RISC V ARCHITECTURE BASED HARDWARE ACCELERATOR FOR k-NN

Submitted as part of Mini-project (ECP67)

BACHELOR OF ENGINEERING

in

Electronics and Communication Engineering

Name of the students:

Satwik Kamath - 1MS21EC098

Shreesha T P - 1MS21EC102

Shreeya R - 1MS21EC103

Shreya P Manchala - 1MS21EC105

Under the guidance of

Dr. V Anandi
Associate Professor
Dept. of E & C,
RIT, Bangalore

Department of Electronics and Communication

RAMAIAH INSTITUTE OF TECHNOLOGY

(Autonomous Institute, Affiliated to VTU)
Accredited by National Board of Accreditation & NAAC with 'A+' Grade
MSR Nagar, MSRIT Post, Bangalore – 560054

2024

CERTIFICATE

This is to certify that the project work titled **"RISC V ARCHITECTURE BASED HARDWARE ACCELERATOR FOR k-NN"** is carried out by Satwik Kamath (1MS21EC098), Shreesha T P (1MS21EC102), Shreeya R (1MS21EC103), Shreya P Manchala (1MS21EC105) bonafide students of Ramaiah Institute of Technology, Bangalore, as part of Mini-project work carried out in sixth semester of Bachelor of Engineering in Electronics and Communication during the year 2024. It is certified that all corrections / suggestions indicated for Internal Assessment have been incorporated in the report. The report has been approved as it satisfies the academic requirements prescribed.

Dr. V Anandi
Associate Professor,
Department of E & C
RIT, Bangalore

Dr. Maya V Karki
Professor and HOD,
Department of E &C
RIT, Bangalore

Name & Signature of Examiners with Date:

1)

2)

DECLARATION

We hereby declare that the mini-project entitled "RISC V ARCHITECTURE BASED HARDWARE ACCELERATOR FOR KNN" has been carried out independently at Ramaiah Institute of Technology under the guidance of Dr. V Anandi, Associate Professor, Department of Electronics and Communication Engineering, RIT, Bangalore.

Signature of Students:

1) Satwik Kamath (1MS21EC098)
2) Shreesha T P (1MS21EC102)
3) Shreeya R(1MS21EC103)
4) Shreya P Manchala(1MS21EC105)

Place: Bangalore

Date: 19-07-2024

ACKNOWLEDGEMENT

The immense satisfaction that accompanies the successful completion of the project would be incomplete without the mention of the people who made it possible. We consider it our honour to express our deepest gratitude and respect to the following people who always guided and inspired us during the course of the Project.

We are deeply indebted to **Dr. N. V. R. Naidu**, Principal, RIT, Bangalore for providing us with a rejuvenating undergraduate course under a highly creative learning environment.

We are much obliged to **Dr. Maya V Karki**, Professor & HOD, Department of Electronics and Communication Engineering, RIT, Bangalore for her constant support and motivation.

We sincerely thank our guide **Dr. V Anandi**, Associate Professor, Department of Electronics and Communication Engineering, RIT, Bangalore and express our humble gratitude for her valuable guidance, inspiration, encouragement and immense help which made this project work a success.

We would like to extend my heartfelt gratitude to **Dr. Raghuram S** and **Dr. Lakshmi Shrinivasan** for their insightful review of our project. Their expertise and constructive feedback have been invaluable in enhancing the quality and scope of our work.

We sincerely thank all the faculty members of Department of E&C, RIT for their kind support to carry out this project successfully.

Last but not the least we would like to express our heartfelt gratitude to our parents, relatives and friends for their constant support, motivation and encouragement.

ABSTRACT

The relentless demand for smarter, more responsive devices has pushed embedded systems to their limits. Applications ranging from image recognition to natural language processing require significant computational power, often exceeding the capabilities of general-purpose processors on resource-constrained platforms. This has led to the rise of hardware accelerators, specialized circuits designed to accelerate specific tasks, offering substantial performance and energy efficiency gains. Traditionally, hardware accelerators were often tied to specific processor architectures and relied on proprietary instruction set architectures. This approach, while effective for targeted applications, suffers from several drawbacks: limited flexibility, interoperability challenges, and vendor lock-in.The emergence of RISC-V, an open-source and royalty-free ISA, presents a compelling alternative for hardware acceleration. This project focuses on designing and implementing a dedicated hardware accelerator that can efficiently execute the K-Nearest Neighbor (k-NN) algorithm, a widely used machine learning technique. The RISC-V architecture and instruction set are leveraged to address the computational complexity, performance bottlenecks, and energy efficiency concerns associated with existing hardware implementations of k-NN. The proposed accelerator architecture is designed to achieve significant speedup and energy savings while using minimum hardware components. Key features of the design include customized instruction set, parallelism and pipelining, reduced power consumption, and optimized area utilization. By leveraging the open-source and customizable nature of RISC-V, this project presents a compelling approach to addressing the computational demands of modern machine learning applications, offering significant improvements in speed, power efficiency, and adaptability. The accelerator will be designed using Verilog and implemented on FPGA development boards compatible with RISC-V architectures. Comprehensive testing and validation will be conducted to ensure the correctness, reliability, and robustness of the accelerator across various datasets and input scenarios. The project aims to develop a highly optimized and efficient hardware accelerator for kNN computations by exploiting the flexibility and customization capabilities of the RISC-V ISA. The accelerator will be designed to achieve real-time processing of large datasets while minimizing energy consumption and maximizing scalability across different applications and platforms.

List of figures

List of Acronyms

Acronym	Abbreviation
RISC V	Reduced Instruction Set Computing - 5th version
ISA	Instruction Set Architecture
IFU	Instruction Fetch Unit
ALU	Arithmetic and Logic Unit
kNN	k Nearest Neighbours

List of Tables

Contents

CHAPTER 1

INTRODUCTION

The growing demand for smarter, more responsive devices has pushed embedded systems to their limits. Applications like image recognition and natural language processing require significant computational power, often beyond what general-purpose processors can provide on resource-constrained platforms. This has led to the rise of hardware accelerators, specialized circuits designed to accelerate specific tasks, offering significant performance and energy efficiency gains.

Traditionally, hardware accelerators were tied to specific processor architectures and relied on proprietary instruction set architectures (ISAs). This approach, while effective for targeted applications, has several drawbacks:

Limited Flexibility: Proprietary ISAs restrict customization for evolving algorithms or changing application requirements.

Interoperability Challenges: Integrating with systems using different processor architectures or ISAs is complex and often inefficient.

Vendor Lock-in: Reliance on proprietary solutions limits design choices and can increase development costs. RISC-V, an open-source and royalty-free ISA, presents a compelling alternative for hardware acceleration. RISC-V's modular and extensible nature allows developers to tailor the instruction set and microarchitecture to precisely match the target application's requirements. This flexibility introduces a new era of hardware acceleration, characterized by:

1. Performance: Customization for Specific Workloads: RISC-V enables creating application-specific instruction sets, allowing complex operations, such as the kNN algorithm, to be implemented directly in hardware, significantly reducing execution time. Parallelism and Pipelining: RISC-V's modularity facilitates highly parallel architectures, exploiting data-level parallelism and optimizing throughput through pipelined execution.

2. Improved Power Efficiency: Reduced Instruction Overhead: Customizing the instruction set eliminates unnecessary instructions, reducing clock cycles per operation and lowering power consumption. Selective Hardware Integration: RISC-V's modularity allows designers to include

only necessary hardware modules, eliminating idle, power-hungry components for a leaner, more energy-efficient design.

3. Optimized Area Utilization: Compact Design Footprint: Tailoring the instruction set and microarchitecture results in a more compact design, beneficial for size-constrained applications like wearables or mobile platforms. Scalability for Future Needs: The modular nature of RISC-V allows incremental additions of new instructions or processing elements, ensuring the accelerator can adapt to future demands without a complete redesign.

The kNN algorithm, widely used in machine learning and pattern recognition, exemplifies the benefits of RISC-V-based acceleration. Its computationally intensive nature often causes performance bottlenecks on traditional platforms. A RISC-V-based accelerator can be tailored to exploit kNN's inherent parallelism by:

- Implementing distance calculations and sorting operations directly in hardware.
- Parallelizing the search across multiple data points or dimensions.
- Optimizing data movement and memory access patterns.

This customized approach results in significant speedup and reduced energy consumption compared to software implementations or generic hardware accelerators. In conclusion, RISC-V is set to revolutionize hardware acceleration. Its open-source nature, modularity, and extensibility empower developers to create highly customized and efficient accelerators tailored to specific applications. As the demand for intelligent devices continues to grow, RISC-V-based accelerators offer a compelling path towards optimal performance and energy efficiency within the constraints of resource-limited platforms.

1.1 Problem Statement

Design and implement a specialized hardware accelerator capable of efficiently executing K-Nearest Neighbors (KNN) computations. The primary objectives are real-time processing, reduced energy consumption, and enhanced scalability for deployment. Achieving real-time processing involves leveraging extensive parallelism and hardware optimization techniques to ensure timely responses to input queries, even for large datasets. Additionally, the RISC-V architecture and instruction

set contribute to energy efficiency. The open-source nature of RISC-V allows customization of the accelerator to meet diverse application requirements and computing platforms, ensuring compatibility and scalability.

1.2 Objectives and Scope

1.2.1 Objectives

- The primary focus is on addressing the computational complexity, performance bottlenecks, and utilization concerns associated with existing hardware implementation of KNN.
- The project aims to achieve significant speedup while using minimum hardware components such as flipflops and LUTs.

1.2.2 Scope

- Develop a hardware accelerator architecture for KNN computations using the RISC-V ISA
- Implement the hardware accelerator design using Verilog Conduct performance analysis of the accelerator to assess its speedup, energy efficiency, and scalability compared to existing hardware-based KNN implementation.
- Conduct comprehensive testing and validation to ensure the correctness, reliability and robustness of the hardware accelerator across various datasets and input scenarios.

CHAPTER 2

LITERATURE SURVEY

The literature survey in this chapter explores advancements in reconfigurable hardware for implementing the K Nearest Neighbor (KNN) algorithm. It covers FPGA-based innovations enhancing performance, flexibility, and efficiency in machine learning, including high-speed KNN optimization, real-time 3D LiDAR processing in autonomous vehicles, and RISC-V SoC design using high-level synthesis, highlighting substantial improvements in speed, power efficiency, and adaptability.

A study in [1] explores KNN implementation within reconfigurable hardware to meet high-performance Machine Learning needs. Highlighting ASIC limitations, it proposes a versatile architecture allowing post-fabrication updates to datasets, algorithm parameters, and distance metrics for ASICs and FPGAs. Implemented on the Genesys 2 board with Xilinx's Kintex-7 FPGA, it achieves 110 MHz frequency, ¡0.4 watts power consumption, and supports both fixed-point and IEEE 754 single-precision floating-point arithmetic, demonstrating efficiency and consistency with simulation results.The proposed hardware architecture excels in reconfigurability, supporting both fixed-point and floating-point operations, achieving up to 109.12 MHz, and consuming 0.359 watts, ideal for advanced machine learning applications.

The paper[2] explores optimizing the DCT-KNN algorithm for COVID-19 classification using FPGA and OpenCL, achieving a 44x speedup and 97.8 percent accuracy. Implemented on Intel De5a-net Arria-10, the study employs parallelization and optimization techniques like loop unrolling and MK-sorting. This approach significantly reduces execution time from 3108.249 seconds to 4.45 seconds, highlighting the effectiveness of FPGA-based accelerators and optimization techniques for high-speed, accurate classification tasks.

The paper[3] explores efficient 3D LiDAR data processing for autonomous vehicles, focusing

on the KNN algorithm. It proposes the Double-Segmentation-Voxel-Structure (DSVS) framework for FPGA implementation, leveraging FPGA's flexibility and computational power. The DSVS framework maximizes parallelism and minimizes data movement, achieving superior performance and energy efficiency. Experiments show a 2.1x and 6.2x improvement in energy efficiency over existing FPGA and GPU solutions, highlighting FPGA's potential for robust, energy-efficient autonomous navigation.

The study in [4] explores the design and implementation of a System on Chip (SoC) based on the RISC-V RV32I instruction set architecture using high-level synthesis (HLS). It details the process from understanding RISC-V to FPGA implementation. The author covers the HLS design methodology using C++, Verilog simulation for validation, and gate-level design with the 'yosys' tool. The successful implementation on a Basys3 FPGA board shows efficient resource utilization and power consumption. Future directions include exploring multi-core RISC-V SoC architectures for enhanced performance and functionality.

CHAPTER 3

METHODOLOGY AND PROPOSED DESIGN

The design process was initiated by defining the project's scope and objectives. Clear goals were set to develop a RISC-V based hardware accelerator optimized for k-nearest neighbors (k-NN) computations. Initial discussions and planning laid the groundwork for comprehensive research and development.The figure 3.1 shows the approach tht was followed while designing the hardware accelerator.

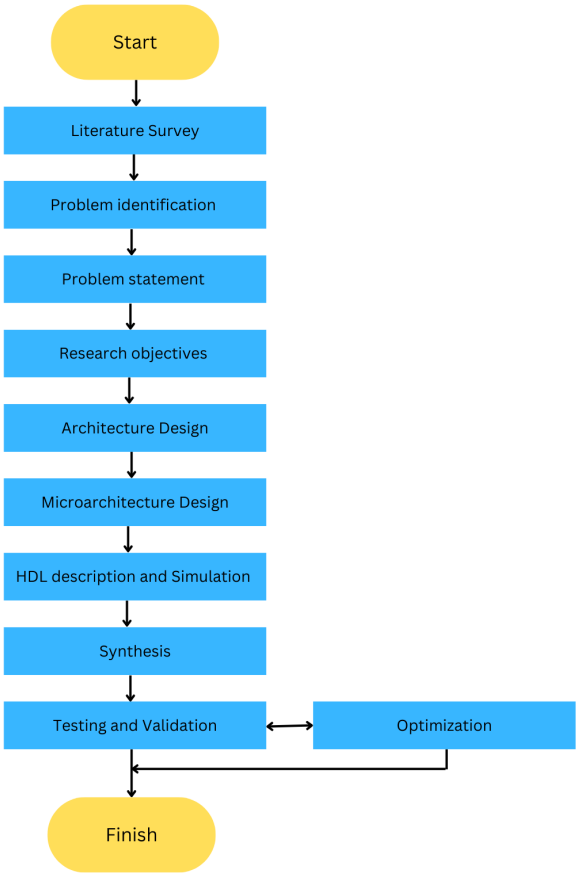

Figure 3.1: Flow Chart describing the process of building the hardware accelerator

The project aimed to develop a RISC-V based hardware accelerator optimized for k-nearest

neighbors (k-NN) computations, starting with defining objectives and conducting a thorough literature review to identify gaps in current implementations. Key challenges, such as computational complexity and energy efficiency, were addressed by formulating a problem statement and research objectives. The design phase involved creating a high-level and detailed microarchitecture tailored for k-NN, followed by HDL description and simulation to verify functionality. The design was then synthesized for FPGA implementation, with iterative optimization to enhance performance and resource efficiency. Comprehensive testing validated the accelerator's performance, leading to a finalized design ready for practical application, demonstrating significant improvements in computational efficiency and scalability.

3.1 kNN ACCELERATOR

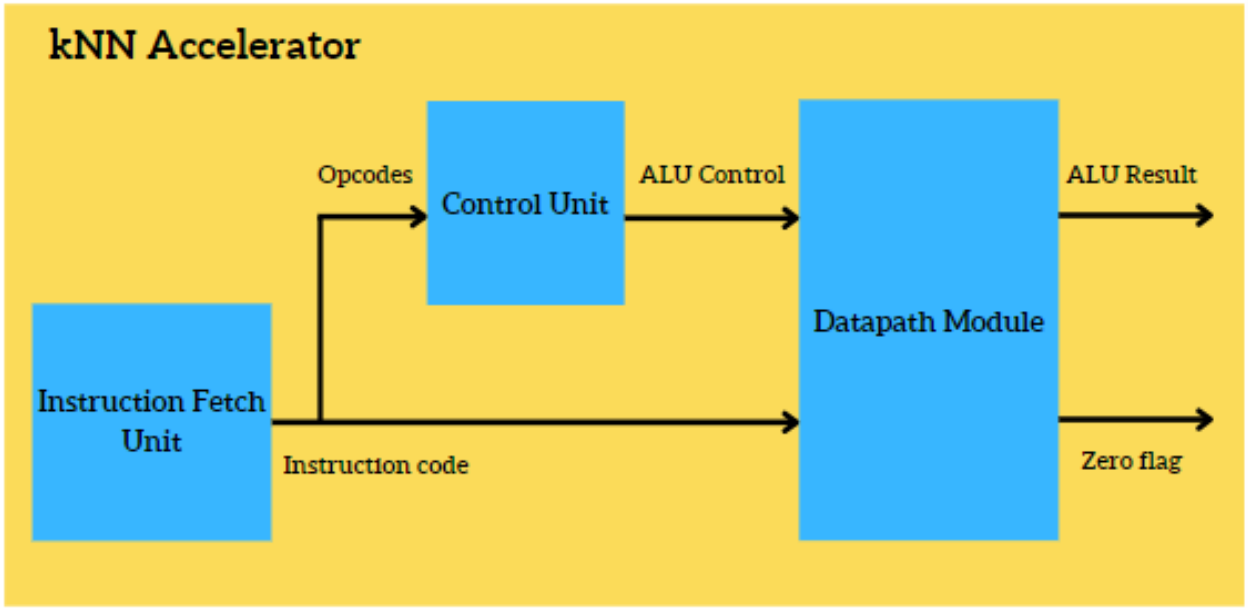

Figure 3.2: Block diagram of kNN Accelerator

In designing a RISC-V based hardware accelerator for the k-Nearest Neighbors (k-NN) algorithm using a single cycle microarchitecture, the processor executes each instruction in one clock cycle, ensuring simplicity and predictability.

The process begins with the Instruction Fetch Unit (IFU) retrieving instructions from the Instruction Memory using the Program Counter (PC). The Control Unit decodes the fetched instructions, generating control signals that guide the Datapath's operations.

The Datapath facilitates dataflow between components like Register Files, the Arithmetic Logic Unit (ALU), and Block Memory (BRAM). Register Files temporarily store operands and intermediate results, enabling quick data access during computations. The ALU performs arithmetic and logical operations, such as distance calculations for the k-NN algorithm. BRAM provides dedicated storage for datasets, distances, and results, ensuring efficient data handling.

The processor continuously cycles through instruction fetch, decode, execute, and store until all instructions are processed. This streamlined approach leverages single cycle execution's simplicity and predictability, optimizing performance for k-NN computations.

Here is a Finite State Diagram depicting the various states of the processor:

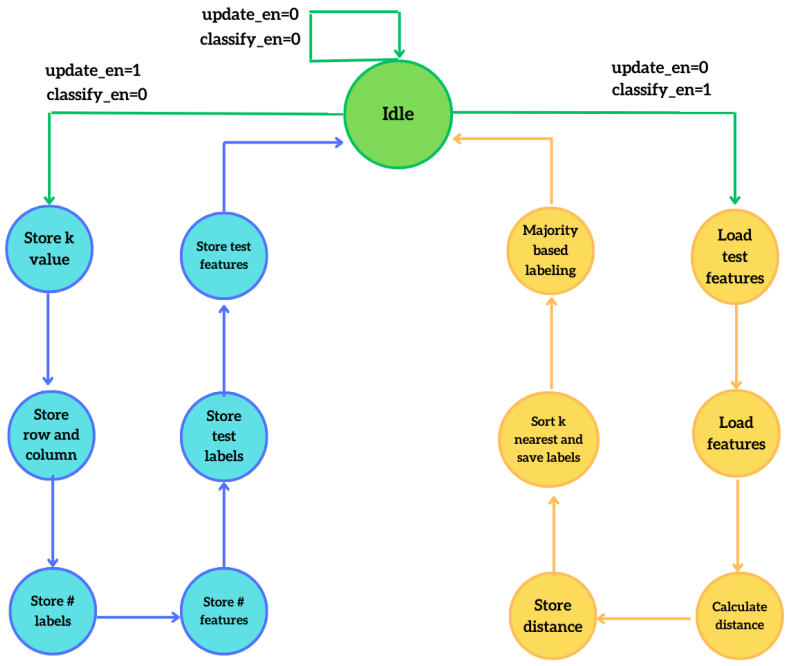

Figure 3.3: Finite State Machine

3.2 Instruction Fetch Unit (IFU)

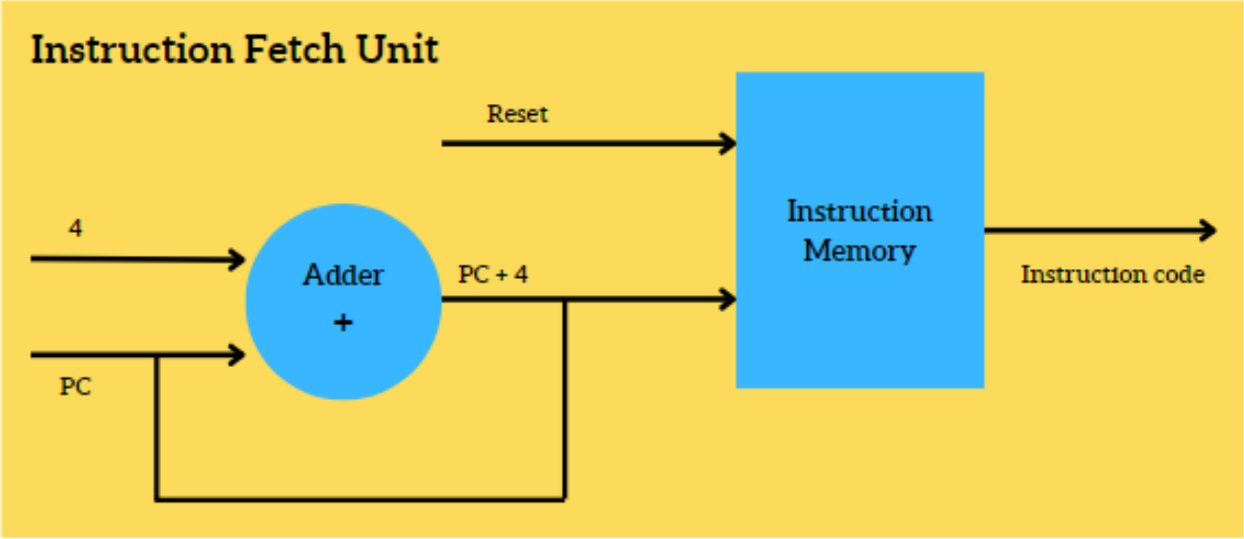

Figure 3.4: Block diagram of IFU

The Instruction Fetch Unit (IFU) is a critical component within the processor responsible for retrieving instructions from memory. Using the Program Counter (PC), it fetches an instruction in one clock cycle, increments the PC to the next address, and forwards the instruction for decoding and execution. Operating synchronously with the processor clock, the IFU ensures timely delivery of instructions for orderly program execution. The IFU fetches a single instruction from memory in one clock cycle and makes the instruction data available for decoding and execution. Concurrently with instruction fetching, the IFU increments the PC to point to the next instruction address, facilitating sequential instruction access and orderly code execution. The IFU accesses instruction memory using the current PC value, retrieves the instruction data, and forwards it for further processing. After fetching an instruction, the IFU increments the PC to ensure continuous program flow and step-by-step execution of the code.

3.3 Control Unit

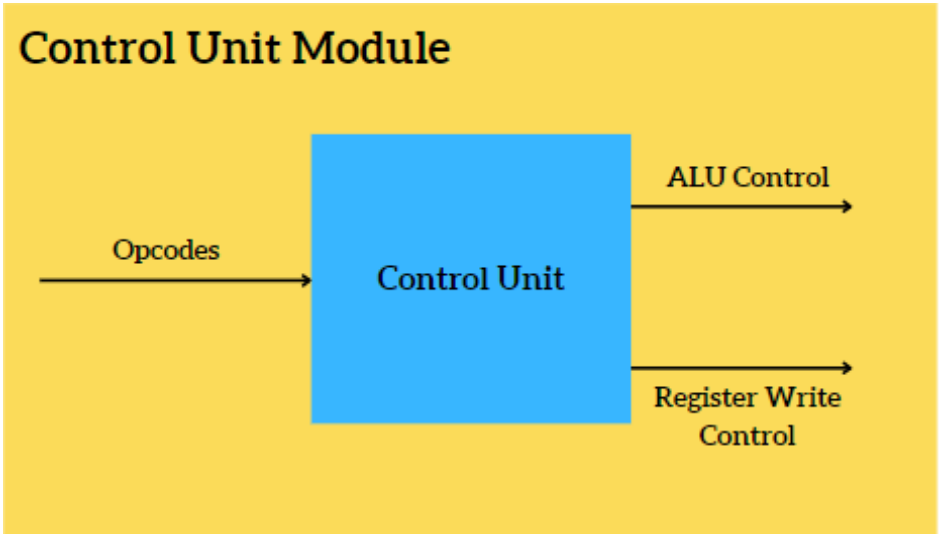

Figure 3.5: Block diagram of Control Unit

The Control Unit decodes the fetched instruction and generates the necessary control signals to coordinate the operations of other blocks. It determines the required operation based on the instruction and manages the data flow and control signals to direct the operations of the ALU, Register Files, and Memory. This ensures that each instruction is executed correctly and efficiently.

Here are the Opcodes used in the control unit:

Base Integer ISA: Load Instruction

Opcode	Instruction	funct3/funct5	Description
0000011	BRAM-1	000	Performs a memory load operation from Data Memory (Block RAM 1). Typically used for loading integer data from BRAM-1 into a register.
0000011	BRAM-2	011	Performs a memory load operation from Label Memory (Block RAM 2). Used for loading integer data from BRAM-2 into a register.

Floating Point ISA: Arithmetic Instructions

Opcode	Instruction	funct3/funct5	Description
1010011	FADD	00001	Performs single-precision floating-point addition. Adds two single-precision floating-point numbers and stores the result.
1010011	SUBMUL	00000	Custom instruction for floating-point subtraction and multiplication. Subtracts two floating-point numbers and multiplies the result with itself, storing the final result.
1010011	FSQRT	00010	Computes the square root of a single-precision floating-point number. Calculates the square root of a floating-point value and stores the result.

Floating Point ISA: Load Instruction

Opcode	Instruction	funct3/funct5	Description
0000111	FLW	-	Loads a single-precision floating-point value from memory into a register.

Base Integer ISA: Branch Instructions

Opcode	Instruction	funct3/funct5	Description
1100011	BLT	001	Branch instructions for control flow operations based on conditions. Determines control flow based on comparisons (equal, not equal, less than, etc.) between two registers.

3.4 Datapath

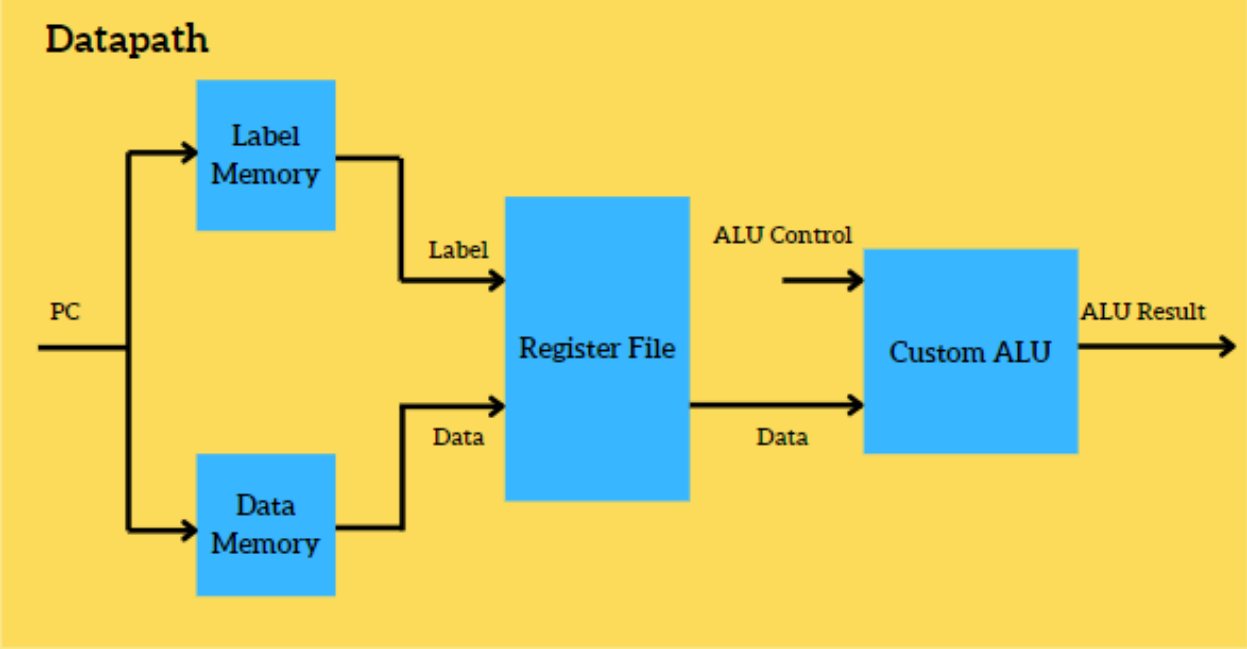

Figure 3.6: Block diagram of Datapath

The Datapath is the network that connects all functional units (Register Files, ALU, Memory) and carries out the data processing. It includes buses and multiplexers to route data between different components, facilitating the movement of data needed for computations. The Datapath ensures that data flows smoothly between the units, enabling efficient execution of instructions. The Datapath module serves as the core computational engine within the processor architecture, integrating essential components such as the Arithmetic Logic Unit (ALU), Register Files, and two Block RAMs (BRAMs) for data and label storage. This module orchestrates the flow of data and instructions through the processor pipeline, enabling efficient computation and data manipulation.

3.5 Register Files

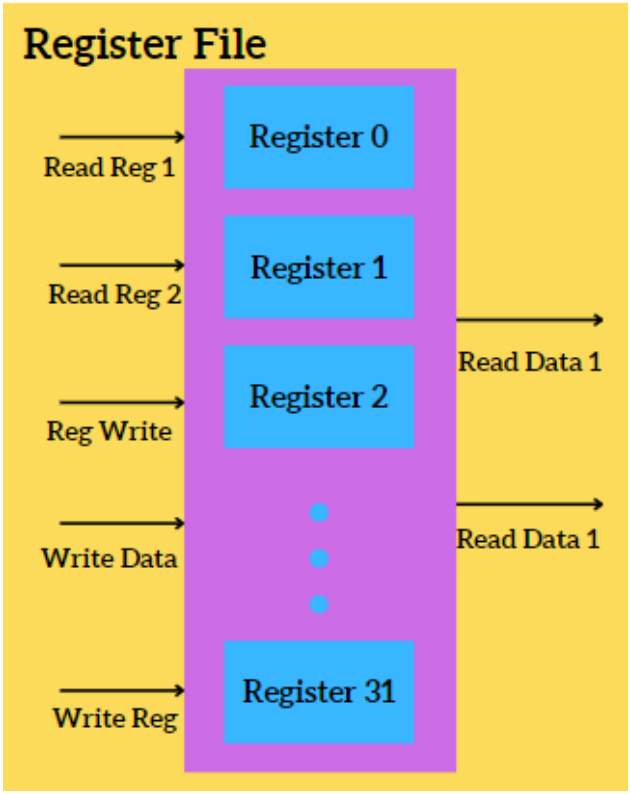

Figure 3.7: Block diagram of Register File

Register Files Overview

Register Files are fast storage registers used for storing operands and computation results, enabling simultaneous reading from multiple registers and writing to registers. In the context of k-NN, they hold dataset points, intermediate values, and results for distance calculations and comparisons.

Functionality Overview

- **Register Configuration:**
  - Number of Registers: 32 registers, each 32 bits.
- **Operational Capabilities:**
  - **Read Operations:** Simultaneous reading from two registers in a single clock cycle.

– **Write Operation:** Writing to one register in a single clock cycle.

Detailed Operation

- **Reading from Registers:** Dual-read capability allows concurrent access to two registers in one clock cycle, optimizing instruction execution.
- **Writing to Registers:** Supports writing to a single register per clock cycle for storing computation results, intermediate values, or data from memory.

3.6 Arithmetic Logic Unit (ALU)

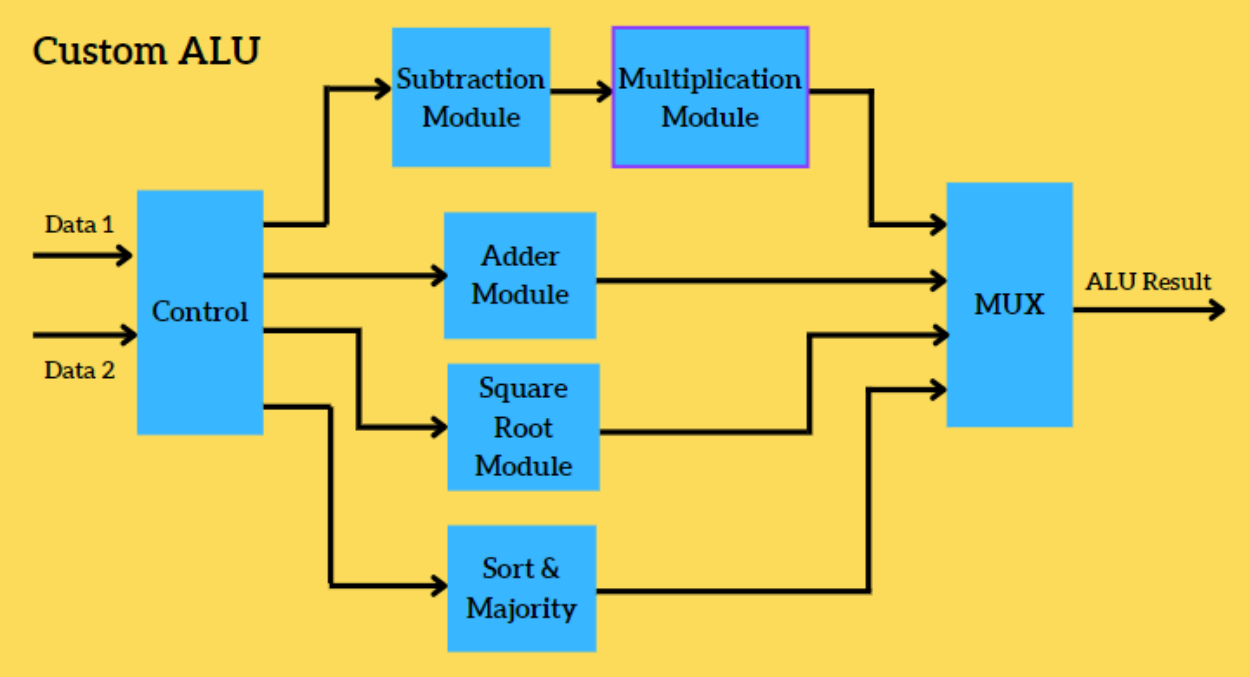

Figure 3.8: Block diagram of ALU

Arithmetic Logic Unit (ALU) Overview

The ALU performs arithmetic and logic operations essential for k-NN, including addition, subtraction, multiplication, and comparisons for distance calculations. It handles both integer and floating point arithmetic, and specialized tasks like sorting and majority determination.

Functionality Overview

- **Integer Operations:**
  - Integer Addition (ALU Control: 0110)
  - Integer Multiplication (ALU Control: 0111)
- **Floating Point Operations:**
  - Floating Point Addition (ALU Control: 0001)
  - Floating Point Square Root (ALU Control: 0010)
  - Floating Point Subtraction and Squaring (ALU Control: 0000)
- **Specialized Operations:**
  - Sorting and Majority (ALU Control: 0011): Sorts floating point values and determines majority labels based on sorted results.

Design and Implementation

The ALU module uses digital logic circuits controlled by signals for specific operations. It optimizes performance for integer and floating point computations, supporting various k-NN algorithm tasks efficiently.

3.7 Block Memory

Block Memory is used to store the dataset, intermediate distances, and final results in k-NN computations. It includes memory for dataset points, the query point, and computed distances, facilitating efficient read and write operations crucial for distance calculations and sorting.

Functional Operation

- **Data Processing Flow:**
  - **Instruction Fetch:** The Datapath fetches instructions via the Instruction Fetch Unit (IFU), directing them for execution.

- **Data Operations:** ALU operations perform arithmetic calculations and logical comparisons on data from Register Files or BRAMs.
- **Data Storage:** Dedicated BRAMs store and retrieve data points and labels, supporting operations like sorting, classification, or pattern recognition.

3.8 RISC-V ISA

3.8.1 RV32I Base Integer Instruction Set

RISC-V is an open-source instruction set architecture (ISA) based on the Reduced Instruction Set Computing (RISC) principles. It aims to provide a standardized, modular, and extensible ISA that can be freely used, modified, and implemented by anyone, unlike proprietary ISAs such as x86 and ARM.

Integer Register-Register Operations

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	src2	src1	ADD/SLT/SLTU	dest	OP
0000000	src2	src1	AND/OR/XOR	dest	OP
0000000	src2	src1	SLL/SRL	dest	OP
0100000	src2	src1	SUB/SRA	dest	OP

Figure 3.9: Integer Register-Register Operations

RV32I defines arithmetic R-type operations where ADD and SUB use rs1 and rs2 as operands, storing results in rd. SLT and SLTU compare values, while AND, OR, and XOR perform bitwise logical operations. SLL, SRL, and SRA execute shifts based on rs2's lower 5 bits.

Integer Load and Store Instructions

Integer Register-Immediate Instructions

31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
I-immediate[11:0]	src	ADDI/SLTI[U]	dest	OP-IMM
I-immediate[11:0]	src	ANDI/ORI/XORI	dest	OP-IMM

Figure 3.10: Integer Load and Store Instructions

RV32I is a load-store architecture where memory access is restricted to load and store instructions. Arithmetic operations manipulate CPU registers. It supports a 32-bit byte-addressable, little-endian user address space. Load (I-type) and store (S-type) instructions calculate effective addresses by adding register rs1 to a sign-extended 12-bit offset. Loads transfer memory values to register rd, ensuring data integrity, while stores move the value from register rs2 into memory.

Conditional Branches Instructions

31	30 25	24 20	19 15	14 12	11 8	7	6 0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
1	6	5	5	3	4	1	7
offset[12,10:5]		src2	src1	BEQ/BNE	offset[11,4:1]		BRANCH
offset[12,10:5]		src2	src1	BLT[U]	offset[11,4:1]		BRANCH
offset[12,10:5]		src2	src1	BGE[U]	offset[11,4:1]		BRANCH

Figure 3.11: Conditional Branches Instructions

Branch instructions in RV32I use the B-type format with a 12-bit B-immediate field for signed offsets. They add this offset to the current PC to determine the target address, comparing values from two registers. BEQ and BNE branch based on whether registers rs1 and rs2 are equal or unequal, respectively. BLT and BLTU branch if rs1 is less than rs2, using signed and unsigned comparisons, respectively.

3.8.2 RV32 "F" Standard Extension for Single-Precision Floating-Point

The "F" standard extension adds single-precision floating-point computational instructions compliant with the IEEE 754-2008 arithmetic standard, suitable for applications like the kNN algorithm.

Single-Precision Load and Store Instructions

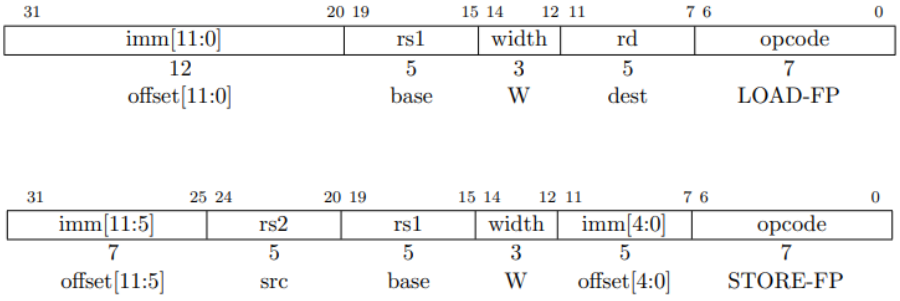

Figure 3.12: Single Precision Load Store Registers

Floating-point load and store instructions use the same base+offset addressing mode as the integer base ISA, with a base address in register rs1 and a 12-bit signed byte offset. The FLW instruction loads a single-precision floating-point value from memory into floating-point register rd. FSW stores a single-precision value from floating-point register rs2 to memory.

Single-Precision Floating-Point Computational Instructions

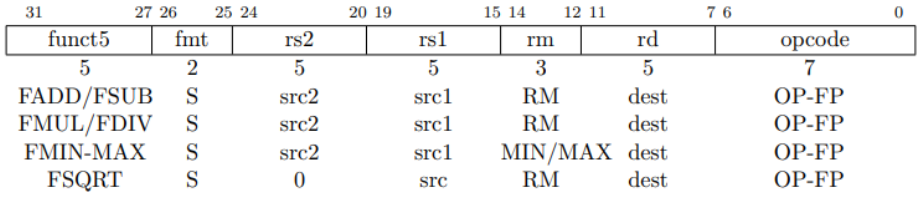

Figure 3.13: Single-Precision Floating-Point Computational Instructions

RISC-V's floating-point arithmetic instructions include operations like FADD.S (addition), FSUB.S (subtraction), FMUL.S (multiplication), FDIV.S (division), FMIN.S (minimum), FMAX.S (max-

imum), and FSQRT.S (square root) for 32-bit single-precision floating-point numbers. These instructions ensure efficient computation and accuracy, with specific rounding modes selected via the rm field.

3.8.3 Custom Floating Point Instructions

Custom instructions derived from the RV32F extension optimize the RISC-V architecture for a hardware accelerator for the k-Nearest Neighbors (k-NN) algorithm, integrating multiple operations into single instructions to boost efficiency.

Custom Instruction Design and Functionality

The custom instructions streamline computational tasks in the k-NN algorithm, reducing the instruction count and enhancing processing speed.

1. **SUBMUL**

   Instruction Details:

   - **Format**: Custom R-type
   - **Description**: Performs subtraction between two floating-point values from source registers rs1 and rs2. The resulting value is then multiplied by itself, and the final result is stored in destination register rd.
   - **Operation**: rd = (rs1 - rs2) * (rs1 - rs2)

2. **MKTFM (Move K to Sort Module)**

   Instruction Details:

   - **Format**: I-type
   - **Description**: Moves a specific value (k) from the register file to the Sort module for sorting operations.
   - **Operation**: rd = rs1 + imm, where imm = 0

3. **MLTFM (Move Label to Sort Module)**

   Instruction Details:

- **Format**: I-type
- **Description**: Moves label information to the Sort module, useful for categorization during sorting.
- **Operation**: rd = rs1 + imm, where imm = 0

4. **MSTFM (Move Square Root to Sort Module)**

   Instruction Details:

   - **Format**: I-type
   - **Description**: Moves a computed square root value to the Sort module, essential for distance calculations in sorting.
   - **Operation**: rd = rs1 + imm, where imm = 0

CHAPTER 4

RESULTS

4.1 kNN Accelerator

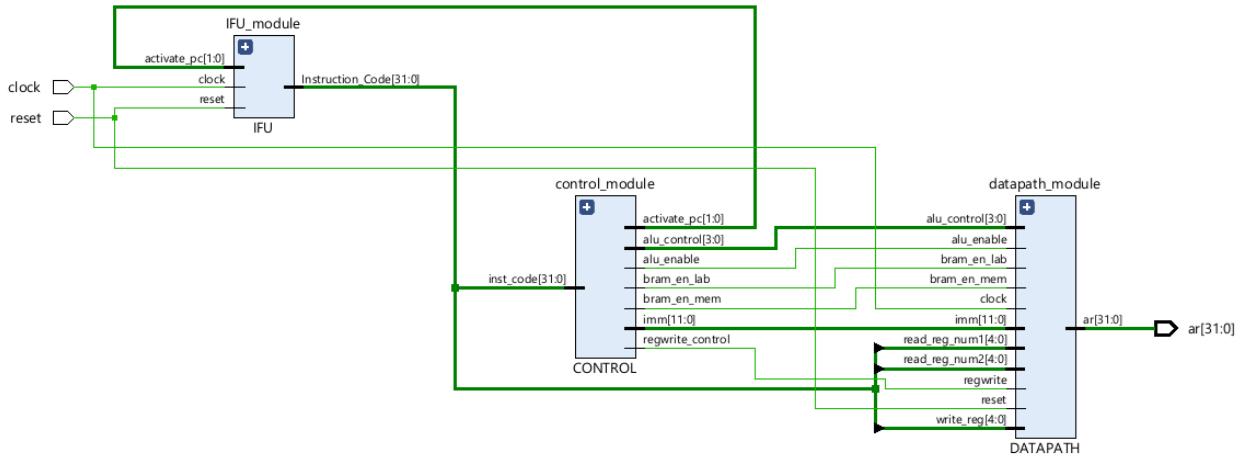

Figure 4.1: RTL Synthesis of kNN ACCELERATOR

The result obtained by RTL synthesis of the kNN Accelerator is shown in Figure 4.1. The netlist contains the IFU, Control unit and the Datapath module.

4.2 Instruction Fetch Unit

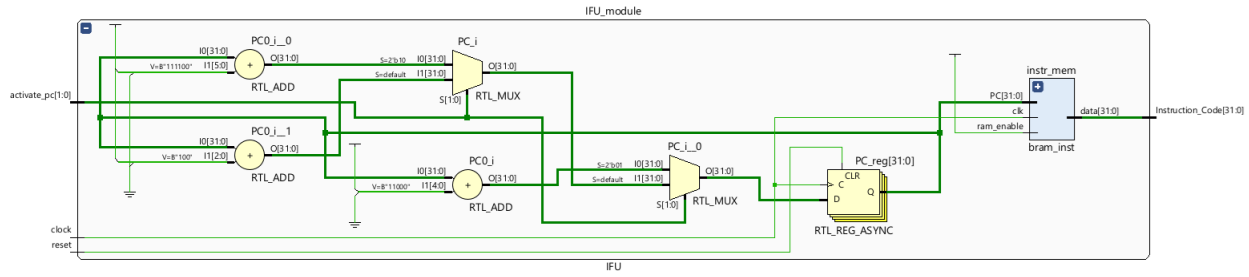

Figure 4.2: RTL Synthesis of Instruction Fetch Unit

The result obtained by RTL synthesis of the IFU is shown in Figure 4.2. The netlist contains the instruction memory and the program counter.

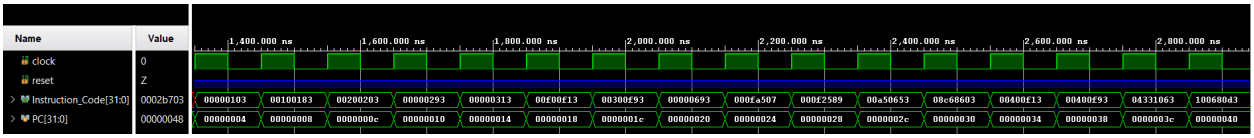

Figure 4.3: Waveforms of Instruction Fetch Unit

The functionality of the IFU is verified and the resulting waveforms are obtained by simulating the IFU, depicted in the Figure 4.3. Initially, the Program Counter (PC) starts at 0, and the instruction memory is preloaded with the necessary instructions. On each positive clock cycle, the Instruction Fetch Unit (IFU) retrieves the instruction located at the address specified by the current value of the PC. After fetching the instruction, the IFU increments the PC by 1, preparing it for the next instruction fetch. This sequential incrementing of the PC ensures that with every positive clock edge, the IFU continues to fetch the next instruction in the sequence. As a result, the IFU maintains a steady and systematic flow of instruction fetching, enabling smooth execution of the program.

4.3 Control Unit

The result obtained by RTL synthesis of the Control Unit is shown in Figure 4.4. The netlist contains the logic to provide the different ALU control values.

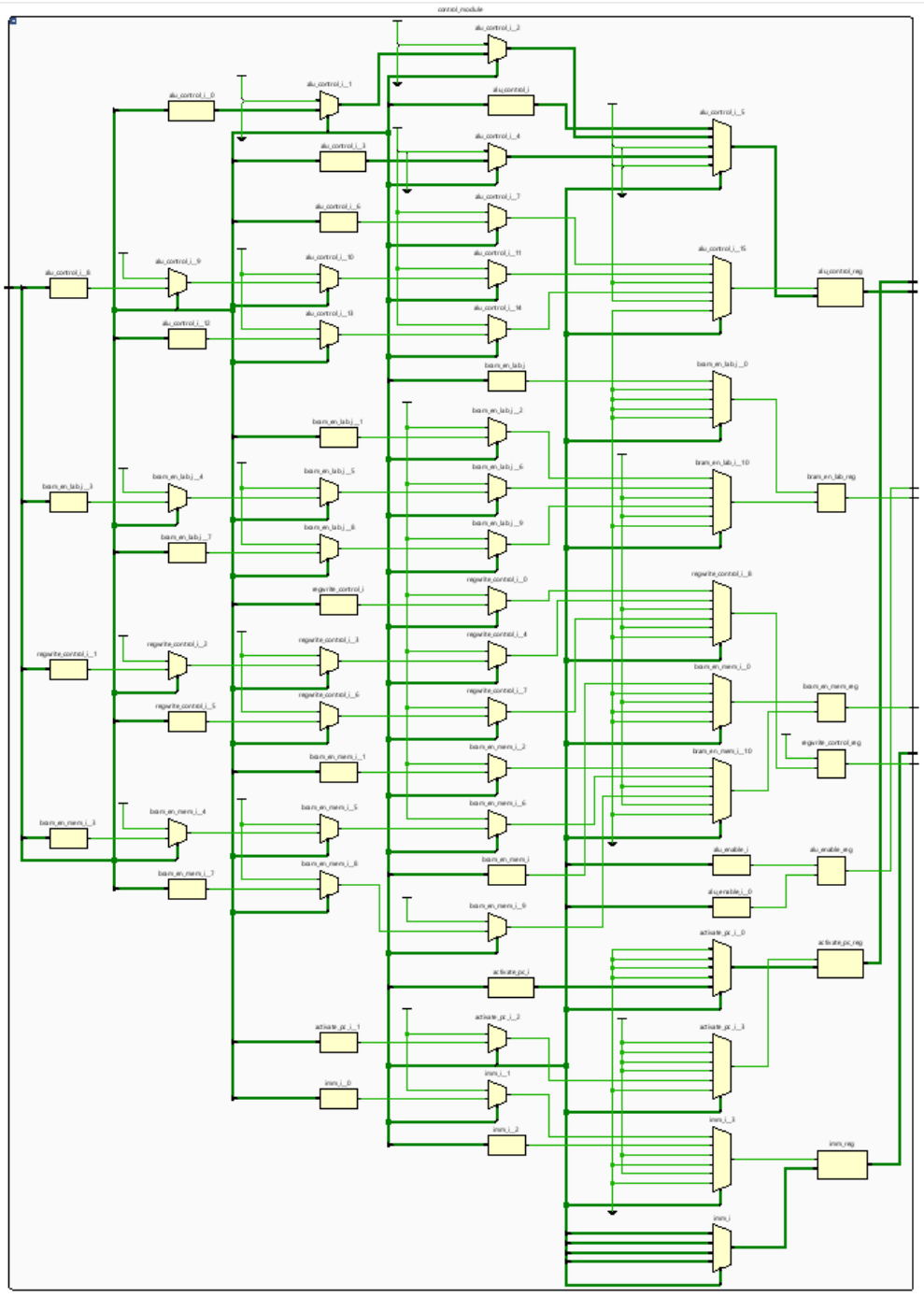

Figure 4.4: RTL Synthesis of Control Unit

The functionality of the Control Unit is verified and the resulting waveforms are obtained by simulating the Control Unit, depicted in the Figure 4.5.

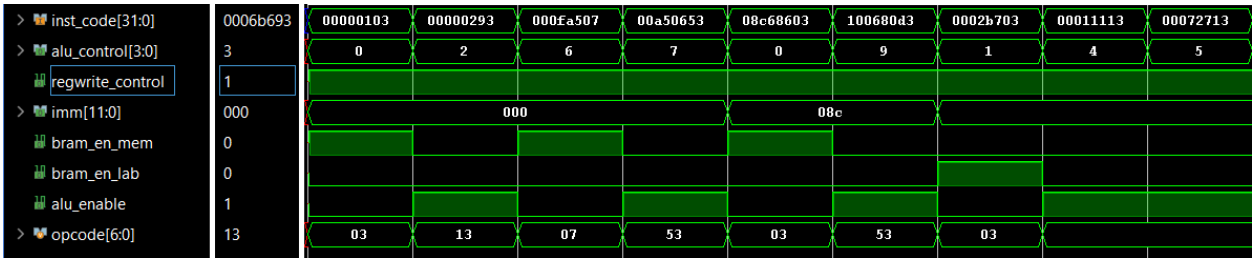

Figure 4.5: Waveforms of Control Unit

Initially, the Control Unit (CU) decodes the instruction codes and sets the corresponding control signals. For the instruction with opcode 03 (`00000103`), it enables register write and BRAM data memory. Conversely, for the instruction with opcode 53 (`00A50653`), it enables the ALU and BRAM data memory. On each positive clock edge, the CU decodes the current instruction and adjusts the control signals accordingly. This ensures that the correct operations are performed for each opcode, maintaining the proper execution flow of the instructions.

4.4 Datapath

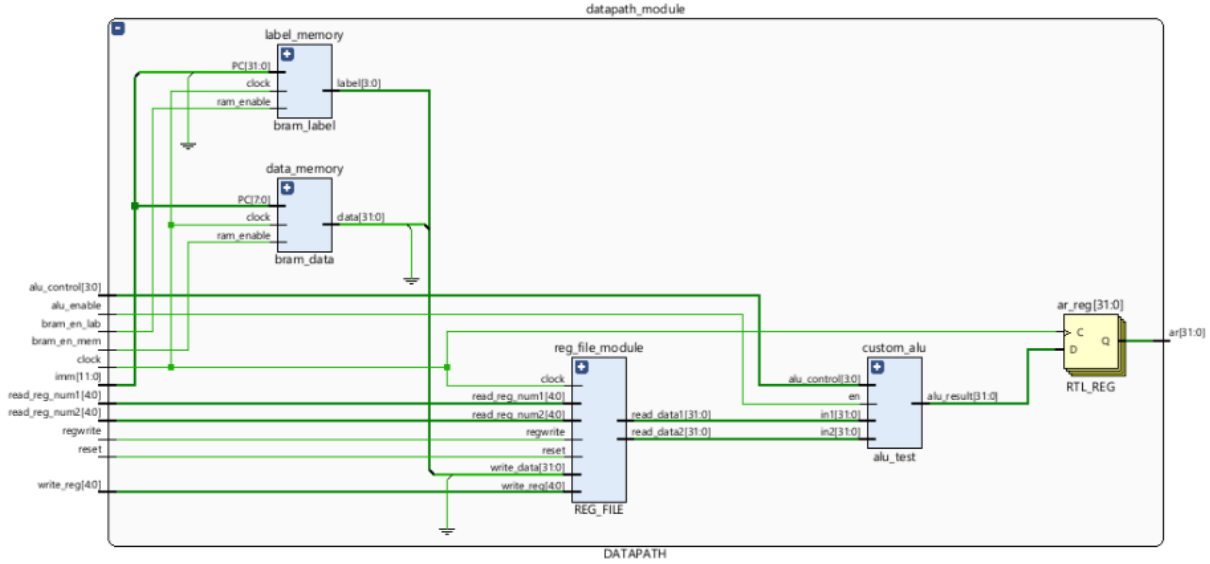

Figure 4.6: RTL Synthesis of Datapath

The result obtained by RTL synthesis of the Datapath is shown in Figure 4.6. The netlist contains the label and data memories, register file module and the custom ALU module.

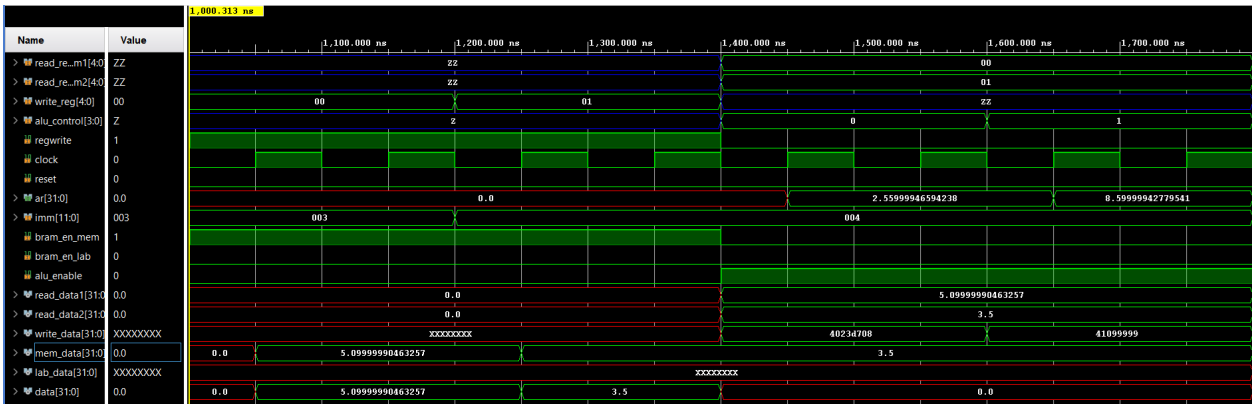

Figure 4.7: Waveforms of Datapath

The functionality of the Datapath is verified and the resulting waveforms are obtained by simulating the Datapath, depicted in the Figure 4.7. When `regwrite` = 1 and `write_reg` is 0,

`bram_en_mem` = 1, `imm` = 03, data is read from address 03 in BRAM memory and written to register 0. When `regwrite` = 1 and `write_reg` is 1, `bram_en_mem` = 1, `imm` = 04, data is read from address 04 in BRAM memory and written to register 1. When `read_reg_1` = 00, `read_reg_2` = 01, `alu_control` = 0000 (subtraction and multiplication), and `alu_enable` = 1, the data is read from registers 0 and 1, and the result of the subtraction and multiplication appears on the `alu_result`. When `read_reg_1` = 00, `read_reg_2` = 01, `alu_control` = 0001 (addition), and `alu_enable` = 1, the data is read from registers 0 and 1, and the result of the addition appears on the `alu_result`.

4.5 Register File

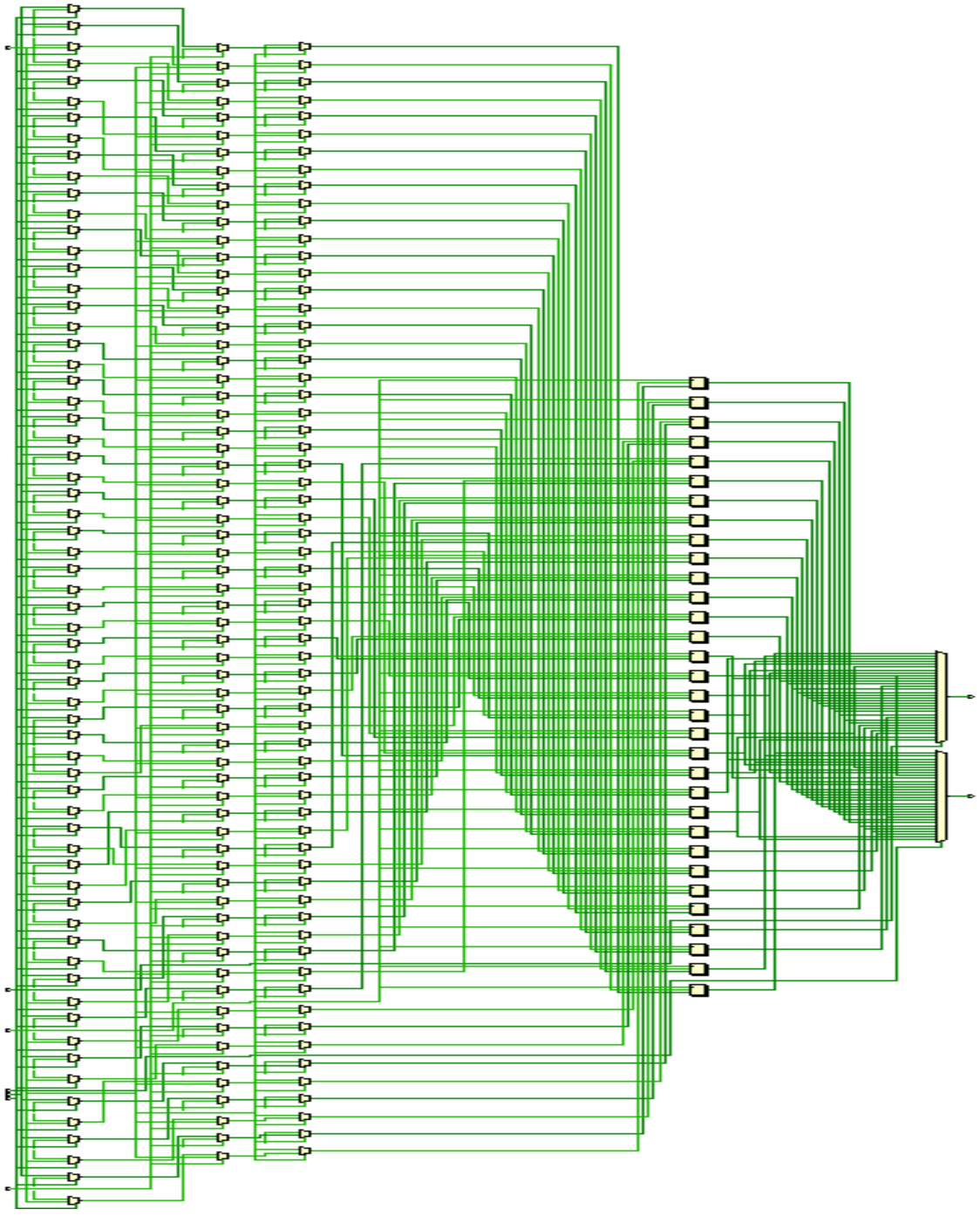

Figure 4.8: RTL Synthesis of Register File

The result obtained by RTL synthesis of the Register File is shown in Figure 4.8. The netlist contains the 32 registers.

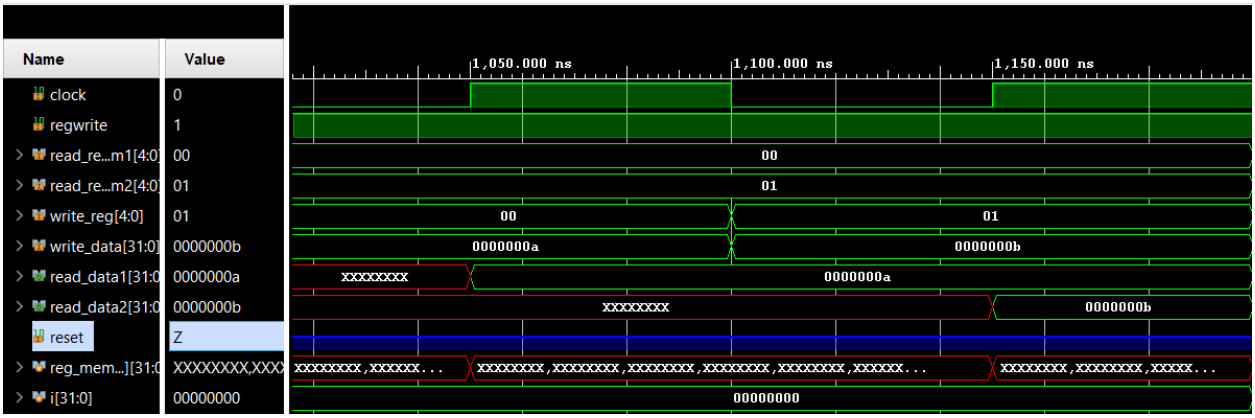

Figure 4.9: Waveforms of Register File

The functionality of the Register File is verified and the resulting waveforms are obtained by simulating the Register File, depicted in the Figure 4.9. The register file starts with default values, and read and write operations occur each clock cycle. In the first clock cycle, the value 'a' is written to register 00, and data from registers 00 ('a') and 01 (initial/default value) are read. In the second clock cycle, the value 'b' is written to register 01, and data from registers 00 ('a') and 01 ('b') are read.

4.6 Arithmetic and Logic Unit(ALU)

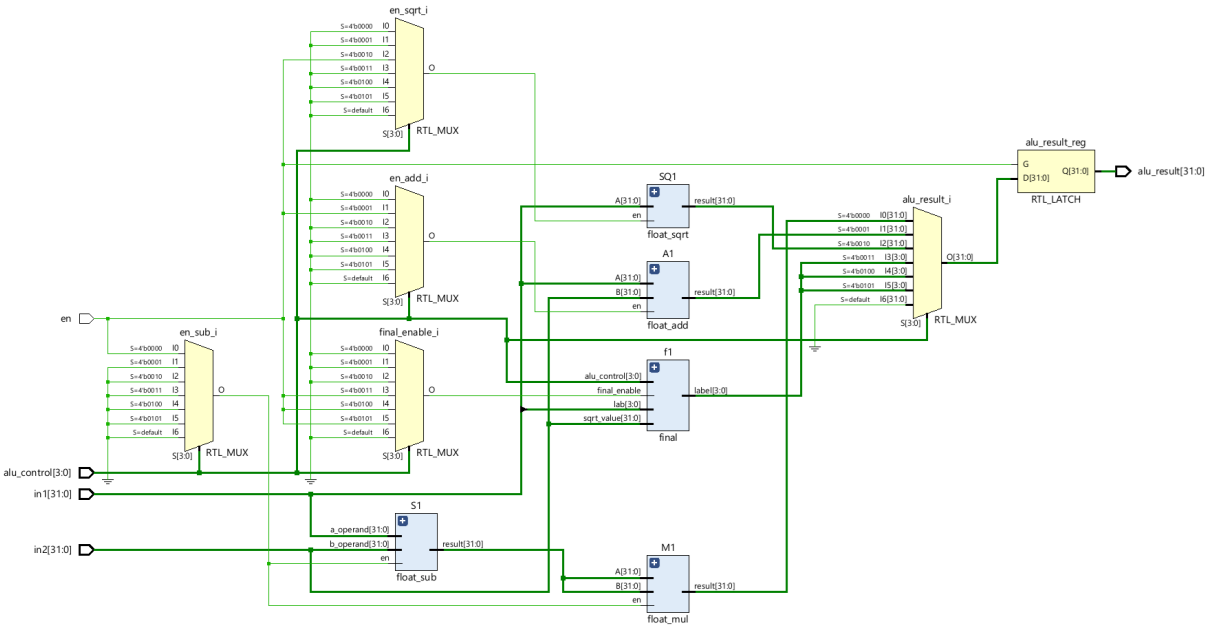

Figure 4.10: RTL Synthesis of Arithmetic Logic Unit

The result obtained by RTL synthesis of the ALU is shown in Figure 4.10. The netlist contains a subtraction module, addition module, multiplication module, square root module and a sort and majority module.

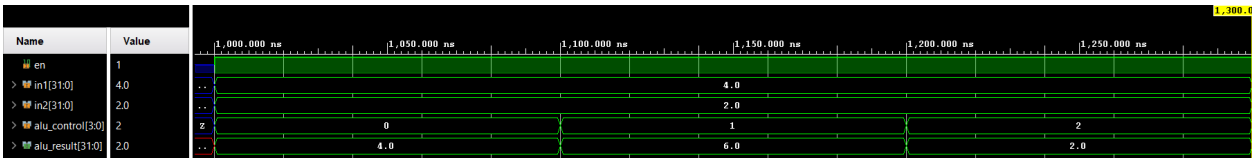

Figure 4.11: Waveforms of ALU

The functionality of the ALU is verified and the resulting waveforms are obtained by simulating the ALU which are depicted in the Figure 4.11. With ALU inputs `in1 = 4` and `in2 = 2`, the operation is determined by the `alucontrol` signal. When `alucontrol = 0`, it performs floating

point subtraction followed by squaring the result, giving `alu_result = 4`. When `alucontrol = 1`, it performs floating point addition, resulting in `alu_result = 6`. When `alucontrol = 2`, it performs the square root of `in1`, resulting in `alu_result = 2`.

References

1. Yacoub, M. H., Ismail, S. M., Said, L. A., Madian, A. H., Radwan, A. G. (2024). Reconfigurable hardware implementation of K-nearest neighbor algorithm on FPGA. AEU-International Journal of Electronics and Communications, 173, 154999.

2. Almomany, A., Ayyad, W. R., Jarrah, A. (2022). Optimized implementation of an improved KNN classification algorithm using Intel FPGA platform: Covid-19 case study. Journal of King Saud University-Computer and Information Sciences, 34(6), 3815-3827.

3. Bhattacharyya, T., Ghosal, P., Deb, S. (2024, January). Vigil: A RISC-V SoC Architecture for 2-fold Hybrid CNN-kNN based Fall Detector Implementation on FPGA. In 2024 37th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID) (pp. 372-377). IEEE.

4. He, Y. (2021, April). Design and implementation of convolutional neural network accelerator based on RISCV. In Journal of Physics: Conference Series (Vol. 1871, No. 1, p. 012073). IOP Publishing.

5. Huang, W., Wu, H., Chen, Q., Luo, C., Zeng, S., Li, T., Huang, Y. (2021). FPGA-based high-throughput CNN hardware accelerator with high computing resource utilization ratio. IEEE Transactions on Neural Networks and Learning Systems, 33(8), 4069-4083.

6. Zhu, C., Huang, K., Yang, S., Zhu, Z., Zhang, H., Shen, H. (2020). An efficient hardware accelerator for structured sparse convolutional neural networks on FPGAs. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 28(9), 1953-1965.

7. Kamaleldin, A., Hesham, S., G ohringer, D. (2020). Towards a modular RISC-V based many-core architecture for FPGA accelerators. IEEE Access, 8, 148812-148826.

8. Zhang, G., Zhao, K., Wu, B., Sun, Y., Sun, L., Liang, F. (2019, April). A RISC-V based hardware accelerator designed for Yolo object detection system. In 2019 IEEE International Conference of Intelligent Applied Systems on Engineering (ICIASE) (pp. 9-11). IEEE.

9. Lu, L., Xie, J., Huang, R., Zhang, J., Lin, W., Liang, Y. (2019, April). An efficient hardware accelerator for sparse convolutional neural networks on FPGAs. In 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM) (pp. 17-25). IEEE.

10. Danopoulos, D., Kachris, C., Soudris, D. (2019, July). FPGA acceleration of approximate knn indexing on high-dimensional vectors. In 2019 14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC) (pp. 59-65). IEEE.