IMPLEMENTATION OF A NEURAL NETWORK-BASED ASIC CHIP FOR MOBILE DNA DEVICES

ZHONGPAN WU

A DISSERTATION SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCES
YORK UNIVERSITY
TORONTO, ONTARIO

JAN, 2025

Abstract

Portable DNA sequencing, particularly using nanopore technology, has the potential to revolutionize genomics by making it accessible in a wide range of environments. However, current state-of-the-art devices face significant challenges due to the lack of integrated bioinformatics processing capabilities. This research addresses these challenges by developing specialized System-on-Chip (SoC) architectures designed for real-time bioinformatics analysis, integrating both a machine learning (ML)-based basecalling accelerator and an Edit Distance (ED) accelerator for sequence comparison.

The proposed SoC architecture, based on an open-source RISC-V core, features hardware accelerators tailored for the computational demands of nanopore DNA sequencing. Performance evaluation was conducted in two stages: first through FPGA prototyping, followed by integration into a fabricated SoC. The FPGA prototyping demonstrated nearly 2,000× speedup for ML-based basecalling compared to a standalone RISC-V core, while maintaining an accuracy rate of 83.7%. It also

showed an 11.5x and 1.2x energy efficiency improvement over x86 CPUs and high-end GPUs, respectively. The ED accelerator for sequence comparison achieved a 538x boost in energy efficiency compared to commercial CPUs.

The fabricated SoC, implemented in a 22-nm CMOS process, successfully demonstrated the feasibility of integrating advanced bioinformatics tasks into a single, power-efficient chip. Evaluation of the fabricated SoC confirmed its capability for real-time, mobile DNA sequencing with high accuracy, reduced power consumption, and significantly improved processing speed, all while reducing dependency on external computational devices. This research represents a significant step towards realizing a fully integrated, stand-alone DNA sequencing solution, capable of performing comprehensive bioinformatics analyses in real time.

Acknowledgements

First and foremost, deep gratitude is extended to my daughter, Naomi Wu, whose laughter and curiosity have been a constant source of joy and motivation throughout this journey. To my wife, Wei Gu, endless appreciation for the unwavering support, patience, and belief, even during the most challenging moments. Her love and encouragement have been the true foundation of this work.

Beyond my family, sincere appreciation goes to my supervisor, Sebastian Magierowski, for his exceptional guidance, wisdom, and mentorship. His insightful feedback and commitment to excellence have been instrumental in shaping both this research and my growth as a scholar. A heartfelt acknowledgment is also due to my co-supervisor, Ebrahim Ghafar-Zadeh, whose expertise and thoughtful advice have been invaluable throughout this process.

Special thanks to colleagues Karim Hammad, Abel Beyene, and Yunus Dawji. Karim's willingness to collaborate and engage in insightful discussions has sparked new ideas. Abel's enthusiasm and analytical perspective have been crucial in refin-

ing key aspects of this work. Yunus brought valuable teamwork and positive energy to every endeavor, helping to overcome many challenges together, which has been greatly appreciated.

Gratitude is also extended to Speidieh Asgari for her generous support, both academically and personally, and for consistently offering assistance whenever needed. Amin Savari's technical insights and constant willingness to brainstorm through challenging problems have had a significant impact on the success of this research.

To all those mentioned, profound thanks for making this journey rewarding, for the support given through every stage of this endeavor, and for the friendships and contributions that have made a lasting difference.

Table of Contents

List of Tables

List of Figures

Abbreviations

FPGA	Field-Programmable Gate Array
SoC	System on Chip
LP	Low Power
ASIC	Application-Specific Integrated Circuit
RISC-V	Reduced Instruction Set Computing Fifth Edition
NGS	Next-Generation Sequencing
HGP	Human Genome Project
SBL	Sequencing by Ligation
SBS	Sequencing by Synthesis
ONT	Oxford Nanopore Technologies
DC	direct current
DNA	Deoxyribonucleic Acid
BC	Basecalling
CPU	Central Processing Unit

GPU Graphics Processing Unit

HLS High-Level Synthesis

ML Machine Learning

EDA Electronic Design Automation

PPA Performance, Power, and Area

HMM Hidden Markov Model

RIFFA Reusable Integration Framework for FPGA

PCIe Peripheral Component Interface Express

CPU Central Processing Unit

NNs Neural Networks

CNN Convolutional Neural Network

RNN Recurrent Neural Network

RTL Register-Transfer Level

TLB Translation Lookaside Buffer

ASR Automatic Speech Recognition

RX Receive

TX Transmit

PFD Process Flow Diagram

COPS Core-Operations-Per-Second

FIFO First-In-First-Out

IP Intellectual Property

CMOS Complementary Metal-Oxide-Semiconductor

GCUPS Giga Cell Updates Per Second

ISA Instruction Set Architecture

TDP Thermal Design Power

1 Introduction

Portable DNA sequencing devices benefit greatly from the ONT nanopore sequencing technology, and the market potential for these devices is both significant and expanding. However, their broader adoption is impeded by several challenges, particularly computational and energy efficiency bottlenecks in performing basecalling and sequence alignment tasks directly on portable devices [1]. Currently, these devices rely heavily on general-purpose GPUs running on a host machine to handle such tasks. Although embedded GPUs provide some advantages, they fall short of delivering adequate performance within the strict power constraints required for portable devices. Moreover, embedded GPUs often rely on active cooling systems, which are impractical for hand-held sequencing devices, further limiting their suitability for truly portable applications.

The thesis proposes the development of a dedicated application-specific SoC, uniquely tailored to meet the demands of nanopore sequencing technology. This SoC is designed from the ground up with a focus on power efficiency, compactness,

and high-performance bioinformatics processing. By moving away from general-purpose GPUs and toward a specialized chip architecture, the thesis aims to overcome the existing limitations and enable next-generation portable sequencing devices that are smaller, faster, and more energy-efficient.

1.1 Background and Motivation

1.1.1 Nanopore Sequencing Technology

Nanopore sequencing, as depicted in Fig. 1.1, is a revolutionary technique that enables the direct reading of nucleotide sequences by threading a DNA or RNA molecule through a nanopore [2, 3]. The nanopore, represented by a genetically engineered, tube-shaped protein in the diagram, allows only one nucleic acid molecule to pass through at a time. As the DNA or RNA strand translocates through the nanopore, it disrupts the ionic current, resulting in small electrochemical current fluctuations, approximately 10 pA in amplitude [4]. These current fluctuations, which are indicative of the unique sequence of nucleotides (A, C, G, T), are recorded by a CMOS readout chip [5]. The analog signals generated are inherently noisy due to factors such as thermal fluctuations, electronic interference, and stochastic molecular interactions within the nanopore [6]. Despite the challenges, the bioinformatics engine, processes and analyzes these signals to infer the nucleotide sequence,

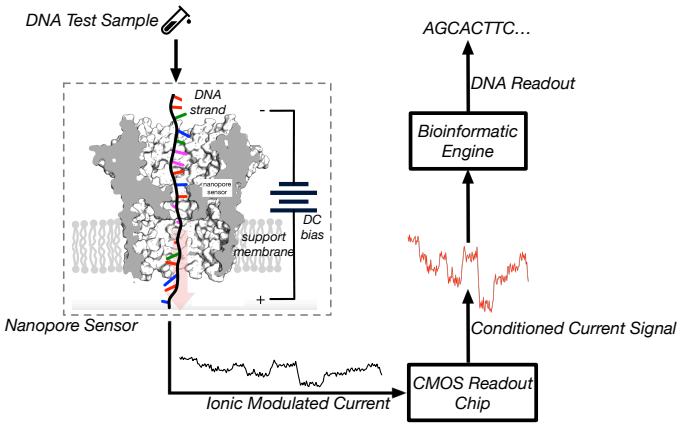

Figure 1.1: The diagram illustrates the nanopore sequencing process, highlighting DNA translocation through a nanopore sensor, signal capture by a CMOS readout chip, and data processing by the bioinformatics engine.

thereby completing the "read" of the sensed DNA or RNA. This method drastically reduces the size of sequencing systems compared to traditional desktop units, enabling portable DNA analysis [7].

A key parameter in nanopore sequencing is the "k-mers", which are sequence length of κ nucleotides (bases) in the DNA or RNA molecule translocating the nanopore at a time [7]. For example, a 5-mer represents a sequence of five consecutive nucleotides like "AGCTA". The nanopore sensor detects changes in ionic current as nucleotides pass through the pore, and these changes are influenced by the specific k-mers occupying the pore at any given time. Because the nanopore accommodates multiple nucleotides simultaneously, the ionic current at any moment is a composite signal affected by several adjacent bases within the pore. This means that the measured signal corresponds to the specific k-mer currently passing

through the nanopore [8]. Different k-mers produce distinct current levels, enabling the differentiation of sequences based on their electrical signatures.

An example of this concept is shown in the idealized ionic current signal depicted in Figure 1.2. The figure illustrates a time-series signal without noise, highlighting the discrete current levels associated with different k-mers as the DNA strand moves through the nanopore. Each segment in the signal corresponds to a specific k-mer occupying the nanopore resulting in a modulated current level. As the k-mer sequence changes due to the movement of the nucleic acid strand, the ionic current shifts to a new level creating a step-like pattern. This pattern enables the identification of nucleotide sequences based on the observed current levels.

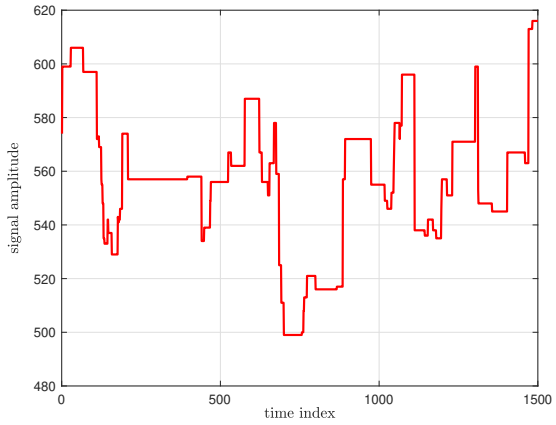

Figure 1.2: An example of the time-series signal available from a nanopore sensor. Each level is indicative of a new portion of DNAs (k-mer) translocating through the sensor.

Once the raw signals are captured, they are processed through a complex pipeline to generate accurate genomic sequences. This pipeline, depicted in Fig-

ure 1.3, consists of several stages that transform digitized electrical signals into high-quality genomic assemblies.

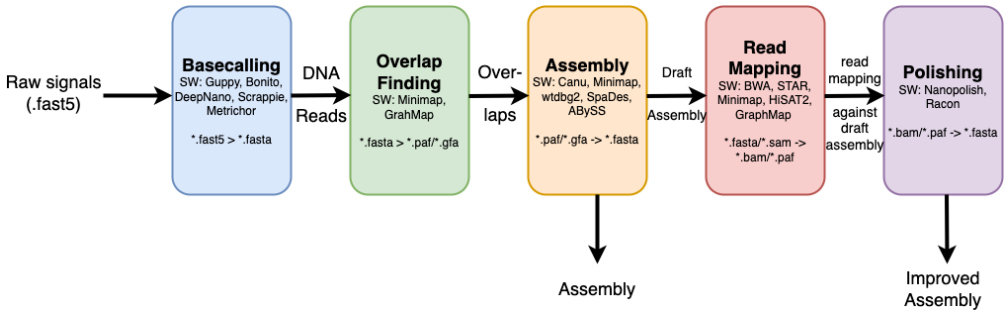

Figure 1.3: An illustrative representation of the nanopore sequencing pipeline, showing the transformation from raw electrical signals to improved genomic assemblies through basecalling, overlap finding, assembly, read mapping, and polishing stages.

The pipeline begins with basecalling, the first and most computationally demanding stage of nanopore sequencing. This task involves interpreting complex, noisy signals to determine the nucleotide sequence of the DNA or RNA molecule. Several challenges make basecalling difficult: first, the electrical signals are influenced by k-bases simultaneously, making it hard to accurately identify the corresponding k-mers combination especially for nanopore models with relatively large values of k [9]. Second, the signals are inherently noisy, affected by experimental conditions, nanopore variations, and stochastic molecular behaviors [10]. To overcome these challenges, sophisticated computational algorithms and machine learning models are employed [11, 12, 13, 14]. These models are trained on known sequences to learn the relationship between k-mers and their corresponding signal

patterns. By recognizing these patterns, basecalling software predicts the most likely nucleotide sequence that generated the observed signals.

After basecalling, the next step in the nanopore sequencing pipeline is the overlap finding where overlapping regions between DNA reads are identified. This step is crucial for genome reconstruction as it aligns and merges reads that cover overlapping regions of the DNA strand. Overlap finding is conceptually related to the sequence alignment task as it requires comparing segments of DNA to determine how closely they match. In particular, the process involves measuring similarities and differences between sequences based on edit distance calculations [15]. ED measures the number of changes—insertions, deletions, or substitutions—required to transform one sequence into another, making it an important computational tool for overlap detection. By efficiently calculating these similarities, overlap finding ensures that reads are correctly assembled into a reference genome.

Once overlapping reads are identified, the next stage is assembly where the reads are combined to generate a draft genome. Assembly tools like Canu and Minimap [16, 17] take the overlapping regions and merge them into contiguous sequences. This process results in a draft sequence, which, while largely complete, may still contain gaps or errors due to sequencing imperfections or unresolved overlaps.

Following the assembly stage, the read mapping stage aligns the raw reads

back to a reference genome to identify discrepancies or gaps. Tools like BWA and Minimap [17, 18] are used to refine the assembly ensuring accuracy by detecting areas where the reads and the reference genome do not perfectly match. This step is essential for correcting errors introduced during assembly.

Finally, the polishing stage improves the accuracy of the draft genome. Tools like Nanopolish and Racon [19, 20] remap the reads against the draft assembly, identifying and correcting errors to produce a higher-quality polished genome. Polishing ensures that the final output is a reliable representation of the original DNA sequence suitable for downstream analysis or applications.

Nanopore sequencing technology, through its unique approach to processing raw electrical signals and assembling genome sequences, offers several advantages over traditional sequencing methods. These mechanisms make nanopore sequencing a powerful tool for various applications in genomics and molecular biology. Some of the key advantages include:

- **Long Read Lengths:** Capable of reading lengths exceeding hundreds of kilobases, facilitating the assembly of complex genomes and the detection of structural variations.

- **Real-Time Data Acquisition:** The technology allows for real-time sequencing, providing immediate access to data as the nucleic acid strand passes

through the nanopore. This feature is beneficial for time-sensitive applications such as clinical diagnostics, outbreak monitoring, and environmental surveillance [21].

- **Portability:** Nanopore sequencing devices like the MinION [22] are compact and portable, enabling sequencing in remote or resource-limited settings. This portability expands the reach of genomic analysis to fieldwork, point-of-care diagnostics, and educational environments [23, 24, 25].

- **Direct Sequencing of Native Molecules:** Nanopore sequencing can analyze native DNA or RNA molecules without the need for amplification or labeling [26].

- **Versatility in Sample Types:** The technology is capable of sequencing a wide range of nucleic acid types, including single-stranded DNA, double-stranded DNA, RNA, and even modified nucleic acids. This versatility makes it suitable for diverse applications.

- **Cost-Effectiveness:** As the technology matures, nanopore sequencing offers a cost-effective solution for whole-genome sequencing and targeted sequencing projects, making high-throughput sequencing more accessible.

These advantages stem from the fundamental design of nanopore sequencing, where the direct detection of nucleotides passing through a pore allows for flexibility

and scalability that is not easily achievable with other sequencing platforms. The ability to produce long reads and perform real-time analysis, combined with the portability of the devices, positions nanopore sequencing as a transformative tool in genomics research and clinical applications.

Despite these advantages, challenges remain in achieving high accuracy due to the complex nature of the signals and inherent noise. Continued development of computational methods and hardware solutions aims to address these challenges, enhancing the utility and reliability of nanopore sequencing technology. Such methods and solutions are the main subject of this thesis.

1.1.2 Nanopore Sequencing Limitation

While nanopore sequencing technology holds immense promise, current portable sequencing devices such as the OxfordNanopore™ MinION MK1B and MK1C exhibit limitations that impact their performance, particularly in real-time sequencing applications. Recognizing these limitations is essential for developing solutions that enhance the utility and efficiency of nanopore sequencing in various settings.

The MinION MK1B [27] is a compact portable nanopore sequencer that relies on an external computer connected via USB for data processing and basecalling—the computational task of converting raw electrical signals into nucleotide sequences. This dependency on an external computing device diminishes the overall porta-

bility and convenience of the system. Meanwhile, the basecalling speed is vastly relies on the performance of the host machine, so that the real-time sequencing is an uncertainty in some cases (e.g., field-based sequencing applications [28, 29]). Additionally, the need for continuous power supply for both the sequencer and the external computer limits its applicability in remote or resource-limited environments.

To mitigate some of these challenges, the MinION MK1C was introduced with an integrated compute module, enabling on-board data processing without the need for an external computer. The MK1C features a touchscreen interface and is designed to function as a standalone device for nanopore sequencing. It incorporates an embedded system capable of handling the computational demands of basecalling, aiming to enhance portability and ease of use.

The MK1C utilizes an NVIDIA™ Jetson TX2 module to perform the intensive computations required for real-time basecalling. The Jetson TX2 features a 256-core NVIDIA™ Pascal GPU and 8 GB of LPDDR4 memory, offering up to 1.33 teraflops (TFLOPs) of computational performance (FP16). Its power consumption ranges from approximately 7.5 watts to 15 watts, depending on the performance mode. While the integration of such a powerful GPU enhances the device's computational capabilities, the MK1C still faces significant challenges in performing real-time sequencing, especially when processing data from all available channels

Figure 1.4: Oxford Nanopore MionION portabel Seqeuncing Devices: **Left**: MinION MK1B [30]; **Right**: MinION MK1C. Compared to MK1B, MK1C contains an embeded GPU for basecalling on device [31].

at maximum throughput.

To elucidate these limitations, it is essential to consider the data rates involved in nanopore sequencing. The MinION flow cell contains up to 512 nanopore channels (N_{channels}) that can operate simultaneously. Each channel samples the ionic current at a rate of up to 4 kHz (R_{sample}), resulting in the generation of over 2 million samples per second that require processing in real-time:

$$S_{\text{total}} = N_{\text{channels}} \times R_{\text{sample}} = 512 \times 4000 = 2{,}048{,}000\,\text{samples/second} \tag{1.1}$$

Processing this immense volume of data in real-time requires highly efficient computational resources. The Jetson TX2, being a general-purpose GPU, is not specifically optimized for the specialized computations involved in basecalling neu-

ral networks. This lack of specialization leads to inefficiencies in computational performance and energy consumption. The high power consumption, ranging up to 15 watts, and the associated heat generation pose challenges for thermal management and limit the device's battery life, which are critical factors for portable and field-deployable sequencing devices.

Moreover, the Jetson TX2's architecture is not tailored to the specific data flow and computational patterns of basecalling algorithms. Neural network models used in basecalling involve operations like matrix multiplications and convolutions that can benefit significantly from hardware acceleration tailored to these tasks. General-purpose GPUs may not provide the most efficient execution of these operations, resulting in sub-optimal performance and longer processing times per sample.

These limitations highlight a significant gap between the current nanopore sequencing technology and the need for a portable and energy-efficient device with real-time performance and high basecalling accuracy without reliance on bulky hardware or excessive power consumption.

1.1.3 Motivation and Design of the Proposed ASIC for Mobile DNA Sequencing

The limitations of current nanopore sequencing devices, such as the MinION MK1B and MK1C, emphasize the need for more efficient and portable solutions capable of performing real-time, high-accuracy sequencing without excessive power consumption or reliance on bulky external hardware. Developing a specialized ASIC specifically targeted for nanopore sequencing offers a promising avenue to address these challenges.

An ASIC designed specifically for DNA sequencing applications offers substantial improvements in computational efficiency, power consumption, and processing speed. By tailoring the hardware to execute basecalling algorithms optimally, an ASIC can perform the necessary computations more efficiently than general-purpose GPUs. This specialization leads to faster processing times per sample, enabling real-time analysis of the high-volume data streams generated by nanopore sequencers.

The proposed system-on-chip (SoC), implemented as an ASIC, leverages the RISC-V architecture, an open-standard Instruction Set Architecture (ISA) that offers several compelling reasons for its selection. Firstly, RISC-V is open-source, eliminating licensing fees and restrictions associated with proprietary architectures.

This availability reduces development costs and fosters innovation by allowing designers to modify and extend the ISA to meet specific application needs. Secondly, the modular and extensible design of RISC-V enables the inclusion of custom extensions and specialized instructions. This flexibility is crucial for bioinformatics applications, where specialized computational tasks like basecalling can benefit from tailored hardware support. Additionally, the growing ecosystem of RISC-V tools and development environments facilitates efficient development and optimization of both software and hardware components within the ASIC-based SoC.

1.2 The SoC Functional Requirements

The proposed SoC supports multiple use cases in mobile DNA sequencing. Around these two stages, the SoC can support applications such as pathogen detection, pre-basecalling, and real-time species classification. The following sections describe each use case, the challenges they present, and how the SoC architecture addresses these challenges.

1.2.1 Real-time Basecalling & Alignment

The primary focus of the proposed SoC is to achieve effective real-time basecalling and alignment, addressing the challenges of processing speed, power efficiency, and accuracy. Real-time basecalling involves converting raw signals from the sequenc-

ing device into nucleotide sequences (A, T, C, G), while alignment matches these sequences to a reference genome to identify genetic variations or pathogens. These processes are often bottlenecks in the sequencing workflow, particularly when rapid results are needed. To overcome these challenges, the SoC is designed to achieve a processing speed of 2 million samples per second, as specified in Eq. 1.1, while maintaining power consumption under 1 Watt for practical use in field environments.

1.2.2 Pathogen/Virus Detection

Pathogen and virus detection requires rapid identification of genetic sequences to provide timely interventions especially in healthcare [32] and public health [33] scenarios. The primary challenge here is to process sequencing data quickly enough to detect pathogens in real-time without requiring extensive computational infrastructure.

The proposed SoC addresses this challenge by using its integrated accelerators to perform both basecalling and alignment efficiently. By doing so, it provides a self-contained system that can handle the entire process from raw signal acquisition to identification of pathogens in the field. This integration reduces the need for cloud-based processing and ensures that sensitive health data remains localized while also accelerating the response time for diagnostics.

1.2.3 Pre-filtering for Basecalling

Data pre-filtering is a crucial step that reduces the volume of data before it is processed by downstream algorithms. In a typical sequencing process, raw signals generated by sensors contain noise that can negatively impact the accuracy of basecalling. The SoC employs specialized filters to efficiently identify and retain high-quality signal data, thereby reducing computational requirements and ensuring that the subsequent steps operate on cleaner and more reliable data.

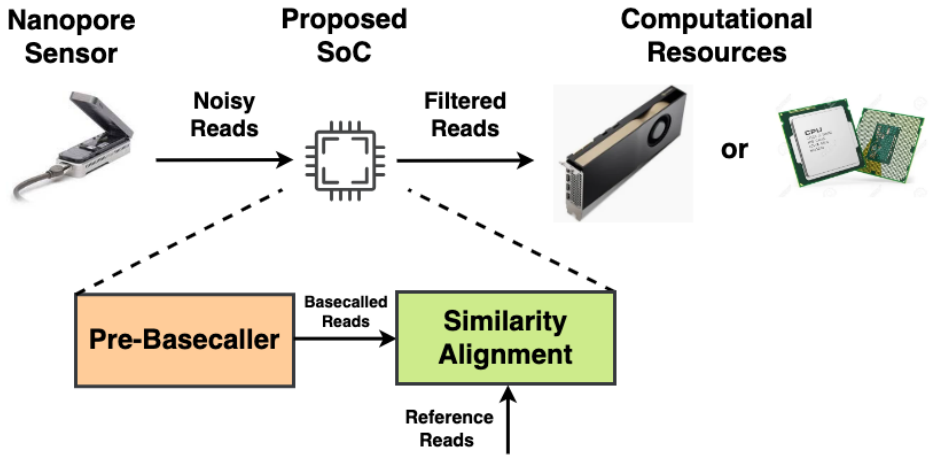

Figure 1.5: Diagram illustrating the SoC's role in pre-basecalling and filtering noisy reads before passing them to a GPU for final basecalling.

The diagram presented in Fig. 1.5 illustrates the complete pre-basecalling workflow, showcasing the essential role of the proposed System-on-Chip (SoC) within this sequencing pipeline. The nanopore sensor generates noisy reads in the form of raw signals, which are then processed by the SoC. This initial phase—referred

to as pre-basecalling—is aimed at rapidly filtering and basecalling these raw reads locally on the SoC, leveraging its specialized bioinformatics accelerators.

By performing pre-basecalling directly on the SoC, the architecture minimizes the volume of noisy data transmitted for further analysis. The filtered reads are passed along to external computational resources, such as GPUs or CPUs, for high-accuracy basecalling and further downstream analysis, if needed. This staged approach not only enhances efficiency but also makes the workflow more adaptable to different levels of available computational power. The proposed SoC should also integrates an initial similarity alignment step, which performs lightweight alignment of the pre-basecalled reads against reference sequences. This alignment, achieved within the SoC, allows for real-time detection and identification of pathogens or species, particularly in field environments where computational resources are limited.

1.2.4 Real-time Species Classification

Real-time species classification is another key use case for the SoC. The ability to quickly classify species based on their genetic material is essential in applications such as biodiversity monitoring and food safety. The challenge lies in the need to process and analyze genetic data on-site without relying on cloud-based solutions, which may introduce latency and privacy concerns.

The SoC's accelerators provide the computational power needed to perform species classification directly in the field. By running the classification algorithms locally, the SoC can reduce latency and ensure data privacy, making it particularly useful for field-based ecological studies and real-time monitoring of ecosystems.

1.3 Thesis Contribution

This thesis explores the development of a groundbreaking neural network-based application-specific integrated circuit (ASIC) designed specifically for mobile DNA sequencing devices. Built on the open-source RISC-V architecture, the proposed design leverages its modularity and extensibility to integrate custom instructions and specialized accelerators tailored for bioinformatics workflows. By addressing the computational and power efficiency challenges inherent to nanopore-based portable sequencing, this work establishes a significant milestone in the field. The contributions span both software and hardware, marking a novel and comprehensive approach to solving these challenges. The key contributions and innovative aspects include:

- **Software Contribution**
  - **HMM-Based Basecalling Algorithm**

    A hidden Markov model (HMM) basecaller was developed to illustrate

the fundamental principles of basecalling algorithms. This served as an example to demonstrate how basecalling can be implemented and provided a reference point for understanding the computational demands of the task.

– **Lightweight Neural Network Based Basecalling Model**

A complete basecalling software pipeline was built to evaluate and compare various neural network models for their suitability in portable DNA sequencing. After comprehensive consideration of factors such as computational efficiency, accuracy, and compatibility with hardware constraints, a Conv1D architecture was selected as the core of the basecaller. This led to the development of a new basecaller, SoCall, which was fully trained on GPUs using PyTorch and optimized with techniques like quantization and batch normalization folding to enhance efficiency. To ensure seamless deployment on the ASIC chip, C code was developed to integrate the basecaller with the hardware, incorporating specialized optimizations to efficiently invoke the accelerator and maximize performance.

– **Sequence Alignment Algorithm**

A sequence alignment algorithm was implemented to complement the

Edit Distance (ED) accelerator in the SoC. This algorithm facilitates fast and efficient nucleotide alignment, addressing one of the key computational bottlenecks in the nanopore sequencing workflow.

- **Hardware Contributions**

  - **HMM Accelerator Development and Evaluation**

    The hardware contributions began with the implementation of an HMM accelerator. This design was prototyped on an FPGA platform, communicating with a host PC via PCIe. The performance data gathered informed subsequent design improvements and the development of the first ASIC implementations.

  - **Customized Systolic Array Accelerator**

    Open-source acceleration architectures were reviewed, leading to the selection and customization of a systolic array-based design. This design was optimized for running the convolutional basecalling model, with FPGA implementation providing performance feedback to refine the architecture. The second ASIC chip includes a 4×4 systolic array specifically configured to meet the computational requirements of the neural network-based basecalling model.

  - **Edit Distance Engine Implementation** A dedicated hardware ac-

celerator for Edit Distance calculations was designed and implemented. This accelerator enables on-chip sequence alignment and comparison, addressing one of the key bottlenecks in the nanopore sequencing pipeline.

The contributions presented in this thesis span both the software and hardware domains. On the software side, they include the design and development of all basecalling and sequence alignment algorithms. On the hardware side, the contributions encompass the implementation of the HMM-based basecalling accelerator, the edit distance accelerator, and the customization of the systolic array-based basecalling accelerator for the second chip. Furthermore, the FPGA implementation and evaluation were solely conducted by the author. The author also contributed to specific stages of the chip fabrication process, including synthesis, pad-ring design, and FPGA testing. Other tasks associated with backend digital chip fabrication, such as floorplanning and place-and-route, were carried out by teammates.

1.4 Thesis Outline

This thesis is organized into eight chapters each focusing on different aspects of developing a novel chip design for miniature DNA sequencers. The main goal is to build specialized novel chips which pave the way for new possibilities in mobile DNA sequencing. Below is a brief overview of each chapter:

- **Chapter 1: Introduction** This chapter introduces the motivation behind the research and the significance of developing specialized chips for miniature sequencers. It discusses the potential impact on mobile DNA sequencing technology and the novelty of this work.

- **Chapter 2: Review of Basecalling Algorithms**

  This chapter reviews the evolution of basecalling algorithms, from early statistical models like HMMs to modern neural network-based approaches, including CNNs, RNNs, and transformers. It highlights their computational demands, accuracy improvements, and the challenges of real-time processing in portable sequencing devices. By examining notable basecallers and their hardware requirements, this chapter lays the groundwork for developing a new basecalling approach tailored to resource-constrained environments.

- **Chapter 3: Review of Hardware Acceleration Architectures**

  This chapter provides a comprehensive review of hardware acceleration architectures, focusing on their suitability for the computational demands of modern basecalling. It examines general-purpose GPUs, domain-specific TPUs, and RISC-V-based accelerators like Gemmini, evaluating their performance, energy efficiency, and scalability. The analysis highlights the advantages of systolic arrays as the foundation for a custom SoC, emphasizing their high par-

allelism, predictable data movement, and adaptability to resource-constrained mobile environments. These insights guide the research direction toward developing specialized accelerators tailored for portable DNA sequencing, building upon the foundational work in HMM-based SoC designs.

- **Chapter 4: First SoC: HMM Basecalling Accelerator** This chapter introduces the design, implementation, and evaluation of the first SoC tailored for HMM-based DNA basecalling. It begins with an exploration of the HMM algorithm and its adaptation for hardware acceleration. The chapter details the architectural design, featuring a RISC-V core integrated with a specialized HMM basecalling accelerator, and highlights innovations such as the inclusion of a traceback block to enhance performance.

  The chapter also covers the SoC's FPGA-based prototyping and subsequent ASIC evaluation, showcasing significant speedups and energy efficiency gains compared to CPU-based implementations and other platforms like the ARM Cortex-A53 and Intel Xeon. By examining different SoC configurations (core-only, core+AccelA, core+AccelB), the trade-offs in computational efficiency, area usage, and power consumption are evaluated. These results establish a foundation for future advancements in portable, efficient, and high-performance DNA sequencing systems.

- **Chapter 5: Second SoC: ML Basecaller Accelerator and ED Accelerator**

  This chapter focuses on the design, implementation, and evaluation of the second SoC, incorporating two accelerators: the Basecalling Accelerator and the Edit Distance Accelerator. The chapter also highlights the modular design of the basecalling software and its integration with hardware accelerators to ensure efficient execution within the SoC.

  The hardware evaluation demonstrates the performance and energy efficiency improvements achieved by the accelerators across FPGA and ASIC implementations. It also discusses the impact of architectural optimizations, such as cache configurations and memory usage, on the overall system performance. This chapter underscores the effectiveness of combining tailored software and hardware solutions to meet the computational and energy requirements of mobile DNA sequencing pipelines.

- **Chapter 6: Conclusion and Future Work** This chapter concludes the thesis by summarizing key contributions, reflecting on insights gained, and proposing future research directions. It highlights the development of the world's first specialized SoCs for nanopore sequencing, designed to address the unique demands of real-time DNA sequencing. The SoCs integrate tailored

hardware accelerators for basecalling and sequence alignment, demonstrating significant advancements in computational performance and energy efficiency within portable sequencing devices. Future work suggests adopting advanced ML models, integrating additional bioinformatics functions, employing dynamic power management, and scaling to smaller technology nodes. The chapter concludes by emphasizing the thesis's role in advancing portable, real-time bioinformatics systems for diverse applications.

2 Review of Basecalling Algorithms

To develop an efficient SoC solution for nanopore sequencing, it is crucial to first focus on the basecalling stage, as it forms the foundation for the entire sequencing workflow. This chapter begins by exploring the development and evolution of basecalling algorithms, focusing on their model architectures, computational demands, and performance trade-offs.

Basecalling is a critical step in nanopore sequencing, where raw electrical signals are converted into nucleotide sequences. This process requires advanced computational methods to ensure high accuracy and efficiency. Over time, basecalling algorithms have evolved from early statistical models to modern machine learning-based approaches, reflecting advancements in computational power and algorithmic sophistication.

2.1 Hidden Markov Model-Based Basecalling

The earliest basecalling methods relied on statistical models such as Hidden Markov Models (HMMs). HMMs represented the sequencing process as a series of states, each corresponding to a specific nucleotide or k-mer. A k-mer is a sequence of k consecutive nucleotides (e.g., "AGCT" for $k = 4$) that influences the signal as the DNA strand passes through the nanopore. HMMs modeled the probabilistic transitions between these states to decode the sequence of nucleotides from the raw signals. These models were computationally lightweight and provided a foundation for the basecalling task.

ONT's early Albacore and the open-source Nanocall [11] were both based on HMM principles, employing similar frameworks to translate raw nanopore signals into nucleotide sequences. Albacore introduced refinements to the HMM-based approach, optimizing transition and emission probabilities for improved decoding accuracy and computational efficiency. In contrast, Nanocall was designed as a simpler, accessible tool, offering a lightweight solution for researchers but falling short in precision and robustness compared to Albacore.

As illustrated in Fig. 2.1, HMMs consist of states (yellow circles) that represent k-mers, with transitions between states governed by transition probabilities (black arrows). Each state emits a signal (red wavy arrows), which is modeled by emission

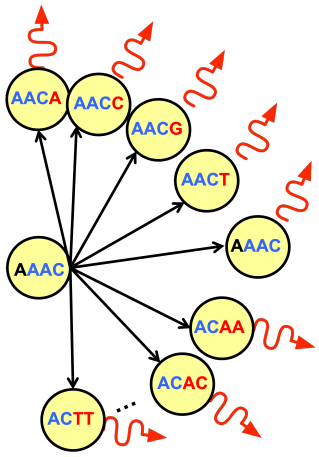

Figure 2.1: HMM for basecalling in nanopore sequencing, showing k-mer states (yellow circles), state transitions (black arrows), and emission probabilities (red wavy arrows) for signal generation.

probabilities to account for the raw signals produced by the nanopore sensor as it reads each k-mer. While these models provided a solid foundation, their inherent limitations in handling signal variability and achieving high accuracy in noisy environments motivated the development of more advanced basecalling methods.

2.2 Neural Network-Based Basecalling

The transition to neural network-based methods marked a significant turning point in basecalling, addressing many of the limitations of earlier approaches. Neural networks introduced the ability to model complex and non-linear relationships in raw nanopore signals, enabling superior accuracy, adaptability, and noise handling compared to traditional methods. Over time, a variety of architectures, including

convolutional, recurrent, and transformer-based models, have been adopted, each contributing unique strengths to the field.

While these advancements revolutionized basecalling, they also introduced substantial computational demands, particularly for real-time and portable sequencing applications. Meeting these demands has driven the development of both optimized algorithms and specialized hardware solutions. The next section will discuss examples of state-of-the-art basecallers, highlighting how architectural innovations have pushed the boundaries of sequencing technology.

2.3 Exploring Neural Network-Based Basecalling Models

In recent years, various neural network architectures have been applied to the task of basecalling, including Recurrent Neural Networks (RNNs)[34, 35], Convolutional Neural Networks (CNNs)[36, 37], and Transformer models[14, 38, 39]. Each of these architectures has demonstrated the capacity to handle the complex nature of basecalling, but their application introduces different trade-offs depending on the hardware platform and performance criteria. In the following subsections, several notable basecalling models that employ these architectures are presented, highlighting their contributions and limitations in the evolution of basecalling techniques.

2.3.1 Scrappie

The Scrappie model, as depicted in Fig. 2.2, represents one of the initial attempts to leverage deep learning for basecalling from ONT. Scrappie is implemented in C language and utilizes Intel SIMD (Single Instruction, Multiple Data) instructions to accelerate its computations. Additionally, it incorporates BLAS (Basic Linear Algebra Subprograms) libraries to further optimize matrix operations and boost the overall model performance.

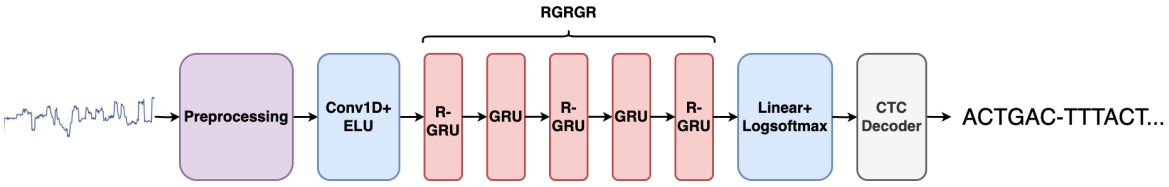

Figure 2.2: Scrappie basecaller architecture: Signals pass through preprocessing, Conv1D+ELU, alternating GRU layers (RGRGR), and a linear layer with LogSoftmax, followed by CTC decoding.

The Scrappie RNN-based basecaller, as depicted in Fig. 2.2, processes nanopore sequencing data through a structured series of neural network layers. It starts with a preprocessing block that normalizes the raw electrical signal from the nanopore device, ensuring that the model receives consistent input regardless of variability in the signal. This is followed by an initial Conv1D layer with ELU activation to extract useful features and stabilize the output.

The core of Scrappie consists of a sequence of bidirectional Gated Recurrent Units (RGRGR), which aim to capture both forward and backward temporal de-

pendencies in the input features, as shown in the diagram. By repeating R-GRU and GRU layers, the model leverages the ability of recurrent architectures to understand long-range dependencies that are crucial in basecalling. These layers form the heart of the model, capturing the rich temporal patterns present in the sequencing data.

After processing through the recurrent layers, the features are passed to a linear layer with a LogSoftmax function, projecting them into probabilities corresponding to each possible nucleotide base. The CTC (Connectionist Temporal Classification) decoder then takes these logarithmic probabilities and generates the final sequence of nucleotides, overcoming the variable length of input signals and providing a clear sequence prediction. This RNN-based approach enables Scrappie to manage the sequential nature of nanopore signals effectively, balancing complexity and accuracy to suit its intended hardware environment (i.e., typically a CPU setup).

The Scrappie model is trained using a tool called Sloika [40], which is designed to provide an effective framework for training recurrent neural network models on nanopore sequencing data. Sloika utilizes CrossEntropy as the loss function which requires alignment between raw signals and reference sequences during training. This alignment is achieved using a pretrained model that matches base sequences with corresponding raw signals. Sloika's training process leverages GPU platforms for efficient parallel computation. The training of Scrappie models presents chal-

lenges due to its reliance on the Sloika tool, which is based on the now-obsolete Theano library. Theano's discontinuation makes it difficult to maintain or update the training framework, reducing its compatibility with modern computing environments. This adds obstacles for researchers who need to train the Scrappie models from scratch or modify them for specific use cases.

2.3.2 Guppy

Guppy is a proprietary basecaller developed by Oxford Nanopore Technologies (ONT) to convert raw electrical signals from nanopore sequencers into DNA nucleotide sequences. The core of Guppy's algorithm utilizes bi-directional Recurrent Neural Networks (Bi-RNNs), specifically bi-directional Long Short-Term Memory (Bi-LSTM) layers [41]. These Bi-LSTMs help capture context from both preceding and succeeding data points, providing a more accurate representation of the signal's dependencies for basecalling.

As a production-grade solution, Guppy has been a mainstay in ONT's sequencing toolkit for several years. However, in 2023, Guppy was designated as a legacy product and is no longer actively supported, with ONT recommending users to migrate to its successor, Dorado, which offers improved capabilities and integration within ONT's sequencing software environment, MinKNOW [42].

Guppy is available in both CPU and GPU versions. The GPU version of Guppy

offers significant speed advantages over the CPU version, making it more suitable for larger datasets or high-throughput sequencing projects. The high-accuracy models of Guppy, in particular, demands considerable computational resources when run on CPUs, often necessitating the use of dedicated compute clusters or powerful workstation setups. The GPU version of Guppy, while significantly faster, has specific hardware requirements, and its installation is restricted to compatible graphics cards, preventing virtualization. This highlights the resource-intensive nature of the basecalling process, particularly when aiming for high-accuracy outputs.

Despite its capabilities, Guppy is a proprietary product, meaning that the underlying details of its implementation are not publicly accessible. This limits the users' ability to customize or extend its functionality, but ensures that it remains a reliable and optimized solution for ONT's sequencing platforms.

2.3.3 Bonito/Dorado

Building on the insights gained from models like Scrappie, the Bonito basecaller (i.e., developed by Oxford Nanopore Technologies) represents a significant advancement in neural network-based basecalling. Bonito is designed to enhance both accuracy and explore various model architectures with higher accuracy compared to earlier models, making it primarily research-oriented for better DNA sequencing solution. The model leverages the efficiency of convolutional operations, partic-

ularly depthwise and pointwise convolutions, so called "Time-Channel Separable Convolutions" (TCS-Conv), inspired by techniques from speech recognition models such as NVIDIA's QuartzNet [43]. Furthermore, Bonito has recently begun incorporating other architectures such as GRU/LSTM and Transformer-based models to further improve its performance.

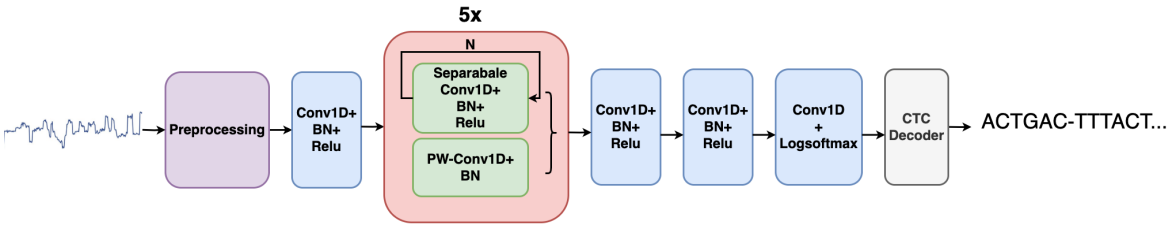

Figure 2.3: The Bonito model's architecture: composed of Temporal Convolutional Stack (TCS) blocks with residual connections, designed to efficiently process nanopore sequencing data.

In the Fig. 2.3, the model begins with the preprocessing stage. The subsequent Conv1D block, integrated with Batch Normalization (BN) and ReLU activation, is similar to that used in Scrappie model as mentioned previously.

The Bonito model then replaces RNN layers with purely conv1D layers to enhance parallel processing capabilities and reduce training times. The architecture consists of 5 large blocks (in red), each of these large blocks contains repeated separable convolutional blocks combined with batch normalization and ReLU activations. Within each separable convolution block, a depthwise convolution operation is followed by a pointwise convolution (i.e., highlighted in green). The number of

times this separable block is repeated within each larger block is represented by N, which can vary across different larger blocks. This design uses a residual connection built by pointwise convolutions followed by batch normalization to ensure stability during training. The final steps involve a CTC decoder which translates the output probabilities into a sequence of base predictions. This convolution-based architecture is well-suited for running on a specialized hardware like GPUs due to its inherent parallelism (with reduced dependence on recurrent structures) and improved efficiency in handling the high-throughput demands of real-time DNA sequencing.

Bonito models are trained using the CTC loss, which is particularly well-suited for tasks like basecalling, where the alignment between input data and target output is not explicitly given. During training, the model learns to map raw nanopore sequencing signals to nucleotide sequences without requiring pre-segmented labels. Instead, CTC loss provides an efficient way to handle variable-length outputs which enables the model to predict possible alignments of the input signal with the output sequence. Training Bonito also involves leveraging large datasets of nanopore signals and corresponding reference sequences to maximize the model's accuracy and generalization capabilities. By default, the dataset used to train Bonito contains 66,000 reads from different species, providing a diverse set of raw nanopore signals and reference nucleotide sequences. This diverse dataset helps the model generalize

across multiple organisms, enhancing its ability to perform accurate basecalling on different types of biological samples. The training process ensures the model can effectively learn from various signals, representing different biological sequences, which improves the reliability of Bonito in real sequencing scenarios for a wide range of different species.

The Bonito model's contributions are multifaceted. Its use of depthwise and pointwise convolutions, inspired by the TCS blocks, significantly reduces the number of parameters compared to standard convolutional layers, making the model lightweight while retaining its capacity for complex feature extraction. The incorporation of residual connections further enhances the model's trainability and robustness, allowing it to achieve state-of-the-art accuracy in basecalling tasks.

Dorado represents ONT's most recent iteration following Bonito with enhanced performance and adaptability. Dorado maintains the foundational architecture of Bonito which is based on CNNs for rapid and efficient processing of raw nanopore sequencing signals. However, it incorporates additional optimizations to improve both speed and accuracy, providing a more refined and high-performing solution for modern sequencing requirements.

Dorado is specifically optimized for high-performance NVIDIA GPUs, particularly the A100 and H100 models, ensuring significant improvements in the computational efficiency of basecalling when using these hardware resources. The GPU

compatibility enhances the bascalling throughput, making Dorado suitable for intensive sequencing scenarios that demand real-time or near-real-time results. Dorado is also compatible with other NVIDIA GPUs equipped with at least 8 GB of memory and the Volta architecture or newer.

These advancements reflect ONT's commitment to iterative improvements in its basecalling tools. By leveraging enhanced GPU optimizations, Dorado outperforms its predecessors, including Bonito, in terms of processing power and scalability. Dorado's targeted optimizations are designed to meet the growing demands of genomic research, providing users with faster and more accurate basecalling capabilities, especially in high-throughput environments.

2.3.4 Mauler

The Mauler model, reported in [44], utilizes an encoder-decoder architecture with global attention, as illustrated in Fig. 2.4. The model processes raw nanopore signals using bidirectional GRU layers which leverage both past and future contexts to capture the underlying structure of the data. The global attention mechanism enhances the model's ability to focus on relevant portions of the input, improving accuracy, particularly in noisy signal environments.

Mauler's key contribution lies in its use of an encoder-decoder framework with global attention which effectively manages the complex temporal dependencies

present in nanopore signals. This global attention mechanism also facilitates interpretability by identifying critical regions in the input that influence predictions. However, the attention mechanism and GRU layers make Mauler computationally intensive that limits its suitability for resource-constrained hardware.

The Mauler model utilizes a training approach similar to Scrappie, leveraging cross-entropy loss. However, instead of relying solely on raw nanopore signals, the Mauler model uses alignments that have already been generated by another basecaller, Chiron [12]. Chiron's basecalling output provides pre-aligned bases and corresponding raw signals, which serve as the ground truth labels for training Mauler. This approach helps to reduce the computational burden of alignment during training, as Mauler can directly use the already aligned sequences to focus on refining its basecalling accuracy. Additionally, the training process is demanding: training with 4000 reads takes approximately two days on an NVIDIA™ TITAN X, highlighting scalability challenges for large-scale applications.

Despite these limitations, Mauler demonstrated significant improvements in basecalling accuracy compared to earlier models. Its use of bidirectional GRU layers and global attention provided a deeper understanding of the raw signal, contributing valuable insights into handling complex DNA sequencing data.

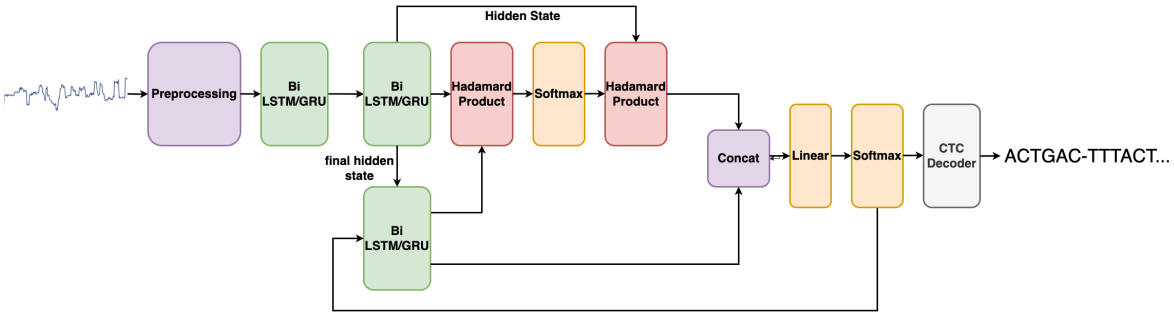

Figure 2.4: The Mauler model's encoder-decoder architecture with global attention, highlighting the bidirectional GRU layers and the attention mechanism used to process raw nanopore signals.

2.3.5 Deepnano-Blitz

DeepNano-Blitz is a basecaller developed to address the limitations of traditional basecalling software for Oxford Nanopore MinION sequencers [45]. This basecaller, originating from academic research, was designed for speed and compatibility with portable computing devices. Unlike many GPU-based basecallers that require significant computational power, DeepNano-Blitz can keep pace with the demands of a MinION run using only a quad-core laptop CPU. This makes it particularly suitable for real-time field deployment and monitoring, providing a practical solution for portable sequencing applications without the need for specialized hardware like GPUs.

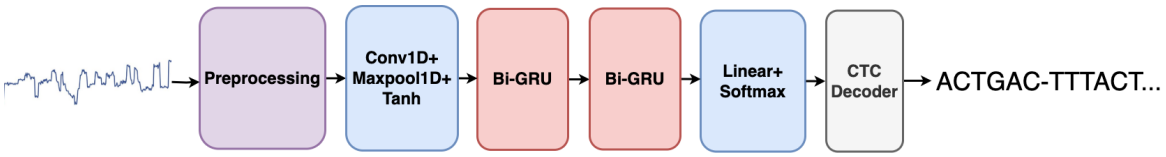

Figure 2.5: Overview of the DeepNano-Blitz architecture. The architecture consists of preprocessing, convolutional, and bidirectional LSTM/GRU blocks, followed by a linear output layer and CTC decoding.

In the depicted architecture in Fig. 2.5, DeepNano-Blitz utilizes a sequence of functional blocks, starting with a preprocessing block to normalize and prepare the raw electrical signals for further processing. This is followed by a Conv1D block, integrated with max-pooling and tanh activation, to perform feature extraction on the processed signals. The core feature extraction stage consists of bidirectional GRU layers that capture long-range dependencies in the sequence data. The output is then passed through a linear layer with a softmax function followed by a CTC decoder to generate the final nucleotide sequence.

The DeepNano-Blitz basecaller model utilizes bidirectional GRU nodes of different sizes, including configurations with 48, 56, 64, 80, 96, and 256 nodes. The accuracy obtained from these experiments ranged from 84% to 88%, depending on the network size. In terms of performance, DeepNano-Blitz can achieve speeds ranging from 1 to 4 million samples per second, demonstrating its ability to effectively handle the data throughput of MinION sequencing devices without requiring GPU acceleration.

2.3.6 DeepNano-Coral

The DeepNano-Coral basecaller [37], designed for nanopore sequencing on the Coral Edge TPU (Tensor Processing Units), leverages an architecture similar to Bonito that focuses on separable convolutions to efficiently process sequencing data while reducing computational complexity. The architecture uses depthwise and pointwise convolutions, providing efficient feature extraction with reduced parameter count and computational needs. This design is particularly well-suited to the limitations of the Coral Edge TPU [46] which has only 8 MB of memory and only supports 8-bit integer operations, necessitating careful optimization for both the model size and performance.

The Coral Edge TPU is a compact energy-efficient device primarily designed for vision tasks which presented challenges when adapting conventional neural networks for basecalling. However, DeepNano-Coral effectively repurposed the TPU for nanopore sequencing by using separable convolutions which are more hardware-friendly for a device constrained by memory and data precision. This architecture not only supports real-time processing but also operates at a speed of 1.54 million signal samples per second. This makes it competitive with other real-time models like Guppy-fast and DeepNano-blitz while consuming significantly less energy.

DeepNano-Coral's accuracy is sufficient for applications like monitoring species

composition in environmental or clinical samples. With its energy usage ranging between 0.6–0.7 Wh per 40.8 Mbp of data, it significantly outperforms Bonito-fast in energy efficiency while offering comparable speed. This makes DeepNano-Coral a practical and cost-effective solution for field deployments where power availability and portability are critical considerations. By optimizing the architecture for the Coral Edge TPU, DeepNano-Coral bridges the gap between efficient edge computing and the demands of nanopore sequencing in various practical applications.

2.3.7 SACall

Transformer models have recently gained popularity in the field of basecalling due to their ability to effectively capture long-range dependencies in sequences and their inherent parallelism capabilities. SACall [14] is a Transformer-based basecaller that builds upon these advantages, aiming to provide higher accuracy than other basecalling models. The architecture of SACall, as illustrated in Fig. 2.6, leverages the strengths of the transformer model to process the raw electrical signals from nanopore sequencing and output nucleotide sequences.

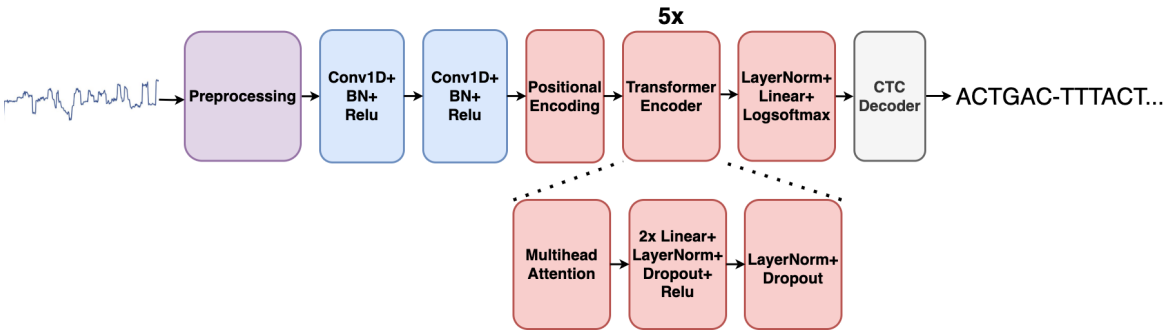

Figure 2.6: The SAcall basecaller includes preprocessing, convolutional layers, Transformer encoders, and a CTC decoder to generate nucleotide sequences.

The SACall model begins with the preprocessing stage where the raw signals generated by the nanopore sequencer are segmented into manageable pieces and normalized to mitigate inconsistencies in sequencing conditions. These preprocessed signals are then passed through a series of convolutional layers: the first two Conv1D layers, each combined with Batch Normalization (BN) and ReLU activation, extract relevant features from the input signal while preserving computational efficiency.

Following the feature extraction, positional encoding is applied to provide the sequential information required by the transformer model. The encoded sequence then passes through the core of the SACall architecture, comprising 5 transformer encoder blocks. Each transformer encoder block includes a multihead attention mechanism that captures interdependencies between positions in the sequence followed by two linear layers with layer normalization, ReLU activation, and dropout for enhanced regularization. The residual connections in the transformer encoder

blocks help facilitate gradient flow during training.

The final part of the SACall architecture includes a linear projection layer combined with LogSoftmax activation to transform the output from the transformer encoders into base probabilities. The CTC decoder takes these probabilities and translates them into nucleotide sequences (A, C, G, T), providing the final output. The use of CTC loss enables SACall to handle variable-length outputs, making it well-suited for processing sequences of differing lengths.

SACall's architecture enables the model to capture long-range dependencies efficiently while the use of convolutional feature extraction and positional encoding ensures that the model has a good representation of the sequential information before applying the transformer blocks. This combination makes SACall an effective choice for real-time basecalling tasks on platforms with adequate computational resources. Compared to recurrent models, SACall's use of parallelizable attention mechanisms significantly improves the inference speed making it a potential choice for large-scale sequencing experiments.

2.3.8 NN-Based Basecalling Models Summary

The presented basecallers' architectures exhibit a range of advantages and limitations, particularly in terms of computational efficiency, accuracy, and hardware requirements. Tab 2.1 offers a comprehensive comparison of these basecallers, sum-

marizing the key features that define their respective performance capabilities.

Table 2.1: Summary of basecalling models discussed in this chapter, highlighting platform usage, model architecture, size, parameter count, accuracy, and training methodologies.

Model Name	**Scrappie**	**Bonito**	**Mauler**	**DeepNano-Blitz(80)**	**DeepNano-Coral**	**SACall**
Model Arch	GRU	TCS-CNN	GRU	GRU	CNN	Transformer
Platform	CPU	GPU	GPU	CPU	TPU	GPU
Library	BLAST	Pytorch	Keras	Rust&MKL	TF-Lite	Pytorch
Model Size(MB)	5.9(T)	2.9(B)	3(B)	1.5(B)	4.6(B)	24.5(B)
Parameters (K)	387	733	365	161	2386	4834
Accuracy(%)	80.2	92.7	86.4	90.2 [45]	88.2 [37]	91.1
Training Method	CELoss	CTCLoss	CELoss	CTCLoss	CTCLoss	CTCLoss

The comparison reveals the diversity of approaches in terms of model architectures and hardware platforms, from the lightweight CPU-based Mauler and DeepNano-Blitz models to the GPU-optimized SACall and Bonito basecallers. Each model represents a different balance between computational requirements and achievable accuracy, reflecting the constraints and design choices made to optimize for specific platforms or use cases. Notably, the SACall and Bonito models, leveraging CNN and transformer architectures, aim to maximize accuracy and speed through the use of sophisticated hardware accelerations such as GPUs.

On the other hand, models like DeepNano-Coral have been designed with ef-

ficiency in mind utilizing edge hardware such as the TPU to perform real-time basecalling with minimized power consumption. This distinction highlights the versatility required to cater for different deployment scenarios, ranging from field applications, which prioritize energy efficiency, to more computationally demanding laboratory environments that focus on maximizing throughput and accuracy.

2.4 Challenges and Limitations of Current Basecallers

Despite significant advancements in basecalling algorithms, several challenges and limitations remain, particularly when addressing the need for real-time processing in portable sequencing devices. These obstacles must be addressed to fully realize the potential of nanopore sequencing technology in diverse environments.

2.4.1 High Computational Demands

Current basecalling models, despite their advancements, face significant challenges that hinder their deployment in real-time and portable sequencing systems. One of the most pressing issues is the high computational demand associated with modern basecallers. Models like SACall and Bonito, which leverage advanced neural network architectures such as transformers and CNNs, require substantial processing power and memory to achieve their high accuracy of 91.1% and 92.7% respectively. These computational requirements make it difficult to implement such models on

resource-constrained devices while maintaining real-time performance.

2.4.2 Power and Resource Inefficiency

Another critical limitation lies in power and resource efficiency, particularly for portable sequencing applications. While models like DeepNano-Coral have been designed for edge hardware such as TPUs to improve efficiency, they still rely heavily on a host PC for data preprocessing and communication with the TPU. This reliance reduces the overall portability and autonomy of the system, as the host PC adds additional power and space requirements. Although DeepNano-Coral demonstrates a balance between computational efficiency and accuracy (88.2%), its deployment scenario is not fully optimized for standalone operation in resource-limited environments.

2.4.3 Dependence on External Powerful Computing Environments

Many modern basecallers rely heavily on external GPUs or powerful computing environments to operate efficiently. While these platforms enable fast processing and high throughput, they introduce significant challenges in terms of portability and cost. For instance, Bonito's relatively compact TCS-CNN model architecture (2.9 MB) still requires GPU support to process large datasets effectively. This dependence limits its applicability in scenarios where access to high-performance

hardware is restricted or impractical, such as remote fieldwork.

2.5 Chapter Summary

The chapter provides a comprehensive review of the evolution and state-of-the-art advancements in basecalling algorithms, highlighting the transition from early statistical models like HMMs to modern machine learning-based approaches, including convolutional, recurrent, and transformer architectures. Each method is analyzed for its computational requirements, accuracy, and adaptability to real-time processing challenges. The chapter also discusses notable basecallers such as Scrappie, Bonito, and SACall, alongside efficiency-focused solutions like DeepNano-Coral.

The review also examines the role of hardware acceleration in addressing the computational demands of modern basecallers, focusing on GPUs, TPUs, and other specialized platforms. This understanding lays the groundwork for evaluating the performance of our own basecaller and its integration with hardware accelerators. By exploring these trade-offs, this chapter sets the stage for designing and optimizing our SoC solution to balance power efficiency, real-time processing, and accuracy, as detailed in the following chapters.

This chapter reviews the development and progression of basecalling algorithms, emphasizing their role in nanopore sequencing. It examines the evolution from early statistical approaches, such as HMMs, to advanced neural network-based meth-

ods, including CNN, RNN, and transformer architectures. By analyzing notable basecallers like Scrappie, Bonito, and SACall, the chapter identifies trends in computational demands, accuracy requirements, and efficiency challenges. The focus remains on the computational characteristics and trade-offs of these algorithms, setting the foundation for exploring how modern basecalling techniques influence system design and optimization, as covered in subsequent chapters.

3 Review of Hardware Acceleration Architectures

While the previous chapter explored algorithmic aspects (including both HMM and Neural Network approaches), this chapter delves into the hardware side. Specifically, it examines hardware acceleration architectures that can help offload and optimize the intensive computational demands inherent in modern basecalling pipelines.

Many mainstream solutions for accelerating neural networks or HMM-like computations revolve around high-throughput parallel architectures (GPUs) [47, 48], specialized ASIC designs (e.g., TPU [49]), or domain-specific accelerators (such as NVDLA [50], and Gemmini [51]). Each of these architectures reflects a set of trade-offs in performance, power consumption, scalability, programmability, and ease of integration. By reviewing these designs in this chapter, we can identify which architectural principles best address the needs of portable sequencing devices, including real-time throughput, low power operation, and compact form factors.

3.1 GPU-Based Acceleration

General-purpose Acceleration like Graphics Processing Units (GPUs) are widely used to accelerate computationally intensive tasks due to their highly parallel architecture [52]. Their ability to handle matrix-heavy operations makes them a viable option for neural network-based basecalling [53]. The following section provides an overview of their architectural features and limitations in the mobile environment.

3.1.1 GPU Architectural Overview

A GPU is built around the principle of massive parallelism and is originally designed for graphics rendering, which naturally involves substantial amounts of data-level parallelism. In modern GPUs used for deep learning inference, the architecture is composed of multiple core clusters, called Streaming Multiprocessors (SMs) in NVIDIA designs [54]. Each multiprocessor can execute thousands of lightweight threads in parallel, commonly following a Single Instruction, Multiple Threads (SIMT) model [55]. By scheduling groups of threads (warps or wavefronts) to run the same instruction, the hardware can amortize control overhead and achieve high throughput.

The GPU memory hierarchy typically includes global memory (off-chip DRAM, such as High Bandwidth Memory (HBM) or Graphics Double Data Rate (GDDR)),

faster on-chip caches or shared memory, and a large register file for thread-specific data [56]. Performance in GPU-based inference is heavily dependent on how effectively the workload uses shared memory and on how well memory accesses can be coalesced into large contiguous blocks. Current GPU generations also incorporate specialized units—NVIDIA calls them Tensor Cores—that handle mixed-precision matrix-multiplication [57]. These units dramatically accelerate neural network computations by performing multiple fused multiply-add operations in parallel, often at reduced precision (e.g., FP16 or INT8), helping to balance accuracy with performance [58].

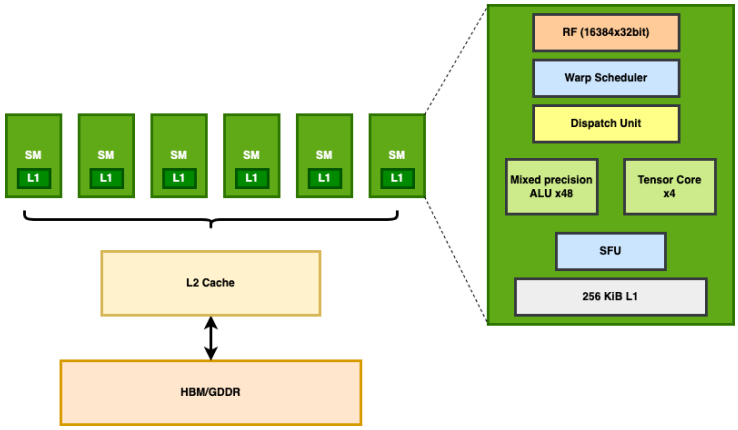

Figure 3.1: Simplified GPU architecture featuring multiple Streaming Multiprocessors (SMs) that share an L2 cache and connect to off-chip GDDR memory.

Fig. 3.1 depicts a typical GPU architecture, showing the SMs and their key elements. Each SM integrates local caches (L1), mixed-precision ALUs, tensor cores, and warp schedulers to handle computationally intensive tasks. The L2 cache

bridges the SMs to global memory implemented with HBM, optimizing data flow and minimizing latency for large genomic datasets. The integration of caches at multiple levels reduces latency and enhances data accessibility for compute units. HBM or GDDR SDRAM provides the necessary bandwidth for large genomic datasets, ensuring that input signals can be processed with minimal delay [59].

The use of GPUs in basecalling workflows significantly accelerates the process, reducing the time required for real-time sequencing. This acceleration facilitates rapid analysis and is particularly beneficial in scenarios where timely results are critical. However, challenges such as power consumption and optimization complexity must be considered when integrating GPUs into portable sequencing systems. Despite these challenges, GPUs continue to play a pivotal role in advancing the capabilities of bioinformatics tools, contributing to the ongoing improvements in DNA sequencing technology.

3.1.2 Architectural Limitations

Although GPUs can deliver exceptional performance on matrix-heavy tasks such as convolutions and fully connected layers, they often demand significant power [60]. Desktop- or data center-grade GPUs typically carry TDP (Thermal Design Power) values ranging from 150W to over 300W [61], reflecting their high computational density and memory bandwidth requirements. While embedded or mobile GPUs

exist with much lower power envelopes (e.g., around 10–20W [62]), they also feature fewer cores and reduced performance.

For basecalling, a GPU may be effective if the workload is amenable to batch processing or if it can be processed in parallel streams. However, the overhead of a general-purpose software stack and the GPU's comparatively higher power consumption pose challenges for battery-powered and thermally constrained systems. Moreover, embedded GPUs still demand careful thermal management, which can complicate the design of portable devices [63]. As a result, while GPUs remain a viable and flexible solution, they can be overshadowed by dedicated accelerators designed for narrower—but more power-efficient—goals in highly constrained environments like mobile DNA sequencing.

3.2 TPU-Based Acceleration

The Tensor Processing Unit (TPU) represents a significant milestone in the evolution of domain-specific accelerators, purpose-built to optimize machine learning workloads. With a design tailored for neural networks, TPUs excel at executing matrix-heavy operations such as those found in convolutional and dense layers. Their architecture emphasizes energy efficiency, high throughput, and reduced latency, making them well-suited for computationally intensive applications. By leveraging systolic arrays for efficient matrix multiplication and employing on-chip

memory for reduced data movement, TPUs deliver a balance of performance and power efficiency that has set a benchmark for hardware acceleration in machine learning. This section explores the TPU's architecture and the principles that contribute to its high performance.

3.2.0.1 TPU Architecture Overview

The Google™ TPU embodies a dataflow-centric approach that diverges from the parallel thread-based execution seen in GPUs [64]. As illustrated in Fig. 3.2, the TPU design incorporates several distinct components arranged to maximize throughput for matrix-intensive operations while limiting off-chip data transfers. At a high level, a PCI-e interface conveys data between the TPU and a host processor at a rate often measured in tens of gigabytes per second. On the external memory side, multiple DDR3 channels collectively furnish bandwidth on the order of 30 GiB/s, supplying the weights and activations needed for deep neural network inference. Inside the chip, an on-chip buffer serves as unified local storage, allowing frequent reuse of activations and partial sums without recurring round-trips to external DRAM. Beyond the memory path, the TPU architecture features a dedicated Matrix Multiply Unit (MMU) based on systolic array that can sustain tens of thousands of multiply-accumulate operations each cycle. This core is responsible for repeatedly processing small chunks of input data, streamed in from the memory

subsystem, and partial sums that accumulate into final layer outputs. Once the multiplications are complete, subsequent hardware blocks manage activations and pooling, ensuring that common neural network functions can be performed without returning to off-chip memory [65].

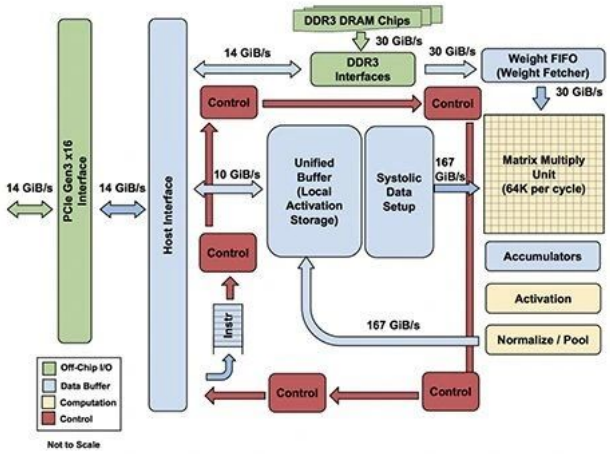

Figure 3.2: Overview of Google™ TPU architecture, highlighting the key components such as the MMU, Unified Buffer, Weight FIFO, and DDR3 interfaces [66].

A defining characteristic of the TPU's core is its reliance on a systolic array to implement the MMU. Fig. 3.3 depicts the essence of a systolic dataflow, in which processing elements (PEs) are arranged in a two-dimensional grid, with each PE receiving new data at a regular cadence. Matrix elements from two input matrices, often referred to as activations and weights, enter from different edges of the array. Each PE multiplies these elements, accumulates partial sums, and passes results along to its neighbor. This wave-like movement of data greatly diminishes the

need to read and write external memory at each step because once data arrives in the array, it remains in flight until processing is complete. As a result, memory bandwidth is used more efficiently, yielding higher performance per watt. In a TPU-like architecture, weight data can stay loaded in the array for reuse across multiple sets of activations, and partial sums exit only after traveling through the entire pipeline of PEs. This arrangement suits the repeated multiply-accumulate patterns found in many convolutional or fully connected layers, as each filter or kernel can persist inside the array while different slices of input data stream through.

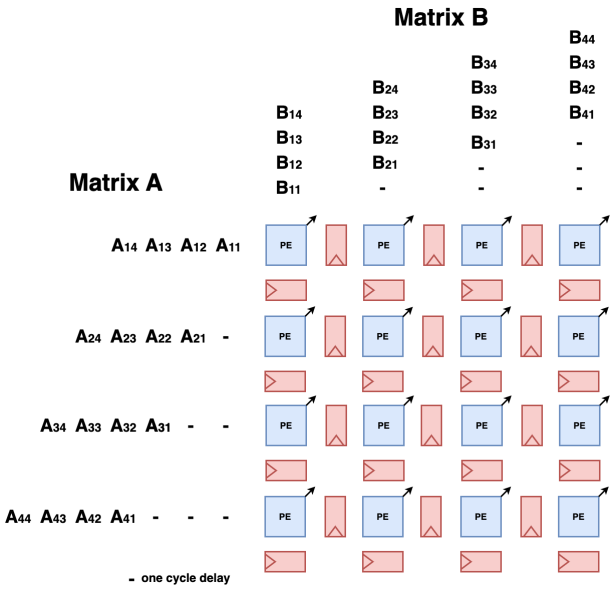

Figure 3.3: Illustration of a 4×4 systolic array for matrix multiplication.

In practical terms, the TPU's balance of local buffering, structured data movement, and specialized matrix hardware drives down the overhead commonly associated with high-volume neural network computations. The on-chip controllers,

which coordinate data transfers between the PCIe interface, DRAM, and the unified buffer, keep the systolic array fed with new data at a rate that matches its throughput capability. Substantial on-chip bandwidth, cited at around 167 GiB/s in TPU designs, ensures that data does not stall while traversing from the buffer to the multiply array.

For workloads like basecalling that often hinge on dense matrix operations, the TPU's architecture can yield substantial gains if layer dimensions and memory layouts map well to the array's shape and dataflow scheme. However, some overhead may arise if the neural network model deviates from standard matrix multiplication patterns, or if the array's capacity exceeds what the application can effectively utilize. Despite these nuances, the TPU's systolic array stands as a prominent reference model, demonstrating how extensive parallelism and minimized off-chip traffic can coalesce into an efficient platform for real-time inference.

3.2.0.2 Architectural Limitations

Although the TPU architecture demonstrates impressive features such as high computational throughput and energy efficiency, it still faces notable limitations when applied in mobile environments. In data center deployments, a single TPU can operate with a TDP ranging roughly from 75W to over 200W (depending on the generation) [49], making it less feasible for battery-operated devices without spe-

cialized cooling solutions.

Even the Edge TPU, commonly known as the Coral TPU, as shown in Fig. 3.4, operates at a significantly lower TDP of around 5–10W [67], making it a highly energy-efficient option for edge AI tasks. Featuring a 64×64 systolic array and 8 MB of on-chip memory, the Coral TPU is well-suited for machine learning inference within its constrained power budget. However, despite its compact and power-efficient design, the Coral TPU relies heavily on a host CPU for critical functions such as control, data preprocessing, and task management. This reliance necessitates frequent communication between the host CPU and the TPU over USB, introducing additional latency and increasing power consumption. These architectural inefficiencies are particularly challenging in mobile environments, where minimizing data transfer and achieving a higher degree of integration are critical. As a peripheral device, the Coral TPU is limited in its ability to meet the requirements of seamless, end-to-end processing, making it less suited for applications demanding fully autonomous operation.

Separately, another drawback of the Coral TPU is the high energy and latency costs associated with off-chip memory access during operation. While its on-device memory can cache smaller models, larger or streaming workloads require frequent external memory access, which incurs significant energy overhead and slows processing. This is a critical limitation in mobile environments, where efficiency and

responsiveness are paramount. By contrast, a tightly integrated SoC can be designed to optimize memory hierarchies, ensuring data is processed efficiently without excessive off-chip communication, thus improving both energy efficiency and throughput.

Finally, the fixed configuration of the Coral TPU restricts its adaptability to domain-specific tasks. While it excels at general-purpose neural network inference, mobile sequencing involves unique computational patterns, such as alignment and error correction, that require greater flexibility. A custom SoC equipped with a configurable systolic array can address these diverse demands while maintaining scalability, efficiency, and performance.

In summary, while the Coral TPU showcases the potential of TPU architecture, it falls short in meeting the autonomy, scalability, and domain-specific requirements of mobile sequencing. These limitations highlight the importance of designing a dedicated SoC tailored to bioanalysis workflows, enabling efficient, fully integrated, and high-performance processing in mobile environments.

Figure 3.4: The Coral TPU, featuring a 64×64 systolic array and 8 MB of on-chip memory for efficient machine learning inference [68].

3.3 Requirements and Constraints for the Proposed SoC

The design of a SoC for basecalling must address a unique set of requirements and constraints that stem from the computational demands of modern neural network models and the operational limitations of portable DNA sequencing devices. Basecalling algorithms, whether convolutional, recurrent, or transformer-based, demand high computational throughput to process real-time nanopore data streams. At the same time, the SoC must adhere to strict constraints on power consumption, thermal performance, and form factor to enable deployment in compact, battery-operated devices. Achieving this balance requires specialized hardware architectures optimized for dense linear algebra operations, efficient data movement, and reduced precision arithmetic, while ensuring that basecalling accuracy and reliability remain uncompromised. The following subsections explore these requirements

in detail.

3.3.1 Performance Demands of Basecalling

Neural network models used in basecalling can vary significantly, from convolution-based methods to those involving recurrent or transformer layers. Despite their differences, most models rely heavily on dense linear algebra operations and typically need to process streaming data in real time. The combination of high-throughput requirements and matrix-heavy operations underscores the importance of specialized hardware that can handle large amounts of multiply-accumulate computations efficiently.

The performance demands of basecalling stem from both throughput and latency considerations. Throughput requirements grow as more sequencing channels generate raw signals simultaneously, creating a stream of data that must be converted into nucleotide sequences without delaying the overall sequencing process. Latency becomes critical in certain real-time applications, such as selective sequencing, where immediate decisions about read processing can significantly impact experimental outcomes. Another key challenge is maintaining basecalling accuracy. Although reduced-precision arithmetic, such as 8-bit integer operations, is commonly employed in hardware accelerators to improve efficiency, it must be carefully implemented to minimize any impact on the final accuracy of basecalling results.

3.3.2 Mobile DNA Sequencing Constraints

In a mobile DNA sequencing context, power and form-factor constraints dominate the design space. A portable sequencer usually operates from a battery or a limited power source and cannot dissipate excessive heat. These limits reduce the viability of hardware architectures that draw tens or even hundreds of watts. The device form factor also restricts the size of the SoC, making large and power-hungry accelerators less feasible. In addition, shared memory bandwidth among CPU cores, the accelerator, and other subsystems can introduce bottlenecks if not managed carefully.

Thermal considerations are particularly important. Portable sequencing devices are not equipped with the same cooling solutions found in servers or desktops. Consequently, hardware running at high throughput must also be capable of operating within tight thermal envelopes. Another constraint is the need for a robust software ecosystem that supports on-device neural inference. The specialized capabilities of the accelerator must be exposed via toolchains, libraries, or frameworks that allow basecalling software to leverage the hardware's strengths without requiring excessive manual optimization.

3.4 Architecture Choice for the SoC Design Research

Selecting the right hardware architecture for the SoC is critical for achieving the performance, power efficiency, and flexibility required for DNA basecalling and broader bioinformatics applications. The computational demands of modern neural network models, coupled with the constraints of mobile DNA sequencing, necessitate a design optimized for matrix-heavy operations and real-time performance. Among the various options, systolic arrays stand out as a compelling choice for this research.

3.4.1 Advantages of Systolic Arrays for SoC Design

The decision to adopt a systolic array architecture stems from its ability to efficiently handle dense linear algebra operations, particularly matrix multiplications, which are central to most neural network models. Systolic arrays structure minimizes data movement, maximizing reuse of intermediate results and significantly reducing energy consumption—a crucial factor for portable devices.

Moreover, systolic arrays offer high scalability and customizability. By varying the size of the array (total number of PEs) or the precision of computations (e.g., INT8), the architecture can be tailored to meet different performance and power requirements. This flexibility is essential for supporting diverse workloads, from lightweight real-time models to more complex, high-accuracy models. The

predictable data movement and simplicity of control within systolic arrays further enhance their suitability for integration into real-time SoC environments.

3.4.2 Gemmini Overview

While the number of accelerators available in the RISC-V ecosystem remains relatively small compared to other architectures, Gemmini stands out as a well-known and impactful project developed by the University of California, Berkeley (UCB). Its open-source nature, coupled with a modular and customizable design, has made it a popular choice for researchers and developers exploring high-performance hardware within the RISC-V ecosystem [51, 69, 70]. By adopting Gemmini as the core accelerator for this SoC design, the research not only benefits from an efficient, well-documented hardware platform but also aligns with the broader goals of leveraging open, extensible, and energy-efficient architectures in the RISC-V community.

Gemmini is designed to seamlessly integrate into RISC-V systems, providing an efficient and flexible solution for matrix-heavy computations. Built around a systolic array, Gemmini excels in accelerating operations central to modern machine learning workloads, such as matrix multiplications and convolutions. Fig. 3.5 illustrates the high-level architecture of Gemmini and its integration with a RISC-V Rocket core.

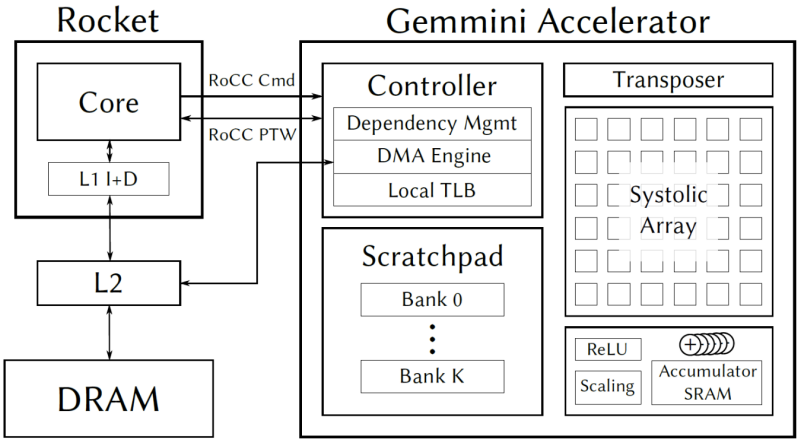

Figure 3.5: High-level architecture of the Gemmini accelerator, showcasing its integration with the RISC-V Rocket core [71].

At the heart of the Gemmini architecture is the systolic array, which performs dense matrix multiplications with high computational efficiency. By structuring computations across an array of interconnected PEs, the systolic array minimizes data movement and maximizes data reuse. To manage dataflow and ensure efficient computation, Gemmini incorporates a scratchpad memory that provides temporary storage for input and output matrices. The scratchpad is organized into multiple banks, allowing for parallel access to data, which reduces memory contention and increases throughput. Gemmini also features a controller responsible for orchestrating operations between the systolic array, the scratchpad memory, and the Rocket core (an open source RISC-V). The controller handles tasks such as dependency management, data transfers via the DMA engine, and address translation through the local Translation Lookaside Buffer (TLB).

In practice, this architecture allows Gemmini to operate as a specialized engine for deep learning workloads, particularly matrix-heavy tasks found in convolutional, recurrent, or transformer-based networks. Its configurable parameters, such as the size of the systolic array and scratchpad, give designers the flexibility to trade off die area, power, and performance. By pairing the Rocket core's programmability with Gemmini's specialized dataflow, the system can rapidly move between general-purpose tasks and computationally intensive basecalling kernels, achieving an effective balance for on-device, real-time DNA analysis.

Further details on Gemmini's architecture and its integration for machine-learning-based basecalling accelerators are discussed in § 5.3.

3.4.3 Relevance to Mobile DNA Sequencing

A Gemmini-based systolic array offloads the matrix-heavy operations of deep learning models from the main CPU, reducing both power draw and heat buildup. Its configurable parameters—such as the array size or the bit-precision of computations—allow tailoring the accelerator's performance to match specific power and latency requirements. Some basecalling models can leverage lower-precision arithmetic, enabling further energy savings without significantly compromising accuracy.

Because Gemmini relies on a localized scratchpad and predictable dataflows, it minimizes off-chip memory accesses. This design choice is essential for extending

energy efficiency while maintaining sufficient throughput for continuous nanopore data streams. Moreover, integrating Gemmini into a RISC-V SoC supports dynamic workload management and adaptive power scaling, both of which are key to running real-time inference under strict thermal limits. The open-source nature of Gemmini also facilitates customization of the hardware-software stack, ensuring that the accelerator can evolve alongside future basecalling techniques and sequencing protocols.

3.5 Chapter Summary

This chapter reviewed prominent hardware acceleration architectures, including GPUs, TPUs, and accelerators in RISC-V ecosystems like Gemmini. GPUs demonstrate versatility and high throughput but face power and thermal constraints in mobile environments. TPUs, while energy-efficient for matrix-heavy workloads, are hindered by reliance on host CPUs and off-chip memory. Systolic arrays emerged as the particularly well-suited choice for DNA basecalling due to their scalability, high parallelism, and energy efficiency.

Gemmini, a RISC-V-based systolic array accelerator, was highlighted as a standout solution for the proposed SoC design. Its open-source, configurable nature enables seamless integration and efficient execution of matrix-dominant workloads, making it a natural fit for real-time mobile DNA sequencing devices. By leveraging

the architectural advantages of systolic arrays and Gemmini, the research positions itself to address the unique computational and operational challenges of portable bioinformatics.

4 First SoC: HMM Basecalling Accelerator

This chapter introduces the design, hardware implementation, and performance evaluation of the first specialized SoC for DNA basecalling. Built around the RISC-V processor architecture and equipped with a dedicated HMM accelerator, this SoC marks an initial step toward creating optimized hardware solutions for HMM-based DNA sequencing. The chapter covers the underlying algorithm, the detailed hardware design process, and performance measurements, demonstrating the feasibility and effectiveness of this architecture while setting the stage for subsequent advancements.

4.1 Algorithm Development

The success of an embedded System-on-Chip (SoC) for mobile DNA sequencing is largely determined by the algorithms chosen to support its core bioinformatics tasks. Each algorithm's computational requirements, accuracy, scalability, and

The section 4.1, 4.3 and 4.6 has published in [72].

compatibility with specialized hardware architectures must be carefully evaluated to achieve the desired balance of performance, energy efficiency, accuracy and real-time sequencing capabilities.

To begin our exploration of basecalling methodologies, we start with the HMM approach [73]. Historically, HMM were among the first algorithms used for basecalling due to their effectiveness in modeling the sequential nature of the data and their capability to handle noise and variability in signal levels. HMM are well-suited for this task as they provide a probabilistic framework for modeling the temporal structure of signals, making them a natural choice for interpreting the noisy data generated by nanopore sequencing.

In HMM basecalling, the primary objective is to articulate the DNA bases encoded by the N-event time-series $\boldsymbol{x} = (x_i)_{i=0}^{N-1}$, which represents the nanopore's output signal. The challenge lies in the noisy characteristics of the output signal, both in amplitude and time [4]. To address this, the basecaller employs a form of sequence detection [74], utilizing the HMM to model the sensor and extract meaningful sequences from the noisy data.

Based on the sensor HMM and on $\boldsymbol{x}$, the basecaller computes $4^{k-mer} \times N$ 'emission' probabilities, $\epsilon_j(x_i)\ \forall\ (j,i)$, associated with the states that the nanopore can possibly exhibit over N event observations. These calculations are represented by the trellis diagram in Fig. 4.1 where the circles denote (in part) the aforementioned

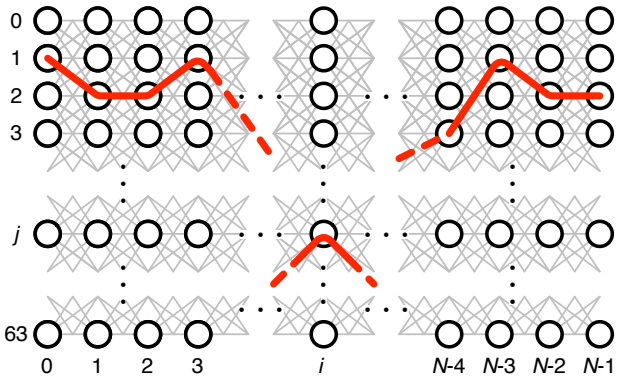

Figure 4.1: Trellis diagram of the basecaller's computational process, with the x-axis representing time frames and the y-axis showing all 64 possible computational states.

state emission probability calculations and the number of states is assumed to be 64.

The HMM also specifies the 'transition' probabilities, $\tau(j_{i-1}, j_i)$, (grey lines in Fig. 4.1) that quantify the probability of nanopore states at event $i-1$ evolving to states at event i. Generally, any state at event i may have 4^{k-mer} such connections within the trellis. However, as we note below, in the HMM, only a unique set of 21 transitions to each state j, $\omega(j)$, is accounted for.

Using an iterative dynamic programming technique (the Viterbi algorithm), the basecaller utilizes its emission and transition values to compute the most likely sequence of states through the trellis, the path $\boldsymbol{\pi}^* = (\pi_i^*)_{i=0}^{N-1}$ (represented by the red line in Fig. 4.1), and thus, the most likely sequence, $\hat{\boldsymbol{a}} = (\hat{a}_i)_{i=0}^{N-1}$, of bases associated with the measured time-series.

Algorithm 1 64-State HMM Basecalling Algorithm

1: *trellis construction:*
2: $\alpha_0(j) \leftarrow \epsilon_j(x_0)\ \forall\ j \in \{0, \ldots, 63\}$
3: *Iterate:*
4: **for** $i \leftarrow 1, N-1\ \ \&\ \ \forall\ j$ **do**
5: $\quad \alpha_i(j) \leftarrow \epsilon_j(x_i) \max_{\nu \in \omega(j)}[\alpha_{i-1}(\nu)\tau(\nu, j)]$
6: $\quad \beta_i(j) \leftarrow \arg\max_{\nu \in \omega(j)}[\alpha_{i-1}(\nu)\tau(\nu, j)]$
7: **end for**
8: *EndState:*
9: $\pi^*_{N-1} \leftarrow \arg\max_j[\alpha_{N-1}(j)]$
10: $\hat{a}_{N-1} \leftarrow \pi^*_{N-1}\ \&\ 3$
11: *Traceback:*
12: **for** $i \leftarrow N-1$ **to** 1 **do**
13: $\quad \pi^*_{i-1} \leftarrow \beta_i(\pi^*_i)$
14: $\quad \hat{a}_{i-1} \leftarrow \pi^*_{i-1}\ \&\ 3$
15: **end for**

The pseudocode listed in Algorithm 1 provides a succinct description of the basecaller's computational process, we highlight its pertinent components now.

The *Iterate* block comprises a significant part of the basecaller and is the focus of the accelerator hardware. This block is as nested for-loop sweeping across all 64 possible state calculations for all N events being measured. Therein, the computation in line 5 combines the emission, $\epsilon_j(x_i)$, and transition, $\tau(\nu, j)$, probabilities (we elaborate on these terms shortly) to update a running measure of each state's *posterior* $\alpha_i(j)$; formally the basecaller's main computation results in estimates of the maximum likelihood for each state after N input observations:

$$\alpha_N(j) = \alpha_0(j) \prod_{i=1}^{N-1} \epsilon_j(x_i)\tau(\nu, j)|_{\nu \in \omega(j)}. \tag{4.1}$$

In concert (line 6), a *pointer*, $\beta_i(j) \in \{0, \ldots, 63\}$, is computed; $\beta_i(j)$ denotes the state at event $i-1$ most likely to have transitioned into state j at event i. $\beta_i(j)$ is drawn, as noted above, from a set $\omega(j)$ of 21 possible predecessors to j (elaboration below).

Keeping track of the most likely preceding steps allows the *Traceback* block to sequentially consult the accumulated pointers and to compute the most likely sequence of states, $\boldsymbol{\pi}^*$, and bases $\hat{\boldsymbol{a}}$ thus completing the basecalling effort.

To complete our sketch, we highlight some details related to the emission and transition terms. The emission characteristics relate the probability that an event x_i may be associated with any of the expected outputs μ_j. This is usually depicted according to the Gaussian distribution as

$$\epsilon_j(x_i) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp[(x_i - \mu_j)^2/2\sigma^2], \tag{4.2}$$

where σ denotes the standard deviation of the observations around the anticipated levels, a measure of the noise in the system.

The term $\tau(\nu, j)$ denotes the probability that a state, $\nu \in \{0, 1, \ldots, 63\}$, transitions to a state j. As noted above, for each j, we account for 21 such transitions; one for a transition via a stay mechanism where $\nu = j$; another four that account for four possible step transitions,

$$\nu = 16{\cdot}l + \lfloor j/4 \rfloor$$

with $l \in \{0, 1, 2, 3\}$; and another 16 that account for 16 possible skip transitions over one base,

$$\nu = 4 \cdot L + \lfloor j/16 \rfloor$$

where $L \in \{0, \ldots, 15\}$. Thus the set of states preceding j, $\nu \in \omega(j)$, consists of $1 + 4 + 16 = 21 = |\omega(j)| \ \forall \ j$.

Although the HMM basecaller exhibited computational efficiency, rendering it a viable candidate for initial hardware implementation, its basecalling accuracy was suboptimal compared to more sophisticated models. As nanopore technology continued to evolve, the complexity and variability of the raw signals increased, rendering the simplistic assumptions of the HMM insufficient. The HMM's reliance on a fixed probabilistic model struggled to adapt to the intricacies of modern nanopore sequencing, which involves more nuanced signal behaviors and noise patterns. As a result, while HMM remains a foundational approach and was suitable for our initial basecalling chip, it becomes clear that more advanced methods are required to achieve the accuracy necessary for practical DNA sequencing applications, particularly as the technology evolved.

4.2 SoC Architecture Overview

The first SoC design features a 64-bit RISC-V core with a 5-stage in-order pipeline, sourced from an open-source project [75, 76]. The SoC integrates components to

effectively handle DNA basecalling demands, including a 16KB instruction cache (ICache) and a 16KB data cache (DCache), each configured as 4-way associative [77]. These caches serve both the RISC-V core and the basecalling (BC) accelerator, optimizing data access and reducing latency. Additionally, an L2 cache serves the entire SoC, enhancing data access speeds and supporting system performance Fig 4.2 illustrates this overall architecture, depicting the main components and their interconnections.

The BC accelerator is designed for bioinformatics analysis, specifically to accelerate the DNA basecalling process. It works alongside the RISC-V core, where the core manages control and orchestration, while the BC accelerator offloads computationally heavy tasks, boosting overall performance and efficiency. Shared cache resources between the RISC-V core and the accelerator ensure seamless communication and efficient resource utilization.

The system also includes on-chip memory for the BC accelerator, with a capacity of 72 KB, ensuring rapid data access during computations. The SoC incorporates features like a branch history table (BHT) [78], branch target buffer (BTB) [79], and return address stack (RAS) [80] to improve instruction flow, further enhancing the performance of the RISC-V core. These features contribute to a robust design capable of meeting the performance needs of portable DNA sequencing applications.

Table 4.1: The First SoC Processor and Accelerator Parameters

Design Parameter	Value
RISC-V ISA	RISCV64G
Data width	64 bit
Instruction Cache (4-way)	16 KiB
Data Cache (4-way)	16 KiB
Translation lookaside buffer (TLB) entries	8
Branch history table (BHT) entries (2-b)	4096
Branch target buffer (BTB) entries	62
Return address stack (RAS) entries	2
Accelerator on-chip memory	72 KiB

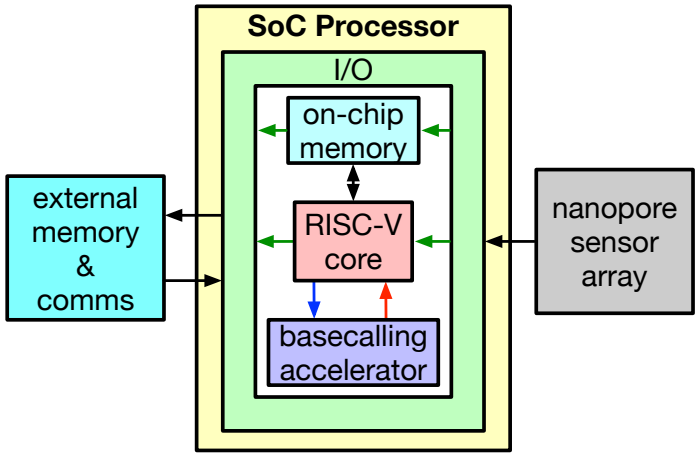

Figure 4.2: System Overview of the RISC-V and the Basecalling (BC) Accelerator in the SoC.

Table 4.1 provides a detailed summary of the SoC's design parameters, including the processor and accelerator specifications. Key design features such as the instruction and data cache sizes, translation lookaside buffer (TLB) entries, and other control-related modules are highlighted, which together contribute to the overall performance and efficiency of the system.

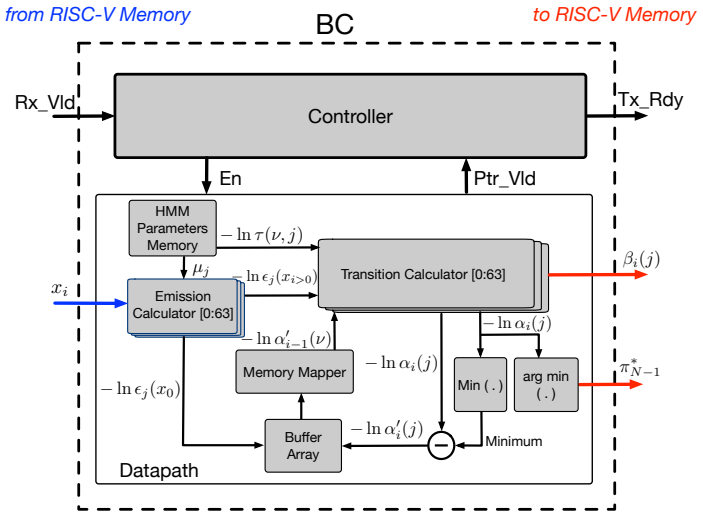

Figure 4.3: The architecture of the proposed basecalling acceleration engine. Note: trellis contruction shown only.

4.3 HMM Basecalling Acceleration Design

The block diagram shown in Fig. 4.3 represents the general architecture of our proposed basecalling acceleration engine, in essence an implementation of the 64-step inner-loop (lines 5 and 6) of Algorithm 1. The engine receives its input, the event x_i, from the RISC-V CPU's main memory and sends its output, the event pointers $\beta_i(j)$ and the end-state index π^*_{N-1}, back to the RISC-V CPU's main memory in a real-time streaming fashion. More details about the memory interface are provided in § 4.4.

A couple of design choices immediately occur for this system. First, as previously mentioned and also detailed below, to accommodate the level of independence present across the 64 iterations of states j, we employ a fully-unrolled loop-level

parallel hardware implementation. Thus, as illustrated in Fig. 4.3, 64 parallel `emission` and `posterior` calculation blocks ("slices") comprise the datapath and simultaneously compute the 64 pointers $\beta_i(j)$ corresponding to each new event input x_i. These slices carry out the majority of the design's arithmetic.

A second important consideration, directed at improved computational speed and efficiency, is to simplify the arithmetic computations of $\epsilon_j(x_i)$ and $\alpha_i(j)$ themselves. In this vein, following standard practice [81], the basecalling engine computes the negative logarithm of the underlying probabilities rather than the exact probabilities themselves. This requires that the *max* operation be replaced with a *min* and, most importantly, it reduces (4.2) to

$$-\ln \epsilon_j(x_i) = (x_i - \mu_j)^2 \tag{4.3}$$

where the terms, $\ln(2\pi)$ and $1/2\sigma^2$, have been removed as they serve only as constant offsets to the calculations, without any contribution to the relative probabilities of the states upon which the choice of a final optimal sequence relies. Thus, the single state emission calculation is comprised of a 2-input subtractor and a 2-input multiplier as portrayed by Fig. 4.4. It should be noted that the arithmetic logic in Fig. 4.4, as well as in Fig. 4.5, is pipelined by placing registers in between consecutive logic stages. For simplicity, the provided figures do not include those pipelining registers and other detailed logic of the actual hardware synthesized by the design tools.

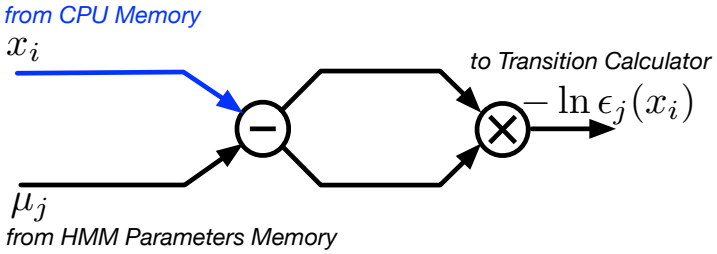

Figure 4.4: The arithmetic arrangement of a single slice from the emission calculator.

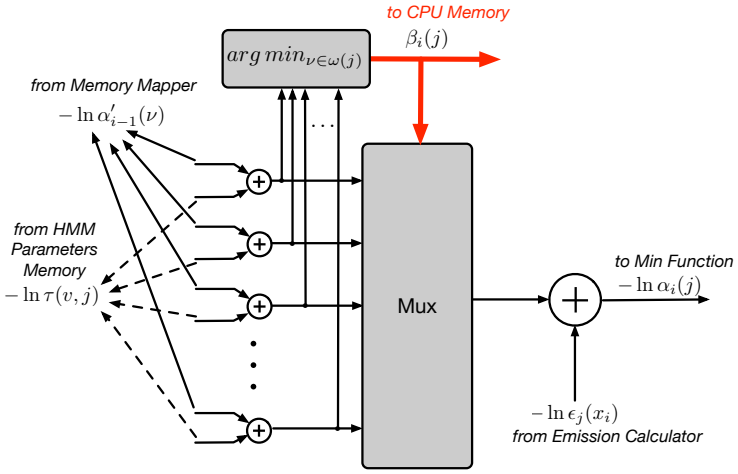

Figure 4.5: The architecture of single slice from the Transition Calculator.

Similarly, the hardware-computed logarithmic posterior and pointer updates become

$$\ln \alpha_i(j) \leftarrow \ln \epsilon_j(x_i) + \min_{\nu\in\omega(j)} [\ln \alpha_{i-1}(\nu) + \ln \tau(\nu, j)] \tag{4.4}$$

$$\beta_i(j) \leftarrow \arg\min_{\nu\in\omega(j)} [\ln \alpha_{i-1}(\nu) + \ln \tau(\nu, j)] \tag{4.5}$$

The function of each building block for the proposed engine architecture in Fig. 4.3 is as follows.

- The `Controller` block maintains the data flow configuration and timing for all other components of the engine. In particular, it signals enable permissions for each block to indicate the start and end of operation cycles, and receives valid status signals indicating elapsed calculations for event pointers and the end-state pointer. It also manages the engine's external communication with the CPU, relying on valid and ready signals to register the arrival of new input events and to indicate that the engine is done calculating a new set of 64 pointer values, respectively.

- The `Transition Calculator` block is the engine's main processing chain used to compute the 64 posteriors and pointers for each new incoming event x_i using a parallel structure. Each slice of the Transition Calculator handles the computations in (4.4) and (4.5) for each new event. Fig. 4.5 shows the detailed structure of each slice instance which is composed of twenty one 2-input adders, arg min function, multiplexer, and 2-input adder. The twenty one adders are responsible for adding $\ln \alpha_{i-1}(\nu)$ to $\ln \tau(\nu, j)$ as in (4.4). The arg min function and the multiplexer are then used to find the state pointer $\beta_i(j)$ as in (4.5) and the minimum value as in (4.4), respectively. More specifically, the arg min function is implemented as a 21-input comparator circuit which outputs the index of the minimum value of the 21 input values coming from the adder circuits. The last adder finalizes the addition of $\ln \epsilon_j(x_i)$ to

the mux output and calculates $\ln \alpha_i(j)$ as in (4.4).

- The `HMM Parameters Memory` is a memory block which stores the HMM's transition and emission models' parameters (i.e., $\ln \tau(\nu, j)$ and μ_j, respectively).

- The `Emission Calculator` block calculates the negative logarithm of the 64 emission probabilities $-\ln \epsilon_j(x_i)$ for each new input event x_i according to (4.3) and Fig. 4.4. Similar to the Transition Calculator block, the 64 emission calculations are executed in a parallel fashion using 64 identical hardware instances (slices) based on the emission model parameters (i.e., μ_j) stored in the `HMM Parameters Memory` block.

- The `Min` block ensures that the engine's recursive computations across the succession of input events do not experience overflow, a functionality not described in the Algorithm 1 pseudocode. Thus, the Min block finds the minimum of the 64 posteriors calculated by the Transition Calculator block for each event. The minimum value is then subtracted from the event posteriors before storing the result (i.e., $\ln \alpha'_i(j)$) in the buffer array.

- The `arg min` block outputs the end-state index (i.e., π^*_{N-1}) for the last event as in line 9 of Algorithm 1. In particular, the arg min block is implemented as a 64-input comparator circuit which outputs the index of the minimum value

of the 64 posterior values coming from the Transition Calculator block at the end of processing event x_N.

- The `Buffer Array` block is used to store the normalized posteriors for each event (i.e., $\ln \alpha'_i(j)$) in preparation for the iteration triggered by the arrival of the next event x_i.

- The `Memory Mapper` block is used to allocate the posterior set, $\omega(j)$, designated to each instance of the transition calculator block based on the unique 21 transition states to its state index j. Specifically, the memory mapper is a crossbar switch (with fixed interconnect links) which maps the 21 posterior values to each state slice based on its index `j`. For each state slice instance, the 21-posteriors set is fetched from the 64 normalized posteriors calculated for the previous event and stored in the buffer array.

For the first event ($i = 0$) only the 64 emission values (i.e., $-\ln \epsilon_j(x_o)$) need to be calculated (since all preceding values are assumed to be zero). Thus, the engine's controller puts the Transition Calculator in an idle state for the first input event and only the Emission Calculator block is allowed to initialize the Buffer Array.

For the second and all other subsequent events, the engine operates in the same manner as follows. The HMM Parameters Memory, Emission Calculator and the Memory Mapper provide the Transition Calculator with the logarithm of tran-

sition probabilities, the current event's emission probabilities, and the previous event's posterior probabilities. Based on these three inputs, the transition calculator evaluates the current event posteriors and pointers according to (4.4) and (4.5), respectively. Only the pointers are sent to the component that performs the traceback computation (either CPU or another attached accelerator). Once the engine reaches the last event ($i = N - 1$), the end-state index (i.e., the arg min function's output) is sent to the RISC-V CPU with the pointers (i.e., equivalent to line 9).

To further enhance the performance of the BC accelerator, we introduced a traceback block that eliminates the need to send out pointers and end-state indices. Instead, the traceback block directly sends out the decoded state sequence, thereby reducing communication overhead and speeding up the overall processing.

The traceback block is designed to wait until all pointers corresponding to an M-long event sequence are accumulated in the pointer buffer. Once the trellis constructor hardware signals the delivery of all trellis pointers, the traceback state sequence calculation begins, iterating $M - 1$ times through the traceback loop as depicted in Fig. 4.6.

During each iteration (e.g, pointers for each event coming out from BC), the traceback mechanism retrieves a string of N relative trellis pointers from the pointer buffer, corresponding to the current event. This string is accessed using the address signal `m`, and it is read from the pointer buffer in a last-in-first-out (LIFO) manner

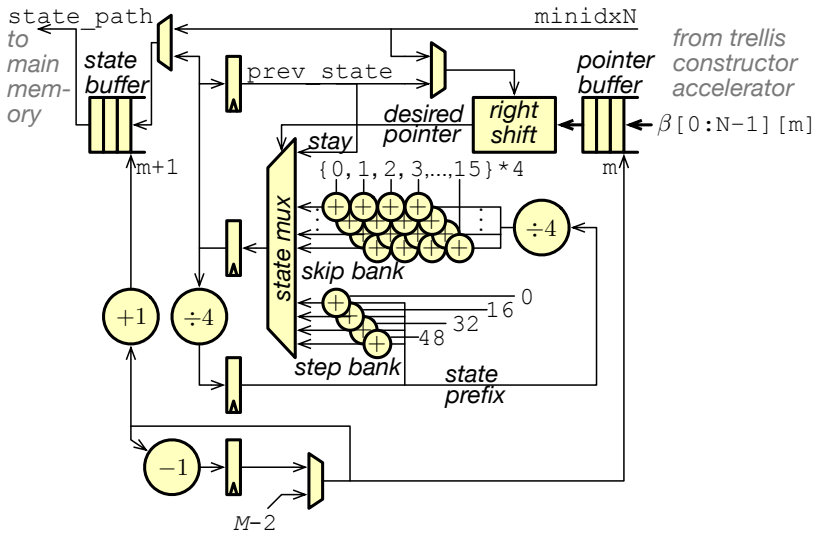

Figure 4.6: The traceback acceleration block diagram [82].

starting from location `m`$=M-2$. From this fetched string, the desired pointer is extracted by shifting the pointer string to the right by `prev_state` bytes, with the least-significant byte being selected as the desired pointer. At the start of execution, the initial pointer is determined using the `minidxN` value, which is provided by the previous BC accelerator.

By incorporating the traceback block, the basecalling accelerator becomes more efficient by directly producing the final decoded sequence without the need for intermediate pointer transmission, thereby significantly improving the overall throughput of the system.

4.4 Accelerator Integration

The integration of the Hidden Markov Model (HMM) basecalling accelerator with the RISC-V processor is a key aspect of the overall SoC architecture, facilitating efficient computation and data movement. The basecalling accelerator is connected to the Rocket RISC-V CPU and L1 Data Cache via the RoCC (Rocket Custom Coprocessor) interface [76], as illustrated in Fig. 4.7. The design ensures that the RISC-V core can offload basecalling computations to the dedicated accelerator, enhancing performance for DNA sequencing operations.

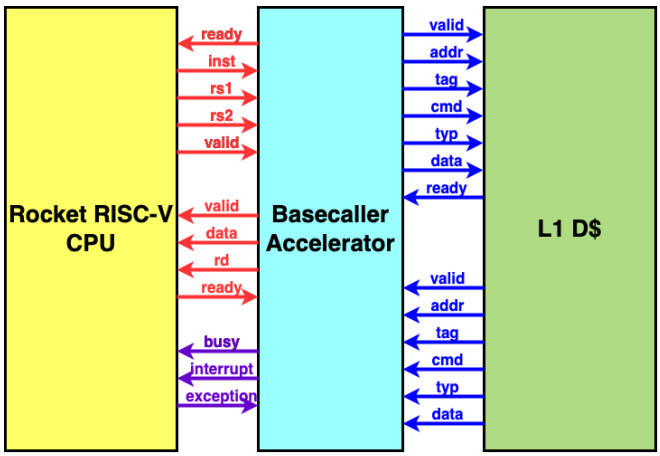

Figure 4.7: RISC-V CPU and the BC accelerator integration through RoCC Interface.

The BC accelerator connects to the RISC-V CPU through command and operand signals, which are used for communication between the CPU and the accelerator. In the Fig. 4.7, the red signals represent these command and operand lines. The inst signal from the CPU provides specific commands, while `rs1` and `rs2` carry

the operands needed for basecalling computations. The accelerator processes these instructions and sends the results back to the CPU using the `rd` signal. Signals like `valid` and `ready` (also shown in red) help synchronize these transfers, ensuring that data moves between components only when both sides are prepared, thus avoiding timing conflicts.

The accelerator is also tightly integrated with the L1 D$, which serves as a shared memory resource. The interaction between the basecalling accelerator and the L1 D$ involves address, data, and control signals to manage data access efficiently, represented by blue lines in Fig. 4.7. The `addr` signal determines the memory address for accessing the cache, while control signals such as `tag`, `cmd`, and `typ` specify the type of operation being performed. Data is transferred using dedicated data lines, which enable the accelerator to quickly read from or write to the cache. The `valid` and `ready` signals are crucial here as well, ensuring that memory requests and responses occur without delays, reducing overall latency and improving efficiency.

Control and status signals are represented in purple and are essential for effective integration. The BC accelerator uses signals like `busy`, `interrupt`, and `exception` to maintain proper coordination with the RISC-V CPU. The `busy` signal informs the CPU when the accelerator is occupied, preventing new commands until the current task completes. The `interrupt` signal allows the accelerator to notify the

CPU upon task completion or when an important event occurs, and the `exception` signal reports any operational errors, ensuring robust error handling.

This integration through the RoCC interface allows the RISC-V CPU to delegate computationally intensive tasks to the basecalling accelerator while maintaining overall control of the sequencing process. The accelerator specializes in performing the complex calculations required for DNA basecalling, which helps to reduce the computational load on the CPU. The shared use of the L1 Data Cache enables rapid access to data for both the CPU and the accelerator, minimizing latency and improving the system's overall efficiency.

The tightly integrated workflow between the CPU, BC accelerator, and L1 cache creates an efficient processing environment suitable for real-time DNA sequencing. The combination of command, data, and control signals ensures efficient communication and workload distribution, avoiding bottlenecks and resource contention. This setup allows for a well-balanced combination of computational power and energy efficiency, making the SoC a robust platform for mobile DNA sequencing applications.

4.5 Control and Communication

The BC accelerator communicates with the RISC-V processor through custom instructions used in software, which are specifically designed to control and manage

its operations. These custom instructions are issued through the RoCC interface, providing the necessary control to orchestrate the computations required for the HMM used in basecalling. Fig. 4.8 illustrates the custom instruction format, which is fundamental for effective control of the BC accelerator and ensures seamless interaction between the accelerator and the Rocket RISC-V CPU. It consists of several fields that control various aspects of communication and data transfer. The `FUNCT` field (31-25) is used to specify the function to be executed by the accelerator, while `RS2` and `RS1` (25-20 and 20-15) provide the source register operands that include the necessary parameters for executing a given function. The `RD` field (15-12) is used to specify the destination register where the result of the accelerator's operation will be written. Finally, the `OPCODE` field (7-0) indicates that this is a custom RoCC instruction intended for the basecalling accelerator.

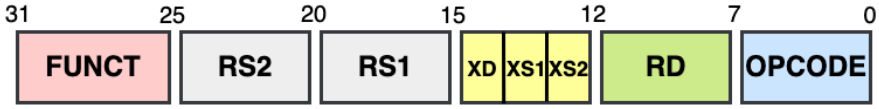

Figure 4.8: Custom 32-bit RISC-V Instruction Format for the BC Accelerator.

Table 4.2 provides an overview of the specific instructions used to operate the BC accelerator. For example, a `FUNCT` value of 0 is used to reset the BC accelerator, while a value of 1 sends the number of events. Instructions with `FUNCT` values from 2 to 4 are used to send model parameters, such as the mean (μ_j), transition (τ_j), and standard deviation (σ_j). Finally, the instruction with `FUNCT` value 5 is used to

send the event data and receive the decoded state path from the accelerator.

Table 4.2: The custom RISC-V instructions to operate the BC accelerator. Note: The OPCODE is zero by default.

FUNCT	RS2	RS1	RD	DESCRIPTION
0	0	0	0	**Reset the BC**
1	0	N	0	**Send Number of the events**
2	0	μ_j	0	**Send the model (1_{st} addr) to BC**
3	0	τ_j	0	**Send the transition (1_{st} addr) to BC**
4	0	σ_j	0	**Send the standard deviation (1_{st} addr) to BC**
5	0	x_i	$\hat{a}_{0\rightarrow N}$	**Send events (1_{st} addr), and wait to receive the state path.**

By utilizing these custom instructions, the RISC-V CPU is able to precisely control each phase of the basecalling process. This flexibility enables a well-coordinated sequence of operations, from initializing the BC accelerator, sending necessary parameters, and processing events, to ultimately retrieving the final state path. The custom instruction interface, coupled with the efficient communication mechanism through the RoCC, provides a high level of performance, making the basecalling accelerator a highly efficient component for real-time DNA sequencing tasks.

4.6 FPGA Evaluation and Measurements

To evaluate the performance of the first SoC design, FPGA prototyping was utilized as a crucial step in verifying the functionality and performance of the HMM acceler-

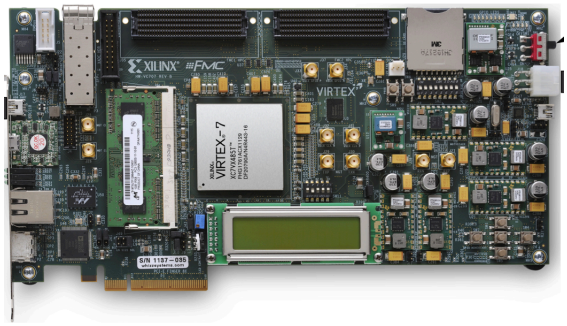

Figure 4.9: The system hardware emulation using Xilinx VC707 FPGA board.

ator within a flexible and iterative development environment. This process involved testing key architectural components, validating the functionality of the proposed accelerator, and benchmarking basecalling performance under realistic workloads. The insights gained from the FPGA evaluation provided valuable feedback for refining the design and optimizing the accelerator's performance in subsequent ASIC implementation.

4.6.1 FPGA Implementation

The proposed acceleration engine, illustrated in § 4.3 is linked to and coordinated with a x86 CPU through PCIe bus. This interface allows the proposed FPGA

accelerator to serve as a rapid real-time co-processor for a CPU basecaller and thus allows it to fit conveniently within a larger DNA sequencing pipeline. To simplify the understanding of this communication framework, Fig. 4.10 shows the high-level architecture of the CPU-FPGA communication interface. The proposed core acceleration engine (i.e., discussed in the previous subsection) is implemented on a Virtex-7 FPGA device shown in Fig. 4.9 whereas the basecaller's main C-program is running on a host CPU.

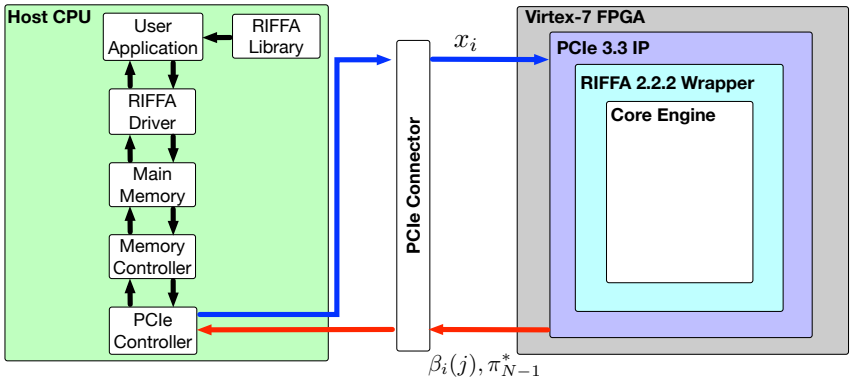

Figure 4.10: High-level architecture for the CPU-FPGA hardware interface.

On the FPGA (hardware) side, the proposed core engine is wrapped by the Reusable Integration Framework for FPGA Accelerators (RIFFA) [83]. RIFFA is an open-source collection of software libraries and hardware designs to enable real-time streaming between CPUs and FPGAs through PCIe. In particular, RIFFA allows stitching the core engine to a PCIe endpoint (i.e., Xilinx PCIe IP) using so-called Rx and Tx engines which receive and generate PCIe packets, respectively.

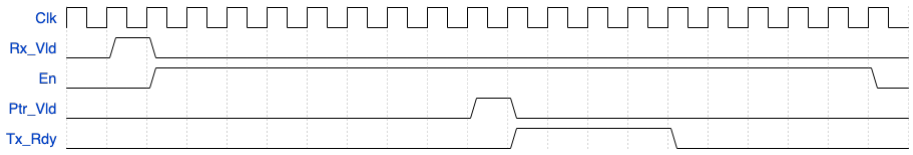

Figure 4.11: Simplified timing diagram for RIFFA interface/core engine handshaking protocol

The RIFFA packet size may be configured by the user to 32-bits, 64-bits or 128-bits.

On the CPU (software) side, the HMM basecaller (i.e., the user application) described in Algorithm 1 is implemented using C programming language. As discussed in the previous subsection, this code offloads the basecaller's computation-intensive loop in lines 5 and 6 of Algorithm 1 to the proposed acceleration engine. This takes place with the aid of a simple RIFFA API that includes send and receive functions that are inserted by a developer directly in a C program to facilitate data exchange between the CPU and hardware-mapped kernels (i.e., `fpga_send()` for CPU transmission to the FPGA, and `fpga_recv()` for CPU reception from the FPGA). The RIFFA driver then allows these API functions to access the CPU's main memory. The CPU's PCIe controller then allows communicating the CPU's main memory data via the memory controller in the form of PCIe packets. As previously illustrated in Fig. 4.3, the data sent from the CPU to the FPGA is the sequence of nanopore events x_i whereas the FPGA sends back 64 pointers $\beta_i(j)$ for each event as well as the *end-state index* of the last event (i.e., π^*_{N-1}) to the CPU.

The interaction between the proposed acceleration engine and the RIFFA in-

terface signals can be illustrated with the aid of Fig. 4.11. This simplified timing diagram provides a high-level description for the processing of a single event (which repeats until the last event). First, the CPU-side basecalling C program, with the aid of the RIFFA driver, packs the N events in a sequence of 128-bit event message packets. Each event is alloted a 32-bit word, thus, each message packet carries four events. When the C program hosted by the CPU invokes the `fpga_send()` function to send a message packet to the FPGA over the PCIe bus, the arrival of the packet is signified by the status signal `Rx_Vld`.

Once the valid signal is received by the basecalling engine's Controller, it enables the appropriate datapath blocks for processing the first event in the received message packet using the `En` signal for 18 clock cycles (i.e., the overall latency of the data path blocks for processing a single event).

As soon as the *arg min* block has the 64 states' pointers calculated, the Transition Calculator block sends the `Ptr_Vld` signal to the Controller. The Controller then arranges the 64 calculated pointers (each allotted 1 byte) into four 128-bit pointer message packets, queues these message packets on the core engine's output data port, and asserts a ready signal, `Tx_Rdy` informing the remainder of the transmission-side IP that data intended for the CPU is ready to be sent. Four cycles of the basecalling engine's clock are needed to send the four pointer messages. The processing of the remaining three input events in the received message packet

is handled in the same fashion before receiving the following packet's valid signal.

For the first event the Controller does not signal `Tx_Rdy` since no pointers are calculated for the first event. On the other hand, at the last event the Controller signals `Tx_Rdy` for five clock cycles (instead of four) to send one extra RIFFA packet carrying the end-state index (i.e., π^*_{N-1}) to the CPU.

4.6.2 FPGA Measurements

The performance of the proposed FPGA-accelerated basecalling engine (§ 4.3) for Timp's basecaller in [73] is evaluated in terms of speed and power consumption across different accuracy levels. The performance of the basecalling engine is compared with a traditional CPU-based implementation coded in C. The CPU used for the evaluation is a 12-core Intel Xeon E5-2620 v3 clocked at 2.4GHz with 32GB of RAM. The basecalling engine is designed and implemented using Xilinx Vivado HLx v2016.1 tools. The target FPGA device is a Virtex-7 (XC7VX485T-2FFG1761C), which contains 75,900 slices, each comprising 4 Look-Up Tables (LUTs) and 8 Flip Flops (FFs).

To guide the FPGA implementation, we built a bit-true MATLAB system-level simulator to evaluate the fixed-point basecaller's prediction accuracy performance at different bit-widths. To account for possible noise effects in the measurement system, we examined the basecaller's behaviour at different input signal-to-noise

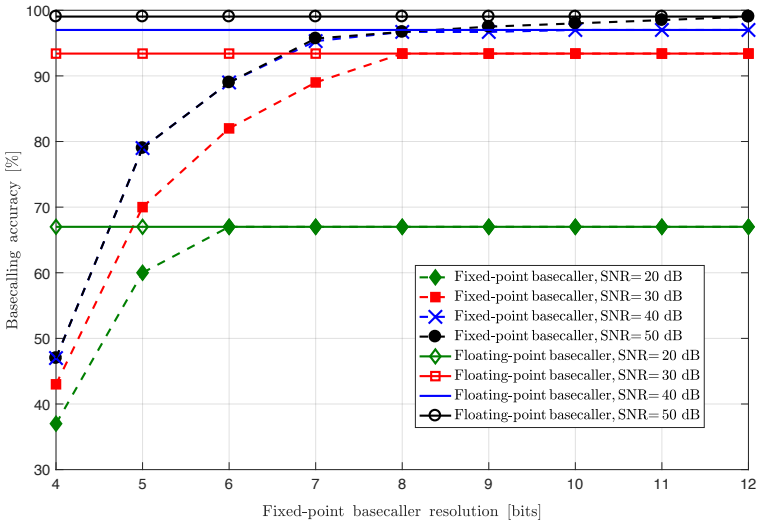

Figure 4.12: Fixed-point basecaller accuracy as a function of bit resolution under different input SNR settings.

ratios (SNRs) from the sensor channel. In addition to the sensor's additive noise, the HMM basecaller's accuracy depends on its ability to mitigate the stay and skip translocation mechanisms discussed above. Lacking detailed reports on this issue, we have assumed a 10% stay probability and a 10% skip probability as exemplary values for hardware analysis.

The simulation results depicted in Fig. 4.12 show how the accuracy of the fixed-point basecaller changes with the bit-width at different SNR values for the sensor channel. At each SNR, the accuracy is measured with respect to a floating-point equivalent. As expected, the accuracy for the fixed-point basecaller increases with increasing bit-width until it saturates at the value attained by its floating-point counterpart.

From another perspective, the results show that the bit-width value required to saturate the fixed-point basecaller accuracy increases with increasing SNR (e.g., 6-bits for 20dB, 8-bits for 30dB, etc). This could be justified by the dominance of the sensor channel SNR over the bit-width in determining the fixed-point basecaller's accuracy. In particular, in case of a highly corrupted sensor signal (i.e., low SNR) the ability of the fixed-point basecaller to make accurate predictions of the original DNA sequence is very low due to the noise distortion level regardless of its computational resolution (bit-width). That is why, for instance, at 20-dB SNR the accuracy (i.e., 67%) at a resolution of 6-bits is the same as that for 12-bits. On the other hand, in case of a low-noise signal (i.e., high SNR signal), the major limiting factor for the fixed-point basecaller's prediction accuracy is its resolution. This could be seen in the 50-dB SNR case at which the basecaller's resolution must be 12-bits to match the floating-point accuracy of 99.04%. It is worth mentioning here that besides the sensor additive noise and the basecaller's resolution, the skip and stay state transitions are still an additional source of discrepancy for the sensor signal and limit the basecaller's accuracy, a deep study of these effects is beyond the scope of this thesis.

The results presented in Fig. 4.13 demonstrate a dramatic improvement in the basecalling speed for the proposed FPGA-accelerated basecalling engine compared to a CPU-only implementation, thanks to the parallel computing which may be

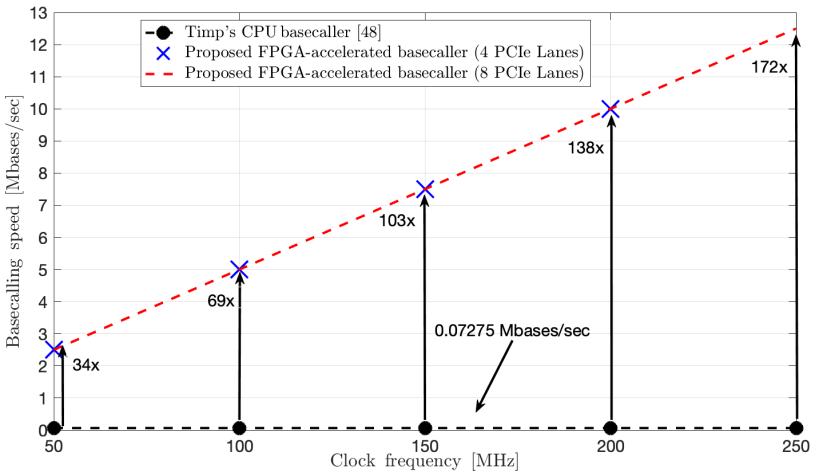

Figure 4.13: Measured FPGA-accelerated and CPU-only basecalling speed for Timp's basecaller [73] as a function of FPGA clock frequency. Numbers denote the relative speed improvement of the FPGA-accelerated design relative to the CPU alone.

realized in FPGA devices. As expected, the basecalling speed improvement tracks the clock frequency. The Register Transfer Language (RTL) model for the basecaller is successfully synthesized and implemented on the target FPGA device at a clock frequency of 250 MHz when using 8 lanes of the Xilinx PCIe 3.3 IP. At this frequency the FPGA-accelerated basecalling speed exceeds the CPU-only implementation by a factor of 172×. In addition, it could be observed that despite the accelerated basecaller attaining similar speeds on both the 4-lane and 8-lane PCIe configurations (i.e., the bandwidth of the 4-lane PCIe channel is accommodating the basecalling engine bit rate), only the 8-lane interface-based basecaller could be implemented at a 250-MHz clock frequency. This pertains to the higher AXI clock frequency (i.e., used to generate the basecalling engine clock) of the 8-lane interface

(250 MHz) compared to that for the 4-lane (125 MHz). However, the higher maximum clock frequency attained by the 8-lane interface compared to the 4-lane comes at the expense of the circuit power consumption as will be shown in the following paragraphs.

It should be noted that the 172× speed-up in favor of the proposed accelerator compared to the CPU is due to the parallel computing for three major parts of Algorithm 1. The first part is unrolling the inner loop in lines 5 and 6 for the 64 hidden states of the HMM. The second part is the 21 transition additions in (4.4) executed for each of the 64 states' calculations. The third part is the normalization subtraction which is executed in parallel on the FPGA accelerator for the 64 posteriors output from the Transition Calculator block (in Fig. 4.3) at the end of the processing cycle of each input event. Therefore, the aforementioned parallel computations lead to a speed-up factor of 147× (i.e., (21x64+64)/(2.4GHz/250MHz)) for 250MHz clocked FPGA compared to the 2.4GHz CPU. The residual speed-up factor is attributable to other PC computer architecture-related factors (e.g., cache misses, procedure calls, multi-core communications, etc.).

The practical significance of the results reported in Fig. 4.13, should be considered with respect to the corresponding power consumption and hardware utilization costs. This will better enable the system designer (or even the application layer user) to select the most efficient configuration for the whole system to meet the

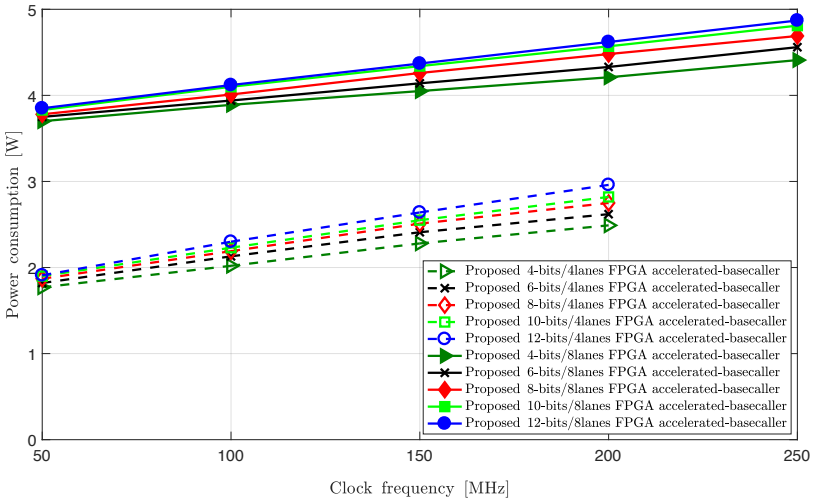

Figure 4.14: FPGA-accelerated basecalling power consumption.

target application requirements (e.g., energy-efficiency).

We start by considering the measured average operational power consumption of the FPGA accelerator in Fig. 4.14 as a function of clock frequency. Both the 4 and 8-lane configurations consume on average 8% and 16%, respectively, of the CPU's average "incremental" power, 25.16 W (i.e., the extra power consumed *while* the CPU executes the basecalling algorithm, otherwise the PC's total consumed power is 87.15 W). For further illustration, in Fig. 4.15, the results of Fig. 4.13 and Fig. 4.14 are utilized to demonstrate the energy efficiency (bases computed per joule) of the proposed accelerator compared to the CPU. Fig. 4.15 confirms that the proposed accelerator is more energy-efficient than the CPU by at least 225× for the 8-lane design at 50 MHz (and 1390× for the 4-bit design with 4 lanes at 200 MHz). The relative energy-efficiency improvement of the proposed accelerator

with increasing clock frequency reflects its consistent linear increase in basecalling speed (as shown in Fig. 4.13) relative to a marginal increase in the corresponding power consumption (as shown in Fig. 4.14). The power-frequency change ratio demonstrated in Fig. 4.14 is $\sim 1 : 3.5$ (rather than the theoretical $1 : 1$). This pertains to the fact that the power consumption reported in Fig. 4.14 encapsulates not only that of the core basecalling engine, but also the PCIe and RIFFA IP while the abscissa refers only to the basecalling engine's clock frequency. Thus, increasing the clock frequency only increases the power consumption of the basecalling engine while the power consumed by the communications IP portions remains invariant. Moreover, the power of the basecalling engine occupies an average of 8.25% of the total power consumption budget of the accelerator circuit reported in Fig. 4.14. Due to these facts, the marginal increase in the power consumption with the clock frequency illustrated in Fig. 4.14 could be understood.

It should be noted that the PCIe IP and RIFFA IP in the proposed accelerator communicates with a different and independent clock frequency (i.e., AXI interface clock) to the basecalling clock frequency (i.e., used in Figs. 4.13, 4.14, 4.15). Hence, the divergence of the EE lines when increasing the basecalling clock frequency is notable in Fig. 4.15 especially for the 4-lanes curves for different circuit resolution. This stems from the fact that at low basecalling clock frequencies the dynamic power consumed by the core basecalling engine is still not big enough compared to the

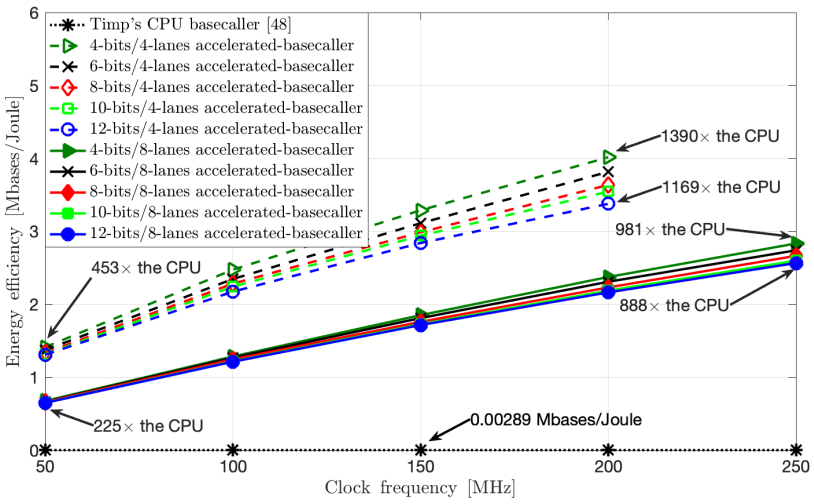

Figure 4.15: Basecalling energy-efficiency of the FPGA and CPU, numeric labels denote the relative efficiency improvement of the FPGA.

bigger constant power (i.e., dependant on the constant AXI clock) consumed by the PCIe and RIFFA logic elements to show a reasonable difference in the total power consumed by the whole circuit. However, at higher basecalling clock frequencies, the dynamic power of the core basecalling engine starts to show more impact as it becomes relatively closer to the constant power component for the PCIe and RIFFA IPs.

In addition to the power consumption, the hardware utilization and the energy density (E-D) (i.e., elaborated below) of our proposed basecalling engine on the target Virtex-7 FPGA device is illustrated in Table 4.3. The table shows the resource utilization percentages for the LUTs, FFs and DSP slices with different complexities (i.e., resolution) of our basecalling circuit. As expected, resource usage

increases with bit-width. Table 4.3 does not show a substantial difference between using 4-PCIe lanes and 8-PCIe lanes. However, 14% of the Gigabit Transceiver (GTX) resource (i.e., not reported in the table) is utilized in case of the 4-lanes design (i.e., using 4 transceivers out of 28) compared to 29% for the 8-lanes (i.e., using 8 transceivers out of 28). It is worth mentioning that the aforementioned GTX resource utilization for the 4 and 8-lane designs directly justifies the power offset between both designs as noticed in Fig. 4.14. In addition to the resource utilization, Table 4.3 also shows the E-D at the maximum clock frequency (that is, 200 MHz for the 4-lane interface and 250 MHz in case of 8-lanes interface) for each design resolution. The basecaller is assumed to decode a long genome with a length of 1 Gbp. The E-D metric (i.e., measured in Joules/slice) is another important metric that determines the energy efficiency of the utilized FPGA resources by combining the actual number of the utilized FPGA slices and its corresponding energy consumption. All the E-D values reported in Table 4.3 are in mJoules/slice. The increase in the ED for both 4 and 8 lanes accelerators with increasing resolution shows consistency with the results in Fig. 4.15 which conveys that the lower the basecaller's resolution, the higher the energy efficiency.

Finally, it is worth mentioning that the utilization of the proposed FPGA accelerator in the HMM basecalling function in this work does not limit its applicability to other HMM-based algorithms such as Pair HMM[84] and Profile HMM[85]. In

Table 4.3: FPGA device utilization, performance density and E-D.

	4-PCIe Lanes				8-PCIe Lanes			
	LUT	FF	DSP	E-D	LUT	FF	DSP	E-D
4-bits	12%	6%	53%	1.47	12%	6%	53%	1.76
6-bits	14%	7%	55%	1.66	15%	7%	55%	1.95
8-bits	17%	8%	55%	1.72	18%	8%	55%	2.0
10-bits	20%	9%	57%	1.87	21%	9%	57%	2.05
12-bits	23%	10%	59%	1.91	24%	11%	59%	2.15

case of Pair HMM, the accelerator can be efficiently utilized to accelerate the initialization and recursion sections of the algorithm similar to the initialize and iterate blocks, respectively, of the Algorithm 1. On the hand, the accelerator can also speed-up the database search for the Profile HMM algorithm by implementing parallel instances of accelerated Pair HMMs to rapidly identify the sequence family of an arbitrary DNA sequence.

4.7 ASIC Evaluation and Measurements

The evaluation of the first SoC's ASIC implementation provided critical insights into the design's real-world performance. By fabricating and testing the chip, key

metrics such as area, power consumption, and energy efficiency were assessed to validate the design objectives and identify opportunities for further optimization. The ASIC evaluation also highlighted the integration and functionality of the HMM accelerator within a fixed hardware environment, offering a comprehensive understanding of the SoC's capabilities and limitations in handling basecalling tasks. These results served as a foundation for iterative improvements in future SoC designs, ensuring better scalability and efficiency for DNA sequencing applications.

4.7.1 System Setup for ASIC Validation

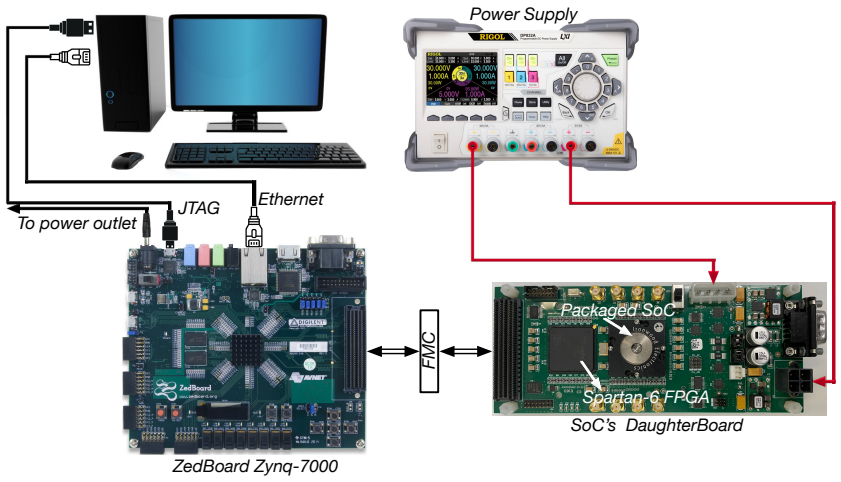

Figure 4.16: SoC test setup involving the customized test board, ZedBoard, and 2-channels power supply.

The test setup for the System-on-Chip (SoC), as illustrated in Fig. 4.16, involves multiple components to ensure a comprehensive evaluation of the design. It includes a customized test board equipped with a Spartan-6 FPGA, used for initial

signal processing and configuration, as well as a ZedBoard based on the Zynq-7000 platform, which serves as the core system controller. Additionally, the setup utilizes a two-channel power supply to precisely control and monitor power delivery to both the customized test board and the SoC chip. A computer is also included for programming, monitoring, and data collection through interfaces like JTAG and Ethernet. This integrated test system was originally developed and described by the BSG group [86].

The details of the SoC test system used for evaluation are illustrated in Fig. 4.17. The x86 CPU serves as the primary control station. It is responsible for programming both the Zedboard and the FPGA on the test board. Additionally, it provides remote terminal access through an Ethernet connection, allowing the operator to issue commands, program devices, and conduct debugging and testing tasks efficiently. The ZedBoard, integrates an ARM host CPU (Processing System, PS) and Custom FPGA Logic (Programmable Logic, PL). The ARM host CPU runs a front-end server called `fesvr` C executable, which communicates with the Control and Status Registers (CSRs) of the RISC-V core on the SoC. The `fesvr` also handles transferring data from the SD card to the DDR memory of the RISC-V core, enabling it to access necessary data during basecalling operations. The ARM CPU is responsible for Ethernet communication with the x86 CPU, allowing for remote management of the system. The Zedboard's programmable logic contains

the Custom FPGA Communications Logic (PL), which facilitates data transfer between the ARM CPU and the test board. This connection is achieved using the AXI interface for internal communication between the PS and PL. The Zedboard and the test board are connected via an FMC (FPGA Mezzanine Card) connector, which provides high-speed, reliable communication between the two components. On the test board, the SoC includes the basecalling accelerator. The Custom FPGA Communications Logic on the test board handles data transmission between the SoC and the Zedboard. Commands and data are sent from the ARM CPU to the SoC, and the basecalling accelerator processes these tasks as required. JTAG is utilized for low-level programming and debugging of the FPGA and the SoC. This connection allows direct access to internal registers, enabling detailed testing and validation of the system.

4.7.2 ASIC Implementation

As discussed in § 4.3, the BC accelerator is composed of the trellis construction and traceback components. When designing the first chip, we wanted to evaluate how different hardware configurations, specifically involving the traceback block, could improve the overall basecalling speed.

To achieve this, two accelerators were incorporated into the chip design. The first and second accelerators are nearly identical, except that the first accelerator

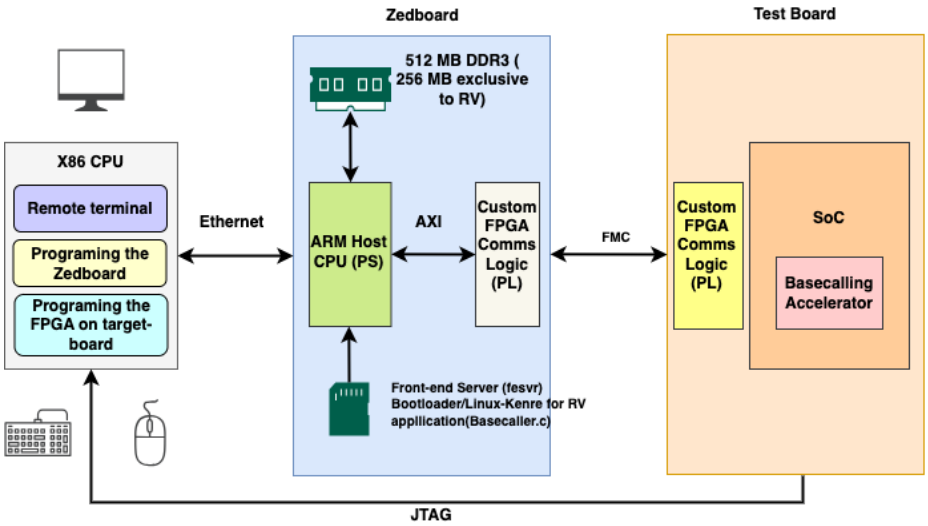

Figure 4.17: A detailed diagram illustrating the interaction between the x86 CPU, ZedBoard, and the test board containing the SoC for Measurements.

(AccelA) relies on the RISC-V CPU to handle the traceback operations, while the second accelerator (AccelB) includes a dedicated traceback block, as shown in Fig. 4.6. This setup allowed an assessment of the performance impact of having an exclusive hardware module for traceback.

Fig. 4.18 illustrates the overall SoC design, showcasing how these two accelerators fit into the broader system. By comparing these designs, valuable insights were gained into the benefits of integrating traceback directly into the hardware to minimize latency and improve basecalling throughput.

Following the successful FPGA evaluation of the HMM-based basecalling accelerator, it was integrated into a full SoC for an extensive round of testing and performance validation. To simulate a realistic operational environment for this

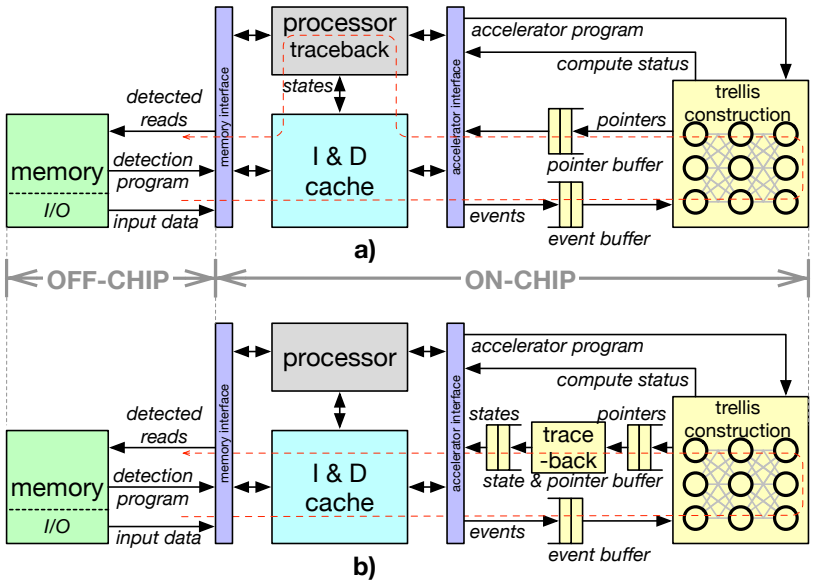

Figure 4.18: The SoC architectur features two accelerators [82]: a) AccelA: the accelerator for trellis only; b) the accelerator for both trellis and traceback.

SoC, a custom-designed evaluation platform was employed. This platform utilized two interconnected FPGAs, creating a hardware environment that effectively mirrored the behavior of the final SoC. This setup allowed for rigorous testing of the RISC-V-based Rocket core, coupled with the HMM accelerator, under real-world conditions.

The SoC was fabricated using the 22-nm fully-depleted silicon-on-insulator (FD-SOI) process offered by GlobalFoundries™. Fig. 4.19 shows the die shot of the fabricated chip, and Fig. 4.20 highlights the primary hardware blocks including the RISC-V core, its associated cache memory, and two specialized accelerators, referred to here as AccelA and AccelB. These accelerators are tailored to accelerate

different stages of the basecalling process. The chip itself was wirebonded to a ball grid array (BGA) substrate and mounted in an elastomer socket to facilitate detailed testing and evaluation.

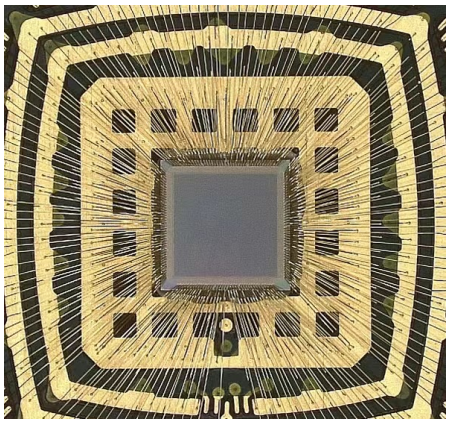

Figure 4.19: Die shot of the fabricated chip, showing the wire bonding connections and pad layout for the first SoC. The central region contains the core logic, surrounded by the pad ring for external I/O.

In terms of physical layout, the RISC-V core occupies an area of 0.355 mm^2, while AccelA and AccelB occupy areas of 0.395 mm^2 and 0.401 mm^2, respectively. For the RISC-V core, around 30% of the silicon area is dedicated to its logic and register blocks, with the remainder taken up by ICache and DCache, which are crucial for reducing memory access latency and ensuring efficient data throughput. The accelerators, on the other hand, show a more significant contribution of area dedicated to logic, accounting for approximately 45% of their total area. The rest of the area is mainly occupied by buffer memory, such as event and pointer buffers, which are vital for managing data flow and synchronization during the

acceleration of basecalling operations. These specialized buffers help to maintain the accelerators' efficiency by ensuring data is readily available for processing.

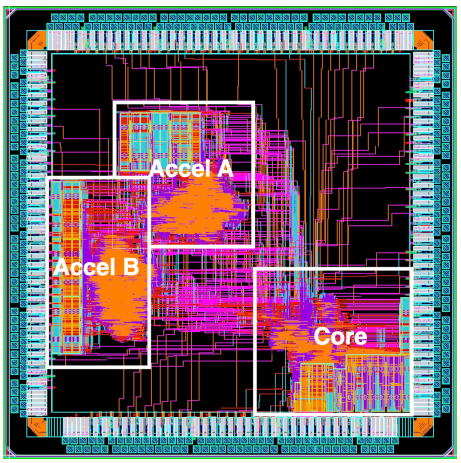

Figure 4.20: SoC architecture integrating two dedicated HMM BC accelerators: AccelA, specialized for parallel trellis construction, and AccelB, designed for both parallel trellis construction and pipelined Viterbi traceback.

4.7.3 ASIC Measurements

The performance of the SoC was compared against three additional platforms to highlight its scalability and efficiency: the Tensilica Xtensa LX6 microprocessor in the ESP32 microcontroller (240 MHz) for cost-effective applications, the ARM Cortex-A53 core in the Xilinx Zynq UltraScale+ MPSoC ZCU106 for high-end embedded use, and the Intel Xeon E5-2620 v3 (2.4 GHz, 12 cores) representing desktop-level computing. These platforms span three distinct Instruction Set Architectures (ISAs): RISC-V for the custom SoC, ARMv8 for the Cortex-A53, and x86 for the Intel Xeon processor. All tests were performed using default compiler

settings with their respective GNU toolchains: RISC-V toolchain v5.3.0, Xilinx-adapted ARM toolchain v11.2.0, and x86 toolchain v12.2.1. Additionally, the Cortex-A53 was tested with and without SIMD (Single Instruction, Multiple Data) enabled to showcase the performance gains from vectorized processing.

Fig. 4.21 illustrates the comparative performance, measured in basecalling speed (kilobases per second), as a function of clock frequency across the platforms. The SoC was evaluated under three workload partitioning configurations: core-only, core with AccelA, and core with AccelB. Results show that the core+AccelB configuration significantly outperformed the other SoC setups, achieving speed-ups of approximately 87× and 260× over core+AccelA and core-only, respectively. This improvement is attributed to AccelB's dedicated traceback block, enabling accelerated execution of critical functions within the DNA sequence detection program.

The comparative analysis also highlights the scalability of the SoC relative to other platforms. The ESP32 microcontroller, representing cost-effective embedded systems, demonstrated limited scalability due to its low clock speed and processing power. The ARM Cortex-A53, tested over a frequency range from 300 MHz to 1200 MHz, showed notable improvements with SIMD enabled, underscoring the benefits of vectorized processing. The Intel Xeon processor, operating at a fixed 2.4 GHz, served as a benchmark for desktop computing performance. The results demonstrate the versatility of the SoC platform, combining the flexibility of the RISC-V

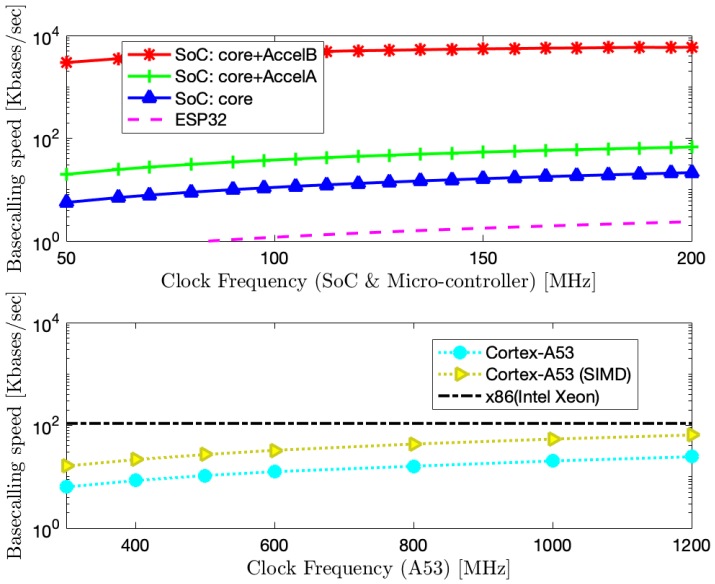

Figure 4.21: Performance comparison of the SoC platform with ESP32, ARM Cortex-A53, and Intel Xeon E5-2620 v3. The comparison is shown for different workload partitioning schemes on the SoC, as well as for different clock frequencies. The evaluation highlights the scalability of the SoC platform in relation to the other systems.

core with the efficiency of the HMM-based accelerator. By leveraging architectural enhancements such as AccelB, the SoC achieves competitive performance across a range of computing environments, from embedded to desktop-level applications.

Fig. 4.22 illustrates the energy efficiency of different SoC configurations, including the core alone and with specialized accelerators, across various clock frequencies. The top plot highlights the energy efficiency of the SoC configurations operating between 50 MHz and 200 MHz, while the bottom plot focuses on the Cortex-A53 processor's efficiency across frequencies from 400 MHz to 1200 MHz.

The SoC uses a nominal supply voltage of 0.8 V for its 22-nm circuits, but

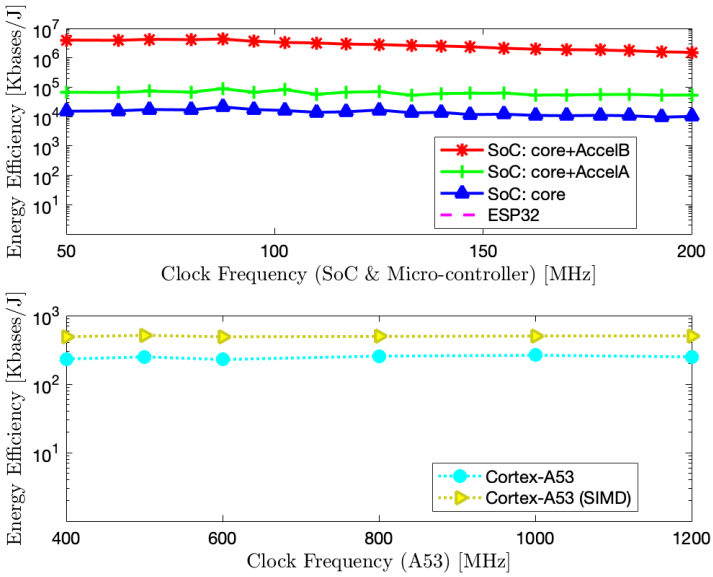

Figure 4.22: Energy efficiency of different configurations across varying clock frequencies.

can operate at reduced voltage levels, down to 0.49 V at a 50 MHz clock and 0.64 V at 200 MHz. Despite operating at these lower voltages, the configuration involving the core with AccelB achieves the highest energy efficiency, consistently ranging between 10^6 and 10^7 Kbases per Joule across all tested frequencies. The configuration of the core with AccelA also exhibits enhanced efficiency compared to the core-only setup, but it doesn't reach the level of the AccelB setup. In comparison, the RISC-V core operates with a significantly lower efficiency, staying within the 10^3 to 10^4 Kbases per Joule range, while the ESP32 reaches only 0.019 Kbases per Joule, making it difficult to represent effectively in the diagram and highlighting its limited efficiency.

Table 4.4: Performance comparison of different SoC configurations at 250 MHz.

Platform	HW job	SW job	Runtime (Cycles)	Number of logic cells	Power (mW)
Core	-	Trellis+Traceback	10511397	119884	19
Core+AccelA	Trellis	Traceback	3332256	276450	18.4
Core+AccelB	Trellis+Traceback	-	16612	282380	20.2

In the second plot, energy efficiency is compared for the Cortex-A53 processor, both in its standard form and with a SIMD enhancement. The data shows that the efficiency remains stable across clock frequencies from 400 MHz to 120 MHz, with the SIMD version providing a slight advantage, reaching approximately 10^3 Kbases per Joule.

In order to further test how the accelerators handle the HMM BC algorithm, we conducted experiments to evaluate the performance of the SoC in three different configurations: `Core`, `Core+AccelA`, and `Core+AccelB`. As shown in Tab. 4.4, the baseline `Core` configuration handles all computational tasks in software, resulting in moderate power consumption (19 mW) and resource usage (119,884 logic cells), with a runtime of 10,511,397 cycles.

Introducing hardware accelerators, as seen in `Core+AccelA` and `Core+AccelB`, allows specific tasks to be offloaded, which increases the utilization of logic cells, thus increasing the area and power consumption, but can reduce runtime significantly by optimizing task execution efficiency. The `Core+AccelA` configuration offloads the trellis computation to hardware, resulting in a runtime reduction of

68% reduction compared to the baseline and using an increase of additional 30% in logic cells, with a slight decrease in power consumption to 18.4 mW.

The `Core+AccelB` configuration fully offloads both trellis and traceback to hardware, achieving a drastic reduction in runtime (a 633× speedup compared to the baseline), with an increase in logic cell usage of additional 35% compared to the baseline and 2% compared to `Core+AccelA`, and an increase in power consumption of 6.3% compared to the baseline and 9.8% compared to `Core+AccelA`. These metrics highlight the trade-offs between improved processing speed and increased hardware resource usage.

Providing these quantitative metrics helps to better illustrate the trade-offs involved in each configuration. This analysis suggests that careful consideration is needed when deciding how much of the workload to move to hardware, balancing power efficiency, resource usage, and overall system performance for mobile DNA sequencing tasks. These insights will guide us in designing the next generation of chips, where we can further optimize task partitioning and accelerator integration to achieve better performance and power efficiency.

4.8 Chapter Summary

This chapter detailed the design, hardware implementation, and performance evaluation of the first SoC developed for HMM-based DNA basecalling. The chapter

began by outlining the theoretical foundation of the algorithm and its suitability for basecalling tasks. Hardware adaptations necessary to optimize the algorithm for the constraints of a specialized SoC were discussed, emphasizing the integration of a RISC-V core with a dedicated HMM accelerator.

The architectural overview provided insights into the SoC's components, including its memory hierarchy, integration with the basecalling accelerator, and support for parallel processing. The design trade-offs between computational efficiency, area usage, and power consumption were presented in detail, including the addition of a traceback block for enhanced performance.

Evaluation results from both FPGA prototyping and ASIC testing were highlighted, showcasing significant performance improvements in speed and energy efficiency compared to software-only implementations and alternative platforms. Comparisons across different configurations—core-only, core with AccelA, and core with AccelB—illustrated the impact of hardware acceleration on runtime and scalability. Additionally, the SoC's competitive performance relative to commercial platforms such as the ARM Cortex-A53 and Intel Xeon processors demonstrated its capability for real-time DNA sequencing.

This work extends the field by demonstrating the scalability and practicality of accelerator-based SoC designs, providing a blueprint for future innovations in portable and efficient DNA sequencing systems.

5 Second SoC: ML Basecaller Accelerator and ED Accelerator

This chapter explores the design and enhancements implemented in the second SoC, which transitions from using HMMs to neural network-based models for basecalling. The development focuses on identifying an optimal architecture to achieve improved performance, with a systolic array-based accelerator emerging as a key solution due to its advantages in efficiently handling neural network computations. The design leverages an open-source accelerator, chosen for its configurability and seamless integration into the RISC-V architecture as a coprocessor.

To expand the functionality of the SoC, it also incorporates an additional accelerator specifically designed to accelerate Edit Distance (ED) calculations, which are essential for sequence comparison tasks in DNA sequencing workflows.

The section 5.1, 5.2, 5.4, 5.3 and 5.7 are submitted to [87].

5.1 Algorithm Development

Building on the insights from analyzing state-of-the-art basecaller architectures in Chapter 2, a new software basecalling system was developed for the second SoC, which incorporates neural network acceleration. This system is designed to be highly adaptable, enabling the integration and evaluation of various neural network models using real sequencing data. The selection of the most suitable model for deployment was guided by assessments of its architectural simplicity, computational efficiency, memory usage, and performance on practical sequencing tasks.

The chosen model balances minimal complexity with high accuracy, ensuring efficient execution within the constraints of the hardware. Its compact size and reduced number of operations enable faster processing while conserving energy, making it ideal for the resource-limited environment of the SoC. The model's performance was also assessed in terms of its compatibility with the systolic array-based accelerator, which optimizes matrix operations central to the neural network's functionality.

5.1.1 Basecalling Software and Neural Network Design

Today, many of the classification problems fall into the deep learning domain, and so our basecaller design assumes this to be the case as well. The question of which

type of deep learning network to employ, the "core architecture", will be addressed in 5.1.2. Presently, however, we consider the broader context of the basecalling system architecture in which such models, be they recurrent, convolutional, or attention-based, may be embedded in a modular fashion.

The proposed basecalling software system architecture is illustrated in Fig. 5.1. It is designed for deployment on an SoC for DNA sequencing applications and aims to serve as a common host for different types of neural network model. The modular nature of this architecture also allows flexibility in evaluating various models, ultimately selecting the one that optimally meets the performance and power constraints of the SoC.

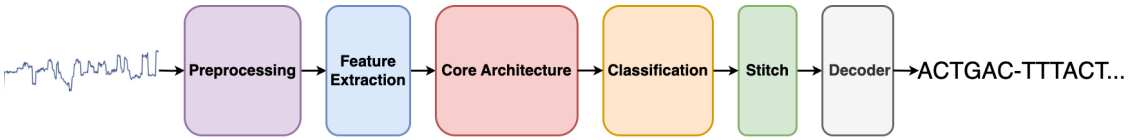

Figure 5.1: High-level architecture of the proposed software basecalling system.

The system begins with the preprocessing stage, where the raw electrical signals generated by the nanopore sequencer are segmented into smaller arrays of normalized raw samples. The segmentation refers to the practical preference of breaking up otherwise large input time series into smaller subsets appropriate for handling by a given machine learning network. In long-read nanopore sequencers, a single DNA strand could easily be represented by 100s of thousands of raw mea-

surement samples. Since signal dependence across the entirety of such a run is unlikely (dependencies across roughly 300 samples are much more common), computing efficiency is better accommodated by segmenting such inputs into smaller and overlapping sequences. The normalization function of the preprocessing stage is necessary to address variations such as noise and inconsistencies across different sequencing conditions, providing a standardized input for downstream processing.

The preprocessed signal then passes through an initial feature extraction stage composed of Conv1D, Batch Normalization (BatchNorm), and ReLU. Conv1D sweeps across each time-series input and outputs a corresponding set of sequences across its output channels; thus, a 1D input is effectively converted into a multi-dimensional representation. The BatchNorm helps maintain more consistent distributions to help stabilize the training process and ReLU naturally enables learning across multiple layers. These computations provide an effective starting point for capturing important features while keeping the computational complexity manageable. The feature extraction stage presents a possible target for execution of the heterogeneous computing elements of the SoC.

The signal then enters the core architecture block, which forms the heart of the basecaller. As elaborated in § 5.1.2, this stage can be configured with various neural network models, known options include LSTM, GRU, transformer encoder or Conv1D-based approaches. The core architecture is applied to capture temporal

relationships in the sequence data, which is crucial for basecalling accuracy. The computational load presented by this stage is an obvious target for specialized acceleration hardware on the SoC.

After processing through the core architecture, the data moves to the classification stage that computes a probability distribution, via logSoftmax, on core architecture's output state vectors. This reflects the common practice of basecallers to represent their outputs in terms of a vocabulary of stays, bases, and/or k-mer states (the representation of the groups composed of k-mer bases). This stage also includes the option of including an intervening feedforward or convolutional layer as a further means to create alternate projections of the state outputs onto their probability estimates.

The stitch stage then reconstructs the sequence first segmented by the preprocessing stage. This is done while ensuring consistency in base predictions, especially when the model processes the input in batches.

Finally, the decoder generates the nucleotide sequence by converting the fixed-length features into a variable-length sequence. This is a common part in sequence labeling or sequence translation approaches of which basecalling is a type. Options here include a beam search algorithm, conditional random field approaches, or connectionist temporal classification decoder. The question of how to handle this in the SoC (heterogeneous blocks, in the core accelerators, as a separate accelerator)

remains an open question in the literature.

Our partitioning of the basecalling system into dedicated functional stages strongly suggests its amenability to a multicore SoC design composed of general-purpose processors and specialized accelerators. By utilizing all available resources, the SoC can assign different stages of the pipeline to the most suitable components: preprocessing and operations like activations or decoding can be handled by the general-purpose processor cores, while computationally intensive tasks, such as matrix multiplications in the model, are accelerated by dedicated hardware. A balanced allocation optimizes performance and power efficiency, enabling the system to effectively meet the real-time requirements of DNA sequencing.

5.1.2 Core Architectures

Various neural network approaches have been applied to the task of basecalling nanopore measurements. In our proposed system of Fig. 5.1, these approaches are situated in the core architecture block. Common styles include recurrent neural networks (RNNs) [34, 35], convolutional neural networks (CNNs) [36, 37], and transformer methods [14, 38, 39]. The choice of which model to use depends on its ability to achieve an optimal balance between accuracy, efficiency, and hardware compatibility. All of these being of course influenced by the intended use.

To determine the most suitable model for the SoC, we designed and evaluated

Table 5.1: Core Architecture Comparison Overview. Note: MACs/Raw considers the entire model.

Architecture	**GRU**	**LSTM**	**Conv1D**	**PE + TE**
Layers	2 (Bi)	2 (Bi)	4	2
Parameters (K)	438	578	446	611
MACs/Raw (K)	417.8	557.0	445.9	795.5
Eval Accuracy (%)	89.5	90.4	85.5	80.9
Training time/epoch (min)	12.5	12.5	11.0	20.3

several neural network architectures. Table 5.1 summarizes the candidates, which are chosen; these include RNNs based on GRU and LSTM nodes, Conv1D networks, and transformer encoder (TE) [88] with positional embedding (PE) [89] models. The choices are representative of methods currently popular in the literature, but the implementation details of these networks (i.e., their layer and parameter count) was strongly influenced by the hardware constraints our mobile SoC will likely face. In particular, anticipating that low cost SoCs might scarcely accommodate 1-MiB of data memory requirement per chip, hence, we constrained our model sizes to roughly 500K parameters. Although large basecalling models well exceed this, many basecalling proposals fall inside our considered parameter-size range [45].

The decision to constrain the parameter size to approximately 500K was based on the need to balance computational load with the power and memory limitations of an embedded SoC. To better understand this constraint, consider the following factors:

- **On-Chip Memory Limitations:** The SoC in question must support real-time basecalling, but embedded SoCs typically have limited on-chip SRAM available, often in the range of 256KB to 1MB. A model with millions of parameters would require significant memory for both storage and execution. Each parameter, when using 8-bit quantization, requires 1 byte, and with 500K parameters, the total model size is around 500KB. This size was selected to ensure that the entire model (or a significant portion of it) could be stored on-chip, avoiding the need for frequent off-chip DRAM access, which incurs both latency and higher energy costs.

- **Energy Efficiency Considerations:** For embedded systems operating below 1 Watt, energy consumption is a crucial factor. Off-chip memory access is known to be a significant source of energy expenditure. By limiting the model size to 500K parameters, the aim was to minimize the number of memory transactions needed during inference, thereby reducing the overall energy consumption. Larger models would not only demand more power for computation but would also require additional off-chip data transfers, which could exceed our energy budget.

- **Computational Complexity:** The MAC (Multiply-Accumulate) operations required for each layer increase with the number of parameters, impacting

both execution time and energy consumption. A model with 500K parameters results in an overall manageable number of MAC operations, given the capabilities of our SoC's accelerator. For instance, a systolic array with 1,024 MAC units could efficiently manage the workload associated with 500K parameters, allowing for effective throughput in real-time sequencing scenarios while maintaining energy efficiency.

- **Memory Bandwidth and Latency**: Real-time sequencing requires low-latency computation to keep up with the data flow from the sequencer. With a parameter size of 500K, the model can be efficiently divided into smaller blocks that fit into the cache, allowing for streamlined processing. This helps reduce latency by avoiding the bottlenecks associated with memory bandwidth limits and excessive data transfer delays.

Therefore, the choice of approximately 500K parameters reflects the need to balance model complexity with the limited power and memory capacity of an embedded SoC. By keeping the model size at this level, it becomes feasible to achieve real-time sequencing performance without incurring prohibitive energy and memory access costs. This parameter size allows the model to run within the power envelope of less than 1 Watt, while still providing sufficient accuracy for practical DNA basecalling tasks in an embedded context.

When comparing these architectures, several important insights emerge beyond just accuracy and computational load. The GRU and LSTM models demonstrate superior accuracy, primarily due to their ability to capture relevant dependencies in the sequential data produced during basecalling. However, this strength comes at the cost of increased complexity. Specifically, GRU and LSTM computations involve multiple gate operations, which require sequential data dependencies that are difficult to parallelize and these dependencies significantly hinder their ability to meet the stringent requirements of real-time sequencing. While these models involve vector multiplications, which can be efficiently managed by the accelerator, they also require additional non-linear activations such as the sigmoid and tanh functions. These will increase the complexity of the SoC, adding to design and resource costs. Offloading these activations to the processor seems a possible solution, but this will introduce more memory transmission bottlenecks since transferring data back and forth to DRAM for processing leads to considerable delays. Consequently, the necessity for these non-linear activations and the memory bandwidth bottlenecks limit the suitability of GRU and LSTM models for a hardware-optimized SoC that targets efficient real-time basecalling.

The TE model, despite its success in other sequence-based applications, did not meet expectations in the basecalling context we studied. Transformers are well-known for capturing global dependencies and excelling in parallelism. However, the

observed accuracy was lower compared to GRU/LSTM and Conv1D models, potentially due to the high sensitivity of transformers to hyperparameters or the need for additional domain-specific tuning. This observation also aligns with findings in [90] which indicate that RNN-based models (such as LSTM) outperform transformer models in all evaluated metrics for basecalling. RNNs appear to be better suited for capturing time-axis relationships between sequential samples, whereas the attention mechanism in transformers is seemingly less effective for this specific type of data. This suggests that, despite their popularity, current transformer incarnations may be less capable of handling the temporal dependencies required for accurate basecalling compared to RNNs.

Balancing computational efficiency and model accuracy, Conv1D stands out as a candidate with both potential advantages and trade-offs. Conv1D offers a balanced approach between simplicity and efficiency. While it achieves an evaluation accuracy of 85.5%, which is lower compared to GRU and LSTM, it remains a viable option for real-time applications due to its architectural simplicity and parallelization. In many scenarios, achieving extremely high accuracy is not essential, such as in virus/pathogen detection, species classification, and even in pre-filtering [91], which aids in raw data reduction before higher-accuracy basecalling. These tasks can tolerate moderate accuracy (e.g., 5% accuracy loss) in favor of reduced complexity and increased processing speed, making Conv1D an appealing choice for such cases. Ad-

ditionally, Conv1D may not fully capture long-range dependencies as effectively as GRU or LSTM, but it compensates by being more hardware-friendly, especially for accelerators like systolic arrays, which are well-suited to handling convolution operations. With 445.9K MAC operations and 446K parameters, Conv1D has a moderate computational footprint, allowing it to be processed efficiently without the additional overhead of gate functions (e.g., Hadamard product) and non-linear activations, such as those used in GRU and LSTM. For example, if implemented in an SoC with a 1,024 MAC unit systolic array running at 1 GHz, Conv1D's architecture could theoretically achieve significant throughput. Given the need for 445.9K MACs per raw input, the estimated processing capability would be approximately 230 kbases/s, fulfilling the real-time throughput goal mentioned earlier, making it a promising candidate for real-time DNA sequencing applications where both power efficiency and computational speed are paramount.

This balance between computational simplicity, efficient memory access, and hardware compatibility makes Conv1D a feasible choice for basecalling tasks. It ensures that the system can take full advantage of specialized hardware accelerators without being constrained by the complexity introduced by sequential architectures like GRU and LSTM. Although Conv1D does not achieve the highest accuracy among the evaluated models, its compatibility with hardware acceleration and its efficient resource utilization make it a strong candidate for real-time, power-efficient

basecalling on a custom SoC.

5.1.3 Basecaller comparison

Building upon this decision, we managed to run the proposed basecalling model architecture on an x86 platform to evaluate its accuracy and speed before applying it to the SoC. The model was implemented using PyTorch and deployed on a GPU using PyTorch CUDA. The platform contains an Intel™ Xeon W2123 (3.6 GHz) CPU and a NVIDIA™ Titan X RTX GPU with 24GB GDDR and 64GB DDR3 system memory.

Then we compared our proposed basecaller with some state-of-the-art basecallers on the test platform. As presented in Table 5.2. The proposed basecaller was initially tested on a general GPU platform to verify its accuracy before evaluating it on the SoC. While our basecaller achieves an accuracy of 84.7%, which is lower compared to Bonito's 92.7%, it offers a favorable balance between model size, computational efficiency, and scalability. Bonito achieves high accuracy but requires 733K parameters and significant GPU resources, making it less suitable for resource-constrained environments. DeepNano-coral, which leverages TPU acceleration, also features a larger parameter count of 2,386K, resulting in a model size of 4.6 MB, and achieves an accuracy of 90.1%. However, its larger parameter size and the requirement of TPU hardware present challenges in terms of scalability for

embedded applications. Scrappie, designed primarily for CPU use, has a smaller parameter count of 387K and a model size of 5.9 MB in text format, but its accuracy of 80.2% is significantly lower, making it less ideal for applications demanding higher performance.

Table 5.2: Comparison of the proposed basecaller with existing state-of-the-art basecallers. "B" denotes a binary file, and "T" denotes a text file.

Model Name	**Bonito**	**Scrappie**	**DeepNano-Coral**	**Proposed Model**
Test Platform	GPU	CPU	TPU	GPU
Parameters (K)	733	387	2386	446
Model Size (MB)	2.9 (B)	5.9 (T)	4.6 (B)	1.8 (T)
Speed (kilo-raws/s)	220	41	2190	250
Accuracy (%)	92.7	80.2	90.1 [45]	84.5

In contrast, our proposed basecaller, with 446K parameters and a model size of only 1.8 MB, strikes an optimal balance between accuracy, model size, and computational requirements. Despite not achieving the highest accuracy, it maintains a compact architecture and computational efficiency, making it highly suitable for deployment on platforms with stringent power and computational limitations, such as FPGAs and ASICs. This balance makes it a viable option for achieving real-time sequencing in an embedded system environment.

5.1.4 Sequence Comparison Algorithm

Sequence comparison allows us to quantify and localize the differences between sequences, such as insertions, deletions, and substitutions. Two prominent algorithms were evaluated for this purpose: the wavefront alignment (WFA) [92] and the edit distance [93] algorithms. Both of these methods have distinct characteristics and serve different roles in DNA data analysis, aspects which must be carefully considered when choosing an algorithm for implementation on our NanoSoC with limited computational and memory resources.

WFA is a relatively new approach that employs heuristics for sequence alignment. These exploits are particularly effective when sequences are highly similar. By dynamically extending the "wavefront" to cover only regions of interest within the alignment matrix, it significantly reduces computational demands. In the best-case scenario, where sequences have minimal differences, wavefront alignment achieves a computational complexity of $O(n \cdot s)$, where s is the edit distance between the strings being compared and n is the length of one of the query sequences. However, in the worst case computational complexity can increase to $O(m \cdot n)$, where m and n are the lengths of the two sequences (query and reference). The memory complexity in these cases can also grow from $O(s^2)$ to $O(n^2)$, as the wavefront must expand to cover all possible differences, increasing the resource

requirements considerably in complex alignments.

Alas, the case of dealing with many poorly matched sequences is exactly the initial scenario we envision for embedded SoCs in the mobile sequencing context. Specifically, for tasks like pathogen detection, a computing system will be faced with the need to sift through many divergent sequences in search for pathogen templates. Also, as basecalling accuracy is traded for reduced power, the clash with WFA's heuristics will diminsh its advantage.

In contrast, the ED algorithm provides a simpler solution for sequence comparison. The standard implementation of ED uses a dynamic programming approach that requires $O(m \cdot n)$ space to build an alignment matrix for sequences of lengths m and n, but use-case needs can allow room for optimizations. For example, in the ED implementation listed in Algorithm 2, we reduce memory requirements to $O(n + 1)$, by utilizing a one-dimensional array (dp) to store only the current row of the alignment matrix while using an additional variable ($prev$) to hold the previous value needed for in-place updates. This optimization eliminates the need for a full two-dimensional matrix, significantly reduces memory usage, and avoids costly and power-intensive external DRAM accesses. Furthermore, the inclusion of the `offset` parameter is crucial in restricting the range of sequence alignment, particularly in cases where local or banded alignments are necessary. The offset limits the region in which the algorithm computes the edit distance, focusing on a diagonal

band around the alignment path. This allows the algorithm to handle cases where the sequences are approximately aligned, avoiding unnecessary computations over irrelevant portions of the sequences.

Algorithm 2 Edit Distance with Dynamic Programming and Memory Optimization

Require: Two strings $str1$ and $str2$, their lengths m and n, and an offset $offset$
Ensure: Edit distance between $str1$ and $str2$

```
1: function editDistDPMem(str1, m, str2, n, offset)
2:     Declare integer array dp[n + 1]
3:     for i = 0 to n do
4:         dp[i] ← i
5:     end for
6:     for i = 1 to m do
7:         prev ← dp[0]
8:         dp[0] ← i
9:         for j = offset + 1 to n do
10:            temp ← dp[j − offset]
11:            if str1[i − 1] = str2[j − 1] then
12:                dp[j − offset] ← prev
13:            else
14:                dp[j − offset] ← 1 + min(dp[j − 1 − offset], dp[j − offset], prev)
15:            end if
16:            prev ← temp
17:        end for
18:    end for
19:    return dp[n − offset]
20: end function
```

To further optimize the algorithm for hardware implementation, we encode the sequences for efficient memory transfer. Specifically, every 32 bases are packed into a single 64-bit integer, where each base (A, C, G, T) is represented using 2 bits (e.g., 00 for A, 01 for C, 10 for G, and 11 for T). If the sequence length is not

divisible by 32, padding with zeros is applied to fill out the 64-bit integer. By doing so, we reduce the overall size of the sequence data, which significantly decreases the memory bandwidth required to transfer sequences to and from memory during edit distance calculations.

By focusing on the ED algorithm, we ensured that our SoC design has a chance to benefit from a consistent computational demand and reduced memory usage. These aims are aligned with the goals of low power consumption and real-time operation. These characteristics also make ED a more promising choice for initial SoC implementations, facilitating efficient DNA sequencing while minimizing the complexities associated with more dynamic and resource-intensive algorithms. The simplicity, reduced memory footprint, and predictable workload of ED make it well-suited for hardware acceleration and integration into a power-efficient, real-time sequencing SoC.

5.2 SoC Architecture Overview

The proposed SoC architecture is shown in Fig. 5.2. It features two RISC-V tiles (cores) containing an in-order core with a split L1 cache. Each tile is also tied to a dedicated accelerator, one for basecalling, referred to as the Basecalling (BC Accelerator) and one for sequence alignment, referred to as the Edit Distance (ED Accelerator). The accelerators interface with the cores via RoCC (Rocket Custom

Coprocessor) [94] instructions, enabling efficient offloading of computational tasks. Each accelerator uses command (`cmd`) and response (`resp`) signals for control and status exchanges. Both tiles connect to a unified L2 cache which manages memory coherence and access to external DDR memory.

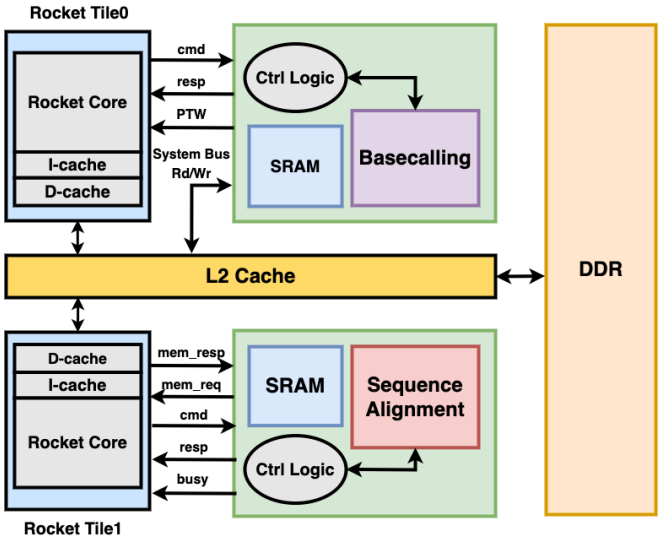

Figure 5.2: The Proposed Second SoC Architecture.

This design balances performance and power efficiency. By including two cores, the architecture supports parallel task execution, separating basecalling and comparison workloads between the accelerators. This dual-core setup provides efficient multitasking and flexibility for users to manage and allocate computational resources based on specific sequencing tasks, which is especially useful in a real-time DNA sequencing context. It also aligns with architectural philosophies found in embedded system [95, 96].

Table 5.3 summarizes the design parameters of the SoC. The architecture includes a split L1 cache for each core, with 16 KiB instruction and 16 KiB data caches, both configured as 4-way associative. The unified L2 cache, shared by both cores, is 256 KiB and configured as 8-way associative to improve data access efficiency and reduce bottlenecks. The SoC also includes specialized branch prediction units, such as the BHT and BTB, which help maintain efficient instruction flow. On-chip memory exclusively for accelerators is 352 KiB, providing the necessary resources for both basecalling and edit distance computations.

Table 5.3: The Second SoC Processor and Accelerator Parameters

Design Parameter	Value
RISC-V ISA	RISCV64G
Number of Tiles	2
Data width	64 b
Instruction Cache (4-way)	32 KiB
Data Cache (4-way)	32 KiB
L2 Cache (8-way)	256 KiB
Translation lookaside buffer (TLB) entries	8
Branch history table (BHT) entries (2-b)	4096
Branch target buffer (BTB) entries	62
Return address stack (RAS) entries	2
Accelerator on-chip memory	352 KiB

5.3 BC Acceleration Customizations

The BC accelerator, as depicted in Fig. 5.3, is designed to execute matrix multiplications central to modern basecalling algorithms. At its core, the accelerator

employs our customized version of the Gemmini systolic array architecture [97] to achieve high throughput and low latency for matrix multiplication. By processing large blocks of data in parallel, the systolic array is well-suited to the repetitive nature of convolutional operations and fully connected layers in deep learning models used for basecalling.

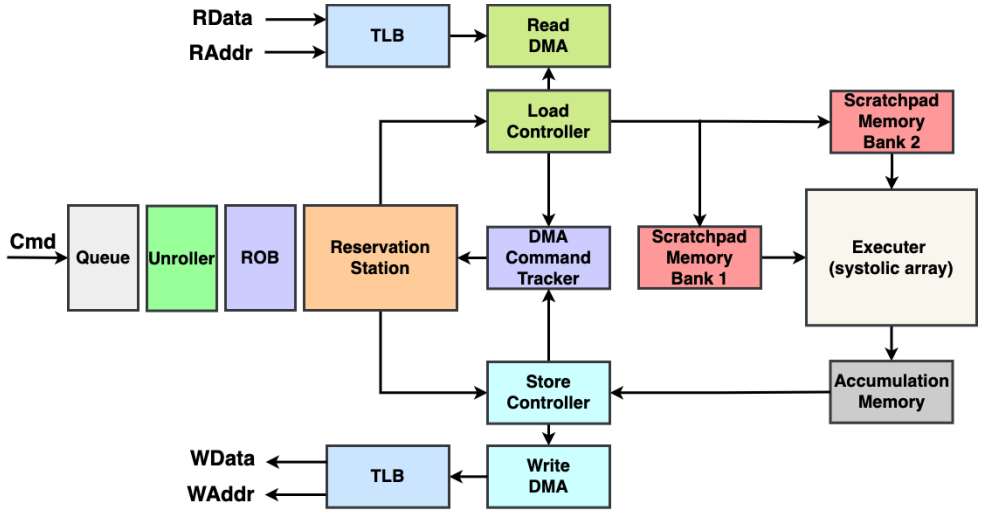

Figure 5.3: A systolic array-based basecaller accelerator block diagram.

In designing this accelerator, we chose a 32×32 systolic array configuration of processing elements (PEs) to strike an optimal balance between computational performance and hardware area. The selected array size provides 1,024 MAC units, which meets the real-time processing requirements for DNA sequencing workloads given realistic clock speeds, as demonstrated in our earlier analysis in § 5.1.2.

The system starts with the `Queue`, which receives high-level commands from its RISC-V core. These commands are then sent to the `Unroller` which decomposes

each matrix-level operation into smaller atomic instructions. These instructions dictate fundamental tasks such as data movements between DRAM and scratchpad, between scratchpad and systolic array, and also between systolic array and accumulation memory.

The `Reorder Buffer` (ROB) in Gemmini plays a crucial role in maintaining the correct execution sequence, particularly within its decoupled access-execute architecture. In this design, the `Load Controller`, `Store Controller`, and `Executer` operate independently and may process instructions concurrently and out-of-order with respect to each other. This architecture enhances performance by enabling parallel execution of tasks that are not interdependent. To ensure correctness, the ROB detects hazards between instructions belonging to different controllers. It tracks dependencies and ensures that instructions are only issued to their respective controllers when all inter-controller dependencies have been resolved. By doing so, the ROB prevents conflicts and maintains the logical sequence of execution across the different components of the architecture.

The `Reservation Station` issues these operations based on resource availability, making sure that the compute and memory units are utilized without introducing bottlenecks. The `TLB` translates virtual addresses to physical addresses, enabling the `Direct Memory Access` (DMA) module to move data efficiently between external DRAM and internal memory structures like Scratchpad Memory.

The DMA reduces the burden on the main processor by handling these memory transactions independently, facilitating uninterrupted matrix operations. The `DMA Command Tracker` keeps data transfers in sync, preventing conflicts and ensuring orderly execution. Once transferred, data is stored temporarily in scratchpad Memory which provides a controlled and low-latency storage area for frequently accessed data such as weights and input matrices. Data movement between Scratchpad Memory and the systolic array is managed by the `Load Controller` and `Store Controller`.

`Accumulation Memory` stores partial results during multiply-accumulate operations, minimizing the need for frequent access to external memory, which enhances both speed and energy efficiency. To further optimize performance, the Accumulation Memory integrates adders directly within the memory array. These adders, sized to match the systolic array dimensions (DIM), enable efficient in-memory accumulation of partial sums. The Executer manages the core computational unit, the systolic array, which is represented in Fig. 5.3. This block contains numerous Multiply-Accumulate (MAC) units that execute parallel operations in a pipelined manner. The `Executer` directs the flow of data through the MAC units, fully exploiting the available parallelism to achieve high computational throughput, which is essential for deep learning workloads like basecalling.

The detailed architecture of the `Executer` block is illustrated in Fig. 5.4. The

systolic array within the `Executer` is the core computational engine, designed for high-throughput neural network operations. The array operates by processing data through a grid of interconnected processing elements (PEs), each capable of performing a multiply-accumulate (MAC) operation. Inputs to the systolic array are fed from two main sources: Scratchpad Memory Bank1 supplies the input activations, while Scratchpad Memory Bank 2 provides the weights.

This dual-bank setup is critical, as it allows simultaneous access to activations and weights, ensuring that the MAC units are fully utilized. Each PE in the systolic array receives a stream of activations and weights. The PEs are arranged in a grid where activations flow horizontally, weights flow vertically, and the results propagate diagonally. This data flow is staggered across multiple cycles, which introduces specific cycle latencies for different PEs. These latencies arise because data must traverse through intermediate PEs before reaching the final computation points, ensuring that all partial sums are processed sequentially and correctly. To implement this staggered data flow, registers are embedded within each PE to hold the intermediate values of activations, weights, and partial sums. These registers synchronize the data flow, ensuring that the inputs to each PE are available at the correct cycle. By pipelining the operations within the array, the systolic architecture achieves a high level of throughput while minimizing idle time for each PE.

The partial sums are accumulated as the data flows through the array, allow-

ing the systolic array to compute the result of matrix multiplications efficiently. This architecture minimizes data movement and leverages the regularity of neural network computations to maximize throughput. At the end of the operation, the computed partial sums are passed to the Accumulation Memory, where they are further processed or stored temporarily. This tightly integrated design ensures that the `Executer` block can handle the demanding computational requirements of neural network models used in basecalling with high efficiency and low latency.

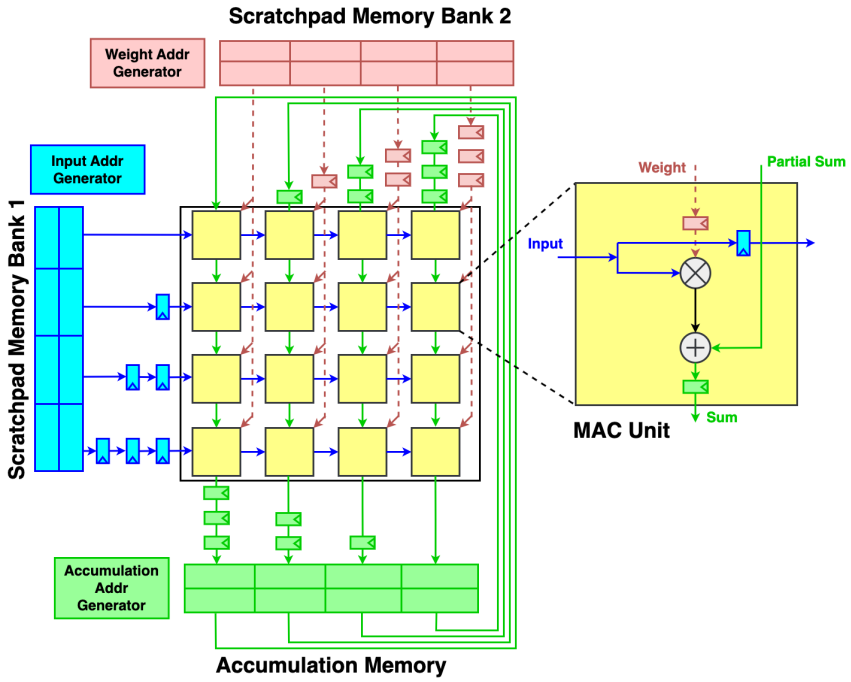

Figure 5.4: Detailed diagram of `Executer`, showing the 4×4 systolic array configuration, including input and weight address generators, MAC units, and accumulation memory components.

The flow of data through the systolic array is carefully orchestrated to maximize efficiency. The `Address Generator` and `Weight Address Generator` ensure that

the correct data is fetched and sent to the MAC units at each clock cycle. Data is streamed horizontally and vertically across the array, allowing each MAC unit to contribute to multiple calculations in a pipelined manner, thereby improving the overall throughput. Partial sums produced by the MAC units are then stored in the `Accumulation Memory`, which acts as an intermediate storage buffer before final results are passed back to the main system memory. The `Accumulation Address Generator` coordinates the writing of these partial results, ensuring that data is stored in an organized manner to facilitate subsequent processing steps.

5.4 ED Acceleration Design

The ED accelerator for sequence comparison is depicted in Fig. 5.5. As with the BC accelerator, its communication also employs a RoCC interface. In this case, to exchange command and response messages with its tile's core and data with its tile's L1 cache. The system uses Query SRAM and Reference SRAM to store the query and reference sequences, respectively. These memory blocks provide quick access and efficient storage, facilitating the rapid comparison of sequences.

The comparison is performed by distributing segments of the query and reference sequences to multiple Edit Distance units through a `Distributer` in Fig. 5.5. This parallel processing approach enhances efficiency by allowing simultaneous comparison of different sequence segments. Each ED unit, labeled `ED 1`, `ED 2`, `ED 3`, and `ED`

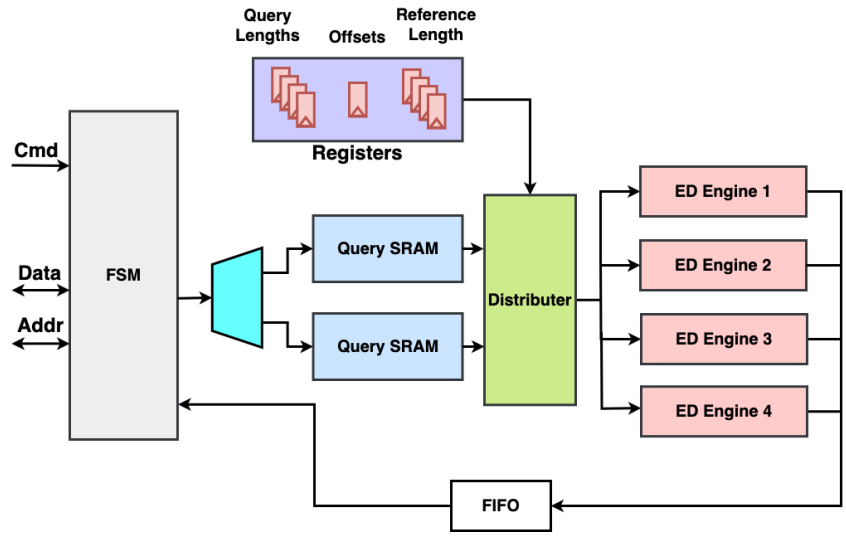

Figure 5.5: The sequence comparison accelerator block diagram.

4, computes the edit distances. The `Distributer` organizes the sequences into sets, specifically Set 1, Set 2, Set 3, and Set 4, where each set represents a combination of query and reference segments. These sets are then sent to the corresponding ED units.

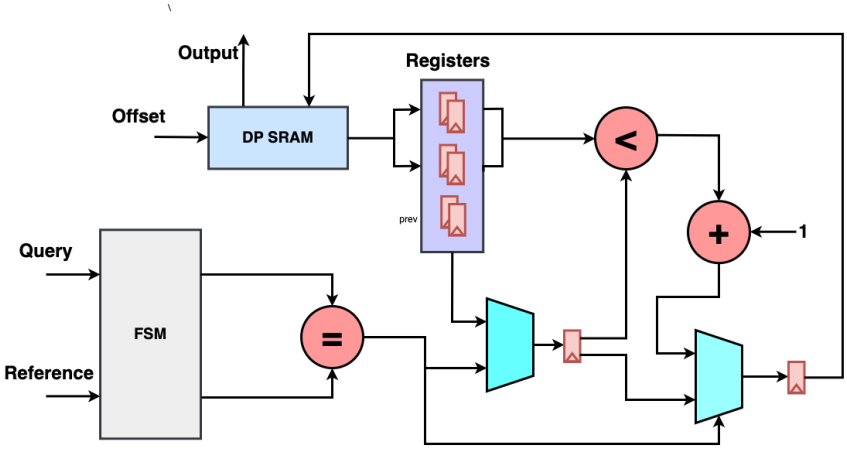

Figure 5.6: The sequence comparison accelerator block diagram.

In Figure 5.6, each ED cell receives a specific combination of query and reference segments. The `FSM` (Finite State Machine) controls the flow of operations, managing the sequence of actions required for comparison. The DP SRAM stores intermediate values during the processing of sequences. Registers hold lengths and offsets of the sequences, ensuring correct data segment processing. The `Multiplexer` (Mux) highlighted in blue aids in selecting between different data sources based on control signals from the FSM. By leveraging the RoCC interface and the described architectural components, the design achieves efficient data processing and interaction with the Rocket's L1 cache. This integration ensures high-performance execution of basecalling operations. The comparison process benefits from parallel processing, reducing the time required to accurately match query sequences to reference sequences.

The FSM coordinates the flow by controlling read and write operations to the query SRAM and the reference SRAM. For each position in the query sequence, the FSM sets the `prev` variable to the initial value (the first value of the SRAM) and updates the address of the current index to the current position index. When the comparison is complete, the score of the current indexed position stored in the SRAM is updated. This process continues for each segment in parallel across all ED units. Additionally, the FSM manages the sequence of actions required for comparison, while the `Dual Port (DP) SRAM` stores intermediate values during the

processing of sequences.

5.5 Control and Communication

The integration of the BC accelerator into the RISC-V architecture involves close interaction between the accelerator and the RISC-V processor through specialized interfaces (shown in Fig. 5.2) and instruction sets (shown in Fig. 4.8). Similar to the first chip, which featured the HMM-based basecalling accelerator, the accelerator in the second chip also utilizes the RoCC interface, and both BC and ED accelerators have unique `OPCODE` in the ISA so that the core can identify each. In our SoC, the BC's `OPCODE` is **3**, and the ED's is **0**.

With the inclusion of DMA functionality, the BC accelerator can directly access memory through the L2 cache via TileLink [98], a system bus within the SoC. This capability enables efficient handling of the large data streams required for basecalling. Conversely, the ED accelerator primarily uses the L1 cache. This design choice is motivated by the relatively small size of DNA sequence data, which can be represented compactly as 64-bit integers, making L1 cache sufficient for its storage. Moreover, after the BC accelerator completes its operations, the processed data resides in the L1 cache rather than the L2 cache. This is because the final steps of the basecalling process are performed by the RISC-V core itself, ensuring efficient access to the data directly from the L1 cache without incurring additional

latency from accessing L2.

The use of the RoCC interface by both accelerators enables efficient communication with the RISC-V core, allowing parallel operation of the BC and ED accelerators. This parallelism minimizes data transfer latency and significantly improves overall system throughput. By integrating these specialized accelerators, the SoC enhances its computational capabilities, enabling high-performance execution of bioinformatics tasks.

5.5.1 BC Accelerator Instructions

The BC accelerator is equipped with a set of specialized instructions designed to enable seamless interaction with the RISC-V core. As the second SoC employs the same protocol as the first SoC, the instruction format remains consistent (Fig. 4.8). An overview of the key instructions available for the BC accelerator is provided below:

- `mvin rs1 rs2` (`FUNC: 2`)

  moves data from DRAM to the internal memory `scratchpad` or `accumulation`. The `rs1` register contains the DRAM virtual memory address, while `rs2` register holds three components: the lowest 32 bits represent the address in the scratchpad memory (with MSB as 0) or in the accumulator memory (with MSB as 1), and the highest 32 bits contain row (highest 16 bits) and column

information of the destination.

- `mvout rs1 rs2` (`FUNC: 3`)

  moves data from the internal memory to DRAM. The `rs1` register shoul contains the virtual address of the DRAM. The `rs2` register contains the internal memory address in the lowest 32 bits, with the row location represented in the highest 16 bits and the column occupying the next 16 bits of `rs2`.

- `config_ld rs1 rs2` (`FUNC: 0`, `rs1[1:0]=01`)

  sets the stride for the `mvin` command. `rs1` specifies the stride for internal memory, while `rs2` sets the DRAM stride in bytes. This is particularly useful when the strides differ between DRAM and internal memory.

- `config_st rs1 rs2` (`FUNC: 0`, `rs1[1:0]=10`)

  sets the stride of the `mvout` command. This is useful when the strides are different in bytes between the dram and internal memory.

- `config_ex rs1 rs2` (`FUNC: 0`, `rs1[1:0]=00`)

  configures the multiplication command, including settings such as dataflow, activation in `rs1`. The `rs2` is used to set the number of bits for shifting to convert the partial sum from int32 back to int8.

- `preload rs1 rs2` (`FUNC: 6`)

  preloads the Matrix `B` into systolic array and sets the output for Matrix `C`.

`rs1` is the address of the Matrix `B` in the scratchpad, while `rs2` contains the address of the output Matrix `C` in the accumulation memory.

- `compute_preloaded rs1 rs2` (`FUNC: 4`)

  multiplies Matrix `A` with the preloaded Matrix `B`. `rs1` contains the address of Matriix `A`, and `rs2` is not used.

The matrix multiplication process on the accelerator involves a carefully sequenced set of instructions to manage data flow and computation. Initially, the stride for the data transfer must be set to ensure proper alignment. Following this, the `mvin` command is issued to move input activations and weights from memory to the scratchpad. This operation requires specifying the source memory addresses and the corresponding destination addresses in the scratchpad, enabling precise data placement for computation.

After the data has been moved, one of the matrices is preloaded into the systolic array within the PEs. This step ensures the array is primed with the necessary data before computation begins. The compute command then triggers the matrix multiplication, with the systolic array performing multiply-accumulate operations across its grid of processing elements. The results are accumulated in a dedicated SRAM during this process.

Once the computation is complete, the final step is to retrieve the results from

the accumulation SRAM and store them back into the main memory. This streamlined workflow, assuming the matrix dimensions fit within the systolic array, highlights the efficiency and structured approach required to maximize the accelerator's performance.

5.5.2 ED Accelerator Instructions

The ED accelerator uses the RoCC interface for communication with the RISC-V core. The ED accelerator is responsible for performing efficient sequence comparison. It uses a set of custom RISC-V instructions, similar to the BC accelerator, to handle data transfers and compute operations. Table 5.4 illustrates the custom instructions used to operate the ED accelerator:

Table 5.4: The custom RISC-V instructions to operate the ED accelerator. Note: The OPCODE is zero by default.

FUNC	RS2	RS1	RD	DESCRIPTION
0	N_Q	N_R	0	**Send quantity of Queries and References**
1	L_Q	Q	0	**Send both Queries lengths and Queries**
2	L_R	R	0	**Send both References lengths and References**
3	0	O_R	0	**Send Reference offsets**
4	0	0	$E_{N_Q \times N_R}$	**Wait for Results**

As detailed in Table 5.4, these instructions leverage the unique `OPCODE` of **0**, allowing the core to identify and operate the ED accelerator with precision. The

first instruction initializes the computation by specifying the number of query (N_Q) and reference (N_R) sequences. This step is critical for setting up the accelerator's processing environment. Subsequently, the second and third instructions transmit the lengths and actual data of the query and reference sequences (L_Q, Q, L_R, R) to the accelerator. These instructions ensure that the accelerator has all the necessary inputs to perform the edit distance calculation effectively. Additionally, the fourth instruction allows for the transfer of reference offsets (O_R), enabling more complex alignment tasks where sequences might not align from their starting indices.

The final instruction signals the accelerator to provide the computed results ($E_{N_Q \times N_R}$). By using these specialized instructions, the core and accelerator communicate seamlessly, optimizing data transfer and minimizing latency. This instruction-driven control mechanism highlights the tight integration between the RISC-V core and the ED accelerator, ensuring efficient execution of computationally intensive alignment tasks.

5.6 Preparing Models for Hardware Deployment through Software Optimization

Optimizing machine learning models for hardware deployment involves tailoring the model architecture and operations to align with the constraints and capabili-

ties of specific hardware accelerators. This section discusses the key optimization techniques applied to prepare models for efficient execution on hardware, including floating-point and quantized model preparation. General optimization strategies, such as quantization, BatchNorm folding, and converting convolutions to matrix multiplications, are explored in the following subsections. Additionally, matrix tiling is addressed to integrate accelerator instructions seamlessly into the model's runtime execution, ensuring optimal performance during inference.

5.6.1 Foundational Optimization Techniques

Optimizing machine learning models for hardware deployment requires tailoring their structure and operations to align with the computational and memory constraints of the target hardware. This subsection details three foundational techniques:

- `Quantization` Quantization converted the model's weights and activations from 32-bit floating-point (FP32) to 8-bit integer (INT8) precision. This approach significantly reduced memory usage and enabled the model to leverage the accelerator's hardware-friendly integer arithmetic, resulting in faster computation and lower power consumption. The INT8 format ensured that the model retained high accuracy while operating efficiently on the hardware.

- `BatchNorm Folding` BatchNorm layers, which are added after each convolutional layer during training to stabilize and accelerate convergence, introduce overhead during inference. To eliminate this overhead, BatchNorm parameters are folded into the preceding convolutional layers. By integrating these parameters into the convolution's weights and biases, the number of parameters and operations required during inference was reduced. This optimization streamlined the model and made it more efficient for deployment on hardware.

- `Conv to MatMul` To take full advantage of the accelerator's systolic array, which is optimized for matrix multiplication, all convolutional operations are transformed into matrix multiplication equivalents. This transformation involved restructuring the convolution operation into a series of matrix multiplications by reshaping the input data and weights into matrices. This step ensured that the model's operations are fully compatible with the hardware's computational units, maximizing acceleration efficiency and throughput.

5.6.2 Matrix Tiling

The BC accelerator is targeted to efficiently execute neural network models by handling the core computational task—matrix multiplication. The critical part of neural network model execution involves performing matrix multiplications, which are resource-intensive and need to be optimized for both speed and memory usage.

In this section, we explore how the BC accelerator manages these operations, focusing on the tiling process, memory management, and the efficient execution of matrix multiplications.

To perform matrix multiplication on the systolic array, matrices are divided into smaller submatrices or tiles, which fit within the dimensions of the systolic array. The size of each tile matches the systolic array size, denoted as `DIM`. If the dimensions of the matrices are not evenly divisible by `DIM`, padding is applied to make them compatible with the systolic array, as described in lines 1 of Algorithm 3. This padding ensures that all matrix tiles have consistent dimensions, which simplifies processing and maximizes the utilization of the systolic array. Another important consideration is the capacity of the accelerator's internal memories. The entire matrices cannot be stored at once, so the data must be processed in multiple stages, which involves dynamically managing memory tiles. The algorithm handles this by dividing the matrices into tiles and then determining how many tiles can fit into the available memory, ensuring efficient usage without exceeding the hardware's constraints. Algorithm 3 provides a detailed breakdown of how tiled matrix multiplication is carried out using a systolic array. The algorithm is structured using multiple loop levels, each serving a specific function in the overall multiplication process:

- **Loop Level 1** (lines 10): This level iterates over the larger matrix tiles

that have been created by dividing the padded matrices. The outer loops handle the traversal of these larger tile blocks across the dimensions I0, J0, and K0, dividing the padded matrices into tiles that can be processed by the accelerator. It defines the order in which these tiles are loaded and executed, ensuring that the computational workload is divided into manageable parts. Essentially, this loop level deals with selecting and working on a particular segment or tile of the input matrices that will be loaded into the systolic array for processing.

- **Loop Level 2** (lines 14): After selecting the tiles in Loop Level 1, Loop Level 2 focuses on processing smaller portions within each tile. This level is crucial for managing memory transfers between the accelerator's internal scratchpad and the systolic array. The tiling factor computed earlier is used to further split the matrix tiles into sub-tiles that can fit comfortably within the scratchpad memory. Loop Level 2 ensures that the memory is used optimally by loading only the necessary data for each compute operation, keeping in mind the constraints of internal memory.

- **Loop Level 3** (lines 18): This is the most fine-grained level of the computation, responsible for handling the submatrix multiplication. In this level, the operation is executed directly by the accelerator hardware using specialized

instructions discussed in § 5.5.1. The systolic array performs these operations in a highly parallel fashion, with each processing element (PE) working on separate elements, which maximizes computational throughput.

Algorithm 3 Tiled Matmul for Systolic Array Algorithm

Require: Matrix $A(I, K)$, Matrix $B(K, J)$, Systolic Array Size: `DIM`
Ensure: Product Matrix $C(I, J)$
Compute the matrix size after padding:
p*added_I* $= I \div$ `DIM` $+ (I \mod$ `DIM` $\neq 0)$, do same for p*added_J* and p*added_K*
Compute the tiling factor based on the scratchpad memory size:
t*iled_I* $=$ p*added_I* $\div$ `DIM`, do same for t*iled_J* and t*iled_K*
if *tiled_I* + *tiled_J* + *tiled_K* > MAX_TILES **then**
    Memory Outage Issue
end if
Compute the number of tiles to be handled:
$I0 = padded\_I/(tiled\_I \times DIM) + (padded\_I\%(tiled\_I \times DIM) \neq 0)$, do same for $J0$ and $K0$
Loop Level 1:
for $i0 = 0$ to $I0 - 1$ **do**
    for $j0 = 0$ to $J0 - 1$ **do**
        for $k0 = 0$ to $K0 - 1$ **do**
            Loop Level 2:
            for $j1 = 0$ to $tiled\_J - 1$ **do**
                for $k1 = 0$ to $tiled\_K - 1$ **do**
                    for $i1 = 0$ to $tiled\_I - 1$ **do**
                        Loop Level 3:
                        for $j2 = 0$ to $DIM - 1$ **do**
                            for $k2 = 0$ to $DIM - 1$ **do**
                                for $i2 = 0$ to $DIM - 1$ **do**
$C((i0 \times tiled\_I + i1) \times \texttt{DIM} + i2, (j0 \times tiled\_J + j1) \times \texttt{DIM} + j2) +=$
$A((i0 \times tiled\_I + i1) \times \texttt{DIM} + i2, (k0 \times tiled\_K + k1) \times \texttt{DIM} + k2) \times$
$B((k0 \times tiled\_K + k1) \times \texttt{DIM} + k2, (j0 \times tiled\_J + j1) \times \texttt{DIM} + j2)$
                                end for
                            end for
                        end for
                    end for
                end for
            end for
        end for
    end for
end for

5.7 FPGA Evaluation and Measurements

5.7.1 FPGA Implementation

The hardware potential of our second SoC, referred to as NanoSoC, is evaluated using a Xilinx VC707 field-programmable gate array (FPGA), as shown in Fig. 5.7. NanoSoC combines various processing styles and runs a RISC-V version of Linux OS, which helps manipulate the file system, control data, and manage program flow. As a result, we are in a good position to estimate the full-stack potential of our work, ranging across software, computing, and memory impacts.

Implemented directly on the FPGA, the design features a 64-bit RISC-V (ISA version RV64GC) processor core and its associated BC and ED accelerators. This setup allows for detailed testing by simulating realistic task scheduling, memory management, and I/O operations, while enabling real-time adjustments and iterations during the design phase. By running basecalling and sequence comparison code written in C on the Linux OS, we can assess NanoSoC's performance under multitasking scenarios. Thus, the FPGA provides an accurate environment to capture the system's behavior, ensuring that performance measurements reflect practical deployment conditions. NanoSoC operates at a clock frequency of 30 MHz, utilizing a 5-stage in-order Rocket microarchitecture with an integrated Floating Point Unit (FPU) [99].

Due to the limitations of available FPGA resources, the maximum systolic array size was constrained to 16×16, rather than the originally proposed 32×32 configuration. Despite this limitation, evaluating the performance using the smaller 16×16 array offers valuable insights into the system's behavior. It allows us to identify the overall system bottlenecks and evaluate the impact of a reduced systolic array on both throughput and energy efficiency. This assessment helps refine optimization strategies, including dataflow scheduling and workload distribution across processing elements, ensuring that the final SoC design can make the most effective use of a larger array when implemented in ASIC form.

The FPGA includes 1-GB DDR3 memory dedicated to our NanoSoC design, while an SD card is used for storing the file system and Linux kernel. To ensure a fair comparison of energy efficiency between the GPU, x86 CPU, and our proposed ASIC-based accelerators, we needed to address the memory transfer bottlenecks. The AXI bus that connects the processor and accelerators to the memory controller runs at 30 MHz, which limits data transfer rates. Additionally, the SD card introduces slower data access speeds, further affecting performance. To mitigate these bottlenecks, we chose to use a large raw dataset consisting of approximately 36,000 raw samples from *Klebsiella pneumoniae* [100], corresponding to around 4,500 base pairs. This dataset is used to evaluate processing speed, power consumption, and basecalling accuracy during sequencing analysis, with the reference sequence for

ED also stored on the SD card.

Using a large dataset serves several essential purposes: it minimizes the impact of I/O bottlenecks by ensuring that data loading times do not interfere with performance measurements, allows the evaluation to focus solely on computational efficiency, and ensures that the entire dataset can be fully loaded into the FPGA's DRAM. By doing so, we can confidently capture the computational capabilities of each platform without being affected by delays in data transfer, ensuring that the results provide an accurate reflection of real-world performance.

To evaluate the performance of the BC and ED accelerators, we alternated their operation to avoid resource conflicts and thus help clarify their individual characteristics. Using the Rocket core's control and status registers (CSRs) in Linux user mode, we recorded execution cycle counts, helping provide detailed insight into hardware performance. Power consumption was monitored using the TI Fusion Digital Power Designer [101].

5.7.2 FPGA Measurements

The FPGA measurements offer valuable insights into the performance of various components within the SoC, with a particular focus on the impact of cache configurations on the processor. Key aspects include evaluating how cache size influences the overall performance of the SoC, encompassing both the cores and accelerators,

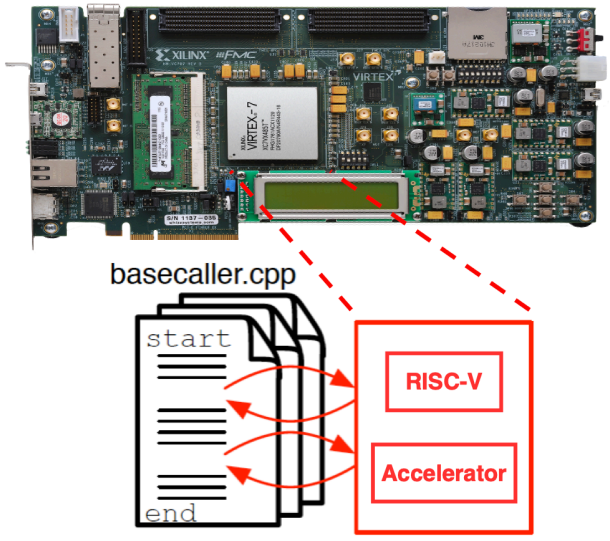

Figure 5.7: Software/Hardware arrangement of the NanoSoC system hardware emulation using Xilinx VC707 FPGA board.

analyzing the internal memory structure of the BC accelerator, and assessing the role of cache utilization in the ED accelerator. These studies underscore the importance of efficient memory hierarchies in optimizing data flow, reducing latency, and ensuring high-throughput performance in computational tasks.

5.7.2.1 RISC-V Multi-level Caches

In our evaluation of the system's full-stack hardware performance, we explored various configurations across NanoSoC's main computing components, the processors and accelerators, as well as its on-chip memory effects, and the L1 and L2 cache sizes. Cache configuration plays a pivotal role not only in determining memory access latency, but also in influencing other critical metrics such as power consump-

tion and chip area. Larger cache sizes typically reduce access latency, which directly improves performance by allowing faster data retrieval. However, increasing cache size comes at a cost both in terms of energy consumption, due to the higher power required to access larger memory, and in terms of area, as more silicon is needed to accommodate the additional memory cells. In the context of our SoC design, striking the right balance between these trade-offs is critical. Performance gains achieved by expanding the cache must be weighed against the increased energy requirements and the physical area occupied by the chip. This becomes especially important in constrained environments such as mobile DNA sequencing devices.

To better understand performance scaling, we also considered the power law of cache misses [102, 103], which suggests that the miss rate decreases as a power-law function of the cache size. Essentially, the larger the cache, the fewer the cache misses, but with diminishing returns as cache size increases further. This behavior aligns with our findings, where increasing cache size did reduce cache misses, yet the performance gains are not proportional due to the diminishing returns effect. This power law relationship helps guide our design decisions, ensuring that cache resources are allocated efficiently without overcommitting to areas with minimal performance returns.

This formulation reflects the fact that while increasing the cache size leads to performance improvements, the rate of improvement slows as the cache grows larger.

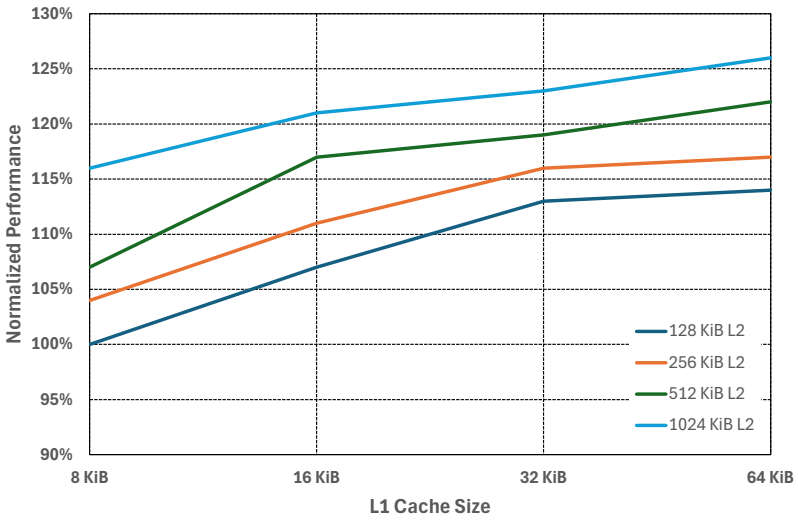

Figure 5.8: Comparison of normalized BC performance across different L1 and L2 cache sizes on NanoSoC.

For instance, doubling the L1 cache from 32 KiB to 64 KiB may provide only a 10% performance gain, despite the area cost nearly doubling [104].

This trend is clearly demonstrated in our experimental results discussed below, where we evaluate the impact of varying L1 and L2 cache sizes on both the BC and ED accelerators. By examining the performance across different cache configurations, we observe similar diminishing returns, reinforcing the theoretical expectations.

Fig. 5.8 highlights the impact of varying L1 and L2 cache sizes on the performance of the BC on NanoSoC. As seen on the x-axis, the L1 cache size is incremented from 8 KiB to 64 KiB, while the y-axis represents normalized performance, with the baseline set to 100% for an 8 KiB L1 and 128 KiB L2 configuration. The

lines depict various L2 cache sizes and how performance scales with changes in the L1 cache size.

Focusing first on the 128 KiB L2 configuration (dark-blue line), increasing the L1 cache size from 8 KiB to 16 KiB leads to a performance improvement of 7%. As the L1 cache is further increased to 32 KiB, performance continues to improve, reaching 113%. However, as the L1 cache size grows to 64 KiB, the performance increase becomes marginal, reaching only 114%. This behaviour highlights that while increasing the L1 cache size improves performance, the gains diminish beyond 32 KiB. The reason for this diminishing return lies in the working set of data for the basecalling task, which, beyond a certain cache size, can fit within the cache, thereby reducing cache misses.

In the basecalling application under consideration, the working set refers to the portion of the dataset that the processor actively uses during computation. The basecaller model has been selected as the most optimized model, as discussed in the previous section. Once the working set fits entirely within the cache, additional cache size offers minimal performance improvement, as the system already has sufficient space to avoid most memory accesses. For this specific problem size, the working set fits within a 32-KiB cache, as demonstrated by the performance plateau when moving to a 64-KiB L1 cache. Changes in the problem size, such as increasing the data processed during basecalling or altering memory access pat-

terns, would theoretically affect the performance. However, within the scope of this study, we have found that the current configuration is well-suited to the targeted basecalling application, as indicated by the performance stability observed beyond a 32-KiB cache. The focus here is to demonstrate how effectively the cache is used for this specific problem size, which reflects a realistic scenario for our basecalling application.

The performance-to-area ratio provides further insight into the tradeoffs. While increasing the L1 cache from 8 KiB to 16 KiB yields a substantial 7% performance improvement, the area required for this increase is significant. Doubling the cache typically requires roughly twice the silicon area, yet the performance gains taper off significantly beyond 32 KiB. For instance, moving from 32 KiB to 64 KiB results in less than a 1% increase in performance, highlighting the inefficiency of further cache expansion when weighed against area and power constraints. In other words, the performance boost does not scale proportionally with the area investment after a certain point.

Examining the impact of L1 cache size across other L2 configurations reveals similar trends but with more pronounced initial gains. For example, with a 512 KiB L2 cache, increasing the L1 cache from 8 KiB to 16 KiB results in a 10% performance gain, improving from 107% to 117%. As the L1 cache size increases further, the gains diminish. Increasing the L1 to 32 KiB results in a 2% improvement, reaching

119%, while further enlarging it to 64 KiB adds only 3%, bringing performance to 122%. A similar pattern is evident for the 1024 KiB L2 configuration, where increasing the L1 cache from 8 KiB to 16 KiB results in a 5% gain, but beyond 32 KiB, the performance increases slow significantly, reaching around 126% with a 64 KiB L1 cache. These diminishing returns suggest that increasing L1 cache size provides diminishing benefits once the working set is sufficiently cached, even when the L2 cache is large.

Similarly, scaling the L2 cache size while keeping the L1 cache constant follows a comparable pattern of diminishing returns. At an 8 KiB L1 cache, increasing the L2 cache from 128 KiB to 256 KiB improves performance by 4%, and doubling the L2 cache again provides only a 3% gain, moving from 104% to 107%. The largest L2 cache configuration yields an additional 9% increase, bringing performance to 116%. However, at larger L1 cache sizes (e.g., 32 KiB), increasing the L2 cache from 512 KiB to 1024 KiB results in only a 4% performance gain. These diminishing returns indicate that once the L2 cache becomes large enough to hold most of the model's weights and activations, further cache increases have little impact on performance. As with the L1 cache, the performance-to-area ratio becomes unfavourable after a certain threshold, meaning the benefits do not scale linearly with area or power consumption.

The diminishing returns observed in both L1 and L2 cache scaling follow a

common pattern seen in many programs, including basecalling. This effect occurs because, initially, increasing cache sizes reduces memory access latency, particularly between the SoC and external memory, which leads to significant performance improvements. In the case of basecalling, which involves processing sequential signal data through convolutional layers and other operations, the working set (model parameters, activations, and input data) eventually fits within the cache. Once this happens, further cache increases provide minimal gains, as the existing cache is efficiently handling most data access requests. Consequently, beyond a certain cache size, the additional area and power required to increase the cache provide diminishing performance improvements, making further scaling less cost-effective. While this pattern is not unique to basecalling, the specific data access patterns of the task emphasize the point at which cache scaling becomes inefficient for the SoC.

Shifting our focus to the ED accelerator, we selected a problem size representative of typical DNA sequencing use cases by comparing pairs of DNA sequences, each consisting of 1,000 bases in length. This specific problem size was employed to benchmark the accelerator's performance and generate the corresponding plot. Fig. 5.9 presents the normalized performance of ED across varying L1 and L2 cache sizes, following the same structure as Fig. 5.8. The x-axis reflects the L1 cache size, incrementing from 8 KiB to 64 KiB, and the y-axis represents the normalized per-

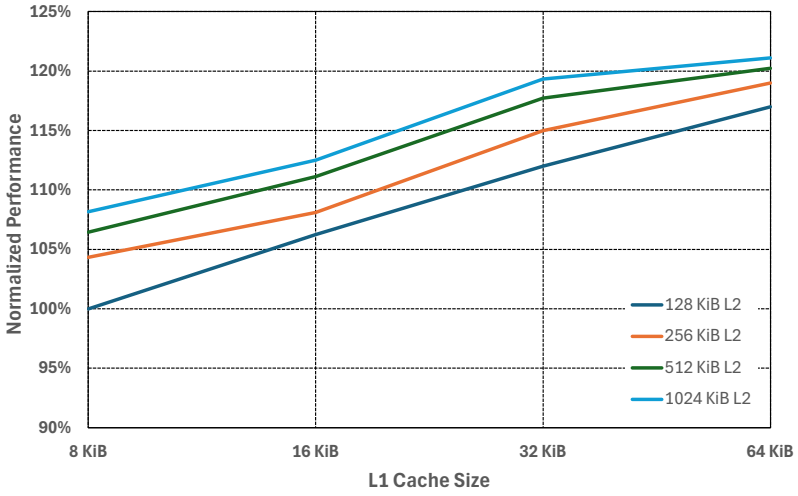

Figure 5.9: Comparison of normalized ED performance across different L1 and L2 cache sizes on NanoSoC.

formance, with the baseline set to 100% at 8 KiB L1 and 128-KiB L2. The lines in the graph represent the performance of different L2 cache configurations (from 128 KiB to 1024 KiB) in combination with the varying L1 cache sizes.

In contrast to the BC accelerator, the ED accelerator exhibits a stronger reliance on the size of the L1 cache for performance improvements. Specifically, within the 128 KiB L2 configuration, increasing the L1 cache from 8 KiB to 16 KiB results in a performance improvement from 100% to 106%. Further enlarging the L1 cache to 32 KiB and 64 KiB enhances performance to 112% and 117%, respectively. This consistent upward trend underscores the ED accelerator's dependence on larger L1 cache sizes to reduce memory access latency and improve data locality for sequence data. Unlike the BC accelerator, which begins to plateau in performance gains, the ED accelerator continues to benefit from increased cache sizes within the tested

range, indicating that its computational demands are more tightly coupled with cache performance. The absence of a clear saturation point suggests that the ED accelerator could continue to gain performance with even larger caches, highlighting the importance of adequately provisioning cache hierarchies to support the computational demands of real-time DNA sequence comparison tasks within our NanoSoC design.

When we examine the larger L2 cache sizes, the trends remain consistent. For example, with a 512 KiB L2 cache, increasing the L1 cache from 8 KiB to 16 KiB results in a 5% performance gain, from 106% to 111%. Moving to 32 KiB L1 improves performance by an additional 7%, bringing it to 118%, but the gains from further increasing the L1 cache to 64 KiB are minimal, raising performance by just 2%. The largest L2 configuration, 1024 KiB, follows the same pattern: a modest 5% gain from increasing L1 cache to 32 KiB, but only a 2% gain from 32 KiB to 64 KiB L1. This diminishing return shows that, beyond a certain point, increasing L1 cache size provides little additional benefit since the data required for ED computations already fits within the cache.

From a performance-to-area perspective, enlarging both L1 and L2 caches results in diminishing performance gains. Doubling the L1 cache size from 16 KiB to 32 KiB provides noticeable improvements, but increasing it further to 64 KiB leads to a marginal gain despite the significant increase in area. Similarly, expanding L2 cache

size from 128 KiB to 512 KiB improves performance, but moving to 1024 KiB offers little additional benefit. This indicates that after a certain point, the performance boost does not scale proportionally with the area and power cost, especially once the working set fits comfortably in the cache.

These findings suggest that a 32 KiB L1 and 512 KiB L2 cache configuration strikes the right balance between performance improvement and resource efficiency. This setup provides sufficient cache space to handle the ED accelerator's memory demands while avoiding excessive area and power consumption. The same configuration, as seen in the BC analysis, offers optimal performance for both accelerators, ensuring the SoC maximizes efficiency for real-time sequencing tasks.

5.7.2.2 BC Accelerator

As part of our efforts to optimize the accelerator's performance, we investigated varying the sizes of both the scratchpad and accumulation memories, as shown in Fig. 5.10. It provides a visual representation of how different memory configurations influence the accelerator's performance. The sizes of the scratchpad (dark blue bars) and accumulation memory (orange bars) are depicted on the left y-axis, while the right y-axis displays the normalized performance. For this analysis, we use a configuration with both scratchpad and accumulation memories set at 64 KiB as the baseline (Start of the left red line), against which all other configurations are

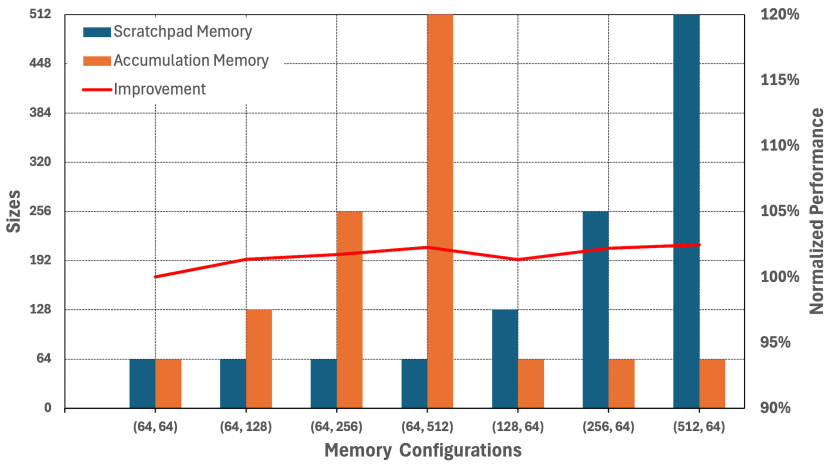

Figure 5.10: Impact of scratchpad (dark blue) and accumulation (orange) memory size changes on the BC accelerator's performance (red line). Bar labels on x-axis denote (scratchpad size and accumulator size in KiB).

compared to measure performance gains.

The experimental results reveal that increasing the memory sizes beyond the baseline configuration yields minimal performance improvements. Specifically, enlarging the scratchpad memory from 64 KB to higher capacities results in only a marginal increase in normalized performance. A similar trend is observed when scaling the accumulation memory size; the performance gains are negligible (only 1-2%) despite the additional memory resources. These findings indicate that the BC accelerator's performance is not significantly sensitive to the sizes of the scratchpad and accumulation memories within the tested range. The minimal impact suggests that the accelerator efficiently utilizes memory resources and that further increases do not contribute to substantial performance enhancements. This efficiency allows for optimization of the accelerator's design by selecting memory sizes that balance

performance with area and power constraints, without compromising the overall system throughput.

Based on these observations, we decided to configure our SoC with a 256 KiB scratchpad memory and a 64 KiB accumulation memory for the BC accelerator. This configuration offers an optimal balance between performance and resource utilization. The choice of a 256 KiB scratchpad memory ensures that the accelerator can efficiently handle the required data without incurring unnecessary area and power overheads associated with larger memory sizes. While increasing the scratchpad memory beyond 256 KiB showed no significant performance gains, this size provides sufficient capacity to accommodate the working datasets effectively.

To further evaluate the overall performance of the BC accelerator, we conducted a series of experiments to compare the execution of the basecalling algorithm across various computational platforms. Specifically, we implemented the basecalling algorithm in three different ways: (1) a floating-point runtime executed on a RISC-V CPU, (2) a fixed-point (int8) version also running on the RISC-V CPU, and (3) the fixed-point version running on our custom BC accelerator.

As shown in Fig. 5.11, using the floating-point CPU implementation as the baseline, we observed that the fixed-point CPU implementation achieved only a modest performance improvement of 1.1×. This limited gain suggests that simply converting to int8 arithmetic on the CPU does not yield significant speedups.

Several factors could contribute to this minimal improvement. One possibility is that the RISC-V CPU has efficient hardware support for floating-point operations, which narrows the performance gap between floating-point and integer computations. Additionally, the fixed-point implementation may not fully leverage CPU-specific optimizations such as vectorization or parallelism, which are crucial for maximizing performance gains. Memory access patterns and the overhead of data type conversions might also offset the expected benefits of using smaller data types.

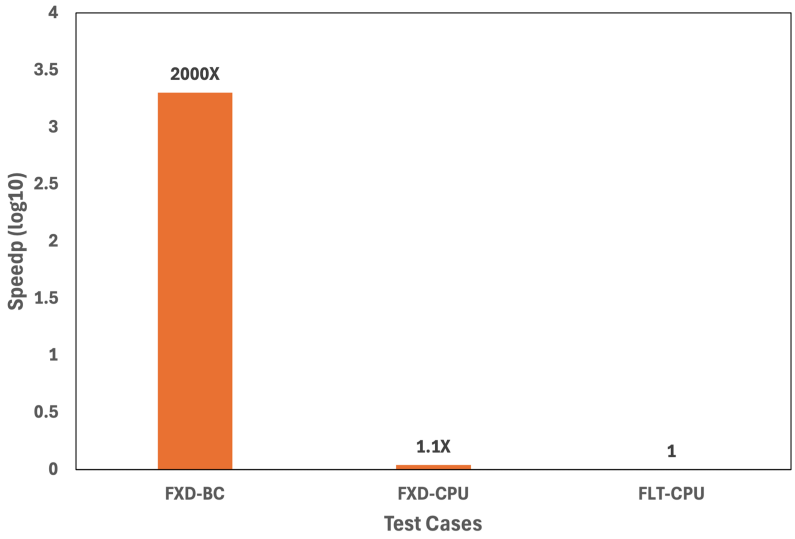

Figure 5.11: Cycle count comparison for three basecalling test cases: hardware accelerated fixed-point (Fxd-CPU + BC-accel), fixed-point on CPU (Fxd-CPU), and floating-point on CPU (Flt-CPU).

In stark contrast, the fixed-point implementation running on the BC accelerator demonstrated a dramatic performance improvement, achieving a speedup of approximately 2000× compared to the floating-point CPU baseline. This substantial acceleration is measured in terms of cycle counts.

Energy efficiency is a crucial factor in evaluating the performance of computational platforms for basecalling, especially in resource-constrained environments like mobile DNA sequencing devices. Achieving a balance between processing speed and power consumption is essential for practical deployment. In this context, we compared the basecalling performance and energy efficiency of three platforms: a high-performance GPU (NVIDIA Titan RTX), a typical x86 CPU (Intel Xeon W-2123), and BC implemented on an FPGA. The comparison of basecalling performance across the GPU, CPU, and our FPGA-based BC accelerator, as shown in Table 5.5, reveals some insights into both the computational capabilities and energy efficiency of these platforms, particularly in the context of mobile DNA sequencing.

Table 5.5: Performance comparison of GPU, CPU, and NanoSoC (FPGA) for basecalling.

Platform	Freq (MHz)	Samples/s	Output Bases	Bases/s	Active Power(W)	Bases/Joule
GPU	1770	1.5e5	4574	10654.5	60	177.6
CPU (x86)	3600	1.3e4	4612	958.8	51	18.8
NanoSoC (FPGA)	30	363.8	4612	26.0	0.12	216.6

While the GPU delivers the highest raw throughput at 10,654.5 bases per second, this comes with a significant tradeoff in power consumption, drawing 60-W and achieving a moderate energy efficiency of 177.6 bases per joule. Its parallel architecture is ideal for tasks requiring high throughput, but the substantial power

demands limit its practicality in energy-constrained environments like mobile sequencing devices. The x86 CPU, with a throughput of 958.83 bases per second and 51-W consumption, offers lower performance and an even lower energy efficiency of 18.8 bases per joule, making it less suitable for long-term, battery-powered applications where energy conservation is key.

In contrast, the FPGA-based BC accelerator, though operating at a modest 30 MHz and delivering a much lower throughput of 26.0 bases per second, excels in energy efficiency, consuming only 0.12 W (active power) to achieve 216.6 bases per joule— 1.2 times more efficient than the GPU and over 11.5 times more efficient than the CPU. This makes the BC accelerator particularly suitable for energy-sensitive applications such as mobile DNA sequencing, where minimizing power consumption outweighs the need for maximum throughput.

From these measurements, if the BC accelerator were implemented on an ASIC running at 1.5 GHz, the expected throughput would be approximately 1,300 bases per second, surpassing the CPU's performance significantly. To match or exceed the GPU's performance of 10,654.5 bases per second, it would require roughly 9-10 BC accelerator cores operating in parallel. This setup would not only achieve comparable throughput to the GPU but would do so with significantly greater energy efficiency. Given the FPGA-based BC's energy consumption of just 0.12 W at 30 MHz, scaling up to an ASIC implementation at 1.5 GHz would retain a

substantial advantage in terms of bases per joule compared to the GPU, making it a highly energy-efficient solution for high-throughput basecalling tasks. With its low power consumption and scalability, the BC accelerator stands out as a promising solution for high-throughput and energy-efficient basecalling tasks.

In addition, we integrated clock gating features into the BC part of the NanoSoC on the FPGA. Clock gating is a power-saving technique that reduces dynamic power consumption by selectively disabling the clock signal to certain parts of the circuit when they are not in use. This is particularly important in reducing unnecessary switching activity, which is a major contributor to power consumption.

5.7.2.3 ED Accelerator

An evaluation of NanoSoc's ED accelerator enabled performance potential is summarized in Table 5.6. Therein we present a comparison of the performance and the energy-efficiency of sequence comparison across four different platforms.

The key metrics evaluated include Giga Cell Updates Per Second (GCUPS), runtime, power consumption, and the energy-efficiency (measured in GCUPS-per-joule). The ED accelerator achieves a GCUPS/Joule of 1.415, which is approximately 381× more efficient than the RISC-V processor which only achieves 3.71×10^{-3} GCUPS/Joule. This stark difference highlights the superior efficiency of the specialized ED accelerator compared to a general-purpose processor like the RISC-V.

Table 5.6: Performance and power consumption comparison of the ED accelerator across different platforms.

Platform	GCUPS ($\times 10^{-3}$)	Runtime (ms)	Active Power (mW)	GCUPS/Joule
ED Accelerator @30MHz	92.0×10^{-3}	4.13×10^{3}	(ED only) 65	1.415
RISC-V @30MHz	0.26×10^{-3}	112.70×10^{3}	70	3.71×10^{-3}
ARM @1.2GHz	13.13×10^{-3}	3.93×10^{3}	150	0.0875
x86 @3.6GHz	134.08×10^{-3}	0.37×10^{3}	51,000	2.63×10^{-3}

The ARM and x86 processors, operating at higher clock frequencies, also show lower energy-efficiency compared to the ED accelerator. In particular, the proposed ED accelerator demonstrates 16× and 538× better energy-efficiency than the ARM and the x86 processors (despite its high GCUPS), respectively. These results emphasize the significant trade-offs between raw performance and energy-efficiency. Hence, the specialized ED accelerator strikes a better balance between computational capability and power consumption, making it a particularly attractive solution for battery-limited applications.

While the ED accelerator demonstrates substantial performance and energy-efficiency improvements over general-purpose platforms, we do not provide direct comparisons with other similar specialized accelerators in the literature. This is largely due to significant differences in the underlying alignment algorithms, as

well as distinct design goals and target applications. Many existing accelerators are optimized for specific workloads and architectures that are not directly comparable to our SoC, which is designed with a focus on power efficiency for mobile and embedded environments. To maintain a fair and consistent analysis, we have chosen to compare our results against platforms that share similar general-purpose characteristics and operational constraints.

5.8 ASIC Evaluation and Measurements

The evaluation and measurement of the ASIC implementation are conducted using the same test platform and SoC test system as those employed for the first chip. Additionally, both chips share identical packaging and substrate sizes, ensuring uniformity in their physical designs.

5.8.1 ASIC Implementations

A NanoSoC-like implementation in ASIC form was investigated. Certain practical constraints and fabrication issues presently prevent demonstrating the full potential of a NanoSoC chip. Although the root cause of these issues is still being isolated, it appears that certain physical challenges impact less than 1% of the total computations. Consequently, gross system power and performance metrics remain valid and provide valuable insights into the chip's capabilities.

The physical implementation achieved provides a valuable assessment of the practical challenges and the potential for realistic bioinformatic SoCs, serving as a useful guide for researchers. A photograph of the fabricated ASIC die is shown in Fig. 5.12. This image provides a visual representation of the physical layout, including the arrangement of input/output (I/O) pads and the core computational area. The die's structure reflects the constraints and design considerations discussed earlier, including the allocation of space for the computational core and peripheral components.

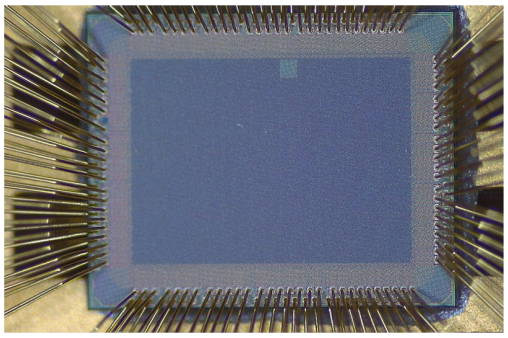

Figure 5.12: Photograph of the fabricated ASIC die showcasing the layout of the computational core and I/O pad arrangements.

The SoC was fabricated using GlobalFoundries™ 22-nm CMOS fully-depleted silicon-on-insulator (FDSOI) process. The cost and performance tradeoffs offered by this technology are promising for embedded systems at scales comparable to the NanoSoC concept[105]. The layout of the implemented system is shown in Fig. 5.13. The chip area, 5 mm^2, is constrained by the space provided by the fab house for the design submission. Larger designs could potentially be pursued for commercial

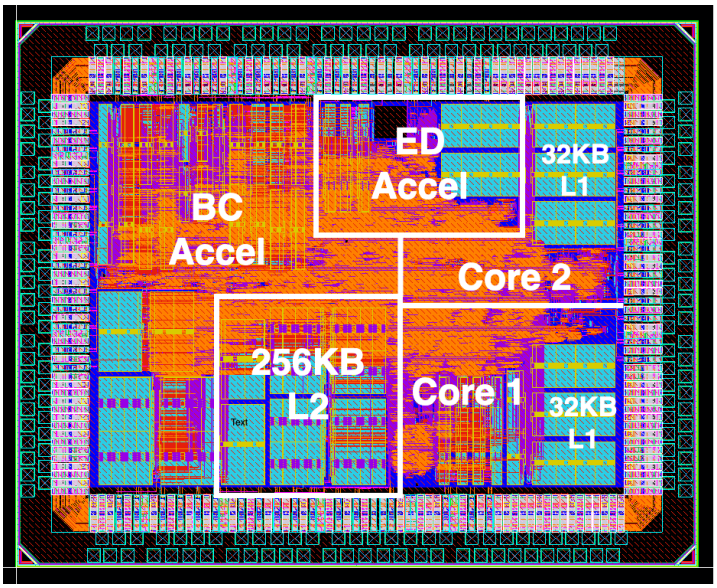

Figure 5.13: Layout of the proposed 5 mm² ASIC, featuring dual in-order 5-stage RISC-V cores (RV64GC), each paired with dedicated ML-Based BC and ED accelerators.

applications. In accommodating all the requirements of a practical system, the available space must be shared among the SoC's core computational components and the communications blocks responsible for data and program transfer between the chip and external memory or data ports. Additionally, significant area is occupied by the 205 input/output (I/O) pads and their signal-conditioning pad drivers, which consume over 40% of the available chip space, leaving approximately 3 mm^2 for the main components of the NanoSoC

As with the NanoSoc proposal, the design in this space features two tiles, each composed of in-order 5-stage RISC-V pipelines configured as 64-bit Rocket micro-architectures. Each core has 32-KiB of L1 cache, split evenly between instruction

and data. As described in § 5.2, versions of the BC and ED accelerators are also attached to each of these RISC-V tiles. The tiles communicate between each other and to external memory via L2 which is 256 KiB. Including the internal memories of the accelerators the SoC contains a total of 672 KiB of SRAM. Unfortunately the space requirements committed to I/O and on-chip memories allowed us to implement only a 4×4 systolic array, purely floating point, implementation for the BC accelerator. Further, a wiring bug in our physical implementation corrupts communications between ED and its tile and also causes intermittent calculation error in our BC accelerator.

The final chip could boot Linux and correctly run C programs therein at a maximum operating frequency of 250 MHz, a value largely limited by our need to provide off-chip clocking. The technology itself has the potential to exceed 1-GHz operating speeds. At 250 MHz, the system's Linux-only power consumption is 34 mW from a 0.8-V supply. When basecalling on just a single RISC-V core, the power consumption rises by 12 mW albeit at an achieved rate of only 6 bases/s - a clear limitation of unaccelerated operation. However, with BC acceleration engaged (albeit with incorrect computations as noted above) this calculation is executed 16× faster, in line with expectations for a 4×4 array. This is achieved with 7.5× improvement in energy efficiency over a core-only run. Based on these measurements, scaling to a 32×32 array at 1-GHz does then hold the promise of operating at

25 kbases/s, about a fourth of the maximum practical measurement throughput of current miniature devices, and hence a means of approaching reasonable operation. The transition to purely integer-based computations offers further opportunities for speed-up.

5.8.2 ASIC Measurements

The performance of the BC accelerator was evaluated in its ability to perform matrix multiplications, a core operation in many neural network-based applications. Fig. 5.14 compares the execution time (in clock cycles) for matrix multiplication across various matrix sizes, ranging from 32×32 to 2048×2048, on both the BC accelerator and the CPU (RISC-V core).

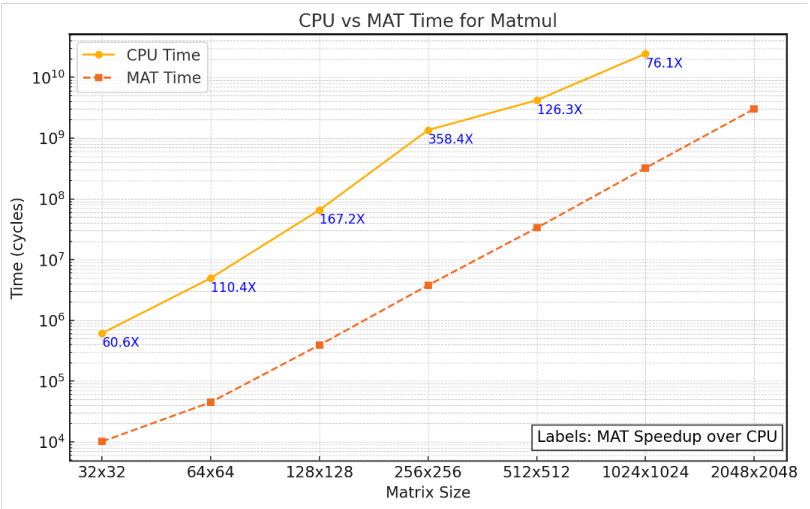

Figure 5.14: Performance comparison of matrix multiplication on the BC accelerator and the RISC-V CPU. The plot shows execution time (in clock cycles)

The results demonstrate that the BC accelerator significantly outperforms the CPU, particularly as the matrix size increases. On average, the accelerator achieves a speedup of approximately 165× across the evaluated matrix sizes. This includes substantial gains for smaller matrices, such as 32×32, as well as impressive performance for larger matrices, such as 512×512 and 1024×1024, highlighting the accelerator's efficiency across a range of computational demands.

The non-linear scaling of speedup across different matrix sizes highlights the efficiency of the systolic array architecture used in the BC accelerator. Larger matrices leverage the parallelism of the accelerator more effectively, resulting in significantly reduced computation time compared to the sequential processing of the RISC-V CPU. This performance gain validates the architectural design of the BC accelerator, particularly in computationally intensive tasks like matrix multiplication.

From an implementation perspective, these results emphasize the suitability of the BC accelerator for neural network workloads, where matrix multiplication operations dominate the computation. The combination of high throughput and significant power efficiency improvements makes this accelerator an ideal candidate for integration into the mobile DNA sequencing pipeline.

Beside the matrix performance evaluation, the baseline power consumption of the SoC is analyzed to investigate the impact of reducing supply voltage (Vdd) on power savings at different operating frequencies while maintaining chip stability.

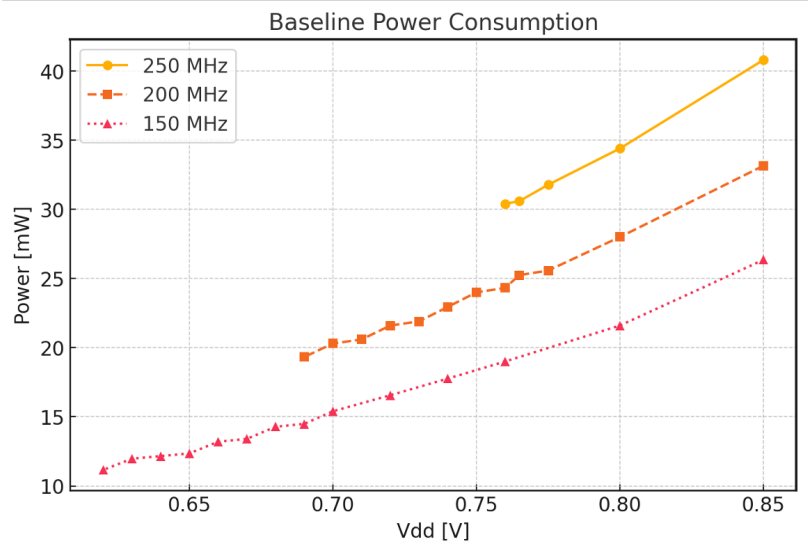

Figure 5.15: Baseline power consumption of the ASIC at different operating frequencies (150 MHz, 200 MHz, and 250 MHz) and its supply voltages (Vdd).

Fig. 5.15 shows the power consumption trends at 150 MHz, 200 MHz, and 250 MHz frequencies across a range of Vdd values, from 0.65 V to 0.85 V.

As expected, the power consumption decreases significantly as Vdd is lowered, following the quadratic relationship between power and supply voltage in CMOS circuits. At 250 MHz, power consumption drops from approximately 43 mW at 0.85 V to about 30 mW at 0.70 V, while at 150 MHz, the reduction is even more pronounced, starting at just above 12.5 mW at 0.65 V. This highlights the effectiveness of lowering Vdd as a strategy for reducing power consumption, particularly at lower operating frequencies.

This analysis underscores the trade-offs involved in optimizing for energy efficiency. At lower frequencies, the BC accelerator benefits from greater power reduc-

tions without compromising stability, making it ideal for energy-sensitive mobile sequencing applications. For performance-critical tasks requiring higher frequencies, the power savings are still significant, though the operational voltage range narrows.

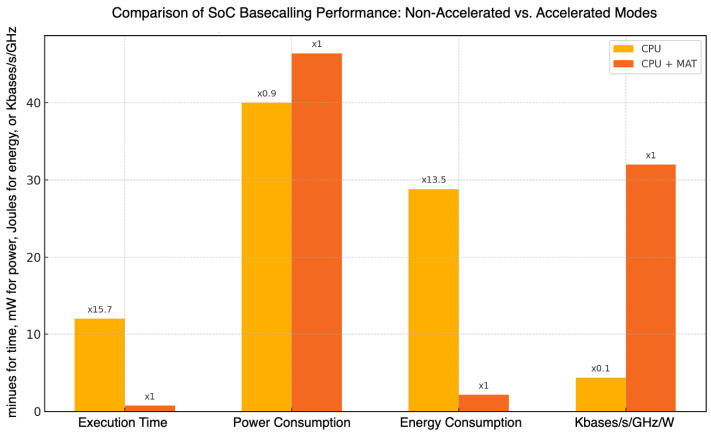

Figure 5.16: Comparison of SoC basecalling performance in non-accelerated (CPU-only) and accelerated (CPU+BC) modes.

Fig. 5.16 presents a detailed measurement of the basecalling model introduced in § 5.1.2, comparing the performance of the RISC-V core alone (CPU) with the RISC-V core augmented by the BC accelerator (CPU+BC). The comparison is based on four key metrics: execution time, power consumption, energy consumption, and throughput (Kbases/s/GHz/W). These measurements highlight the impact of hardware acceleration on the basecalling workload, a critical component of the DNA sequencing pipeline.

For execution time, the CPU+BC system achieves a 15.7x reduction, demonstrating the accelerator's ability to efficiently handle the computational demands of the basecalling model. By leveraging the MAT for matrix-heavy operations, the system significantly reduces processing latency, which is essential for enabling real-time basecalling.

The power consumption comparison shows a modest increase for the CPU+BC system. However, when viewed in conjunction with execution time, the 13.5× reduction in energy consumption highlights the BC's capability to balance performance and energy efficiency. This result underscores the suitability of the accelerated system for mobile sequencing applications, where power constraints are a major design consideration.

Throughput, as measured in Kbases/s/GHz/W, shows consistent improvement with the CPU+BC system, ensuring scalability and robustness for varying basecalling workloads. The combined improvements make the CPU+BC configuration a compelling choice for deploying the basecalling model in real-world scenarios.

Overall, the CPU+MAT system delivers transformative improvements, including a 15× speed-up in execution time and a 13× boost in energy efficiency. These advancements validate the MAT accelerator as a crucial element in improving the performance and energy efficiency of basecalling models, significantly enhancing their role in the broader DNA sequencing pipeline.

5.9 Chapter Summary

This chapter presented the development and evaluation of the second SoC, combining both software and hardware innovations to accelerate DNA sequencing tasks, particularly basecalling and sequence alignment.

From the software perspective, a modular basecalling system was designed to support various neural network architectures. The Conv1D-based model was selected for its balance of computational simplicity and sufficient accuracy (84.5%), enabling compatibility with hardware accelerators. Optimization techniques such as quantization, BatchNorm folding, and convolution-to-matrix conversion were applied to ensure efficient execution on the SoC.

On the hardware side, the SoC was evaluated on both FPGA and ASIC platforms. The FPGA implementation, utilizing a 16×16 INT8 systolic array accelerator, was optimized through cache size adjustments tailored to the BC and ED algorithms. This resulted in significant performance gains, including a 2000× speedup for basecalling compared to the RISC-V CPU at the same frequency. Additionally, the FPGA-based BC accelerator demonstrated impressive energy efficiency, achieving 1.2× the efficiency of a GPU and 11.5× that of an x86 CPU. The ED accelerator on the FPGA exhibited a 30× higher throughput and 381× greater energy efficiency compared to the RISC-V CPU, underscoring the benefits of hardware acceleration

for sequence alignment tasks.

The ASIC implementation, despite utilizing a smaller systolic array, achieved a higher clock frequency of 250 MHz and validated the architectural design by delivering up to a 358× speedup. The ASIC consumed significantly less power (3.8 mW) than the FPGA (120 mW), further highlighting its energy efficiency. For basecalling, the ASIC demonstrated a 15.7× reduction in execution time and a 13.5× improvement in energy consumption compared to the RISC-V CPU. The ED accelerator on the FPGA platform maintained its advantages, achieving 16× higher energy efficiency than ARM processors and 7× greater efficiency than x86 CPUs.

Together, these software and hardware innovations underline the potential of the second SoC to meet the computational and energy demands of modern DNA sequencing pipelines, making it a strong candidate for real-world deployment in resource-constrained environments.

6 Conclusion and Future Work

The thesis aimed to address the challenges associated with real-time bioinformatics processing in portable DNA sequencing applications. This was achieved through the development of algorithms and the design of two SoC ASICs tailored for computationally intensive tasks, including basecalling and sequence alignment. The work focused on bridging the gap between the computational demands of DNA sequencing pipelines and the resource constraints of portable sequencing devices, emphasizing performance, energy efficiency, and integration.

6.1 Thesis Conclusion

This thesis tackled the core challenges of enabling real-time bioinformatics processing in portable DNA sequencing devices by designing and implementing specialized software and hardware solutions. Through this work, significant advancements were made in addressing the computational and energy constraints associated with nanopore sequencing, culminating in the development of the world's first SoC specif-

ically tailored for this application.

The software innovations provided the foundation for a flexible and adaptable basecalling pipeline, incorporating computational models like HMMs, Conv1D, GRU, LSTM, and transformer encoders. This modularity ensured the ability to evaluate diverse algorithms while optimizing for specific constraints of portable sequencing environments. The adoption of Conv1D-based basecalling, combined with optimizations such as INT8 quantization and batch normalization folding, demonstrated a careful balance between computational efficiency and accuracy. Similarly, the optimized ED algorithm proved essential for handling the intensive sequence alignment tasks central to the pipeline.

The hardware contributions pushed the boundaries of portable bioinformatics systems. The first SoC design established the feasibility of embedding HMM-based basecalling accelerators, offering an initial step toward integrated sequencing solutions. Building upon this, the second SoC introduced a neural network-based approach, integrating a systolic array and specialized ED accelerator to deliver unprecedented performance improvements. The FPGA validations and ASIC implementations demonstrated real-world applicability, achieving significant speed and energy efficiency gains.

In conclusion, this work represents a major step forward in the evolution of portable DNA sequencing technology. By demonstrating real-time, energy-efficient

processing capabilities in a compact, scalable SoC, it opens the door for widespread adoption in resource-constrained environments. Beyond the immediate contributions, the thesis sets the stage for future exploration of advanced machine learning architectures, expanded bioinformatics functionality, and further improvements in power management. Together, these efforts pave the way for next-generation portable sequencing devices that can revolutionize genomics research and diagnostics, bringing powerful tools to labs, clinics, and remote settings worldwide.

6.2 Reflections and Impact of Contributions

This research journey was not merely about achieving technical milestones but also about gaining deeper insights into the intricate interplay between algorithm design and hardware architecture. A key learning was the critical importance of co-design, where software and hardware are developed collaboratively to address specific challenges in computational complexity, memory constraints, and power efficiency. This approach provided a framework for identifying trade-offs and optimizing solutions tailored to the unique requirements of portable bioinformatics systems.

From a broader perspective, this work highlighted the evolving landscape of bioinformatics processing, where traditional computational methods fall short in resource-constrained environments. The systematic comparison of models, including HMMs, Conv1D, and neural networks, demonstrated the significance of sim-

plicity, scalability, and accuracy in model selection, especially when transitioning to hardware. The Conv1D-based basecaller emerged as a key contribution, bridging the gap between high performance and hardware efficiency, offering a practical path forward for similar computational pipelines.

On the hardware side, this research underscored the transformative potential of specialized architectures. The design and validation of the two SoCs highlighted the advantages of tailoring accelerators—such as systolic arrays and ED-specific modules—for bioinformatics workloads. This specialization not only improved speed and energy efficiency but also demonstrated the feasibility of embedding complex bioinformatics tasks into compact and portable platforms. The ASIC measurements, achieving unprecedented efficiency metrics, reinforce the potential for such designs to revolutionize mobile sequencing applications.

What was ultimately learned is that the field of portable DNA sequencing requires an integrated approach that goes beyond solving immediate computational challenges. It demands a vision for how bioinformatics can evolve through hardware and software innovation. This work contributes to that vision by establishing a methodology for hardware-software co-design, demonstrating the value of modular and adaptable frameworks, and pushing the boundaries of what is possible in energy-efficient, real-time sequencing devices. These lessons extend beyond this project, offering a blueprint for addressing computational challenges in other do-

mains of portable, resource-constrained systems.

6.3 Future Work

6.3.1 Enhanced ML Architectures for Basecalling

Future work should consider the adoption of more advanced ML models for basecalling, such as transformer-based models that could further improve accuracy and adaptability to varying sequencing conditions. This would require modifications to the accelerator design to efficiently handle the increased computational complexity associated with such models. Exploring different quantization and pruning techniques could also help reduce the computational load while maintaining accuracy, which would be crucial for real-time applications.

6.3.2 Integration of Additional Bioinformatics Functions

The proposed SoC architecture currently focuses on basecalling and sequence alignment. Future iterations could integrate additional bioinformatics functions, such as variant calling and genome assembly, to create a more comprehensive sequencing platform. Adding these capabilities would reduce the need for external computational resources, thereby achieving a truly stand-alone sequencing device.

6.3.3 Advanced Power Management Techniques

Power efficiency remains a critical aspect of mobile sequencing devices. Future research could investigate the use of dynamic voltage and frequency scaling (DVFS) and power gating techniques to further reduce power consumption, especially during idle periods or low-demand tasks. Implementing such techniques would enhance the energy efficiency of the SoC, making it more practical for field applications where battery life is a limiting factor.

6.3.4 ASIC Implementation and Scaling

The current SoC designs were implemented using a 22-nm CMOS process. Future work could focus on scaling the design to more advanced technology nodes, such as 7-nm or 5-nm, which would provide higher performance and lower power consumption. Additionally, a full custom ASIC implementation could further optimize the SoC's performance, area, and power characteristics, potentially making it suitable for mass production in commercial portable DNA sequencers.

6.3.5 Broader Applicability and Field Testing

Finally, the proposed SoCs should be tested in broader real-world scenarios, including field deployments in diverse environmental conditions. Such testing would

help refine the design to ensure robustness and reliability. Expanding the scope of applications beyond DNA sequencing, such as RNA sequencing or metagenomics, could also increase the utility and impact of the developed SoC.

6.4 Concluding Remarks

The work presented in this thesis represents a significant advancement in the field of portable genomics by developing specialized SoCs for real-time DNA sequencing. Starting from an HMM-based accelerator and advancing to an NN-based approach with integrated sequence alignment, the research demonstrates the feasibility of embedding sophisticated bioinformatics computations within a compact hardware platform. The proposed architectures provide substantial improvements in speed, energy efficiency, and computational capability, contributing to the vision of a truly portable, stand-alone DNA sequencing device. Moving forward, the integration of more advanced ML models, additional bioinformatics functions, and enhanced power management techniques will be key to furthering the impact and practicality of these systems for in-field genomic analysis.

Bibliography

[1] ZhongPan Wu, Karim Hammad, Robinson Mittmann, Sebastian Magierowski, Ebrahim Ghafar-Zadeh, and Xiaoyong Zhong. Fpga-based dna basecalling hardware acceleration. In *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1098–1101. IEEE, 2018.

[2] Daniel Branton, David W Deamer, Andre Marziali, Hagan Bayley, Steven A Benner, Thomas Butler, Massimiliano Di Ventra, Slaven Garaj, Andrew Hibbs, Xiaohua Huang, et al. The potential and challenges of nanopore sequencing. *Nature biotechnology*, 26(10):1146–1153, 2008.

[3] David Deamer, Mark Akeson, and Daniel Branton. Three decades of nanopore sequencing. *Nature biotechnology*, 34(5):518–524, 2016.

[4] Wenqing Shi, Alicia K Friedman, and Lane A Baker. Nanopore sensing. *Analytical chemistry*, 89(1):157–188, 2017.

[5] Ralph MM Smeets, Ulrich F Keyser, Nynke H Dekker, and Cees Dekker. Noise in solid-state nanopores. *Proceedings of the National Academy of Sciences*, 105(2):417–421, 2008.

[6] Raj D Maitra, Jungsuk Kim, and William B Dunbar. Recent advances in nanopore sequencing. *Electrophoresis*, 33(23):3418–3428, 2012.

[7] Yunhao Wang, Yue Zhao, Audrey Bollas, Yuru Wang, and Kin Fai Au. Nanopore sequencing technology, bioinformatics and applications. *Nature biotechnology*, 39(11):1348–1365, 2021.

[8] Minsoung Rhee and Mark A Burns. Nanopore sequencing technology: nanopore preparations. *TRENDS in Biotechnology*, 25(4):174–181, 2007.

[9] Alberto Magi, Roberto Semeraro, Alessandra Mingrino, Betti Giusti, and Romina D'aurizio. Nanopore sequencing data analysis: state of the art, applications and challenges. *Briefings in bioinformatics*, 19(6):1256–1272, 2018.

[10] Alessio Fragasso, Sonja Schmid, and Cees Dekker. Comparing current noise in biological and solid-state nanopores. *ACS nano*, 14(2):1338–1349, 2020.

[11] Matei David, Lewis Jonathan Dursi, Delia Yao, Paul C Boutros, and Jared T Simpson. Nanocall: an open source basecaller for Oxford Nanopore sequencing data. *Bioinformatics*, 33(1):49–55, 2017.

[12] Haotian Teng, Minh Duc Cao, Michael B Hall, Tania Duarte, Sheng Wang, and Lachlan JM Coin. Chiron: translating nanopore raw signal directly into nucleotide sequence using deep learning. *GigaScience*, 7(5):giy037, 2018.

[13] Xuechun Xu, Nayanika Bhalla, Patrik Ståhl, and Joakim Jaldén. Lokatt: A hybrid dna nanopore basecaller with an explicit duration hidden markov model and a residual lstm network. *BMC bioinformatics*, 24(1):461, 2023.

[14] Neng Huang, Fan Nie, Peng Ni, Feng Luo, and Jianxin Wang. Sacall: a neural network basecaller for oxford nanopore sequencing data based on self-attention mechanism. *IEEE/ACM transactions on computational biology and bioinformatics*, 19(1):614–623, 2020.

[15] Miten Jain, Sergey Koren, Karen H Miga, Josh Quick, Arthur C Rand, Thomas A Sasani, John R Tyson, Andrew D Beggs, Alexander T Dilthey, Ian T Fiddes, et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature biotechnology*, 36(4):338–345, 2018.

[16] Sergey Koren, Brian P Walenz, Konstantin Berlin, Jason R Miller, Nicholas H Bergman, and Adam M Phillippy. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome research*, 27(5):722–736, 2017.

[17] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.

[18] H Li. Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. *arXiv preprint arXiv:1303.3997*, 2013.

[19] Nicholas J Loman, Joshua Quick, and Jared T Simpson. A complete bacterial genome assembled de novo using only nanopore sequencing data. *Nature methods*, 12(8):733–735, 2015.

[20] Robert Vaser, Ivan Sović, Niranjan Nagarajan, and Mile Šikić. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome research*, 27(5):737–746, 2017.

[21] Lauren M Petersen, Isabella W Martin, Wayne E Moschetti, Colleen M Kershaw, and Gregory J Tsongalis. Third-generation sequencing in the clinical laboratory: exploring the advantages and challenges of nanopore sequencing. *Journal of clinical microbiology*, 58(1):10–1128, 2019.

[22] Alexander S Mikheyev and Mandy MY Tin. A first look at the oxford nanopore minion sequencer. *Molecular ecology resources*, 14(6):1097–1102, 2014.

[23] Lucky R Runtuwene, Josef SB Tuda, Arthur E Mongan, and Yutaka Suzuki.

On-site minion sequencing. *Single molecule and single cell sequencing*, pages 143–150, 2019.

[24] Shinnosuke Komiya, Yoshiyuki Matsuo, So Nakagawa, Yoshiharu Morimoto, Kirill Kryukov, Hidetaka Okada, and Kiichi Hirota. Minion, a portable long-read sequencer, enables rapid vaginal microbiota analysis in a clinical setting. *BMC medical genomics*, 15(1):68, 2022.

[25] Sophie Zaaijer, Columbia University Ubiquitous Genomics 2015 class, and Yaniv Erlich. Using mobile sequencers in an academic classroom. *elife*, 5:e14258, 2016.

[26] Cornell Institute of Biotechnology. Oxford nanopore sequencing, 2024. Accessed: October 2024.

[27] Oxford Nanopore Technologies. Minion mk1b sequencer specifications, 2024. Accessed: October 2024.

[28] Sarah S Johnson, Elena Zaikova, David S Goerlitz, Yu Bai, and Scott W Tighe. Real-time dna sequencing in the antarctic dry valleys using the oxford nanopore sequencer. *Journal of biomolecular techniques: JBT*, 28(1):2, 2017.

[29] Max Bloomfield, Samantha Hutton, Charles Velasco, Megan Burton, Miles Benton, and Matt Storey. Oxford nanopore next generation sequencing in a

front-line clinical microbiology laboratory without on-site bioinformaticians. *Pathology*, 56(3):444–447, 2024.

[30] Oxford Nanopore Technologies. MinION Mk1B Specifications, 2024. Accessed: 2024-11-25.

[31] Oxford Nanopore Technologies. MinION Mk1C User Manual, 2024. Accessed: 2024-11-25.

[32] Xiong Ding, Michael G Mauk, Kun Yin, Karteek Kadimisetty, and Changchun Liu. Interfacing pathogen detection with smartphones for point-of-care applications. *Analytical chemistry*, 91(1):655–672, 2018.

[33] Ruth R Miller, Vincent Montoya, Jennifer L Gardy, David M Patrick, and Patrick Tang. Metagenomics for pathogen detection in public health. *Genome medicine*, 5:1–14, 2013.

[34] Vladimír Boža, Broňa Brejová, and Tomáš Vinař. Deepnano: deep recurrent neural networks for base calling in minion nanopore reads. *PloS one*, 12(6):e0178751, 2017.

[35] Adrien Jarretier-Yuste, Chloé Goldsmith, Céline Robardet, Stefan Duffner, and Marc Plantevit. Basecalling by deep learning on minion nanopore sequencing data.

[36] Zhimeng Xu, Yuting Mai, Denghui Liu, Wenjun He, Xinyuan Lin, Chi Xu, Lei Zhang, Xin Meng, Joseph Mafofo, Walid Abbas Zaher, et al. Fast-bonito: A faster deep learning based basecaller for nanopore sequencing. *Artificial Intelligence in the Life Sciences*, 1:100011, 2021.

[37] Peter Perešíni, Vladimír Boža, Broňa Brejová, and Tomáš Vinař. Nanopore base calling on the edge. *Bioinformatics*, 37(24):4661–4667, 2021.

[38] Shuwei Li, Zhiru Guo, Ao Shen, Zheqi Yu, Wei Mao, Shaobo Luo, and Hao Yu. Baseformer: Transformer based base-caller for fast and accurate next generation sequencing. In *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 463–466. IEEE, 2022.

[39] Xuan Lv, Zhiguang Chen, Yutong Lu, and Yuedong Yang. An end-to-end oxford nanopore basecaller using convolution-augmented transformer. In *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 337–342. IEEE, 2020.

[40] Oxford Nanopore Technologies. Sloika: A toolkit for training and deploying neural network basecallers, 2024. Accessed: 2024-11-04.

[41] Oxford Nanopore Technologies. Taiyaki: Tools for training and applying neural network basecallers to oxford nanopore sequencing data, 2024. Accessed:

2024-11-04. Content includes tools for neural network basecaller training and application on Oxford Nanopore sequencing data.

[42] Oxford Nanopore Technologies. Guppy protocol, 2024. Accessed: 2024-11-04.

[43] Samuel Kriman, Stanislav Beliaev, Boris Ginsburg, Jocelyn Huang, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, and Yang Zhang. Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6124–6128. IEEE, 2020.

[44] Mahdieh Abbaszadegan. An encoder-decoder based basecaller for nanopore dna sequencing. 2019.

[45] Vladimír Boža, Peter Perešíni, Broňa Brejová, and Tomáš Vinař. DeepNanoblitz: a fast base caller for MinION nanopore sequencers. *Bioinformatics*, 36(14):4191–4192, 2020.

[46] Jonah Sengupta, Rajkumar Kubendran, Emre Neftci, and Andreas Andreou. High-speed, real-time, spike-based object tracking and path prediction on google edge tpu. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 134–135. IEEE, 2020.

[47] Roksana Hossain, Robinson Mittmann, Ebrahim Ghafar-Zadeh, Geoffery G Messier, and Sebastian Magierowski. Gpu base calling for dna strand sequencing. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 96–99. IEEE, 2017.

[48] Shishir Reddy, Ling-Hong Hung, Olga Sala-Torra, Jerald P Radich, Cecilia CS Yeung, and Ka Yee Yeung. A graphical, interactive and gpu-enabled workflow to process long-read sequencing data. *BMC genomics*, 22:1–8, 2021.

[49] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, et al. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–14, 2023.

[50] Gaofeng Zhou, Jianyang Zhou, and Haijun Lin. Research on nvidia deep learning accelerator. In *2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, pages 192–195. IEEE, 2018.

[51] Abraham Gonzalez, Jerry Zhao, Ben Korpan, Hasan Genc, Colin Schmidt, John Wright, Ayan Biswas, Alon Amid, Farhana Sheikh, Anton Sorokin, et al. A 16mm 2 106.1 gops/w heterogeneous risc-v multi-core multi-accelerator soc

in low-power 22nm finfet. In *ESSCIRC 2021-IEEE 47th European Solid State Circuits Conference (ESSCIRC)*, pages 259–262. IEEE, 2021.

[52] Stephen W Keckler, William J Dally, Brucek Khailany, Michael Garland, and David Glasco. Gpus and the future of parallel computing. *IEEE micro*, 31(5):7–17, 2011.

[53] Ryan R Wick, Louise M Judd, and Kathryn E Holt. Performance of neural network basecalling tools for oxford nanopore sequencing. *Genome biology*, 20:1–10, 2019.

[54] John Nickolls and William J Dally. The gpu computing era. *IEEE micro*, 30(2):56–69, 2010.

[55] William J Dally, Stephen W Keckler, and David B Kirk. Evolution of the graphics processing unit (gpu). *IEEE Micro*, 41(6):42–51, 2021.

[56] Jack Choquette. Nvidia hopper gpu: Scaling performance. In *2022 IEEE Hot Chips 34 Symposium (HCS)*, pages 1–46. IEEE Computer Society, 2022.

[57] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro*, 41(2):29–35, 2021.

[58] Piotr Kluska, Adrián Castelló, Florian Scheidegger, A Cristiano I Malossi, and

Enrique S Quintana-Ortí. Qattn: Efficient gpu kernels for mixed-precision vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3648–3657, 2024.

[59] Tal Ridnik, Hussam Lawen, Asaf Noy, Emanuel Ben Baruch, Gilad Sharir, and Itamar Friedman. Tresnet: High performance gpu-dedicated architecture. In *proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 1400–1409, 2021.

[60] Muthukumaran Vaithianathan, Mahesh Patil, Shunyee Frank Ng, and Shiv Udkar. Comparative study of fpga and gpu for high-performance computing and ai. *ESP International Journal of Advancements in Computational Technology (ESP-IJACT)*, 1(1):37–46, 2023.

[61] Anne C Elster and Tor A Haugdahl. Nvidia hopper gpu and grace cpu highlights. *Computing in Science & Engineering*, 24(2):95–100, 2022.

[62] Leonidas Kosmidis, Iván Rodriguez, Jérôme Lachaize, Jaume Abella, Olivier Notebaert, Francisco Cazorla, and David Steenari. Embedded gpu benchmarking for high-performance on-board data processing. In *European Workshop on On-Board Data Processing (OBDP2019)*, volume 29, 2019.

[63] Jincong Lu and Sheldon X-D Tan. Thermal map dataset for commercial

multi/many core cpu/gpu/tpu. In *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, pages 1–7, 2024.

[64] Yu Emma Wang, Gu-Yeon Wei, and David Brooks. Benchmarking tpu, gpu, and cpu platforms for deep learning. *arXiv preprint arXiv:1907.10701*, 2019.

[65] Jeff Jun Zhang, Tianyu Gu, Kanad Basu, and Siddharth Garg. Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator. In *2018 IEEE 36th VLSI Test Symposium (VTS)*, pages 1–6. IEEE, 2018.

[66] Timothy Prickett Morgan. First in-depth look at google's tpu architecture, April 2017.

[67] Pilsung Kang and Athip Somtham. An evaluation of modern accelerator-based edge devices for object detection applications. *Mathematics*, 10(22):4299, 2022.

[68] Coral products, 2024.

[69] Dennis Agyemanh Nana Gookyi, Michael Wilson, Roger Kwao Ahiadormey, Derek Kwaku Pobi Asiedu, Paul Danquah, and Raymond Gyaang. The efficiency of convolution on gemmini deep learning hardware accelerator. In *2023 IEEE AFRICON*, pages 1–5. IEEE, 2023.

[70] Mahmoud Nazzal, Deepak Vungarala, Mehrdad Morsali, Chao Zhang, Arnob Ghosh, Abdallah Khreishah, and Shaahin Angizi. A dataset for large language model-driven ai accelerator generation. *arXiv preprint arXiv:2404.10875*, 2024.

[71] Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard Mao, Albert Ou, Colin Schmidt, Samuel Steffl, John Wright, Ion Stoica, Jonathan Ragan-Kelley, Krste Asanovic, Borivoje Nikolic, and Yakun Sophia Shao. Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration. In *Proceedings of the 58th Annual Design Automation Conference (DAC)*, 2021.

[72] Zhongpan Wu, Karim Hammad, Ebrahim Ghafar-Zadeh, and Sebastian Magierowski. FPGA-accelerated 3rd generation DNA sequencing. *IEEE Transactions on Biomedical Circuits and Systems*, 14(1):65–74, 2019.

[73] W. Timp, J. Comer, and A. Aksimentiev. DNA Base-calling from a Nanopore using a Viterbi algorithm. *Biophysical J.*, 102(10):L37–L39, May 2012.

[74] J. R. Barry, E. A. Lee, and D. G. Messerschmitt. *Digital Communication*, volume 1. Springer, 3 edition, 2004.

[75] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar

Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, et al. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *IEEE Micro*, 40(4):10–21, 2020.

[76] Krste Asanovic, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, et al. The rocket chip generator. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 4:6–2, 2016.

[77] E Malathy and Chandra Segar Thirumalai. Review on non-linear set associative cache design. *IJPT*, 8(4):5320–5330, 2016.

[78] David R Kaeli and Philip G Emma. Branch history table prediction of moving target branches due to subroutine returns. *ACM SIGARCH Computer Architecture News*, 19(3):34–42, 1991.

[79] Chris H Perleberg and Alan Jay Smith. Branch target buffer design and optimization. *IEEE transactions on computers*, 42(4):396–412, 1993.

[80] Aasheesh Kolli, Ali Saidi, and Thomas F Wenisch. Rdip: Return-address-stack directed instruction prefetching. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 260–271, 2013.

[81] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[82] Abel Beyene, Zhongpan Wu, Yunus Dawji, Karim Hammad, and Sebastian Magierowski. A high-efficiency soc for next-generation mobile dna sequencing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2024. Submitted.

[83] Matthew Jacobsen, Dustin Richmond, Matthew Hogains, and Ryan Kastner. RIFFA 2.1: A Reusable Integration Framework for FPGA Accelerators. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 8(4):22:1–22:23, 2015.

[84] D. Sampietro, C. Crippa, L. Di Tucci, E. Del Sozzo, and M. D. Santambrogio. FPGA-based PairHMM Forward Algorithm for DNA Variant Calling. In *IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 1–8, July 2018.

[85] Sean R. Eddy. Accelerated Profile HMM Searches. *PLoS Computational Biology*, 7(10):e1002195–16, Oct. 2011.

[86] Bespoke Siligon Group. BaseJump: Building the DNA For Open Source ASIC Systems. https://bjump.org, 2023.

[87] Zhongpan Wu and if applicable] [Other Authors. Towards an embedded soc for mobile dna sequencing applications. *IEEE Transactions on Computational Biology and Bioinformatics*, 2024. Submitted for publication.

[88] Di Xu, Peiheng Zhang, Yang Zhang, Shunchen Shi, and Xueqi Li. Transcaller: An end-to-end accelerated transformer-based nanopore basecaller on gpus. In *2023 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 704–711. IEEE, 2023.

[89] Xue-song Tang, Kuangrong Hao, and Hui Wei. A bio-inspired positional embedding network for transformer-based models. *Neural Networks*, 166:204–214, 2023.

[90] Marc Pagès-Gallego and Jeroen de Ridder. Comprehensive benchmark and architectural analysis of deep learning models for nanopore sequencing basecalling. *Genome Biology*, 24(1):71, 2023.

[91] Meryem Banu Cavlak, Gagandeep Singh, Mohammed Alser, Can Firtina, Joël Lindegger, Mohammad Sadrosadati, Nika Mansouri Ghiasi, Can Alkan, and Onur Mutlu. Targetcall: Eliminating the wasted computation in basecalling via pre-basecalling filtering. *arXiv preprint arXiv:2212.04953*, 2022.

[92] Santiago Marco-Sola, Juan Carlos Moure, Miquel Moreto, and Antonio Es-

pinosa. Fast gap-affine pairwise alignment using the wavefront algorithm. *Bioinformatics*, 37(4):456–463, 2021.

[93] William R Pearson and Webb Miller. [27] dynamic programming algorithms for biological sequence comparison. In *Methods in enzymology*, volume 210, pages 575–601. Elsevier, 1992.

[94] Davide Pala. *Design and programming of a coprocessor for a RISC-V architecture*. PhD thesis, Politecnico di Torino, 2017.

[95] Steve Heath. *Embedded systems design*. Elsevier, 2002.

[96] Gabriela Nicolescu and Pieter J Mosterman. *Model-based design for embedded systems*. Crc Press, 2018.

[97] Hasan Genc, Seah Kim, Amid, et al. Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration. pages 769–774.

[98] Henry Cook, Wesley Terpstra, and Yunsup Lee. Diplomatic design patterns: A tilelink case study. In *1st Workshop on Computer Architecture Research with RISC-V*, volume 23, 2017.

[99] Torbjørn Viem Ness. Low power floating-point unit for risc-v. Master's thesis, NTNU, 2018.

[100] Monash University. Dataset. https://bridges.monash.edu/articles/dataset/Population_genomics_of_Klebsiella_pneumoniae_-_supporting_material/9246884, 2024. Accessed: 2024-08-21.

[101] Texas Instruments. Fusion digital power designer gui for isolated power applications, 2014.

[102] Allan Hartstein, Vijayalakshmi Srinivasan, T Puzak, and P Emma. On the nature of cache miss behavior: Is it 2. *The Journal of Instruction-Level Parallelism*, 10:1–22, 2008.

[103] Wikipedia contributors. Power law of cache misses — wikipedia, the free encyclopedia, 2024. Accessed: 2024-10-03.

[104] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Morgan kaufmann, 2017.

[105] Eric Flamand, Davide Rossi, Francesco Conti, Igor Loi, Antonio Pullini, Florent Rotenberg, and Luca Benini. GAP-8: A RISC-V SoC for AI at the Edge of the IoT. In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 1–4. IEEE, 2018.