Delft University of Technology
Master of Science Thesis in Embedded Systems

# Seneca-Lite: Open-source RISC-V based Multi-Core Neuromorphic Platform

**Yashwanth Gopinath**

# Seneca-Lite: Open-source RISC-V based Multi-Core Neuromorphic Platform

Master of Science Thesis in Embedded Systems

Embedded Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

In collaboration with:
IMEC, Netherlands

Yashwanth Gopinath
Student Number: 5731143

August 29 2024

**Author**
Yashwanth Gopinath (Y.Gopinath@student.tudelft.nl)
**Title**
Seneca-Lite: Open-source RISC-V based Multi-Core Neuromorphic Platform
**MSc Presentation Date**
August 29 2024

**Graduation Committee**

| | |
|---|---|
| Thesis Advisor: | Prof. Said Hamidioui, TU Delft |
| Supervisor: | Dr. Rajendra Bishnoi, TU Delft |
| Supervisor: | Gert-Jan Van Schaik, IMEC |
| External Member: | Prof. Ranga Rao Venkatesha Prasad, TU Delft |

## Abstract

Neuromorphic architectures are energy efficient architectures for executing spiking neural networks. Current open-source neuromorphic hardware projects are either experimentation platforms (RANC, ODIN) or neural network accelerators (Open-Spike, SNE), there are no direct processing platforms that support AI and ML applications. Seneca-Lite is an open-source RISC-V based multicore neuromorphic platform. The goal of Seneca-Lite is to enable new possibilities for AI and ML applications and foster more collaboration in this field. The platform is intended for research and academic purposes and furthering the field of neuromorphic computing.

The Seneca-Lite platform utilizes the Ibex core, a highly parameterizable open-source 32 bit RISC-V processor. Each core in the Seneca-Lite platform contains a Network On Chip (NoC) router, local memories, network message FIFOs and interconnects. The multi-core platform is designed to facilitate messaging between cores via the NoC. The NoC used in Seneca-Lite is the same as the NoC used in RANC (Reconfigurable Architecture for Neuromorphic Computing). The number of cores in the system is parameterized and can be controlled by the user depending on their need. Since the platform is open-source, many of the internal parameters (Ibex parameters, FIFO parameters etc) can be tweaked by the user as per their target application.

The completed neuromorphic platform is benchmarked for various state-of-the-art applications and compared to other neuromorphic platforms.

# Preface

This thesis, submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer and Embedded Systems Engineering (CESE) at the Faculty of Electrical Engineering, Mathematics, and Computer Science (EEMCS), Delft University of Technology, represents the culmination of research conducted in collaboration with IMEC Eindhoven from September 2023 to August 2024. This industrial thesis was undertaken under the guidance of Dr. Rajendra Bishnoi and Gert-Jan Van Schaik (IMEC), with Prof. Said Hamdioui serving as the thesis advisor.

The primary objective of this research is to develop an open-source neuromorphic platform, addressing the current gap in such platforms for academic and developmental purposes. Seneca-Lite is an innovative, RISC-V, event-driven platform created using open-source tools and components, designed to be lightweight and user-friendly. Additionally, it aims to reduce energy consumption compared to existing neuromorphic platforms.

This research endeavor has been an immensely enriching experience, providing me with hands-on exposure to all facets of embedded systems design. I have gained valuable insights into various computer architectures, RTL implementations of complex designs, and the integration of system-level components. Throughout this project, I have greatly benefited from the invaluable insights and guidance provided by Amirreza Yousefzadeh, Gert-Jan Van Schaik and Dr. Rajendra Bishnoi, which have been instrumental in all aspects of the research.

Yashwanth Gopinath
Delft, August 2024

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Neuromorphic computing represents a transformative approach to artificial intelligence (AI) and machine learning (ML) that emulates the human brain's architecture and functionality. This field has gained significant traction due to its potential to deliver highly efficient, low-power, and fast processing solutions, which are crucial for real-time applications in edge computing environments [28]. Unlike traditional von Neumann architectures, neuromorphic systems integrate processing and memory storage, thereby reducing latency and energy consumption. Such capabilities make neuromorphic computing ideal for applications in the Internet of Things (IoT), autonomous vehicles, and various sensor-driven environments, where efficient processing and immediate response are paramount [25].

Despite the advancements in neuromorphic computing, there remains a significant research gap in the availability of a fully open-source, scalable neuromorphic platform. Existing platforms are either experimental or focus narrowly on neural network acceleration without addressing broader system-level challenges. The primary objective of this thesis is to design, implement, and verify a multi-core neuromorphic processor that utilizes open-source components and architectures. Seneca-Lite aims to be competitive in performance, energy efficiency, and scalability when benchmarked against other neuromorphic processors.

The motivation behind developing an open-source neuromorphic platform stems from the need to accelerate access to advanced computing resources[25]. Currently, many neuromorphic platforms are either proprietary or focused on specific aspects of neural network acceleration, limiting broader academic and industrial collaboration. By offering an open-source solution, Seneca-Lite aims to facilitate widespread experimentation, innovation, and adoption in the field of neuromorphic computing. This approach can significantly enhance the development of scalable and efficient systems for a wide array of ML and AI applications[13].

Seneca-Lite is designed as an open-source, RISC-V based multi-core neuromorphic platform, optimized for efficiency and scalability. The platform leverages a novel Network-on-Chip (NoC) design to enhance inter-core communication and overall system performance. The choice of RISC-V as the underlying architecture offers several advantages, including extensibility, modularity, and a vibrant open-source ecosystem. Seneca-Lite aims to bridge the gap in existing technologies by providing a fully open-source, scalable neuromorphic platform that can support diverse ML and AI applications.

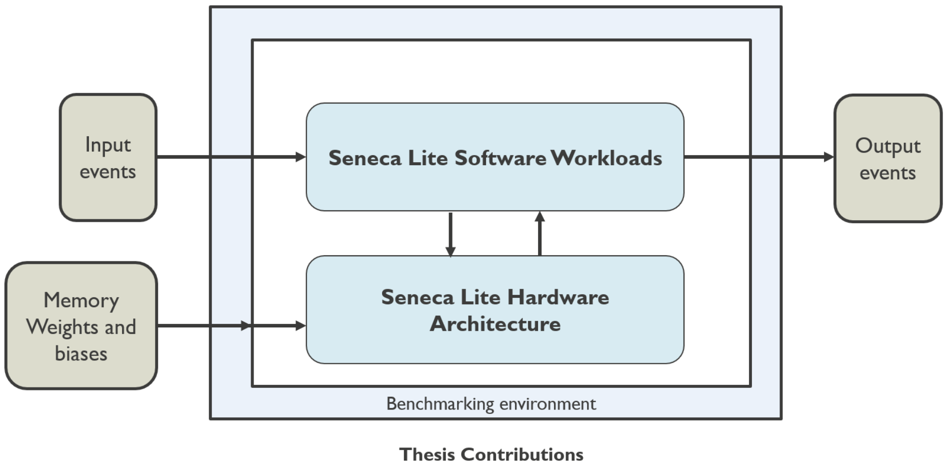

Input events
Memory Weights and biases
Seneca Lite Software Workloads
Seneca Lite Hardware Architecture
Output events
Benchmarking environment
Thesis Contributions

Figure 1.1: **Summary of thesis contributions for Seneca-Lite**

This thesis contributes to the field of neuromorphic computing by introducing an efficient and flexible open-source platform, Seneca-Lite, with a novel optimized router for enhanced NoC communication. In Figure 1.1, we see the software and hardware contributions and also includes input neuromorphic events (sensor inputs), weights and biases for the application. The key contributions include:

- Design and Hardware Implementation: Developing the multi-core architecture of Seneca-Lite.
- Verification: Implementing and verifying the design through simulations and test cases
- Event driven dataflow: The design of multi-core system and communication network is optimised for neuromorphic data flows, suited for ML and AI applications.
- Benchmark with End-to-End Software Workloads: Demonstrating the practical utility of Seneca-Lite through end-to-end applications and ASIC simulations.

The report is structured as follows:

- **Chapter 2 - Background:** An overview of Spiking Neural Networks (SNN), neuromorphic computing paradigm and other relevant technologies used in this research.
- **Chapter 3 - Literature Study:** A comprehensive review of existing neuromorphic architectures and their analysis.
- **Chapter 4 - Proposed Seneca-Lite Architecture:** Detailed description of the proposed architecture and design objectives

- **Chapter 5 - Seneca-Lite Hardware Implementation:** Design choices and hardware implementation steps for Seneca-Lite.
- **Chapter 6 - Results and Discussion:** Presentation of the software benchmarks, synthesis results, and analysis of these outcomes.
- **Chapter 7 - Conclusions and Future Work:** Summarizing the findings, discussing the implications, and outlining potential future research directions.

# Chapter 2

# Background

In this chapter, some of the technology used in the research is explained in detail. Each of these sections correspond to technology used in the design and implementation chapters.

## 2.1 Spiking Neural Networks (SNNs)

Spiking Neural Networks (SNNs) represent a third generation of neural network models, significantly more biologically plausible than traditional artificial neural networks (ANNs) or even deep neural networks (DNNs). Unlike ANNs and DNNs, which rely on continuous activation functions and weighted sums, SNNs introduce the concept of timing into the model, mimicking the way neurons in the brain communicate through discrete spikes or action potentials [11, 18].

In SNNs, the basic computational unit is a spiking neuron [11]. Unlike traditional neurons that produce continuous outputs, spiking neurons emit a spike only when their membrane potential—a value representing the neuron's internal state—exceeds a certain threshold. This spike is then transmitted to connected neurons, influencing their membrane potentials.

- **Membrane Potential:** Each spiking neuron maintains a membrane potential that accumulates input signals over time. Inputs from other neurons arrive as spikes, which can either increase (excite) or decrease (inhibit) the membrane potential [11]. The dynamics of the membrane potential are often modeled using differential equations, such as the Leaky Integrate-and-Fire (LIF) model [16], which describes how the potential decays over time if no input is received .

- **Spike Generation:** When the membrane potential crosses a specific threshold, the neuron generates a spike, which is then propagated to other neurons. After spiking, the neuron undergoes a refractory period during which it cannot spike again, allowing the membrane potential to reset and preventing continuous firing [18].

- **Synaptic Weights:** Similar to traditional neural networks, SNNs use synaptic weights to determine the strength of connections between neurons [11]. These weights influence how much the membrane potential of a neuron is affected by incoming spikes. Learning in SNNs involves adjusting these synaptic weights, often using spike-timing-dependent plasticity (STDP), a biologically inspired learning rule that modifies weights based on the relative timing of spikes from pre- and post-synaptic neurons [18].

### 2.1.1 Neuron Architecture

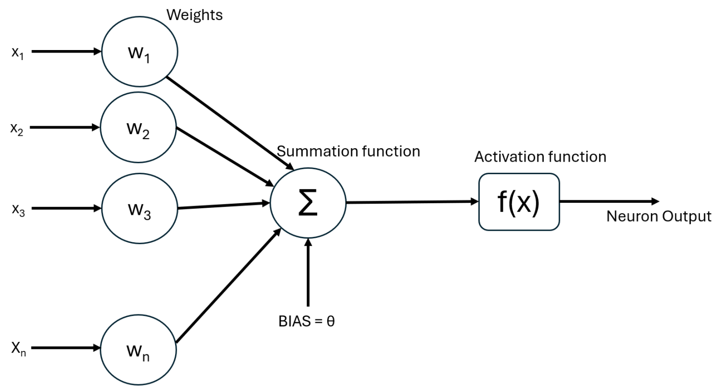


Figure 2.1: **Typical SNN neuron architecture**

Figure 2.1 represents the basic structure of a spiking neural network (SNN) neuron, highlighting the key components involved in processing inputs and generating outputs. Here's a breakdown of each component of the diagram:

- Input Signals (x1,x2,. . . . . . ,xn): These are the inputs to the neuron, typically representing the spikes or action potentials received from other neurons in the network. Each input corresponds to a spike event that can either increase or decrease the neuron's membrane potential.

- Weights (w1,w2,. . . ,wn): Each input signal is associated with a synaptic weight. These weights determine the strength of the connection between the input and the neuron. The weight can be positive (excitatory) or negative (inhibitory), influencing how the input affects the neuron's membrane potential.

- Summation Function: The summation function combines all the weighted input signals and adds a bias ($\theta$). This bias can be seen as a threshold that the neuron needs to surpass to produce an output. The output of this function is the neuron's membrane potential, which is the accumulated signal over time.

- Activation Function (f(x)): The activation function in an SNN is typically modeled by a function that decides whether the neuron will "fire" (produce a spike) based on the membrane potential. If the combined input (the summation) exceeds a certain threshold, the neuron fires and generates an output spike. In traditional neural networks, this might be a sigmoid or ReLU function, but in SNNs, it's more commonly a function that models the spike generation process, such as the Leaky Integrate-and-Fire (LIF) model.

- Neuron Output: If the activation function is triggered (i.e., the neuron fires), the output is a spike that will be transmitted to other neurons in the network. The timing of these spikes carries information, unlike in traditional neural networks where the output is a continuous value.

### 2.1.2 Challenges and Research Directions

Despite their potential, SNNs face several challenges that have limited their widespread adoption compared to traditional neural networks:

- Training Complexity: Training SNNs is more complex than training conventional ANNs. The non-differentiable nature of spike generation poses difficulties for gradient-based optimization methods like backpropagation, which are standard in training ANNs and DNNs. Researchers have developed various approaches to overcome this, including surrogate gradient methods, which approximate the gradient of the spiking neuron model, and biologically inspired learning rules like STDP (spike-timing-dependent plasticity) [23].

- Limited Tooling and Frameworks: While there are a growing number of tools and frameworks for developing SNNs (such as NEST, BindsNET, and SpiNNaker), they are not as widely used or widely supported as those for traditional neural networks. This limits accessibility for researchers and developers.

- Computational Overheads: Although SNNs are energy-efficient in hardware, their simulation on conventional CPUs and GPUs can be computationally expensive due to the need to track the dynamics of membrane potentials and spikes over time. This has spurred interest in specialized neuromorphic hardware that can natively support SNN computations [23].

## 2.2 Neuromorphic Computing

Neuromorphic computing is an emerging field of technology that seeks to replicate the structure and function of the human brain in silicon-based systems. Unlike traditional computing, which relies on the von Neumann architecture where memory and processing units are separate, neuromorphic systems integrate these components, much like the brain's neurons and synapses [13]. The goal is to create machines that can process information more efficiently, adapt to new situations, and learn from their environment with minimal energy consumption. This approach leverages the principles of spiking neural networks (SNNs), where information is transmitted through spikes, or discrete electrical signals, in a manner similar to how biological neurons communicate [23, 13].

However, implementing neuromorphic computing in hardware poses significant challenges. One of the primary difficulties is the need for specialized hardware that can support the parallel, distributed processing nature of neural networks. Traditional CPUs and GPUs, although powerful, are not optimized for the asynchronous, event-driven operations characteristic of neuromorphic systems. This has led to the development of neuromorphic hardware platforms such as IBM's TrueNorth [1], Intel's Loihi [5], and SpiNNaker [10], which are designed to mimic the brain's architecture. These platforms use custom circuits and memory structures to emulate synaptic connections and neural activity, allowing for real-time processing of sensory data and other complex tasks.

Despite these advances, several challenges remain in the development of neuromorphic hardware. First, the scalability of these systems is a major concern. As the number of neurons and synapses in a neuromorphic chip increases, so does the complexity of managing their interactions and ensuring reliable communication between them. Additionally, the power consumption and heat dissipation in densely packed neuromorphic chips must be carefully managed to prevent performance degradation. Another challenge is the lack of standardized development tools and frameworks, which makes it difficult to design, simulate, and optimize neuromorphic systems across different platforms. These hurdles must be overcome to fully realize the potential of neuromorphic computing in applications such as artificial intelligence, robotics, and autonomous systems [13].

## 2.3 Ibex RISC-V Core

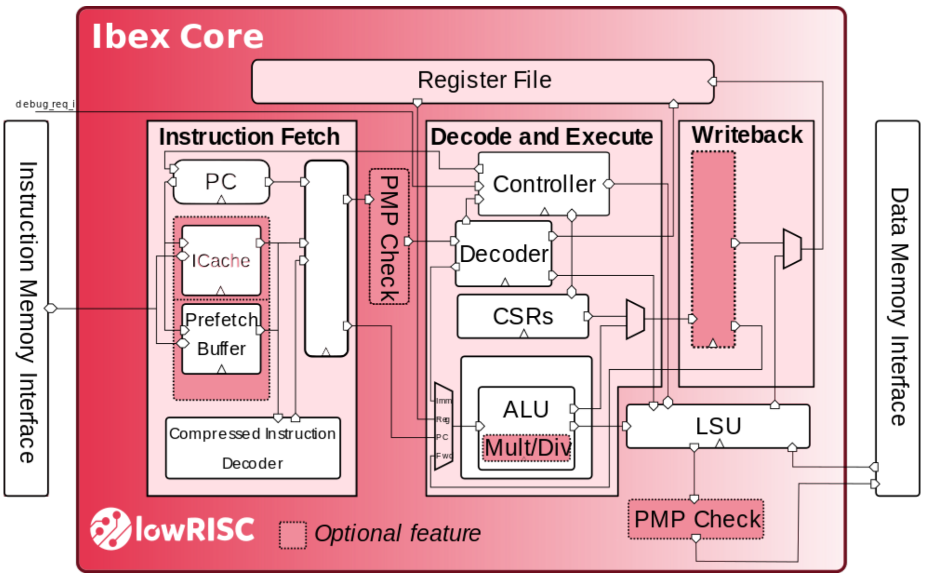


Figure 2.2: **Ibex RISC-V core [17]**

The Ibex RISC-V core is a production-quality, open-source 32-bit processor designed to meet the diverse needs of modern computing applications [17]. Developed by lowRISC, the Ibex core is highly parametrizable, offering flexibility in various configurations and making it an ideal choice for a wide range of applications, including neuromorphic computing platforms. The core's design emphasizes low power consumption, scalability, and efficient performance, aligning well with the requirements of edge computing and AI-driven tasks [25]. A high level block diagram of the Ibex RISC-V core is shown in Figure 2.2. Several key features of the Ibex RISC-V Core can be leveraged in Seneca-Lite.

- Open Source: The Ibex core is open-source, promoting transparency, collaboration, and innovation within the developer community. The source code is available on GitHub, allowing for extensive customization and optimization [17].
- Highly Parametrizable: Ibex supports the RISC-V IMAC instruction set, which includes integer (I), multiplication and division (M), atomic (A), and control and status register (C) operations. This flexibility allows for tailored performance and functionality based on specific application needs. Additionally, the core can be configured with different pipeline stages to balance between performance and area efficiency [6].
- Low Power Configurations: Ibex is designed with power efficiency in mind, making it suitable for energy-constrained environments like edge devices and IoT applications [17]. It incorporates techniques such as clock gating and dynamic voltage scaling to minimize power consumption [6].
- ASIC and FPGA Synthesis Support: The Ibex core supports synthesis for both Application-Specific Integrated Circuits (ASICs) and Field- Programmable Gate Arrays (FPGAs). This versatility allows developers to deploy the core in various hardware environments, facilitating rapid prototyping and deployment [17, 22].
- Successful Tape-Outs: Ibex has been successfully taped out in multiple projects, demonstrating its reliability and readiness for production use. These tape-outs validate the core's design and performance in real-world applications [22, 4].

### 2.3.1 Benefits of Using Ibex in Neuromorphic Platforms

Neuromorphic computing platforms aim to mimic the human brain's efficiency and capabilities, making low power consumption, flexibility, and scalability essential. The Ibex RISC-V core meets these criteria, offering several benefits for neuromorphic applications:

- Energy Efficiency: Neuromorphic platforms require processors that can operate with minimal power consumption. Ibex's low power configurations and energy-saving techniques make it an ideal candidate for such applications [6], ensuring prolonged operation in power-sensitive environments.
- Scalability: The ability to scale the core's performance and area usage through parametrization allows for the development of neuromorphic systems that can grow and adapt to increasing computational demands. This scalability ensures that the platform can handle more complex tasks as they arise.
- Customizability: The open-source nature of Ibex allows for extensive customization to meet the specific needs of neuromorphic applications. Developers can modify the core's functionality, optimize its performance, and add new features to support unique processing requirements.

- Versatile Deployment: Support for both ASIC and FPGA synthesis allows neuromorphic platforms to be rapidly prototyped and deployed in various hardware environments. This flexibility accelerates development cycles and enables quick adaptation to new technological advancements [22, 4].

The Ibex RISC-V core, with its open-source nature, high parametrizability, low power consumption, and support for both ASIC and FPGA synthesis, is an excellent choice for neuromorphic computing platforms. Its features align perfectly with the demands of energy-efficient, scalable, and secure AI applications. By leveraging the capabilities of the Ibex core, developers can build robust and versatile neuromorphic systems capable of advancing the state-of-the-art in AI and machine learning.

## 2.4 Tile Link Interconnects

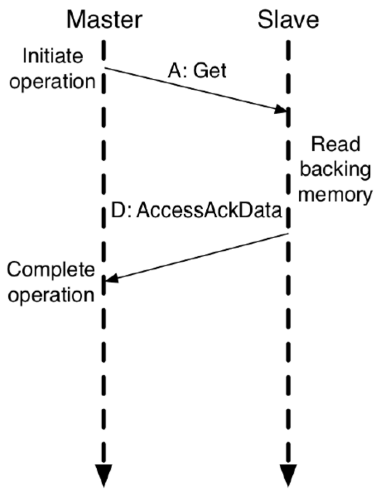


Figure 2.3: **TL-UL interconnect memory operations [20]**

TileLink is a high-performance, cache-coherent interconnect protocol used primarily in RISC-V based systems. It is designed to facilitate communication between processors, memory units, and various peripheral devices. TileLink offers several key features, including support for coherent caches, efficient transaction processing, and scalability, making it an ideal choice for complex system-on-chip (SoC) designs [20].

### 2.4.1 TileLink Uncached Lightweight (TL-UL)

TileLink Uncached Lightweight (TL-UL) is a simplified subset of the TileLink protocol. It is designed for scenarios where full cache coherence is not required, focusing instead on low-latency and efficient data transfer [20]. TL-UL is particularly well-suited for use in applications where power efficiency and area optimization are critical, such as in neuromorphic computing platforms like Seneca-Lite.

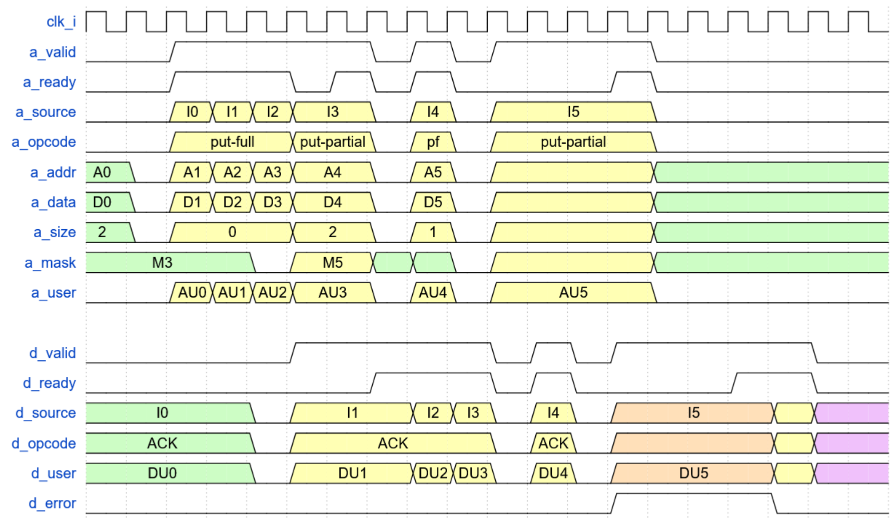


Figure 2.4: **TL-UL interconnect memory operations featuring six write transactions (4 full, 2 partial)[20]**

## Key Features of TL-UL [20]

- Simplified Protocol: TL-UL streamlines the standard TileLink protocol by removing cache coherence mechanisms, which reduces complexity and resource requirements. This simplification leads to more efficient communication in systems where cache coherence is unnecessary.

- Low Latency and High Efficiency: By focusing on uncached transactions, TL-UL minimizes latency and maximizes data transfer efficiency. Additionally, by reducing the overhead associated with cache coherence, TL-UL conserves energy and reduces the silicon area required for implementation.

- Scalability: TL-UL supports a scalable architecture, allowing for easy expansion as system requirements grow. This is crucial for developing neuromorphic systems that may need to scale up to accommodate increasing computational demands.

- Flexibility: The protocol is highly flexible, supporting a wide range of data widths, transaction types, and system configurations. This adaptability ensures that TL-UL can be tailored to meet the specific needs of various applications.

- Open Source: Like the broader TileLink protocol, TL-UL is open source, promoting community collaboration and continuous improvement.

Figure 2.4 details the write transaction of TL-UL. All signals represented with 'a' are the request and the signals represented with 'd' is the response. There are 6 total operations occurring here. The memory write operations are to multiple addresses with its corresponding acknowledgements.

**Advantages of TL-UL in Seneca-Lite**

TL-UL's simplified protocol reduces the design complexity of the interconnect system within Seneca-Lite. This leads to easier integration and faster development cycles. The low-latency and efficient data transfer capabilities of TL-UL enhance the overall performance of the Seneca-Lite platform. This is particularly beneficial for real-time neuromorphic applications that require rapid data processing and response.

By minimizing power consumption, TL-UL aligns well with the low-power requirements of neuromorphic computing. TL-UL's scalable architecture ensures that Seneca-Lite can grow to meet increasing computational demands without significant redesign. This scalability is essential for future-proofing the platform against evolving AI and ML workloads.

The open-source nature of TL-UL allows for extensive customization and TL-UL's compatibility with existing RISC-V ecosystems facilitates seamless integration with other components and peripherals used in the Seneca-Lite platform. This compatibility streamlines development and enhances system coherence.

The TileLink Uncached Lightweight (TL-UL) protocol offers a robust and efficient interconnect solution for the Seneca-Lite neuromorphic platform. Its emphasis on low latency, energy efficiency, and scalability makes it an ideal choice for real-time, power-constrained applications. By leveraging TL-UL, Seneca-Lite can achieve high performance and adaptability, ensuring it meets the demands of advanced AI and ML tasks.

## 2.5 Survey of Open-Source NoCs

- **Split Merge NoC:** The Split Merge NoC features a dynamic routing algorithm that efficiently balances load across the network by splitting and merging data packets. This approach reduces latency and improves throughput but can be complex to implement and may require significant hardware resources [12].

- **Hermes NoC:** Hermes NoC employs a deterministic routing algorithm, ensuring predictable and stable network performance. It is designed for low power consumption and high reliability, making it suitable for safety-critical applications. However, its fixed routing paths may limit flexibility and adaptability to varying workloads[15].

- **FPGA-NoC:** This NoC is introduced as part of the CONNECT architecture, which is customizable and tailored for FPGAs[21]. FPGA-NoC is tailored specifically for FPGA implementations, optimizing the use of FPGA resources while providing flexible and high-performance communication. It supports various topologies and routing algorithms, offering adaptability but potentially at the cost of increased complexity and power usage[21].

- **OpenNoC:** OpenNoC provides an open-source framework for building customizable NoCs. It supports multiple topologies and routing protocols, allowing for extensive optimization. Its open-source nature promotes collaboration and innovation, but it may require substantial effort to achieve specific performance targets[24].

- **FlooNoC:** FlooNoC focuses on fault tolerance and reliability, incorporating mechanisms to handle errors and failures gracefully[7]. It is designed to maintain performance even in the presence of hardware faults, making it ideal for critical applications. However, the added fault tolerance features introduce additional overhead.

- **RANC NoC:** RANC NoC is optimized for simplicity, fault tolerance, and application independence. It utilizes a 2D mesh topology and dimension order routing, ensuring low-latency and efficient communication. Its straightforward design minimizes complexity and power consumption, making it an excellent choice for neuromorphic systems[19].

In the above list, each Network-On-Chip (NoC) is explained in brief. They are compared with one another for use in a neuromorphic system. These comparisons are detailed in Tables 2.1, 2.2 and 2.3. In Table 2.1, the comparison is straightforward. In Table 2.2, we compare scalability, which is the measure of how easily we can extend the NoC to a higher number of cores. Area and energy metrics are stadard comparison. It is important to note here that RANC NoC has high area and energy but there is energy conservation due to its spike based approach[19]. Out of the seven NoCs compared, only the Hermes NoC supports fault tolerance and recovery[15]. This fault recovery mechanism also causes high latency while recovering from a fault. Similarly, FlooNoC has high latency only under heavy load conditions[7]. For all three comparison tables, NA indicates a lack of available information.

| **NoC Technology** | **Network Topology** | **Routing** |
|---|---|---|
| **Split Merge** | Simple 2D Mesh | Dimension order routing(DOR) |
| **Hermes NoC** | Simple 2D Mesh | Dimension order routing(DOR) |
| **FPGANoC** | Simple 2D Mesh | Wormhole Routing |
| **OpenNoC** | Uni-directional deflection Torus (cyclic mesh) | Hot-potato routing |
| **FlooNoC** | 2D mesh (Large data channels) | Dimension order routing(DOR) |
| **RANC** | Simple 2D Mesh | Dimension order routing(DOR) |
| **Binary Tree Based NoC** | Binary Tree | Tree traversal routing |

Table 2.1: **Comparison of different open-source NoCs based on topology and routing**

| **NoC Technology** | **Scalability** | **Area** | **Energy** | **Fault Tolerance** |
|---|---|---|---|---|
| **Split Merge** | Medium | High | High | No |
| **Hermes NoC** | High | Low | Low | Yes |
| **FPGANoC** | Low | Low | Low | No |
| **OpenNoC** | High | Medium | Low | No |
| **FlooNoC** | Low | Low | Low | No |
| **RANC** | NA | High* | High* | No |
| **Binary Tree Based NoC** | Very Low | Low | Low | No |

Table 2.2: **Comparison of different open-source NoCs based on area, scalability, energy consumption and fault tolerance.**

| **NoC Technology** | **Latency** | **Design Standard** | **Performance** |
|---|---|---|---|
| **Split Merge** | Low Latency (High loads) | NA | 3X |
| **Hermes NoC** | High* | NA | 1X |
| **FPGANoC** | Low | Wishbone | 1X |
| **OpenNoC** | Low | AXI4 | 2X |
| **FlooNoC** | Medium* | AXI4 | High |
| **RANC** | NA | Spike-based | NA |
| **Binary Tree Based NoC** | Depends on level in binary tree | NA | Depends on core proximity |

Table 2.3: **Comparison of different open-source NoCs based on latency, design standard, and performance over baseline.**

**RANC NoC: The Optimal Choice for Neuromorphic Systems**
The RANC NoC stands out as the best NoC for neuromorphic systems due to its combination of simplicity and efficiency due to its spike-based design[19]. Neuromorphic systems require robust, low-latency communication to emulate the rapid and parallel processing of the human brain. The RANC NoC's 2D mesh topology and dimension order routing provide the necessary performance while maintaining low power consumption, which is crucial for energy-efficient neuromorphic platforms.

# Chapter 3

# Literature Study

## 3.1 Reconfigurable Architecture for Neuromorphic Computing (RANC)

RANC addresses the growing need for efficient and resilient Network-on-Chip (NoC) architectures tailored for neuromorphic computing systems. Neuromorphic systems, which are designed to mimic the structure and function of the human brain, require specialized communication architectures to handle the high-throughput and low-latency requirements of spiking neural networks (SNNs). Traditional NoC architectures, while effective for general-purpose computing, often fall short in meeting the unique demands of neuromorphic workloads such as energy efficiency, and application awareness [19].

RANC (Reconfigurable Architecture for Neuromorphic Computing) addresses this gap by providing an ecosystem that supports both software simulation and hardware emulation of neuromorphic architectures. It enables researchers to prototype, modify, and optimize neuromorphic designs before committing to expensive silicon fabrication. The ecosystem is composed of highly configurable components that mimic the behavior of biological neurons, allowing researchers to experiment with different architectural configurations and evaluate their performance in real-time.

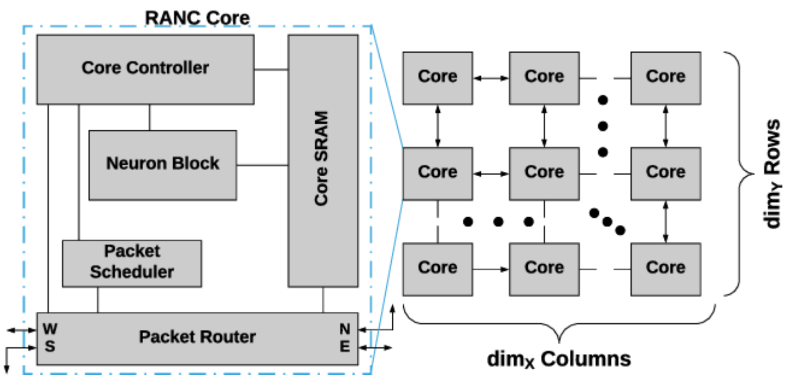


Figure 3.1: **High-level architecture of RANC[19]**

Key features of the RANC architecture include:

- Open-Source: Unlike many other neuromorphic platforms, the RANC architecture is open-source and available to all. The project is maintained and improved constantly by the community.

- Application-Aware Components: RANC allows for the customization of neuron behaviors, network topologies, and data flows based on the specific requirements of an application. This feature makes RANC suitable for a wide range of applications, from traditional SNN-based tasks to non-neuromorphic workloads like vector-matrix multiplication (VMM).

- Scalability and Flexibility: The architecture is scalable, supporting large networks with hundreds of thousands of neurons and synapses. It is also highly flexible, allowing for easy modifications to core components, such as the neuron block, core controller, and packet router, without requiring changes to the entire system.

- Hardware Emulation: The platform includes an FPGA emulation environment, enabling hardware designers to test and validate their designs in a realistic setting. This allows for precise evaluation of factors like resource usage, latency, and energy consumption.

The paper presents a series of experiments that demonstrate RANC's ability to replicate the behavior of IBM's TrueNorth architecture [1, 19]. Through case studies involving the MNIST dataset and EEG data classification, the authors show that RANC can accurately emulate TrueNorth's performance while also providing insights into architectural bottlenecks that could lead to inefficiencies in certain applications [19].

One of the significant advantages of RANC is its ability to perform detailed cycle-by-cycle analysis of neuromorphic applications. For example, the authors illustrate how RANC can be used to optimize the mapping of VMM operations onto neuromorphic hardware, reducing resource usage by eliminating redundant neuron and synapse duplications that are necessary in fixed architectures like TrueNorth [19].

## 3.2 Seneca Neuromorphic Architecture

The Seneca architecture is developed by IMEC. The high level diagram is shown in Figure 3.2.

The Seneca architecture, designed for energy and area-efficient neuromorphic processing, integrates several key components and features tailored for FPGA environments[25]:

- Neuromorphic Cores: Seneca utilizes 8 SIMD-based Neural Processing Elements (NPEs) designed to enhance parallel processing and maximize computational efficiency.

- Hierarchical Memory: It features a hierarchical memory structure that includes registers, local, and global memory, which is crucial for optimizing data reuse and minimizing latency in data access.

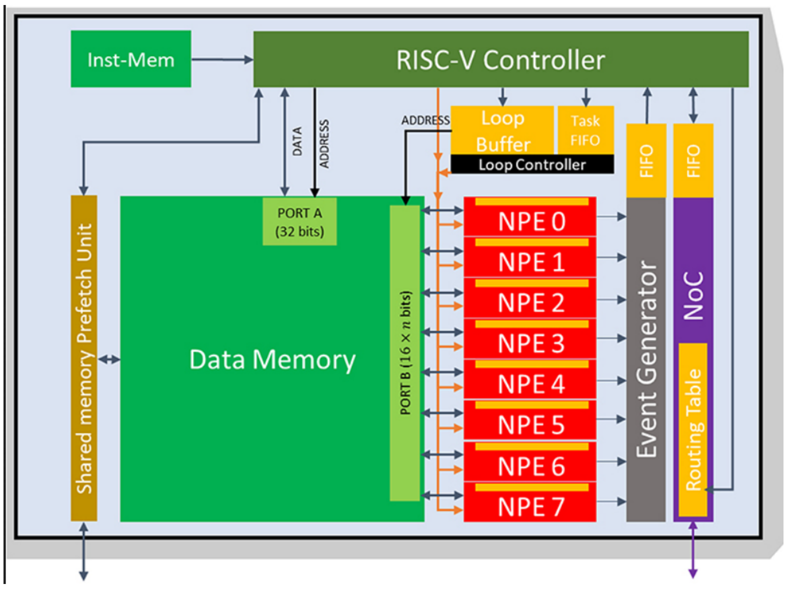


Figure 3.2: **High-level block diagram of Seneca neurmorphic core[25]**

- RISC-V Controller: A core component, the RISC-V processor, is responsible for data pre-processing and the generation of micro-tasks. It effectively manages the execution flow and task distribution across NPEs.
- Loop Controller: This controller ensures that tasks are processed in parallel across the NPEs, enhancing the system's overall throughput and performance.
- Energy and Area Optimization: The architecture is specifically optimized for low power consumption and minimal area usage by leveraging accelerators. This optimization is critical for applications requiring high computational efficiency with constrained power and space.
- Design Flexibility: The Seneca architecture supports various configurations and tuning of parameters to meet specific application needs, making it versatile and adaptable.

This architecture, as illustrated in Figure 3.2, uses advanced FPGA-based design strategies that leverage modern hardware components and software techniques to achieve high performance and efficiency.

## 3.3 Intel Loihi

Intel's Loihi is a neuromorphic research chip designed to mimic the human brain's functionality using artificial neurons and synapses, achieving high efficiency in both energy use and computational power[5]. It represents a significant advance in neuromorphic computing, aiming to provide faster processing, lower power consumption, and on-the-fly learning capabilities that are not feasible with traditional computing architectures.

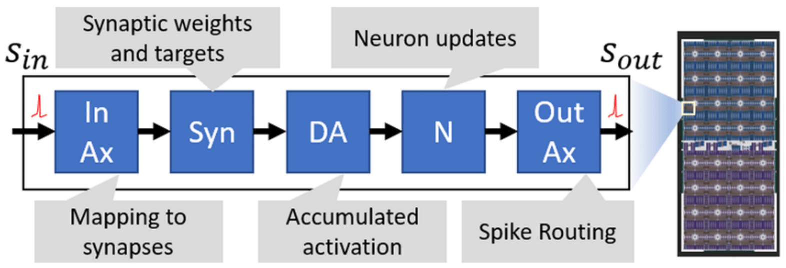


Figure 3.3: **Single core design of Loihi 2 architecture**[14]

**Key Components of Loihi**

- Neuromorphic Cores: Loihi integrates 128 neuromorphic cores. Each core is capable of implementing spiking neural networks, which are inspired by the way neurons in the human brain communicate via spikes[5].
- On-chip Learning: One of the standout features of Loihi is its ability to learn directly on the chip without needing to be pre-trained in a data center. This is achieved through spike-timing-dependent plasticity (STDP), a form of synaptic adaptation that is key to learning in biological brains.
- Asynchronous Design: The chip uses an asynchronous spiking mechanism that allows it to be highly efficient, processing information only when necessary, unlike traditional processors that continuously cycle whether processing is needed or not.
- Energy Efficiency: Loihi is designed to be extremely energy efficient, consuming a fraction of the power used by conventional processors performing similar tasks. This makes it ideal for edge computing applications where power availability is limited.
- Scalability: The architecture of Loihi allows for scaling up by connecting multiple chips to form a more extensive network of neurons and synapses, making it suitable for complex applications requiring vast neural networks.
- Communication: It uses a mesh network that allows for high-speed communication between cores and between chips, facilitating complex and large-scale neural architectures [5, 14].
- Programmability: Despite its specialized nature, Loihi supports flexible programming for a wide range of neuromorphic algorithms, which can be tailored to specific tasks like sensory processing or pattern recognition.
- Integration with External Systems: Loihi can interface with conventional systems via high-speed communication links, making it versatile for integration into broader computing systems or standalone applications in robotics, healthcare, and more.

Intel's development of Loihi underscores a significant shift towards hardware that can support more naturalistic forms of machine learning, akin to human cognitive processes, potentially revolutionizing how tasks are approached in AI research and applications[14]. The next generation of Loihi chips titled Loihi 2 were introduced in 2021.

## 3.4 ODIN Spiking Neural Network (SNN)

The ODIN processor is designed by Charlotte Frenkel at Université catholique de Louvain. Introduced in 2019, ODIN is a digital spiking neuromorphic processor that integrates online learning capabilities directly onto the chip. Fabricated using 28-nm FDSOI CMOS technology, ODIN showcases a compact design, measuring just 0.086 mm$^2$, and is highly energy-efficient with a minimum energy consumption of 12.7 picojoules per synaptic operation[9].

The core of ODIN consists of a 256-neuron 64k-synapse crossbar neurosynaptic core. This design enables ODIN to support spike-driven synaptic plasticity (SDSP), which allows for adaptive learning directly on the chip. The neurons in ODIN can emulate up to 20 different Izhikevich behaviors, making it a versatile tool for simulating cortical spiking neurons.

ODIN's design highlights its potential as a general-purpose experimentation platform for bio-inspired edge computing. It's particularly suited for applications that require efficient, low-power processing of sensory data, such as vision or motor control. Despite its small size, ODIN can be deployed on small-scale FPGAs, allowing for flexible usage in various computational environments[9].

ODIN stands out not only for its technical specifications but also as the first fully open-source neuromorphic chip at its time of publication, promoting transparency and accessibility in neuromorphic research[8, 9].

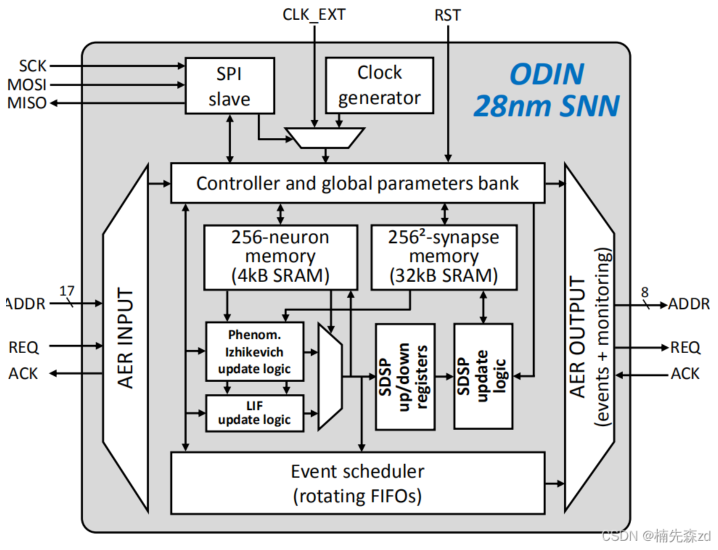

CLK_EXT
RST
SCK
MOSI
MISO
SPI slave
Clock generator
ODIN 28nm SNN
Controller and global parameters bank
256-neuron memory (4kB SRAM)
256²-synapse memory (32kB SRAM)
ADDR
17
REQ
ACK
AER INPUT
Phenom. Izhikevich update logic
LIF update logic
SDSP up/down registers
SDSP update logic
AER OUTPUT (events + monitoring)
8
ADDR
REQ
ACK
Event scheduler (rotating FIFOs)

Figure 3.4: **Block diagram for 256 neuron ODIN neuromorphic processor [9]**

# Chapter 4

# Proposed Seneca-Lite Architecture

This chapter discusses the design objectives of the Seneca-Lite architecture and the choices made during its hardware implementation. Both the hardware implementation and software workloads are considered while designing the architecture.

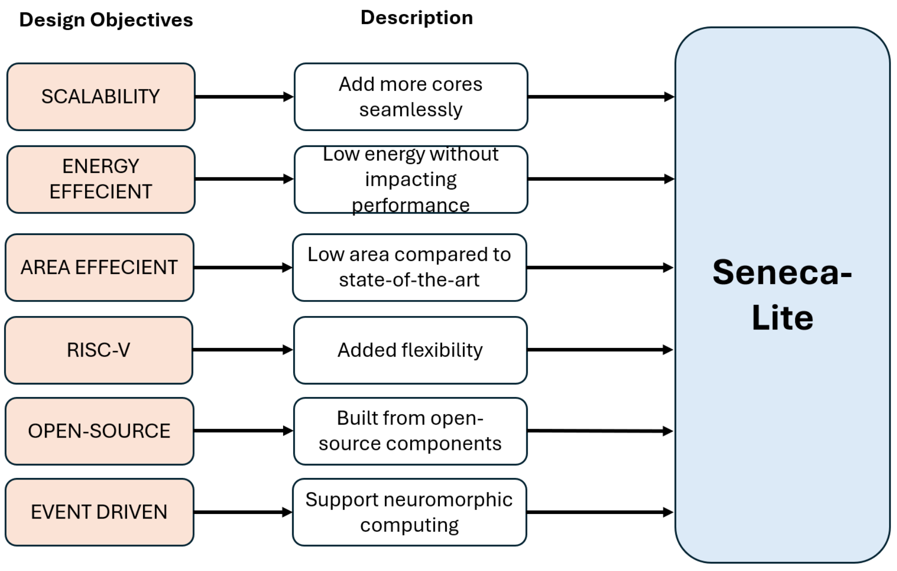
## 4.1 Seneca-Lite Design Overview

The Seneca-Lite is designed with some key objectives in mind, these include:

- Scalable: The design should be scale with an increase in cores. This allows support for larger ML and AI applications without compromising functionality.
- Energy/Area Efficient: Seneca-Lite is meant to be a lightweight neuromorphic platform. Design choices should align to make the design more area and energy efficient.
- RISC-V Based: The key processing core used in Seneca-Lite design should use the RISC-V instruction set architecture, which is preferred due to its flexibility [25].
- Open-Source: The components used in the design should be open-source. To ensure collaboration and experimentation, the design should be accessible to all.
- Event Driven Architecture: Since the goal is to design a neuromorphic platform, the architecture should adhere to event driven design subsection 4.1.1 explains the event driven data flow in detail.

### 4.1.1 Event Driven Data Flow

The Seneca-Lite architecture should employ an event-driven data flow to optimize the processing efficiency and power consumption for neuromorphic comput-



Design Objectives
Description
SCALABILITY
Add more cores seamlessly
ENERGY EFFECIENT
Low energy without impacting performance
AREA EFFECIENT
Low area compared to state-of-the-art
RISC-V
Added flexibility
OPEN-SOURCE
Built from open-source components
EVENT DRIVEN
Support neuromorphic computing
Seneca-Lite

Figure 4.1: **Seneca-Lite design objectives**

ing tasks. This methodology ensures that the system processes data only when specific events occur, reducing unnecessary energy expenditure and latency[13].

As Seneca-Lite is a multicore platform, we can divide the data flow into two categories, event driven data flow in a single core and event driven data flow through the Network On Chip (NoC).

### Event Driven Data Flow at Core Level

In a single-core setup, the data flow begins with input events, which are typically sensor data or incoming signals that need processing. The core components of this data flow include:

- Input Handling:
  - Event Detection: Sensors or external interfaces detect events, triggering the processing core.
  - Pre-Processing: Initial data handling is performed to filter and format the data for efficient processing.
- Core Processing:
  - Task Acceptance/Forwarding: The RISC-V processor in each core accepts the data and processes it or it forwards the packet to the appropriate core via the NoC.
  - Parallel Processing: Multiple neurosynaptic processing elements (NPEs) execute tasks in parallel, leveraging the architecture's inherent parallelism.
- Output Generation: This includes the post processing of data and it should formatted in the expected manner.

**Network on Chip (NoC) Neuromorphic Flow in Multi-Core Design**

The NoC in Seneca-Lite is pivotal for facilitating communication between multiple cores, ensuring data can be shared and processed efficiently across the system.

- Event Propagation: An event detected by one core can propagate through the NoC to other cores that need to process related data.
- Routing: The NoC routing should direct data packets to the appropriate cores. This method ensures low latency and efficient data transfer.
- Packet Creation: Data from the initiating core is encapsulated into packets containing both the payload and routing information.
- Transmission: These packets traverse the NoC, which can manage data flow and prevent bottlenecks.
- Aggregation: Results from multiple cores are aggregated as needed to produce the final output. This can be done in a designated core or using an external handler.
- Output Handling: The aggregated data is then transmitted to the designated output interfaces, completing the event-driven processing cycle.

## 4.2 Neuromorphic Architecture

Similar to the data flow, the architecture of the design can be divided into two sections, core level architecture and system level architecture. The system level architecture includes multiple single cores connected via the Network On Chip (NoC).

### 4.2.1 Core Level Seneca-Lite Architecture

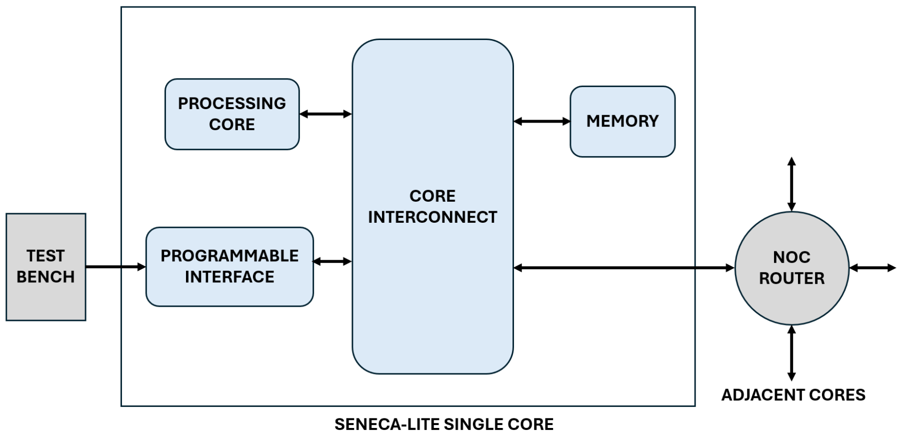


Figure 4.2: **Seneca-Lite single core proposed design**

Figure 4.2 showcases the proposed architecture for a single core of Seneca-Lite. The key components and interconnections are highlighted below:

- Processing Core: A central processing unit which is responsible for executing tasks and processing data. The central processing unit is also responsible for creating packets to send to other cores.

- Memory: Local memory unit used for storing data and instructions needed by the processing core. High-speed access and sufficient capacity to ensure smooth data flow and processing efficiency.

- Programmable Interface: Interface to allow programmability and customization of core functionalities. It should have control over both data and instruction memory.

- Core Interconnect: An internal interconnect system facilitating communication between the processing core, memory, and programmable interface. High-bandwidth, low-latency connections to ensure efficient data transfer and synchronization.

- NoC (Network on Chip) Node: Node facilitating communication with adjacent cores in a multi-core system. This module should be scalable to accommodate additional cores without significantly impacting performance.

The modules and the core interconnections are for specific purposes and should be designed with the purpose in mind.

- Processing Core to Memory: Direct, high-speed connection to enable quick access to stored data and instructions.

- Processing Core to Programmable Interface: Bi-directional communication to allow the core to send, receive and process messages over the NoC. Bi-directionality is beneficial for acknowledgement of operations.

- Core Interconnect to NoC Node: Seamless integration to ensure data can be efficiently transferred to and from adjacent cores.

Additionally, the architecture should be designed to support easy scalability, allowing additional cores to be integrated into the NoC with minimal impact on overall performance. Design emphasis should be on power efficiency, with components and interconnections optimized for low power consumption without compromising processing capabilities.

### 4.2.2 System Level Architecture

Seneca-Lite is a scalable and flexible platform. The number of cores in the design depends on the requirement of the application workload. All individual cores function as one single unit as detailed in subsection 4.2.1. The inputs and outputs to each cores are through the Network on Chip(NoC).

### Network On Chip (NoC)

The Network on Chip is the most critical aspect in terms of latency and power consumption. The chosen NoC needs to be flexible, open-source and scalable with number of cores. Figure 4.3 is an example of a 3x3 9-core Seneca-Lite design.

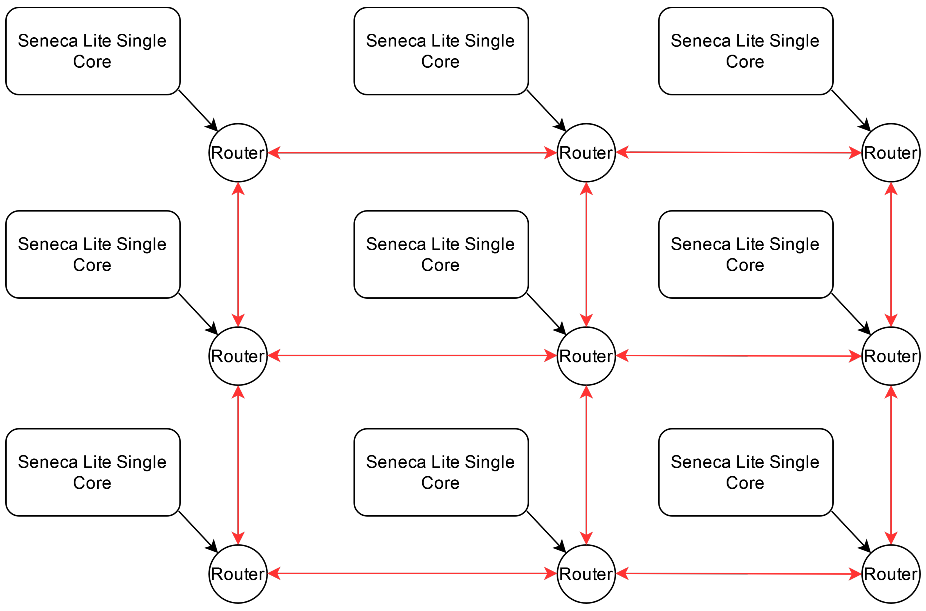


Figure 4.3: **Seneca-Lite 3x3 proposed architecture**

The Network on Chip has many key components and they each have their own set of requirements for Seneca-Lite:

- NoC Nodes: Each NoC node acts as a communication hub for its corresponding processing core and must support routing, data packet handling, and inter-core communication.
- Routing Algorithm: It should implement an efficient routing algorithm to ensure low-latency communication.The algorithm should be robust, capable of handling high traffic and minimizing congestion.
- Data Packet Handling: Each NoC node should manage data packets, including encapsulation, transmission, reception, and decapsulation.
- Interconnect Fabric: A high-bandwidth, low-latency interconnect fabric connecting all NoC nodes. It should support concurrent data transfers to prevent bottlenecks.
- Synchronization: Ensure proper synchronization between nodes to manage data consistency and timing with mechanisms for clock synchronization across the NoC.
- Modular Design: The NoC should be modular, allowing easy addition of nodes to scale the system as needed.

Based on the design objectives detailed in the previous sections, the components for Seneca-Lite were selected through a comprehensive literature study. Various options for each component were surveyed and compared. The comparison analysis and the rationale behind the selection of specific components are explained in chapter 3. The features outlined in this chapter enable Seneca-Lite to support a wide range of AI and ML applications effectively.

# Chapter 5

# Seneca-Lite Hardware Implementation

## 5.1 Single Core Seneca-Lite Hardware Implementation

The Seneca Lite single core architecture is designed to efficiently handle neuromorphic computing tasks by integrating key components and ensuring seamless data flow between them. This detailed explanation covers the important connections and functions of each module in the design, as illustrated in Figure 5.1.

The single core architecture consists of the following primary components:

- Ibex RISC-V Core
- TileLink Uncached Lightweight (TL-UL) Crossbar (XBAR)
- Memory Units (Data Memory and Instruction Memory)
- NoC Interface
- FIFO Buffers
- NoC Router

Each of these components plays a crucial role in ensuring the efficient operation of the Seneca Lite single core.

#### Ibex RISC-V Core

The Ibex core is the central processing unit responsible for executing instructions and processing data. It is a highly parametrizable, low-power 32-bit RISC-V processor. It is used in the IMC configuration, supports integer, multiplication and control/status instructions. The Ibex core is critical for executing the neuromorphic algorithms and managing data processing tasks efficiently[17].

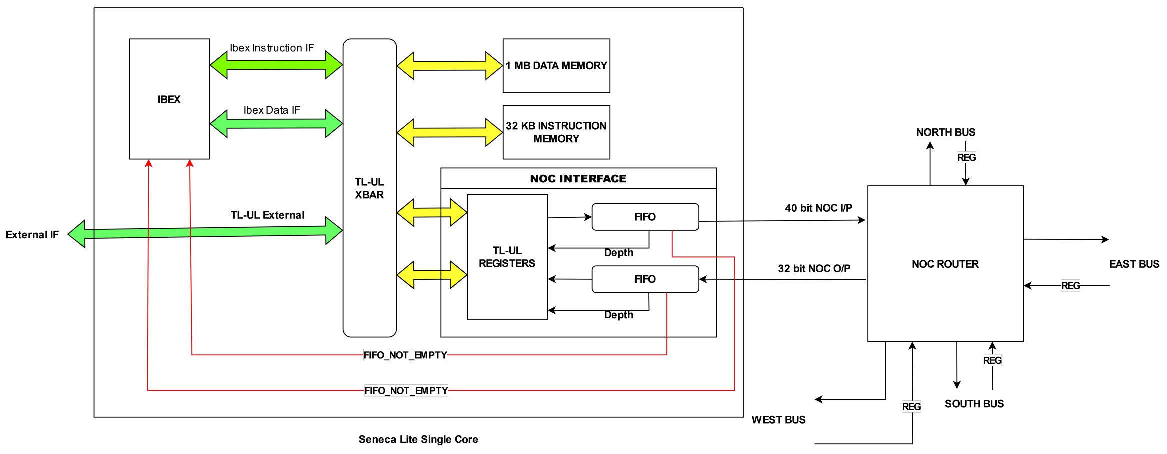


Figure 5.1: **Seneca-Lite single core hardware architecture**

### TileLink Uncached Lightweight (TL-UL) Crossbar (XBAR)

The TL-UL XBAR acts as an interconnect fabric, facilitating communication between the Ibex core, memory units, NoC interface, and other peripherals. It supports the TL-UL protocol, which is designed for low-latency, efficient data transfer without the need for cache coherence.

The TL-UL XBAR ensures seamless data flow between the core and other components, reducing latency and improving overall system efficiency. Memory Units

The memory units consist of: 1 MB Data Memory: Used for storing data required during computation and 32 KB Instruction Memory: Stores the instructions to be executed by the Ibex core. The memory sizes are determined using the minimum requirements of the software workload. The instruction memory hold the instructions decoded from the software and the data memory holds the weights and biases of the application.

| **Intra-core Devices** | **Base Address** | **Size Byte** |
|---|---|---|
| Instruction Memory | 0x00000000 | 0x08000 (32KB) |
| Data Memory | 0x100000 | 0x100000 (1024KB) |
| NoC Interface | 0x200000 | 0X8000 (32KB) |

Table 5.1: **Address ranges for single core TL-UL interface devices**

Table 5.1 lists the various device interfaces present for the single core Seneca-Lite design. All TL-UL interfaces are either a host or a device interface as described in section 2.4. The host interfaces in a single core are the external interface, Ibex data and instruction interfaces. All the devices in Table 5.1 are connected to all host interfaces.

### NoC Interface

The NoC (Network on Chip) interface connects the single core to the broader NoC, enabling communication with other cores in a multi-core setup. It includes

FIFO buffers and TL-UL registers.

- FIFO Buffers: These buffers are used to temporarily store data packets before they are transmitted across the NoC. They help in managing data flow and preventing data loss during transmission. Each FIFO has a depth of 4 which can be changed in the top level file.
- TL-UL Registers: These registers hold the control and status information required for TL-UL transactions, ensuring proper communication protocol adherence.

The NoC interface, along with the FIFO buffers and TL-UL registers, ensures reliable and efficient communication between the core and other components in the network, facilitating scalable and flexible system design.

**Connections and Data Flow**

- Ibex to TL-UL XBAR: The Ibex core communicates with other components via the TL-UL XBAR. This connection ensures that data and instructions can be fetched from memory and processed efficiently.
- TL-UL XBAR to Memory Units: The XBAR routes data and instruction requests from the Ibex core to the respective memory units, ensuring that the core has access to the necessary resources for computation.
- TL-UL XBAR to NoC Interface: The XBAR also connects to the NoC interface, enabling the core to send and receive data packets across the NoC. This connection is crucial for inter-core communication and data sharing in a multi-core setup.
- NoC Interface to NoC Router: The NoC interface sends data packets to the NoC router, which routes them to their destination within the network. This ensures efficient communication and data transfer across the network.

The Seneca-Lite single core architecture, with its integration of the Ibex RISC-V core, TL-UL XBAR, memory units, NoC interface, FIFO buffers, and NoC router, is designed to efficiently handle neuromorphic computing tasks. Each component and connection is optimized to ensure low-latency, efficient data transfer, and scalable performance, making it an ideal choice for advanced AI and ML applications.

### 5.1.1 Control Mechanism

In the Seneca-Lite architecture, maintaining configuration control from the top level is crucial. A central component of this architecture is the Ibex processing core, which plays a pivotal role in the single-core configuration. All instructions intended for execution are loaded into the instruction memory, from which the Ibex core sequentially executes each instruction. As detailed in Table 5.1 , the Ibex core accesses the instruction memory using the base memory address 0x00000000.

During the process of loading instructions into the instruction memory, it is imperative to place all cores into a reset mode to prevent them from executing any instructions prematurely. To facilitate this, a configuration register located at the top level is provided, which grants access to the reset control of each individual core. By writing to the control register at address 0x600000, we can assert control over the reset state of the cores. Writing all 1's to this register will place all cores into reset mode, effectively halting instruction execution. Conversely, writing 0's will release the cores from reset, enabling them to execute instructions. Additionally, this mechanism allows for selective control, wherein some cores can remain in reset mode while others are permitted to execute instructions.

## 5.2 Network on Chip (NoC) Router

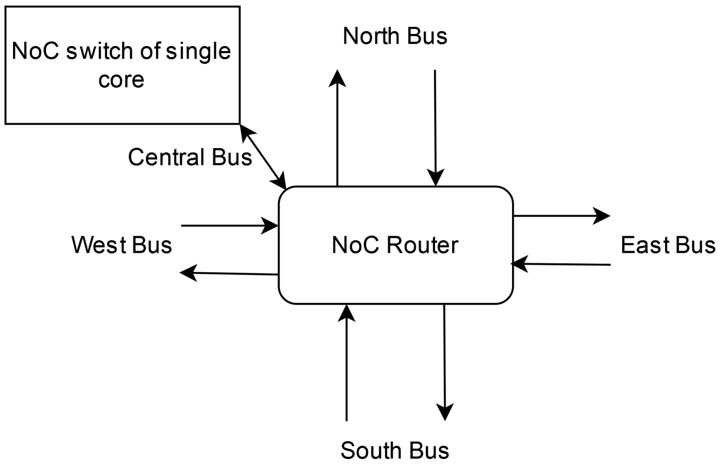


Figure 5.2: **Seneca-Lite NoC router design**

The Network-on-Chip (NoC) router is a fundamental component in the Seneca-Lite architecture, responsible for facilitating efficient communication between multiple cores within the system. As the name suggests, the NoC router acts as a central hub that directs data packets between different processing cores, ensuring that information is transmitted accurately and efficiently across the chip.

The primary role of the NoC router is to manage data traffic within the NoC. In a multi-core system like Seneca-Lite, each core performs specific tasks and needs to communicate with other cores to share data, synchronize processes, or distribute workloads. The NoC router handles these communications by receiving data packets from one core and routing them to the appropriate destination core.

The NoC router is connected to multiple other NoC routers, each associated with a single core. These switches act as interfaces between the core and the NoC, packaging data into packets that the NoC router can process. The router uses dimension-order routing (e.g., XY routing in a 2D mesh topology) to determine the most efficient path for each packet to reach its destination. This router design is the same as the one used in RANC [19]. The psudo-code for the routing is shown in Figure 5.3.

```
function ROUTE(packet, core, dx, dy)
    if dx < 0 then
        route(packet, core.east, dx+1, dy)
    else if dx > 0 then
        route(packet, core.west, dx−1, dy)
    else if dy < 0 then
        route(packet, core.south, 0, dy+1)
    else if dy > 0 then
        route(packet, core.north, 0, dy−1)
    else
        core.accept(packet)
    end if
end function
```

Figure 5.3: **Psudocode for Dimension Order Routing [19]**

- Input Buffers: These buffers temporarily store incoming data packets from the NoC switches before they are processed by the router. This helps to manage data flow and prevent congestion within the NoC.
- Routing Logic: The routing logic determines the best path for each data packet based on the destination address. It ensures that packets are routed through the network with minimal delay for 2D mesh topology (Figure 5.3).
- Crossbar Switch: The crossbar switch connects the input and output ports of the router, enabling the simultaneous transmission of multiple data packets. This component is critical for maintaining high throughput in the NoC.
- Output Buffers: Once the routing logic determines the path, the data packets are stored in output buffers before being sent to the next NoC router or the destination core.

In Seneca-Lite, the NoC router is crucial for maintaining the system's performance and scalability. As a neuromorphic computing platform designed for efficient and parallel processing, Seneca-Lite relies heavily on the NoC router to manage data-intensive tasks without compromising speed or energy efficiency.

The NoC router enables seamless communication between cores, allowing the system to scale effectively as more cores are added. This is particularly important in neuromorphic systems, where processing elements must frequently exchange data to simulate neural networks.

Additionally, the router's ability to efficiently manage data traffic contributes to the overall energy efficiency of the Seneca-Lite system. By minimizing the distance data packets need to travel and reducing congestion, the NoC router helps to lower power consumption, which is a critical factor in the design of energy-efficient neuromorphic systems.

### 5.2.1 NoC Packet Structure

The packet structure depicted in Figure 5.4 consists of two main fields: an 8-bit header and a 32-bit data payload.

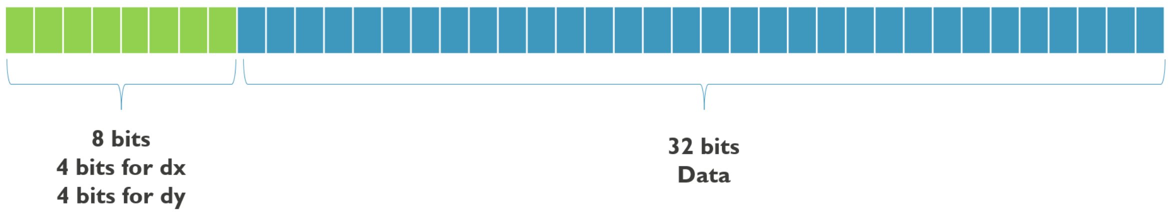


Figure 5.4: **Packet structure for the NoC hardware implementation**

- **Header (8 bits):** The first 8 bits of the packet are divided into two 4-bit segments for routing as explained in Figure 5.3.
  - 4 bits for dx: These bits represent the horizontal (x-axis) offset or distance between the current core and the destination core in a network. The value of dx determines the direction the packet should be routed horizontally.
  - 4 bits for dy: These bits represent the vertical (y-axis) offset or distance between the current core and the destination core. The value of dy determines the direction the packet should be routed vertically.
- **Data (32 bits):** The remaining 32 bits of the packet contain the actual data payload that needs to be delivered to the destination core. This data can represent any kind of information that the packet is meant to carry across the network.

The only constraint by the packet structure is the limitation of 4 bits for dx and dy. This restricts our range of values for dx and dy to -8 to +7 and it limits routing to 7 cores north, east and 8 cores south and west. In the design, all TL-UL buses are 32-bit wide and to accomodate the 40 bit packet structure, we write to temporary registers that correspond to dx and dy.

## 5.3 System Level Hardware Implementation

### 5.3.1 Module Hierarchy

In the previous sections we discussed in detail about the single core and its functionality. The completed single core Seneca-Lite design consists of the Ibex core, NoC interface and memory. Multiple single cores connected together form a network of processing neuromorphic cores which form the Seneca-Lite architecture.

Figure 5.5 shows the distinction between modules and their hierarchy. Each NoC Node consists of the Seneca-Lite core and the NoC router corresponding to the core. The NoC node is the key module used for all interconnections in the top level.

### 5.3.2 Top Level NoC connections

One of the key design objectives is ensuring scalability, which is achieved through the top-level interconnections by allowing a customizable number of cores. The

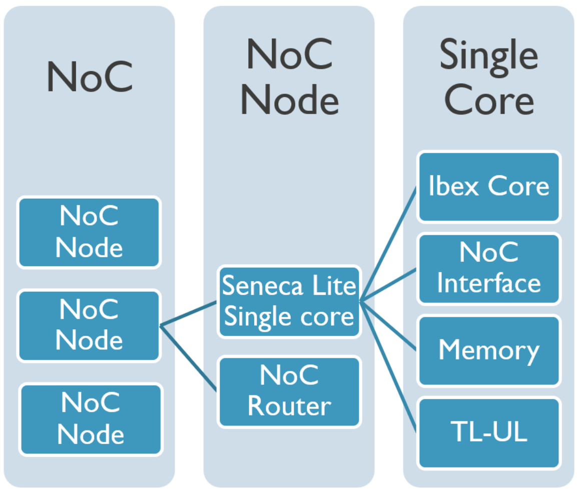


Figure 5.5: **Module hierarchy for Seneca-Lite architecture**

top-level file defines the architecture's rows and columns as parameters. Based on the workload requirements, the architecture can be tailored to any desired number of rows and columns.

The interconnections facilitating this customization are created dynamically. These top-level connections are established using a generate block in Verilog. When connecting the different cores, it is crucial to properly name and categorize them for future reference. All cores fall into one of three categories:

- Corner Case: These cores are positioned at the corners of the architecture. Their connections are limited to only two directions, with all other router connections being unused and inactive in these cores.
- Edge Case: These cores are located along the edges of the architecture. They have one connection that is invalid, while the remaining three connections are part of the grid, linking to other cores.
- Normal Case: These cores are situated in the middle of the architecture, where connections from all four sides are valid.

To access these processing elements from the NoC we use the external interfaces that are a part of each Seneca-Lite single core.

### 5.3.3 4-core architecture

In the previous subsection, we discussed the customization of the number of cores. In this section, we will explore a template architecture created for the KeyWord Spotting (KWS) Benchmark, which is explained in detail in subsection 6.3.1.

The system consists of 2 rows and 2 columns interconnected via the NoC (Network on Chip). All cores in this architecture are corner cases, meaning they have only two valid NoC connections. The architecture is illustrated in detail in Figure 5.6. The Seneca-Lite single core, combined with the router, forms the NoC Node. These NoC Nodes are interconnected and are represented in red in the figure. Several additional components are included for enhanced functionality:

- I/O Module: The input/output module is used for streaming in input packets and reading the output. It is connected to the first core directly through the NoC Router.

- External Interfaces XBAR: The external TL-UL interfaces of all the cores are connected to a common TL-UL crossbar. The XBAR allows us to control all device memories using one direct connection. To access each core externally, we use the address mapping as shown in Table 5.2.

| **DEVICES** | **BASE ADDRESS** | **SIZE BYTE** |
|---|---|---|
| CORE 0 | 0X000000 | 0X8000 (32KB) |
| CORE 1 | 0X100000 | 0X8000 (32KB) |
| CORE 2 | 0X200000 | 0X8000 (32KB) |
| CORE 3 | 0X300000 | 0X8000 (32KB) |
| I/O Module | 0X400000 | 0X8000 (32KB) |

Table 5.2: **Address ranges for different cores in a 4-core Seneca-Lite architecture**

- Zynq Processor: Developed by Xilinx, Zynq is a unique System on chip that integrates ARM processing with FPGA(Field Programmable Gate Array) fabric. Due to its high performance, versatility and flexibility. This processor allows us complete control over the rest of the architecture.

- AXI to TL-UL interface: The Zynq processor uses the AXI(Advanced eXtensible Interface) standard. To connect it to our system we connect the main TL-UL host to the AXI of Zynq. The Zynq serves as the host interface for all debugging and software tasks.

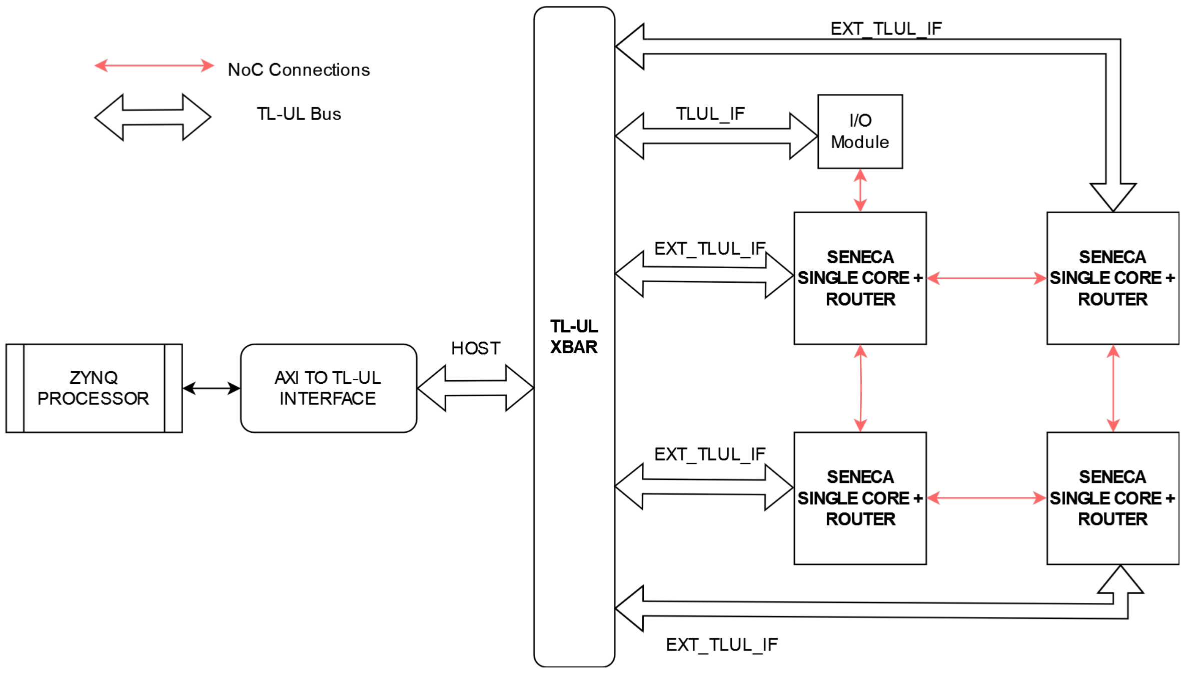


Figure 5.6: **Seneca-Lite top level diagram for a 4-core architecture**

# Chapter 6

# Results and Discussion

## 6.1 Experimental Setup

In the Seneca-Lite architecture, the main processing cores are the Ibex RISC-V cores. The software for each core is the set of instructions that runs on the Ibex RISC-V cores. These cores are encapsulated with memory and interconnect and connect to each other via the NoC (Network On Chip). The steps followed for loading software and starting the architecture is the same. The flow starts by temporarily loading the testbench with weights and biases. These can be changed to any other metric used for computation as well.

If the precompiled software is not present, then we use the RISC-V Compiler Toolchain provided by lowRISC[17]. Once we have the resulting bitfile, we can set all the cores in reset using the configuration registers discussed in subsection 5.1.1. Once the cores are in reset, we start loading the weights, biases and software one core at a time. Once all the cores are loaded the cores are taken off reset and the instructions are executed.

All hardware results are obtained from post synthesis gate level simulation. For simulation of the design, both pre- and post-synthesis was carried out using Xilinx Vivado and NCSim. Power measurement was done using Cadence JOULES. The clock frequency used for the entire simulation was 500MHz.

## 6.2 Seneca-Lite Single Core Results

Seneca-Lite single core was synthesized as a single unit and both synthesis and energy results were obtained from this experiment. Since Seneca-Lite has a NoC as well. Each NoC router is considered as a part of its respective single core.

All performance and power results of Seneca-Lite were obtained using Cadence's toolset[2]. The RTL simulation was done using the NCSim toolchain and Cadenca JOULES was used for average power consumption and energy metrics for specific workloads.

FPGA related architecture and metrics were obtained using Xilinx Vivado with Zynq UltraScale+ RFSoC ZCU111 Evaluation Kit[26] as the target platform.

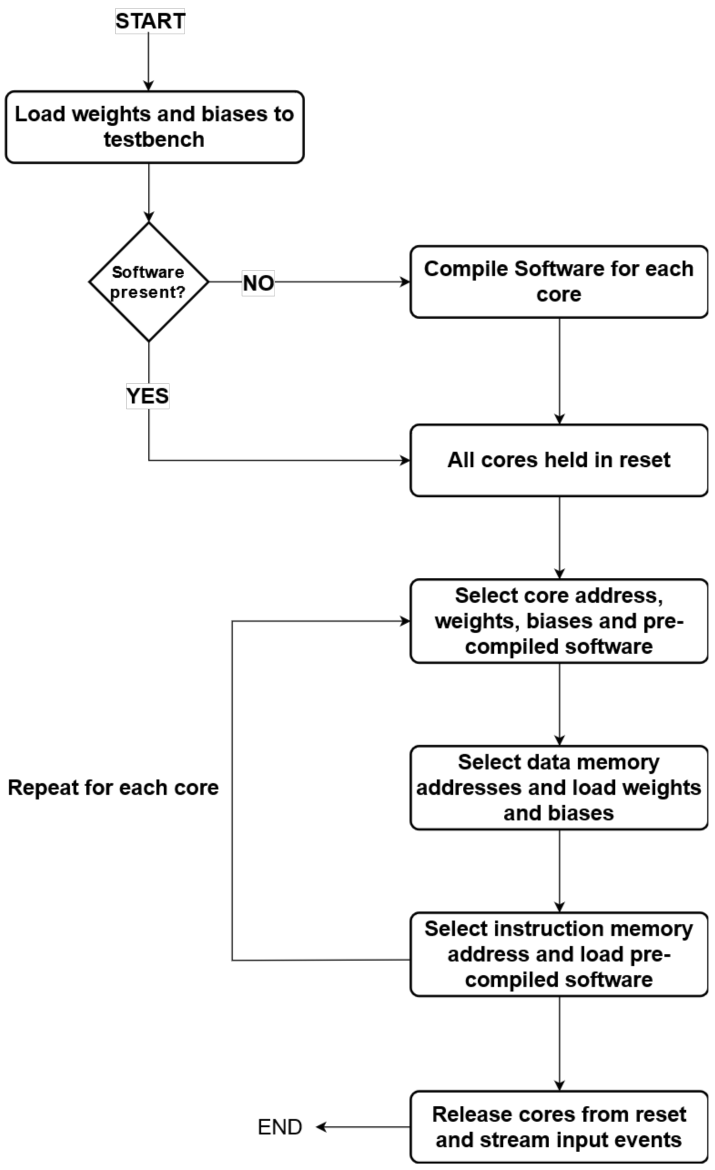

START
Load weights and biases to testbench
Software present?
NO
Compile Software for each core
YES
All cores held in reset
Select core address, weights, biases and pre-compiled software
Repeat for each core
Select data memory addresses and load weights and biases
Select instruction memory address and load pre-compiled software
END
Release cores from reset and stream input events

Figure 6.1: **Control Flow for the RISC-V processing cores in Seneca-Lite architecture**

### 6.2.1 Synthesis Results

The Seneca-Lite single core consists of both data and instruction memory, the Ibex processing core, NoC interface and the NoC router and the intra-core interconnects.

The single core area is calculated for GF-22nm FDX technology node. The JOULES area calculations provide the single core area as:

$SingleCoreArea = 0.0924mm^2$

The FPGA synthesis results of the Seneca-Lite single core module are listed in Table 6.1. The breakdown between different modules is detailed in Table 6.2. Different top level components in the single core of Seneca-Lite have their distribution in different components. It is important to note that the Ibex processing core utilises most of the resources of the single core.

| | |
|---|---|
| CLB LUTs | 3339 |
| CLB Registers | 2123 |
| CARRY8 | 20 |
| F7 Muxes | 256 |
| F8 Muxes | 128 |
| F9 Muxes | 0 |
| Block RAM Tile | 32 |
| URAM | 0 |
| DSPs | 4 |

Table 6.1: **Seneca-Lite single core utilisation on ZCU111**

| **Module** | **Data Memory** | **Ibex Core** | **Inst. Memory** | **NoC Interface** | **TL-UL XBAR** |
|---|---|---|---|---|---|
| **CLB LUTs** | 92 | 2777 | 90 | 19 | 361 |
| **CLB Registers** | 109 | 1828 | 109 | 19 | 58 |
| **CARRY8** | 0 | 17 | 0 | 0 | 3 |
| **F7 Muxes** | 0 | 256 | 0 | 0 | 0 |
| **F8 Muxes** | 0 | 128 | 0 | 0 | 0 |
| **F9 Muxes** | 0 | 0 | 0 | 0 | 0 |
| **Block RAM** | 16 | 0 | 16 | 0 | 0 |
| **DSPs** | 0 | 4 | 0 | 0 | 0 |

Table 6.2: **Seneca-Lite single core utilisation breakdown between modules on ZCU111**

### 6.2.2 Energy Metrics

For the Seneca-Lite single core, energy estimation was carried out using Cadence JOULES. The top level component for this experiment was seneca_lite_single_core and the estimated energy consumption is the average power consumption for the single core. This is calculated by toggling the design lines and taking the average.

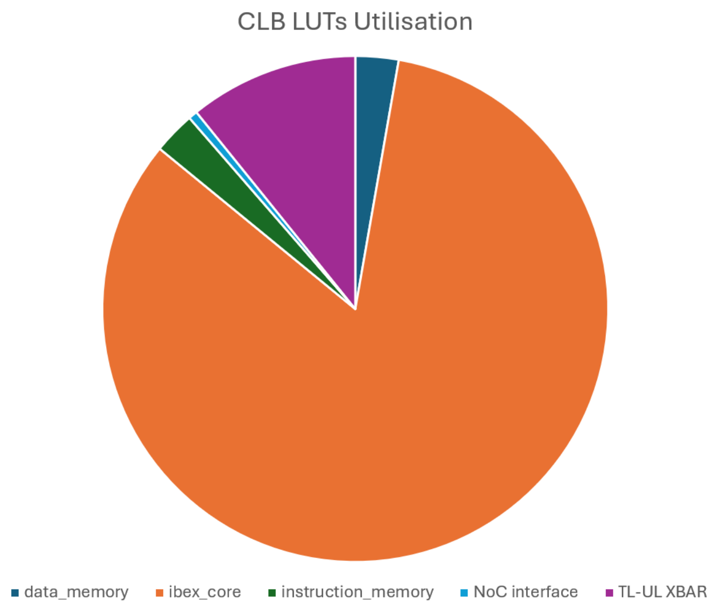


Figure 6.2: **Configurable Logic Blocks(CLBs) LUTs utilisation by submodules of Seneca-Lite core**

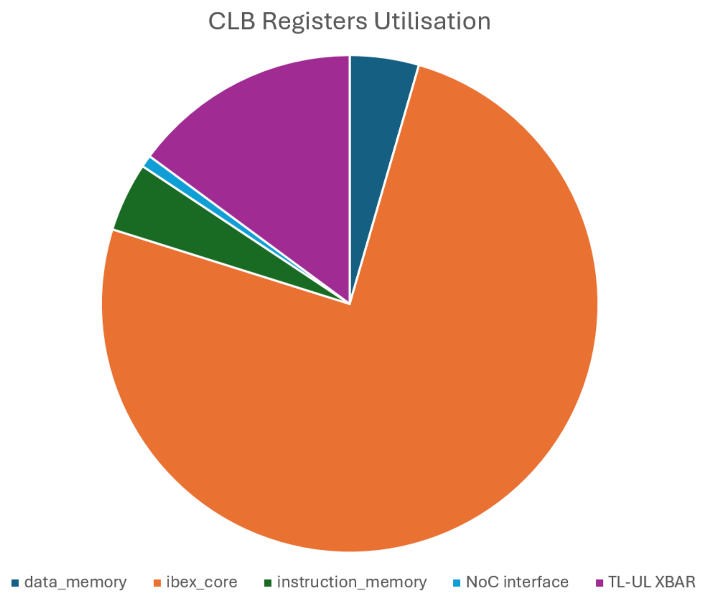


Figure 6.3: **Configurable Logic Blocks(CLBs) Registers utilisation by submodules of Seneca-Lite core**

## 6.3 System Level Results

### 6.3.1 Software Benchmark 1- Keyword Spotting Software

The first benchmark used is a 3-layer Fully Connected (FC) neural network used for keyword spotting (KWS). This software was designed to run on a single core and is optimized for embedded systems[27]. The code integrates essential features such as event-driven processing, interrupt handling, and neural network computation through layers, using quantized weights for efficiency.

This software leverages event-driven data flow to process neural network layers, making it suitable for neuromorphic architectures like SENECA[27]. This makes the software a desirable benchmark for the Seneca-Lite platform as well. The focus is on efficient handling of input events, layer-wise computation, and optimized memory usage to deliver high performance in constrained environments.

The software is designed to operate on a 3-layer fully connected neural network. The first layer processes input events, and the second and third layers continue to propagate and transform these inputs using weights and biases. Quantization of weights is used to optimize the performance and reduce the memory footprint, which is crucial in embedded systems[27].

#### Functionality and Workflow

- Event Processing: The system processes input events through the messaging interface. These events are typically neuron activations from the previous layer in a neural network. The address and value of each event are extracted and used to update the neuron values in the first layer by multiplying the event value with the corresponding weights [27].
- Layer Computation: Once all the input events are processed, the neuron values for the first layer are computed by summing the weighted inputs and adding a bias. These results are then used as inputs for the second layer, and the process is repeated until the third layer is computed.
- Post-processing: After the network has processed all layers, the system triggers post-processing, which involves interpreting the results, generating an output (e.g., detecting if a keyword was spotted), and preparing for the next round of input events.

By processing events asynchronously and using interrupts, the system efficiently manages CPU resources, minimizing idle time and power consumption.

Furthermore, the modular design allows for easy extension and optimization, making it a robust solution for real-time keyword spotting in low-power devices[27].

#### Seneca-Lite Setup and Workload Mapping

For verifying the functionality of the KWS software, a 4-core 2x2 Seneca-Lite architecture is used. The workload is a 3-layer fully connected network. Each of the layers are run on each core. Each core and its workload distribution are listed in Table 6.3. All communication between cores occurs over the Network-On-Chip (NoC).

| Core | Workload |
|---|---|
| Core 0 | Handling input stream and synchronization of other cores |
| Core 1 | Runs Layer 1 of the Fully Connected (FC) Neural Network |
| Core 2 | Runs Layer 2 of the Fully Connected (FC) Neural Network |
| Core 3 | Runs Layer 3 of the Fully Connected (FC) Neural Network |

Table 6.3: **Core workload distribution of KWS benchmark**

### 6.3.2 Software Benchmark 2- Hand Gesture Recognition Software

E. Ceolini et.al[3] present a benchmark for hand-gesture recognition using a combination of electromyography (EMG) and event-based camera sensor data. The study focuses on implementing this recognition system using neuromorphic computing platforms, particularly Intel's Loihi processor and the ODIN + MorphIC systems, and comparing their performance with traditional machine learning approaches[3].

The dataset generated relies on two sensors, a Dynamic Vision Sensor (DVS) and an Active Pixel Sensor (APS). The APS uses a 240x180 pixels resolution camera. The APS dataset and its network are used because of its efficiency in event driven inference framework.

The APS feature network consists of three convolutional layers and one fully connected layer. The first convolutional layer takes the greyscale image as input and outputs 8 feature maps using a 3x3 kernel. From Figure 6.4, CONV2 and CONV3 produce 16 and 32 feature maps respectively. The final layer of the feature network is the fully connected later that takes the flattened output from previous convolutional layers and maps it to 128 features.

The APS classification network consists of one fully connected network that produces an output that corresponds to one of five hand gestures. It maps the 128 features generated to one of five classes.

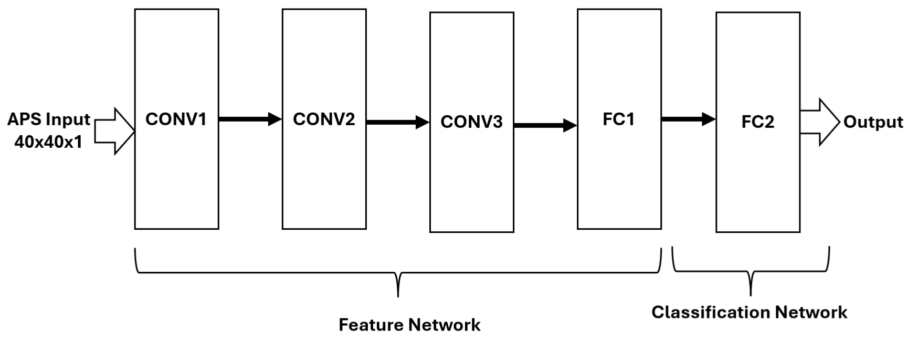


Figure 6.4: **APS neural network**

#### Seneca-Lite Implementation and Workload distribution

The entire network is implemented on a 3x3 9 core Seneca-Lite architecture. In the 9 cores, only 7 cores are used for active computation and running the neural network. CONV2 and CONV3 are run on two cores due to issues with memory sizes. The rest of the layers are run on a single core. The messages between cores are relayed via the Network-on-chip (NoC). The rest of the architecture has idle cores but the NoC of the entire 3x3 architecture is utilised. The input is streamed in through the input module as discussed in section 5.3.

### 6.3.3 Benchmarking Results

Since the Seneca-Lite platform is flexible and can be extended to any number of cores. The system level benchmarking is carried out using two architectures- 2x2 4-core system and the 3x3 9-core system.

#### 2x2 architecture synthesis Results

The 2x2 architecture is the system top and area is calculated for GF-22nm FDX technology node. The JOULES area calculations provide the system level area as:
$2 \times 2\, architecture(total\, area) = 0.416mm^2$
Similarly, for the APS benchmark, we use a 3x3 architecture.
$3 \times 3\, architecture(total\, area) = 1.124mm^2$

## Energy Metrics

#### Key Word Spotting benchmark

The 4 core design utilised for the KWS code. The energy calculations were done using JOULES for the post-synthesis simulation waveforms generated using Xilinx Vivado and NCSim. The total energy consumed by the four cores and the Network-On-Chip (NoC) is:
$Energy\, of\, KWS\, Task = 0.839\mu J$

#### APS Hand Gesture Recognition benchmark

In the 3x3 9-core design, the APS task has 7 active cores and 2 inactive cores. Similar to the previous benchmark, the same toolset is used for calculating the energy metrics for the APS benchmark.
$Energy\, of\, APS\, GR\, Task = 9.263\mu J$

### 6.3.4 Comparison with state of the art

#### KWS Benchmark

| Architecture | Total Energy (µJ) | Area (mm2) | Technology |
|---|---|---|---|
| Seneca | 1.2 | 1.8 | GF 22nm |
| Seneca-Lite | 0.839 | 0.416 | GF 22nm |

Table 6.4: **Comparison of Seneca and Seneca-Lite for KWS benchmark**

Table 6.4 provides a comparative snapshot of the two neuromorphic designs, Seneca and Seneca-Lite, in terms of their energy consumption and area, both fabricated using the GlobalFoundries (GF) 22nm technology process.

Seneca consumes more energy at 1.2 µJ compared to Seneca-Lite which uses only 0.839 µJ. This represents a significant 30.1% reduction in energy consumption for Seneca-Lite. In terms of area, Seneca occupies 1.8 mm$^2$ while Seneca-Lite uses just 0.416 mm$^2$, marking a striking 76.9% reduction in area. The differences are illustrated graphically in Figure 6.5. Such metrics highlight Seneca-Lite's advantages in energy efficiency and compactness, making it potentially more suitable for applications where power consumption and space are critical constraints, like embedded systems or portable devices.

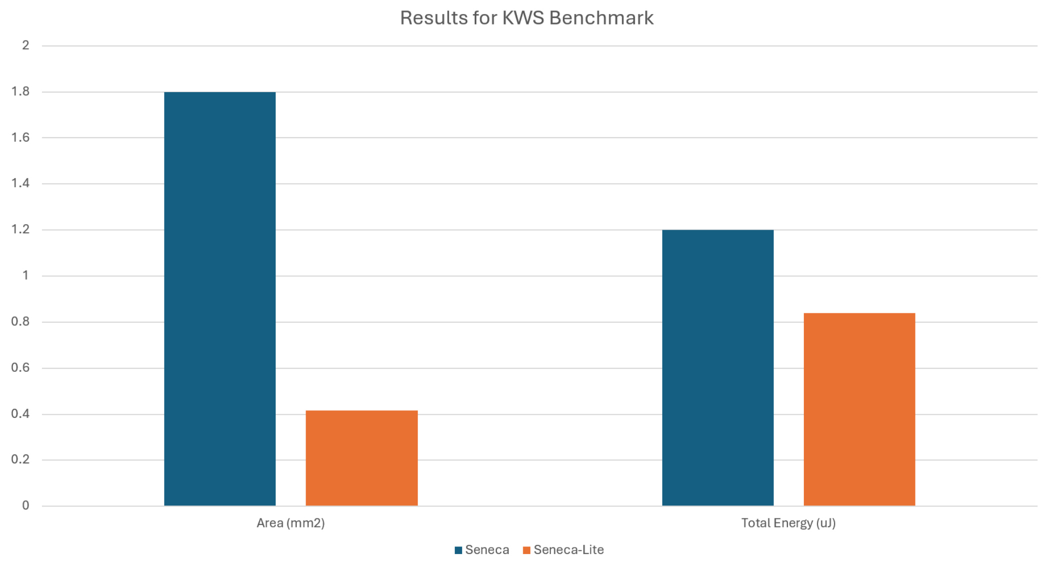


Figure 6.5: **Comparison of area and energy metrics for KWS benchmark**

The reduction in both area and energy without a drastic decrease in performance capabilities suggests that Seneca-Lite is a lightweight version of Seneca, focusing on power and space efficiency while potentially sacrificing some computational power or flexibility offered by the fuller Seneca architecture. These differences could align well with specific use cases of key word spotting software where smaller, energy-efficient chips are preferable, especially in battery-operated or wearable devices.

## APS Benchmark

The APS benchmark is a subset of the hand gesture recognition software described in [3]. The Table 6.5 features the results of 4 different neuromorphic architectures. The results for Loihi and ODIN are published in [3]. Seneca was benchmarked and compared in [27]. Seneca-Lite was benchmarked using the 3x3 architecture as described in subsection 6.3.2.

In Table 6.5, it is important to note that Silicon area and technology nodes are different for different cores. The silicon area is the total area of the utilized

| **Architecture** | **Accuracy (%)** | **Energy ($\mu$J)** | **Area (mm$^2$)** | **No. of cores** |
|---|---|---|---|---|
| **Loihi**[**3**, **5**] | 92.1 | 815.3 | 39 | 95 |
| **ODIN + MorphIC**[**3**, **9**] | 85.1 | 57.2 | 2.86 | 4 |
| **Seneca**[**27**] | 94.75 | 16.9 | 1.88 | 4 |
| **Seneca-Lite** | 90.25* | 9.62 | 1.124 | 9 |

Table 6.5: **Comparison of the results of different architectures for APS benchmark**

cores. The memory capacities and technology nodes of each core are: Loihi = 2 Mb in 14 nm, ODIN = 286kb in 28 nm, MorphIC = 576 Kb in 65 nm, and SENECA is 2.3 Mb in 22 nm.

In the comparison of the architectures presented in the table, Seneca-Lite demonstrates notable distinctions when evaluated against other architectures such as Spiking CNN (Loihi), ODIN + MorphIC, and Seneca. These differences can be summarized as follows:

- Accuracy: Seneca-Lite achieves an accuracy of 90.2%, which, while slightly lower than the other architectures, particularly Seneca (94.75%) and Spiking CNN (Loihi) (92.1%), remains competitive, especially considering its other advantages. Since, we do not use all 9 cores, the NoC bottleneck can cause some packets to arrive later than expected. With multiple cores working on the same layer, the communication delay leads to drop in accuracy.

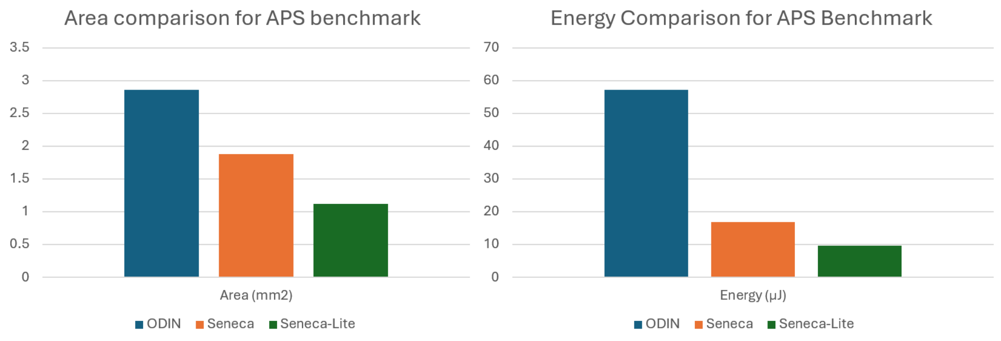


Figure 6.6: **Comparison of area and energy metrics for APS benchmark**

- Inference Energy: One of the most significant contrasts lies in the inference energy consumption. Seneca-Lite uses only 9.62 µJ, which is substantially lower than the Spiking CNN (Loihi) at 815.3 µJ—a stark contrast. It also outperforms ODIN + MorphIC (57.2 µJ) and even Seneca (16.9 µJ), highlighting Seneca Lite's superior energy efficiency.

- Area: In terms of the area occupied, Seneca-Lite occupies 1.124 mm$^2$, which is the smallest among the architectures compared. This is significantly less than Spiking CNN (Loihi), which occupies 39 mm$^2$ and even outperforms ODIN + MorphIC (2.86 $\mathrm{mm}^2$) and Seneca (1.88 $\mathrm{mm}^2$), indicating its compactness and efficient design.

The difference between Seneca-Lite's energy consumption (9.62 µJ) and that of Spiking CNN (Loihi) (815.3 µJ) is particularly stark, with Seneca-Lite being approximately 85 times more energy-efficient.

These comparisons highlight Seneca-Lite as a highly energy-efficient and compact architecture, making it particularly suitable for applications where power and space are at a premium, even if it slightly sacrifices accuracy compared to some other high-accuracy models.

# Chapter 7

# Conclusions and Future Work

## 7.1 Conclusions

The research presented in this thesis has introduced Seneca-Lite, an open-source, RISC-V-based, multi-core neuromorphic platform designed to meet the growing demands of AI and ML applications. The architecture of Seneca-Lite is carefully crafted to balance efficiency, scalability, and flexibility, providing a robust platform for academic research and development in the field of neuromorphic computing. The performance of Seneca-Lite was evaluated through extensive benchmarking, and the results underscore its competitiveness compared to existing neuromorphic platforms.

As discussed in chapter 4, the primary design objectives for Seneca-Lite included scalability, energy and area efficiency, adherence to an event-driven architecture, and the use of open-source components. Each of these objectives was carefully integrated into the system's architecture and implementation:

- Scalability: The design was required to scale with the number of cores, allowing it to support increasingly complex workloads. This objective was achieved by parameterizing the architecture, particularly in the top-level interconnections, which allow the number of rows and columns of cores to be customized based on the specific requirements of the application. This scalability was demonstrated in the implementation of various core configurations, including a 4-core and a 9-core system, with efficient intercore communication facilitated by a dynamic Network on Chip (NoC).

- Energy and Area Efficiency: Seneca-Lite was designed to be a lightweight neuromorphic platform, prioritizing both energy and area efficiency. The choice of the Ibex RISC-V core, known for its low power consumption, and the integration of the TileLink Uncached Lightweight (TL-UL) protocol, which minimizes latency and energy usage, were critical in achieving these goals. From subsection 6.3.4, the results showed that Seneca-Lite operates with significantly lower energy consumption compared to other architectures like the Spiking CNN (Loihi) and ODIN + MorphIC, particularly highlighting its suitability for energy-constrained environments.

- Event-Driven Architecture: The architecture of Seneca-Lite was designed to support event-driven data flow, which is essential for neuromorphic computing applications. This approach ensures that the system processes data only when necessary, reducing unnecessary power usage and latency. The implementation of this event-driven approach at both the core level and system level was successfully demonstrated, enabling efficient handling of spiking neural network workloads.

- Open-Source: A key goal of this project was to contribute to the open-source community by providing a platform that could be freely accessed, modified, and extended by researchers and developers. Seneca-Lite was built using open-source components, including the Ibex RISC-V core and TL-UL protocol, ensuring that it remains accessible and adaptable for future research and development efforts.

The benchmarking results revealed that Seneca-Lite performs well when compared to other neuromorphic architectures.

While Seneca-Lite's accuracy of 90.25% is slightly lower than that of the Seneca architecture and Spiking CNN (Loihi), it is still within a competitive range, particularly for lightweight and energy-efficient applications.

The architecture's compact design, occupying only 1.124 $\text{mm}^2$, further establishes Seneca-Lite as an area-efficient solution, which is significantly smaller compared to other platforms like Loihi, which occupies 39 $\text{mm}^2$.

## 7.2 Future Work

While Seneca-Lite represents a significant step forward in the development of scalable and efficient neuromorphic platforms, there are several avenues for future work:

- Advanced Routing Algorithms: While the current implementation uses a straightforward routing algorithm for the NoC, future iterations could benefit from more sophisticated algorithms that further reduce latency and improve fault tolerance. This also includes introducing multicasting to the architecture for faster response times.

- Integration with Emerging Technologies: As new AI and ML techniques emerge, integrating these into the Seneca-Lite platform could further enhance its applicability and performance in cutting-edge research areas.

- Broader Application Support: Expanding the range of applications that can be efficiently executed on Seneca-Lite, such as more complex spiking neural networks or hybrid AI models, would increase the platform's utility across various fields.

- FPGA and ASIC implementation: Current Seneca-Lite architecture was verified with benchmarks and simulations. The verification can be extended to hardware testing with FPGAs and custom chips for more accurate results.

# References

[1] Filipp Akopyan et al. “TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10 (2015), pp. 1537–1557.

[2] Inc. Cadence Design Systems. *Tools A-Z*. `https://www.cadence.com/en_US/home/tools/tools-a-z.html`. Accessed: Aug. 20, 2024.

[3] Enea Ceolini et al. “Hand-Gesture Recognition Based on EMG and Event-Based Camera Sensor Fusion: A Benchmark in Neuromorphic Computing”. In: *Frontiers in Neuroscience* 14 (2020). URL: `https://api.semanticscholar.org/CorpusID:220963473`.

[4] Nguyen Cong Dao et al. “FlexBex: A RISC-V with a Reconfigurable Instruction Extension”. In: *2020 International Conference on Field-Programmable Technology (ICFPT)* (2020), pp. 190–195. URL: `https://api.semanticscholar.org/CorpusID:233991737`.

[5] Mike Davies et al. “Loihi: A neuromorphic manycore processor with on-chip learning”. In: *Proceedings of the 25th ACM international conference on Architectural support for programming languages and operating systems*. 2018, pp. 409–412.

[6] Islam Elsadek and Eslam Yahya Tawfik. “RISC-V Resource-Constrained Cores: A Survey and Energy Comparison”. In: *2021 19th IEEE International New Circuits and Systems Conference (NEWCAS)* (2021), pp. 1–5. URL: `https://api.semanticscholar.org/CorpusID:235639171`.

[7] Tim Fischer et al. “FlooNoC: A Multi-Tb/s Wide NoC for Heterogeneous AXI4 Traffic”. In: *IEEE Design Test* 40.6 (2023), pp. 7–17. DOI: `10.1109/MDAT.2023.3306720`.

[8] Charlotte Frenkel. *ODIN: Online-learning Digital Spiking Neural Network Processor*. `https://github.com/ChFrenkel/ODIN`. 2019.

[9] Charlotte Frenkel et al. “A 0.086-mm$^2$ 12.7-pJ/SOP 64k-Synapse 256-Neuron Online-Learning Digital Spiking Neuromorphic Processor in 28-nm CMOS”. In: *IEEE Transactions on Biomedical Circuits and Systems* 13.1 (2019), pp. 145–158. DOI: `10.1109/TBCAS.2018.2880425`.

[10] Steve Furber et al. “The SpiNNaker project”. In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665.

[11] Wulfram Gerstner and Werner M Kistler. “Spiking neuron models: Single neurons, populations, plasticity”. In: *Cambridge University Press* 21.1 (2002), pp. 219–222.

[12] Yutian Huan and André DeHon. “FPGA optimized packet-switched NoC using split and merge primitives”. In: *International Conference on Field-Programmable Technology* (2012), pp. 47–52. DOI: `10.1109/FPT.2012.6412110`.

[13] Giacomo Indiveri et al. “Neuromorphic silicon neurons”. In: *Frontiers in Neuroscience* 5 (2011), p. 73.

[14] Intel. *Loihi 2: Next-Generation Neuromorphic Research Chip*. Online. Available from Intel Newsroom. Sept. 2021. URL: `https://www.intel.com/content/www/us/en/newsroom/news/next-gen-neuromorphic-chip.html`.

[15] Costas Iordanou et al. “Hermes: Architecting a top-performing fault-tolerant routing algorithm for Networks-on-Chips”. In: *2014 Eighth IEEE/ACM International Symposium on Networks-on-Chip (NoCS)*. 2014, pp. 178–179. DOI: `10.1109/NOCS.2014.7008782`.

[16] Eugene M Izhikevich. “Simple model of spiking neurons”. In: *IEEE Transactions on Neural Networks* 14.6 (2003), pp. 1569–1572.

[17] lowRISC. *Ibex User Manual*. Accessed: 2024-07-30. 2024. URL: `https://ibex-core.readthedocs.io`.

[18] Wolfgang Maass. “Networks of spiking neurons: the third generation of neural network models”. In: *Neural Networks* 10.9 (1997), pp. 1659–1671.

[19] Joshua Mack et al. “RANC: Reconfigurable Architecture for Neuromorphic Computing”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.11 (Nov. 2021), pp. 2265–2278. ISSN: 1937-4151. DOI: `10.1109/tcad.2020.3038151`. URL: `http://dx.doi.org/10.1109/TCAD.2020.3038151`.

[20] OpenTitan Project. *Tile Link Uncached Lightweight Interconnect Bus*. Accessed: 2024-07-30. 2024. URL: `https://opentitan.org/book/hw/ip/tlul/`.

[21] Michael Papamichael and James C. Hoe. “CONNECT: Re-Examining Conventional Wisdom for Designing NoCs in the Context of FPGAs”. In: *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*. New York, NY, USA: ACM, 2012, pp. 37–46. DOI: `10.1145/2145694.2145703`. URL: `https://dl.acm.org/doi/10.1145/2145694.2145703`.

[22] Michael Rogenmoser and Luca Benini. “Trikarenos: A Fault-Tolerant RISC-V-based Microcontroller for CubeSats in 28nm”. In: *2023 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS)* (2023), pp. 1–4. URL: `https://api.semanticscholar.org/CorpusID:263608627`.

[23] Kaushik Roy, Abhronil Jaiswal and Priyadarshini Panda. “Towards spike-based machine intelligence with neuromorphic computing”. In: *Nature* 575.7784 (2019), pp. 607–617.

[24] Kuladeep Sai Reddy and Kizheppatt Vipin. “OpenNoC: An Open-Source NoC Infrastructure for FPGA-Based Hardware Acceleration”. In: *IEEE Embedded Systems Letters* 11.4 (2019), pp. 123–126. DOI: `10.1109/LES.2019.2905019`.

[25] Guangzhi Tang et al. “SENECA: building a fully digital neuromorphic processor, design trade-offs and challenges”. In: *Frontiers in Neuroscience* 17 (2023). ISSN: 1662-453X. DOI: 10.3389/fnins.2023.1187252. URL: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2023.1187252.

[26] Inc. Xilinx. *ZCU111 Evaluation Kit.* https://www.xilinx.com/products/boards-and-kits/zcu111.html. Accessed: Aug. 20, 2024.

[27] Yingfu Xu et al. “Optimizing event-based neural networks on digital neuromorphic architecture: A comprehensive design space exploration”. English. In: *Frontiers in Neuroscience* 18 (2024). ISSN: 1662-4548. DOI: 10.3389/fnins.2024.1335422.

[28] Feichi Zhou and Yang Chai. “Near-sensor and in-sensor computing”. In: *Nature Electronics* 3 (2020), pp. 664–671. URL: https://api.semanticscholar.org/CorpusID:228820372.