BACHELOR OF TECHNOLOGY
PROJECT REPORT

Enhancing Performance of Neural Networks Using Custom Extension of RISC-V Processor

Submitted as Part of the Academic Work for the Bachelor of Technology (B.Tech) Program

Under the Supervision of:

Prof. Roy P. Paily

By:

Bhavik Jain
Roll No.: 220102023

Department of Electronics and Electrical Engineering
Indian Institute of Technology
Guwahati, Assam, India
November, 2024

Declaration

I hereby declare that the project report titled: **"Enhancing Performance of Neural Networks Using Custom Extension of RISC-V Processor"** submitted by me, **Bhavik Jain** (Roll no. 220102023), is an authentic work carried out by me as part of my academic curriculum.

This project has not been submitted to any other institute or university for the award of any degree or diploma. All sources of information and materials used in the project have been duly acknowledged.

ACKNOWLEDGMENTS

I express my heartfelt gratitude to my supervisor, **Prof. Roy P. Paily**, for their invaluable guidance, support, and encouragement throughout the course of this project.

I also thank the faculty members and staff of the **Department of Department of Electronics and Electrical Engineering** at **Indian Institute of Technology, Guwahati** for their assistance and for providing me with the resources and opportunities needed for the successful completion of this project.

Finally, I extend my gratitude to my family and friends, whose constant motivation and support made this endeavor possible.

Contents

Chapter 1

Introduction

1.1 Background and Context

RISC-V, an open-source instruction set architecture (ISA), has gained significant attention for its modularity, flexibility, and ability to support custom extensions. As machine learning (ML) becomes a cornerstone of modern computing, the demand for hardware capable of efficiently executing ML workloads has surged. Tasks such as matrix multiplication, softmax activation, and other ML operations are computationally expensive and require optimized processing capabilities.

Traditional general-purpose processors are not well-suited for the high computational demands of ML applications, often resulting in bottlenecks in performance and power consumption. While GPUs and TPUs offer solutions for accelerating ML workloads, they are typically expensive, power-hungry, and less viable for on-chip or embedded systems.

1.2 The Problem

General-purpose instruction sets are not inherently designed for ML-specific operations, leading to inefficiencies in critical tasks such as matrix computations and activation functions. This results in higher power consumption, increased latency, and suboptimal utilization of hardware resources, particularly in embedded systems where power and area are constrained.

1.3 Motivation

The modularity of the RISC-V ISA offers a unique opportunity to address these challenges by introducing custom instructions tailored for ML-specific tasks. By embedding operations like matrix multiplication and softmax directly into the instruction set, it is possible to achieve significant improvements in processing speed, energy efficiency, and resource utilization. Such enhancements can bridge the gap between the performance of general-purpose processors and specialized accelerators like GPUs or TPUs.

1.4 Objectives

This project aims to:

- Design and implement custom instructions for ML-specific operations within a RISC-V processor.
- Evaluate the impact of these instructions on performance, power efficiency, and resource utilization.
- Provide a comparative analysis between the customized processor and baseline implementations.

Chapter 2

Literature Review

2.1 RI5CY Processor

RI5CY is a 4-stage in-order 32-bit RISC-V processor core developed as part of the PULP (Parallel Ultra-Low Power) platform. It extends the RISC-V ISA with features like hardware loops, post-increment load/store instructions, and additional ALU operations, making it highly customizable for specific tasks. [1].

Compared to other RISC-V cores like BOOM or Rocket, RI5CY's lightweight, power-efficient design makes it ideal for embedded ML workloads. Specialized processors like Google's TPUs and NVIDIA's Tensor Cores further highlight the growing need for hardware acceleration of ML-specific tasks.

Despite its advantages, extending RI5CY poses challenges, including balancing ISA compatibility, pipeline efficiency, and power consumption. Validation and testing of new instructions are also critical. However, emerging trends in energy-efficient ML hardware and sparse tensor operations offer opportunities to address these challenges. RI5CY's modularity and extensibility position it well for advancing on-chip performance for ML-specific workloads.

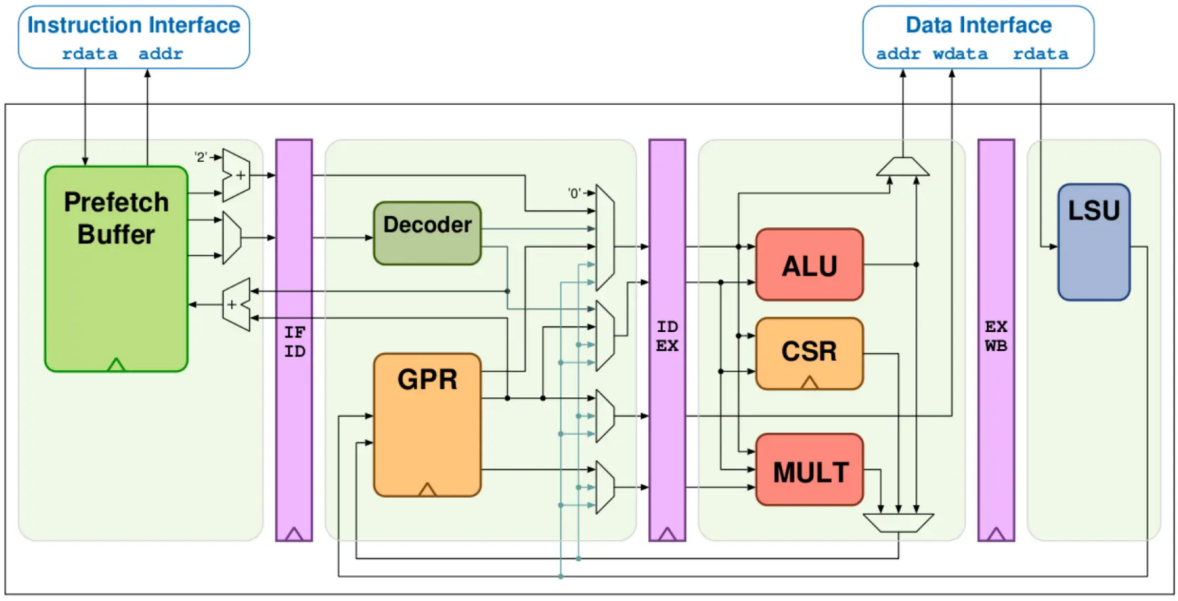

Figure 2.1: Block diagram of RI5CY Processor [1]

2.2 Transform Networks

Transformer networks, introduced by Vaswani et al. (2017) [2] have become foundational in natural language processing (NLP) due to their ability to efficiently capture long-range dependencies through a self-attention mechanism. Unlike traditional recurrent models, transformers process sequences in parallel, improving both training speed and scalability. The self-attention mechanism computes attention scores between all elements in a sequence, allowing the model to focus on important parts of the input dynamically.

Transformers are the backbone of models like BERT and GPT, which excel in NLP tasks such as machine translation and text generation. The architecture has also been adapted to computer vision with Vision Transformers (ViT), showing competitive performance against convolutional neural networks (CNNs) on image classification tasks.

Despite their advantages, transformers are computationally expensive due to the quadratic complexity of self-attention. Hardware accelerators, such as Google's TPUs and NVIDIA's Tensor Cores, have been designed to optimize matrix operations central to transformer models, speeding up training and inference.

Recent efforts focus on lightweight transformers, such as MobileBERT and TinyBERT, to reduce the computational overhead for edge devices. Techniques like sparse attention

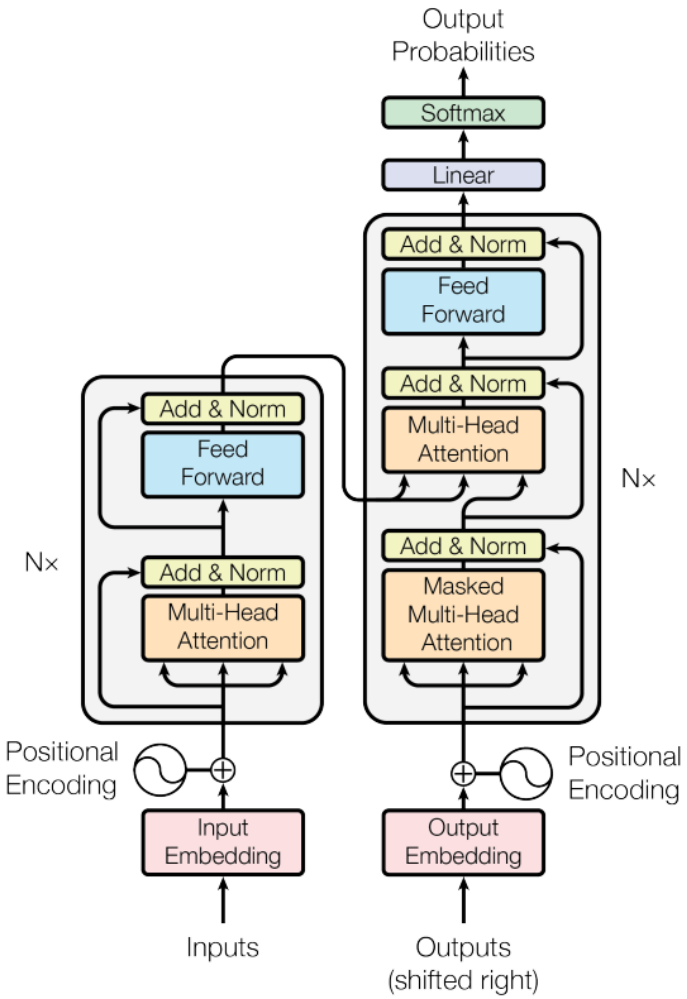

Figure 2.2: The Transformer model architecture [2]

mechanisms are being explored to mitigate the high computational cost.

Customizing processors, such as RI5CY with instructions optimized for ML workloads like matrix multiplication and softmax, offers a promising path to enhancing transformer performance, particularly in resource-constrained environments.

In summary, transformers have revolutionized AI applications, especially in NLP and computer vision, and the demand for specialized hardware accelerators continues to grow to support their computational needs.

Chapter 3

Completed Work

3.1 Designing a 3 Stage Pipelined RISC-V Processor

As part of the groundwork for this project, I initially designed a 3-stage RISC-V processor to gain a deeper understanding of pipelining and its challenges. The 3-stage pipeline was implemented with fetch, decode and execute stages, focusing on the core principles of instruction flow, hazard management, and data dependencies. This effort allowed me to explore fundamental pipelining techniques, such as instruction forwarding, stall mechanisms and interrupt handling in a simplified context.

Designing the 3-stage processor provided valuable insights into pipeline control and optimization, particularly in resolving hazards that arise due to data and control dependencies.

The following diagram illustrates the proposed design of the 3-stage processor.

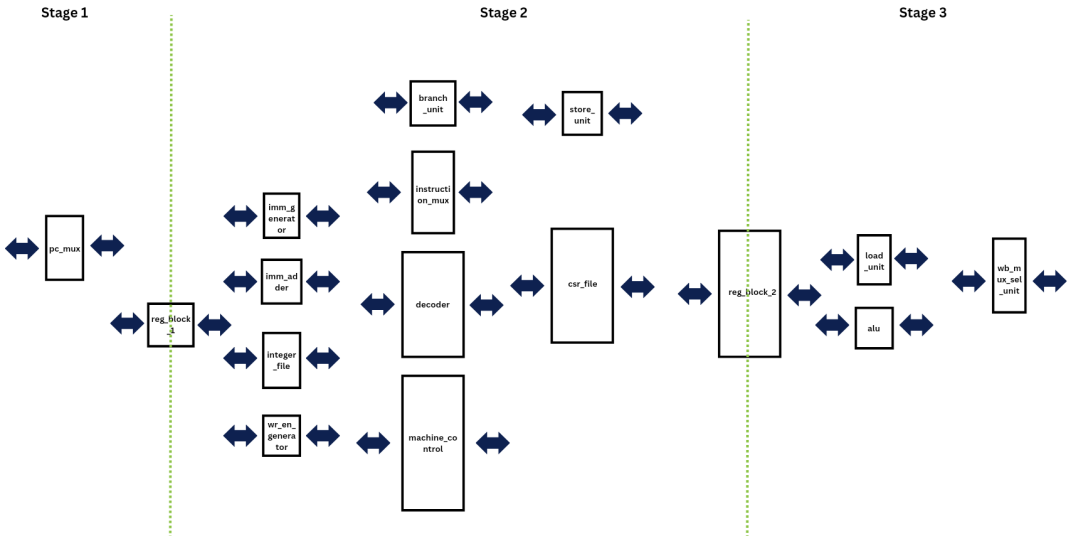

Figure 3.1: Block diagram for 3-stage RISC-V

For the implementation of the proposed design, I wrote the processor code in Verilog, a hardware description language commonly used for designing digital circuits, and ran it on Vivado, an advanced FPGA design suite by Xilinx. The schematic generated by Vivado of the final design is as given below.

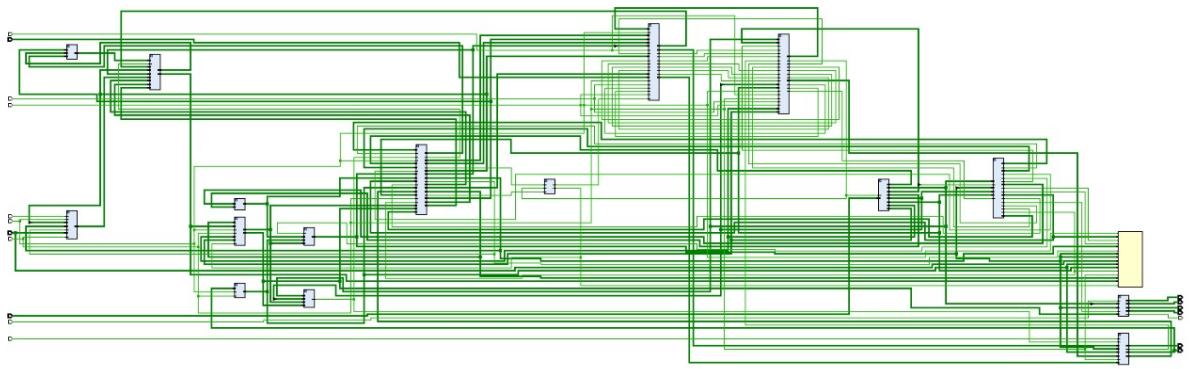

Figure 3.2: Vivado Schematic

The design was tested using a testbench, where the input was the machine code for a basic “Hello!” program. The expected output of this simulation was the generation of a file containing the string ”Hello!” written twice.

To validate the functionality of the processor, I generated corresponding simulation waveforms in Vivado. The simulation results confirmed that the processor successfully executed the program, and the output file contained the string “Hello!” written twice as expected. Snapshots of the Vivado simulation waveform and the generated “Hello World” output file have been attached to demonstrate the correctness of the design.

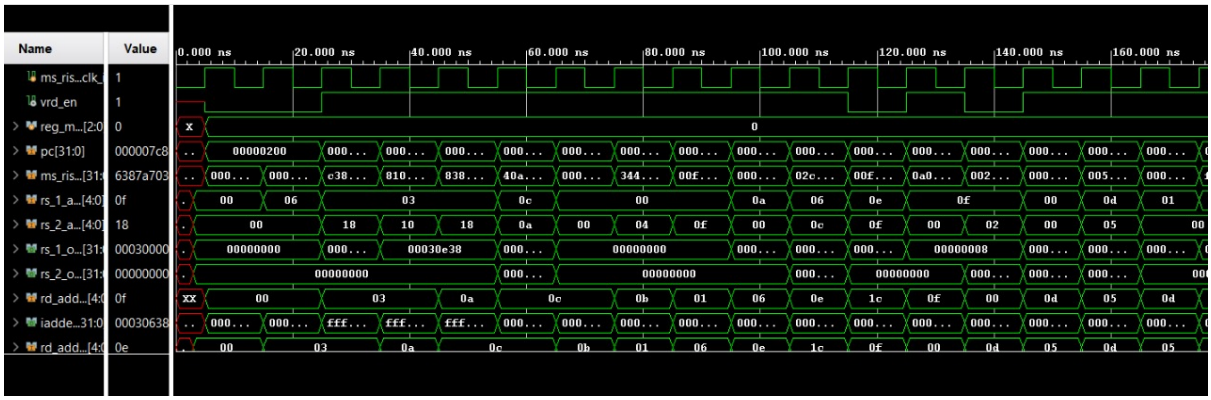

Figure 3.3: The Simulation Waveform

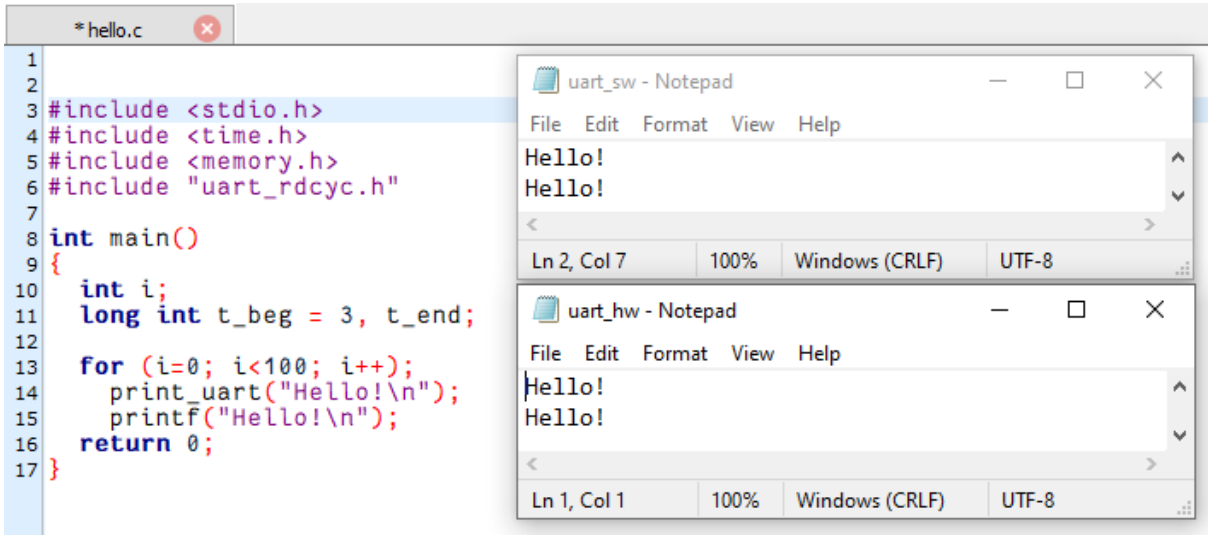

Figure 3.4: The C code snippet which was converted to machine code and finally loaded into the designed processor to get the output

These learnings have been instrumental in preparing for the next phase of this project, where the focus shifts to extending the functionality of a more complex 4-stage RI5CY processor.

3.2 Undesrstanding the RI5CY Processor and Learning to Add Custom Instructions

The next phase of the project focused on studying how custom instructions can be added to the RI5CY processor for enhancing machine learning workloads. This study was guided by the work presented in [3], [4] , where the RI5CY processor was modified to support convolutional neural network (CNN) operations such as Convolution, ReLU, and Max Pooling.

The paper detailed a comprehensive approach to extending the RI5CY processor's instruction set architecture (ISA) with custom instructions optimized for CNN workloads. These modifications enabled significant improvements in computational efficiency and energy consumption for machine learning tasks.

To integrate these instructions, the authors modified the RI5CY processor pipeline in several ways:

1. Instruction Decoding: Enhanced the instruction decoder to recognize and handle custom operations. Data Path Modifications: Added execution units to implement CNN-specific tasks seamlessly.

2. Pipeline Integration: Addressed potential pipeline hazards and ensured compatibility with existing instructions.

3. Furthermore, the paper emphasized the testing and validation processes, which demonstrated the processor's ability to achieve substantial performance gains for CNN workloads while maintaining low power consumption.

In this, I used the RISC-V GNU Toolchain for compiling and assembling the RISC-V machine code, which is essential for the development of custom instructions and verifying the functionality of the processor. The RISC-V GNU Toolchain is a collection of software tools designed to support the development of RISC-V applications, providing a complete environment for building and running code on RISC-V processors.

The toolchain includes several key components:

1. GCC (GNU Compiler Collection): This is used to compile the source code into RISC-V assembly code. It supports various optimizations for efficient code generation.

2. Binutils: A set of utilities for manipulating binary files, such as linking and assembling the object files generated by GCC.

3. GDB (GNU Debugger): A tool for debugging RISC-V code, which is useful for identifying and fixing issues in the custom instructions and processor functionality.

4. Newlib: A C library that provides standard functions for embedded systems programming, often used for developing applications that run on RISC-V processors.

By leveraging the RISC-V GNU Toolchain, I was able to efficiently generate machine code, simulate its execution on my custom processor, and validate its behavior with test programs. This toolchain is an essential part of the development workflow for RISC-V-based systems, offering a robust, open-source alternative for building and debugging applications on RISC-V hardware.

Similar to the 3 stage processor designed before, the "Hello World" program was run on Linux environment using the the GNU Toolchain compiler and Verilator simulator.

The corresponding C code used and the commands run on the terminal are attached below.

```
18
19 #include <stdint.h>
20 #include <stdio.h>
21
22 int main(int argc, char* arg[]) {
23
24         printf("%d: Hello World !", 0);
25
```

Figure 3.5: C code for Hello World program

```
vboxuser@BTP:~$ cd core-v-verif/cv32e40p/sim/core/
vboxuser@BTP:~/core-v-verif/cv32e40p/sim/core$ make custom CUSTOM_PROG=hello-world
/home/vboxuser/core-v-verif/mk/Common.mk:79: Must define a DPI_DASM_SPIKE_REPO to use the common makefile
*******************************************************************************
* Cloning CV32E40P RTL model
*******************************************************************************
git clone https://github.com/openhwgroup/cv32e40p /home/vboxuser/core-v-verif/core-v-cores/cv32e40p; cd /h
ore-v-verif/core-v-cores/cv32e40p; git checkout 523b80f
```

Figure 3.6: Command to run the C program on the RI5CY Processor

```
[tb_top_verilator] finished dumping memory

HELLO WORLD!!!
This is the OpenHW Group CV32E40P CORE-V processor core.
CV32E40P is a RISC-V ISA compliant core with the following attributes:
        mvendorid = 0x602
        marchid   = 0x4
        mimpid    = 0x0
        misa      = 0x40001104
        XLEN is 32-bits
        Supported Instructions Extensions: MIC

TOP.tb_top_verilator @ 134810: EXIT SUCCESS
- /home/vboxuser/core-v-verif/cv32e40p/tb/core/tb_top_verilator.sv:83: Verilog $finish
vboxuser@BTP:~/core-v-verif/cv32e40p/sim/core$
```

Figure 3.7: Generated output

By studying this implementation, I gained valuable insights into extending the RI5CY processor for domain-specific tasks. This understanding will inform my efforts to add custom instructions for other machine learning operations, such as Matrix Multiplication and Softmax, in subsequent phases of my project.

Chapter 4

Future Plans

While many researchers have already customized processors like RI5CY for CNNs and other machine learning networks, designing custom instructions specifically for Transform Networks is important for several reasons. Transform Networks, have become increasingly prominent in recent machine learning applications, especially for natural language processing (NLP) tasks like language translation, text generation, and sentiment analysis. These networks rely heavily on operations like matrix multiplication, softmax, and activation functions—similar to CNNs—yet they require a different set of optimizations due to the unique structure of self-attention mechanisms and large-scale data processing involved in tasks such as sequence modeling.

The primary motivation behind designing custom instructions for Transform Networks is to optimize hardware performance for operations unique to the architecture of Transform Networks, particularly:

1. Self-Attention Mechanisms: These require efficient computation of dot-products and scaling operations, which differ from traditional convolution operations in CNNs. Tailored instructions can speed up these operations while reducing power consumption.

2. Sequence Parallelism: Transform Networks process input sequences (e.g., sentences or text) in parallel rather than spatially, which necessitates optimized data flow and memory handling.

3. Scalability and Efficiency: Transform Networks often require handling large datasets and weights, demanding significant memory bandwidth and efficient matrix operations. Custom instructions can reduce overheads associated with data movement and memory access.

By specifically targeting the needs of Transform Networks, I aim to design custom instructions that optimize these critical operations and make the processor more suited to the increasing demand for performance in NLP and other sequence-based applications. This focus will ensure that the processor architecture remains adaptable and efficient for cutting-edge machine learning models, ultimately contributing to faster training and inference times in Transform Network-based applications.

Thus, future plans for the project are as follows:

1. Designing optimized custom instructions for ML-specific ALU operations like matrix multiplication (MatMul), ReLU, and Softmax.

2. Integrating these custom instructions into the RI5CY processor.

3. Verifying the output by running a pre-designed transform model and ensuring the correctness of the hardware implementation.

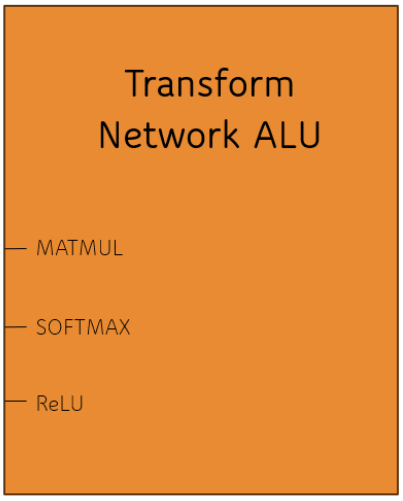

Figure 4.1: The proposed design of the ALU

Chapter 5

References

1. PULP Platform Team. *RI5CY User Manual*. ETH Zurich and University of Bologna, 2021. Available at: `https://pulp-platform.org/docs/ri5cy_user_manual.pdf`.

2. Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I. (2017). Attention is All you Need. Neural Information Processing Systems.

3. S. Wang et al., "Optimizing CNN Computation Using RISC-V Custom Instruction Sets for Edge Platforms," in IEEE Transactions on Computers, vol. 73, no. 5, pp. 1371-1384, May 2024.

4. QmppmQ. "Modifications to RI5CY Processor for CNN Operations". GitHub Repository. Available at: `https://github.com/QmppmQ/riscv`.