuit network (EMT-type). We assume that the simulation is at discrete time with constant time step. Based on the relation between the calculation time and the real-time system operation, we will distinguish two types of simulations, namely offline and real-time simulation.

2.3.1 Offline Simulation

An offline simulation refers to the execution of the circuit model, usually on a computer, in order to have an idea of its behavior. It offers a wide variety of numerical methods and programming flexibility. Some of the objectives of using offline simulation is to optimize the accuracy for the different modeling techniques and help to find the range of values for the parameters that must be tuned on the network whilst maximizing the computing speed. In an offline simulation, the execution time of a calculation step differs from the network response time (real time). In a real-time simulation, the execution of a calculation step is synchronized with the response time of the system. On a discretized time basis with a constant computation step, the offline simulation starts each computation step independently of the real time. Two conditions arise when comparing the time evolution of the offline simulation and the network response as shown in figure Fig. 2.3:.a. If the requirements on computation are low comparative to the computation power, the time for a simulation step can be lower than the network response. This condition leads to an accelerated simulation as shows the upper part of figure Fig. 2.3:.a. However, the regular condition is that the complexity of the models on modern networks requires a high amount of computations. The time consumed by the computation is often longer than the time response of the network as shown in the bottom

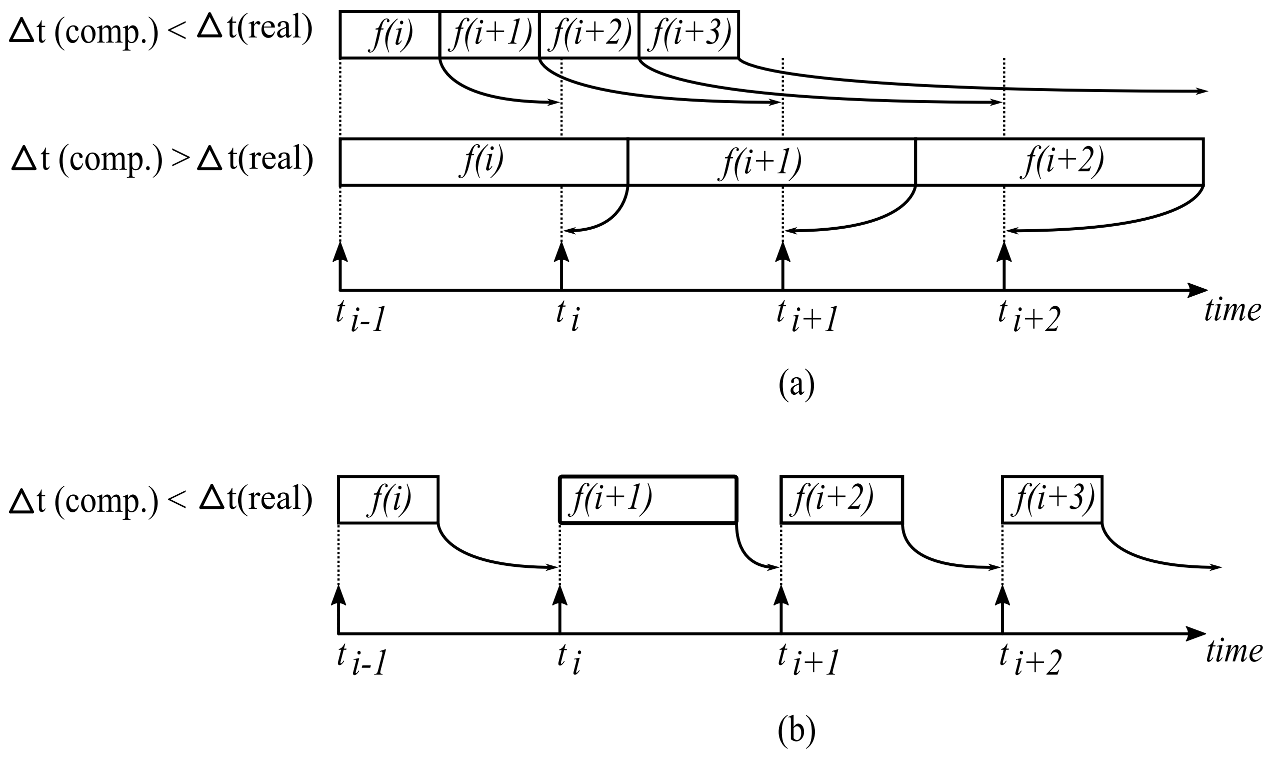


Figure 2.3: Types of simulation: (a) Offline simulation. (b) Real-time simulation

part of Fig. 2.3:.a. As offline simulation doesn't have time constraints, the simulation can be performed with a higher precision considering the models and mathematical operations.

2.3.2 Real-Time Simulation

In a real-time simulation, Fig.2.3: .b, the execution time of a computation step must always be smaller than the real time and the start of the simulation step is synchronized with the beginning of the computation step. The idle time (difference between the computation time and the real time) must suffice to treat inputs and outputs. The time constraint limits the flexibility in programming and the precision on the results. However, As the results are synchronized with the real time, these tools offer the possibility of being interfaced with real systems.

2.3.3 Use of Real-Time Simulators

HIL simulation requires that all calculations and communication between the simulator and the rest of the system be performed in a time smaller than the established calculation step [28].

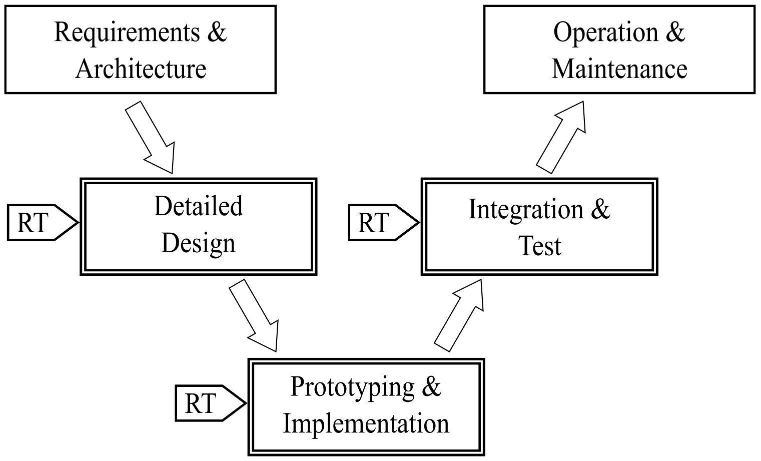


Figure 2.4: Use of real-time simulation. (Inspired from [1])

First, the simulator receives the input signals coming from devices on the connection environment. Then the simulator calculates and sends the new operating conditions (state) back to the rest of the system (or DUT). This completes the simulation cycle. The HIL technique allows changes to be made quickly if the results are not compliant. It also allows validating operation while avoiding significant power consumption and minimizing potential risks associated with malfunction. Real-time simulators are used in different phases of project development [1]. Fig. 2.4: shows different phases in which real-time simulators are used. In the design phase, the speed of the simulator helps reduce simulation time. The real-time simulator is also used for prototyping and testing real components of systems such as HVDC, FACTS, and their control, thus facilitating their implementation and verification.

2.4 Commercial Simulation Tools

The increasing industry demand has pushed the creation or adaptation of multiple tools for offline and for real-time simulation that continues to evolve.

2.4.1 Offline Simulation Tools

Offline simulation is performed on hardware platforms with one or more general-purpose processors (CPUs). Several offline simulation tools use the nodal method proposed in 1969 by Dommel for transient analysis in electrical networks [29]. These tools are known as the

EMT (electromagnetic transient) type. Among them we have: ATP [30] and PSCAD / EMTDC [31], which uses the trapezoidal integration method with a fixed calculation step. Also, EMTP-RV, the simulation software developed by Hydro-Québec is based on a modified version of nodal analysis (MANA) [24, 32]. A second group of tools is based on the state representation method and allows the simulation to be carried out with a fixed or variable calculation step. These simulators are used for modeling systems in electronics, automation, transport and electrical networks, among others. The best known are Hydro-Québec's SimPowerSystems SPS [20] and PLECS from PLEXIM [21]. These two programs are marketed as a toolbox in Matlab / Simulink. PLECS is also sold as a stand-alone application.

2.4.2 CPU-Based Real-Time Simulators

The first commercial digital simulator was developed by RTDS Technologies and used to test HVDC devices [33]. This simulator was organized in racks whose hardware platform was formed by three main modules. The processing module, made up of two DSPs, had a few digital and analog input and output ports. The interface module used to communicate the rack to the workstation (host). The inter-rack communication module used to interconnect up to four racks. The software platform consisted of a graphical interface (PSCAD) for inputting circuits. A compiler produced the code to be executed in the processing module, the circuit distribution reports, and a set of libraries with the models of the components encoded in machine language. This simulator was limited in terms of interfaces, libraries of components, and computing capacity. However, it allowed the testing of protective relays and control equipment [34]. This technology continues to evolve and several other commercial digital simulators have been developed to cover many applications in modern industry [35], among them, are:

- The modern version of the RTDS simulator, from RTDS Technologies. It uses RSCAD software for development. The platform is configured in racks based on the RISC PowerPC processor. This simulator is used for testing protection and control systems, HVDC, FACTS and HIL among others.
- HYPERSIM [15] developed by the Hydro-Québec Research Institute. The open hardware platform is based on parallel computers and capable of interfacing with FPGAs. The Hypersim Toolkit has a model library, schematic editor, data acquisition system, and signal processing system. This tool is capable of automatically distributing the simulation tasks on the different available cores in the system. This simulator is mainly used for electromagnetic transient simulations in huge electrical networks (FACTS and HVDC among others). The calculation steps are greater than 10 μs.

- ARTEMiS [36] from OPAL-RT, this simulator has a hardware platform based on multi-core processors and RT-Lab software. Capable of interfacing with Matlab/Simulink, it contains model libraries of components for the simulation of power systems and power electronics. This simulator is used in the design and testing of protective equipment, relays and controllers in power electronics and HIL.
- rtxd from ADI [37] This simulator has a CPU-based hardware platform and uses ADvantageDE for development. It is mainly used in the simulation of avionics systems.

Processor-based real-time simulators offer modularity, programming flexibility and support for multiple data types which ease the implementation of the ever-increasing size and complexity systems. However, data transmission between processors and I/O interfaces introduces delays that limit the minimum computational step to several microseconds [23].

2.4.3 FPGA-Based Real-Time Simulators

The advances in technology, the increasing I/O requirements for data transfer, and the increased needs for computing capacity at a short time step have pushed the introduction of FPGAs in the development of simulators. Mixed platforms (FPGA-CPU) and pure FPGA platforms have been introduced into the market. Among them we can find the following:

- OPAL-RT's eFPGAsim [38] uses FPGAs to simulate power electronics, control systems, electrical protection and HIL. It uses the homemade solver (eHS) to solve the system of equations and is supported by several editors (PSIM, SPS, PLECS).
- dSPACE [39] with a platform based on CPU and on FPGA uses a library for the simulation of power electronic systems (RTI) under Simulink for development. It is mainly used for the implementation and testing of control systems in mechatronics.
- Typhoon RTDs [40] uses the Typhoon schematic editor. It is used for testing power electronics controllers.
- PLECS RT BOX by PLEXIM [41] on a SoC platform (Zync7000 CPU-FPGA) uses the editor and the PLECS code generator under Matlab. It is used for HIL testing of power converters.
- Speedgoat and Simulink real-time use combined CPU-FPGA platforms to perform HIL and RCP.

These tools are used in various industrial sectors for hardware-in-loop simulation (HIL), rapid prototyping of power systems, power electronics and their control. Several conditions to be considered when building FPGA simulators will be discussed in the next chapter.

2.5 Conclusion

This chapter presented the concepts related to the simulation of electrical networks. It presented the main mathematical formulations for state-space and nodal analysis. A view of how this formulation can be solved using matrix-vector multiplication was also presented. The more frequently switch models used for real-time simulation and the consideration for FPGA implementations were presented. The special consideration shown for the ADC model presented its principal benefit and drawback. This model will be used for the work performed in this research. The difference between offline and real-time simulation was treated showing their benefits and drawbacks. In the same line, the conditions for the utilization of real-time simulation were shown. Finalizing this chapter, we presented a brief summary of the commercial tools to perform offline simulation. Similarly, the best-known commercial tools for real-time simulation based on CPU or CPU/FPGA were presented. The latest is the most related to this research. This work is centered on the development of real-time simulation of power converters on FPGA. For this reason, the more recent techniques used for the development of real-time simulators on FPGA will be presented in the following chapter.

CHAPTER 3 LITERATURE REVIEW

3.1 Introduction

Real-time simulation of electrical networks has passed from using scaled model circuit representations to computer-aided simulations. During the 1990s, the use of simulators with one or more general-purpose processors (CPUs) was common. However, the latency associated with their communication systems limited their usage when short time steps were required. The FPGA has been introduced to real-time simulators as mechanisms to improve the I/O interface. The growth in the computing power of FPGAs has promoted the arrival of simulators using CPU and FPGA as processing elements. In recent literature, we can find the design of simulators purely based on FPGAs for systems with high dynamic characteristics such as power converters. The latest is the core of this research.

As a general rule, the development of real-time simulators starts by creating models and algorithms that describe the behavior of the electrical network to be simulated. At this point, offline simulation tools can be used to verify the specs. Once its behavior has been validated, the portion of the network to be simulated on FPGA is separated. The discretization chosen for the representation of the equations depends on the precision, time constraint, and stability required. The main constraints to fulfill are calculation time, accuracy, and the area of the design. However, the design of real-time simulator of power converters on FPGA has multiple challenges:

- Time constraints: In modern power converters, for example, the switching frequencies are high [42]. It is suggested that the calculation step (Ts) be 20 times smaller than the system dynamics (transient and harmonics), and 50 to 100 smaller than the switching period [6, 43]. This allows to have a better accuracy, and to catch changes inferior to 2 % of the duty cycle. Thus, if we assume a switching frequency of 10 kHz, the computation time must be less than 1 μs.

- Area: FPGAs have limited resources, this condition limits the size of the network to be simulated while keeping the required calculation time-step fixed.

- Numerical resolution: The numerical representation, fixed-point or floating point, is chosen according to the dynamic range of values. In modern applications, operators capable of handling floating point numbers are increasingly required. Several works have deepened the development of floating point operators.

- Modeling methods: The circuit component model must have enough detail for an accurate representation of the real power circuit. However, good optimizations should be achieved to reduce resources consumption and make the circuit fit on the FPGA.
- Design approaches: The complexity inherent to FPGA development limits its utilization and increases the time to market. However, this limitation can be mitigated if appropriated tools or development environments are available to the application specialists.

This chapter presents the state-of-the-art of FPGA-based real-time simulators for electrical networks. It presents the limitations found in currents development and puts our research in context. It also presents the commercial tools and design approaches used in the development of the real-time simulation for power converters on FPGAs. This includes the use of the high-level synthesis (HLS), overlay architectures, and LID approaches in the design of FPGA computing engines, whether in the context of real-time simulation or as calculation acceleration units over FPGA. At the end of this chapter, the works most related to this research are summarized and compared.

3.2 FPGA-based Real-Time Methods for Simulation of Power Electronics

Modern power converters are known for their high-dynamic operation which increases the challenges bound to their real-time simulation. Multiple methods have been proposed at the application level to meet the requirements. These methods aim to increase parallelism, reduce the simulation time step and simplify the power converters' real-time simulation.

3.2.1 Latency Insertion Method

The LIM method proposes to represent the circuit using connections of LIM compatible branches (series circuit R, L, and voltage source), and LIM compatible nodes (parallel circuit G, C, and current sources) [44]. Branches and nodes are computed and updated alternatively as time progresses. Originally proposed for offline simulations, this method improves the simulation time. It has been adapted to be used in FPGA as the LB-LMC method presented in [25] which separates the circuits in two parts (linear and nonlinear). The linear part will be solved with the nodal method. Each nonlinear element is treated independently and is represented as a current or voltage source (with its discretized equation of state explicitly) whose solution can be calculated in one step with the previous step results. LB-LMC method has been used in other publications and more recently is the core of the project

ORTiS Code Gen [45]. A new version of LIM for FPGA is presented in [46]. The circuit is divided in LIM compatible branches (series R, L, V) and nodes (parallel G, C, I). A hardware instance is created for each LIM branch and node to offer a higher granularity and to increase parallelism. The number of operations for each LIM entity remains constant. However, the FPGA occupation grows as the number of LIM entities increases.

3.2.2 Switching Elements Based Partitioning

The circuit partitioning is one of the most used methods. It allows dividing large circuits into multiple subcircuits to increase parallelism. In power converters, the dynamics of switches are fast comparatively to the dynamics of large capacitors and inductors. Often, parallel capacitors and serial inductors are used to determine the partition points [47]. Although the partitioning techniques were originally proposed for offline simulation, modern adaptations have made it possible to use them in real time with FPGA. Some of them are presented below.

The network tearing technique (NTT) [48], used for power converters proposes to divide the circuit into separate parts (subcircuits) and replace these with multi-port networks using the hybrid Norton-Thévenin equivalent. The resistive model is used for switches. Only the current and voltage variables in each port are considered when putting all the modules together. Using this method, a Neutral Point Clamped (NPC) converter of 18 switches is divided into 5 subcircuits, 3 of which are formed by the branches containing the switches of the converter. With this distribution, a reduction in memory requirements is obtained (from 218 systems of equations to 26 per branch) to keep the different configurations of the switches in the circuit. An extension for the utilization of NTT for more generic topologies is presented in [49].

A switching network partitioning (SNP) method for multiple converters using the two-value resistor switch model is presented in [50]. The SNP method proposes to analyze the power converter in three stages: i) equivalent circuit external to the switching network, ii) equivalent variables for the switching network, iii) equivalent of switching network variables as sources. One entity determines the states of all switches considering only the gates control signals and the external circuit. However it requires 2^l permutations to be stored in RAM for each converter, where l is the number of legs in the converter.

As a sub-module must wait for the result of another, partitioning the circuit often introduces latency that can cause instability. To tackle this effect, the predictor-corrector method [10] proposes to separate the switch network from the circuit element part. Then, compute them at the same time with a parallel architecture. At each time step, the predictor generates the results for the future time step based on the results obtained by the corrector. This architecture is known to spend the double of DSP resources.

3.2.3 Methods for Naturally Commuted Devices

The voltage and current on the terminals of a naturally commuted switch (e.g., a diode) determines its status ON-OFF. As these variables evolve, the conditions of the switch can change producing errors in the results. Multiple iterations are often needed to correctly establish the switch status. However, the iterations increase the computational demand which limits the real-time simulation when small time-steps are required.

The Zero Regulation method (ZR) proposed in [51] presents a method for reducing the zero-crossing error. An SLR (series load resonant converter) is used as a case study case. A threshold is set around the zero-crossing (chattering region) to limit the iterations. The predictor-corrector is used to determine the condition. A flag is used to identify if the simulation is in the zero-crossing zone, the values around the zero point are counted (the zero is marked as a region). This limits the value of the chattering to be neglected and reduces the iterations. This technique reduces the zero-crossing error from 20% to 0.3% potentially.

The Direct Mapped Method (DMM [52]) proposes to eliminate the iterations for circuits with diodes. In a single-phase full rectifier, the possible combinations of the diode statues are obtained by a map that links the history terms to the possible statuses of the diodes. The regions obtained by the map determine the status of all diodes, avoiding iterations. Two units are used to solve the system, one to calculate historic and outputs. The second to evaluate the mapping function. The results report a time-steps of 25 ns for an LLC converter.

The mentioned methods intend to simplify the analysis of the network or optimize and facilitate the implementation of FPGA. However, they can be considered custom solutions from the point of view that if the circuit changes, the hardware must be synthesized again. Some of them can even require further adjustments.

3.3 Hardware Development Approaches

3.3.1 RTL

The FPGA design flow requires several steps to be taken to ensure the functionality of the circuits implemented. The first step is the circuit description. Traditionally, the hardware description languages VHDL and Verilog are used at this stage. The quantity and types of

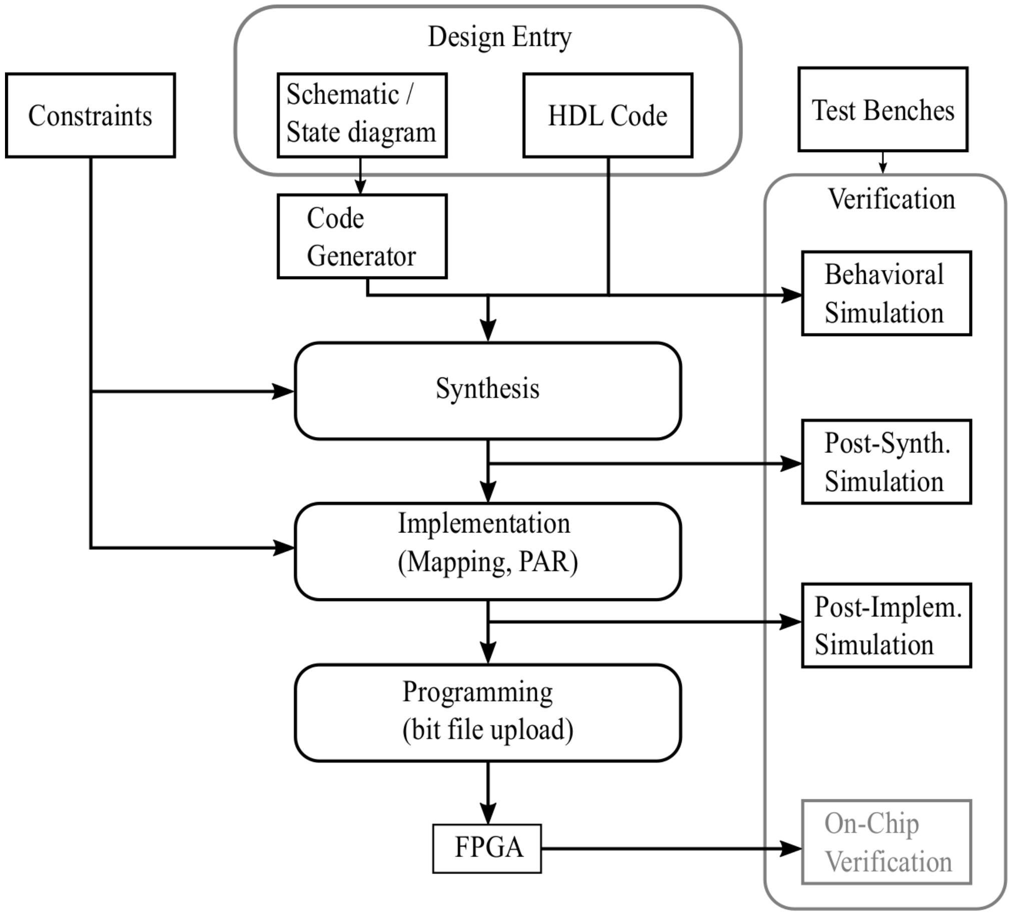


Figure 3.1: FPGA Design Flow

available operators (IPs) depend on the capacity and the technology of the FPGA chosen as a development platform. With these elements in hand, the hardware expert carries out a description (behavioral or structural) of the circuit based on the specification and the algorithms. It is a manual process, prone to errors, and requires great skill from the designer. Once the description step is done, the rest of the process can be automated at a certain degree by using the tools. In a second step, the circuits are synthesized. This task is performed using tools like Xilinx ISE / Vivado, Altera Quartus, Actel Libero, Synopsis Symplify. The third step concerns the implementation process (map, place and route) which is specific to each manufacturer. Fig. 3.1: shows the design flow over FPGAs. The simulation is a fundamental step that allows verifying the functioning of the described circuit. It can be performed at different stages of the development. FPGA manufacturers offer tools for simulation purpose e.g., ISIM from Xilinx. However, other tools such as ModelSim from Mentor Graphics are often used. Finally, the circuit is implemented and programmed on the FPGA.

In the literature, there are several examples of real-time simulator development that have gone through an RTL process. The implementation of an electronic power converter model on FPGA with the switches replaced by its ADC model and the equations gathered according to the nodal method is proposed in [53]. A modular scalable HIL simulator for the validation of designs in power electronics is proposed in [40]. Other examples of sub-microsecond real time simulators are presented in [42, 54]. These examples use the fixed point numerical representation, which facilitates the design task and the realizations with small steps of calculation. However, this representation limits the range of values and introduces precision errors. The use of floating point representation is now possible on FPGAs. Different examples of implementation exist in the literature [27, 48].

The FPGA development process is well defined. If the RTL approach is used, it requires some expertise in digital circuit development. This expertise isn't necessarily a requirement for application engineers. For this reason, producing specialized equipment can be a slow and painful process. There are intermediate tools, like System Generator, that facilitate the generation of test vectors for verification easing the development process.

3.3.2 High-Level Synthesis Approach (HLS)

The difficulty inherent to developing systems on FPGAs using a low-level (RTL) approach increases the time to market for products. This condition has caused developers in different fields to consider the utilization of abstractions such as high-level synthesis (HLS). This way, the developer can focus on the development of the application without considering the details of implementation while the HLS tool automatically generates the VHDL-RTL code. The code generated can be easily enchained to the traditional FPGA design flow to obtain an implementation. Fig. 3.2: gives a general view of the development process using HLS.

The idea of HLS dates to the 1970s [55]. The first HLS generations intended to facilitate the synthesis of ASICs. However, they were hardly accepted due to the personalized languages usage that involved a significant learning time, poor quality of results (QoR) or simply a poorly targeted customer.

Early attempts at HLS include tools such as CMU-DA [56, 57], which used the instruction set processor language (ISP) for behavioral description, then performed an intermediate transformation of the data path and generated the RTL. MIMOLA [58] used to design processors. HAL [59] and Cathedral-II [60] used for signal processing. Looking for a language generalization to reducing the learning curve and make users to adopt the idea of HLS. Several tools have taken C/C ++ as an entry point and created extensions such as Handel C. [61], SpecC [62] and the SystemC library [63]. The latest is specially adapted to support some

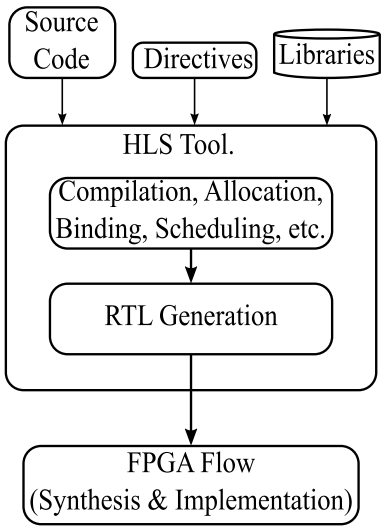

Source Code
Directives
Libraries
HLS Tool.
Compilation, Allocation, Binding, Scheduling, etc.
RTL Generation
FPGA Flow (Synthesis & Implementation)

Figure 3.2: HLS Design Flow

characteristics for the generation of material such as timing, synchronization, concurrency, hierarchy. These extensions limited the complex constructions of programming languages which are hard to implement on hardware (i.e., recursion and dynamic memory allotments).

The continued effort to make FPGAs easier to use, and the adoption of the C language, led to greater acceptance by developers. Several tools capable of transforming the C code into RTL specifically intended to FPGAs appeared on the market. Among them are CASH [64], C2H [65], Impulse C [66], Catapult [67], PICO [68], AutoPilot (ESL) [69], GAUT [70], ROCCC [71] and LegUp [72]. Many of these tools are built on standard C compilers. The C code goes through several transformations and optimizations such as the elimination of unused code, unrolling loops (creation of the data flow graph [DFG]). Then, the constraints that characterize the hardware (FPGA platform) and the constraints inserted by the developer are regarded. The process is completed by scheduling operations, instantiating operators and generating the RTL code.

In the generation of hardware, the optimizations made for the compiler on the source code have an impact on the performance, the size of the generated material and on the compilation time as studied in [73]. Some of the most used HLS tools in the market come from biggest FPGA manufacturers (Xilinx and Intel), and from the producers of test and simulation tools (Matworks and National Instruments). These HLS tools become widely used on the market.

- Xilinx's Vivado HLS [2]: This tool uses C, C ++, or SystemC code as input to generate the hardware. The design flow is shown in Fig. 3.3:. It uses the AutoPilot compiler (originally developed by AutoESL) for generating the RTL code. The tool enables the implementation and reuse of test benches for simulation and verification, provides packaging capabilities and performance estimation. It provides directives to allow the user to control several optimization techniques such as unrolling/pipelining loops and partitioning of arrays.

- Intel OpenCL SDK [3]: In this tool, the OpenCL C code is transformed and divided into Verilog code for the hardware (accelerator) and into dynamic libraries for the host (CPU) as shown in Fig. 3.4:. An emulator makes it possible to verify the functioning of the code before profiling functions and applying optimizations for implementation on FPGA. The code Verilog is implemented with Quartus II.
- MathWorks HDL coder [74]: generates synthesizable HDL Code form Symulink models or Matlab functions. The data types are transformed from floating point to fixed point before the HDL generation. The work flow is integrated with Vivado and Quartus to synthesize and program the FPGA.
- LabVIEW FPGA [75]: offers the possibility of creating a virtual instrument (VI) using the graphical approach given by LabVIEW. The Xilinx tool chain is used for synthesis purposes.

The use of commercial HLS tools facilitates the design exploration and is suggested as an alternative to reduce design effort and increase productivity [76, 77]. These tools are already used in various fields: including cryptography [78], telecommunications, image processing [79], video and finance [80]. The latter presents a performance study for the acceleration of benchmark implementations in pricing. It compares the usage of Vivado HLS, Intel OpenCL SDK, and other CPU and GPU-based platforms. The study reports an acceleration of 221× for HLS tools compared to 31× for CPUs multi-core, 60× for GPUs, and 207× for Xeon Phi. These studies focus on computational acceleration and don't focus on the real-time simulation. The use of the HLS approach in the real-time simulator development is a field to be exploited which literature continues to increase. One of the first applications shows the usage of Vivado HLS (VHLS) [81] for implementing a real-time simulator for a half-bridge inverter to control an induction heater. Since then, multiple simulators have been proposed using HLS approach [50, 82].

In the frame of this work, we used VHLS to carry out a comparative study between VHLS and custom design in the development of real-time simulators. VHLS makes design exploration easier and reduces the need for detailed knowledge of FPGA development. However, the

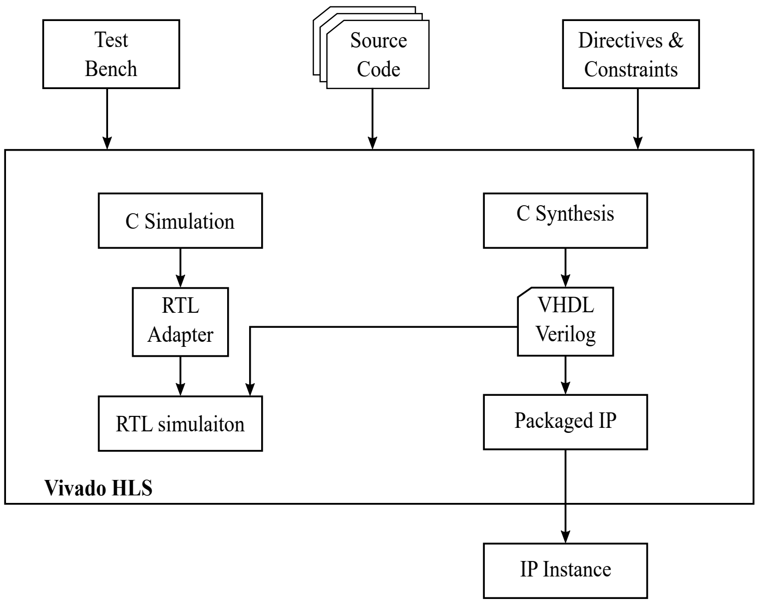


Figure 3.3: Design flow with Vivado HLS. (Inspired from [2])

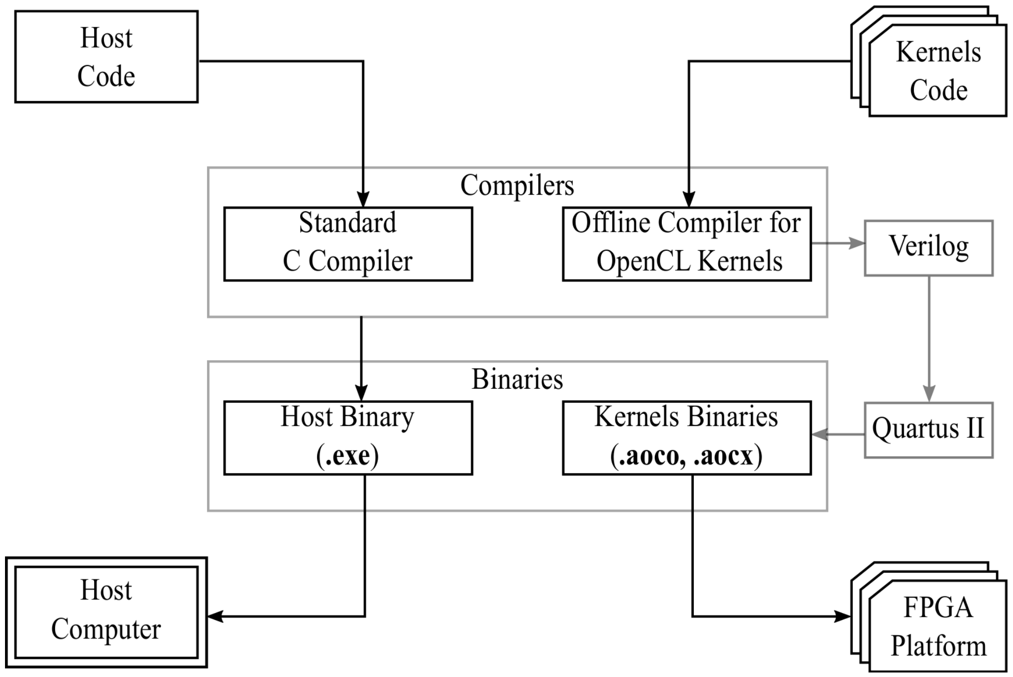


Figure 3.4: Design flow with Intel SDK. (Inspired from [3])

performance obtained with these tools is still behind that obtained with a custom RTL development. The methods, approaches and results of this study are discussed in Chapter 5.

A study presented in [83] compares the utilization of IP, Matlab, Vivado-HLS, Labview, and VHDL as design approaches for the development of real time simulators for power converts. A buck and a full-bridge converters are used as case study with implementation in fixed and floating point. Due to the relatively simple nature of the models, a small time step was obtained in each case. The time step varies from 50.7 ns to 375 ns, and from 12.8 ns to 150 ns for the buck converter in modes fixed point and floating point respectively. Similarly, the time step varies from 70 ns to 158 ns, and form 19.9 ns to 575 ns in fixed-point and floating point respectively for the full-bridge converter with losses. The complexity of design goes from very high (VHDL) to low (LabVIEW). This study put Vivado HLS as the best choice for the trade-off between design effort and performance.

3.3.3 Overlay Architectures (OA)

The digital design over FPGA for a range of applications can be made at different levels, each of them presenting its benefits and drawbacks. If a custom RTL approach is followed, all the resources of the FPGA will be available to the designer. This implies that higher performances can be reached. However, the difficulty of designing over FPGAs implies a recurrent need of an FPGA designer which increases the costs. Additionally, the design iteration cycle tends to be long due to the synthesis and implementation processes. A synthesis process can take one day or more depending on the complexity of design. The use of HLS approaches aims to simplify the applications implementation over FPGA. It allows the use of programming languages as C/C++ to facilitate the access of FPGAs to software developers and application engineers. When using HLS, the efficiency of designs can be reduced. Although modern FPGA technology and HLS tools have improved enough to increase the efficiency of the design, the synthesis process remains a time-consuming task that affects design iteration time. Overlay Architectures have been introduced in projects like ZUMA [84], Quku [85], and DeCo [9] as an attempt to tackle these drawbacks and improve productivity [86]. An OA is a regular arrangement of interconnected tiles forming an easily re-configurable fabric implementation (hardware architecture) over FPGA. A software tool is used to extract the DFG from the application and map it into the hardware architecture.

The way the tiles are interconnected defines the topology of the OA. Topologies with high connectivity offer multiple alternatives for data path creation. This increases the flexibility allowing many types of applications to be mapped on it. Some of these topologies are bi-dimensional arrays of tiles. Each tile exchanges data (back and forward) with its fully connected four neighbors (top, bottom, left and right). The Torus [8] and The Nearest Neighbor [4, 87] are good examples of highly connected OA topologies. However, the high

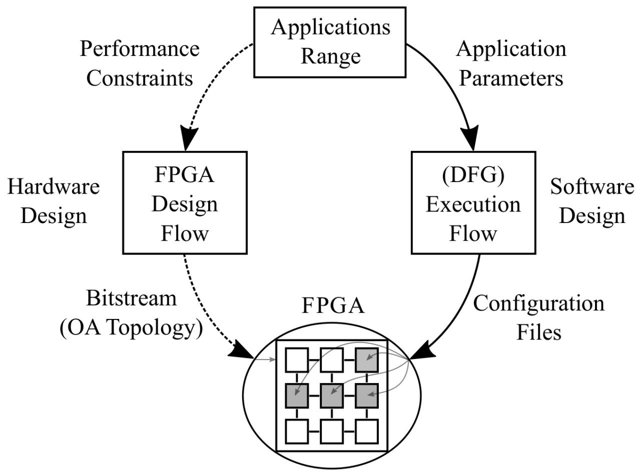


Figure 3.5: OA Design —General flow (Inspired from [4])

connectivity increases the overhead and affects complexity, cost and performance. Furthermore, the possibility of having multiple paths for data to reach their destination increases the design effort for control and synchronization. Linear OA topologies [9, 88] group the tiles on levels or stages allowing simplex data transfers between one stage and the subsequent. The limited interconnection scheme reduces the flexibility and range of applications. However, substantial gains can be obtained on simplicity and performance.

To have a good trade-off when developing an OA, the range of applications on which it will be used must be well known. This allows to have a perfect view of the performance requirements and the surface constraints for the hardware implementation (generation of the bit stream), which helps the hardware designer to choose an appropriated topology. Similarly, the software tool is used to automatically extract the application parameters to generate the OA configuration files. A global view of this process is presented in Fig 3.5:. The generation of the bitstream and the software for mapping is a one-time investment activity. Then, the application engineers can use the OA and the design interface to run their applications in a shorter design cycle. The design exploration can be done without considering the details of the FPGA implementation neither re-synthesizing the AO. Multiple commercial tools, for real-time simulation, use OA on their solutions over FPGA.

OPAL-RT's eHS is a good example of OA architecture utilization for the implementation of general propose solvers [89–91]. eHS uses the ADC model for switches. Sub microseconds time-step simulations of multiple converters configurations can be programmed with different scenarios. Its architecture is based on customized floating point arithmetic operators to achieve low latency. The implementation is done using System Generator following an RTL approach. The modification of eHS architecture to generate a new version of this tool remains a hard task that only a hardware specialist can perform. Another commercial OA example is offered by typhoon [40]. It is composed by an array of standard processing cells (SPC) connected to the I/O system, signal generator, and specialized solvers through a low latency NoC. A central control unit is used to synchronize operations. Each SPC contains a linear state-space solver, and a programmable state machine is used as a topology selector. The matrix-vector multiplication is done using a MAC unit. The stored data are floating point while the accumulator is in extended fixed point. This implies a conversion fix to float for each result. In this platform the circuit is analyzed and partitioned offline. The matrices related to each subcircuit are mapped to the different SPC and stored in their internal memories.

It is worth noticing that updating these tools for supporting new characteristics (e.g., upgrading its basic operator characteristics) remains a task reserved for the hardware expert that can take long development time.

3.3.4 Latency Insensitive Design (LID)

The maximal clock frequency at which a digital design can work is constrained by the critical path —longest propagation delay caused by the logic between registers. To improve the timing constraints it is usual to pipeline. This technique consists in splitting the logic by an appropriate register insertion to reduce the critical path. The depth of pipelines on all paths must be equivalent for warranting data synchronicity. The iterations needed to improve the timing of the system involves running the synthesis process to detect the critical path and may require some hand work which can take a long time. The use of LID helps to tackle this issue. LID proposes to build distributed digital systems by connecting pre-designed modules using channels. Thus, the process of design can be split as i) functional modules design, ii) interface communication protocols [92]. The designers can agree on the protocol to be used, and concentrate most of their energy in the design of the functional modules following a correct-by-construction approach [93].

A LID compliant module is a wrapper (known as a shell) containing the basic functional module (pearl), some memory arrangement and a control unit as shown in Fig. 3.6::

- The pearl is the main processing element that should operate only when all input data are available and the module connected to the output port is available to receive new data. It is required for the pearl to be stallable.
- The memories offer data contention to warrant the correct functioning of the module independently from input arrivals. Normally these memories are FIFO structures.

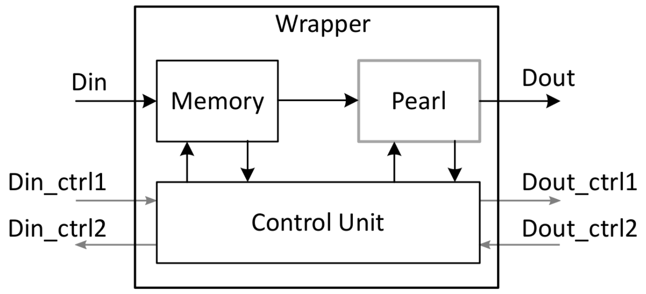


Figure 3.6: LID module —General structure(Inspired from [5])

- The control unit implements the communication protocol, manages the status of the memory and monitors the status of the module connected to the outputs. It activates the pearl when all the input data required for processing have been received and the module connected to the output is available to receive new data.

- The communication port (channel) is formed by data signals and control signals. Normally, two opposite control signals are used for handshaking. In Fig. 3.6:, these signals are identified as *xxx_ctrl1* and *xxx_ctrl2* on each port where *xxx* is the name of the port. *xxx_ctrl1* is a forward signal indicating that a data has been produced. In most applications it is referenced as *valid* or *void* signals. *xxx_ctrl2* serves as back pressures signal to advise if the module is able to receive new data. Depending on the protocol, this signal can be treated as ready, stop or increment in credit base systems [94]. In recent developments, the channel communication schemes have been generalized and used in HLS tools to interconnect modules [95,96] and expanded to be used in multiple FPGA applications [97].

The main drawback of the LID approach is the overhead due to wrapping elements. It can cost around 10% in LUTs and registers for systems with good local connectivity and medium granularity [5]. However the increased modularity benefits offered by LID help to reduce the design time [98] and make it a desirable approach to follow. Although the LID approach has been used on multiple applications (i.e., video processing) [99], we did not find any explicit evidence of LID utilization for the hardware design of real time simulators for power converters. In this work, we propose the utilization of LID for the implementation of real time simulators. It will be presented in chapter 7.

3.4 Summary

A condensed view of the works most related to this research project is presented in this section. Although it wasn't always possible, we extracted the most relevant information and presented it in a condensed way. Table 3.1: presents the design reference, the platform used, and the synthesis results. Table 3.2: presents the design tool (or approach), the simulated circuit size (defined by the number of states on the circuit—a switch is considered as two states), and the performance when the information was available. One factor we include as comparative point is the computing power (CP) measured in GFLOPS. In most of the applications, this value wasn't explicitly mentioned. To put them in common ground, we decided to make a global conversion if the information wasn't explicitly available. Considering that two DSPs are needed to implement a single FP operator, we have $CP = (FREQ_{(MHz)}/1000) * DSP/2$. We can see that a higher CP is required for obtaining sub-microsecond time steps as the circuit size increase. However, the FP number format adds latency on operators which impacts negatively the Ts. When the solutions use the FXP approach, as done by Liu (2020) [10] and Milton (2017) [100], the circuits can be solved in one clock cycle that corresponds to the time step. This way a small time step can be reached with implementations at a low frequency.

Although it depends on the designer's experience, the use of the RTL approach presents a resource economizing —particularly DSPs— comparatively to HLS especially if FP is used. The highest implementation frequencies are reached when pipelining is used, which is common for circuits using FP format. For two circuits of similar size with implementations in different numeric format, the FXP seems to be privileged by its performance. However, the dynamic range of values offered by FP make it preferable for keeping an acceptable precision in generic applications. Except for Montaño (2018) [11], the implementations presented in this table are mainly customized design for a circuit and don't present any information about scaling up and performance affectation. The synthesis process must be relaunched if any modification must be done to the circuit to be simulated.

In Chapter 8, we'll take back the information presented in Tables 3.1: and 3.2:, and we'll add the information related to our solution to perform a comparison with our results.

Table 3.1: Synthesis Results from State of the Art

Ref.	**FPGA**	**Registers**	**LUT**	**RAM**	**DSP**	**Freq. (MHz)**
M. Dagbagi (2016) [6]	Artix 7	1,197 (9.0%)	1,463 (11.0%)	NR[1]	186 (85.0%)	100
H. Chalangar(2020) [52]	Kintex 7	2,373 (0.6%)	1,223 (0.6%)	22 (4.9%)	88 (10.5%)	320
C. Liu(2020) [10]	Kintex 7	45,509 (9.0%)	54,366 (21.4%)	91 (11.4%)	210 (13.6%)	40
F. Montaño(2018) [11][a]	Kintex 7	83,272 (20.4%)	73,161 (35.9%)	NR	703 (83.7%)	100
F. Montaño(2018) [11][b]	Kintex 7	21,136 (5.2%)	35,726 (17.5%)	NR	128 (15.2%)	100
M. Milton(2017) [100]	Virtex 7	8,932 (1.5%)	87,525 (29.0%)	NR	1,884 (67.0%)	20
R. Mirzahosseini(2019) [50]	Virtex 7	147,317 (24.3%)	142,296 (46.9%)	258 (12.5%)	361 (12.9%)	175
C. Liu (2018) [101]	Kintex 7	42,642 (11.2%)	47,028 (25.8%)	91 (11.6%)	120 (17.7%)	200
T. Ould-Bachir(2015) [48]	Virtex 6	58,216 (19.3%)	29108 (19.3%)	41 (9.8%)	116 (15.1%)	400
D. Majstorovic (2011) [40]	Virtex 5	NR[1]	NR[1]	117 (89.0%)	40 (14.0%)	200

[1]NR:Not Reported, *a*: solution 1, *b*: solution 2

Table 3.2: Design Characteristics from State of the Art

Ref.	**Number Format**	**Design Tool**	**Application**	**Size (Nb States)**	**Ts (ns)**	**CP (GFLOPS)**
M. Dagbagi (2016) [6]	FXP[1]	NR[4]	VSR	21	420	9.30
H. Chalangar (2020) [52]	FXP	NR	LLC	22	40	14.10
C. Liu (2020) [10]	FXP	NI[5]-HLS	AC-DC-AC	25	25	4.20
F. Montaño (2018) [11][a]	SFP[2]	HLS	NPC	41	630	35.10
F. Montaño (2018) [11][b]	CFP[2]	RTL	NPC	41	310	6.40
M. Milton (2017) [100]	FXP	HLS	VSC($\times$6)	140	50	18.84
R. Mirzahosseini (2019) [50]	SFP[3]	HLS	VSC	64	180	31.59
C. Liu (2018) [101]	FXP	NI-HLS	NPC	41	40	12.00
T. Ould-Bachir (2015) [48]	CFP	SG[6]-RTL	NPC	41	750	23.20
D. Majstorovic (2011) [40]	FP-FXP	RTL	AC-DC-AC	37	1000	12.00

[1]FXP:Fixed Point, [2]CFP:Custom Floating Point, [3]SFP: Single Floating Point, [4]NR:Not Reported, [5]NI:National Instruments, [6]SG:System Generator, *a*: solution 1, *b*: solution 2

3.5 Conclusion

This literature review provided an overview of the various aspects related to our research subject. We presented the different approaches and considerations for the design and implementation of real time simulators for electrical networks. In this journey, we have found that a part of the present solutions use a low-level approach, therefore difficult to update. On the other hand, today's HLS tools don't perform as well as those of a hardware development expert. This is in part due to the lack of mechanisms for inserting dedicated high-performance arithmetic operators. However, the utilization of HLS approach has increased. It can be considered to facilitate the development of solvers on FPGA. Latency insensitive design has good potential for increasing modularity and facilitating design, but hasn't been reported for applications in real-time simulation. In contrast, overlay architectures are used for implementing a generic solver for real time simulators to facilitate the access of FPGA to application engineers and software developers. Finally, we present a condensed view of the works more related to our research. It puts in evidence the technique of design followed as well as the performance obtained. These tables will be used in chapter 8, when we present a discussion of our contribution.

CHAPTER 4 CONTENT ORGANIZATION AND RESEARCH APPROACH

This thesis follows a by-article format to include the realized work on the frame of this doctoral research. The three following chapters present the research work submitted to peer review and published as three different papers. The methodology for the implementation of real-time simulators on FPGA is introduced and analyzed. The papers are complementary and present the incremental contributions as steps followed to reach the proposed methodology for developing real-time simulators on FPGA covered in this research. The papers have been chronologically organized in the body of this work, starting with the state of the art for the implementation of real-time simulators for custom design and high-level synthesis including their benefits and limitations. In a second step, the analysis and implementation of an efficient interconnection system are presented. It introduces the latency insensitive design (LID) paradigm as a development approach. The final paper presents the whole methodology which integrates LID and OA for the development of the power converter's real-time simulators.

The existent literature shows that real-time simulators implemented in FPGAs allow small time-step solvers implementation. Most of these implementations are custom RTL designs with few possibilities of reconfiguration. The complexity associated with the RTL design requires the knowledge of hardware specialists and limits the intervention of the application specialists or software developers which is presented as a drawback. HLS approaches are more friendly for application specialists and software developers. Although the HLS approach has already been used for the implementation of solvers for small applications, their performances are reduced compared to custom RTL design. This condition established the starting point of our research where a comparative analysis between RTL versus HLS design is performed. This study led us to a first publication and framed certain potential weaknesses observed with the use of HLS in its current state which allowed us to propose an alternative solution and gave way to other publications.

The first article presents an analysis for the utilization of commercial tools for the implementation of real-time simulators over FPGA following a high-level synthesis (HLS) approach. This analysis allowed us to compare the ratio cost/performance obtained for a design developed by a hardware designer against a design using an approach HLS generated with Vivado HLS. The results obtained during this phase have shown us that there are still some limitations on the performance/resource ratio. The custom design outperforms HLS design when the required time step is inferior to 1 μs. One of the main HLS limitations, detected at that moment, is that the tool doesn't offer the possibility to insert any custom-made operator neither a mechanism for addressing data to them, which could be more cost/performance efficient. However, as some collected results show that under μs simulations can be achieved with HLS, this approach remains an interesting option because it eases the design process.

When small calculation time is required, it is an obligation for the data propagation mechanisms to be as efficient and as fast as possible. Furthermore, as the FPGA resources are limited, their utilization must be conservative and the integration of the system should be relatively simple. To facilitate the design integration, the introduction of latency insensitive design paradigm is considered for creating a high-performance interconnection mechanism (MIN), interfacing the arithmetic operators (DPs) and the memory system. Taking into account the conditions before mentioned, it will be possible to generate efficient, high performance real-time simulators in a resource constraint device in a relatively simpler manner.

The second article tackles the problem of transferring data at high throughput in a cost/performance efficient way. In this work, we propose the architecture of a scalable Multi-stage Interconnection Network (MIN) that can be used for multiprocessor systems. The MIN is an array of basic cells or switches 2×2. It supports multicasting, broadcasting, and data contention capabilities with a straightforward routing algorithm for address coding/decoding. The pipelined structure and the simplicity of its design allow it to operate at a high frequency (>500 MHz). In this phase, we use LID methodology.

The third article introduces a design methodology for the development of the complete real-time simulation system. This methodology proposes the utilization of a programmable overlay architecture with a decentralized control scheme. It follows the LID paradigm for the implementation of real-time simulators. The architecture can be an alternative to the high-level approach parallel to the one studied in the first article. It integrates the MIN presented in the second article. The control unit is integrated into each functional block. The data transfers are performed based on events on their point-to-point communication channels instead of signal timing. This methodology allows the introduction of cost-effective custom floating-point operators. The proposed architecture has been used to evaluate the time step vs. resource consumption for a wide range of power converters sizes. The results show that sub-microsecond time-step operation and high computational performance (300 GFLOPS) are possible using the proposed architectures.

A general discussion and a conclusion complete the dissertation and synthesize the work realized in this research.

CHAPTER 5 ARTICLE 1: AN EVALUATION OF A HIGH-LEVEL SYNTHESIS APPROACH TO THE FPGA-BASED SUB-MICROSECOND REAL-TIME SIMULATION OF POWER CONVERTERS

F. Montaño, T. Ould-Bachir, and J. P. David, "An evaluation of a high-level synthesis approach to the FPGA-based sub-microsecond real-time simulation of power converters," IEEE Trans. Ind. Electron., vol. 65, no. 1, pp. 636–644, Jan 2018.

Abstract

This paper evaluates the benefits of using a high-level synthesis (HLS) tool to develop FPGA-based real-time simulators for power electronics systems. The investigated workflow generates a synthesizable hardware description from a system level C-code along with a set of directives that specify performance criteria such as area utilization and timing closure requirements. The performance of the HLS approach is evaluated for different circuit sizes and target clock frequencies. Results show that HLS can be used for Hardware-in-the-loop (HIL) applications when the circuit to be simulated is small and the target clock frequency is not too high (up to 100 MHz). For larger circuits and higher clock frequencies, HLS will either require a simulation time-step that is too large for real-time simulation purposes, or will tend to use almost all of the FPGA resources.

Keywords

Power electronics, real-time simulation, HIL, FPGA, high-level synthesis.

5.1 Introduction

Hardware-in-the-loop simulation is recognized as an economic and efficient industrial prototyping approach for the design of power system controllers [102]. The high switching frequency requirements of modern power converters ($>$ 20kHz) [48, 103–106] are however very challenging for real-time simulators because the simulation time-steps for such systems must be in the sub-microsecond range to meet criteria such as a fine time resolution on the gate signals and accurate simulation results [107]. In the recent years, FPGA devices have been used in HIL simulators to fulfil these requirements, to enhance their computing capabilities and to reduce the total latency of the simulation loop [108].

However, FPGA programming is a difficult task that requires specialized skills in hardware design. FPGA designs are usually based on Register Transfer Level (RTL) abstraction to describe a digital circuit. At that level, registers contain the current state of the system.

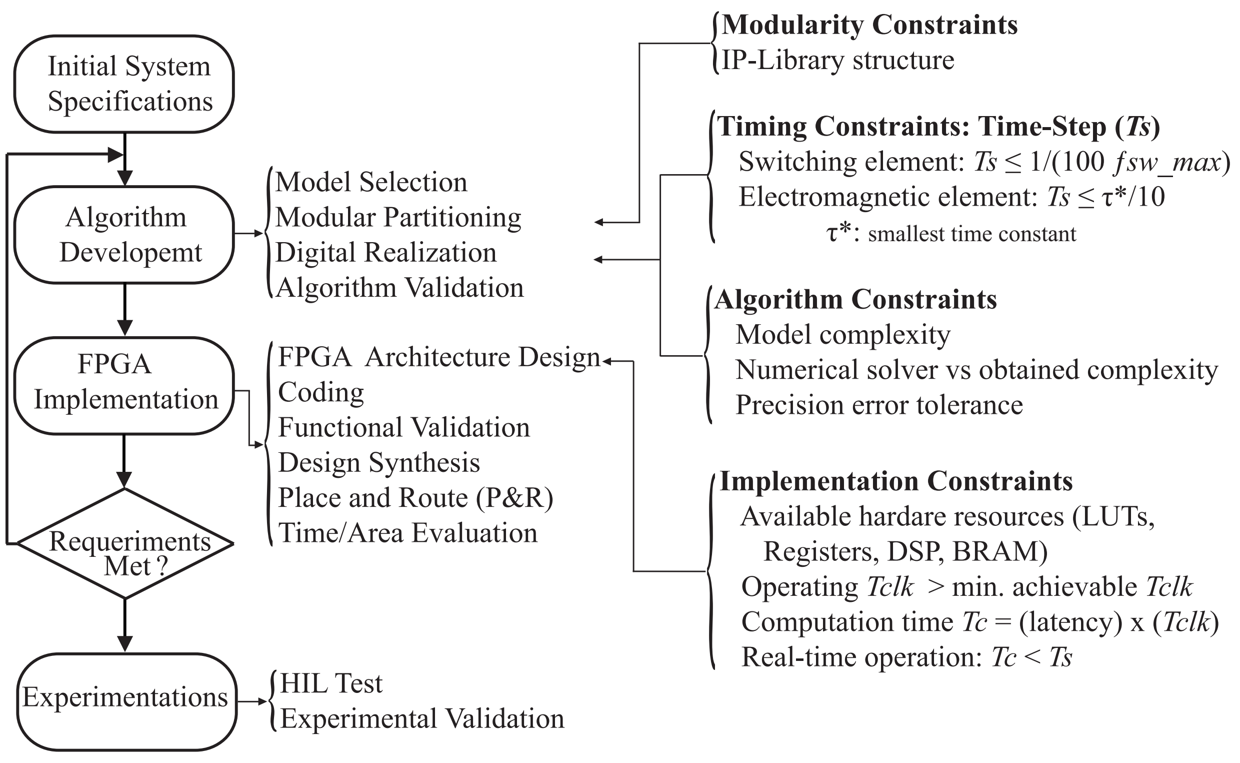

Initial System Specifications
Algorithm Developemt
FPGA Implementation
Requeriments Met ?
Experimentations
Model Selection
Modular Partitioning
Digital Realization
Algorithm Validation
FPGA Architecture Design
Coding
Functional Validation
Design Synthesis
Place and Route (P&R)
Time/Area Evaluation
HIL Test
Experimental Validation
Modularity Constraints
IP-Library structure
Timing Constraints: Time-Step (Ts)
Switching element: Ts ≤ 1/(100 fsw_max)
Electromagnetic element: Ts ≤ τ*/10
τ*: smallest time constant
Algorithm Constraints
Model complexity
Numerical solver vs obtained complexity
Precision error tolerance
Implementation Constraints
Available hardare resources (LUTs, Registers, DSP, BRAM)
Operating Tclk > min. achievable Tclk
Computation time Tc = (latency) x (Tclk)
Real-time operation: Tc < Ts

Figure 5.1: Design guideline for FPGA-based real-time simulator design. Adapted from [6]

At each clock cycle, the state is updated with a combinatorial function of itself and inputs. The functions involved are typically implemented with Arithmetic and Logic Units (ALU) or dedicated resources [109]. The low abstraction level of RTL increases the development cost and the time to market of FPGA-based simulators for HIL systems, thus restricting their use.

A useful design guideline to develop FPGA-based real-time simulators is presented in [6] and summarized in Fig. 5.1:. Passing from system specification to HIL experimental validation involves several steps including modeling, partitioning, coding and validation. Many constraints must be satisfied including libraries and resources availability, timing and accuracy.

The development of FPGA-based systems is in itself a multi-step process as depicted in Fig. 5.2: [7]. Passing from system description to a circuit description requires several steps on three different axes (behaviour, structure, physical). Most error prone conditions are present in algorithmic development, hardware description and their verification (typically done by human beings) while logic and circuit transforms are automatically computed by tools, which are supposed to reduce the presence of errors.

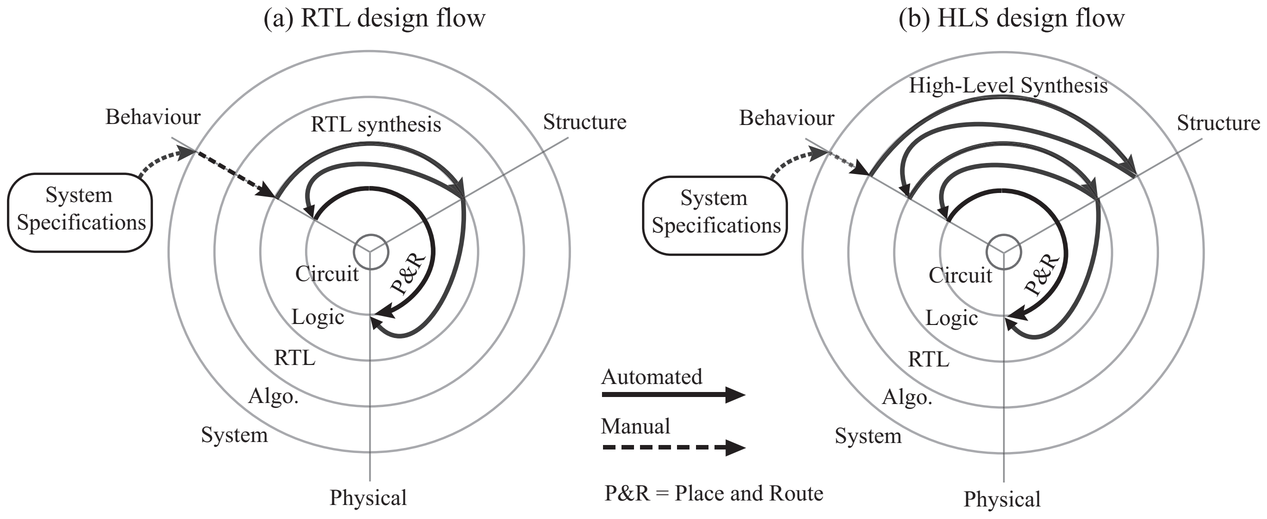


Figure 5.2: Y-chart for (a) RTL and (b) HLS. Adapted from [7]

Working at an abstraction level higher than RTL, with dedicated languages and tools, is believed to reduce the need for hardware specialists and to increase productivity [76, 77]. Important research efforts have been invested in the recent years to allow efficient exploitation of the FPGA potential with high-level programming languages [110].

Nowadays, High Level Synthesis tools available in the market such as Vivado HLS (VHLS) from Xilinx [111], and OpenCL SDK from Altera [112] allow the use of high-level languages to ease design and verification of hardware. Since hardware under development must meet certain performance criteria, HLS tools offer the use of configurations and directives to support pipelining, duplication, and other digital design techniques used to improve hardware performance [113]. Recent works suggest that such tools could be employed for developing high performance computing units, designated hereafter as Hardware Solvers (HS), for real-time simulation applications [81, 110].

The main goal of this paper is to evaluate the benefits of using VHLS to design FPGA-based hardware solvers with reduced simulation time-steps for HIL applications. The evaluation takes into account the design operation frequency, the resources requirements as well as the accuracy of the simulation.

The remainder of this article is organized as follows: Section 5.2 reviews the works related to this research. Section 5.3 presents a design exploration comparing the performances of VHLS against custom design. Section 5.4 presents case studies where actual power converters are considered. A conclusion is given in Section 5.5.

5.2 Related Work

FPGA devices are used in HIL simulators to model and simulate power converters with small time-steps. Reported time-steps vary from 12.5 ns [54, 107] to 0.5-1 μs [40, 53]. Most of the time, a low-level design approach is used to describe the FPGA design, whether using Hardware Description Languages (HDLs) such as VHDL or Verilog or specialized block-set toolboxes such as Xilinx's System Generator [114]. Additional examples can be found in [27, 48, 115, 116]. The number format adopted for these implementations are either fixed-point arithmetic [42, 53, 107] or floating-point arithmetic [27, 40, 48]. Fixed-point designs are characterized by smaller footprints and time-steps, whereas floating-point designs offer better accuracy and larger dynamic range.

The HLS approach to FPGA-based real-time simulation in the field of power electronic systems is at its early stage of exploration. Very few works have been reported to date. This includes [117], an FPGA-based feedback controller; as well as [81, 110], a controller for a resonant half bridge power converter, and its HIL emulator for testing heating appliances. In each case, high-level description was demonstrated to fulfill the expected performances and the small simulation time-step was highlighted. However the small size of the simulated circuits does not exploit the full potential of current FPGA and the related works don't explore the possible pitfalls associated with larger circuits.

More generally, none of the aforementioned works apprehended the evaluation of the HLS approach in broader terms for the HIL simulation context. Such evaluations do exist for other application fields: [80] offers a comparison between using HLS tools and other technologies in financial engineering; [78] compares HLS to hand-written code in cryptography; [79] evaluates HLS for image processing; [76] evaluates the use of HLS in telecommunications. However, these studies are concerned with using HLS for developing FPGA-accelerated computing engines, whereas the objective of this study is to conduct a thorough evaluation of the benefits of HLS for hardware solvers with low calculation time-steps as the primary target result.

5.3 Design Exploration

A Hardware Solver performs mainly a matrix-vector multiplication — please refer to Appendix 5.6 for details about power circuit modeling. This operation has a reduced data dependency, which allows a high degree of parallelism. To evaluate the effectiveness of VHLS for the development of HSs, we conduct a design exploration study investigating the relationship between the size of the electric circuit, the achievable time-step, the working frequency of the FPGA design, and the cost of the implementation. All the HSs are designed using:

1. Custom design approach;
2. VHLS approach constrained by resources;
3. VHLS approach constrained by latency.

5.3.1 Custom Design

In this approach, the HS is implemented using System Generator v.14.4 from Xilinx. Fig. 5.3: presents the architecture of the custom designs, which consists of five fundamental modules:

1. The Updater is formed of two logic blocks that analyze gating signals and state variables and dynamically update the vector registers file.
2. The Keeper contains the vector registers and the matrix representing network equations. The matrix is saved in RAM blocks arranged to allow simultaneous access to multiple elements at a time. RAM blocks are filled using a Matlab m-script. The circuit inputs are directly sent to the input vector registers.
3. The Matrix-Vector Multiplier is formed by eight dot-product (DP) units working in parallel. The DP operator is a custom design optimized for low-latency floating-point operation [118]. Each DP has eight multipliers connected to an adder tree and terminated by an accumulator as depicted in Fig. 5.4:. Each DP uses 16 DSP blocks. Table 5.1: shows the latency of the DPs for each working frequency.
4. The Output Selector routes matrix-vector multiplication results. History terms (referred to as states in the following) are fed back to the Updater module for the next computing cycle, while the remainder of the results form the outputs of the HS.
5. The Control Unit schedules all operations for vector updating, memory data reading, dot-product operation, data feedback and outputs activation. It is a state machine formed by a counter and memory blocks. Since the scheduling is determined during design, the content of the memory blocks is precomputed by an m-script with respect to circuit specifications, and the target simulation time-step.

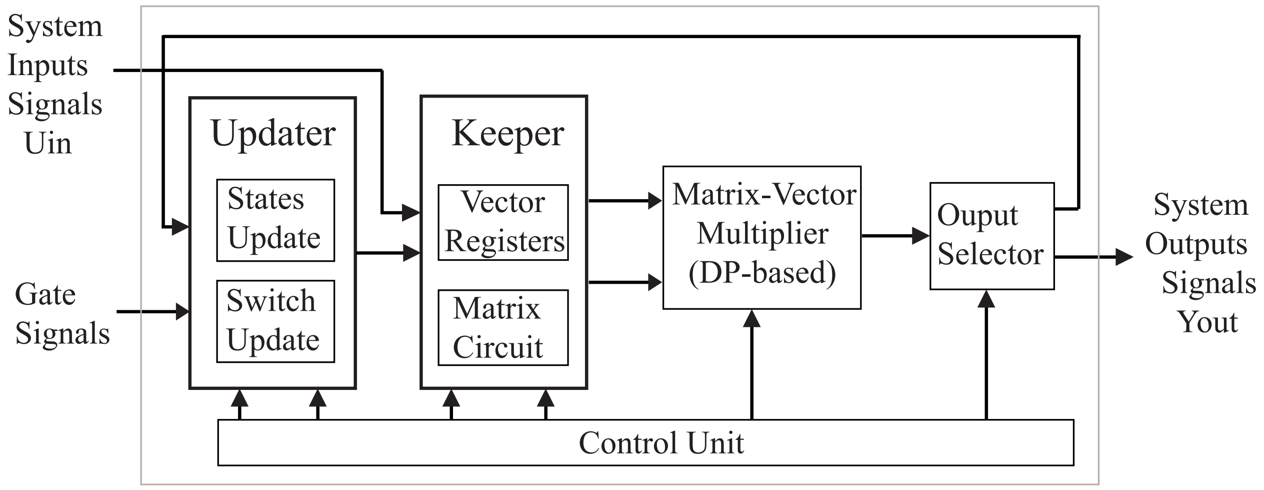


Figure 5.3: Custom design hardware solver architecture

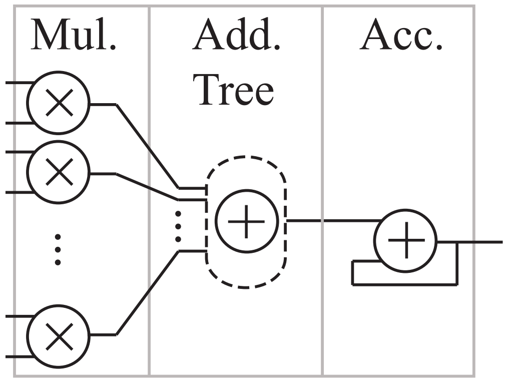


Figure 5.4: DP operator structure

Table 5.1: DP Latency for the Various Operation Frequencies

Frequency (MHz)	40 - 50	100	200	250	320
Latency (Clk Ticks)	2	5	13	14	16

5.3.2 VHLS Approaches

For the two VHLS approaches discussed in Section 5.3.2 and Section 5.3.2, the HSs are described by the C-code of Fig. 5.5: and implemented using Vivado HLS 2015.2. Even though directives can be introduced directly in the source code as pragmas, we have grouped them in reusable attachable directive files to facilitate design explorations without changing

```c
void hw_solver(
  float y[nb_outputs],
  float u[nb_inputs],
  int gates){

  static int sw = 0;
  static float x[nb_states] = {...};
  static float swhist[2*nb_switches] = {...};
  const float A[nb_rows][nb_columns] = {...};
  float b[nb_columns];
  int i, j;

  // updates switch status
  sw = switch_update(gates, sw, swhist);

  vector_forming: for(i=0; i<nb_columns; ++i){
    // puts u, x and
    // sub values from swhist into b
  }

  // Matrix-vector multiplication
  float * results = mvmult(A, b);

  assign_vects: for(i=0; i<nb_rows; ++i){
    // puts sub values from results into
    // swhist, x and y
  }
}
```

Figure 5.5: Excerpt from the C-code used for the design of the HS in VHLS

the source code. The whole design of the solver is implemented using the code structure of Fig. 5.5::

- `switch_update(...)` updates switch statuses according to the gating signals, previous switch statuses, and the currents and voltages associated with the switches (`swhist`). This function is not used in this design exploration but is used in Section 5.4.

- The `vector_forming` loop groups circuit inputs, states and switches conditions to update the state/input vector.

```
set_directive_array_partition -type complete
    -dim 1 "hw_solver" yout
set_directive_array_partition -type complete
    -dim 1 "hw_solver" uin
set_directive_top "hw_solver"
set_directive_pipeline "hw_solver"
set_directive_allocation -limit 64
    -type operation "multiplication" fmul
set_directive_allocation -limit 64
    -type operation "multiplication" fadd
```

Figure 5.6: Directives for the VHLS approach constrained by resources

```
set_directive_array_partition -type complete
    -dim 1 "hw_solver" yout
set_directive_array_partition -type complete
    -dim 1 "hw_solver" uin
set_directive_pipeline -II $II "hw_solver"
set_directive_latency -max $lat "hw_solver"
set_directive_top "hw_solver"
```

Figure 5.7: Directives for VHLS approach constrained by latency

- `mvmult(...)` takes the vector composed by `vector_forming` and performs matrix-vector multiplication. This is the most time and resource consuming function. It is composed of two nested loops with a multiply-accumulate operation at the inner loop level.
- The `assign_vects` loop separates the matrix-vector multiplication results and builds the state vector (`x`), switch voltages and currents (`swhist`), and the output vector (`y`).

VHLS automatically generates inputs/outputs, control signals, internal control logic, and performs scheduling and binding processes. Voltages and currents use floating-point variables whilst an integer variable is used for gating signals. VHLS maps the variables at the top level function to input/output ports automatically. Matrix values are internally saved in a constant matrix that can be mapped into memory blocks or registers.

VHLS Approach Constrained by Resources

In our first attempt, and in order to provide a fair comparison between the custom designs and HLS results, VHLS is constrained by resource utilisation in a way that allows the tool to match the computing power of the custom designs. Hence, allocation directives are applied to limit the number of arithmetic operators to 64 adders and 64 multipliers. The single precision multipliers and adders instantiated by VHLS require respectively 3 and 2 DSP blocks, which leads to a consumption of $(2+3)\times 64 = 320$ DSP blocks. Fig. 5.6: shows the directives used when VHLS is constrained by resources.

VHLS Approach Constrained by Latency

In a second round of HLS implementations, pipeline and latency directives are set to force VHLS to generate a design with the lowest time-step possible for each solver design. In this case, the tool is free to use as much resources as needed. Fig. 5.7: shows the directives used when the tool is constrained by latency, aiming for the minimum achievable time-step.

5.3.3 Implementation Results

The design exploration targets Xilinx Kintex XC7K325T-FFG676-1 FPGA [119]. A wide range of clock frequencies are investigated, namely 40, 50, 100, 200, 250, and 320 MHz (resp. 25, 20, 10, 5, 4 and 3.125 ns). The custom designs are capable of handling all frequencies, whereas VHLS designs were only successful handling frequencies up to 200 MHz. The designs are tested for various number of states ranging between 2 and 60, and their perfoamce evaluated by assessing the achievable minimum time-step and the FPGA area occupation (cost). Results are presented in Fig. 5.8: and Fig. 5.9:. Dashed lines are synthesis estimates resulting from situations where either VHLS encouters timings issues or the resources utilization exceeds the capacity of the target FPGA.

Fig. 5.8: gives the minimum time-step for the three design approaches as a function of the FPGA clock period and the number of states in the circuit. For a given clock frequency, it is observed that the time-step grows quadratically with respect to the circuit size. The growth rate is more pronounced for lower frequencies. Hence, achieving high clock frequencies is crucial when designing HSs, more so when large circuits are considered, a goal that VHLS presenly fails to fulfill for clock frequencies exceeding 100 MHz (respectively, a clock period of less than 10 ns). However, the clock frequency has little impact when the circuit size is small.

Fig. 5.9: gives the cost for the three design approaches as a function of the FPGA clock period

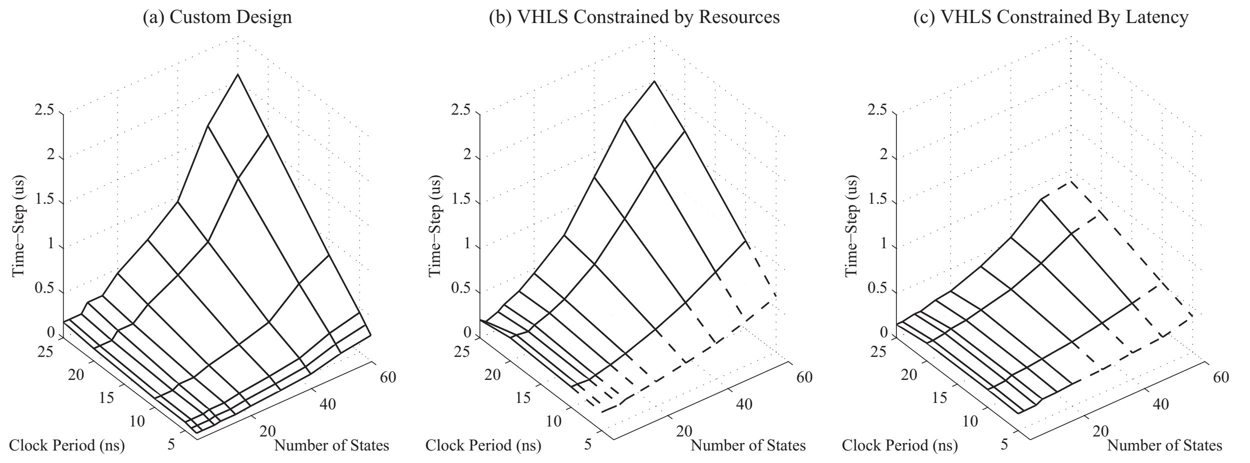


Figure 5.8: Minimum Time-step: (a) Custom design; (b) VHLS constrained by resources; (c) VHLS constrained by latency

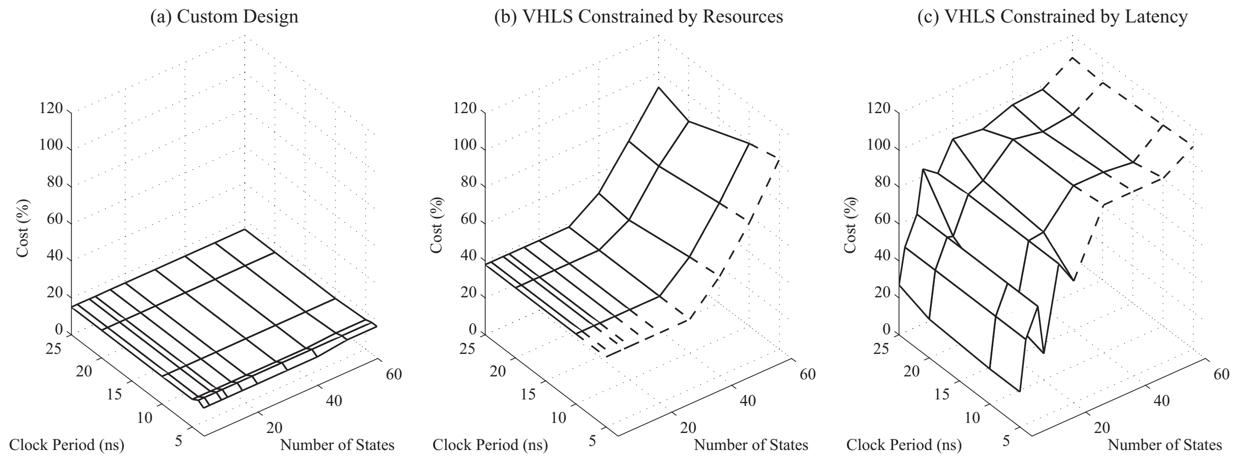


Figure 5.9: Implementation cost: (a) Custom design; (b) VHLS constrained by resources; (c) VHLS constrained by latency

and the number of states in the circuit. Typically, the cost of an FPGA implementation is a composite metric given by the number of Look-Up Tables (LUTs), the number of registers, and the number of DSP and RAM blocks. Here, a global cost is considered, computed as the maximum percentage of the resources used among LUTs, registers, DSP and RAM blocks. Such a metric is a good indicator of the number of HSs that can be implemented on a single target FPGA.

From Fig. 5.9:.a, one observes that custom designs show a flat 15% area occupation for all clock frequencies and circuit sizes. The area occupation is dominated by the DSP consumption. When VHLS is constrained by resources (Fig. 5.9:.b), the area occupation is also dominated by the DSP consumption, at least for circuits with up to 30 states. For circuits with more than 30 states, the resource consumption is dominated by reconfigurable logic and grows quickly with respect to the circuit size. Finally, when VHLS is constrained by latency (Fig. 5.9:.c), the area occupation is very high (close to 100%) for both DSP blocks and reconfigurable logic.

5.4 Case Study: Power Converter

This section considers a three-level Neutral-Point Clamped (NPC) converter connected to an RLE load, as illustrated in Fig. 5.10:. This is equivalent to a 41-state circuit.

The switches in the circuit are modeled using the Associated Discrete Circuit (ADC) model [26] to keep network equations fixed regardless of switch statuses — refer to Appendix 5.6 for a presentation of the switch model. Table 5.2: gives the parameters for the power converter and for its control.

5.4.1 Implementation Results

The hardware solver for the power converter is conceived using the three aforementioned design approaches, i.e. *i*) Custom design; *ii*) VHLS constrained by resources; *iii*) VHLS constrained by latency. All implementations target a 100 MHz clock frequency (10 ns clock period), a design trade-off motivated by the results of Section 5.3.3 and aiming to guarantee fairness in the comparison of VHLS results against our custom implementations. Table 5.3: gives synthesis estimates for the three implementation approaches.

The custom design clearly outperforms VHLS results in terms of resources and speed. The minimum time-step is smaller by more than two folds, and the reconfigurable resource consumption by 1.5 to 2 folds. DSP block utilization for VHLS is increased by 150-450% due to the use of 3 DSP blocks per floating-point multiplier and 2 DSP blocks per floating-point adder, whereas the custom designs consumes only 2 DSP blocks per floating-point multiplication. Even if VHLS appears to be a very good choice because of its ease of use, when it comes to implementation of real power converters, VHLS is presently not cost-effective for real-time simulation requirements.

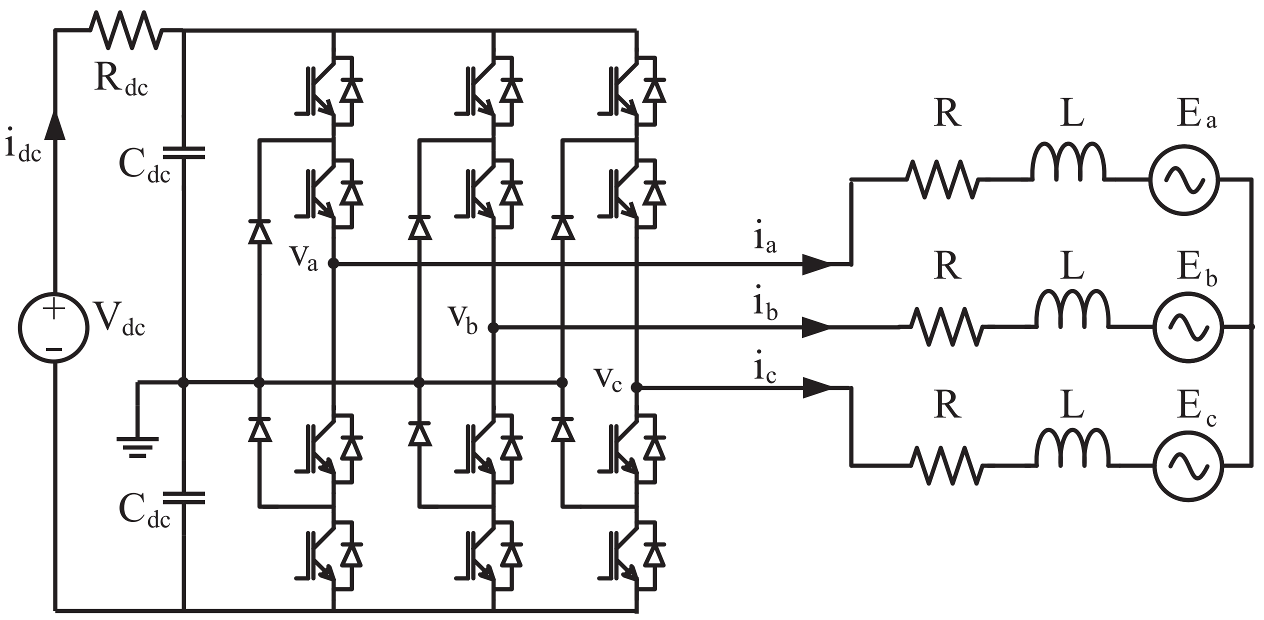


Figure 5.10: Case study: NPC converter connected to an RLE load

Table 5.2: Circuit Parameters

Parameter	Value
V_{dc}	100 V
R_{dc}	0.4 Ω
C_{dc}	1 mF
R	0.4 Ω
L	0.1 mH
PWM Modulation Freq.	400 Hz
Modulation Index	0.8
PWM Carrier Freq.	25 kHz

5.4.2 Offline Simulation Results

In this section, we evaluate the correctness of the computations executed by the proposed implementations. This is done from within Matlab, in which test vectors are generated and simulation results compared against reference outputs. Expected simulation results are generated by running a m-script file using single and double precision arithmetics. Double precision results serve as a reference to all implementations, whereas single precision results are used to make sure that FPGA-based simulations compare favourably against IEEE-754 complying code.

The results presented hereafter consider the NPC converter with a time-step of 630 ns. The simulation is executed for a runtime of 10 ms during which the inverter is controlled in open loop by a sine PWM: the modulation sine wave has a frequency of 400 Hz, the carrier frequency is 25 kHz. A modulation index of 0.8 is used. For this evaluation, we only use the results from the implementation where VHLS is constrained by latency.

Fig. 5.11:.a presents the load current on phase a: i_a. Fig. 5.11:.b shows superimposed relative errors for single precision computations (m-code), the custom design as well as for VHLS. It demonstrates that both designs are similar in their accuracy performance, and that the overall relative error is close to what is expected from a single precision IEEE-754 compliant code ('Single' in Fig. 5.11:.b), which is a relative error of less than 0.01%. The relative error is higher when i_a crosses zero, but this is to be expected as confirmed by the 'Single' curve in Fig. 5.11:.b. Fig. 5.11:.c shows a close-up view from the relative error and shows that all three curves are almost identical.

In a similar way, Fig. 5.12:.a presents the voltage on capacitor C_{dc}: V_{CAP}. Fig. 5.12:.b. show the superimposed relative errors for single precision computations, custom design and VHLS. The error remains always inferior to 0.01%.

It is worth mentioning that the custom design is able to run the NPC at 310 ns (Table 5.3:). The 630 ns time-step was used instead in order to keep a certain level of fairness towards VHLS. It is well known that reducing the simulation time-step increases accuracy. This observation simply confirms our conclusion that presently RTL design outperforms VHLS for larger circuits.

Table 5.3: Synthesis Results for the Kintex XC7K325T

	NPC Converter			
Metrics	Custom Design	VHLS (*DSP)	VHLS (*Ts)	Available
Registers	21,136	50,635	83,272	407,600
LUTs	35,726	52,249	73,161	203,800
DSP	128	320	703	840
Clock (ns)	10	10	10	N/A
Min. Δt (ns)	310	680	630	N/A

(*DSP)= Constrained by Resources, (*Ts) = Constrained by Latency

5.4.3 Online Simulation Results

This section presents the implementation and real-time simulation of the NPC converter. The FPGA design is composed of three main blocks:

1. A VHLS block constrained by latency for the NPC HS.
2. A PWM block for generating gates and E_{abc} signals;
3. An I/O interface controller.

Table 5.4: gives the resource consumption for the real-time implementation. The complete system was implemented on a OP5707 [120] target, equipped with a Xilinx Virtex 7 7VX485TFFG1761-2.

Offline simulation results for a 20 ms simulation period have been generated using SimPowerSystems (SPS) to validate real-time results. The simulation encompasses three sequences:

1. Sequence 1: The PWM is active during 7.5 ms, putting the NPC in inversion mode.
2. Sequence 2: Gate signals are turned off during 4.5 ms to force currents to zero;
3. Sequence 3: Sinusoidal E_{abc} voltages are applied to put the converter in rectification mode for a 8 ms period.

Fig. 5.13: shows the currents on the load for offline simulation. Fig. 5.14: shows the load currents from the real-time simulation, captured by a Infiniium 54845A Oscilloscope. A gain of 0.01 is applied on the analog outputs, yielding a 50 V/div. Similarly, Fig. 5.15: and Fig. 5.16: show the voltage on capacitor during inversion mode for offline and real-time simulation respectively. The channel is set to 20 V/div. Figs. 5.13:, 5.14:, 5.15: and 5.16: show a perfect match between SPS and the real-time simulation.

5.5 Conclusion

This paper presented an evaluation for the use of Vivado HLS as a design approach for the FPGA-based real-time simulation of power electronic systems. This work is the first of its kind to investigate in broad terms the use of Vivado HLS for HIL applications. The area occupation and minimum time-step of the VHLS design obtained during design exploration show that, for small circuits, VHLS produces hardware solvers with sub-microsecond time-steps and a footprint comparable to what an experienced hardware designer can achieve.

Table 5.4: Implementation results for the Virtex 7VX485T

Resource	Used	Available
LUTs	88,403	303,600
Registers	94,187	607,200
DSP	754	2,800
Block RAM	45	1,030

The simulation accuracy was also evaluated and was found to be very acceptable with a relative error inferior to 0.01%. However, for larger circuits, VHLS presently fails to fulfill HIL requirements in a cost-effective manner. Nevertheless, the simplicity of using higher level hardware description tools and the good performance of VHLS for the real-time simulation of small power converters suggests a future trend in the HIL practice where HLS will play an important role.

5.6 Appendix—Power Circuit Modeling

Network equations are obtained using a fixed time-step discretization. All components are replaced by their companion circuit, a Norton equivalent resulting from an implicit integration rule in which the current source is a history term. Switches are modeled as small inductor when closed, and small capacitor when open, which is known in the literature as the Associated Discrete Circuit (ADC) model of the switch [26]. In this work, network equations are assembled using the the Modified-Augmented Node Analysis (MANA) [24] and using backward Euler integration rule, yielding:

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{5.1}$$

where $\mathbf{A}$ is the MANA matrix, $\mathbf{x}$ is a vector containing unknown node voltages and currents entering the independent voltage sources, and $\mathbf{b}$ is a vector composed of known sources and history terms.

By choosing appropriate values for the L_{sw} and C_{sw} associated each switch, such that $C_{sw}/\Delta t = \Delta t/L_{sw}$, network equations become fixed irrespective of the switch statuses. The inverse of the invariant $\mathbf{A}$ matrix is then precomputed $\mathbf{A^{-1}}$ and its values used to build a state-space alike computing scheme, as suggested in [23]:

$$\begin{bmatrix} \mathbf{v}^{n+1} \\ \mathbf{y}^{n} \end{bmatrix} = \begin{bmatrix} \mathbf{W_{vu}} & \mathbf{W_{vx}} \\ \mathbf{W_{yu}} & \mathbf{W_{yx}} \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n} \\ \mathbf{x}^{n} \end{bmatrix}, \tag{5.2}$$

where $\mathbf{x}^n$ is a sub-vector of $\mathbf{v}^n$ that is consistent with switch statuses, $\mathbf{u}^n$ is the input vector, $\mathbf{x}^n$ is a state (history term) vector, and $\mathbf{y}^n$ is the output vector.

The switch status of an IGBT-diode pair is updated according to:

$$s^{n+1} = c^{n+1} + s^n(i_s^n \leq 0) + \overline{s^n}(v_s^n < 0) \tag{5.3}$$

where c^{n+1} is the current command at the IGBT gate, s^n, v_s^n and i_s^n are respectively the switch status, the voltage and the current associated with the switch.

The switch status of a diode is updated according to:

$$s^{n+1} = s^n(i_s^n \geq 0) + \overline{s^n}(v_s^n \geq 0) \tag{5.4}$$

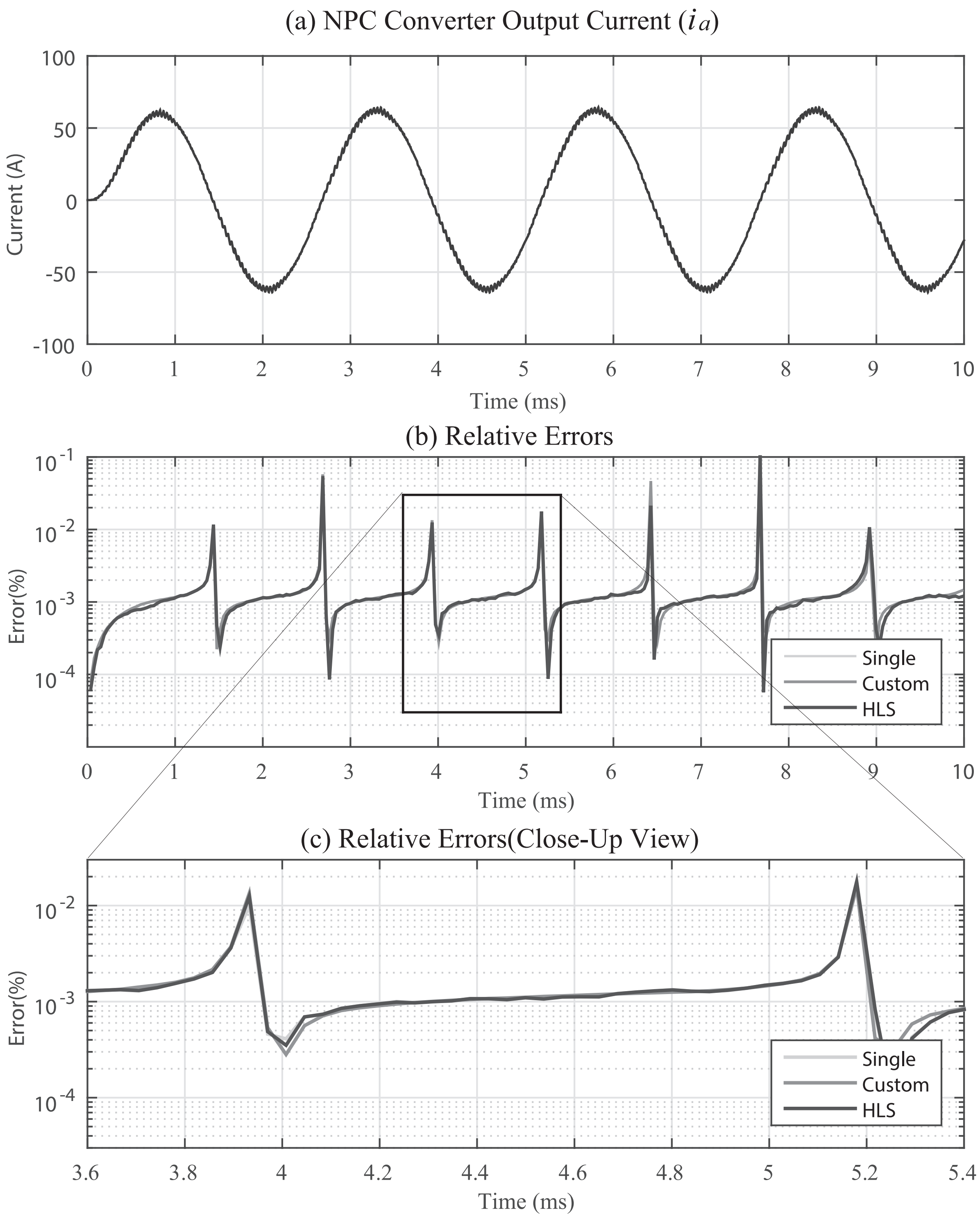


Figure 5.11: Simulation results: (a) Load current i_a; (b) Relative errors; (c) Close-up view on relative errors

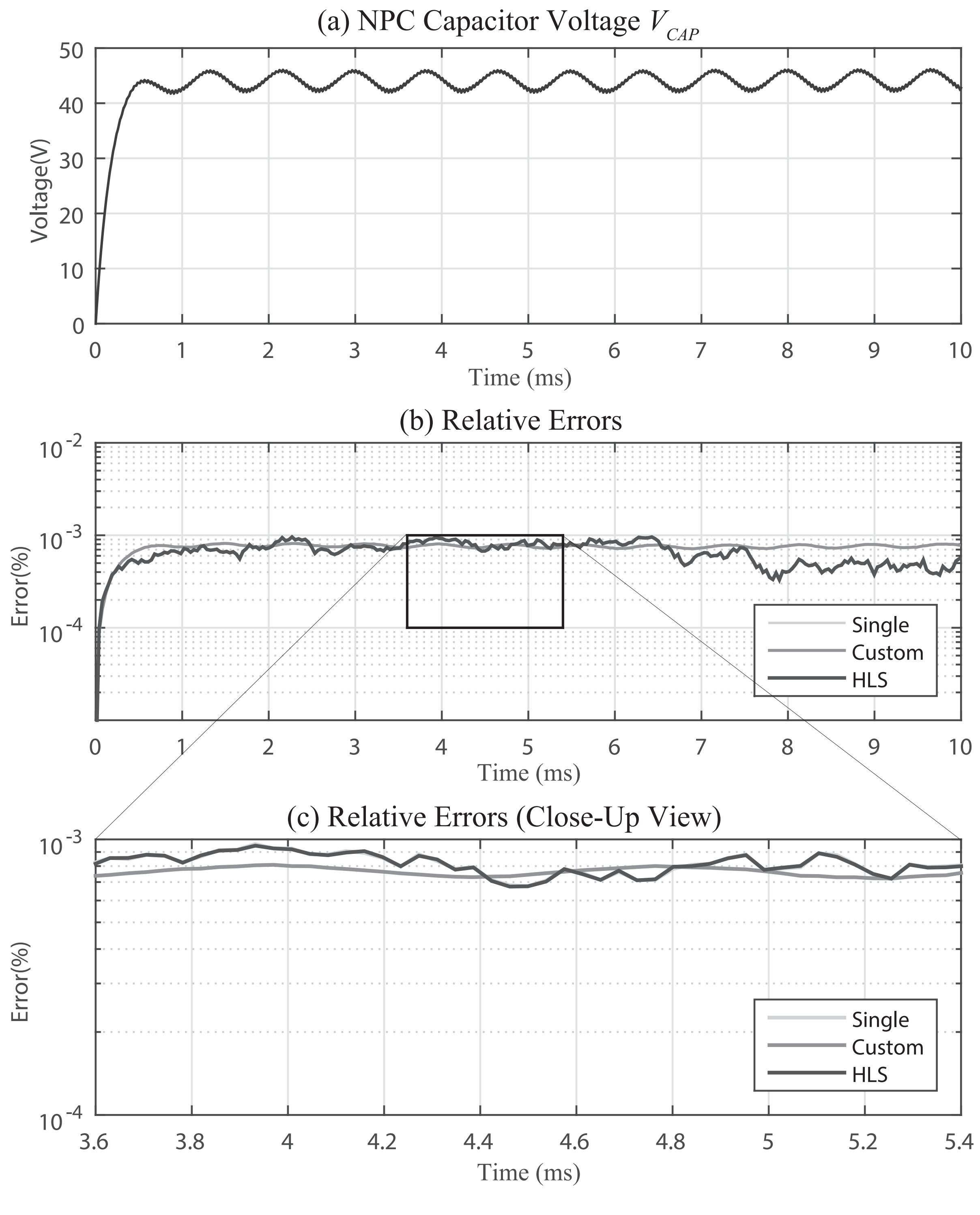


Figure 5.12: Simulation results: (a) Capacitor voltage V_{CAP}; (b) Relative errors; (c) Close-up view on relative errors

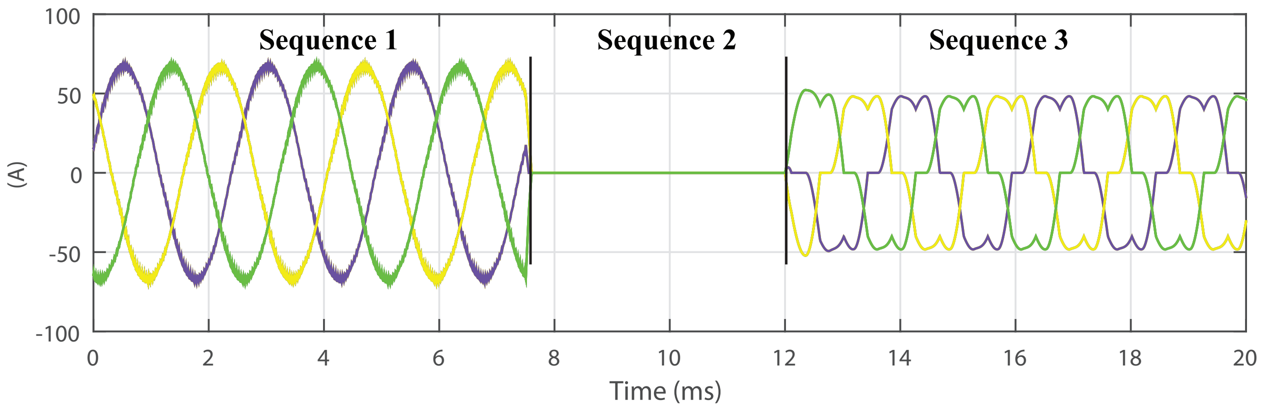
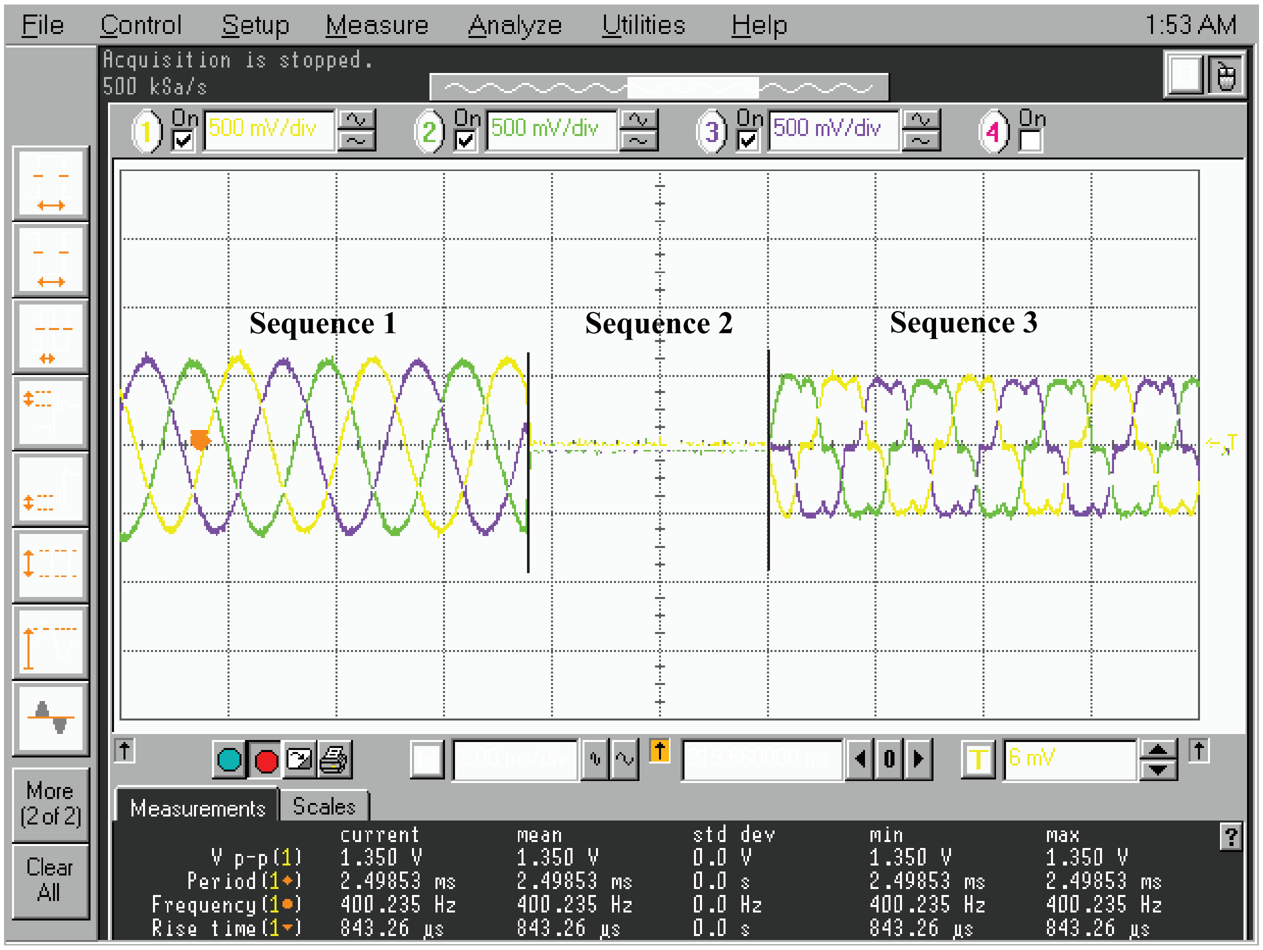


Figure 5.13: Load currents from a SPS offline simulation with 3 sequences: (1) Inversion mode; (2) Gates turned-off; (3) Rectification mode



```
Acquisition is stopped.
500 kSa/s
```

```
               current       mean          std dev    min           max
   V p-p(1)    1.350 V       1.350 V       0.0 V      1.350 V       1.350 V
  Period(1)    2.49853 ms    2.49853 ms    0.0 s      2.49853 ms    2.49853 ms
Frequency(1)   400.235 Hz    400.235 Hz    0.0 Hz     400.235 Hz    400.235 Hz
Rise time(1)   843.26 µs     843.26 µs     0.0 s      843.26 µs     843.26 µs
```

Figure 5.14: Load currents from an online simulation with 3 sequences: (1) Inversion mode; (2) Gates turned-off; (3) Rectification mode

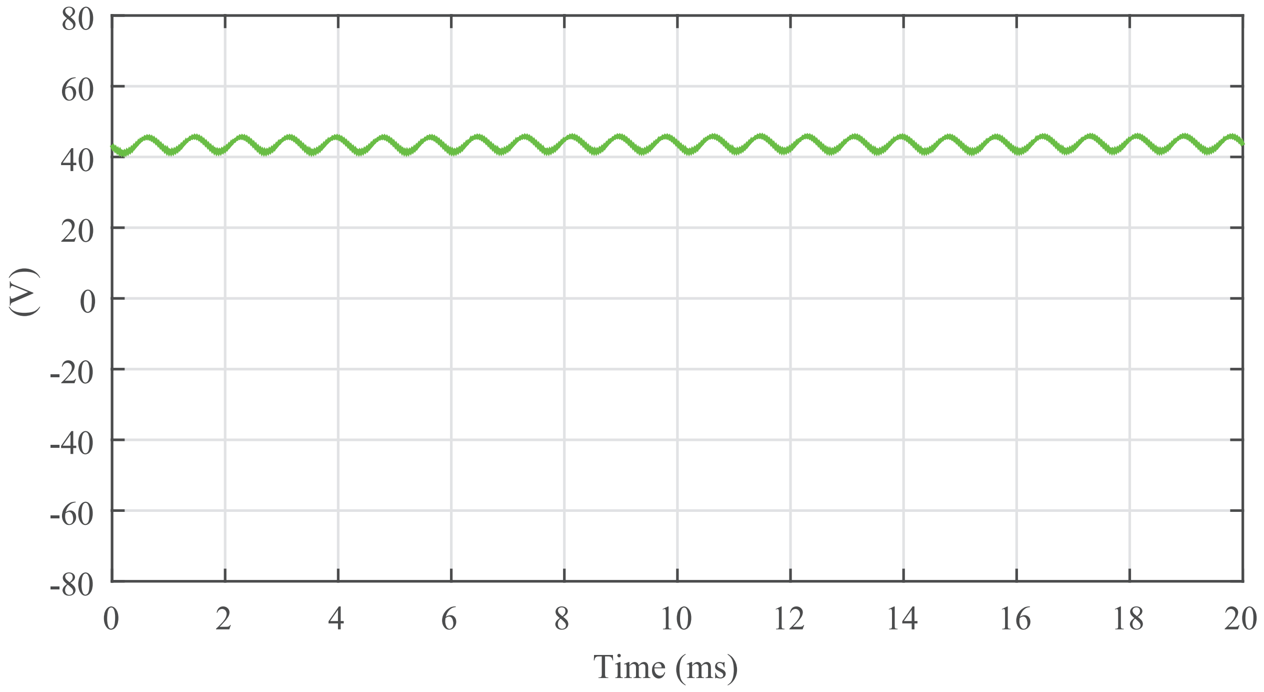
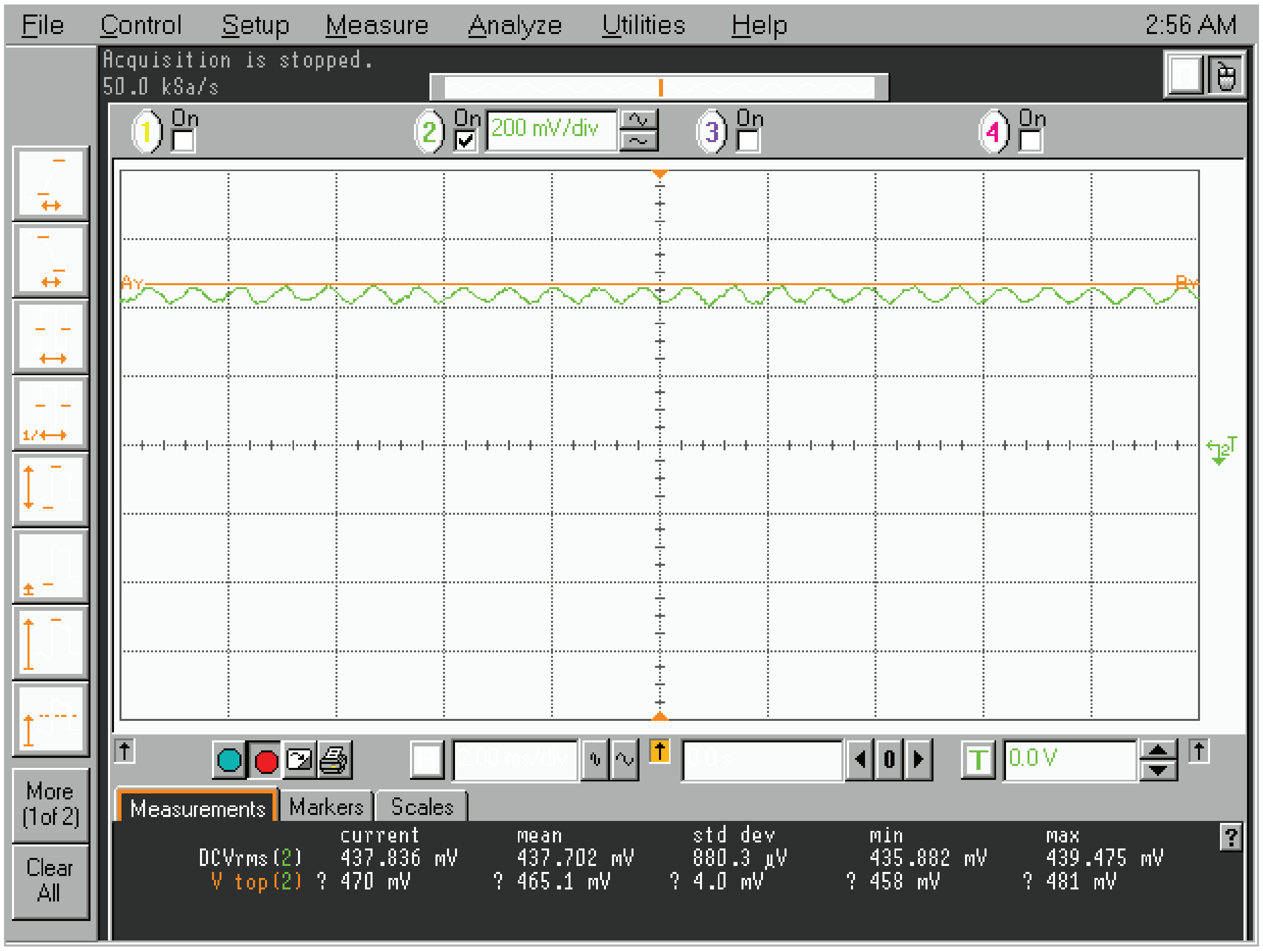


Figure 5.15: Capacitor voltage from an SPS offline simulation in inversion mode



```
Acquisition is stopped.
50.0 kSa/s
```

```
                 current        mean           std dev        min            max
     DCVrms(2)   437.836 mV     437.702 mV     880.3 µV       435.882 mV     439.475 mV
     V top(2)  ? 470 mV       ? 465.1 mV     ? 4.0 mV       ? 458 mV       ? 481 mV
```

Figure 5.16: Capacitor voltage from an online simulation in inversion mode

CHAPTER 6 ARTICLE 2: A LOW-LATENCY RECONFIGURABLE MULTISTAGE INTERCONNECTION NETWORK

Montaño, F., Ould-Bachir, T., Mahseredjian, J., & David, J. P. (2019, May). "A low-latency reconfigurable multistage interconnection network," In 2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE),(pp. 1-4).

Abstract

This paper presents a low-latency multistage interconnection network. The proposed architecture features simplicity of design, a straightforward address encoding/decoding scheme, and provides non-blocking as well as multi-casting and broadcasting capabilities. It uses the latency-insensitive design methodology, a paradigm that eases the design process while ensuring correctness of data transfers. The design is tailored to multiprocessor reconfigurable devices. Our results show that the proposed interconnection has a small footprint, a very high throughput, and that it can operate at high clock frequencies (> 500 MHz) on recent FPGAs both from Xilinx and Intel.

Keywords

Multistage interconnection network, low-latency communication, latency-insensitive design, FPGA.

6.1 Introduction

The performance of multiprocessor based systems depends heavily on the efficiency of their interconnection networks. Ideally, all the processors are linked. However, keeping a direct interconnection can be cost prohibitive as the number of processors increases. This is more accentuated when the processors lie on a single die, where constraints on size, latency and routing algorithms become challenging [121–123].

An interconnection network allows data exchange among the different nodes connected to it. A node is usually formed by a processor, a memory or a combination of the two. The way nodes are interconnected defines the topology of the network. The taxonomy for interconnection topologies is shown in Fig. 6.1: [124, 125]. Networks belong to one of two categories, static or dynamic, depending on whether the edges connecting nodes can be reconfigured or not. Static topologies consist of fixed interconnections defined at build time. Each node has a direct connection to its neighbors, forming a network of one or multiple dimensions such as stars, rings, meshes and cubes. In such networks, the links are passive and it is the node's role to route the data. In dynamic topologies, the nodes are indirectly connected through one or more links connected to a set of reconfigurable switches forming an active interconnection network (AIN).

The crossbar and the Multistage Interconnection Networks (MINs) are the two main types of AIN. In a crossbar, each input port has a direct link to all output ports. An N-node crossbar requires N^2 connections, and each node has a fan-out of N. Hence, the quadratic surface cost is the main drawback of the crossbar structure [126]. MINs solve the problem by splitting the crossbar into a succession of intermediate crossbars of smaller size [127].

The MIN is formed by a number of switches[1] grouped in stages. MINs are named based on their interconnection scheme: Baseline, Benes, Butterfly and Omega are various kinds of MINs. It has been shown that all kinds of MINs are equivalent in terms of their routing capablities [128]. Each switch has k input and k output channels that are used to connect to other switches from neighbouring stages. The number of interconnections in a MIN is given by $(N/k)log_k(N)$ [129]. When $k = 2$, a unique path exists between a given input-output port pair, which may potentially result in blocking situations. Redundancy of paths can be introduced by increasing the number of channels (k) at each cell, or by adding more stages. However, these solutions affect the performance, cost, and complexity of the control algorithms [130–132].

This paper presents an FPGA-based implementation of a generic, scalable, non-blocking MIN. The proposed MIN has multi-casting and broadcasting capabilities. It uses 2×2 switches and a simple routing algorithm. The remainder of this paper is organized as follows: Section 6.2 presents the detailed description of the architecture and its logical operation. Synthesis results of the design exploration are presented in Section 6.3. Section 6.4 concludes the paper.

6.2 MIN Architecture

6.2.1 Butterfly Topology

The butterfly topology has been selected for this work because of its simplicity. The work can be easily adapted to other topologies such as Omega, Baseline, or Benes. Fig. 6.2: presents such a topology for a 16×16 MIN. The network takes its name from the butterfly patterns formed by the interconnection links at each stage. It is formed by assembling 2×2 switches, where each switch is a crossbar unit cell that allows one-to-one and one-to-many connections.

[1]The terms "switch" and "cell" are used interchangeably in the text.

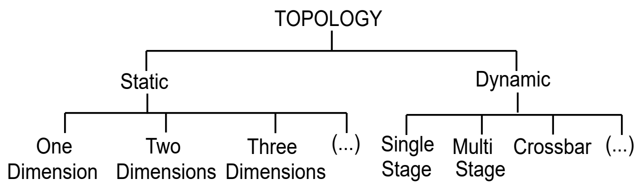


Figure 6.1: Taxonomy of interconnection networks

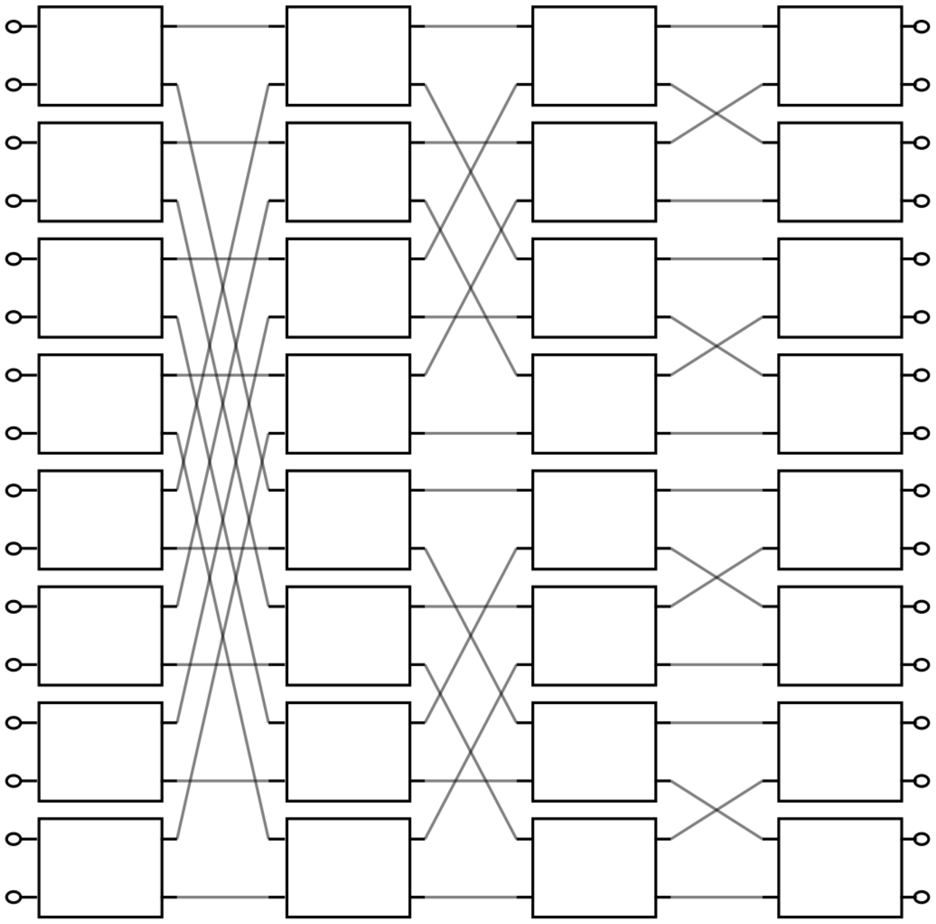


Figure 6.2: Butterfly topology: 16×16 MIN

Fig. 6.3: presents all possible combinations of the 2×2 cell. One-to-one direct and cross connections are presented in Fig. 6.3:.a, in which case data flow freely through the switch. Blocking condition occurs when incoming data on both input channels target the same output, as shown in Fig. 6.3:.b. Fig. 6.3:.c and 6.3:.d show one-to-many connections in non-blocking and blocking condition respectively.

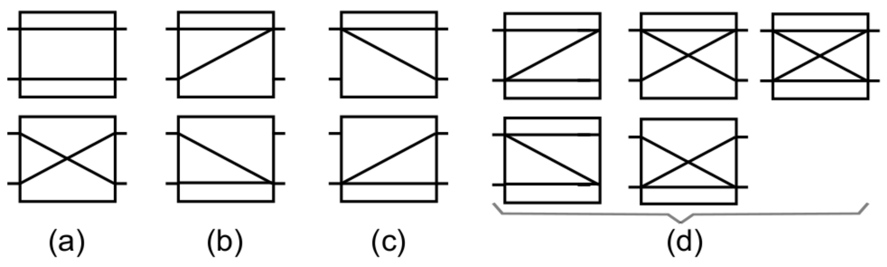


Figure 6.3: Basic switch modes and conditions: a) One-to-one non-blocking; b) One-to-one blocking; c) Multicast non-blocking; d) Multicast blocking

6.2.2 Proposed Switch Architecture

Fig. 6.4:.a presents the proposed 2×2 switch. It is comprised of two fork units at the input, four FIFOs, and two output selectors. Each fork unit receives incoming data from the previous stage and sends them to one or both (broadcast) output channels, according to routing instructions. FIFOs are used to handle congestion and to avoid blocking conditions. The output selector is responsible for sending the data to the next stage. Fig. 6.4:.b shows the one-to-one and one-to-many connections with no data contention. Fig. 6.4:.c. illustrates data contention conditions that our proposed switch solves.

The control logic is built using the Latency Insensitive Design (LID) paradigm. LID uses events-based protocols to transfer useful data throughout communication channels [94]. In the proposed switch, the data is sent if at least one of the two FIFOs is not empty and the cell from the next stage is ready to receive data. Moreover, the selector's routing policy treats the situation where two inputs target the same destination in a way that does not favor a given input over the other. For that purpose, a round robin algorithm is used to manage priorities when both FIFOs (say F_0 and F_1) contain data. Let us suppose that priority is given to F_0 and both FIFOs contain data. Then the selector will read data from F_0 and pass the priority to F_1. The priority will keep passing from one FIFO to the other as long as both have data for that output. If only one FIFO has data, data is read and the priority remains unchanged.

Hence, the proposed MIN uses a FIFO-based LID methodology. It is capable of data exchange, regardless of the latency of its data path and the arriving time of data.

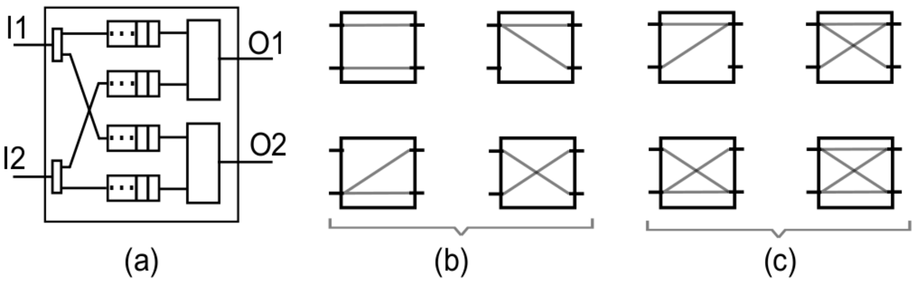


Figure 6.4: Proposed Switch: a) Internal structure; b) Transmission without data contention; c) Transmission with use of data contention

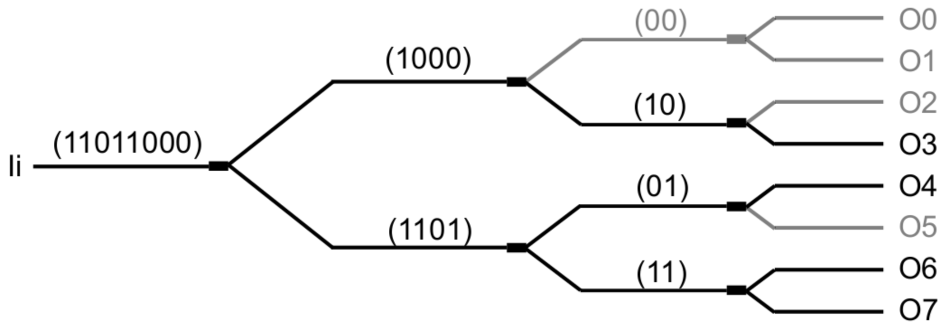


Figure 6.5: Multi-cast routing example: routing commands are given between parenthesis. Target outputs are O3, O4, O6 and O7 , i.e. 11011000

6.2.3 Routing Scheme

The routing command is formed by a string of N bits, where N is the number of output ports in the MIN. Each output port is associated with a single bit in the string such that each bit position corresponds to a port number (e.g. bit 3 represents output port 3). A bit is set in the routing command to indicate that data must reach the corresponding output port, it is reset otherwise. At each stage, the cell splits the incoming routing command into two halves (top and bottom). Each half contains the routing command for a cell from the next stage. Data is sent to the corresponding channel (top or bottom) if the corresponding routing command has at least one bit set. The process of splitting the routing command in two halves and the transfer decision repeat at each stage.

The example of Fig. 6.5: depicts the routing of an incoming packet over an 8×8 MIN. We assume a routing command equal to 11011000, that is output ports 3, 4, 6, and 7 are targeted. Fig. 6.5: illustrates the process of splitting the routing command in two halves and sending data to the next stage. If all bits in a routing command are reset, the process stops for that branch. At the final stage, targeted outputs are reached as expected.

6.2.4 MIN Capacity and Data Contention

We define the capacity of a MIN as the maximum amount of data that the network can hold. The capacity is a function of the number of ports (N) and the FIFO depth (D). The maximal capacity of the MIN is given by $C_{max} = 2 \times N \times log_2(N) \times D$. A MIN with a higher capacity supports more data contention and can receive data even if one of the outputs is momentarily stalled. Data contention occurs when two incoming data at a given cell target the same output channel.

Fig. 6.6: shows the data flow and the effect of data contention on all stages for an 8×8 MIN considering N-to-one transmissions, with destination port 0 stalled. Values indicate originating ports. Superscripts indicate arrival time. Fig 6.6:.a shows the contention at Stage 0 and data at Stage 1 after two passes. In presence of contention, the round robin algorithm is run. Fig. 6.6:.b shows the state of the MIN after 4 passes. Using FIFOs of depth $D = 2$, a total of 28 data are held by the MIN.

6.2.5 Generic Structure

The MIN is implemented using a generic and portable VHDL code, at a Register Transaction Level (RTL) design abstraction. The butterfly topology is generated by assembling so-called blocks, as shown in Fig. 6.7:.a. A block is a collection of switches. As shown in Fig. 6.7:.b, stage i contains 2^i blocks, and each of these blocks are comprised of $N/2^{i+1}$ switches.

At build time, an algorithm set the connections between the outputs of the switches in a given block and the outputs of that block. The algorithm is illustrated in Fig. 7.c. Let us assume that the block contains S switches and $2S$ output ports. Let us label the switches from 0 to $S-1$ from top to bottom and label the output port of the block from 0 to $2S-1$, from top to bottom as well. Then the switches with label s_t ($0 \leq s_t < S/2$) have their output 0 (top) connected to the output port $2s_t$, and their output 1 (bottom) connected to the output port $2s_t + S$, whereas the switches with label s_b ($S/2 \leq s_b < S$) have their output 0 connected to the output port $2s_b + 1 - S$ and their output 1 connected to the output port $2s_b + 1$.

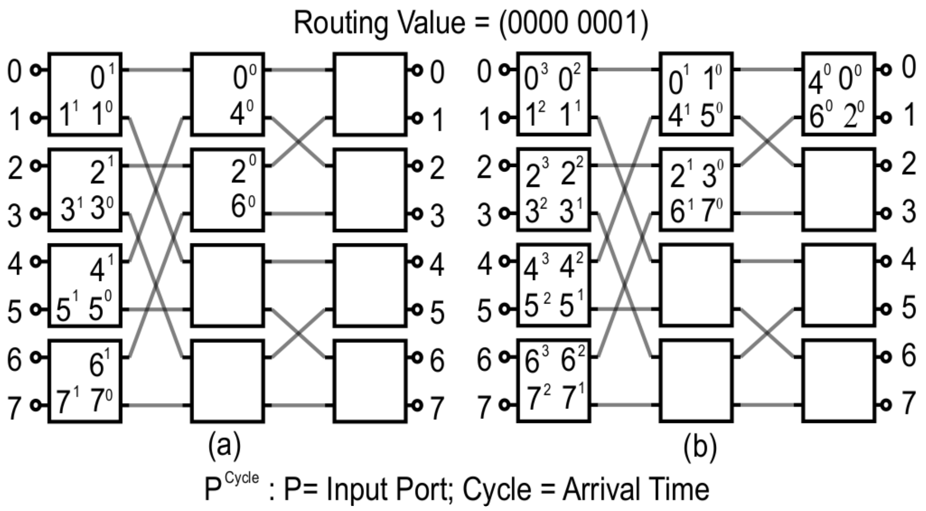

Routing Value = (0000 0001)
(a)
(b)
P^Cycle : P= Input Port; Cycle = Arrival Time

Figure 6.6: Data flow in N-to-one transmission targeting output port 0: a) Status at cycle 2; b) Status at cycle 4

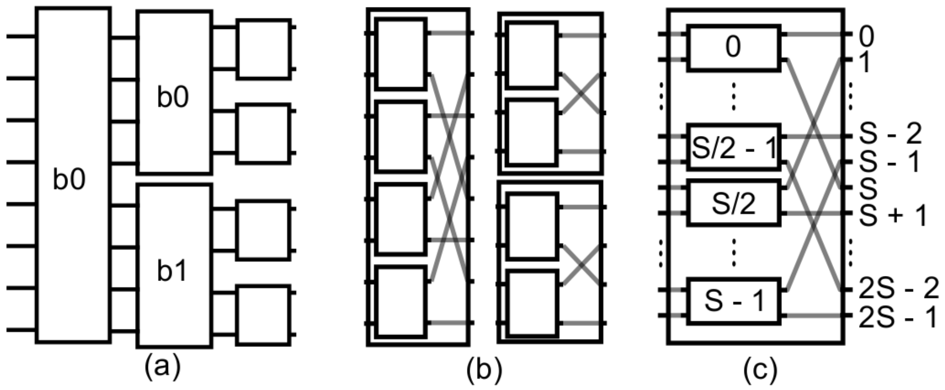


Figure 6.7: Block representation: a) Inter-block connection; b) Output cross connections; c) Switch-Block output interconnection

6.3 Implementation Results

This section presents synthesis results targeting FPGAs Xilinx Virtex-7 and Intel Arria 10 of similar size. The objective is to assess the footprint and performance of the proposed architecture for different MIN parameters. The portability of the design allows us to synthesize the MIN on different platforms with no modification to the code, except for the MIN parameters. For Xilinx, Vivado 2015.3 is used, whereas Quartus Prime 15.1 is used for Intel.

The parameters modified during the exploration include: *i*) N: the number of input and output ports; *ii*) m: the data width at the input ports; and *iii*) D: the depth of the FIFOs.

The baseline configuration is an 8×8 MIN with 32-bit input ports, and 2-word FIFOs. During design exploration, one parameter is changed at a time and the remaining parameters are taken from the baseline configuration. Prior to the architectural exploration, a functional verification of the proposed MIN using the baseline parameters was performed using Xilinx ISIM and Modelsim-Altera in N-to-one and broadcast modes.

Table 6.1: shows synthesis results from design explorations conducted by varying parameters N, m and D of the baseline configuration for Virtex and Arria FPGAs respectively. The parameter N has been varied from 8 to 64, m from 32 to 512, and D from 2 to 64. For each configuration, the table gives the amount of reconfigurable logic used in terms of number of LUTs/ALMs (Xilinx/Intel) and number of registers, as well as the maximum operating frequency.

Clearly N is the parameter that has the most effect on the design cost of the MIN. Results show that the resource consumption grows as $O(N\log(N))$ with the number of ports, which is consistent with the theory. The $O(N\log(N))$ growth rate is confirmed by Fig. 6.8:.a (Xilinx) and Fig. 6.8:.b (Intel) for both LUTs/ALMs and registers. Nevertheless, the proposed design has a very acceptable footprint since a 64×64 MIN (the largest MIN considered here) occupies less than 25% of the available resources. Area occupation grows with m at a slightly sub-linear rate — from $O(m^{0.80})$ to $O(m^{0.93})$ — which is mainly attributable to resource sharing. Finally, we observe an area occupation that is almost constant for various FIFO depths. This observation can be explained by the fact that modern FPGAs have dedicated fabric for small FIFOs. Hence, increasing the depth of the FIFOs comes at almost no cost (up to a certain depth limit), while enhancing MIN's capacity by many folds. It is also worth mentioning that the design is optimally pipelined, thus making it capable of sustaining > 500 MHz clock frequencies on both technologies, Xilinx and Intel. This is beneficial to complex designs because it ensures that the critical paths lies in peripheral modules and remains outside the communication network.

6.4 Conclusion

This paper proposed a latency insensitive design for MINs. The architecture offers multi-casting, broadcasting and data contention capabilities while using a straightforward routing scheme. The portability, compactness and optimal pipelining of the design allows its operation at high frequency (> 500 MHz) at a reasonable cost on both Xilinx and Intel FPGAs. Moreover, the design exploration conducted in this work determined that the main factor affecting the design cost was the number of ports (N), which grows at a rate of $O(N\log(N))$, a result consistent with the theory. The paper also showed that the capacity of the MIN

Table 6.1: Design Exploration

Number of Ports (N)										
	Virtex-7					Arria 10				
Nb. Ports	LUTs	%	Registers	%	F$_{max}$ (MHz)	ALMs	%	Registers	%	F$_{max}$ (MHz)
8	5,028	1.2	4,687	0.5	625	4,260	1.0	6,743	0.4	522
16	13,613	3.1	12,642	1.5	625	12,774	3.0	19,498	1.1	522
32	35,143	8.1	28,040	3.2	625	30,725	7.2	45,258	2.6	522
64	94,944	21.9	68,552	7.9	625	77,788	18.2	107,792	6.3	524
Port Width (m)										
Port Width	LUTs	%	Registers	%	F$_{max}$ (MHz)	ALMs	%	Registers	%	F$_{max}$ (Mhz)
32	5,028	1.2	4,687	0.5	625	4,260	1.0	6,743	0.4	522
64	7,396	1.7	5,967	0.7	625	6,264	1.5	10,071	0.6	522
128	11,662	2.7	8,527	1.0	625	12,932	3.0	16,729	1.0	521
256	20,734	4.8	13,727	1.6	625	23,595	5.5	30,038	1.8	522
512	39,010	9.0	24,052	2.8	625	46,085	10.8	56,668	3.3	523
FIFO Depth (D)										
FIFO Depth	LUTs	%	Registers	%	F$_{max}$ (MHz)	ALMs	%	Registers	%	F$_{max}$ (MHz)
2	5,028	1.2	4,687	0.5	625	4,260	1.0	6,743	0.4	521
4	5,148	1.2	4,831	0.6	625	4,118	1.0	6,925	0.4	520
8	5,239	1.2	4,975	0.6	625	4,319	1.0	7,117	0.4	521
16	5,483	1.2	5,119	0.6	625	4,304	1.0	7,312	0.4	521
32	5,691	1.3	5,263	0.6	625	4,626	1.1	7,501	0.4	518
64	7,227	1.7	5,359	0.6	625	5,047	1.2	7,695	0.5	518

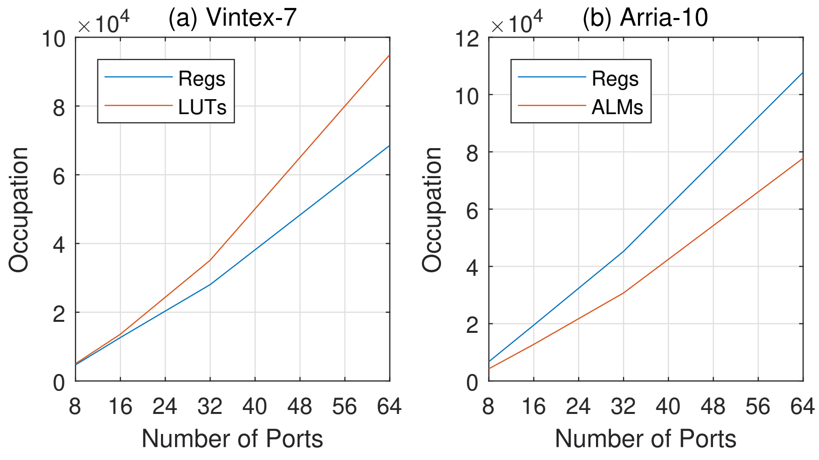


Figure 6.8: Resource consumption as function of number of ports (N)

can be substantially enhanced at almost no cost by deepening FIFOs, which improves its tolerance to stall situations.

CHAPTER 7 ARTICLE 3 : A LATENCY-INSENSITIVE DESIGN APPROACH TO PROGRAMMABLE FPGA-BASED REAL-TIME SIMULATORS

Montaño, F., Ould-Bachir, T., & David, J. P. (2020). "A Latency-Insensitive Design Approach to Programmable FPGA-Based Real-Time Simulators," MDPI-Electronics, 9(11), 1838.

Abstract

This paper presents a methodology for the design of programmable FPGA-based real-time simulators (RTSs) for power electronic circuits (PECs). The programmability of the simulator results from the use of an efficient and scalable Overlay Architecture (OA). The proposed OA relies on a Latency-Insensitive Design (LID) paradigm. LID consists in connecting small processing units that automatically synchronize and exchange data when appropriate. The use of such data-driven architecture aims to ease the design process while achieving a higher computational efficiency. The benefits of the proposed approach is evaluated by assessing the performance of the proposed solver in the simulation of a two-stage ac-ac power converter. The minimum achievable time-step and FPGA resource consumption for a wide range of power converter sizes is also evaluated. The proposed overlays are parameterizable in size, cost effective, provide sub-microsecond time-steps, and offer a high computational performance, with a reported peak performance of 300 GFLOPS.

Keywords

Real-time simulation; HIL; power electronic circuits; FPGA; latency-insensitive design; data-driven architectures

7.1 Introduction

Hardware-in-the-Loop (HIL) simulation is an industrial methodology used in the development of power systems to reduce risks and costs [133, 134]. In a HIL setup, a real controller is connected to a simulated plant in closed loop. Better performances are achieved when the round trip time of the signals is as short as possible, ideally few μs or less. A HIL simulation is typically performed using PC clusters, in which case the simulator offers flexibility and programmability [15]. However, when the power system requirements impose a latency limited or close to 1 μs, Field Programmable Gate Arrays (FPGAs) are usually the best choice for HIL [16].

Recent publications address the FPGA-based real-time simulation problem from an application, system or circuit modeling point of view [10, 52, 135]. Their purpose is either to reduce the simulation time-step, to propose new switch modelling techniques or to augment the parallelism of the simulation. The approach adopted in the paper addresses the subject from a hardware implementation perspective. Our aim is to propose a design methodology that will help reduce development time of an FPGA-based RTS — also designated as a Hardware Solver (HS) in this paper.

The implementation of a HS is typically described at Register Transfer Level (RTL) [109]. The utilization of High-Level Synthesis (HLS) tools has been considered as an alternative to facilitate hardware development on FPGA [11, 81, 110], but has been shown to impose certain limitations on performance and resources utilization [11]. Furthermore, the HLS design approach does not remove the synthesis process stage, which takes from minutes to hours, depending on the complexity of the design.

The main contribution of this paper is to propose a novel design methodology for the development of a HS. The methodology uses an Overlay Architecture (OA) based on a decentralized control scheme using a Latency-Insensitive Design (LID) approach. An OA is a configurable and regular hardware structure composed of functional and interconnection elements. An OA is configurable at two levels: *i*) at the architectural level, to modify the number and characteristics of operators and generate a new bitstream; *ii*) at the software level, which is done by filling the content of embedded memories. On the other hand, LID facilitates the design integration task by allowing data transfers between units on the datapath to be self-synchronized. In the proposed LID approach, processes are synchronous systems driven by the events at their point to point communication channels following a specific protocol. This ensures the correct functioning of processes independently of the latency on the channels that interconnect them. Modules designed with this method are easily integrated to reach the correctness of the system [93].

The remainder of this paper is organized as follows: Section 7.2 presents the works related to this research, that is FPGA-based real-time simulation of power converter techniques referenced in the literature. The purpose of Section 7.3 is to provide the reader with a review of the background material pertaining to overlay architectures and latency-insensitive design. Section 7.4 presents the proposed solution. It reviews the mathematical foundation of our approach, the programming model of the proposed Hardware Solver as well as its building blocks. Section 7.5 presents the simulation and implementation results for the case studies as well as the performance exploration. A discussion is given in Section 7.6.

7.2 Related Work

7.2.1 Switch Model

The design of a HS must obey tight constraints due to the high switching frequencies of semiconductors, which typically require simulation time-steps below 1 μs [6, 11]. To achieve a low time-step solution, the switching behavior is either emulated using an ideal switch model or a switching function [136]. The ideal switch model in turn resorts to one the two commonly used switch models: the resistive switch model (RSM) or the Associated Discrete Circuit (ADC) switch model.

The RSM replaces the switch by a low and high resistor when on and off respectively. It is known to be a more accurate approach than ADC but suffers from certain limitations such as heavy memory requirements for storing precomputed network equations, and higher computational cost due to iterations. The Sherman-Morrison-Woodbury formula to compute the updated inverse matrix on the fly was proposed by Hadizadeh (2010) [137], but such an approach neglects the iterations needed to determine the state of diodes. A method to handle diodes has been proposed by Blanchette (2012) [42], but it introduces unrealistic parasitic elements that alter the behavior of the converter. The use of iterations, the compensation method and circuit partitioning have also been considered [48, 50], but it results in large time-steps. A predictor-corrector algorithm has been used by Liu (2020) [10] to decouple the switches from the circuit elements and to simulate them simultaneously, but such an approach relies on forward integration scheme and may become unstable. Finally, a direct map method was proposed by Chalangar (2020) [52] to simulate a high switching frequency resonant converters, but it is unclear at the moment how the method could apply to more complex topologies.

The ADC switch model [26] replaces the switch by a Norton equivalent with a fixed impedance. It gives a fixed system of equations irrespective of switch statuses, which considerably reduces memory requirements. Because it does not require an iterative solution, the ADC model is less computationally hungry than RSM, and has been widely used for HS implementations on FPGA [6, 53, 138]. However, ADC is prone to fictitious oscillations that can alter the behavior of the converter's model. Various techniques have been proposed for minimizing the effect of these undesired oscillations [139–141]. These techniques often require special routing mechanism in the HS to handle the various conditions the modified algorithm has to account for.

The switching function is a simple model, which replaces the converter by a circuit only consisting of controlled voltage and current sources. Hence, the switching function describes the behavior of the converter through its interfaces. However, by doing so, it fails to accurately emulate transients or certain operational modes of the converter (the rectification for an inverter for instance). The switching function was utilized in works where transients were not the primary focus of the investigation [135, 142, 143].

7.2.2 Hardware Description Languages

The implementation of a HS is mainly obtained using hardware description languages such as Verilog or VHDL at RTL level. Describing a digital design at the RTL abstraction level is known to be a difficult and error prone task. Increasing the level of abstraction for hardware design is supposed to reduce the development time and increase productivity [76], which is why the utilization of High-Level Synthesis (HLS) tools has been considered as an alternative to facilitate hardware development on FPGA [11, 81, 110].

HLS becomes a popular trend for the implementation of hardware accelerators in areas such as image processing, economics, robotics etc. [79, 80, 144]. More recently HLS has also been used for custom implementations of HS for HIL [143, 145] and for its testbenches. The use of HLS for HIL has been shown to impose limitations on hardware performance and resources utilization when intended to be used in HS for HIL [11]. Furthermore, once an HLS code is finished, the development cycle is not completed: the RTL code needs to be generated, the code synthesised and implemented, then tested on the FPGA for timing, accuracy, etc. All these tasks can take from hours to days, depending on the complexity of the design.

7.2.3 Number Format

An FPGA implements real arithmetic computations using either fixed-point or floating-point (FP) format. The fixed-point number is basically an integer scaled by an implicit factor. Fixed point is known to uses less hardware resources, which yields a low latency datapath. However, the fixed-point format suffers from a restricted dynamic range that can restrict its utilization for HIL applications. These limitations have been addressed by means of appropriate optimizations such as scaling the operands to the dimensions of the hardened blocks of the FPGA, or by normalizing all the physical quantities to a per unit scale [42, 52, 146].

On the other hand, the FP number format allows for a larger dynamic range, but its operators are costly in terms of hardware, and require deeper pipelines. This is even more true when double precision is considered. Hence, most HSs reported in the literature make use of single precision FP to save hardware and to reduce latency [50]. The utilization of a non-standard

FP is often privileged in the FPGA context, that is a FP which does not adhere to IEEE-754 standard to provide better resource consumption. For instance, in [147], the authors proposed a low-cost FP with soft normalization and showed the benefits to HIL applications.

The solution considered in this paper is based on a non-standard FP format with intermediate precision, which balances resource consumption, computing power and computational accuracy. The arithmetic operators are built using a fused-datapath approach [148], which consists in removing all intermediate packing and unpacking stages from a complex FP datapath and performing all the computations jointly. All internal additions are performed using the Self-Alignment Format (SAF) [118]. Such an approach was previously adopted in [17, 48, 138].

7.3 Background

Improving the computing performance to reach lower time-steps while keeping a reasonable FPGA resource consumption increases the difficulty associated with the development of a HS. Most of the research devoted to the development of HSs is performed at an application level, where the main energy is invested on the elaboration of new modeling techniques, circuit partitioning schemes, and parallelizing calculations.

Generally, the implementation of a HS follows a centralized design approach [6, 40, 54]. It consists of a datapath, comprised of memory elements and functional units (FU), under the command of a control unit that sequences all data transfers. The sequence of execution determines the timing of data transfers and needs to be designed by taking into account the sequence of reads and writes and the latency of each processing unit.

In this work, we propose a parametrizable and reconfigurable architecture inspired from OAs that uses a modularized LID approach. All data transfers are handled by an efficient simple interconnection scheme that facilitates programmability and system expansion. This section presents an overview of OA and LID to clarify these concepts.

7.3.1 Overlay Architectures

Custom design using RTL is a good way to achieve high performance due to the fine granularity of the hardware description and the direct access to FPGA resources it allows. However the difficulty behind such a low level design is the long time associated with FPGA development cycles. Overlay architectures have been introduced by such initiatives as ZUMA [84], Quku [85], and DeCO [9], with the aim to increase the abstraction level, improve the productivity, ease the programmability, and to reduce design time [86].

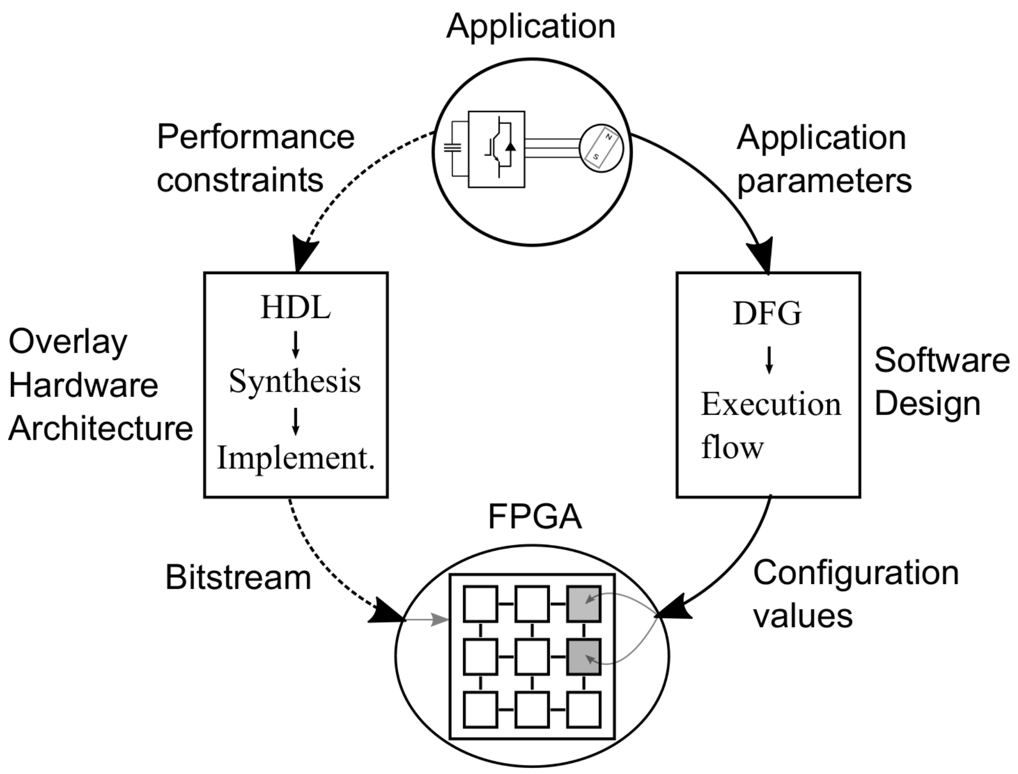


Figure 7.1: Overlay development cycle (Inspired from [4])

An OA is a reconfigurable implementation over a FPGA. An OA is formed by two main parts: an implementation (hardware architecture) and a mapping tool. Fig. 7.1: shows a condensed workflow for OA-based design. For the implementation part (left of Fig. 7.1:), the type of applications determines the OA topology. Performance requirements are then taken into account to define a set of parameters to generate an instance of the OA and to generate the FPGA programming bitstream using the traditional FPGA workflow. Once the hardware is generated, a mapping tool (right of Fig. 7.1:) is used to interpret the application code and to extract its Data Flow Graph (DFG). This way, the application is rapidly mapped on the hardware without passing trough the synthesis process again.

The performance delivered by an OA depends on the selected topology. In the two dimensional array topologies shown in Fig. 7.2:.a and 7.2:.b (nearest neighbor and torus, respectively) each tile is connected to four surrounding tiles (top, bottom, right and left) in a bidirectional way. This results in a high connectivity among units and offers wide configuration possibilities. However, this also implies that data have multiples paths to reach a desired destination, putting extra effort on the control and its synchronization, while also increasing the design complexity and cost due to the overhead related to the interconnection.

In the linear topology shown of Fig. 7.2:.c, the tiles are arranged in stages, and data can only by transmitted from one stage to the following one in a unidirectional way. This topology presents a reduced connectivity which leads to a potentially reduced capability of reconfiguration and reduce the variety of applications but gains in simplicity and in performance.

7.3.2 Latency-Insensitive Design

Timing performance of a large digital design is determined by the delay of its longest combinational path. This is often obtained using pipelining techniques, which is done by inserting register stages withing the datapath. However, the pipeline depth must be matched on all paths and affects of the control sequence. The purpose of a LID approach is to solve this issue by making functional modules independent of the arrival time of their inputs [92]. In LID, the interconnection elements are separate from the functional elements, and correct operation of the system is warranted by construction [93]. Such an approach has the merit to ease the design process by allowing the hardware designer to invest more effort on the design of small processing blocks while their interaction is taken into account at a later stage through interface communication protocols.

A basic functional module is termed a *pearl* in the literature. The *pearl* is supposed stallable, and is encapsulated in a so-called *shell*. As shown in Fig. 7.3:.a, the *shell* is comprised of memory elements, and a control logic. The memory elements are connected to the input channels and help to sustain data contention. The control logic serves the purpose of managing channel's protocol, and serving the module's functionality [94]. All data transfers are then performed only when the *pearl* is available to process them. All data signals are accompanied by control signals for handshake purposes. Two control signals in opposite direction are typically used: A `valid` forward signal to indicate that the source has valid data to send; and a `ready` backward signal to indicate that the target is ready to accept the incoming data. Fig. 7.3:.b illustrates a LID-based system with multiple shells interconnected.

7.4 Proposed Solution

This section presents the proposed solution in a detailed fashion. We start with the mathematical formulation of the power converter model — the ADC switch for modeling the power converter. We then discuss the latency-insensitive design of a HS. The LID HS is an OA, in the sense that it respects the principle of having a fixed, yet parametrized architecture, over which an application is deployed. Interconnection between functional units is implemented using a crossbar that connects all FUs together. The crossbar is implemented using a LID.

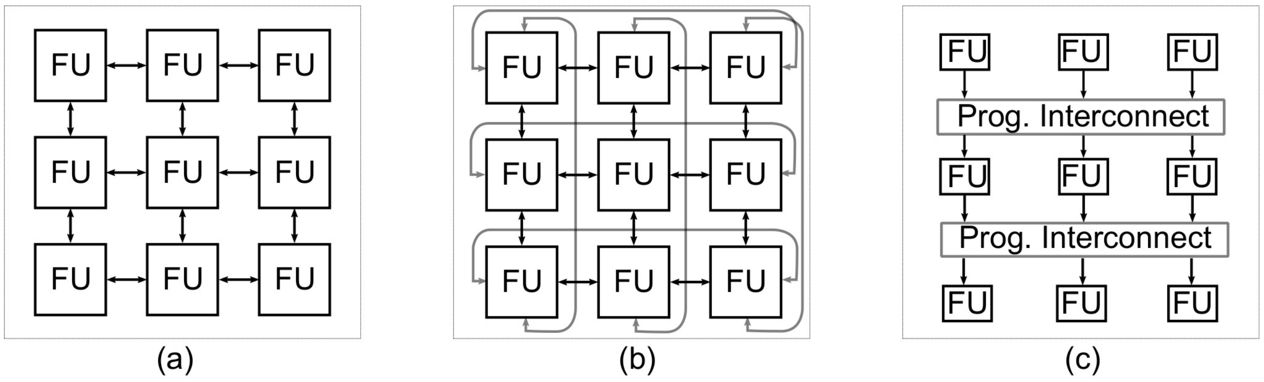


Figure 7.2: Basic topologies of overlay architectures: a) Nearest neighbor [4], b) Torus [8], c) Linear [9]

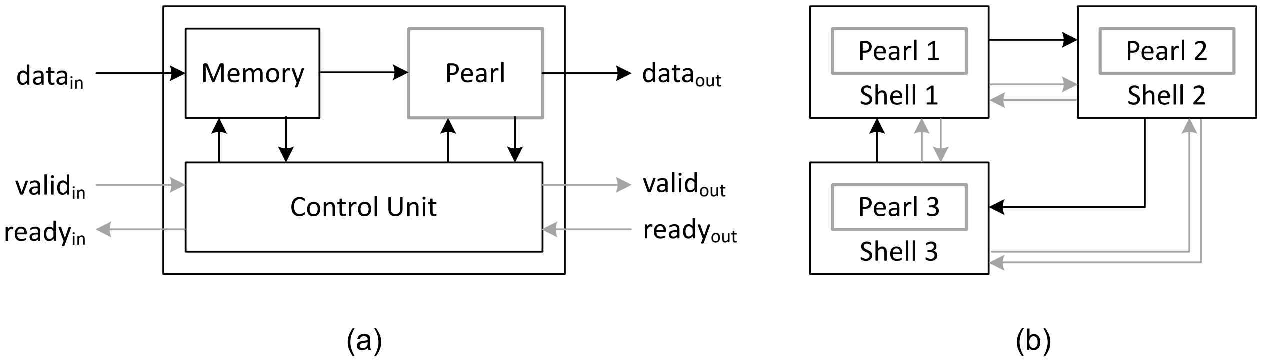


Figure 7.3: (a) General architecture of a shell; (b) LID-based design with interconnected shells

The arithmetic operators use a custom floating point number representation to achieve high performance.

7.4.1 Mathematical Formulation

Using the associate discrete circuit modeling approach, all components are replaced by Norton equivalents resulting from a fixed time-step backward Euler integration rule. The network equations can then be assembled to yield:

$$\mathbf{A}\mathbf{x}_n = \mathbf{b}_n, \tag{7.1}$$

where subscript n denotes present time point, $\mathbf{A}$ is the nodal matrix obtained from the Modified-Augmented Nodal Analysis (MANA) [24], $\mathbf{x}_n$ is a composite vector of unknowns node voltages and voltage source currents, and $\mathbf{b}_n$ is a vector of known input sources ($\mathbf{u}_n$) and history terms. We distinguish between switch history terms ($\mathbf{i}_n^s$) and L/C component history terms ($\mathbf{i}_n^h$). $\mathbf{b}$ being simply a linear combinations of those terms, it can be expressed as:

$$\mathbf{b} = \mathbf{K} \begin{bmatrix} \mathbf{u}_n \\ \mathbf{i}_n^s \\ \mathbf{i}_n^h \end{bmatrix}. \tag{7.2}$$

Using the ADC switch model, the matrix $\mathbf{A}$ is fixed, irrespective of switch statuses, and its inverse can be precomputed. Moreover, to improve the computational efficiency, rewriting the network equations in a state-space like form was found to be convenient [11,23,52,138], which yields:

$$\begin{bmatrix} \mathbf{y}_n \\ \mathbf{i}_{n+1}^{\hat{s}} \\ \mathbf{i}_{n+1}^h \end{bmatrix} = \mathbf{H} \begin{bmatrix} \mathbf{u}_n \\ \mathbf{i}_n^s \\ \mathbf{i}_n^h \end{bmatrix}, \tag{7.3}$$

where $\mathbf{H}$ is a fixed matrix obtained from algebraic manipulations of $\mathbf{A}^{-1}$, and $\mathbf{K}$. The terms with subscript $n+1$ are history terms computed for the next time-point of the simulation, and $\mathbf{y}_n$ are outputs of interest.

It is worth mentioning that $\mathbf{i}^{\hat{s}}$ comprises the switch history terms for switches in ON and OFF states (both are computed), whereas $\mathbf{i}^s$ includes only one of the two terms for each switch, with respect to actual switch status ($\mathbf{i}^{\hat{s}}$ is hence twice as large as $\mathbf{i}^s$). This matter has been thoroughly discussed in [11,23,138]. It is also worth mentioning that history terms are referred to as state variables in the remainder of this text.

7.4.2 Hardware Solver Architecture

Fig. 7.4:.a presents a general structure of a HS architecture. It consists of a linear solver connected in closed loop to a switch update module. The Linear solver feeds vectors $\mathbf{u}_n$, $\mathbf{i}_n^s$, $\mathbf{i}_n^h$ to an MVM module, which in turns produces vectors $\mathbf{y}_n$, $\mathbf{i}_{n+1}^{\hat{s}}$, $\mathbf{i}_{n+1}^h$. The linear solver sends those results to appropriate locations. The switch update module receives $\mathbf{i}_{n+1}^{\hat{s}}$ and extracts the switch history terms $\mathbf{i}_{n+1}^s$, according to the input gates and switch logic [11,23,138].

Fig. 7.4:.b illustrates how the MVM unit is built. It consists of N Dot-Product (DP) units, each DP unit consisting of m parallel multipliers (termed DP-type m) feeding an addition tree, and terminated by an accumulation function, as shown in Fig. 7.4:.c. The DP operator is designed to process $K \times m$ pairs of operands $(x_i^{(k)}, y_i^{(k)})$, $i = 1..m$, $k = 1..K$ in K successive clock cycles, and produces:

$$\sum_{k=1}^{K} \sum_{i=1}^{m} x_i^{(k)} y_i^{(k)} \tag{7.4}$$

7.4.3 Programming Model

Fig. 7.5: presents a more detailed view of the linear solver architecture. It is composed of three stages: *i)* A crossbar (XBAR); *ii)* Memory elements; and *iii)* N DPs. It is worth mentioning that, compared to similar works from the literature, this HS does not contain a global control unit, and all operations are self synchronised thanks to the adopted LID approach. The XBAR is in charge of routing data to appropriate locations. The memory elements are embedded RAM blocks (MTX and VECT in Fig. 7.5:) with read and write logic. The DPs form the functional part of the linear solver. The XBAR is presented in Section 7.4.4, the memory elements in Section 7.4.5, and the DP operators in Section 7.4.6.

The computing power of the linear solver is determined by the number of DP (N) and the number of multipliers each DP has (m), also referred to as DP-type in this paper. Hence, the OA is determined by parameters N and m. Once the overlay is generated, the application can be reprogrammed on top of it. This is done by mapping Eq. (7.3) over the OA, which is solely determined by the number of independent sources, the number of states variables, the number of switches, and the number of outputs. These parameters are used to generate the programming bits of code used to configure the XBAR, the Writer and Reader, and to fill the contents of the VECT and MTX memories.

7.4.4 Interconnection Network

The role of the interconnection network is to move data to the appropriate VECT memory locations from either the inputs of the linear solver, or the outputs of the DPs, as shown in Fig. 7.5:. The data should be able to move freely from any input to any output of the interconnection network, which motivates the use of a crossbar. However, an N-node crossbar requires N^2 connections, and each node has a fan-out of N [126]. Multistage Interconnection Networks (MINs) solve this problem by splitting the crossbar into a succession of intermediate crossbars of smaller size [127]. The MIN considered in this paper is formed by 2×2 cells

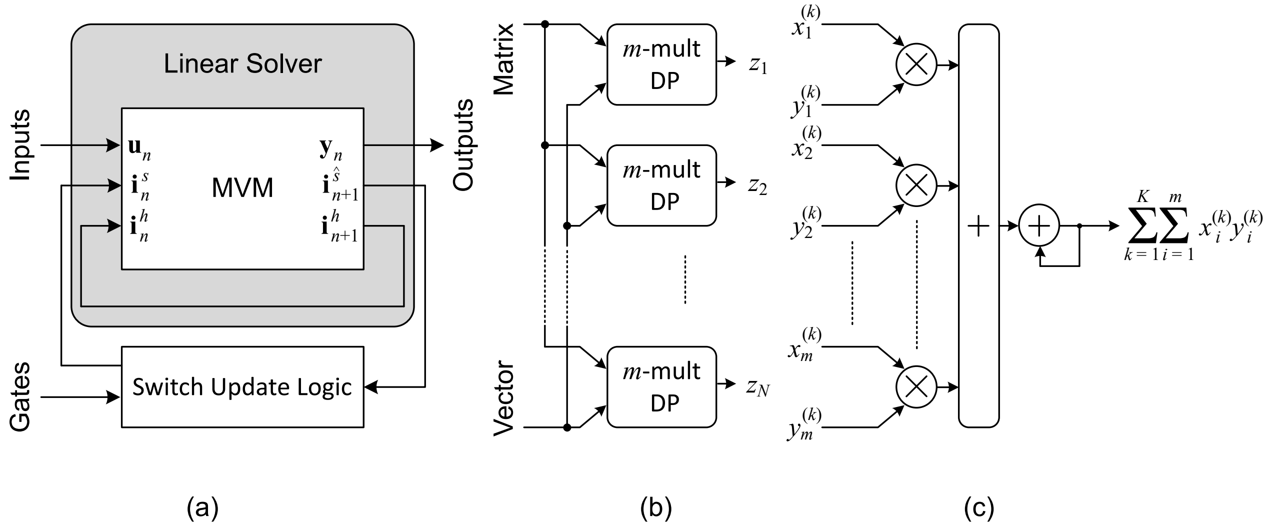


Figure 7.4: Solver Architecture

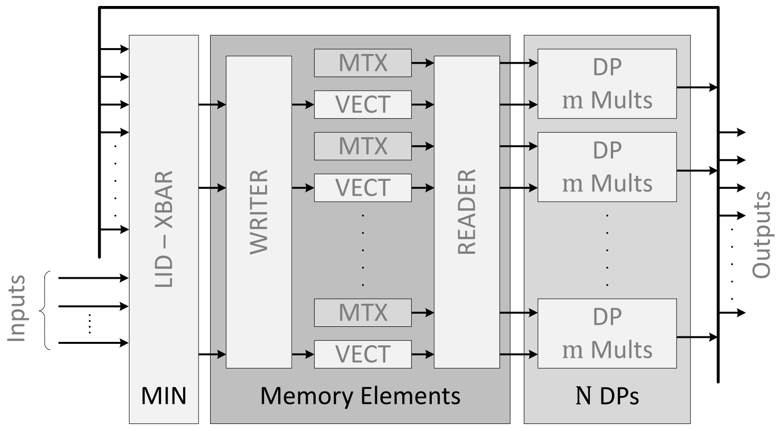


Figure 7.5: Linear Solver Architecture

arranged in a Butterfly topology [128], as shown in Fig. 7.6:.a. Each cell has 2 inputs and 2 outputs channels that are used to connect to neighbouring stages. The number of cells in the Butterfly MIN amounts to $(N/2)log_2(N)$.

For this work, we have adopted the MIN implementation presented in [12], which offers a straightforward routing scheme and such capabilities as multi-casting, broadcasting, and data contention handling. Fig. 7.6:.b illustrates the associated 2×2 cell, which is designed using a LID approach. The cell is composed of two fork units, four FIFOs, and two output selectors. The fork units receive incoming data from the previous stage and route them to one or both output channels on the following stage. In a butterfly MIN, there exists a unique path for each pair of input and output channels, which may result in blocking situations [129]. The

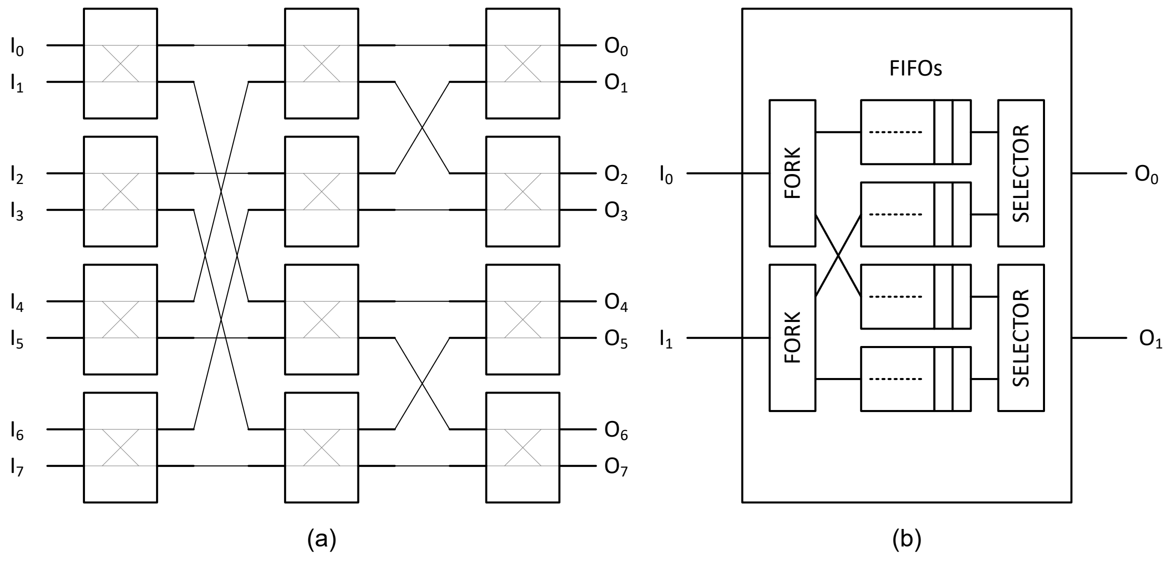


Figure 7.6: Multilevel Interconnection Network. (a) 8×8 Butterfly topology; (b) 2×2 MIN Cell

FIFOs in Fig. 7.6:.b are used to handle contention and avoid blocking. The output selectors are connected to two FIFOs each, and are responsible for sending the data to the next stage. If at least one of the two FIFOs connected to a given selector is not empty and the cell from the next stage is ready to receive data, the data is read and passed forward. When the two FIFOs have data, the selector will read from one of the two FIFOs using a simple round robin scheduling, switching the priority of the FIFOs for the next time the situation occurs.

7.4.5 Memory Elements

The main tasks of the Memory Elements unit of Fig. 7.5: are: *i)* Store circuit network equations entries (MTX), as well as inputs and state variables (VECT); and *ii)* Sequence read and write operations.

The data in MTX is fixed during simulation time while the data in VECT is updated at each time step. The MTX and VECT are implemented using m parallel RAM blocks each, to allow simultaneous access to multiple elements at a time. Moreover, the VECT memories use double buffering technique to avoid read/write overlaps. While one segment of the memory is used for reads, the other is dedicated to writes. Read and write segments are swapped at the following simulation time-point of the simulation.

The round robin rule implemented by the MIN cell of Fig. 7.6:.b provides it with the capability of not prioritising one incoming channel over the other. However, it makes the output order of the MIN harder to predict. A simple way to manage the problem is by adding a tag at the input of the MIN. For each time-step, and each input channel, a tag formed by the input port ID and the arrival order is concatenated with the data token. Hence, the variables are easily identifiable at the output of the MIN, even if they arrive out of order.

The role of the *Reader* block of Fig. 7.5: is to read data from the memory locations when all state variables are available, with respect to the double buffering scheme. Its role is also to make the read order deterministic so that state variables are produced in a specific order, and feed to the input channels of the MIN in a predefined sequence. On the other hand, the role of the *Writer* block of Fig. 7.5: is to identify the incoming variables from their tag and write them to correct memory locations, with respect to the double buffering scheme.

7.4.6 Dot-Product Operator

As discussed previously, a DP operator is an arithmetic unit consisting of m parallel multipliers feeding an addition tree and terminated by an accumulation function, as shown in Fig. 7.4:.c. The DP considered in the paper is based on an approach that combines fused-datapath [148] and self-alignment format (SAF) [118], and is shown in Fig. 7.7:.

Fused-datapath (FDP) is a paradigm that consists in building a complex FP datapath by removing all intermediate packing and unpacking stages and performing all the computations jointly. This is explicit from Fig. 7.7:, where unpacking blocks are found at the inputs and the output of the DP. SAF on the other hand is a format made to process FP additions efficiently and to reduce the latency of the arithmetic operator: all mantissas are aligned with respect to the last bits of the number's exponent (fine grain left shifts), whereas all additions can be performed at a higher clock frequency and in a single clock cycle, since only coarse grain right shift alignments are needed [118].

The FDP/SAF approach was previously adopted in [17, 48, 138] to generate dot product operators for the real-time simulation of power converters, and was shown to be computationally efficient, capable of sustaining higher clock frequencies while offering significant hardware resource savings.

7.5 Results

This section presents synthesis results, time-step evaluation, and simulation assessments for a selection of overlays, along with a discussion. We discuss resource consumption and timing performance for each synthesized architecture, and determine the minimum achievable time-step for each overlay for a variety of circuit sizes. Simulation results for an ac-dc-ac converter test case are presented and used to discuss the computational accuracy of the HS.

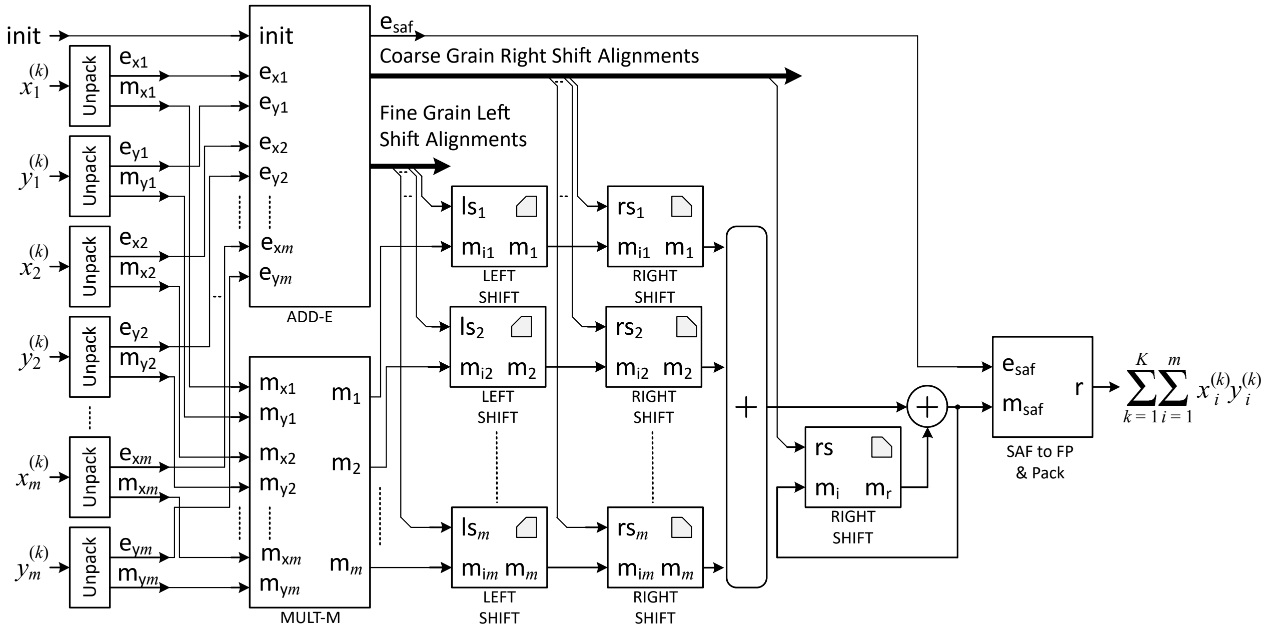

init
Unpack
ADD-E
MULT-M
Coarse Grain Right Shift Alignments
Fine Grain Left Shift Alignments
LEFT SHIFT
RIGHT SHIFT
SAF to FP & Pack
e_{saf}
m_{saf}
$\sum_{k=1}^{K}\sum_{i=1}^{m} x_i^{(k)} y_i^{(k)}$

Figure 7.7: Dot product operator implemented using a fused path self-alignment format

7.5.1 Synthesis Results

The hardware overlay was implemented using vendor-agnostic VHDL, following a generic coding style, free from any directive or primitive specific to a particular technology. The overlay is generated from a set of parameters: that is the number of DPs and their type. The architecture also supports changing the data size and memory depth, but these two parameters were considered with default values, which are sufficient for all the test cases considered for this evaluation. Namely, we have used a non-standard IEEE format with 8-bit exponent and 34-bit mantissa, which is consistent with previously published works such as [23]. The memory depth was fixed to 2^{10} entries, to match the size of BRAMs. The Virtex 7 (7vx485tffg1157-1), was considered for this evaluation. Without being the latest technology, this device offers the resources and timing capabilities to implement the HS of appreciable sizes. Synthesis results were obtained using Vivado 2015.3.

Three OAs have been considered for this study, namely:

- Architecture 1: 4 DPs, of DP-type 8: $(N, m) = (4, 8)$;
- Architecture 2: 8 DPs, of DP-type 16: $(N, m) = (8, 16)$;
- Architecture 3: 16 DPs, of DP-type 32: $(N, m) = (16, 32)$.

Table 7.1: Synthesis Results for Xilinx Virtex 7 (7vx485tffg1157-1)

Architecture	**Registers**	**LUTs**	**RAM**	**DSPs**	**Freq. (MHz)**	**CP (GFLOPS)**
Architecture 1	19,979 (3.3%)	26,913 (8.9%)	48 (4.7%)	128 (4.6%)	303.0	19.39
Architecture 2	78,131 (12.9%)	83,801 (27.6%)	152 (14.8%)	512 (18.3%)	303.0	77.56
Architecture 3	264,645 (43.6%)	286,919 (94.5%)	560 (54.4%)	2,048 (73.1%)	303.0	301.27

Table 7.1: outlines synthesis results for these overlays. It clearly shows that Architecture 3 is indeed the largest overlay than can be implemented on the selected FPGA, given the scarcity of reconfigurable resources (LUTs). It is interesting to discuss the results of Table 7.1: in light of the reported Computing Power (CP) of each overlay, expressed in Giga FLoating-point OPerations per Second (GFLOPS). CP is given at peak performance, and is computed using the formula CP $= N \times (2 \times m) \times f_{max}$, where f_{max} is the maximum sustainable clock frequency, and $N \times (2 \times m)$ is the number of simultaneous FP operations by an (N, m) overlay. It appears that CP is 4 times higher from one set of parameters to the next, which is consistent with the fact that parameter m and N are multiplied by 2 each time we move from Architecture 1 to Architecture 2, and from Architecture 2 to Architecture 3. We see that DSP block utilisation is increased by a factor of 4 each time, whereas registers increase by factors 3.9× and 3.4× respectively; LUTs by 3.1× and 3.4× respectively, block RAMs by 3.2× and 3.7× respectively. This shows that the area occupation scales linearly with the CP (a slightly sub-linear scale, one could say). This is very advantageous and can be attributed to the use of the proposed MIN, which scales as $\mathcal{O}(n \log(n))$ with the number of ports n.

Table 7.2: outlines a set of comparable works from the literature. For each reference, the details of the architecture is not necessarily known, and so CP was approximated by associating each DSP block with a FP operation, as that is the case for the proposed overlays. What Table 7.2: shows is that the proposed overlay are one of the must powerful and, as none of the reported architectures has even exceeded 50 GFLOPS equivalent, whereas so do Architectures 2 and 3. Table Finally, it is worth mentioning that the largest overlay, namely Architecture 3, offers close to 10× as much CP as the most powerful architectures reported in Table 7.2:. This is very promising as more recent FPGAs are larger, and should help to reach 1 TFLOPS of computing power in the near future.

7.5.2 Time-step Exploration

The overlays have been tested for different power converter sizes, by varying the number of states from 5 to 340, or until the time-step exceeded 1 μs. The number of inputs and outputs was fixed to 16, which is reasonable for a wide range of applications. A similar approach was

Table 7.2: Synthesis results from literature

Design	**FPGA**	**Registers**	**LUTs**	**RAM**	**DSPs**	**Freq. (MHz)**	**CP (GFLOPS)**
M. Dagbagi (2016) [6]	Artix 7	1,197 (9.0%)	1,463 (11.0%)	N/A	186 (85.0%)	100	9.30
H. Chalangar(2020) [52]	Kintex 7	2,373 (0.6%)	1,223 (0.6%)	22 (4.9%)	88 (10.5%)	320	14.10
C. Liu(2020) [10]	Kintex 7	45,509 (9.0%)	54,366 (21.4%)	91 (11.4%)	210 (13.6%)	40	4.20
F. Montaño(2018) [11]	Kintex 7	83,272 (20.4%)	73,161 (35.9%)	N/A	703 (83.7%)	100	35.10
M. Milton(2017) [100]	Virtex 7	8,932 (1.5%)	87,525 (29.0%)	N/A	1,884 (67.0%)	20	18.84
R. Mirzahosseini(2019) [50]	Virtex 7	147,317 (24.3%)	142,296 (46.9%)	258 (12.5%)	361 (12.9%)	175	31.59

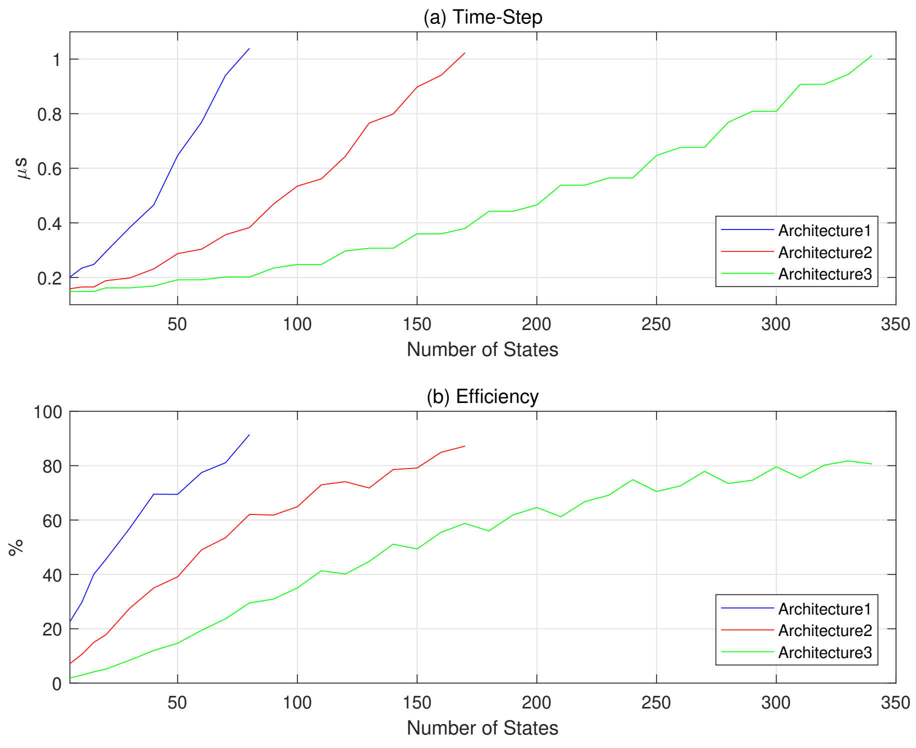


Figure 7.8: Overlay performance. (a) Min. time-step vs nb states; (b) Efficiency

adopted in [11]. Fig. 7.8:.a gives for each overlay the simulation time-step as a function of the number of states. The $\Delta t = 1$ μs limit was chosen as a rule of thumb to accommodate for the high switching frequencies (≥ 10 kHz) of modern power electronics. It is also a time-step that can not be sustained by CPU-based approaches [16], and justifies the use of an FPGA.

It is interesting to see that for a given simulation time-step, the number of states that the solver can handle doubles as we move from Architecture 1 to 2, or from Architecture 2 to 3, whereas the cost in hardware is multiplied by 4. This indicates that it is worth tearing the network up whenever it is possible. For example, if a circuit has 240 states, the time-step is 600 ns using Architecture 3. However, if one can manage a decoupling that breaks the network in two sub-circuits of equal size (120 states), them 2 HS of Architecture 2 could be used instead, for half the cost.

Another way to look at this is from an efficiency point of view. Fig. 7.8:.b shows the efficiency for each overlay and for the various power converter sizes discussed earlier. The efficiency is defined as the ratio of the computing power needed to solve a given problem in the time-step reported in Figure 7.8:.a over the peak performance given in Table 7.1:. In general, the efficiency of an overlay increases with the size of the power converter, although the curve is not monotonic, and slight periodic decreases can be observed. These periodic declines are a side effect of a DP-based overlay, which treats inputs in batches of m pairs of values. This phenomena is more observable with Architectures 2 and 3 as m increases. We also see that all the architectures are not very efficient when the circuit is small, although the simulation time-step is very low in those cases ($\leq$ 200 ns). This means that when the power converter is small, the data spends most of the time in the pipeline or the interconnection network. This last observation is a motivation for low latency arithmetic operators. However, as the network equations get bigger, the latency has limited or no effect.

7.5.3 Open-Loop Test Case 1

We will be considering two open loop test cases and a closed loop test case. For all these tests, we have chosen the AC-DC-AC power converter depicted in Fig. 7.9: as a target circuit. The same circuit was considered in [10]. The circuit parameters are listed in Table 7.3:. Contrarily to [10], we consider here a higher switching frequency, namely 10 kHz instead of 2 kHz.

The circuit has been modeled using the ADC model. As previously discussed, the ADC is known to introduced fictitious oscillations and the conductance needs to be tuned [145]. The tuning was performed using the 2-norm relative error of the load currents [149]:

$$e_{\text{2-norm}}(\%) = 100\frac{||\mathbf{i}_{fpga} - \mathbf{i}_{ref}||_2}{||\mathbf{i}_{ref}||_2}. \tag{7.5}$$

where $\mathbf{i}_{fpga}$ is the current obtained from the FPGA model, whereas $\mathbf{i}_{ref}$ is the reference obtained using the SimPowerSystem (SPS) Matlab toolbox. We have found that the accuracy of the simulation is less sensitive to the conductances associated with the switches at the rectification than the inversion stage. The values of 0.7 ℧ for the rectifier, and 0.02 ℧ for the inverter side were found to give satisfactory results. Our investigation also showed that the model behaves better when the simulation time-step is small, as evidenced by Table 7.4:, which gives the 2-norm relative error for a wide range of time-steps. Typically, for real-time simulation purposes, an error below 5% is required. Hence, a time-step of 300 ns or less is targeted for

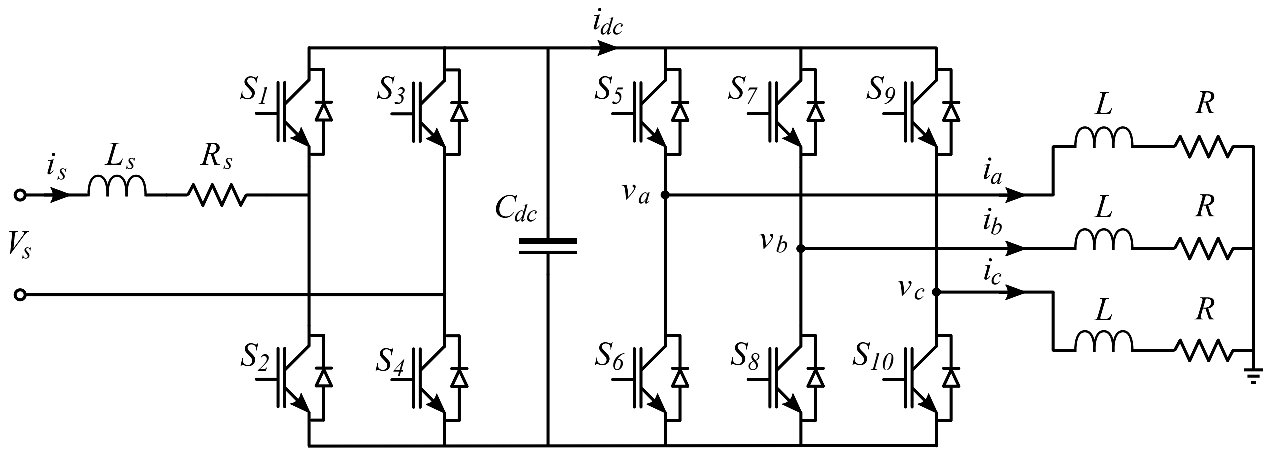


Figure 7.9: AC-DC-AC Power Converter. Adapted from [10]

Table 7.3: ACDCAC Converter parameters

Parameter	**Value**	**Units**
V_s Peak	3600	V
V_s Freq	50	Hz
L_s	2	mH
R_s	39	mΩ
L	2	mH
R	10	Ω
C_{dc}	3.01	mF
F_{mod}	50	Hz
F_c	10	kHz
Mod Index	0.8	

Table 7.4: 2-Norm Error Relative Error

Signal	**@100 ns**	**@150 ns**	**@200 ns**	**@250 ns**	**@300 ns**	**@400 ns**	**@500 ns**	**@1 μs**
i_a	3.26	3.66	4.07	4.50	4.92	5.79	6.67	10.94
i_b	3.19	3.59	4.00	4.43	4.84	5.72	6.60	10.87
i_c	2.92	3.30	3.70	4.12	4.53	5.40	6.27	10.55

this test case.

Here are the simulation time-steps for the AC-DC-AC for the various AOs:

- Architecture 1: 55 clock cycles, i.e. 200 ns @275MHz;
- Architecture 2: 44 clock cycles, i.e. 160 ns @275MHz;

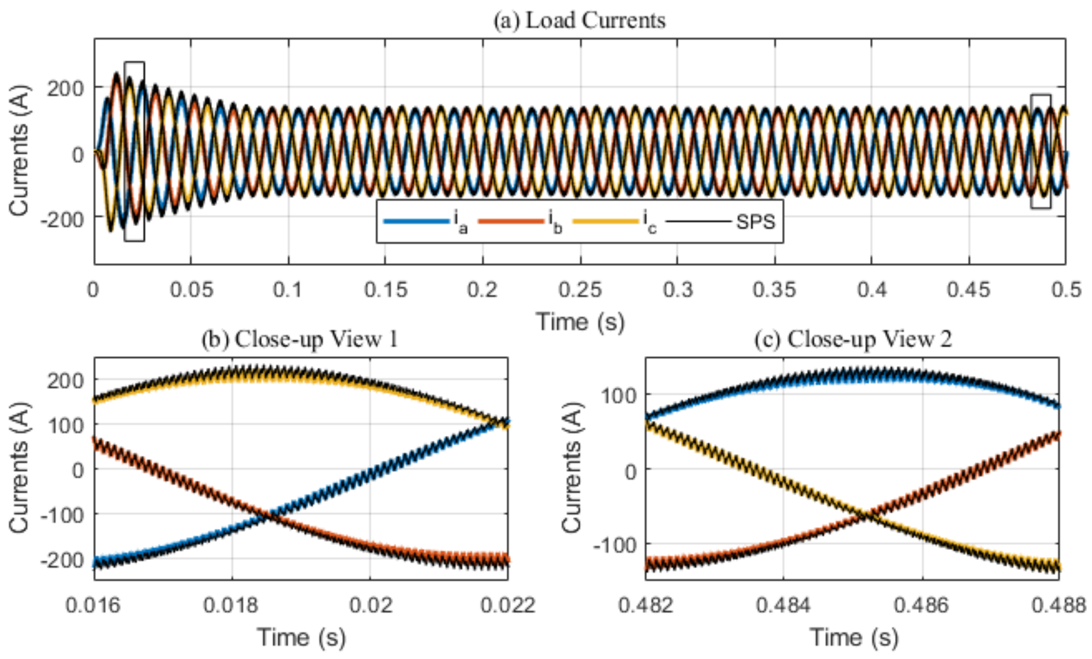


Figure 7.10: 3-phase load currents: (a) Whole sequence view with indications for close-up views; (b) Close-up view #1; (c) Close-up view #2

- Architecture 3: 41 clock cycles, i.e. 149 ns @275MHz.

From the results above, it appears that Architecture 1 has sufficient performance to simulate the AC-DC-AC converter. Fig. 7.10:.a presents the load currents obtained with a time-step of 200 ns using Architecture 1 and over a time span of 0.5 s. Two close-up views illustrated as rectangles in Fig. 7.10:.a — that is, a short time following start-up (0.016 s) and during steady state (0.482s) — are shown in Fig. 7.10:.b and Fig. 7.10:.c. Fig. 7.10: shows the good agreement of the FPGA simulation with the SPS reference.

Table 7.5: compares the Total Harmonic Distortion (THD) of the FPGA results generated a time-step of 200 ns with the THD from the SPS model. Such an evaluation can be useful to determine if the FPGA-based simulation is wrongly introducing new harmonics to the generated signals; or to the contrary, if it is reducing its harmonic content. As shown in Table 7.5:, the SPS reference and the HS show similar THD, are in the same range with a relative error below 3%.

We have also assessed the computational accuracy of the hardware DP operators by comparing the FPGA results to a Matlab code implementing the ADC switch model. The resulting relative errors are calculated for each time-point of the simulation using the Eq. 7.6:

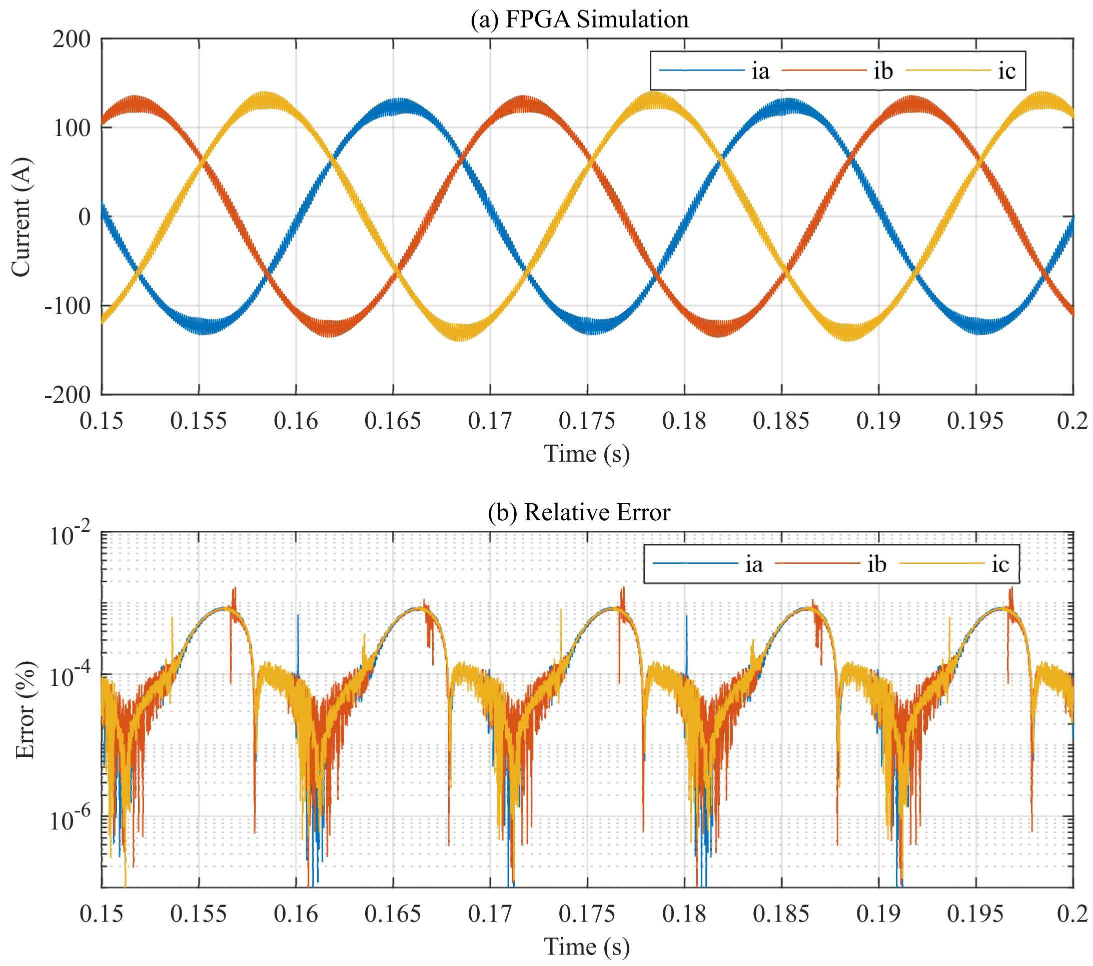


Figure 7.11: Close-up view of 3-phase load currents (a) HS FPGA-simulation; (b) Relative error for each time-point of the simulation

Table 7.5: THD Comparison

Signal	**SPS (%)**	**LID-HS (%)**	**Error (%)**
i_a	4.24	4.37	2.99
i_b	4.08	4.14	1.42
i_c	4.14	4.24	2.46

$$e_{rel}(\%) = 100\frac{|i_{fpga} - i_{mtlb}|}{|i_{mtlb}|}. \tag{7.6}$$

where i_{mtlb} is the matlab results used as a reference. We have found that relative error varies between $10^{-3}\%$ and $10^{-4}\%$, as shown in Fig. 7.10:.b. Fig. 7.11:.b also shows that the relative increases at certain moments during the simulation, over the $10^{-3}\%$ threshold. However, by correlating these moments with the current signals from Fig. 7.10:.a, it appears that these spikes occur only during zero crossing of the currents, and as such can be neglected.

7.5.4 Open-Loop Test Case 2

For the second open-loop test case, the test sequence starts with the amplitude of the ac voltage input set to $|V_s| = 1,800$ V. At $t = 0.1$ s, the amplitude is set to $|V_s| = 3,600$ V and maintained at that level for the rest of the remainder of the test. A third test sequence starts at $t = 0.25$ s, when a change in load resistor is applied, moving from $R = 10\ \Omega$ to $R = 5\ \Omega$.

Fig. 7.12: presents the load currents for this second test. Fig. 7.12:.a identifies the position of the close-up views shown in Fig. 7.12:.b, Fig. 7.12:.c and Fig. 7.12:.d, each from one test sequence. The results shows the very good agreement between the simulation and the SPS reference under changing working conditions.

7.5.5 Closed-Loop Test Case

In this section, we assess the performance of the HS in a closed-loop configuration. The digital proportional resonant (PR) controller of Fig. 7.13: was connected to the system. The controller is formed by a proportional gain k_p added to a resonant path formed by a gain k_i and a resonant filter. The filter equation is given by:

$$H_{\mathrm{r}}(z) = \frac{b_0 z + b_1 z^1 + b_2 z^2}{a_0 z + a_1 z^1 + a_2 z^2}, \tag{7.7}$$

The parameters of the controller (filter coefficients and gains) are listed in Table 7.6:, and were calculated using the process described in [150]. For this test, the load currents are sampled at 100 ksps, which is typical of low-cost A/D, and accounts for the worst case scenarios.

Fig. 7.14: shows the load currents obtained in close-loop over a time span of 0.5 s. The test is comprised of two test sequences: From $t = 0$ s to $t = 0.15$ s, the load resistor is $R = 5\ \Omega$; After $t = 0.15$ s the load resistor changes to $R = 10\ \Omega$. Fig. 7.14:.a identifies the position of the close-up views shown in Fig. 7.14:.b, Fig. 7.14:.c and Fig. 7.14:.d, one from each test sequence (resp. Fig. 7.14:.b and Fig. 7.14:.d) and the load transition instant (Fig. 7.14:.c).

The close-up views show that the controller compensates for the change in load and presents very fast response time at transition (so that is not even visible on the figure). Most notably, we see a very good agreement between the model and the SPS reference in all the conditions.

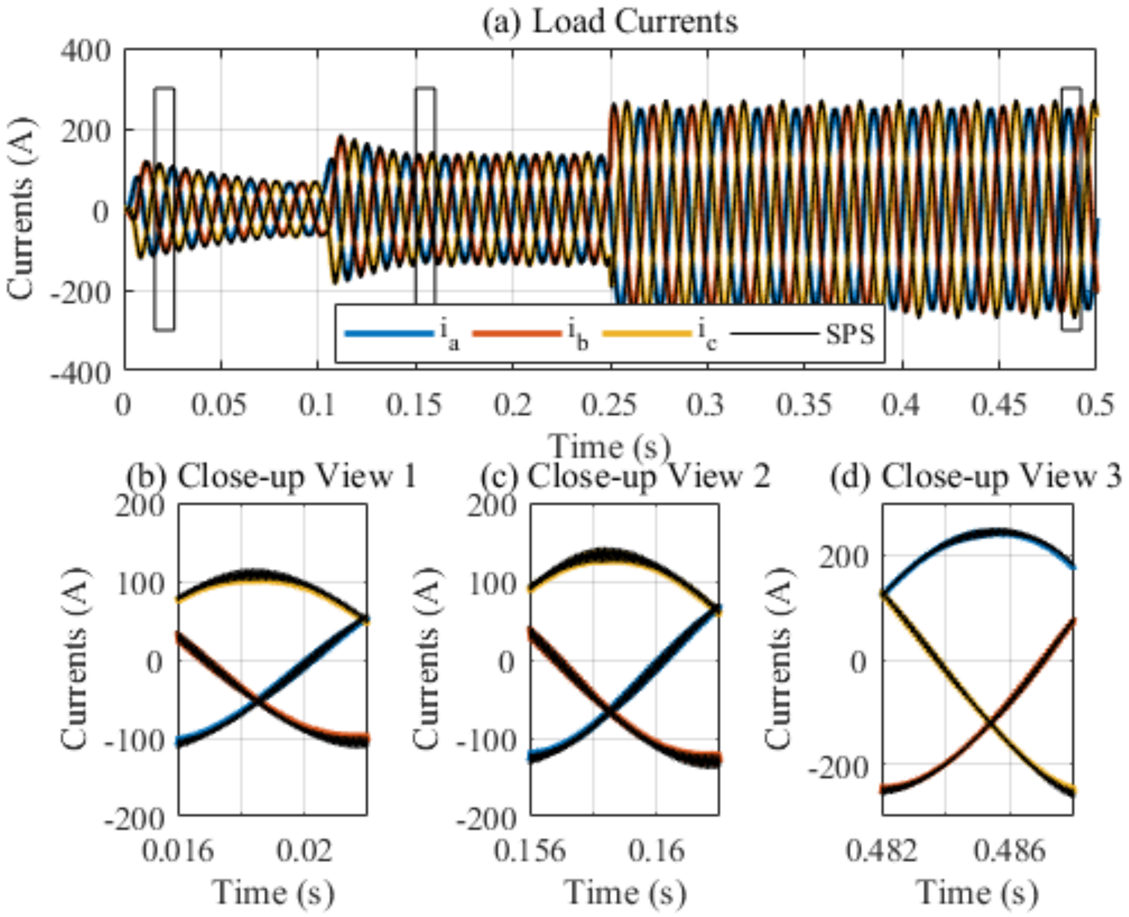


Figure 7.12: Three-phase load currents: (a) Whole sequence view with indications for close-up views; (b) Close-up view #1; (c) Close-up view #2; (c) Close-up view #3

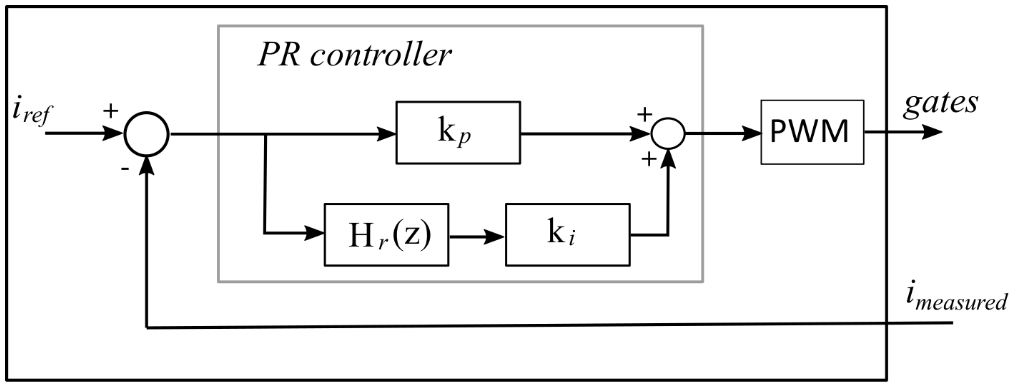


Figure 7.13: Control strategy block diagram

Table 7.6: PR controller parameters

Parameter	Variable	Value	Units
Switching Frequency	F_c	10	kHz
Sampling Frequency	F_a	100	kHz
Grid Frequency	F_{mod}	50	Hz
Resonant Frequency	F_r	50	Hz
Resonant Bandwidth	B_r	1.5	Hz
Damping factor	ζ	0.95	-
Proportional Gain	k_p	1	-
Resonant Gain	k_i	0.4501	-
Filter parameter	b_0	9.424777960769380e-05	-
Filter parameter	b_1	-9.424731452853712e-05	-
Filter parameter	b_2	0	-
Filter parameter	a_0	1	-
Filter parameter	a_1	1.999895887530370	-
Filter parameter	a_2	0.999905756661575	-

7.6 Discussion

The methodology presented in this paper is an alternative design approach based on an overlay architecture, and aims at reducing the gap between RTL and HLS. New overlays can be generated by simply changing design parameters at the top level. It is difficult to establish a fair comparison among results in the literature and our approach due to the differences on technologies, type of circuit simulated, numerical representation, etc. However, we used the computing power as metric to assess the capability of a given design and were able to show that the proposed overlays are among the most capable in the literature.

All generated architectures (Architectures 1, 2 and 3) are sub-microsecond capable as shown in Fig. 7.8:. Typically, for a given circuit size, larger overlays result in smaller time-steps, but are offer less efficiency. For instance, a circuit with 50 states will be simulation at 660 ns (70% efficiency) on Architecture 1, 300 ns (40% efficiency) on Architecture 2, and 200 ns (18% efficiency) on Architecture 3. This is explained in part by the fact that DP inputs are treated in batches, and zero padding is required when the vector size is not an integer multiple of the DP type.

That being said, the main concern of a real-time simulator designer is achieving a small simulation time-step, which is clearly a target met for this work. For example, for the circuit discussed in [6], a time-step of 300 ns can be reached by Architecture 1, instead of the 500 ns reported in the paper. Also, for the circuit discussed in [50], Architectures 2 can reach a time-step of 200 ns, instead of the 800 ns reported in the paper.

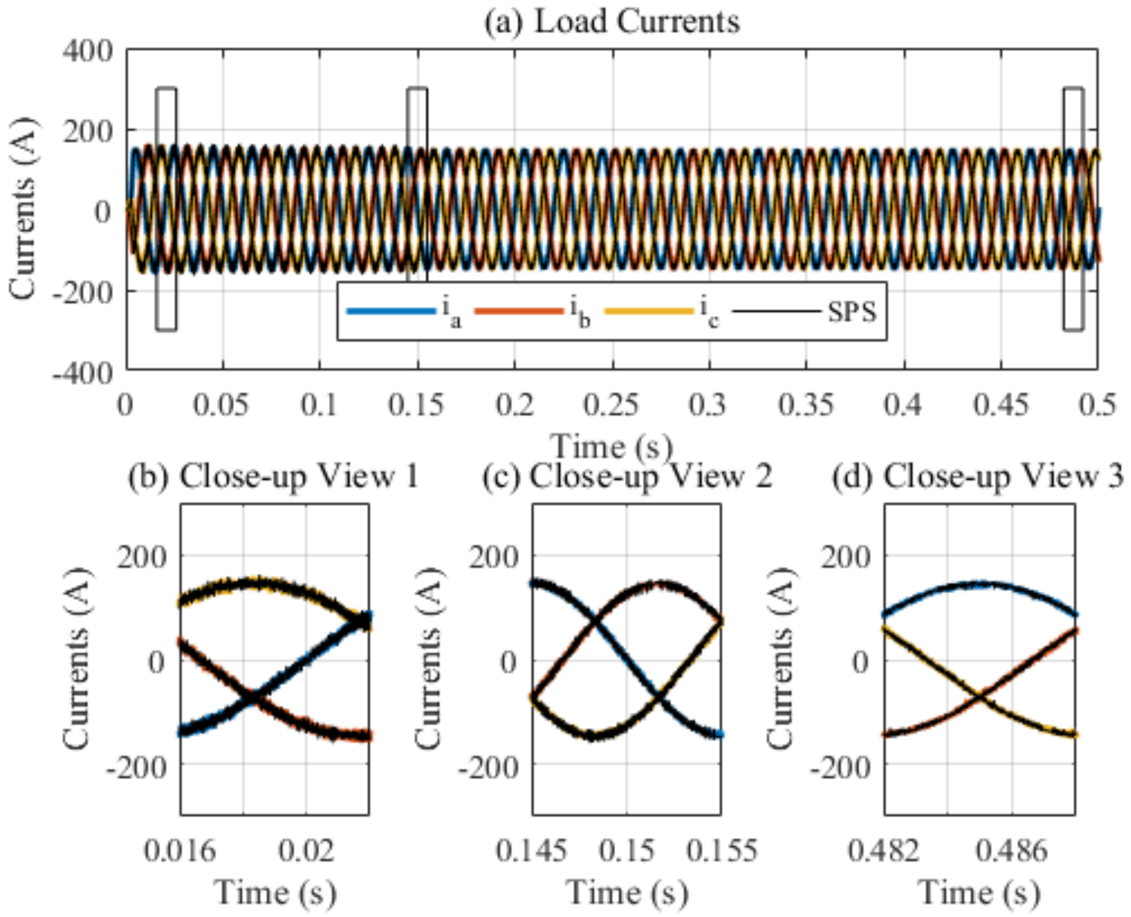


Figure 7.14: Three-phase load currents in closed-loop: (a) Whole sequence view with indications for close-up views; (b) Close-up view #1; (c) Close-up view #2 and (d) Close-up view #3

At the present time, the synthesis of the overlay architecture is done manually, and it is still the work of the developer to determine the set of parameters (DP number and type) to choose from to find the architecture that best suits the circuits size vs. simulation times-step required. That could be improved by a automating the analysis from the circuit specifications (size and target time-step).

The main limitation of this methodology is that the minimum time-step for a given circuit can no be foretold due non-deterministic behavior of the MIN (round robin). Hence, a simulation must be executed to determine the time-step. A fast simulation tool could improve this drawback, for instance by having cycle-accurate models of the overlay.

7.7 Conclusions

This paper presented a new methodology for the design of programmable HS for the FPGA-based real-time simulation of power converters. The programmability of the HS results from the use of an overlay architecture that separates the execution of an application from the hardware development. The overlay on the other hand is implemented using a latency insensitive design approach that makes all modules auto-synchronised, allows modularity and eases integration. The overlay was implemented using a vendor-agnostic VHDL code, that uses generic parameters to easily scale the hardware solver to the desired performance. To achieve high performance and reach small time-steps, low latency custom floating-point operators that combine a fused datapath approach and a self-alignment format, were used. Three different overlays were generated and implemented on a Xilinx Virtex 7 FPGA, and were shown to achieve high performance while being cost effective. An in-depth analysis of the performances of the overlays has shown that the area occupation scales linearly with the computing power; that the decoupling can achieve higher efficiency at a lower hardware cost; and that the low latency of the arithmetic operators matters only for small circuit sizes. A test AC-DC-AC test case was also considered. A time-step of 200 ns was achieved on the smallest overlay without any decoupling. The test case showed the good agreement of FPGA-based results with a Simulink/Matlab reference and confirmed the computational accuracy achieved by the proposed arithmetic units.

CHAPTER 8 GENERAL DISCUSSION

The use of real-time simulators facilitates the test and development of new electrical networks. However, the development of the simulators themselves is a complex task. In this context, the method proposed in this work intended to ease the way for application specialists to implement real-time simulators for power electronics over FPGAs. This work was performed in three main steps. First, we analyzed the benefits and limitations of using HLS for the design of HS comparably to custom design. Then we proposed the design of a fast connection system and introduced the use of LID as design technique. Finally, we combined the LID and the OA approaches to ease the design of re-configurable and re-programmable architectures and automate hardware generation.

In chapter 5, we have shown that, although the use of high-level synthesis eases the process of hardware generation and reduces the time to market, there are still limitations around the applications to be implemented and the performance obtained. We considered the operation frequency, the resource consumption and the accuracy of the simulation for circuits of different sizes. Nevertheless, we mainly focused on the achievable time step. Special directives were used to control the HLS tool operation by latency and by resource usage. The results were compared to a RTL custom design. The custom design could be managed to implement all circuit sizes (2–60 states) at all implementation frequencies (40 MHz - 320 MHz). The resource usage remained almost flat. The time step increased with the circuit size. For a circuit of 60 states, it has a maximal time step of 2.5 μs when working at 40 MHz and less than 0.5 μs when working at 320 MHz. Vivado HLS was not able to implement most of the hardware solvers for frequencies higher than 100 MHz. However, some HLS implementations were valid (i.e., for circuits of 60 states at frequencies of 100 MHz or less, when constrained by resources, and 50 states at the same range of frequencies when constrained by latency). This showed that HLS approach can be used for the implementation of HS for real-time simulation, but there is still some work to do for improving the capacities of the tool. The RTL design used a specialised pipelined approach in the custom floating point operators that improved their performanace. This has been our first contribution in this domain. The associated article has been cited 45 times, according to Google Scholar, at the time of this writing.

In chapter 6, we presented a new re-programmable connection mechanism for multiprocessor systems. It manages the routing of data among the different nodes. The data length can be easily expanded facilitating the support for any numeric format. We have shown that the proposed programmable multistage interconnection network offers high throughput using a light design and routing scheme. Its self-pipelined scheme allows it to work at frequencies up to 500 MHz, which enables its integration to a high-performance system. The proposed MIN architecture has easy non-centralized routing protocol and presents low footprint. This MIN was thought as a strategy to tackle limitations like those found in chapter 5 related to the methods for linking efficient custom modules. The MIN is the connection system in the architectures generated for the real-time hardware solvers presented in chapter 7.

Chapter 7 integrated the MIN, the floating point operators forming the dot product operators, and the memory modules to obtain a hardware simulator with the proposed methodology. It used the combined approaches of OA (that allows reprogramming without expending time in the synthesis process) and LID (that facilitates the modularity of design and its integration). These combined approaches have never been used before for implementing real-time simulators. This methodology allowed the generation of configurable architectures for the implementation of HS for real-time simulation. The generated architectures work at frequencies greater than 250 MHz and produce results of sub-microsecond time step. The studies performed over multiple circuit configurations showed that the smallest architecture (namely architecture 1) can simulate circuits with more than 50 states at time steps below 1 μs with the efficiency of 80% approximately. Bigger architectures can simulate bigger circuits in an under-microsecond range. It can go up to 150 states and 300 states for architectures 2 and 3 respectively, with an efficiency of 80% approximately.

The summarized results from the literature presented in chapter 3 have been retaken in this section with our results added at the three final rows of the tables and presented in tables 8.1: and 8.2:. As multiple types of circuits can be simulated in each of our generated architectures, we use the CP and the reachable time step as a comparison point with the existent literature. The three architectures (Architecture 1, 2, 3) offer a computing power of 19.39 GFLOPS, 77.56 GFLOPS and 301.27 GFLOPS respectively. These values are comparable or superior to those presented in the literature. If we consider the Architecture 1, we can see that any circuit with 70 or fewer states can be simulated in less than 1 μs. The 128 required DSP blocks make it comparable or better than most of the analyzed literature, even for those having FXP numerical format. Architecture 3 presents the highest resource consumption. However, It is capable of solving the biggest circuit (340 states) in sub-microsecond range. The second most consuming resources solution solves a circuit with 140 states using FXP. As a case study, an AC-DC-AC converter with 25 states has been simulated on the three generated architectures. They work at 275 MHz obtaining time steps of 200 ns, 160 ns, and 149 ns for architectures 1, 2 and 3 respectively. As the architectures can be reprogrammed the circuit can increase or decrease in a certain range keeping the same resources consumption.

Table 8.1: Synthesis Results from State of the Art

Ref.	**FPGA**	**Registers**	**LUT**	**RAM**	**DSP**	**Freq. (MHz)**
M. Dagbagi (2016) [6]	Artix 7	1,197 (9.0%)	1,463 (11.0%)	NR[1]	186 (85.0%)	100
H. Chalangar(2020) [52]	Kintex 7	2,373 (0.6%)	1,223 (0.6%)	22 (4.9%)	88 (10.5%)	320
C. Liu(2020) [10]	Kintex 7	45,509 (9.0%)	54,366 (21.4%)	91 (11.4%)	210 (13.6%)	40
F. Montaño(2018) [11][a]	Kintex 7	83,272 (20.4%)	73,161 (35.9%)	NR	703 (83.7%)	100
F. Montaño(2018) [11][b]	Kintex 7	21,136 (5.2%)	35,726 (17.5%)	NR	128 (15.2%)	100
M. Milton(2017) [100]	Virtex 7	8,932 (1.5%)	87,525 (29.0%)	NR	1,884 (67.0%)	20
R. Mirzahosseini(2019) [50]	Virtex 7	147,317 (24.3%)	142,296 (46.9%)	258 (12.5%)	361 (12.9%)	175
C. Liu (2018) [101]	Kintex 7	42,642 (11.2%)	47,028 (25.8%)	91 (11.6%)	120 (17.7%)	200
T. Ould-Bachir(2015) [48]	Virtex 6	58,216 (19.3%)	29108 (19.3%)	41 (9.8%)	116 (15.1%)	400
D. Majstorovic (2011) [40]	Virtex 5	NR[1]	NR[1]	117 (89.0%)	40 (14.0%)	200
*F. Montaño (2020) [13][a]	Virtex 7	19,979 (3.3%)	26,913 (8.9%)	48 (4.6%)	128 (4.6%)	303
*F. Montaño (2020) [13][b]	Virtex 7	78,131 (12.9%)	83,801 (27.6%)	152 (14.8%)	512 (18.3%)	303
*F. Montaño (2020) [13][c]	Virtex 7	264,645 (43.6%)	286,919 (94.5%)	560 (54.4%)	2048 (73.1%)	303

[1]NR:Not Reported, *a*: solution 1, *b*: solution 2, *c*: solution 3, $*$: this work

Table 8.2: Design Characteristics from State of the Art

Ref.	**Number Format**	**Design Tool**	**Application**	**Size (Nb States)**	**Ts (ns)**	**CP (GFLOPS)**
M. Dagbagi (2016) [6]	FXP[1]	NR[4]	VSR	21	420	9.30
H. Chalangar (2020) [52]	FXP	NR	LLC	22	40	14.10
C. Liu (2020) [10]	FXP	NI[5]-HLS	AC-DC-AC	25	25	4.20
F. Montaño (2018) [11][a]	SFP[2]	HLS	NPC	41	630	35.10
F. Montaño (2018) [11][b]	CFP[2]	RTL	NPC	41	310	6.40
M. Milton (2017) [100]	FXP	HLS	VSC($\times$6)	140	50	18.84
R. Mirzahosseini (2019) [50]	SFP[3]	HLS	VSC	64	180	31.59
C. Liu (2018) [101]	FXP	NI-HLS	NPC	41	40	12.00
T. Ould-Bachir (2015) [48]	CFP	SG[6]-RTL	NPC	41	750	23.20
D. Majstorovic (2011) [40]	FP-FXP	RTL	AC-DC-AC	37	1000	12.00
*F. Montaño (2020) [13][a]	FP	RTL	Generic	<70	<1000	19.39
*F. Montaño (2020) [13][b]	FP	RTL	Generic	<170	<1000	77.56
*F. Montaño (2020) [13][c]	FP	RTL	Generic	<340	<1000	301.27

[1]FXP:Fixed Point, [2]CFP:Custom Floating Point, [3]SFP: Single Floating Point, [4]NR:Not Reported, [5]NI:National Instruments, [6]SG:System Generator, *a*: solution 1, *b*: solution 2, *c*: solution 3, $*$: this work

It is important to notice that with the presented method a new architecture can be synthesized just by modifying some parameters. Additionally, each generated architecture can be reprogrammed to solve a new circuit without going through a synthesis process. The programming tool takes the circuit matrix and the architecture parameters to generate the configuration files of the reprogrammable modules in the OA (MIN, memory elements and switch update logic). Most of the designs shown in the literature are custom solutions and are not easy to modify. For HLS approaches, modifications are easier, but synthesis is still required.

CHAPTER 9 CONCLUSION AND RECOMMENDATIONS

In this thesis, a methodology for the development of real-time simulators over FPGA was proposed. This methodology takes advantage of the overlay architecture and the latency insensitive design paradigms to generate reconfigurable and reprogrammable data-driven architectures to implement hardware solvers for real-time simulation. The different stages of this research, including the analysis of the utilization of existing tools (advantages and limitations), the system design, and modules development were studied and presented. The results were discussed and compared to other works to highlight the contribution of this work. The proposed methodology was used to implement three hardware solvers architectures for testing its applicability for performing real time simulation of power converters over FPGA. Multiples architectures can be generated by a simple modification of parameters. However, only three architectures where generated and tested due to resources limitation on FPGA devices. The generated architectures are big enough to fit in a Xilinx Virtex-7 FPGA. They are capable of working at frequencies greater than 275 MHz. In all cases, a design exploration was performed to show the behavior of the reached time-step and computation efficiency for different circuits sizes on each architecture. An AC-DC-AC converter was used as study case for testing the performance of the architectures. A simulation time-step of 200 ns was reached. On simulated signals, a 2-Norm relative error below to 5% and a THD = 4.37% were obtained — a THD error of 3.0% comparative to results obtained with SPS.

9.1 Synopsis of the work

We presented our research work heading towards the development of a methodology in order to increase the level of abstraction for the design and development of real-time simulators on FPGAs. The combination of latency insensitive design approach and overlay architectures has been used essentially to isolate the processes of module-level design and system integration.

The organization of this thesis allows that each chapter introduces an original contribution that arises from this research. In chapter 5, a study of using a commercial HLS tool (Vivado HLS) for developing real-time hardware solvers is presented. This study has been realized to analyze the benefits and drawbacks of using a high-level approach comparatively to custom design for developing the hardware solvers for real-time simulation. In this study, the design exploration was done over two dimensions. First constraining by resources and second constraining by latency. It has been shown that custom design outperforms HLS tools. However, when the circuit to be simulated is small and the time-step requirements are high enough it is possible to generate solvers with real-time capabilities using HLS. Furthermore, the simplicity of coding in C++ and performing design space exploration by a simple modification of directives makes HLS an attractive approach. In chapter 6, the architecture of a multistage interconnection network (MIN) using a latency insensitive design approach is presented. The topology and straightforward routing schemes make the MIN a light design with high throughput, capable of working at frequencies higher than 500 MHz. Its structure is conformed by a reconfigurable number of 2×2 basic cells organized in a butterfly topology. Each cell provides multi-cast and buffering capabilities. This way, the whole structure supports data contention and non-blocking functioning. The chapter presented a design exploration for this MIN for different values of the parameters such as the number of ports, port width, and FIFOS depth.

In chapter 7, the complete methodology for the design of real-time simulators for power electronics converters on FPGAs was presented. This chapter presented the integration of all the system components, conformed mainly by the MIN, memories, switch block, and floating-point dot product operators. All blocks were arranged to form a linear OA with each block adapted to support LID which makes them auto-synchronized and facilitates their integration. The methodology was used to generate three HS Architectures. The generated architectures are reprogrammable and capable of performing at sub-microsecond real-time simulation. A study of the performance of these architectures has been performed and compared to existing literature. An AC-CD-AC converter was used as a test case, the smallest architecture was shown to work at time-steps of 200 ns. The signals were compared to an SPS reference with a relative error in load currents amplitude inferior to $10^{-3}\%$ and a THD error below 3% which confirmed the accuracy obtained by the arithmetic operators.

9.2 Limitations of the Proposed Solution

The main limitation of this methodology is that the minimum time-step for a given circuit can no be foretold due non-deterministic behavior of the MIN (round-robin). Hence, a simulation must be executed to determine the time-step. A fast simulation tool could compensate this drawback, for instance by having cycle-accurate models of the overlay.

The same methodology can be used to introduce new components to improve the library. Currently, the component library only considers the IGBT switch and linear components. Although a wide set of power converters can be simulated with the presented solution, some limitations will appear if one intends to simulate power converters with different components (e.g. circuits with saturable transformers). To cope with these limitations, the library should be improved.

From the software point of view, the tool requires some values to be manually entered (e.g., circuit matrix, architecture parameters) before the automatic generation of OA configuration files. Improving the tool to generate automatically the matrix and choose the appropriate architecture based on the power converter will reduce the development cycle.

9.3 Future Research

The works performed in this research allowed the generation of multiple reprogrammable architectures with configuration of different parameters. The latency-insensitive design paradigm has been used to isolate the design process of a module and the integration of the system following the principle of correct construction. However, multiple improvements can be brought to this work by making it more general and adapting it for future studies and developments, for example:

- By creating a mathematical model or simulation tool that can pre-calculate the time step for any given circuit on any generated architecture.
- By creating an interface that allows to easily extract the matrix of any power circuit and calculate the parameters to generate the simulator architecture that offers the best cost/performance ratio.
- By increasing the library of operators or modules that support the different components instantiated in the circuit models

REFERENCES

[1] X. Guillaud, M. O. Faruque, A. Teninge, A. H. Hariri, L. Vanfretti, M. Paolone, V. Dinavahi, P. Mitra, G. Lauss, C. Dufour, P. Forsyth, A. K. Srivastava, K. Strunz, T. Strasser, and A. Davoudi, "Applications of real-time simulation technologies in power and energy systems," *IEEE Power and Energy Technology Systems Journal*, vol. 2, no. 3, pp. 103–115, Sept 2015.

[2] "Vivado design suite user guide high-level synthesis," https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug902-vivado-high-level-synthesis.pdf, accessed: 2020-10-05.

[3] "Intel® FPGA SDK for OpenCL™ProEdition high-level synthesis," https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/opencl-sdk/aocl_getting_started.pdf, accessed: 2020-10-05.

[4] C. Liu, H. Ng, and H. K. So, "Quickdough: A rapid FPGA loop accelerator design framework using soft CGRA overlay," in *2015 International Conference on Field Programmable Technology (FPT)*, 2015, pp. 56–63.

[5] K. E. Murray and V. Betz, "Quantifying the cost and benefit of latency insensitive communication on FPGAs," in *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 223–232. [Online]. Available: https://doi.org/10.1145/2554688.2554786

[6] M. Dagbagi, A. Hemdani, L. Idkhajine, M. W. Naouar, E. Monmasson, and 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 Trans. Ind. Electron.*, vol. 63, no. 2, pp. 1179–1190, 2016.

[7] W. Meeus, K. Van Beeck, T. Goedemé, J. Meel, and D. Stroobandt, "An overview of today's high-level synthesis tools," *Design Automation for Embedded Systems*, vol. 16, no. 3, pp. 31–51, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10617-012-9096-8

[8] N. Kapre and J. Gray, "Hoplite: Building austere overlay NoCs for FPGAs," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–8.

[9] A. K. Jain, X. Li, P. Singhai, D. L. Maskell, and S. A. Fahmy, "DeCo: A DSP block based FPGA accelerator overlay with low overhead interconnect," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2016, pp. 1–8.

[10] C. Liu, H. Bai, S. Zhuo, X. Zhang, R. Ma, and F. Gao, "Real-time simulation of power electronic systems based on predictive behavior," *IEEE Trans. Ind. Electron.*, vol. 67, no. 9, pp. 8044–8053, 2020.

[11] F. Montaño, T. Ould-Bachir, and J. P. David, "An evaluation of a high-level synthesis approach to the FPGA-based sub-microsecond real-time simulation of power converters," *IEEE Trans. Ind. Electron.*, vol. 65, no. 1, pp. 636–644, Jan 2018.

[12] F. Montaño, T. Ould-Bachir, J. Mahseredjian, and J. P. David, "A low-latency reconfigurable multistage interconnection network," in *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, 2019, pp. 1–4.

[13] F. Montaño, T. Ould-Bachir, and J. P. David, "A latency-insensitive design approach to programmable FPGA-based real-time simulators," *MDPI-Electronics*, vol. 9, no. 11, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/11/1838

[14] J. Mahseredjian, V. Dinavahi, and J. Martinez, "Simulation tools for electromagnetic transients in power systems: Overview and challenges," *IEEE Trans. Power Del.*, vol. 24, no. 3, pp. 1657–1669, July 2009.

[15] C. Larose, S. Guerette, F. Guay, A. Nolet, T. Yamamoto, H. Enomoto, Y. Kono, Y. Hasegawa, and H. Taoka, "A fully digital real-time power system simulator based on pc-cluster," *Mathematics and Computers in Simulation*, vol. 63, no. 3, pp. 151 – 159, 2003, modelling and Simulation of Electric Machines, Converters and Systems.

[16] G. Lauss, M. O. Faruque, K. Schoder, C. Dufour, A. Viehweider, and J. Langston, "Characteristics and design of power hardware-in-the-loop simulations for electrical power systems," *IEEE Trans. Ind. Electron.*, vol. 63, no. 1, pp. 406–417, 2016.

[17] T. Ould-Bachir, H. Saad, S. Dennetière, and J. Mahseredjian, "CPU/FPGA-based real-time simulation of a two-terminal MMC-HVDC system," *IEEE Trans. Power Del.*, vol. 32, no. 2, pp. 647–655, April 2017.

[18] H. Dommel, *EMTP Theory Book, 2nd edition.* MicroTran Power Analysis Corporation, 1992.

[19] V. Jalili-Marandi and V. Dinavahi, "SIMD-based large-scale transient stability simulation on the graphics processing unit," *IEEE Transactions on Power Systems*, vol. 25, no. 1, 2010.

[20] "Simscape™ Electrical™ user's guide (specialized power systems)," https://www.mathworks.com/help/pdf_doc/physmod/sps/powersys_ug.pdf, accessed: 2020-10-02.

[21] J. H. Alimeling and W. P. Hammer, "PLECS-piece-wise linear electrical circuit simulation for simulink," in *Proceedings of the IEEE 1999 International Conference on Power Electronics and Drive Systems. PEDS'99 (Cat. No.99TH8475)*, vol. 1, 1999, pp. 355–360 vol.1.

[22] T. Ould Bachir and J. David, "FPGA-based real-time simulation of state-space models using floating-point cores," in *Proceedings of 14th International Power Electronics and Motion Control Conference EPE-PEMC 2010*, 2010, pp. S2–26–S2–31.

[23] T. Ould-Bachir, C. Dufour, J. Bélanger, J. Mahseredjian, and J.-P. David, "Effective floating-point calculation engines intended for the FPGA-based HIL simulation," in *International Symposium on Industrial Electronics (ISIE)*, Hangzhou, China, May 2012, pp. 1363–1368.

[24] J. Mahseredjian, S. Dennetière, L. Dubé, B. Khodabakhchian, and L. Gérin-Lajoie, "On a new approach for the simulation of transients in power systems," *Electric Power Systems Research*, vol. 77, no. 11, pp. 1514–1520, 2007, selected Topics in Power System Transients - Part II.

[25] A. Benigni and A. Monti, "A parallel approach to real-time simulation of power electronics systems," *IEEE Transactions on Power Electronics*, vol. 30, no. 9, pp. 5192–5206, 2015.

[26] P. Pejovic and D. Maksimovic, "A method for fast time-domain simulation of networks with switches," *IEEE Trans. Power Electron.*, vol. 9, no. 4, pp. 449–456, July 1994.

[27] M. Dagbagi, L. Idkhajine, E. Monmasson, and I. Slama-Belkhodja, "FPGA implementation of power electronic converter real-time model," in *Int. Symp. Power Electro. Electr. Drives, Automation Motion*, June 2012, pp. 658–663.

[28] J. Bélanger, P. Venne, and J.-N. Paquin, "The what, where and why of real-time simulation," vol. 1, no. 0, pp. 37–49, 2010.

[29] H. Dommel, "Digital computer solution of electromagnetic transients in single-and multiphase networks," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-88, no. 4, pp. 388–399, April 1969.

[30] Y. H. Song, K. Xie, and C. Ferguson, "Development of an ATP-based fault simulator for underground power cables," in *2000 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No.00CH37077)*, vol. 1, 2000, pp. 757–761 vol.1.

[31] M. A. Hannan and A. Mohamed, "PSCAD/EMTDC simulation of unified series-shunt compensator for power quality improvement," *IEEE Transactions on Power Delivery*, vol. 20, no. 2, pp. 1650–1656, 2005.

[32] J. Mahseredjian and F. Alvarado, "Creating an electromagnetic transients program in Matlab: MatEMTP," *IEEE Trans. Power Del.*, vol. 12, no. 1, pp. 380–388, jan 1997.

[33] R. Kuffel, J. Giesbrecht, T. Maguire, R. P. Wierckx, and P. McLaren, "RTDS-a fully digital power system simulator operating in real time," in *Proceedings 1995 International Conference on Energy Management and Power Delivery EMPD '95*, vol. 2, 1995, pp. 498–503 vol.2.

[34] P. G. McLaren, R. Kuffel, R. Wierckx, J. Giesbrecht, and L. Arendt, "A real time digital simulator for testing relays," *IEEE Transactions on Power Delivery*, vol. 7, no. 1, pp. 207–213, 1992.

[35] C. Dufour, S. Cense, V. Jalili-Marandi, and J. Bélanger, "Review of state-of-the-art solver solutions for HIL simulation of power systems, power electronic and motor drives," in *2013 15th European Conference on Power Electronics and Applications (EPE)*, 2013, pp. 1–12.

[36] C. Dufour, J. Mahseredjian, J. Bélanger, and J. L. Naredo, "An advanced real-time electro-magnetic simulator for power systems with a simultaneous state-space nodal solver," in *2010 IEEE/PES Transmission and Distribution Conference and Exposition: Latin America (T D-LA)*, 2010, pp. 349–358.

[37] "Understandingthe real-time executive daemon (rtxd) for linux," https://www.adi.com/downloads/files/Understanding-rtxd-for-Linux.pdf, accessed: 2020-09-22.

[38] "Pragmatic real-time simulation on FPGA for modern electronic systems," https://opal-rt.com/systems-efpgasim/#, accessed: 2020-09-23.

[39] "Scalexio," https://www.dspace.com/shared/data/pdf/2020/dSPACE-SCALEXIO_Product-information_08-2020_English.pdf, accessed: 2020-09-23.

[40] D. Majstorovic, I. Celanovic, N. D. Teslic, N. Celanovic, and V. A. Katic, "Ultralow-latency hardware-in-the-loop platform for rapid validation of power electronics designs," *IEEE Trans. Ind. Electron.*, vol. 58, no. 10, pp. 4708–4716, Oct. 2011.

[41] "PLECS the simulatioon platform for power electronic systems," https://www.plexim.com/sites/default/files/rtboxmanual.pdf, accessed: 2020-09-23.

[42] H. F.-Blanchette, T. Ould-Bachir, and J.-P. David, "A state-space modeling approach for the FPGA-based real-time simulation of high switching frequency power converters," *IEEE Trans. Ind. Electron.*, vol. 59, no. 12, pp. 4555–4567, December 2012.

[43] W. Li, G. Joos, and J. Belanger, "Real-time simulation of a wind turbine generator coupled with a battery supercapacitor energy storage system," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 4, pp. 1137–1145, 2010.

[44] J. Schutt-Aine, "Latency insertion method (LIM) for the fast transient simulation of large networks," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 48, no. 1, pp. 81–89, 2001.

[45] M. Milton and A. Benigni, "ORTiS solver codegen: C++ code generation tools for high performance, FPGA-based, real-time simulation of power electronic systems," *SoftwareX*, vol. 13, p. 100660, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711021000054

[46] M. Milton and A. Benigni, "Latency insertion method based real-time simulation of power electronic systems," *IEEE Transactions on Power Electronics*, vol. 33, no. 8, pp. 7166–7177, 2018.

[47] T. Kato, K. Inoue, and Y. Fujiwara, "Multirate analysis method for a power electronic system by multi-domain partitioning," in *2010 IEEE 12th Workshop on Control and Modeling for Power Electronics (COMPEL)*, 2010, pp. 1–8.

[48] T. Ould-Bachir, H. F.-Blanchette, and K. Al-Haddad, "A network tearing technique for FPGA-based real-time simulation of power converters," *IEEE Trans. Ind. Electron.*, vol. 62, no. 6, pp. 3409–3418, 2015.

[49] M. K. Namboothiripad, M. J. Datar, M. C. Chandorkar, and S. B. Patkar, "FPGA accelerator for real-time emulation of power electronic systems using multiport decomposition," *IEEE Transactions on Industry Applications*, vol. 56, no. 6, pp. 6674–6686, 2020.

[50] R. Mirzahosseini and R. Iravani, "Small time-step FPGA-based real-time simulation of power systems including multiple converters," *IEEE Trans. Power Del.*, vol. 34, no. 6, pp. 2089–2099, 2019.

[51] C. Liu, H. Bai, R. Ma, Y. Wang, and F. Gao, "Suppression of chattering in the real-time simulation of the power converter," *IEEE Transactions on Power Electronics*, pp. 1–1, 2021.

[52] H. Chalangar, T. Ould-Bachir, K. Sheshyekani, and J. Mahseredjian, "A direct mapped method for accurate modeling and real-time simulation of high switching frequency resonant converters," *IEEE Trans. Ind. Electron.*, pp. 1–1, 2020.

[53] M. Matar and R. Iravani, "FPGA implementation of the power electronic converter model for real-time simulation of electromagnetic transients," *IEEE Trans. Power Del.*, vol. 25, no. 2, pp. 852–860, April 2010.

[54] A. Myaing and V. Dinavahi, "FPGA-based real-time emulation of power electronic systems with detailed representation of device characteristics," *IEEE Trans. Ind. Electron.*, vol. 58, no. 1, pp. 358–368, 2011.

[55] G. Martin and G. Smith, "High-level synthesis: Past, present, and future," *IEEE Design & Test of Computers*, no. 4, pp. 18–25, 2009.

[56] S. Director, A. Parker, D. Siewiorek, and D. Thomas, "A design methodology and computer aids for digital VLSI systems," *IEEE Transactions on Circuits and Systems*, vol. 28, no. 7, pp. 634–645, 1981.

[57] C. Y. Hitchcock and D. E. Thomas, "A method of automatic data path synthesis," in *20th Design Automation Conference Proceedings*, 1983, pp. 484–489.

[58] P. Marwedel, "The MIMOLA design system: Tools for the design of digital processors," in *21st Design Automation Conference Proceedings*, 1984, pp. 587–593.

[59] P. G. Paulin, J. P. Knight, and E. F. Girczyc, "Hal: A multi-paradigm approach to automatic data path synthesis," in *23rd ACM/IEEE Design Automation Conference*, 1986, pp. 263–270.

[60] H. Man, J. Rabaey, J. Vanhoof, G. Goossens, P. Six, and L. Claesen, "CATHEDRAL-II-a computer-aided synthesis system for digital signal processing vlsi systems," *Computer-Aided Engineering Journal*, vol. 5, pp. 55 – 66, 05 1988.

[61] "DK4 Handel-C language reference manual," https://babbage.cs.qc.cuny.edu/courses/cs345/Manuals/HandelC.pdf, accessed: 2020-10-05.

[62] "SpecC language reference manual," http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.113.743&rep=rep1&type=pdf, accessed: 2020-10-05.

[63] "SystemC 2.0.1 language reference manual," http://homes.di.unimi.it/~pedersini/AD/SystemC_v201_LRM.pdf, accessed: 2020-10-05.

[64] M. Budiu, G. Venkataramani, T. Chelcea, and S. C. Goldstein, "Spatial computation," *SIGPLAN Not.*, vol. 39, no. 11, p. 14–26, Oct. 2004. [Online]. Available: https://doi.org/10.1145/1037187.1024396

[65] "Nios II C2H Compiler user guide," https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_nios2_c2h_compiler.pdf, accessed: 2020-10-05.

[66] "CoDeveloper from impulse accelerated technologies," https://www.yumpu.com/en/document/read/5485572/codeveloper-from-impulse-accelerated-technologies, accessed: 2020-11-12.

[67] T. Bollaert, *Catapult Synthesis: A Practical Introduction to Interactive C Synthesis*. Dordrecht: Springer Netherlands, 2008, pp. 29–52. [Online]. Available: https://doi.org/10.1007/978-1-4020-8588-8_3

[68] S. Aditya and V. Kathail, *Algorithmic Synthesis Using PICO*. Dordrecht: Springer Netherlands, 2008, pp. 53–74. [Online]. Available: https://doi.org/10.1007/978-1-4020-8588-8_4

[69] Z. Zhang, Y. Fan, W. Jiang, G. Han, C. Yang, and J. Cong, *AutoPilot: A Platform-Based ESL Synthesis System*. Dordrecht: Springer Netherlands, 2008, pp. 99–112. [Online]. Available: https://doi.org/10.1007/978-1-4020-8588-8_6

[70] P. Coussy, C. Chavet, P. Bomel, D. Heller, E. Senn, and E. Martin, *GAUT: A High-Level Synthesis Tool for DSP Applications*. Dordrecht: Springer Netherlands, 2008, pp. 147–169. [Online]. Available: https://doi.org/10.1007/978-1-4020-8588-8_9

[71] J. Villarreal, A. Park, W. Najjar, and R. Halstead, “Designing modular hardware accelerators in C with ROCCC 2.0,” in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, 2010, pp. 127–134.

[72] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, “LegUp: High-level synthesis for FPGA-based processor/accelerator systems,” in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA ’11. New York, NY, USA: Association for Computing Machinery, 2011, p. 33–36. [Online]. Available: https://doi.org/10.1145/1950413.1950423

[73] Q. Huang, R. Lian, A. Canis, J. Choi, R. Xi, N. Calagar, S. Brown, and J. Anderson, “The effect of compiler optimizations on high-level synthesis-generated hardware,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 3, May 2015. [Online]. Available: https://doi.org/10.1145/2629547

[74] “HDL Coder™ Reference Matlab & Simulink,” https://www.mathworks.com/help/pdf_doc/hdlcoder/hdlcoder_ref.pdf, accessed: 2021-04-14.

[75] L. Estrada, N. Vázquez, J. Vaquero, Á. de Castro, and J. Arau, “Real-time hardware in the loop simulation methodology for power converters using labview fpga,” *Energies*, vol. 13, no. 2, p. 373, 2020.

[76] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, “High-level synthesis for FPGAs: from prototyping to deployment,” *IEEE Trans. Comput-Aided Design Integr. Circuits Syst.*, vol. 30, no. 4, pp. 473–491, 2011.

[77] S. Sinha and T. Srikanthan, “High-level synthesis: Boosting designer productivity and reducing time to market,” *IEEE Potentials*, vol. 34, no. 4, pp. 31–35, July 2015.

[78] E. Homsirikamol and K. Gaj, “Can high-level synthesis compete against a hand-written code in the cryptographic domain? a case study,” in *International Conference on Reconfigurable Computing and FPGAs (ReConFig).* IEEE, 2014, pp. 1–8.

[79] O. Reiche, K. Häublein, M. Reichenbach, M. Schmid, F. Hannig, J. Teich, and D. Fey, “Synthesis and optimization of image processing accelerators using domain knowledge,” *Journal of Systems Architecture*, pp. –, 2015.

[80] G. Inggs, S. Fleming, D. Thomas, and W. Luk, “Is high level synthesis ready for business? a computational finance case study,” in *Field-Programmable Technology (FPT), 2014 International Conference on*, Dec 2014, pp. 12–19.

[81] O. Jiménez, O. Lucía, I. Urriza, L. A. Barragan, D. Navarro, and V. Dinavahi, "Implementation of an FPGA-based online hardware-in-the-loop emulator using high-level synthesis tools for resonant power converters applied to induction heating appliances," *IEEE Trans. Ind. Electron.*, vol. 62, no. 4, pp. 2206–2214, 2015.

[82] D. Tormo, R. Vidal-Albalate, L. Idkhajine, E. Monmasson, and R. Blasco-Gimenez, "Study of system-on-chip devices to implement embedded real-time simulators of modular multi-level converters using high-level synthesis tools," in *2018 IEEE International Conference on Industrial Technology (ICIT)*, 2018, pp. 1447–1452.

[83] E. Zamiri, A. Sanchez, M. Yushkova, M. S. Martínez-García, and A. de Castro, "Comparison of different design alternatives for hardware-in-the-loop of power converters," *Electronics*, vol. 10, no. 8, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/8/926

[84] A. Brant and G. G. F. Lemieux, "ZUMA: an open FPGA overlay architecture," in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, 2012, pp. 93–96.

[85] S. Shukla, N. W. Bergmann, and J. Becker, "QUKU: a two-level reconfigurable architecture," in *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI)*, 2006, pp. 1–6.

[86] Q. Ijaz, E.-B. Bourennane, A. K. Bashir, and H. Asghar, "Revisiting the high-performance reconfigurable computing for future datacenters," *Future Internet*, vol. 12, no. 4, p. 64, Apr 2020. [Online]. Available: http://dx.doi.org/10.3390/fi12040064

[87] D. Capalija and T. S. Abdelrahman, "A high-performance overlay architecture for pipelined execution of data flow graphs," in *2013 23rd International Conference on Field programmable Logic and Applications*, 2013, pp. 1–8.

[88] X. Li, A. Jain, D. Maskell, and S. A. Fahmy, "An area-efficient FPGA overlay using DSP block based time-multiplexed functional units," *arXiv preprint arXiv:1606.06460*, 2016.

[89] C. Dufour, S. Cense, T. Ould-Bachir, L. Grégoire, and J. Bélanger, "General-purpose reconfigurable low-latency electric circuit and motor drive solver on FPGA," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 3073–3081.

[90] J. Bélanger, A. Yamane, A. Yen, S. Cense, and P. Robert, “Validation of eHS FPGA reconfigurable low-latency electric and power electronic circuit solver,” in *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, 2013, pp. 5418–5423.

[91] T. Ould-Bachir and J.-P. David, “Performing floating-point accumulation on a modern FPGA in single and double precision,” in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, North Carolina, USA, 2-4 2010, pp. 105–108.

[92] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, “Theory of latency-insensitive design,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 9, pp. 1059–1076, Sep. 2001.

[93] L. P. Carloni, K. L. McMillan, A. Saldanha, and A. L. Sangiovanni-Vincentelli, “A methodology for correct-by-construction latency insensitive design,” in *Proceedings of the 1999 IEEE/ACM International Conference on Computer-aided Design*, ser. ICCAD ’99. Piscataway, NJ, USA: IEEE Press, 1999, pp. 309–315. [Online]. Available: http://dl.acm.org/citation.cfm?id=339492.340032

[94] M. Abbas and V. Betz, “Latency insensitive design styles for FPGAs,” in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2018, pp. 360–3607.

[95] L. P. Carloni, “From latency-insensitive design to communication-based system-level design,” *Proceedings of the IEEE*, vol. 103, no. 11, pp. 2133–2151, 2015.

[96] S. Dai, A. Klinefelter, H. Ren, R. Venkatesan, B. Keller, N. Pinckney, and B. Khailany, “Verifying high-level latency-insensitive designs with formal model checking,” *arXiv preprint arXiv:2102.06326*, 2021.

[97] K. E. Fleming, M. Adler, M. Pellauer, A. Parashar, A. Mithal, and J. Emer, “Leveraging latency-insensitivity to ease multiple FPGA design,” in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA ’12. New York, NY, USA: Association for Computing Machinery, 2012, p. 175–184. [Online]. Available: https://doi.org/10.1145/2145694.2145725

[98] M. Vijayaraghavan and Arvind, “Bounded dataflow networks and latency-insensitive circuits,” in *2009 7th IEEE/ACM International Conference on Formal Methods and Models for Co-Design*, 2009, pp. 171–180.

[99] M. R. Casu and L. Macchiarulo, "A new approach to latency insensitive design," in *Proceedings of the 41st Annual Design Automation Conference*, ser. DAC '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 576–581. [Online]. Available: https://doi.org/10.1145/996566.996725

[100] M. Milton, A. Benigni, and J. Bakos, "System-level, FPGA-based, real-time simulation of ship power systems," *IEEE Transactions on Energy Conversion*, vol. 32, no. 2, pp. 737–747, 2017.

[101] C. Liu, R. Ma, H. Bai, F. Gechter, and F. Gao, "A parallel solver to the three-level VSC modeling for HIL application," in *2018 IEEE Transportation Electrification Conference and Expo (ITEC)*, 2018, pp. 108–113.

[102] O. Faruque, T. Strasser, G. Lauss, V. Jalili-Marandi, P. Forsyth, C. Dufour, V. Dinavahi, A. Monti, P. Kotsampopoulos, J. Martinez, K. Strunz, M. Saeedifard, X. Wang, D. Shearer, and M. Paolone, "Real-time simulation technologies for power systems design, testing, and analysis," *IEEE Power Energy Technol. Syst. J.*, vol. 2, no. 2, pp. 63–73, June 2015.

[103] L. Huang, A. P. Hu, A. Swain, and X. Dai, "Comparison of two high frequency converters for capacitive power transfer," in *Energy Conv. Congress and Expo. (ECCE).* IEEE, 2014, pp. 5437–5443.

[104] D. Stepins, "An improved control technique of switching-frequency-modulated power factor correctors for low thd and high power factor," *IEEE Transactions on Power Electronics*, vol. 31, no. 7, pp. 5201–5214, July 2016.

[105] W. Zhu, K. Zhou, M. Cheng, and F. Peng, "A high-frequency-link single-phase PWM rectifier," *IEEE Trans. Ind. Electron.*, vol. 62, no. 1, pp. 289–298, 2015.

[106] T. Wang, S. Sen, and M. Amirabadi, "Soft switching high frequency AC-link DC-DC buck-boost converters," in *App. Power Electron. Conf. and Expo. (APEC).* IEEE, 2015, pp. 57–64.

[107] G. Parma and V. Dinavahi, "Real-time digital hardware simulation of power electronics and drives," *IEEE Trans. Power Del.*, vol. 22, no. 2, pp. 1235–1246, April 2007.

[108] J. Rodriguez-Andina, M. Valdes-Pena, and M. Moure, "Advanced features and industrial applications of FPGAs: A review," *IEEE Trans. Ind. Electron.*, vol. 11, no. 4, pp. 853–864, Aug 2015.

[109] F. Vahid, *Digital Design with RTL Design, Verilog and VHDL.* John Wiley & Sons, 2010.

[110] D. Navarro, O. Lucia, L. A. Barragan, I. Urriza, and O. Jimenez, "High-level synthesis for accelerating the FPGA implementation of computationally demanding control algorithms for power converters," *IEEE Trans. Ind. Informat.*, vol. 9, no. 3, pp. 1371–1379, 2013.

[111] Xilinx, *UG902 v2016.1: Vivado Design Suite User Guide – High Level Synthesis*, May 2016.

[112] Altera, *UG-OCL002 v2015.11.02: Altera SDK for OpenCL – Programming Guide*, 2015.

[113] P. Li, P. Zhang, L.-N. Pouchet, and J. Cong, "Resource-aware throughput optimization for high-level synthesis," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: ACM, 2015, pp. 200–209.

[114] Xilinx, *UG640 v11.4: System Generator for DSP – User Guide*, December 2009.

[115] L. Herrera, C. Li, X. Yao, and J. Wang, "FPGA-based detailed real-time simulation of power converters and electric machines for EV HIL applications," *IEEE Trans. Ind. Appl.*, vol. 51, no. 2, pp. 1702–1712, March 2015.

[116] O. Lucía, I. Urriza, L. Barragán, D. Navarro, O. Jimeénez, and J. Burdío, "Real-time FPGA-based hardware-in-the-loop simulation test bench applied to multiple-output power converters," *IEEE Trans. Ind. Appl.*, vol. 47, no. 2, pp. 853–860, March 2011.

[117] R. Velazquez, O. Lucia, D. Navarro, L. Barragan, J. Artigas, and C. Sagues, "Design of an FPGA-based full-state feedback controller using high level synthesis tools," in *Workshop on Control and Modeling for Power Electronics (COMPEL), 2014 IEEE 15th*, June 2014, pp. 1–6.

[118] T. Ould-Bachir and J.-P. David, "Self-alignment schemes for the implementation of addition-related floating-point operators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 6, no. 1, pp. 1–21, 2013.

[119] Xilinx, *UG470 v1.1: 7 Series FPGAs Configuration, User Guide*, 2016.

[120] OPAL-RT, "OP5707 Simulator," 2017. [Online]. Available: http://www.opal-rt.com/simulator-platform-op5707/

[121] M. Ramakrishna, V. K. Kodati, P. V. Gratz, and A. Sprintson, "GCA:Global congestion awareness for load balance in networks-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 7, pp. 2022–2035, 2016.

[122] A. Kostrzewa, S. Saidi, and R. Ernst, "Dynamic control for mixed-critical networks-on-chip," in *2015 IEEE Real-Time Systems Symposium*, 2015, pp. 317–326.

[123] A. Karkar, T. Mak, K. Tong, and A. Yakovlev, "A survey of emerging interconnects for on-chip efficient multicast and broadcast in many-cores," *IEEE Circuits and Systems Magazine*, vol. 16, no. 1, pp. 58–72, 2016.

[124] S. Thuseethan and S. Vasanthapriyan, "Spider-web topology: A novel topology for parallel and distributed computing," in *2015 3rd International Conference on Information and Communication Technology (ICoICT)*, 2015, pp. 34–38.

[125] T. yun Feng, "A survey of interconnection networks," *Computer*, vol. 14, no. 12, pp. 12–27, dec 1981.

[126] C. Cakir, R. Ho, J. Lexau, and K. Mai, "Modeling and design of high-radix on-chip crossbar switches," in *Proceedings of the 9th International Symposium on Networks-on-Chip*, ser. NOCS '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2786572.2786579

[127] L. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," in *Proceedings of the 1st Annual Symposium on Computer Architecture*, ser. ISCA '73. New York, NY, USA: Association for Computing Machinery, 1973, p. 21–28. [Online]. Available: https://doi.org/10.1145/800123.803967

[128] Chuan-Lin Wu and Tse-Yun Feng, "On a class of multistage interconnection networks," *IEEE Transactions on Computers*, vol. C-29, no. 8, pp. 694–702, 1980.

[129] Agrawal, "Graph theoretical analysis and design of multistage interconnection networks," *IEEE Transactions on Computers*, vol. C-32, no. 7, pp. 637–648, 1983.

[130] F. Bistouni and M. Jahanshahi, "Rearranging links: a cost-effective approach to improve the reliability of multistage interconnection networks," *International Journal of Internet Technology and Secured Transactions (IJITST)*, vol. 8, no. 3, pp. 336–371, 2018.

[131] F. Bistouni and M. Jahanshahi, "Pars network: A multistage interconnection network with fault-tolerance capability," *Journal of Parallel and Distributed Computing*,

vol. 75, pp. 168 – 183, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731514001476

[132] W. Kabaciński and M. Michalski, "The control algorithm and the FPGA controller for non-interruptive rearrangeable log2(n, 0, p) switching networks," in *2013 IEEE International Conference on Communications (ICC)*, 2013, pp. 3840–3845.

[133] A. S. Vijay, S. Doolla, and M. C. Chandorkar, "Real-time testing approaches for microgrids," *IEEE Journal of Emerging and Selected Topics in Power Electronics*, vol. 5, no. 3, pp. 1356–1376, Sept 2017.

[134] J. Hollman and J. Marti, "Real time network simulation with PC-cluster," *IEEE Transactions on Power Systems*, vol. 18, no. 2, pp. 563 – 569, may 2003.

[135] M. Milton, A. Benigni, and A. Monti, "Real-time multi-FPGA simulation of energy conversion systems," *IEEE Transactions on Energy Conversion*, vol. 34, no. 4, pp. 2198–2208, Dec 2019.

[136] H. Jin, "Behavior-mode simulation of power electronic circuits," *IEEE Trans. Power Electron.*, vol. 12, no. 3, pp. 443–452, May 1997.

[137] A. Hadizadeh, M. Hashemi, M. Labbaf, and M. Parniani, "A matrix-inversion technique for FPGA-based real-time EMT simulation of power converters," *IEEE Trans. Ind. Electron.*, vol. 66, no. 2, pp. 1224–1234, Feb 2019.

[138] T. Ould-Bachir, C. Dufour, J. Bélanger, J. Mahseredjian, and J. P. David, "A fully automated reconfigurable calculation engine dedicated to the real-time simulation of high switching frequency power electronic circuits," *Mathematics and Computers in Simulation*, vol. 91, pp. 167–177, 2013.

[139] Q. Mu, J. Liang, X. Zhou, Y. Li, and X. Zhang, "Improved ADC model of voltage-source converters in DC grids," *IEEE Trans. Power Electron.*, vol. 29, no. 11, pp. 5738–5748, 2014.

[140] C. Dufour, "Method and system for reducing power losses and state-overshoots in simulators for switched power electronic circuit," U.S. Patent 9,665,672, May 30, 2017.

[141] K. Wang, J. Xu, G. Li, N. Tai, A. Tong, and J. Hou, "A generalized associated discrete circuit model of power converters in real-time simulation," *IEEE Trans. Power Electron.*, vol. 34, no. 3, pp. 2220–2233, 2019.

[142] M. Milton, A. Benigni, M. Vygoder, J. Gudex, and R. Cuzner, "Power electronic system real-time simulation on national instruments FPGA platforms," in *2019 IEEE Electric Ship Technologies Symposium (ESTS)*, Aug 2019, pp. 32–38.

[143] M. Milton and A. Benigni, "Software and synthesis development libraries for power electronic system real-time simulation," in *IEEE Electric Ship Technologies Symposium (ESTS)*, 2019, pp. 368–376.

[144] S. Mahmood, P. Shydlouski, and M. Hubner, "An application specific framework for HLS-based FPGA design of articulated robot inverse kinematics," in *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec 2018, pp. 1–6.

[145] X. Guo, J. Yuan, Y. Tang, and X. You, "Hardware in the loop real-time simulation for the associated discrete circuit modeling optimization method of power converters," *Energies*, vol. 11, no. 11, p. 3237, Nov 2018. [Online]. Available: http://dx.doi.org/10.3390/en11113237

[146] E. Zamiri, A. Sanchez, A. de Castro, and M. S. Martínez-García, "Comparison of power converter models with losses for hardware-in-the-loop using different numerical formats," *Electronics*, vol. 8, no. 11, p. 1255, Nov 2019. [Online]. Available: http://dx.doi.org/10.3390/electronics8111255

[147] A. Sanchez, A. de Castro, M. S. Martínez-García, and J. Garrido, "LOCOFloat: a low-cost floating-point format for FPGAs: application to HIL simulators," *Electronics*, vol. 9, no. 1, p. 81, Jan 2020. [Online]. Available: http://dx.doi.org/10.3390/electronics9010081

[148] M. Langhammer, "High performance matrix multiply using fused datapath operators," in *Asilomar Conference on Signals, Systems and Computers*, Oct 2008, pp. 153–159.

[149] W. Gautschi, *Numerical analysis : an introduction.* MA, Boston: Birkhauser, 1997.

[150] T. D. C. Busarello, J. A. Pomilio, and M. G. Simoes, "Design procedure for a digital proportional-resonant current controller in a grid connected inverter," in *2018 IEEE 4th Southern Power Electronics Conference (SPEC)*, 2018, pp. 1–8.