

# **HARDWARE ACCELERATION OF MATRIX MAC PROCESSOR ON FPGA**



**Session: BSc. Spring 2024**

**Project Supervisor: Fahad Bin Muslim**

**Submitted By:**

**Saad Khan (Team Lead)**

**Mahnoor Maleeka**

**Syed Zaeem Shakir**

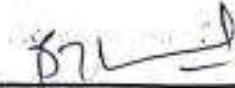
---

**Faculty of Computer Science and Engineering**

**Ghulam Ishaq Khan Institute of Engineering Sciences &  
Technology**

## Certification

This is to certify that Saad Khan, 202414 and Mahnoor Maleeka, 2020217 and Syed Zaem Shakir, 2020487 have successfully completed the final project **Hardware Acceleration of MATRIX MAC Processor on FPGA**, at the Ghulam Ishaq Khan Institute of Engineering Sciences & Technology to fulfill the partial requirement of the degree **Computer Engineering**.



**External Examiner**

Dr. Salman Ahmed

Assistant Professor,  
Computer Systems Engineering,  
UET Peshawar



**Project Supervisor**

Dr. Fahad Bin Muslim

Assistant Professor,  
Computer Engineering  
GIKI, Topi.



**Head of the Department**

Department of Computer Engineering and Data Science, GIK

Head of  
Computer Engineering & Data Science  
FCSE, GIK Institute of Engineering  
Science and Technology, Topi

## **Abstract**

Increased demands for computer hardware accelerators that speed up computational tasks are coming from different domains, where the degree of complexity is massive, e.g., artificial intelligence, scientific computing, or data analytics. This thesis covers the hardware implementation of a RISC-V-based processor and of the accelerator units that will run on it, which are designed to speed up a particular type of matrix operations. By exploiting the capabilities offered by Field Programmable Gate Array (FPGA) platforms, Intellera can focus on addressing the performance issues tied to matrix operations with a view of offloading the CPU-intensive portion to a customized hardware accelerator.

Our project starts with a detailed architecture design examination of the processor accompanied with acceleration hardware techniques, which consequently facilitates the design of the Intellera processor. The crucial elements of the processor including the custom instruction set architecture (ISA), are efficiently produced with pipelining design techniques and hardware accelerators using hardware description languages like Verilog for their implementation.

In conclusion, the development of Intellera is the high point of the design in the sphere of processor hardware acceleration. The successful development and testing of the Intellera processor set the stage for the research and creativity of the coming-age technology of hardware-accelerated computing for the years to come. Strides in efficiency, collaboration, development, and eventually deployment in real life are marked, adding all systems together to form a basis for future progress in high-performance computing architecture.

## Undertaking

I certify that the project **Hardware Acceleration of MATRIX MAC Processor on FPGA** is our own work. The work has not, in whole or in part, been presented elsewhere for assessment. Where material has been used from other sources it has been properly acknowledged/ referred.



---

Saad Khan

2020414



---

Mahnoor Maleeka

2020217



---

Syed Zaeem Shakir

2020487

## Acknowledgement

Hereby, we would like to extend our heartfelt gratitude to Dr. Fahad bin Muslim, who provided essential support and informative mentoring during this project process. Leadership, persistence, and inspiration are the integrals of my research and unique contributions to outcomes.

Our thanks also go to the faculty and administration of the Ghulam Ishaq Khan Institute of Engineering Sciences and Technology (GIKI) for their help and crucial support in learning and in research.

In addition to that, we are grateful to Mr. Kashif for also contributing the same through valuable knowledge and information on the possibility to build an Intellera processor development. His achievements, professional acumen, and co-existence had vastly expanded the scope of our project and improved the social value.

We would also like to thank all our colleagues who come forward to volunteer their time, talent, and dedication for the project's completion. Our devotion and teamwork have become the resource that enables us to handle any challenges and accomplish the tasks given.

Finally, we will express gratitude to our families and friends for their unwavering support, firm attitude and constant inspiration in the whole trying time. Our parents have always been there to support us.

Working as a unit was the fundamental part that enabled the project to accomplish its purpose. Many people also were involved in this work. We wholeheartedly glory in the indispensable efforts of every stakeholder who in one way or another helped to the greatness of this project.

## Table of Contents

|   |     |
|---|-----|
| Session: BSc. Spring 2024.....  | 1   |
| Abstract.....   | ii  |
| Undertaking.....  | iii |
| Acknowledgement .....   | iv  |
| Chapter 1: Introduction.....  | 10  |
| 1.1 Background of the Project: .....                                  | 10  |
| 1.1.1 CISC and RISC Architectures.....                                | 10  |
| 1.1.2 GPU and TPU Technologies .....                                  | 10  |
| 1.2 Motivation Behind the Project:.....                               | 11  |
| 1.3 Scope of Intellera: .....   | 12  |
| 1.4 Objectives: .....   | 12  |
| Chapter 2: Literature Survey.....                                     | 14  |
| 2.1 Evolution of RISC-V Architecture: .....                           | 14  |
| 2.2 Advancements in Pipeline Architectures: .....                     | 14  |
| 2.3 Matrix MAC Units and Their Integration in Processors:.....        | 14  |
| 2.4 FPGA Implementation and Its Strategic Importance:.....            | 15  |
| 2.5 Challenges and Opportunities in FPGA-Based Processor Design:..... | 15  |
| 2.6 Study of Hazard Units in Pipelined Processors: .....              | 15  |
| Chapter 3: Design (Systems Requirements/Specifications) .....         | 16  |
| 3.1 Systems Requirements:.....  | 16  |
| 3.1.1 Functional Requirements: .....                                  | 16  |
| 3.1.2 Functional Requirements with Traceability information .....     | 17  |
| 3.1.3 Non-functional Requirements .....                               | 19  |
| 3.2 System Specifications: .....                                      | 20  |
| 3.2.1 Architectural Design .....                                      | 20  |
| 3.2.2 Intellera Development View .....                                | 21  |
| 3.2.3 Hardware Interface.....   | 21  |
| 3.2.4 Software Interfaces .....                                       | 23  |

|   |    |
|---|----|
| Chapter 4: Proposed Solution (Methodology, Implementation) .....      | 23 |
| 4.1 Control Unit: .....   | 24 |
| 4.2 Single Cycle Processor: .....                                     | 25 |
| 4.3 Five Staged Pipelined Processor: .....                            | 26 |
| 4.4 Control Hazard Unit: .....  | 26 |
| 4.5 MAC Module: .....   | 27 |
| 4.5.1 Custom Instruction Set for Matrix Manipulation (32 bits): ..... | 28 |
| 4.5.2 Supported Instruction Set and Functionalities: .....            | 29 |
| 4.6 MAC Decoder: .....  | 30 |
| 4.6.1. Inputs: .....  | 30 |
| 4.6.3. Decoding Logic: .....  | 31 |
| 4.6.4. Integration with Control Unit: .....                           | 31 |
| 4.6.5. Key Roles in Matrix Operations: .....                          | 31 |
| 4.7 Register File: .....  | 31 |
| 4.8 Control Unit for Matrix Operations: .....                         | 32 |
| 4.9 Data Memory: .....  | 32 |
| 4.10 Five Stage Pipelined Processor with Matrix MAC: .....            | 33 |
| 4.11 UART Transceiver with Matrix MAC: .....                          | 34 |
| Chapter 5: Results and Discussion .....                               | 35 |
| 5.1 Benchmarking Tests Setup: .....                                   | 35 |
| 5.2 Performance Evaluation: .....                                     | 36 |
| 5.2.1 Resource Utilization .....                                      | 36 |
| 5.2.2 Operational Frequency .....                                     | 36 |
| 5.2.3 Power Efficiency .....  | 36 |
| 5.2.4 Throughput Analysis .....                                       | 36 |
| 5.2.5 Latency Measurements .....                                      | 37 |
| Chapter 6: Conclusion and Future Work .....                           | 38 |
| 6.1 Conclusion: .....   | 38 |
| 6.2 Achievements: .....   | 38 |
| 6.3 Lessons Learned: .....  | 38 |

|  |    |
|--|----|
| 6.4 Future Work: .....                                     | 39 |
| 6.4.1 Implementing MMU as an In-built System .....         | 39 |
| 6.4.2 Expansion to System-on-Chip (SoC) Architecture ..... | 39 |
| 6.4.3 Broadening the Custom Instruction Set .....          | 39 |
| GLOSSARY .....   | 40 |
| REFERENCES .....   | 42 |
| APPENDIX A .....   | 44 |
| APPENDIX B .....   | 45 |



## List of Tables

|   |    |
|---|----|
| Table 3.1 Functional Requirement 1 .....                            | 17 |
| Table 3.2 Functional Requirement 2 .....                            | 18 |
| Table 3.3 Functional Requirement 3 .....                            | 18 |
| Table 3.4 Functional Requirement 4 .....                            | 19 |
| Table 3.5 Functional Requirement 5 .....                            | 19 |
| Table 4.1: Modified MAC ISA .....                                   | 29 |
| Table 5.1: Benchmarking and Testing of Matrix MAC Instr. ....       | 35 |
| Table 5.2 Performance Metrics before and after Matrix MAC Unit..... | 36 |
| Table A.1 RISC-V ISA.....   | 44 |

## List of Figures

|   |    |
|---|----|
| Figure 1.1: GPU Architecture .....  | 11 |
| Figure 3.1 Architectural Design of Complete Processor .....   | 20 |
| Figure 3.2 Processor Development Stages.....  | 21 |
| Figure 4.1: Control Unit with ALU Decoder .....   | 24 |
| Figure 4.2: Complete Single Cycle Processor Architecture.....   | 25 |
| Figure 4.3: Complete 5 Staged Pipelined Processor Architecture .....  | 26 |
| Figure 4.4: Complete 5 Staged Pipelined Processor with Hazard Unit Architecture                               | 26 |
| Figure 4.5: Internal Architecture of Matrix MAC Unit .....  | 27 |
| Figure 4.6: Modified MAC Decoder .....  | 30 |
| Figure 4.7: Modified Control Unit.....  | 32 |
| Figure 4.8: Complete 5 Staged Pipelined Processor with Hazard & Matrix MAC Unit<br>Architecture.....          | 33 |
| Figure 4.9: UART Interfacing.....   | 34 |
| Figure 5.1 Behavioral simulation of MAC Instructions, Testing Addition and Multiplication of<br>Matrices..... | 35 |

# Chapter 1: Introduction

Nowadays, the ongoing extraordinarily fast growth in computational requirements throughout the different fields faced technology providers with the main challenge of finding fast, effective solutions. Processors that can power complex AI algorithms and computational tasks are now more in demand as the need for high performance computing machines that can-do real-time processing is now being exhausted. In response to the rise of such multimedia applications together, Intellera project is born for a pioneering venture in designing and building a high-speed RISC-V-based processor with special-purpose hardware accelerators optimized for matrix manipulation.

## *1.1 Background of the Project:*

### 1.1.1 CISC and RISC Architectures

Complex Instruction Set Computing (CISC) and Reduced Instruction Set Computing (RISC) represent two fundamental approaches to processor design. CISC architectures, like those found in x86 processors, are characterized by a wide range of complex instructions allowing for more functions within a single instruction. This complexity can lead to slower performance in some cases due to the intricate decoding required. In contrast, RISC architectures streamline operations by focusing on a smaller set of instructions, which can be executed more rapidly, providing advantages in power efficiency and performance predictability, particularly useful in embedded systems [1].

### 1.1.2 GPU and TPU Technologies

Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) have revolutionized areas requiring massive parallelism such as graphics rendering and machine learning. GPUs, at first developed for use in imaging, turned out to be the saviors of general-purpose computing because they are designed in such a way that the processor can take multiple calculations at a given time. While TPUs, on the other hand, these are geared accelerators from Google, which are specially designed in order to speed up tensor operations within neural network environments, providing

orders of magnitude of a performance gain with regard to processing times and power consumption as compared to the traditional CPU processors [2].

*1.2 Motivation Behind the Project:*

The primary motivation for the Intellera project is rooted in the limitations of traditional computational architectures when dealing with specific applications such as AI and large-scale data analytics. In terms of the market, high-priced TPU and GPU chips are being used over-

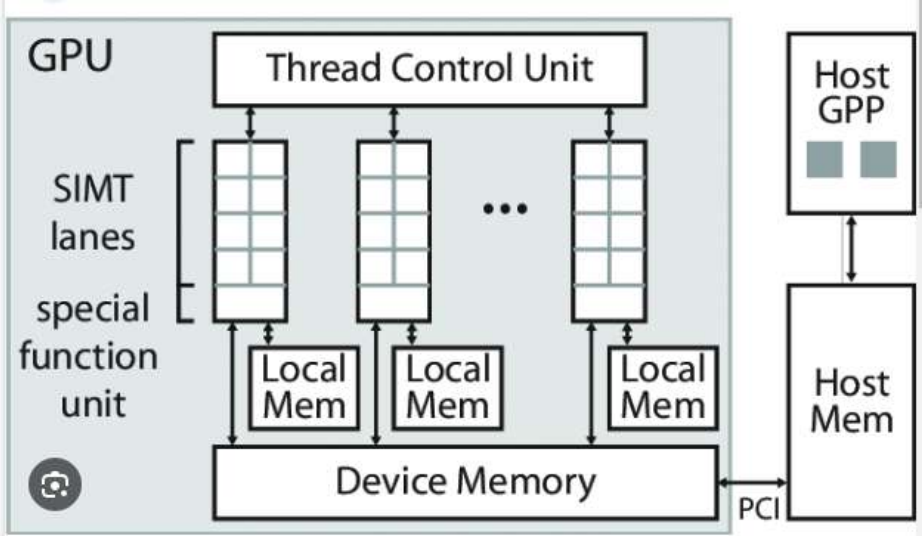


Figure 1.1: GPU Architecture

precatory, therefore creating room for a specific and integrated approach. With the employment of matrix processing features right on-chip a RISC-V based processor, there is the chance to crucially decrease power consumption and pricing, making data centers available for the more compact, portable and resource-saving environments.

The following is also crucial, which is the memory access patterns, and their association with matrix operations, which require high memory efficiency. Inefficient memory routes can cause capacity loads that will result in lowering down the performance. Matrix assignments might constitute a headache for processors in a row, because these can stall machine due to lack of efficient memory access. As a result, processors will need more time to perform operations, and the efficiency overall would decline. Besides, energy efficiency is one of the major parameter in modern processor design, contemporaneous processor considered to be fast but however its normal operation will consume excessive power under heavy computational like matrix multiplication. Such limitations indicate the imperativeness of designing structures which seek speeding up

memory locality, energy efficiency, scalability, and flexibility to meet the expected challenges in the coming times.

### *1.3 Scope of Intellera:*

The scope of the Intellera project is both ambitious and technically demanding. Such a development includes the making of a RISC-V processor with a built-in Matrix MAC unit in its inner make-up framework. This integration seeks to increase a processor's capabilities to make specific progress on mission-critical tasks like matrix multiplication, the backbone of artificial intelligence and machine learning processes. The project use FPGA for sample and optimization processor architecture. Among the most advantageous qualities of the FPGAs are their ability to be reconfigured in a hardware fashion and rapid prototyping of designs that are utilized individually to enhance performance and resource utilization; thus these devices prove convenient for iterative design processes.

Further on, the technical design and implementation come with validation and testing phases in large scope. These stages are crucial disciplines to be understood well that the processor can match its theoretical performances and at the same time operate reliably under real time applications. We will perform thorough testing, covering a range of manufacturing scenarios in order to reveal and eliminate any false operating of the processor. Finally, designing a chip with enough functionality to fulfill industry needs is the main goal which should be achieved through device design that is high-performance, energy-efficient and reliable enough to be used in numerous electronic products.

### *1.4 Objectives:*

The objectives of the Intellera project are various:

**Architectural Design and Enhancement:** To design a 5-stage pipelined 32-bit RISC-V processor that can add a matrix MAC unit for matrix operation and a hazard detection unit that helps to manage the data dependence.

**FPGA Implementation:** Building the required processor and placing it onto an FPGA platform for trial periods, hence it could be modularized and real-time performance tuning purposes.

**Performance Optimization:** To deliver an operating unit that is able to do something a CPU, GPU and TPU setup can do- yet with reduced power consumption and operation latency.

**Comprehensive Validation:** To conduct extensive testing to ensure stability, accuracy, and efficiency of the processor under various computational loads.



## Chapter 2: Literature Survey

In the chapter on literature survey from Intellera project, a thorough review of latest research papers, publications, and innovations in RISC-V (Reduced Instruction Set Computer) architecture, accelerators for matrix manipulation using hardware by FPGA, and relevant technologies will be presented thereby. The first chapter is meant to define the theoretical scheme, highlight specific innovations and progress, critique the approaches and methods which have already been developed, and to find the void spaces or distinct areas for future work.

### *2.1 Evolution of RISC-V Architecture:*

The RISC-V architecture has rapidly become a focal point for innovation in processor design, offering a license-free ISA that enables broad modifications and customizations. Studies like that of Dennis et al. [3] illustrate the practical applications of RISC-V in embedded systems, demonstrating how it can be implemented to achieve significant efficiencies on FPGA platforms. Similarly, Malone et al. [4] discuss the adoption of RISC-V in radiation-tolerant FPGAs, highlighting its potential for space applications where robustness and reliability are paramount. The adaptability and the fact that it can be used for many purposes that have limited costs are the essential features of RISC-V that are critical for narrow down efforts like Intellera where we aim to push the boundaries of advanced computational abilities but still keep the costs relatively low.

### *2.2 Advancements in Pipeline Architectures:*

Pipelined architecture design plays a great role in nowadays processors, so to improve the overall processor performance. The path which starts with simple pipelines, as explained in Qingxin and Li [5], direct pipelines with more levels yield better result especially for custom applications such as embedded systems. A 5 stage pipeline would be a good implementation in RISC-based processors which greatly enhance the speed of processing the instructions so that it wouldn't take so long for those that require high computational resource with minimal latency.

### *2.3 Matrix MAC Units and Their Integration in Processors:*

The capability to incorporate MAC units straight into the computation units is a major improvement by far, as it makes it possible to complete computation tasks involving complex math operations much faster. Krishna [6] undergoes an in-depth review of MAC designs implemented with trees like Booth tree and Wallace, underline how these designs consume power and alter

performance in FPGA chips. The incorporation of such units is indeed both mostly essential and advantageous in digital signal processing and artificial intelligence, where matrix operations are usually rapid and efficient. This coincides with project's Intellera goal to review the default setting in RISC-V processor configuration thus bypassing the need for expensive external accelerators like the GPU and TPU.

#### *2.4 FPGA Implementation and Its Strategic Importance:*

FPGA technology is the most important tool in the design and detection of innovative processors given the fact that it is multi-purpose and very efficient ( indicates that this is a permanent trend). The work [7] of Kamaleldin on a modular RISC-V many-core architecture for FPGA accelerators shows the optimized performance of FPGAs in option minimization, cost reduction, and system flexibility by means of scaling up the number of cores. It is vital to the Intellera project as it is the FPGA technology that is essential for the prototype and refinement of the processing design of the processor which ensures an always-optimized performance and the ability to adapt.

#### *2.5 Challenges and Opportunities in FPGA-Based Processor Design:*

From the point of view of Jacobsen et al. [8] in their analysis of the RIFFA framework, the introduction of engines of processors into FPGAs creates new possibilities and problems. This is the foundation of the FPGA integrative development block, which successfully connects FPGA accelerators to the traditional computing systems overcoming the gap of the performance readiness of the integrated system. Insights from such research can actually be priceless for the Intellera project as they help to design a processor that not only caters to predetermined performance criteria but also can be later easily integrated into already built up technology ecosystems.

#### *2.6 Study of Hazard Units in Pipelined Processors:*

The detection units of hazards are of fundamental importance to accomplishing productive flow processors as they detect and rectify data, control, and structural hazards that might compromise, processors' performance. In addition, Navin et al, come up with an innovative technique which uses a co-processor to do away with the control hazards, an effort aimed at boosting the pipeline's efficiency [9].



## Chapter 3: Design (Systems Requirements/Specifications)

In this Chapter we consider the detailed design of the Intellera system, with a focus on the systems requirements and specifications that are necessary for development and implementation of the system. In the design phase of the hardware model, the focus is on the architecture of hardware, software integration, and performance targets, enabling matrix manipulation to be efficient on the FPGA-based platform.

### *3.1 Systems Requirements:*

Systems specifications for the Intellera project state the system component and functionality as well as the expected performance metrics to be met to realize the objectives. These stipulations serve as a framework for the design and development phases.

#### 3.1.1 Functional Requirements:

- 1) **Basic RISC-V ISA:** Intellera processor is required to execute the simple “RISC-V instructions” adhering to the RISC-V instruction set architecture rules.
- 2) **Custom ISA:** In order to do that, the entire system will be made from a processor specially designed for a particular instruction set architecture (ISA), where matrix operations are the primary focus.
- 3) **Matrix Multiplication:** The processor needs to be so fast in its respective operations for the matrix multiplication instruction being inputted.
- 4) **Matrix Addition:** It should also perform matrix addition precisely, which involves computations according to implied derivatives.
- 5) **Register File:** Register file should be present in order to address memory requirements of data and results during execution of instructions.
- 6) **Data Memory Access:** Memory will support processor with ‘read’ and ‘write’ commanding operations indication. It should be provided with a mechanism to resolve the memory access conflict hazards.
- 7) **Arithmetic and Logic Unit (ALU):** An ALU is needed to implement only the operations of the RISC-V and the Matrix Multiplication instructions included in Verilog.
- 8) **Control Unit:** The control unit must produce instructions, develop the flow of instructions and execute them, including branch and jump instructions.

9) **Compatibility:** The processor (the RV32I ISA standard) must be compliant with the RISC-V instruction set. On the other hand, it must also include specific Matrix MAC instructions.

10) **Hazard Unit:** In the valuator there should be the Hazard Unit that would settle all possible hazards. Among the mentioned errors, control, Structure and Data Hazards stand out as the ones that are the most commonly occurring types of errors.

11) **Pipelining:** The processor shall have at least five pipeline stages, bunch of them, such as fetch, decode, execute, and memory write-backs, so as to better complete instructions.

12) **FPGA Implementation:** The hardware design of the processor and its components need to be such that it act together with the FPGA mechanism capitalizing on the latter's hardware prototyping advantages.

### 3.1.2 Functional Requirements with Traceability information

Table 3.1 Functional Requirement 1

|                                |  |     |                         |     |                        |     |                   |    |    |
|--------------------------------|--|-----|-------------------------|-----|------------------------|-----|-------------------|----|----|
| <b>Requirement ID</b>          | 01   |     | <b>Requirement Type</b> |     | Functional             |     | <b>Use Case #</b> |    | 00 |
| <b>Status</b>                  | <b>New</b>   | yes | <b>Agreed-to</b>        | yes | <b>Baselined</b>       | yes | <b>Rejected</b>   | No |    |
| <b>Parent Requirement #</b>    | N/A  |     |                         |     |                        |     |                   |    |    |
| <b>Description</b>             | The processor must execute basic RISC-V instructions, adhering to the RISC-V ISA.  |     |                         |     |                        |     |                   |    |    |
| <b>Rationale</b>               | This requirement ensures compatibility with the standard RISC-V instruction set, forming the foundation for further customizations |     |                         |     |                        |     |                   |    |    |
| <b>Source</b>                  |  |     |                         |     | <b>Source Document</b> |     | -                 |    |    |
| <b>Acceptance/Fit Criteria</b> | Successful execution of standard RISC-V instructions.  |     |                         |     |                        |     |                   |    |    |
| <b>Dependencies</b>            | None   |     |                         |     |                        |     |                   |    |    |
| <b>Priority</b>                | <b>Essential</b>   | yes | <b>Conditional</b>      | no  | <b>Optional</b>        | No  |                   |    |    |
| <b>Change History</b>          | None   |     |                         |     |                        |     |                   |    |    |

Table 3.2 Functional Requirement 2

|                                |  |     |                         |            |                        |     |                   |    |  |
|--------------------------------|--|-----|-------------------------|------------|------------------------|-----|-------------------|----|--|
| <b>Requirement ID</b>          | 02   |     | <b>Requirement Type</b> | Functional |                        |     | <b>Use Case #</b> | 00 |  |
| <b>Status</b>                  | <b>New</b>   | yes | <b>Agreed-to</b>        | yes        | <b>Baselined</b>       | yes | <b>Rejected</b>   | No |  |
| <b>Parent Requirement #</b>    | 01   |     |                         |            |                        |     |                   |    |  |
| <b>Description</b>             | The processor will support a custom instruction set architecture (ISA) optimized for matrix operations.        |     |                         |            |                        |     |                   |    |  |
| <b>Rationale</b>               | Custom instructions tailored for matrix operations will improve performance and efficiency for specific tasks. |     |                         |            |                        |     |                   |    |  |
| <b>Source</b>                  |  |     |                         |            | <b>Source Document</b> | -   |                   |    |  |
| <b>Acceptance/Fit Criteria</b> | Successful execution of custom Matrix MAC instructions.  |     |                         |            |                        |     |                   |    |  |
| <b>Dependencies</b>            | Requirement #1 (Basic RISC-V ISA).   |     |                         |            |                        |     |                   |    |  |
| <b>Priority</b>                | <b>Essential</b>   | yes | <b>Conditional</b>      | no         | <b>Optional</b>        | No  |                   |    |  |
| <b>Change History</b>          | None   |     |                         |            |                        |     |                   |    |  |

Table 3.3 Functional Requirement 3

|                                |   |     |                         |            |                        |     |                   |    |  |
|--------------------------------|---|-----|-------------------------|------------|------------------------|-----|-------------------|----|--|
| <b>Requirement ID</b>          | 03  |     | <b>Requirement Type</b> | Functional |                        |     | <b>Use Case #</b> | 00 |  |
| <b>Status</b>                  | <b>New</b>  | yes | <b>Agreed-to</b>        | yes        | <b>Baselined</b>       | yes | <b>Rejected</b>   | No |  |
| <b>Parent Requirement #</b>    | 01, 02  |     |                         |            |                        |     |                   |    |  |
| <b>Description</b>             | The processor must efficiently execute matrix multiplication instruction.   |     |                         |            |                        |     |                   |    |  |
| <b>Rationale</b>               | Matrix multiplication is a fundamental operation in various computational tasks and requires optimized execution. |     |                         |            |                        |     |                   |    |  |
| <b>Source</b>                  |   |     |                         |            | <b>Source Document</b> | -   |                   |    |  |
| <b>Acceptance/Fit Criteria</b> | Efficient execution of matrix multiplication with specified performance metrics.                                  |     |                         |            |                        |     |                   |    |  |
| <b>Dependencies</b>            | Requirement #2 (Custom ISA).  |     |                         |            |                        |     |                   |    |  |
| <b>Priority</b>                | <b>Essential</b>  | yes | <b>Conditional</b>      | no         | <b>Optional</b>        | No  |                   |    |  |
| <b>Change History</b>          | None  |     |                         |            |                        |     |                   |    |  |

Table 3.4 Functional Requirement 4

|                                |   |     |                         |            |                        |                   |                 |    |
|--------------------------------|---|-----|-------------------------|------------|------------------------|-------------------|-----------------|----|
| <b>Requirement ID</b>          | 04  |     | <b>Requirement Type</b> | Functional |                        | <b>Use Case #</b> | 00              |    |
| <b>Status</b>                  | <b>New</b>  | yes | <b>Agreed-to</b>        | yes        | <b>Baselined</b>       | yes               | <b>Rejected</b> | No |
| <b>Parent Requirement #</b>    | 02  |     |                         |            |                        |                   |                 |    |
| <b>Description</b>             | It should also perform matrix addition precisely, adhering to specified computational requirements.               |     |                         |            |                        |                   |                 |    |
| <b>Rationale</b>               | Matrix multiplication is a fundamental operation in various computational tasks and requires optimized execution. |     |                         |            |                        |                   |                 |    |
| <b>Source</b>                  |   |     |                         |            | <b>Source Document</b> | -                 |                 |    |
| <b>Acceptance/Fit Criteria</b> | Matrix addition is a common operation in matrix processing tasks and must yield accurate results.                 |     |                         |            |                        |                   |                 |    |
| <b>Dependencies</b>            | Requirement #2 (Custom ISA).  |     |                         |            |                        |                   |                 |    |
| <b>Priority</b>                | <b>Essential</b>  | yes | <b>Conditional</b>      | no         | <b>Optional</b>        | No                |                 |    |
| <b>Change History</b>          | None  |     |                         |            |                        |                   |                 |    |

Table 3.5 Functional Requirement 5

|                                |  |     |                         |            |                        |                   |                 |    |
|--------------------------------|--|-----|-------------------------|------------|------------------------|-------------------|-----------------|----|
| <b>Requirement ID</b>          | 05   |     | <b>Requirement Type</b> | Functional |                        | <b>Use Case #</b> | 00              |    |
| <b>Status</b>                  | <b>New</b>   | yes | <b>Agreed-to</b>        | yes        | <b>Baselined</b>       | yes               | <b>Rejected</b> | No |
| <b>Parent Requirement #</b>    | N/A  |     |                         |            |                        |                   |                 |    |
| <b>Description</b>             | A register file should be available to store and access data and results during instruction execution. |     |                         |            |                        |                   |                 |    |
| <b>Rationale</b>               | Register files provide fast data access, crucial for efficient instruction execution and data storage. |     |                         |            |                        |                   |                 |    |
| <b>Source</b>                  |  |     |                         |            | <b>Source Document</b> | -                 |                 |    |
| <b>Acceptance/Fit Criteria</b> | Reliable storage and retrieval of data using the register file.  |     |                         |            |                        |                   |                 |    |
| <b>Dependencies</b>            | None   |     |                         |            |                        |                   |                 |    |
| <b>Priority</b>                | <b>Essential</b>   | yes | <b>Conditional</b>      | no         | <b>Optional</b>        | No                |                 |    |
| <b>Change History</b>          | None   |     |                         |            |                        |                   |                 |    |

### 3.1.3 Non-functional Requirements

1) Power Efficiency: The design will emphasize on power efficiency with a view of limiting the power consumption of the electronic gadget. As a result, it will be fit to be used as a processor in embedded applications.

2) Latency: The processor should be performing matrix operations and MAC processing as fast as possible in order to reduce latency (for both standard RISC-V and Matrix MAC instructions) to offer real-time processing capabilities.

3) Resource Utilization: FPGA resource efficiency could be improved by optimizing the LUTs, flip-flops, and memory blocks, which are mainly used.

4) Scalability: The architecture incorporated to the processor should be able to act as an enabler for the future advancements envisaged in the implementation or expansion of the instruction set.

Furthermore, this architecture should be tailored to optimally handle accordingly various kinds of matrix sizes and complexities.

5) Performance: Benchmark distribution shall not be inferior to advocated and take the place of matrix arithmetic performance.

6) Compatibility: Where applicable, compatibility with existing software or systems will be maintained.

7) Testing and Verification: Complete testing providing suites and uniformity for correctness and functioning of applications need to put in place. Unit and substitute testing will be used to confirm that code is working properly.

8) Documentation: It has to ensure there is detailed description of the hardware and software components to ease the use and maintenance of software especially by its users.

### 3.2 System Specifications:

The Intellera RISC-V unit utilizes standard RISC-V instructions and supports special operations of matrix for its effectiveness. Moreover, it can be programmed to display special operations which are specific to the matrix of binary operations. It reflects the combination of hardware architecture patterns, software integration structure, and performance criteria to be the basis of the matrix manipulation in a field-programmable gate array (FPGA) platform..

#### 3.2.1 Architectural Design

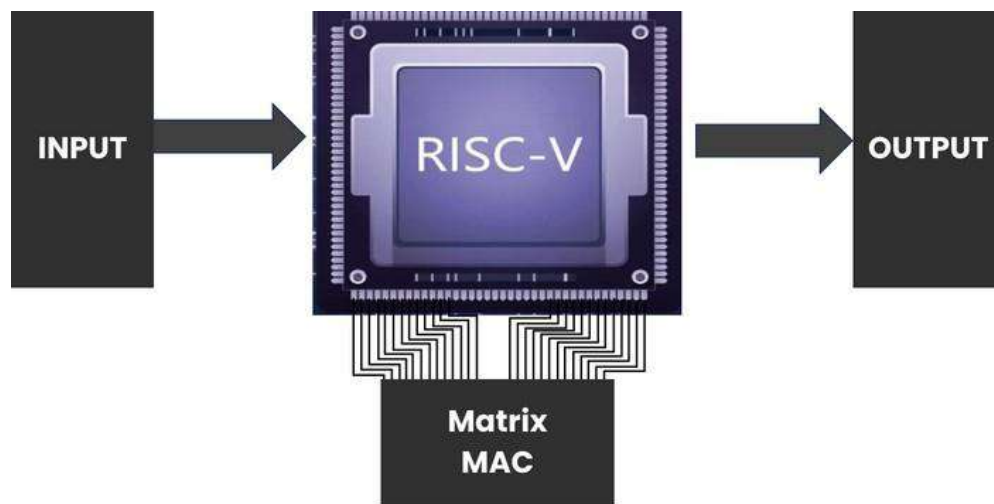


Figure 3.1 Architectural Design of Complete Processor

The Intellera's own system includes a fitting RISC-V processor for fast matrix operations and data flow synchronization as well as memory units to handle matrix data and code for operations. At the heart of it, we use a Matrix Acceleration Unit (MAC Unit). This dedicated hardware, designed

for matrix multiplication, ensures faster performance. The I/O (Input/Output) Unit is an external data exchange facilitator, which helps Intellera to be running alongside other software applications without problematic switching.

3.2.2 Intellera Development View

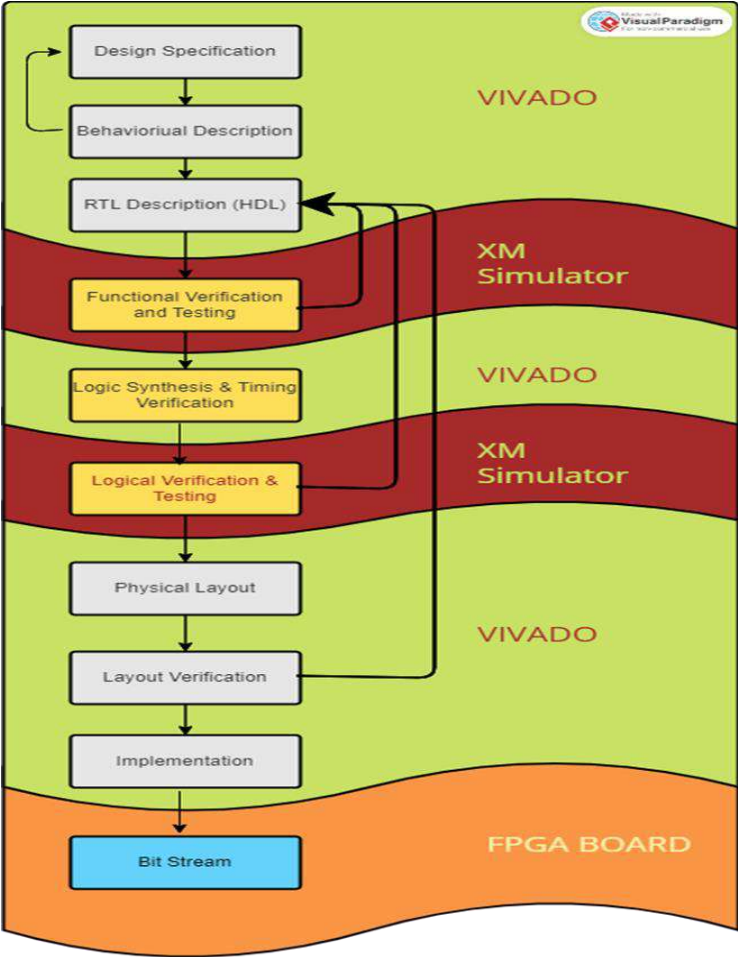


Figure 3.2 Processor Development Stages

The Intellera's instruction with five stages includes creating specifications and behavior description, refining RTL description into detailed one using HDL and validating the RTL simulation through both physical and logical testing. The complexity becomes evident that begins with the mechanism converting the design to logic, and proceeding to the timing and layout validation, which cumulates in the bit stream generation and implementation.

3.2.3 Hardware Interface

The mainstream of our FPGA-based RISC-V processor with a Matrix MAC module software foresee the architecture of the hardware interface at which different hardware components interact

with each other through altered connections. MAIConfigURES the RISC-V processor with the embedded Matrix MAC core and the interfaces to external power and data sources.

### 1. Components:

- **FPGA Board:** This FPGA board is our project's hardware platform, which was chosen for its affordability and availability. It provides the RISC-V class of microprocessors, cores of the MAC Module and other required devices.
- **RISC-V Processor with Matrix MAC Module:** The heart of our project, this integrated unit combines the 32-bit RISC-V processor with the Matrix MAC module. It is responsible for executing instructions, including Matrix MAC operations.
- **Power Supply:** The FPGA board must obtain a steady power supply for the function to work. The device is connected to a power cord so it keeps the device running at optimal performance.
- **Laptop/PC:** The laptop or the PC used for programming is like an interface for the FPGA-based system. It is the programming language that is used for the FPGAs, testing, new experience and analyzing the system.

### 2. Interactions:

- **Data Input:** Input data for processing is provided to the combined RISC-V processor and Matrix MAC module. The data may be generated internally or as external sensors or sources feeding.
- **Data Output:** The data processed represents output by the forum unit joined and it is obtained and later passed to external devices or processing stages.

### 3. Connections:

- **FPGA to Power Supply:** The FPGA board needs that it is connect to a reliable power supply in order to ensure the correct voltage and current level for proper functioning.
- **FPGA to Laptop/PC:** The FPGA board is then linked to the input interface of a computer be it a laptop or a personal computer, either USB or other suitable interfaces. As a result, it enables the utilization of the programming language in debugging as well as data transfer between the board and the development environment.

### *3.2.4 Software Interfaces*

The interface of the software describes the involved tools and software components in the development, programming, testing, and verification of our FPGA-based RISC-V processor with a Matrix MAC module.

#### Tools and Software:

**Vivado:** The Vivado is the tool-box of the application program used for FPGA development. It is the one that helps apply the Verilog code and then synthesize for the RISC-V processor as well as the Matrix MAC module. Also, Vivado can produce a bitstream which will be useful in the programming of the FPGA. Both this process and the Vivado tool itself offer a quick result and a huge advantage in the development of the final device.

**Venus:** Venus represents an online tool employed ensuring RISC-V assembly code productivity. It enables the programmer to write the assembly code for the RISC-V architecture and then convert it into the hexadecimal and/or binary machine code, which can further be used by the user on the modified RISC-V processor designed in the Vivado.

**DigitalJS:** DigitalJS is a software product intended for the development of FPGA-based System algorithms incorporating visual schematics as well. It aids in the design and documentation of the processor's architecture and connections.

## **Chapter 4: Proposed Solution (Methodology, Implementation)**

In this chapter of the document, we explore a solution to the Intellera's project, where we discuss our methodology as well as the setup specifications of the accelerator which is on FPGA for computing matrix MAC operations. The proposed solution for Intellera is progressively illustrated, in a way which takes the discussion from a single-cycle processor to pipelining and then



incorporates MAC unit. Step-by-step details are highlighted, bringing Intellera's process into view as the architecture is explained.

#### 4.1 Control Unit:

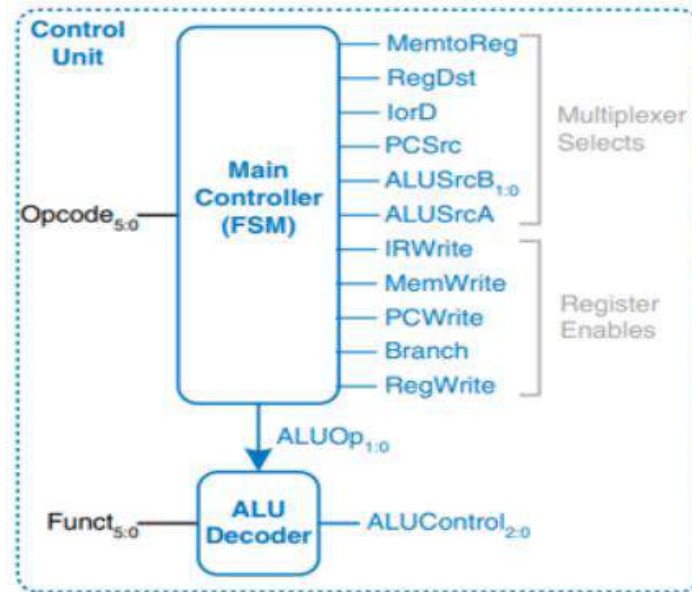


Figure 4.1: Control Unit with ALU Decoder

The first phase of designing the Intellera chip the set chip involves creating the instruction control unit (IA) block responsible for decoding instruction and coordinating the operation of the ALU and memory units. Attached to it there is a FSM (central unit) referred to as the main controller that processes instructions and outputs signals in accordance with the instruction types used in the program sequencing and control flow. The time state diagrams including the delay for each of the subway statuses was suggested as a desired option to attain for the formalism of the logics in the Intellera main controller [10]. This FSM interprets opcodes and routes signals to various operations including instruction memory storage (IRWrite) and the final collation of register destinations (RegDst).

In the context of an 8-bit RISC controller IP core architecture into fetch, decode, execute units, and stage control shows the inevitable need for subdivision of the CU's major responsibilities for its effective functioning [11].

## 4.2 Single Cycle Processor:

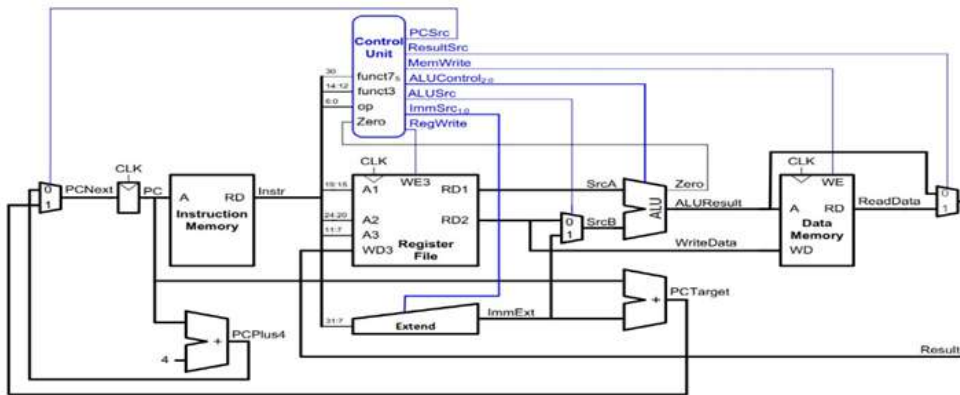


Figure 4.2: Complete Single Cycle Processor Architecture

Furthermore, the proposed by Intellera processor cycle operating on RISC-V instruction set coded on FPGA chip is the new matrix multiplication algorithm acceleration technology. At the core of this architecture lies the ability to complete the stages of instruction processing—fetch, decode, execute, write-back—within a singular clock cycle. In the fetch stage, toggles and the clock signal direct the PC to retrieve instructions from memory, incrementing the PC by four to prepare for the next operation. The decode phase involves the Control Unit deciphering instructions from the IR, with opcodes identifying the specific operation, while the execute stage sees the transmission of control signals to data path components, with the ALU performing computations as instructed. Through this procedure, the final stage, that is the write-back stage, redresses the Register File with the results from either the ALU or from memory, and closes the instruction execution. This streamlined process enables Intellera to handle complex operations like matrix MAC rapidly and efficiently, reflecting advancements in FPGA-based processor design and control unit functionality in microprocessors.

### 4.3 Five Staged Pipelined Processor:

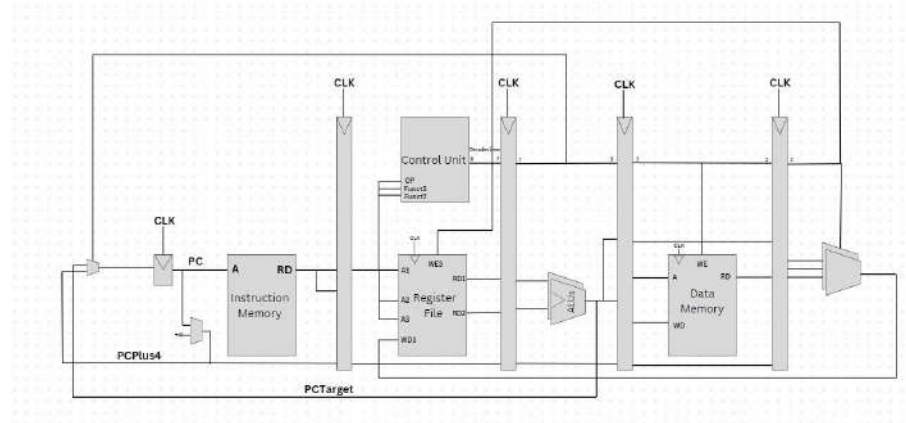


Figure 4.3: Complete 5 Staged Pipelined Processor Architecture

The improved Intellera method makes 5 pipelined stages of correlation between its processor architecture and the one reviewed above. This enables the instruction sets to be executed in a sequence and ensure faster task completion. This setup enables instructions to be processed in different stages of fetching, decoding, reading operands (registers fetching), execution, and writing results (write-back) of each pipeline stage at the same time. Each stage is designed to function in tandem with the next, ensuring continuous operation and efficient use of the ALU, thus significantly enhancing the throughput and reducing the latency involved in processing each instruction. This method ensures that the processor components are utilized effectively, maintaining constant activity and improving overall performance.

### 4.4 Control Hazard Unit:

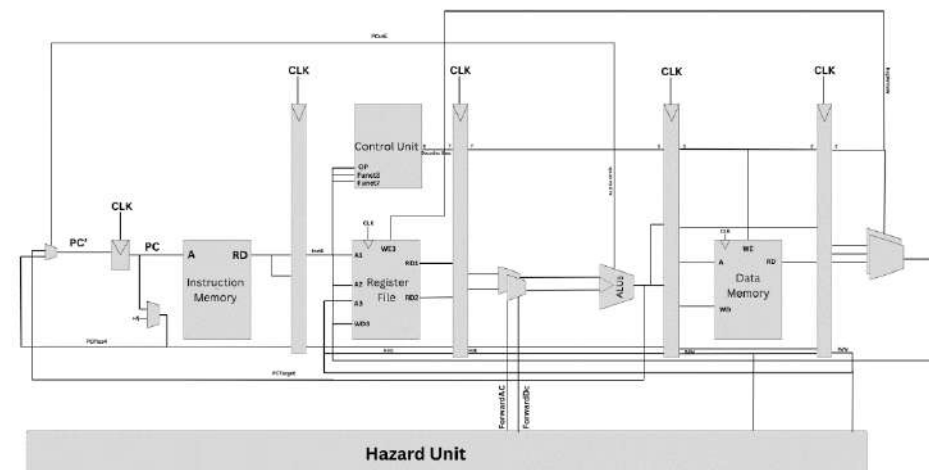


Figure 4.4: Complete 5 Staged Pipelined Processor with Hazard Unit Architecture

In the pipelined design of the Intellera RISC-V processor, each instruction is processed through several consecutive stages, enhancing throughput, and minimizing idle times within the CPU architecture. Unlike a single-cycle design, this pipelined approach allows multiple instructions to be in different stages (fetch, decode, execute, write-back) simultaneously, leveraging an "assembly line" effect that maintains a steady stream of instruction processing. However, this methodology introduces control hazards such as branch and jump hazards. Branch hazards occur when the processor must wait to determine the outcome of a branch instruction before proceeding, potentially stalling the pipeline. Jump hazards similarly disrupt the flow by changing the instruction sequence unexpectedly.

These challenges necessitate advanced solutions to maintain efficiency. For instance, branch prediction and techniques like dynamic branch prediction are utilized to minimize stalls associated with control hazards. These methods predict the behavior of branch instructions to keep the pipeline filled [12]. Enhanced pipelined architectures further mitigate these hazards by optimizing the instruction flow and minimizing penalties associated with incorrect predictions, ensuring that operations like matrix multiplication on the FPGA are executed with minimal delay and higher efficiency.

4.5 MAC Module:

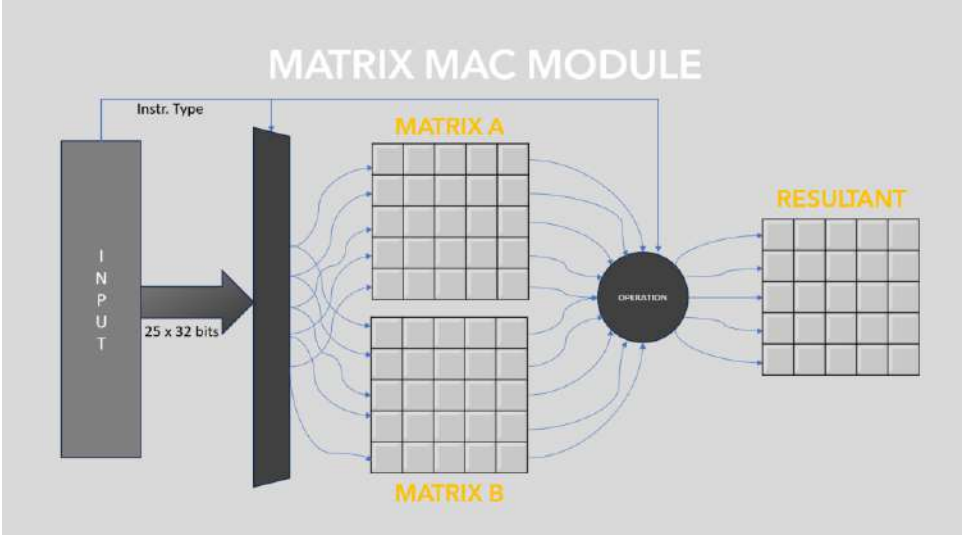


Figure 4.5: Internal Architecture of Matrix MAC Unit

The MAC module serves as the core for fixed-point matrix computations. It manages three internal matrices:

Matrix A: Stores the first operand matrix with 25 registers (A1 to A25).

Matrix B: Stores the second operand matrix with 25 registers (B1 to B25).

Result Matrix (R): Holds the final outcome of the operation, also with 25 registers (R1 to R25).

Each register within the module is 32 bits wide.

The module is equipped with:

25 Input Ports (32 bits each): These ports facilitate loading data from the main memory into the corresponding registers within the matrices.

25 Output Ports (32 bits each): These ports are used to store the results calculated by the MAC unit back into the main memory.

4-bit MAC Control Signal Input (MACControl[3:0]): This dedicated input port receives control signals that instruct the module on the specific operation to perform (e.g., multiplication, addition, subtraction).

#### 4.5.1 Custom Instruction Set for Matrix Manipulation (32 bits):

To seamlessly integrate with the RISC-V architecture, a custom instruction format is designed specifically for matrix operations. Here's a breakdown of the 32-bit instruction format:

- Opcode (7 bits, bits 6-0): This field identifies the exact type of matrix instruction being issued.
- F3 (3 bits, bits 9-7), F7 (2 bits, bits 31-30): These bits select the operation to be performed. F7 is also called MAC-OP.
- Row (10 bits, bits 19-10): This field specifies the starting row address within the chosen matrix (relevant for load instructions).
- Offset (10 bits, bits 29-20): This field defines the offset value used to access elements within a particular row (relevant for load instructions).

#### 4.5.2 Supported Instruction Set and Functionalities:

The table below provides a comprehensive overview of the supported instructions, their corresponding control signals, and their functionalities:

*Table 4.1: Modified MAC ISA*

| Instruction        | F7 | F3  | MAC Control Signals | Description                                 |
|--------------------|----|-----|---------------------|---|
| LMAC A Row, Offset | 00 | 000 | 0000                | Load Matrix A from memory into registers    |
| LMAC B Row, Offset | 00 | 001 | 0001                | Load Matrix B from memory into registers    |
| CLR A              | 01 | 000 | 0010                | Clear all registers in Matrix A             |
| CLR B              | 01 | 001 | 0011                | Clear all registers in Matrix B             |
| CLR R              | 01 | 010 | 0100                | Clear all registers in Result Matrix (R)    |
| CLR ALL            | 01 | 011 | 0101                | Clear all registers in A, B, and R matrices |
| MAC M              | 10 | 000 | 0110                | Perform Matrix Multiplication ( $A * B$ )   |
| MAC ADD            | 10 | 001 | 0111                | Perform Matrix Addition ( $A + B$ )         |
| MAC SUB            | 10 | 010 | 1000                | Perform Matrix Subtraction ( $A - B$ )      |
| SUB MAC            | 10 | 011 | 1001                | Subtract Matrix A from B ( $B - A$ )        |
| STR R              | 11 | 000 | 1010                | Store Result Matrix (R) back to main memory |

## 4.6 MAC Decoder:

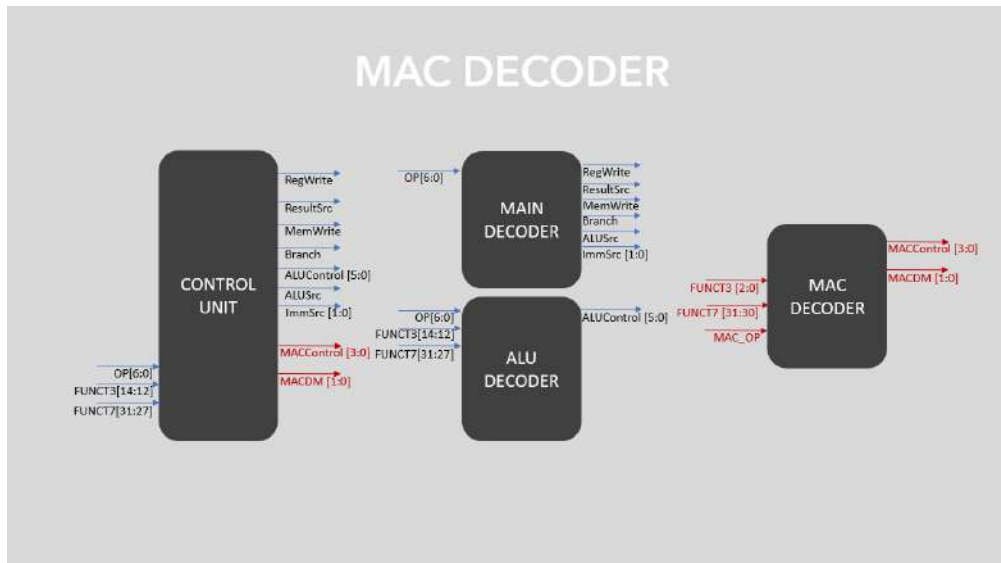


Figure 4.6: Modified MAC Decoder

Here's a more comprehensive interpretation of the MAC decoder's functionality, incorporating insights from your previous description:

### 4.6.1. Inputs:

**funct3 (3 bits):** Extracted directly from the 32-bit instruction format. It pinpoints the precise type of matrix instruction being executed.

**MAC\_OP (1 bit):** Signifies whether the incoming instruction is specifically intended for the MAC module.

**MACOP (2 bits):** Differentiates between categories of matrix instructions, including:

- Load operations (LMAC A, LMAC B)
- Clear operations (CLR A, CLR B, CLR R, CLR ALL)
- Store operations (STR R)
- MAC operations (MAC M, MAC ADD, MAC SUB, SUB MAC)

### 4.6.2. Outputs:

**MACDM (2 bits):** These bits convey signaling information to the main memory, indicating matrix operations:

- 00: No matrix operation is involved.
- 01: Load Matrix A instruction is being executed.
- 10: Load Matrix B instruction is being executed.

- 11: Store Resultant Matrix instruction is being executed.

MACControl (4 bits): These bits generate the control signals that ultimately govern the MAC module's behavior, specifying the exact operation to be performed.

#### 4.6.3. Decoding Logic:

- The decoder meticulously analyzes the combination of funct3, MAC\_OP, and MACOP input bits to accurately determine the intended matrix instruction.
- Based on this analysis, it generates the corresponding MACControl signals to guide the MAC module toward the correct execution.
- Simultaneously, it produces the MACDM signals to alert the main memory about ongoing matrix operations, ensuring proper memory access and data transfer.

#### 4.6.4. Integration with Control Unit:

The MAC decoder operates as a cohesive component within the broader control unit of your RISC-V processor.

It collaborates with other decoders (e.g., main decoder, ALU decoder) to collectively handle the full spectrum of instructions, including both conventional instructions and the newly introduced matrix-specific instructions.

#### 4.6.5. Key Roles in Matrix Operations:

**Instruction Identification:** Accurately pinpoints the type of matrix instruction being issued.

**Signal Generation:** Produces control signals (MACControl) to initiate specific operations within the MAC module.

**Memory Communication:** Employs MACDM signals to coordinate with the main memory for loading and storing matrix data.

#### 4.7 Register File:

The register in the Intellera processor I/O, especially created for MAC module, has a dedicated port M\_A input (20 bits) which is suggested to speed up the matrix operations. In addition, this port obeys block and row addresses which matrix arithmetic instructions provide, allowing the entries of matrixes to get pushed to the right registers directly without further reading mechanism. Sequentially the adaptation combines the data processing effortlessly for matrix operations and noticeably enhances the speed of retrieving the required information consequently benefiting the entire execution of matrix tasks. The management of this feature in the control unit guarantees that



only the M\_A input is selectively activated to achieve high-quality synchronous operation between a register file, a control unit and a MAC module, thereby bringing the performance of the processor to a maximum level when performing matrix manipulation.

#### 4.8 Control Unit for Matrix Operations:

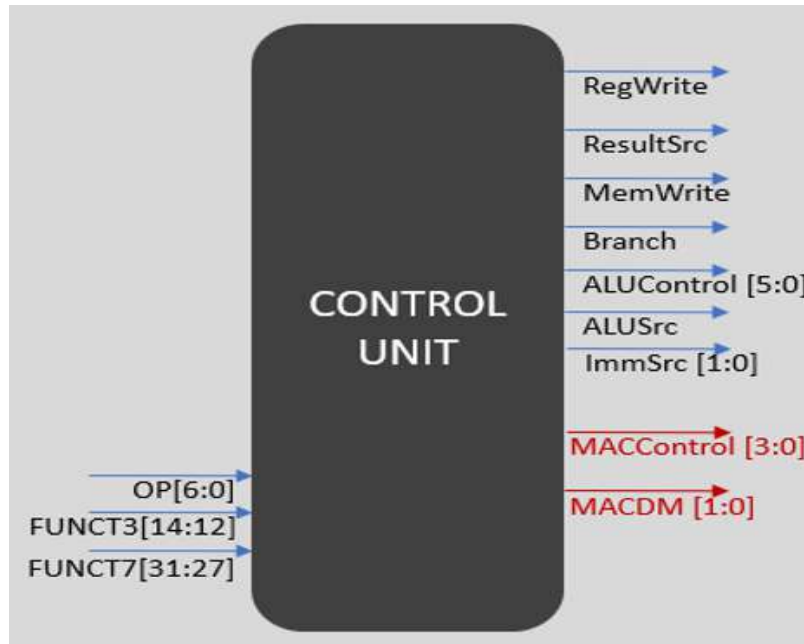


Figure 4.7: Modified Control Unit

The Intellera processor's control unit has been enlarged in its scope to include matrix operations, incorporating more control signals (MAC\_OP and MACOP) that are dedicated for the specific sake for MAC module tasks. This is due to the fact that the exchange allows to perform not only the common RISC-V functions but also specific commands pertaining to a matrix. A well-tailored dedicated MAC Decoder, developing in parallel main decoder and ALU decoder, generates specific control signals for the MAC block, according to the specific matrix instruction. This is an implementation of peculiar matrix calculations, whereby precision of the data processing is highly optimized, hence the improvement of the processor dedicated functions.

#### 4.9 Data Memory:

The data memory used in the Intellera processor was greatly improved to tackle matrix operations with specific devices that were just added like a MACDM Input and expanded data ports among other things. The 2-bit decoder (MACDM Input), triggered by a memory operation or matrix transfer, selects the desired placement. The choice is complemented by the selection of either the source matrices A, B, or the operation result to be saved or loaded in memory. Besides this,

simultaneous use of 25 input and output ports allows retrieval of data in the optimal time frame between a memory and a MAC module. The control unit makes memory access offsets calculations, that are based on Matrix row and offset information, that is carried to the MA\_M input, in order to achieve maximum precision in data retrieval and storage. In addition, those additors support not only a flexible and speedy approach to handling data for matrices but also an optimal result in terms of data transfer and computation inside the framework itself.

#### 4.10 Five Stage Pipelined Processor with Matrix MAC:

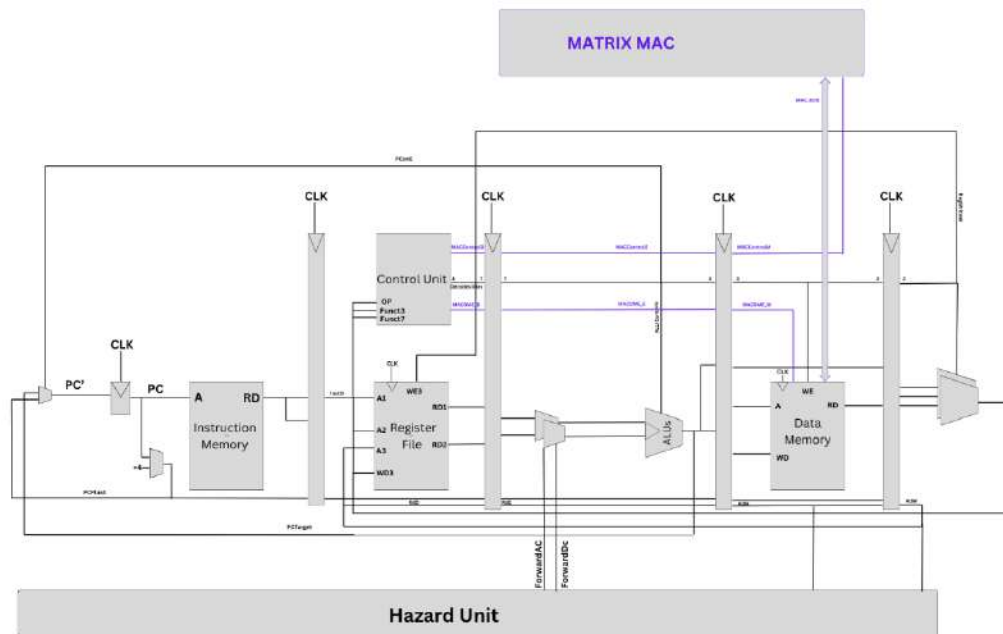


Figure 4.8: Complete 5 Staged Pipelined Processor with Hazard & Matrix MAC Unit Architecture

Intellera utilizes a 5-stage pipelined processor model to enhance the execution of matrix multiplication operations critical for its functionality. By dividing the instruction-processing cycle into five stages—Instruction Fetch, Instruction Decode/Register Read, Matrix Address Calculation, Matrix Operand Access, and Execute/Write Back—Intellera enables parallel processing of instructions, significantly increasing throughput. This method ensures continuous operation by keeping key components like the ALU active, reducing idle cycles and maximizing efficiency. Each stage is meticulously designed to handle specific tasks efficiently, from fetching instructions from memory to executing operations and storing results, thereby streamlining the entire computation process within the processor.

#### 4.11 UART Transceiver with Matrix MAC:

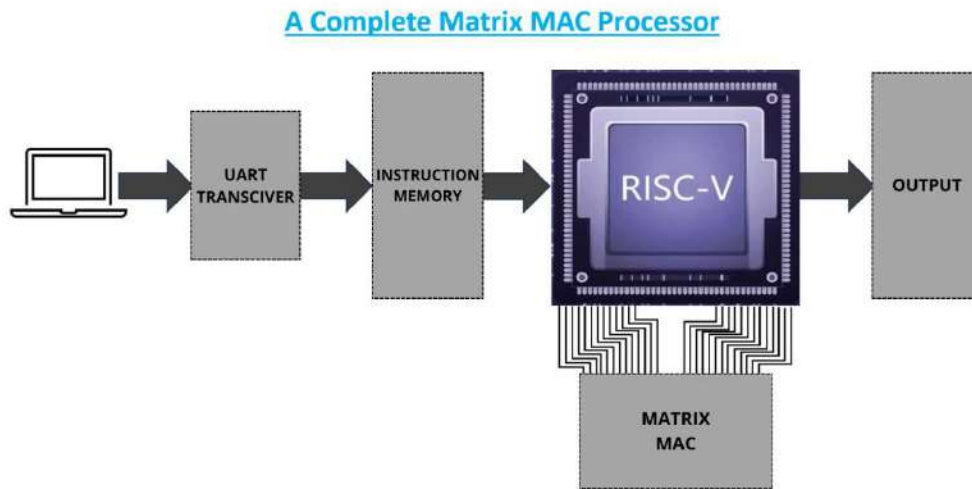



Figure 4.9: UART Interfacing

To develop the UART transceiver module in Verilog, we implemented it with the transceiver for data frame transmission and reception that were 8-bit, at one bit for start and stop bits respectively. The transceiver data is being transmitted asynchronously in a serial/synchronous fashion like a two-way street. Each 8 digital bits that make an individual instruction are transferred and received through four separate 8-digital bits packets due to the UART's 8-bit communication limit. The moment a byte has arrived the transceiver module uses a state machine to step through four states that each of them corresponds a bit within a byte. These memory bytes are loaded into a holding register in a buffer and temporarily hit. Once the full set of bytes is obtained, the module joins them into an orderly regiment of 32-bit instruction. This transfer is total and the instructions are the integral part of the instruction memory where they are kept into reserve until they are eventually deciphered for execution by the processor. Through this method, tasks are broken down into smaller steps of instructions; the processor receives and executes these steps even though they are conveyed in a step-by-step or packetized manner.

## Chapter 5: Results and Discussion

### 5.1 Benchmarking Tests Setup:

Table 5.1: Benchmarking and Testing of Matrix MAC Instr:



| Instruction | MAC Control | Opcode  | Funct3 | MAC_OP | Type   | Tested | Test Case    |
|-------------|-------------|---------|--------|--------|--------|--------|--------------|
| LMAC A      | 0000        | 1110111 | 000    | 00     | M-Type | YES    | LMACA 400, 5 |
| LMAC B      | 0001        | 1110111 | 001    | 00     | M-Type | YES    | LMACA 425, 5 |
| CLR A       | 0010        | 1110111 | 000    | 01     | M-Type | YES    | CLR A        |
| CLR B       | 0011        | 1110111 | 001    | 01     | M-Type | YES    | CLR B        |
| CLR R       | 0100        | 1110111 | 010    | 01     | M-Type | YES    | CLR R        |
| CLR All     | 0101        | 1110111 | 011    | 01     | M-Type | YES    | CLR ALL      |
| MACM        | 0110        | 1110111 | 000    | 10     | M-Type | YES    | MAC M        |
| MAC Add     | 0111        | 1110111 | 001    | 10     | M-Type | YES    | MAC ADD      |
| MAC Sub     | 1000        | 1110111 | 010    | 10     | M-Type | YES    | MAC SUB      |
| SUB MAC     | 1001        | 1110111 | 011    | 10     | M-Type | YES    | SUB MAC      |
| STR R       | 1010        | 1110111 | 000    | 11     | M-Type | YES    | STR R        |

To evaluate the performance and functionality of the Intellera processor, a group of benchmarking tests that had been designed and run were employed to assess Intellera processor's performance and usability. The benchmark tests were made to determine the correctness of the information throughput, speed, and rate of the operations executed by Intellera's matrix MAC resources.

They cover multiple matrix operations such as loading matrix operands, putting registers into clear state, performing matrix multiplication, matrix addition, matrix subtraction and storing the results. Every scenario was creatively considered to check for the precision and authentic output of the Intellera processor's matrix MAC units.

Following which the benchmarking tests were performed and the outcomes were analyzed through simulation to measure how the Intellera processor achieved high efficiency and accuracy for matrix manipulation.

Figure 5.1 Behavioral simulation of MAC Instructions, Testing Addition and Multiplication of Matrices

## 5.2 Performance Evaluation:

Table 5.2 Performance Metrics before and after Matrix MAC Unit

| Parameters    | 5 stage Pipelining with Hazard Unit | 5 Stage Pipelined Processor with Matrix MAC |
|---------------|-------------------------------------|---|
| Lookup Tables | 283                                 | 6338  |
| Flip Flops    | 223                                 | 712   |
| Frequency     | 333 MHz                             | 333 MHz                                     |
| Power         | 0.185 W                             | 0.637 W                                     |

### 5.2.1 Resource Utilization

This section compares the resource utilization, specifically the number of Lookup Tables (LUTs) and Flip Flops used in both processor configurations. The processor with the Matrix MAC utilizes significantly more LUTs (6338 vs. 283) and Flip Flops (712 vs. 223), reflecting the complexity and increased capability provided by the Matrix MAC unit [13].

### 5.2.2 Operational Frequency

Both configurations operate at the same frequency of 333 MHz, suggesting that the integration of the Matrix MAC unit does not adversely affect the clock speed of the processor [14].

### 5.2.3 Power Efficiency

Despite the increased functionality, the power consumption of the processor with the Matrix MAC unit is significantly higher (0.637 W vs. 0.185 W). That shows the implications of this increased power demand and explores potential optimizations to mitigate power consumption while maintaining performance [15].

### 5.2.4 Throughput Analysis

Throughput can be calculated as the reciprocal of the latency (time per operation) multiplied by the clock frequency (operations per unit time):

$$\text{Throughput} = (1 / \text{Latency}) * \text{Clock frequency}$$

$$\text{Throughput} = \left(\frac{1}{3.729}\right) * 333$$

$$\text{Throughput} = 89 \text{ MOPS}$$

### 5.2.5 Latency Measurements

Latency can be defined as the time it takes for a signal to propagate through a certain portion of the processor. In this case, we can calculate the latency for a basic operation, such as a register-to-register operation, assuming it occurs within a single clock cycle.

$$Latency = TSU + TH + TW$$

$$Latency = 2.016 ns + 0.213ns + 1.50ns$$

$$Latency = 3.729 ns$$

## **Chapter 6: Conclusion and Future Work**

The detailed elaboration of the Intellera processor, which is basically a hardware RISC-V accelerator done on FPGA to handle matrix MAC operations, is an exceptional advancement in hardware acceleration for matrix computation. As a wrap-up, this chapter is a summary of the prominent results and the final inferences.

### *6.1 Conclusion:*

The time will come that the process of making and examining the Intellera processor is certainly considered as the most delicate point of affirmation of the RISC-V systems when we talk of the processing system. What has been done then is solidify the framework of building up a MAC Unit to a pipelined architecture and tackling the problems and requirements of high-speed matrix operations. Technical findings of the research show that Intellera is a good evidence for the fact that we can successfully achieve good performance with ease with the use of dedicated hardware accelerators in processors - continuously computation, lower power usage, and massive improvement throughput.

Intellera's architecture reflects a meticulous balance between innovation and practical implementation, leveraging of the flexibility of FPGA is done which not only enables the system to perform remarkably but also retain the compatibility and scalability. The placement of the hazard unit illustrates an integrated technique that tackles the issue of instructional pipelines' stalls caused by dependencies. This ensures instruction flows among the units will be smoother and efficient.

### *6.2 Achievements:*

The outcome of the project is by no means only about performance metrics, it covers complex setting up of standard that can be modified according to the context. Successfully developing custom MAC instructions has paved the way for unique signal processing functionalities embedded directly in the processor, which has the potential to be an outstanding edge in tasks like microprocessor systems, digital signal processing, and artificial neural networks.

### *6.3 Lessons Learned:*

Through the design and testing phases of Intellera, valuable insights have been gained thorough test program - the complexity of processor design as well as the trade-offs in hardware vs software

requirements has been revealed. The implementation of MAC unit into a 5-stage pipeline FPGA architecture has many challenges that go beyond the current capability of the FPGA technology. It has encouraged a deeper comprehension of both the potential and the limitations of this area of technology.

#### *6.4 Future Work:*

The Intellera chip is just beginning as it aims to mark major milestones in the history of space exploration. The development of many ways is obvious in the short term, and these could translate to sustained improvements of the processor's efficiency and the ability to handle the more complex and advanced demands in the computing world.

##### **6.4.1 Implementing MMU as an In-built System**

One of the most promising directions for future work is the implementation of the implicit Memory Management Unit (the MMU) is close to the present work. An MMU that is built in would allow the Intellera so that it could take care of more complicated memory addressing, management, and protection features, therefore facilitating the development of advanced operating system solutions and increasing the sophistication of application scenarios [16].

##### **6.4.2 Expansion to System-on-Chip (SoC) Architecture**

Moving Intellera from a pure perception platform to a SOC implementation it can be suitable for a variety of applications. An SoC integration would be a solid choice as it would allow for a greater number of peripheral interfaces, special purpose coprocessors, and dedicated hardware accelerators, ultimately making Intellera a more useful and powerful processing platform [17].

##### **6.4.3 Broadening the Custom Instruction Set**

Building upon the current custom instruction set, Intellera could be enhanced by adding more specialized instructions, which would cater to a wider array of matrix operations and potentially other types of data-intensive computations.



# GLOSSARY

1. ALU (Arithmetic Logic Unit) - A digital circuit used to perform arithmetic and logical operations.
2. CISC (Complex Instruction Set Computing) - A type of processor design where each instruction can execute several low-level operations.
3. Control Unit - The component of the processor that directs the operation of the processor by managing the instruction cycle.
4. FPGA (Field-Programmable Gate Array) - An integrated circuit designed to be configured by a customer or a designer after manufacturing.
5. FSM (Finite State Machine) - A computational model that can be in exactly one of a finite number of states at any given time.
6. GPU (Graphics Processing Unit) - A specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device.
7. Hazard Detection Unit - Part of a processor used to detect and manage situations that could cause incorrect behavior in pipelined architectures.
8. ISA (Instruction Set Architecture) - The part of the computer architecture related to programming, including the native data types, instructions, registers, addressing modes, memory architecture, interrupt and exception handling, and external I/O.
9. Lookup Tables (LUTs) - A memory resource in FPGAs used to implement logic functions.
10. MAC (Multiply-Accumulate) Unit - A hardware element that performs multiply-and-accumulate operations, often used in digital signal processing.
11. Matrix Operations - Mathematical operations involving matrices, including addition, subtraction, multiplication, and others.
12. MMU (Memory Management Unit) - A hardware component responsible for handling accesses to the memory requested by the CPU, performing tasks such as virtual address translation, physical address generation, memory protection, and cache control.
13. Pipelining - A technique where multiple instructions are overlapped in execution in a processor.
14. Register File - A small, fast storage element in a processor that holds the operands (data) and results of the operations executed by the ALU.

15. RISC (Reduced Instruction Set Computing) - A type of processor design that allows every instruction cycle to use a single clock cycle by using a small and highly optimized set of instructions.
16. RTL (Register Transfer Level) - A high-level abstraction layer for a digital circuit which is used in hardware description languages to describe the operations, timing, and structure of electronic systems.
17. System-on-Chip (SoC) - An integrated circuit that integrates all components of a computer or other electronic systems into a single chip.
18. TPU (Tensor Processing Unit) - An integrated circuit developed specifically for accelerating tensor calculations.

## REFERENCES

- [1] Patterson, D. A., & Hennessy, J. L. (2017). Computer Organization and Design MIPS Edition.
- [2] Jouppi, N. P., Young, C., Patil, N., & Patterson, D. (2017). "In-Datcenter Performance Analysis of a Tensor Processing Unit."
- [3] D. Dennis et al., "Single cycle RISC-V microarchitecture processor and its FPGA prototype," IEEE Symposium on Integrated Circuits and Devices, 2017.
- [4] S. J. Malone, P. Saenz, and P. Phelan, "RISC-V Processors for Spaceflight Embedded Platforms," IEEE Aerospace Conference, 2023.
- [5] L. Qingxin and P. Li, "A 8-bit MCU design using a four-pipeline architecture," IEEE International Conference on Computer and Communication Systems, 2002.
- [6] N. Krishna, "Performance Analysis of MAC Unit using Booth, Wallace Tree, Array and Vedic multipliers," International Journal of Engineering Research & Technology, 2020.
- [7] A. Kamaleldin, S. Hesham, and D. Göringer, "Towards a Modular RISC-V Based Many-Core Architecture for FPGA Accelerators," IEEE Access, 2020.
- [8] M. Jacobsen, Y. Freund, and R. Kastner, "RIFFA: A Reusable Integration Framework for FPGA Accelerators," IEEE Symposium on Field-Programmable Custom Computing Machines.
- [9] A. Habibizad Navin, Ehsan Lahouti, Mahmoud Lotfi Anhar, M. Mirnia, "A new method to prevent control hazard in pipeline processor by using an auxiliary processing unit," 2010.
- [10] M. Miroshnyk, S. Poroshyn, A. Shkil, E. Kulak, I. Filippenko, D. Kucherenko, Y. Pakhomov, S. Juliia, M. Goga, "Design of Logical Control Units Based on Finite State Machines' Patterns," 2018.
- [11] R. P. Aneesh, K. Jiju, "Design of FPGA based 8-bit RISC controller IP core using VHDL," 2012.
- [12] I. Thanga Dharsni, Kirti S. Pande, Manoj K. Panda, "Optimized Hazard Free Pipelined Architecture Block for RV32I RISC-V Processor," 2022.
- [13] W. Wang et al., "A universal FPGA-based floating-point matrix processor for mobile systems," 2014.
- [14] R. Divakaran et al., "Implementation and Verification of RISC Processor on FPGA Using ChipScope Pro Tool," 2019.

- [15] K. Sano et al., "Performance Evaluation of Finite-Difference Time-Domain (FDTD) Computation Accelerated by FPGA-based Custom Computing Machine," 2009.
- [16] S. Sajin et al., "Design of a Multi-Core Compatible Linux Bootable 64-bit Out-of-Order RISC-V Processor Core," in Proc. IEEE Int. Conf. on VLSI Design, 2023.
- [17] R. Krishnamurthy and L. Zhao, "Energy-Efficient and Ultra Low Voltage Design of Sub-14nm SoCs and Microprocessors: Challenges and Opportunities," in Proc. IEEE Symp. on Circuits and Systems, 2016.

## APPENDIX A

Table A.1 RISC-V ISA

| Instruction | ALU Control | Opcode  | Funct3 | Funct7  | Type   | Tested | Test Cases                         |
|-------------|-------------|---------|--------|---------|--------|--------|------------------------------------|
| ADD         | 00000       | 0110011 | 000    | 0000000 | R-type | YES    | add x3, x1, x2                     |
| SUB         | 00001       | 0110011 | 000    | 0100000 | R-type | YES    | sub x4, x1, x2<br>sub x5, x2, x1   |
| MUL         | 00010       | 0110011 | 000    | 0000001 | R-type | YES    | mul x6, x1, x2                     |
| AND         | 00011       | 0110011 | 111    | 0000000 | R-type | YES    | and x6, x1, x2                     |
| XOR         | 00101       | 0110011 | 100    | 0000000 | R-type | YES    | xor x10, x1, x2                    |
| OR          | 00100       | 0110011 | 110    | 0000000 | R-type | YES    | or x7, x1, x2                      |
| SLT         | 01100       | 0110011 | 010    | 0000000 | R-type | YES    | slt x11, x2, x1                    |
| LW          | N/A         | 0000011 | 010    | N/A     | I-type | YES    | lw x25, -11(x31)<br>lw x12, 8(x10) |
| LI          | N/A         | 0010011 | 000    | N/A     | I-type | YES    | li x2, -2<br>li x10, 400           |
| ADDi        | N/A         | 0010011 | 000    | N/A     | I-type | YES    | addi x1, x0, 15                    |
| SLLI        | 00110       | 0010011 | 001    | 0000000 | I-type | YES    | slli x12, x1, 3                    |
| SRLI        | 00111       | 0010011 | 101    | 0000000 | I-type | YES    | srli x13, x2, 3                    |
| SW          | N/A         | 0100011 | 010    | N/A     | S-type | YES    | sw x6, -3(x31)<br>sw x4, 8(x10)    |
| BGE         | 01000       | 1100011 | 101    | N/A     | B-type | YES    | bge x2, x1, -56                    |
| BEQ         | 01001       | 1100011 | 000    | N/A     | B-type | YES    | beq x1, x1, 56                     |
| BNE         | 01010       | 1100011 | 001    | N/A     | B-type | YES    | bne x2, x1, 12                     |
| BLT         | 01011       | 1100011 | 100    | N/A     | B-type | YES    | blt x2, x1, test2                  |

## **APPENDIX B**

GitHub Repository Link:

<https://github.com/theuppercaseguy/FYP--Risc-V-32-bit-Matrix-Mac>