Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

Enhanced Run-Time Reconfigurable Myokinetic Interface for Prosthetic Control of Artificial Hands

Davi de Alencar Mendes

DISSERTAÇÃO DE MESTRADO

PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS MECATRÔNICOS

Brasília

2024

Mendes, Davi de Alencar.
M538e Enhanced Run-Time Reconfigurable Myokinetic Interface for Prosthetic Control of Artificial Hands / Davi de Alencar Mendes; orientador Daniel Mauricio Muñoz Arboleda. -- Brasília, 2024.
87 p.

Dissertação de Mestrado (Programa de Pós-Graduação em Sistemas Mecatrônicos) -- Universidade de Brasília, 2024.

1. FPGA. 2. Zynq SoC. 3. Partial Reconfiguration. 4. Myokinetic Interface. I. Arboleda, Daniel Mauricio Muñoz, orient. II. Título

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

Aprimoramento de uma Interface Miocinética Dinamicamente Reconfigurável para Controle Protético de Mãos Artificiais

Davi de Alencar Mendes

Dissertação de Mestrado submetida ao Departamento de Engenharia Mecânica da Universidade Brasília como parte dos requisitos necessários para a obtenção do grau de Mestre

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Brasília
2024

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

Enhanced Run-Time Reconfigurable Myokinetic Interface for Prosthetic Control of Artificial Hands

Davi de Alencar Mendes

Dissertação de Mestrado submetida ao Departamento de Engenharia Mecânica da Universidade Brasília como parte dos requisitos necessários para a obtenção do grau de Mestre

Trabalho aprovado. Brasília, 19 de Julho de 2024:

Prof. Dr. Daniel Mauricio Muñoz Arboleda, UnB/FT/ENM
Orientador

Prof. Dr. Renato Coral Sampaio, UnB/FGA
Examinador interno

Prof. Dr. Vanderlei Bonato, USP/ICMC
Examinador externo

Prof. Carlos Eduardo da Silva Santos, IFTo
Examinador externo

Brasília
2024

“Para:
Meus amados pais José Ricardo & Eunice Mendes,
Querida irmã Sarah Mendes Barbosa & Ruy Barbosa,
Minha esposa Gabriela Chiesse.”

DAVI MENDES

Acknowledgements

Primeiramente, agradeço à Deus, pois toda inteligência e sabedoria provêm dEle. Aos meus pais, José Ricardo & Eunice Mendes, obrigado pelo amor e apoio incondicional, sou eternamente grato por tudo que fizeram, pela dedicação e suporte aos meus estudos, sem vocês nada disso seria possível. Tenho muito orgulho de ser filho de vocês e minha admiração por tudo que são e conquistaram é eterna. À minha irma, Sarah, que está comigo desde sempre, obrigado por acreditar no meu potencial. À minha amada esposa, Gabriela Chiesse, pelo companheirismo diário, carinho e paciência, sem os quais eu não chegaria até aqui, obrigada por me motivar a ir mais longe e estar comigo em cada etapa dessa jornada. À minha avó, Izabel, agradeço pelo amor diário e por ser uma figura essencial em nossa família - esse trabalho também é para a senhora.

Ao meu Professor Orientador Daniel Muñoz, agradeço pelas aulas na graduação que me fizeram perceber o excelente docente que é, pelo amparo imensurável durante todo o mestrado, pelas palavras de encorajamento e, principalmente, por confiar em mim para realizar esse trabalho. Sou eternamente grato à todas as oportunidades que me deu, carregarei com carinho tudo o que fez por mim.

Por fim, agradeço a CAPES e a UnB pelo suporte financeiro dedicado ao desenvolvimento da pesquisa.

Abstract

In recent years, FPGAs have become more popular in embedded systems, both as a main computation resources and as hardware accelerators. Partial Run-Time Reconfiguration of FPGAs is a compelling design concept for general purpose reconfigurable systems for its flexibility and extensibility. In the recent literature research for the Myokinetic Control Interface – a Human-Machine Interface (HMI) based on implanted magnets and gross muscular motion – embedded solutions have been proposed to the development of a self-contained transcutaneous magnet localizer system. Previous works show that, when designed properly a magnet localizer system can improve its performance with reduced logic utilization taking advantage of a partial run-time reconfigurable architecture. Unfortunately, there are problems to face: the reconfiguration overhead is not negligible compared with nowadays CPUs performance, and elevated power consumption. In an attempt to cover this gap, here we present a novel implementation for a run-time reconfigurable architecture used to estimate magnet displacement using data-driven models implemented as hardware accelerators capable of tracking five magnets. The architecture is implemented on both AMD Xilinx FPGA and SoC-FPGA devices aimed at reducing the reconfiguration overhead with the pre-fetching (*hardware pipelining*) of reconfigurable modules (RM). The system exhibits short execution time (6.67 ms – SoC-FPGA, and 5.97 ms – FPGA) and power consumption (1.635 W – SoC-FPGA, and 1.203 W – FPGA). The system proved able to localize magnets with high accuracy (RMSE ranging from $\approx$ 0.076 mm to $\approx$ 0.043 mm). The obtained reconfiguration throughput ($\sim$ 399 MB/s) using the internal configuration port is considered optimal. In conclusion, we demonstrated the design and implementation of an enhanced run-time reconfigurable architecture applied to myokinetic interface paving the way towards the development of a self-contained FPGA-based magnet localizer.

Keywords: FPGA. Zynq SoC. Partial Reconfiguration. Myokinetic Interface.

Resumo

Nos últimos anos, FPGAs tem se tornado mais populares em sistemas embarcados, tanto como principais recursos computacionais quanto como aceleradores em hardware. A Reconfiguração Dinâmica Parcial é um conceito de projeto atraente para sistemas reconfiguráveis de proposito geral devido à sua flexibilidade e extensibilidade. Na literatura acadêmica recente a respeito da Interface Miocinetica – uma Interface Homem-Máquina (IHM) baseada em ímãs implantados e movimento muscular (contração e elongamento) – soluções embarcadas tem sido propostas para o desenvolvimento de um sistema localizador de ímãs transcutâneo autônomo. Trabalhos anteriores mostram que, quando projetado adequadamente, um sistema localizador de ímãs pode melhorar seu desempenho com utilização reduzida de lógica, ao adotar uma arquitetura reconfigurável em tempo de execução. Infelizmente, há problemas a serem enfrentados: a sobrecarga de reconfiguração não é negligenciável em comparação com o desempenho das CPUs atuais, e o consumo elevado de energia. Em uma tentativa de cobrir essa lacuna, apresentamos aqui uma nova implementação para uma arquitetura reconfigurável em tempo de execução usada para estimar o deslocamento de ímãs utilizando modelos orientados por dados implementados como aceleradores de hardware capazes de rastrear cinco ímãs. A arquitetura é implementada em dispositivos AMD Xilinx FPGA e SoC-FPGA visando reduzir a sobrecarga de reconfiguração com *pre-fetching* (*hardware pipelining*) de módulos reconfiguráveis (MR). O sistema apresenta reduzido tempo de execução (6,67 ms – SoC-FPGA, e 5,97 ms – FPGA) e consumo de energia (1,635 W – SoC-FPGA, e 1,203 W – FPGA). O sistema demonstrou ser capaz de localizar ímãs com alta precisão (RMSE variando de $\approx$ 0,076 mm a $\approx$ 0,043 mm). A taxa de transferência de reconfiguração obtida ($\sim$ 399 MB/s) utilizando a porta de configuração interna é considerada ideal. Em conclusão, demonstramos o projeto e a implementação de uma arquitetura reconfigurável em tempo de execução aprimorada aplicada à interface miocinética, abrindo caminho para o desenvolvimento de um localizador de ímãs autônomo baseado em FPGA.

Palavras-chave: FPGA. Zynq SoC. Partial Reconfiguration. Myokinetic Interface.

List of Figures

List of Tables

Contents

1 Introduction

1.1 Problem Description

Restoring dexterous motor functions equivalent to those of the human hand, after amputation, is considered one of the major goals in rehabilitation engineering and applied neuroscience. Reaching this goal requires a successful execution of the two key components of a prosthesis: the artificial hand and the Human-Machine Interface (HMI). The artificial hand must be capable of movements and graps comparable to those of the natural hand, while the HMI should allow effortless control of such movements, bridging the artificial hand and the sources of volition [1].

Currently, sEMG (Surface Electromyography) is an extensively used signal in biomedical HMI applications for prosthetic control (ESPOSITO et al., 2021), for it is easy to access and provides an intuitive control strategy to reproduce the function of a biological limb. This biopotential contains information about neural signals transmitted from the brain to the muscles to perform a motor task. Hence, it allows for capturing the subject's movement intention. EMG-controlled artificial limbs are referred to as myoelectric prostheses ranging in invasiveness (from surface sensors to intra-neural electrodes). Despite its widespread usage, myoelectric prosthetic control suffers from fundamental issues regarding signal acquisition and the lack of accessible independent control sources for the realization of simultaneous control of multiple DoFs (Degrees of Freedom) (ESPOSITO et al., 2021).

An innovative solution abandoning the paradigm of transducing electrical signals was proposed recently for hand prosthetic control by inferring muscles contractions from the deflection of permanent magnets implanted into the amputee's residual muscles, the Myokinetic Control Interface (TARANTINO et al., 2017) (GHERARDINI; MASIERO, et al., 2023). For that, an array of 3-axis magnetic field sensors disposed around the forearm exploits the magnetic field variations caused by the implanted magnetic markers (MMs) displacement (elongation, and contraction). The study of (MORADI et al., 2022) presents the first clinical implementation of implanted magnetic markers for an amputee's prosthetic hand. In their proposed approach, the magnetic field signals directly drive the prosthesis hand skipping the magnet localization step. This is done by detecting gesture types and grades of motion using artificial neural networks and convolutional neural networks. More recently, the first-in-human implementation of a self-contained myokinetic interface was implemented: a transradial amputee received the implantation of six magnets and was able to control a robotic hand following direct and pattern recognition control strategies exploiting

[1] i.e. the exercise of choosing or willing; a state of choice

magnet localization data, derived through numerical approximation methods (CIPRIANI et al., 2023).

Indeed, to estimate the magnet's displacements, two main strategies can be adopted: (a) to solve the inverse problem of magnetostatics, which commonly is addressed by optimization solvers (BRUCKNER et al., 2017) and requires numerical approximations as reported in (TARANTINO et al., 2017) (CLEMENTE et al., 2019) and (IANNICIELLO, 2024); (b) to use data-driven methods and machine learning (ML) models for mapping experimental magnetic measurements to the magnet's displacements (MENDEZ, 2021; MENDEZ et al., 2022).

In (TARANTINO et al., 2017), magnets were analytically modeled as point dipoles, and localization was obtained through the Levenberg–Marquardt (LM) optimization algorithm. The output of the optimization problem is a functioning solution to the inverse problem of magnetostatics. However, the LM algorithm does not provide a fixed execution time, which represents a major drawback for real-time constraints (ZHOU et al., 2019). The work of (TARANTINO et al., 2017) utilized a tracking algorithm that was ineffective in providing estimations as fast as the sensor acquired information. More specifically, while sensors could acquire ∼75 samples per second (one sample every 13 ms), 45 ms were needed for localizing four magnets. In (CLEMENTE et al., 2019), a fully embedded system was presented, which proved capable of tracking up to five magnets in less than ∼4 ms using 32 magnetic field sensors with a power consumption of 980 mW (550 mW for the acquisition unit and 430 mW for the computational unit). In this case, the time needed for sampling the sensors readings, and sending them to the computational unit was ∼24 ms.

The elevated data transfer latency limitation was recently addressed with the design of a new modular, parallel architecture capable of acquiring synchronized samples from one acquisition unit (sampling rate of 100Hz) up to eight acquisition units (sampling rate of 38Hz), significantly increasing the system output rate (IANNICIELLO, 2024). Despite these advances in system bandwidth, limitations on the reliability of the localization output, employed to control the prosthesis, emerged during the first-in-human study of the system (CIPRIANI et al., 2023). In fact, when using localization data to train pattern recognition algorithms, the repeatability of the feature set may be affected by the accuracy of the localization algorithm, based on numerical approximation methods, which can deteriorate due to several factors. For instance, factors intrinsic to the optimization technique, such as the presence of local minima in the solution or the need for user-defined initial conditions, can induce localization errors and thus variability in the feature set. The latter aspect posed challenges for developing robust classifiers during the in-human pilot study (CIPRIANI et al., 2023).

To overcome these drawbacks, here we suggest to exploit data-driven algorithms in place of numerical approximation methods for deriving magnet displacement. The work

of (MENDEZ et al., 2022) demonstrated the feasibility of using data-driven methods implemented on hardware to retrieve the position of a single magnet, to control a hand prosthesis. Field Programmable Gate Arrays (FPGAs) were used to obtain hardware parallel architectures for two regression models: a) Linear Model, and b) Radial-Basis Function Neural Network model (RBFNN). Ad-hoc solutions with 8 parallel operators for the linear model and 8 parallel neurons for the RBFNN model were obtained, being able to estimate the magnet displacement in 4.8 μs and 12.07 μs for the linear and RBFNN models, respectively. Although both models performed with good accuracy, the RBFNN model performed with higher precision than the linear one, as repeatability of the measurements is the most important characteristic of the myokinetic interface for prosthetic control. However, the main drawback of the RBFNN model is its extensive hardware occupation, with a utilization of near 95% of the Look-up Tables (LUTs) for a ZYNQ-7020 SoC-FPGA device, dissipating a power of 209 mW and 545 mW for the linear and RBFNN models, respectively.

In (MENDEZ, 2021), a Partial Run-Time Reconfigurable (RTR) SoC-FPGA architecture was exploited to improve the efficiency of the aforementioned system (MENDEZ et al., 2022) both in terms of performance and energy. The proposed RTR FPGA architecture implemented the Linear Model, RBFNN Model, and a Multilayer Perceptron Model (MLP, akin to a feedforward artificial neural network) to derive magnet displacements. Regarding data processing, the magnetic sensor data was further processed into a reduced set of features using principal component analysis (PCA). In addition, a single reconfiguration partition (RP) (placed in an entire device clock region) provides the reconfigurable slot for the hardware accelerators. The SoC-FPGA architecture is capable of tracking five magnets within $\sim$ 9.5 ms of total execution time requiring two reconfiguration steps (reconfiguration time of $\sim$ 4.74 ms). The total power consumption was estimated in 1719 mW, of which: 1514 mW are dedicated to ARM Cortex A9 Core, and 162 mW are dissipated by the programmable logic (PL). Despite of its advantages, the RTR-based solution has some shortcomings when it comes to power consumption and reconfiguration overhead. The hard-core ARM processor is power demanding (over 88% of total power consumption) and the usage of a single large RP makes so that partial bitstreams are large in size (over 1 MB) with considerable dedicated reconfiguration time impacting the overall system performance.

1.2 Research Questions

In this context, the present work is dedicated to answering the questions:

- What are the advantages, limitations and peculiarities for the implementation of a data-driven magnet tracking system for the Myokinetic Control Interface with run-time reconfiguration capabilities on a FPGA platform and on a SoC-FPGA platform?

- What guidelines (constraints, design flow, design methods) and optimizations must be used to assure that RTR architecture meets/exceeds previous works in power efficiency and hardware utilization, mitigating the reconfiguration overhead?

These key points represents the core of the presented work and could contribute to the research in this field by providing technological and scientific advancements in a core component for the Myokinetic Control Interface using reconfigurable hardware.

1.3 Objectives

This work's purpose is to develop a run-time reconfigurable (RTR) architecture on FPGA and SoC-FPGA embedded platforms for a Myokinetic Human-Machine Interface (HMI) used for prosthetic limbs (i.e. robotic hands). In this proposal, the tracking of multiple magnets is done by hardware accelerators based on data-driven models with adaptive capabilities using dynamic partial reconfiguration (DPR).

1.3.1 Specific Objectives

In the proposal, the specific objectives are to:

- Devise a hardware implementation at the RTL level capable of processing myokinetic magnetic sensor data for the localization of multiple magnets aimed at the control of prosthetic limbs. It should be capable of estimating the displacement of five MMs (magnetic markers) *virtually implanted* in an anatomically relevant forearm mockup used for the experimental data acquisition.
- Demonstrate the RTR architecture for the myokinetic magnet tracking in FPGA and SoC-FPGA embedded platforms. This is done to exploit optimizations on the DPR hardware implementation to pursue cost-effective and power-efficient solutions considering real-time operational constraints and reduced logic utilization.
- Characterize the embedded implementation according to its localization accuracy, power consumption and total execution time, aiming at reducing the reconfiguration time overhead by implementing a more efficient DPR architecture with hardware pipelining capabilities.

1.4 Contributions

Within the research work, there are three types of expected contributions, namely: (i) Scientific, (ii) Technological, and (iii) Academic contributions.

- Scientific:
  - Implementation of a novel FPGA-based real-time embedded system for the Myokinetic Control Interface for multi-magnet localization with dynamic partial reconfiguration and hardware pipelining (pre-fetching) capabilities. It is also intended to provide a SoC-FPGA implementation to assess advantages and limitations in the novel implementation.
- Technological:
  - Design & Development of an embedded RTR system under FPGA and SoC-FPGA targets with reusable components across implementations using hardware description language (HDL) and High-Level Synthesis (HLS) for the RTL implementation. If design goals are attained, the implementation will be more power efficient with increased performance and accuracy.
  - Improvements to the IP-Cores from LEIA Laboratory used in the implementation of Linear Regression and Radial-Basis Function Neural Network.
- Academic:
  - Continue the ongoing collaboration with the Biorobotics Institute at the Sant'Anna School of Advanced Studies about the MYKI Project.
- Journal Articles
  - **Run-Time Reconfiguration for Tracking Multiple Magnets with a Myokinetic Interface**. Sergio A. Pertuz, Davi A. Mendes, Marta Gherardini, Daniel M. Muñoz, Helon Vicente Hultmann Ayala and Christian Cipriani. *IEEE Transactions on Medical Robotics and Bionics*. Submission Date: July./2024. Status: Publication Accepted.
- Conference Papers
  - **A Comparative Analysis of HDL and HLS for Accelerating Machine Learning based Strain Estimation with Ultrasonic Guided Waves**. Davi A. Mendes, Gabriel Reves, M. A. Pastrana, Pedro H. Domingues, Helon V. H. Ayala, Alan C. Kubrusly, Daniel M. Muñoz and Carlos H. Llanos. *XIII Brazilian Symposium on Computing Systems Engineering*. Submission Date: July/2023. Status: Published (DOI: 10.1109/SBESC60926.2023.10324053).
  - **Implementation of a PID Controller using Online Tuning Applied to a Mobile Robot Obstacle Following/Avoidance**. M. A. Pastrana, L. H. Oliveira, D. A. Mendes, D. L. Silva, J. Mendoza-Peñaloza and Daniel M. Muñoz. *2024 Argentine Conference on Electronics (CAE)*, Submission Date: September/2023. Status: Published (DOI: 10.1109/CAE59785.2024.10487152)

– **Teaching Control Theory using Mobile Robot Obstacle Following/Avoidance with CoppeliaSim and MFO Algorithm.**. M. A. Pastrana, J. Bautista, J. Mendoza-Peñaloza, L. H. Oliveira, D. A. Mendes and Daniel M. Muñoz. *2023 Latin American Robotics Symposium (LARS), 2023 Brazilian Symposium on Robotics (SBR), and 2023 Workshop on Robotics in Education (WRE)*. Submission Date: July/2023. Status: Published (DOI: 10.1109/LARS/SBR/WRE59448.2023.10333042).

1.5 Document Organization

The work is divided into five chapters including the Introduction from Chapter 1. Chapter 2 dedicated to the research theoretical foundations regarding FPGAs, SoC-FPGA and its design flows and design methodologies. The Myokinetic Interface literature research is also presented in Chapter 2. Chapter 3 presents the development of the proposed run-time reconfigurable architecture from the experimental modeling used for training the machine-learning models used in estimating magnet displacement to their implementation at the RTL-level. In addition, we also present the implemented design for the DPR system for the FPGA and SoC-FPGA platforms. Chapter 4 presents the obtained results of the SW/HW co-design implementation in both the SoC-FPGA and FPGA platforms. The analysis includes the design utilization for the proposed static and reconfigurable logic and a system performance evaluation metrics based on in-target execution results. Lastly, 5 presents the final remarks for the report with suggestions for future works.

2 Theoretical Foundation

This chapter provides an overview about the tools and methods used in the development/research of reconfigurable hardware. The focus is on the relevant aspects of FPGAs & SoC-FPGAs, such as the design flow, toolchain and Dynamic Partial Reconfiguration (DPR) as a mean of implementation of RTR-based systems and reconfiguration strategies and their technological aspects.

In addition, the chapter also provides an overview about the state-of-the-art regarding the Myokinetic Control Interface (HMI) and the machine learning algorithms used to devise the magnet tracking scheme used in the hardware implementation.

2.1 FPGA & SoC-FPGA

Field Programmable Gate Arrays (FPGAs) are Integrated Circuits (ICs) that can be reprogrammed after fabrication with strong spatial (parallel) computing style similar to Application Specific Integrated Circuits (ASICs). ASICs are, in general, more efficient than instruction flow processors, which are oriented towards a temporal computing style (sequential). Reconfigurable hardware combines efficiency and flexibility for the realization of a multitude of complex applications. These applications are often constrained by power consumption and performance requirements with the need to adaptation in different operating environments. As ASICs may lack the desired flexibility and might pose a development overhead due to its increased design complexity, FPGAs arise as a viable option for modern engineering applications. In order to achieve higher levels of integration, security and reliability System-On-Chip FPGAs with on-chip fixed-function processing subsystems have recently emerged as potential contenders for high-end processing applications. SoC-FPGAs combine the advantages of hardware programmability with the software programmability found in modern processing cores.

The primary function of the FPGA is to implement programmable logic, which can be used by end customers to create new hardware devices. In its essence, FPGAs are built around an array of programmable logic block embedded in a sea of programmable interconnect. The array is often referred to as the programmable logic fabric (or *fabric* for short) (CHURIWALA, 2016). Figure 1 shows a basic architecture of an FPGA.

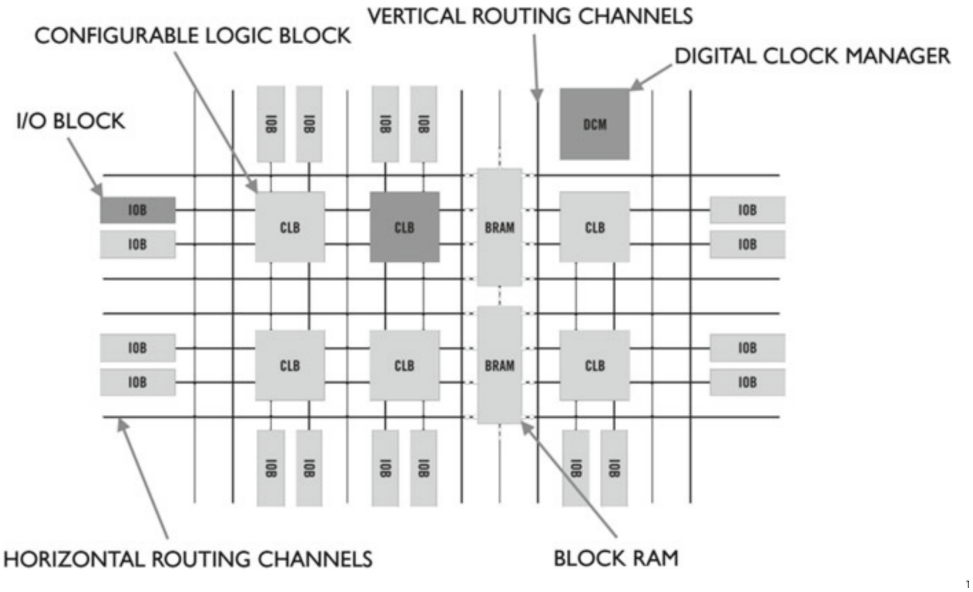


Figure 1 – Basic FPGA Architecture. Adapted from (CHURIWALA, 2016).

As the FPGA market matured, it became clear that special functionality needed its own dedicated functions built from hard gates, such as: carry chains, block RAM, DSP blocks, transceivers and others. This hardening improves not only cost but also frequency substantially.

Within any one FPGA family, devices will share a common fabric architecture with a different amount of programmable logic. This enables users to match their logic requirements and application I/O requirements with the available options for device packages. FPGA devices also provide options for multiple speed grades and temperature grades and voltage levels. The highest speed devices are typically 25% faster than lower speed devices (CHURIWALA, 2016).

Modern FPGAs commonly operate at 100-500 Mhz. In general, most logic designs run in the mid-frequency range. The highest frequency designs are typically DSP designs with extensive BRAM blocks. A logic design might exploit multiple clock domains across a wide frequency range depending on its purpose and clocking requirements.

In the following, we present a high level overview of FPGA architectures and the associated design flow for FPGA designs.

Programmable Interconnect

In order to enable arbitrary logic networks, a set of wires is woven through the FPGA fabric. The interconnect wires architecture varies from generation to generation and is hidden from the user by the tools.

Programmable Logic Block

An array of programmable logic blocks are embedded into the programmable interconnect – called CLBs (configurable logic blocks) in Xilinx devices. Today, a single logic block consists of one or more programmable logic functions implemented as a 4-6 bit configurable lookup table (LUT), a configurable carry chain, and configurable registers. These *configurable* hard blocks can be configured through the FPGAs configuration memory to be used in user's defined logic (CHURIWALA, 2016).

The combination of a LUT, carry chain, and register is called a *logic cell*, which is a standard unit for measuring FPGAs capacity.

Memory

The spatial computing inherent to FPGAs is powered by dataflow and memory. Programmable logic designs commonly use a combination of local memories embedded in the FPGA fabric and external DDR memories. Within the logic fabric, memory can be implemented as discrete registers, shift registers, distributed RAM, or block RAM. System memory access to external DDR memory is done via a bus interface which is usually an AXI protocol internal to the FPGA.

In general, registers are flip-flops – used for status and control datapaths, pipelining, and shallow (1-2 deep) FIFOs. Shift registers are commonly used for delay buffer elements and pipeline balancing in DSP designs.

DSP Blocks

Modern FPGAs contain discrete multipliers to enable efficient DSP processing. These applications are built using pipelines or flow graphs of DSP operations and data streaming. Pipelines are implemented for optimal throughput to compute at every clock cycle. Xilinx FPGAs contain a DSP block known as a DSP48, which supports an 18-bit $\times$ 25-bit multiplier, a 48-bit accumulator, and a 25-bit pre-adder (UG479…, 2018). In addition, up to four levels of pipelining can be supported for operating up to 500 MHz. The DSP48 supports integer math and the implementation of a 32-bit floating-point multiplier requires two DSP48 blocks and several hundred LCs.

2.1.1 FPGA Design Flow

Figure 2 presents the design flow with FPGAs with a high-level summary of a more traditional design flow for a hardware-only system. The individual design steps are as follows:

1. **System Design Entry**: This step comprises a functional description using design sources (RTL, Block Diagram, IP, Netlist) in which basic design constraints are taken into account for area, performance and I/O for instance. Behavioral logic simulation is used in analysis and verification of the design and allows for functional iterations on the design for resource tuning.

2. **Implementation**: Implementation step uses the output from the top-down synthesis of the overall RTL design to place and route the netlist onto the available device resources of the target part. The EDA tool works to satisfy logical, physical, and timing constraints of the design.

3. **Design Verification**: This step is shared with other design steps as verification is spread out and produces valuable insights to optimize, validate and modify the design at each stage of the process. Behavioral and structural logic simulation are used to verify the correctness of the design across the process.

4. **Hardware Bring-Up Validation**: In this phase, in-circuit validation is performed using the bitstream generated from the implementation results. This bitstream is used by the target part configuration logic to program the device in hardware debug and validation tasks.

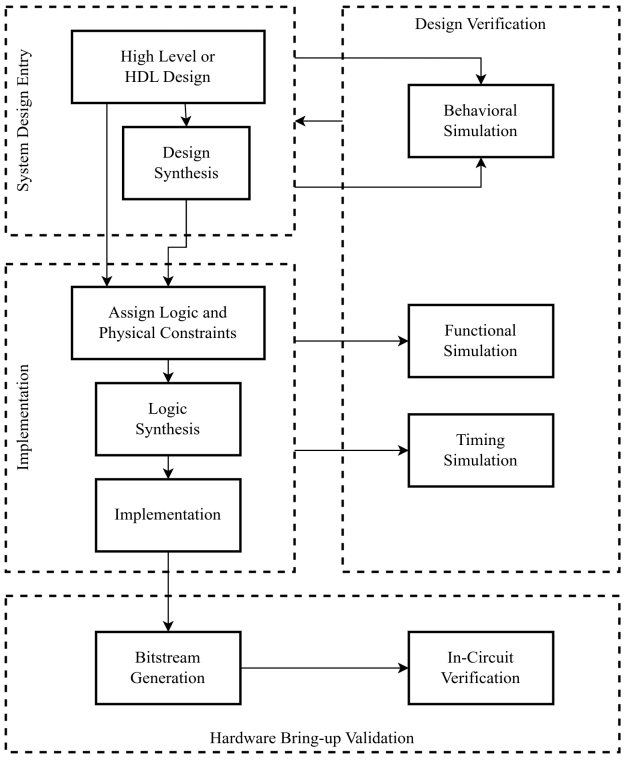


Figure 2 – Design Flow for FPGA Hardware System Development. Adapted from (UG892…, 2022).

Often, FPGA designs are developed based on high level functional specifications. These high level specifications might not cover key features such as timing, power, and

size of the design. Despite not being an explicit part of the design flow shown in Fig. 2, the identification of functional requirements, definition & refinement of performance and design constraint requirements are an essential process input to the system design entry stage.

2.1.2 HLS Design Flow

High-Level Synthesis (HLS) is the process of compiling a software description (e.g., in C or C++ or OpenCL) into a digital circuit. HLS aims to increase designer productivity by allowing a higher abstraction level that eases and shortens the hardware design process (SIMPSON, 2015). While this has enabled a wider audience to target spatial computing architectures, the optimizations principles known from traditional software design are no longer sufficient to implement high-performance code, due to fundamentally distinct aspects of hardware design (FINE LICHT et al., 2021).

The HLS converts a pragma-assisted procedural description to a functional equivalent behavioral description in a HDL such as Verilog or VHDL. This requires mapping variables and operations to corresponding constructs, the scheduling operations according to their inter-dependencies. A central task in HLS is to transform the untimed representation into a timed representation such that the throughput requirements are satisfied, which for pipelined sections might require the circuit to accept a new input every cycle. Coarse-grained control flow is implemented with state machines, while computations and fine-grained control flow are organized in pipelines.

Most effort invested by an HLS programmer lies in guiding the scheduling process to implement deep, efficient pipelines, but logical synthesis is considered when choosing data types and buffer sizes, and post-implementation place & route can ultimately bottleneck applications once the desired parallelism has been achieved, requiring the developer to adapt their code to aid this process (CONG et al., 2011).

HLS tools typically rely on functional verification of a particular circuit through hardware simulation and software/hardware cosimulation. However, performing exhaustive hardware simulations may become unfeasible or time-consuming as designs increase in complexity. Furthermore, the lack of formal proof on the correctness of particular synthesis steps and the resulting RTL modules prevents the adoption of HLS in domains where design iterations are significantly more expensive (CONG et al., 2011).

Figure 3 shows the AMD Xilinx Vitis HLS Design Flow. This HLS framework supports C and C++ to generate RTL modules packaged as IP-Cores for the Vivado Design Suite. It is possible to use optimization directives to modify and control the implementation of the internal logic and I/O ports.

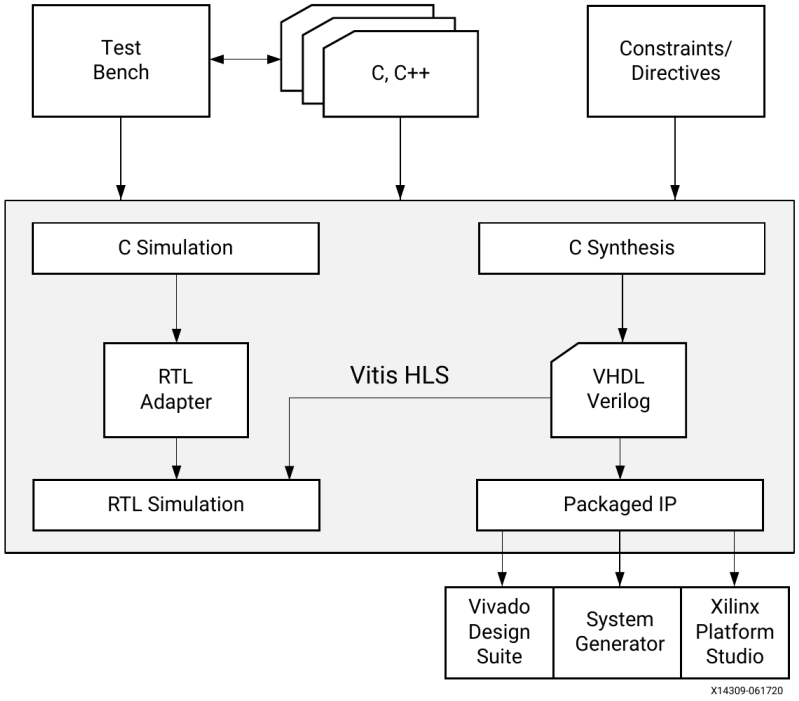


Figure 3 – Vitis HLS Design Flow. Adapted from (UG1399…, 2022).

Vitis HLS inputs include (UG1399…, 2022):

- Primarily, C functions (C and C++11/C++14).
- Design Constraints with specifications for: clock period, clock uncertainty, and the device target.
- Optional Directives for the implementation of specific behaviors, such as: AXI interface configuration, dataflow, and others.
- C test bench and associated files to simulate the C function prior to synthesis, and to verify the RTL output using C/RTL Co-simulation.

The following are Vitis HLS Outputs (UG1399…, 2022):

- Primarily, RTL implementation files in HDL formats available in both Verilog, and VHDL standards. These can be synthesized and implemented into Xilinx devices using the Vivado Design Suite.
- Reports generated as a result of simulation, synthesis, C/RTL co-simulation, and generating output.

2.2 Partial Reconfiguration

FPGA technology provides the flexibility of programming and reprogramming a device in the field without the need to go through re-fabrication. Dynamic Partial Reconfiguration (DPR) takes this one step further, allowing the dynamic modification of *part* of an operating FPGA design without impacting the rest of the design.

Any system with functions that can be time-multiplexed stands to benefit from taking advantage of Partial Reconfiguration. However, that does not mean that DPR can be applied into any FPGA design with ease or without major changes to the architectural design. DPR allows functions to be switched on hardware, similar to a microprocessor's ability to switch between tasks in software.

Hardware coprocessing is achieved by off-loading compute-intensive functions from the central processor to a dedicated hardware, executing with lower power and latency. Having dedicated hardware for each function is an inefficient use of resources. Partial reconfiguration allows a *library* of hardware functions to be partially reconfigured onto the same set of FPGA resources on-demand.

DPR-based designs consists of three basic parts. The Static is the portion of the design that is expected to function at all times and does not possess run-time reconfigurability. The Reconfigurable Partition (RP) is the instance or level of hierarchy within multiple Reconfigurable Modules (RMs) are defined and implemented. Each Reconfigurable Module represents one of the time-multiplexed functions (hardware co-processors) that will be switched in and out of the FPGA (Figure 4).

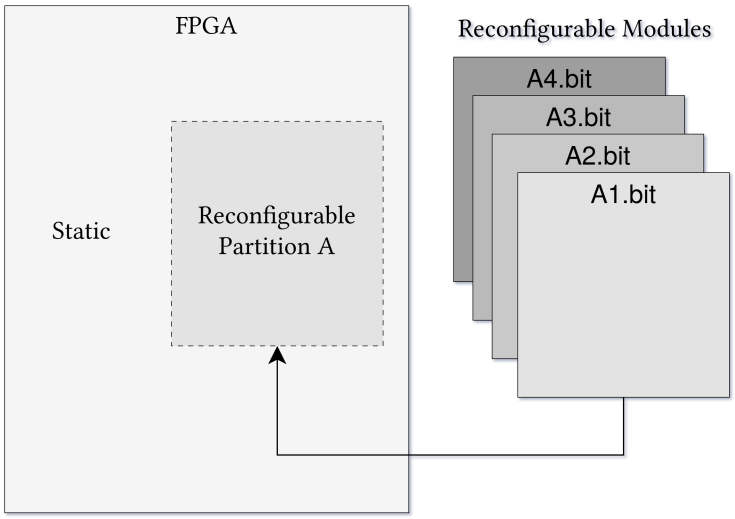


Figure 4 – Dynamic Partial Reconfiguration – Single RP scheme: concept and terminology.

DPR-based designs can contain one or more RPs, each of which must occupy a mutually exclusive physical area (programmable block) of the FPGA determined during the floorplanning. The programmable block must contain the aggregated resources required to

individually implement each of the RMs associated with it. In each device family, different resource types and granularity of the physical area can be reconfigured dynamically.

Partial bitstreams must be generated for each RM in each RP as well as a full bitstream which contains the data for both the static and the RMs being implemented. The full bitstream is used for initial configuration of the FPGA, while partial bitstreams can be loaded via the FPGA external configuration ports or via the internal configuration ports which can be incorporated into the static portion of the design.

In order to take full advantage of the potential benefits of Partial Reconfiguration for a given application, the FPGA device family; design structure modeling and support functions for DPR must be considered prior to starting the design.

2.2.1 PR Design Flow

The PR tool flow involves a number of simple steps (Fig. 5):

1. Synthesize the Static region with RPs as black boxes.
2. Synthesize each RM separately in a bottom-up scheme (out-of-context – OOC). This allows this portion to be stitched into the rest of the design at a later stage.
3. Create a physical area constraint (programmable block) to define the RP for each RM. This area should contain all the resources required for each of the RMs and its routing.
4. Implement the static logic with one RM per RP with a fully routed design.
5. Lock the static placement and routing.
6. Add a different RM to static-only design to each RP, implement, and store the fully routed design. Repeat for multiple configurations of all RMs.
7. Run partial verification utility and DRC on every routed design.
8. Generate bitstreams for each configuration with full bitstreams and partial bitstreams for each RM.

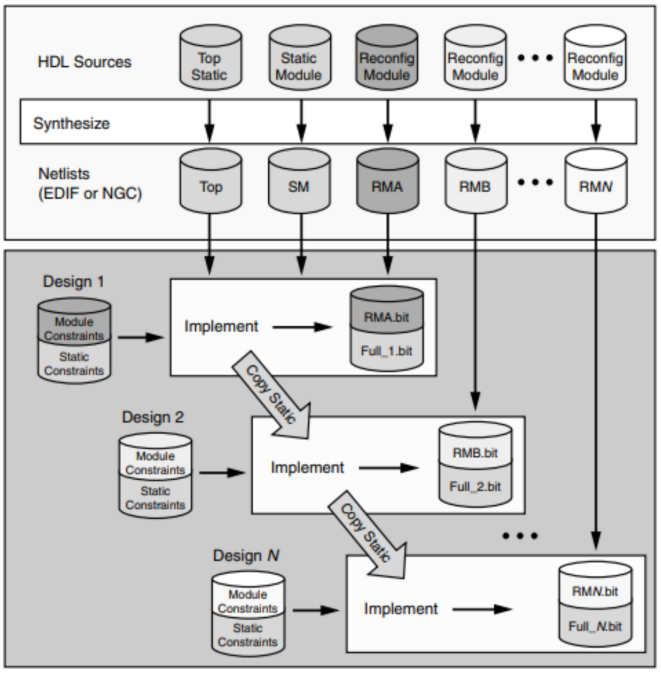


Figure 5 – Partial Reconfiguration Design Tool Flow. Adapted from (UG947…, 2022).

Static/Reconfigurable Decoupling

The ports of the instance will be the partition pins of the RP. These must be the union of the pins of all the RMs associated with that RP. These partition pins are used to bridge the static/reconfigurable areas of the design and must be decoupled when loading a new RM into the RP as unknown logic states can propagate across the static logic causing issues.

The decoupler must be driven from the static logic to provide stable inputs during partial reconfiguration. When considering bus interfaces, the decoupler might need to implement additional functionality to deal with handshakes, pending transfers. The following signals must be considered for decoupling:

- All the control signals generated from the RPs.
- In the event of the RM being loaded without a full reset before an operation, all control signals into the RP must be decoupled.
- Clock inputs must be decoupled if the RM contains logic that can initialized without being qualified by a decoupled control signal.

2.2.2 Configuration Management

Storing and managing partial bitstreams is a major concern in a DPR design. Storage of partial bitstreams is typically outside the FPGA, either on a nonvolatile flash memory, on high-speed DDR SDRAM volatile memory, or other data transfer protocol (PCIe, Ethernet).

Managing these partial bitstreams often requires a processing unit (hard-core or soft-core processor) or dedicated logic into the static region of the FPGA.

Depending on the location of the partial bitstreams and the management engine used, selected configuration ports can be used to configure the FPGA on AMD Xilinx devices, namely:

- ICAP (Internal Configuration Access Port): Preferable choice for configuration management internally to the FPGA. Requires a reconfiguration controller and dedicated logic to drive the ICAP interface.
- PCAP (Processor Configuration Access Port): The primary configuration mechanism for ZYNQ-7000 SoC FPGA designs.
- MCAP (Media Configuration Access Port): Provides access to configuration memory from PCIe block on UltraScale devices.
- JTAG: For test and debug.
- Slave SelectMAP: Mostly used to perform full and partial reconfiguration, especially when using an external processor.

On AMD Xilinx 7-Series, the internal configuration access port (ICAP) clock frequency is limited up to 100 MHz with a 32-bit data path (UG909…, 2022). Given a theoretical bandwidth of 400 MB/s, it should not be a bottleneck in many scenarios. Nevertheless, there are some applications that requires high reconfiguration throughput to sustain delivering large partial bitstreams in a timely manner. In order to achieve the maximum physical limit of ICAP, the data storage and reconfiguration controller must be designed to operate within the maximum bandwidth.

2.2.3 DPR Infrastructure: Xilinx Dynamic Function eXchange (DFX)

The AMD Xilinx Dynamic Function eXchange (DFX) is a comprehensive solution for dynamic partial reconfiguration on Xilinx devices. These IP-Cores are available to assist users in quick and easily implementing key aspects of a reconfigurable design (UG909…, 2022).

These Dynamic Function eXchange IPs are:

- **Dynamic Function eXchange Controller**: The DFX Controller provides management functions for self-controlling partially reconfigurable designs. Intended for enclosed systems where all of the RMs are known to the controller. Supports hardware and software triggers in addition to the AXI4-Lite register control interface. Bitstreams

are fetched from an AXI4 bus, and as result, is not tied to any particular storage device. Also, provides additional functionality to enable decoupling and reset after reconfiguration.

- **Dynamic Function eXchange Decoupler**: The DFX Decoupler can be used for managing the boundary between the static logic and an RP during reconfiguration. It can be customized for the number of interfaces, decoupling functionality, status and control via AXI4-Lite interface.
- **Dynamic Function eXchange AXI Shutdown Manager**: The DFX AXI Shutdown Manager can be used to make AXI interfaces between a RP and the static logic safe during reconfiguration as failures in AXI transfers could cause system deadlock. It is used to handle the termination of transfers, for the RM might not be able to complete them.
- **Dynamic Function eXchange Bitstream Monitor**: The DFX Bitstream Monitor can be used to identify partial bitstreams as they flow through the design. This information can be used for debugging or system applications such as blocking bitstream loads.

DFX Guidelines for 7-Series and Zynq Devices

Some design requirements are unique to DFX and specific to 7-Series and ZYNQ-7000 SoC-FPGA devices. With the Reset After Reconfiguration feature (`RESET_AFTER_RECONFIG`), the reconfiguring region is held in a steady state during partial reconfiguration, and then all logic in the new RM is initialized to its starting values. This feature behaves in the same manner as the initial configuration of the FPGA. In order to apply Reset After Reconfiguration, the height of the RP must align to clock region boundaries. Otherwise, any static logic placed between the RP and the clock region boundary would be affected after partial reconfiguration.

The width of the RP must be set so that the left and right edges of the programmable block (Pblock) rectangle make the most efficient use of interconnect and clocking resources. These edges should be placed between two resource columns and not between two interconnect columns, allowing the placer and router tools the full use of all resources in both static and reconfigurable logic. If this rule is not followed, the tool might prohibit the usage of interconnect columns in the Pblock.

Vivado 2021.2 introduced an IP Integrator (Block Design) project based environment for DFX applications (UG947…, 2022). The DFX IP Integrator Project Flow includes:

- Within a Block Design, creating Block Design Containers (BDC) to identify hierarchy in a project.

- Defining BDCs as Reconfigurable Partition within the design hierarchy.
- Populating a set of RMs for each RP.
- Creating a set of top-level and module-level synthesis runs.
- Creating a set of related implementation runs.
- Managing dependencies as sources, constraints, or options are modified.
- Checking DRC rules and results.
- Verifying configurations.
- Generating compatible sets of full and partial bitstreams.

The DFX BDC Project Flow is particularly interesting because it is IP-centric and based on the IP Integrator (Block Diagram). In this methodology, each RM is contained within a block design with a fixed set of input/output ports. Figure 6 shows a DFX block diagram.

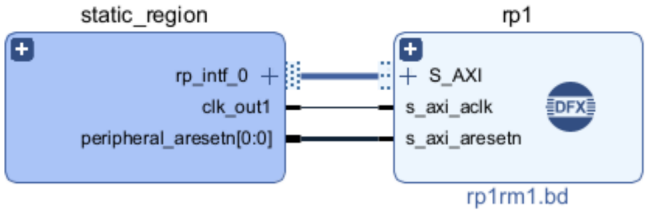


Figure 6 – DFX Block Design Container (BDC) in IP-Integrator (Block Diagram). Adapted from (UG947…, 2022).

2.3 The ZYNQ-7000 SoC Family

The AMD Xilinx ZYNQ-7000 is an All-Programmable SoC (System-On-Chip) combining all aspects of a digital system: processing, high-speed logic, interfacing, memory, and so on. All of these functions are combined into a lower cost, with faster and more secure data transfers between various system elements, lower power consumption, smaller physical size, and better reliability solution integrated at the chip level. The Zynq SoC comprises two main parts: a Processing System (PS) around a dual-core ARM Cortex-A9 processor, and Programmable Logic (PL) – a 7-Series FPGA. Data-links between the PL and PS are made using industry standard Advanced eXtensible Interface (AXI) connections (UG585…, 2023) depicted in Fig. 7.

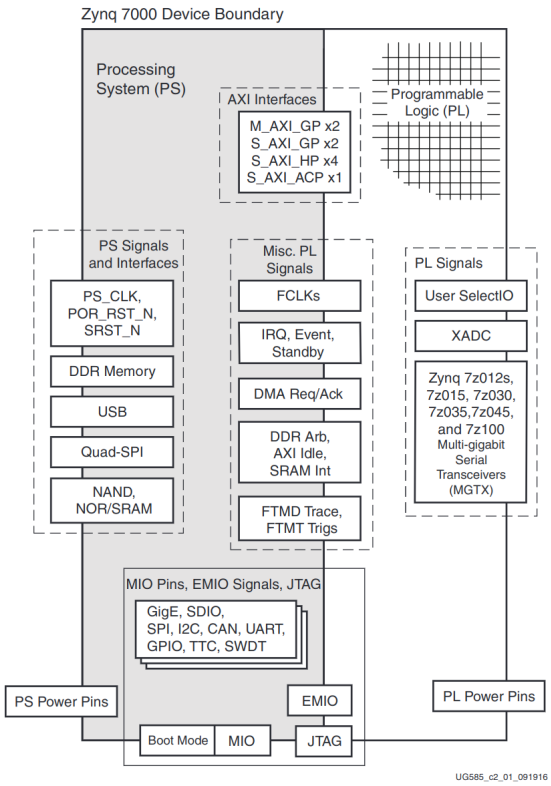


Figure 7 – Zynq SoC – Signals, Interfaces and Pins. Adapted from (UG585…, 2023).

2.3.1 Processing System – PS

The Zynq processing system encompasses a set of associated resources besides the ARM processor forming an Application Processing Unit (APU), and further peripheral interfaces, cache memory, memory interfaces, interconnect, and clock generation circuitry (UG585…, 2023).

The APU is primarily comprised of two ARM processing cores; NEON Media Processing Engine and Floating Point Unit (FPU); a Memory Management Unit (MMU); and a Level 1 cache memory. The APU also includes a Level 2 cache memory, and On Chip Memory (OCM). The architecture diagram is shown in Fig. 8.

The ARM Cortex-A9 can operate at up to 1Ghz, depending on the particular Zynq device. Each of the cores has separate Level 1 caches for data and instructions, both of which are 32 KB; The larger Level 2 cache of 512 KB is shared between the two cores for instructions and data, and there is a further 256 KB of on-chip memory within the APU.

The Snoop Control Unit (SCU) interfaces between the processors and cache memories to ensure cache coherency, also initiates and controls access to the Level 2 cache, arbitrating between requests from the two cores when necessary. The SCU also manages PS-PL transactions via the Accelerator Coherency Port (ACP).

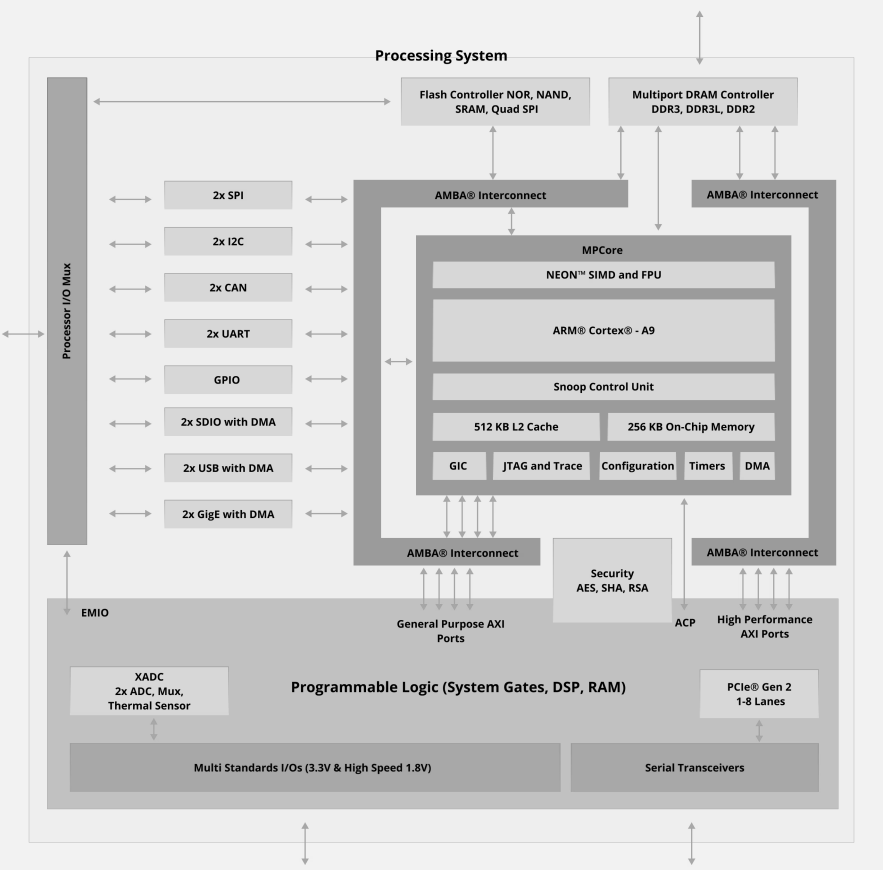


Figure 8 – Zynq SoC - Simplified Architecture Diagram. Adapted from (UG585…, 2023).

Besides that, the Zynq PS features a variety of interfaces between the PS and external components. These I/O includes standard communications interfaces, and General Purpose Input/Output (GPIO). Table 1 presents the relevant interfaces (UG585…, 2023).

Table 1 – I/O Peripheral Interfaces

I/O Interface	Description
SPI (2)	*Serial Peripheral Interface* *Either in master or slave mode (4-pin)*
I2C (2)	*I^2C bus* *Compliant with I2C spec. version 2.* *Supports master and slave modes.*
CAN (2)	*Controller Area Network* *Bus interface controller compliant with* *ISO 118980-1, CAN 2.0A and CAN 2.0B.*
UART (2)	*Universal Asynchronous Receiver Transmitter* *Low rate data modem interface for* *serial communication.*
GPIO	General Purpose Input/Output 4 banks, each of 32 bits.
SD (2)	For interfacing with SD card memory.
USB (2)	Universal Serial Bus Compliant with USB 2.0 in host, device, or flexible OTG mode.
Gib. Eth.	Ethernet Ethernet MAC peripheral for 10Mbps, 100 Mbps and 1Gbps modes.

2.3.2 Programmable Logic – PL

The second principal part of the Zynq architecture is the programmable logic (PL). This is based on 7-Series devices – Artix-7 and Kintex-7 FPGAs. Table 2 presents the prominent features of each Zynq-7000 device for its PL device.

Table 2 – Zynq-7000 Family Members – Programmable Logic

Zynq PL	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
Logic Cells (K)	28	74	85	125	275	350	444
Block RAM (Mb)	2.1	3.3	4.9	9.3	17.6	19.1	26.5
DSP Slices	80	160	220	400	900	900	2,020
Maximum I/O Pins	100	150	200	250	362	362	400
Maximum Transceiver Count	-	4	-	4	16	16	16

The Z-7010, Z-7015, Z-7020 are cost-optimized devices featuring Artix-7 FPGAs, while the Z-7030, Z-7035, Z-7045, Z-7100 are mid-range devices featuring Kintex-7 FPGAs. The main difference between the specific devices within the Zynq family is the type and quantity of the programmable logic. Each of the family members provides a different amount of general purpose logic, block RAMs, and DSP blocks. Naturally, the overall processing capabilities and I/O of the PL section increases in proportion to its resources.

The PS is kept the same across all family members, with the addition that the maximum frequency of the ARM core differs. The PS on the Artix-7 based devices can be clocked at up to 866 MHz, the Kintex-7 devices up to 1GHz.

2.3.2.1 MicroBlaze Soft Processor Core

For comparison purposes, the alternative to a hard processor is a so called soft processor like the Xilinx MicroBlaze, which is comprised of logical elements of the programmable logic fabric. The implementation of a soft processor is therefore equivalent of any other IP block of an FPGA design.

The MicroBlaze embedded processor soft core is a reduced instruction set computer (RISC) optimized for implementation in AMD Xilinx FPGAs. It is implemented with a Harvard memory architecture; instruction and data access are done in separate address spaces. Figure 9 shows a functional block diagram of the MicroBlaze core.

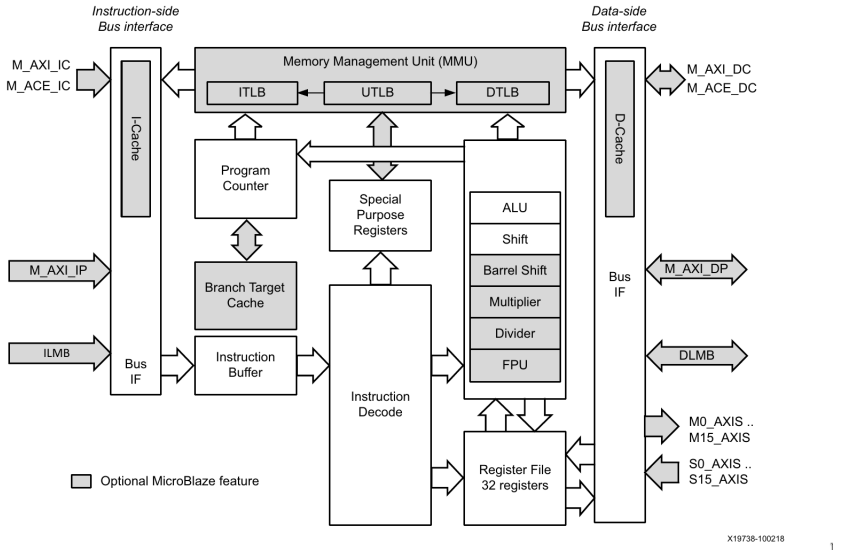


Figure 9 – MicroBlaze Core Block Diagram. Adapted from (UG984…, 2022).

The fixed feature set includes:

- a thirty-two 32-bit or 64-bit general purpose registers.
- 32-bit instruction word with three operands and two addressing modes.
- a 32-bit address bus, extensible to 64 bits.

It can be used with an optional instruction/data cache for improved performance when executing/reading code that resides outside the local memory bus (LMB) address range. Moreover, a large set of additional features enables MicroBlaze implementations for different scenarios with extensive flexibility.

According to Xilinx literature (UG984…, 2022), optimal MicroBlaze configurations can achieve no more than 260 DMIPs (Dhrystone Millions of Instructions Per second) on Zynq in speed grade -3 device, whereas the dual-core ARM is projected to reach 5000 DMIP (2500 DMIPs per core), assuming a PS clock frequency of 1GHz (UG585…, 2023). Therefore, indicating that the ARM core offers about 20 times greater processing performance than a single MicroBlaze core.

Despite its reduced processing capabilities, MicroBlaze can interface with up to 16 AXI4-Stream interfaces, each consisting of one input and one output port. These channels are dedicated uni-directional point-to-point 32-bit data streaming interfaces. Each link provides a low latency dedicated interface to the processor pipeline allowing for an highly efficient solution for transferring data to/from dedicated hardware functions.

2.3.3 Processing System – Programmable Logic Interfaces

The appeal of Zynq SoC-FPGA lies not just in the PL or PS parts but in the composition of a complete embedded solution to form an integrated system. This is mostly due to the adoption of AXI interconnects and interfaces used to bridge the PS-PL boundary. In addition, PS-PL connections can also be made using the EMIO (Extended MIO).

AMD Xilinx contributed strongly to defining AXI4 (part of the ARM AMBA 3.0 open standard) as the definitive interconnect technology for use within FPGA architectures. Briefly, there are three flavours of AXI4, each composing a bus protocol. Furthermore, the AXI4 Standard does not impose strict timing between address and data operations. Unaligned data transfers are also supported independently of bus width. Out-of-order transaction completion is supported through the transaction identifier in AXI side-channel signals.

The specification is geared towards the interface of IP blocks, rather than the interconnect itself. An interconnect is effectively a switch which manages and routes traffic between attached AXI interfaces. There are several interconnects within the PS, some of which directly route to the PL.

- **AXI4**: For memory-mapped links, capable of high-performance with independent read and write channels, which support burst-based access of up to 256 data words. Also supports multiple outstanding addresses for parallel processing of transactions. This is the *full* link specification.
- **AXI4-Lite**: A simplified subset link with support to a single data exchange per transfer (without bursts). It is also memory-mapped resembling a register-like interface in which an address and single data word are transferred.
- **AXI4-Stream**: For high-speed streaming data, supporting burst transfers of unrestricted size. There is no address mechanism. Additional side-channel signals might

be used to transfer positional or sequential information and signaling of packet boundaries. In this interface, a simple FIFO-like handshake is used for data transfers.

The PS-PL interface is divided into AXI General Purpose Ports, AXI High-Performance Ports, and Accelerator Coherency Port (ACP). The General Purpose AXI is a 32-bit data bus, suitable for low and medium rate communications between the PL and PS. The ACP interface is mostly used for cache coherent transactions. The High-Performance ports include FIFO buffers to accommodate read/write bursts for high rate communications. The data width is either 32 or 64 bits, and the PL is the master of all four interfaces. These High-Performance Ports are also attached to the DDR memory controller, which makes them the preferable option when dealing with Direct Memory Access (DMA) inside the PL.

2.4 The Myokinetic Approach for Hand Prosthetic Control

This Section is dedicated to the core concepts of the Myokinetic approach for Hand Prosthetic Control, a Human-Machine Interface based on implanted magnets and gross muscular motion. Furthermore, the experimental data collected from *virtually implanted* magnets in an anatomically relevant forearm mockup is introduced to test and validate the data-driven scheme devised to estimate magnet displacement. In addition, the algorithms used to estimate the magnet displacement are also introduced with details regarding data pre-processing based on the Principal Component Analysis (PCA).

2.4.1 Human-Machine Interface – HMI

In the context of prosthetic control, the HMI should allow effortless control of distinctive movements, bridging the artificial hand and the sources of volition. Novel techniques are in pursuit for a physiologically appropriate interface that could entail simultaneous, direct control over multiple DoFs (Degrees of Freedom). The Myokinetic Control Interface has been proposed to bridge the existing scientific and technological gap embracing the ideia of sensing the magnetic field of implanted magnets in the residual limb muscles, to monitor their contractions and send appropriate commands to the artificial hand (TARANTINO et al., 2017).

As part of such concept, the myokinetic interface requires a transcutaneous magnet localizer that can be integrated in a self-contained limb prosthesis, a feature yet to be realized within the current state of the art (IANNICIELLO, 2024). The magnet localizer is a key component that should translate the magnetic field generated by one or multiple magnetic sources using a set of magnetometers into the position and orientation (pose) of target magnetic markers (MMs).

In magnetic tracking, certain constraints on the geometry of the generated field must be established so that tackling the inverse problem of magnetostatics through the usage of optimization algorithms that minimize the deviation between measured and modeled magnetic field is feasible (BRUCKNER et al., 2017). In this regard, different methods have been proposed for magnet localization: linear methods with closed form solutions that are more sensitive to measurement noise, and nonlinear approaches using Levenberg-Marquardt (LM) algorithm, particle swarm optimization (PSO) with higher computational cost and reduced localization error.

Concerning the sensing system, in (GHERARDINI; MANNINI; CIPRIANI, 2021) the effects related to the number and spatial distribution of magnetic sensors were investigated for the optimal distribution while minimizing the computational cost of localization. Briefly, a simulated proximal amputation with 11 MMs and 480 sensing elements provides the initial distribution for the fast and intuitive proposed sensor set selection strategy. Additionally, (MASIERO et al., 2021) addresses the effects of intrinsic sensor properties such as resolution and localization rate with experimentally verified computer simulations in terms of tracking accuracy and the number of iterations of the LM algorithm.

In (CLEMENTE et al., 2019), an ARM-based embedded solution for the magnet localizer was introduced with the utilization of 32 magnetometers as data providers for the LM algorithm used to devise magnet pose. An Acquisition Unit (AU) was developed to house the tri-axial magnetometers (MAG3110), and a dedicated 16-bit microcontroller (dSPIC33EP) was selected for interfacing with the magnetometers using a I2C serial communication bus. It was reported that 24 ms were necessary to sample and transmit the magnetometers readings to the ARM core in which the localization algorithm ran. Despite its capabilities, the embedded solution suffered from elevated latency data acquisition and transmission. Also, the reported power consumption was of of 980 mW (550 mW for que acquisition unit and 430 mW for the computational unit).

A novel design was proposed in (IANNICIELLO, 2024), with a more modular approach composed of an ARM-based modular solution with capacity for up to eight acquisition units (20 magnetometers each). This design provides a sampling rate for magnetic sensor data of 100 Hz for a single AU and 38 Hz for 8 AUs. The field is digitized in 6.6 ms and additional 2.2 ms are needed for data transfers. The LM algorithm was used to provide magnet localization in parallel to the data sampling and acquisition with mostly constant accuracy and errors below 70 μm. Power consumption ranged from less than 600 mW with one AU to ~1.2 W for eight AUs. It should be noted that the previous design had only a single AU (32 magnetometers) reaching near 1 W of total power, while the improved design was able to adopt 5 AUs (for a total of 100 magnetometers) within the same power budget.

Furthermore, due to the larger system of equations used to derive magnet localization the LM algorithmic latency ranged from 153 μs (one-magnet/AU) to 21.3 ms (five-

magnets/AU) and even higher processing time when multiple movements were considered due to longer iterations (IANNICIELLO, 2024). Table 3 summarizes the works related to the development of myokinetic interfaces, their localization methods, and platform.

Table 3 – Brief Summary for Recent Myokinetic Interface Solutions

Authors	Year	Acquisition Unit		Myokinetic Control Interface	
		Sensors	MMs	Localization Method	Platform
Tarantino et al. (2017)	2017	6	1	MMs as point magnetic dipole: localization through the Levenberg–Marquardt optimization algorithm.	PC
Clemente et al. (2019)	2019	32	1-5		ARM
Masiero et al. (2021)	2021	32	1-10	Sensory system resolution and localization rate simulation and experimental assessment in terms of tracking accuracy and computation time.	PC
Taylor et al. (2021)	2021	96	2	Real-time muscle length tracking in an in vivo turkey model while investigating accuracy, biocompatibility and stability.	PC
Moradi et al. (2022)	2022	3	3	Clinical implementation of a bionic hand controlled with artificial neural networks which extracts human intention directly from the implanted magnet's magnetic fields.	PC
Mendez et al. (2022)	2022	128	1	Data-Driven: Localization through model inference using Linear and RBFNN models.	SoC-FPGA
Ianniciello (2024)	2024	160	8	Acquisition Unit (AU) with 20 sensors (up to 8 AUs). Localization through Levenberg–Marquardt optimization algorithm. Computation time of 21.3 ms per iteration.	ARM
Mendez (2021)	2024	128	5	Data-Driven with Dimensionality Reduction: Localization through run-time reconfiguration (RTR) for model inference using Linear, RBFNN and MLP models.	SoC-FPGA w. DPR

In (MENDEZ et al., 2022), a data-driven method was adopted in place of numerical optimization methods previously used for magnet localization. In this approach, a set of offline experimental data can be used to train, test, and validate machine learning models used to devise magnet displacement from ground-truth data (actual displacement). To that end, a SoC-FPGA was used in a highly parallel implementation for Linear Regression and Radial-Basis Function Neural Network (RBFNN) models within microseconds of computational latency. In the proposal, the number of sensors was also set to 128 (4 AUs). The main shortcoming of this design was that the logic utilization required by the implementation of the magnet localizer might hinder the implementation of additional logic dedicated to control, acquisition, and other functionality required for the eventual self-contained transcutaneous magnet localizer.

In order to mitigate issues in the SoC-FPGA implementation, (MENDEZ, 2021) improved the overall SoC-FPGA proposal with DPR allowing the design architecture to be reconfigured at run-time. This technology allowed the implementation of a set of models used to localize five magnets under real-time constraints with reduced logic utilization. The obtained results pave the way towards the adoption of DPR-based architectures for magnet localization applied to myokinetic interface.

2.4.2 Data-Driven Magnet Tracking applied to the Myokinetic Control Interface

In order to devise precise magnet localization for the myokinetic interface, one must consider which model structure to use based on a fitting solution that accurately represents the system behavior. The general process of model training, testing and cross validation is usually a small problem. The difficulty lies in selecting the proper model structure, hyper-parameters and in devising engineering choices based prior knowledge or physical insight into the system.

2.4.2.1 Experimental Data Acquisition

A Myokinetic Interface mockup (Fig. 10) was used to experimentally assess the accuracy, repeatability, and response time of the HMI prototype. The mockup was carefully designed with anatomic relevant proportions and magnet trajectory. It was intended to replicate the contraction and elongation of the extrinsic muscles of the hand.

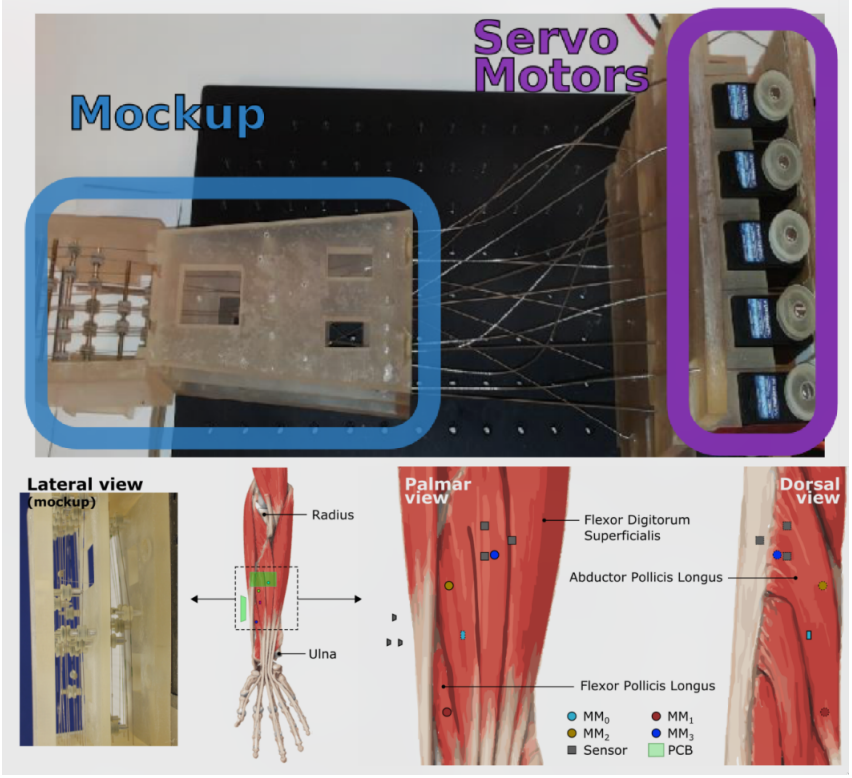


Figure 10 – Myokinetic Interface Mockup. Adapted from (TARANTINO et al., 2017)

The experimental data was collected using different configurations for the displacement of magnets from a single magnet to five magnets. A total of eleven datasets were acquired for the different configurations and displacement profiles utilized (six for a single magnet control; five for five-magnets). The mockup is capable of $\sim$ 10 mm of maximum translation of the wire connected to each magnet with independent control for each magnet

every 50 ms. The input displacement applied to the servos is used as a ground-truth value for the analysis of the retrieved magnet displacement by the myokinetic localizer. The length of the trajectory was derived from anatomical measurements (TARANTINO et al., 2017).

The magnetic fields are sampled through four AUs, each equipped with 32 three-axis magnetometers. For every AU, the sensors were laid out on orthogonal 8×4 grids with 9 mm inter-spacing. The AUs were positioned on four opposite sides of a parallelepiped enclosing the mockup workspace. The sensor readouts were sampled at 20 Hz and all collected signals were stored to be used for offline processing.

Five-Magnet Dataset – Multisine Acquisition Procedure

The five-magnet dataset captures the magnetic sensor readouts from the 4 AUs (total of $32 \times 4 \times 3 = 384$ features) for the simultaneous displacement of five magnets with a multisine displacement trajectory for the servo command.

$$y_k = \frac{A}{2M}\left(\sum_{i=1}^{M}[\sin(2\pi f_i t_s k + \phi_i)] + 1\right)$$

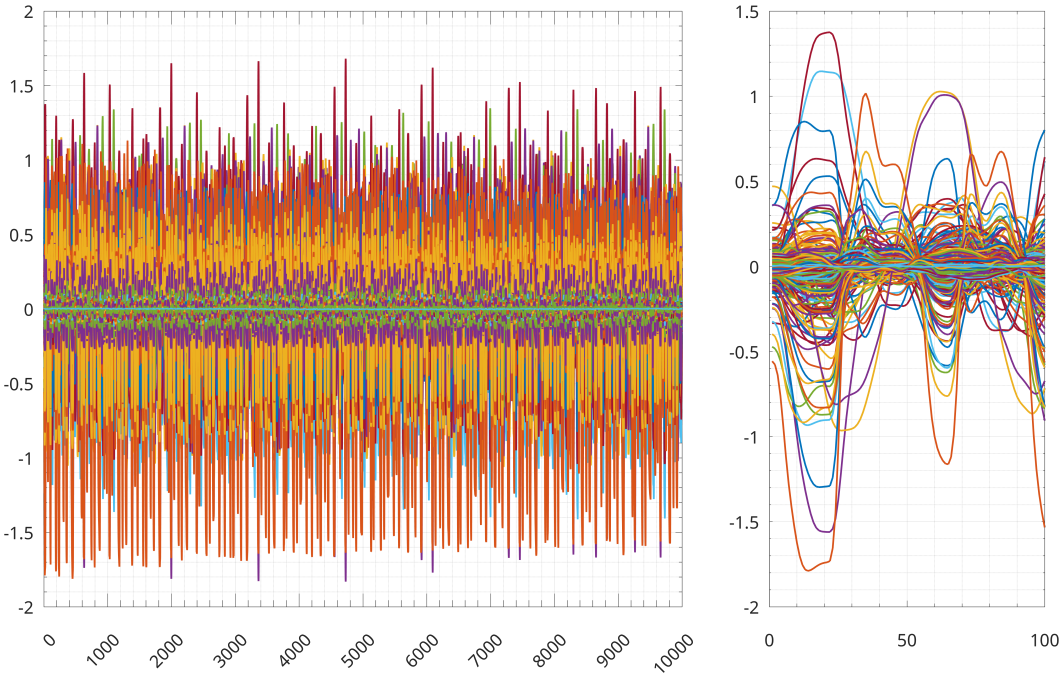
where y_k denotes the servo command trajectory at the discrete time $k = 0, 1, \ldots, N$, A is the desired amplitude for y_k in the range $[0, A]$, M is the number of sine wave given the equally spaced frequency $f_1, f_2, \ldots, f_M$ and $\phi_1, \phi_2, \ldots, \phi_M$ random phases in the range $[0, 90°]$. The multisine trajectory displacement is able to excite the system with simultaneous control of each magnet with smooth trajectories that could give insights into the dynamic behavior of the system.

The acquisition is shown in Figure 11, considering: $A = 10$ mm for the displacement amplitude, $M = 10$ sine waves, and frequencies in a range between 0.1 and 0.5 Hz.

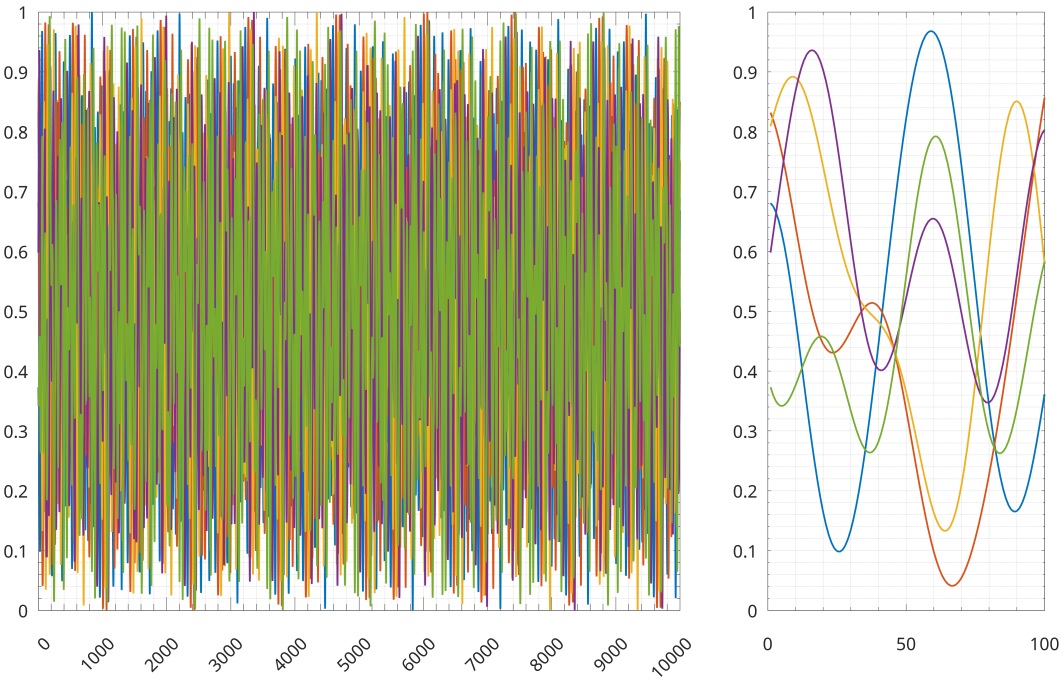
Five-Magnet Dataset – Ramp Acquisition Procedure

The Ramp acquisition uses a ramp-like output to the servo displacement trajectory. A set of four ramps profiles was acquired as it is relevant for the application of magnet displacement in the myokinetic interface. The ramps are configured with positive and negative slopes with varying speed for 20, 40, 60 and 80. Figure 12 depicts the sensed magnetic field and the controlled displacement of the five-magnets during acquisition.

It can be seen that there is a distinguishable relationship between the measured magnetic field and the output displacement, especially in 12 given the simpler magnet trajectory for each magnet.

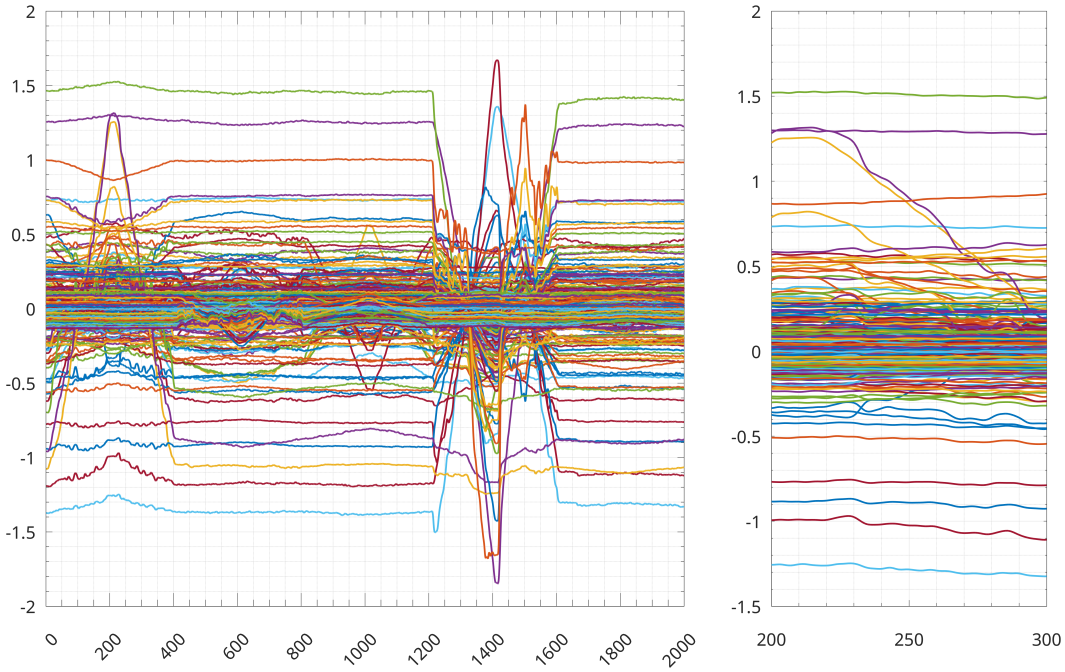


(a) Magnetic Sensor Data

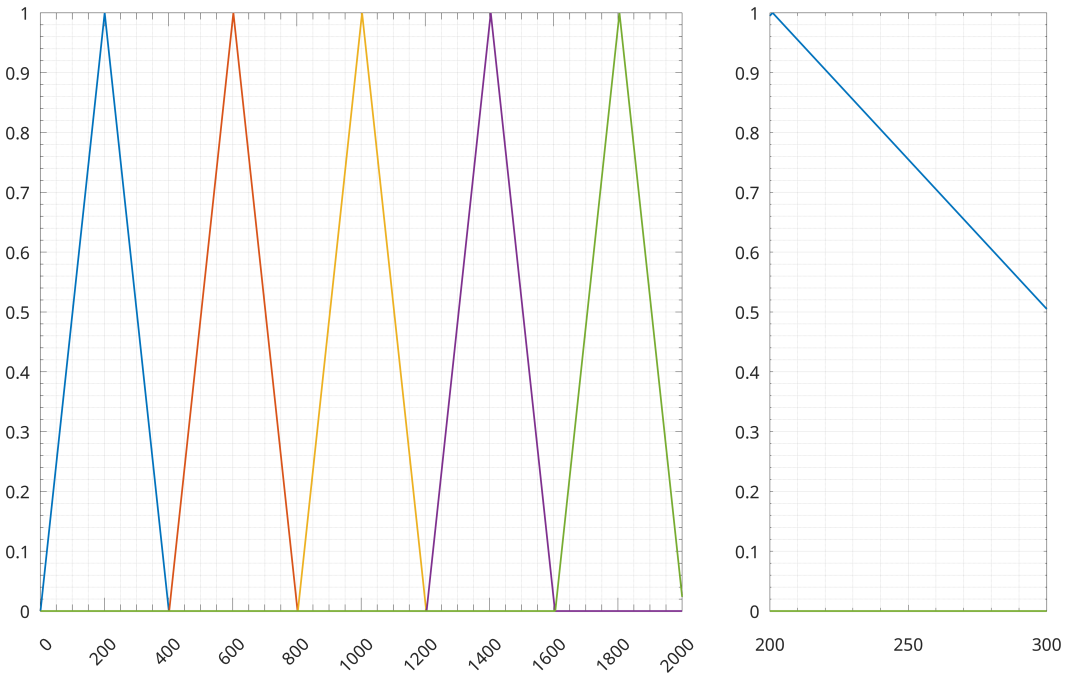


(b) Target Magnet Displacement

Figure 11 – Magnetic Sensor Data for Multisine Acquisition – Five-Magnets Dataset. The x-axis display the number of samples in the dataset. The y-axis depicts the mean-centered readings in Gauss for (a), and the normalized displacement in (b). In both figures, the right plot is the zoomed window view of left plot data.



(a) Magnetic Sensor Data



(b) Target Magnet Displacement

Figure 12 – Magnetic Sensor Data for Ramp Acquisition – Five-Magnets Dataset. The x-axis display the number of samples in the dataset. The y-axis depicts the mean-centered readings in Gauss for (a), and the normalized displacement in (b). In both figures, the right plot is the zoomed window view of left plot data.

In (MENDEZ et al., 2022; MENDEZ, 2021), the multisine acquisition was used in model fitting for training, and testing procedures. The sum of sinusoidal waves is a common excitation signal used in system identification and the parameters used in data acquisition

were selected for the purpose of machine-learning model training. The ramp-like signals were used to validate model inference effectively testing the fitted machine-learning models with a different set of data than the one used to train each model.

In the following, we present some of tools & methods used in data-driven magnet localization for the myokinetic interface.

2.4.2.2 Dimensionality Reduction using Principal Component Analysis (PCA)

PCA is widely used to reduce the data dimensionality of machine learning models. It is used to devise a truncated linear transformation (Eq. 2.1) in which K columns from the $\mathbf{V}$ matrix (i.e., the principal components, PCs) form an orthogonal basis for the K features providing a low-dimensional approximation ($\mathbf{T_K}$) to the high-dimensional $\mathbf{X}$ data in terms of relevant patterns. This transformation is associated with another matrix factorization given by the SVD of $\mathbf{X} \in \mathbb{C}^{n \times m}$ (Eq. 2.2).

$$\mathbf{T_K} = \mathbf{X}\mathbf{V_K} \tag{2.1}$$

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H, \tag{2.2}$$

where $\mathbf{U} \in \mathbb{C}^{n \times n}$ and $\mathbf{V} \in \mathbb{C}^{m \times m}$ are *unitary* matrices and $\mathbf{V}$ holds an orthonormal-basis of corresponding eigenvectors (BRUNTON; KUTZ, 2019). Here ^{H} denotes the complex conjugate transpose.

The PCA has two main stages, the *training* and the *projection stage*. During the *training stage*, principal components are derived using SVD. This stage is computationally complex and time-consuming as it involves computing a matrix factorization (BENGIO; COURVILLE; VINCENT, 2013). The *projection stage* requires the PCs and the input values to perform the dimensionality reduction (see Eq. 2.1).

The usage of PCA for magnetic field data is relevant as the acquired data is composed of 128 three-axial readings (384 total features) and the selection of an optimal set of measurements is capable of providing reduced localization error (GHERARDINI; MANNINI; CIPRIANI, 2021). Therefore, in (MENDEZ, 2021) a dimensionality reduction based on PCA was used to train shallow models with less inputs, yielding more efficient and compact hardware implementations.

2.4.2.3 Regression using Black-Box Data-Driven Models

In the proposal for the development of a magnet localizer architecture for the myokinetic interface, we consider a black-box modeling, in which no physical insight is known, the dynamic behavior of the system is considered to be nonlinear. In this modeling scheme, we attempt to obtain model parameters based on measured input and output data (NELLES,

2020). The hardware implementation for the candidate models will be detailed later in Chapter 3.

We begin by introducing the basic formulation for a set of three models used in the proposed system: Linear Regression, Artificial Neural Networks (ANNs), and Radial-Basis Function Neural Network (RBFNN). Both model structure and parameters are determined from experimental modeling guided towards efficient hardware implementations. The goal is to attain elevated accuracy without resorting to sophisticated and resource-consuming machine learning approaches.

Linear Regression

A linear model may be able to approximate a nonlinear process with reasonable accuracy if its nonlinear characteristic is weak. It is a simple model with a small number of parameters.

$$\hat{y} = \sum_{i=0}^{p} w_i u_i, \text{ with } u_0 = 1$$

Despite of its simplicity, linear models are the standard models with a series of relevant properties (NELLES, 2020):

- Evaluation speed is fast since on p multiplications and additions are required.
- Parameter optimization can be performed very rapidly by a least squares algorithm.
- Structures optimization can be performed efficiently by a linear subset selection technique such as the orthogonal least squares (OLS) algorithm.
- Constraints can be incorporated for the model output and parameters if a quadratic programming algorithm is used instead of the least squares.

Artificial Neural Networks – ANN

Within the context of Artificial Neural Networks (ANNs), the Feedforward Neural Network (FFNN), also called Multilayer Perceptron (MLP) is structured around layers and neurons in which the information flows forward through hidden nodes (if any) to the output nodes; see Fig. 13. In basis function formulation, the FFNN can be written as:

$$\hat{y} = \sum_{i=0}^{M} w_i \phi_i \left(\sum_{j=0}^{p} w_{ij} u_j \right)$$

with the output later weights w_i and the hidden layer weights w_{ij}. The ϕ_i is the output from the activation function.

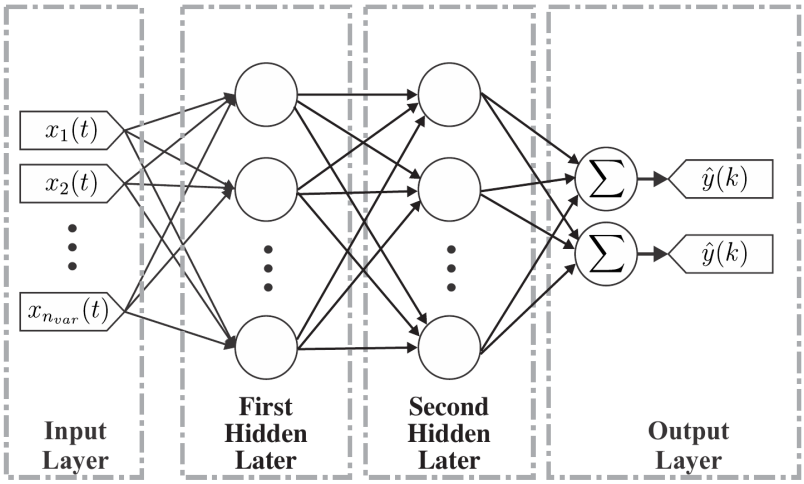


Figure 13 – Feedforward Neural Network Architecture

The FFNN is a universal approximator (NELLES, 2020). This means it can approximate any smooth function to an arbitrary degree of accuracy as the number of hidden layer neuron increases. However, this proof is not constructive in the sense that it determines the desired structure of the neural network. An FFNN is trained by the optimization of these neuron weights.

Some important properties can be summarized:

- Sensitivity to noise is very low since almost all training data samples are exploited to estimate all model parameters.
- Evaluation speed is fast since the number of neurons is relatively small compared with other neural network architectures.
- Parameter optimization is slow due to the usage on nonlinear optimization techniques.
- Structure optimization requires expensive pruning or growing methods.

Radial-Basis Function Neural Network – RBFNN

In contrast to the FFNN, the Radial-Basis Function Neural Network (RBFNN) utilizes a radial construction mechanism. Its operation can be split into two parts. First, the distance of the input vector $\mathbf{u}$ to the center vector $\mathbf{c}$ with respect to the covariance matrix $\mathbf{\Sigma}_i$ is calculated. In the second part, this scalar distance is transformed by the nonlinear activation function $g(x)$ (NELLES, 2020). A typical choice for activation function is the Gaussian function:

$$g(x) = \exp\left(-\frac{1}{2}x^2\right)$$

If several RBF neurons are used in parallel and are connected to an output neuron, the radial basis function network is obtained (Figure 14). It can be described as:

$$\hat{y} = \sum_{i=0}^{M} w_i \phi_i(||\mathbf{u} - \mathbf{c_i}||_{\Sigma_i})$$
$$\phi_i(\mathbf{u}, \theta_i) = \exp\left(-\frac{1}{2}||\mathbf{u} - \mathbf{c_i}||^2_{\Sigma_i}\right)$$

where the hidden layer parameter vector θ_i consists of the coordinates of the center vector c_i and the entries of the inverse covariance matrix, and output layer weights w_i.

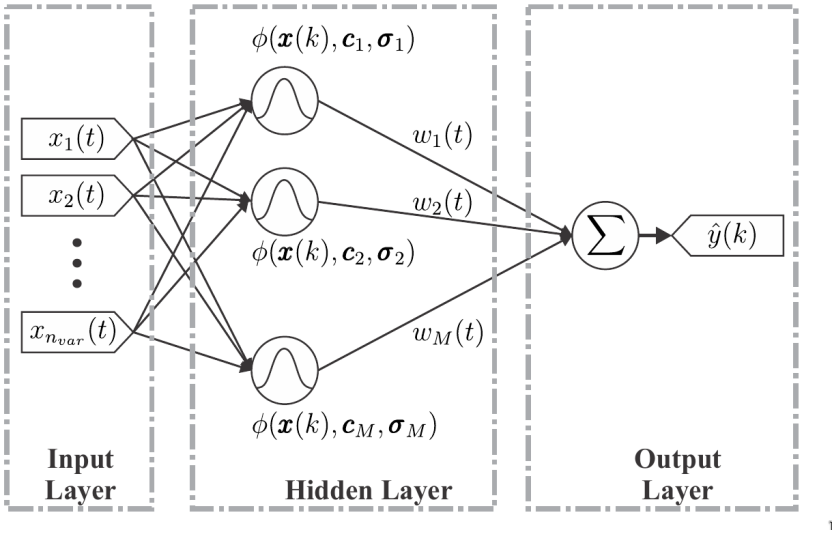


Figure 14 – Radial-Basis Function Network Architecture

Contrary to the FFNN, multiple hidden-layers are not feasible as the neuron outputs of a possible hidden layer would span the input space for the second hidden layer. Due to the lack of interpretation, the hidden layer parameters of the second hidden layer cannot be chosen by prior knowledge, which is a major strength of the RBFNN.

3 Run-Time Reconfigurable Architecture for Myokinetic Magnet Tracking

In this chapter, it is presented the relevant steps taken towards the development of a run-time reconfigurable system, together with its design features, conceptual design, and implemented design. The proof of concept of the prototype exploits an experimental dataset for model training & validation to localize five MMs simulating a multi-DoF prosthetic hand. The dataset was used to experimentally assess the accuracy, repeatability and response time of the proposed magnet tracking system.

3.1 Introductory Remarks

The proposed myokinetic localizer for the eventual control of the hand prosthesis was implemented in two different embedded targets: the SoC-FPGA (ZYNQ-7020 – PYNQ-Z2) development kit and the FPGA (Artix-7 – Arty A7-100T) development kit. The SoC-FPGA integrates the software programmability of an ARM-based processor with the hardware programmability of an FPGA. This was done in order to characterize the proposed DPR system using a hard-core processor (ARM Cortex A9) and a soft-core processor (MicroBlaze) implemented into the FPGA fabric.

At the system level, both implementations (FPGA and SoC-FPGA) must perform a similar set of tasks regarding magnet tracking, which is based on machine learning models implemented as reconfigurable modules. The proposed run-time reconfiguration scheme is also kept the same in both implementations, notably: the partial reconfiguration controller, number of reconfigurable partitions, floorplanning constraints and the data source for partial bitstreams.

At the architectural level, each embedded platform (FPGA or SoC-FPGA) will require different means of implementation for the proposed DPR system specification. This chapter is also dedicated to the analysis of the SW/HW co-design for the proposed myokinetic localizer.

3.2 Proposed Modeling for Five Magnet Tracking

In order to develop an efficient localizer, we are interested in selecting a viable set of estimators for a multi-magnet tracking scenario. This modeling process requires evaluating tradeoffs, such as model complexity, accuracy, and required resources/area in the FPGA fabric.

We adopted experimental data collected in (MENDEZ et al., 2022) and the data modeling methodology presented in (MENDEZ, 2021) to train/test and validate the data-driven models used in magnet tracking for a set of five magnetic markers from the myokinetic experimental dataset. In this proposal, we also adopted the same VHDL code generator tools for the RTL implementation of RBFNN and Linear Regression models with our own improvements for the proposed DPR design, these improvements are detailed in Section 3.3.3 and 3.3.4.

The experimental dataset is used to train the Linear Regression, RBFNN, and FFNN data-driven models to perform regression for the expected magnet displacement for each magnetic marker. This modeling allows the usage of different models for magnetic tracking. In addition, the data dimensionality from the 128 three-axis digital magnetometers might lead to extensive hardware utilization for the implementation of these models in hardware as seen in (MENDEZ et al., 2022). In our proposal, we used the same pre-processing from (MENDEZ, 2021) to perform dimensionality reduction using PCA with our own RTL implementation using high-level synthesis as described in Section 3.3.1. The RTL implementation for the FFNN model was also developed using high-level synthesis and is described in Section 3.3.2.

3.2.1 Model Performance with Unprocessed Magnetic Sensor Features

From the experimental dataset, the multisine samples are used in model training & validation with a 75% partition used for training and the remaining 25% used for testing. The MSE (Mean Squared Error) is used to assess the accuracy of the localizer based on its predicted magnet displacement $\hat{X}$ and the real magnet displacement X. This measure is shown in Eq. 3.1 for a sample of n data points and in Eq. 3.2 calculated in dB.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{X}_i)^2 \tag{3.1}$$

$$\mathrm{MSE}_{\mathrm{dB}} = 10\log_{10}\mathrm{MSE} \tag{3.2}$$

To translate the obtained MSE in dB to a measure of error in magnet displacement it is necessary to scale the measure accordingly. A MSE of -10 dB corresponds to 1 mm^2 (RMSE $\approx$ 0.31 mm) which is around 3.1% of the maximum muscle deflection applied during data acquisition (up to 10 mm).

Another metric used to evaluate model performance is the R^2 (coefficient of determination) as a larger value of R^2 implies in a more successful regression model. It can be calculated as follows in Eq. 3.3 using the SSE (residual sum of squares - Eq. 3.4) and SST (total sum of squares - Eq 3.5):

$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}} \tag{3.3}$$

$$\text{SSE} = \sum_{i=1}^{n} (X_i - \hat{X}_i)^2 \tag{3.4}$$

$$\text{SST} = \sum_{i=1}^{n} (X_i - \bar{X})^2 \tag{3.5}$$

$$\bar{X} = \frac{1}{n} \sum_{i=1}^{n} X_i \tag{3.6}$$

The task of model identification for the multi-magnet tracking is twofold: a) determination of model structure and model parameters estimation and; b) model subset selection. The model structure consists in the determination of kernel parameters and hyperparameters. The parameters estimation step is model-dependent as each model possesses its own training method. The FFNN is trained using backpropagation algorithm for the network weights and bias. The Linear Regression Model is trained using the least-squares approach. The RBFNN kernel centers and widths are estimated using clustering schemes (LLOYD, 1982) and its weights are estimated using the least squares algorithm.

The first approach for the data modeling consists in model training without any pre-processing stages for the magnetic sensor features in order to establish baseline results. The MSE and R^2 for this approach are shown in Table 4.

Table 4 – MSE [dB] and R^2 values for model training with unprocessed magnetic sensor features. Highlighted cells presents the best MSE [dB] (lower is better) for each MM (Magnetic Marker) in the experimental data.

Magnet	**Model**					
	Linear		**RBFNN**		**FFNN**	
	R^2	MSE [dB]	R^2	MSE [dB]	R^2	MSE [dB]
MM1	0.936	-21.9573	0.942	-22.3632	0.932	-21.7635
MM2	0.884	-18.2703	0.653	-14.2600	0.889	-18.4118
MM3	0.894	-18.8868	0.769	-15.8982	0.922	-20.2026
MM4	0.974	-25.6571	0.967	-25.3986	0.971	-25.8990
MM5	0.882	-19.4943	0.364	-13.8027	0.889	-19.8585

The overall results from Table 4 are satisfactory and show that the FFNN obtains the best MSE for four out of five magnets if compared to the other models. It is also necessary to evaluate the required area/logic resources needed for the RTL implementation for these models as the number of input features from 128 magnetic sensors (a total of 384 inputs) will lead to designs with high utilization.

In this regard, (MENDEZ, 2021) proposed the usage of a dimensionality reduction preprocessing stage to save on hardware utilization by selecting shallow models with desirable characteristics based on MSE and R^2 obtained during model training.

3.2.2 Model Performance with Dimensionality Reduction of Magnetic Sensor Features

Using PCA to provide a low dimensional approximation of high dimensional correlated data from the myokinetic experimental dataset allows for retaining as much of the variance in the dataset as possible. Figure 15 presents the singular values and cumulative variance obtained using the multisine acquisition in the myokinetic dataset.

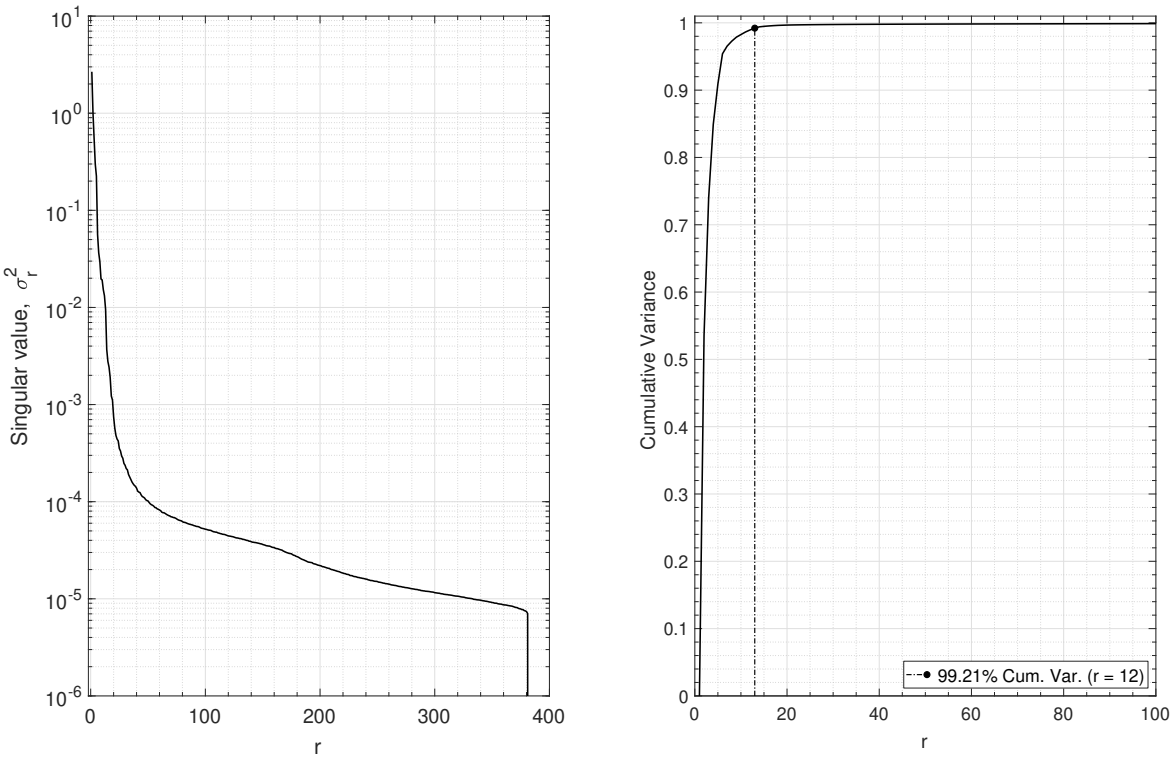


Figure 15 – Singular Values and Cumulative Variance for the magnetic sensor readings from the five-magnet dataset – multisine acquisition.

A low dimensional approximation is given by the truncation of Principal Components (PCs). The selection criteria for the PCs selection might be based on the desired rank, noise magnitude, and the distribution of PCs. It is a common practice to perform the truncation based on the explained variance (typically at a threshold of 99%). For the myokinetic experimental data, this yields 12 PCs as seen in Fig. 15.

Furthermore, the applicability of PCA presents some limitations as the results depend on data alignment (scaling, translation, rotation), and this condition might lead to artificial rank inflation. When considering that a truncated set of PCs is used to obtain the low-rank

approximation for multiple outputs (the displacement of five magnets) it is beneficial to use a selecting criteria based on the correlation between PCs and the desired output.

This correlation ranking method mitigates the effects of rank inflation and selects the components of interest for each magnet. In the implementation, PCs are ranked by their correlation coefficient with the property to be correlated (i.e., the dependent variable). Therefore, we calculated each PC's absolute correlation with every output (i.e., magnet displacement). Results for this ranking strategy are shown in Fig. 16 for the first 12 PCs.

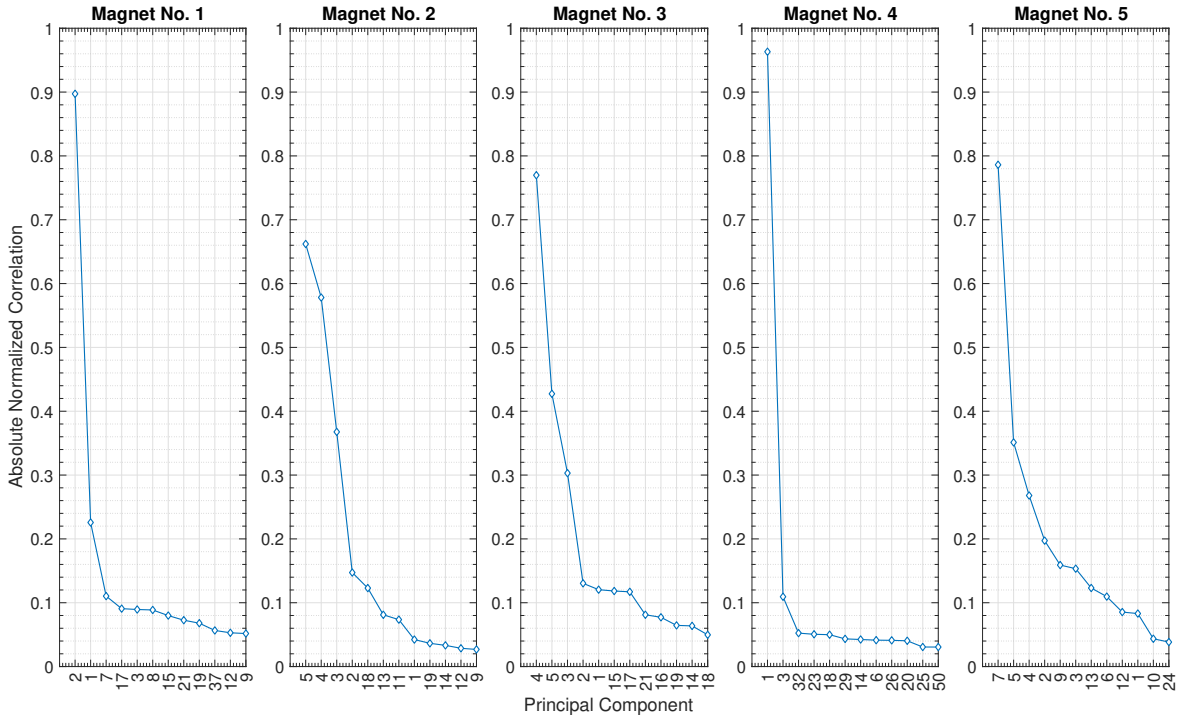


Figure 16 – Correlation ranking of Principal Components for every magnet with first 12 PCs ranked.

In the explained variance ranking strategy, PCs with reduced associated variance might not be selected but it can not be implied that the corresponding PC is unimportant. The correlation-based ranking yields better results in model performance and it can be seen that some magnets share a similar set of PCs with high correlation to its displacement output. This ranking overlap between different magnets reduces the required amount of PCs to be kept in the transformation matrix used to obtain the low dimensional approximation in the hardware implementation.

Regarding model performance, different model schemes can be exploited by selecting at most k PCs to be used in the low-rank approximation following the correlation ranking. In this regard, we conducted tests using $k = [4, 5, 8, 10, 16, 20, 25, 32, 50, 64, 100, 128, 150, 200]$ PCs for model training and evaluation of MSE in dB and R^2.

The analysis of models with reduced magnetic features will provide insights on model selection for hardware implementation based on error criteria (MSE and R^2) and based on

hardware utilization as the reconfigurable partition (RP) has area and resource constraints. Results are shown in Fig. 17 for MSE and Fig. 18 for R^2.

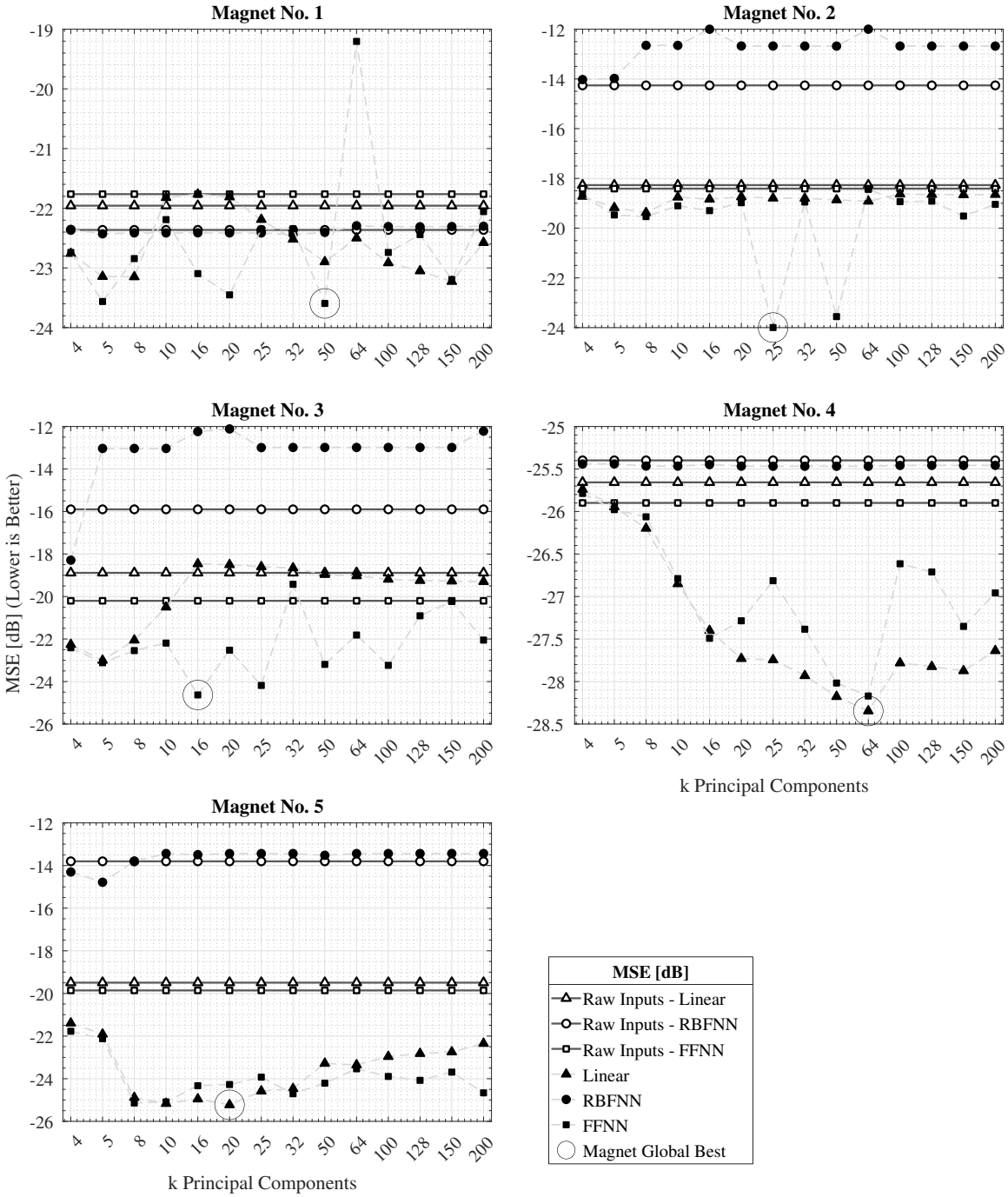


Figure 17 – Model performance MSE (dB) with reduced magnetic features for every Magnet. Baseline results are shown for models with unprocessed features (Raw Inputs). The horizontal axis represents the number of PCs used in model training. The vertical axis represents the obtained MSE in dB (lower is better) and the best metric is highlighted for each magnet.

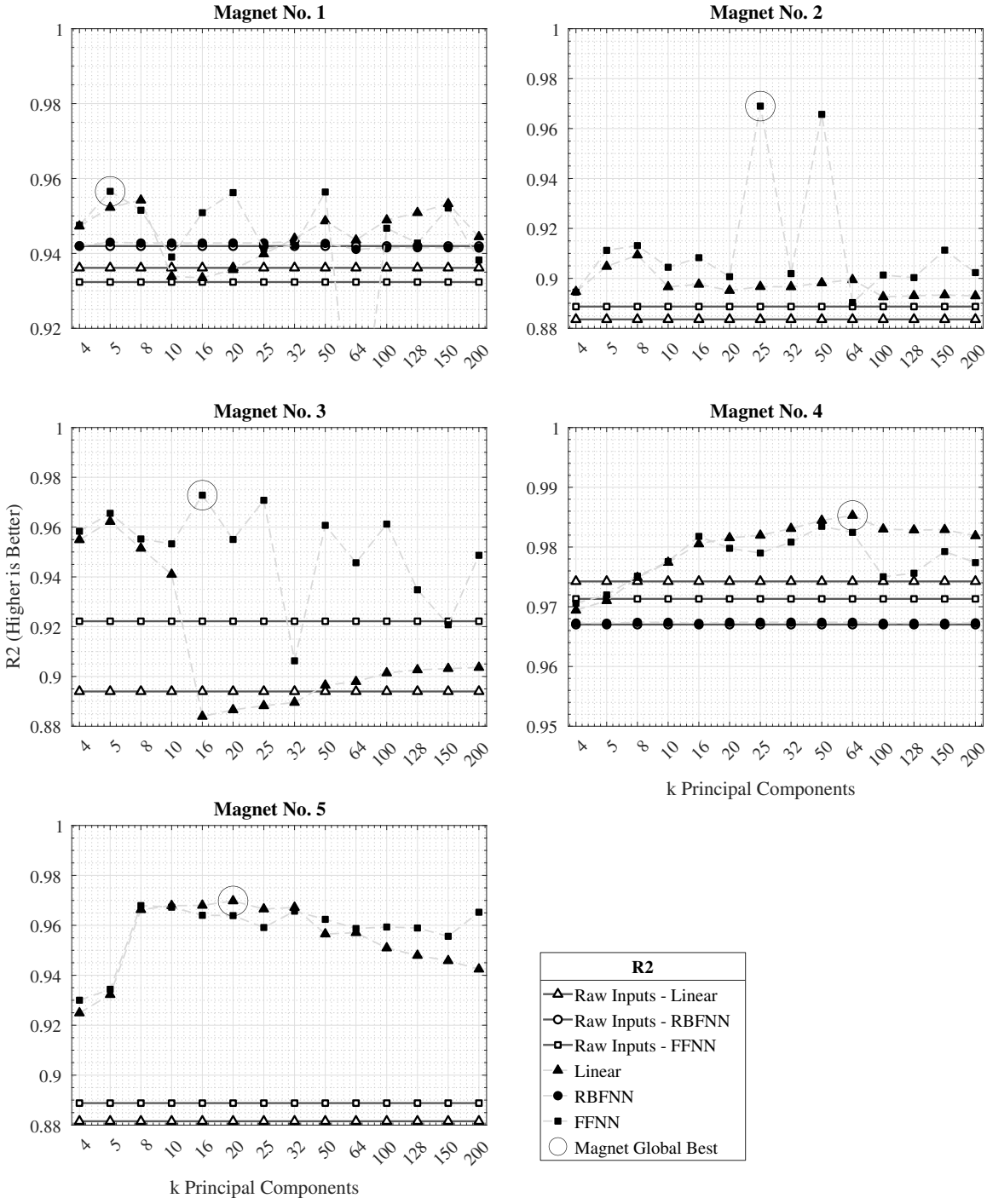


Figure 18 – Model performance R^2 with reduced magnetic features for every Magnet. Baseline results are shown for models with unprocessed features (Raw Inputs). The horizontal axis represents the number of PCs used in model training. The vertical axis represents the obtained R^2 (higher is better) and the best metric is highlighted for each magnet.

From the results, one can conclude that for certain combinations of magnets/models, the usage of reduced magnetic sensor features outperforms models trained with unprocessed features. The FFNN model presents the best results in MSE (dB) for magnets 1, 2, and 3 considering 50, 25, and 16 PCs, respectively. For magnets 4 and 5, the Linear Regression

model with 20 and 64 PCs performs better the FFNN and RBFNN models.

Regarding the hardware implementation, models with a reduced number of inputs (fewer PCs) are preferred due to lower hardware utilization and latency. Table 5 presents the candidate models for hardware implementation of the RTR architecture. It should be noted that 38 out of the total of 384 PCs are needed for the implementation of the dimensionality reduction, since some of the PCs have a high correlation with the output (magnet displacement) across different magnets.

Table 5 – Summary of models selected for hardware implementation.

Magnet No.	**Model**	k PCs	MSE [dB]	R^2
MM1	RBFNN	10	-22.41	0.94
MM2	FFNN	25	-23.99	0.96
MM3	FFNN	16	-24.63	0.97
MM4	Linear Regression	16	-27.40	0.98
MM5	Linear Regression	16	-24.95	0.96

3.3 Machine Learning Models Implementation on Hardware

This section presents the detailed design and means of implementation for the data-driven models used in the proposed RTR architecture. First, the HLS-based implementation for the Dimensionality Reduction using PCA is presented in 3.3.1. Key results are presented regarding the RTL obtained from the imperative code implementation in 3.3.1.1. The proposed HLS-based PCA implementation was also featured in another research paper (MENDES et al., 2023) (Appendix A) with adaptations used in a machine learning-based strain estimation task with ultrasonic guided waves.

The proposal for the FFNN model implementation is also based on HLS is described in Sec. 3.3.2. The Linear Regression, and RBFNN models are derived from HDL code generators allowing the selection of model parameters for its structure (number of operators and parallelism) with parametric floating-point bit-width. The Linear Model is further detailed in Sec. 3.3.3 and the RBFNN model in Sec. 3.3.4.

3.3.1 HLS Implementation for Dimensionality Reduction using PCA

As previously stated, model inference using reduced features is able to provide acceptable accuracy for magnet localization. In the proposal, this HLS kernel for dimensionality reduction using PCA works as a functional block in the embedded design. It must be able to receive input data from magnetic sensors and provide the features used for magnet localization in further stages by the machine learning models.

The proposed kernel implements three processing pipelines: (i) mean-removal, (ii) dimensionality reduction using PCA, and (iii) output data scaling. In addition, a single AXI4 Memory-Mapped interface is used for reading and writing from a memory bank. It also considers a 32-bit data path used for single-precision floating-point operations and a sequential execution mode controlled by an AXI4-Lite interface.

The first pipeline module processes the input data from magnetic sensors with a mean removal operation using values taken from model training. By subtracting the mean of each feature from the data points, mean removal ensures that the data is centered for the dimensionality reduction stage. In the following pipeline stage, the features are transformed into a lower-dimensional subspace spanned by the principal components. The projection stage operates by projecting the magnetic data points onto the principal components subspace. Mathematically, this projection is achieved by computing the dot product between the feature data vector and the matrix containing the principal components which was stored from the singular value decomposition (SVD) performed during model training. Each row in the resulting projected data matrix corresponds to the projection of the corresponding original data point onto the principal components subspace.

The algorithmic implementation consists in a loop re-ordered matrix multiplication, which transposes the iteration space in order to eliminate loop-carried dependencies and enable pipelining (FINE LICHT et al., 2021). This transformation also impacts the memory access pattern with improvements to performance as the input is processed in a contiguous fashion. In the kernel, principal components and expected mean values (used in mean-removal) were mapped into ROM elements.

3.3.1.1 HLS Source & Synthesis Results

As for the Vitis HLS development flow, the design was validated through testbench simulation based on ground-truth results for correctness. In addition, a C/RTL Co-Simulation was used to verify that the RTL is functionally identical to the C implementation. After verification, the RTL design was exported from Vitis HLS as a Vivado IP. Moreover, the exported IP was integrated in Block Design to the ZYNQ PS for implementation on the selected target device (*xc7z020clg400-1*).

The source code for the proposed kernel is shown in Listing 1. Pipelining and unrolling using pragma directives were performed. Input/Output interface configuration is also derived from pragma directives. The source headers and includes are generated automatically from scripts with values from the magnet tracking training set.

Code 1 – HLS PCA IP Source Code

```
1 #include "pca.h"
2
```

```
void dut(const float in[M], float out[N_PCs]) {
#pragma HLS TOP name = dut
// NOTE -- AXI Memory-Mapped interface to the IP
#pragma HLS INTERFACE m_axi port=in  offset=slave depth=M     bundle=gmem
#pragma HLS INTERFACE m_axi port=out offset=slave depth=N_PCs bundle=gmem
// NOTE -- AXI Lite control interface to the IP
#pragma HLS INTERFACE s_axilite port=return

    // Init Constant Values
    const float mn[M] = {
#include "mn.txt"
    };
    const float V[M*N_PCs] = {
#include "V_corr.txt"
    };

    // R/W Signals
    float out_data[N_PCs];

    ap_uint<16> p;
    ap_uint<8> m;

    // PCA ITERATIVE MODULE
iter_in: // PCA ITERATIVE LOOP
    for (p = 0; p < M; ++p) {
        // Mean Removal
        const auto IN = (in[p] -  mn[p]);
    iter_pc: // PCA INNER LOOP
        for (m = 0; m < N_PCs; ++m) {
#pragma HLS PIPELINE II=1
            const auto prev = (p == 0) ? 0 : out_data[m];
            out_data[m] = prev + IN * V[p * N_PCs + m];
#pragma HLS DEPENDENCE variable=out_data false
        }
    }

    // Scaler & Output
    float out_scaled = 0;
iter_scaler: // DATA SCALER MODULE
    for (m = 0; m < N_PCs; m++) {
#pragma HLS PIPELINE II=1
        // Scaler (as float)
        out_scaled = (out_data[m] - SCALER_MIN);
        out_scaled = out_scaled / (SCALER_MAX - SCALER_MIN);
        // Scaler (as int)
        out_scaled = out_scaled * (SCALER_RANGE_MAX - SCALER_RANGE_MIN);
        out_data[m] = out_scaled + SCALER_RANGE_MIN;
    }

iter_output: // DATA OUTPUT MODULE
    for (m = 0; m < N_PCs; m++) {
#pragma HLS PIPELINE II=1
        out[m] = out_data[m];
    }
}
```

Table 6 summarizes the estimated latency and resource consumption for the synthe-

sized design. From Table 6, it is noticed that the PCA processing pipeline takes up ≈ 99.44% of the total latency. This pipeline was implemented with II = 1 (Initiation Interval) for optimal throughput performance.

Tables 6 and 7 consider a synthesis target clock of 100 MHz for the target device in the ZYNQ SoC-FPGA (*xc7z020-clg400-1*). Post-route timing report from Vitis HLS indicates that the design is capable of operating at up to 120.83 MHz (8.276 ns). The obtained latency of 227.84 μs is within the margin of the required time to process magnetic sensor features for real-time operation and the required fabric resources required by the HLS kernel will not impact the DPR system implementation as this logic will be placed in a static (non-reconfigurable) region.

As the Vitis HLS tool implements code operations using specific device resources in the RTL it is worthwhile to analyze the usage of operators and storage elements after C synthesis, as presented in Table 7.

Table 6 – HLS Synthesis Performance & Resource Estimates

Module & Loops	Latency (cycles)	Latency (ns)	Iteration Latency	Initiation Interval	Trip Count	Pipelined	FF	LUT
HLS Design	22784	$2.278 \cdot 10^5$	-	22785	-	✖	2283 (2%)	3496 (6%)
• PCA Iterative Loop	22656	$2.266 \cdot 10^5$	59	-	384	✖	-	-
∘ PC Inner Module	50	500	-	50	-	✖	297 (≈ 0%)	175 (≈ 0%)
∘ PC Inner Loop	48	480	12	1	38	✓	-	-
• PCA Scaler Module	71	710	-	71	-	✖	506 (≈ 0%)	484 (≈ 0%)
∘ PCA Scaler Loop	69	690	33	1	38	✓	-	-
• Output Module	41	410	-	41	-	✖	44 (≈ 0%)	73 (≈ 0%)
∘ Output Loop	39	390	3	1	38	✓	-	-

From Table 7, the synthesized HLS RTL considers an output buffer implemented as a true dual-port RAM used to store partial results processed by intermediate pipelines. Furthermore, each processing module makes use of the single-precision floating-point multiplication/addition/subtraction using DSP-based and/or logical fabric implementation for a total consumption of 7 DSPs (also reported on Table 7). The PCA and Data Scaler modules also implement ROM storage elements for constant values such as mean values coefficients and principal components matrix coefficients for a total consumption of 35 BRAMs units.

Table 7 – HLS Synthesis Binding Operator/Storage Summary

Hierarchy	Op./Storage	Impl.	DSP	BRAM	Latency
Functional Addition × 2	`add`	`fabric`	-	-	0
Single-Precison Subtraction	`fsub`	`fulldsp`	2	-	4
Mean Values	`rom_1p`	`BRAM`	-	1	1
• **PCA Iterative Module**					
Functional Addition × 2	`add`	`fabric`	-	-	0
Principal Components Matrix	`rom_1p`	`BRAM`	-	32	1
• **Data Scaler Module**					
Functional Addition	`add`	`fabric`	-	-	0
Single-Precison Division	`fdiv`	`fabric`	-	-	15
Single-Precison Add/Sub.	`fadd/fsub`	`fulldsp`	2	-	4
• **Data Output Module**					
Functional Addition	`add`	`fabric`	-	-	0
Output Buffer	`ram_s2p`	`BRAM`	-	2	1
TOTAL			7	35	

3.3.2 Feedforward Neural Network Model – HLS-FFNN

A Feedforward Neural Network (FFNN) was developed for the proposed DPR system and will be used in magnet localization receiving reduced features from the PCA module and will produce an estimate of magnet displacement as an output. Once again, this development was done using Vitis HLS and followed the same design flow as the one used to develop the HLS-based PCA IP core.

This HLS-FFNN design is one of the reconfigurable modules (RM) in the DPR system and is constrained by the available fabric resources for its reconfigurable partition (RP). Therefore, special attention was given to utilization and timing constraints so that placement/routing requirements would be met during the DPR design implementation.

The HLS-based FFNN uses AXI Stream interfaces for receiving and transmitting 32-bit data that is used in single-precision floating-point processing pipelines. These stream interfaces also implement the `TLAST` side-channel signal to indicate the boundary of a packet. Figure 19 shows the overall FFNN three-layer structure. The hidden layer uses a hyperbolic tangent sigmoid activation function and the output layer uses a linear activation function.

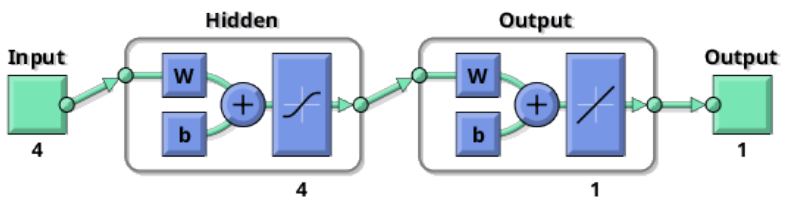


Figure 19 – Feedforward Neural Network with a three-layer structure. The numbers indicate that 4 neurons are used in the hidder layer and a single neuron is used in the output layer.

The HLS-FFNN design implements four pipelines: (i) input propagation into hidden layer weight multiplication; (ii) hidden layer bias accumulation; (iii) hidden layer activation function; and (iv) output layer weight multiplication with accumulation. In the first pipeline, input data is multiplied by the hidden layer weights in loop re-ordered fashion to allow pipelining the inner accumulation loop. The results are passed to the following pipeline for bias accumulation. The hidden layer output is then obtained after intermediate values are passed to the sigmoid (hyperbolic tangent) activation pipeline. The output layer performs weight multiplication and accumulation in a single pipeline to compute the network output.

During development, experimental design space exploration was used to obtain different levels of parallelism (by unrolling loops) and different pipeline organizations (by merging pipelines) searching for a feasible RTL design that would meet utilization, timing, and latency constraints. For instance, the pipelines for the hidden layer bias accumulation and activation function could be unrolled for greater parallelism and reduced latency but it would require extensive fabric resources to be implemented.

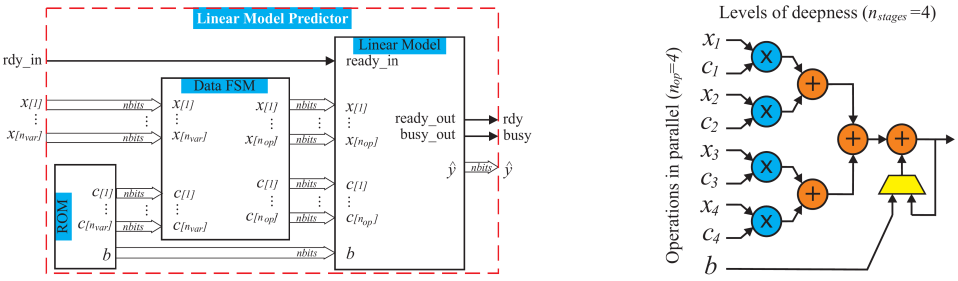
Due to its small scale, all layer weights/bias of its 4 neurons are stored in partitioned arrays using fabric resources. The hyperbolic tangent sigmoid is implemented inline using the equivalence $\tanh(N) = 2/(1 + \exp(-2 \cdot N)) - 1$. This optimization using a single-precision exponential operator reduced resource utilization for the pipeline used in the activation function for the hidden layer in comparison with the standard hyperbolic tangent operator available from Vitis HLS math library.

3.3.3 Linear Regression Model – pLinRGen

A Linear Regression Model (**pLinRGen**) was previously developed in (MENDEZ et al., 2022) with its floating-point operators (multipliers and adders) based on pre-characterized FPLib IP-Cores (MUÑOZ et al., 2010a,b). In addition, FPLib supports parametric floating-point bit-width representation. A trade-off analysis between hardware consumption, computational performance, numerical precision, and latency was performed in order to provide the optimal physical implementation in (MENDEZ, 2021).

In this work the **pLinRGen** was improved by supporting AXI Stream interfaces for receiving and transmitting data. The required changes in the RTL architecture were verified using automatic testbenchs through behavioral and timing simulations. In-circuit tests were also performed to validate the design using the AXI-DMA to perform data transfers. In addition, several scripts were developed to allow IP testing/exporting using the Vivado Design Suite. All changes were submitted to the `LEIA UnB - pLinRGen` private repository.

The overall architecture of the Linear Regression Model is shown below in Figure 20. For a more comprehensive description, the reader is referred to (MENDEZ, 2021).



(a) Linear Regression Top Module View (b) Linear Model Pipeline Architecture

Figure 20 – Linear Regression Model (**pLinRGen**) – 20a: Top-level view with ROM, Control State Machine and Linear Model Pipeline. 20b: Internal pipeline representation. Adapted from (MENDEZ et al., 2022).

The **pLinRGen** tool allows the HDL (VHDL-based) code generation of Linear Regression Models with an internal pipeline. It can be configured through a set of parameters to produce different model schemes, its parameters include:

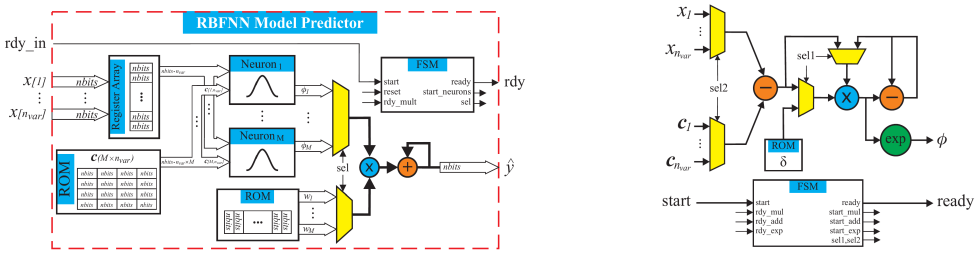
- fp_{width}: The floating-point bit-width representation.
- n_{var}: Number of input variables.
- c_N: Regression coefficients.
- n_{op}: Number of parallel operators from which the number of pipeline stages are derived.

After code generation using the desired set of parameters and the selection of regression coefficients derived from the magnetic tracking training dataset, the AXI-Stream version of the **pLinRGen** IP is exported to the Vivado IP repository for usage in the DPR system implementation.

3.3.4 Radial-Basis Function Network Model – vRBFGen

A Radial-Basis Function Neural Network Model (**vRBFGen**) was previously developed in (AYALA, H. V. H. et al., 2017; AYALA, H. et al., 2016) with its floating-point operators (multiplication, addition, subtraction, exponentiation) based on pre-characterized FPLib IP-Cores (MUÑOZ et al., 2010a,b) with support to parametric floating-point bit-width representation.

In this work, the **vRBFGen** was improved by supporting AXI Stream interfaces for receiving and transmitting data. The required changes in the RTL architecture were verified using automatic testbenchs through behavioral and timing simulations. In-circuit tests were also performed to validate the design using the AXI-DMA to perform data transfers. In addition, a set of scripts was developed to allow IP testing/exporting using the Vivado Design Suite. All changes were submitted to the `LEIA UnB - vRBFGen` private repository.



(a) RBFNN Top Module View

(b) RBF "Neuron" Instance Architecture

Figure 21 – RBFNN Regression Model (**vRBFGen**) – 21a: Top-level view with ROMs, Control State Machine and RBF "Neuron" instances. 21b: RBF "Neuron" with Control State Machine. Adapted from (MENDEZ et al., 2022).

The **vRBFGen** tool provides an HDL (VHDL-based) code generation of Radial-Basis Function Network Models using parallel neurons. It can be configured through a set of parameters to produce different model structures, including:

- fp_{width}: The floating-point bit-width representation.
- n_{var}: Number of input variables.
- M: Number of hidden-layer neurons.
- c_n: $M \times n_{\text{var}}$ vector with the centers for every neuron.
- δ: $M \times 1$ vector for neurons spreads (obtained using $1/(2\sigma^2)$).
- w: $M \times 1$ vector with output layer weights.

3.4 Run-time Reconfigurable System Implementation

This study aims to provide an RTR design capable of tracking two magnets concomitantly (for a total of five magnets) with reduced hardware utilization for the implementation of data-driven models used in magnet tracking. This is done by using two reconfigurable partitions and by constraining the area of these partitions to reduce reconfiguration time. As the reconfiguration time poses an overhead in the total execution time, we are interested in mitigating this by using an RP scheduling scheme as shown in Fig. 22.

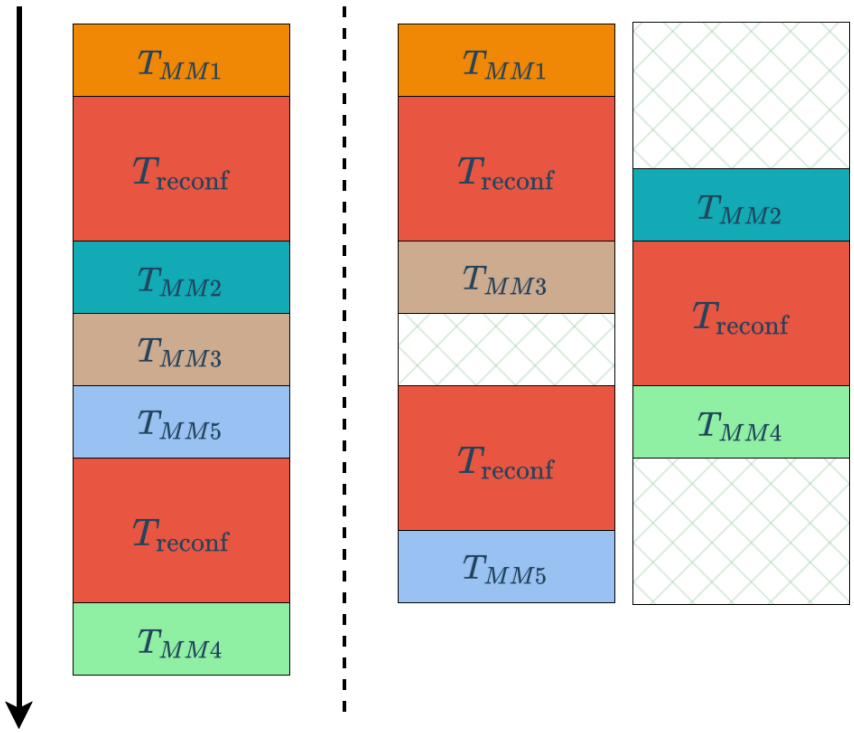


Figure 22 – Scheduling of five magnet tracking using a single RP on the left and two RPs on the right. The scheme with a single RP was taken from (MENDEZ, 2021). The scheme with two RPs is the proposed scheme to reduce the total execution time and increase parallelism. It should be noted that the illustration does not present any time scale for the reconfiguration time and model latency.

In Fig. 22, $T_{MM1}, T_{MM2}, T_{MM3}, T_{MM4}, T_{MM5}$ denote the execution time for tracking magnets 1 to 5, and T_{reconf} is the reconfiguration time for loading a new RM into the RP. In the work of (MENDEZ, 2021), a single RP is used to sequentially load and process each magnet localization. A single module (RM) is shared for magnets 2, 3, and 5 as the same Linear Regression model is used avoiding reconfiguration steps.

Fig. 22 shows how the processing of magnet tracking is done for the proposed dual RP scheme. The performance gain offered by the architecture with two RPs is evident with the capability to mitigate the reconfiguration time and increase parallelism. By using two RPs, we are able to process the sequential magnet localization while the other accelerator is still under reconfiguration, effectively *hidding* the computational latency for magnet localization given that this latency is below the reconfiguration time for the RM.

The proposed design implements a "hardware pipelining" or "hardware pre-fetching" in the RTR architecture with dual RPs. However, this performance gain is not for free, since the system is more complex than the one with a single RP: it requires more hardware resources to be implemented.

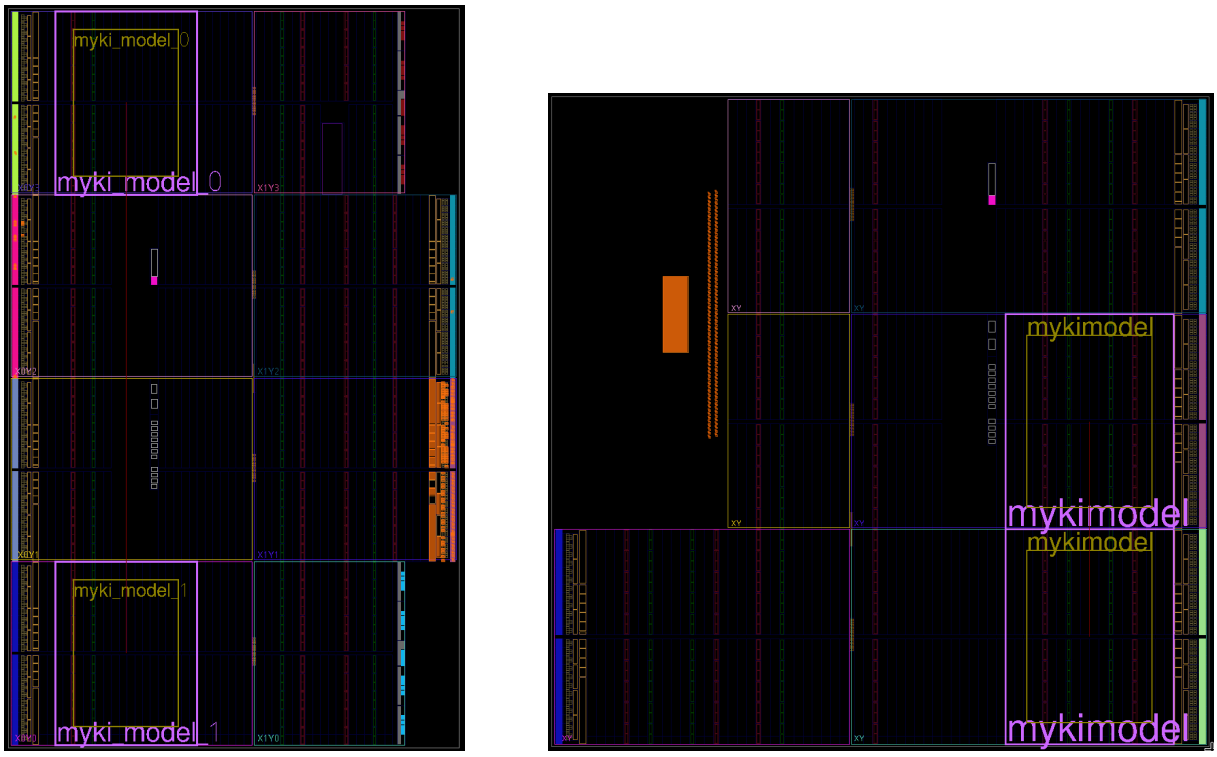
As a longer reconfiguration time is not tolerable, a straightforward way to lower the impact of the reconfiguration overhead is to reduce the dedicated area for the programmable

block used by the RP. In the following, we present the conceptual design (floorplanning, system overview, and block design) used in the implementation of the RTR architecture under the FPGA and SoC-FPGA platforms.

3.4.1 Floorplanning

In the DPR design flow, the floorplanning comprises the allocation of programmable blocks for each RP. This step was done following guidelines presented in Xilinx UG909 for 7-Series Devices (UG909…, 2022). The height for each RP is aligned with clock region boundaries to apply reset after reconfiguration functionality. This option holds internal signals steady during reconfiguration, then issues a masked global reset to the reconfigured logic.

In depth, an entire clock region contains 3200 Slices for the SoC-FPGA and 2600 Slices for the FPGA device. Using an entire clock region for each RP would lead to an undesirable high resource utilization. Thus, the width of the RP was set so that the RP would encompass 1600 Slices for the SoC-FPGA (PYNQ-Z2) and 1700 Slices for the FPGA (ARTY A7) device (about half the area of an entire clock region). By constraining the available area/resources in the RP, it is possible to iterate the selection of model parameters (number of neurons/operators) so that hardware implementations are feasible. Figure 23 depicts the proposed floorplanning.



(a) FPGA (ARTY A7 – *xc7a100tcsg324-1*) (b) SoC-FPGA (PYNQ-Z2 – *xc7z020clg400-1*)

Figure 23 – Proposed floorplanning for FPGA and SoC-FPGA with two Reconfigurable Partitions (RPs).

3.4.2 System Overview & Block Design

The FPGA and SoC-FPGA designs were implemented in Vivado 2022.2 using a Block Design (IP Integrator) flow (UG892…, 2022). In addition, the DFx (Dynamic Function eXchange) design flow and IPs are used to provide run-time reconfiguration capabilities to the design using the floorplanning presented in the previous Section 3.4.1. The DFx IPs used are the DFx Reconfiguration Controller and DFx Decoupler. Briefly, the Reconfiguration Controller delivers the partial bitstream to the ICAP and uses the AXI-HP bus attached to the DDR memory as its data source. The DFx Decoupler is used to decouple signals from/to the RP, avoiding indefinite states in the programmable logic.

Each one of the five RMs is implemented as a child block diagram in a parent block diagram which implements the static logic. The generated RTL modules for the RBFNN, FFNN, and Linear Regression are packaged into AXI-Stream compatible IPs to be implemented in the reconfigurable partitions. The proposed HLS-based PCA IP will be implemented in the static (non-reconfigurable) logic.

The data transfers to/from the RMs are performed by AXI-DMA (Direct Memory Access) blocks. Each RP contains a single input and single output AXI-Stream channel used to provide the input/output data for each model. The HLS PCA IP uses AXI Memory-Mapped interfaces to transfer incoming data from DDR memory and outputs data features used as model inputs. Regarding data storage, an SD card is used to store partial bitstreams and unprocessed magnetic sensor data. The proposed system uses the DDR memory to cache the bitstreams and data buffers used during the system application execution.

Each embedded platform (either FPGA or SoC-FPGA) has its own means of implementing system-level functionalities. Figure 24 presents the Block Design for the SoC-FPGA platform.

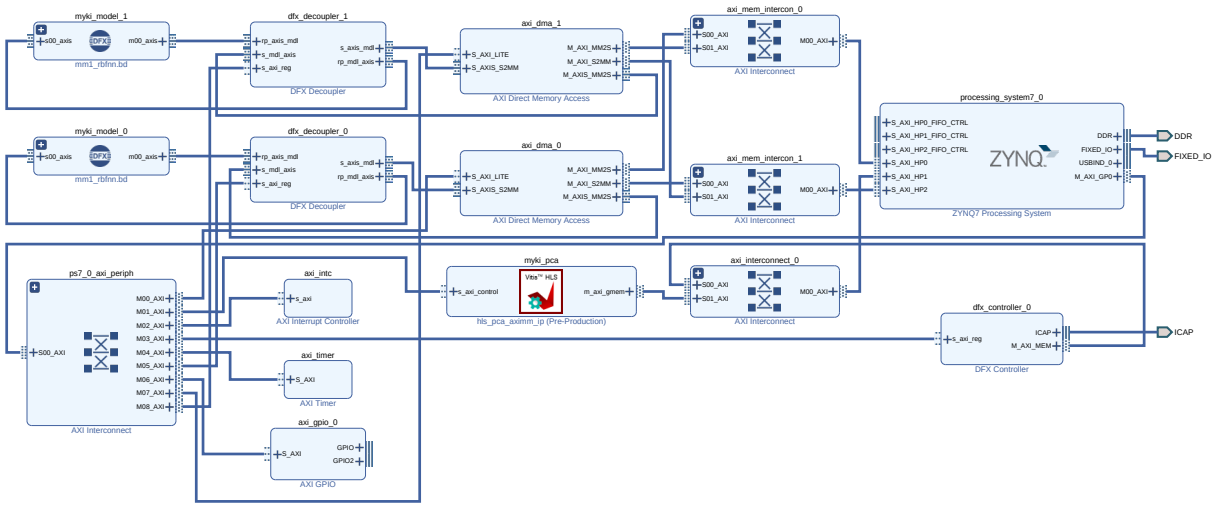


Figure 24 – Vivado SoC-FPGA Block Design for the RTR implementation for the proposed myokinetic interface localizer. The diagram is presented with the "Interfaces View" in which block connections are AXI interfaces.

From the SoC-FPGA Block Design (Fig. 24), the ZYNQ7 Processing System (PS) is configured with: AXI General-Purpose Port 0 for peripherals AXI-Lite control interfaces; AXI High-Performance Interfaces 0 and 2 are connected to AXI-DMAs 1 and 2; AXI High-Performance Interface 1 is connected to the HLS PCA IP and DFx Controller. The SD card and DDR memory are also enabled by default and a clock of 100 MHz is used to supply the PL clock used across all AXI Peripherals and ICAP interface. In addition, the following IPs are also used:

- AXI Interrupt Controller: Interrupt signals from the HLS PCA IP and AXI-DMAs are connected to the PS `IRQ-F2P` Interrupt Port. This is a core component used to check, enable and acknowledge interrupts using AXI-Lite registers.
- AXI Timer: This timer is used to capture events and measure time intervals such as the model execution latency and reconfiguration time.

The platform software is a standalone application (bare-metal). During initialization, the partial bitstreams and the experimental dataset are loaded into the DDR memory from the SD-card storage. The ARM allocates memory buffers used to send/receive data from the PCA pre-processing stage and memory buffers used to send/receive RM data through the AXI-DMA. The system is prepared to perform reconfiguration by software triggers and uses the interrupts from the PCA IP and AXI-DMAs to control the data flow and reconfiguration.

The FPGA design for the RTR architecture requires the implementation of a soft-core processor (MicroBlaze) and a DDR memory controller (MIG 7-Series) in the programmable logic. In addition, a PMOD Micro-SD is attached to the FPGA development kit to allow the host board to read/write nonvolatile storage over an SPI interface.

The overall software capabilities for the FPGA design are limited in comparison with the SoC-FPGA design which contains an ARM-based processing system. This comes at the cost of power consumption as the SoC-FPGA is more power-demanding than its FPGA counterpart. Figure 25 presents the block design for the proposed FPGA-based RTR architecture.

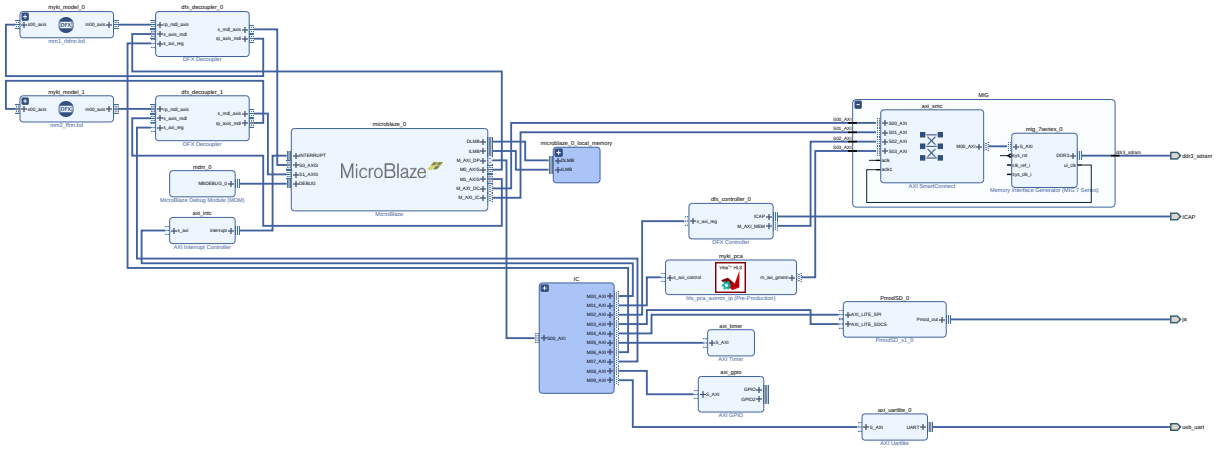


Figure 25 – Vivado FPGA Block Design. The diagram is presented with the "Interfaces View" in which block connections are AXI interfaces.

The FPGA design uses the MicroBlaze soft-core processor with a customized Real-Time preset. The DDR memory is interfaced through the Memory Interface Generator (MIG 7-Series) and is used as data and instruction cache for the soft-core processor. The MicroBlaze is configured to provide 2 AXI-Stream interfaces with input/output ports for data transfers with the RPs. These stream interfaces can be used efficiently with assembly instructions for blocking and non-blocking data transfers.

The AXI Interrupt Controller and AXI-Timer are also used with the same purpose as in the SoC-FPGA platform. The Pmod-SD IP is used to interface with the attached Micro-SD Card Slot and the AXI-UARTLite provides serial communication used to debug the standalone application (bare-metal) for the MicroBlaze.

The RTL description (in VHDL), IP sources, and TCL build scripts were added to the `LEIA UnB - MYKI` private repository along with full, and partial bistreams used in the aforementioned implementation. Also, the C-source code used in the standalone application was also added.

4 Run-Time Reconfigurable Architecture Implementation Results

This Chapter is dedicated to the analysis and discussion of the SW/HW co-design implementation in both the SoC-FPGA and FPGA platforms. Firstly, relevant metrics regarding design utilization for the implemented FPGA and SoC-FPGA platforms described in Sections 4.1, 4.2 are presented. Furthermore, Section 4.3 presents results for the performance metrics regarding latency, reconfiguration time and throughput, power consumption, and computational efficiency.

4.1 Reconfigurable Modules Hardware Utilization

Based on the models from Table 5, the synthesis and implementation were carried out considering: a) $\text{fp}_{\text{width}} = 27$ bits (floating-point bit-width) as it reduces the required number of DSP blocks used for the operators used in the RBFNN and Linear Regression models; b) the FFNN uses standard $\text{fp}_{\text{width}} = 32$ (single-precision) for its operators; c) The number of operators for the Linear Regression model was $n_{\text{op}} = 8$ due to the RP area constraint; d) Both the RBFNN and FFNN contain 4 neurons in the hidden layer.

Table 8 and 9 present the total hardware utilization for the implementation of the proposed models for the FPGA and SoC-FPGA platforms, respectively. It can be seen that the RBFNN is occupying more than 95% of RM Slices. Reducing the RP area beyond the current allocation of approx. 1600/1700 Slices might not be viable for the implementation of all RMs.

Table 8 – Reconfigurable Modules (RMs) Implemented Utilization Summary – FPGA: ARTY A7

Reconfigurable Module	**Reconfigurable Partition (Pblock)**				
	Slice Registers (13600)	Slice LUTs (6800)	Slice (1700)	DSPs (20)	Block RAM Tile (10)
MM1: RBFNN	2400 (17.65%)	5966 (87.74%)	1622 (95.41%)	5 (25%)	0 (0%)
MM2: FFNN	2462 (18.10%)	2243 (32.99%)	944 (55.53%)	14 (70%)	0.5 (5%)
MM3: FFNN	2455 (18.05%)	2233 (32.84%)	943 (55.47%)	14 (70%)	0.5 (5%)
MM4: Lin. Reg.	2480 (18.24%)	3281 (48.25%)	1180 (69.41%)	8 (40%)	0 (0%)
MM5: Lin. Reg.	2480 (18.24%)	3289 (48.37%)	1239 (72.35%)	8 (40%)	0 (0%)

Table 9 – Reconfigurable Modules (RMs) Implemented Utilization Summary – SoC-FPGA: PYNQ-Z2

Reconfigurable Module	**Reconfigurable Partition (Pblock)**				
	Slice Registers (12800)	Slice LUTs (6400)	Slice (1600)	DSPs (35)	Block RAM Tile (20)
MM1: RBFNN	2400 (18.75%)	5905 (92.27%)	1576 (98.5%)	5 (12.5%)	0 (0%)
MM2: FFNN	2231 (17.43%)	2219 (34.67%)	937 (58.56%)	14 (35%)	0.5 (2.5%)
MM3: FFNN	2224 (17.38%)	2208 (34.5%)	878 (54.88%)	14 (35%)	0.5 (2.5%)
MM4: Lin. Reg.	2480 (19.38%)	3317 (51.83%)	1258 (78.63%)	8 (20%)	0 (0%)
MM5: Lin. Reg.	2480 (19.38%)	3317 (51.83%)	1166 (72.88%)	8 (20%)	0 (0%)

From the RMs physical implementation, it is possible to derive the partial bitstream size for each target device used for each RM. The total reconfiguration time is directly related to the size of the partial bitstream and the bandwidth of the configuration port used. For 7-Series devices, both PCAP (Processor Configuration Access Port) and ICAP (Internal Configuration Access Port) are limited to 3.2 Gb/s (32-bit data width with 100 MHz clock) (UG909…, 2022).

Table 10 presents the size of partial bitstreams from the proposed floorplanning and the associated reconfiguration time considering the maximum bandwidth. To achieve the maximum bandwidth, the memory bank providing the partial bitstream data must be capable of handling such transfers. The SoC-FPGA (PYNQ-Z2) provides a DDR3 memory and the FPGA (ARTY A7) provides a DDR3L memory, which can be used to store the partial bitstreams and deliver them to the desired configuration port.

Table 10 – Target Device Partial Bitstream Sizes

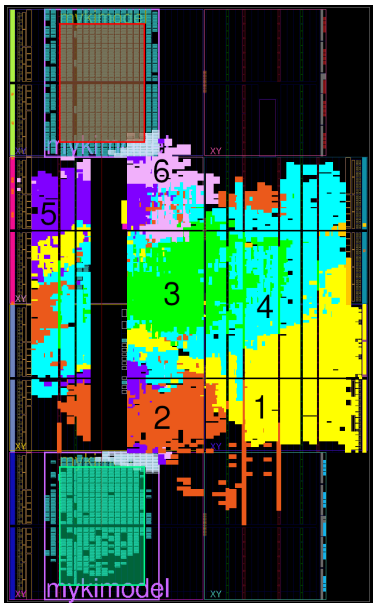
Reconfigurable Partition	*SoC-FPGA PYNQ-Z2*	*FPGA ARTY A7*
Partial Bitstream Size [bytes]	857740	761588
Theoretical Reconfiguration Time @ 400 MB/s	2144.35 μs	1903.97 μs

From a system-level perspective, the reconfiguration time is the determinant factor when accounting for the total execution time for the five-magnet tracking as the reconfiguration time is expected to be much longer than the hardware accelerator latency.

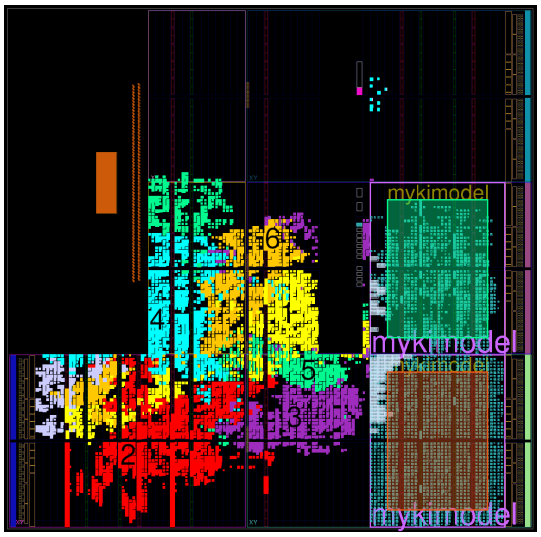
4.2 DPR System Design Utilization

Tables 11 shows the SoC-FPGA hardware utilization and 12 shows the FPGA hardware utilization for the implementations of the block designs (Fig. 25, 24), respectively. Both implementations share the same set of RPs and RMs. The FPGA implementation demands

more resources (almost twice the FFs and LUTs occupation) as it implements the MicroBlaze soft-core processor and the DDR memory interface infrastructure.



(a) FPGA Device View with highlights. Labeled Regions – (1) DDR Interface; (2) HLS-PCA IP; (3) MicroBlaze Core; (4) AXI IPs; (5) DFX Controller & Decoupler; (6) PMOD-SD IP.



(b) SoC-FPGA Device View with highlights. Labeled Regions – (1) AXI-DMA; (2) HLS-PCA IP; (3) DFX Controller & Decoupler; (4) AXI Interconnect; (5) AXI IPs; (6) AXI-DMA.

Figure 26 – Implemented Designs: Device View with highlights and labels.

The proposed floorplanning with reduced area for the RPs proved successful as the FPGA design only occupies ∼43% of total slices and the SoC-FPGA design ∼31% of total slices. Figure 26 shows the device view with highlights.

In (MENDEZ, 2021), the FFNN model was also implemented using HLS considering a parallel neuron structure with a total utilization of 7188 LUTs, and 6354 FFs. In this work, the implementation considers a less parallel approach but with efficient pipeline processing with a total utilization of 3281 LUTs, and 2455 FFs. The present implementation saves logic resources in comparison with the previous implementation.

From the utilization results, it is possible to conclude that the RBFNN model requires a considerable amount of resources for its implementation (about 95% of all RP Slices). The other models reach up to 80% of RP Slice utilization. This is an indication that the RP area could be further reduced to save on logic utilization, partial bitstreams storage, and power dissipation if the RBFNN implementation were to be optimized.

Table 11 – DPR Design Utilization on SoC-FPGA

Hierarchy	SoC-FPGA Implemented Design Utilization									
	Slice Registers (106400)		Slice LUTs (53200)		Slice (13300)		DSPs (220)		Block RAM (140)	
Total – Static Logic	11476	10.80%	9618	18.04%	4188	30.96%	7	3.18%	25.0	17.85%
HLS-PCA	2497	2.35%	2164	4.07%	848	6.38%	7	3.18%	18.5	13.21%
AXI-DMA 0	2094	1.97%	1442	2.71%	660	4.96%	0	0%	3	2.14%
AXI-DMA 1	2032	1.91%	1442	2.65%	632	4.75%	0	0%	3	2.14%
DFX Controller	1520	1.43%	1333	2.51%	545	4.10%	0	0%	0.5	0.36%
AXI Interconnect 0	1344	1.26%	1213	2.28%	466	3.50%	0	0%	0	0%
PS7 AXI Interconnect	664	0.62%	728	1.37%	378	2.84%	0	0%	0	0%
AXI Interconnect 0	501	0.47%	411	0.77%	177	1.33%	0	0%	0	0%
AXI Interconnect 1	350	0.33%	361	0.68%	168	1.26%	0	0%	0	0%
AXI Timer	240	0.23%	289	0.54%	110	0.83%	0	0%	0	0%
AXI Int. Controller	98	0.09%	97	0.18%	34	0.26%	0	0%	0	0%
AXI GPIO	96	0.09%	43	0.08%	35	0.26%	0	0%	0	0%
PS7 System Reset	33	0.03%	17	0.03%	13	0.10%	0	0%	0	0%
DFX Decoupler 0	7	<0.01%	39	0.07%	25	0.19%	0	0%	0	0%
DFX Decoupler 1	7	<0.01%	39	0.10%	27	0.20%	0	0%	0	0%
Reconfigurable Modules										
MM1: RBFNN	2400	2.26%	5905	11.10%	1576	11.85%	5	2.27%	0	0%
MM2: FFNN	2231	2.10%	2219	4.17%	937	7.05%	14	6.36%	0.5	0.36%
MM3: FFNN	2224	2.09%	2208	4.15%	878	6.60%	14	6.36%	0.5	0.36%
MM4: Lin. Reg.	2480	2.33%	3317	6.23%	1258	9.46%	8	3.64%	0	0%
MM5: Lin. Reg.	2480	2.33%	3317	6.23%	1166	8.77%	8	3.64%	0	0%

Table 12 – DPR Design Utilization on FPGA

Hierarchy	FPGA Implemented Design Utilization									
	Slice Registers (126800)		Slice LUTs (63400)		Slice (15850)		DSPs (240)		Block RAM (135)	
Total – Static Logic	19318	15.24%	17745	28.00%	6753	42.60%	12	5.00%	40.5	30.00%
DDR Mem. Interface	10809	8.52%	9362	14.77%	3488	22.01%	0	0%	0	0%
AXI SmartConnect	6735	5.31%	4750	7.49%	1947	12.28%	0	0%	0	0%
MIG 7-Series	4074	3.22%	4615	7.28%	1571	9.91%	0	0%	0	0%
MicroBlaze	2764	2.18%	3388	5.34%	1148	7.24%	5	2.08%	20	14.81%
HLS-PCA	2729	2.15%	2148	3.39%	934	5.89%	7	2.92%	18.5	13.70%
DFX Controller	1517	1.20%	1379	2.18%	535	3.38%	0	0%	0	0%
PMOD-SD IP	585	0.46%	394	0.62%	171	1.08%	0	0%	0	0%
AXI Timer	240	0.19%	289	0.46%	105	0.66%	0	0%	0	0%
AXI Int. Controller	170	0.13%	119	0.19%	44	0.28%	0	0%	0	0%
AXI Interconnect	129	0.10%	301	0.47%	129	0.81%	0	0%	0	0%
MicroBlaze Debug	110	0.09%	93	0.15%	41	0.26%	0	0%	0	0%
AXI UART-Lite	107	0.08%	89	0.14%	33	0.21%	0	0%	0	0%
AXI GPIO	96	0.08%	43	0.07%	37	0.23%	0	0%	0	0%
PS System Reset	34	0.03%	17	0.03%	13	0.08%	0	0%	0	0%
MicroBlaze Memory	14	0.01%	17	0.03%	11	0.07%	0	0%	2	1.48%
DFX Decoupler 0	7	<0.01%	53	0.08%	34	0.21%	0	0%	0	0%
DFX Decoupler 1	7	<0.01%	53	0.08%	30	0.19%	0	0%	0	0%
Reconfigurable Modules										
MM1: RBFNN	2400	1.89%	5966	9.41%	1622	10.23%	5	2.08%	0	0%
MM2: FFNN	2462	1.94%	2243	3.54%	944	5.96%	14	5.83%	0.5	0.37%
MM3: FFNN	2455	1.94%	2233	3.52%	943	5.95%	14	5.83%	0.5	0.37%
MM4: Lin. Reg.	2480	1.96%	3281	5.18%	1180	7.44%	8	3.33%	0	0%
MM5: Lin. Reg.	2480	1.96%	3289	5.19%	1239	7.82%	8	3.33%	0	0%

4.3 DPR System Performance Evaluation

For the performance evaluation of the proposed architectures, it is worthwhile to analyze computational latency, reconfiguration time and throughput, power consumption, and computational efficiency metrics.

The SoC-FPGA architecture uses the ARM core as its main processing unit, while the FPGA design uses the MicroBlaze soft-core. Despite its hardware differences, the software implementation is somewhat similar. The system works based on interrupts triggered in the PL by the HLS-PCA IP and AXI-DMA IPs, the AXI Interrupt Controller is used to manage these interrupt signals. In both implementations, the platform software is bare-metal (standalone) and time interval measurements are taken using the AXI-Timer implemented in the PL, with the capability to capture trigger inputs from the aforementioned interrupt signals. This allows measuring model execution latency, PCA inference latency, and reconfiguration time.

Table 13 shows the measured computational latency for the inference of each model and the reconfiguration time and estimated reconfiguration throughput. It can be seen that the MicroBlaze-based design increases the computational latency for the magnet localization tasks. This is mostly due to the fact the MicroBlaze Core is not able to transfer data at every clock cycle. The stream link from the MicroBlaze to the RP can only transfer data at every 2 clock cycles, while the SoC-FPGA design uses the AXI-HP port and AXI-DMA IP capable of transfer bursts for every clock cycle. It should be noted that both designs use the same frequency of 100 MHz for the PL clock domain. However, the measured latency is acceptable and won't affect system-level performance. It is worth noticing that these measured values are close to the values presented in Table 10 which were estimated using the size of the RPs.

In addition, the reconfiguration throughput is near the theoretical limit of 400 MB/s in both designs. The work of (MENDEZ, 2021) was only able to achieve about 250 MB/s (estimated) of reconfiguration throughput in its implementation (reconfiguration time of 4740.8 μs). From the results, one can conclude that the reconfiguration time still poses an overhead in the total execution time but with the adoption of 2 RPs this effect is partially mitigated.

Table 13 – DPR System Computational & Reconfiguration Latency

Target	**Measured Latency** [μs]					**Reconfiguration Metrics**	
	MM1	**MM2**	**MM3**	**MM4**	**MM5**	**Time** [μs]	**Throughput** [MB/s]
SoC-FPGA	3.8636	7.6421	6.2549	3.8671	3.8636	2145.627	399.76
FPGA	6.0184	13.4654	9.5670	8.5955	8.5948	1907.136	399.33

The latency for the HLS-PCA IP was estimated at 227.84 μs during synthesis. In the SoC-FPGA, the measured latency was 229.04 μs, and on its counterpart (FPGA) was

237.18 μs. As the pre-processing is done on static logic, it can be processed concurrently with magnet localization by each RM.

A total execution time can be calculated using the proposed RP scheduling scheme (Fig. 22) and the measured latency seen from the results and considering that in the worst-case scenario, the PCA inference needs to be sequentially processed before starting magnet localization.

$$
\begin{aligned}
T_{\text{total}} &= T_{PCA} + T_{MM1} + T_{\text{reconfig}} + T_{\text{reconfig}} + T_{\text{reconfig}} + T_{MM5} \\
T_{\text{total (SoC-FPGA)}} &\approx 229.04 + 3.86 + 3 \times 2145.62 + 3.86 \approx 6673.62\ \mu s \\
T_{\text{total (FPGA)}} &\approx 237.18 + 6.01 + 3 \times 1907.13 + 8.59 \approx 5973.17\ \mu s
\end{aligned}
$$

This leads to an approximation for the total execution time for the SoC-FPGA of 6.67 ms and 5.97 ms for the FPGA-based solution. This latency considers the five-magnet tracking tasks and the PCA inference used in the pre-processing stage of magnetic data acquisition. In previous work, (MENDEZ, 2021) obtained approx. 9.5 ms as the DPR solution had to be reconfigured twice for tracking five magnets.

Table 14 presents the power consumption summary for the proposed architectures. The total dynamic power consumption for the SoC-FPGA is 1.493 W, within which 1.257 W are dedicated to the PS. From that, 270 mW is used for the ARM Cortex-A9 dual-core; 630 mW for the DDR3 memory; 16 mW for I/O peripherals; and 344 mW for clocking resources (Processor, Memory, I/O). The FPGA-based design uses nearly 1W of dynamic power. From that, 685 mW are dedicated to the DDR3 Memory Interface; 114 mW to the MicroBlaze soft-core; and 108 mW for clocking resources. In previous work, (MENDEZ, 2021) obtained 1718 mW of total power consumption using a similar SoC-FPGA device.

Table 14 – DPR System Power Summary

Target	**On-Chip Power [W]**			**Reconfigurable Modules [mW]**				
	Dynamic	**Static**	**Total**	**MM1**	**MM2**	**MM3**	**MM4**	**MM5**
SoC-FPGA	1.493	0.142	1.635	60	70	68	53	55
FPGA	1.094	0.108	1.203	80	73	73	48	49

With these metrics for latency and power consumption, it is possible to devise a measure of the number of magnet localizations per second (f_L) per Watt to provide insights into the overall computational efficiency of each solution. The proposed FPGA-based solution is able to achieve 696.19 f_L/W; the SoC-FPGA-based solution 458.48 f_L/W; (MENDEZ, 2021) obtained 305.53 f_L/W. In (IANNICIELLO, 2024), if we consider the minimum power consumption (single AU configuration) of $\sim$ 600 mW and the computational time of 21.3

ms (worst-case scenario per iteration) for tracking eight magnets, this yields $\approx 625.97\ f_L$/W. These baseline results for power efficiency are a clear indicator that the proposed solution on the FPGA platform presents a clear appeal regarding its reduced power consumption and execution time for five-magnet tracking.

While the work of (IANNICIELLO, 2024) is a more complete representation of a transcutaneous magnet localizer for the myokinetic interface, the proposed run-time reconfigurable architecture demonstrated to be capable of real-time operation with accurate localization of multiple magnets. In the proposal, we also demonstrated the complete magnetic data processing pipeline done at the RTL level using a combination of HLS-based IPs and HDL-based IPs. The measured reconfiguration throughput was found to be as close as possible to the theoretical limit for the interface. This reduces the reconfiguration overhead as partial bitstreams can be loaded faster into the RP. The pre-fetching scheme is a considerable advance from previous work (MENDEZ, 2021), as it enables parallel magnet tracking and mitigates the overhead.

It is also worthwhile noting that given the design utilization from the proposed implementation, it is still viable to implement additional modules. These modules could be implemented in parallel for the prosthetic control of artificial hands, such as the UnB-LEIA Hand (PERTUZ; LLANOS; MUNOZ, 2021). For the prosthetic control, there are several systems for filtering, parametric estimation, position and torque control for grasp execution. These tasks could be executed in parallel fashion using a FPGA-based system. From a software perspective, implementing all these tasks might not be viable while using a single microprocessor given the real-time, power requirements for the development of artificial hands.

5 Final Remarks

This brief chapter presents the conclusions and future works derived from the initial research questions, developments and obtained results during the research period.

5.1 Conclusions

The present work was dedicated for attaining the objectives set out in Chapter 1.

First, the hardware implementation of a FPGA-based solution in place of a SoC-FPGA solution proved to be a good decision based on the actual obtained results for power consumption and computational latency. The Zynq SoC-FPGA is a powerful processing unit but given the self-contained nature of the transcutaneous magnet localizer, reduced power consumption is highly desirable. One limitation that must be considered is the need for an external high-speed DDR memory to provide storage capabilities for partial bitstreams. This comes at the expense of power dissipation for the interface and memory controller.

Furthermore, the software application is standalone (bare-metal) and all the data processing is done on the FPGA fabric. One could conclude that the ARM processing cores are sub-utilized and the need for such processing power might not cover for the additional power expense. Also, the MicroBlaze soft-core processor demonstrated similar performance, providing an integrated solution to the RPs using the AXI4-Stream links.

Regarding the DPR design, the RP area is in direct relation to the partial bitstream size, which is strictly tied to the reconfiguration time. The proposed DPR design with two RPs is capable of reducing the reconfiguration overhead but could still benefit from a reduction on reconfiguration time. In fact, the upper bound to the RP area was the logic utilization of the RBFNN model, which implements every neuron in parallel. Also, the proposed solution was carefully designed to achieve the near maximum reconfiguration throughput (400 MB/s).

5.2 Future Work

During the research, some improvement points were noted and could provide interesting research ideas for future works.

With dedicated effort, the proposed solution could be integrated with an Acquisition Unit (AU) capable of sampling and acquiring magnetic field measurements from magnetometers. This is considered the next step toward developing an FPGA-based embedded myokinetic localizer. Expanding the scope of machine-learning models for the estimation of

magnet displacement based on the experimental dataset could reveal candidate models for hardware implementation with greater accuracy.

For the DPR design methodology, assess the usage of compressed partial bitstreams in order to reduce the reconfiguration time overhead. The AMD Xilinx DFX Controller supports bitstream compression but this feature was not investigated.

Regarding the models architecture, implement additional functionality so that model parameters can be changed at run-time. This could be done by adding an AXI-Lite interface to configure model parameters (weights, bias, and others) before initiating data transfers. This can improve the execution time avoiding reconfiguration steps.

References

AMD XILINX. **UG1399 – Vitis High-Level Synthesis User Guide**. July 2022. v2022.1. Cit. on p. 24.

AMD XILINX. **UG479 – 7 Series DSP48E1 Slice**. Mar. 2018. v1.10. Cit. on p. 21.

AMD XILINX. **UG585 – Zynq-7000 Technical Reference Manual**. June 2023. v1.14. Cit. on pp. 30–32, 35.

AMD XILINX. **UG892 – Vivado Design Suite Design Flows Overview**. Apr. 2022. v2022.1. Cit. on pp. 22, 64.

AMD XILINX. **UG909 – Vivado Design Suite Dynamic Function eXchange**. June 2022. v2022.1. Cit. on pp. 28, 63, 68.

AMD XILINX. **UG947 – Vivado Design Suite Tutorial: Dynamic Function eXchange**. May 2022. v2022.1. Cit. on pp. 27, 29, 30.

AMD XILINX. **UG984 – MicroBlaze Processor Reference Guide**. June 2022. v2022.1. Cit. on pp. 34, 35.

AYALA, H.; SAMPAIO, R.; MUÑOZ, D. M.; LLANOS, C.; COELHO, L.; JACOBI, R. Non-linear model predictive control hardware implementation with custom-precision floating point operations. In: 2016 24th Mediterranean Conference on Control and Automation (MED). 2016. P. 135–140. Cit. on p. 60.

AYALA, H. V. H.; MUÑOZ, D. M.; LLANOS, C. H.; COELHO, L. d. S. Efficient hardware implementation of Radial Basis Function Neural Network with customized-precision floating-point operations. **Control Engineering Practice**, Elsevier BV, v. 60, p. 124–132, Mar. 2017. ISSN 0967-0661. DOI: 10.1016/j.conengprac.2016.12.004. Available from: <http://dx.doi.org/10.1016/j.conengprac.2016.12.004>. Cit. on p. 60.

BENGIO, Y.; COURVILLE, A.; VINCENT, P. Representation Learning: A Review and New Perspectives. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Institute of Electrical and Electronics Engineers (IEEE), 2013. DOI: 10.1109/tpami.2013.50. Cit. on p. 43.

BRUCKNER, F.; ABERT, C.; WAUTISCHER, G.; HUBER, C.; VOGLER, C.; HINZE, M.; SUESS, D. Solving Large-Scale Inverse Magnetostatic Problems using the Adjoint Method. **Scientific Reports**, v. 7, 2017. Cit. on pp. 14, 37.

BRUNTON, S. L.; KUTZ, J. N. **Data-Driven Science and Engineering**. Cambridge University Press, 2019. DOI: 10.1017/9781108380690. Cit. on p. 43.

CHURIWALA, S. **Designing with Xilinx® FPGAs: Using Vivado**. Springer International Publishing, 2016. ISBN 9783319424385. Available from: <https://books.google.com.br/books?id=6vFNDQAAQBAJ>. Cit. on pp. 19–21.

CIPRIANI, C. et al. The MyoKinetic prosthetic hand: Implanted magnets restore grasping in humans with upper limb amputation. Preprint available at https://doi.org/10.21203/rs.3.rs-3221346/v1, 2023. Cit. on p. 14.

CLEMENTE, F.; IANNICIELLO, V.; GHERARDINI, M.; CIPRIANI, C. Development of an Embedded Myokinetic Prosthetic Hand Controller. **Sensors**, v. 19, n. 14, 2019. Cit. on pp. 14, 37, 38.

CONG, J.; LIU, B.; NEUENDORFFER, S.; NOGUERA, J.; VISSERS, K.; ZHANG, Z. High-Level Synthesis for FPGAs: From Prototyping to Deployment. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 30, n. 4, p. 473–491, 2011. DOI: 10.1109/TCAD.2011.2110592. Cit. on p. 23.

ESPOSITO, D.; CENTRACCHIO, J.; ANDREOZZI, E.; GARGIULO, G. D.; NAIK, G. R.; BIFULCO, P. Biosignal-Based Human–Machine Interfaces for Assistance and Rehabilitation: A Survey. **Sensors**, v. 21, n. 20, 2021. ISSN 1424-8220. DOI: 10.3390/s21206863. Available from: <https://www.mdpi.com/1424-8220/21/20/6863>. Cit. on p. 13.

FINE LICHT, J. de; BESTA, M.; MEIERHANS, S.; HOEFLER, T. Transformations of High-Level Synthesis Codes for High-Performance Computing. **IEEE Transactions on Parallel and Distributed Systems**, 2021. DOI: 10.1109/TPDS.2020.3039409. Cit. on pp. 23, 55.

GHERARDINI, M.; MANNINI, A.; CIPRIANI, C. Optimal Spatial Sensor Design for Magnetic Tracking in a Myokinetic Control Interface. **Computer Methods and Programs in Biomedicine**, Elsevier, v. 211, p. 106407, 2021. Cit. on pp. 37, 43.

GHERARDINI, M.; MASIERO, F.; IANNICIELLO, V.; CIPRIANI, C. The Myokinetic Interface: implanting permanent magnets to restore the sensory-motor control loop in amputees. **Current Opinion in Biomedical Engineering**, p. 100460, 2023. ISSN 2468-4511. DOI: https://doi.org/10.1016/j.cobme.2023.100460. Cit. on p. 13.

IANNICIELLO, V.; GHERARDINI, M.; CIPRIANI, C. Transcutaneous Magnet Localizer for a Self-Contained Myokinetic Prosthetic Hand. **IEEE Transactions on Biomedical Engineering**, v. 71, n. 3, p. 1068–1075, 2024. DOI: 10.1109/TBME.2023.3325910. Cit. on pp. 14, 36–38, 72, 73.

LLOYD, S. Least squares quantization in PCM. **IEEE Transactions on Information Theory**, Institute of Electrical and Electronics Engineers (IEEE), v. 28, n. 2, p. 129–137, Mar. 1982. ISSN 0018-9448. Cit. on p. 49.

MASIERO, F.; SINIBALDI, E.; CLEMENTE, F.; CIPRIANI, C. Effects of Sensor Resolution and Localization Rate on the Performance of a Myokinetic Control Interface. **IEEE Sensors Journal**, Institute of Electrical and Electronics Engineers (IEEE), v. 21, n. 20, 2021. DOI: 10.1109/jsen.2021.3109870. Cit. on pp. 37, 38.

MENDES, D. A.; REVES, G.; PASTRANA, M. A.; DOMINGUES, P. H.; AYALA, H. V. H.; KUBRUSLY, A. C.; MUÑOZ, D. M.; LLANOS, C. H. A Comparative Analysis of HDL and HLS for Accelerating Machine Learning based Strain Estimation with Ultrasonic Guided Waves. In: 2023 XIII Brazilian Symposium on Computing Systems Engineering (SBESC). 2023. P. 1–6. DOI: 10.1109/SBESC60926.2023.10324053. Cit. on p. 54.

MENDEZ, S. P. **RUN-TIME RECONFIGURATION FOR EFFICIENT TRACKING OF IMPLANTED MAGNETS WITH A MYOKINETIC CONTROL INTERFACE APPLIED TO ROBOTIC HANDS**. Apr. 2021. PhD thesis – Universidade de Brasília. Available from: <https://repositorio.unb.br/handle/10482/41876>. Cit. on pp. 14, 15, 38, 42, 43, 48, 50, 59, 62, 69, 71–73.

MENDEZ, S. P.; GHERARDINI, M.; PAULA SANTOS, G. V. de; MUNOZ, D. M.; AYALA, H. V. H.; CIPRIANI, C. Data-Driven Real-Time Magnetic Tracking Applied to Myokinetic Interfaces. **IEEE Transactions on Biomedical Circuits and Systems**, Institute of Electrical and Electronics Engineers (IEEE), v. 16, n. 2, p. 266–274, Apr. 2022. DOI: 10.1109/tbcas.2022.3161133. Cit. on pp. 14, 15, 38, 42, 48, 59–61.

MORADI, A.; RAFIEI, H.; DALIRI, M.; AKBARZADEH-T., M.-R.; AKBARZADEH, A.; NADDAF-SH., A.-M.; NADDAF-SH., S. Clinical implementation of a bionic hand controlled with kineticomyographic signals. **Scientific Reports**, v. 12, n. 1, p. 14805, 2022. ISSN 2045-2322. DOI: 10.1038/s41598-022-19128-1. Cit. on pp. 13, 38.

MUÑOZ, D. M.; SANCHEZ, D. F.; LLANOS, C. H.; AYALA-RINCÓN, M. FPGA based floating-point library for CORDIC algorithms. In: 2010 VI Southern Programmable Logic Conference (SPL). Porto de Galinhas, Brazil: IEEE, 2010a. DOI: 10.1109/spl.2010.5483002. Cit. on pp. 59, 60.

MUÑOZ, D. M.; SANCHEZ, D. F.; LLANOS, C. H.; AYALA-RINCÓN, M. Tradeoff of FPGA Design of a Floating-point Library for Arithmetic Operators. **Journal of Integrated Circuits and Systems**, Journal of Integrated Circuits and Systems, 2010b. Cit. on pp. 59, 60.

NELLES, O. **Local linear neuro-fuzzy models: Fundamentals**. Springer International Publishing, 2020. P. 393–445. Cit. on pp. 43–45.

PERTUZ, S. A.; LLANOS, C. H.; MUNOZ, D. M. Development of a Robotic Hand Using Bioinspired Optimization for Mechanical and Control Design: UnB-Hand. **IEEE**

Access, Institute of Electrical and Electronics Engineers (IEEE), v. 9, p. 61010–61023, 2021. DOI: 10.1109/access.2021.3073010. Cit. on p. 73.

SIMPSON, P. A. **FPGA Design: Best Practices for Team-based Reuse**. 2. ed.: Springer International Publishing, 2015. ISBN 978-3-319-17923-0,978-3-319-17924-7. Cit. on p. 23.

TARANTINO, S.; CLEMENTE, F.; BARONE, D.; CONTROZZI, M.; CIPRIANI, C. The myokinetic control interface: tracking implanted magnets as a means for prosthetic control. **Scientific Reports**, Nature Publishing Group, v. 7, n. 1, 2017. Cit. on pp. 13, 14, 36, 38–40.

TAYLOR, C. R.; SRINIVASAN, S. S.; YEON, S. H.; O'DONNELL, M. K.; ROBERTS, T. J.; HERR, H. M. Magnetomicrometry. **Science Robotics**, v. 6, n. 57, eabg0656, 2021. DOI: 10.1126/scirobotics.abg0656. Cit. on p. 38.

ZHOU, J.; CHEN, X.; CHANG, U.; LU, J.-T.; LEUNG, C. C. Y.; CHEN, Y.; HU, Y.; WANG, Z. A Soft-Robotic Approach to Anthropomorphic Robotic Hand Dexterity. **IEEE Access**, v. 7, p. 101483–101495, 2019. DOI: 10.1109/ACCESS.2019.2929690. Cit. on p. 14.

Appendix

APPENDIX A – Related Articles

A Comparative Analysis of HDL and HLS for Accelerating Machine Learning based Strain Estimation with Ultrasonic Guided Waves

Davi A. Mendes*, Gabriel Reves*, M. A. Pastrana*, Pedro H. Domingues†,
Helon V. H. Ayala†, Alan C. Kubrusly‡, Daniel M. Muñoz* and Carlos H. Llanos*
* *Department of Mechanical Engineering*
University of Brasília - UnB
Brasília, Brazil
Email: davi.mendes@ieee.org, gabriel.reves@gmail.com, mariopastrana403@gmail.com, damuz@unb.br, llanos@unb.br

† *Department of Mechanical Engineering*
Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, Brazil
Email: phd.engmec@gmail.com, helon@puc-rio.br

‡ *Center for Telecommunications Studies*
Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, Brazil
Email: alan@cpti.cetuc.puc-rio.br

***Abstract*—In the field of nondestructive testing and structural health monitoring, ultrasonic waves are widely utilized to identify defects and characterize materials. Recently, data-driven machine learning models have been proposed for strain estimation using shallow-models and Principal Component Analysis (PCA). However, little research effort has been guided towards the development of real-time strain estimation hardware accelerators. This study presents a novel comparative analysis of hardware implementations of PCA on a low-cost SoC-FPGA using High-level Synthesis (HLS) and HDL-based architectures. The comparison was conducted in terms of relevant metrics: hardware occupation, latency, and computational efficiency. Additionally, we demonstrate a scalability analysis considering floating-point bit-width representation and the number of operators. The proposed HDL-based architecture was able to achieve similar performance in comparison with the HLS-based implementation. The advantages of the proposed hardware accelerators are shown by their real-time inference capabilities, low power consumption, and reduced hardware utilization associated with low latency and elevated computational efficiency.**

***Index Terms*—FPGA; HLS; Principal Component Analysis; Machine Learning; Ultrasonic Guided Waves; Strain Estimation.**

I. Introduction

Ultrasonic waves are widely used in nondestructive tests and structural health monitoring fields in order to identify defects and characterize materials [1]. Due to the acoustoelastic effect [2], ultrasonic waves can also be used to measure mechanical stress by observing subtle time-of-flight variations within the received signals [3], [4]. Thin structures behave as ultrasonic waveguides allowing propagation over long distances and providing information on the condition of a larger area [5]. Ultrasonic guided waves propagate under distinct, generally dispersive, wave modes [6], which can be simultaneously generated due to a given excitation signal [7].

The stress dependence of ultrasonic guided waves is, however, complex not only due to wave mixing within the received signal [8], but due to the different stress sensitivity of each guided wave mode [9], [10].

Recently, data-driven machine learning techniques have been applied in the evaluation and monitoring of mechanical stress [11] and plastic strain of metallic plates [12].

In [13], an online stress monitoring system for metallic plates was proposed. The method consists of an unidimensional Convolutional Neural Network (1D-CNN) trained with the Lamb wave responses sensed in the plate in different static loadings, which proved effective in the estimation of the stress under static and dynamic loadings.

Similar approach was used in [14] to assess the stress level of pretensioned rods for civil construction, considering three types of grout materials and noise levels. The CNN presented an accuracy of 100% in identifying the prestressing level of the rods when in 90dB-noise and maintained an acceptable accuracy in other noise levels. Contrasting, in [4], a support vector regressor model was trained with features extracted from ultrasonic guided waves signals using Wavelet Packet (WPD) and Singular Value Decomposition (SVD).

Developing real-time monitoring of mechanical stress requires a detailed study into numerical precision and computational performance while considering power and computational constraints present in embedded solutions. The authors of [11] explored different machine-learning models to estimate the mechanical stress of an aluminum plate to which tensile stress was applied demonstrating that the models based on Principal Component Analysis (PCA) exhibit superior accuracy and require a smaller model size than previous models.

In general, dedicated hardware accelerators are extremely helpful, specially in low-cost embedded systems with con-

strained performance. Since PCA is computationally expensive including a large number of arithmetic operations on vast datasets, significant research efforts have been invested into designing efficient and optimized PCA architectures using Field Programmable Gate Arrays (FPGAs) to accelerate its execution for dimensionality reduction and feature extraction.

In spite of the fact that HDL-based RTL implementation using VHDL or Verilog is still predominant in FPGA development, High-Level Synthesis (HLS) leverages programming languages such as C/C++ to provide hardware layouts from imperative code and annotated directives. HLS tools can provide faster development time and allows for guided design exploration with detailed resource and timing reports. Furthermore, the quality of the implementation depends on the intricacy of the compiler and the designers ability to generate efficient hardware. In this regard, the work of [15] presents several transformations and guidelines for developing performance oriented hardware layouts from HLS codes.

Nowadays, HLS-based designs have been proposed as efficient PCA hardware accelerators, for instance, in [16], a fall detection system was presented employing PCA as a dimensionality reduction method for a decision tree classifier implemented in Programmable Logic (PL) on a Zynq SoC. In [17], the authors demonstrate an HLS-based PCA implementation for spectral image processing with 12 channels on a Zynq SoC. The eigenproblem is solved in the Processing System (PS) while the remaining steps are mapped and executed in the PL. In addition, the work of [18] presents a HLS design space exploration for PCA projection applied to spectral image processing. In [19], [20], a guided design space exploration is used to evaluate a flexible PCA hardware implementation for floating-point and fixed-point arithmetic using an SVD Core. The authors of [21] proposed an efficient HLS implementation for a linear discriminant analysis classifier using PCA theory to reduce computational complexity by: (i) providing low-dimensional features and (ii) avoiding matrix inversion operations on an AMD Xilinx Virtex UltraScale series FPGA.

Alternatively, the work of [22] presents an HDL-based implementation of the full PCA algorithm split across processing elements mapped in a powerful AMD Xilinx Virtex-7 FPGA. It features a MicroBlaze soft-core processsor and the HDL architecture is responsible for carrying out dimensionality reduction on hyperspectral images up to 256 channels with real-time capabilities. A parametric HDL-based floating-point algorithmic implementation for the PCA was proposed in [23] for embedded machine learning applications. It considers a tunable representation and evaluates the numerical precision, throughput and power dissipation for different bit-width representations.

This study aims at providing hardware architectures for accelerating embedded machine learning algorithmic implementations in which the PCA is used for dimensionality reduction with a given set of recorded principal components. In this proposal, we adopted experimental data collected in [8] and the data modeling methodology presented in [11] to provide an accurate and efficient hardware accelerator in a strain estimation task with real-time capabilities on a low-cost Zynq-7000 SoC.

Regarding the RTL design, our exploration was divided into (i) a high-level synthesis approach using Vitis HLS and (ii) a parametric HDL IP-core. This allowed us to analyze the shortcomings of each approach in detail and point out how these problems affect our machine-learning accelerator.

Therefore, our study provides a novel comparative analysis that takes into account the design space exploration and the code transformations described in [15] for the HLS-based architecture, providing an optimized kernel. Additionally, it also features a conceptual design and implementation for a parametric HDL IP-Core with streaming interfaces used in cooperating ARM-FPGA computation.

Moreover, for each approach, we provide relevant results for the synthesized design, interfacing schemes, and task-load distribution, as well as implementation results for hardware occupation, computational latency and efficiency. As the HDL-based architecture uses tunable floating-point operators, a tradeoff analysis for hardware consumption and dissipated power was also performed for different bit-width representations.

II. Methods

A. Strain Estimation Dataset

a) Experimental Dataset: In [8], an aluminum plate with $800\times100\times3$mm dimensions had its farthest ends attached to a bridge structure, where one extremity was fixed while the other moved under the action of an endless screw. A transmitting piezoelectric transducer was used to excite the plate with a 2.5 MHz bandwidth sinc-like broad-band pulse, inducing multiple Lamb modes to be propagated in the plate, which are recorded by a receiving piezoelectric transducer, 700 mm away from the transmitter. Tensile loads were applied to the plate by turning the endless screw, promoting deformations between 0 to 150 μm/m. These deformations were recorded by a resistive strain gauge in the center of the plate. Finally, the plate excitation process was performed 499 times for different strain levels in the aforementioned range. The acquired signals were sampled at 10 MHz, with 8,192 samples, comprising the observable interval from 130.0 μs to 949.1 μs.

b) Principal Component Analysis: PCA is widely used to reduce the data dimensionality of machine learning models. It is used to devise a truncated linear transformation (Eq. 1) in which K columns from the $\mathbf{V}$ matrix (i.e., the principal components, PCs) form an orthogonal basis for the K features providing a low-dimensional approximation ($\mathbf{T_K}$) to the high-dimensional $\mathbf{X}$ data in terms of relevant patterns. This transformation is associated with another matrix factorization given by the SVD of $\mathbf{X} \in \mathbb{C}^{n\times m}$ (Eq. 2).

$$\mathbf{T_K} = \mathbf{X}\mathbf{V_K} \tag{1}$$

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H, \tag{2}$$

where $\mathbf{U} \in \mathbb{C}^{n \times n}$ and $\mathbf{V} \in \mathbb{C}^{m \times m}$ are *unitary* matrices and $\mathbf{V}$ holds an orthonormal-basis of corresponding eigenvectors [24]. Here ^{H} denotes the complex conjugate transpose.

The PCA has two main stages, the *training* and the *projection stage*. During the *training stage*, principal components are derived using SVD. This stage is computationally complex and time-consuming as it involves computing a matrix factorization [25]. The *projection stage* requires the PCs and the input values to perform the dimensionality reduction (see Eq. 1).

c) Data Modeling: Guided by the data modeling methodology and model construction procedure presented in [11] and the experimental data from [8] we demonstrate a novel SoC-FPGA hardware implementation for a shallow machine learning model with dimensionality reduction using PCA. In the inference stage, the hardware accelerator receives the input signal with 8192 samples which is reduced to a set of 9 features after dimensionality reduction. Then, linear regression is applied to determine the estimated strain level for the excitation process.

B. Proposed Approach

1) HLS – High Level Synthesis: The proposed HLS kernel implements three processing pipelines: (a) dimensionality reduction using PCA, (b) data scaling and (c) linear regression. In this kernel, three input ports and a single output port were mapped into separate AXI4 Memory-Mapped interfaces. This interface protocol provides independent read and write channels, which support burst-based access and provide a queue for outstanding transactions. The kernel implementation considers a 32-bit data path used for single-precision floating-point operations and a sequential execution mode controlled by an AXI4-Lite interface.

The first processing module computes the dimensionality reduction applied to the normalized (zero-mean) input data. At this point, the required input data, mean values and the principal components are read from AXI4 Memory-Mapped Interfaces. This algorithmic implementation consists in a loop re-ordered matrix multiplication which transposes the iteration space in order to eliminate loop-carried dependencies and enable pipelining [15]. The subsequent processing module transforms the reduced PCA features by scaling each one to unitary range. Finally, the scaled features are used as inputs for the Linear Regression module.

As for the Vitis HLS development flow, the design was validated through testbench simulation based on ground-truth results for correctness. After a C/RTL Co-Simulation, the RTL design was exported from Vitis HLS as a Vivado IP. Moreover, the packaged IP-Core was integrated in Vivado to the Zynq PS for implementation onto the selected target (*xc7z020clg400-1*).

2) HDL – Hardware Description Language: The proposed HDL IP-Core implements a parametric tree-reduction multiply-add architecture (i.e. the *vector multiplication module*) that calculates a scalar product between two vectors. This architecture is synchronized to a Serial-Input Parallel-Output (SIPO) register which stores the input data and the principal components (PCs).

On the PL side, the architecture is connected to an AXI-DMA bridge using a single input and a single output AXI-Stream interface with a 32-bit data path. Figure 1 shows the proposed high-level interfacing scheme. At the PS side, the Zynq processor interfaces with the DMA using High-Performance AXI ports for data transfers and a simple AXI-Lite interface for flow control. In short, the AXI-DMA bridge provides a seamless connection between the SIPO register used in the HDL-IP Core and an addressable memory bank stored into the PS.

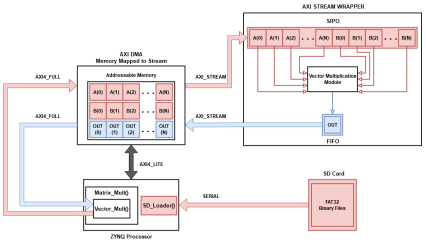


Fig. 1. Interfacing scheme for the proposed HDL IP-Core.

Regarding the dataflow, the SIPO register simultaneously loads the multiplication inputs into the first operand level while the vector multiplication is being performed, thus enabling the loading of new input data to the SIPO register in a pipelined fashion. Figure 2 shows the fine-grained details of the module implemented in the PL. Using a SIPO input register and a First-In-First-Out (FIFO) output register eases the development of additional logic required for the AXI-stream handshake protocol.

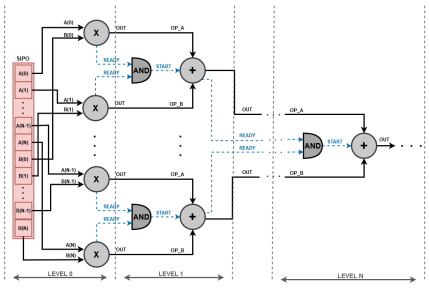


Fig. 2. The *Vector Multiplication Module* (depicted in the *AXI Stream Wrapper* block in Fig. 1).

The floating-point operators used for the multipliers and adders are based on pre-characterized FPLib IP-Cores [26], [27]. In addition, FPLib supports parametric floating-point bit-width representation. This could be exploited in a trade-off analysis between hardware consumption, computational performance, and numerical precision in order to provide the optimal physical implementation [27].

TABLE I
HLS SYNTHESIS PERFORMANCE & RESOURCE ESTIMATES

Module & Loops	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Initiation Interval	Pipelined	DSP	FF	LUT
HLS Design	73846	$7.385 \cdot 10^5$	-	73847	-	-	✖	7 (3%)	6136 (5%)	11809 (22%)
• PCA Iterative Module	73744	$7.374 \cdot 10^5$	-	73744	-	-		2 (≈0%)	1485 (1%)	1047 (1%)
○ PCA Iterative Loop	73742	$7.374 \cdot 10^5$	24	9	8192	9	✓	-	-	-
• PCA Scaler Module	21	210	-	21	-	-		-	319 (≈0%)	102 (≈0%)
○ PCA Scaler Loop	19	190	12	1	9	1	✓	-	-	-
• Lin. Regression Module	52	520	-	52	-	-		-	511 (≈0%)	359 (≈0%)
○ Linear Regression Loop	50	500	27	20	2	20	✓	-	-	-

It is possible to assign the amount of parallel multiply/add operators in the tree-reduction architecture depending on the application requirements. In our proposal, we considered two main variations for implementation using 5 and 100 *cores*, structured with 6 and 7 levels of depth, respectively.

The 5 cores size parameter was selected for implementation because it matches the latency for filling up the input buffer and for processing the actual vector multiplication, while the 100 cores was chosen because it was the highest number of cores viable for implementation on the FPGA using single-precision floating-point operators. Each adder stage is connected to the previous multiplier stage through an AND gate connected to the multiplier `ready` signal and adder `start` signal. This is done because the floating-point operators can only receive new inputs after the output is processed.

The PS-PL task handling was done as follows: The PL component was tasked with multiplying a tiled segment of the matrix multiplication, while the PS component was responsible for streaming data to the PL, handling any interruptions created by the architecture, accumulating partial results into a complete result, and aligning memory sections to prepare them for transmission to the AXI-DMA bridge. This task distribution for matrix multiplication is shown in Eq. 3.

$$C_{ij} = \underbrace{\sum_{1}^{M}}_{\textbf{PS}} \underbrace{\sum_{k}^{cores+k} A_{ik} \cdot B_{kj}}_{\textbf{PL}}, \tag{3}$$

with M being the number of necessary calls of the implemented architecture, calculated as $M = \lceil 8192/cores \rceil$. This means that for the 100 *cores* implementation, 82 calls are needed for a single result, but, if 5 *cores* are used, 1640 calls are needed. Besides, the dataset and the required set of principal components were stored in a SD Card external memory, and were loaded as the processing occurs.

III. RESULTS

This section discusses the obtained results in the implementation with the HLS and HDL architectures evaluated on the low-cost AMD Xilinx ZYNQ SoC (*xc7z020clg400-1*) used in the Zybo Z7-20 development kit. The clock frequency is 100 MHz.

For the proposed HLS approach, Table I summarizes the estimated latency and resource consumption for the synthesized design. From Table I, it is noticed that the PCA processing pipeline takes up $\approx 99.86\%$ of the total latency. This pipeline was implemented with II $= 9$ (Initiation Interval) for optimal throughput performance with unrolling for nested loops processing each one of the 9 output samples.

From Table II, the synthesized HLS RTL considers (i) an accumulation buffer implemented as a true dual-port RAM used to store partial results processed by the first pipeline and (ii) an intermediate buffer implemented as ROM for the scaled data output from the data scaler pipeline. Furthermore, each processing module makes use of the single-precision floating-point multiplication/addition/subtraction using DSP-based and/or logical fabric implementation for a total consumption of 7 DSPs (also reported on Table I). The PCA Scaler and Linear Regression modules also implement ROM storage elements for constant values such as: data scaler coefficients and regression coefficients.

TABLE II
HLS SYNTHESIS BIND OPERATOR/STORAGE SUMMARY

Hierarchy	Op./Storage	Impl.	Latency
Single-Precision Mul. (3 DSP)	`fmul`	`maxdsp`	4
Single-Precison Add/Sub. (2 DSP)	`fadd/fsub`	`fulldsp`	4
Functional Addition ×3	`add`	`fabric`	0
Accumulation Buffer	`ram_t2p`	`BRAM`	1
Scaled Data Buffer	`rom_np`	`BRAM`	1
PCA Iterative Add. (2 DSP)	`fadd`	`fulldsp`	4
Scaler Coef. Parameters	`rom_1p`	`BRAM`	1
Regression Coef. Parameters	`rom_1p`	`BRAM`	1

Table III reports the target device utilization for each architecture. The HLS-based architecture is tightly contained with a significant consumption of BRAM resources, mainly used in the AXI4 Memory-Mapped ports to store interface data from the read channel.

In opposition, the HDL w. 100 *cores* architecture was designed for parallelism/throughput with a trade-off between performance and area in the hardware implementation as shown by the elevated count of utilized DSPs and LUTs. The architecture with 5 *cores* is able to overcome the HLS design by reducing LUT usage in 2.8% and Slice Registers (FF) by 3.36% with an overhead of 3 DSP units.

Additionally, Fig. 3 exhibits the device implementation view with cell highlight for the proposed RTL components. Indeed,

TABLE III
IMPLEMENTED DESIGN HARDWARE UTILIZATION

Design Information	LUT (53200)	FF (106400)	BRAM (140)	DSP (220)
HDL w. 5 cores	4794 (9.01%)	4817 (4.53%)	3.5 (2.5%)	10 (4.55%)
• HDL RTL	1789 (3.36%)	1291 (1.21%)	0 (0%)	10 (4.55%)
HDL w. 100 cores	44935 (84.46%)	26140 (24.57%)	3.5 (2.5%)	200 (90.91%)
• HDL RTL	41930 (78.81%)	22615 (21.26%)	0 (0%)	200 (90.91%)
HLS Architecture	5106 (9.60%)	7157 (6.73%)	52.5 (37.50%)	7 (3.18%)
• HLS RTL Only	3276 (6.16%)	4865 (4.57%)	52.5 (37.50%)	7 (3.18%)
◦ PCA Pipeline	847 (1.59%)	1423 (1.34%)	-	2 (0.91%)
◦ Scaler Pipeline	51 (0.10%)	191 (0.18%)	-	-
◦ Regression P.	357 (0.67%)	493 (0.46%)	-	-

HDL-based solution with 100 *cores* takes up almost all of the available fabric logic as it is expected from Table III data.

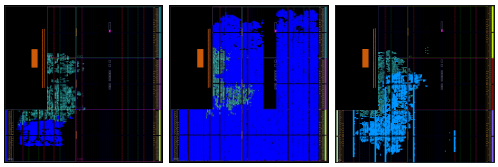


Fig. 3. Device implementation view. Left: HDL-based with 5 cores; Center: HDL-based with 100 cores; Right: HLS-based.

The computational latency, as shown in Table IV, was measured using an AXI-Timer while performing in-circuit verification. The HLS approach fared the worst and the HDL-based architecture with 100 *cores* performed the best, being almost 15% faster for a single execution in the model inference task. The architecture with 5 *cores* is quite similar to the HLS solution. The ARM latency was outperformed by the hardware accelerators with a speedup of: ≈ 14 [HDL: 100 *cores*], ≈ 11.7 [HDL: 5 *cores*] and ≈ 11.8 [HLS] respectively.

In addition, Table IV also presents the computational latency for the test set inference, which consists in sequentially processing all the 250 recorded runs.

TABLE IV
MEASURED COMPUTATIONAL LATENCY

Computational Latency	HDL Architecture (5 *cores*)	HDL Architecture (100 *cores*)	HLS Architecture	ARM
Single Execution [ms]	1.688	1.414	1.6710	19.792
Test Set Inference [ms]	422.0	353.5	416.0	4948.0

In order to evaluate computational efficiency, Table V presents the power report with respect to the architecture PS and PL dynamic/static power consumption. From the data, it is observed that the HDL architecture with 100 *cores* is more power intensive than the HLS counterpart by almost 37.5%, while the HDL architecture with 5 *cores* is only 3% more power intensive.

To put the data into perspective, evaluating each proposed architecture in terms of inference frequency (f_L) per watt, the

TABLE V
POWER REPORT SUMMARY

On-Chip	HDL 5 cores	HDL 100 cores	HLS Architecture
Dynamic (W)	1.436	1.948	1.387
• PS Dynamic (W)	1.404	1.404	1.258
• PL Dynamic (W)	0.032	0.544	0.129
◦ Proposed Design (W)	0.006	0.521	0.113
Device Static (W)	0.138	0.153	0.142
Total On-Chip Power (W)	1.575	2.101	1.529

5 *cores* HDL-based design exhibits ≈ 376.1 f_L/W, the 100 *cores* HDL design exhibits ≈ 336.6 f_L/W and the HLS-based design exhibits ≈ 391.4 f_L/W. Although the HDL architecture with 100 *cores* is slightly faster it is not as efficient as the HLS-based which provides a 16% improvement over its counterpart. The HDL architecture with 5 *cores* achieves similar latency and total on-chip power when compared to the HLS-based solution which maintains a marginal improvement of $\approx 4\%$ in computational efficiency over the 5-*core* architecture.

To assess the scalability of the HDL-based implementation, we also considered three distinct bit-width floating point representations: 16 bits with $e = 5$ exponent bits and $m = 10$ mantissa bits, 27 bits ($e = 8, m = 18$) and single-precision with 32 bits ($e = 8, m = 23$). Furthermore, four different *core* sizes, namely 5, 10, 50, and 100, were evaluated. Table VI provides a summary of the resource and power consumption for each combination.

TABLE VI
SCALABILITY ANALYSIS FOR THE BIT-WIDTH AND NUMBER OF CORES.

HDL Design Parameters	LUT %	DSP %	FF %	On-chip power (W)
5 cores, 16 bits	979 (1.84%)	5 (2.27%)	479 (0.45%)	0.01
10 cores, 16 bits	1944 (3.65%)	10 (4.55%)	890 (0.84%)	0.017
50 cores, 16 bits	9868 (18.55%)	50 (22.73%)	4189 (3.94%)	0.066
100 cores, 16 bits	19794 (37.21%)	100 (45.45%)	8290 (7.79%)	0.123
5 cores, 27 bits	2021 (3.80%)	5 (2.27%)	772 (0.73%)	0.018
10 cores, 27 bits	4040 (7.59%)	10 (4.55%)	1443 (1.36%)	0.033
50 cores, 27 bits	20138 (37.85%)	50 (22.73%)	6833 (6.42%)	0.141
100 cores, 27 bits	40269 (75.69%)	100 (45.45%)	13534 (12.72%)	0.262
5 cores, 32 bits	2117 (3.98%)	10 (4.55%)	912 (0.86%)	0.025
10 cores, 32 bits	4217 (7.93%)	20 (9.09%)	1708 (1.61%)	0.045
50 cores, 32 bits	21043 (39.55%)	100 (45.45%)	8103 (7.62%)	0.193
100 cores, 32 bits	42040 (79.02%)	200 (90.91%)	16054 (15.09%)	0.365

Table VI reveals that the hardware resource consumption, including LUTs, FF, DSP and on-chip power escalates significantly as the bit-width representation increases. For instance, the 16-bit implementation generally reduces by half the required hardware resources in comparison the single-precision implementation. Moreover, deploying an architecture with 100 *cores* using a higher bit-width (27 or 32 bits) representation takes up a considerable portion of the available fabric logic, which in turn might restrict the implementation of additional modules required for a specific application.

IV. CONCLUSION

This study conducted a comparative analysis between the High-Level Synthesis (HLS) and hand-written HDL focus-

ing on the implementation of Principal Component Analysis (PCA) for dimensionality reduction applied to mechanical strain estimation using ultrasonic guided waves.

The proposed HDL architecture utilizes a SIPO register and a multiply-add tree, which uses parametric floating-point arithmetic representation. The SIPO register is written through the use of an AXI-Stream protocol with Direct Memory Access. In order to investigate the impact of different architecture parameters such as bit-width representation, number of *cores*, and SIPO size, a custom HDL code generator was developed, which can be utilized to implement various hardware solutions. The proposed cooperating computation using PS-PL for the HDL-based solutions also proved to be efficient and was able to provide similar computational latency in comparison to the HLS-based solution with complete hardware implementation.

The analysis of resources and power consumption revealed that the proposed HDL architecture with 5 *cores* achieves better computational efficiency compared to the HDL design with 100 *cores*. Even though the HLS-based design consumes more hardware resources, it outperforms the HDL design with 5 *cores* by approximately 4% in terms of computational efficiency.

The HLS-based solution also proved to have a faster development pace and was also able to provide insightful reports during the initial design space exploration. In addition, the efficient implementation of memory-mapped interfaces provided by the HLS tool highlights its capabilities for the proposed application.

Finally, a scalability analysis was performed in terms of bit-width and number of parallel *cores*, demonstrating that HDL-based solution using half-precision (16 bits) floating-point representation is able to reduce the hardware occupation by half of the resources required the 27-bit-width solution while maintaining the same latency and throughput.

In future works, the implementation of a more diverse set of hardware machine-learning techniques intend to be integrated into the proposed PCA architecture enabling the real-time estimation of mechanical strain using ultrasound waveguides with low-cost SoC-FPGA.

REFERENCES

[1] Z. Yang, H. Yang, T. Tian, D. Deng, M. Hu, J. Ma, D. Gao, J. Zhang, S. Ma, L. Yang, H. Xu, and Z. Wu, "A review in guided-ultrasonic-wave-based structural health monitoring: From fundamental theory to machine learning techniques," *Ultrasonics*, 2023.

[2] Y. Pao, W. Sachse, and H. Fukuoka, "Acoustoelasticity and ultrasonic measurements of residual stresses," *Phys. Acoust.*, 1984.

[3] V. V. Mishakin, S. Dixon, and M. D. G. Potter, "The use of wide band ultrasonic signals to estimate the stress condition of materials," *Journal of Physics D: Applied Physics*, 2006.

[4] Q. Ji, L. Jian-Bin, L. Fan-Rui, Z. Jian-Ting, and W. Xu, "Stress evaluation in seven-wire strands based on singular value feature of ultrasonic guided waves," *Structural Health Monitoring*, 2021.

[5] E. H. Ling and R. H. A. Rahim, "A review on ultrasonic guided wave technology," *Australian Journal of Mechanical Engineering*, 2020.

[6] J. L. Rose, *Ultrasonic Guided Waves in Solid Media*. Cambridge University Press, 2014.

[7] H. Nishino, K. Yoshida, H. Cho, and M. Takemoto, "Propagation phenomena of wideband guided waves in a bended pipe," *Ultrasonics*, 2006, proceedings of Ultrasonics International (UI'05) and World Congress on Ultrasonics (WCU).

[8] A. C. Kubrusly, N. Pérez, T. F. Oliveira, J. C. Adamowski, A. M. B. Braga, and J. P. Von de Weid, "Mechanical strain sensing by broadband time reversal in plates," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 2016.

[9] N. Gandhi, J. E. Michaels, and S. J. Lee, "Acoustoelastic lamb wave propagation in biaxially stressed plates," *The Journal of the Acoustical Society of America*, 2012.

[10] A. C. Kubrusly, A. M. B. Braga, and J. P. von der Weid, "Derivation of acoustoelastic lamb wave dispersion curves in anisotropic plates at the initial and natural frames of reference," *The Journal of the Acoustical Society of America*, 2016.

[11] C. D. V. Holguin, H. V. H. Ayala, and A. C. Kubrusly, "Improved stress estimation with machine learning and ultrasonic guided waves," *Experimental Mechanics*, 2021.

[12] R. B. Vieira and J. Lambros, "Machine learning neural-network predictions for grain-boundary strain accumulation in a polycrystalline metal," *Experimental Mechanics*, 2021.

[13] H. Lim and H. Sohn, "Online stress monitoring technique based on lamb-wave measurements and a convolutional neural network under static and dynamic loadings," *Experimental Mechanics*, 2020.

[14] Z. Zhang, F. Tang, Q. Cao, H. Pan, X. Wang, and Z. Lin, "Deep learning-enriched stress level identification of pretensioned rods via guided wave approaches," *Buildings*, 2022.

[15] J. de Fine Licht, M. Besta, S. Meierhans, and T. Hoefler, "Transformations of high-level synthesis codes for high-performance computing," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[16] A. A. S. Ali, M. Siupik, A. Amira, F. Bensaali, and P. C. de-la Higuera, "HLS based hardware acceleration on the zynq SoC: A case study for fall detection system," in *2014 IEEE/ACS 11th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 2014.

[17] M. Schellhorn and G. Notni, "Optimization of a principal component analysis implementation on field-programmable gate arrays (fpga) for analysis of spectral images," in *2018 Digital Image Computing: Techniques and Applications (DICTA)*, 2018.

[18] R. Marino, J. M. Lanza-Gutierrez, T. Riesgo, and M. Holgado, "Design space exploration for PCA implementation of embedded learning in FPGAs," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018.

[19] M. A. Mansoori and M. R. Casu, "Efficient FPGA implementation of PCA algorithm for large data using high level synthesis," in *2019 15th Conference on Ph.D Research in Microelectronics and Electronics (PRIME)*. IEEE, 2019.

[20] ——, "HLS-based flexible hardware accelerator for PCA algorithm on a low-cost ZYNQ SoC," in *2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*. IEEE, 2019.

[21] D. Peng and J. Sha, "Efficient HLS implementation of fast linear discriminant analysis classifier," *IEEE Embedded Systems Letters*, 2021.

[22] D. Fernandez, C. Gonzalez, D. Mozos, and S. Lopez, "FPGA implementation of the principal component analysis algorithm for dimensionality reduction of hyperspectral images," *Journal of Real-Time Image Processing*, 2016.

[23] M. Franceschi, A. Nannarelli, and M. Valle, "Tunable floating-point for embedded machine learning algorithms implementation," in *2018 15th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*. IEEE, 2018.

[24] S. L. Brunton and J. N. Kutz, *Data-Driven Science and Engineering*. Cambridge University Press, 2019.

[25] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

[26] D. M. Muñoz, D. F. Sanchez, C. H. Llanos, and M. Ayala-Rincón, "FPGA based floating-point library for CORDIC algorithms," in *2010 VI Southern Programmable Logic Conference (SPL)*. Porto de Galinhas, Brazil: IEEE, 2010.

[27] ——, "Tradeoff of FPGA design of a floating-point library for arithmetic operators," *Journal of Integrated Circuits and Systems*, 2010.