

Design and Implementation of Matrix Algebra Accelerators on FPGA for Signal Processing Applications using High-Level Synthesis

A thesis submitted in the year 2025 to the University of Hyderabad
in partial fulfilment of the award of a **Ph.D.** degree

in

Electronics Science

by

VENKATA SIVA KUMAR K

(Reg.NO: 18PHPE04)



Center for Advanced Studies in Electronics Science and Technology

School of Physics

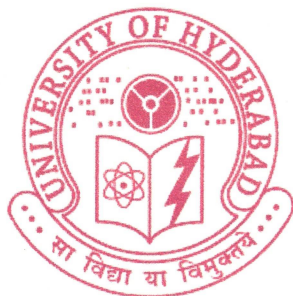
University of Hyderabad

(P.O.) Central University,

Gachibowli, Hyderabad – 500046

Telangana

India



CERTIFICATE

This is to certify that the thesis entitled “**Design and Implementation of Matrix Algebra Accelerators on FPGA for Signal Processing Applications using High-Level Synthesis**” submitted by **Mr. Venkata Siva Kumar K** bearing registration number 18PHPE04 in partial fulfilment of the requirements for the award of **Doctor of Philosophy** in Electronics Science in the Center for Advanced Studies in Electronics Science and Technology (CASEST), School of Physics is a bonafide work carried out by him under my supervision and guidance.

This thesis is free from plagiarism and has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Further, the undertaken work has the following publications before submission of the thesis for adjudication and has produced evidence for the same in the form of the reprint in the relevant area of his research:

1. **Venkata Siva Kumar K**, Vijitha Naikoti, Samrat L.Sabat, et al. “FPGA-Based Hardware Accelerator for Matrix Inversion”. In: SN Computer Science journal 4.2 (2023), p.147 (5),doi:10.1007/s42979-022-01542-x, ISSN:2661-8907.

and

has made presentations in the following conferences:

1. **Venkata Siva Kumar K**, Venkat Reddy Kopparthi, Samrat L.Sabat et al.
“System on chip implementation of floating point matrix inversion using modified Gram-Schmidt based QR decomposition on Pynq FPGA”. In: 2021 IEEE International Symposium on Smart Electronic Systems (iSES). IEEE. 2021, pp. 84–88 (5).
2. **Venkata Siva Kumar K**, Venkat Reddy Kopparthi and Samrat L.Sabat,
“Hardware-Software Co-Design for Whitening using Zero-Phase Component Analysis,” 2022 IEEE International Symposium on Smart Electronic Systems (iSES), Warangal, India, 2022, pp. 36-41, doi: 10.1109/iSES54909.2022.00019.
3. **Venkata Siva Kumar K** and Samrat L.Sabat, “Hardware-Efficient Implementation of Principal Component Analysis Using High-Level Synthesis,” 2024 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), Bangalore, India, 2024, pp. 1-5, doi: 10.1109/CONECCT62155.2024.10677202..

Further, the student has passed the following courses towards the fulfilment of coursework requirements for a PhD.

S.No.	Course Code	Name	Credits	Pass/Fail
1	PE701	SEMICONDUCTOR DEVICES	4	Pass
2	PE702	VLSI Digital Design	4	Pass
3	PE704	RESEARCH METHODOLOGY	4	Pass
4	PE705	ANALOG AND MIXED SIGNAL IC	4	Pass

Samrat Sabat
Prof. Samrat L.Sabat 2-9-2025

Supervisor

Samrat Sabat
2-9-2025
Head, CASEST
HEAD, CASEST
(Centre for Advanced Studies in
Electronics Science & Technology)
School of Physics
University of Hyderabad
Hyderabad-500 046.

[Signature] 3/9/25
Dean, School of Physics
संकाय अध्यक्ष / Dean
भौतिकी संकाय / School of Physics
हैदराबाद विश्वविद्यालय
UNIVERSITY OF HYDERABAD
हैदराबाद / HYDERABAD-500 046. भारत / INDIA.

DECLARATION

I hereby declare that the matter embodied in this thesis titled “**Design and Implementation of Matrix Algebra Accelerators on FPGA for Signal Processing Applications using High-Level Synthesis**” submitted by me under the guidance and supervision of Prof. Samrat L. Sabat, Center for Advanced Studies in Electronics Science and Technology (CASEST), School of Physics, University of Hyderabad is a bonafide research work. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma. I hereby agree that my thesis can be deposited in Shodhganga/INFLIBNET. A report on plagiarism statistics from the University Librarian is enclosed.

Date: 01-09-2025

Place: Hyderabad

K. Venkata Siva Kumar

Mr. Venkata Siva Kumar K

RELEVANCE TO SUSTAINABLE DEVELOPMENT GOALS

Thesis Title: Design and Implementation of Matrix Algebra Accelerators on FPGA for Signal Processing Applications using High-Level Synthesis

Registration No.: 18PHPE04

Supervisor: Prof. Samrat L. Sabat

Centre: Center for Advanced Studies in Electronics Science and Technology (CASEST)

School: School of Physics

This thesis, “Design and Implementation of Matrix Algebra Accelerators on FPGA for Signal Processing Applications using High-Level Synthesis”, is aligned with several United Nations Sustainable Development Goals (SDGs). The relevance of this work to the SDGs is outlined as follows:

SDG 3: Good Health and Well-Being

Matrix algebra enables healthcare data analysis, disease modelling (e.g., COVID-19 tracking), and hospital resource optimization, thereby supporting informed medical

decision-making and improved health outcomes.

SDG 7: Affordable and Clean Energy

Matrix-based optimization techniques contribute to the efficient distribution and management of renewable energy resources such as solar and wind power.

SDG 9: Industry, Innovation, and Infrastructure

Matrices are widely applied in engineering and manufacturing processes, including structural analysis, robotics, and control systems, promoting innovation and resilient infrastructure.

SDG 11: Sustainable Cities and Communities

Matrix methods aid in smart transportation system design, traffic optimization, and public service management, supporting data-driven and sustainable urban development.

SDG 13: Climate Action

Large-scale matrix computations underpin climate modelling, disaster forecasting, and greenhouse gas emission analysis, enabling accurate predictions and effective climate policy evaluation.

DEDICATED

To my parents

K Durga Rao & K Subbulamma

To my parents in law

N Mahammadu & N Naguramma

To my wife

Mubeena

To my siblings

Kanakarao, Kameswaramma, Padma kumari, Bhulakshmi & Farook

(who gave me not only life and also their endless support, encouragement, and love)

ACKNOWLEDGEMENTS

It's a great pleasure to express my thanks to the people who have contributed directly or indirectly to the successful completion of my Ph.D. work, and I would like to convey my heartfelt thanks to each of them.

Firstly, I would like to express my sincere gratitude to my research supervisor **Prof. Samrat L. Sabat** for his constant support, encouragement, and for giving me the opportunity to work on the exciting research topic “Design and Implementation of Matrix Algebra Accelerators on FPGA for Signal Processing Applications using High-Level Synthesis”. He guided me with his valuable suggestions throughout the progress of my research work. The research has been a learning and growing experience for me.

I am grateful to my doctoral committee members **Prof M. Ghanashyam Krishna, and Dr. Bhawna Gomber** for the discussions, suggestions, encouragement, and support throughout my Ph.D. I always feel happy to have such a friendly and comfortable faculty at CASEST.

I take this opportunity to thank **Prof. S V S Nageswara Rao, Prof. K. C. James Raju, Prof G. Rajaram, Dr. Anjali Priya, Dr. Pratap Kollu** for their valuable suggestions during the research work. I thank current and former Deans, School of Physics and Heads, CASEST for their cooperation in providing

facilities and infrastructure in the CASEST, School of Physics.

I would also like to thank the University Grant Commission for providing fellowship, C2S program of MeitY and Institution of Eminence, University of Hyderabad for providing CAD tools and financial support to carry out this work. I extend my sincere gratitude to **Dr. Rangababu** for his valuable suggestions and interest in my work.

I would like to thank **P. Ravibabu**, and other technical and non-teaching staff members of the CASEST for their help and co-operation, during my research tenure. I am thankful to my friends **K. Venkata Reddy, Anil Kali, B bhavana, R. Sai Prasad Goud, M. Rahul, and Thulasiram Varma** for their co-operation and suggestions during my study. I would also like to thank MathWorks, Jupyter for providing the tools.

LIST OF PUBLICATIONS

Journal publications:

1. **Venkata Siva Kumar K**, Vijitha Naikoti, Samrat L.Sabat, et al. “FPGA-Based Hardware Accelerator for Matrix Inversion”. In: SN Computer Science Journal 4.2 (2023), p. 147 (5), doi:10.1007/s42979-022-01542-x.

Conference proceedings:

1. **Venkata Siva Kumar K**, Venkat Reddy Kopparthi, Samrat L.Sabat et al. “System on chip implementation of floating point matrix inversion using modified Gram-Schmidt based QR decomposition on Pynq FPGA”. In: 2021 IEEE International Symposium on Smart Electronic Systems (iSES). IEEE. 2021, pp. 84–88 (5).
2. **Venkata Siva Kumar K**, Venkat Reddy Kopparthi and Samrat L.Sabat, “Hardware-Software Co-Design for Whitening using Zero-Phase Component Analysis,” 2022 IEEE International Symposium on Smart Electronic Systems (iSES), Warangal, India, 2022, pp. 36-41, doi: 10.1109/iSES54909.2022.00019.
3. **Venkata Siva Kumar K** and Samrat L.Sabat, “Hardware-Efficient Implementation of Principal Component Analysis Using High-Level Synthesis,”

2024 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), Bangalore, India, 2024, pp. 1-5, doi: 10.1109/CONECCT62155.2024.10677202.

4. S Srinu, **Venkata Siva Kumar K**, Samrat L.Sabat et al. “System on Chip implementation of dual-band spectrum sensing using energy detection algorithm,” 2022 IEEE 19th India Council International Conference (INDICON), Kochi, India, 2022, pp. 1-6, doi: 10.1109/INDICON56171.2022.10039768.

ABSTRACT

Matrix algebra plays a vital role in numerous scientific and engineering applications, including computer graphics, data analysis, and machine learning. The growing demand for efficient processing of large-scale matrix operations has led to the development of hardware accelerators tailored for matrix computations. This research focuses on the development of hardware accelerators for matrix algebra operations critical to scientific computing, machine learning, and signal processing. Emphasis is placed on accelerating complex operations such as matrix inversion, eigenvalue decomposition, and matrix square root computation, which are essential for algorithms like Principal Component Analysis (PCA) and Zero-phase Component Analysis (ZCA).

Various matrix decomposition techniques were explored and analyzed using MATLAB to identify methods that improve computational efficiency and numerical stability. Among these, QR decomposition using the Modified Gram-Schmidt (MGS) approach was found to be particularly effective for inverting large square matrices. Hardware implementations on FPGA using AXI-Lite and DMA interfaces revealed that MGS-based inversion offers lower latency and better figure of merit compared to classical methods, especially for matrix sizes up to 100×100 .

To address the challenge of inverting rank-deficient matrices, SVD-based inversion was implemented using Lanczos and implicit triQR algorithms. Prior to

this, eigenvalue decomposition was realized in hardware to support SVD, and performance metrics were recorded for different matrix dimensions. Further, the ZCA whitening process was implemented on a Pynq FPGA through hardware–software co-design. Two integration scenarios were examined to compare resource utilization and acceleration performance, with eigenvalue decomposition handled in hardware. Finally, a complete PCA implementation was achieved in hardware, demonstrating reduced resource consumption and competitive execution time relative to existing approaches.

The overall work contributes scalable and efficient hardware designs for core matrix operations, supporting high-performance, real-time applications in embedded and edge computing environments.

CONTENTS

1	Introduction	1
1.1	Different matrix algebra methods and their applications	2
1.2	Motivation	3
1.3	Thesis aim and Objectives	4
1.4	Thesis Contributions	6
1.5	Organisation of the Thesis	7
2	Background and Literature Review	9
2.1	Matrix decomposition methods	9
2.1.1	LU Decomposition	10
2.1.2	Cholesky Decomposition	12
2.1.3	Modified Cholesky Decomposition	13
2.1.4	QR Decomposition	14

2.1.5	Singular Value Decomposition	15
2.2	Data pre-processing	17
2.3	Hardware and Software Co-Design	19
2.4	High-Level Synthesis design space exploration	20
2.4.1	Optimization techniques in HLS	22
2.4.1.1	Loop Optimizations	22
2.4.1.2	Function Pipeline optimization	23
2.4.1.3	Dataflow optimization	23
2.4.2	Data transfer interfaces	24
2.5	Pynq FPGA	26
2.5.1	Pynq-Z1 board setup	28
3	Hardware Acceleration of Eigenvalue Decomposition and Square Root of Matrix	29
3.1	Chapter Overview	29
3.2	Introduction	30
3.2.1	Analytical method	31
3.2.2	Jacobi method	31
3.2.3	Power method	34
3.2.4	Givens Rotation method	36
3.2.5	Householder method	37
3.2.6	QR method	39

3.2.7	Hessenberg method	39
3.3	Eigenvalue decomposition using Lanczos algorithm	42
3.4	Square root of a matrix	46
3.4.1	Schur decomposition method	47
3.4.2	Newton-Raphson method	48
3.4.3	Eigenvalue decomposition with Lanczos algorithm	49
3.5	Hardware Results	51
3.5.1	Eigenvalue Decomposition	51
3.5.2	Square root of the matrix using Eigen Decomposition	54
3.6	Conclusions	55
4	Hardware Accelerator for Matrix Inversion	57
4.1	Chapter Overview	57
4.2	Introduction	58
4.3	Matrix inversion using QR decomposition algorithm	60
4.3.1	QR Decomposition	60
4.3.2	QR Decomposition using modified Gram-Schmidt Algorithm	60
4.3.3	Matrix inversion using MGS based QR decomposition algorithm	61
4.3.4	Simulation results for MGS-based matrix inversion	62
4.3.5	Hardware results	65
4.3.5.1	QR decomposition	66
4.3.5.2	Matrix inversion using modified Gram Schmidt	68

4.4	Matrix inversion using Lanczos and implicit TriQR algorithms	71
4.4.1	Simulation results for SVD based matrix inverse	74
4.4.2	Hardware results	75
4.5	Conclusions	80
5	Hardware implementation of Whitening algorithm	81
5.1	Hardware-Software Co-Design for Whitening using ZCA	82
5.1.1	Introduction	82
5.1.2	Whitening with ZCA algorithm	85
5.1.3	Results and Discussion	86
5.1.4	Conclusions	90
5.2	Hardware implementation of whitening using PCA	92
5.2.1	Introduction	92
5.2.2	Principal Component Analysis algorithm	93
5.2.3	Hardware Implementation	95
5.2.4	Conclusions	98
6	Conclusions and Future Work	99
6.1	Conclusions	99
6.2	Future work	101
	Bibliography	102

LIST OF FIGURES

2.1	Matrix decomposition	9
2.2	Matrix decomposition methods are suited to specific matrix characteristics	16
2.3	Data preprocessing process [47]	17
2.4	Hardware and software co-design framework [59]	19
2.5	HLS Design Implementation flow	21
2.6	Loop Pipelining optimization [68]	22
2.7	Loop unroll optimization [68]	23
2.8	Dataflow optimization [70]	24
2.9	AXI4 interfaces	25
2.10	DMA interface [74]	26
2.11	Pynq framework architecture [77]	27
3.1	Jacobi plane of rotation [88]	32

3.2	Plane of reflection for Householder transformation [96]	37
3.3	SoC interface architecture of Eigenvalue Decomposition	52
3.4	Overlay of eigenvalue decomposition using AXI-Lite interface	53
3.5	SoC interface architecture of Square root of matrix	54
3.6	Overlay of matrix square root using AXI-Lite interface	55
4.1	Computation time(sec) vs Matrix size of Householder, CGS and MGS	62
4.2	MSE vs Matrix size of Householder, CGS and MGS	64
4.3	PSNR vs Matrix size of Householder, CGS and MGS	64
4.4	MSE of matrix inversion using QR decomposition for different matrix sizes	65
4.5	SoC interface architecture of the QR decomposition using AXI-Lite interface	66
4.6	SoC interface architecture of the QR decomposition using DMA interface	66
4.7	Overlay of QR -based MGS algorithm using AXI-Lite interface	67
4.8	Overlay of QR -based MGS algorithm using DMA interface	67
4.9	SoC interface architecture of the Matrix inversion IP	68
4.10	Overlay of matrix inverse using QR -based MGS algorithm	69
4.11	Illustration of loop pipelining optimization technique [122]	70
4.12	Overlay of matrix inverse using linear algebra package	71
4.13	The block-level representation of matrix inverse computation using the Lanczos algorithm	74

4.14	MSE of matrix inversion using SVD for full rank matrices	74
4.15	MSE of matrix inversion using SVD for singular matrices	75
4.16	SoC design of matrix inverse using Lanczos SVD	76
4.17	Overlay of matrix inverse using Lanczos SVD	76
5.1	Scenario 1 for implementation of whitening with ZCA	84
5.2	Scenario 2 for implementation of whitening with ZCA	84
5.3	Pynq SoC architecture for implementation of whitening with ZCA .	86
5.4	Software implementation of whitening with ZCA	87
5.5	Overlay design of Scenario 1 for implementation of whitening with ZCA	87
5.6	Overlay design of Scenario 2 for implementation of whitening with ZCA	88
5.7	Hardware-software co-design implementation of whitening with ZCA .	90
5.8	Block level representation of PCA implementation	95
5.9	Pynq SoC architecture for PCA	95
5.10	Overlay design of PCA	96
5.11	Power utilization of PCA implementation for a matrix size of 16×30	97

LIST OF TABLES

2.1	Key summary of different matrix decomposition techniques	16
2.2	Comparison of different whitening techniques	18
3.1	Summary of eigenvalue decomposition methods	41
3.2	Resource utilization of eigenvalue decomposition for different input feature dimensions	53
3.3	SW and HW execution time of eigenvalue decomposition for different input feature dimensions	53
3.4	Resource utilization of square root of matrix for different input feature dimensions	55
3.5	SW and HW execution time of square root of matrix for different input feature dimensions	55
4.1	MATLAB simulation results for various matrix inversion decomposi- tion methods	63

4.2	Resources and latency comparison of floating point QR decomposition of matrix size 100×100	67
4.3	Performance analysis of MGS based matrix inversion without inner loop pipelining optimization for various matrix sizes	69
4.4	Performance analysis of MGS-based matrix inversion with inner loop pipelining optimization for various matrix sizes	70
4.5	HAF of MGS matrix inverse using without and with inner loop pipelining optimization for different matrix sizes	71
4.6	Performance analysis of LAP QR based matrix inversion with various matrix sizes	72
4.7	Performance analysis of SVD based matrix inversion for different matrix sizes without inner loop pipelining optimization	77
4.8	Performance analysis of SVD based matrix inversion for various matrix sizes with inner loop pipelining optimization	77
4.9	HAF of SVD matrix inverse using without and with inner loop pipelining optimization for different matrix sizes	78
4.10	HET and FoM comparison of matrix inverse using different algorithm techniques	78
4.11	Performance comparison of different matrix inversion algorithms for matrix size 25×25	79
5.1	Comparison of resource utilization for different data feature dimensions	88
5.2	SW and HW execution time of eigenvalue decomposition for different input feature dimensions	89

5.3	SW and HW-SW co-design execution time of whitening with ZCA for different data feature dimensions	90
5.4	Comparison of resource utilization, throughput for various matrix size of eigenvalue decomposition	96
5.5	Performance comparison of PCA implementation for matrix size 16 $\times 30$	97

LIST OF ABBREVIATIONS

SDG	Sustainable Development Goal
CPU	Central Processing Unit
GPUs	Graphics Processing Units
TPUs	Tensor Processing Units
ASICs	Application-Specific Integrated Circuits
LU	Lower-Upper
CGS	Classical Gram-Schmid
MGS	Modified Gram-Schmidt
SVD	Singular Value Decomposition
PYNQ	Python Productivity for Zynq
ZCA	Zero-phase Component Analysis
MIMO	Multiple-Input Multiple-Output
CNN	Convolutional Neural Network
PCA	Principal Component Analysis
HDL	Hardware Description Language
FPGA	Field Programmable Gate Array
HLS	High Level Synthesis
SoC	System-on-Chip
RTL	Register Transfer Level

DSP	Digital Signal Processor
CLB	Configurable Logic Block
LUT	Look Up Table
RAM	Random Access Memory
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
IP	Intellectual Property
EVD	Eigen Value Decomposition
MSE	Mean Square Error
PSNR	Peak Signal Noise Ratio
DMA	Direct Memory Access
DDR	Double Data Rate
AXI	Advanced eXtensible Interface
BRAM	Block Random Access Memory
HAF	Hardware Acceleration Factor
SET	Software Execution Time
HET	Hardware Execution Time
SW CT	SoftWare Computation Time
HW CT	HardWare Computation Time
GR	Givens Rotation
FoM	Figure of Merit
PS	Processing System
PL	Programmable Logic
WO	Without
LAP	Linear Algebra Package
EDP	Energy Delay Product
MAE	Maximum Absolute Error
HW-SW	Hardware-Software
ARM	Advanced RISC Machine
SW	Software
HW	Hardware

CHAPTER 1

INTRODUCTION

Matrix operations such as eigenvalue decomposition, inverse, singular value decomposition on a large-order matrix, require a considerable amount of time. Hence, matrix algebra accelerators play a vital role in hardware system design, and their primary purpose is to accelerate specific tasks [1–3]. Matrix algebra is crucial in various computational fields, such as machine learning, physics, computer graphics, engineering, statistics and economics [4–7]. Hardware accelerators are built to manage the extensive parallelism needed for matrix operations, such as multiplication, addition, inversion, square-root and eigenvalue decomposition [8, 9]. Hardware accelerators platform for matrix algebra, includes Graphics Processing Units (GPUs), Field-Programmable Gate Array (FPGA), Tensor Processing Units (TPUs) and Application-Specific Integrated Circuits (ASICs) [10].

Using specialized hardware to accelerate matrix computations can result in substantial energy savings. This is due to the fact that hardware accelerators can execute numerous operations simultaneously, decreasing the number of data passes required and thus lowering power consumption compared to general-purpose processors. Moreover, these accelerators are engineered to optimize energy efficiency

per operation, making them more suitable for large-scale computations [11]. Many scientific and industrial applications involve solving large-scale matrix problems that surpass the capabilities of conventional CPUs. Hardware accelerators can manage these extensive tasks by distributing the workload across thousands of parallel processing units, making it possible to tackle problems that were previously impractical to solve [12].

Certain matrix operations and algorithms can be greatly optimized when executed on hardware accelerators. For instance, deep learning algorithms, which depend heavily on matrix multiplications, can see substantial speed improvements with TPUs and GPUs. These hardware solutions are tailored to enhance data flow and reduce memory access latency, resulting in much faster execution times. We know the emerging fields such as deep learning, big data analytics, and quantum simulations demand extensive matrix algebra computations. Hardware accelerators are adapting to these new demands by incorporating specialized instructions and architectures, ensuring they can meet the growing performance requirements.

Matrix algebra tasks frequently require significant data movement between memory and processing units. Hardware accelerators usually feature enhanced memory bandwidth and advanced memory hierarchies specifically optimized for matrix operations, resulting in more efficient memory usage and reduced bottlenecks [13]. Overall, hardware accelerators for matrix algebra are motivated by their capacity to deliver high performance, energy efficiency, scalability, real-time processing, and cost savings in handling computationally intensive tasks.

1.1 Different matrix algebra methods and their applications

This section briefly reviews the major matrix algebra methods and their respective applications.

1. Basic Matrix algebra Operations

Basic matrix operations include matrix subtraction, addition, transposition and matrix multiplication. These matrix operations are the basic building

blocks for more advanced methods in matrix computations.

Applications: Used for solving systems of equations and processing scaling and rotation in computer graphics.

2. Matrix inverse

In general matrix inverse is used to solve a set of linear equations.

Applications: The matrix inverse operation is frequently used in statistics, data science, Cryptography, Computer graphics, communication and control systems.

3. Eigenvalues and Eigenvectors

Eigenvalue decomposition is the technique used in linear algebra.

Applications: Eigenvalues and Eigenvectors are used in principal component analysis, quantum mechanics, and stability analysis of differential equations.

1.2 Motivation

Matrix algebra forms the computational backbone of many high-performance applications, including artificial intelligence, machine learning, scientific simulations, and data analytics. With the exponential growth in data-driven applications, there is a rising demand for faster and more efficient matrix computations. Traditional CPU-based software solutions often fail to meet performance, power, and latency requirements, especially for large-scale matrices. This gap has led to the increased adoption of hardware accelerators such as GPUs, TPUs, and FPGAs.

The Key motivations for exploring hardware acceleration in matrix algebra include:

1. Increasing Scale and Complexity of Matrix Operations

Modern applications like deep learning involve large-dimensional matrix operations during training and inference. As model sizes grow, matrix computation becomes a bottleneck. Hardware accelerators offer significant speedups for such operations.

2. Low Latency

Real-time applications in robotics, autonomous systems, and computer vision demand minimal latency. Hardware accelerators enable faster execution of matrix operations, crucial for time-sensitive tasks like image recognition.

3. High-Throughput and Parallelism

Matrix operations such as multiplication, inversion, and eigenvalue decomposition are highly parallelizable. Hardware accelerators leverage this to perform multiple computations concurrently, significantly boosting throughput.

4. Energy efficiency

As matrix size and operation frequency increase, CPU-based computation becomes power-intensive. Custom hardware accelerators can perform these tasks more efficiently, reducing energy consumption while maintaining performance.

1.3 Thesis aim and Objectives

The primary aim of this thesis is to design and develop efficient hardware accelerators for key matrix algebra operations to meet the computational demands of modern data-driven applications. These include artificial intelligence, signal processing, communications, and other domains where real-time performance, low latency, and energy efficiency are critical.

Thesis Objectives:

- To analyze the computational performance and numerical stability of matrix inversion techniques evaluate various matrix decomposition-based inversion methods (e.g., QR, SVD) in a software simulation environment (MATLAB) to identify suitable algorithms for hardware acceleration.
- To implement a hardware accelerator for QR decomposition using the Modified Gram-Schmidt (MGS) method Design a floating-point QR decomposition

core using high-level synthesis (HLS) on Pynq-Z1 FPGA with loop pipelining optimization and AXI bus interfaces.

- To explore and implement efficient eigenvalue decomposition techniques on FPGA Implement eigenvalue decomposition using Lanczos and implicit TriQR algorithms and evaluate their hardware acceleration factor and resource utilization across different matrix sizes.
- To design and compare matrix inversion architectures using MGS and Lanczos-SVD techniques. Build SoC-based hardware implementations of matrix inversion using both full-rank and rank-deficient decomposition methods. Apply pipelining optimizations and compare execution time and accuracy across implementations.
- To develop a hardware-software co-design framework for whitening transformation. Implement whitening using Zero-phase Component Analysis (ZCA) in Hardware-Software co-design mode and Principal Component Analysis (PCA) in full hardware mode, measuring acceleration factors and resource efficiency.
- To evaluate and report hardware performance metrics across all designs. Analyze hardware acceleration factor (HAF), execution time, numerical stability, and resource utilization for all implemented modules and applications, thereby demonstrating the feasibility and scalability of the proposed hardware accelerators.

1.4 Thesis Contributions

Matrix algebra comprises a set of fundamental operations widely used in real-world applications such as signal processing, communication systems, cryptography, and artificial intelligence. Hardware accelerators, being custom-designed systems, offer significant speed and efficiency for such compute-intensive tasks. The primary objective of this thesis is to design and implement hardware accelerators for matrix algebra operations with a focus on improving computational speed and efficiency through algorithmic and architectural optimization.

The key contributions of this thesis are as follows:

- Analyzed and compared the simulation results, computational performance, and numerical stability of various matrix inversion methods based on different decomposition techniques using MATLAB.
- Designed and implemented a System-on-Chip (SoC) QR decomposition architecture using the Modified Gram-Schmidt algorithm with floating-point precision on the Pynq-Z1 FPGA. The implementation utilizes loop pipelining optimizations and AXI-Lite/DMA interfaces.
- Implemented and evaluated hardware acceleration for eigenvalue decomposition and matrix square root computation using Lanczos and implicit TriQR algorithms across various matrix sizes on the Pynq platform.
- Performed a performance analysis of matrix inversion using MGS and Lanczos-based SVD approaches, employing inner loop pipelining for improved throughput.
- Developed a whitening transformation module using Zero Component Analysis through a hardware-software co-design approach, and evaluated the hardware acceleration factor.
- Built a complete SoC implementation of whitening using Principal Component Analysis, employing AXI-Lite data transfer between the ARM processing system and the custom IP core on the Pynq-Z1 FPGA.

1.5 Organisation of the Thesis

The entire thesis is presented in six chapters.

Chapter 1: Introduction: This chapter introduces the significance of hardware accelerators in modern computing and their relevance to sustainable development goals. It presents the motivation behind the research, discusses the challenges in matrix algebra computation, and outlines the objectives, contributions, and structure of the thesis.

Chapter 2: Background and Literature Review: This chapter reviews the concepts of matrix decomposition techniques and evaluates their advantages and limitations based on matrix properties and size. It also provides an overview of the high-level synthesis design flow, optimization strategies, and the hardware platform used in this work.

Chapter 3: Hardware implementation of Eigenvalue decomposition and square root of matrix This chapter focuses on the implementation of eigenvalue decomposition using Lanczos and implicit TriQR algorithms. A hardware IP core is developed on the Pynq-Z1 FPGA using AXI-Lite interfacing. The hardware acceleration factor and resource utilization for various matrix sizes are reported. Additionally, the square root of a matrix is computed using the eigen decomposition approach.

Chapter 4: Hardware Accelerator for Matrix Inversion: This chapter details the implementation of matrix inversion for full-rank matrices using QR decomposition (MGS variant) and for rank-deficient matrices using SVD with Lanczos and TriQR algorithms. These methods are implemented using SoC design on the Pynq-Z1 FPGA. Comparative analysis includes execution time, resource usage, and accuracy, with and without loop optimization.

Chapter 5: Hardware implementation of Whitening algorithm: This chapter presents the implementation of whitening transformations using both ZCA and PCA methods. Whitening is realized in two modes: hardware-software co-design and full hardware implementation. The performance, acceleration factor, and resource utilization are evaluated for varying matrix sizes.

Chapter 6: Conclusions and future works: This chapter summarizes the results obtained throughout the research conducted in the thesis. It consolidates the main findings and conclusions from Chapters 3 to 5, highlighting the contributions made to the hardware accelerator for matrix algebra.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

2.1 Matrix decomposition methods

Matrix decomposition is a foundational concept in applied statistics and linear algebra, with far-reaching implications for scientific and engineering disciplines [14–16]. Matrix decomposition, widely known as matrix factorization, constitutes a foundational operation in linear algebra whereby a given matrix undergoes disassembly into simpler, more manageable constituents as shown in Figure 2.1. These constituents often expose hidden patterns or inherent properties within the original matrix, thereby facilitating its analysis and manipulation [17, 18].

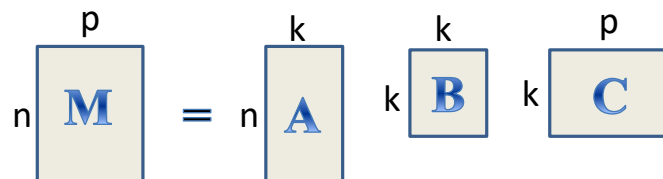

$$\begin{matrix} & p \\ n & \boxed{\mathbf{M}} \end{matrix} = \begin{matrix} & k \\ n & \boxed{\mathbf{A}} \end{matrix} \begin{matrix} & k \\ k & \boxed{\mathbf{B}} \end{matrix} \begin{matrix} & p \\ k & \boxed{\mathbf{C}} \end{matrix}$$

Figure 2.1: Matrix decomposition

Matrix decomposition generally serves two fundamental purposes: improving computational efficiency and simplifying analytical processes. In practical applications, many matrix operations—such as determinant calculation, matrix inversion, least-squares fitting, and solving linear systems—cannot always be computed efficiently through direct methods. By decomposing these complex problems into simpler problems, such as diagonal or triangular systems, computational efficiency is significantly enhanced, and the overall analytical process becomes more tractable [19, 20].

In the literature, there exist numerous methods for matrix decomposition as listed below:

- LU Decomposition [21]
- Cholesky Decomposition [22]
- Modified Cholesky Decomposition [23]
- Singular Value Decomposition [24]
- QR Decomposition [25]
- Eigenvalue Decomposition [26]

2.1.1 LU Decomposition

The lower upper (LU) decomposition method is used to decompose a non-singular matrix B into lower and upper triangular matrices as L and U [27].

$$B_{m \times m} = L_{m \times m} \times U_{m \times m} \tag{2.1}$$

It is useful for solving systems of linear equations; once the input matrix is decomposed into L and U , solving equations becomes faster and easier. Partial pivoting is employed in LU decomposition to enhance numerical stability, while

Algorithm 1 LU decomposition algorithm

Input: Square matrix $B \in \Re^{m \times m}$ **Output:** Lower (L) and upper (U) triangular matrix.

```

1: Initialize  $L$  and  $U$  as a zero matrix with the same dimensions as  $B$ 
2: for each row  $p = 1$  to  $m$  do
3:   for each column  $t = p$  to  $m$  do
4:      $U[p, t] = B[p, t] - \sum_{r=1}^{p-1} L[p, r] \cdot U[r, t]$ 
5:   end for
6:   for each row  $t$  from  $p + 1$  to  $m$  do
7:      $L[t, p] = \frac{B[t, p] - \sum_{r=1}^{p-1} L[t, r] \cdot U[r, p]}{U[p, p]}$ 
8:   end for
9: end for
10: Return Lower ( $L$ ) and upper ( $U$ ) triangular matrix.

```

it's effective for smaller matrices, its hardware implementation complexity increases significantly for larger ones, impacting computational efficiency [28].

For large-scale matrices, parallelized LU factorization or block recursive LU decomposition is used to expedite computation and efficiently accommodate the matrix size. Parallelizing LU factorization improves performance, reduces data reading and execution times, with a complexity of $O(m^3)$ [29–31]. In contrast, block-recursive LU decomposition, as outlined in LU , increases a complexity of $O(m^2)$. Dividing the original matrix into four square blocks, each termed a block without loss of generality, allows for reliable decomposition tailored to the memory capacity of the computing machine. As the number of blocks increases, memory usage and computational costs decrease by enhancing scalability [32].

The major disadvantages of this method are

- It requires additional memory to store L and U .
- It is applicable only if the matrix is non-singular, meaning it is invertible, and every leading principal minor is nonzero.
- Not efficient for sparse matrix.

2.1.2 Cholesky Decomposition

Cholesky decomposition is frequently utilized to decompose positive symmetric matrices when solving least squares problems in signal processing, machine learning, and data regression applications [33, 34]. The Cholesky decomposition uniquely decomposes a symmetric matrix B into either a lower triangular matrix L or an upper triangular matrix U .

$$B_{m \times m} = L_{m \times m} \times L_{m \times m}^T \quad (2.2)$$

This method is applicable only to positive definite and diagonally dominant dense matrices. The complexity of cholesky decomposition is of the order $O(m^3)$ [35]. The

Algorithm 2 Cholesky decomposition algorithm

Input: Lets B be the positive-definite matrix $B \in \Re^{m \times m}$

Output: L be the lower triangular matrix such that $B = LL^T$

```

1: Initialize  $L$  as a zero matrix having size  $m \times m$ 
2: for each row  $r = 1$  to  $m$  do
3:   for each column  $t = 1$  to  $r$  do
4:     if  $r = t$  then
5:        $L[r, r] = \sqrt{B[r, r] - \sum_{k=1}^{r-1} L[r, k]^2}$ 
6:     else
7:        $L[r, t] = \frac{1}{L[t, t]} (B[r, t] - \sum_{k=1}^{t-1} L[r, k] \cdot L[t, k])$ 
8:     end if
9:   end for
10: end for
11: Return Lower triangular matrix  $L$ 

```

major disadvantages of this method are

- Cholesky Decomposition's inherent lack of suitability for parallelization can restrict its scalability on modern computing architectures.
- It can become unstable when computing the inverse of near-singular or rank-deficient matrices.

2.1.3 Modified Cholesky Decomposition

Let B be the symmetric matrix that is decomposed into a L , a lower triangular matrix, and a diagonal matrix D using LDL decomposition such that [36]

$$B_{m \times m} = L_{m \times m} \times D_{m \times m} \times L_{m \times m}^T \quad (2.3)$$

Algorithm 3 Modified Cholesky decomposition

Input: Let's B be the positive-definite matrix $B \in \Re^{m \times m}$

Output: L be the lower triangular matrix such that $B = LDL^T$

```

1: Initialize  $L$  as a zero matrix of size  $m \times m$ 
2: for each row  $q = 1$  to  $m$  do
3:   for each column  $h = 1$  to  $q$  do
4:     if  $q = h$  then
5:        $D[q, q] = B[q, q] - \sum_{y=1}^{q-1} L[q, y]^2 D_{yy}$ 
6:     else
7:        $L[q, h] = \frac{1}{D[h, h]} \left( B[q, h] - \sum_{y=1}^{h-1} L[q, y] \cdot L[h, y] \cdot D[h, y] \right)$ 
8:     end if
9:   end for
10: end for
11: Return  $L, D$ 
```

LDL decomposition is notably advantageous when working with symmetric matrices, as it offers superior computational efficiency compared to the more general LU decomposition [37]. Successful FPGA compilation reports demonstrate that the LDL -based method consumes fewer resources and computes faster compared to QR -based implementations [38]. Using LDL^T factorization, we circumvent the need for square root computations in Cholesky decomposition and devise an efficient algorithm to symbolically compute generalized inverses of polynomial matrices [39]. The major disadvantages of this method include numerical stability, algorithm complexity, restricted parallelism, increased storage demands, and limited applicability.

2.1.4 QR Decomposition

The cornerstone of numerous advanced signal processing algorithms lies in algebraic matrix operations. Abundant examples abound in wireless communication, spanning multiple-input-multiple-output (MIMO) systems, beamforming, multi-user detection, and cancellation [40, 41]. Among the essential operators that facilitate these matrix operations is QR factorization.

The given matrix $B \in \mathfrak{R}^{m \times m}$ undergoes QR decomposition, resulting in an orthogonal matrix (Q) having matrix size $m \times m$ and an upper triangular matrix (R) having matrix size $m \times m$, given as

$$B_{m \times m} = Q_{m \times m} \times R_{m \times m} \quad (2.4)$$

QR decomposition technique is numerically stable compared to Cholesky, LDL and

Algorithm 4 QR decomposition algorithm

Input: $B \in \mathfrak{R}^{m \times m}$

Output: Orthonormal matrix Q , upper triangular matrix R

```

1: for  $t = 1$  to  $m$  do
2:    $q_t = B_t$ 
3:   for  $i = 1$  to  $i < t$  do
4:      $r_{i,t} = q_i^T \times B_t$ 
5:      $q_t = q_t - r_{i,t} \times q_i$ 
6:   end for
7:    $r_{t,t} = ||q_t||$ 
8:    $q_i = q_i / r_{t,t}$ 
9: end for

```

LU decomposition methods. This method is applicable for well-conditioned matrices, including full-rank non-square or non-singular matrices [42]. Various methods exist for computing QR decomposition, each with its own set of advantages and disadvantages due to their unique solution processes. These techniques include the Gram-Schmidt ortho-normalization method, Householder reflections, and Givens rotations. The major disadvantages of this method are: The matrix inversion algo-

rithm based on QR decomposition fails when the input matrix is a singular square or rank-deficient non-square matrix.

2.1.5 Singular Value Decomposition

Singular value decomposition stands as a pivotal factorization method for both real and complex matrices, finding wide-ranging applications across engineering domains such as signal processing, image processing, robotics, and bio-informatics [43, 44]. The matrix B is decomposed into orthogonal matrices U, V and S . The S matrix consists of singular values that are present in a diagonal, and all remaining elements are zero.

$$B_{M \times M} = U_{M \times M} \times S_{M \times M} \times V_{M \times M}^T \quad (2.5)$$

Algorithm 5 Singular Value Decomposition (SVD)

Input: $B \in \mathbb{R}^{M \times M}$

Output: U, S , and V^T Matrices are decomposed such that $B = USV^T$

- 1: Evaluate the eigenvalues and eigenvectors of $A^T A$ and AA^T
 - 2: Form matrix V with eigenvectors of $A^T A$ (columns of V)
 - 3: Form matrix U with eigenvectors of AA^T (columns of U)
 - 4: Sort the eigenvalues in descending order and arrange corresponding eigenvectors in U and V
 - 5: Form the diagonal matrix S with sorted eigenvalues (singular values) on the diagonal
 - 6: **Return** U, S , and V^T
-

Despite its versatility, the computation of SVD poses a formidable challenge, often proving complex and time-consuming. The major disadvantages of this method are high memory requirements and high computation time.

Figure 2.2 provides the possibility of applying a matrix decomposition method to the given input matrix, and Table 2.1 presents the key summary of different matrix decomposition techniques outlined in the literature.

All decomposition methods summarise that LU decomposition helps solve linear

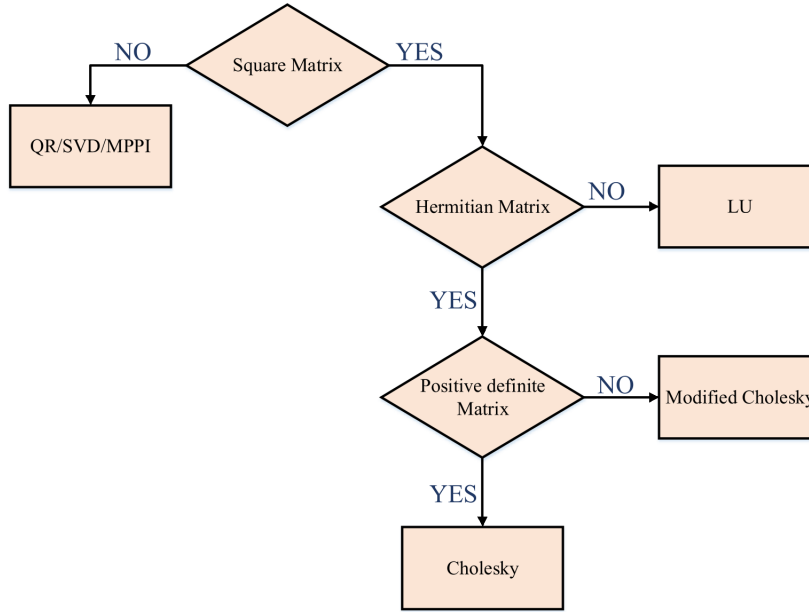


Figure 2.2: Matrix decomposition methods are suited to specific matrix characteristics

Table 2.1: Key summary of different matrix decomposition techniques

Decomposition	Applicable Matrices	Key Property	Applications
LU	Non Singular and Square	Lower and upper triangular matrices	For solving linear systems
Cholesky	Positive-definite	Lower triangular matrix and its transpose	System solving, Optimization
QR	Square or Rectangular	Unitary and upper triangular matrices or Orthogonality	Eigenvalue problems, Least Square
SVD	Rectangular or Square	Unitary matrices Orthogonal	Dimensional reduction, PCA

systems through forward and backwards substitution. Cholesky decomposition is twice as fast when applied to symmetric and positive definite matrices. Modified Cholesky decomposition addresses nearly positive-definite cases by adding a minor correction to the diagonal. QR decomposition can be used for any matrix, providing stability for least-squares problems. Lastly, SVD applies to any square or rectangular matrix, and it is the most robust choice for tasks such as low-rank approximation and dimensionality reduction.

2.2 Data pre-processing

Typically, data from real-world scenarios is incomplete, inconsistent, and contains inaccurate data sets. While training a convolutional neural network (CNN) using these raw data features may lead to inaccurate classification [45]. Data pre-processing must be applied before training the CNN model to obtain an accurate classification [46]. The steps for data preprocessing as shown in Figure 2.3.

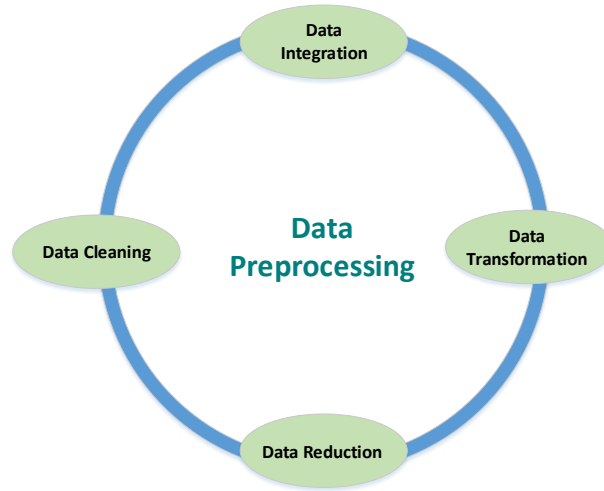


Figure 2.3: Data preprocessing process [47]

Data preprocessing is crucial for several reasons as outlined below:

- **Data Quality Improvement:** In nature, raw data usually contains noisy, inconsistent, and missing values. The preprocessing technique helps clean up and improve its quality[48].
- **Enhanced Model Performance:** We know that well-prepared data can improve the accuracy of the model by applying scaling, normalization, and transformation [49].
- **Data Integration:** Integrate data from different sources to make a single data set to provide a unified view for resolving conflicts between data sources [50].

- **Reduction of Complexity:** Dimensionality reduction methods help to reduce the algorithms' computational requirements by removing redundant features [51].
- **Facilitation of Analysis:** After cleaning the raw data, it is easier to explore and analyze the patterns and relationships of the data, which aids in better data decision making.

The whitening technique is a standard preprocessing technique used in machine learning that transforms random variables into new random variables satisfying orthogonality [52]. The whitening process is a linear transformation that provides an n -dimensional random X vector into a new random vector Z given as [53]:

$$Z = W.X \quad (2.6)$$

The advantage of rotational freedom, many possible whitening procedures are available. Different whitening techniques are used, such as zero-phase component analysis (ZCA), principal component analysis (PCA), Cholesky matrix decomposition, etc. [54].

Table 2.2: Comparison of different whitening techniques

Feature	PCA	ZCA	Cholesky
Purpose	Dimensionality reduction of input data	Data whitening with preserve structure	Whitening input data
Transformation	Projects onto principal axes	Uses eigenvectors for decorrelation	Cholesky factor used for whitening
Output Structure	Reduced in dimensionality	Similar structure to original input data	Uncorrelated features
Computation	Higher due to eigenvalue Complexity	Similar to the PCA but preserve structure	Faster than PCA for large datasets
Using for	Feature extraction and visualization	Image processing and neural networks	For data preprocessing

The above whitening technique is chosen based on the specific requirement of the application, with ZCA often preferred for image data to maintain similarity preservation of input data, PCA is used for dimensionality reduction to maintain

model efficiency in machine learning [55], and Cholesky decomposition whitening is faster without a focus on structure preservation compared to input data.

2.3 Hardware and Software Co-Design

In the 1990s, the growing demand for embedded computing systems led to the emergence of hardware-software co-design [56]. The framework of hardware-software co-design is a design methodology in system building that focuses on the combined growth of hardware and software rather than implementing them separately. The hardware and software co-design framework is shown in Figure 2.4. It shows that a system's performance and functionality depend on the hardware and software working together optimally [57, 58]. Therefore, the rapid growth of hardware-software co-design helps to attain system-level goals by leveraging the synergy between hardware and software through parallel development. The nature of co-design challenges varies depending on the application domain, implementation technology, and design methodology, which is essential for maintaining commercial competitiveness.

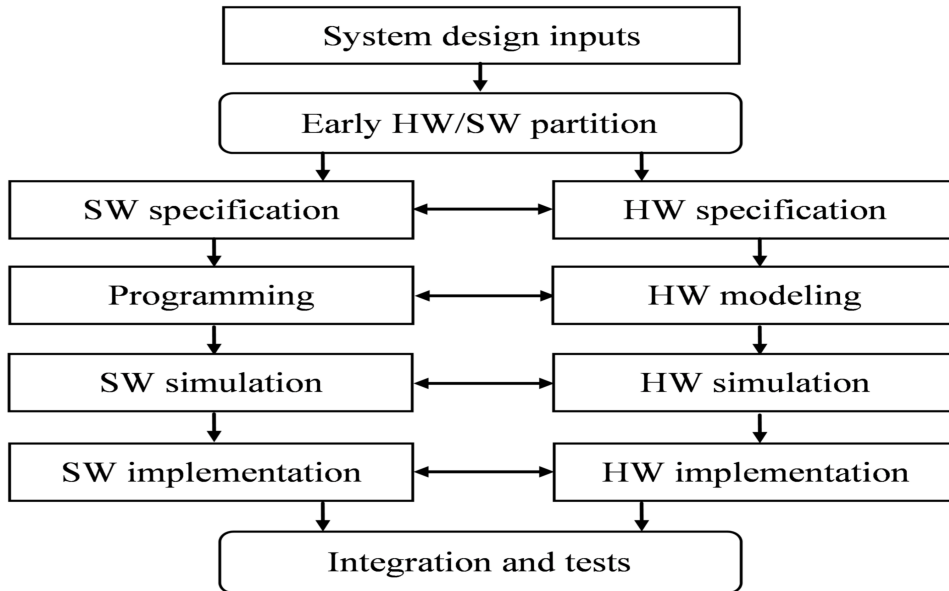


Figure 2.4: Hardware and software co-design framework [59]

Designing an embedded computer system to meet its performance, cost, and design time goals involves a hardware-software co-design approach, where the development of hardware and software components mutually influence each other. This

field focuses on creating systems with programmable CPUs and various hardware components, balancing the roles of software and hardware, or reconfigurable logic, and addressing trade-offs between different implementation technologies.

System design involves implementing desired functionality using physical components. The process must begin with specifying this functionality. The partitioning algorithm improves the overall execution time of the application by optimizing the execution times of software and hardware, the communication time, and the data-path reconfiguration time. It is time to fully embrace hardware-software co-design, fostering collaboration between programming languages and architecture to overcome legacy computing constraints [60, 61].

2.4 High-Level Synthesis design space exploration

With rapid growth in embedded system design, High-Level Synthesis (HLS) is becoming a key role in digital system design. HLS bridges the gap between hardware implementations and high-level algorithmic descriptions. The main advantage of HLS is translating high-level programming languages like SystemC, C, and C++ into hardware description languages (HDLs) such as Verilog or VHDL. HLS tool can carry out tasks like scheduling, pipelining, loop unrolling, and resource allocation to achieve the required hardware specifications while maintaining the functionality of the original algorithm [62].

High-level synthesis is increasingly popular for designing high-performance and energy-efficient heterogeneous systems, reducing time to market, and managing complex systems. HLS allows designers to work at a higher abstraction level by using software programs to specify hardware functionality. It is particularly useful for designing FPGA circuits, enabling easy refinement and replacement of hardware implementations in the target device. Recent years have seen significant activity in HLS research, with numerous tools available from both industry and academia. These tools differ in input languages, internal optimizations, and output quality, making performance comparisons challenging. However, selecting the most suitable HLS for specific algorithms depends on factors such as area, throughput, and

programmer experience [63]. The HLS design flow is shown in Figure 2.5.

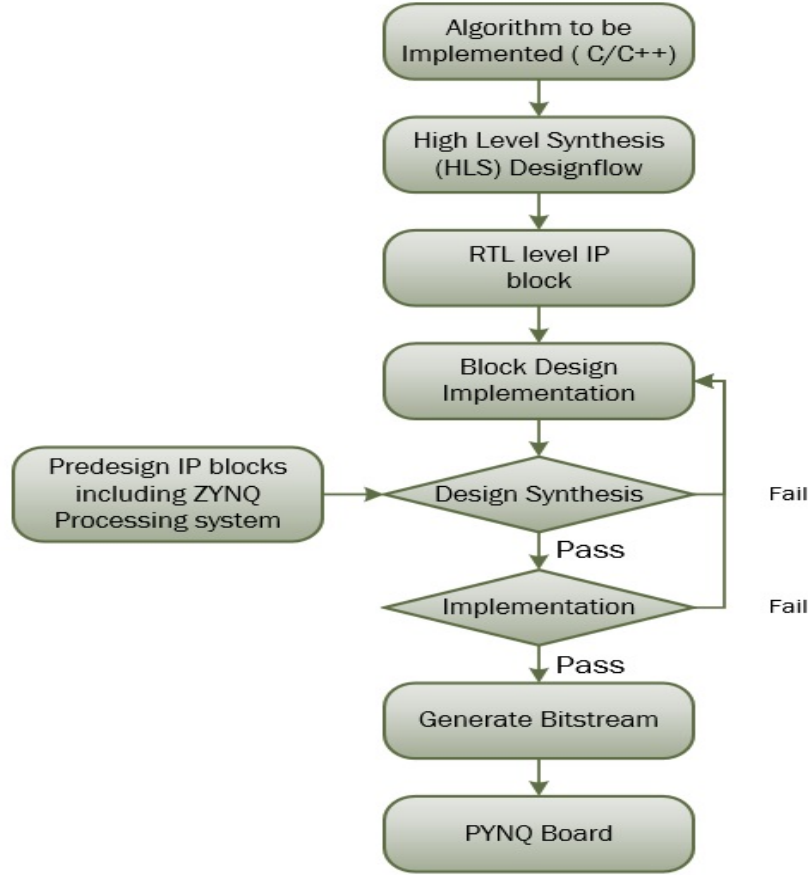


Figure 2.5: HLS Design Implementation flow

HLS enables designers to generate multiple hardware microarchitectures from high-level behavioral descriptions, facilitating the exploration of trade-offs between performance, area, and power. Design space exploration (DSE) in HLS is essential, as the number of possible configurations grows rapidly with synthesis options. Heuristic methods are often employed to efficiently search for Pareto-optimal solutions, guiding the selection of the best hardware design for a given application [64]. This flexibility is particularly valuable for matrix inversion, where algorithmic and architectural choices significantly impact hardware efficiency. While not specific to matrix inversion, advances in HLS for accelerator design—including speculative execution and memory-aware intermediate representations—highlight the trend toward higher abstraction and automation in hardware design. These approaches aim to simplify the mapping of complex algorithms to hardware, improve productivity, and enable efficient exploitation of parallelism and memory hierarchies [65, 66].

2.4.1 Optimization techniques in HLS

Optimization techniques play a crucial role in High-Level Synthesis, enabling the generation of efficient hardware implementations. These techniques are designed to trade-offs among key design metrics such as performance, latency, area, and power consumption [67].

The primary categories of optimization techniques include:

2.4.1.1 Loop Optimizations

Loop optimizations are critical in HLS to reduce execution latency, improve throughput, and maximize resource efficiency. They enable the transformation of loop structures to better suit hardware execution.

(a) Loop Pipelining

Loop pipelining transforms a loop by overlapping the execution of its successive iterations. This technique divides the operations within the loop into pipeline stages, allowing a new iteration to begin before the previous one completes, thus improving overall throughput. The cycle-by-cycle execution pipeline schedule diagram for a loop is shown in Figure 2.6.

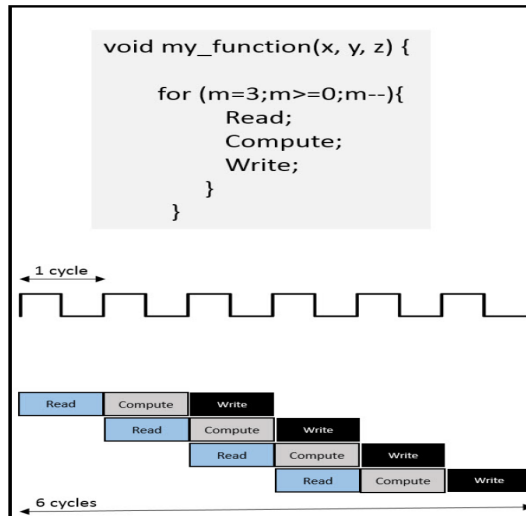


Figure 2.6: Loop Pipelining optimization [68]

(b) **Loop unroll**

Loop unrolling is used to parallelize operations and process more iterations in a single clock cycle, as shown in Figure 2.7. So, loop unrolling improves the performance of loops by reducing latency when synthesizing hardware and effectively increases throughput.

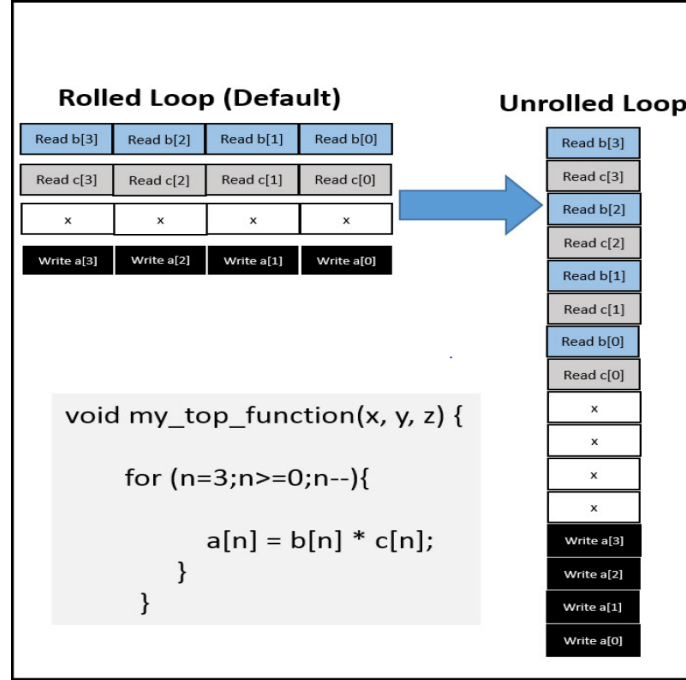


Figure 2.7: Loop unroll optimization [68]

2.4.1.2 Function Pipeline optimization

It is similar to a loop pipeline; in this optimization technique, different stages of a function can be executed in parallel. This is helpful when the sequence of multiple operations is present in a function, and in this case, with the help of optimization techniques before the completion of the previous iteration, the new iteration will come to process.

2.4.1.3 Dataflow optimization

The dataflow optimization technique is used in high-level synthesis (HLS) to improve parallelism and throughput in FPGA designs [69]. By applying the dataflow

directive, different functions or loops within a hardware block can be scheduled to operate concurrently, allowing data to flow between them through FIFOs or channels without waiting for the entire operation to complete. This pipelined execution model enables multiple stages of computation to process data simultaneously, significantly increasing overall performance, especially in streaming or iterative applications. Dataflow optimization is particularly beneficial in designs where tasks can be decomposed into independent stages, such as image processing pipelines, signal processing chains, or machine learning inference engines.

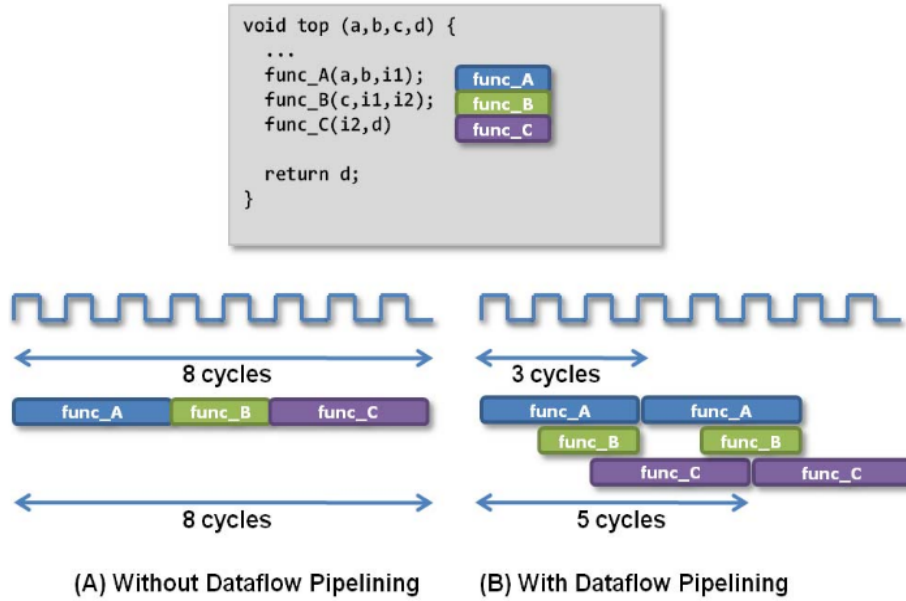


Figure 2.8: Dataflow optimization [70]

The Figure 2.8 compellingly illustrates the differences in function execution with and without dataflow pipelining. In scenario (A), functions are executed sequentially, resulting in significantly higher latency. However, in scenario (B), the implementation of dataflow pipelining allows for overlapping execution, dramatically reducing the total latency from 8 clock cycles down to just 5 clock cycles. This demonstrates the powerful impact of dataflow pipelining on performance efficiency.

2.4.2 Data transfer interfaces

In HLS, data transfer interface mechanisms are used to move data between the synthesized hardware design module (FPGA) and external devices, such as memory

or other peripherals[71]. We know that the selection of interface mechanisms will impact performance, especially in data-intensive applications. Different data transfer interfaces can be defined based on how data can be accessed in our design and control requirements.

The available primary data transfer interfaces are shown in Figure 2.9.

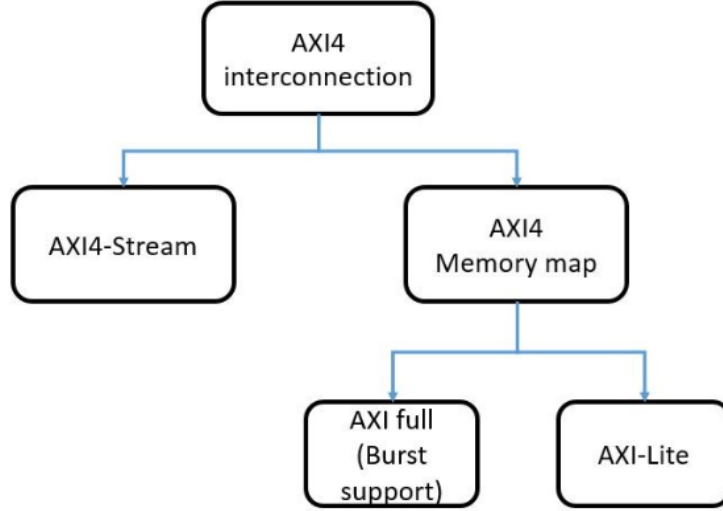


Figure 2.9: AXI4 interfaces

1. AXI4 (Memory-Mapped Interface)

The AXI4 (Advanced eXtensible Interface 4) memory-mapped protocol is a high-performance, synchronous communication interface designed to connect master and slave components in system-on-chip (SoC) and FPGA-based designs. It supports burst-based read and write transactions with separate address and data channels, allowing for high-throughput access to memory and peripherals. AXI4 can handle multiple outstanding transactions and out-of-order responses, making it ideal for complex, high-speed systems that require efficient memory access. Its flexibility and scalability have established it as a standard choice for interfacing processors, memory controllers, and high-bandwidth peripherals in modern digital designs.

2. AXI4-Lite

AXI4-Lite is a simplified version of the AXI4 (Advanced eXtensible Interface) protocol, specifically designed for low-throughput, memory-mapped commu-

nication. It is commonly used to configure and control hardware components in FPGA and SoC designs. The interface supports basic read and write operations using a reduced set of signals, but it does not support burst transfers. This makes AXI4-Lite ideal for connecting to control registers and lightweight peripherals. Its straightforward design allows for easy integration with standard IP cores, and it is often used alongside AXI4 or AXI4-Stream interfaces in more complex systems.

3. AXI4-Stream

The AXI4-Stream protocol is a streamlined data transfer interface designed for transmitting large volumes of data between components, particularly in FPGA and SoC designs, as shown in Figure 2.10. It operates without the need for addressing, making it ideal for continuous data flow applications, such as video processing, audio pipelines, and high-speed data acquisition. Communication is facilitated through a handshake mechanism that uses TVALID and TREADY signals to ensure proper synchronization. Additionally, optional signals like TLAST and TUSER can provide control information or metadata for packet-based processing[72, 73].

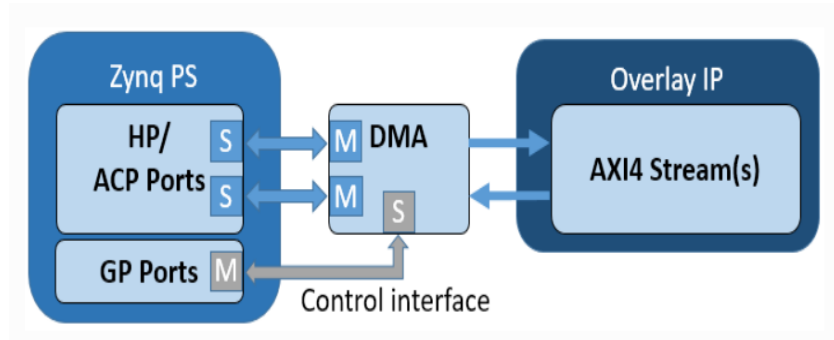


Figure 2.10: DMA interface [74]

2.5 Pynq FPGA

In the present work, the Pynq-Z1 development board has been employed for the hardware implementation of the proposed algorithms. The board is initialized by installing the Pynq (Python productivity for Zynq) operating system (OS) onto an

SD card, enabling the system to boot directly through the Pynq OS. This operating system provides a pre-configured environment that integrates essential Python libraries along with Jupyter notebook support, thereby facilitating rapid prototyping and hardware-software interaction. The Pynq-Z1 board offers multiple connectivity interfaces, including USB, Ethernet, PMODs, HDMI (input and output), and JTAG. X. It is equipped with 1 GB of DDR3 RAM, which provides sufficient memory to execute a Linux-based system efficiently, and incorporates 4 GB of onboard eMMC storage for the operating system and data management [75]. Due to these features, the board finds extensive applications in embedded system development, particularly in domains such as networking, real-time signal processing, and high-performance computing. Furthermore, even in cases where algorithmic complexity or hardware resource limitations may arise, system design can be efficiently realized through a hardware-software co-design approach [76].

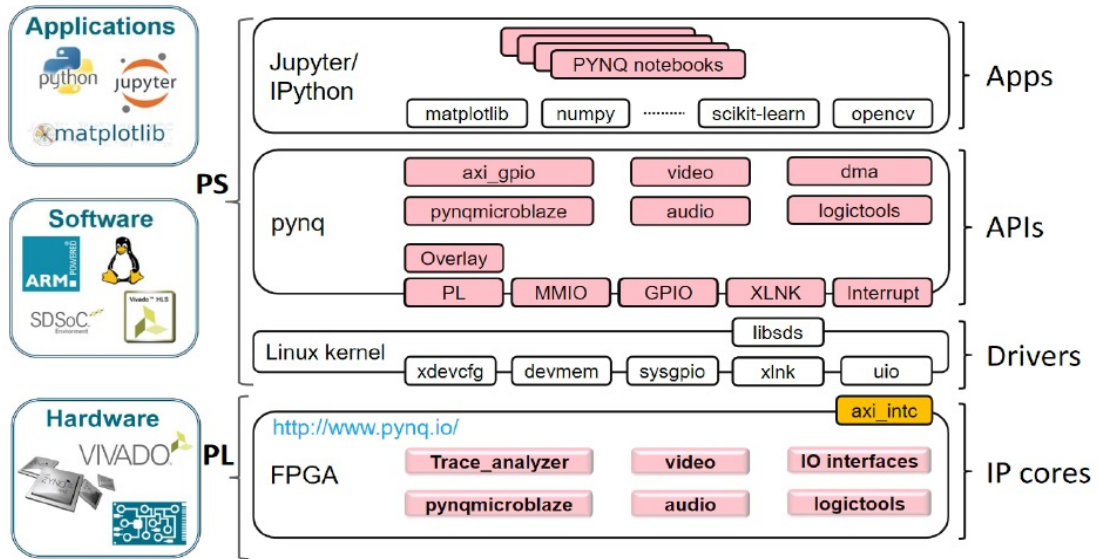


Figure 2.11: Pynq framework architecture [77]

The architecture of the Pynq system is organized into three distinct layers: the application layer, the software layer, and the hardware layer, as shown in Figure 2.11. The application layer is primarily based on the Jupyter Notebook environment, providing an interactive, browser-based interface for executing Python scripts, data visualization, and hardware control. This layer also integrates tools such as IPython and Matplotlib, enabling seamless interaction with the underlying hardware through

user-friendly computational workflows [78].

The software layer operates on the Zynq processing system and is built upon a Linux kernel, typically Ubuntu, with an embedded Python environment. Within this layer, the Pynq API library plays a crucial role by establishing communication between the software and hardware domains. Through this API, hardware overlays can be accessed and controlled via Python scripts, allowing users to configure, monitor, and dynamically reprogram FPGA logic functions [79].

At the lowest level, the hardware layer comprises the FPGA fabric and the processing system. The FPGA portion is configured using overlays, which represent the hardware design abstraction. These overlays encapsulate the programmable logic and can be dynamically invoked or modified through software APIs, thereby enabling flexible hardware–software co-design and runtime adaptability [80].

2.5.1 **Pynq-Z1 board setup**

The Pynq-Z1 board setup involves flashing a pre-built Pynq Linux image onto a microSD card, inserting it into the board, and powering it via USB or an external supply. Once booted, the board can be accessed through a web-based Jupyter Notebook interface over Ethernet or USB, enabling rapid prototyping and control of hardware accelerators using Python and overlays.

CHAPTER 3

HARDWARE ACCELERATION OF EIGENVALUE DECOMPOSITION AND SQUARE ROOT OF MATRIX

3.1 Chapter Overview

Efficient computation of eigenvalues for symmetric matrices plays a vital role in a wide range of scientific and engineering applications. It forms the basis of several key algorithms, such as Zero Phase Component Analysis (ZCA) and Principal Component Analysis (PCA), and is particularly critical in real-time applications, including optical flow estimation. In this work, eigenvalue decomposition is implemented using the Lanczos and implicit TriQR algorithms, followed by the development of a hardware eigenvalue decomposition IP core on the Pynq-Z1 FPGA platform. An AXI-Lite interface is employed to facilitate communication between the processor and the hardware IP. The proposed hardware design is evaluated in terms of acceleration factor and resource utilization across varying input matrix dimensions.

Experimental results demonstrate a hardware acceleration factor ranging from $8\times$ to $18\times$ for matrix sizes from 10×10 to 20×20 . Furthermore, matrix square root computation is realized using the obtained eigenvalues and eigenvectors for different matrix sizes, and corresponding performance metrics, including acceleration factors and resource utilization, are reported.

3.2 Introduction

The computation of eigenvectors and eigenvalues is the most challenging step in many applications, such as machine learning and communication systems [81]. This section provides an overview of various approaches used to solve eigenvalue problems and explore the relationships between existing algorithms.

Suppose B is the square matrix that has matrix size $m \times m$, eigenvalue (λ), and eigenvector (v). The relationship between them is given as

$$Bv = \lambda Iv \tag{3.1}$$

$$Bv - \lambda Iv = 0 \tag{3.2}$$

$$(B - \lambda I)v = 0 \tag{3.3}$$

B matrix is considered defective when the algebraic multiplicity of an eigenvalue is not equal to its geometric multiplicity. The calculation of eigen pairs λ, v for such matrices presents challenges. This raises the critical question of how to efficiently evaluate eigenvalues and eigenvectors for both non-defective and defective matrices while ensuring high levels of accuracy and precision.

In general, the eigenvalues can be obtained using different methods such as analytical method, Jacobi, Power, GolubKhan, Givens Rotation, Householder, QR method and Hessenberg techniques [82, 83].

3.2.1 Analytical method

This analytical method to evaluate the eigenvalue and eigenvector is efficient for small matrices [84].

Suppose B be the square matrix of size $m \times m$ and compute the eigenvalue decomposition from the characteristic polynomial by solving:

$$\det(B - \lambda I) = 0 \quad (3.4)$$

where λ is the eigenvalue and matrix I is known as the identity matrix of size $m \times m$.

From equation (3.4), the characteristic polynomial of matrix B can be obtained, which yields m roots corresponding to the m eigenvalues of B . The associated eigenvectors are then determined from these eigenvalues. However, this approach is generally practical only for matrices of size less than 5, since for larger matrices of size more than 5, solving the characteristic polynomial to obtain the roots becomes computationally difficult [85].

3.2.2 Jacobi method

The Jacobi method is a robust iterative algorithm that has proven essential in various fields where symmetric matrices are prevalent. Its applications span computational physics, engineering challenges, structural mechanics, principal component analysis (PCA) in statistics, and quantum mechanics [86]. A significant advantage of the Jacobi method is its ability to guarantee real eigenvalues for symmetric matrices.

Jacobi used orthogonal similarity transformations to convert a real symmetric matrix into a diagonal matrix, where diagonal entries are the eigenvalues [87]. These transformations are based on a plane of rotation called the Jacobi rotation, as shown in Figure 3.1.

In Figure 3.1, the rotation angle θ between XY plane and X^1Y^1 , then $R(x, y)$ plan is transformed to

$$x = OE + EF = OE + CD = x' \cos \theta + y' \sin \theta \quad (3.5)$$

$$y = RD - FD = RD - EC = y' \cos \theta - x' \sin \theta \quad (3.6)$$

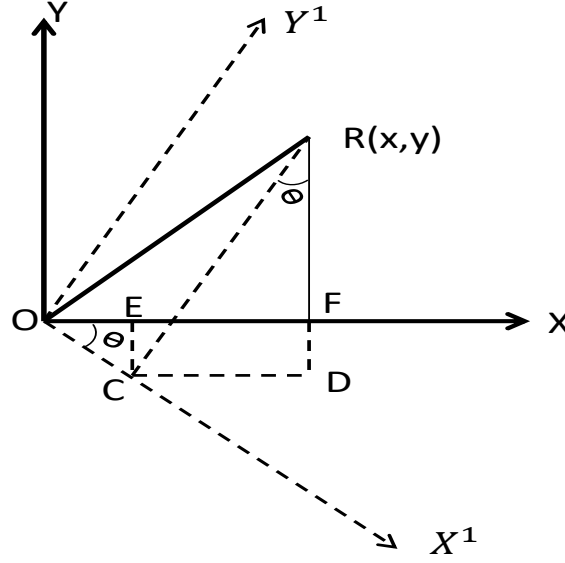


Figure 3.1: Jacobi plane of rotation [88]

Q be the orthogonal change of basis is formulated as shown below

$$Q = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix} = Q_r \quad (3.7)$$

The mapping of position vectors with

$$Q^{-1} = Q^T = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad (3.8)$$

From the Equation (3.8), it is clearly given that the plane of rotation is an orthogonal matrix. Therefore, For $m \times m$ square matrix of m dimensional space, The plane of rotation $R(p, q, \theta)$ as shown in equation (3.9).

$$R = \begin{bmatrix} 1 & 0 & . & . & \dots & 0 & 0 \\ 0 & \ddots & . & . & . & . & . \\ . & & \cos \theta & 0 & \sin \theta & . & . \\ 0 & \dots & 0 & 1 & 0 & 0 & 0 \\ . & . & -\sin \theta & 0 & \cos \theta & . & . \\ \vdots & . & . & . & . & \ddots & \vdots \\ 0 & . & . & . & . & \dots & 1 \end{bmatrix} \quad (3.9)$$

Let's consider a real symmetrix square matrix B as

$$B = \begin{bmatrix} b_{11} & \dots & b_{1p} & \dots & b_{1q} & \dots & a_{1m} \\ b_{21} & \dots & b_{2p} & \dots & b_{2q} & \dots & b_{2m} \\ \vdots & \ddots & & & & & \vdots \\ b_{p1} & \dots & b_{pp} & \dots & b_{pq} & \dots & b_{pm} \\ \vdots & & & \ddots & & & \vdots \\ b_{q1} & \dots & b_{qp} & \dots & b_{qq} & \dots & b_{qm} \\ \vdots & & & & & \ddots & \vdots \\ b_{m1} & \dots & b_{mp} & \dots & b_{mq} & \dots & b_{mm} \end{bmatrix} \quad (3.10)$$

Finding the maximum off-diagonal element in a matrix after each rotation can be quite time-consuming. This challenge led to the development of the Cyclic Jacobi method [90], which computes the rotation matrices in sequence. As the size of the matrix increases, the number of required rotation matrices also grows, resulting in a slow convergence rate. For a matrix of size $m \times m$, the minimum number of rotation matrices needed is $\frac{m \times (m-1)}{2}$.

The success of the Jacobi method in diagonalizing real symmetric matrices with orthogonal transformations (specifically Givens rotations) and reliable accuracy have inspired many researchers to develop similar methods. However, the Jacobi technique is applicable for smaller symmetric matrices and it is not suitable for large-size matrices.

Algorithm 6 Jacobi algorithm[89]

Input: $B \in R^{m \times m}$, The diagonal matrix is evaluated using similarity transformation $B = R_{pq}^T B R_{pq}$

Output: B and V

```

1: Let  $V = \text{Identify matrix}$ ,  $\text{max iter} = m$ ,  $\text{iter} = 1$ ,  $\text{threshold} = \epsilon$ 
2: while (  $\text{iter} < m$  ) do
3:   Find out the maximum off-diagonal element of  $B$ , then note down the  $q^{th}, p^{th}$ 
      column and row
4:   if (  $\text{max} > \text{threshold}$  ) then
5:     Evaluate the rotation angle that is  $\theta$ , update the matrix  $R_{pq}$  ;
6:      $B = R_{pq}^T B R_{pq}$  ;
7:      $V = V \times R_{pq}$ ;
8:      $\text{iter} = \text{iter} + 1$ ;
9:   else
10:    return  $B$  and  $V$ ;
11:   end if
12: end while

```

3.2.3 Power method

The Power method is an iterative technique used to compute the dominant eigenvalue of a matrix, along with its corresponding eigenvector, particularly for large square matrices [91]. Originally proposed in 1913, this method requires the matrix to be diagonalizable, implying the existence of a complete set of linearly independent eigenvectors. In addition, the matrix must possess a unique largest eigenvalue (in magnitude), ensuring that for a suitably chosen initial vector x , the sequence of iteratively generated vectors converges to a scalar multiple of the dominant eigenvector.

Let's consider a matrix B of size m having m eigenvalues with sequence of

$$|\lambda_1| > |\lambda_2| \geq \dots |\lambda_{n-1}| > |\lambda_m| \quad (3.11)$$

The vector x can be represented by a full set of m orthogonal eigenvectors

$v_1.v_2.....v_m$ is given as

$$x = \alpha_1 v_1 + \alpha_2 v_2 \alpha_m v_m \quad (3.12)$$

Where $\alpha_1, \alpha_2 \alpha_m$ are coefficients.

$$B^p x = \lambda_1^p \left[\alpha_1 v_1 + \left(\frac{\lambda_2}{\lambda_1} \right)^p \alpha_2 v_2 + \left(\frac{\lambda_3}{\lambda_1} \right)^p \alpha_3 v_3 + \cdots \left(\frac{\lambda_m}{\lambda_{m-1}} \right)^p \alpha_m v_m \right] \quad (3.13)$$

$$\text{As } P \rightarrow \infty, B^p x \rightarrow \lambda_1^p \alpha_1 v_1 \quad (3.14)$$

$$\lambda_1 = \lim_{P \rightarrow \infty} \frac{(B^p x)_r}{(B^{p-1} x)_r} r = 1, 2, 3 \dots \dots m \quad (3.15)$$

Here λ_1 is denoted as the larger eigenvalue.

Algorithm 7 Power method [92]

Input: $B \in R^{m \times m}$

Output: Λ^k and v^k

- 1: **for** $k = 1, 2, 3 \dots n$ **do**
 - 2: $q^k = B v^{k-1}$
 - 3: $v^k = (q^k) / ||q^k||_2$.
 - 4: $\Lambda^k = (v^k)' B^k$.
 - 5: return v^k and Λ^k
 - 6: **end for**
-

To find the smallest eigenvalue, the same approach is followed by shifting the input matrix B by λ_1 i.e., $B - \lambda_1 I$. The significant advantage of the power method is that it is simple and applicable to larger matrices. The main drawback of this method is that its convergence acceleration depends on the most significant eigenvalue and second-largest eigenvalue, and it is also very time-consuming. Later, several popular iterative algorithm methods are proposed to find eigenvalues and eigenvectors, like Arnoldi, QR, and Lanczos.

3.2.4 Givens Rotation method

The Jacobi method determines eigenvalues by iteratively diagonalising the matrix, but it can be inefficient due to repeated transformations that undo previously achieved zeros. As a result, an improved method known as Givens Rotation has been developed, building upon the Jacobi approach to overcome these limitations. The robust approach to evaluating eigenvalues and eigenvectors of input square matrices involves reducing the matrix into a simpler form and initiating an iterative step process [93]. Specifically, for symmetric matrices, achieving a tridiagonal form is advantageous because this transformation can be achieved in a finite number of steps. Once in tridiagonal form, computations involving the matrix can be performed more efficiently for subsequent operations.

Algorithm 8 Givens Rotation algorithm [94]

Input: $B \in R^{m \times m}$, The diagonal matrix is evaluated using similarity transformation $B = R_{pq}^T B R_{pq}$

Output: B and V

- 1: initialize $V = \text{Identify matrix}$, $maxiter = k(k < m)$, $iter = 1$ threshold $= \epsilon$
 - 2: **while** ($iter < k$) **do**
 - 3: Find out the maximum off-diagonal element of B , then note down the q^{th}, p^{th} column and row
 - 4: **if** ($max > threshold$) **then**
 - 5: evaluate the rotation angle θ to 0, b_{rq} element of B , where $r = p - 1$;
 - 6: Form the rotation matrix R_{pq} by following rotation angle θ ;
 - 7: $B = R_{pq}^T B R_{pq}$;
 - 8: $V = V \times R_{pq}$;
 - 9: $iter = iter + 1$;
 - 10: **else**
 - 11: return the matrices B and V ;
 - 12: **end if**
 - 13: **end while**
 - 14: return the matrix B
-

3.2.5 Householder method

The householder technique is more stable in solving the eigenvalues compared to the Givens Rotation method. This method can give a sequence of orthogonal transformations by transforming the real symmetric square matrix to a tridiagonal matrix [95], and it can be obtained from the plane of reflection as shown in the figure 3.2.

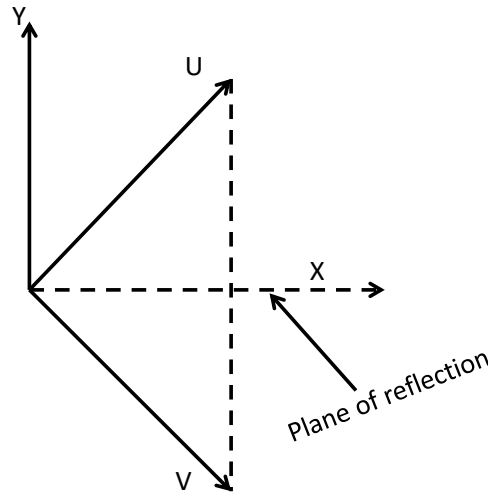


Figure 3.2: Plane of reflection for Householder transformation [96]

Let us consider u, v are vectors, $\|v\| = \|u\|$ and $y = \frac{u-v}{\|u-v\|}$ as shown figure 3.2

Householder reflection matrix $H = (I - 2yy^T)$ is symmetric and orthogonal.

(3.16)

x is any vector that is orthogonal to y can be written as

$$Hx = (I - 2yy^T)x = Ix - 2yy^Tx = Ix = x \quad (3.17)$$

Hx gets mapped to the same vector x

when the Householder plane of reflection is projected on y

$$Hy = (I - 2yy^T) y = Iy - 2yy^T y = y - 2yI = y - 2y = -y \quad (3.18)$$

From figure 3.2 it is observed that

$$Hv = u; Hu = v \quad (3.19)$$

where u and v both are orthogonal to each other.

let us consider a $m \times m$ symmetric matrix B and;

$$\text{Let } u_1 = \begin{bmatrix} b_{21} & b_{31} & \dots & \dots & b_{m1} \end{bmatrix}^T \text{ and } v_1 = \|u_1\| e_1 \quad (3.20)$$

Construct the orthogonal similarity transformation matrix as shown equation 3.21.

$$R_1 = \begin{bmatrix} 1 & 0 \\ 0 & H_{m-1} \end{bmatrix} \quad (3.21)$$

matrix B is transformed using similarity transformation as shown in equation

$$B^1 = R_1^T B R_1 = \begin{bmatrix} 1 & 0 \\ 0 & H_{m-1} \end{bmatrix} \begin{bmatrix} b_{11} & u^T \\ u & B_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & H_{m-1} \end{bmatrix} \quad (3.22)$$

$$B^1 = \begin{bmatrix} b_{11} & v^T \\ v & H_{m-1} B_1 H_{m-1} \end{bmatrix} \quad (3.23)$$

$v_2 = \|u_2\| e_1$ for R_2

$$R_2 = \begin{bmatrix} I_2 & 0 \\ 0 & H_{m-2} \end{bmatrix} \quad (3.24)$$

after $m - 2$ steps with $R = R_1 R_2 \dots R_{m-2}$

The symmetric tridiagonal matrix can be formed using equation 3.24. By using Strum sequence property, we can easy determine the eigenvalue.

3.2.6 QR method

The QR technique became more popular around the 1960s and also it is applicable to square matrices to evaluate eigenvalues and eigenvectors. This QR technique factorizes matrix B into matrix Q and matrix R using the Gram-Schmidt technique.

Algorithm 9 QR algorithm [97]

Input: $B \in R^{m \times m}$

Output: B_m and V_m

- 1: Initialise $B_0 = B, V = I$
 - 2: **for** $t = 1, 2, 3, \dots, m$ **do**
 - 3: $[Q_t, R_t] = qr(B_{t-1})$
 - 4: $B_t = R_t \times Q_t$.
 - 5: $V_t = V_{t-1} \times Q_t$.
 - 6: **end for**
 - 7: return B_m and V_m
-

The matrix B converges to a form that makes it easy to identify its eigenvalues. The eigenvalues of B_t and B_{t-1} are identical due to their similarity [98]. While QR factorization is typically fast, it can become time-consuming and expensive when repeated multiple times. To reduce the cost per iteration and speed up convergence, several shifting methods are employed. After 1970, the eigenvalue problem for dense matrices of moderate size was essentially solved, prompting researchers to seek approaches for higher-dimensional matrices. Unfortunately, the QR method is not directly preferred for computations; it is generally applied to matrices that have been transformed into a simpler form, such as tridiagonal matrices.

3.2.7 Hessenberg method

Due to unbalanced, defective information in non-symmetric matrices, finding the eigenvalues and eigenvectors is challenging, leading to higher roundoff errors. Additionally, defective matrices cannot have a complete set of eigenvectors, making them non-diagonalizable. Hessenberg's method is suitable for resolving the eigen-

value problem in non-symmetric matrices [99]. Similar to symmetric matrices, non-symmetric matrices are first reduced to a more straightforward form to efficiently compute eigenvalues [100].

The matrix form of the Hessenberg matrix is given below

$$H = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{bmatrix} \quad (3.25)$$

Except for the first sub-diagonal row, everywhere zeros below the diagonal element of the Hessenberg matrix. We can get the Hessenberg matrix through a sequence of Householder transformations. After getting the Hessenberg matrix, we employed the QR or QL iterative process to find the eigenvalues and eigenvectors.

The Table 3.1 summarizes various eigenvalue decomposition methods, each designed for specific types of matrices and computational needs. Analytical methods are effective for small matrices, typically those with dimensions up to 5, as they directly solve the characteristic polynomial. However, as the size of the matrices increases, iterative techniques become more practical. The Power method is commonly used to find dominant eigenvalues in general matrices. In contrast, the Jacobi and Givens Rotation methods are better suited for symmetric and real matrices. Although the Jacobi method is accurate, it tends to converge slowly. On the other hand, Givens Rotation offers improved computational efficiency for larger matrices. The QR and Hessenberg methods are widely used for general or non-symmetric matrices. The QR method is stable but may experience slow convergence when eigenvalues are repeated. More advanced techniques, such as Householder and Lanczos, are particularly effective for large symmetric matrices. The Lanczos method, in particular, is efficient due to its reliance on vector-based operations and its low memory requirements.

Eigen Method	Matrix characteristics	Analysis
Analytical	General	This simple technique is suitable for matrices up to size 5 to extract eigenvalues from the characteristic polynomial.
Jacobi	Symmetric and Real	It works best for lower-order matrices. The input matrix can be transformed into a diagonal matrix for accurate eigenvalue evaluation, but convergence may be slow.
Power	General	It is a stable method to find the extreme eigenvalues iteratively for the higher-order matrices. convergence depends on the ratio between the two largest eigenvalues.
Givens Rotation	Symmetric and Real	It is suitable for higher-order matrices and it has a computational advantage compared to the Jacobi method.
Householder	Symmetric and Real	The technique is stable with minimal errors. However, if the eigenvalues are similar, convergence may decline, requiring more storage as the matrix is updated each iteration.
QR	General	The eigenvalues are diagonal entities of the (R) matrix. Its converges become very slow when repeated eigenvalues are present.
Hessenberg	Non-symmetric	This method converts the input matrix to the Hessenberg matrix and employs the QR or QL technique to evaluate eigenvalues
Lanczos	Symmetric and Real	It requires less computational effort compared to the Householder method due to the vector-based nature of its matrix operations. Additionally, it has minimal storage requirements because the original matrix is updated only once.

Table 3.1: Summary of eigenvalue decomposition methods

3.3 Eigenvalue decomposition using Lanczos algorithm

Suppose matrix B is a large, sparse, positive definite, and symmetric matrix; then, the finding of eigenvalues and eigenvectors is advantageous using the Lanczos method. The Lanczos algorithm is an iterative method that Cornelius Lanczos can devise to find the eigenvalues.

Inputs of algorithm:-

B is input matrix, k is number of iterations need to perform, q_0 - initialize the orthonormal unit vector.

Output of algorithm:-

$\mathcal{T}$ is Tridiagonal matrix and Q is Orthogonal matrix.

where,

$$\mathcal{T} = \begin{bmatrix} \alpha_1 & \beta_1 & & 0 \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{m-1} \\ 0 & & \beta_{m-1} & \alpha_m \end{bmatrix} \text{ and } Q = [q_1, \dots, q_m]$$

where $q_1, \dots, q_m$ are the column vectors of orthogonal Q matrix

$\beta_2, \beta_3, \dots, \beta_m$ and $\alpha_1, \alpha_2, \dots, \alpha_m$ are sub-diagonal and diagonal elements of $\mathcal{T}$.

The elements $(\alpha_1, \alpha_2, \dots, \alpha_m, \beta_2, \beta_3, \dots, \beta_m)$ of the tridiagonal matrix $\mathcal{T}$ are computed using orthogonal similarity transformation as shown below

$$\mathcal{T} = Q^T B Q \quad (3.26)$$

Equation 3.26 can be rewritten as

$$B Q = Q \mathcal{T} \quad (3.27)$$

$$B[q_1, q_2, \dots, q_k] = [q_1, q_2, \dots, q_k] \mathcal{T} \quad (3.28)$$

From equation 3.28 we can get

$$B q_k = q_k \alpha_k + q_{k-1} \beta_{k-1} + \beta_k q_{k+1} \quad (3.29)$$

multiplying equation 3.29 by q_k^T on both sides then

$$q_k^T B q_k = q_k^T q_k \alpha_k + q_k^T q_{k-1} \beta_{k-1} + q_k^T \beta_k q_{k+1} \quad (3.30)$$

The property of orthonormality $q_k q_{k-1}^T = 0 = q_k q_{k+1}^T$. From this Equation 3.30 becomes

$$\alpha_k = q_k^T B q_k \quad (3.31)$$

$$r_k = \beta_k q_{k+1} = B q_k - q_k \alpha_k - q_{k-1} \beta_{k-1} \quad (3.32)$$

$$r_k = B q_k - \alpha_k I q_k - q_{k-1} \beta_{k-1} \quad (3.33)$$

$$r_k = (B - \alpha_k I) q_k - q_{k-1} \beta_{k-1} \quad (3.34)$$

$$r_k = \beta_k q_{k+1} \quad (3.35)$$

$$q_{k+1} = \frac{r_k}{\beta_k} \quad (3.36)$$

$$\text{Since } \|q_{k+1}\| = 1 \quad (3.37)$$

$$\beta_t = \|R_t\| \quad (3.38)$$

Algorithm 10 Lanczos algorithm

Input: $\Sigma \in \mathbb{R}^{m \times m}$, t .

Output: $Q \in \mathbb{R}^{m \times m}$ and $\mathcal{T} \in \mathbb{R}^{m \times m}$.

- 1: Initialize $t = 0$, R_0 , $q_0 = q_1$ and $\beta_0 = 1$
 - 2: **while** $j=0$ to t or $\beta_t \neq 0$ **do**
 - 3: $q_{t+1} = R_t / \beta_t$
 - 4: $t = t + 1$
 - 5: $\alpha_t = q_t^T \cdot \Sigma \cdot q_t$
 - 6: $R_t = (X - \alpha_t \cdot I) \cdot q_t - q_{t-1} \cdot \beta_{t-1}$
 - 7: $\beta_t = \|R_t\|$
 - 8: **end while**
 - 9: β_t and α_t are sub-diagonal and diagonal elements of $\mathcal{T}$. augmenting column vectors of q_t provides Q
-

From the above algorithm, we can easily say that the matrix B is not changing during the entire process, so that this method is useful for large sparse matrices. Usually, this algorithm is considered unstable compared to the Householder method due to the loss of orthogonality in the Lanczos vector, but it has advantages in storage, speed, and simplicity. To maintain orthogonality in the Lanczos vector, the reorthogonalization technique is used. After getting the Tridiagonal matrix $\mathcal{T}$ and

the Orthogonal matrix Q , then we will apply the Implicit TriQR algorithm to get eigenvalues and eigenvectors.

Implicit TriQR

The main aim of implicit triQR algorithm is to calculate the eigenvectors and eigenvalues by reducing the tridiagonal matrix $\mathcal{T}$ to the diagonal matrix. This can be done by a sequence of an orthogonal transformation with help of Givens Rotation method in an iterative process along with Wilkinson shift technique is used to accelerate the convergence.

Inputs to Implicit TriQR algorithm:-

- (i) Input matrix are tridiagonal matrix $\mathcal{T}$
- (ii) Orthogonal matrix Q

Output of Implicit TriQR algorithm:-

- (i) Eigenvalues D
- (ii) Eigenvectors V

1. Let $\mathcal{T} \in \mathbb{R}^{m \times m}$ with $\alpha_1, \alpha_2 \dots \alpha_m$ are diagonal elements and $\beta_2, \beta_3, \dots \beta_m$ are sub-diagonal elements
2. Initialize $Iter_{max} = t, m = n$
3. While ($m > 1$)
4. $g = \frac{(\alpha_{m-1} - \alpha_m)}{2}$
5. if ($g == 0$)
6. $s = \alpha_m - |\beta_m|$
7. else
8. $S = \frac{\alpha_m - |\beta_m| |\beta_m|}{g + \text{sign}(g) * \sqrt{g^2 + \beta_m^2}}$
9. end if

10. $x = \alpha_1 - s, y = \beta_2$
11. for($l = 1 : \text{to } m - 1$)
12. If ($m > 2$)
13. $[c, s] = \text{givensrotation}(x, y)$
14. else
15. compute $[c, s]$, such that $P^T \mathcal{T}_{2 \times 2} P$ is diagonal
16. end if
17. $w = c \times x - s \times y$
18. $d = \alpha_l - \alpha_{l-1}$
19. $z = 2 \times c \times s \times \beta_{l+1} + d \times s^2$
20. $\alpha_l = \alpha_l - z \alpha_{l+1} = \alpha_{l+1} + z$
21. $\beta_{k+1} = d \times c \times s + \beta_{l+1} (c^2 - s^2), x = \beta_{l+1}$
22. if ($l > 1$)
23. $\beta_l = w$
24. if ($l < m - 1$)
25. $y = -s \times \beta_{l+2}$
26. end if
27. $Q(:, l : l + 1) = Q(:, l : l + 1) \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$
28. end for
29. if($(|\beta_m| < 0.001 \times (|\alpha_m - 1| + |\alpha_m|)) \parallel \text{iter} < Iter_{max} = t$)
30. $m = m - 1, \text{iter} = 0$
31. end if

32. end while

33. After getting convergence $\alpha_1, \alpha_2 \dots \alpha_m \Rightarrow D, Q_{m \times m} \Rightarrow V$

After getting tridiagonal matrix $\mathcal{T}$ and the orthogonal matrix Q from lanczos algorithm then we will apply Implicit TriQR algorithm to get eigenvalue and eigenvectors as shown below.

$$\mathcal{T} - k_i I \rightarrow QR \quad (3.39)$$

where, k_i is the Wilkinson shift.

$$\mathcal{T} = \left(\begin{array}{cc|cccc} \tau_{0,0} - \kappa_i & \tau_{1,0} & 0 & 0 & \cdots & 0 \\ \tau_{1,0} & \tau_{1,1} - \kappa_i & \tau_{1,2} & 0 & \cdots & 0 \\ \hline 0 & \tau_{2,1} & \tau_{2,2} - \kappa_i & \tau_{3,2} & \ddots & \vdots \\ 0 & 0 & \tau_{3,2} & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \tau_{m-2,m-2} - \kappa_i & \tau_{m-1,m-2} \\ 0 & 0 & \cdots & 0 & \tau_{m-1,m-2} & \tau_{m-1,m-1} - \kappa_i \end{array} \right) \quad (3.40)$$

If we choose Wilkinson shift [101] as an eigenvalue of $\begin{pmatrix} \tau_{m-2,m-2} & \tau_{m-1,m-2} \\ \tau_{m-1,m-2} & \tau_{m-1,m-1} \end{pmatrix}$ that is closest to $\tau_{m-1,m-1}$, which can globally converge to annihilate off-diagonal elements to get faster results. For that, we can find out Givens Rotation matrix $G_0^T = \begin{pmatrix} \gamma_0 & \sigma_0 \\ -\sigma_0 & \gamma_0 \end{pmatrix}$, to annihilate all off-diagonal elements. After getting the eigenvalue and eigenvector, then sort the eigenvalues in descending order, and the eigenvector matrix is updated respectively.

3.4 Square root of a matrix

Square root computation is a fundamental but complex operation, especially for non-perfect squares due to their irrational nature and the iterative methods required. Two prominent hardware-based algorithms are the Newton-Raphson Inverse method and the Non-Restoring method. Comparative studies show that both algorithms can be effectively synthesized for hardware, with trade-offs in terms of speed, area, and

implementation complexity. The Newton-Raphson method is iterative and well-suited for hardware parallelization, while the Non-Restoring method offers a more straightforward hardware implementation but may be less efficient for large-scale computations.

The Schur decomposition, Eigenvalue Decomposition are the most prominent approaches for computing Matrix square root. The Schur method computes the square root of a matrix by first reducing it to Schur form ($A = QSQ^H$), then recursively finding the square root of the upper triangular matrix S . This approach is known for its numerical stability and efficiency, especially for general matrices, and is widely used in practice [102–104]. Extensions allow for real arithmetic throughout and address the existence and conditioning of real square root [104]. Blocking and recursive blocking further enhance performance, especially on modern hardware, by leveraging efficient matrix multiplication routines [105].

3.4.1 Schur decomposition method

Suppose B be the square matrix having $m \times m$ to find the square root matrix C such that:

$$C^2 = B$$

Where the square root of a matrix can be obtained using the Schur decomposition [106].

The implementation steps for finding the square root of a matrix are given below:

1. **Input:** B be the square matrix and B having matrix size $m \times m$.
2. **Schur Decomposition:** Evaluate the Schur decomposition of B :

$$B = QRQ^*$$

where Q matrix is a unitary and R matrix is an upper triangular respectively.

3. Square Root of R :

Evaluate the square root of the matrix R for that each diagonal element r_{ii} of R can be computed as:

$$\sqrt{r_{ii}}$$

For compute the off-diagonal elements of R , use the property of upper triangular matrices. If R has the form:

$$T = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{nn} \end{pmatrix}$$

after that, we can compute the square root $R^{1/2}$ as:

$$R^{1/2} = \begin{pmatrix} \sqrt{r_{11}} & \frac{r_{12}}{2\sqrt{r_{11}}} & \cdots & \frac{r_{1n}}{2\sqrt{r_{11}}} \\ 0 & \sqrt{r_{22}} & \cdots & \frac{r_{2n}}{2\sqrt{r_{22}}} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sqrt{r_{nn}} \end{pmatrix}$$

4. Finally, we can get the square root C using below property:

$$C = QR^{1/2}Q^*$$

Square root of a matrix using the Schur decomposition method is good for general square matrices.

3.4.2 Newton-Raphson method

The square root of a matrix is not always unique or even real. However, if the input matrix is a positive definite matrix, a unique principal square root exists that is also positive definite, and numerical methods like the Newton-Raphson iteration can efficiently compute this square root of the matrix [107].

The algorithm steps for finding the square root of a matrix are given below:

Input: B be the square matrix and B having matrix size $m \times m$.

Output: The square root matrix C such that $C^2 = B$:

Steps:

Iteration:

- Set $\text{tol} \leftarrow 10^{-10}$
- $\text{max_iter} \leftarrow 100$
- Choose an initial matrix for $C \leftarrow \text{identity_matrix}(m)$

Initialization:

- Evaluate c_{next}

$$C_{\text{next}} \leftarrow 0.5 \times (C + BC^{-1})$$

- Check for convergence

$$\text{if } \|C_{\text{next}} - C\| < \text{tol}$$

return square root matrix C_{next}

- Upadating $C \leftarrow C_{\text{next}}$

If no convergence:

The convergence is not met within the specified maximum number of iterations.

This technique is efficient and works well for non-singular matrices. The initial guess of the matrix is significantly important in the Newton-Raphson method to achieve convergence.

3.4.3 Eigenvalue decomposition with Lanczos algorithm

The eigenvalue method computes the square root by diagonalizing the matrix, taking the square roots of the eigenvalues and evaluate the product of the matrix of eigenvectors, the square root of the diagonal matrix of its eigenvalues, and the inverse or transpose the matrix of eigenvectors.

The algorithmic steps to find the square root using eigen-decomposition method is listed below:

Input: B be the square matrix B having matrix size $m \times m$.

Output: The square root matrix C such that $C^2 = B$:

1. Evaluate the eigenvalue decomposition:

Suppose a square matrix B having matrix size $m \times m$ and perform the eigenvalue decomposition:

$$B = V\Lambda V^{-1} \quad (3.41)$$

where, Λ is the eigenvalues and V is the eigenvectors.

2. Evaluate the Square Roots of Eigenvalues:

Find the square root of each eigenvalue to get a new diagonal matrix :

$$\sqrt{\Lambda} = \text{diag}(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \dots, \sqrt{\lambda_m})$$

where λ_i are the eigenvalues of matrix B

3. Calculate the square root matrix C :

$$C = \sqrt{B} = V\sqrt{\Lambda}V^{-1} \quad (3.42)$$

In equation 3.41, here, eigenvalues and eigenvectors are extracted with help of the Lanczos algorithm and implicit triQR algorithm due to the advantage of the matrix operations being vector-based and also requiring minimal storage because the original matrix is updated only once.

Recent research explores hardware acceleration for both Schur and eigenvalue-based methods. Blocked and recursive Schur algorithms exploit parallelism and high-performance BLAS libraries, achieving significant speedups on CPUs and GPUs [105]. Mixed-precision and parallel implementations further reduce computation time, especially for high-accuracy requirements [108]. For eigenvalue computations, parallel Jacobi methods and FPGA implementations have demonstrated improved efficiency, particularly in real-time applications [109]. GPU acceleration and instruction-level parallelism are increasingly leveraged for large-scale problems.

3.5 Hardware Results

3.5.1 Eigenvalue Decomposition

Recent advances in high-level synthesis (HLS) have enabled efficient hardware accelerator designs for eigenvalue decomposition (EVD), a core operation in scientific computing, signal processing, and machine learning. Eigenvalue decomposition is computationally intensive and often a bottleneck in applications such as principal component analysis (PCA) and direction-of-arrival (DoA) estimation. Hardware acceleration, especially using FPGAs and GPUs, is crucial for real-time and large-scale processing. HLS tools, such as Xilinx Vivado HLS, allow designers to describe algorithms at a higher abstraction level, improving productivity and enabling rapid design space exploration.

Emerging hardware accelerators, such as GPUs, offer immense computational capacity, but memory bandwidth limitations can make EVD memory-bound rather than compute-bound. Recent research proposes algorithmic optimizations—such as improved tridiagonalization and randomized methods—to better exploit hardware parallelism and alleviate memory bottlenecks, achieving significant speedups over state-of-the-art libraries [110]

Before the hardware implementation of eigenvalue decomposition, we first implemented the algorithm on software and reported the software execution time. After the software implementation, it was implemented on hardware using a 32-bit floating-point data type on the Pynq-Z1 board for various matrix sizes and reported resource utilisation and hardware execution time. Finally reported hardware acceleration factor.

The SoC hardware implementation of eigenvalue decomposition on Pynq-Z1 board has two stages. In the first stage, the hardware of eigenvalue decomposition is developed using HLS technique. We used pipeline directive i.e., pragma, to improve the hardware latency. We designed eigenvalue decomposition hardware using the Lanczos and implicit TriQR algorithm due to its advantages mentioned before. In

the HLS, we followed the development flow as mentioned below:

- Designed the hardware eigenvalue decomposition using pipeline directive and validated its functionality.
- Synthesized the design and created the hardware IP.
- Created an Overlay for this IP block combined with ZYNQ IP core, then synthesized and implemented the design on Pynq-Z1 board.
- Realised the design on the Pynq-Z1 board in Jupyter Notebook using Python.

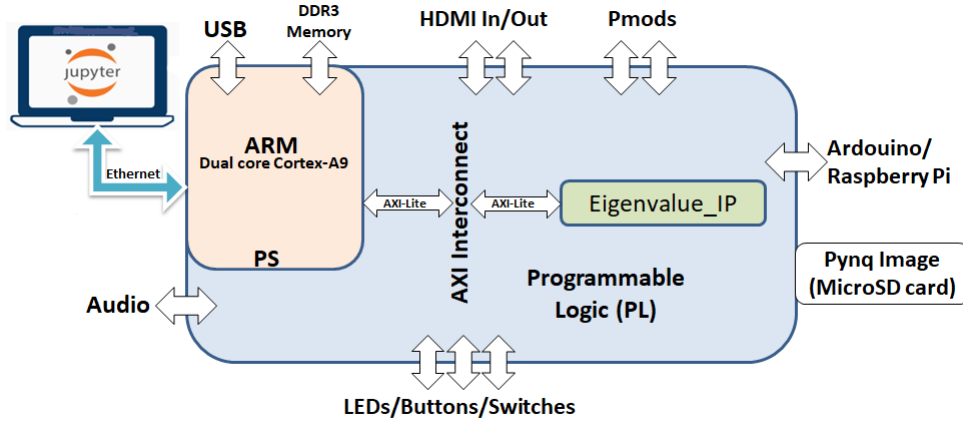


Figure 3.3: SoC interface architecture of Eigenvalue Decomposition

We designed the floating-point IP for the eigenvalue decomposition unit. The input matrix is applied to the eigenvalue decomposition hardware IP using AXI-Lite data transfer mode such as interfaces. The SoC hardware architecture of the design and overlay design of the eigenvalue decomposition of a matrix are shown in Figure 3.3 and Figure 3.4, respectively.

We calculated the software execution time (SET) of eigenvalue decomposition on ARM dual core Cortex-9 processor and used it to figure out the hardware acceleration factor (HAF) [111] as

$$\text{HAF} = \frac{\text{SOFTWARE EXECUTION TIME (SET)}}{\text{HARDWARE EXECUTION TIME (HET)}}. \quad (3.43)$$

is 0.0008s to 0.8724s for the matrix dimension from 8×8 to 100×100 respectively. It is observed that the hardware acceleration factor improved from approximately 55x to 62x for the matrix size 8×8 to 100×100 respectively.

3.5.2 Square root of the matrix using Eigen Decomposition

In the second stage, the eigenvalue decomposition is used to design the square root of matrix using the same procedure as mentioned above and realised on the Pynq-Z1 board with AXI-Lite data transfer interface. Floating-point hardware IP for the square root of matrix is designed using eigenvalue decomposition with help of Lanczos and implicit TriQR algorithms. The SoC hardware architecture of the design and overlay design of the square root of a matrix are shown in Figure 3.5 and Figure 3.6, respectively.

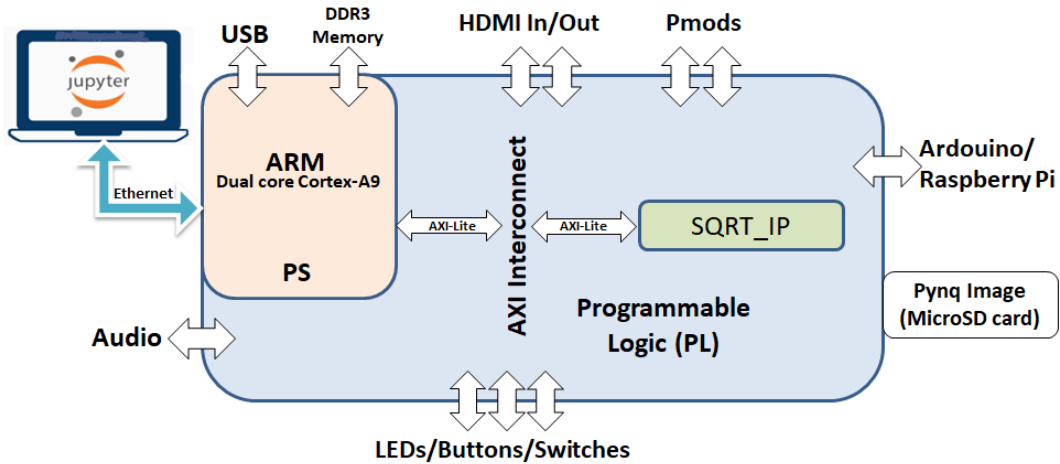


Figure 3.5: SoC interface architecture of Square root of matrix

Eigenvalue decomposition was implemented to compute the square root of a matrix on the Pynq-Z1 FPGA using a 32-bit floating-point data type. The corresponding resource utilization and hardware execution time are reported in Table 3.4 and Table 3.5. Furthermore, the hardware acceleration factor was evaluated, and it was observed that the FPGA-based implementation achieves approximately 55 $\times$ to 60 $\times$ speedup for matrix sizes ranging from 10×10 to 100×100 respectively.

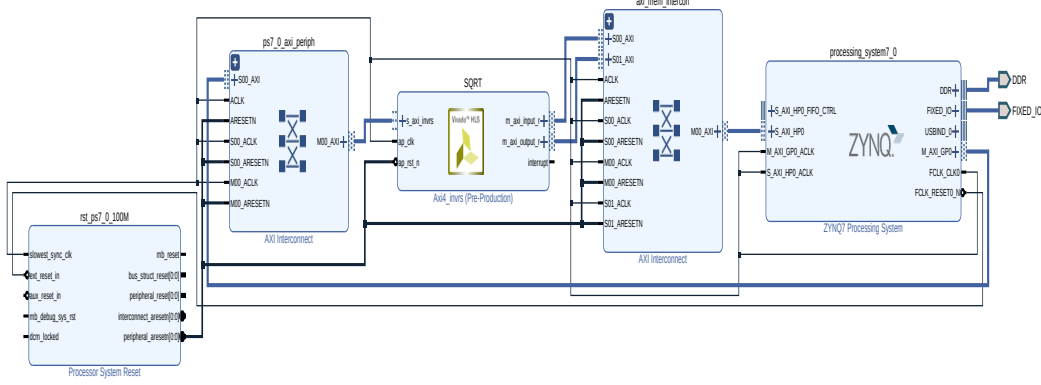


Figure 3.6: Overlay of matrix square root using AXI-Lite interface

Table 3.4: Resource utilization of square root of matrix for different input feature dimensions

Resources	Square root of matrix			
Matrix Dimension	10×10	25×25	50×50	100×100
fmax(MHz)	109.08	109.08	109.08	109.08
Slice LUTs	14985(28.17%)	15554(29.24%)	15246(28.66%)	15377(28.90%)
Slice Registers	17365(16.32%)	18017(16.93%)	18142(17.05%)	18410(17.30%)
BRAM	7.50(5.36%)	11.50(8.21%)	38(27.14%)	134(95.71%)
DSP	66(30.00%)	66(30.00%)	66(30.00%)	66(30.00%)
LUTRAM	526(3.02%)	693(3.98%)	372(2.14%)	372(2.14%)
BUFG	1(3.13%)	1(3.13%)	1(3.13%)	1(3.13%)

Table 3.5: SW and HW execution time of square root of matrix for different input feature dimensions

Matrix_size	SET (sec)	HET (sec)	HAF
10×10	0.0874	0.0016	55.43
25×25	1.123	0.0193	58.19
50×50	8.7755	0.1441	60.94
100×100	64.607	1.0709	60.32

3.6 Conclusions

The hardware implementation of the square root of a matrix is the most crucial block in scientific computing, computer graphics and machine learning applications. We know that Schur decomposition is mostly effective for symmetric positive def-

inite matrices and suitable initial guess is very important for Newton's method. So that Eigenvalue decomposition is a prominent and straightforward calculation of the matrix square root. In this eigenvalue decomposition uses lanczos and implicit TriQR algorithm for calculating the eigenvalue and eigenvector. Firstly, we implemented eigenvalue decomposition on Pynq-Z1 FPGA and analyzed HAF for different matrix sizes. After that we implemented the square root of the matrix with help of eigenvalue decomposition on the FPGA and reported software and hardware execution time for that reported HAF. In Future, we will implement fixed point IP for eigenvalue decomposition and square root of matrix to improve computational efficiency and resource utilization.

CHAPTER 4

HARDWARE ACCELERATOR FOR MATRIX INVERSION

4.1 Chapter Overview

Matrix inversion is a computationally expensive operation in many scientific applications. Performing matrix inversion of rank deficient large order matrices is still a challenge due to its computational overhead. This chapter presents the hardware implementation of matrix inversion with (i) Singular value decomposition (SVD) based on Lanczos and implicit triQR algorithms, (ii) QR method that uses modified Gram-Schmidt (MGS) algorithm, and (iii) inbuilt linear algebra package QR inverse using Xilinx Vivado high level synthesis platform. All the three algorithms are implemented on Pynq-Z1 FPGA using System on Chip (SoC) approach. The resource utilization along with accuracy, hardware execution time with and without loop optimization are reported for the aforementioned matrix inversion techniques of different matrix sizes. We achieved the hardware acceleration factor with loop optimization as 111x and 104x, respectively, for the matrix inversion of the MGS

algorithm and SVD based on Lanczos algorithm with respect to the software implementation execution time on Pynq-Z1 FPGA for the matrix size 90×90 .

4.2 Introduction

Matrix inversion is a fundamental and computational expensive unit in different scientific and engineering applications such as image recognition, least-square problem solving, regression, and multiple-input multiple-output (MIMO) wireless communication [112, 113].

The inverse of a matrix A can be represented as A^{-1} , multiplied with A , which gives an Identity matrix (I).

$$A^{-1} \times A = A \times A^{-1} = I \quad (4.1)$$

I is a square identity matrix with all diagonal elements containing value one and the remaining off-diagonal elements having value zero.

In general, the conditions for the existence of inverse are:

1. The matrix must be a square matrix.
2. The matrix must not have zero determinant value ($\det(A) \neq 0$).

Suppose matrix A is having a size 2×2 :

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad (4.2)$$

A^{-1} can be evaluated as

$$A^{-1} = \frac{\text{adj}(A)}{\det(A)} = \frac{[\text{Co-factor of matrix}(A)]^T}{\det(A)} \quad (4.3)$$

Where, $\det(A) = a_{11} \times a_{22} - a_{12} \times a_{21}$

The Co-factor of matrix A can be evaluated as $C_{pq} = (-1)^{p+q} \times \det(M_{pq})$, where $\det(M_{pq})$ is the minor of a_{pq} .

Suppose a matrix has a lower order; then, we can easily find the inverse of a matrix using an analytical technique. If the order of the matrix increases, it isn't easy to evaluate the inverse of the matrix. Matrix decomposition techniques are used to evaluate the inverse of a matrix to mitigate the above issue.

There exist different techniques such as determinant method and decomposition methods for matrix inversion depending on the matrix characteristics. For a non-singular matrix the inverse can be computed easily using Moore-Penrose method i.e., $A^\dagger = (A^T A)^{-1} A^T$ [114]. However, it numerical instability for Ill-conditioned matrices. Moreover the algorithmic complexity of inverse algorithms increases with the increase in matrix size. Hence, several decomposition algorithms such as lower-upper (LU) decomposition, Cholesky decomposition, modified Cholesky decomposition, QR decomposition, and singular value decomposition (SVD) are proposed to compute the matrix inverse [115].

From these algorithms, LU decomposition can be applied only when the matrix is non-singular square [116], however, it has stability issue. Cholesky decomposition technique can be applied only when the matrix is symmetric, positive definite and non-singular square matrix, whereas modified Cholesky can be applied for a negative semi-definite matrix [117]. QR decomposition is accurate and stable compared to LU decomposition and Cholesky decomposition [117]. For a non-singular matrix, QR decomposition can be performed using classical Gram-Schmidt (CGS), Household transformation, or Givens Rotation (GR) techniques [118]. The Givens Rotation method is suitable when the matrix is sparse [119]. The lack of parallelization in Household transformation is the main bottleneck for its embedded hardware implementation [120].

Thus, classical Gram-Schmidt technique is being applied for non-sparse matrix. Although CGS is simpler to implement but it is less stable [120]. Hence, modified Gram-Schmidt is an alternate stable technique being used to compute the inverse of a

non-sparse and non-singular matrix. The matrix inversion using QR decomposition is applicable only for a full-rank matrix, it fails when the input matrix is a rank-deficient i.e. singular. In the current work, matrix inversion is performed using QR decomposition based on modified Gram-Schmidt algorithm and SVD based on lanczos and with triQR algorithm.

4.3 Matrix inversion using QR decomposition algorithm

4.3.1 QR Decomposition

The matrix A is decomposed to matrices Q and R as

$$A = QR \quad (4.4)$$

where, Q and R represents the orthogonal matrix and upper triangular matrix respectively.

$$Q = \begin{bmatrix} q_{00} & q_{01} & q_{02} \\ q_{10} & q_{11} & q_{12} \\ q_{20} & q_{21} & q_{22} \end{bmatrix}, R = \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ 0 & r_{11} & r_{12} \\ 0 & 0 & r_{22} \end{bmatrix}$$

If the matrix is non-singular, then the Q-R factorization is unique.

4.3.2 QR Decomposition using modified Gram-Schmidt Algorithm

From the Equation 4.4, The matrix A can be decomposed as QR with help of modified Gram-Schmidt algorithm. In the MGS algorithm, the diagonal elements of the matrix R are the Euclidean norm of the columns of the input matrix columns. Columns of Q matrix are calculated by the division of the intermediate matrix columns and diagonal elements of the matrix R . Finally, non-diagonal elements of the matrix R are calculated by projecting the columns of the matrix Q onto the intermediate matrix columns. The detailed algorithmic steps for decomposing a matrix A into Q and R is given in Algorithm 11.

Algorithm 11 Calculate $Q, R = qrMGS(A)$

Require: Input $= A \in \mathbb{R}^{M \times N}$

Ensure: Output $= Q, R = qrMGS(A)$

```

1: for  $g = 1 : N$  do
2:    $R(g, g) = \|A(1 : M, g)\|_2$ 
3:    $Q(1 : M, g) = A(1 : M, g) / R(g, g)$ 
4:   for  $t = g + 1 : N$  do
5:      $R(g, t) = Q(1 : M, g)^T * A(1 : M, t)$ 
6:      $A(1 : M, t) = A(1 : M, t) - Q(1 : M, g) * R(g, t)$ 
7:   end for
8: end for

```

4.3.3 Matrix inversion using MGS based QR decomposition algorithm

After QR decomposition, the inverse of matrix A from equation (4.4) is given as

$$A^{-1} = (QR)^{-1} = R^{-1} \cdot Q^{-1} \quad (4.5)$$

Algorithm 12 Calculate $R^{-1} = inv(R)$

Input $= R \in \mathbb{R}^{N \times N}$

Output $= R^{-1} = inv(R)$

```

1: for  $(g = 1 : N)$  do
2:   for  $(p = 1 : g - 1)$  do
3:      $R^{-1}(p, g) = R^{-1}(p, (1 : g - 1)) * R((1 : g - 1), g)$ 
4:   end for
5:    $R^{-1}((1 : g - 1), g) = R((1 : g - 1), t) / R(g, g)$ 
6:    $R^{-1}(g, g) = 1 / R(g, g)$ 
7: end for

```

Since Q satisfies the orthogonality property $Q^{-1} = Q^T$. Hence,

$$A^{-1} = R^{-1} \cdot Q^T \quad (4.6)$$

We know that R is the upper triangular matrix, and its diagonal elements have non-zeros. In such cases, the inverse of R can be evaluated using algorithm 12.

4.3.4 Simulation results for MGS-based matrix inversion

In order to evaluate the computation time, PSNR and mean square error performance of the matrix inversion using MGS based QR algorithm, we did the following simulation in MATLAB environment for different matrices dimensions such as 32×32 , 256×256 , 512×512 , 1024×1024 and 2048×2048 .

We considered Householder transformation, classical Gram-Schmidt, and modified Gram-Schmidt techniques to perform the inverse using QR decomposition. The computation time for inversion, mean square error (MSE) between MATLAB *pinv* function, and the QR decomposition-based inverse are evaluated along with the peak signal to noise ratio (PSNR) as shown in Table 4.1 and Figures 4.1-4.3.

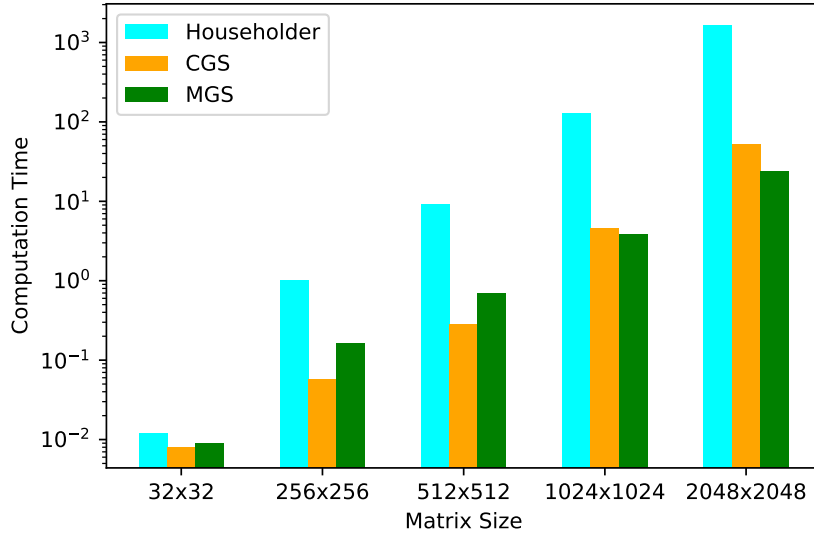


Figure 4.1: Computation time(sec) vs Matrix size of Householder, CGS and MGS

From the Figure 4.1, it was observed that, for lower order matrix, CGS based inverse algorithm took less computation time compared to Householder and MGS based inverse method, whereas for higher-order matrices, MGS based inverse took less computation time compared to the CGS and Householder based inverse.

Matrix Random		pinv (Matlab)	QR (Householder)	QR (CGS)	QR (MGS)	MyQR (LAPACK)
2x2	Elapse time(sec)	0.0045	0.0032	0.0051	0.0045	0.0073
	MSE	.	$2.3e^{-31}$	$3.4e^{-31}$	$3.4e^{-31}$	$3.9e^{-31}$
	PSNR	.	306	304.58	304.586	298
32x32	Elapse time(sec)	0.0053	0.0121	0.0081	0.0087	0.0168
	MSE	.	$4.4e^{-30}$	$4.2e^{-26}$	$7.1e^{-28}$	$6.8e^{-27}$
	PSNR	.	293	253.68	271	262
100x100	Elapse time(sec)	0.0191	0.1892	0.0156	0.035	0.053
	MSE	.	$1.2e^{-29}$	$2.6e^{-26}$	$7.2e^{-29}$	$6.7e^{-28}$
	PSNR	.	291	255.68	281.38	272.12
256x256	Elapse time(sec)	0.0317	1.019	0.058	0.1608	0.2193
	MSE	.	$3.1e^{-27}$	$6.4e^{-23}$	$3.9e^{-27}$	$2.6e^{-26}$
	PSNR	.	269	221.9	263	252
512x512	Elapse time(sec)	0.1228	9.196	0.2879	0.7075	1.416
	MSE	.	$5.2e^{-28}$	$5.2e^{-22}$	$1.5e^{-27}$	$5.6e^{-27}$
	PSNR	.	285	212.83	269.30	256
1024x1024	Elapse time(sec)	0.777	128.43	4.573	3.89	17.424
	MSE	.	$3.8e^{-25}$	$1.8e^{-18}$	$2.1e^{-24}$	$8.3e^{-24}$
	PSNR	.	248	177	236	226
2048x2018	Elapse time(sec)	7.802	1664.8	52.74	23.99	139.86
	MSE	.	$1.2e^{-28}$	$3.2e^{-24}$	$1.8e^{-27}$	$8.3e^{-26}$
	PSNR	.	279	234	267	253

Table 4.1: MATLAB simulation results for various matrix inversion decomposition methods

From the Figure 4.2, it was observed that for both lower and higher-order matrices, Householder transformation gives less MSE compared to the CGS and MGS-based inverse. The accuracy obtained by MGS and Householder inverse is comparable but better than CGS for both lower and higher-order matrices.

From the Figure 4.3, we can infer that, as far as the PSNR is concerned, for both lower and higher-order matrices, Householder transformation performs slightly better than the MGS inverse, whereas the MGS inverse performs better than the CGS inverse. Since Householder is not suitable for parallelization, the MGS-based

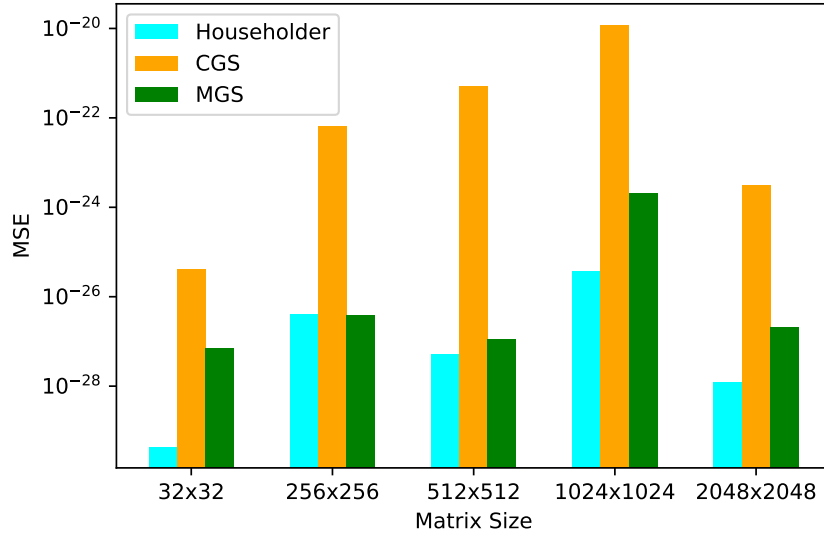


Figure 4.2: MSE vs Matrix size of Householder, CGS and MGS

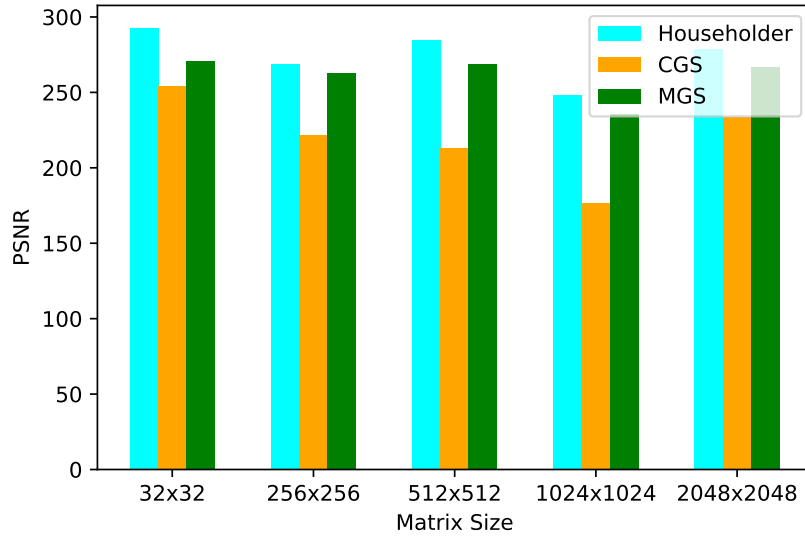


Figure 4.3: PSNR vs Matrix size of Householder, CGS and MGS

inverse algorithm is the good choice for computing the matrix inverse among the CGS and Householder-based inverse techniques.

MGS based algorithm is concluded as the good choice for computing the matrix inverse among the CGS and Householder based inverse technique.

In Figure 4.4, we applied random input matrix sizes up to 1000×1000 for the MGS based matrix inverse and reported the MSE between the inverse obtained

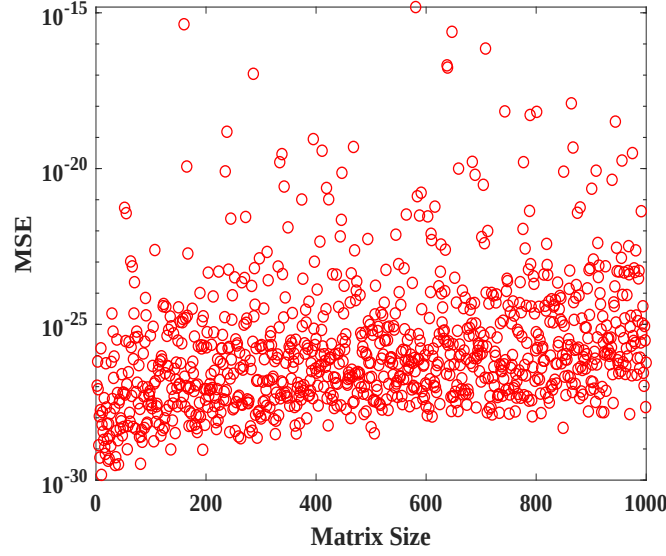


Figure 4.4: MSE of matrix inversion using QR decomposition for different matrix sizes

using QR decomposition and reference algorithm ($pinv$). We have considered Matlab inbuilt $pinv$ as the reference algorithm. Random input matrices of different sizes are considered input matrices that need to be inverted. Mean Square Error (MSE) for a matrix of size t is obtained as

$$\text{MSE}(t) = \frac{1}{t \cdot t} \sum_{i=0}^{t-1} \sum_{j=0}^{t-1} \left(A_{i,j}^{pinv} - A_{i,j}^{QRinv} \right)^2 \quad (4.7)$$

where A^{pinv} is the inverse of matrix A using Matlab inbuilt $pinv$ and A^{QRinv} is the inverse of matrix A using MGS based QR decomposition algorithm.

From Figure 4.4, it is observed that the mean square error for QR inverse based on MGS algorithm is within an error limit of $\leq 10^{-15}$.

4.3.5 Hardware results

The SoC hardware implementation of matrix inversion on the Pynq-Z1 board has two stages. In the first stage, the hardware of QR decomposition is developed using HLS technique. We used pipeline directive i.e., pragma, to improve the hardware latency. In the second stage, the QR decomposition core is used to design the

matrix inversion IP on the Pynq-Z1 board.

4.3.5.1 QR decomposition

We designed a QR decomposition using the MGS algorithm, a 32-bit floating-point data type is implemented on the Pynq-Z1 FPGA board that has a processor supporting both AXI-Lite and DMA interfacing methods. The SoC hardware architecture of the design is shown in Fig. 4.5, Fig. 4.6.

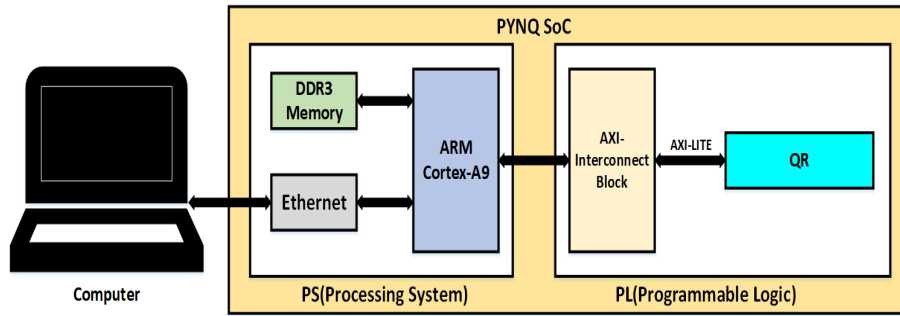


Figure 4.5: SoC interface architecture of the QR decomposition using AXI-Lite interface

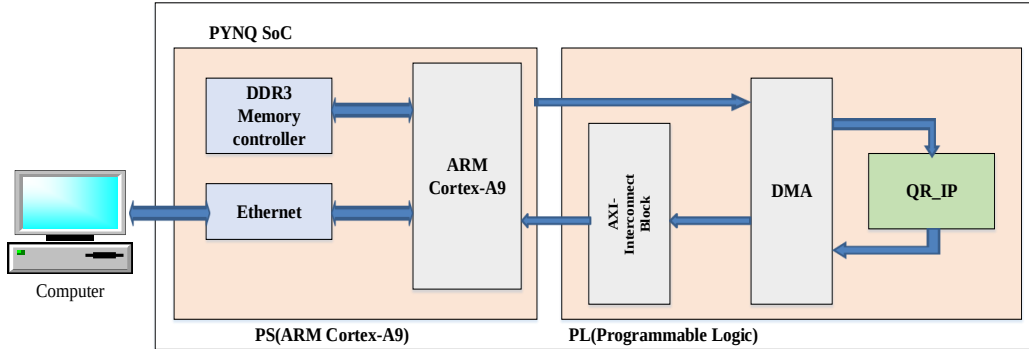


Figure 4.6: SoC interface architecture of the QR decomposition using DMA interface

We created the overlay for QR decomposition based on MGS algorithm using XILINX VIVADO 2019.2 as shown in Figure 4.7 and Figure 4.8.

After the realization of QR decomposition, we extracted the resource utilization and latency for AXI-Lite and DMA interface as shown in Table 4.2.

For a fair comparison of resources, we calculated a figure of merit (FoM) pa-

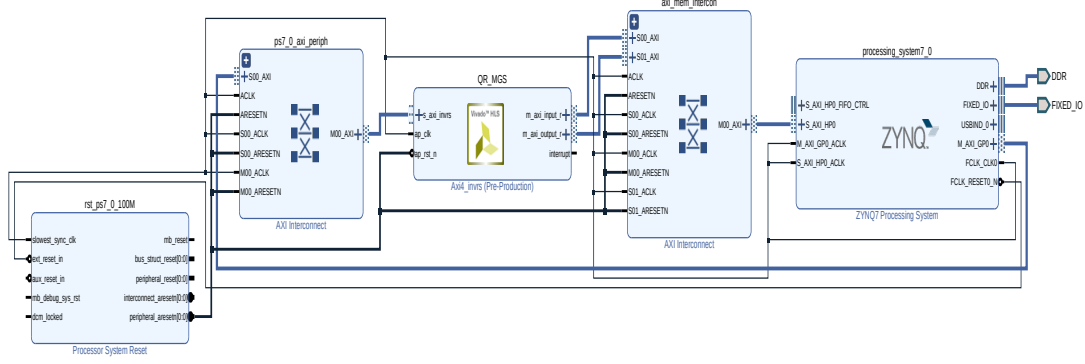


Figure 4.7: Overlay of QR-based MGS algorithm using AXI-Lite interface

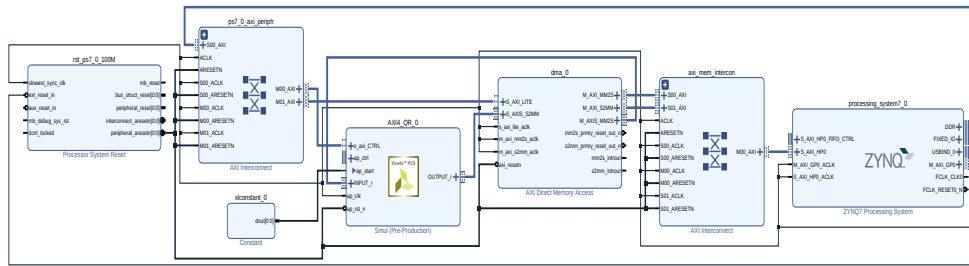


Figure 4.8: Overlay of QR-based MGS algorithm using DMA interface

Table 4.2: Resources and latency comparison of floating point QR decomposition of matrix size 100×100

Ref	Algo.	Without Optimization	Unrolling	Pipelining	Data transfer Interface	Resource Utilization	FoM	Latency (msec)
[121]	CGS	✓	-	-	AXI-BRAM	BRAM=2, DSP=6 FF=4391, LUT=6093	84000	209
[121]	CGS	-	✓	-	AXI-BRAM	BRAM=50, DSP=19 FF=5632, LUT=7914	109472	141
Prop. work	MGS	✓	-	-	AXI4-Lite	BRAM=166, DSP=7 FF=4406, LUT=6253	88040	232
Prop. work	MGS	-	-	✓	AXI4-Lite	BRAM=166, DSP=7 FF=4484, LUT=6349	89432	131
Prop. work	MGS	✓	-	-	DMA	BRAM=96, DSP=7 FF=2494, LUT=4164	54912	231
Prop. work	MGS	-	-	✓	DMA	BRAM=96, DSP=7 FF=2605, LUT=4207	56144	131

parameter as

$$FoM = [(LUT + FF) \times 8] + [(BRAM + DSP) \times 16] \quad (4.8)$$

where LUT, FF, BRAM, and DSP represent the number of Lookup Tables, Flip Flops, Block RAMs, and DSP slices used by the SoC.

The latency and FoM of the proposed method are compared with the classical Gram-Schmidt algorithm [121]. Our implementation QR decomposition based on MGS algorithm with DMA interface has less FoM and latency than classical Gram-Schmidt implementation on Zedboard [121]. The effect of HLS optimization such as loop unrolling and pipeline is clearly observed in latency and resources.

4.3.5.2 Matrix inversion using modified Gram Schmidt

The matrix inversion using MGS algorithm is implemented on the Pynq-Z1 FPGA, using AXI-Lite interfacing, between Processing System (PS) and Programmable Logic (PL) of Pynq FPGA. The SoC interface architecture and Overlay design of matrix inversion using MGS algorithm as shown in Figure 4.9 and Figure 4.10. We report the resource utilization, power consumption, hardware execution time (HET) and mean square error for different matrix size in Table 4.3.

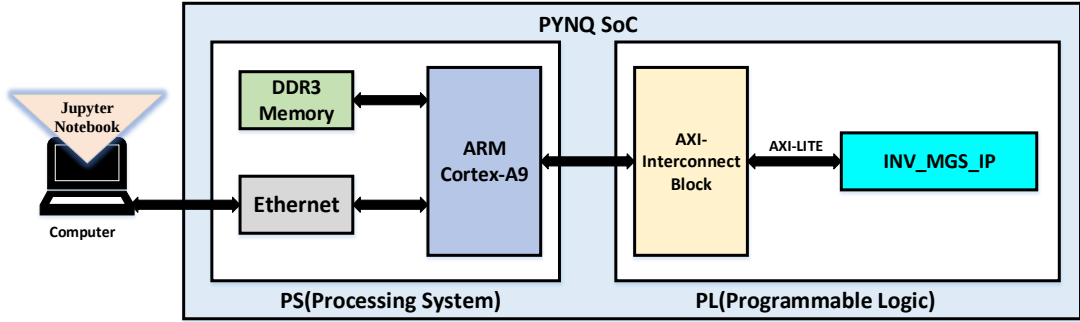


Figure 4.9: SoC interface architecture of the Matrix inversion IP

From the Table 4.3, it is observed that for lower-order matrix, MGS method gives less MSE, but for the higher-order matrix, the accuracy decreases. Similarly the HET increases with increase in matrix dimension.

Pipelining is an optimization process that decreases the execution time of a function or loop. Figure 4.11 analyses the instruction execution without (WO) and with loop pipelining optimization technique. Figure 4.11 illustrates that, if a task

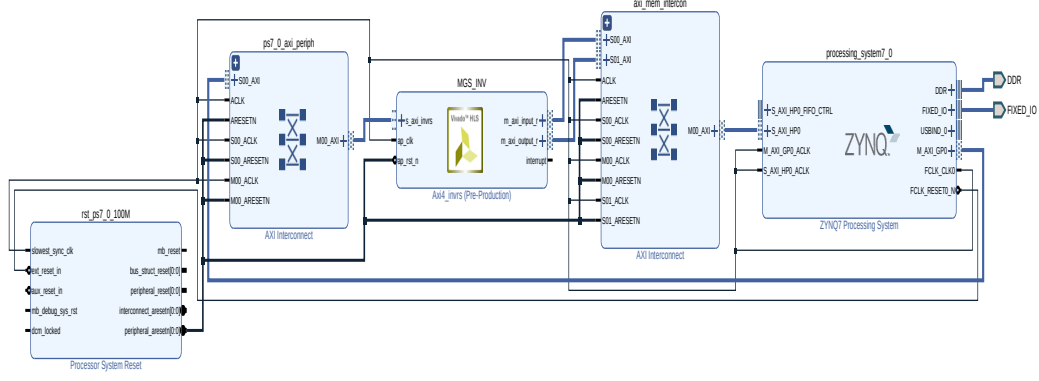


Figure 4.10: Overlay of matrix inverse using QR-based MGS algorithm

Table 4.3: Performance analysis of MGS based matrix inversion without inner loop pipelining optimization for various matrix sizes

Matrix size	16×16	25×25	32×32	50×50	90×90
fmax(MHz)	114.29	114.29	114.29	114.29	103.80
Slice LUTs	4387(8.25%)	4631(8.70%)	4460(8.38%)	4620(8.68%)	4703(8.84%)
Slice Registers	5195(4.88%)	5527(5.19%)	5267(4.95%)	5734(5.33%)	5771(5.42%)
BRAM	5(3.57%)	8(5.71%)	9(6.43%)	33(23.57%)	65(46.43%)
DSP	5(2.27%)	6(2.73%)	5(2.27%)	5(2.27%)	5(2.27%)
LUTRAM	197(1.13%)	205(1.18%)	198(1.14%)	206(1.18%)	207(1.19%)
Static Power(Watts)	0.137	0.138	0.138	0.141	0.147
Dynamic Power(Watts)	1.367	1.374	1.372	1.432	1.506
IP Power(Watts)	0.097	0.103	0.101	0.161	0.235
SoC Power(Watts)	1.504	1.512	1.51	1.573	1.653
HET(sec)	0.0013	0.0045	0.0092	0.0342	0.1965
MSE	$2.59e^{-13}$	$3.17e^{-13}$	$1.61e^{-11}$	$1.97e^{-11}$	$7.41e^{-11}$

requires eight clock cycles to execute, with enabling loop pipelining technique, the same task can complete the execution in four clock cycles. In this current work, we used loop pipelining optimization method to speed up the hardware execution time.

We applied inner loop pipelining optimization to the MGS based matrix inversion algorithm. The resources, MSE and HET are tabulated in Table 4.4. It shows that with inner loop pipelining optimization technique, there is substantial improvement in HET at the cost of negligible change in resources.

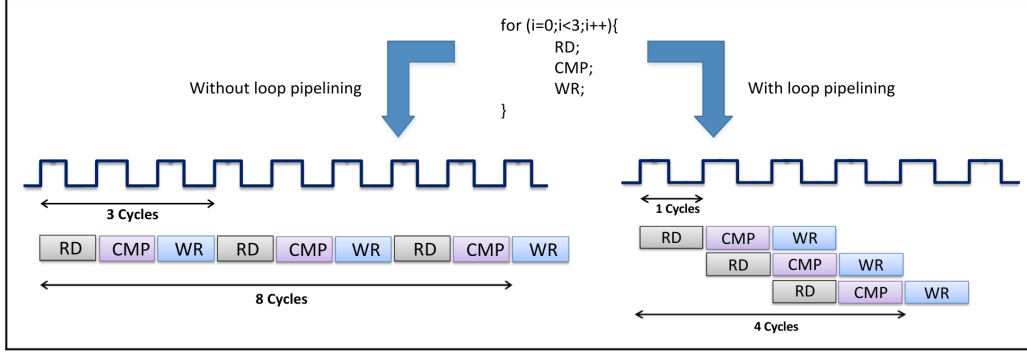


Figure 4.11: Illustration of loop pipelining optimization technique [122]

Table 4.4: Performance analysis of MGS-based matrix inversion with inner loop pipelining optimization for various matrix sizes

Matrix Size	16×16	25×25	32×32	50×50	90×90
fmax(MHz)	109.92	103.80	109.92	103.80	103.80
Slice LUTs	4684(8.80%)	4928(9.26%)	4778(8.98%)	4962(9.33%)	5124(9.63%)
Slice Registers	5358(5.04%)	5607(5.27%)	5450(5.12%)	5734(5.39%)	5860(5.51%)
BRAM	5(3.57%)	8(5.71%)	9(6.43%)	33(23.57%)	65(46.43%)
DSP	5(2.27%)	10(4.55%)	5(2.27%)	9(4.09%)	9(4.09%)
LUTRAM	210(1.21%)	223(1.28%)	213(1.22%)	228(1.31%)	230(1.32%)
Static Power(Watts)	0.137	0.138	0.137	0.141	0.146
Dynamic Power(Watts)	1.361	1.370	1.362	1.425	1.496
IP Power(Watts)	0.091	0.099	0.092	0.155	0.225
SoC Power(Watts)	1.499	1.508	1.5	1.567	1.642
HET(sec)	0.0008	0.0024	0.0053	0.0181	0.1044
MSE	$2.59e^{-13}$	$3.17e^{-13}$	$1.61e^{-11}$	$1.97e^{-11}$	$7.41e^{-11}$

For different matrix size, the HAF with and without inner loop pipelining optimization is reported in Table 4.5. From the Table 4.5, we observe that, decrease in HET due to inner loop pipelining increases the HAF of the algorithm.

Further, we implemented the matrix inversion using linear algebra package (LAP) QR decomposition algorithm available in Vivado HLS tool as shown in Figure 4.12 and report the performance for various matrix sizes in the Table 4.6.

From the Table 4.6, it is observed that matrix inversion using LAP QR decomposition gives comparable accuracy as MGS based method, however it consumes

Table 4.6: Performance analysis of LAP QR based matrix inversion with various matrix sizes

Matrix Size	16 × 16	25 × 25	32 × 32	50 × 50	90 × 90
fmax(MHz)	114.29	103.80	114.29	103.80	103.80
Slice LUTs	9934(18.67%)	10153(19.08%)	9969(18.74%)	10186(19.15%)	10419(19.58%)
Slice Registers	11278(10.60%)	11588(10.99%)	11324(10.64%)	11625(10.93%)	12270(11.53%)
BRAM	9(6.43%)	15(10.36%)	17(12.14%)	52(36.79%)	106(75.36%)
DSP	34(15.45%)	39(17.73%)	34(15.45%)	39(17.73%)	39(17.73%)
LUTRAM	776(4.46%)	830(4.77%)	732(4.21%)	844(4.85%)	674(3.87%)
Static Power(Watts)	0.140	0.141	0.141	0.146	0.154
Dynamic Power(Watts)	1.451	1.479	1.454	1.527	1.625
IP Power(Watts)	0.183	0.21	0.186	0.259	0.357
SoC Power(Watts)	1.591	1.62	1.595	1.673	1.78
HET(sec)	0.0026	0.0028	0.0034	0.0072	0.0301
MSE	$1.28e^{-13}$	$1.9e^{-13}$	$4.03e^{-12}$	$5.60e^{-12}$	$2.57e^{-12}$

The inverse of a square non-singular matrix A is computed using SVD as

$$A_{N \times N}^{-1} = V_{N \times N} \times S_{N \times N}^{-1} \times U_{N \times N}^T, \quad (4.10)$$

where S^{-1} is the inverse of the matrix S . Similarly, the pseudo-inverse of a non-singular rectangular matrix $A \in \mathbb{R}^{M \times N}$ can be computed using

$$A_{N \times M}^{\dagger} = V_{N \times N} \times S_{N \times M}^{\dagger} \times U_{M \times M}^T, \quad (4.11)$$

where $S^{\dagger}$ is the pseudo-inverse of S . The detailed steps for computing pseudo-inverse of the matrix A using SVD based on Lanczos and implicit triQR algorithm is given below.

Initially the input matrix is transformed to a symmetric matrix $B \in \mathbb{R}^{N \times N}$ as

$$B_{N \times N} = [A^T \times A]_{N \times N}. \quad (4.12)$$

The Lanczos algorithm is applied on B to get the orthogonal matrix Q and tridiagonal matrix T such that

$$Q_{N \times N}^T \times B_{N \times N} \times Q_{N \times N} = T_{N \times N}. \quad (4.13)$$

After obtaining the T and Q matrices, implicit triQR algorithm is applied on T and Q to get the eigenvalues or diagonal matrix $D \in \mathbb{R}^{N \times N}$ and eigen vector matrix $V \in \mathbb{R}^{N \times N}$ respectively.

The matrix B can be represented as In terms of the D and V as shown below

$$B_{N \times N} = V_{N \times N} \times D_{N \times N} \times V_{N \times N}^{-1}. \quad (4.14)$$

From Equation (4.14), the singular values of matrix A is calculated by applying square root to the diagonal elements of D . The rank of the matrix A can be used to obtain $S \in \mathbb{R}^{K \times K}$ by taking only singular values greater than the SVD tolerance limit. Since S is a diagonal matrix, and inverse of S i.e $S^\dagger \in \mathbb{R}^{K \times K}$ can be obtained by reciprocating the diagonal elements of S . The eigen vectors $U \in \mathbb{R}^{M \times K}$ is calculated by considering $k = K$ (number of eigenvalues) columns of $V_{N \times N}$.

$$U_{M \times K} = A_{M \times N} \times V_{N \times K} \times S_{K \times K}^\dagger. \quad (4.15)$$

From Equation (4.11) the pseudo-inverse $A^\dagger$ can be rewritten as

$$A_{N \times M}^\dagger = V_{N \times K} \times S_{K \times K}^\dagger \times U_{K \times M}^T. \quad (4.16)$$

Substituting Equation (4.15) in Equation (4.16) gives,

$$A_{N \times M}^\dagger = V_{N \times K} \times S_{K \times K}^\dagger \times (S_{K \times K}^\dagger)^T \times V_{N \times K}^T \times A_{M \times N}^T. \quad (4.17)$$

Since the singular values are square root of eigenvalues, Equation (4.17) can be rewritten as

$$A_{N \times M}^\dagger = V_{N \times K} \times D_{K \times K} \times V_{N \times K}^T \times A_{M \times N}^T. \quad (4.18)$$

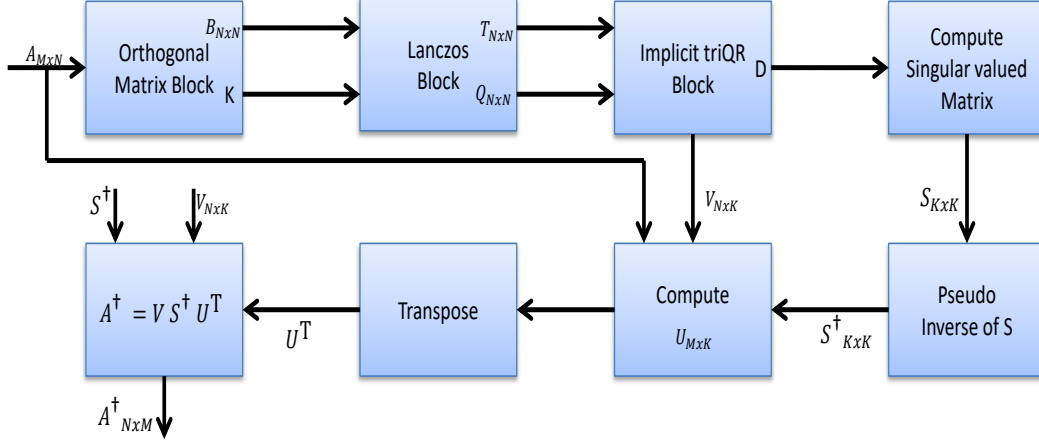


Figure 4.13: The block-level representation of matrix inverse computation using the Lanczos algorithm

The block-level computation of the matrix inverse using the Lanczos method is shown in Fig. 4.13.

4.4.1 Simulation results for SVD based matrix inverse

We applied SVD based on Lanczos and implicit triQR algorithm to compute pseudo-inverse of both full rank and singular matrices (rank deficient) of different sizes. The MSE is obtained same as Equation (4.7) except $Q_{inv}(A)$ is replaced with $S_{inv}(A)$.

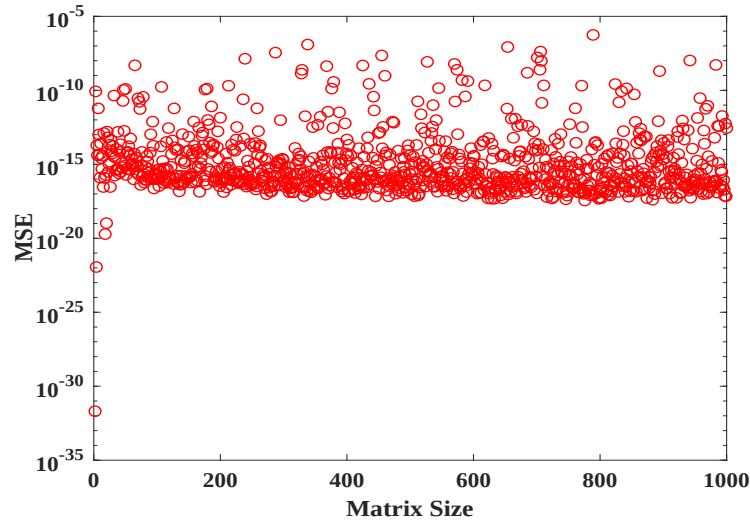


Figure 4.14: MSE of matrix inversion using SVD for full rank matrices

where, $Sinv(A)$ is the inverse of matrix A using SVD based on Lanczos algorithm. MSE of matrix inversion using SVD for full rank and rank deficient matrix are plotted in Figure 4.14 and Figure 4.15 respectively.

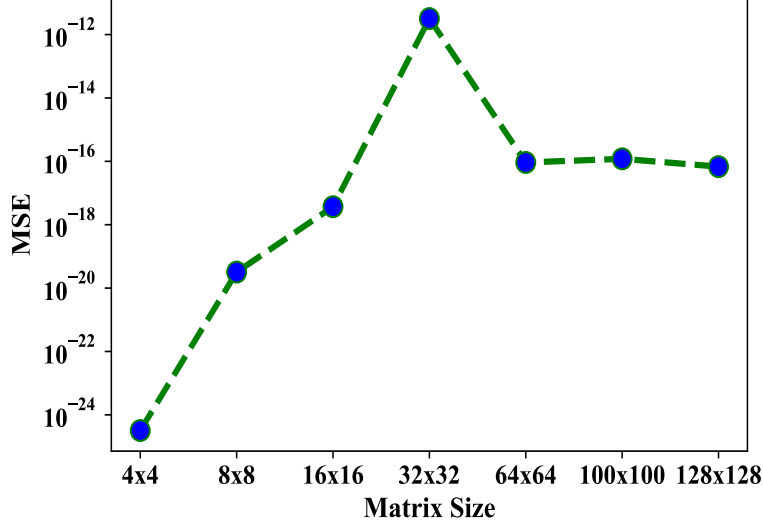


Figure 4.15: MSE of matrix inversion using SVD for singular matrices

From Figure 4.14 and Figure 4.15, it is observed that the mean square error for matrix inversion is below 10^{-07} of the inverse using the Lanczos algorithm.

4.4.2 Hardware results

A 32-bit floating-point IP is created for the matrix inversion using SVD decomposition and implemented on the Pynq-Z1 FPGA with AXI-Lite interfacing. The SoC architecture and overlay design of matrix inverse using SVD as shown in Figure 4.16 and Figure 4.17 respectively.

We synthesized the design with and with-out inner loop pipelining optimization technique. The hardware resources, execution time and mean square error for the inversion IP with and with-out inner loop pipelining optimization for different matrix dimensions are reported in Table 4.7 and Table 4.8.

From the Table 4.7, it is observed that for lower-order matrix, Lanczos based matrix inverse provides less MSE, but for the higher-order matrix, the accuracy decreases. Similarly the HET increases with increase in matrix dimension.

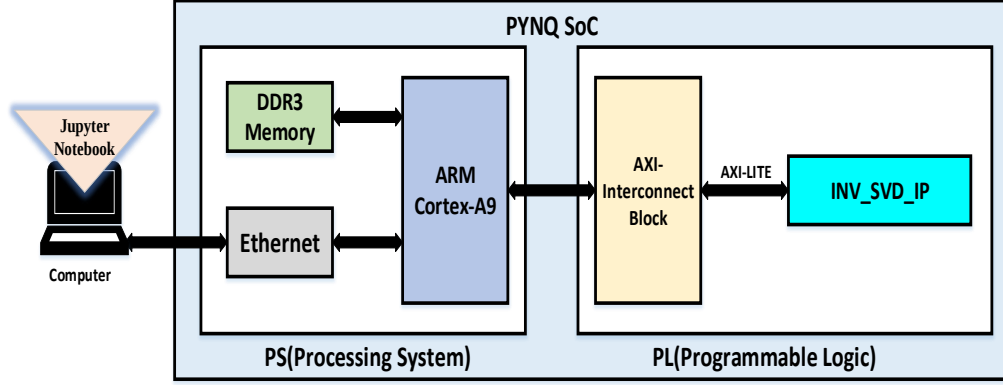


Figure 4.16: SoC design of matrix inverse using Lanczos SVD

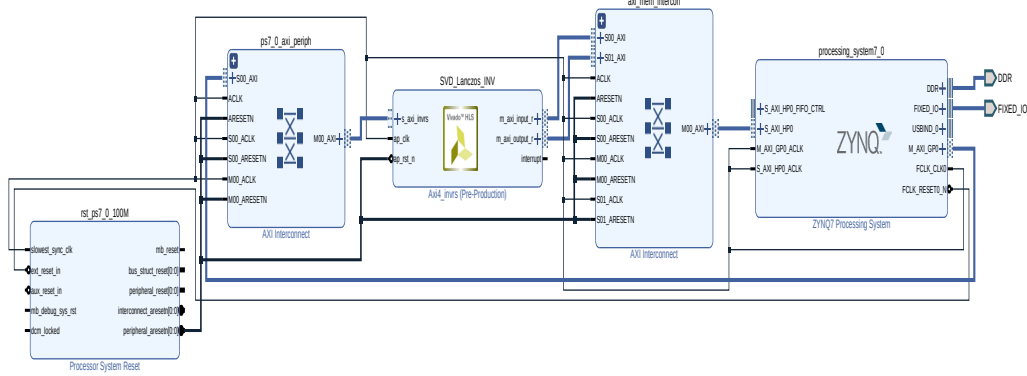


Figure 4.17: Overlay of matrix inverse using Lanczos SVD

We evaluated the hardware acceleration factor (HAF) of matrix inversion IP using SVD based on Lanczos and implicit triQR with and without inner loop pipelining optimization. The hardware acceleration factor for matrices of different size are tabulated in the Table 4.9.

From Table 4.9, it is observed that inner loop pipelining optimization gives higher HAF compared to the design without inner loop pipelining optimization, irrespective of the order of the matrix.

The FoM and HET for all three implementation methods for different matrix sizes are reported in Table 4.10. From the Table 4.10, it is observed that for lower order matrix, MGS based QR inverse took less hardware execution time and less FoM compared to linear algebra package QR inverse, whereas for higher order matrix linear algebra package QR inverse took less hardware execution time. FoM of linear

Table 4.7: Performance analysis of SVD based matrix inversion for different matrix sizes without inner loop pipelining optimization

Matrix Size	16×16	25×25	32×32	50×50	90×90
fmax(MHz)	108.58	108.75	114.29	114.29	114.29
Slice LUTs	14521(27.30%)	15326(28.81%)	14628(27.50%)	14941(28.08%)	15081(28.35%)
Slice Registers	16500(15.51%)	17402(16.36%)	16587(15.59%)	17607(16.55%)	17826(16.75%)
BRAM	9(6.43%)	14(9.64%)	17(12.14%)	47(33.21%)	87(61.79%)
DSP	55(25%)	55(25%)	55(25%)	55(25%)	55(25%)
LUTRAM	555(3.19%)	757(4.35%)	438(2.52%)	370(2.13%)	372(2.14%)
Static Power(Watts)	0.144	0.145	0.145	0.151	0.159
Dynamic Power(Watts)	1.622	1.638	1.632	1.709	1.815
IP Power(Watts)	0.351	0.367	0.362	0.439	0.544
SoC Power(Watts)	1.766	1.783	1.777	1.86	1.973
HET(sec)	0.0061	0.0231	0.0462	0.1671	0.9562
MSE	$1.71e^{-11}$	$2.07e^{-11}$	$9.69e^{-08}$	$1.91e^{-07}$	$6.97e^{-08}$

Table 4.8: Performance analysis of SVD based matrix inversion for various matrix sizes with inner loop pipelining optimization

Matrix size	16×16	25×25	32×32	50×50	90×90
fmax(MHz)	108.75	103.80	114.29	103.80	103.80
Slice LUTs	15699(29.51%)	16559(31.13%)	15828(29.75%)	16179(30.41%)	16593(31.19%)
Slice Registers	17252(16.21%)	17940(16.86%)	17395(16.35%)	18069(16.98%)	18374(17.27%)
BRAM	10(7.14%)	15(10.56%)	18(12.86%)	47(33.57%)	87(62.14%)
DSP	55(25%)	60(27.27%)	55(25%)	58(26.36%)	58(26.36%)
LUTRAM	540(3.10%)	708(4.07%)	443(2.55%)	386(2.22%)	389(2.24%)
Static Power(Watts)	0.144	0.145	0.145	0.150	0.157
Dynamic Power(Watts)	1.609	1.626	1.628	1.683	1.753
IP Power(Watts)	0.339	0.356	0.357	0.413	0.482
SoC Power(Watts)	1.754	1.771	1.773	1.833	1.909
HET(sec)	0.0037	0.0121	0.0243	0.0932	0.5411
MSE	$1.71e^{-11}$	$2.07e^{-11}$	$9.69e^{-08}$	$1.91e^{-07}$	$6.97e^{-08}$

algebra package QR inverse is nearly double than MGS based QR inverse technique. These two methods are applicable for non-singular matrices only. Although SVD based on Lanczos inverse took more HET and FoM compared to linear algebra package QR inverse and MGS-based QR inverse methods, it is applicable for both singular and non-singular matrices.

Table 4.9: HAF of SVD matrix inverse using without and with inner loop pipelining optimization for different matrix sizes

Matrix Size	SET	WO inner loop pipelining		With inner loop pipelining	
	(sec)	HET (sec)	HAF	HET (sec)	HAF
16×16	0.3771	0.0061	61	0.0037	102
25×25	1.3672	0.0234	58	0.0121	112
32×32	2.7232	0.0461	59	0.0243	109
50×50	9.7614	0.1679	58	0.0932	104
90×90	56.2591	0.9559	59	0.5411	104

Table 4.10: HET and FoM comparison of matrix inverse using different algorithm techniques

Matrix Size	LAP QR		QR (MGS)		SVD (Lanczos)	
	FoM	HET(sec)	FoM	HET(sec)	FoM	HET(sec)
16×16	170384	0.0026	80496	0.0008	264648	0.0037
25×25	174792	0.0028	84568	0.0024	277192	0.0121
32×32	171160	0.0034	82048	0.0053	266952	0.0243
50×50	175944	0.0072	86240	0.0181	275664	0.0932
90×90	183832	0.0301	89056	0.1044	282056	0.5411

Table 4.11 compares resource utilization, FoM, computation time, power consumption, energy delay product (EDP) and maximum absolute error (MAE) of the SVD and MGS based matrix inverse algorithms for matrix size 25×25 .

EDP is computed as

$$EDP = \frac{P}{f_{max}^2}, \quad (4.19)$$

where P and f_{max} represents total power consumed by the IP and its maximum operating frequency respectively.

The hardware accuracy i.e. maximum absolute error (MAE) between the input matrix and the reconstructed matrix is computed from the inverse matrix using

pseudo inverse property $A_r = AA^{-1}A$ as

$$MAE = |A - A_r|, \quad (4.20)$$

where A and A_r represents the input matrix and reconstructed matrix evaluated using a pseudo inverse property.

Table 4.11: Performance comparison of different matrix inversion algorithms for matrix size 25×25

Ref	Algo.	fmax (MHz)	Resource Utilization	FoM	Computation time(msec)	Power (mW)	EDP (fJ)	MAE
[123]	GR	NA	BRAM=10, DSP=32 FF=6741, LUT=7143	111744	1.05	124	NA	$2.27e^{-07}$
[123]	Gauss Jordan	NA	BRAM=3, DSP=5 FF=1801, LUT=2209	32208	3.91	123	NA	$1.56e^{-06}$
Prop. work	MGS	114	BRAM=8, DSP=10 FF=4928, LUT=5607	84568	2.4	99	0.0076	$1.79e^{-06}$
[123]	SVD	NA	BRAM=14, DSP=49 FF=9313, LUT=11340	166232	11.24	126	NA	$3.09e^{-07}$
Prop. work	Lanczos	103.8	BRAM=15, DSP=60 FF=17940, LUT=16559	277192	12.1	356	0.0331	$1.59e^{-05}$

From the Table 4.11, it is observed that Givens Rotation (GR) and MGS-based matrix inverse algorithm take less FoM, computation time, and power than the proposed SVD-based on Lanczos matrix inverse algorithm. GR and MGS-based matrix inverse algorithm works for only non-singular matrices, but SVD-based on Lanczos matrix inverse works for both singular and non-singular matrices. Gauss Jordan-based matrix inverse algorithm takes less FoM, power, and computation time compared to the proposed SVD-based inverse algorithm, but if the order of matrix increases, then Gauss Jordan-based matrix inverse becomes slow. Similarly, the previously implemented SVD-based matrix inverse algorithm [123] takes less FoM and power, but computation time is almost comparable to the proposed SVD-based on Lanczos matrix inverse algorithm. SVD based on Lanczos is a more appropriate when the input is a large sparse matrix.

4.5 Conclusions

In this work, we designed and realized accelerators for matrix inverse algorithms namely (i) QR decomposition based on modified Gram-Schmidt, (ii) SVD based on Lanczos and implicit triQR algorithm, and (iii) linear algebra package QR inverse of Xilinx Vivado HLS on SoC platform in Pynq Jupyter note book environment. After realization, the functionality, area, timing, and power are extracted and compared. We reported the hardware acceleration factor for MGS-based matrix inversion and SVD-based matrix inversion techniques. We included reorthogonalization to increase the stability of Lanczos algorithm. The proposed SVD-based matrix inverse is the most efficient decomposition technique in finding the pseudo-inverse of a rank-deficient and rectangular matrix. In future, we will develop an optimized and fixed-point matrix inversion hardware IP of the proposed matrix inversion algorithms to minimise the hardware resources.

CHAPTER 5

HARDWARE IMPLEMENTATION OF WHITENING ALGORITHM

Chapter Overview

Whitening is a data preprocessing technique used in machine learning to decorrelate features and achieve unit variance. This process helps in preventing certain features from dominating the analysis. Further, it improves the convergence of learning algorithms, and ensures more isotropic data distribution. ZCA (Zero-Phase Component Analysis) and PCA (Principal Component Analysis) are the standard methods used for whitening.

In this chapter, we implemented whitening with ZCA using a hardware and software co-design framework. We report the hardware acceleration factor and resource utilization for ZCA whitening across different input dimensions. The implementation results demonstrate that HW-SW co-design achieved hardware acceleration ranging from 8x to 18x for various input data feature dimensions between 1000 x 10 and 1000 x 20.

Additionally, we implemented a complete whitening process using PCA on an FPGA. Our proposed method shows superior resource utilization compared to existing implementations, while achieving comparable hardware execution time and frequency, particularly for an input matrix size of 16 x 30.

5.1 Hardware-Software Co-Design for Whitening using ZCA

Zero-phase component analysis (ZCA) can de-correlate the input data in the data preprocessing method. It addressed the stochastic axis-swapping issue in the PCA pre-processing method. There are three main computation blocks in ZCA evaluation. The first one is the evaluation of the covariance matrix, the second one is the eigenvalue decomposition, and finally, the evaluation of the whitened matrix. Among the three, Eigenvalue decomposition is a major computational block in ZCA. With the help of the HW-SW co-design framework, the ZCA algorithm is implemented on a Pynq FPGA in two scenarios by partitioning the ZCA algorithm. Firstly, we run only eigenvalue decomposition on the hardware side and the remaining covariance and whitened matrix run on the processing system. Second, the eigenvalue decomposition and whitened matrix are processed on the hardware IP, and the remaining covariance matrix evaluation is performed on the processing system.

5.1.1 Introduction

Signal de-correlation is an essential pre-processing tool in feature extraction and machine learning. Whitening is a pre-processing technique that transforms the original correlated signal into many de-correlating signals with identical covariances [124]. Principal component analysis (PCA) whitening is a popular and standard technique in machine learning where the signal vector is rotated by a rotation matrix and scaled by the square root of the eigenvalues of the signal covariance matrix. The stochastic axis swapping problem in principal component analysis can be addressed with the whitening of zero-phase component analysis [125]. In ZCA, the transformed

PCA whitening is further rotated by the same rotation matrix [126]. The rotation matrix is derived from the eigenvectors of the signal covariance. In literature ZCA has been used in different applications such as spectroscopic analysis [127], digit recognition [128].

ZCA is a potential tool for many embedded applications such as autonomous underwater vehicles [129], face recognition[130], etc. It has three main computations, such as evaluation of (i) covariance matrix, (ii) eigenvalue decomposition and (iii) whitened matrix. Among these three units, eigenvalue decomposition of the covariance matrix is computationally intensive. The accuracy and speed of the ZCA algorithm depend on the method of eigenvalues computation, the target hardware platform for implementation and the implementation strategies.

Throughput, latency, and energy consumption are three main constraints for designing the hardware system for any application. The leading hardware platforms for system design are central processing units, field programming gate arrays, graphical processing units, and application-specific integrated circuits[131]. Low throughput, high latency, and energy efficiency challenges are present on embedded devices with GPUs or CPUs, but several attractive features such as parallelism and high throughput are available with FPGAs and ASICs for hardware system design [132]. Designing a new hardware architecture for ASIC chips is inflexible, time-consuming, and too expensive. Hence FPGA based system design/prototyping is a popular choice. Further, the availability of an on-chip processor, on-chip bus, and flexibility for implementing an intense computational algorithm in an FPGA-based system-on-chip (SoC) is an added advantage to meet the design time requirement [133].

Algorithm implementation on the FPGA fabric requires resources like LUT, DSP 48 slices, and shift registers. However, the hardware fabric resources are limited on an FPGA. In the co-design strategy, part of the algorithm is ported into the on-chip processor of the FPGA, and the remaining is implemented on the FPGA fabric as a hardware IP [134, 135]. The processor and the hardware IP are interfaced with on-chip buses. The choice of the component that will be implemented on hardware/software depends on the profiling results of the algorithm [136]. To

maximize the advantages of the hardware capability, the algorithm component that requires maximum computation time is ported into the hardware IP.

The current work aims to explore the speed performance of the ZCA algorithm on the Pynq-Z1 platform using the hardware-software (HW-SW) co-design framework with help of High-level Synthesis tool. Further, we explored the HW-SW co-design framework for realizing the same in two different scenarios as shown in Figure 5.1 and Figure 5.2. In scenario (i), we implemented only eigenvalue decomposition on hardware IP, and other functions such as evaluation of covariance matrix and whitened matrix are executed on the ARM Cortex-A9 processor. In scenario (ii), we implemented the computation of eigenvalue decomposition, and whitened matrix on the hardware IP, whereas the covariance matrix computation on the processor. In both the scenarios, we compare the resources and speedups achieved using these two scenarios.

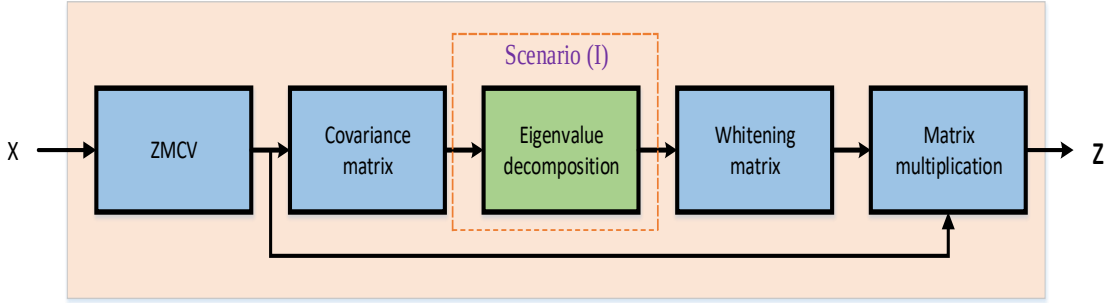


Figure 5.1: Scenario 1 for implementation of whitening with ZCA

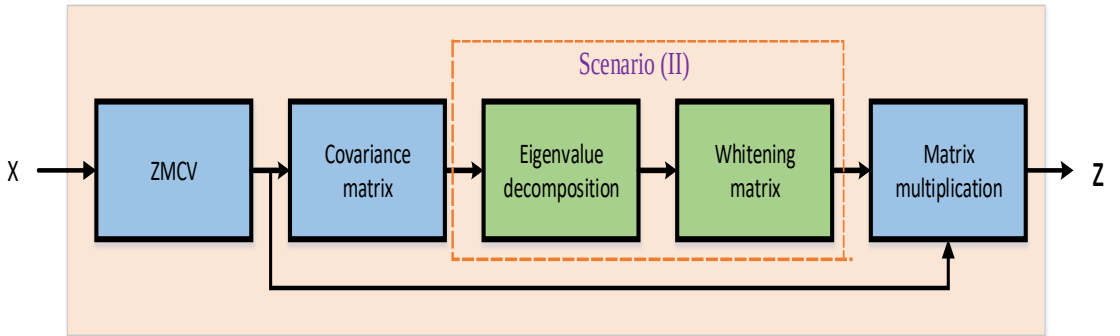


Figure 5.2: Scenario 2 for implementation of whitening with ZCA

5.1.2 Whitening with ZCA algorithm

Mathematically, whitening transforms a random vector $\mathbf{x} = (x_1, x_2, \dots, x_i)$ into a set of new random variables $\mathbf{Z}$ using a whitening matrix $\mathbf{W}$: $\mathbf{Z} = \mathbf{W}\mathbf{x}$ that satisfies $Cov(\mathbf{Z}) = \mathbf{I}$. $\mathbf{I}$ defines the identity matrix [137]. For whitening, one of the main requirement is that the signal need to be of zero-mean. The steps for computing ZCA of a vector $\mathbf{x}$ is detailed below:

1. Find the zero-mean signal by subtracting the mean

$$\hat{x}_i = x_i - x_\mu \quad (5.1)$$

where mean is defined as

$$x_\mu = \frac{1}{m} \sum_{i=1}^m x_i \quad (5.2)$$

2. Find the covariance of zero-mean signal as

$$\Sigma = E(\hat{\mathbf{x}}\hat{\mathbf{x}}^T) = \frac{1}{m} \sum_{i=1}^m (\hat{x}_i) (\hat{x}_i)^T \quad (5.3)$$

3. Decompose the covariance matrix Σ as

$$\Sigma = V.D.V^T \quad (5.4)$$

where D and V are eigenvalues and eigenvectors of the covariance matrix respectively.

4. Evaluate the whitened matrix for ZCA as

$$\mathbf{W} = \Sigma^{-\frac{1}{2}} \quad (5.5)$$

5. Evaluate the ZCA of signal as

$$\mathbf{Z} = \mathbf{W}.\hat{\mathbf{x}} \quad (5.6)$$

Further, $\mathbf{Z}$ satisfies

$$E[\mathbf{Z}\mathbf{Z}^T] = \mathbf{W}\Sigma\mathbf{W}^T = \mathbf{I}. \quad (5.7)$$

where $\mathbf{W}$ is the whitened matrix that satisfies $\mathbf{W}\Sigma\mathbf{W}^T\mathbf{W} = \mathbf{W}$. Thus $\mathbf{W}^T\mathbf{W} = \Sigma^{-1}$.

Substituting Equation(5.5) in Equation(5.6) we get

$$\mathbf{Z} = \Sigma^{-\frac{1}{2}}.\hat{\mathbf{x}} \quad (5.8)$$

By substituting Equation(5.4) in Equation(5.8), the ZCA of signal vector $\mathbf{x}$, i.e., Equation(5.8) can be rewritten as

$$\mathbf{Z} = V.D^{-\frac{1}{2}}.V^T.\hat{\mathbf{x}} \quad (5.9)$$

The evaluation of Equation (5.9) requires the computation of eigenvalues and eigenvectors, which is computationally intensive. We used the Lanczos algorithm followed by an implicit TriQR algorithm to find the covariance matrix's eigenvalues. The detailed steps are shown in Chapter 3.

5.1.3 Results and Discussion

ZCA whitening is implemented on Pynq-Z1 board using the HW-SW co-design as shown in Figure 5.3. Subsequently, it was implemented on PS, to compare the execution time performance.

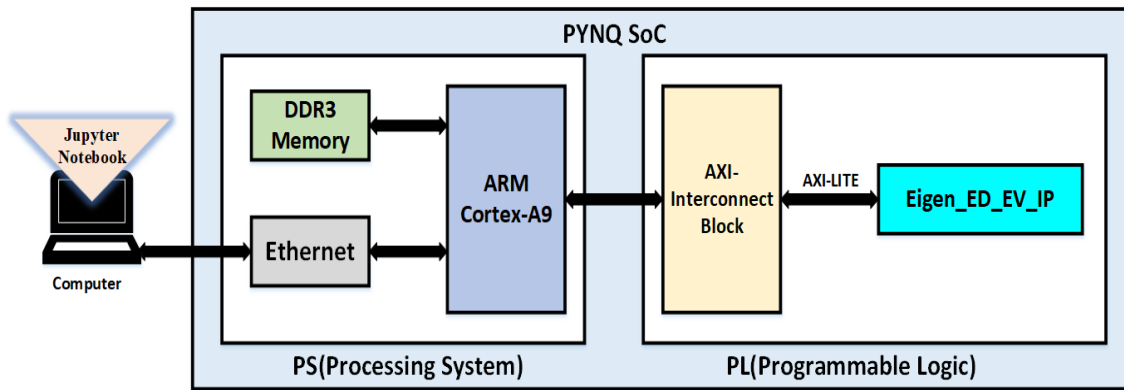


Figure 5.3: Pynq SoC architecture for implementation of whitening with ZCA

We used a random input vector for ZCA whitening for the test case. Figure 5.4 shows the scatter plot of the original signal, mean subtracted signal, and whitened signal for the data feature dimension 1000×20 . From Figure 5.4, we observe that

the ZCA minimized the correlation in the signal. The functional results demonstrate the working of the ZCA algorithm on the processor.

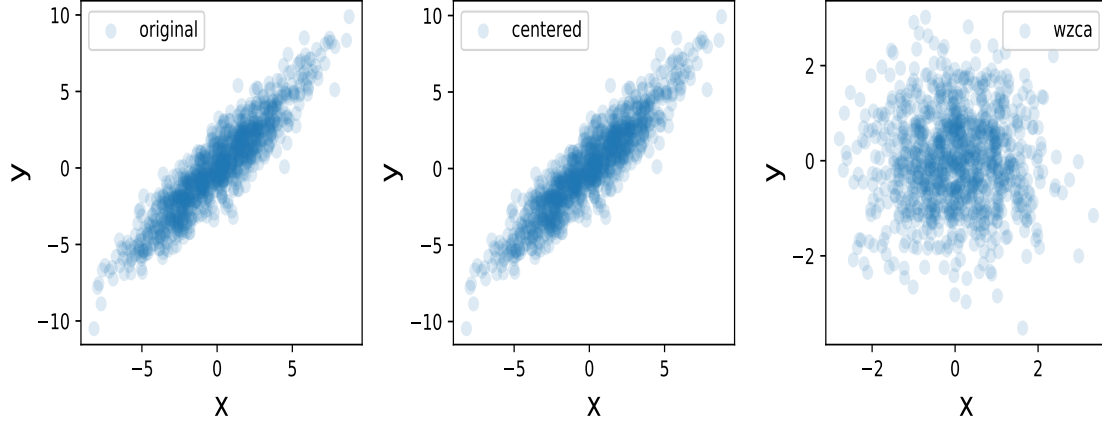


Figure 5.4: Software implementation of whitening with ZCA

In the HW-SW co-design framework, we considered two scenarios. In the first scenario, eigenvalue decomposition Equation (5.4) was implemented on HW as an IP, and the remaining algorithmic steps such as Equation (5.3), Equation (5.5) and Equation (5.9) were implemented on PS. The overlay design of scenario 1 is shown in figure 5.5.

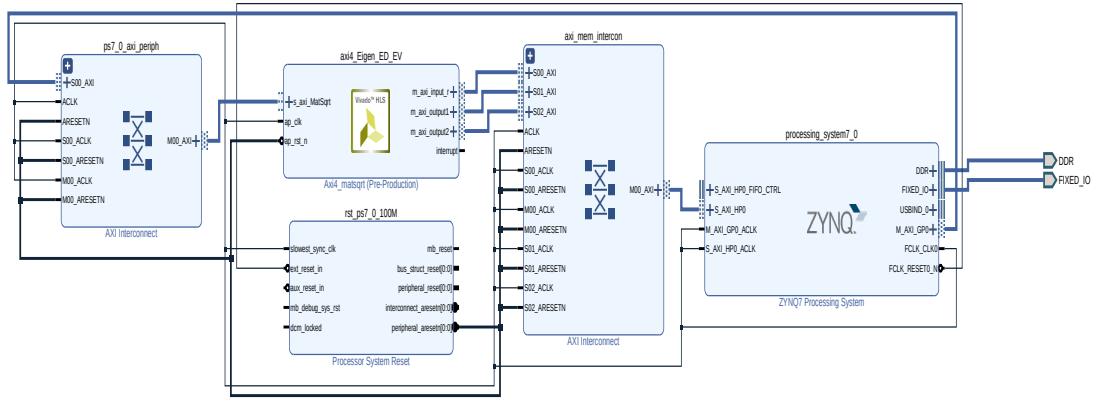


Figure 5.5: Overlay design of Scenario 1 for implementation of whitening with ZCA

In the second scenario, the computations of eigenvalue decomposition Equation (5.4) and whitened matrix (5.5) were implemented on HW as an IP, whereas the covariance matrix computation Equation (5.3) and Equation (5.9) were implemented on PS. The overlay design of scenario 2 is shown in figure 5.6.

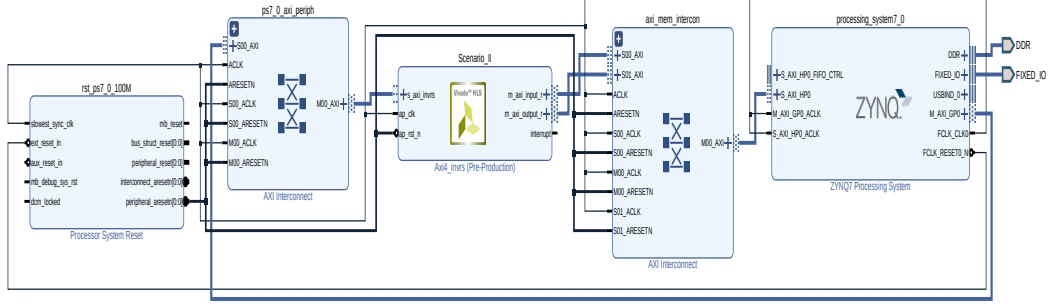


Figure 5.6: Overlay design of Scenario 2 for implementation of whitening with ZCA

The maximum frequency of operation (f_{max}), along with different resource utilization for different input data dimensions, is shown in Table 5.1. It demonstrates that when both eigenvalue decomposition and whitened matrix are implemented on hardware as an IP, resources like LUTs and slice registers increase marginally compared to evaluating eigenvalue decomposition on the hardware IP.

Table 5.1: Comparison of resource utilization for different data feature dimensions

Resources	Eigenvalue Decomposition (EVD)			(EVD + Whitenen Matrix)		
Data Dimension	10×10	15×15	20×20	10×10	15×15	20×20
f_{max} (MHz)	109.08	109.08	109.08	109.08	109.08	109.08
Slice LUTs	13688 (25.73%)	13725(25.80%)	14114 (26.53%)	18157(34.13%)	18193(34.20%)	18516(34.80%)
Slice Registers	15531(14.60%)	15590 (14.65%)	15745(14.80%)	20415(19.19%)	20480(19.25%)	20637(19.40%)
BRAM	8 (5.71%)	8 (5.71%)	8 (5.71%)	8(5.71%)	8(5.71%)	8(5.71%)
DSP	50 (22.73%)	50 (22.73%)	50 (22.73%)	66(30%)	66(30%)	66(30%)
LUTRAM	572 (3.29%)	576 (3.31%)	765 (4.40%)	593(3.41%)	596(3.43%)	753(4.33%)
BUFG	1 (3.13%)	1 (3.13%)	1 (3.13%)	1(3.13%)	1(3.13%)	1(3.13%)

However, it is interesting to observe that the resources such as BRAMs, DSP slices are not increasing even if the input dimension changes from 10×10 to 20×20 . It is because we are changing only the dimension of input matrix size, not the algorithm's logic. All three dimensions of matrix size fit into 8 BRAMs and used the same numbers of DSP slices resulting in almost the same resources. If we further increase the dimension of the matrix size, then the usage of BRAMs increases, and the remaining resources LUT and FF increase little.

Further, we calculated the software and hardware execution time of eigenvalue decomposition for different input sizes and tabulated the software (SW) and hardware (HW) execution time in Table 5.2. SW and HW execution time refers to the

time when eigenvalue decomposition is executed on PS (ARM Cortex-A9 processor) and PL (hardware IP), respectively. From these execution times, we calculated the hardware acceleration factor (HAF).

Table 5.2: SW and HW execution time of eigenvalue decomposition for different input feature dimensions

Data Feature Dimensions	Eigenvalue Decomposition		
	SW (msec)	HW (msec)	HAF
10×10	70.13	1.26	55.55
15×15	205.26	3.65	56.23
20×20	470.18	8.22	57.19

From Table 5.2, we observe that for the matrix dimension 10×10 , the SW execution time for eigenvalue decomposition on the Cortex-A9 processor is approximately 70 msec. In contrast, it is 1.2 msec on hardware IP resulting in hardware acceleration of approximately 55x. We also observe that the HAF increases marginally with an increase in the covariance matrix dimension, i.e., signal length. After obtaining, a sufficient acceleration factor for the eigenvalue decomposition, we realized ZCA whitening computation using the HW-SW co-design framework, where eigenvalue decomposition and whitened matrix computations are on hardware IP; covariance matrix and ZCA computations are carried out on the processor. We intentionally executed the covariance computation on the processor to save resources on FPGA for performing other applications. For obtaining the SW execution time, the complete ZCA whitening algorithm from Equation (5.1) to Equation (5.9) was executed on the processor.

Secondly, we implemented ZCA whitening in an HW-SW co-design framework in two scenarios as mentioned above and their execution time for whitening the different dimensions of the data features are listed in Table 5.3. We observed that the HAF increases with the dimension of the data features. Table 5.3 infers that in scenario (ii) we get improved HAF than scenario (i). The HW-SW co-design implementation results of ZCA whitening for a random input vector having the data feature dimension 1000×20 as shown in Figure 5.7. It is identical to SW

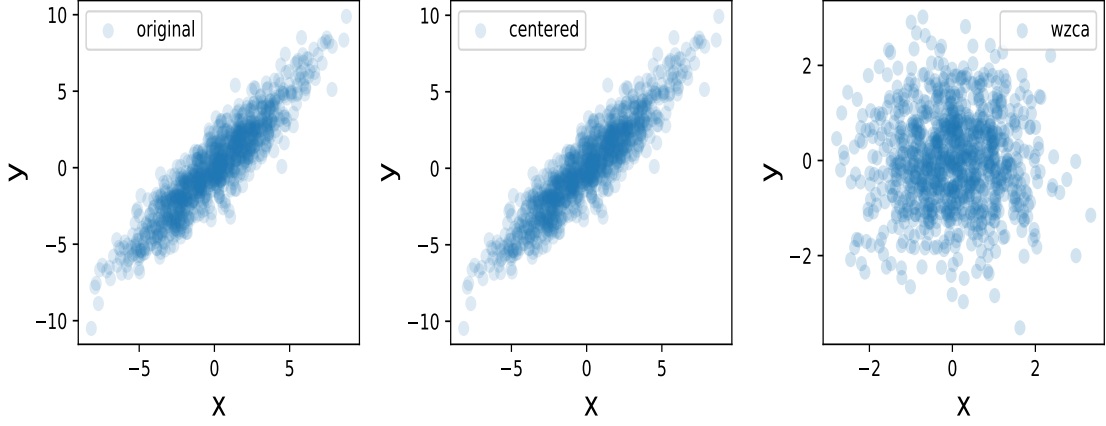


Figure 5.7: Hardware-software co-design implementation of whitening with ZCA

Table 5.3: SW and HW-SW co-design execution time of whitening with ZCA for different data feature dimensions

Data Feature Dimensions	Whitening				
	SW (msec)	HW-SW Co-Design			
		Scenario (i) (msec)	HAF	Scenario (ii) (msec)	HAF
1000×10	79.43	12.06	6.58	9.92	8.00
1000×15	207.63	18.22	11.39	14.98	13.86
1000×20	501.71	33.35	15.04	27.61	18.11

implementation results shown in Figure 5.4.

From Table 5.3, we observe that the SW execution time for complete whitening on Cortex-A9 processor is approximately 80 msec. In contrast, the execution time is approximately 12 msec for scenario (i) and 10 msec for scenario (ii), resulting in hardware acceleration of approximately 6x for scenario (i) and 8x for scenario (ii) respective to the matrix dimension 1000×10 . We also observe that the HAF increases with an increase in covariance matrix dimension.

5.1.4 Conclusions

In this work, we presented the ZCA whitening algorithm on a Pynq-Z1 FPGA in the HW-SW co-design framework using the Vivado HLS tool. At the beginning,

eigenvalue decomposition was evaluated using the Lanczos and implicit TriQR algorithms. It was implemented on the Pynq-Z1 FPGA, and its hardware acceleration was evaluated with varied input dimensions. Its promising acceleration factor led to developing an integration of both eigenvalue decomposition and the whitened matrix on the Pynq-Z1 FPGA as a standalone HW IP. We obtained a higher acceleration factor compared to only accelerating the eigenvalue decomposition for the implementation of ZCA whitening using HW-SW co-design framework. This is due to two reasons: (i) when only eigenvalue decomposition was executed on HW, the bus transaction time is more than when both eigenvalue decomposition and the whitened matrix were implemented as HW IP. Secondly, we achieve more acceleration when more computation is performed on HW. However, the choice entirely depends on the available resources in the hardware logic of the FPGA. The HW acceleration factor can also be improved by implementing a fixed-point HW IP. In the future, we will implement covariance computation and realize the complete ZCA whitening in a standalone fixed-point HW IP for accelerating the whitening algorithm.

5.2 Hardware implementation of whitening using PCA

Data whitening is a commonly used technique for preprocessing raw data to reduce redundancy. The Principal Component Analysis (PCA) method is often employed in the whitening process for dimensionality reduction. Eigenvalue decomposition plays a pivotal role in PCA, enhancing its computational efficiency. The current work presents a hardware-efficient implementation of PCA utilizing eigenvalue decomposition with the Lanczos and implicit TriQR algorithms on the Pynq-Z1 FPGA platform. The hardware implementation of PCA leverages the Xilinx Vivado High-Level Synthesis (HLS) platform, employing loop pipeline optimization to enhance performance. The interface can be established between the hardware IP logic unit and the processor with the help of AXI-Lite.

5.2.1 Introduction

Data whitening is a crucial pre-processing technique widely utilized in various applications like big data and machine learning [138–140]. It transforms correlated signals with identical covariances into de-correlated signals, aiding in feature extraction [141]. The principal component analysis is a fundamental technique in whitening data, reducing correlated variables to fewer uncorrelated ones by providing principal components. Due to its dimensional reduction property, PCA finds applications in diverse fields including hyper-spectral imaging, microwave imaging, pattern recognition, and computer vision [142–144]. Notably, recent studies have shown PCA’s effectiveness in face recognition [145, 146] and machine defect classification [147]. Additionally, PCA clustering models have been analyzed for brain tumor segmentation in T1w MRI images, showcasing improved performance rates [148].

While various prototypes for PCA exist in the literature, hardware implementations on FPGA remain limited due to complexity and resource constraints. Notable works include network-on-chip implementations [149], hardware implementations for network intrusion detection systems [150], and reconfigurable hardware architectures

[151]. Optimization of PCA implementations on FPGA for spectral image analysis has also been explored [152].

Notably, Lanczos and the implicit TriQR algorithm require minimal storage and have reduced computational requirements, making them suitable for eigenvalue decomposition in PCA implementations.

In recent years, significant research efforts have focused on implementing Principal Component Analysis (PCA) on various platforms including CPUs, GPUs, and FPGAs [153]. Among these platforms, FPGA implementations offer superior computational performance compared to CPUs and GPUs [154]. The design of hardware on FPGAs typically involves using HDL languages like VHDL or Verilog. However, as embedded systems grow in complexity, designing hardware at the register-transfer level (RTL) can pose challenges in finding the optimal hardware architecture.

To address this challenge, High-Level Synthesis (HLS) has emerged as an alternative approach. HLS has gained popularity in FPGA hardware design due to its ability to decrease design development time and make it easy to market on time [155]. The proposed hardware PCA has been implemented on the Pynq-Z1 board.

5.2.2 Principal Component Analysis algorithm

In multivariate statistical analysis, PCA became a more prominent technique for dimensional reduction on input data to remove the redundancy information. It means transforming high-dimension datasets to lower dimensions by retaining the most information [156]. The input data $\mathbf{X} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^{m \times n}$. In the PCA algorithm, $\mathbf{X}$ is the input of PCA and produces a lower dimensionality array $\mathbf{Z}$ through the steps as mean, covariance, eigenvalue decomposition and sorting shown below steps.

1. Evaluate the zero-mean signal:

One of the main requirements for whitening data is that the signal needs to be of zero mean. Before evaluating the zero mean signal, find the input data's

mean.

$$\mu_k = \frac{1}{m} \sum_{i=1}^m x_{ik} \quad (5.10)$$

Zero-mean signal:

$$\hat{x}_i = x_k - \mu_k \quad (5.11)$$

2. Evaluate the covariance of zero-mean signal as

$$\Sigma = \frac{X^T \times X}{m} \quad (5.12)$$

3. Evaluate the eigenvalue decomposition to the covariance matrix

$$[D, V] = EVD(\Sigma) \quad (5.13)$$

where V and D are eigenvectors and eigenvalues of the covariance matrix respectively.

4. Sorting eigenvalues and eigenvectors:

$$\begin{aligned} [D_s, U_s] &= Sort(D) \\ V_s(:, j) &= V(:, U_s(k)) \end{aligned}$$

5. Find the whitened matrix for PCA as

$$\mathbf{Z}_{pca} = D^{-\frac{1}{2}} \cdot V^T \cdot \hat{\mathbf{x}} \quad (5.14)$$

In the PCA algorithm, the mean, covariance, and eigenvalue decomposition are the most computationally intensive components. Initially, mean values for each input data column are computed to facilitate data normalization, followed by the calculation of covariance from the normalized data. In the subsequent step, we employ the Lanczos and implicit TriQR algorithms to determine eigenvalues and eigenvectors from the covariance matrix.

After getting eigenvalues and eigenvectors, according to eigenvalues, these eigenvectors are sorted in descending order in the third step. Finally, the normalized

data from the input is projected into the principal components space to reduce redundancy in the input information. The block diagram representation of PCA calculation is illustrated in Figure 5.8.

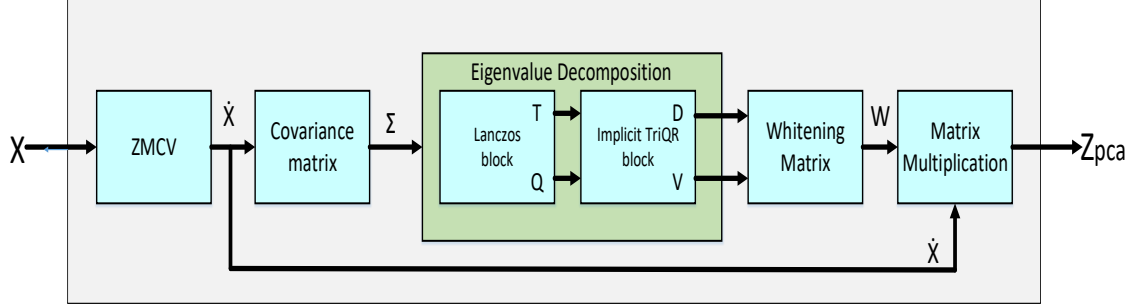


Figure 5.8: Block level representation of PCA implementation

5.2.3 Hardware Implementation

In this current work, we implemented eigenvalue decomposition using the Lanczos algorithm followed by the implicit TriQR algorithm, and the entire PCA on the Pynq-Z1 board, comparing it with previous implementations. The system-on-chip (SoC) implementation and overlay design of the PCA algorithm is illustrated in Figure 5.9 and Figure 5.10. After this implementation, we reported resource utilization, critical frequency, latency, and hardware execution time, and the clock period was set to 10 nanoseconds.

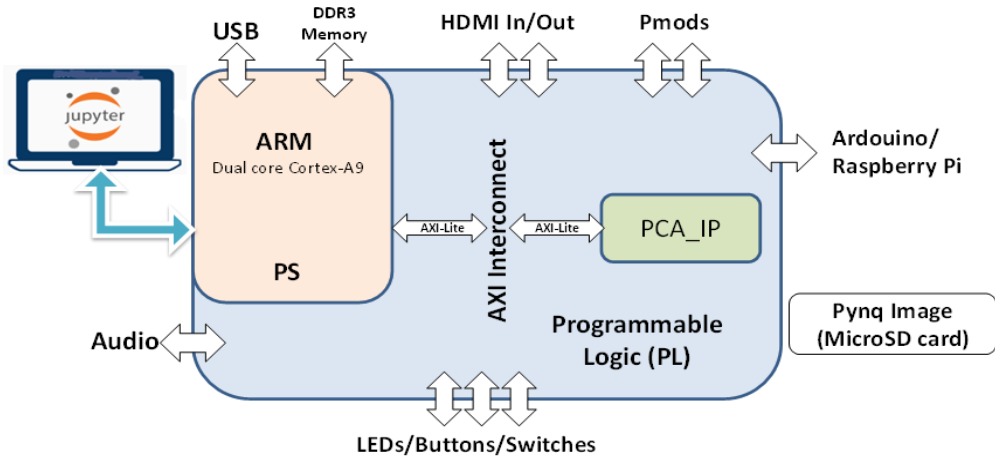


Figure 5.9: Pynq SoC architecture for PCA

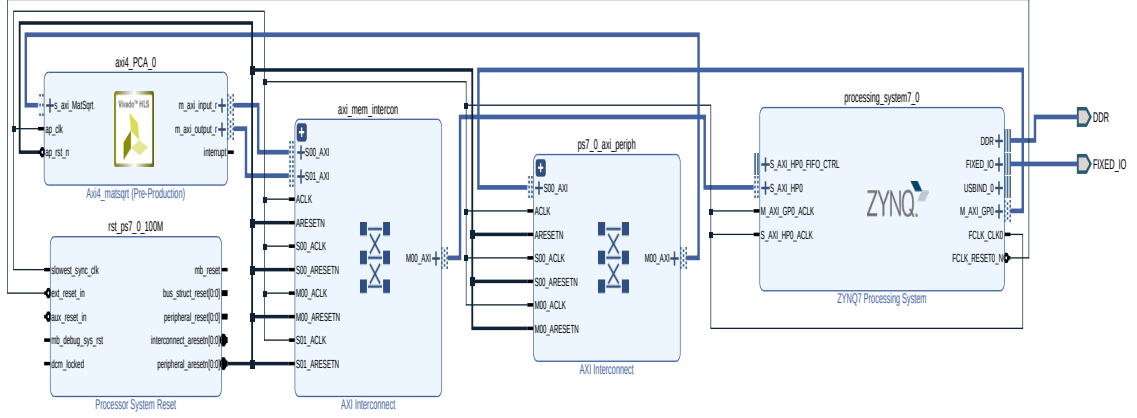


Figure 5.10: Overlay design of PCA

In this section, we employed a 32-bit floating-point data type in our PCA implementation on the Pynq-Z1 board for a matrix size of 16×30 with and without loop pipeline optimization. We reported resource utilization, critical frequency, latency, and hardware execution time, comparing them with a previous hardware PCA implementation. Additionally, to mitigate hardware execution time, we employed loop pipeline optimization in this implementation. The functionality with and without loop pipeline optimization is illustrated in Figure 4.11. Following the application of loop pipeline optimization, hardware execution time decreased from 9.27 milliseconds to 5 milliseconds with a slight increment in resource utilization.

Table 5.4: Comparison of resource utilization, throughput for various matrix size of eigenvalue decomposition

Ref.	Matrix Size	LUT	Reg	BRAM	DSP	Freq(Mhz)	Throughput
[151]	8×8	26994	33009	6	320	250	1.96×10^4
P.W.	8×8	13560	15387	8	50	112	0.12×10^4
[151]	16×16	100914	144866	11	1152	240	4.50×10^3
P.W.	16×16	13588	15558	8	50	110	0.23×10^3

First, we implemented eigenvalue decomposition to compare it with the previous eigenvalue decomposition implementation [151]. From Table 5.4, it is evident that the proposed eigenvalue decomposition, utilizing the Lanczos algorithm followed by the implicit TriQR algorithm, exhibited lower resource utilization compared to

the previous implementation [151], while providing comparable throughput across different matrix sizes such as 8×8 and 16×16 .

Table 5.5: Performance comparison of PCA implementation for matrix size 16×30

Work	Optimization	LUTs	Regs.	BRAMs	DSPs	Freq.(MHz)	Latency	Time (msec)
[157]	-	18884	13425	6	95	116	31707056	273
[151]	-	347269	275185	285	3170	183	289352	1.6
[158]	pipeline	145220	66790	132	385	95	64253	0.675
P.W	-	15009	20003	10	55	128.93	885012	9.27
P.W	Loop pipeline	16289	20638	11	55	128	465740	5

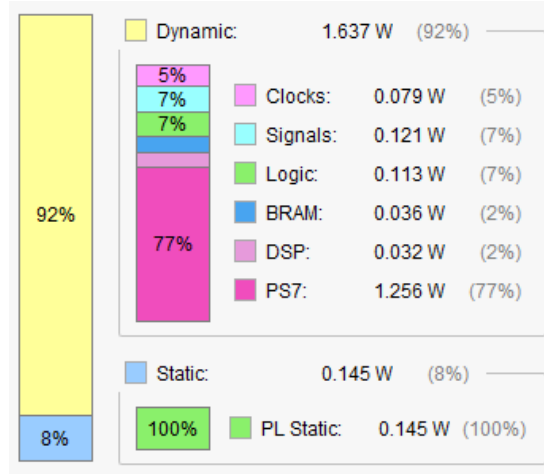


Figure 5.11: Power utilization of PCA implementation for a matrix size of 16×30

After the hardware implementation of the PCA algorithm, we compared it with the previous PCA implementation by Ali et al. [157] in Table 5.5. In the Table, latency is expressed as the number of clock cycles, and Time refers to the execution time, respectively. It is observed that our proposed implementation demonstrated reduced hardware execution time while utilizing similar resource utilization. When compared to previous implementations by Korat et al. [151] and Mansoori et al. [158], our implementation exhibited a slight increase in hardware execution time. Still, it consumed less resource utilization, as summarized in Table 5.5 and On-Chip power utilization reported as shown in Figure 5.11.

5.2.4 Conclusions

In this study, we successfully implemented a hardware-efficient PCA algorithm on a Pynq-Z1 FPGA using the Vivado HLS tool. Eigenvalue decomposition stands out as a crucial component in PCA implementation, and we utilized the Lanczos and implicit TriQR algorithms for this purpose. These algorithms offer computational efficiency due to their vector-based matrix operations, resulting in minimal storage requirements as the original matrix is updated only once during computation. First, we implemented eigenvalue decomposition and achieved less resource utilization and comparable throughput with previous implementation for the different matrix sizes. After that, PCA implementation demonstrates comparable hardware execution time while consuming fewer hardware resources compared to previous implementations. Additionally, we employed loop pipeline optimization to further reduce execution time. Looking ahead, we aim to explore additional optimization techniques and leverage fixed-point hardware IP to accelerate the PCA algorithm, thereby achieving even greater reductions in hardware execution time.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

This study highlights the significance of hardware accelerators in modern system design, particularly in the context of matrix algebra operations that form the foundation of numerous scientific and engineering applications. Matrix algebra plays a crucial role in accelerating application design, and this thesis has primarily focused on three core operations: matrix inversion, eigenvalue decomposition, and matrix square root computation. These operations, together with whitening transformations, were successfully implemented on the Pynq FPGA platform. In the context of artificial intelligence and embedded system applications, matrix inversion, eigenvalue decomposition, and matrix square root are indispensable computational building blocks.

A comparative study of different matrix inversion algorithms was first conducted using the MATLAB environment. The analysis revealed that, from both stability and computational efficiency perspectives, the QR decomposition-based approach offers advantages over LU, Cholesky, and modified Cholesky-based meth-

ods. Building on this insight, a hardware IP for single-precision floating-point QR decomposition was developed using the Modified Gram–Schmidt (MGS) technique, supporting AXI-Lite and DMA interfaces. The implementation demonstrated superior performance, with reduced latency and resource utilization, compared to the classical Gram–Schmidt-based QR decomposition realized through loop unrolling and AXI-BRAM data transfer. Furthermore, QR-based matrix inversion using the MGS algorithm for a 25×25 matrix achieved lower power consumption and resource usage, while maintaining comparable execution time relative to the QR-based Givens rotation method. The maximum absolute error remained within acceptable limits, confirming its suitability for embedded applications.

While QR-based inversion methods are effective for non-singular matrices, they fail in the presence of singular inputs. To address this limitation, a Singular Value Decomposition (SVD)-based inversion approach was proposed, utilizing the Lanczos method followed by the implicit TriQR algorithm. The SVD-based inversion successfully handled singular matrices. Although the Lanczos-based SVD approach exhibited a slight increase in power consumption and Figure of Merit (FoM), the execution time remained nearly comparable to QR-based inversion for matrices of size 25×25 . Moreover, the Lanczos-based SVD proved more effective for large sparse matrices, making it a promising alternative for complex applications.

In addition to inversion, eigenvalue decomposition was implemented on the Pynq FPGA using a combination of the Lanczos and implicit TriQR algorithms. The obtained eigenvalues and eigenvectors were subsequently employed to realize the matrix square root operation. The performance of these implementations was evaluated by comparing hardware and software execution times, from which hardware acceleration factors were derived. Results demonstrated substantial speedups, confirming the efficacy of FPGA-based acceleration for these matrix operations.

Finally, the thesis extended these foundational operations to the whitening process, a critical step in data preprocessing for applications such as Zero-phase Component Analysis (ZCA) and Principal Component Analysis (PCA). Whitening with ZCA was implemented using hardware–software co-design across multiple matrix sizes. The hardware acceleration factor was systematically evaluated, showing

consistent performance improvements over software-only execution. The complete whitening process was implemented using PCA on the Pynq FPGA, where results indicated reduced resource utilization and competitive execution times, particularly for matrices of size 16×32 . These results underline the efficiency of vector-based computations inherent in the Lanczos and implicit TriQR algorithms.

In summary, this thesis establishes a comprehensive framework for implementing and accelerating key matrix algebra operations on FPGA platforms. The proposed methodologies not only optimize resource utilization, power efficiency, and latency, but also provide scalable solutions for real-time embedded system applications.

6.2 Future work

While this thesis has demonstrated the efficient implementation of matrix inversion, eigenvalue decomposition, matrix square root, and whitening transformations on the Pynq FPGA platform, several avenues remain open for further research and development:

Scalability to Larger Matrices: The current implementations have been evaluated for moderate matrix sizes (up to 25×25 , 16×32 and 100×100). Extending these designs to handle significantly larger matrices, commonly encountered in scientific computing and machine learning, will require optimized memory management and parallel computation strategies. Techniques such as block matrix decomposition and hierarchical processing could be explored to improve scalability.

Parallel and Pipelined Architectures: Although loop pipelining and DMA-based optimizations have been employed, further improvements could be achieved by exploiting deep pipelining and parallelism at both the algorithmic and hardware levels. Implementing systolic array architectures or tensor-core-inspired designs may provide substantial acceleration for large-scale matrix operations.

Low-Precision and Mixed-Precision Computing: With the growing interest in approximate computing and quantization for AI workloads, future work could ex-

plore low-precision and mixed-precision arithmetic (e.g., FP16, INT8). This would not only reduce resource utilization but also improve performance while maintaining acceptable accuracy for many machine learning applications.

Integration with AI/ML Workloads: The matrix algebra operations implemented in this thesis—particularly eigenvalue decomposition and whitening—are fundamental in machine learning algorithms such as PCA, ICA, and dimensionality reduction techniques. Future research could integrate the developed hardware IPs into complete ML pipelines on FPGA platforms, targeting real-time applications in computer vision, signal processing, and biomedical engineering.

Support for Sparse Matrices: The Lanczos-based SVD method shows promise for large sparse matrices. Extending this work to design specialized sparse-matrix accelerators could provide significant advantages for graph processing, large-scale scientific simulations, and natural language processing applications.

BIBLIOGRAPHY

- [1] Biagio Peccerillo et al. “A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives”. In: *Journal of Systems Architecture* 129 (2022), p. 102561.
- [2] Tom Blank. “A survey of hardware accelerators used in computer-aided design”. In: *IEEE Design & Test of Computers* 1.3 (2009), pp. 21–39.
- [3] Anu Samanta, Indranil Hatai, and Ashis Kumar Mal. “A survey on hardware accelerator design of deep learning for edge devices”. In: *Wireless Personal Communications* 137.3 (2024), pp. 1715–1760.
- [4] Barry M Wise and Neal B Gallagher. “An introduction to linear algebra”. In: *Critical reviews in analytical chemistry* 28.1 (1998), pp. 1–20.
- [5] Calyampudi Radhakrishna Rao and M Bhaskara Rao. *Matrix algebra and its applications to statistics and econometrics*. World Scientific, 1998.
- [6] Shayle R Searle and Andre I Khuri. *Matrix algebra useful for statistics*. John Wiley & Sons, 2017.
- [7] Xian-Da Zhang. “A matrix algebra approach to artificial intelligence”. In: (2020).

- [8] Ling Zhuo and Viktor K Prasanna. “High-performance designs for linear algebra operations on reconfigurable hardware”. In: *IEEE Transactions on Computers* 57.8 (2008), pp. 1057–1071.
- [9] Yunwen Xu et al. “An improved predictive controller on the FPGA by hardware matrix inversion”. In: *IEEE Transactions on Industrial Electronics* 65.9 (2018), pp. 7395–7405.
- [10] Neelesh Ranjan Srivastava and Vikas Mittal. “A Review on Hardware Implementations of Signal Processing Algorithms”. In: *Latest Trends in Renewable Energy Technologies: Select Proceedings of NCRESE 2020* (2021), pp. 295–302.
- [11] Chao Wang et al. “Reconfigurable hardware accelerators: Opportunities, trends, and challenges”. In: *arXiv preprint arXiv:1712.04771* (2017).
- [12] Christoforos Kachris and Dimitrios Soudris. “A survey on reconfigurable accelerators for cloud computing”. In: *2016 26th International conference on field programmable logic and applications (FPL)*. IEEE. 2016, pp. 1–10.
- [13] Hiroyuki Usui et al. “DASH: Deadline-aware high-performance memory scheduler for heterogeneous systems with hardware accelerators”. In: *ACM Transactions on Architecture and Code Optimization (TACO)* 12.4 (2016), pp. 1–28.
- [14] R Oishi-Tomiyasu et al. “Application of matrix decomposition algorithms for singular matrices to the Pawley method in Z-Rietveld”. In: *Applied Crystallography* 45.2 (2012), pp. 299–308.
- [15] AS Morse. “A gain matrix decomposition and some of its applications”. In: *Systems & Control Letters* 21.1 (1993), pp. 1–10.
- [16] Rick Ma et al. “Decomposition approach for low-rank matrix completion and its applications”. In: *IEEE Transactions on Signal Processing* 62.7 (2014), pp. 1671–1683.
- [17] Jun Lu. “Matrix decomposition and applications”. In: *arXiv preprint arXiv:2201.00145* (2022).

-
- [18] Fuzhen Zhang. “A matrix decomposition and its applications”. In: *Linear and Multilinear Algebra* 63.10 (2015), pp. 2033–2042.
- [19] Genevera I Allen, Logan Grosenick, and Jonathan Taylor. “A generalized least-square matrix decomposition”. In: *Journal of the American Statistical Association* 109.505 (2014), pp. 145–159.
- [20] Israel Gohberg, Marinus A Kaashoek, and Ilya M Spitkovsky. “An overview of matrix factorization theory and operator applications”. In: *Factorization and Integrable Systems: Summer School in Faro, Portugal, September 2000* (2003), pp. 1–102.
- [21] Gil Shabat et al. “Randomized LU decomposition”. In: *Applied and Computational Harmonic Analysis* 44.2 (2018), pp. 246–272.
- [22] Aravindh Krishnamoorthy and Deepak Menon. “Matrix inversion using Cholesky decomposition”. In: *2013 signal processing: Algorithms, architectures, arrangements, and applications (SPA)*. IEEE. 2013, pp. 70–72.
- [23] Haw-ren Fang and Dianne P O’leary. “Modified Cholesky algorithms: a catalog with new approaches”. In: *Mathematical programming* 115.2 (2008), pp. 319–349.
- [24] Reza Pourramezan et al. “A real-time synchrophasor data compression method using singular value decomposition”. In: *IEEE Transactions on Smart Grid* 13.1 (2021), pp. 564–575.
- [25] Phuong Thi Nha, Ta Minh Thanh, and Nguyen Tuan Phong. “Consideration of a robust watermarking algorithm for color image using improved QR decomposition”. In: *Soft Computing* 26.11 (2022), pp. 5069–5093.
- [26] Stephan Weiss, Ian K Proudler, and Fraser K Coutts. “Eigenvalue decomposition of a parahermitian matrix: Extraction of analytic eigenvalues”. In: *IEEE Transactions on Signal Processing* 69 (2021), pp. 722–737.
- [27] M Rafique. “The Solution of a System of Linear Equations by an Improved LU-Decomposition Method”. In: *American Journal of Scientific and Industrial Research, Science Huß* (2015).

- [28] Yang Xia et al. “End-to-End LU Factorization of Large Matrices on GPUs”. In: *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 2023, pp. 288–300.
- [29] Santiago Badia, Alberto F Martín, and Ramon Planas. “Block recursive LU preconditioners for the thermally coupled incompressible inductionless MHD problem”. In: *Journal of Computational Physics* 274 (2014), pp. 562–591.
- [30] Ramesh Ragala and G Bharadwaja Kumar. “Recursive block LU decomposition based ELM in apache spark”. In: *Journal of Intelligent & Fuzzy Systems* 39.6 (2020), pp. 8205–8215.
- [31] Sameer Deshmukh et al. “O (N) distributed direct factorization of structured dense matrices using runtime systems.” In: *Proceedings of the 52nd International Conference on Parallel Processing*. 2023, pp. 1–10.
- [32] Iria CS Cosme et al. “Memory-usage advantageous block recursive matrix inverse”. In: *Applied Mathematics and Computation* 328 (2018), pp. 125–136.
- [33] Thomas Bondo Pedersen et al. “The versatility of the Cholesky decomposition in electronic structure theory”. In: *Wiley interdisciplinary reviews: Computational molecular science* 14.1 (2024), e1692.
- [34] Stefan Chmiela et al. “Accurate global machine learning force fields for molecules with hundreds of atoms”. In: *Science Advances* 9.2 (2023), eadf0873.
- [35] Sadiqur Rahaman et al. “Complexity analysis of matrix decomposition algorithms for linear MIMO detection”. In: *2016 5th International Conference on Informatics, Electronics and Vision (ICIEV)*. IEEE. 2016, pp. 927–932.
- [36] Venkata Reddy Kopparthi, Rangababu Peesapati, and Samrat L Sabat. “System on chip implementation of low complex orthogonal matching pursuit algorithm on FPGA”. In: *2020 6th International Conference on Signal Processing and Communication (ICSC)*. IEEE. 2020, pp. 178–184.
- [37] Domen Stadler et al. “Modified Cholesky decomposition for solving the moment matrix in the radial point interpolation method”. In: *International Journal of Computational Methods* 11.06 (2014), p. 1350088.

-
- [38] Ahmed A Hussain et al. “FPGA-based hardware implementation of computationally efficient multi-source DOA estimation algorithms”. In: *IEEE access* 7 (2019), pp. 88845–88858.
 - [39] Thomas McSweeney. “Modified Cholesky decomposition and applications”. PhD thesis. University of Manchester, 2017.
 - [40] Jongin Park, Seok-Min Wi, and Jin S Lee. “Computationally efficient adaptive beamformer for ultrasound imaging based on QR decomposition”. In: *IEEE transactions on ultrasonics, ferroelectrics, and frequency control* 63.2 (2016), pp. 256–265.
 - [41] Ali Umut Irturk. “Implementation of QR decomposition algorithm using FPGAs”. PhD thesis. Citeseer, 2007.
 - [42] Takeshi Terao, Katsuhisa Ozaki, and Takeshi Ogita. “LU-Cholesky QR algorithms for thin QR decomposition”. In: *Parallel Computing* 92 (2020), p. 102571.
 - [43] Yousef Jaradat et al. “A tutorial on singular value decomposition with applications on image compression and dimensionality reduction”. In: *2021 international conference on information technology (ICIT)*. IEEE. 2021, pp. 769–772.
 - [44] Jifei Miao et al. “Quaternion higher-order singular value decomposition and its applications in color image processing”. In: *Information Fusion* 92 (2023), pp. 139–153.
 - [45] Kuntal Kumar Pal and KS Sudeep. “Preprocessing for image classification by convolutional neural networks”. In: *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. IEEE. 2016, pp. 1778–1781.
 - [46] Alejandra Sarahi Sanchez-Moreno et al. “Efficient face recognition system for operating in unconstrained environments”. In: *Journal of Imaging* 7.9 (2021), p. 161.

- [47] Vranda Jajoo and Sanjay Tanwani. “A Survey on Data Preprocessing Techniques in Stream Mining”. In: *International Conference on Information and Communication Technology for Intelligent Systems*. Springer. 2024, pp. 93–106.
- [48] Shuihua Wang et al. “Advances in data preprocessing for biomedical data fusion: An overview of the methods, challenges, and prospects”. In: *Information Fusion* 76 (2021), pp. 376–421.
- [49] Khaled Mahmud Sujon et al. “When to use standardization and normalization: empirical evidence from machine learning models and XAI”. In: *IEEE access* (2024).
- [50] Nelly Tochi Nwosu. “Reducing operational costs in healthcare through advanced BI tools and data integration”. In: *World Journal of Advanced Research and Reviews* 22.3 (2024), pp. 1144–1156.
- [51] Papia Ray, S Surender Reddy, and Tuhina Banerjee. “Various dimension reduction techniques for high dimensional data analysis: a review”. In: *Artificial Intelligence Review* 54.5 (2021), pp. 3473–3515.
- [52] Hongfei Zhang et al. “Crack detection based on convnext and normalization”. In: *Journal of Physics: Conference Series*. Vol. 2289. 1. IOP Publishing. 2022, p. 012022.
- [53] Bijan G Mobasser and Amro Lulu. “Radiometric identification of signals by matched whitening transform”. In: *Sensors* 21.24 (2021), p. 8398.
- [54] Ziyuan Wang and You Wu. “Investigating the effectiveness of whitening post-processing methods on modifying LLMs representations”. In: *2023 IEEE 35th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE. 2023, pp. 813–820.
- [55] G Thippa Reddy et al. “Analysis of dimensionality reduction techniques on big data”. In: *Ieee Access* 8 (2020), pp. 54776–54788.
- [56] Wayne H Wolf. “Hardware-software co-design of embedded systems”. In: *Proceedings of the IEEE* 82.7 (2002), pp. 967–989.

-
- [57] Giovanni De Michell and Rajesh K Gupta. “Hardware/software co-design”. In: *Proceedings of the IEEE* 85.3 (1997), pp. 349–365.
- [58] Qingcheng Xiao et al. “HASCO: Towards agile hardware and software co-design for tensor computation”. In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2021, pp. 1055–1068.
- [59] Dake Liu. “Application specific instruction set DSP processors”. In: *Handbook of Signal Processing Systems*. Springer, 2010, pp. 415–447.
- [60] Wenqing Li et al. “TCADer: A Tightly Coupled Accelerator Design framework for heterogeneous system with hardware/software co-design”. In: *Journal of Systems Architecture* 136 (2023), p. 102822.
- [61] Vittoriano Muttillio et al. “Systemc-based co-simulation/analysis for system-level hardware/software co-design”. In: *Computers and Electrical Engineering* 110 (2023), p. 108803.
- [62] Philippe Coussy et al. “An introduction to high-level synthesis”. In: *IEEE Design & Test of Computers* 26.4 (2009), pp. 8–17.
- [63] Francisco J Iñiguez-Lomeli, Carlos H Garcia-Capulin, and Horacio Rostro-Gonzalez. “A hardware architecture for single and multiple ellipse detection using genetic algorithms and high-level synthesis tools”. In: *Microprocessors and Microsystems* 111 (2024), p. 105106.
- [64] B. Schafer and Z. Wang. “High-Level Synthesis Design Space Exploration: Past, Present, and Future”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39 (2020), pp. 2628–2639.
- [65] J. Gorius, S. Rokicki, and S. Derrien. “SpecHLS: Speculative Accelerator Design Using High-Level Synthesis”. In: *IEEE Micro* 42 (2022), pp. 99–107.
- [66] C. Schlaak, T. Juang, and C. Dubach. “Memory-Aware Functional IR for Higher-Level Synthesis of Accelerators”. In: *ACM Transactions on Architecture and Code Optimization (TACO)* 19 (2022), pp. 1–26.

- [67] Jason Cong et al. “FPGA HLS today: successes, challenges, and opportunities”. In: *ACM Transactions on Reconfigurable Technology and Systems (TRETs)* 15.4 (2022), pp. 1–42.
- [68] Murali Ravi, Angu Sewa, Siva Sankara Sai Sanagapati, et al. “FPGA as a hardware accelerator for computation intensive maximum likelihood expectation maximization medical image reconstruction algorithm”. In: *IEEE Access* 7 (2019), pp. 111727–111735.
- [69] Chenfeng Zhao et al. “HLPerf: demystifying the performance of HLS-based graph neural networks with dataflow architectures”. In: *ACM transactions on reconfigurable technology and systems* 18.1 (2024), pp. 1–26.
- [70] 5. *Optimizing with Dataflow — Vitis™ Tutorials 2022.1 documentation*. URL: https://xilinx.github.io/Vitis-Tutorials/2020-1/docs/build/html/docs/Hardware_Accelerators/Design_Tutorials/01-convolution-tutorial/dataflow.html.
- [71] Zhonghao Zhang and Zhengxiang Li. “ARM and FPGA heterogeneous accelerated processing system based on HLS and PCIe”. In: *2021 4th international conference on information and computer technologies (ICICT)*. IEEE. 2021, pp. 300–304.
- [72] BB Shabarinath and Muralidhar Pullakandam. “SoC-based real-time SVM classification with integrated training using HLS and PYNQ”. In: *Microprocessors and Microsystems* 101 (2023), p. 104878.
- [73] Shuai Li et al. “Novel CNN-based AP2D-net accelerator: An area and power efficient solution for real-time applications on mobile FPGA”. In: *Electronics* 9.5 (2020), p. 832.
- [74] *DMA — Python productivity for Zynq (Pynq) v1.0*. URL: <https://pynq.readthedocs.io/en/v2.3/pynq-libraries/dma.html>.
- [75] Mohamed El-Hadedy et al. “RECO-ASCON: Reconfigurable ASCON hash functions for IoT applications”. In: *Integration* 93 (2023), p. 102061.
- [76] Seifeddine Messaoud et al. “Deep convolutional neural networks-based Hardware–Software on-chip system for computer vision application”. In: *Computers & Electrical Engineering* 98 (2022), p. 107671.

-
- [77] Adam Taylor. “Hello PYNQ — the PYNQ edition of the MicroZed Chronicles!” In: (Aug. 2025). URL: <https://www.hackster.io/news/hello-pynq-the-pynq-edition-of-the-microzed-chronicles-55353a56c678>.
 - [78] Eilen García Rodríguez et al. “Methodology for online detection and classification of power quality disturbances based on FPGA”. In: *Applied Soft Computing* (2025), p. 112813.
 - [79] Mohammad A Al-Jarrah, Amin Jarrah, and Amal Alawaisah. “Automotive engine idle speed controller: Nonlinear model predictive control utilizing the firefly algorithm”. In: *Computers and Electrical Engineering* 108 (2023), p. 108688.
 - [80] Cuong V Nguyen et al. “A non-contact infection screening system using medical radar and Linux-embedded FPGA: Implementation and preliminary validation”. In: *Informatics in medicine unlocked* 16 (2019), p. 100225.
 - [81] Alaa Tharwat. “Principal component analysis-a tutorial”. In: *International Journal of Applied Pattern Recognition* 3.3 (2016), pp. 197–240.
 - [82] Paul A. White and Robert R. Brown. “A Comparison of Methods for Computing the Eigenvalues and Eigenvectors of a Real Symmetric Matrix”. In: *Mathematics of Computation* 18.87 (1964), pp. 457–463. ISSN: 00255718, 10886842.
 - [83] Lorentz Jäntschi. “Eigenproblem basics and algorithms”. In: *Symmetry* 15.11 (2023), p. 2046.
 - [84] Golub Gene H and Henk A Van der Vorst. “Eigenvalue computation in the 20th century”. In: *Journal of Computational and Applied Mathematics* 123.1-2 (2000), pp. 35–65.
 - [85] Douglas N Arnold et al. “Computing spectra without solving eigenvalue problems”. In: *SIAM Journal on Scientific Computing* 41.1 (2019), B69–B92.
 - [86] Di Yan, Wei-Xing Wang, and Xiao-Wei Zhang. “High-performance matrix eigenvalue decomposition using the parallel jacobi algorithm on FPGA”. In: *Circuits, Systems, and Signal Processing* 42.3 (2023), pp. 1573–1592.

- [87] Xiao-Wei Zhang et al. “High-performance of eigenvalue decomposition on FPGA for the DOA estimation”. In: *IEEE Transactions on Vehicular Technology* 72.5 (2022), pp. 5782–5797.
- [88] *Jacobi Eigenvalue Algorithm for symmetric Real matrices*. URL: <https://pages.hmc.edu/ruye/MachineLearning/lectures/ch1/node2.html>.
- [89] Wei Yu Mei et al. “An implementation of matrix eigenvalue decomposition with improved Jacobi algorithm”. In: *2010 First International Conference on Pervasive Computing, Signal Processing and Applications*. IEEE. 2010, pp. 952–955.
- [90] Hamza Fawzi and Harry Goulbourne. “Faster proximal algorithms for matrix optimization using Jacobi-based eigenvalue methods”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 11397–11408.
- [91] Toby RF Phillips et al. “An autoencoder-based reduced-order model for eigenvalue problems with application to neutron diffusion”. In: *International Journal for Numerical Methods in Engineering* 122.15 (2021), pp. 3780–3811.
- [92] Rohit B Chaurasiya and Rahul Shrestha. “Hardware-efficient and fast sensing-time maximum-minimum-eigenvalue-based spectrum sensor for cognitive radio network”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 66.11 (2019), pp. 4448–4461.
- [93] Ilayda Koseoglu et al. “An FPGA implementation of Givens rotation based digital architecture for computing eigenvalues of asymmetric matrix”. In: *2021 13th International Conference on Electrical and Electronics Engineering (ELECO)*. IEEE. 2021, pp. 470–474.
- [94] Ammar Mesloub, Adel Belouchrani, and Karim Abed-Meraim. “New algorithms on complex joint eigenvalue decomposition based on generalized Givens rotations”. In: *2019 27th European Signal Processing Conference (EUSIPCO)*. IEEE. 2019, pp. 1–5.
- [95] Uppu Eshwar, Soumyajit Chatterjee, and Sathya Peri. “Enhancing QR Decomposition: A GPU-Based Approach to Parallelizing the Householder Algorithm with CUDA Streams”. In: *International Conference on Distributed Computing and Intelligent Technology*. Springer. 2024, pp. 146–161.

-
- [96] *ALAFF Householder transformation*. URL: <https://www.cs.utexas.edu/~flame/laff/alaff/chapter03-householder-transformation.html>.
 - [97] Eyaya Fekadie Anley. “The QR Method for Determining All Eigenvalues of Real Square Matrices”. In: *Pure and Applied Mathematics Journal* 5.4 (2016), pp. 113–119.
 - [98] Eyaya Fekadie Anley. “The QR Method for Determining All Eigenvalues of Real Square Matrices”. In: *Pure and Applied Mathematics Journal* 5.4 (2016), pp. 113–119. DOI: 10.11648/j.pamj.20160504.15.
 - [99] Maryam Mohammadi and Fatemeh Nejati. “Watermarking based on Hessenberg matrix decomposition”. In: *Soft Computing Journal* 9.1 (2021), pp. 146–157.
 - [100] Xian-Ming Gu et al. “A Hessenberg-type algorithm for computing PageRank Problems”. In: *Numerical Algorithms* 89.4 (2022), pp. 1845–1863.
 - [101] Richard Bruno Lehoucq. *Analysis and implementation of an implicitly restarted Arnoldi iteration*. Rice University, 1995.
 - [102] Åke Björck and Sven Hammarling. “A Schur method for the square root of a matrix”. In: *Linear Algebra and its Applications* (1983), pp. 127–140.
 - [103] Y. Liu et al. “An Algebraic Multilevel Schur Complement Preconditioner Based on Eigenvalue Deflation”. In: *Numerical Linear Algebra with Applications* 32 (2025).
 - [104] N. Higham. “Computing real square roots of a real matrix”. In: *Linear Algebra and its Applications* (1987), pp. 405–430.
 - [105] E. Deadman, N. Higham, and R. Ralha. “Blocked Schur Algorithms for Computing the Matrix Square Root”. In: *Numerical Analysis and Optimization*. Springer, 2012, pp. 171–182.
 - [106] Matthew I Smith. “A Schur algorithm for computing matrix p th roots”. In: *SIAM journal on matrix analysis and applications* 24.4 (2003), pp. 971–989.
 - [107] Saba Akram and Quarrrat Ul Ann. “Newton raphson method”. In: *International Journal of Scientific & Engineering Research* 6.7 (2015), pp. 1748–1752.

- [108] Z. Bujanović, D. Kressner, and C. Schröder. “Iterative refinement of Schur decompositions”. In: *Numerical Algorithms* 92 (2022), pp. 247–267.
- [109] Z. Shi, Q. He, and Y. Liu. “Accelerating Parallel Jacobi Method for Matrix Eigenvalue Computation in DOA Estimation Algorithm”. In: *IEEE Transactions on Vehicular Technology* 69 (2020), pp. 6275–6285.
- [110] Hansheng Wang et al. “Extracting the Potential of Emerging Hardware Accelerators for Symmetric Eigenvalue Decomposition”. In: *ArXiv abs/2410.02170* (2024).
- [111] Valery Sklyarov et al. “Analysis and comparison of attainable hardware acceleration in all programmable systems-on-chip”. In: *2015 Euromicro Conference on Digital System Design*. IEEE. 2015, pp. 345–352.
- [112] Chuan Zhang et al. “On the low-complexity, hardware-friendly tridiagonal matrix inversion for correlated massive MIMO systems”. In: *IEEE Transactions on Vehicular Technology* 68.7 (2019), pp. 6272–6285.
- [113] Jingcan Zhu et al. “A new-type zeroing neural network model and its application in dynamic cryptography”. In: *IEEE Transactions on Emerging Topics in Computational Intelligence* (2024).
- [114] Lin Li and Jianhao Hu. “An efficient second-order neural network model for computing the Moore–Penrose inverse of matrices”. In: *IET Signal Processing* 16.9 (2022), pp. 1106–1117.
- [115] Jun Lu. “Numerical matrix decomposition”. In: *arXiv preprint arXiv:2107.02579* (2021).
- [116] Ahmed A Hussain et al. “FPGA hardware implementation of DOA estimation algorithm employing LU decomposition”. In: *IEEE access* 6 (2018), pp. 17666–17680.
- [117] Depeng Yang, Gregory D Peterson, and Husheng Li. “Compressed sensing and Cholesky decomposition on FPGAs and GPUs”. In: *Parallel Computing* 38.8 (2012), pp. 421–437.

-
- [118] Chong Yeam Tan et al. “Efficient hardware-accelerated pseudoinverse computation through algorithm restructuring for parallelization in high-level synthesis”. In: *International Journal of Circuit Theory and Applications* 50.2 (2022), pp. 394–416.
- [119] Javier Hormigo and Sergio D Muñoz. “Efficient floating-point givens rotation unit”. In: *Circuits, Systems, and Signal Processing* 40 (2021), pp. 2419–2442.
- [120] M Shibin Fabi and Jayaraj U Kidav. “FPGA Accelerated QRD-Based Matrix Inversion Core for Signal Processing”. In: *International Conference on Communication, Electronics and Digital Technology*. Springer. 2023, pp. 335–351.
- [121] Parth Desai, Semih Aslan, and Jafar Saniie. “FPGA implementation of Gram-Schmidt QR decomposition using high level synthesis”. In: *2017 IEEE International Conference on Electro Information Technology (EIT)*. IEEE. 2017, pp. 482–487.
- [122] Niansong Zhang. *Loop optimization in HLS*. Aug. 2022. URL: https://www.zzzdavid.tech/loop_opt/.
- [123] S Chetan et al. “Hardware Implementation of Floating Point Matrix Inversion Modules on FPGAs”. In: *IEEE 32nd International Conference on Microelectronics (ICM)*. 2020, pp. 1–4.
- [124] Kiran Maharana, Surajit Mondal, and Bhushankumar Nemade. “A review: Data pre-processing and data augmentation techniques”. In: *Global Transitions Proceedings* 3.1 (2022), pp. 91–99.
- [125] Agnan Kessy, Alex Lewin, and Korbinian Strimmer. “Optimal whitening and decorrelation”. In: *The American Statistician* 72.4 (2018), pp. 309–314.
- [126] Tai Yin Chiu. “Understanding Generalized Whitening and Coloring Transform for Universal Style Transfer”. In: *Proc. 2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 4451–4459. DOI: 10.1109/ICCV.2019.00455.

- [127] Y. T. Lin et al. “Feature Extraction of Fourier Infrared Signals from Pyrolysis Products based on ZCA and PSO”. In: *Proc. 2020 IEEE Congress on Evolutionary Computation (CEC)*. 2020, pp. 1–7. DOI: 10.1109/CEC48606.2020.9185916.
- [128] Lei Huang et al. “Decorrelated Batch Normalization”. In: *Proc. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 791–800. DOI: 10.1109/CVPR.2018.00089.
- [129] Gregory Murad Reis et al. “A Whitening-Based Tracking Algorithm for Autonomous Underwater Vehicles”. In: *Proc. 2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO)*. 2018, pp. 1–6. DOI: 10.1109/OCEANSKobe.2018.8559277.
- [130] Qian Yu Li, Jian Guo Jiang, and Mei Bin Qi. “Face recognition algorithm based on improved deep networks”. In: *Acta Electronica Sinica* 45.3 (2017), p. 619.
- [131] Pingfan Meng, Matthew Jacobsen, and Ryan Kastner. “FPGA-GPU-CPU heterogenous architecture for real-time cardiac physiological optical mapping”. In: *Proc. 2012 International Conference on Field-Programmable Technology*. 2012, pp. 37–42. DOI: 10.1109/FPT.2012.6412108.
- [132] Kim Bjerger, Jonathan Horsted Schougaard, and Daniel Ejnar Larsen. “A scalable and efficient convolutional neural network accelerator using HLS for a system-on-chip design”. In: *Microprocessors and Microsystems* 87 (2021), p. 104363.
- [133] Kyuho Shim et al. “System-level simulation acceleration for architectural performance analysis using hybrid virtual platform system”. In: *Proc. 2012 International SoC Design Conference (ISoCC)*. 2012, pp. 402–404. DOI: 10.1109/ISoCC.2012.6407126.
- [134] MD Edwards, J Forrest, and AE Whelan. “Acceleration of software algorithms using hardware/software co-design techniques”. In: *Journal of systems architecture* 42.9-10 (1997), pp. 697–707.
- [135] Flavio Ponzina et al. “A hardware/software co-design vision for deep learning at the edge”. In: *IEEE Micro* 42.6 (2022), pp. 48–54.

-
- [136] Dmitrijs Zaporanuks and Matthias Hauswirth. “Algorithmic profiling”. In: *Proceedings of the 33rd ACM SIGPLAN conference on Programming Language Design and Implementation*. 2012, pp. 67–76.
- [137] Lei Huang et al. “An investigation into the stochasticity of batch whitening”. In: *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*. 2020, pp. 6439–6448.
- [138] Harald Martens et al. “Pre-whitening of data by covariance-weighted pre-processing”. In: *Journal of Chemometrics: A Journal of the Chemometrics Society* 17.3 (2003), pp. 153–165.
- [139] Neha Wadia et al. “Whitening and second order optimization both make information in the dataset unusable during training, and can reduce or prevent generalization”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 10617–10629.
- [140] Zenggang Xiong et al. “Dynamic texture classification based on 3D ICA-learned filters and fisher vector encoding in big data environment”. In: *Journal of Signal Processing Systems* 94.11 (2022), pp. 1129–1143.
- [141] K Venkata Siva Kumar, Kopparthi Venkata Reddy, and Samrat L. Sabat. “Hardware-Software Co-Design for Whitening using Zero-Phase Component Analysis”. In: *Proc. 2022 IEEE International Symposium on Smart Electronic Systems (iSES)*. 2022, pp. 36–41. DOI: 10.1109/iSES54909.2022.00019.
- [142] Jian Zeng et al. “A review of the discriminant analysis methods for food quality based on near-infrared spectroscopy and pattern recognition”. In: *Molecules* 26.3 (2021), p. 749.
- [143] Thierry Bouwmans et al. “On the applications of robust PCA in image and video processing”. In: *Proceedings of the IEEE* 106.8 (2018), pp. 1427–1457.
- [144] Fernando De la Torre and Michael J Black. “Robust principal component analysis for computer vision”. In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. Vol. 1. IEEE. 2001, pp. 362–369.

- [145] Alex Pentland. “Looking at people: Sensing for ubiquitous and wearable computing”. In: *IEEE Transactions on Pattern analysis and machine intelligence* 22.1 (2000), pp. 107–119.
- [146] Maxim A Grudin. “On internal representations in face recognition systems”. In: *Pattern recognition* 33.7 (2000), pp. 1161–1177.
- [147] Arnaz Malhi and Robert X Gao. “PCA-based feature selection scheme for machine defect classification”. In: *IEEE transactions on instrumentation and measurement* 53.6 (2004), pp. 1517–1525.
- [148] Irem Ersöz Kaya et al. “PCA based clustering for brain tumor segmentation of T1w MRI images”. In: *Computer methods and programs in biomedicine* 140 (2017), pp. 19–28.
- [149] Thanit Karnthak and Pinit Kumhom. “A hardware implementation of PCA based-on the networks-on-chip paradigm”. In: *2012 International Symposium on Communications and Information Technologies (ISCIT)*. IEEE. 2012, pp. 834–839.
- [150] Abhishek Das et al. “An efficient FPGA implementation of principle component analysis based network intrusion detection system”. In: *Proceedings of the conference on Design, automation and test in Europe*. 2008, pp. 1160–1165.
- [151] Uday A Korat and Amirhossein Alimohammad. “A reconfigurable hardware architecture for principal component analysis”. In: *Circuits, Systems, and Signal Processing* 38 (2019), pp. 2097–2113.
- [152] Mathias Schellhorn and Gunther 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)*. IEEE. 2018, pp. 1–6.
- [153] Mario Vestias and Horácio Neto. “Trends of CPU, GPU and FPGA for high-performance computing”. In: *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE. 2014, pp. 1–6.

- [154] Christian Brugger et al. “A quantitative cross-architecture study of morphological image processing on CPUs, GPUs, and FPGAs”. In: *2015 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*. IEEE. 2015, pp. 201–206.
- [155] Sakari Lahti et al. “Are we there yet? A study on the state of high-level synthesis”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.5 (2018), pp. 898–911.
- [156] Ali Asghar Sharifi Najafabadi and Farah Torkamani Azar. “Removing redundancy data with preserving the structure and visuality in a database”. In: *Signal, Image and Video Processing* 13.4 (2019), pp. 745–752.
- [157] Ali Amine Ait Si et al. “Hardware PCA for gas identification systems using high-level synthesis on the Zynq SoC”. In: *Proc. 2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS)*. IEEE. 2013, pp. 707–710.
- [158] Mohammad Amir Mansoori and Mario R Casu. “High Level Design of a Flexible PCA Hardware Accelerator Using a New Block-Streaming Method”. In: *Electronics* 9.3 (2020), p. 449.

Design and Implementation of Matrix Algebra Accelerators on FPGA for Signal Processing Applications using High-Level Synthesis

by Venkata Siva Kumar K

Submission date: 02-Sep-2025 02:44PM (UTC+0530)

Submission ID: 2740114247

File name: Venkata_Siva_Kumar_K.pdf (4.66M)

Word count: 25373

Character count: 127895

Scanned
2-9-2025

Design and Implementation of Matrix Algebra Accelerators on FPGA for Signal Processing Applications using High-Level Synthesis

ORIGINALITY REPORT

31%

SIMILARITY INDEX

14%

INTERNET SOURCES

29%

PUBLICATIONS

4%

STUDENT PAPERS

PRIMARY SOURCES

1

Venkata Siva Kumar K, Venkata Reddy Kopparthi, Samrat L. Sabat. "Hardware-Software Co-Design for Whitening using Zero-Phase Component Analysis", 2022 IEEE International Symposium on Smart Electronic Systems (iSES), 2022

Publication

9%

2

link.springer.com

Internet Source

6%

3

Venkata Siva Kumar Kokkiligadda, Vijitha Naikoti, Gaurao Sunil Patkotwar, Samrat L. Sabat, Rangababu Peesapati. "FPGA-Based Hardware Accelerator for Matrix Inversion", SN Computer Science, 2023

Publication

5%

4

K Venkata Siva Kumar, Venkata Reddy Kopparthi, Samrat L. Sabat, Thulasiram Varma.K, Rangababu Peesapati. "System on chip implementation of floating point matrix inversion using modified Gram-Schmidt based QR decomposition on PYNQ FPGA", 2021 IEEE International Symposium on Smart Electronic Systems (iSES), 2021

Publication

4%

5

Submitted to University of Hyderabad, Hyderabad

1%

SI. NOS 1 to 4 above are candidate's own work. Excluding these four sources, the plagiarism score will be less than 10%. Samrat L. Sabat 2-9-2025

Samrat L. Sabat
2-9-2025

Prof. SAMRAT L. SABAT
Centre for Advanced Studies in
Electronic Sciences & Technology (CAEST)
School of Physics, University of Hyderabad
Gachowli, Hyderabad-500 046. IN

6	repositorio.unal.edu.co Internet Source	1 %
7	fenix.tecnico.ulisboa.pt Internet Source	<1 %
8	Leslie Hogben, Richard Brualdi, Anne Greenbaum, Roy Mathias. "Handbook of Linear Algebra", CRC Press, 2006 Publication	<1 %
9	necst.it Internet Source	<1 %
10	www.duo.uio.no Internet Source	<1 %
11	Submitted to Utah Education Network Student Paper	<1 %
12	www.escholar.manchester.ac.uk Internet Source	<1 %
13	dx.doi.org Internet Source	<1 %
14	Mi Lu, XianGzhen Qiao, Guanrong Chen. "Parallel computation of the modified extended kalman filter", International Journal of Computer Mathematics, 1992 Publication	<1 %
15	Submitted to Universidade do Porto Student Paper	<1 %
16	eprints.usq.edu.au Internet Source	<1 %
17	Submitted to BENEMERITA UNIVERSIDAD AUTONOMA DE PUEBLA BIBLIOTECA Student Paper	<1 %

18	vitela.javerianacali.edu.co Internet Source	<1 %
19	Submitted to Universidad Nacional de Colombia Student Paper	<1 %
20	hal.archives-ouvertes.fr Internet Source	<1 %
21	apo-tokyo.org Internet Source	<1 %
22	d-nb.info Internet Source	<1 %
23	www.docdroid.net Internet Source	<1 %
24	Santanu Saha Ray. "Numerical Analysis with Algorithms and Programming", CRC Press, 2018 Publication	<1 %
25	dokumen.pub Internet Source	<1 %
26	dspace.tul.cz Internet Source	<1 %
27	Francesco Ratto, Ángela Porras Máinez, Carlo Sau, Paolo Meloni et al. "An Automated Design Flow for Adaptive Neural Network Hardware Accelerators", Journal of Signal Processing Systems, 2023 Publication	<1 %
28	xilinx.github.io Internet Source	<1 %
29	"2019 6th International Conference on Signal Processing and Integrated Networks (SPIN)",	<1 %

2019 6th International Conference on Signal Processing and Integrated Networks (SPIN), 2019

Publication

30	ia903006.us.archive.org Internet Source	<1 %
31	Srinivasan Gopalakrishnan. "Wave Propagation in Materials and Structures", CRC Press, 2019 Publication	<1 %
32	Submitted to University of Glasgow Student Paper	<1 %
33	faculty.csu.edu.cn Internet Source	<1 %
34	hdl.handle.net Internet Source	<1 %
35	lup.lub.lu.se Internet Source	<1 %
36	webthesis.biblio.polito.it Internet Source	<1 %
37	"A hardware/software co-design case study on MPEG AAC audio decoder", International Journal of Embedded Systems, 2005 Publication	<1 %
38	Ricardo Lent. "Towards Cognitive Networking using FPGA-Based Acceleration", IEEE Journal of Radio Frequency Identification, 2021 Publication	<1 %
39	Submitted to University of Greenwich Student Paper	<1 %
40	irgu.unigoa.ac.in Internet Source	<1 %

41	www.wiseguyreports.com Internet Source	<1 %
42	Andrei D. Polyanin, Alexander V. Manzhirov. "Handbook of Mathematics for Engineers and Scientists", Chapman and Hall/CRC, 2019 Publication	<1 %
43	Submitted to Ho Chi Minh University of Technology and Education Student Paper	<1 %
44	Jeff Calder, Peter J. Olver. "Linear Algebra, Data Science, and Machine Learning", Springer Science and Business Media LLC, 2025 Publication	<1 %
45	Nazarov, M.. "A mixed hook-length formula for affine Hecke algebras", European Journal of Combinatorics, 200411 Publication	<1 %
46	Submitted to University of Lincoln Student Paper	<1 %
47	Submitted to University of Stellenbosch, South Africa Student Paper	<1 %
48	pea.lib.pt.ehu Internet Source	<1 %
49	people.math.ethz.ch Internet Source	<1 %
50	Longju Liu, Jie Yang, Mei Wu, Jinlu Liu, Wei Huang, Yang Li, Bingjie Xu. "A Post-Processing Method for Quantum Random Number Generator Based on Zero-Phase Component Analysis Whitening", Entropy, 2025 Publication	<1 %

51

Xiaokun Yang. "Integrated Circuit Design - IC Design Flow and Project-Based Learning", CRC Press, 2024

Publication

<1%

52

ntrs.nasa.gov

Internet Source

<1%

Exclude quotes On

Exclude matches < 14 words

Exclude bibliography On