System-Level Design in the Era of Brain-Computer Interfaces

Guy Eichler

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2025

Abstract

System-Level Design in the Era of Brain-Computer Interfaces

Guy Eichler

Brain-computer interfaces (BCIs) hold significant potential to not only transform healthcare, by enabling individuals with neurological impairments to interact with the world, but also to enhance computational models and technology, opening new avenues for human-computer interaction and advancing the capabilities of artificial intelligence (AI) and machine learning (ML). However, BCI systems must overcome several obstacles to transition from research lab experiments into real-world applications. One major challenge is the need for modern BCI systems to be mobile, wireless, and include an implanted system-on-chip (SoC) that interfaces with the brain. This requirement introduces safety concerns and imposes physical constraints on BCI systems. A second challenge is that BCI systems must integrate an ever-increasing number of sensors to support large-scale data acquisition (DAQ), enabling a better understanding of the brain. Consequently, these systems must handle growing volumes of neural data, pushing wireless transmission data rates, power consumption, and overall feasibility of implant-based BCI systems to their limits. To address these challenges, one potential solution is to integrate computation for BCI applications in specialized hardware close to the neural data source, a common approach in I/O-bound systems. Nonetheless, this solution introduces its own set of complexities, as BCI applications often rely on computationally intensive machine-learning models that must be optimized in order to be executed in real time, while meeting the physical constraints of implant-based BCI systems. For this reason, understanding the full structure of the BCI system, including all of its components, is key

for future development in the BCI field. In addition to the implanted SoC that transmits the neural data, implant-based BCI systems also incorporate a mobile, wearable SoC that receives the transmitted data. This wearable SoC must meet the physical constraints of wearable devices, which are less stringent than those of implanted devices. As long as transmission data rates between the two SoCs are sufficient, the wearable SoC can execute BCI applications either fully or in collaboration with the implanted SoC, enabling the design of practical BCI systems capable of running meaningful applications for public use. The field of BCI is multidisciplinary and holds transformative potential for both medicine and technology. To fully realize this potential, computer architects and engineers must understand the general system-level structure of BCI systems, as well as their constraints and components. Thus, my thesis is that *to unlock the full potential of the BCI field, BCI system development must be properly defined and standardized, with a clear target BCI system, an understanding of its constraints, and a distinction between three overlapping time domains that emphasize different levels of research and development, with progress continuing concurrently in each domain.* I support this thesis by presenting a three-dimensional approach to restructuring system-level design in the BCI field. Each dimension corresponds to a time domain, representing an independent area of research and development: (1) Pre-BCI – designing implant-based BCI systems that support large-scale data acquisition and wireless communication; (2) Intra-BCI – developing methodologies to integrate real-time computation and complete BCI applications into BCI systems; and (3) Post-BCI – specializing computational data flows to interact seamlessly with the biological brain. To guide system development in these domains, I discuss the unique constraints of modern BCI systems, relying on trends in machine learning and neural interface design. I define the self-contained BCI system as our target system, capable of large-scale DAQ, high-throughput data transmission and application-level computation. I also explore potential future trends, including the use of brain-inspired neuromorphic computing as an intermediate step in the computational flow. Most importantly, I demonstrate how development can proceed concurrently across all three dimensions to overcome the key obstacles in BCI system development and accelerate the realization of fully functional BCI systems for real-world BCI applications.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my advisor, Professor Luca Carloni. His unwavering support, insightful discussions, and invaluable feedback were instrumental throughout my PhD journey. His encouragement and guidance provided me with the freedom and resources to explore my passion for innovation and research.

I would also like to thank Professor Ken Shepard, who supported me greatly during my time at Columbia University. His trust in me provided the opportunity to work as part of a brilliant collaborative team on the design of emerging brain-computer interface technologies, which inspired many of my research ideas and ultimately helped shape my PhD dissertation.

I would like to specifically thank the incredible mentors I had throughout my PhD journey. Among them are Paolo Mantovani and Giuseppe Di Guglielmo, two research scientists who created the research infrastructure I used from the beginning of my PhD and were always available for advice and help. Additionally, Luca Piccolboni and Davide Giri, two senior PhD students who were part of my research group in the early years of my PhD, showed me how to become a productive and effective PhD student and how to transition my research ideas from theoretical concepts into full implementations and published research papers.

I would also like to thank the other members of the system-level design research group in the computer science department at Columbia University, who collaborated with me on various research projects during my PhD. Specifically, I am grateful to Biruk Seyoum, Kuan-Lin Chiu, and Joseph Zuckerman, who co-authored some of my papers. I would also like to acknowledge other distinguished external collaborators, including Nanyu Zeng, Taesung Jung, Yatin Gilhotra, and Judicael Clair. To all my collaborators, thank you for enabling my research and for allowing me to contribute to your work, which enriched my PhD journey.

Finally, I would like to extend my thanks to all my colleagues throughout the years, my friends, and, of course, my family. Special thanks to my parents and brothers for their unwavering support and encouragement, which helped me reach this point in my academic journey and achieve my goals and dreams.

Preface

Chapter 1

Introduction

1.1 The Promise of Brain-Computer Interfaces

The main purpose of brain-computer interface (BCI) systems is to establish a direct connection between the brain and external devices [1, 2, 3]. In these systems, neural data is recorded from the brain, processed, and used to generate outputs that serve two primary goals.

The first goal is to apply this information in various assistive technologies for future public use [4, 5]. For example, neural data from motion-related brain regions can be processed to control prosthetic limbs and robotic arms [6, 7, 8, 9]. Similarly, neural data from vision-related regions can aid in vision restoration [10, 11, 12]. Additionally, processed brain information has been shown to assist in authenticating individuals [13] and holds potential for supporting cryptographic protocols used to implement information security and data privacy [14].

The second goal of processing brain data is to advance our computational models toward the next generation of artificial intelligence (AI) [15, 16]. By studying interactions between neurons in the brain, we can gain insights into how the brain executes computational tasks. These insights can then be used to replicate these interactions, advancing AI through the creation of more accurate and efficient brain-inspired computational models [17, 18, 19].

Ultimately, the development of assistive technologies benefits from enhancing our computational models. Developing highly efficient and accurate models improves the processing of neuronal activity, deepens our understanding of the brain, and enables a more meaningful translation of brain intent, which supports the research and development of BCI applications for public use [20].

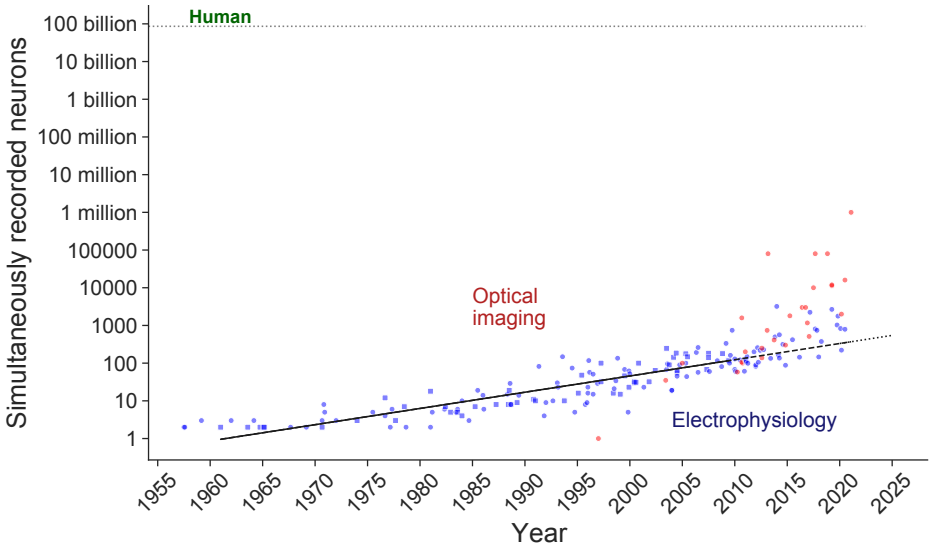


Figure 1.1: The scale of neural recordings over time. Adapted from the work of Urai *et al.* [25].

1.2 The Evolution of BCI Systems through Scalable Neural Interfaces

The key to understanding complex interactions between neurons and overall brain functions lies in the interplay of massive populations of neurons. With the human brain containing approximately 100 billion neurons, modern BCI systems aim to achieve high resolution and high throughput in neural data acquisition [21, 22, 23, 24]. To support these features, the sensing component within BCI systems must continuously improve. This technology, known as the *neural interface (NI)*, plays a critical role in advancing our understanding of the brain. The drive to enhance brain understanding has led to exponential advances in NI technology.

As shown in Figure 1.1, the capacity of NIs to conduct massively parallel recordings of neuronal activity through sensors, or *channels*, has approximately doubled every seven years since the first BCIs emerged in the 1960s [25, 26].

To accommodate the increasing resolution and throughput of neural data from the brain, BCI systems have evolved from integrating low-resolution, non-invasive electroencephalography (EEG) channels [27, 28, 29, 30] to incorporating high-resolution, invasive, and implantable NIs [31, 22, 32, 33, 34, 35]. Invasive NIs, capable of integrating large-scale numbers of channels, are now at the forefront of research and development [21, 32, 22, 34, 36].

1.3 Challenges in BCI System-Level Design

Figure 1.2 illustrates an abstraction of modern BCI system-level designs. These designs aim to create self-contained, implant-based BCI systems. A *self-contained BCI system* is capable of executing BCI applications. It includes an implanted system-on-chip (SoC) responsible for managing data acquisition (DAQ) from the NI and wirelessly communicating neural data [22, 21, 32, 37, 38, 39]. A key constraint on the implanted SoC is that it must limit power consumption to prevent the surrounding temperature from rising above 1.5°C, as this could cause cellular damage to the brain tissue [40, 41]. The system also includes a wearable SoC that intercepts neural data from the implanted SoC, performs *external processing*, and sends control commands to external devices [42].

Self-contained BCI systems that execute BCI applications must support running BCI computations. To enable this, one approach is to handle *application-level computation* as part of the external processing on the wearable SoC [43, 44]. This approach avoids integrating application-level computations into the implanted SoC, which typically involves advanced machine learning (ML) techniques that are computationally intensive and power hungry. By offloading computation to wearable devices [30], which have less strict power constraints than implantable devices, the system can operate within power limits. However, this approach must also ensure that neural data be transferred quickly and reliably from the implanted SoC, enabling BCI applications to run in real-time and be effective in real-world use cases. This requirement becomes increasingly challenging as the number of channels integrated into the NI grows, leading to a higher volume of neural data that must be transmitted in real time.

To avoid reaching the limits of data transmission rates and power consumption for communication on the implanted SoC, a second approach integrates application-level computation within the implanted SoC [45, 46]. However, since modern BCI applications rely on computationally intensive ML-based algorithms, a major challenge lies in implementing these algorithms in a lightweight manner that fits within the tight power budget of the implanted SoC. To achieve this, the implanted

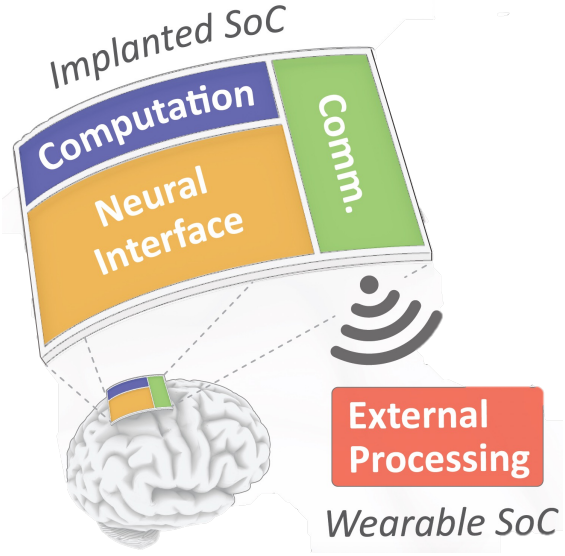


Figure 1.2: BCI system-level design.

SoC must carefully balance wireless communication and on-chip computation.

To date, neither of these two approaches has resulted in a single self-contained, implant-based BCI system that has been successfully tested *in vivo* (on live subjects), proven safe and reliable, and made available to the public, suggesting that current strategies may need to be refined.

The obvious potential of the BCI field has led to increased research into emerging BCI applications using complex computations, as well as designs for highly advanced implanted SoCs that integrate thousands of channels. However, financial reports predict a slow progression toward the commercial viability of real-world BCI systems [47]. The reasons for this conclusion range from ethical considerations that limit the promotion of such devices to safety concerns that slow down the approval of BCI systems for broader experimentation. From an ethical perspective, we aim to demonstrate that BCI systems offer superior performance and an improved quality of life compared to other available solutions. In this way, BCI systems could become the preferred solution. From a safety perspective, we must develop BCI systems with a clear understanding of their physical limitations, which should guide the design process and motivate the development of unique design solutions within the BCI domain.

As computer architects, while we can only speculate on the next breakthrough in BCI algorithm development, we have a responsibility to promote better design methodologies and establish an

organized, standardized framework that provides guidelines and a common taxonomy for BCI system development [48]. In this way, we can foster better discussions among researchers in emerging fields, encourage collaboration, and accelerate the development of designs with improved performance and practical use cases.

For this reason, in this dissertation, I aim to consolidate system-level design in the BCI field, with a specific focus on the integration of computation within BCI systems. Based on research and development goals, I divide the work on BCI systems into three focus areas, each defined as a distinct time domain with respect to BCI. In doing so, I hope to promote a more organized approach to BCI system research and development and clarify the dependencies between various research perspectives that offer solutions at different levels of the BCI system.

1.4 Thesis Scope

Thesis. To unlock the full potential of the BCI field, BCI system development must be properly defined and standardized, with a clear target BCI system, an understanding of its constraints, and a distinction between three overlapping time domains that emphasize different levels of research and development, with progress continuing concurrently in each domain.

1.4.1 Research Contributions

In my research, I contribute to multiple aspects of BCI system-level design. As stated in my thesis, I distinguish between three focus areas of contribution, each defined as a time domain. The purpose of this classification is to indicate the prioritization between these areas. A time domain that precedes others is more fundamental for creating a functional BCI system based on current technology, while a later time domain plays a crucial role in enhancing the system to make BCIs attractive and widely accessible to the public.

Pre-BCI. This focus area involves designing implant-based BCI systems that support large-scale neural data acquisition, wireless communication, and establish the foundation for real-time computation within the BCI system. At this stage, implementing computation is not the primary

focus, as long as we are capable of supporting real-time wireless transmission of neural data. In this section, I present our efforts to create a standard for implant-based BCI system design, starting with the definition of a practical target system. I then discuss my work on developing microcontrollers for both the implanted SoC and the wearable SoC, which are responsible for connecting the system, ensuring smooth operation, and supporting essential functionalities for advancing BCI research. Specifically, I present two implant-based BCI systems: **BISC** [21, 49] and **IBISX** [50], each integrating a different type of neural interfaces. As part of a collaborative effort, I developed the control flows for these BCI systems, both of which were successfully tested *in-vivo* and *in-vitro*.

Intra-BCI. This focus area involves integrating hardware-based computation into the implant-based BCI system, making it self-contained. As the number of channels in neural interfaces grows exponentially, I analyze the integration of machine-learning applications within the BCI system, scaling them according to the increasing number of channels. Specifically, I introduce **MINDFUL**, an analytical framework for estimating lower bounds on power and area costs in implanted SoCs that integrate hardware-based deep neural networks (DNNs). Using MINDFUL, I investigate the feasibility of scaling the neural interface while embedding DNNs into the implanted SoC. The conclusion is that more effective system-level optimizations are needed for combining implanted SoCs and cutting-edge computations specialized for BCI applications. Based on these findings, I present my further contributions focused on integrating computation into the wearable SoC, an intermediate step toward the final goal of embedding computation directly into the implanted SoC. In this context, I introduce **MasterMind** [43], a many-accelerator SoC architecture for real-time BCI applications. MasterMind demonstrates a methodology for integrating BCI algorithms within the system, focusing initially on the constraints of the wearable SoC. This enables full BCI applications to run in real-time, as long as real-time wireless transmission of neural data from the implanted SoC is maintained. Additionally, I present **MindCrypt** [44], an SoC following a similar design methodology. MindCrypt utilizes the brain as a resource by implementing a custom random number generator that generates random bits from neural data, integrated into a heterogeneous

SoC. These random numbers support ML and security applications essential for a self-contained BCI system. I also explore dynamic partial reconfiguration (DPR) on FPGA as a potential solution for future BCI applications on constrained devices.

Post-BCI. This focus area addresses the next level of computation in BCI systems. As neural interfaces and the number of neurons analyzed in parallel grow, our computational models must be updated to accommodate this new reality. Currently, most computational models used in BCI applications are based on general models that are also applied in other fields, with the primary difference being the data used to train these models. I argue that we must tailor our computational models and algorithms to the specific needs of the BCI system and the brain. Only by doing so we can fully exploit the increased volume of neural data, achieve higher accuracy, and ensure the feasibility of integration within the BCI system. First, I introduce **KalmMind** [51, 52], a configurable Kalman filter design framework for embedded Brain-Computer Interfaces. KalmMind offers a modular hardware architecture for the Kalman filter (KF), a key algorithm for motion decoding in BCI. It features dynamic accuracy tuning embedded within the KF algorithm, enhancing energy efficiency and specialization for motion decoding from diverse neural data in BCI systems. In addition, we must acknowledge that the scale of modern deep neural networks (DNNs) is growing further apart from the biological brain. To maintain an efficient interface capable of real-time computation within the constraints of BCI systems, emerging models like spiking neural networks (SNNs) should be considered. While neuromorphic computing based on SNNs is functionally closer to the biological brain, it is expected to require much less energy than DNNs for executing learning-based tasks. I propose that SNNs could serve as an interface between biological neurons and DNNs, with the goal of integrating all three to collaboratively solve tasks, improving computational efficiency. To explore this, I present **SpikeHard** [53], an efficiency-driven accelerator design that enables the integration of neuromorphic hardware into heterogeneous SoCs. SpikeHard offers a novel approach to seamlessly executing neuromorphic workloads alongside traditional ML workloads on these SoCs.

1.4.2 Structure of the Dissertation

The dissertation begins with **Part I: Preface**, including **Chapter 1** for the introduction and **Chapter 2** for background information. It then explores each of the time domains in BCI system-level design, organized into three parts.

In **Part II: Pre-BCI**, **Chapter 3** covers the development of highly-configurable BCI systems, focusing on the design of microcontrollers for the implanted and wearable SoCs, and presenting the **BISC** and **IBISX** implant-based BCI systems.

In **Part III: Intra-BCI**, **Chapter 4** introduces **MINDFUL** and discusses the scalability of BCI systems and applications based on the current state of the art. **Chapter 5** continues with **MasterMind**, explaining the methodology for implementing BCI applications and integrating accelerators into the wearable SoC. **Chapter 6** presents **MindCrypt**, demonstrating how the brain can serve as a resource in the BCI system.

In **Part IV: Post-BCI**, **Chapter 7** presents a discussion about the current state of computation in BCI, its reliance on ML models, and potential future directions including brain-inspired computation. **Chapter 8** describes **KalmMind**, showing how small modifications to a popular algorithm can enhance BCI applications. **Chapter 9** introduces **SpikeHard**, laying the groundwork for combining SNNs and DNNs in one SoC to support next generation BCI applications.

The dissertation ends with **Part V: Epilogue**, offering conclusions and outlining future work.

Chapter 2

Background

2.1 Neural Interface Design

The neural interface (NI) translates neural signals into a digitized form. Its design, particularly the type and number of channels, determines data resolution, throughput and computational capacity for BCI applications.

Non-Invasive Neural Interfaces. Electroencephalogram (EEG) technology is the common sensing technology used in non-invasive NIs. In EEG, the channels consist of large electrodes attached to the scalp, capturing the electrical activity of large populations of neurons [27, 28, 54]. These interfaces are external to the body, eliminating the need for a surgical procedure. This aspect contributes greatly to the popularity of non-invasive NIs. However, EEG-based NIs can only record the aggregated activity of many neurons at the level of brain regions. In addition, non-invasive NIs typically use a small number of channels. This results in relatively low resolution and throughput of brain data. For this reason, EEG NIs have shown limited success in solving fine classification problems with ML methods and are used only in relatively coarse classification problems [28].

Invasive Neural Interfaces. Many NIs directly interface with the brain tissue. These NIs commonly use micro-electrodes for interfacing with the brain [31, 22, 21, 55, 56, 38, 57, 23, 32]. These NIs are invasive and come in various shapes and sizes, each with distinct characteristics. One design features a shank that penetrates the cortex of the brain, with electrodes positioned on its surface [58]. Another involves multiple wires carrying electrodes, strategically inserted into specific cortical regions [22]. Both are considered highly invasive approaches, requiring craniotomy and brain penetration, which carry risks, especially during potential device removal. A less

invasive alternative uses an endovascular stent with electrodes positioned within the brain's blood vessels [39]. While this method avoids craniotomy and leverages well-established techniques from cardiology to streamline system development, its structure limits electrode placement and is expected to constrain neural data resolution.

In response, a new family of minimally invasive NIs has emerged [21, 56, 38]. These NIs use electrocorticography (ECoG) with micro-electrode arrays (MEAs) placed on the cortex of the brain. This approach reduces surgical risks and simplifies device removal. High-resolution brain data is achievable when the electrodes are numerous and densely integrated [24]. Additionally, neural imagers represent a novel class of minimally invasive NIs. Instead of electrodes, they use single-photon avalanche diodes (SPADs) to detect photons emitted by neurons during spiking electrical activity [50]. Neurons are modified via optogenetics to express light-emitting indicators for photon emission [35]. With adequate resolution and advanced light filtering, neural imagers enable precise mapping of neuronal circuits in the brain.

Goals of Neural Interface Design. As a goal for NI design, minimally invasive NIs must support high-quality recording. This capability is essential for an accurate evaluation of neuron activity [31]. Another goal is a stable and long-term recording capability. The NI must provide valid brain data over long periods of time. This capability usually depends on two main features. The first one is device resistance to general degradation that will affect the signal quality. The second one is the biocompatibility of the device with minimal immune response from the tissue. Minimizing the invasiveness of the device helps with this requirement.

Another goal is high-throughput and high-density recording capabilities. Monitoring a large number of neurons simultaneously is assumed to improve the understanding of deeper functions of the brain [26, 31, 25]. The NI must offer insights into interactions between individual neurons and allow the distinction of specific neurons among large populations of neurons [59].

2.2 Wireless RF Communication in BCI

Traditional BCI systems predominantly relied on wired connections, significantly limiting their mobility and practical application, especially in the human setting [60, 36, 32, 57].

The introduction of wireless technologies in BCIs has effectively mitigated these constraints, paving the way for research opportunities and diverse applications in neuroprosthetics and rehabilitation. However, standard wireless communication protocols like Wi-Fi or Bluetooth are not ideally suited for large-scale NIs; they either consume excessive power or cannot support the required data rates. Custom-designed solutions using radio frequency (RF) are imperative to achieve optimal energy efficiency and prevent overheating [61, 62, 63, 21, 50]. The data rate requirements of wireless BCIs scale almost linearly with the number of channels, while optimal proximity between the implanted SoC and an external antenna is critical for maximizing energy efficiency.

2.3 BCI Applications

Among BCI applications, some provide real time predictions in order to control external devices [6, 64, 65, 66], while others have the goal of unlocking a deeper understanding of different regions of the brain [67, 68, 69]. We classify these two groups as *online* and *offline* BCI applications, respectively.

Online BCI applications. BCI applications provide solutions in the fields of motion, vision, and speech. For example, brain data from the motor cortex is decoded into commands to move a prosthesis or a cursor on a screen [6, 64, 70, 71]. Audio is synthesized from brain activity in the anterior speech area of the brain [72, 73, 74]. Neural data from the visual cortex is decoded into shapes and images, and the results are used to help with vision loss [10, 75, 11].

BCIs detect brain activity and use processing engines to decode the intention of the brain. This computation must be executed in real time, i.e., the computational output should be ready to actuate control before the individual notices any delay [76]. In general, real time in BCI has been defined in different ways. Some define real time as the execution of the entire application within

the reaction time of the brain, which is ~$0.18sec$ [43, 44], while others generalize this approach and define real-time as the execution of a task within a specific window that is set by a specific application [77]. Other works define real time in BCI as the ability to process data with a specific data rate that matches the sampling rate of the NI [46, 45].

Offline BCI applications. BCI applications implement ML models that require the acquisition of large amounts of high-resolution brain data [69, 78, 68, 67]. For instance, BCI applications have been developed to analyze neurons as a model for artificial neural networks [78, 69, 79, 80]. Offline BCI applications also provide insights for developing online BCI applications.

Offline BCI applications do not require the computation to be done in real time; however, data must be reliable, accurate, and diverse to gain meaningful insights about the brain. For this reason, the most important component of these applications is the quality of raw neural data. The quality of the data is based on the type of NI and its channel count. The growth of the NI in terms of channel count is crucial for developing novel offline BCI applications and for the progress of brain research.

Computation in BCI applications. Traditionally, BCI applications have utilized linear control algorithms, for example, the Kalman and Wiener Filters [81, 82, 83]. However, new methods that utilize machine learning (ML) and, specifically, deep neural networks (DNNs) provide higher accuracy for decoding brain data [83, 84, 71].

As the dimensions of input data increase, the computational complexity and model size of DNNs deteriorate greatly [26, 85]. This results in computational requirements that grow faster than the input, making the computations more resource intensive [26, 86]. With neural recordings arriving simultaneously from many channels, an increase in channel count is expected to cause an increase in computational demands within BCI applications [87].

In this work, we emphasize the importance of understanding the potential scaling of computation in DNN-based BCI applications and of predicting the feasibility of implementation before applying it to real-world BCI systems [26, 25, 37, 88].

2.4 Heterogeneous SoC Architectures for BCI

The development of heterogeneous SoC architectures for the Internet-of-Things (IoT) has given rise to solutions for running time-sensitive applications with higher performance and lower resource utilization on constrained devices [89, 90, 91, 30, 92].

As modern BCIs are based on implanted and wearable devices, some BCI SoC architectures are designed to be self-contained and include the NI, the wireless communication, and computation engines [45, 46, 77]. In the case of a non-invasive NI, this solution is ideal, as the NI is external to the body, and the power consumption is only constrained by the limits of the power source [77, 93]. In the case of an invasive NI, however, power consumption of implanted SoCs must be limited in order to avoid cellular damage in the brain [45, 46, 40, 41].

Figure 1.2 presents the general approach, where the BCI system includes an implanted SoC that integrates an NI, wireless communication, on-chip computation, and a wearable SoC that executes external processing. These BCI systems can be classified into two groups according to the type of computation on the implanted SoC.

The first group offers computation that only prepares neural data to be transferred via wireless communication. In this case, the data to transmit is digitized raw neural data, and the communication must operate at a data rate that can handle the total number of channels in the NI and the sampling rate of the NI [21, 22]. Application-level computation is implemented as external processing in a wearable SoC in the body area network (BAN) [43, 29, 30].

The second group offers an implementation of application-level computation on the implanted SoC. In this case, the final output of the computation is transmitted to the wearable SoC. The final output is much smaller in size compared to the raw neural data. Consequently, transmission data is significantly reduced [46, 45, 87].

One of our goals is to find a design balance that allows us to include an NI, communication, and computation in one implanted SoC while resulting in a low amount of power consumption for safety measures.

2.5 The HiWA Algorithm

The HiWA algorithm transforms a source dataset into a target dataset by minimizing a distance metric [65], and leverages the Sinkhorn algorithm [94] that enables highly efficient computation. HiWA was successfully applied for matching the neural activity from the brain of non-human primates to body movements in a process called Distribution Alignment [95]. HiWA provides a proof of concept that the data recorded from the motor cortex of the brain can be matched to body movements recorded independently, and compared across time, space, and even different subjects.

Figure 2.1 reports the inner loop of HiWA, which is the core part of the algorithm [65]. HiWA takes four main inputs (line 2). Matrix $X_{p\times m}$ represents a cluster of 3D points extracted from the multi-dimensional brain data (after applying Factor Analysis [96]), while matrix $Y_{q\times m}$ is a cluster of 3D points from the database of body movements. Values p and q are the number of points in the two clusters and m is the number of dimensions represented in the datasets ($m = 3$). The scalar $P = \frac{1}{\#clusters}$ is used as a normalization factor in SVD. Matrix T is used as a step size, similar to the one used in gradient descent.

First, HiWA initializes Q (line 3) and R (line 4). Matrix Q encodes a correspondence between 3D points in X and Y (points $X(k)$ and $Y(l)$). Then, HiWA calls SVD and Sinkhorn in a loop that converges when a norm check on R is satisfied (line 5). R is used to rotate cluster X and to reduce its distance from Y. In SVD (line 9), matrix A represents the next rotation of cluster X towards cluster Y. It is calculated by multiplying the two clusters with their correspondence matrix Q, added with the step size T. Matrix A is then decomposed by the linear algebra method $svd()$ and assembled into matrix R by multiplying its singular vectors matrices U and V. By multiplying R with X, SVD rotates the data in the cluster. Then, it returns R, $X \cdot R$, and Y^T. In Sinkhorn (line 14), matrix C represents the initial distances between 3D points in X and Y (points $X(k)$ and $Y(l)$). The global variable γ is a *regularization parameter* set by the user and used to produce matrix K from matrix C. The variable max_iter, which is also set by the user, determines the maximum number of loop iterations in Sinkhorn. The loop (line 18) completes when a convergence check is satisfied

1: **global variables**: $p, q, m, \gamma, max\_iter$

2: **function** HiWA_CORE($X_{p\times m}, Y_{q\times m}, P_{1\times 1}, T_{m\times m}$)
3: Initialize $Q_{p\times q} = \mathbb{1}_{p\times q}/(p \cdot q)$
4: Initialize $R_{m\times m} = \mathbb{1}_{m\times m}$
5: **while** $||R - R_{prev}|| > 0.01$ **do**
6: $R, XR, Y^T \leftarrow \text{SVD}(X, Y, T, P, Q)$
7: $Q, sum(C * Q) \leftarrow$ SINKHORN($X \cdot R, Y^T$)
8: **return** $R, sum(C * Q)$

9: **function** SVD(X, Y, T, P, Q)
10: $A = 2P(Y^T \cdot Q^T \cdot X) + T$
11: $U, V, S = svd(A)$
12: **return** $R = U \cdot V^T, X \cdot R, Y^T$

13: **function** SINKHORN(X, Y)
14: $C \leftarrow C(k, l) = ||X(k) - Y(l)||^2$
15: $K = exp(-C/\gamma)$
16: Initialize $\vec{b}_{q\times 1} = \mathbb{1}_{q\times 1}/q$
17: **for** i from 0 to max_iter **do**
18: $\vec{a} = \mathbb{1}_{p\times 1}/(p \odot (K \cdot \vec{b}))$
19: $\vec{b} = \mathbb{1}_{q\times 1}/(q \odot (K^T \cdot \vec{a}))$
20: **if** $(\vec{a}, \vec{b})$ converged **then**
21: $break$
22: $Q = diag(\vec{a}) \cdot K \cdot diag(\vec{b})$
23: **return** $Q, sum(C * Q)$

Notations:
$exp(\cdot), /, \odot, sum(\cdot), *$ are elementwise operators
R_{prev} is the previous R
$diag(x)$ is a diagonal matrix with vector x as the diagonal
$svd(\cdot)$ is the original Singular Value Decomposition
$||\cdot||$ is $norm$

Figure 2.1: The core of the Hierarchical Wasserstein Alignment (HiWA) algorithm.

on vectors $\vec{a}$ and $\vec{b}$ or the number of iterations reaches max_iter. The elementwise sum of $C * Q$ is the Sinkhorn distance between the two clusters. The Sinkhorn algorithm returns the realized sum, and the current correspondence matrix Q. The two outputs provide the current assessment of similarity between clusters X and Y. After the HiWA core computation completes on two clusters, matrix R and the distance $sum(C * Q)$ are returned and stored (line 8).

The algorithm of Figure 2.1 is executed across all clusters pairs. Each result from a pair of clusters is completely independent from the other pairs, and thus can be analyzed in parallel. After all cluster pairs have been analyzed, the best match between brain data cluster and a movement cluster can be determined (the best match is the shortest Sinkhorn distance between clusters).

2.6 The Kalman Filter Algorithm

Motion Decoders for BCI. Traditionally, BCI applications of motion decoding from neural data have utilized linear methods for predicting kinematics (e.g., the Kalman Filter), which provide stable but limited accuracy [83, 97]. New methods of motion decoding use non-linear deep neural networks [98, 70, 99, 97]. While these methods can provide high accuracy in some cases, they

1: **function** KALMAN_FILTER($F, Q, H, R, \vec{x}_{n-1}, P_{n-1}, \vec{z}_n$)
2: //Predict
3: $\vec{x}_n = F \cdot \vec{x}_{n-1}$
4: $P_n = F \cdot P_{n-1} \cdot F^t + Q$
5: //Update
6: $\vec{y} = \vec{z}_n - (H \cdot \vec{x}_n)$ //Innovation
7: $S = H \cdot P_n \cdot H^t + R$
8: $K = P_n \cdot H^t \cdot S^{-1}$ //Compute Kalman Gain
9: $\vec{x}_n = \vec{x}_n + K \cdot \vec{y}$
10: $P_n = (I - K \cdot H) \cdot P_n$
11: **return** $\vec{x}_n, P_n$

Figure 2.2: The Kalman Filter algorithm.

can impose long training times and also suffer from overfitting, due to the high complexity and variability of neural data [98, 83]. As a result, another group of decoders is emerging. These decoders use linear methods, such as the Kalman Filter (KF), as the main decoding component in combination with modern non-linear ML techniques to improve accuracy [64, 100, 101, 102, 103]. In this way, these decoders not only utilize the robustness of well-proven algorithms like the KF but also have the improved accuracy of non-linear methods.

The Kalman Filter Algorithm in Detail. The KF is a recursive algorithm that uses a series of measurements over time to predict the state of desired variables [104, 105, 81]. Figure 2.2 reports the main function that is executed at each time step of the algorithm. Each step outputs a prediction state vector ($\tilde{x}_n$), which holds a value for each of the desired variables, and the updated covariance matrix (P_n), which estimates the current accuracy of the results.

We use x as the dimension of the state vector and z as the dimension of the measurement vector. The KF algorithm receives 5 matrices as inputs: *(i)* the previous $P_{x\times x}$, *(ii)* $F_{x\times x}$ – the state transition model, which represents the probability of transitioning from one state to another, *(iii)* $Q_{x\times x}$ – the process noise covariance, which determines the uncertainty in the transition between states, *(iv)* $H_{z\times x}$ – the observation model over the measurements, which models the relationship between the different states and the observations made, and *(v)* $R_{z\times z}$ – the observation noise covariance, which describes the error in the measurements. In the traditional KF, the matrices F, Q, R, H remain constant between consecutive iterations of the KF and constitute the *KF model*. At each iteration,

Table 2.1: The Accuracy of the KF with Different Methods

Accuracy Metric	Gauss [115]	IFKF [109]	Taylor [110]	SSKF [116]	Newton [117]
MSE	3.8×10^{-12}	53.8	0.05	0.1	6.6×10^{-6}
MAE	7×10^{-7}	2.7	0.08	0.06	0.0004
*Max. Difference (%)	0.008	2.2×10^{4}	9.7×10^{2}	5.3×10^{2}	4
*Avg. Difference (%)	0.0001	350	9	4.8	0.035

*These scores are normalized with respect to the original KF output [83].

the KF receives the previous $\vec{x}_{n-1}$, the previous covariance matrix P_{n-1} and a new measurement vector $\vec{z}_n$.

The KF executes the "predict" step and then the "update" step. Calculating the Kalman gain K by inverting the matrix $S_{z\times z}$ lies at the core of the KF computation.

Kalman Filter Design for BCI. With the increase in the number of channels in BCIs [25], the size of the measurements increases. Consequently, the KF requires large matrix operations, particularly the inversion of a large matrix. For edge devices, this requirement is often addressed by a KF hardware accelerator [106, 107, 108]. In many cases, the KF incorporates an approximation method, sacrificing accuracy to meet throughput and power constraints [109, 110, 111]. We aim to design accelerators that introduces no more than ~10% error compared to the standard KF, ensuring adequate control over fine motor tasks with an embedded BCI [112, 113, 114].

Table 2.1 reports KF accuracies when integrated with several candidate computation techniques from literature, which we implemented in software. For all methods, the KF predicts motion based on neural data from the brain of a non-human primate (NHP) [118, 83] and runs for 100 KF iterations. The KF output is compared to the one provided by Glaser *et al.* [83].

Gaussian elimination (Gauss) [115] is the standard method for matrix inversion [46, 108, 107] and provides the most accurate result, as it calculates the matrix inverse directly. However, Gauss suffers from high complexity ($O(n^3)$) and internal dependencies that limit parallel processing. Inverse Free KF (IFKF) [109], Taylor expansion of K (Taylor) [110], Steady-State KF (SSKF) [116] and the Newton-Raphson method (Newton) [117] offer different approximation techniques of K or S^{-1}. IFKF provides the worst accuracy, because it requires dimensionality reduction of the measurements and assumes minimal cross-correlation, despite the high correlation in simultane-

ous neural data measurements [25, 26]. Other methods provide sufficient accuracy when tested on neural data. Nevertheless, Newton provides the best accuracy among these approximations. Our work is the first to embed Newton inside the KF. In Section 8.1, we explain our approach in detail.

2.7 Randomness Generation for BCI Applications

To function as real-world IoT devices, BCI systems must support secure communication and ML-based computation [119, 29, 120, 64, 65, 121]. Security primitives such as AES [122] and RSA [123] are widely used in various applications and rely on random keys to ensure information security and privacy [124]. Kalman Filter is a widely used online learning algorithm for continuous control in BCIs, but it requires periodic calibration or integration with Reinforcement Learning (RL) [64, 100]. RL and other ML algorithms heavily rely on randomness to support adaptiveness [121, 64, 100], enhance the robustness of backpropagation and classification, and reduce training complexity [125, 126, 100].

Random Number Generation in IoT. In practice, random number generation (RNG), which relies on random bit generation (RBG), is typically achieved using algorithms that combine multiple existing sources to harvest entropy. For example, Linux generates randomness from system-level event recordings and the timing of interrupts from various I/O sources [127, 128, 129]. In IoT devices, limited hardware resources and the absence of user inputs make generating random numbers a challenging task [130, 124]. Consequently, traditional approaches that rely on computationally intensive numerical methods or sampling hardware components are often impractical or inefficient [121, 128].

A widely used approach for random number generation in IoT is the use of Physical Unclonable Functions (PUFs) [131, 132, 133]. PUFs leverage manufacturing variations introduced during the fabrication of an integrated circuit (IC), ensuring that no two devices have identical circuit characteristics. A PUF takes an input 'challenge' and combines it with the unique physical properties of the device to produce a 'response' that cannot be replicated by any other device. A strong PUF, which is reliable and capable of generating a large space of responses, can serve as an RNG [133].

However, integrating PUFs into constrained systems is challenging due to unreliability caused by sensitivity to device aging and environmental noise, such as variations in temperature and voltage [134, 133, 135]. Wearable devices, on the other hand, are expected to function reliably in constantly changing environments. To address these challenges, sensor-based PUFs have been proposed for wearable devices [131, 136, 137]. These PUFs are lightweight and rely on algorithms that generate random bits by combining sensor data. Depending on the algorithm and the type of sensor used, this approach can help create a more robust RNG.

The Brain as a Random Number Generator. Empirical evidence of chaos in the brain [138] suggests a degree of unpredictability in the interactions between neurons. However, at lower resolutions, when analysis is conducted on larger regions of the brain, unique patterns emerge that can be used for authenticating individuals [13]. Experiments in neurophysiology [14] and EEG [139, 140] have demonstrated that applying mathematical transformations as RBG algorithms to fine-grained, high-resolution brain data can generate streams of bits that pass statistical tests for randomness [141]. When produced with sufficient throughput, these bits can be used to generate cryptographic keys [14]. By validating the randomness of streams of bits generated from neural data collected through modern electrode-based NIs [31], we can pave the way for integrating a sensor-based PUF that leverages high-resolution brain data in BCI systems.

2.8 Brain-Inspired Neuromorphic Computing

Spiking Neural Network (SNN). The computational model used in neuromorphic computing is the spiking neural network (SNN), which mimics the behavior of the human brain. In particular, an SNN is composed of computational elements called neurons and channels called axons. A neuron is an event-driven state machine controlled by a set of 1-bit inputs. When an input is high, the event is called an input spike. Depending on its state, a neuron can react to a given input configuration by sending an output spike to other neurons via an axon.

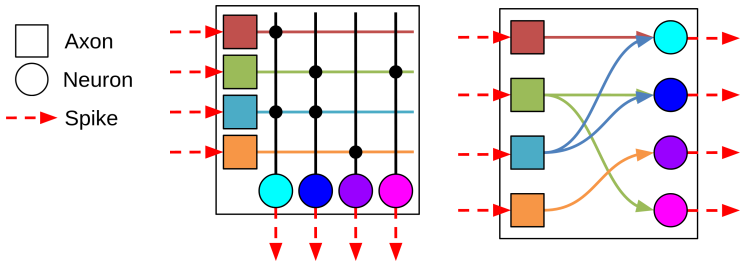
Neuromorphic Cores. In hardware design, the SNN model is split into neuromorphic cores [142, 143], where each core represents a subset of axons and neurons from the SNN. As shown in Fig-

ure 2.3(a), a core implements a crossbar that receives input spikes through its axons. Once an axon receives a spike, the spike triggers all of the connected neurons. If a neuron is ready, it will send an output spike directed to an axon in another core. As shown in Figure 2.3(b), the crossbar can also be viewed as a directed graph that captures the one-way connections between axons and neurons. *In Chapter 9, the term core refers to a neuromorphic core*.

Event-Driven Computation. The computation in an SNN is event-driven. Specifically, the result of a computation depends on the exact timing of events (or spikes). In several architectures, these events are controlled by time intervals delineated by ticks. TrueNorth [142] and RANC [144] use a global *tick control signal* whose pulse notifies all the cores to execute the next step of computation, which involves calculating the next output spikes, and sending them to the corresponding axons. On the other hand, Loihi [143] uses a non-global tick signal. That is, once a core has finished processing the current computation interval, it will synchronize with its adjacent cores to advance to the next interval (i.e. tick).

Open-Source Neuromorphic Hardware. TrueNorth [142] and Loihi 2 [145] are the most prominent neuromorphic chips as of today. Both contain ~1 million neurons, and over 100 million synapses (connections between neurons). However, the designs of these chips are closed-source. In contrast, the Reconfigurable Architecture for Neuromorphic Computing (RANC) [144] has been released open-source and is an excellent baseline design to explore different optimizations for accelerating SNNs with specialized hardware. Similar to the state-of-the-art TrueNorth and Loihi 2, RANC is based on a multi-core architecture that can execute pre-trained SNN models. RANC leverages a similar tick control as in TrueNorth, but unlike Loihi 2, both RANC and TrueNorth do not support broadcasting a spike from one neuron to multiple axons. Thanks to its high reconfigurability, RANC was successfully tested on FPGA with SNN-based CNN classifiers, MNIST [146] image classifiers, and VMM models [147]. Overall, these features make RANC a great candidate for integration in a heterogeneous SoC.

Integration Challenges in a Heterogeneous SoC. The successful integration of neuromorphic hardware as an accelerator in a heterogeneous SoC depends on multiple features. We would like



(a) Crossbar (b) Graph

Figure 2.3: Neuromorphic core depicted as a crossbar and as a graph.

to be able to program the accelerator by means of software applications running on a general-purpose processor [148]. For computational flexibility, the processor should be able to offload the processing of an SNN model to the accelerator at runtime, by transferring input data to the accelerator and receiving output data from the accelerator. To that end, the accelerator interface must be standardized to enable data movement between the accelerator and the rest of the SoC (e.g. main memory and third-party accelerators).

In a heterogeneous SoC, multiple hardware accelerators typically run in parallel after being invoked by a given application. Hence, it is important that a neuromorphic accelerator does not become a performance bottleneck compared to other accelerators. Additionally, if the accelerator is not energy-efficient, the SoC can become unsuitable for real-time applications, as well as for devices in constrained environments [149, 150, 151]. These challenges must be addressed by effectively exploring the design space of SNN accelerators, simplifying their integration in an SoC, as well as prototyping and testing the overall SoC on FPGA [43].

Pre-BCI: Designing Practical and Functional BCI Systems

Chapter 3

Standardization of the BCI System

To build a practical BCI system, we must first focus on the most crucial components that ensure reliable operation while meeting stringent physical constraints. This allows us to establish a baseline, standardized BCI system that can serve as the foundation for future BCI systems. Specifically, we design a standard BCI system to be I/O-centric at the implanted SoC level and computation-ready at the wearable SoC level. At the implanted SoC, we emphasize the integration of a neural interface (NI), wireless communication, and seamless connectivity between the two. At the wearable SoC, we intercept neural data, store it on the device, and prepare the infrastructure necessary for specialized hardware and software to process the data for BCI applications.

Our standard BCI system enables two primary functions: recording and stimulation. Recording involves acquiring neural data via sensors embedded in the neural interface (NI). Stimulation, on the other hand, involves injecting current into neurons interfacing with the NI to elicit a response. Both functionalities are crucial for advancing research and development in the BCI field. Furthermore, these functions must support a degree of configurability to accommodate the diverse types of neural data and stimulation patterns needed to generate a variety of neuronal responses.

In this chapter, I introduce the key components I developed for our standard BCI system. Specifically, I present the architectures of specialized microcontrollers that I designed for both the implanted and wearable SoCs. Additionally, I outline the design process of the BCI system, which can be replicated to streamline future BCI system designs. I played a significant role as part of a large development team that delivered two large-scale, implant-based BCI systems: BISC [21, 49] and IBISX [50], both of which were successfully fabricated and tested *in-vitro* and *in-vivo*.

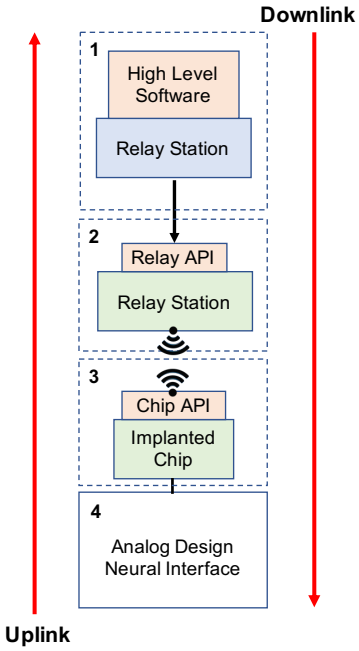


Figure 3.1: BCI system-level flow.

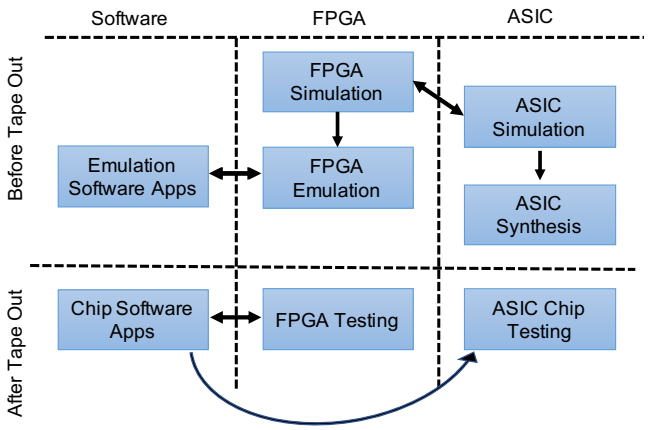


Figure 3.2: BCI system development process.

3.1 System-Level Flow and Design Process

Figure 3.1 illustrates the high-level system-level design flow of our BCI system. In the wearable SoC, referred to as the relay station, software applications running on a general-purpose processor use a specialized application programming interface (API), called the *Relay API*, to initiate instructions implemented within a custom microcontroller on the relay station. This microcontroller operates finite-state machines (FSMs) that trigger instructions from another set of APIs, called the *Chip API*. The Chip API facilitates communication with another custom microcontroller on the implanted SoC. Instructions are transmitted via a short-range radio frequency (RF) link from the relay station to the implanted SoC, where the microcontroller executes these instructions to configure control over the neural interface (NI) and other components of the implanted SoC, enabling wireless transmission and power management. The system supports two data flow directions: the *downlink*, which transfers configuration instructions from the relay station to the implanted SoC, and the *uplink*, which transfers neural data and system status information from the implanted SoC back to the relay station.

Figure 3.2 illustrates the development process. We begin with an FPGA simulation of the system, including both microcontrollers. This is followed by an ASIC simulation of the implanted SoC and its synthesis. In parallel, we emulate the system on FPGA, testing it using software applications running on the general-purpose processor. After the tape-out of the implanted SoC, we combine the fabricated implanted SoC with the wearable SoC (relay station) implemented on FPGA. We then test the integrated system, again using software applications running on the general-purpose processor.

As part of the general system-level flow, the BCI system implements a wireless link protocol comprising both uplink and downlink protocols. The downlink protocol packetizes the information transmitted from the relay station to the implant, while the uplink protocol handles the packetization of information sent from the implant to the relay station. Each packet contains a total of 125 bits. Packets from the relay station are received by the implanted SoC bit-by-bit and stored in a shift buffer, which matches the size of one packet. Decoder modules in both the implanted SoC and the relay station prepare the packets before transmission. These packets carry the information to be transmitted, along with fixed-size segments of bits used for communication synchronization, referred to as 'sync bits,' each occupying a specific position within the packet. Additionally, the decoder adds eight bits of error correction code (ECC), which is computed based on the information in the packet and the sync bits. Encoder modules in both the implant and the relay station probe the shift buffer at each clock cycle, search for the sync bits, and identify when a full packet has been received. Once a complete packet is identified, it is copied from the shift register and forwarded for further processing.

Figure 3.3 presents the communication protocol between the implanted and wearable SoCs in our BCI system, with packet structures for both the downlink and uplink. In both packet types, the parity bits store the ECC code, while the preamble bits remain unused. In the downlink packet, the OPCODE is a unique code that identifies the instruction. The OPERAND bits contain an address to a block in the controller and any additional parameters needed to execute the command. In the uplink packet, the preamble bits synchronize the order of recorded values during recording. The

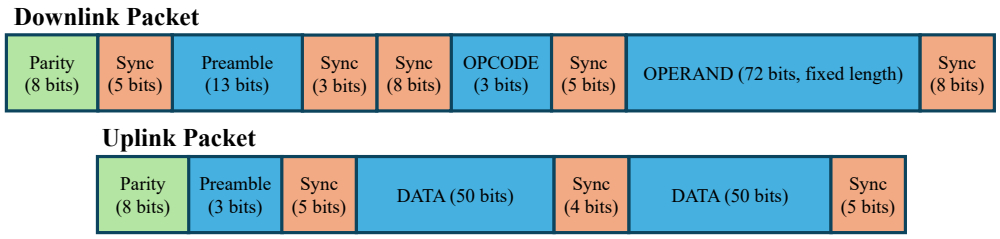


Figure 3.3: Packet structure in the BCI system communication protocol.

DATA bits carry the information from the implant to be processed by the relay station.

3.2 The Implanted Microcontroller

The microcontroller on the implanted SoC manages wireless communication, device configuration, and dynamic neural data acquisition. Figure 3.4 presents a block diagram of the implanted microcontroller marked as logic design. It consists of a main decoder that translates instructions received via the wireless transceiver into actions on the device. The decoder processes one instruction at a time, and while an instruction is being executed, it cannot accept new instructions. The decoder interfaces with five main control blocks:

- **The analog front-end (AFE) control:** Configures static settings for the recording and stimulation functions and generates dynamic control signals to trigger switching activities in the AFE.
- **Gain, ADC and WPT controls:** The gain control configures the settings for the amplifier gain, which receives the neural data samples from the recording channels array. The analog-to-digital converter (ADC) control configures the settings for the 10-bit ADC that takes input from the amplifier. The wireless power transfer (WPT) control configures the settings for wireless power transfer to power all the components of the implanted SoC.
- **The transceiver control:** Receives data samples from the AFE, packetizes them for transmission via the wireless transceiver, and streams the data. During recording, each new sample is added to the next packet. Once a packet is full, it is transmitted, and the next packet begins preparation. The transceiver control can also be configured to packetize and transmit other de-

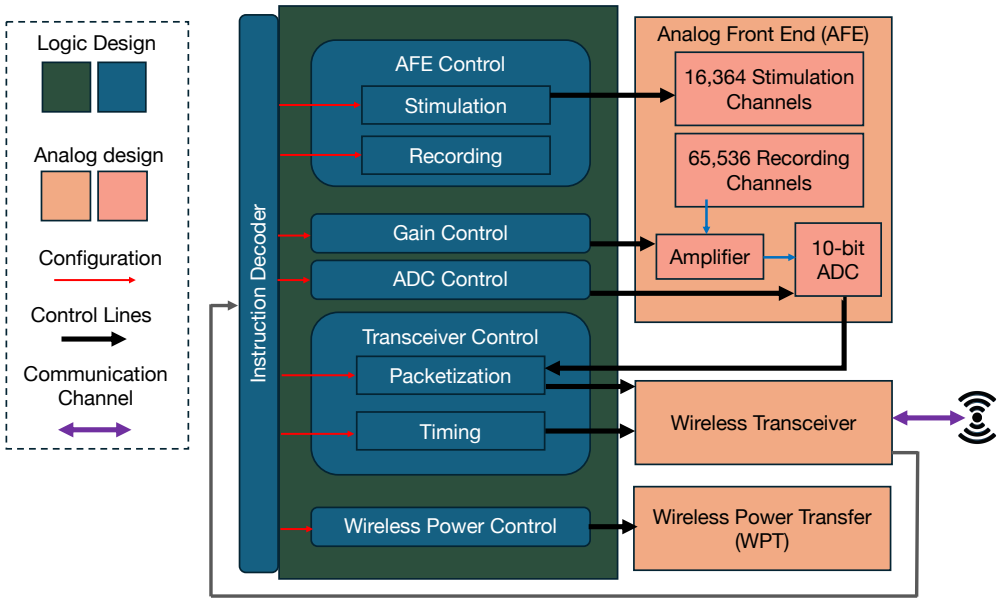


Figure 3.4: Block diagram of the implanted microcontroller (logic design).

vice status information, following a specialized communication protocol.

Table 3.1 summarizes the instruction set architecture (ISA) of the implanted microcontroller, which consists of seven instructions classified into three types: dynamic instructions, static instructions, and query configuration instructions.

Dynamic instructions include recording, stimulation, power-on, and halt. Each of these instructions triggers an event that starts or stops an ongoing process. The recording instruction activates control lines to the AFE, enabling sampling from different sensory channels at every clock cycle. In BISC [21, 49], the stimulation instruction initiates multi-phasic stimulation through a pre-configured subset of electrodes for a set duration. In IBISX [50], stimulation occurs simultaneously with recording, and the stimulation instruction configures registers within the implanted SoC to enable or disable the stimulation pattern during recording. The power-on instruction powers the recording amplifiers, while the halt instruction stops an ongoing recording.

Static instructions include configuration and programming. The configuration instruction sets configuration registers that initialize specialized FSMs in the main microcontroller or are distributed to other functional blocks on the implanted SoC, such as the wireless power transfer block (WPT), the AFE, and the transceiver. These registers define a longer-term state that is updated

Table 3.1: Summary of the ISA of the Implanted Microcontroller (Chip API)

Instruction	Arguments	Description
Recording	Number of samples from the channels	Starts a recording from a pre-configured set of channels
Stimulation	Three time intervals for a stimulation pulse / stimulation pattern	Sends a stimulation pulse from a pre-configured set of channels / configures a stimulation pattern
Configuration	Address for a target block and the configuration values	Configures the blocks in the micro-controller including the control over recording and stimulation
Programming	The coordinates of the pixels targeted for stimulation and their initial polarities	Configures the polarity of the pixels for stimulation (BISC-only)
Query	Address for a specific module	Queries for the current configuration of a specific module in the micro-controller
Power On	N/A	Powers on the analog-to-digital converter (ADC) amplifiers
Halt	N/A	Stops an on-going recording

infrequently. In BISC, the programming instruction sets the specific subset of electrodes used for stimulation and their initial polarity. When stimulation is triggered, this information controls the specific signals sent to the AFE. In IBISX, the programming instruction serves the same purpose as the configuration instruction.

The query configuration instruction allows access to the information stored in the configuration registers of the implanted SoC.

3.3 The Wearable Microcontroller

The microcontroller on the wearable SoC manages wireless communication, internal configuration, and interaction through specialized I/O pins. It sends commands to the implanted SoC, receives responses and recorded neural data, and establishes a connection with an external base station for high-performance processing when needed. The microcontroller includes a main decoder that translates instructions from software running on the relay station. Additionally, the microcontroller has a custom component that processes neural data from the implanted SoC, packaging it in the main memory for access by the software. Recorded neural data are either saved in non-volatile memory on the device or transmitted to a computer base station via wired or wireless Ethernet.

The microcontroller is implemented on the programmable logic (PL) of the FPGA board,

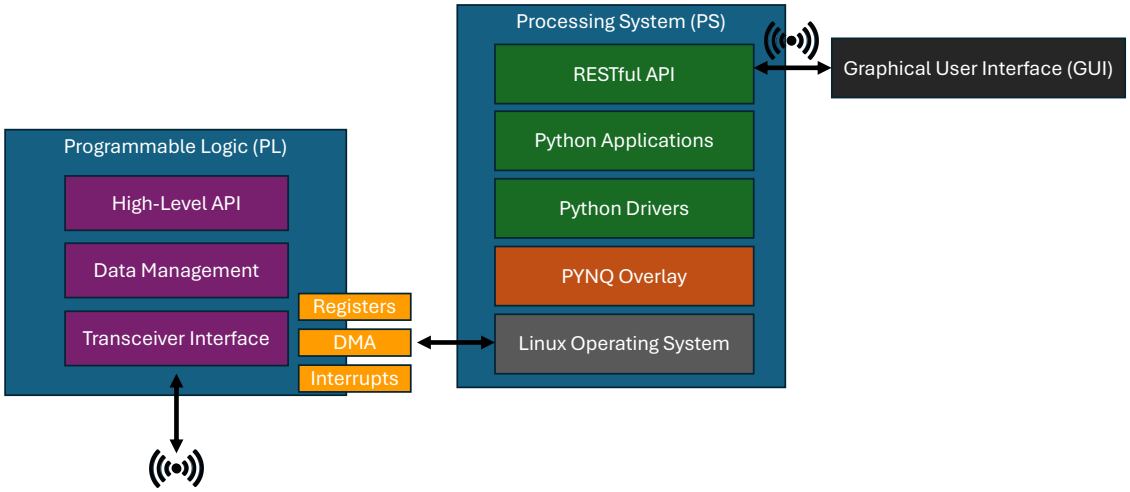


Figure 3.5: Hardware-software stack in the wearable SoC (relay station).

specifically using a Xilinx Zynq-7000 system-on-chip (SoC), which integrates both a processing system (PS) and a programmable logic (PL) unit. The PS features a dual-core ARM Cortex-A9 processor, runs the Linux operating system, and interfaces with a secure digital (SD) card for non-volatile memory. The specialized hardware implementing the microcontroller on the PL includes a high-level application programming interface (API), known as the Relay API, which generates and delivers pre-configured sequences of time-sensitive commands to the implanted device. The hardware implementation of the API is reconfigurable, taking advantage of the flexibility of the PL to allow updates at any time. A Python software framework that utilizes the Relay API runs under the Linux operating system using PYNQ [152] to manage the execution of instructions by the wearable microcontroller.

Table 3.2 summarizes the instruction set architecture (ISA) of the wearable microcontroller (Relay API). Software applications pass instructions to the microcontroller by calling device drivers that set designated memory-mapped IO (MMIO) registers. The microcontroller implements instructions to initiate individual chip-level commands from Table 3.1. Additionally, it executes relay-level instructions to configure parameters within the relay station, such as the address of a memory region for storing incoming neural data, tuning error correction mechanisms, enabling I/O pins, setting communication timeouts, and more. The third type of instruction initiates a sequence of chip-level commands, implemented through specialized finite-state machines (FSMs) that send

Table 3.2: Summary of the ISA of the Wearable Microcontroller (Relay API)

Instruction Type	**Arguments**	**Description**
Chip-level	Same as the chip instruction	Send one command to to be processed by the implanted microcontroller according to the Chip API.
Relay-level	Relay station configuration values	Configures relay station specific attributes that do not directly influence the implanted SoC.
Chip-level sequence	The desired sequence of chip commands and their values	Initiates customized FSMs that automatically send a sequence of commands in a pre-defined order

multiple chip-level commands while relying on synchronization signals between the implanted SoC and the relay station for precise timing.

Figure 3.5 presents the software stack and hardware interface on the relay station. The software stack runs on the processing system (PS), with the top layer being the RESTful API that connects external clients to lower software layers. Commands from the client trigger Python applications that allocate memory via direct memory access (DMA) and set up neural data interception from the implant. Once the setup is complete, the applications use system calls to drivers to access memory-mapped registers, configure hardware in the programmable logic (PL), and retrieve addresses for the allocated memory.

The Python applications and drivers are built on the PYNQ overlay, which abstracts hardware interaction between the PS and PL. After configuring the registers with Relay API command data, the microcontroller in the PL executes the commands, prepares instructions for the implant, and forwards them for processing by the implanted controller. The microcontroller in the PL communicates instructions to the implant through a module interfacing with the transceiver of the relay station. A data management module in the PL intercepts incoming data from the implant and stores it in the allocated memory via DMA. The PS and PL communicate through interrupts, signaling when command execution is complete.

To enable an interactive user experience, a custom graphical user interface (GUI) runs on a high-performance computer base station. The GUI interfaces with the Python framework on the relay station through the RESTful API and displays real-time neural data from the implant.

3.4 Results: Implant-Based BCI Systems

My contributions to the development of a standard BCI system, alongside the contributions of the other members of the development team, have enabled the design of two large-scale, implant-based BCI systems: BISC [21, 49] and IBISX [50]. Both systems were implemented with similar microcontrollers for the implanted and wearable SoCs. Currently, these systems rely on specialized computation executed by software applications running on the PS of the relay station. However, in the future, specialized hardware in the PL can be utilized for hardware acceleration of computation. Both systems are bidirectional BCIs, supporting both recording and stimulation, and offer flexible configuration capabilities.

Figure 3.6 shows the implanted SoC of the BISC system that integrates a neural interface based on a micro-electrode array (MEA). It employs electrocorticography (ECoG) for sensing, where the electrodes interface with the brain cortex without penetration, minimizing brain tissue damage. BISC incorporates 65,536 channels for recording electrical neural data and 16,384 channels for stimulating neurons. Configuration parameters allow recording from selected areas of the MEA at varying resolutions. The data bandwidth limitations of the wireless link in the BISC implanted SoC restrict recording to either 256 channels at $33.9\frac{kSamples}{sec}$ or 1024 channels at $8.475\frac{kSamples}{sec}$.

Figure 3.7 shows the implanted SoC of the IBISX system that integrates an optical neural interface, also called neural imager. It incorporates an array of single-photon avalanche diodes (SPADs) and two sets of micro light-emitting diodes (micro-LEDs). One set of micro-LEDs emits light that induces fluorescent excitation, causing neurons to emit photons when active. These photons are intercepted by the SPADs, which include unique counters that accumulate as more photons are detected. This allows the SPADs to measure the intensity of neural activity in specific neuronal populations. The second set of micro-LEDs emits light that increases neuronal activity, effectively enabling optogenetic stimulation [35]. IBISX integrates 49,152 channels, organized into 256 columns of 192 channels, and includes 768 micro-LEDs. The configuration parameters allow recording from anywhere between one column of channels and the entire array of 256 columns,

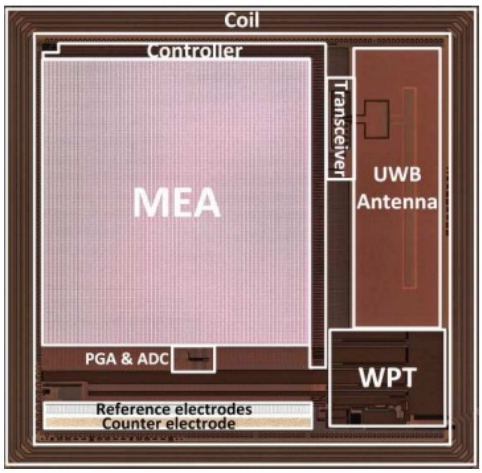


Figure 3.6: The BISC implanted SoC.

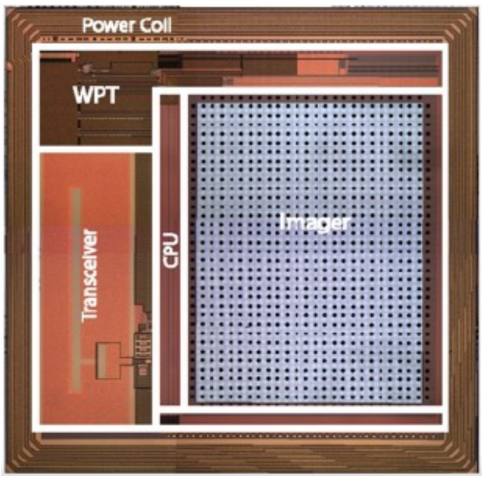


Figure 3.7: The IBISX implanted SoC.

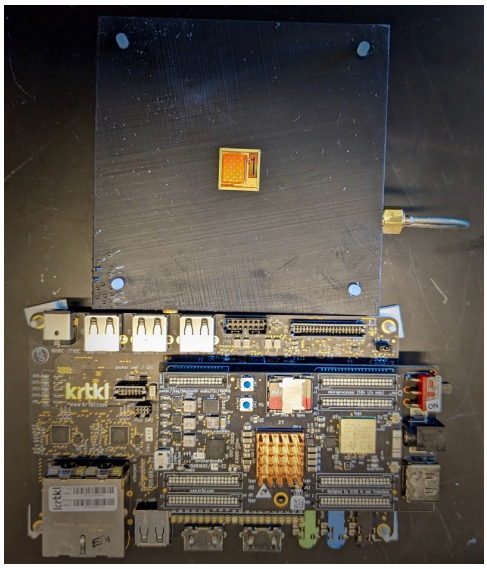


Figure 3.8: The FPGA-based wearable SoC.

with the ability to turn on or off any subset of LEDs. When recording from the full array, the system achieves a recording rate of 176 frames per second.

In both BCI systems, the control flow inside the implanted SoC follows the same general structure. After detecting a voltage spike or intercepting photons, the measurement values are digitized, packetized according to the communication protocol (Figure 3.3), and sent in the uplink direction to the wearable SoC.

The power density of the BISC implanted SoC is $27\frac{mW}{cm^2}$, while that of the IBISX implanted SoC is $33\frac{mW}{cm^2}$. Since both power densities are below the strict $40\frac{mW}{cm^2}$ power density limit for implanted

devices in the brain, both systems are considered safe, as they minimize heat dissipation [40].

Figure 3.8 shows the wearable SoC implemented on the FPGA board, with one of our implanted SoCs shown above it. Configuration parameters are sent in the downlink direction in a similar manner from the wearable SoC to the implanted SoC. The main difference lies in the implementation of instructions that generate chip-level sequences (Table 3.2), which arises from the different ways BISC and IBISX handle recording and stimulation at the neural interface. While in BISC, recording and stimulation are managed as two completely separate functions at the level of the implanted SoC, in IBISX, stimulation is treated as a feature that can be turned on or off during recording. In BISC, chip-level sequences involving both recording and stimulation require synchronization with the implanted SoC and a stalling mechanism, which waits for signals from the implanted SoC before generating the next Chip API instruction. In contrast, in IBISX, stimulation is pre-configured to either be on or off before recording begins. Chip-level sequences in IBISX that involve both recording and stimulation require stopping an ongoing recording, enabling stimulation, and then starting the next recording.

The BISC system has been demonstrated in-vivo [49], with experiments on multiple regions of the brain cortex: the motor cortex of a porcine, the motor cortex of a non-human primate (NHP), and the visual cortex of an NHP. The system successfully demonstrated high-quality chronic recording and accurate decoding of somatosensory, motor, and visual information [49]. The IBISX system has so far been tested in-vitro. However, wired versions of the system, incorporating a similar neural interface design, have been successfully tested in-vivo, specifically targeting the motor cortex of an NHP and validating the advantages of this type of neural interface [153]

Both systems integrate very large-scale neural interfaces and, by enabling wireless communication between the implanted SoC and the wearable SoC, overcome one of the critical barriers to the widespread adoption of BCI technologies: the impractical reliance on wires. The combination of high-density wireless recording, stimulation capabilities, safe operation, and configurability lays the groundwork for the future integration of ML-based BCI applications into the system. This holds immense promise for transforming healthcare by enhancing the usability of BCI systems.

Intra-BCI: Integrating Scalable BCI Applications

Chapter 4

Scalability of BCI Systems and Applications

As implanted BCI SoCs evolve into heterogeneous SoCs that handle more neural data, wireless communication, and computationally intensive applications under physical constraints, their design complexity grows. Thus, understanding the interplay among the main components and establishing a consistent system-level design perspective is crucial for designing safe, effective BCI systems that target real-world BCI applications. Specifically, we address two fundamental questions about the implanted SoC as the main part of the BCI system:

1. How does scaling the neural interface (NI) affect the system constraints?
2. Can we relax the constraints by integrating hardware-based application-level computation?

To answer these questions, I present our work on MINDFUL, a mathematical model that abstracts the combination of a large-scale DAQ, wireless data transmission, and hardware-based application-level computation. We first define the available power budget on the implanted SoC and the real-time constraint affecting all its main components. Then, we introduce the mathematical equations composing the MINDFUL model. The model prioritizes physical safety and provides a lower limit for the total power consumption of the implanted SoC.

We apply our model to published implanted BCI SoCs and analyze their power and area limitations. From our analysis, we realize that future BCI systems must bridge the gap between machine learning (ML) model development and implant-based SoC design. ML models are often developed focusing solely on accuracy by processing large volumes of neural data. In response, the design of BCI SoCs tends to prioritize DAQ and wireless transmission, often relying on non-specialized designs rather than tailoring them to specific applications. We apply our model to state-of-the-art

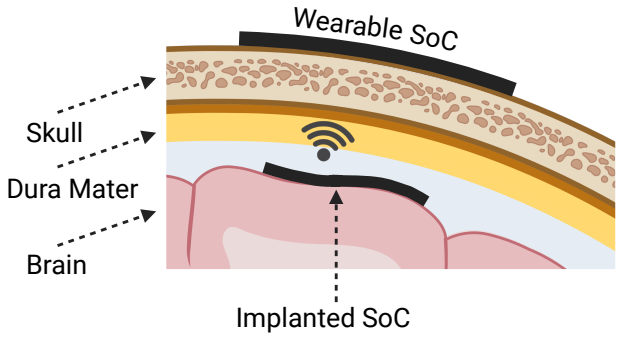


Figure 4.1: A minimally-invasive BCI system.

implanted BCI SoCs and analyze their power and area limitations. Our analysis reveals that future BCI systems must bridge the gap between ML model development and implant-based SoC design. ML models typically prioritize accuracy, processing large volumes of specialized neural data. In contrast, BCI SoC designs focus on general, high-resolution DAQ and high-throughput wireless transmission, often relying on non-specialized architectures instead of tailoring them to specific applications. To advance toward self-contained, implant-based BCI systems, specialized SoC designs for specific BCI applications and ML models tailored to implanted SoC constraints are necessary.

4.1 Modeling Implanted BCI Systems-on-Chip

4.1.1 The Target BCI System

Figure 4.1 presents a BCI system based on a minimally-invasive NI. An implanted chip is positioned on the cortex of the brain in the subdural space between the dura mater that envelopes the brain and the brain tissue. This is our target system, where the implanted SoC communicates with the wearable SoC through short-range wireless communication. The implanted SoC is abstracted as a flat surface that directly interfaces with the brain tissue. This abstraction can be extended to other, less conventional shapes as long as the entire surface area that interfaces with the brain tissue can be calculated.

Form Factor. Compared to other invasive approaches, this form factor offers several advan-

tages. First, post-surgery, the skull is closed and heals naturally, preserving the native biological structure [154]. Second, there is no slit or active opening in the skull, as wireless communication eliminates the need to output wires from the implanted SoC [56]. Third, the implanted SoC directly interfaces with the cortex of the brain, engaging with large populations of neurons and enabling a higher throughput of neural data [39].

Communication. The implanted SoC has the ability to communicate through a custom wireless communication transceiver and antenna that are embedded in the device. The communication is designed to be short-ranged to support better energy efficiency and minimal latency. The implanted SoC must be equipped with a wireless power transfer unit to eliminate the need to frequently replace a battery for the chip [21, 50].

Flexibility and Volumetric Efficiency. Given the non-flat structure of the brain, maintaining a stable interface with cortical neurons requires the implanted SoC to be flexible [38]. Flexibility can be achieved by thinning the chip post-fabrication or using conformable materials [155]. However, the larger the chip, the higher the chance of breaking the chip, and as such, optimizing the area of the chip or splitting it into multiple chips is sometimes necessary [46].

In addition, the area of an implanted BCI SoC is optimized through the metric of *volumetric efficiency* [50, 59, 49]. As the number of channels in the NI increases, the proportion of sensing area on the implanted SoC, where neurons are accessible, must grow at a faster rate than the non-sensing area. Maintaining this balance is essential to allow recording from more neurons.

Data Flows on the Implanted SoC. We assume that each implanted SoC is implemented with a predefined, application-specific pipeline that starts with DAQ and ends with wireless communication. In this case, we consider two potential data flow architectures: *communication-centric* and *computation-centric*.

In Figure 4.2, we present the two data flow architectures. For both architectures, raw neural data from the NI is transferred to the computation stage, followed by the wireless communication stage. In communication-centric, the computation only processes the raw neural data to accommodate the specific wireless transmission protocol on the device. In this case, the transceiver must

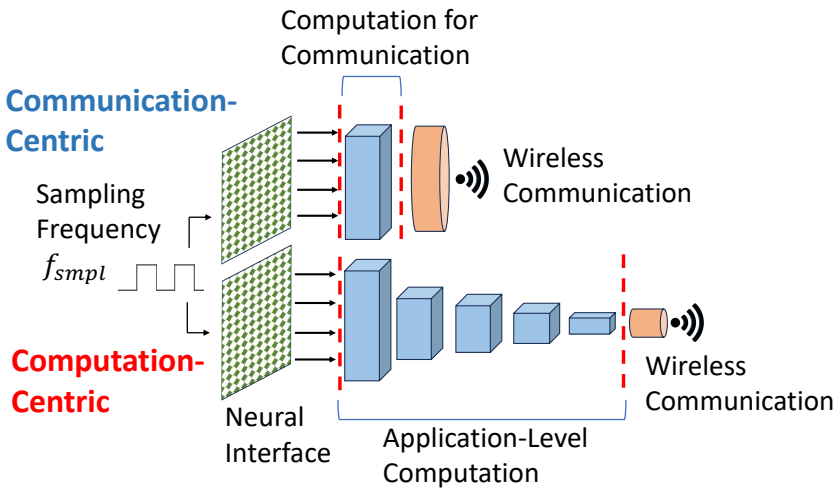


Figure 4.2: Data flow architectures on the implanted SoC.

transmit with a suitable data rate that matches the entire scale of the NI and the sampling rate of the NI (f_{smpl}). In computation-centric, an application-specific computation is embedded in the implanted SoC. This computation operates on the neural data, after which the transceiver transmits the computed results. In many cases, the computation effectively decreases the required transmission data rate. Similarly to communication-centric, computation-centric must accommodate f_{smpl} and process new data at each sampling event.

As we target BCI applications that aim to control external devices, the implanted SoC is required to communicate frequently, transmitting neural data or the computation results to the wearable SoC. The receiving direction is intermittently utilized for tasks such as configuring the SoC and establishing communication protocols [21, 50]. The impact of the receiving direction on the architecture is assumed to be negligible compared to that of the transmitting direction.

We focus on computation using DNNs, which is currently the most promising method for achieving satisfactory accuracy in predicting brain intent [83, 156]. However, our approach is versatile and can be extended to a variety of signal-processing applications.

4.1.2 Power Consumption Limits

Physical safety is a necessary requirement when designing BCI systems. Specifically, the implanted SoC must support long-term use and must not damage the sensitive brain tissue due to high power consumption that results in heat dissipation.

Temperature and Power Density. The brain tissue is among the most heat-sensitive in the human body [157, 41]. Nonetheless, it benefits from one of the highest blood flow rates, which assist in reducing heat [40]. Further research is required to accurately determine the effects of prolonged low-temperature heating on the brain. However, preliminary studies suggest that a temperature increase of up to ~1.5°C, reaching around 38.5°C, may represent the upper safety limit for the brain [40].

The power consumption of an implanted SoC results in heat dissipation from its surface area when higher power consumption leads to increased heat. However, a larger chip surface area can help mitigate a rise in temperature around the device [158]. For this reason, the total power consumption has to match the surface area of the SoC [158]. With the help of the blood flow in the brain, a power density of $40\frac{mW}{cm^2}$ for an implanted SoC is considered as the power limit per unit of area [40].

Solutions for wearable devices tend to be more diverse compared to implantable ones. The power constraint on a wearable SoC within the BAN is much more flexible, and it is typically around $200mW$ [43, 29, 30]. In this context, we assume that the wearable SoC design presents minimal challenges compared to the implanted SoC design.

Power Density into Power Budget. Knowing the maximum allowed *power density* ($P_d = \frac{power}{area}$) for a chip and the area of a chip, we can calculate the maximum power that can be consumed by a chip. We refer to this power limit as the ***power budget***.

Table 4.1 summarizes the key properties of implanted BCI SoC designs that have undergone experiments either *in-vivo* (live subjects) or *ex-vivo* (biological tissue). Only HALO [163, 45] did not report results from such experiments. The designs numbered 1–8 integrate wireless communication and can be abstracted with the form factor presented in Figure 4.1. Designs 9–11 are wired

Table 4.1: Summary of Implanted SoC Designs

#	SoC	NI Type	#Channels	Wireless	In/Ex-Vivo
1	BISC [21, 49]	Electrodes	1024	Yes	Yes
2	Gilhotra *et al.* [50]	SPAD	49152	Yes	Yes
3	Neuralink [22, 154]	Electrodes	1024	Yes	Yes
4	Shen *et al.* [159]	Electrodes	16	Yes	Yes
5	Muller *et al.* [160]	Electrodes	64	Yes	Yes
6	Yang *et al.* [161]	Electrodes	4	Yes	Yes
7	WIMAGINE [162]	Electrodes	64	Yes	Yes
8	HALO [163, 45]	Electrodes	96	Yes	No
9	Wang *et al.* [32]	Electrodes	384	No	Yes
10	Jang *et al.* [57]	Electrodes	1024	No	Yes
11	Pollman *et al.* [36]	SPAD	49152	No	Yes

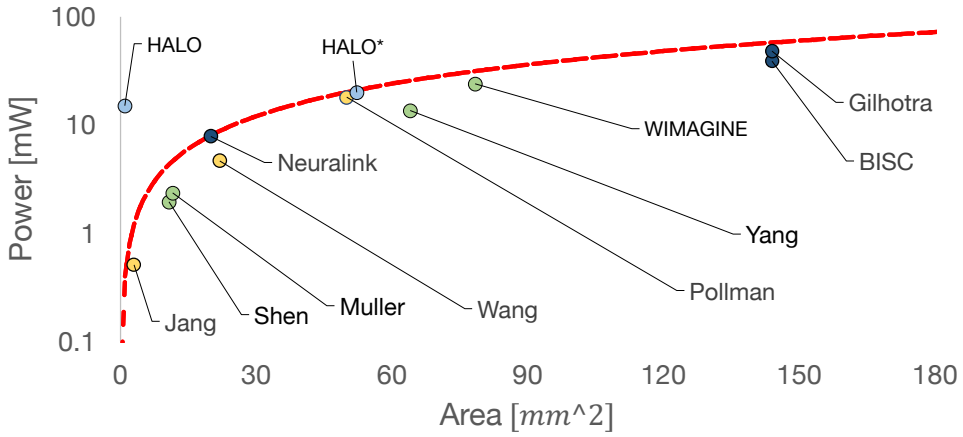


Figure 4.3: Power vs. area of different implanted SoC designs.

and do not support wireless communication.

Figure 4.3 presents the power consumption per area for the implanted SoC designs from Table 4.1. Given that the current standard number of channels is ~1024 channels, we scaled the SoCs designed to support fewer than 1024 channels. This scaling was based on the assumption that the area and power of the SoCs can be characterized on a per-channel basis [22, 32]. We combine this approach with the relationship between area and power in sensory-based designs presented by Simmich *et al.* [164]. However, the reported P_d for HALO had to be scaled down to ensure it fits within the power budget (HALO*). SoCs 2 and 11 can record from up to 49K channels, with a different f_{smpl} for each channel count configuration. We use their nominal parameters for recording from 1024 channels. In Section 4.2, we analyze SoCs 1–8 with our proposed model.

General Power Model. We split the total power of the implanted SoC into sensing power (P_s) and non-sensing power (P_{ns}). While P_s is designated for the operation of the NI, including the power to support recording neural data from the NI and the digitization of the data, P_{ns} includes the rest of the power that supports communication (P_{comm}) and computation (P_{comp}).

As a general rule of thumb, the total power of the implanted SoC (P_{SoC}) must not exceed the power budget (P_{budget}), i.e.:

$$P_{SoC} = P_s + P_{ns} = P_s + P_{comp} + P_{comm} \leq P_{budget} \tag{4.1}$$

Our model estimates P_{SoC} as:

$$P_{SoC} = P_s + \tilde{P}^{app}_{comp}(n_{NI}, t_{comp}) + \tilde{P}^{E_b}_{comm}(d, n_{out}, t_{comm}) \tag{4.2}$$

where $\tilde{P}^{app}_{comp}$ is a function to estimate P_{comp} for a specific application (app), a number of NI channels (n_{NI}) and computation time (t_{comp}); $\tilde{P}^{E_b}_{comm}$ is a function to estimate P_{comm} with a specific transmission energy per bit (E_b), output size for computation (n_{out}), data type size after digitization (d), and communication time (t_{comm}).

An increase in n_{NI} directly increases P_s, followed by an increase in P_{comp} and P_{comm}. In the case of a communication-centric architecture, Equation (4.2) can be simplified with $n_{out} = n_{NI}$.

4.1.3 Scaling the Neural Interface

The NI is the main component of the BCI system, integrating both sensing channels and the circuitry responsible for recording and digitization of neural data.

Figure 4.2 shows that the NI operates under the control of the sampling frequency (f_{smpl}). At each t_{smpl}=$\frac{1}{f_{smpl}}$, n_{NI} channels are sampled for new data. The number of channels integrated in the SoC and f_{smpl} depends on the requirements of the BCI application. Typically, f_{smpl} is set within the range of 1–30KHz [70, 71, 74, 165].

Area per Channel and Power Budget. As illustrated in Figure 4.4, the area of an implanted

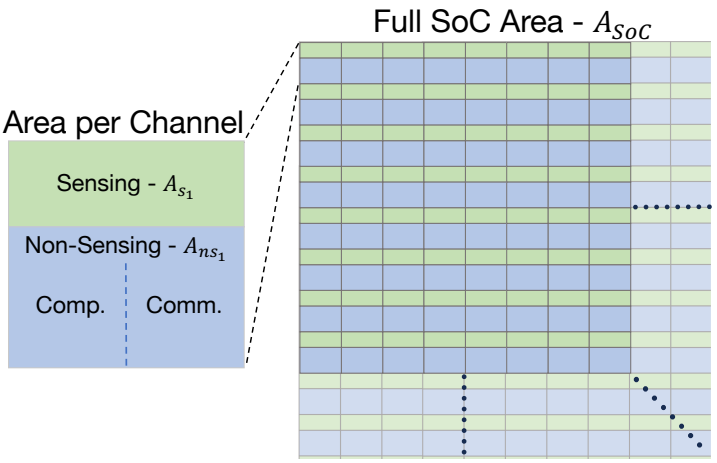


Figure 4.4: Abstraction of area on the implanted SoC.

SoC is usually expressed as the area per channel [22, 32]. The total area of the SoC includes both the sensing area (A_s) and the non-sensing area (A_{ns}). Similarly, we model the area of a single channel as the sum of a sensing area per channel (A_{s_1}) and a non-sensing area per channel (A_{ns_1}). This yields the anticipated total chip area per implanted SoC:

$$A_{SoC} = A_s + A_{ns} = n_{NI} \cdot (A_{s_1} + A_{ns_1}) \tag{4.3}$$

The total P_{budget} for the SoC is given by:

$$P_{budget} = A_{SoC} \cdot P_d^{max} = A_{SoC} \cdot 40 \frac{mW}{cm^2} \tag{4.4}$$

where P_d^{max} is the maximum estimated power density for implanted devices. We calculate the total power consumption (P_{SoC}) based on Equation (4.2). We assume P_s to be linear with respect to n_{NI}:

$$P_s = n_{NI} \cdot P_{s_1} \tag{4.5}$$

where P_{s_1} refers to the power added by adding one channel to the NI, which is assumed to remain constant. This assumption is based on the symmetric structure of integrated channels in modern NIs and the fact that most SoC designs use the uniform terms of area per channel and power per

channel. We assume that most large-scale designs reflect this at the implementation level as well.

Every implanted SoC reports a maximum number of active channels (n_{max}). For a specific design, we assume that estimating A_{SoC} and P_{SoC} for $n_{NI} \leq n_{max}$ is easy and can be calculated linearly with respect to the given design parameters (e.g. A_s, A_{ns}, P_s, P_{ns}).

However, interpolating the area and power of a design with $n_{NI} \geq n_{max}$ is not a trivial task. Above the current standard of 1024 channels, we assume that P_s and A_s continue to scale linearly with the number of channels in the neural interface (n_{NI}), as channels are physically added to the SoC one by one. The parameters of A_{ns} and P_{ns}, which include communication and computation, scale with n_{NI} as well. However, their scaling may vary depending on various design choices in order to avoid exceeding the P_{budget} limit.

4.1.4 The Real-Time Constraint

According to the high-level architecture of the implanted SoC (Figure 4.2), at each recording event, the previous neural data is transferred to the next stage on the implanted SoC and the SoC is ready to receive new samples of neural data. Effectively, this defines a real-time constraint on the implanted SoC, where each stage in the data flow must be executed within t_{smpl}. As a result, Equation (4.2) can be written as:

$$P_{SoC} = P_s + \tilde{P}^{app}_{comp}(n_{NI}, t_{smpl}) + \tilde{P}^{E_b}_{comm}(d, n_{out}, t_{smpl}) \tag{4.6}$$

Assuming communication is isolated as a separate, last stage in the data flow, t_{smpl} determines the transmission data rate (f_d):

$$f_d = \frac{d \cdot n_{out}}{t_{comm}} = \frac{d \cdot n_{out}}{t_{smpl}} \tag{4.7}$$

As for computation, t_{smpl} determines the maximum time that can be spent in each computational stage and the amount of necessary parallelization of processing elements. Specifically, as computation starts immediately after DAQ, the first stage in computation must finish execution within t_{smpl}.

Overall, the real-time constraint has a direct impact on the total power consumption and the total execution time of the data flow on the implanted SoC. At the application level, we assume that the designated BCI application is capable of processing with the throughput of data coming from the NI, and with a total latency that is within the reaction time of the brain.

4.1.5 Scaling Wireless Communication

An implanted SoC does not need to include application-level computation as long as DAQ and wireless communication can work according to the real-time constraint and under P_{budget}.

In this case, all the neural data is transmitted from the SoC after digitization (communication-centric architecture), meaning that the SoC must support the required transmission data rate (f_d from Equation (4.7)), assuming $n_{out} = n_{NI}$. To accommodate this, the antenna on the SoC must support a sufficient bandwidth (BW). For this reason, the antenna has to be able to support a specific bandwidth (BW). Implementing a transceiver that supports a modulation technique is essential to sufficiently utilize this BW.

Energy-Efficient Modulation. Energy-efficient modulation techniques, such as On-Off Keying (OOK), are currently the preferred solutions in implanted SoCs [21, 62, 63].

In the case of OOK, each symbol in the transmission can encode a maximum of 1 bit of information per clock cycle as long as the transmission data rate matches the BW of the available antenna [166]. For instance, if BW=100MHz, an optimal OOK transceiver could transmit at up to f_d=100$Mbps$. However, BW utilization is usually not optimal and depends on various factors in the implementation.

In reality, an OOK transceiver supports wireless transmission with sufficient signal-to-noise ratio (SNR) and a constant energy per bit (E_b) at a maximum data rate that is lower than the physical BW. For example, when a design specifies values of E_b=50$\frac{pJ}{bit}$, n_{NI}=1024, d=10bit and f_{smpl}=8KHz, these parameters correspond to neural data transmission at a data rate of approxi-

mately $82Mbps$. In this case, $\tilde{P}_{comm}^{E_b}$ can be estimated as:

$$\tilde{P}_{comm}^{E_b} = f_d \cdot E_b = \frac{d \cdot n_{NI}}{t_{smpl}} \cdot E_b \tag{4.8}$$

when a larger n_{NI} would require a higher f_d. However, the reported n_{NI} for an SoC design might already correspond to the maximum f_d that the transceiver design could support, while maintaining a constant E_b. If this is the case, Shannon's limit dictates that increasing f_d would likely result in diminishing returns in terms of power consumption and in an escalation of E_b [166, 167].

Design Scenarios. Assuming that the communication components are designed to support a high margin of f_d above that one reported for the SoC, we can then add more channels and increase f_d, while maintaining a constant E_b. In this case, the same antenna and transceiver can be used and we can assume that the total A_{ns} does not need to increase with additional channels. Specifically, we assume $A_{ns_1} = 0$ for each added channel. We define this scenario as the ***high-margin design***.

Assuming the communication components are not designed to support a higher f_d, another option is to increase A_{ns} to support better communication components and a broader BW. In this case, we assume A_{ns_1} to remain approximately equal to the one of the original design. Every additional channel adds a constant P_{ns_1} and a constant A_{ns_1}. This abstraction models a design where each channel operates independently from all other channels. However, as the size of A_{ns} increases linearly, meeting the requirement for flexibility becomes more challenging [38, 155]. Moreover, increasing A_{ns} on the chip reduces volumetric efficiency [59], leaving the neurons beneath this area inaccessible and limiting our progress toward the goal of recording from all neurons in the brain. We classify this option as ***naive design***.

Advanced Modulation. A common approach to support a higher f_d is to use a modulation technique that allows sending more than 1 bit per symbol in one transmission cycle. Quadratic Amplitude Modulation (QAM) is a popular modulation technique that requires a more complicated transceiver design, yet it allows the transmission of multiple bits per symbol [166, 167].

In the case of QAM, E_b changes according to the number of bits per symbol. In this case, we

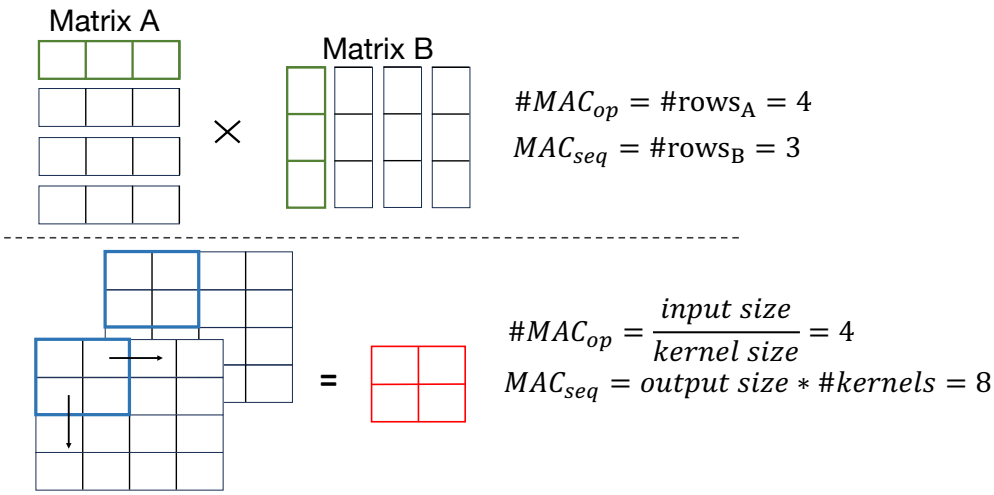


Figure 4.5: $\#MAC_{op}$ and MAC_{seq} example.

assume that the original A_{ns} can also be used for the new implementation and new channels do not increase A_{ns}. In the case of replacing OOK with QAM, this assumption is approximately correct as the same antenna can be used.

In order to estimate $\tilde{P}_{comm}^{E_b}$, we solve the QAM equation and derive E_b for each number of bits per symbol [168, 169, 170]. For example, in a design with $n_{NI} = 1024$, we assume 1 bit per symbol for $n_{NI} \leq 1024$. For $1024 < n_{NI} \leq 2048$, communication will require 2 bits per symbol. For $2048 < n_{NI} \leq 3072$, communication will require 3 bits per symbol and so on. For each interval of 1024 channels we need to solve the QAM equation, derive E_b, and estimate $\tilde{P}_{comm}^{E_b}$. Generally, adding a bit to the symbol is expected to lead to a sharp increase in P_{comm} and, eventually, P_{comm} is expected to make P_{SoC} exceed the available P_{budget}.

4.1.6 Scaling Application-Level Computation

Systems that include DAQ through sensory channels employ specialized hardware for executing application-level computation close to the data source. This strategy increases the system throughput by reducing the wireless transmission data rate [89].

For the BCI system, recent work suggests to embed hardware-based application-level computation at the implanted SoC, in order to address transmission data rate limitations [45, 46].

In parallel, modern BCI applications have shown the best classification accuracy when implemented with DNNs [10, 71, 72, 73, 74, 83]. However, DNNs are computationally intensive and scale super-linearly with respect to their input size [85, 171, 172]. As NIs grow larger, the input size grows as well and the DNN model needs to be scaled to take the additional input into account [25, 26].

We estimate the overhead of implementing DNN-based BCI application-level computation in hardware and adding it to the data flow inside the implanted SoC. Our model is based on calculating the lower bound for hardware overheads in order to check if offloading a hardware-based DNN is theoretically feasible.

Modeling On-Chip Deep Neural Networks. To model the implementation of specialized hardware for DNN computation on the implanted SoC, we start with three main observations:

1. The strict power budget constrains us to first specialize the implanted SoC for a single application and DNN model. In the case where there is enough margin between P_{SoC} and P_{budget}, we can consider adding hardware support for more than one DNN. This aligns with the general trend of creating specialized hardware for a specific application to optimize performance on constrained devices [89, 173, 174, 175].

2. The architecture of the implanted SoC follows the computation-centric architecture as a one-way pipeline. When estimating power consumption, we exclude the overhead of a CPU or centralized main memory, as they are unnecessary for this data flow. This approach is inspired by Ganguly *et al.* [176], who demonstrated that modeling DNNs as non-von Neumann hardware architectures, thereby eliminating the need for a CPU and main memory, can significantly enhance performance.

3. Data dependencies are present between consecutive layers in the DNN, requiring sequential execution in a specific order. In addition, implementing a DNN relies heavily on matrix multiplications and convolutions, which extensively use multiply-and-accumulate (MAC) operations [177, 178]. We rely on these two key attributes when modeling the power, performance

and energy for each layer in the DNN and for the DNN as a whole. While this approach is similar to the one used by NeuralPower [179], we also consider the real-time constraint in the implanted SoC and the cost of customized MAC-based hardware.

We define several terms. A *MAC operation (MAC_{op})* is a sequence of steps, where each step includes a multiplication between two values and an addition with a previously stored value. The number of independent MAC_{op} in a DNN layer is denoted as $\#MAC_{op}$. The *MAC sequence (MAC_{seq})* is the length of a sequence within a MAC_{op}. These metrics, MAC_{op} and MAC_{seq}, represent our MAC-based approach to characterize the computation in DNN layers, which is based on the more general polynomial-based approach given by NeuralPower [179] for describing the operations within DNN layers (e.g. matrix-matrix multiplications and convolutions).

Figure 4.5 shows a simple example. At the top, a matrix-matrix multiplication between $A_{4\times3}$ and $B_{3\times4}$ can be characterized as $\#MAC_{op}$=4 and MAC_{seq}=3. At the bottom, a convolutional layer with two input channels, one output, kernel size of 4, and output size of 4 corresponds to $\#MAC_{op}$=4 and MAC_{seq}=8.

Formulating the Computational Model. By definition, all the MAC_{op} within an individual layer in the DNN are independent and can be computed in parallel, provided that all inputs are available at the start of the computation. In addition, MAC_{seq} is the same for every MAC_{op} within a specific layer.

We assume that the hardware architecture assigns a *MAC hardware unit (MAC_{hw})* for each MAC_{op}, requiring that the corresponding MAC_{seq} is executed by the same MAC_{hw}. When allowed by the timing constraints, one MAC_{hw} can sequentially execute multiple MAC_{op}. This feature enables a lighter, less complex design that may help meet the physical constraints.

We estimate the computation power with respect to the minimal number of MAC_{hw} that we need in order to compute the DNN up to a maximum time (T_{max}). In case that the whole DNN needs to be executed within the real-time constraint, the model sets $T_{max} = t_{smpl}$. For a DNN with N layers, we formulate the process of finding a hardware-based DNN architecture that meets our

needs as a search-optimization problem:

$$[MAC^i_{seq}], [\#MAC^i_{op}] = f^{app}_{MAC}(n_{NI}, DNN),\ i \in [1, N] \tag{4.9}$$

$$E_i = MAC^i_{seq} \cdot t_{MAC}, \quad O_i = \#MAC^i_{op} \tag{4.10}$$

$$T_i = E_i \cdot \left\lceil \frac{O_i}{\#MAC_{hw}} \right\rceil, \quad \sum_{i=1}^{N} T_i \leq T_{max} \tag{4.11}$$

$$\#MAC_{hw} > 0, \quad \#MAC_{hw} \leq max(O_i) \tag{4.12}$$

where f^{app}_{MAC} in Equation (4.9) is a method that outputs the $\#MAC_{op}$ and MAC_{seq} per layer for an application with a specific DNN architecture (DNN). The MAC_{op} and MAC_{seq} of the i-th layer are denoted as MAC^i_{op} and MAC^i_{seq}, respectively. Equation (4.10) defines t_{MAC} as the computation time of one step with one MAC_{hw}. In Equation (4.11), the total runtime of the i-th layer is T_i. The constraints in Equation (4.12) are based on the assumption that computation with $\#MAC_{hw} = 0$ is not possible, and $\#MAC_{hw}$ should not exceed the maximum $\#MAC_{op}$ in a single layer. The second constraint is based on our assumption that a complete MAC_{op} must run on the same MAC_{hw}. The final objective is to minimize $\#MAC_{hw}$.

In accordance to Equation (4.2), the final estimate for the lower bound of P_{comp} is given by:

$$\tilde{P}^{app}_{comp} = \#MAC_{hw} \cdot P_{MAC} \tag{4.13}$$

where P_{MAC} is the power consumption of a single MAC_{hw}.

In scenarios where we allow pipelining of the DNN at the level of layers, Equation 4.11 and Equation 4.12 can be replaced with:

$$T_i = E_i \cdot \left\lceil \frac{O_i}{\#MAC^i_{hw}} \right\rceil, \quad max(T_i) \leq T_{max} \tag{4.14}$$

$$\#MAC_{hw} > 0, \quad \#MAC_{hw} = \sum_{i=1}^{N} \#MAC_{hw}^{i} \leq \sum_{i=1}^{N} O_i \tag{4.15}$$

where $\#MAC_{hw}^{i}$ is the number of MAC units for the i-th layer in the DNN. In this case, the new objective is to minimize $\#MAC_{hw}^{i}$ for each layer. Now, the constraint in Equation (4.15) dictates that $\#MAC_{hw}$ must not exceed the total $\#MAC_{op}$ in the DNN.

Scaling the DNN Computation. In the case of an implanted SoC with application-level computation, the SoC needs to transmit the relatively small final output of the DNN. We expect a significant reduction in P_{comm} in exchange for spending more on P_{comp}, especially when the DNN is scaled for very large numbers of channels. We are interested in understanding if under some circumstances, scaling P_{comp} can be more beneficial for handling the constraints of the SoC compared to the scaling of P_{comm} alone.

To model a realistic effect of larger input data on the size of a DNN as a result of increasing n_{NI}, we scale the DNN size according to the increase in input size. Our scaling approach is based on the fact that in most classification DNNs, the output size of the DNN is not affected by the input size. As the output is usually a group of statistical values, where each of the values corresponds to one label from a fixed set of labels, changes in the input size are expected to change the results of the output but not the number of output values. Nonetheless, changes in input size are expected to affect the structure of the intermediate layers in the DNN [26, 85, 171, 172].

We reflect the relationship between the new inputs and the old inputs in terms of layer size and network depth (number of layers). For a given DNN architecture that matches a neural dataset and a specific n_{NI}, we scale the size of every layer and adjust the network depth according to a parameter, denoted as α, which represents the ratio between the original n_{NI} and the new n_{NI}.

Splitting a DNN in the BCI System. DNN architectures, and specifically convolutional neural networks (CNNs) and multi-layer perceptrons (MLPs) [180, 181, 182], are based on a one-way dataflow that propagates through a sequence of layers. Given that the communication between the implanted SoC and the wearable SoC is wireless, short-range, and supports high data rates [21, 50], a straightforward system-level optimization would be to partition the computation between

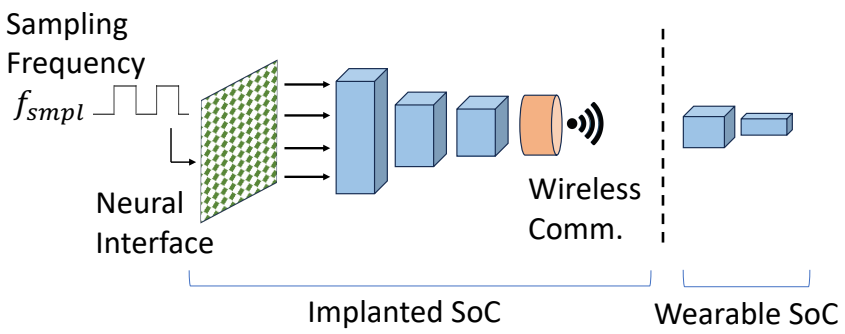


Figure 4.6: Splitting a DNN in the BCI system.

the implanted SoC and the wearable SoC.

Figure 4.6 illustrates this by fitting only a subset of the DNN on the implanted SoC, while the other layers are present on the wearable SoC. This system-level optimization can help reduce P_{comp} on the implanted SoC. However, as we go backwards in the DNN, the output size of the last layer present on the implanted SoC grows and with it also the required f_d. For this reason, with the reduction in P_{comp} we also expect a rise in P_{comm}.

4.2 Model Evaluation: Case Studies

4.2.1 Methodology

We consolidate the mathematical equations and concepts from Section 4.1 into a unified model, which we call MINDFUL. The MINDFUL model enables us to estimate the lower bounds of power and area overheads for both communication-centric and computation-centric architectures of implanted BCI SoCs. For computation-centric architectures, we report the best-performing design, whether pipelined or non-pipelined. We use MINDFUL to analyze SoCs 1–8 from Table 4.1, as they support wireless communication. As baselines, we use the nominal power and area parameters reported for each SoC design in their corresponding papers, scaled as described in Section 4.1.2. To model DNN-based computation, we use an architecture of a DenseNet CNN (DN-CNN) and an architecture of a Multi-Layer Perceptron (MLP), both trained for speech synthesis from ECoG neural data [74]. These DNN types represent the current standard for modern application-level

computation in BCI applications [183, 184]. The original DN-CNN includes 10 layers, while the original MLP consists of 4 layers. These DNNs were originally designed to process input data from n_{NI}=128, sampled at f_{smpl}=2KHz. The output of both DNNs consists of 40 labels, representing 40 speech frequencies that can be used to generate audio [74].

4.2.2 Scaling the Neural Interface with Communication-Centric Architectures

Since the original SoC designs are communication-centric and do not include application-level computation, the reported value of P_{ns} for each SoC includes P_{comp}, which is required to prepare the data for transmission, along with P_{comm}, which is required for transmission. We calculate P_{SoC} based on Equation (4.5). Then, we calculate P_{budget} according to Equation (4.3) and Equation (4.4).

Energy-Efficient Modulation. We model the two design scenarios described in Section 4.1.5: naive and high margin.

Figure 4.7 shows the results of P_{SoC} normalized to P_{budget} for different values of n_{NI}. The top plot shows the results for the naive design and the bottom plot shows the results for the high-margin design. Each bar in the plots is labeled with the corresponding SoC number and is divided into P_s and P_{ns}. For the naive design, the results show that scaling n_{NI} does not change the ratio between P_{SoC} and P_{budget}. This happens because, in the naive case, n_{NI} introduces linear scaling of both P_{SoC} and A_{SoC} by the same factor, which results in a consistent margin of unused P_{budget}. For the high-margin design, ultimately, P_{SoC} exceeds P_{budget} for all SoC designs. This occurs because P_{ns} increases linearly at a rate higher than A_{SoC}, eventually causing P_{SoC} to exceed P_{budget}.

Figure 4.8 shows the results of A_s normalized to the total A_{SoC} for various values of n_{NI}, offering a measure of volumetric efficiency (Section 4.1.1). The left plot displays the results for the naive design, while the right plot shows those for the high margin design. For the naive design, the relative A_s does not improve with the increase in n_{NI} due to A_{ns} increasing at the same rate as A_s. This implies that a naive design does not improve volumetric efficiency for large-scale NIs, which significantly limits neuron accessibility. Arguably, the naive design is equivalent to scaling

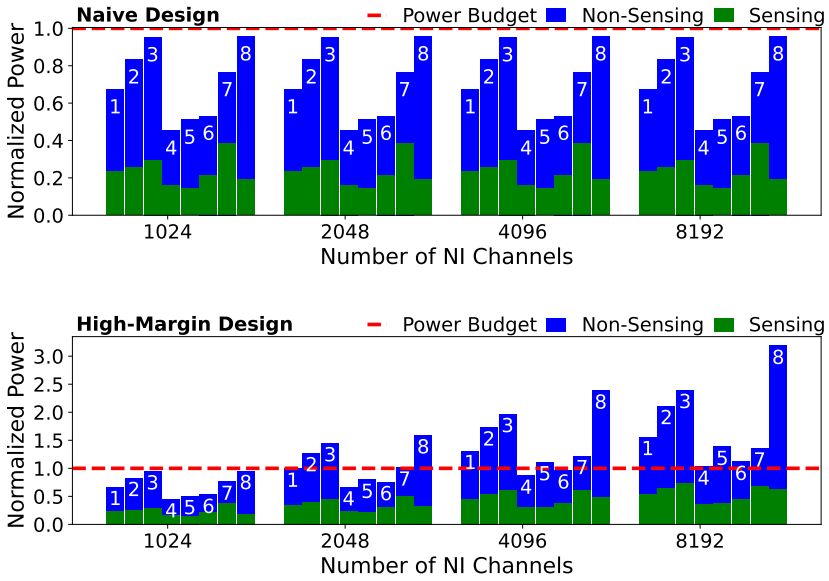


Figure 4.7: Normalized SoC power vs. number of channels.

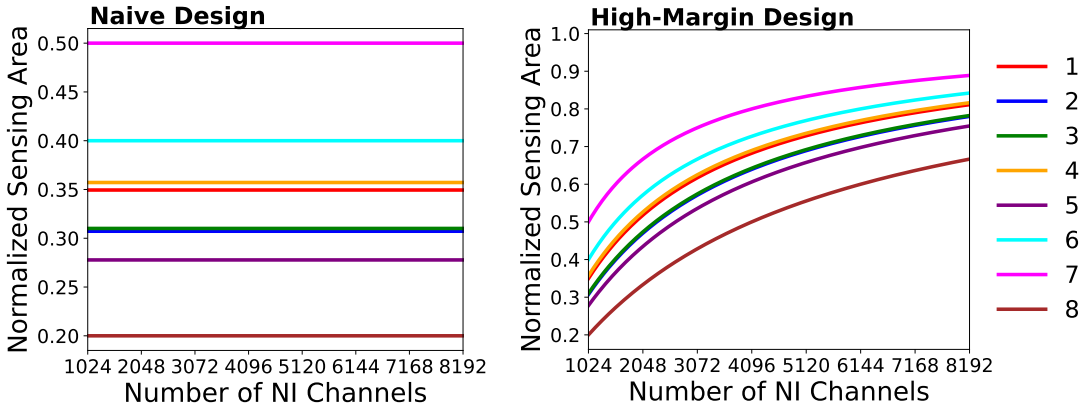


Figure 4.8: Normalized sensing area vs. number of channels.

the number of implanted SoCs, which leads to the same issue of failing to optimize volumetric efficiency. From this metric, the high-margin design is preferable compared to the naive design, while the normalized A_s grows and becomes the dominant area on the SoC.

These two designs illustrate two contradictory approaches: the naive design offers optimal power management on the SoC but results in an infeasible A_{SoC}. The high-margin design achieves better area management but infeasible P_{SoC}. Neither of these design approaches is viable in practice and an intermediate approach is necessary to strike a better balance between area and power.

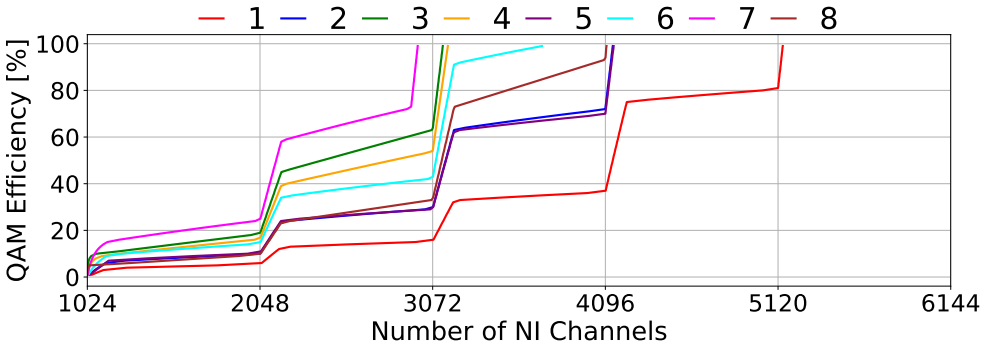


Figure 4.9: Scaling with advanced modulation (QAM).

Advanced Modulation. For advanced modulation, similarly to high-margin design, we assume that above n_{NI}=1024, A_{ns} must not be increased due to limitations of volumetric efficiency. Instead, the existing A_{ns} can be used for the implementation of QAM.

For each n_{NI}, we solve the QAM equation to determine E_b for the required data rate [166, 167]. We assume nominal QAM parameters of BER=10^{-6} , path loss=60dB, and margin=20dB [168, 169, 170]. In our analysis, we focus on *QAM efficiency*, a design parameter that quantifies the power efficiency of the QAM implementation, where higher efficiency results in reduced P_{comm}. However, current QAM designs achieve an efficiency of only ~15% [168, 169, 170].

Figure 4.9 shows the minimum required QAM efficiency in percentage for each SoC design and n_{NI} value. For each n_{NI} value and SoC, the reported QAM efficiency is the minimum achievable, while keeping P_{SoC} within the P_{budget}. Sharp increases in QAM efficiency indicate the addition of 1 bit to each symbol in every QAM transmission cycle (Section 4.1.5). For 20% QAM efficiency, most SoC designs cannot support more than n_{NI}=3000, while the average n_{NI} is only ~2200. However, if high QAM efficiency can be achieved, SoCs 2, 5, and 8, for example, can support up to 4000 channels, a capacity that is not achievable with the high-margin design. In this case, most SoCs demonstrate the potential to integrate 2 to 4 times more channels (n_{NI}) than what is possible with the high-margin design.

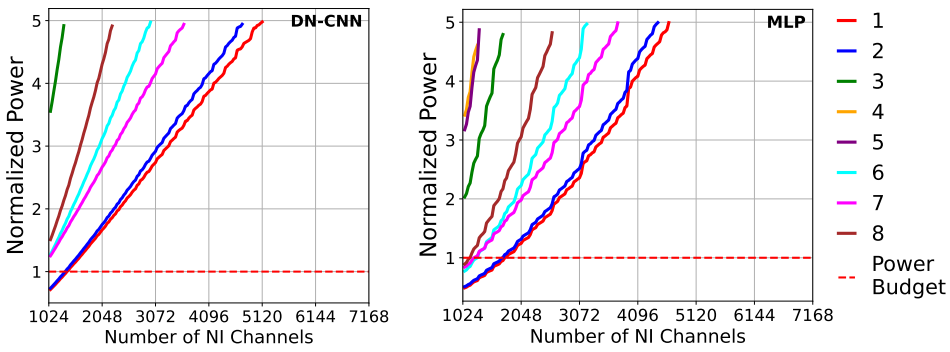


Figure 4.10: DNN-based power consumption.

Overall, advanced modulation can address the increased data rates resulting from larger NIs. However, achieving this requires overcoming challenges in QAM efficiency. Ultimately, even an ideal, yet impractical, QAM efficiency would not suffice to sustain the transmission of all neural data using current SoC designs. To support the continued scaling of NIs, it is necessary to reevaluate implanted SoC designs. This includes reducing P_s and ensuring that the additional P_{ns} from increased communication does not push the total P_{SoC} beyond the P_{budget}.

4.2.3 Scaling the Neural Interface with Computation-Centric Architectures

We use MINDFUL to model the integration of application-level computation, which reduces transmission data rate and eliminates the need for advanced modulation techniques. We assess whether DNNs can be integrated into the implanted SoC designs from Table 4.1. We replace the original P_{ns} reported for each SoC design with the calculated result from MINDFUL, which includes $\tilde{P}_{comp}^{app}$ for the DNN model and the required $\tilde{P}_{comm}^{E_b}$ for wireless transmission.

Offloading DNN Models. We estimate the power consumptions of MLP ($\tilde{P}_{comp}^{MLP}$) and DN-CNN ($\tilde{P}_{comp}^{DN-CNN}$).

Figure 4.10 shows the total P_{SoC} normalized to P_{budget} for each SoC and scaled for different values of n_{NI}. On the left, the results of DN-CNN, while the results for MLP are on the right. For n_{NI}=1024, some SoC designs are unable to integrate the DNNs without increasing P_{SoC} beyond the P_{budget}. For DN-CNN, some SoCs require P_{SoC} that exceeds by 5 times the P_{budget}, and therefore,

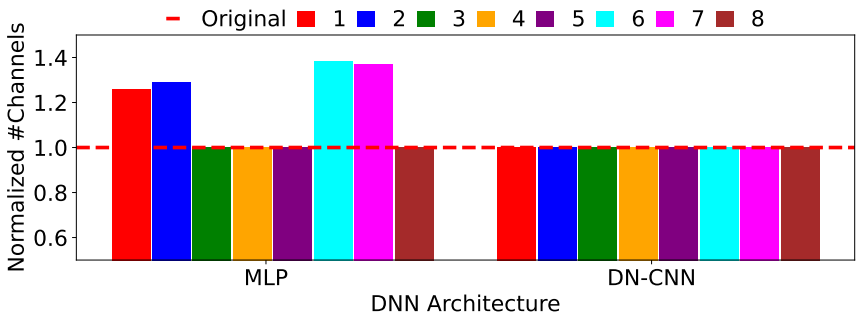


Figure 4.11: Layer reduction optimization.

they are not shown in the plot. The other SoCs are capable of integrating the DNNs, with the best results supporting n_{NI}≈1800 with MLP and n_{NI}≈1400 with DN-CNN. In comparison to using advanced modulation, an average of n_{NI}≈1800 can be achieved with a QAM efficiency of only 13% (Section 4.2.2).

Without optimizations, current DNNs are not suitable for integration on implanted SoCs. Even with an optimistic lower bound for their power overhead that excludes additional integration-related overheads, the implanted SoCs that integrate the DNNs cannot scale to even twice the standard channel count (1024).

Splitting DNN Models. For each SoC, the system-level optimization of splitting the DNN model is applied only if the output size of the final layer on the implanted SoC is small enough to be transmitted at the originally reported data rate for the SoC (Figure 4.6).

Figure 4.11 presents the results of this optimization. We name it ***layer reduction***. The results show the increase in the maximum n_{NI} that the implanted SoC can support when the optimization is applied, normalized to the maximum n_{NI} without the optimization.

The best result is a 40% increase in n_{NI} for SoC 6 with MLP. However, with DN-CNN, the SoC designs did not experience any benefit from layer reduction.

The main reason for these results is that DNNs typically progress from the largest output per layer to the smallest. By offloading only part of the DNN onto the implanted SoC instead of the entire DNN, we reduce P_{comp}. However, by removing the last layers, the output of a partial DNN is often larger than that of the final layer, which increases the data rate required for transmission.

As a result, P_{comm} increases, contributing to a higher total P_{SoC}.

These results suggest that while modern DNNs can improve the accuracy of BCI applications, it is not feasible to implement such models on implanted BCI SoCs without additional optimizations. Network size and implementation must be optimized specifically for BCI applications, ensuring they are as lightweight as possible to fit within the constraints of implant-based BCI systems.

4.2.4 Potential Optimization Strategies

The MLP and DN-CNN, in their current form, are not suitable for hardware implementation on implanted SoCs due to the high P_{SoC} they introduce. Even with the optimization of splitting the DNN layers, satisfactory results remain unachievable. However, we would like to apply additional optimization strategies in order to specialize the implanted SoCs for the DNN models, and enable a successful integration of BCI systems and applications:

- **Neural Data Reduction.** For some applications, using BCIs to record from all accessible neurons has shown to be redundant due to temporal and spatial dependencies between the activities of different neurons [25, 26, 74]. For this reason, computational methods such as neuronal spike sorting [185] are commonly used to reduce the amount of neural data [57, 22, 60, 186]. These methods filter out the data from inactive neurons. Generally, these methods are better suited for implant-based BCI systems than standard compression techniques, as they eliminate the need for additional memory. We modify the value of α from Section 4.1.6 in order to mitigate the effect of scaling the NI on the scale of the DNN. This parameter effectively reduces the size of the DNN [180]. We refer to this optimization as ***channel dropout***,
- **Technology Node.** From our earlier results, hardware implementations of DNNs are expected to be very power consuming. To address this, the MAC_{hw} units can be designed with a more advanced technology node [187, 188]. This optimization is expected to significantly decrease P_{comp} and allow the mapping of larger DNN models on the implanted SoC. However, implanted BCI SoCs are designed mostly with analog circuits (in both the NI and the transceiver), which

do not easily benefit from advanced technology nodes [189]. We refer to this optimization as ***technology scaling***.

- **Density of Sensing Area.** To improve the NI resolution and promote volumetric-efficiency, channels are being integrated with increased density [24]. This reduces A_{s_1} and contributes to the flexibility of the chip [38]. However, it is important to remember that when the total chip area is reduced the power budget decreases [40]. We refer to this optimization as ***channel density***.

We apply the optimizations to the implanted SoCs in four steps:

1. The channel dropout optimization: `ChDr`. We search for the minimum dropout value, while P_{SoC} is still under the P_{budget}.
2. We add layer reduction to mitigate the dropout and allow an increase in DNN model size: `La+ChDr`.
3. We scale the technology node to 12nm instead of 45nm by re-synthesizing the MAC_{hw}, retrieving the updated t_{MAC} and P_{MAC}: `La+ChDr+Tech`.
4. We use 2× the channel density in the NI: `La+ChDr+Tech+Dense`.

Figure 4.12 shows the MLP model size after applying each of the optimization steps, normalized to its original size without optimizations. The results were obtained with n_{NI}= 2048, 4096 and 8192, using the minimum channel dropout required to ensure that P_{SoC} remains within P_{budget}. From left to right, the bars represent the addition of an optimization step for each amount of n_{NI}.

For 2048 channels, `ChDr` reduces the DNN model size by an average of 68% to ensure it fits within the constraints of the BCI system. For 4096 channels, `ChDr` reduces the model size by an average of 94%. For 8192 channels, the average reduction is 98%. In most cases, `La+ChDr` allows maintaining a larger MLP model size. Applying `Tech` reduces P_{SoC} in all cases, allowing for an increase in the acceptable DNN model size. However, for 8192 channels, the average model size remains only 6% of the original. Applying `Dense` reduces A_{SoC}, improving volumetric efficiency, but also leads to a lower P_{budget}. For 8192 channels, only an average of 2% of the original DNN model can fit within the BCI system.

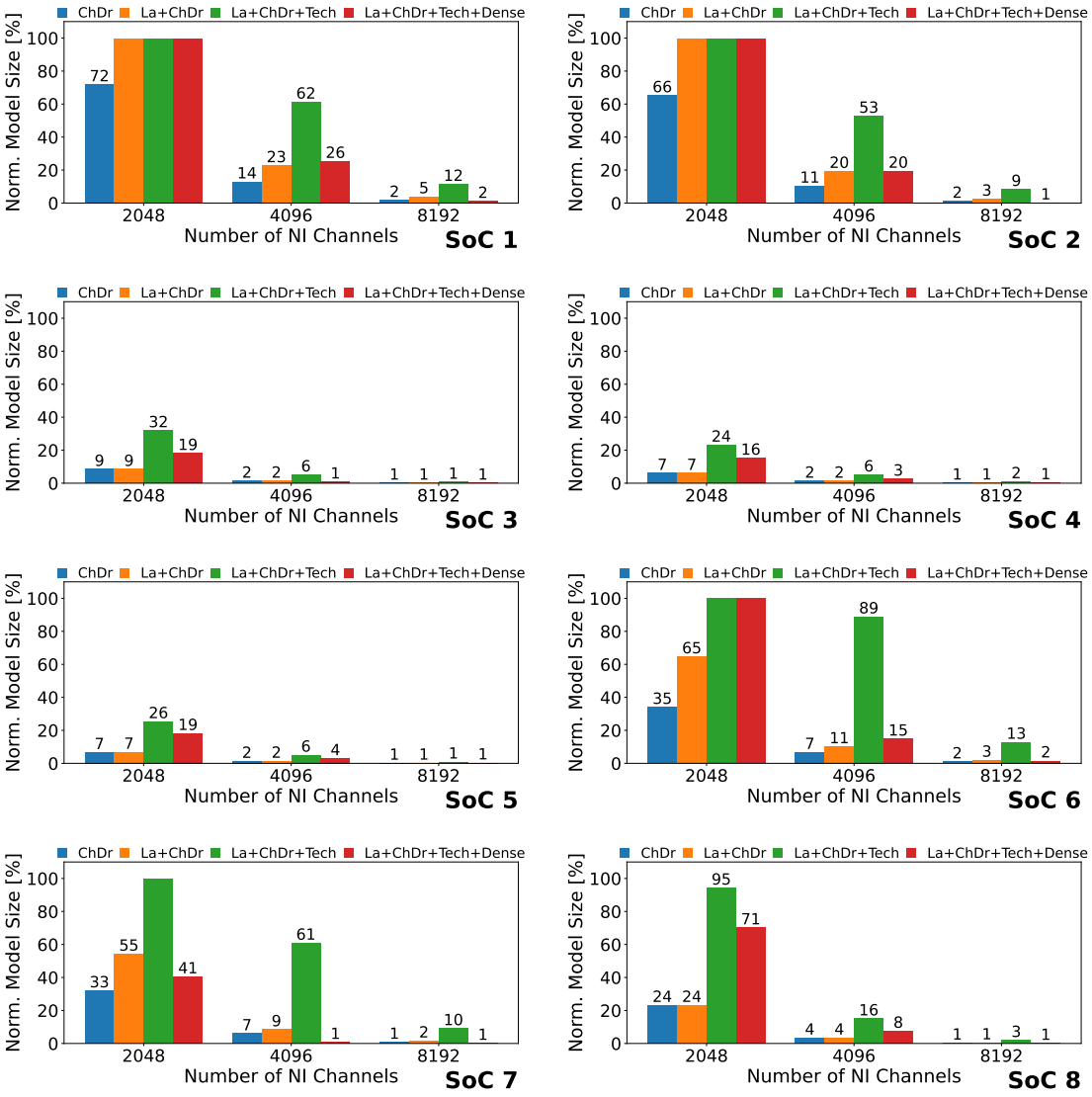


Figure 4.12: Feasible MLP model sizes on the implanted SoCs with respect to the number of channels.

These results emphasize that as the NI scales, BCI-based DNN models must be significantly reduced in size, often to a fraction of their original size, to enable effective integration into the BCI system. System-level optimizations can assist in specializing BCI systems to specific BCI applications and in adapting DNN hardware implementations for implanted SoCs. However, modern DNN models for BCIs need to be redesigned to be lightweight in order to bridge the gap between application demands and system constraints and pave the way for self-contained, implantable, and safe BCI systems.

4.3 Related Work

State-of-the-art implant-based BCI systems focus on the integration of large-scale neural interfaces with thousands of channels [56, 60, 154]. Neuropixels [32, 60] is among the most widely used devices for recording brain data in research settings. The implanted SoC uses a wired connection to external processing [190]. In contrast to wireless BCI systems, wired systems meet different power constraints. The design of Neuralink [22, 154] integrates wireless communication and on-chip spike detection [185]. The implanted SoC replaces part of the skull and interfaces with the brain through wires.

BCI systems have been implemented with the ability to execute real-time ML-based computations [43, 44, 45, 46, 77]. Some systems implement these computations for a relatively small-scale NI of around 100 channels [45]. Other systems scale by employing multiple implanted SoCs to solve a task [46]. However, recent results suggest that these designs would not scale well at the level of an individual SoC with a large-scale NI [87]. For non-invasive BCIs, it was suggested to decompose the computation into smaller operations and execute closer to the data source [77]. Some assume that the implanted SoC can transmit all the neural data in real-time, with computation implemented on the wearable SoC [43, 44]. This solution is possible in case we can support higher data rates with wireless communication or reduce the data rate with hardware-efficient methods for finding patterns in neural activity [186].

Research has been done to define low-power thresholds for implant-based, computation-capable BCIs [88, 156, 191]. Saad *et al.* [191] discuss the importance of integrating low-power, embedded decoding for motor imagery in ECoG-based BCI systems. Silowowski *et al.* [156] analyze the impact of neural data size on DNN models. They show that additional neural data can improve model accuracy for DNNs, but this is not always the case. They recommend to evaluate the significance of added neural data, and to minimize the DNN model size.

Hueber *et al.* [88] presents a benchmark for estimating the overhead of hardware implementations for close-loop BCI computations. In their analysis, they focus on linear methods, artificial

neural networks, and spiking neural networks (SNNs). To estimate the power consumption for each computation, they calculate the number of MAC operations and memory accesses. They conduct experiments with a neural interface with up to 192 channels. They conclude that a hardware implementation of a BCI computation requires a careful consideration of the trade-offs between complexity, latency, and power consumption. They highlight the advantages of implementing SNNs for close-loop BCIs for a better power consumption.

In this work, we focus on open-loop BCI applications, which, in contrast to close-loop applications, rely heavily on wireless communication to interact with external devices. However, the approach to modeling MAC operations is similar across both types of applications. In the future, MINDFUL can be extended for closed-loop BCIs, as well as other computational models.

Chapter 5

Integration of BCI Applications in the BCI system

In a complete BCI system, real-time, scalability, and power constraints must be addressed for the algorithms that analyze brain signals. Real-time analysis of these signals allows the BCI system to use the results to control actuators or deliver feedback to the brain.

Since the integration of computation at the implanted BCI SoC is not yet critical, as custom implantable communication IPs can still transmit all neural data from the neural interface (NI) in real-time [21, 50], the design focus can shift to integrating computation within the less constrained, wearable BCI SoC. This approach still necessitates hardware implementations of BCI algorithms as part of complete BCI systems operating within the body area network (BAN) environment [29]. This represents a necessary intermediate step between proof-of-concept software implementations of machine learning (ML) for BCI and the practical, real-world integration of these computational workloads as close as possible to the neural data source within the implanted BCI SoC. As the number of channels in the NI becomes too large for real-time wireless transmission, these BCI computations must be moved from the wearable BCI SoC closer to the data source, into the implanted BCI SoC. Initially integrating these algorithms into hardware at the wearable BCI SoC serves as a critical step toward achieving this final goal.

For integration in a wearable BCI SoC, and assuming that communication adds negligible runtime overhead, ML-based BCI applications must achieve the real-time requirement of completing execution within 0.18 seconds, which is the average reaction time of the brain [192], while adhering to the low-power constraint of ~200mW for wearable devices in the BAN [30].

To achieve this goal, I present our work on MasterMind [43], a many-accelerator system-

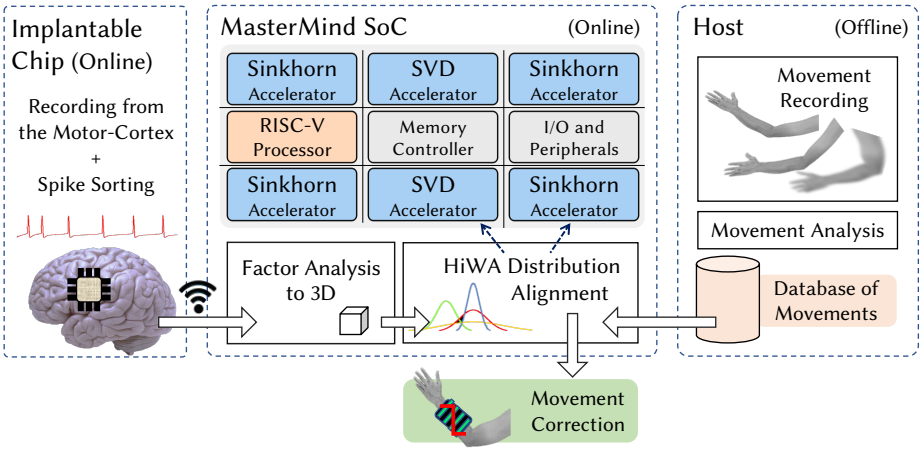


Figure 5.1: The MasterMind many-accelerator SoC in a BCI system.

on-chip (SoC) architecture specialized for the wearable BCI SoC. As illustrated in Figure 5.1, MasterMind integrates a RISC-V processor and multiple instances of accelerators. The accelerators are implemented based on our proposed multi-objective design-space exploration (DSE) methodology, which combines explorations at both the accelerator and SoC levels. The number of accelerator instances in the SoC is configurable at design time, enabling workload parallelization across these instances. MasterMind employs advanced point-to-point (P2P) communication among accelerators, allowing them to operate in a pipeline without software synchronization overhead. This approach significantly reduces memory accesses, resulting in substantial performance and energy-efficiency benefits. MasterMind enables the online deployment of Hierarchical Wasserstein Alignment (HiWA), a promising ML algorithm that can effectively decode brain signals into body movements [65].

We deploy multiple versions of the MasterMind SoC on FPGA and evaluate their performance in executing HiWA by invoking the hardware accelerators. The FPGA-based experiments demonstrate the advantages of our solution compared to software executions with the RISC-V CVA6 processor [193, 194], the ARM Cortex-A53 processor, and the Intel i7 processor.

Overall, MasterMind demonstrates a methodology for successfully integrating BCI algorithms into the wearable BCI SoC. This work represents a step toward standardizing the development process for hardware-based computation in practical BCI systems.

5.1 SoC Architecture

Target System. Figure 5.1 shows the target BCI system for movement correction. In an offline phase, body movements are recorded, analyzed, and stored in a database. In an online phase, an implantable chip, attached to the motor-cortex of the brain, records neural data using customized electrodes integrated in the NI [42]. The data is first processed with spike sorting [185] to identify the occurrences of action potentials (spikes). In our target system, we assume that spike sorting is implemented in hardware and takes place at the level of the implanted chip [195].

The implanted chip sends data through wireless communication to the wearable device hosting the MasterMind SoC. A factor analysis application is executed by the processor tile on the SoC to transform the data from a highly multi-dimensional representation into a 3D representation. Then, MasterMind executes the HiWA algorithm that compares the brain data to different body movements loaded from a database. Eventually, this process finds the best matching movement, and the result can be forwarded to an external device as part of a full BCI application for correcting movement disabilities. Alternatively, the results of HiWA can also be useful as part of a BCI application that executes a feedback loop back to the implanted chip, which actuates electrical stimulation through electrodes attached to the brain [21]. In both example applications, the real-time constraint imposed by the reaction time of the brain [192] should be enforced on the latency of the application when executed in the BCI system.

MasterMind features a scalable tile-based SoC architecture. This architecture combines multiple accelerator tiles with one processor tile, one memory channel and one I/O tile to manage various peripherals. In particular, two accelerator tiles implement the two main computational kernels of the HiWA algorithm: Sinkhorn and SVD.

Our SoC comes with the full software stack for offloading the HiWA computation to the accelerators from a Linux user-space application. In Section 8.3, we show FPGA-based experiments on MasterMind SoCs with up to eight accelerators.

SoC Design Platform. The design of MasterMind leverages ESP, an open-source platform for

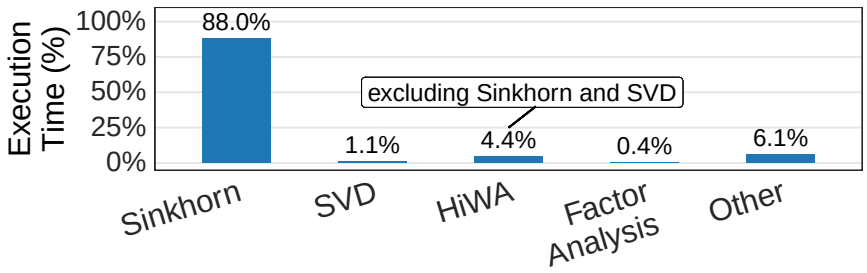


Figure 5.2: Profiling of the HiWA application.

the agile design of heterogeneous SoCs and their prototyping on FPGA [196, 197, 198]. Given a set of hardware accelerators, ESP allows the designer to build a tile-based SoC with the desired mix and location of accelerator tiles, processor tiles, memory tiles (each containing a channel to main memory) and an I/O tile that manages various peripherals. ESP features a multi-plane network-on-chip (NoC) as the main on-chip interconnect. We select the open-source 64-bit RISC-V CVA6 [193, 194] as the main processor.

Software Profiling. We profile the current state-of-the-art Python implementation of the BCI application executing the HiWA algorithm to find the best candidates for hardware acceleration [95]. In Section 8.3, we also evaluate our own optimized C++ implementation. Figure 5.2 shows the execution time breakdown of running the HiWA application on the provided dataset. The Sinkhorn algorithm is the most time consuming task, accounting for 88% of the 12 seconds of total execution time. Instead, the SVD algorithm accounts for 1.1% of the execution time. The remaining part of HiWA contributes an additional 4.4%, while the factor analysis step amounts to 0.4%. Finally, this application spends about 6% of the time for pre-processing tasks, extracting the data clusters and scaling them. The pre-processing tasks are not part of HiWA and can be tackled with various software implementations, which are not the focus of this work.

Based on the software profiling, we implement hardware accelerators for Sinkhorn, the main bottleneck of the algorithm, and for SVD, a component that works in a loop with Sinkhorn. We embed the rest of the computation steps that contribute to the 4.4% of the execution time into our accelerators. Together, the accelerators allow us to isolate most of the computation and offload it into hardware. We design a separate accelerator for SVD due to its computation complexity, short

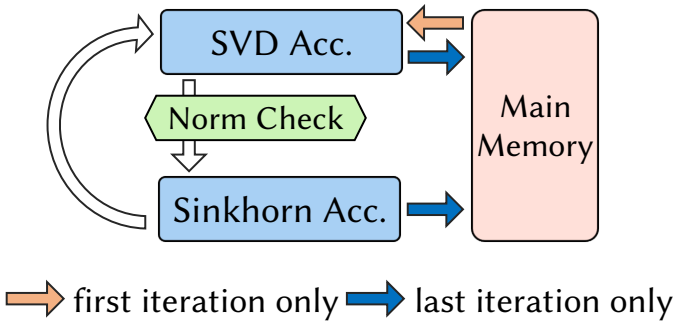


Figure 5.3: Dataflow in the MasterMind SoC.

execution time, and reusability by other algorithms.

Data-Flow in the SoC. Figure 5.3 depicts the data-flow of the HiWA algorithm running on the SoC. The SVD accelerator reads the data from the main memory and generates the inputs for the Sinkhorn accelerator, which in turn generates the next input for SVD. This process continues in a loop until matrix R, which is one of the outputs of SVD, satisfies a norm check. Finally, Sinkhorn stores the results back into main memory. This dataflow can be parallelized over multiple accelerators for a faster execution at the cost of larger energy consumption.

Accelerators Communication and Invocation. The accelerator tiles follow a loosely-coupled accelerator model [148, 199]. They are designed independently from the rest of the system, are connected directly to the system interconnect with a configurable socket, and execute coarse-grained tasks. The socket is configured through memory-mapped registers and handles tasks such as address translation, cache coherence, direct memory access (DMA), and point-to-point (P2P) communication on behalf of the accelerators. P2P communication enables direct data transfer between the accelerators and isolation of their computation from the processor and main memory [200]. On top of the basic P2P hardware support provided by ESP, we developed an advanced hardware-software co-design strategy to support P2P and allow accelerators to work independently in a loop with minimum software assistance. At invocation time, the configuration of the accelerators specifies across different iterations which data should be accessed with P2P and which data should be accessed regularly in main memory.

We leverage the ESP software API to develop Linux device drivers for invoking our accelera-

tors, and set up an on-chip direct P2P communication. The result is a major reduction of off-chip memory accesses, with consequent performance improvements and energy savings (Section 8.3).

The support for P2P in MasterMind can be very useful for a BCI system that executes long iterative computations involving multiple processing elements. For instance, Karageorgos *et al.* [45] suggest the main processing elements that are currently needed for common BCI tasks and their interactions with one another. These tasks in addition to HiWA, spike sorting, and factor analysis can be decomposed into several accelerators in MasterMind. The decomposition helps meeting low-power constraints for individual tasks, while direct data transfer significantly reduces interactions with the processor and main memory, and increases the general performance of the system.

5.2 Accelerators Design

Figure 5.4 and Figure 5.5 show the architectures of our accelerators. We design Sinkhorn in SystemC and synthesize it with Stratus HLS. We design SVD in C++ and synthesize it with Vivado HLS. The accelerators include private local memories (PLMs) to store inputs, outputs, and partial results [201]. Each accelerator can be configured to interact with main memory or directly with another accelerator for sending and receiving data. The PLMs are implemented as multi-bank memories that expose multiple read/write ports, and their size is configurable at design time. The accelerators have several parameters, implemented as memory-mapped registers. They can be divided between operational parameters and communication parameters. The operational parameters are used as constants in the computation kernels. The communication parameters define how the accelerator communicates with the rest of the SoC and which functions must be executed in a particular invocation. In this way, we can handle dynamic changes in the communication between accelerators at run-time and support the P2P feature of the SoC.

Each architecture consists of three main modules operating concurrently in pipeline [202]. The *load* function receives the input data into the PLMs; it uses a multiplexer to select the correct PLM according to the current offset in memory and the operational parameters. The *compute* function processes the data, and the *store* function emits the output data. Both accelerators use 32-bit fixed-

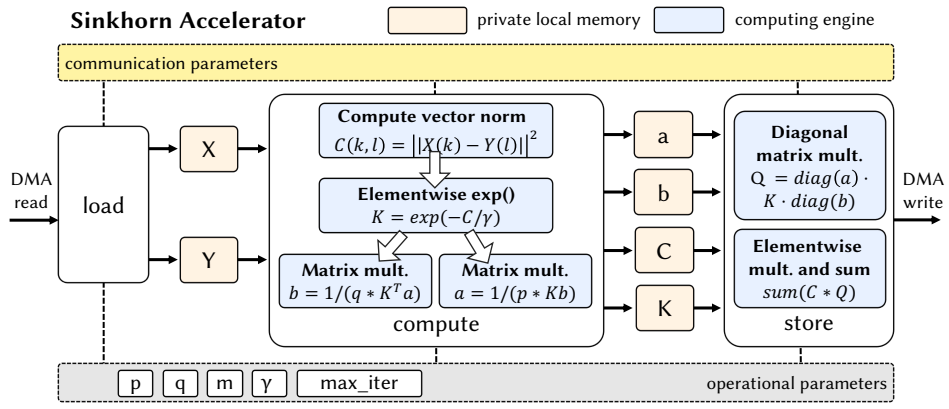


Figure 5.4: The hardware architecture of the Sinkhorn accelerator.

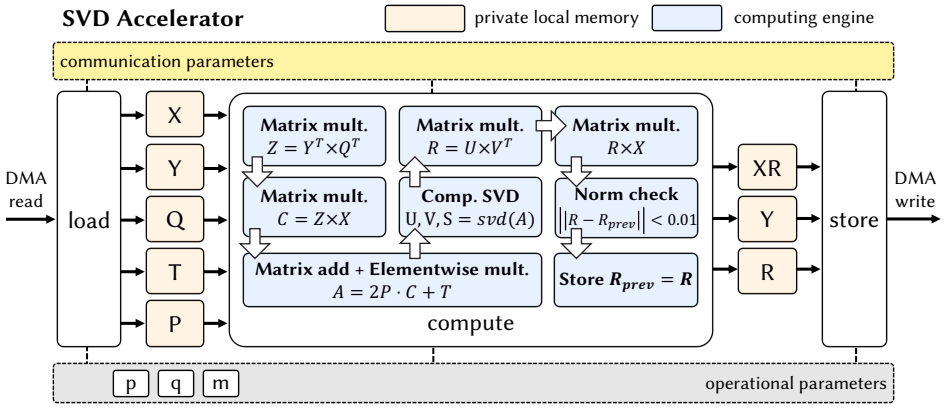


Figure 5.5: The hardware architecture of the SVD accelerator.

point datatype to improve performance and energy efficiency, without sacrificing the computational accuracy of the accelerators ($\geq$ 97%) compared to software implementations [95].

Sinkhorn Accelerator. The accelerator implements the Sinkhorn algorithm that is responsible for most of the execution time of HiWA. The operational parameters of the accelerator are: p, q, m, γ, and max_iter. They define the sizes of the matrices that the accelerator takes as input, the regularization parameter (Section 2.5), and the total number of iterations.

Figure 5.4 shows the architecture of the Sinkhorn accelerator. The accelerator loads the input matrices X and Y. Matrix C is computed by parallelizing the computation of the vector norm between 3D-points in X and Y. To compute elements of matrix K, we implement an efficient module for the $\exp(-x)$ operator as the approximation of the Taylor series. The module updates an initial value in a loop according to the recursive Taylor series formula, and uses only one 8-bit

divisor to reduce area, a 32-bit multiplier, and a 32-bit adder.

The accelerator repeatedly uses K to calculate vectors a and b. The calculation of a depends on b and vice versa; it is done for a total of max_iter iterations, unless the vectors have converged. We pipeline most of the computation by using multi-port memories to implement a, b, and K. The use of costly division operators was mostly avoided by reusing constants ($1/\gamma$), scheduling the division at the last stage of the computation, and implementing a Taylor series module for the inversion operator $1/x$. This module receives an initial value as input, and uses only one multiplier and one adder to calculate the output recursively. Compared to using a regular 32-bit divisor, this method significantly reduces area costs.

The final step is the calculation of matrix Q from K, a, and b, by transforming the vectors a and b into diagonal matrices and by multiplying them with K. Our diagonal matrix multiplication module multiplies three matrices, two of them diagonal, in a fully pipelined manner, while taking advantage of the unique structure of diagonal matrices. When an element of Q is ready, it is stored in a double buffer, multiplied by an element in C, and added to the total sum of the elementwise multiplication. The double buffer is used to preserve the full throughput of the pipeline. It masks the memory access latency in *store* by overlapping communication with computation, and it eliminates the need to store the entire output in a PLM.

SVD Accelerator. The accelerator rotates the cluster with the data from the brain before feeding it back to Sinkhorn. The operational parameters of the accelerator are m, p, and q, which define the sizes of its input matrices.

Figure 5.5 shows the architecture of the SVD accelerator. The accelerator loads as input four matrices (Q, X, Y, and T) and one scalar P. It calculates matrix A as the result of a chain of matrix operations: $2P(C) + T = 2P(Z \cdot X) + T = 2P(Y^T \cdot Q^T \cdot X) + T$, while the main bottleneck of the operations is the multiplication of matrix Y^T with matrix Q^T. For this operation, we use a large factor to unroll the main loop of the multiplication and gain a significant speedup. For other multiplications we use intermediate settings for a good balance on all resources, and used pipelined loops for additions and elementwise multiplications.

The accelerator uses the linear algebra method $svd()$ on matrix A in order to generate its singular vectors matrices U and V. Then, it computes the rotation matrix $R = U \cdot V^T$. The $svd()$ method is another bottleneck of the accelerator as it is an iterative operation with data dependencies. We customized $svd()$ by balancing the different initiation intervals of the module to achieve an efficient pipelined implementation.

Finally, the accelerator rotates X by calculating $X \cdot R$. At the end of the computation, a norm check on R determines the convergence of R, and *store* writes the output into memory.

5.3 Design-Space Exploration

Exploring trade-offs between performance and area is crucial for a BCI system that must meet real-time performance requirements while adhering to strict area and power constraints. A broad DSE at the accelerator level generates a diverse set of implementations, enhancing the reusability of accelerators under varying constraints. At the SoC level, a compositional DSE produces a family of Pareto-optimal SoCs, identifying configurations that improve efficiency in coarse-grained joint computation among accelerators and in their communication within the SoC [202].

DSE at the Accelerators Level. In Sinkhorn, the *compute* function is divided into several computation modules that are executed in an efficient pipeline (Figure 5.4). Inside the modules, we refactor the implementation to expose additional parameters that determine the parallelism of read/write operations and multiplications.

After obtaining a baseline implementation for the Sinkhorn accelerator, we explore its design-space. Specifically, we apply different levels of loop unrolling, loop pipelining, function inlining, custom datapath components, different sizes for the local memories, and different implementations of the division operators in order to avoid resource overheads.

To explore the design-space of the SVD accelerator, we use the C++ traits made available by Vivado HLS, and we customize the implementations of the matrix operations modules.

In all the configurations, the Sinkhorn accelerator expects an input size of 5KB and an output size of 160KB. The SVD accelerator expects an input of 160KB and an output of 5KB. We record

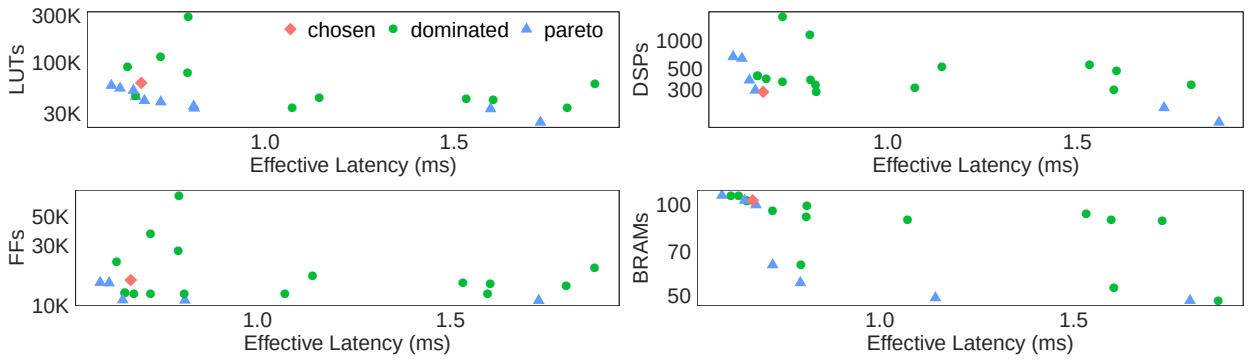


Figure 5.6: Design-space exploration for the Sinkhorn accelerator from four different resource perspectives.

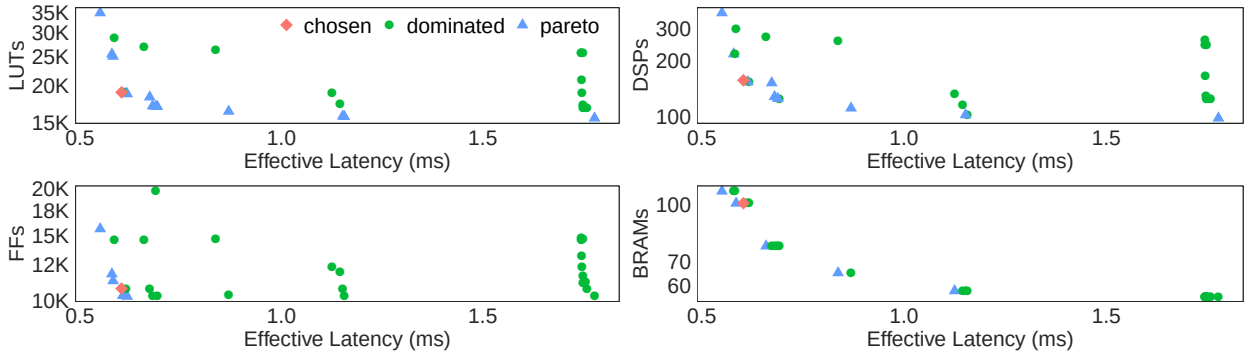


Figure 5.7: Design-space exploration for the SVD accelerator from four different resource perspectives.

the latency as obtained by a cycle-accurate co-simulation with Stratus HLS for Sinkhorn, and Vivado HLS for SVD, with Xilinx Virtex UltraScale+ VCU118 as the target FPGA board. We collect the resource usage reported by Xilinx Vivado after synthesizing the accelerators.

Figure 5.6 and Figure 5.7 show the results of our DSE for Sinkhorn and SVD, respectively, where each design point is characterized in terms of effective latency and FPGA resources (LUTs, FFs, DSPs, and BRAMs). Each configuration for both SVD and Sinkhorn uses different combinations of knobs, resulting in implementations with unique performance and area trade-offs. For each accelerator, we retrieve a set of Pareto-optimal designs for a given FPGA resource.

The availability of a set of Pareto-optimal accelerator designs enhances the reusability of our accelerators across various BCI systems with different physical constraints.

DSE at the SoC Level. The final goal is to integrate our accelerators in a complete SoC. Therefore, we propose a novel approach to evaluate all the possible combinations of the two accelerators. Our approach is based on simulating the computation time, while assuming the accelerators inter-

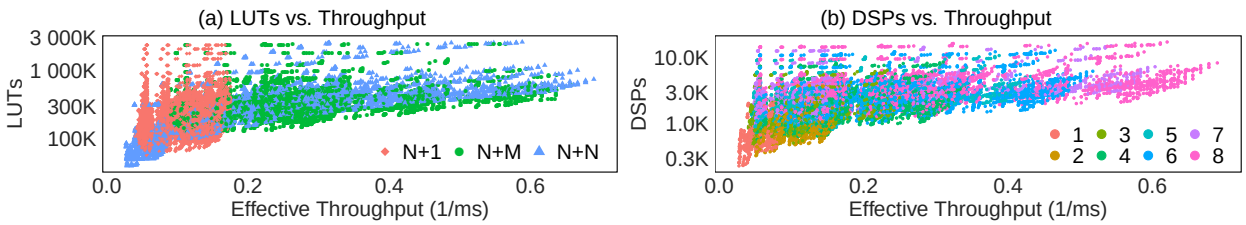


Figure 5.8: Compositional design-space exploration.

act using P2P communication. For every possible pair of accelerators, i.e., a point of Figure 5.6 and Figure 5.7, we vary the number of accelerator instances that are deployed in the SoC. We focus only on the accelerators because our goal is to determine which combinations of accelerators are Pareto-optimal. We did not consider the area of the other SoC components, which remains constant across all the combinations.

To calculate the *area*, we sum the FPGA resources required by the selected accelerator implementations and we multiply them by the number of accelerator instances. To calculate the *throughput*, we simulate the scheduling of the invocations of the accelerators in the SoC. As shown in Figure 5.3, the data-flow of one task starts from one accelerator (SVD) and continues to the next (Sinkhorn), then it repeats for a certain number of iterations K. This loop can be parallelized if there are multiple instances of each accelerator, so that we can execute in parallel T tasks, each requiring K iterations. In the first and last iterations of the loop, the accelerators interact with off-chip memory to read the input and store the output, respectively. Since these operations are performed by all possible accelerator combinations, they can be assumed to have a fixed latency. We also assume that the communication overhead during direct P2P data exchange between accelerators is negligible. This approximation can be refined further by accounting for NoC traffic.

We explore three types of SoC architectures. The first consists of one instance from one accelerator type and N instances of the other type. We call this architecture $N + 1$. The second architecture uses N instances of one type and M of the other type, assuming that N is larger than M. This architecture is called $N + M$. The third has N instances for each accelerator type and we refer to it as $N + N$. We reported values of N that vary from 1 to 8. For each configuration, we consider the total number of tasks T to complete to be equal to N, i.e., the maximum number of

accelerators from one type, and we set the number of iterations for each task to 10. We set the number of tasks to be equal to the maximum number of accelerators from one type (N) in order to be able to offload at least one task to each accelerator.

Figure 5.8 reports the configurations of the simulated SoCs in terms of throughput and resource utilization. Each design point corresponds to a combination of accelerators with a given architecture (indicated by the color and the shape of the point).

Figure 5.8(a) highlights the configurations according to the combination of accelerators, and reports the LUT utilization. The throughput for the $N + 1$ architectures is limited since one accelerator type is bounded to 1. When the throughput is low, these configurations dominate the Pareto curve. Except for the highest values of throughput, for any given value there is an $M + N$ architecture that achieves a lower resource utilization than the $N + N$ ones. In other words, the $M + N$ architectures dominate the Pareto curve. Figure 5.8(b) highlights the configurations according to their maximum number of accelerators from one type (the value of N), and reports the DSP utilization. The results are similar across all resource types.

This information is valuable when designing BCI systems with a limit on the number of accelerators, such as when the communication bandwidth is bounded and the area is constrained. Compositional DSE insights can be used to select the most suitable combination of Pareto-optimal accelerators for the specific system.

5.4 Experimental Results

Experimental Setup. We evaluate our MasterMind SoCs on the Xilinx Virtex UltraScale+ VCU118 FPGA, with a clock frequency of $78MHz$, which is the frequency set by the ESP platform. We compare several FPGA prototypes of the MasterMind SoC, in terms of performance and energy efficiency, with three general-purpose 64-bit processors: Intel i7 8700K ($3.7GHz$), ARM Cortex-A53 ($1.2GHz$), and RISC-V CVA6 [193, 194]. The Intel i7 represents a high-performance processor, the CVA6 processor represents an embedded soft core, and the ARM processor represents an intermediate option in terms of performance and power consumption.

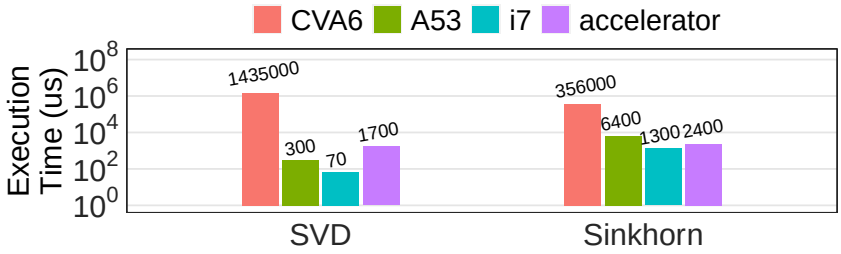


Figure 5.9: Accelerators vs. processors.

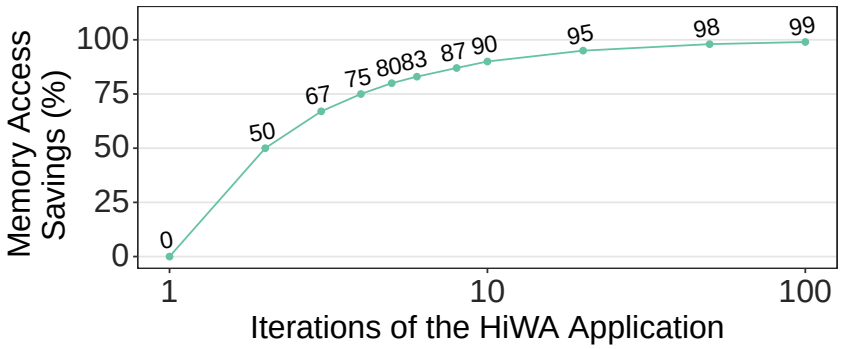


Figure 5.10: Off-chip memory access savings with P2P.

We synthesize MasterMind with Xilinx Vivado 2019.2, which reports that the CVA6 processor tile consumes $0.14W$, while the SVD and Sinkhorn accelerator tiles have power consumptions of $0.11W$ and $0.09W$, respectively. We estimate a TDP of $78.6W$ (the nominal value is $95W$) for the Intel core and of $2.8W$ for the ARM core, based on their datasheets.

Our applications running on CVA6 leverage the ESP run-time API to invoke the accelerators and allocate memory buffers of the sizes that are expected by the accelerators.

Performance of the Accelerators. We first invoke our accelerators in isolation to validate them and to compare their execution times to the ones of the corresponding C++ implementations running on the three general-purpose processors. We develop software implementations of the Sinkhorn and SVD algorithms by leveraging the Eigen C++ Linear Algebra library [203]. Figure 5.9 presents the performance of our accelerators, shown on a logarithmic scale, compared to the corresponding software implementations running on general-purpose processors. Compared to CVA6, the Sinkhorn accelerator achieves a 148× speedup, while the SVD accelerator achieves a 844× speedup. Compared to the other processors, Sinkhorn produces a 1.8× slowdown with

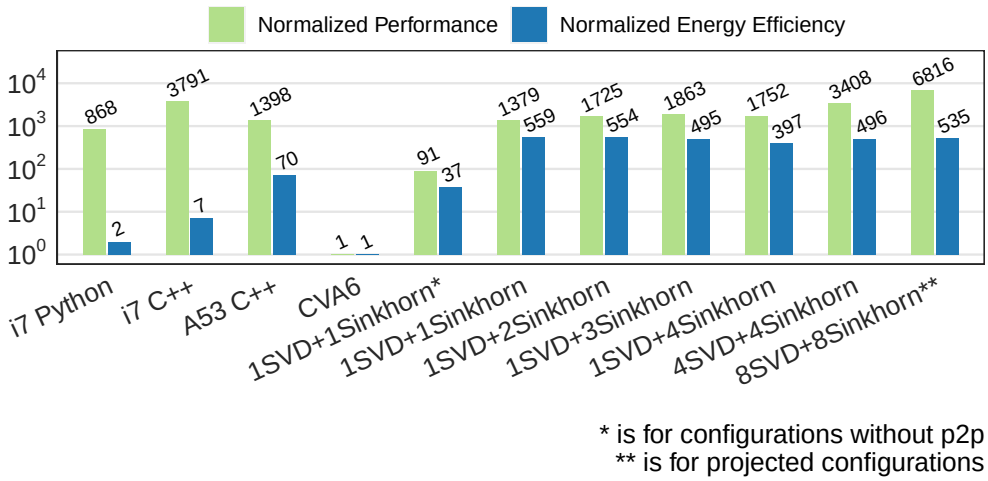


Figure 5.11: MasterMind SoCs vs. general-purpose processors.

respect to Intel i7, and a 3× speedup with respect to ARM A53. SVD produces slowdowns of 24× and 6× compared to Intel i7 and ARM A53, respectively. The slowdowns of both accelerators are to be expected due to a significantly lower clock frequency on FPGA ($78MHz$) compared to the cores ($1.2 - 3.7GHz$).

Performance and Efficiency of the SoC. Our HiWA application invokes the accelerators in a loop on the MasterMind SoC (Figure 5.3). We evaluate the execution of HiWA by scaling up the number of accelerators running in parallel and by experimenting with P2P communication. We run our tests on SoCs containing up to eight accelerators, four per type, and compare our SoCs against the three processors. For Intel i7 and ARM A53, we develop multi-threaded software applications by using the POSIX Threads (Pthreads) library to exploit all available cores. For Intel i7, we also evaluate the state-of-the-art HiWA Python application [95].

Figure 5.11 shows the performance and energy efficiency of running the HiWA applications on the three processors against multiple configurations of our MasterMind SoC. First, we evaluate two runtime configurations, one in which P2P communication is enabled and one in which it is disabled. Then, we increase the number of accelerators. The results across all four platforms (Intel i7, ARM A53, CVA6 and MasterMind SoC) are normalized with respect to the CVA6. On Intel i7, our optimized C++ implementation is three orders of magnitude faster than the CVA6

processor deployed on FPGA, while the Python implementation is two orders of magnitude faster. Even without P2P communication, the MasterMind SoC with one instance of each accelerator achieves 91× speedup and 37× on-chip energy saving over the CVA6 processor. Enabling P2P improves these measures by 1500% (1379× and 559×, respectively), and the SoC surpasses the ARM processor on both measures with 3× speedup and 24× energy efficiency.

Increasing the number of instances only for Sinkhorn helps to achieve a maximum performance gain of 1863× with three instances, and a maximum energy efficiency improvement of 554× with two instances. With four instances of Sinkhorn and one of SVD we encounter diminishing returns. Adding more instances of SVD (for a total of four) increases the performance to 3408×. The MasterMind SoC with four instances for each accelerator achieves a similar performance to the Intel i7, with only 2% of the clock frequency, and 70× better energy efficiency. The same SoC achieves an improvement of 2.4× performance and 7× energy efficiency with respect to the ARM processor. In Figure 5.11, we also project the performance of an SoC containing eight SVD and eight Sinkhorn accelerators. This SoC would be faster than the Intel i7 processor even at $78MHz$ on FPGA, while still being 76× more energy efficient.

Effectiveness of P2P Communication. Figure 5.10 shows the reduction of off-chip memory accesses from enabling the P2P communication between the accelerators. We report the reduction as a function of the number of HiWA iterations before convergence. In our experiments, we use a dataset for which the computation converges after nine iterations, resulting in 89% savings in memory accesses. Therefore, in addition to the gains in terms of on-chip energy efficiency, MasterMind can also minimize off-chip energy consumption, which is a major contributor to the overall energy dissipation of an SoC.

ASIC Projection. In our FPGA-based prototype setting, MasterMind achieves already an impressive improvement for HiWA compared to software-only executions on the Intel, ARM, and RISC-V cores. This result is obtained with a relatively low clock frequency ($78MHz$).

Although we prototyped the MasterMind SoCs on FPGA, the end goal is to build ASIC chips, which would have even better performance and energy efficiency than the FPGA prototypes. We

estimate the average power consumption of the Sinkhorn accelerator with Synopsys Design Compiler. For a $16nm$ LSTP ASIC technology node at $1V$, and with a clock frequency of $1GHz$, the Sinkhorn accelerator has an average dynamic power consumption of $14.64mW$, as opposed to the $90mW$ on FPGA. Combining this estimate with the increased clock frequency, we expect an ASIC implementation of a MasterMind SoC to achieve a further 12.8× speedup and 78.8× better energy efficiency over the results from the FPGA prototype in Figure 5.11. In other words, the 2 seconds execution time of HiWA on Intel i7 will be shortened to roughly 0.15 seconds which is lower than the 0.18 seconds bound, and as much as 13 Sinkhorn tasks working in parallel will dissipate an average power under $200mW$ (Section 4).

These additional improvements demonstrate that the design approach of MasterMind meets the threshold required for enabling real-time, energy-efficient BCI applications on BCI systems.

5.5 Related Work

Multiple algorithms were proposed for analyzing brain-data [65, 66, 64, 204]. Lee *et. al* [65] present HiWA and provide its current state-of-the-art in software [95]; its execution time with the available dataset is around 12 seconds on Intel i7. Our own software implementation takes 2 seconds on Intel i7 and 12 seconds on ARM A53. On FPGA, MasterMind achieves a performance similar to Intel i7, but with a much higher energy efficiency. As mentioned in Section 8.3, we expect even higher gains with ASIC technology. *To the best of our knowledge, MasterMind integrates the first hardware implementation of the HiWA algorithm.*

The SVD and Sinkhorn computation kernels are based on singular value decomposition and the Sinkhorn algorithm [94], respectively. Mohanty *et al.* [205] design a fixed-point Jacobi SVD algorithm on a reconfigurable system. In their work, they analyze the execution time and precision of a fixed-point type architecture called CORDIC on FPGA. They compare it to a SystemC simulation using both fixed-point and floating-point programs. They conclude that a future optimized implementation should be made using HLS on FPGA, as we did for the design of the SVD accelerator in MasterMind. *We are not aware of any hardware implementations of the Sinkhorn*

algorithm other than ours.

In recent years, researchers have focused on the development of hardware-based BCI [45, 206, 207, 208]. The HALO project [45] uses a hardware-software co-design approach to design a computation-capable, low-power and general-purpose architecture for implantable BCIs. Wahalla *et al.* [208] present CereBridge, an FPGA-based real-time processing platform for mobile BCI. These projects show a growing interest in hardware design for BCIs in order to meet the unique requirements of low power, real-time, and mobility to support real-world BCI systems.

There has been research on full BCI systems as well [29, 42, 209]. Miller *et al.* [42] review the current state of BCI that aim to record reliable signals from the human brain surface by means of electrodes (ECoG). The review states that ECoG-based BCI may consist of the following components: (1) an implantable chip for real-time recording and digitization of brain signals; (2) wireless transmission of the digitized signal; (3) real-time analysis of the brain signals and translation into computer commands; and (4) a computerized device that receives these commands and performs an action on the patient. *This vision matches the goals of our research work.*

Chapter 6

The Brain as a System-Level Resource

An essential aspect of a BCI system is its ability to execute diverse applications, including those that leverage machine learning (ML) [64, 65, 100], with a particular emphasis on adaptive control algorithms [6, 210]. However, for a real-world BCI system to be practical, it must also support high-throughput secure communication [29, 119] and ensure privacy preservation [211]. Without robust security and privacy measures for handling brain data, BCI systems would be less appealing for widespread public adoption. Supporting adaptive ML, as well as security and privacy, presents a significant challenge under the physical constraints of the BCI system. To mitigate the need for additional components that must adhere to these constraints, I propose leveraging a resource that is inherently available to the BCI system and remains unaffected by its design: the brain.

For instance, BCI applications often require access to a true high-entropy source of randomness. Randomness is essential for ML algorithms to enhance predictions and achieve higher accuracy [64, 121], as well as for security primitives to generate cryptographic keys [122, 123]. In Internet-of-Things (IoT) devices, traditional methods of harvesting entropy and generating random numbers often become bottlenecks due to resource constraints and the demand for lightweight hardware architectures [124, 127, 128, 130]. Consequently, the use of Physically Unclonable Functions (PUFs) has become ubiquitous [132, 133, 134].

Although incorporating a PUF into a device typically requires minimal additional hardware, traditional PUFs rely on differences in the physical properties of the device that arise from fabrication processes. However, these PUFs are often sensitive to environmental variations, which can compromise reliability in random number generation [134]. As an alternative, sensor-based

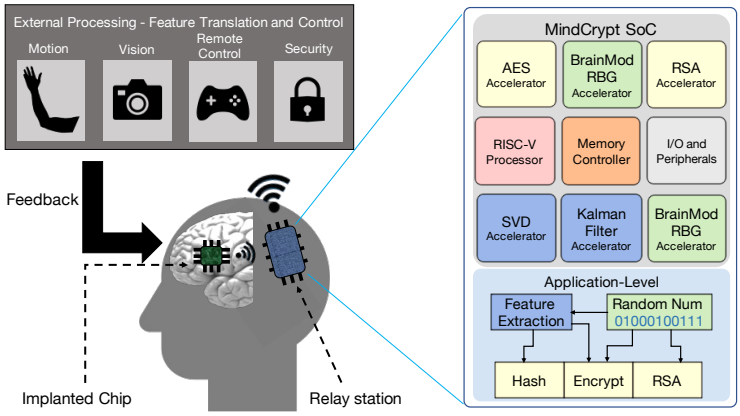


Figure 6.1: MindCrypt SoC as part of a BCI system including an IoT wearable device.

PUFs that utilize raw sensor data have been proposed [131]. Similarly, a reliable, high-throughput, sensor-based PUF leveraging brain data from sensors could provide robust support for BCI systems, which are expected to operate in constantly changing environments.

To achieve this goal, I present our work on MindCrypt [44], an SoC that provides real-time random number generation (RNG) from brain data by leveraging specialized random bit generation (RBG) hardware accelerators. Similar to MasterMind [43], we demonstrate how MindCrypt can be integrated into the wearable BCI SoC, assuming that the transmission of all neural data from the implanted BCI SoC is supported.

Figure 6.1 illustrates a potential real-world BCI system based on the MindCrypt SoC, where on-chip RBG accelerators generate random numbers for various BCI applications running on the BCI-based IoT wearable device. We develop FPGA prototypes of the MindCrypt SoC with various compositions of accelerators. Using software applications running on a 64-bit CVA6 RISC-V processor [193], we evaluate the performance of random number generation (RNG) and prime number generation, comparing them to the Linux-based `/dev/urandom` and `rand()` RNGs. We integrate cryptographic accelerators (AES, SHA2, RSA) into the SoC and analyze their communication with the RBG accelerators using point-to-point (P2P) and direct memory access (DMA). We also use Dynamic Partial Reconfiguration (DPR) to optimize the throughput of random number generation in cases where environmental conditions abruptly change the quality of neural data.

Overall, we demonstrate that BCI devices with access to real-time brain data can achieve high-quality randomness and efficient random number generation. With advancements in implanted BCI SoC designs, our brain-based RBGs could also be integrated at the implant level, enabling random bit generation closer to the neural data source before wireless transmission.

6.1 Randomness Analysis

Recent advancements in recording electrocorticography (ECoG) data directly from the brain have been reported [165]. The resulting database contains recordings from 1024 electrodes (channels) placed on the visual cortex of a non-human primate. This publicly available dataset can be processed using various data analysis tools. Leveraging this ECoG database, we can implement RBG algorithms, generate random bits, and evaluate their quality using the state-of-the-art NIST statistical tests [141].

Brain-Based RBG Throughput. The throughput of a brain-based RBG is determined by the recording capacity of the NI. In this study, we analyze data from the visual cortex of a non-human primate [165], filtered into local field potentials (LFPs) [212]. We analyze a subset of the dataset containing 78 million data points from 64 electrodes, recorded over 41 minutes [165]. This corresponds to a throughput of one value every 0.03 ms from the NI, or a data rate of 33 kHz. In contrast, the full dataset, recorded from 1024 electrodes over the same period, achieves a maximum data rate of 533 kHz. The final throughput is determined by the RBG algorithm and its implementation.

Algorithm for Random Bit Generation. As a baseline, we implemented the RBG algorithm described by Szczepanski *et al.* [14]. We refer to this algorithm as BrainBit and illustrate it in Figure 6.2. BrainBit processes a data value x by first determining whether it deviates beyond a specified threshold from the estimated mean (μ) of the dataset. The threshold is defined by the estimated standard deviation (σ) and a user-defined parameter R. Values within the range $[\mu - R\sigma, \mu + R\sigma]$ are then passed for bit generation. The computation is controlled by another parameter L, which determines the number of sub-intervals into which the range $[\mu - R\sigma, \mu + R\sigma]$ is divided. In line

1: **global variables**: σ, μ, R, L, d, h

2: **function** BRAINBIT(x)
3: Initialize: $result = None$
4: **if** $|x - \mu| \leq R\sigma$ **then**
5: $y = x - (\mu - R\sigma)$
6: $z = 2 \times R\sigma$
7: $w = \lfloor (y/z) \times L) \rfloor$
8: $result = w \mod 2$
9: **return** $result$

10: **function** BRAINMOD(x)
11: Initialize: $result = None$
12: **if** $|x - \mu| \leq R\sigma$ **then**
13: $y = x - \mu$
14: $z = y \times 2^d$
15: $w = \lfloor z \rfloor \mod 2^h$
16: $result = (\sum_h w_b) \mod 2$
17: **return** $result$

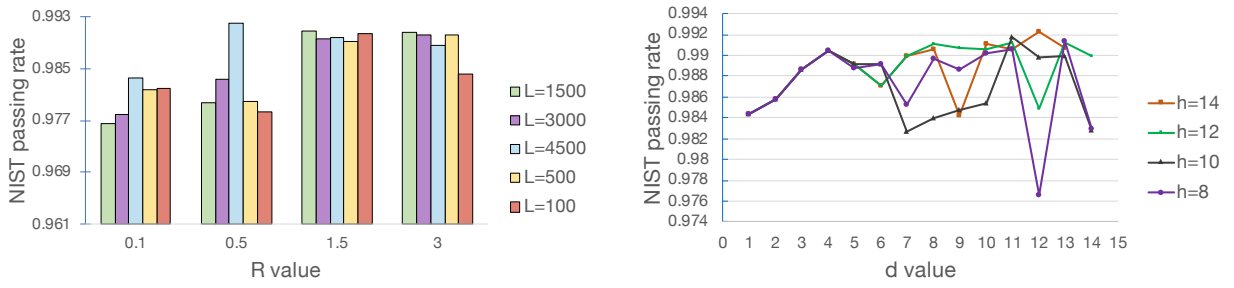
Notations:
R, L, d, h are user defined parameters
μ is the estimated average over all values
σ is the estimated standard deviation over all values
w_b are the bits of the variable w

Figure 6.2: Random bit generation algorithms.

7 of Figure 6.2, the variable w identifies the sub-interval where x is located. If w is even, the result of the bit generation is 0; if it is odd, the result is 1.

The BrainBit algorithm is based on large multiplications and divisions which might not be optimal for hardware implementations. For this reason, we propose a new algorithm, BrainMod, which emphasizes hardware-efficient techniques. BrainMod is reported in Figure 6.2. The algorithm begins by applying the same threshold control as in BrainBit. Then, it calculates the distance of x from the average μ, and multiplies it with 2^d. In line 15, the variable w holds the integer part of z after applying a modulo operation with 2^h. As a final step, the sum of the lower h bits in w is checked for oddness and the result is set accordingly. As BrainMod avoids the use of division operators and relies on powers of 2 for multiplications and modulo operations, it is expected to achieve better resource utilization in a hardware implementation.

Statistical Analysis. We ran the two RBG algorithms with different configurations to generate streams of bits from the LFP dataset. The quality of the randomness of the bit streams was determined using the NIST SP 800-22 statistical test suite [141]. This suite provides a pass or fail score for each type of test, as well as the number of tests passed for each type. In our analysis, we focus on configurations that achieve a pass score for all test types and evaluate the passing rate as the total sum of individual tests passed.



(a) BrainBit

(b) BrainMod

Figure 6.3: Statistical analysis of the bit streams generated from RBG algorithms running on electrophysiological brain data.

Figure 6.3(a) reports the results of the bit streams generated by BrainBit with respect to the values of R and L. All combinations of R, L provided high passing rates of over 96%, which match the standards of other RBGs [121]. For $R = 1.5$, all the values of L produced very similar passing rates of around 99%. For other values of R, the passing rates were less consistent and went down to a minimum of 97.6%. The number of values that passed the threshold condition of BrainBit (Figure 6.2 line 4) changed significantly when we modified R. $R = 1.5$ yielded 91% bits from the original dataset, $R = 3$ yielded 99%, $R = 0.5$ yielded 49%, and $R = 0.1$ yielded only 10%. We conducted the experiments of Section 8.3 with $R = 1.5$ and $L = 1500$, as they give a good compromise between passing rate and bit yield.

Figure 6.3(b) reports the passing rate results of the bit streams generated by BrainMod with respect to the values of d and h when $R = 1.5$. Overall, the passing rate did not go below 97%, similarly to BrainBit, and meets the standards of other RBGs [121]. In BrainMod, the scaling factor 2^d can be implemented as a shift operation of size d, and the modulo with 2^h can be implemented by selecting the lower h bits of the integer part of z. For this reason, when d is small we notice that the results for different values of h are identical. For $d > 5$, the integer part of z (line 14 in Figure 6.2) differentiates between the tested values of h and we could see more variations. When applying the algorithm, we aim at obtaining both good statistical results and good performance. For this reason, we used $\{h = 12, d = 7\}$ in our experiments.

In Section 8.1, we implement the RBG algorithms as hardware accelerators to achieve better

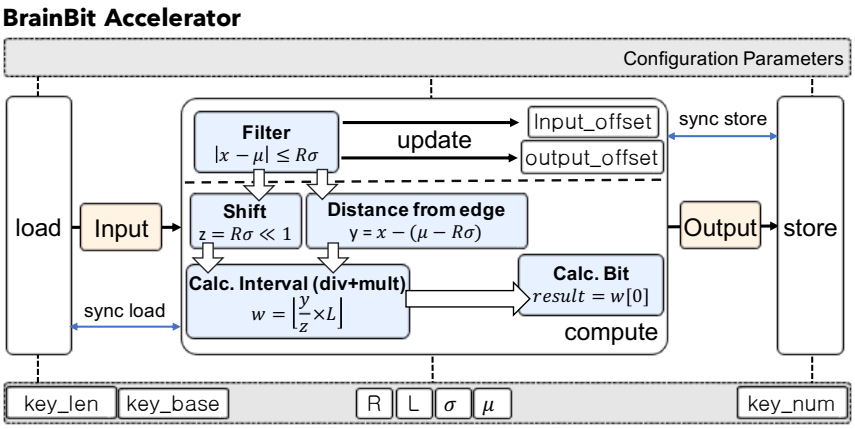


Figure 6.4: The hardware architecture of the BrainBit RBG accelerator.

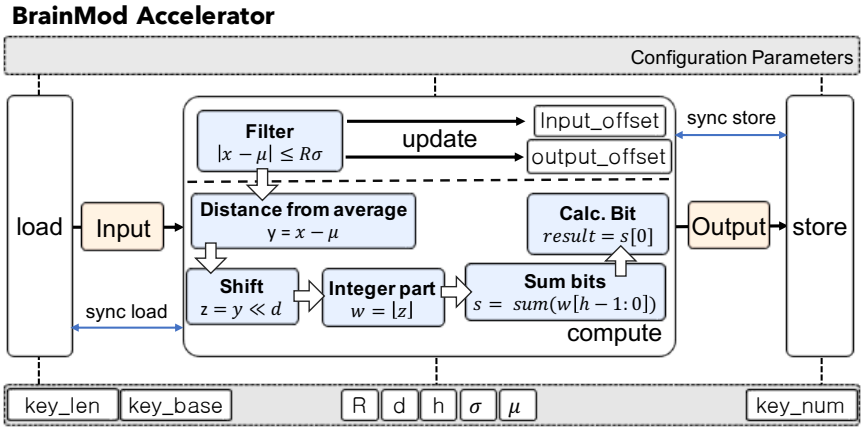


Figure 6.5: The hardware architecture of the BrainMod RBG accelerator.

performance, improved energy efficiency, and to eliminate the need for a processor in generating random numbers, which is advantageous for IoT devices [130].

6.2 Random Bit Generation Accelerators

Figure 6.4 and Figure 6.5 show the architectures of the RBG accelerators for BrainBit and BrainMod, respectively. We designed the accelerators in C/C++ and synthesized them using Vivado HLS. Our implementations extend the original algorithms (Figure 6.2) to generate random bit streams of fixed length, referred to as keys. The accelerators include private local memories (PLMs) to store inputs and outputs [201]. The PLMs are implemented as multi-bank memories that expose multiple read/write ports and their size is configurable at design time. The accelerators

are configured to operate using a 32-bit fixed-point datatype.

Communication and Computation Configurability. To handle the communication with main memory and other accelerators in the SoC, both accelerators use configuration registers that set the size and structure of the input and output data. As such, `key_len` sets the length of a key to produce, while `key_base` indicates the upper bound for the total number of keys that can be generated from the data. `key_num` holds the number of keys the accelerator will produce consecutively. The configuration registers `R`, `L`, `d`, `h`, σ, and μ configure the computation for different datasets of brain data.

Data-Flow in the Accelerators. Each accelerator consists of three main functions that operate concurrently in a pipeline [202]. The *load* function retrieves a `key_len` number of 32-bit fixed-point data values from main memory and stores them in the input PLM [201]. The *compute* function, fully pipelined to accelerate computation, implements the bit generation algorithms. In both accelerators, the computation begins by checking each value stored in the input PLM against the initial filter condition (lines 4 and 12 in Figure 6.2). If the value lies within the configured range, $[\mu - R\sigma, \mu + R\sigma]$, further computation is performed; otherwise, the value is discarded.

In BrainBit (Figure 6.4), the lower edge of the range is calculated and subtracted from the input value. The total size of the range is determined using a shift operation on $R\sigma$. As the final step, the accelerator calculates w (Section 6.1). The computation utilizes a 32-bit multiplier and a 32-bit divisor. The least significant bit of w is then stored in the output PLM as the generated random bit, forming the latest part of a new key.

BrainMod (Figure 6.5) avoids using a 32-bit multiplier and a 32-bit divisor, which are expected to increase the overall area and power on FPGA. First, the average μ is subtracted from the input value. Next, a 32-bit shift operator is applied, and the integer part of the result is taken. The lower h bits are summed, and the least significant bit of the sum is stored in the output PLM as part of a new key.

In both accelerators, after generating enough 32-bit words to assemble a key of `key_len` bit length, the *store* function transmits the key. This process continues until `key_num` keys have

been generated. To accommodate 64-bit DMA requests, the loops inside *load* and *store* feature an unrolled nested loop that reads and writes two 32-bit values in parallel. The PLM is partitioned to enable parallel read/write operations.

Input/Output Offset Control. The first step inside *compute* discards input values that do not meet the filter criteria (Fig 6.2, line 4). While *load* fetches only one `key_len` at a time, any discarded bit in *compute* requires an additional DMA read request to complete the key.

To handle this, the accelerators store offsets for both the input and output in local 32-bit registers, and use synchronization between the main functions (*load, compute, store*) [213]. If a value from the input is discarded, the input offset is incremented while the output offset remains unchanged. The computation continues until all input values in the PLM are read. At this point, a mismatch between the input and output offsets triggers a signal from *compute* to initiate an additional DMA read request by *load*. A signal is also sent from *compute* to stall the next write request by *store*. After the next DMA read, the input offset is reset to zero, while the output offset is updated based on the previously generated bits in the key. Once the output offset reaches `key_len`, the key is complete, and a signal is sent to *store* to initiate a DMA write.

6.3 MindCrypt SoC

MindCrypt provides the infrastructure for a many-accelerator SoC that supports configurable random number generation for applications that run on a wearable BCI device. The architecture of MindCrypt allows the combination of multiple accelerators with a RISC-V processor, a memory channel, and an I/O interface in a single SoC. A typical MindCrypt SoC combines one or more RBG accelerators with other accelerators that implement computation kernels and benefit from a high throughput of random numbers.

Target System. Figure 6.1 illustrates a potential BCI system powered by a MindCrypt SoC. An implanted chip uses a large array of electrodes to record the electrical activity of neurons on the outer cortex of the brain. Using a low-power, short-range wireless connection, the chip transmits the recorded brain data to an external relay station designed with a MindCrypt SoC.

Since most BCI computations are based on learning and predicting brain intentions, we include efficient accelerators for Kalman Filter and Singular Value Decomposition (SVD) to perform real-time feature extraction [64, 65]. To extract the features, applications running on the device use random numbers generated by brain-based RBGs. Further processing of the features by an external device is required for actuating and correcting disabilities via a feedback mechanism. Secure communication of the features to another device is achieved by encrypting them with an AES accelerator, which uses brain-based random keys generated by one of the RBG accelerators. The RBG accelerator supplies the AES keys to an on-chip RSA accelerator, enabling them to be sent to an external device via asymmetric encryption. The ciphertext is transmitted from the AES accelerator to the external device, where it is decrypted using the brain-based key.

SoC Design Platform. We implement MindCrypt by leveraging ESP, an open-source platform for the design of heterogeneous SoCs and their prototyping on FPGA [196]. ESP provides the template for a tile-based architecture that can be configured and specialized with the desired mix and placement of accelerator tiles, processor tiles, memory channel tiles and an I/O tile that manages various peripherals. ESP features a multi-plane network-on-chip (NoC) as the main on-chip interconnect. We selected the 64-bit RISC-V CVA6 processor [193] as the host processor.

Accelerator Communication and Invocation. The accelerator tiles follow a loosely-coupled accelerator model [148]. The RBG accelerators were designed independently from the rest of the system so they can run separately from the processor [130]. In Section 8.3, we implement MindCrypt SoCs with crypto accelerators which were designed with Catapult HLS as part of HARDROID [214]. Following their integration, the accelerators are connected directly to the system interconnect via a configurable tile socket [198], which is provided by ESP. The socket is configured through memory-mapped registers and handles address translation, cache coherence, DMA, and P2P communication. To take advantage of the P2P feature, we designed our RBG accelerators to work independently on arbitrary-size inputs with minimum software assistance. We also modified the crypto accelerators to support a similar feature.

Table 6.1: FPGA Resources and Power consumption in MindCrypt SoC

Component	LUT	FF	BRAM	DSP	Power[W]	Time[ms]
BrainBit	11441	6730	4	22	0.068	89.8
BrainMod	5929	2379	4	16	0.01	88.3
CVA6	56191	35752	36	27	0.128	N/A
AES	69075	28290	14	3	0.108	4.9
RSA	126973	57941	0	0	0.277	6874.4
SHA2	32796	20756	2	0	0.408	5.1

*Execution time is measured for 1536 bits

6.4 Experimental Results

Experimental Setup. We evaluate our MindCrypt SoCs on the Xilinx Virtex UltraScale+ VCU118 FPGA with a clock frequency of $78MHz$, which is the frequency set by the ESP platform. We develop bare-metal and Linux software applications that run on the CVA6 processor and leverage the ESP run-time API to invoke our RBG accelerators and the crypto accelerators.

Table 6.1 reports the FPGA resource usage and power consumption of the accelerators and the processor, as provided by post-implementation reports from Xilinx Vivado, along with the execution time of each accelerator in isolation on FPGA. The RBG accelerators were tasked with generating 1536 random bits, while the crypto accelerators processed an input size of 1536 bits. This input size was chosen because it corresponds to a 1024-bit key and a plaintext size of 16 32-bit words for RSA. For a fair comparison, the same input size was set for the other crypto accelerators, and the output size of the RBG accelerators is adjusted accordingly. The power consumption of BrainMod is nearly 7× lower than that of BrainBit, which was expected due to the more hardware-oriented features of BrainMod (Section 6.1).

Performance of the RBG Accelerators. The average time required to read a value from memory, send it to an RBG accelerator, and generate one random bit is 0.05 ms (Table 6.1). While the NI is actively recording, providing a data value every 0.03 ms, the throughput of random numbers depends on the number of RBG accelerator instances and their latency. Increasing the number of instances can support higher throughput, enabling the system to meet the needs of different applications.

We compare the generation of different bit lengths by each of our RBG accelerators against

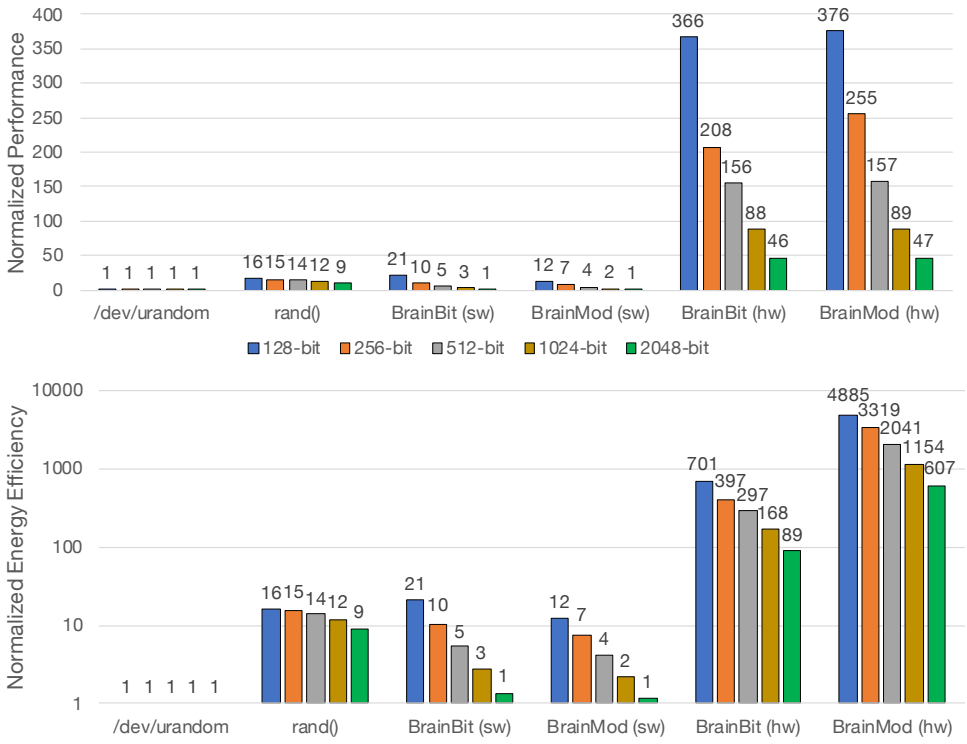


Figure 6.6: Performance and energy efficiency of the RBG accelerators.

equivalent software implementations (written in C) running on the CVA6 processor. We also evaluate the same generation using two Linux-based generators: *(1)* `/dev/urandom` [128] and *(2)* the `rand()` function in C. The interface to `/dev/urandom` allows the request of any specific number of random bytes, while `rand()` generates 32-bit words, which we concatenate to produce the required bit lengths.

Figure 6.6 presents our results, normalized to the performance of `/dev/urandom`. The RBG accelerators achieve up to 376× better performance for 128-bit generation, and at least 46× for 2048-bit. For 128-bit, BrainMod achieves a maximum of 4885× better energy efficiency and a minimum of 607× for 2048-bit. BrainMod also provides 7× better energy efficiency than BrainBit.

Random Prime Number Generation. As the RSA execution time is the largest among the accelerators (Table 6.1), the effective throughput of the system may depend on the number of RSA instances and the utilization rate of the RSA accelerator. Since RSA relies on the availability of prime numbers, we investigate the use of our RBG accelerators as prime number generators. This

Table 6.2: The Amount of Prime Numbers / 1000 Generated Numbers

bit length	BrainBit	BrainMod	/dev/urandom	rand()
256-bit	5.47	5.51	5.6	11
512-bit	2.87	2.83	2.59	6.65
1024-bit	1.16	1.46	1.08	2.74

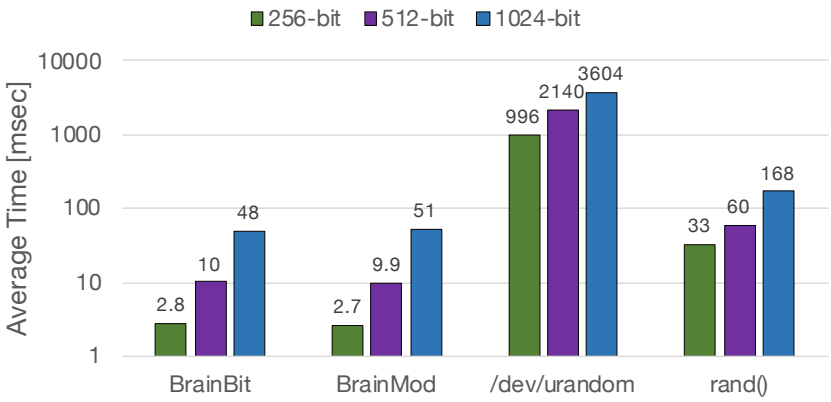


Figure 6.7: Average time to generate a prime number

approach also provides additional insight into the entropy and quality of the brain as a source of randomness, particularly when combined with BrainBit and BrainMod.

We configure the accelerators to generate 1000 random numbers of 256-bit, 512-bit, and 1024-bit lengths and run the Miller-Rabin probabilistic primality test [215] on the CVA6 processor. We also perform the same test on numbers generated using `/dev/urandom` and `rand()`. The test is conducted for 100 iterations. The average number of prime numbers generated by each RNG is summarized in Table 6.2. The number of prime numbers generated by our RBG accelerators is comparable to that of `/dev/urandom`. The amount produced by `rand()` is slightly higher; however, we observe noticeable patterns and a lower quality of randomness. Figure 6.7 presents the average time to generate a prime number from each of the RNGs on a logarithmic scale. For 256-bit, our RBG accelerators provide an average throughput gain of 368× over `/dev/urandom` and 11× over `rand()`. For 512-bit, the speedup provided by the RBG accelerators is 214× over `/dev/urandom`, and for 1024-bit, the maximum speedup is 70×. Compared to `rand()`, the throughput increases by 6× for 512-bit and 3× for 1024-bit.

Overall, the RBG accelerators provide a significant performance advantage over the other

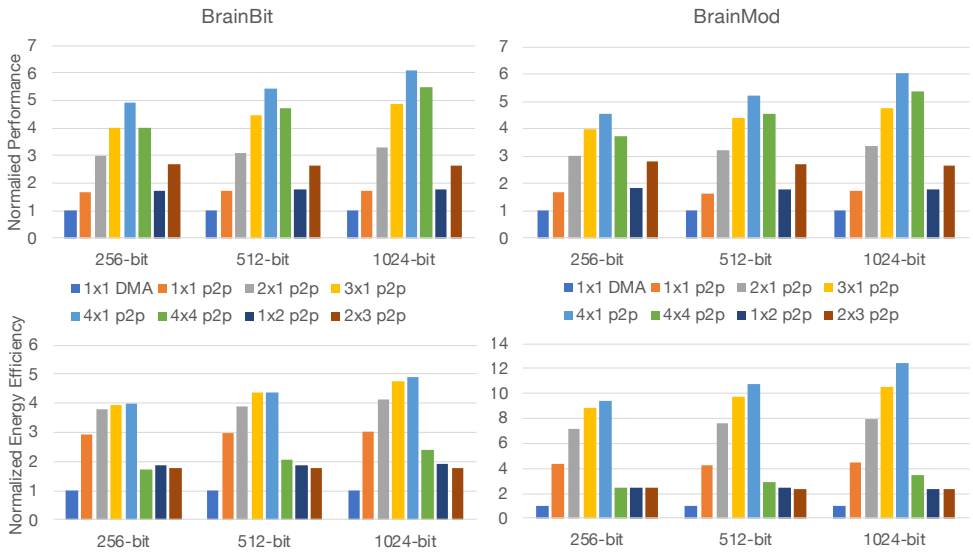


Figure 6.8: Performance and energy efficiency for different SoCs with RBG×AES accelerators communicating via P2P.

RNGs. The results demonstrate the potential of MindCrypt as a viable prime number generation engine for asymmetric cryptography algorithms like RSA, which rely on the availability of large prime numbers.

Performance and Efficiency with AES. To investigate the encryption throughput MindCrypt provides, we invoke both the RBG accelerators and the AES accelerator on FPGA. We evaluate the execution of their joined computation by scaling up the number of accelerators of each type. We run our tests on MindCrypt SoCs containing up to four accelerators per type, and compared the cases of communication via P2P and communication via DMA.

Figure 6.8 shows the normalized performance and energy efficiency of invoking the accelerators. Each AES receives a total of 256-bit, 512-bit, or 1,024-bit random bits, with the first 128 bits used as encryption keys and the remaining bits as 32-bit plaintext words. For each combination of accelerators, the results are normalized with respect to the performance of one RBG accelerator and one AES communicating via DMA. We report the results from unique configurations with varying numbers of instances for each accelerator type.

For the case of one accelerator of each type, enabling P2P provides a 1.7× speedup and 4.4× better energy efficiency (BrainMod). For 1024-bit and four instances of the RBG accelerators,

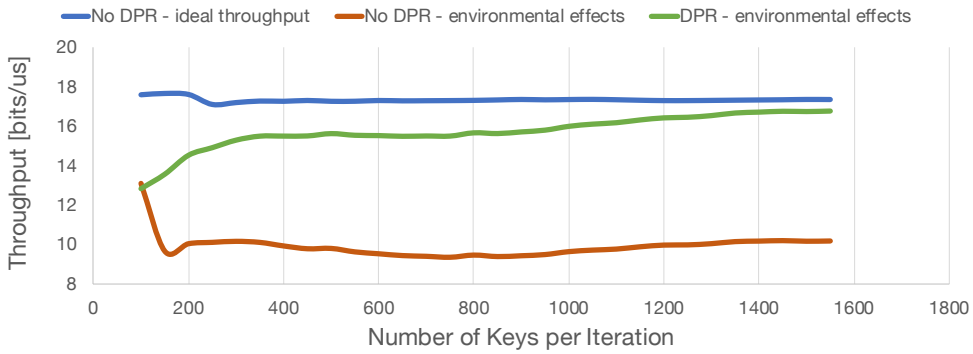


Figure 6.9: DPR throughput gains.

we achieve a maximum of 6.1× speedup and 12.4× energy efficiency. Adding more instances of AES provides moderate speedup and energy efficiency gains. This is expected due to the shorter execution time of AES compared to that of the RBG accelerators (Table 6.1).

The results show that, in addition to the security advantages of isolating the generated random bits from off-chip memory and the processor, using P2P benefits both performance and energy efficiency. We expect P2P to yield similar results when experimenting with other crypto accelerators as well.

Throughput Optimization with DPR. Environmental factors such as temperature and noise can affect the performance of the NI, resulting in some brain data being discarded by the RBG accelerators, which leads to a reduction in total throughput (Section 8.1). To address this, we aim to increase throughput by using an SoC with multiple instances of RBG accelerators reading brain data from more channels. However, BCI-based IoT devices are constrained in area and resources, limiting the ability to host many accelerators. As we target the relay-station component of a BCI system (Figure 6.1), which can be implemented on FPGA, we augment an FPGA implementation of the MindCrypt SoC with a DPR flow to multiplex BrainMod accelerators with other accelerators within the same FPGA area [216]. When a lower RBG throughput is detected, our software triggers a partial reconfiguration that temporarily replaces one of the less frequently used tiles with an additional BrainMod accelerator. As long as the power consumption of BrainMod is lower than that of the other accelerator, the total power consumption remains unaffected. The output of the

RBG accelerators can be detected using the monitors provided as part of ESP [217].

Figure 6.9 shows our evaluation of the performance of DPR in generating different numbers of keys over multiple iterations. The brain data was altered during 60% of the execution time to simulate low throughput. For fewer keys, DPR is not effective, and the RBG throughput remains the same as in the case with no DPR. For higher numbers of keys, DPR improves throughput, which nearly reaches the ideal value with no environmental effects. These results demonstrate that, in the case of an FPGA-based implementation, DPR can be highly effective in maintaining the throughput of MindCrypt, while having a negligible impact on energy efficiency [218].

6.5 Related Work

Random number generation in IoT has been emphasized in literature [121, 124, 130, 135, 136]. Kietzmann *et al.* [121] investigate the generation of random numbers using both software and hardware generators from the perspective of an operating system (OS) on IoT devices. They emphasize the importance of random number generation for emerging AI and security applications in IoT and discuss the increasing processing demands of ML and RL on IoT devices. These demands drive the adoption of hardware platforms that feature low-power RISC-V processors and hardware accelerators. In Section 8.3, by running software applications on a RISC-V processor, we demonstrate that our hardware accelerators provide superior random number generation in terms of throughput and energy efficiency when compared to Linux-based software RNGs.

Authentication and random number generation in IoT have been enabled via PUFs based on ring oscillators, radio frequency, power distribution, and more [132, 133, 134, 135]. However, to eliminate the need for modifying the fabrication of the IC and to enhance reliability, harvesting entropy from sensor-based data has been proposed as an alternative [131, 136, 137]. Hillerström *et al.* [131] propose to build a sensor-based PUF by combining raw data from multiple sensors and applying randomness extraction algorithms to transform it into uniformly distributed sequences. Wallace *et al.* [136] build the SensoRNG framework which collects data from on-board sensors to produce random numbers via a mixing algorithm. In Section 6.1, we apply our RBG algorithms

to brain data recorded from multiple sensors (electrodes) to extract uniformly random bits and validate them using the NIST test suite [141], the standard to verify RBGs [121, 131, 136].

The design of BCI systems as secure healthcare wearable devices has been discussed [29, 119, 120, 154, 219]. Taleb *et al.*[219] explain that security and privacy are essential requirements for devices in wireless body-area networks (WBANs). Lin *et al.*[13] demonstrate how online brain data can be used for authentication. Szczepanski *et al.* [14] present the only existing algorithm in the literature that converts neurophysiological brain data into pseudo-random bits and test it on brain data from a cat. They suggest that, in the future, random bits could be used to generate cryptographic keys. In Section 6.1, we implement BrainBit based on the description provided by Szczepanski *et al.* [14]. We then develop the hardware-efficient BrainMod algorithm.

BCI algorithms based on adaptive ML are more effective in decoding brain intent as the NI alignment and the neural activity change over time [64, 65, 100, 120]. Degenhart *et al.* [64] align neural activity from the brain of a non-human primate to body movements using ML. The implementation runs a stabilizer to support the Kalman Filter, and the model tuning involves using random values at various stages. Zhang *et al.* [100] implement an RL-based Kalman Filter, with RL known for its heavy use of random numbers due to the reliance on randomized algorithms [121]. In Section 8.1, we present the implementation of our RBG algorithms as reconfigurable hardware accelerators. In Section 8.3, we demonstrate that our accelerators offer better throughput for random number generation compared to other RNGs, making them more suitable for generating random numbers in adaptive ML applications.

MindCrypt is the first work to utilize real brain data and realize a fully heterogeneous SoC capable of providing random numbers for various applications on BCI-based IoT devices. With MindCrypt, we aim to foster the future development of BCI systems that can leverage the brain to enhance the viability of the system, bringing it one step closer to becoming a self-contained, real-world solution.

Post-BCI: Specializing BCI Computation

Chapter 7

Can Biological Neural Networks Interface with Artificial Intelligence?

The development of BCI applications that interpret brain intent and facilitate interactions with external devices has greatly benefited from artificial intelligence, particularly machine learning (ML) models [70, 97, 98, 99], control algorithms [81, 116, 220], and their combination [64, 71, 100, 221]. These techniques have enabled BCI applications to achieve sufficient accuracy for potential public use. For example, in motion decoding, where neural measurements are translated into movement, accuracies of around 10% error compared to intended movement are currently achievable, ensuring reliable control over fine motor tasks [112, 113, 114].

The computational methods developed for BCI applications often rely on techniques not specifically designed for processing large-scale neural data in a self-contained BCI system. These methods typically involve deep neural networks (DNNs), such as convolutional neural networks (CNNs) [10], recurrent neural networks (RNNs) [222], multi-layer perceptrons (MLPs) [223], and even large language models (LLMs) [224, 225]. Although the architectures of these DNNs may differ, they share a general approach of training on large-scale datasets, similar to models used in other domains. As BCI applications continue to evolve, we can expect cutting-edge DNN models to enhance both accuracy and efficiency in processing large volumes of neural data.

One major challenge is that to improve accuracy, the computational power required by DNN models has been growing exponentially, currently doubling every few months [17, 226]. While more advanced DNN models could enhance the accuracy and variety of BCI applications and enable better processing of large-scale brain data, the growing scale of these models is becoming difficult to sustain, leading to increasing computational power and energy consumption, far ex-

ceeding improvements in hardware performance [227, 228, 229]. Although techniques like quantization [230] can help reduce the physical size and computational demands of DNNs, the overall trend remains toward larger models, making these solutions more applicable in the short term.

The computational power required for BCI applications is expected to grow with the increasing scale of DNNs and the growing volume of neural data [25, 26]. Given these trends, the realization of self-contained BCI systems that rely solely on implanted SoCs and wearable SoCs to execute full real-time BCI applications will eventually become infeasible. Consequently, BCI application development must be re-evaluated to achieve higher accuracy and improve system performance.

One option is to extend the BCI system to rely on outsourced computation from high-performance machines capable of running very large DNNs. For the largest models, these machines would typically be located in data centers and accessed through cloud computing platforms [231]. This approach introduces two fundamental challenges:

1. *Real-time execution of BCI applications:* The additional time required to prepare data for wireless transmission, combined with the communication delay to distant data centers through cloud environments [232, 233], could compromise the real-time execution of BCI applications. This may create bottlenecks and dataflow complexities that could negatively affect the real-time capabilities of the system. To address this, efforts would need to focus on optimizing the outsourcing process from the wearable SoC through hardware-software co-design.

2. *Information security and data privacy:* Significant concerns exist around information security and data privacy in BCI applications [211, 234, 235]. Running real-time BCI applications on external machines would exacerbate these concerns, as brain data would be analyzed by third parties who may prioritize commercial interests over the security and privacy of individuals [236]. Solutions such as homomorphic encryption [237, 238] and zero-knowledge proofs [239] could help mitigate these risks, but these technologies are still largely under research.

A second option, avoiding these two fundamental challenges, is to keep the BCI system self-contained. This requires relying less on general DNN scaling trends and instead adopting lightweight,

specialized computation techniques for energy-efficient hardware-based processing of brain data. This option prioritizes energy efficiency and aims to keep the BCI system self-contained, potentially indefinitely.

Building on this approach, one direction is to focus on utilizing classic control algorithms, which have proven highly effective in implementing feasible BCI applications [81, 83, 116]. Revisiting and redesigning these computational techniques can improve their accuracy and specialization for BCI by leveraging insights into neuronal interactions, enabled by the increasing scale of neural interfaces and advancements in neural data research.

By following this strategy of improving the accuracy of classic control algorithms for processing neural data, enhancing the overall accuracy of BCI applications, and effectively specializing BCI computation to meet BCI system requirements, we can help avoid the computational and scaling demands of newer DNN-based ML models.

Another direction involves revisiting general DNN trends. The most popular DNN models today, LLMs, are based on transformers, which rely on large matrix multiplications, similar to MLPs—the original DNN model directly inspired by brain computation [240, 241]. This suggests that future computation trends may evolve further toward brain-inspired computation to achieve greater computational accuracy. In this case, brain-inspired neuromorphic computing may offer us a solution. By converting DNN models into spiking neural network (SNN) models that mimic the operation of biological neural networks, similar computations can be performed with potentially much lower computational power and energy consumption [242, 243].

As shown in Figure 7.1, neuromorphic computing is gaining significant traction, with projections indicating SoCs capable of supporting 10 billion neurons on-chip by 2026 [19]. However, our ability to leverage neuromorphic computing for complex tasks remains limited due to challenges in training SNNs and the lack of collaborative efforts among research groups, which has resulted in the absence of common hardware and software standards in the neuromorphic computing community [17, 19]. From a user perspective, a significant challenge lies in providing benchmarks and identifying suitable applications that can effectively benefit from neuromorphic models [19, 244].

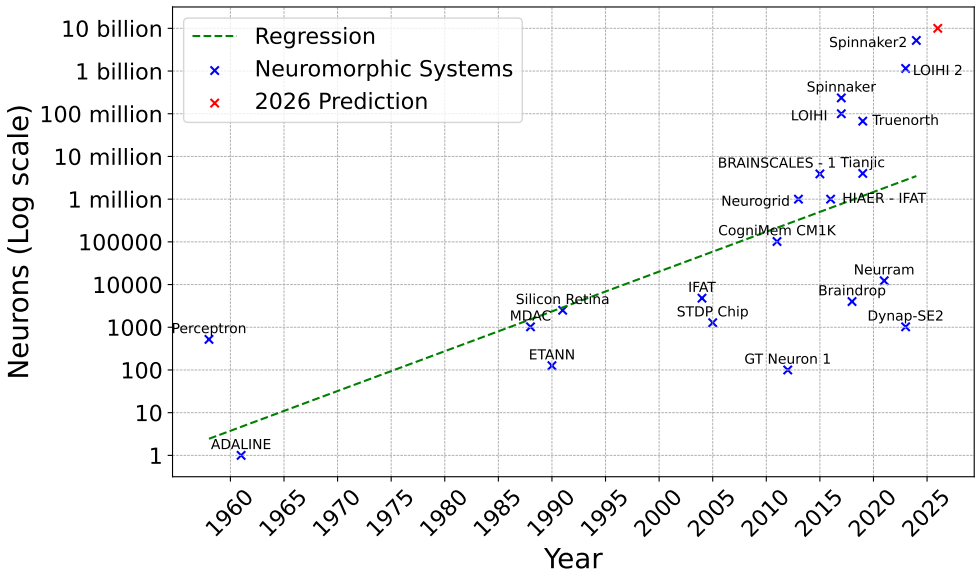


Figure 7.1: The scale of neuromorphic SoCs over time. Based on the work of Kudithipudi *et al.* [19].

For now, neuromorphic computing can be considered for smaller, intermediate tasks that help relax constraints on the computational dataflow in BCI systems, which begins with biological neural networks in the brain and ends with artificial neural networks in hardware and software. Examples of this approach already exist in systems embedding wetware [245], also known as organoids [246], where reservoirs of biological neurons trained for specific computational tasks are integrated with physical hardware to jointly solve these tasks [247]. In such cases, neuromorphic SNNs have already been integrated with biological neurons [245, 248]. Interactions with the brain could take a similar direction, as demonstrated by BCI applications utilizing SNNs [249, 250, 251].

As the first step in the computational dataflow, we could benefit from models that mimic interactions between biological neurons. This approach bridges the gap between the high abstraction of AI computation in the digital domain and the fine-grained, complex interactions of biological neurons. Combined with utilizing the brain as a resource in the system, as demonstrated previously in Chapter 6, this approach could enable the implementation of superior computational dataflows and support self-contained BCI systems. These systems would integrate precise joint computation between hardware, software, artificial neural networks, and biological neural networks in the brain, achieving unprecedented accuracy with highly efficient energy consumption.

Chapter 8

Algorithm-Hardware Co-Design for BCI

Research on BCI applications has recently focused on improving computational accuracy by utilizing new ML techniques, while giving less attention to the feasibility of real-world embedded BCI systems [7, 25, 64, 71, 83, 97, 100, 252]. While ML models show great potential for enhancing the accuracy of BCI applications, achieving an implant-based, wireless BCI system that supports mobility, real-time computation, and low power consumption remains a significant challenge.

Nonetheless, control algorithms relying on linear computation methods are still considered the standard for prediction applications in BCI [64, 71, 81, 83, 105, 221]. These algorithms may achieve lower maximum accuracy compared to deep learning; however, they are considered more robust, as they are less prone to issues such as overfitting and long training times, and they require less computational power.

In this chapter, I focus on one of the most prominent control algorithms for motion prediction in BCI—the Kalman Filter (KF). Specifically, the KF is an algorithm used in motion prediction that estimates movement based on measurements of neural data, providing predictions at each time step for the position and velocity of specific body parts. These predictions bypass the central nervous system to actuate a prosthesis or move a body part [6, 253].

However, existing hardware implementations of the KF are not tailored to BCIs [107, 108, 109, 110]. They rely on techniques that are not optimized for processing diverse, high-dimensional neural data and fail to fully utilize its potential for real-world embedded BCI applications.

For this reason, I introduce KalmMind [51, 52], a framework for the development and design-space exploration (DSE) of configurable KF hardware accelerators tailored to BCI applications.

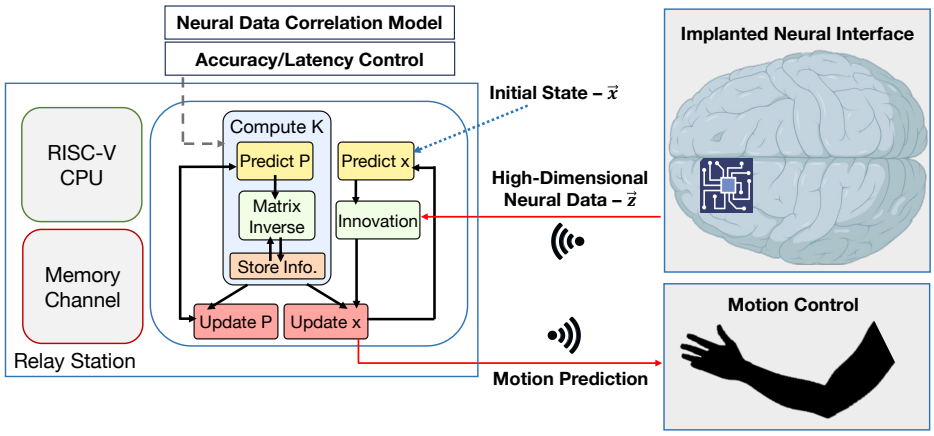


Figure 8.1: KalmMind-based hardware accelerator in a BCI system.

Figure 8.1 provides a high-level overview of a KalmMind accelerator in a BCI system.

Our approach focuses on the computation of the matrix inverse, the primary bottleneck of the KF. KalmMind incorporates a simple yet effective technique based on the iterative Newton-Raphson method [117, 254], which serves as the first approximation technique for controlling computational intensity by leveraging spatio-temporal correlations in measurements of neural activity across consecutive KF iterations [25, 26].

I further demonstrate the flexibility of KalmMind by integrating additional approximation techniques within the KF, enabling comprehensive DSE to minimize latency and maximize accuracy across neural datasets from various sources. Our modular architecture allows for the implementation of a large group of configurable KF hardware accelerators [110, 111, 116].

I evaluate and compare these accelerators by integrating them into a complete SoC that includes a RISC-V CVA6 processor [193]. The SoC is prototyped on FPGA, and experiments demonstrate that KalmMind-based KF accelerators meet the real-time requirements of BCI [76], while maintaining sufficiently low power consumption for a wearable SoC at the BAN [30].

With its high design flexibility and fine-grained control over latency and accuracy, KalmMind creates new opportunities for real-time KF predictions in embedded BCI systems.

8.1 KalmMind

Our goal is to design KF hardware accelerators that are specialized for the requirements of an embedded BCI system. Our architecture offers design flexibility and control over the potential trade-off between computational accuracy and latency.

Reorganization of the KF Algorithm. Traditionally, the KF begins by computing initial predictions of $\tilde{x}_n$ and P_n, then updates them with additional information from the KF model at each iteration (Figure 2.2). While the main bottleneck of the KF is the computation of K (line 8), K is independent from the current $\tilde{x}_n$, the current $\tilde{z}_n$ and the innovation y (line 6). K only depends on the initial prediction of P_n (line 4) and S (line 7). As a result, we reorganize the KF in order to leverage parallel processing and allow more flexibility for hardware design.

In Figure 8.1, we identify the main modules of the KF and their dependencies, then isolate the computation of K (compute K), enabling its easy modification with different computation techniques. Specifically, since K is independent from the input measurements, new measurements can be processed in parallel to the compute K module. In addition, we can switch between different matrix inversion modules. In some cases, we can even pre-compute K or S^{-1}, pre-load them into the memory of the device, and avoid their online computation [116]. Since our KF is designed for BCI, we propose a technique to efficiently compute the inverse matrix by storing and propagating information from earlier iterations of processed neural data.

Calculation of the Matrix Inverse. We define *calculation* as directly calculating the matrix inverse without using any approximations. Gauss is the standard calculation method for the matrix inverse. While generally accurate, it relies on floating-point divisions, which can introduce numerical errors. Moreover, each element in the inverse depends on the calculation of other elements, limiting the ability to decompose the calculation and exploit parallel processing. Alternative methods such as LU factorization [255], Cholesky decomposition (Cholesky) [256] and QR decomposition (QR) [257] have been proposed. These methods can reduce both the number of operations and the numerical errors. However, they still demonstrate dependencies between operations and

experience increased memory usage.

Matrix Inverse Approximation. Reducing computational latency is typically achieved by using *approximation* techniques [258, 109, 110]. Approximation requires fewer computational steps and may reduce dependencies between operations, thereby facilitating more parallel processing.

The Newton-Raphson method (Newton) is one of the oldest methods for approximating the matrix inverse [117, 259]. As it requires simple matrix multiplications, it allows for parallel processing of the computation [254]. The method is recursive, and increasing the recursion depth with more iterations improves the approximation of the inverse [260]. Specifically, an iterative approximation of a matrix inverse is given by:

$$V_{i+1} = f\_approx(V_i, A) \tag{8.1}$$

where $f\_approx(\cdot)$ computes an approximated matrix inverse V_{i+1} from the previous approximation V_i and the matrix to invert A. The Newton method defines the iterative process as:

$$V_{i+1} = V_i \cdot (2I - A \cdot V_i), \quad i = 0, 1, \ldots, m-1 \tag{8.2}$$

where the iterative process executes m iterations and V_m is the final output of the process. Since it does not involve divisions, approximation is less prone to numerical errors.

For the Newton method to converge after a minimal number of iterations, it is crucial to select a reliable *initial seed*. The initial seed V_0 must comply with the constraint:

$$\|I - A \cdot V_0\|_2 < 1 \tag{8.3}$$

which means that the seed is not too far from the optimal A^{-1}.

Combining Calculation and Approximation. Neural datasets are highly diverse, as they are recorded from different brain regions and neural interfaces, each requiring varying levels of computational accuracy and real-time performance.

We propose a new technique, which interleaves two different methods between consecutive KF iterations: one for calculation of the inverse and another for approximation of the inverse. The advantage of our technique lies in balancing a highly accurate but costly calculation with a faster, less accurate approximation.

At each iteration of the KF, one of the two methods is selected. The frequency of calculating the inverse is set by a user-configuration parameter (*calc_freq*). The inverse is calculated at every n-th iteration of the KF where $n \,\%\, calc\_freq = 0$.

In approximation, S_n is the matrix to invert at the n-th iteration of the KF (Figure 2.2). The choice of V_0 is based on one of the two policies we develop, which initialize the approximation using a matrix inverse computed at a previous KF iteration. We propose two *seed policies* as follows:

$$V_{1,n} = S_{n-1}^{-1} \cdot (2I - S_n \cdot S_{n-1}^{-1}) \tag{8.4}$$

$$V_{1,n} = S_j^{-1} \cdot (2I - S_n \cdot S_j^{-1}), \quad j = n - n \,\%\, calc\_freq \tag{8.5}$$

where $V_{1,n}$ is the result of the first internal iteration of inverse approximation in the n-th iteration of the KF. Equation (8.4) defines a straightforward policy for initializing the approximation with $V_0 = S_{n-1}^{-1}$, the matrix inverse from the previous KF iteration (n-1). Equation (8.5) defines a policy where only a calculated inverse can serve as a seed, mitigating potential inaccuracies inherent in an approximated inverse. It sets $V_0 = S_j^{-1}$ where j is the last iteration that has utilized an inverse calculation.

These policies are particularly effective for BCI, as they utilize the strong temporal and spatial correlations between neural data measurements, leveraging an inverse matrix computed for previous measurements to approximate the inverse matrix for current measurements [25, 26].

Overall, using approximation with a small number of recursive iterations is expected to yield lower accuracy but improved latency. Frequent use of calculation should enhance accuracy with higher latency. Nonetheless, calculation may introduce numerical errors, potentially degrading accuracy compared to approximation with a high count of inner iterations.

8.2 Configurable Hardware Acceleration

In this section, we describe our configurable KF hardware architecture, including the modifications described in Section 8.1.

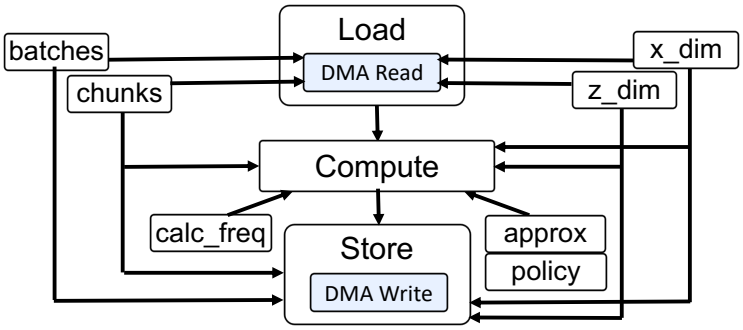


Figure 8.2: The main functions and configuration registers in the KalmMind accelerator architecture.

High-Level Architecture. The accelerator consists of three main functions: *load*, *compute* and *store* (Figure 8.2). It exposes 7 memory-mapped configuration registers that control its communication with main memory and, in particular, the matrix operations executed by the *compute* function (Figure 8.3).

The `x_dim` and `z_dim` registers configure the dimensions of the matrices (F, Q, H, R, P) and the sizes of the vectors $(\vec{x}, \vec{z})$ to be expected by the accelerator. The `chunks` and `batches` registers configure the number of KF iterations to be done in one invocation of the accelerator and provide fine-grained control of the number and size of its direct-memory access (DMA) transactions. The `approx`, `calc_freq` and `policy` registers control the dataflow of computation inside the accelerator by selecting the data path used to invert S at each iteration of the KF (Figure 8.3). Specifically, `calc_freq` sets the frequency (in terms of KF iterations) of calculating the inverse; `approx` determines how many internal iterations of approximation will be executed per KF iteration; `policy` sets the seed policy according to Equation (8.4) or Equation (8.5).

Communication and Local Memories. The *load* function loads F, Q, H, R and the initial $\vec{x}_0, P_0$ from main memory (Section 2.6) and stores them in private local memories (PLMs) inside the accelerator [201]. The PLMs are implemented as multi-bank memories that expose multiple read/write ports; their size is configurable at design time. The matrices F, Q, H, R can be reused in consecutive iterations of the KF without reloading them from main memory. For each subsequent DMA transaction, `chunks` configures the number of measurement vectors to be loaded from main memory. *load* receives (`chunks` × `z_dim`) measurements and stores them in the PLM. The *store*

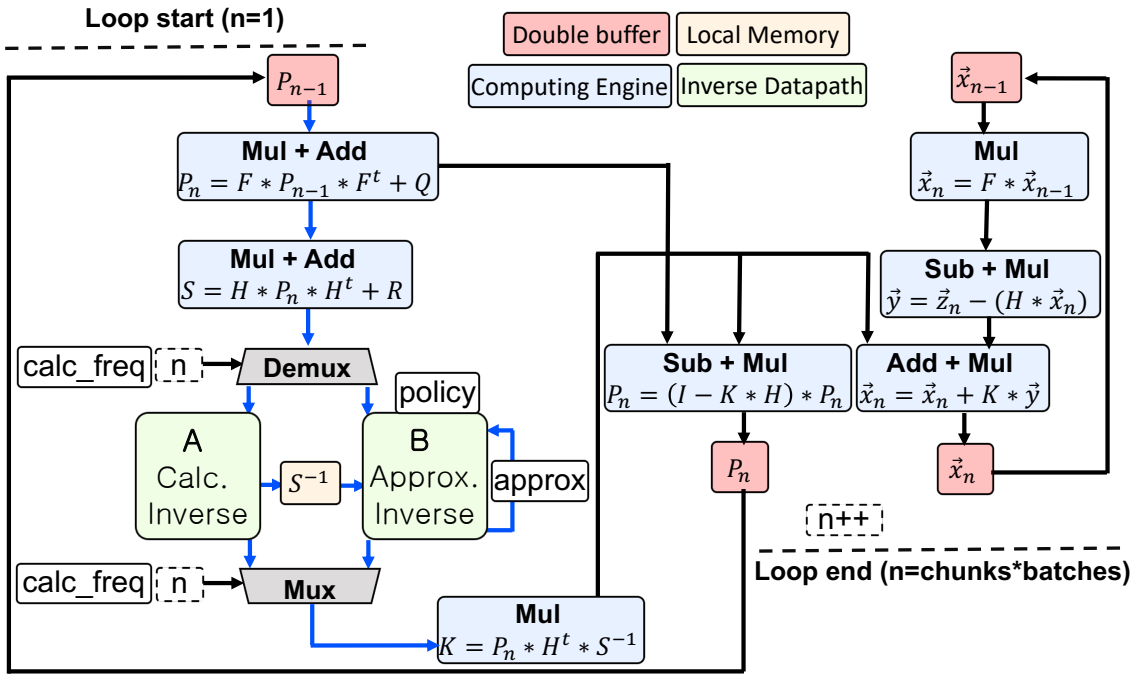


Figure 8.3: The hardware architecture of computation in the KalmMind accelerator.

function sends the computed state vectors ($\vec{x}_n$) and the state covariance matrices (P_n) to main memory. The `batches` register configures the total number of DMA transactions to be executed in one invocation of the accelerator.

KF Computation. Figure 8.3 shows how the *compute* function implements the KF with blue arrows to mark the computation path of the Kalman gain K. The state vector $\vec{x}_n$ is small, as it represents motion kinematics (Section 2.6); hence, every computation with dimension `x_dim` is relatively short. As a result, we chose to implement these computations with the objective of reusing hardware resources instead of increasing throughput. With the exception of the matrix inverse, all matrix operations in Figure 8.3 are fully pipelined. The inner-most accumulation loops are not unrolled in order to save hardware resources.

At each KF iteration, *compute* uses a double-buffer to store both the previous and new $\vec{x}_n$ and P_n (Figure 8.3). The double-buffers are swapped at the end of every iteration. The total number of iterations in one invocation is set to (`batches` × `chunks`).

Thanks to our modular architecture, we are also able to replace the computation of K at design time with an approximation technique (Taylor) or a constant K (SSKF) and easily change the

datatype between floating-point and fixed-point [111].

Implementation of the Matrix Inverse. At each KF iteration, a matrix inversion method is chosen according to the current KF iteration (`n`) and the configuration of `calc_freq`.

Path *A* in Figure 8.3 implements a calculation method for matrix inversion. We refactor the computation to minimize the number of read/write interactions with the PLM and to be able to fully pipeline the computation of the inner-most loop. In Section 8.3, we implement this path with Gauss, Cholesky and QR. We also replace it with a pre-computed constant S^{-1} (SSKF Inverse), inspired by the work of Malik *et al.* [116].

Path *B* in Figure 8.3 implements an approximation method. For most of our accelerators, we implement the Newton method. We use 8 multiply-and-accumulate (MAC) units in parallel to accelerate the computation of matrix multiplications. In its first internal iteration, path *B* uses the seed specified by `policy`. If set to 0, the policy is set to Equation (8.5) and the most recently calculated inverse from path *A* is used as the seed. If set to 1, the policy is set to Equation (8.4) and the inverse from the previous KF iteration is used instead. Path *B* performs a fixed number of internal iterations as set by `approx`.

Overall, path *B* is expected to provide a better throughput compared to path *A* when `approx` is small, while the accuracy of path *A* per iteration should be higher. When `approx` grows larger, the accuracy is improved and can surpass that of path *A*, due to numerical errors in the inverse calculation. Interleaving the two paths in different ways offers various combinations of latency and accuracy in the KF.

8.3 Experimental Evaluation

Methodology: We selected three datasets of electrocorticography neural data from distinct brain regions: the motor cortex of a non-human primate (NHP) [118], the somatosensory cortex of an NHP [99], and the hippocampus of a rat [261]. These datasets were originally tested with a KF using an inverse method that combines Gauss with LU factorization implemented with Python NumPy [83]. We refer to the NumPy implementation as *reference*. When a pure Gauss

is employed in all KF iterations, we refer to it as *baseline*. We design hardware accelerators in C/C++ with Vivado HLS 2019.2. We name the accelerators according to their embedded calculation/approximation methods or with a designated name. The accelerators use 32-bit floating-point data types unless specified otherwise.

SoC Integration: We synthesize our hardware accelerators with Vivado 2019.2 and leverage the open-source ESP platform [196]. ESP provides a tiled SoC architecture connected by a network-on-chip (NoC) with a library of heterogeneous components to facilitate "mix-and-match" SoC design. By following the accelerator integration flow [262], we utilize ESP for its implementations of DMA, memory-mapped registers, and interrupts that allow for seamless usage of an accelerator within a complete heterogeneous SoC architecture. We further leverage ESP to generate FPGA prototypes of complete SoCs combining accelerators along with a 64-bit CVA6 RISC-V processor [193], an I/O tile, and a memory channel tile. We test the SoCs on the Xilinx Virtex UltraScale XCVU440 FPGA board with a clock frequency of 78MHz, which is set according to the critical path of CVA6. We develop custom Linux software applications that run on the CVA6 and invoke the accelerators.

Accuracy Analysis. We analyze the predictions provided by the Gauss/Newton accelerator with different configurations. The motor dataset dimensions are $\{x{=}6, z{=}164\}$, the somatosensory dataset dimensions are $\{x{=}6, z{=}52\}$, and the hippocampus dimensions are $\{x{=}6, z{=}46\}$. We run the accelerator in simulation for 100 iterations on each of the datasets. For `approx`, we use a range of 1-6, which means that Newton can have up to 6 internal iterations. For `calc_freq`, we use a range of 0-6 which means that Gauss can run in each iteration of the KF (`calc_freq=1`), every `calc_freq` iterations (`calc_freq=2-6`), or only at the first iteration (`calc_freq=0`). We compare the output at each KF iteration to the output from the reference and calculate three accuracy metrics: *(i)* Mean Square Error (MSE), *(ii)* Mean Absolute Error (MAE), and *(iii)* the normalized maximum difference between one output and its expected value (MAX DIFF).

Figure 8.4 visualizes the accuracy results: purple denotes the highest accuracy while red denotes the lowest. For each pair of `calc_freq` and `approx`, we report the better result between the

Table 8.1: Accuracy Ranges with Three Neural Datasets

	MSE	MAE	Max Diff.
Motor	2.1×10^{-13}–1.1×10^{-6}	2×10^{-7}–1.6×10^{-4}	4.3×10^{-5}–1.91
Soma.	2.2×10^{-13}–9.9×10^{-6}	2.3×10^{-7}–5.1×10^{-4}	3.5×10^{-5}–5.3
Hippo.	3.1×10^{-11}–7.1×10^{-11}	1.2×10^{-6}–2.2×10^{-6}	8.2×10^{-5}–2.1×10^{-3}
Baseline	4.8×10^{-13}, 3×10^{-13}, 3.5×10^{-11}	2.7×10^{-7}, 2.7×10^{-7}, 1.4×10^{-6}	1.1×10^{-4}, 8.5×10^{-5}, 3.8×10^{-4}

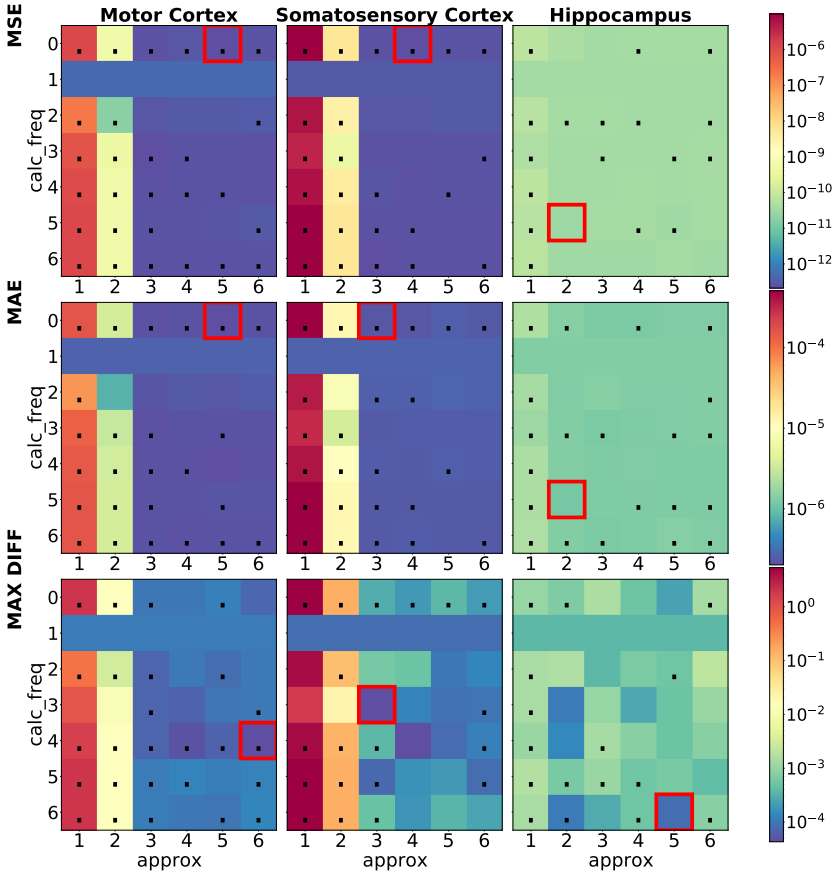


Figure 8.4: Accuracy analysis across neural datasets, configuration parameters, and different metrics.

seed policies (`policy=1` is marked with a dot). For each metric and dataset, the highest accuracy is outlined in red.

The results show that, thanks to the flexibility of the accelerator, we can adjust the computation across a wide range of accuracies. Table 8.1 summarizes the accuracy ranges to which the accelerator can be configured for each dataset and metric. In all cases, we can even find a configuration that provides better accuracy than the baseline. This occurs because Newton can reduce the numerical errors introduced by Gauss by avoiding floating-point divisions. The best result is for MAX DIFF and the hippocampus, with a 78% improvement in accuracy.

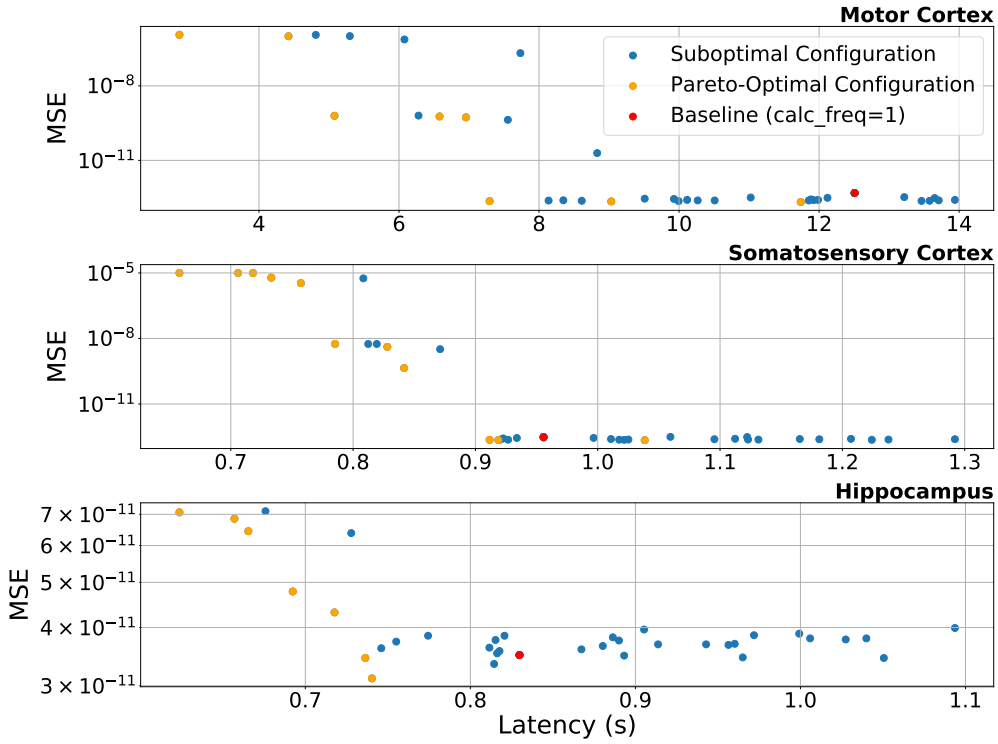


Figure 8.5: Latency vs. accuracy with the Gauss/Newton accelerator.

It is important to note that, with the same set of configurations, each neural dataset achieves its best accuracy with a different configuration. In addition, the neural datasets from the NHP correspond to different accuracy ranges compared to the dataset from the rat. This highlights the importance of matching a neural dataset with a specific KF configuration to achieve sufficient accuracy and optimize performance. A configurable KF, such as the one we propose, offers this flexibility and fosters more effective and specialized BCI applications.

Accuracy vs. Latency. We combine the previous analysis with the latency of the Gauss/Newton accelerator when running on FPGA. Figure 8.5 shows the results for each of the datasets with the MSE metric. The plots highlight Pareto-optimal points, each representing a specific tradeoff between accuracy and latency. The Pareto-optimal points result from different interleaving patterns of the two matrix inverse methods; for example, the point that provides the least latency in each plot has `approx=1` and `calc_freq=0`, while the point with the best accuracy in each plot has `approx≥2`. In each plot, we can find configurations with comparable or better accuracy than the baseline with less latency and ultimately a better performance of the KF.

Table 8.2: FPGA Resources and Performance across KF Implementations/Accelerators

Type	Method	LUT	FF	BRAM	DSP	Power [W]	Perf. [sec]	Energy [J]	Accuracy [MSE]
Software	Intel i7	N/A	N/A	N/A	N/A	78.6	0.065	5.1	3.8×10^{-12}
	CVA6	43996	29922	36	27	0.177	1927	341	1.3×10^{-12}
Hardware: Calc./Approx. Datapath	Gauss/Newton	22119	18725	228	252	0.185	2.8–8.9	0.52–1.64	1.03×10^{-12}–1.1×10^{-6}
	Cholesky/Newton	22429	20126	360	268	0.207	2.8–11.5	0.58–2.38	1.05×10^{-12}–1.1×10^{-6}
	QR/Newton	24842	21259	385	258	0.236	3.04–9.6	0.72–2.27	1.02×10^{-12}–1.1×10^{-6}
	Gauss/Newton FX32	19646	12131	195.5	217	0.146	4.25	0.354	5.9×10^{-2}–0.46
	Gauss/Newton FX64	34831	26109	369	534	0.18	2.44–11.3	0.44–2.04	1.9×10^{-5}–0.24
Hardware: One-way Datapath	LITE	15591	13405	146.5	193	0.114	2.688	0.306	1.14×10^{-6}
	LITE FX64	14782	8075	267	347	0.11	2.268	0.249	1.14×10^{-6}
	SSKF/Newton	18798	16961	204.5	240	0.158	0.53–11.6	0.08–1.82	9.9×10^{-13}–6.3×10^{-5}
	SSKF	8403	6752	19.5	102	0.051	0.03	0.0015	7.63×10^{-3}
	Taylor	15006	13437	118	230	0.155	1.203	0.186	2.3×10^{-3}
	Gauss-Only	12386	10290	102.5	153	0.098	12.507	1.225	1.3×10^{-12}

Overall, this analysis provides valuable insights for a KF hardware design tailored to specific application constraints.

Comparative Analysis of KF Implementations. We use KalmMind to design a set of hardware accelerators. Table 8.2 summarizes their FPGA resources, power, performance range, energy range, and accuracy range, measured while running 100 KF iteration on FPGA. It also includes the baseline implementation on an Intel i7 processor with a clock frequency of 3.7GHz, and on the CVA6 with a clock frequency of 78MHz (the same as the accelerators). We experiment with the largest dataset, the motor dataset, which requires a KF to run in less than $50ms$ per iteration for real-time BCI execution [76, 118].

One group of accelerators is implemented with unique calculation/approximation datapaths, using Gauss, QR, and Cholesky calculation methods alongside Newton approximation. The other group is implemented with one-way datapaths, where each accelerator uses either a calculation or approximation method:

1. **LITE** is designed to run Newton with one internal iteration and a seed from the previous KF iteration. In the first KF iteration, LITE loads a pre-computed seed from main memory.
2. **SSKF** embeds the method by Malik *et al.* [116]. The method is BCI-specific and approximates a constant K instead of the compute K module (Figure 8.1).

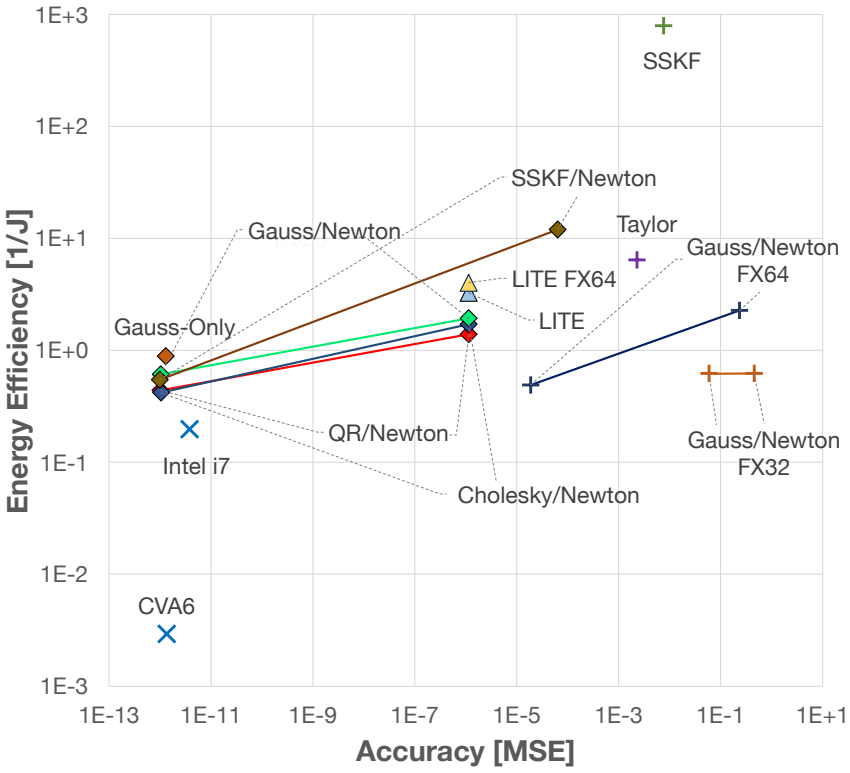


Figure 8.6: Accuracy vs. energy efficiency.

3. **SSKF/Newton** approximates a constant S_{const}^{-1} as in SSKF, however, it supports the option to run iterations of Newton to improve the accuracy of S_{const}^{-1}.

4. **Taylor** integrates the method by Liu *et al.* [110], which approximates K at every KF iteration, avoiding the matrix inverse.

5. **Gauss-Only** always calculates the inverse with our optimized implementation of Gauss, representing the state-of-the-art in matrix inverse calculation.

6. **FX64** and **FX32** are accelerators that operate with 64-bit and 32-bit fixed-point data types, respectively [111].

All accelerators, except Gauss-Only, demonstrate real-time execution by completing 100 KF iterations in under 5 seconds. All accelerators meet the low-power constraint, consuming up to $\sim 200mW$. This proves that KalmMind successfully produces accelerators suitable for real-time embedded BCI systems.

Figure 8.6 shows the tradeoff between accuracy and energy efficiency. The energy efficiency was calculated by inverting the energy result. Among the accelerators that use both calculation and approximation, Gauss/Newton has the best energy efficiency with a 10× improvement compared to Intel i7 and 655× compared to CVA6. SSKF provides the best energy efficiency with a 346× improvement over Gauss/Newton. This is expected as SSKF is the only accelerator that does not compute K. However, the accuracy of SSKF is $10^9\times$ worse than Gauss/Newton and $10^3\times$ worse than LITE. Our SSKF/Newton offers the widest range of accuracy, achieving up to 15.3× better energy efficiency compared to the standard Gauss (Gauss-Only).

Overall, KalmMind compensates for higher average power consumption, resulting from a more complex datapath, by offering substantially better energy efficiency and greater control over accuracy. KalmMind accelerators can be roughly divided into three tiers of accuracy. As accuracy decreases from the highest to the lowest tier, energy efficiency improves. Depending on the needs of the application, the neural data, and the system, the optimal KF accelerator can be chosen.

8.4 Discussion and Related Work

The KF has been used frequently for motion decoding in BCI [7, 64, 81, 83, 101, 112, 116, 220]. The KF models we use to demonstrate KalmMind are trained according to the method of Wu *et al.* [81], and provided by Glaser *et al.* [83].

Usually, BCI applications utilize a Gauss-based KF [7, 64, 83, 100, 252]. They use the KF with additional ML models and continuously update the KF model. KalmMind can optimize the KF part of these applications for real-world embedded BCIs by managing accuracy, latency, and energy efficiency.

In Section 2.6, we evaluate 4 different approximation techniques from literature [109, 110, 116, 117]. While using Newton [117] achieves the most accurate results for motion prediction from neural data, we use it in Section 8.1 to design a BCI-tailored technique that relies on propagating information between consecutive iterations of the KF. We are not aware of other implementations that use Newton within the KF. We suppose that this is because, prior to this work, no suitable

seed policy has been identified. In addition, we design optimized accelerators for fixed-point datatypes [111], Taylor [110], and SSKF [116], which is a specialize method for BCI [112, 220].

Other hardware implementations of the KF have been created to address computation in a mobile edge setting [46, 106, 108, 107, 110, 111, 263, 264, 265, 266]. They are optimized for speed and low power consumption, but most are designed for domains other than BCI, have been tested with relatively small measurement vectors, and rely on costly calculations rather than approximations. However, SCALO [46] is an accelerator-rich distributed BCI system, where the inverse matrix for the KF is computed using Gauss. With KalmMind, they could leverage approximations with sufficient accuracy, reduce data dependencies, and enhance the scalability of the BCI system.

KalmMind is the first architecture to facilitate the design of configurable KF hardware accelerators for BCI, offering flexibility and uniquely supporting fine-grained control over latency and accuracy to address the diversity of brain data.

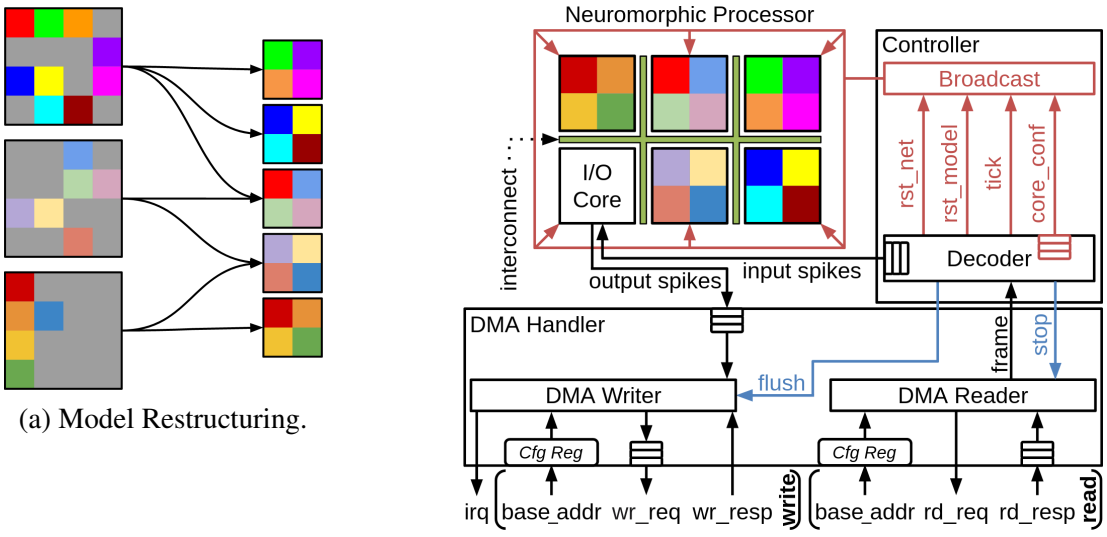
Chapter 9

Neuromorphic Computing in Heterogeneous SoCs

Inspired by the biological nature of computation in the brain, neuromorphic computing is a field of AI that promises higher energy efficiency for smart embedded devices [149, 150, 151] and autonomous machines [267, 268, 269]. In neuromorphic computing, the learning model is the Spiking Neural Network (SNN). Similar to biological neural networks, an SNN consists of neurons that communicate through channels called axons [270], with information flowing through the network via events known as spikes [271].

While specialized hardware for SNNs can be highly efficient [142, 143, 272, 273], the ultimate goal is to achieve the computation and communication efficiency of the human brain [274]. In contrast, traditional approaches to ML, which rely on dense computation, face limitations in scalability and energy efficiency [227, 228, 229, 275]. To address these challenges, DNNs can, in some cases, be converted into SNNs [276].

Several neuromorphic processors have already been integrated into SoCs in an effort to standardize their use, including Loihi 2 [145], TrueNorth [142], and Akida [277, 278]. However, these designs primarily focus on enabling the neuromorphic processor itself. Given that neuromorphic computing is unlikely to dominate the AI field or replace deep learning in the near future—due to challenges in model training and difficulties in identifying complete applications that fully benefit from a neuromorphic approach [19]—greater attention should be directed toward integrating SNNs with other components in a heterogeneous SoC. The goal is to design heterogeneous, many-accelerator SoCs capable of accelerating complex applications by utilizing SNNs alongside other computational kernels [150, 275, 279, 280].



(a) Model Restructuring.

(b) The SpikeHard Accelerator

Figure 9.1: SpikeHard accelerator design and integration.

Specifically, our goal is to support BCI applications with a computational dataflow that starts from biological neurons and ends with modern AI. As discussed in Chapter 7, we aim to leverage a similar approach to wetware [245, 246, 247], enabling high-performance BCI applications that integrate biological neurons, SNNs, and DNNs to solve complex computational tasks in real time within a self-contained BCI system.

Towards this goal, I present SpikeHard [53], a runtime-programmable neuromorphic hardware accelerator designed with a focus on high efficiency, scalability, and seamless integration into heterogeneous many-accelerator SoCs.

Figure 9.1(a) illustrates the first stage of our design methodology, which incorporates an innovative restructuring of SNNs to optimally remap an SNN model onto a desired hardware architecture. This approach enables multi-objective DSE, addressing strict performance and energy-efficiency requirements in a many-accelerator SoC. Figure 9.1(b) shows the architecture of the SpikeHard accelerator, which integrates the optimized hardware for an SNN as a neuromorphic processor, and includes an interface for handling direct-memory access (DMA) transactions along with a customized controller that enables run-time reconfigurability. These features enable the SpikeHard accelerator to be successfully integrated into a heterogeneous SoC.

We deploy many-accelerator SoCs that integrate different versions of SpikeHard on FPGA to classify the MNIST dataset [146] and approximate Vector-Matrix Multiplication (VMM). Using software applications running on a 64-bit CVA6 RISC-V processor [193], we demonstrate that SpikeHard can be invoked alongside other accelerators without compromising the overall performance of the applications. To the best of our knowledge, SpikeHard is the first work to standardize a neuromorphic accelerator integrated into a heterogeneous many-accelerator SoC.

9.1 Model Restructuring

Fitting an SNN model to a neuromorphic architecture is a non-trivial task. For this reason, we develop a model restructuring method that partitions an SNN into fine-grained groups of neurons and axons. The method reorganizes these groups efficiently in neuromorphic cores by solving an optimization problem that minimizes resource utilization. In Section 9.4, we show that model restructuring can be used productively to expand the design space of the SpikeHard accelerator.

9.1.1 Model Generation

Our SNN models and neuromorphic hardware architecture for MNIST classification and VMM are based on a generation tool provided by RANC [144]. The hardware architecture (e.g. the neuron and axon capacities of a core, and number of cores) is given as input alongside training data. Then, RANC generates the SNN model in a sequential manner, where the connections between axons and neurons are calculated at the level of each individual core, according to the architectural parameters.

The original model generation of RANC has two main weaknesses. First, the sequential generation at the level of each core imposes a dependency between the computational model and the hardware architecture. Second, after model generation, there is no process that checks for redundancy and efficient resource utilization.

As part of developing SpikeHard, we initially focus on the second weakness by optimizing SNNs that were generated by RANC with a dependency to a hardware architecture. In particular,

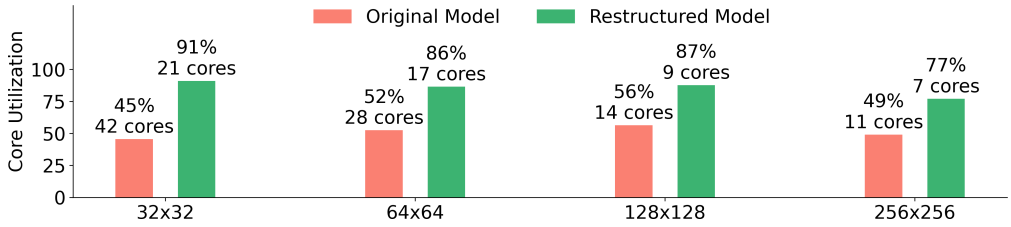


Figure 9.2: Core utilization per core capacities: Axons × Neurons.

we analyze the original placement of neurons and axons, and provided an optimized placement according to the original capacity of each core. Figure 9.2 shows that our method dramatically improves inner core utilization from an average of 50% for the original model to 85%, while also reducing the overall core count by 40%. In this way, we eliminate the redundancies at the level of a core and of the full neuromorphic processor, thereby solving the second weakness.

In addition, we succeed in transferring SNNs generated for specific hardware architectures into other architectures (Section 9.4). In doing so, we partially address the first weakness by relaxing the dependency between model generation and the hardware architecture. In the future, our method could be incorporated in the initial model generation itself, as long as the connections between neurons and axons in the SNN can be determined. Our model restructuring method is detailed in the following subsections.

9.1.2 Minimal Connected Component Analysis

Given an initial mapping of an SNN to cores, we analyze the connections between neurons and axons in the SNN so as to partition the SNN into Minimal Connected Components (MCCs). Inspired by graph theory [281], MCCs are the smallest subsets of connected neurons and axons in a core. That is, for a set S of neurons and axons in a core to be an MCC, two conditions must be satisfied:

1. All neurons and axons in S are connected only to neurons and axons in S.
2. No subset of S can be removed from S without breaking a neuron-axon connection.

During MCC analysis, we only consider intra-core connections from axons to neurons, and not inter-core connections from neurons to axons. However, inter-core connections are preserved by

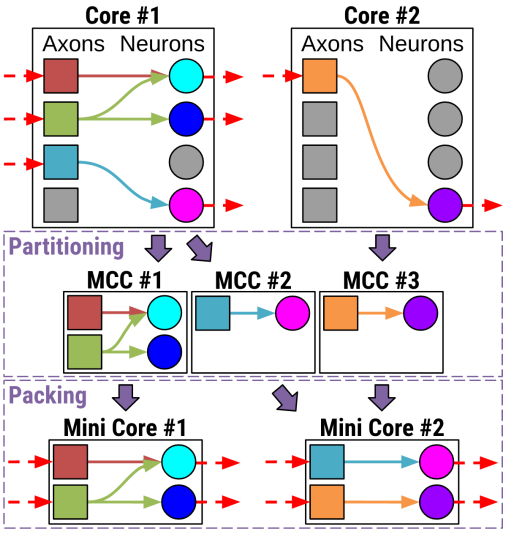


Figure 9.3: Partitioning cores into MCCs and packing them into smaller cores.

appropriately updating the output axon identifier of each neuron at the end of model restructuring.

Figure 9.3 illustrates how the original cores are partitioned into MCCs and then packed into optimized smaller cores. The packing step is formulated as an optimization problem similar to *bin packing* [282, 283], where a set of items of varying sizes must be placed into a minimum number of bins with limited capacity. In our case, the items are MCCs and the bins are the cores, with the capacity of a core being two-dimensional, determined by the number of neurons and axons available. We formulate our MCC packing problem as an Integer Linear Program (ILP), which has the following indicator variables:

$$x_{ij}, y_j \in \{0, 1\} \qquad \forall i \in \{1, ..., \#\text{MCCs}\}, j \in \{1, ..., \#\text{Cores}\}$$

where x_{ij} is 1 only when the i-th MCC is placed in the j-th core, and y_j is 1 only when the j-th core is not empty. The objective is to maximize inner core utilization by minimizing the number

of non-empty cores: min $(\sum_j y_j)$. The solution is subject to the following constraints:

$$\forall i.\ \sum_j x_{ij} = 1 \tag{9.1}$$

$$\forall j.\ \sum_i x_{ij} n_i \leq y_j N \tag{9.2}$$

$$\forall j.\ \sum_i x_{ij} a_i \leq y_j A \tag{9.3}$$

where N is the neuron capacity per core, A the axon capacity per core, n_i the number of neurons in the i-th MCC, and a_i the number of axons in the i-th MCC. Constraint (9.1) ensures that each MCC is placed in exactly one core. Constraints (9.2) and (9.3), respectively, prevent the number of neurons and axons in each core from exceeding the capacity of the core. The optimal solution to this ILP can be computed using an off-the-shelf ILP solver, notably SCIP [284]. This forms the basis for our model restructuring algorithm.

9.1.3 Core Placement within the Neuromorphic Processor

The solution to the MCC packing problem provides an optimized clustering of MCCs for a fixed number of cores with specific capacity. However, the location of cores within the neuromorphic processor should also be taken into account. As illustrated in Figure 9.1(b), the interconnect between cores is a 2D grid that allows for communication between all cores. An increase in interconnect traffic is expected between a core with a neuron that sends spikes, and a core with an axon that receives these spikes. Hence, the placement of cores inside SpikeHard can determine the level of congestion and might affect the overall performance of the accelerator. In this work, however, we focus on MCC packing itself and leave this exploration for future work. Instead, we organize the utilized cores so that the resulting shape of the grid of cores is as close as possible to a square while also not exceeding the maximum grid dimensions of the new architecture. In this way, spike traffic should already be distributed across the interconnect.

9.1.4 Model Restructuring Algorithm

Figure 9.4 shows the `restructure_model` function, which is the entry point of the optimization algorithm that restructures SNNs. The function takes as input: (1) *old_model* - the SNN model to be restructured that has already been mapped to an existing architecture; (2) *new_core_capacity* - a 2D tuple specifying the axon and neuron capacity of each core in the new architecture; and (3) *new_max_grid_dims* - a 2D tuple specifying the maximum dimensions for the grid of cores in the new architecture. The only requirement on *new_core_capacity* is that each MCC of *old_model* must be able to fit within a new core. Given *new_core_capacity*, *new_max_grid_dims* can be automatically deduced by gradually increasing its value until model restructuring succeeds.

We first `partition` the model into its constituent MCCs (Section 9.1.2). The implementation is described in Figure 9.5. At a higher level, for every core used by the model, we perform the following. For each utilized axon within the core, we skip processing the axon if it has already been added to an MCC. Otherwise, we create a new MCC containing the axon along with the neurons that receive spikes from the axon. We then find all the axons that send spikes to each of these neurons and add them to the MCC as well. This process continues recursively until no new axons nor neurons are added to the MCC, yielding a complete MCC.

We then `pack` the MCCs of the model into the minimum number of new cores by solving the MCC packing problem with the approach described in Section 9.1.2, where the new neuron capacity N and axon capacity A are given by *new_core_capacity*, and the maximum number of new cores (so as to bound j) is the product of the dimensions defined by *new_max_grid_dims*.

Given the clustering of MCCs to a minimum number of new cores, the final step is to `place` these cores into a 2D grid. The pseudocode for this procedure is shown in Figure 9.6. The resulting grid shape is as close as possible to a square while also not exceeding the maximum grid dimensions given by *new_max_grid_dims*; noting that grid position $(0, 0)$ is reserved for the specialized I/O core (Figure 9.1(b)).

```
1: function RESTRUCTUREMODEL(old_model, new_core_capacity, new_max_grid_dims)
2:   mccs ← PARTITION(old_model)
3:   new_used_cores ← PACK(mccs, new_core_capacity, new_max_grid_dims)
4:   new_model ← PLACE(new_used_cores, new_max_grid_dims)
5:   return new_model
```

Figure 9.4: Main function for model restructuring.

```
1: function PARTITION(model)
2:   mccs ← ∅
3:   for core ∈ cores utilized by model do
4:     rem_axons ← set of axons within core utilized by model
5:     while rem_axons ≠ ∅ do
6:       mcc_axons, mcc_neurons, axons_to_add_to_mcc ← ∅
7:       axons_to_add_to_mcc.add(rem_axons.get_first_element())
8:       while axons_to_add_to_mcc ≠ ∅ do
9:         axon_i ← axons_to_add_to_mcc.pop()
10:        mcc_axons.add(axon_i)
11:        rem_axons.remove(axon_i)
12:        for neuron ∈ neurons receiving spikes from axon_i do
13:          if neuron ∉ mcc_neurons then
14:            mcc_neurons.add(neuron)
15:            for axon_j ∈ axons sending spikes to neuron do
16:              if axon_j ∉ (axons_to_add_to_mcc ∪ mcc_axons) then
17:                axons_to_add_to_mcc.add(axon_j)
18:      mcc ← (mcc_axons, mcc_neurons)
19:      mccs.add(mcc)
20:  return mccs
```

Figure 9.5: Function to partition model into its constituent MCCs

9.2 Neuromorphic Accelerator

Figure 9.1(b) shows the encapsulation of the neuromorphic processor in a programmable hardware accelerator. The accelerator integrates an interface that supports both 32-bit and 64-bit DMA, and a controller that enables runtime reconfiguration and programmability of SpikeHard. To test the accelerator, we implement the design using Verilog.

```
1:  function PLACE(cores, max_dims)
2:    model ← initialize_empty_model()
3:    x_dim, y_dim ← 1
4:    while (cores.size() + 1) > (x_dim · y_dim) do
5:      if x_dim ≠ max_dims.x and y_dim = max_dims.y or y_dim ≥ x_dim then
6:        x_dim += 1
7:      else
8:        y_dim += 1
9:     assert x_dim ≤ max_dims.x and y_dim ≤ max_dims.y
10:   for i ← 1 to cores.size() do
11:     y ← ⌊i/x_dim⌋
12:     x ← i − y · x_dim
13:     model.add_core(cores.pop(), x, y)
14:   return model
```

Figure 9.6: Function to place cores within the neuromorphic processor

9.2.1 Runtime Programmability

The original neuromorphic hardware that RANC provides allows the SNN model to be configured only at design time [144]. Specifically, each core stores model parameters, such as how neurons and axons are connected to one another, in immutable local buffers initialized via Verilog initial blocks. Consequently, in the case of an FPGA implementation, loading a new model to RANC requires synthesizing a new bitstream and reconfiguring the FPGA, which is time consuming and impractical for real-time systems. In the case of an ASIC implementation, the immutable buffers would have to be initialized using an additional mechanism instead of initial blocks. Nevertheless, without programmability, the ASIC design would be unable to accommodate multiple SNN models at runtime.

In contrast, SpikeHard is a flexible hardware accelerator whose configuration can be programmed at runtime to process various SNN models. The neuromorphic processor inside SpikeHard uses the same core architecture as RANC with a few changes to support runtime programmability. To load a new model at runtime, SpikeHard overwrites the aforementioned local buffers with new model parameters. To this end, as illustrated in Figure 9.1(b), the new model parameters are broadcasted one after the other to all cores while being streamed from main memory.

Table 9.1: Commands Broadcasted to all Cores of the Neuromorphic Processor

Name	Description
`rst_net`	Reset dynamic state of loaded model
`rst_model`	Unload model from accelerator
`tick`	Signal beginning of next computation interval
`core_conf`	Configure neurons and axons of a given core

Alongside each parameter is a pointer, which specifies the destination core and buffer address. The destination core then writes the parameter to the specified address. Given a SpikeHard accelerator embedded with a grid of cores and a fixed core capacity, multiple different SNN models can be offloaded onto the accelerator by simply restructuring the models to fit within the core capacity and the grid dimensions (as described in Section 9.1.4).

Section 2.8 explains how a *tick control signal* marks the timing of computation events for a given SNN running on the neuromorphic hardware. The rate at which the tick control signal is asserted is referred to as the tick frequency. RANC sets a constant tick frequency at design time for all SNN models. However, SpikeHard supports a programmable tick frequency that can be tuned and optimized at runtime for each SNN model in our test environment (Section 9.4).

The customized controller provided by SpikeHard (Figure 9.1(b)) decodes data fetched via DMA, resulting in the execution of one of four commands listed in Table 9.1. The `core_conf` command modifies the model parameters of a specific core. The `rst_net` command initializes the loaded model to be ready for the next computation. The `rst_model` command unloads the current model from the accelerator. The `tick` command marks the beginning of the next computation interval. All the information to execute these commands is passed to the broadcast block, which in turn configures the cores.

9.2.2 DMA Handler

To be successfully integrated into a general-purpose SoC, SpikeHard must support off-chip memory access. To this end, we implemented a DMA handler that can read and write to main memory via DMA transactions (Figure 9.1(b)).

Table 9.2: Data Frames

Acronym	Frame Type	Description	Payload
RST	Reset	Reset model or network	N/A*
CD	Core Data	Model parameters for a given core	Raw core-local buffer data
IN	Input Spikes	Send spikes to specific axons	Contiguous array of spikes
OUT	Output Spikes	Output spikes for a given tick	Contiguous array of spikes
TICK	Tick	Tick a given number of times	N/A*
TERM	Terminate	End-of-File (EOF) token	N/A*

*Frame does not have a payload.

At invocation time, base memory addresses are loaded into 32-bit configuration registers. The DMA reader is then able to send read requests to main memory containing an address along with the number of bytes to read. Read responses from main memory are stored in a 4-slot buffer to enable pipelining and forwarded by the DMA reader to the controller, which decodes the data. Write requests are similarly stored in a 4-slot buffer with an address alongside the data to be stored in main memory. A write response signals that the DMA write was accepted.

The DMA handler reads input frames, which hold the information to program and run the SNN model on SpikeHard, and writes output frames that contain the computation results. Each frame has a fixed-size header of 128 bits, where the first 3 bits indicate the frame type, and the other bits are frame-type dependent. A frame may also have a payload adjacent to the header, where the payload size is stored in the header. Table 9.2 gives a summary of each frame type and its contents. The unit of data transferred on the DMA bus is equal to the bus width, which is either 32-bit or 64-bit. Thus, the end of each payload is zero-padded so that each frame is aligned to the bus width; thereby making the payloads aligned as well. No padding is used within a payload as SpikeHard appropriately parses data of arbitrary word width. Each DMA transaction can span multiple units of data. In this way, reading and writing a frame to memory takes at most two transactions, one for the header and optionally another one for the payload.

9.2.3 Data Protocol

To read a stream of input frames and produce a stream of output frames, SpikeHard uses the data protocol illustrated in Figure 9.7 (the acronyms are defined in Table 9.2).

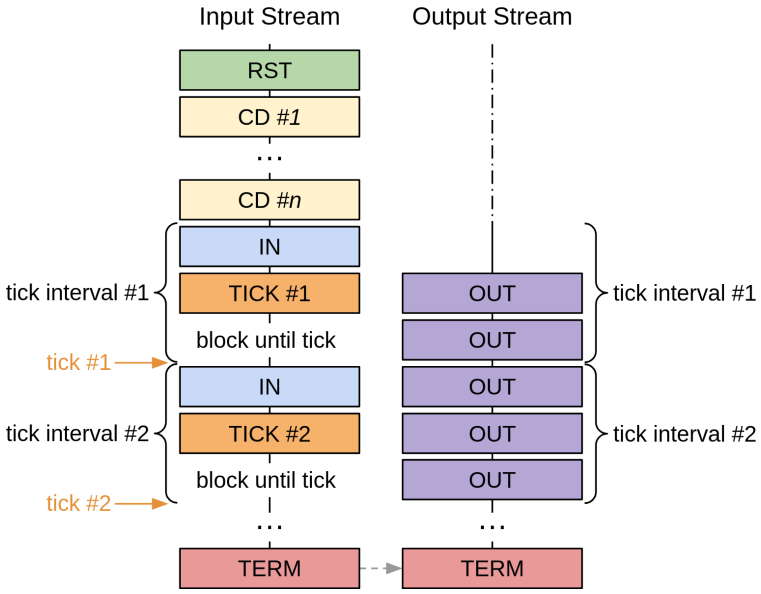


Figure 9.7: Sequence of reads from input stream and writes to output stream.

First, SpikeHard reads a reset (RST) frame. This can initiate a hard reset that will unload the model (`rst_model`) by disabling all the cores, or initiate a soft reset, which will only initialize the state of all neurons in the SNN and remove any spikes in transit (`rst_net`).

Then, core data (CD) frames are read to configure the cores (`core_conf`). Each CD frame contains a core identifier in the header, and the model data in the payload. Cores that do not receive a CD frame stay disabled.

After the model is loaded, input spikes (IN) frames are read. These spikes represent the input data (e.g. MNIST images [146] to classify). IN frames are decoded into input spikes, which get transferred to the designated I/O core (Figure 9.1(b)). The I/O core is a specialized core that receives all the input spikes and sends them to the other cores through the interconnect [**ranc**].

Following an IN frame, a tick (TICK) frame is read. A TICK frame specifies a number of ticks, and a time interval (in clock cycles). The controller will only proceed to decoding the next input frame after ticking the specified number of times, where a `tick` is generated every time a counter has reached the specified number of clock cycles. In the meantime, during each tick time interval, output spikes are tagged with the current interval identifier, and sent out as output spikes (OUT) frames, where the frame header stores the interval identifier and the payload contains the spikes.

The computation is finished once a terminate (TERM) frame is read. This causes the controller to send a `flush` signal to the DMA writer, and a `stop` signal to the DMA reader to prevent it from fetching more frames (Figure 9.1(b)). The `flush` signal tells the DMA writer to send out all remaining OUT frames and finish execution. At the end, the DMA writer also sends a TERM frame to mark the end of the output stream.

9.2.4 Accelerator Interface Scalability and Reusability

Scalable DMA interface. The SpikeHard accelerator interface supports DMA, which is necessary for loosely-coupled accelerators in heterogeneous SoCs, especially when multiple accelerators are concurrently accessing memory [148, 199]. Moreover, the accelerator interface is scalable, as it allows for the addition of new input and output frame types in the future. Additionally, there are no current restrictions on the location of specific frame types or their ordering in the input stream. Thus, the sequence of input frames discussed in Section 9.2.3 can be reordered to achieve more advanced behaviors unsupported by RANC. For instance, a CD frame can be sent after a TICK frame, which would appropriately modify the specified core. In this way, a model can be partially reconfigured during execution.

SNN-specific optimized tick frequency. In both RANC and SpikeHard, every neuron-axon pair within a core is processed sequentially once per tick. For each pair, if the axon propagates a spike, the state of the connected neuron is updated. The update time for a core is deterministic and has time complexity $O(AN)$, where A and N are the number of axons and neurons per core, respectively. Accordingly, RANC uses an independent tick generator implemented in hardware with an immutable tick frequency specified at design time based on the capacity of each core.

To improve performance and energy-efficiency, the tick frequency must be maximized within the limits of functional correctness of the neuromorphic computation. However, maximizing an immutable tick frequency with respect to the expected computation time within a core can create issues when scaling to a larger system. Specifically, the effective maximum tick frequency can vary during runtime as it depends on the amount of spike traffic and congestion not only within

the neuromorphic processor but also to and from main memory. If the tick frequency is too high, spikes may not arrive at their destination on time.

Since different SNN models can generate dissimilar spike traffic, models running on the same hardware can have different optimal tick frequencies. The optimal frequency may also vary during model execution due to changing spike traffic patterns. For instance, the input vector for VMM is provided as input spikes at the start, and no further input is provided during subsequent ticks while the model converges to a solution.

Runtime tick frequency tunability. The optimal tick frequency also depends on the larger system surrounding the neuromorphic accelerator including the application using it. Specifically, instructions being sent to the accelerator at runtime can affect the optimal tick frequency. For example, partially reconfiguring the loaded model during execution when sending CD frames after a TICK frame, can add significant latency, which would require the tick frequency to be temporarily slower. The execution time of instructions is also non-deterministic due to main memory access. This includes writing OUT frames to main memory, which must be done within a single tick period. Since SpikeHard is intended to be part of a larger system, congestion and other overheads within the system may be hard to anticipate at design time, and so the ability to dynamically modify the tick frequency is an important feature.

Altogether, tick generation must be tunable at runtime and synchronizable with the accelerator execution flow. This fine-grained control over the performance of the accelerator is made possible with TICK frames, which ensure that instructions are executed at the desired tick; noting that a TICK frame is only processed after all preceding frames have been executed. This also removes the need for a safety margin in the tick frequency, which would otherwise limit performance.

To alert when the tick frequency is too high, RANC uses error flags that are exposed as read-only top-level hardware signals. More precisely, if a tick occurs before all neuron-axon pairs are processed, or a spike is unable to arrive at its destination on time, a corresponding error flag is set. In SpikeHard, these flags are instead accessed from the Linux application via debug bits in the output-stream TERM frame. Accordingly, if a debug bit is set, this signals that the tick frequency

was too high and must therefore be decreased before re-execution. In Section 9.4.2, we leverage these debug bits to tune the tick frequency for our experiments.

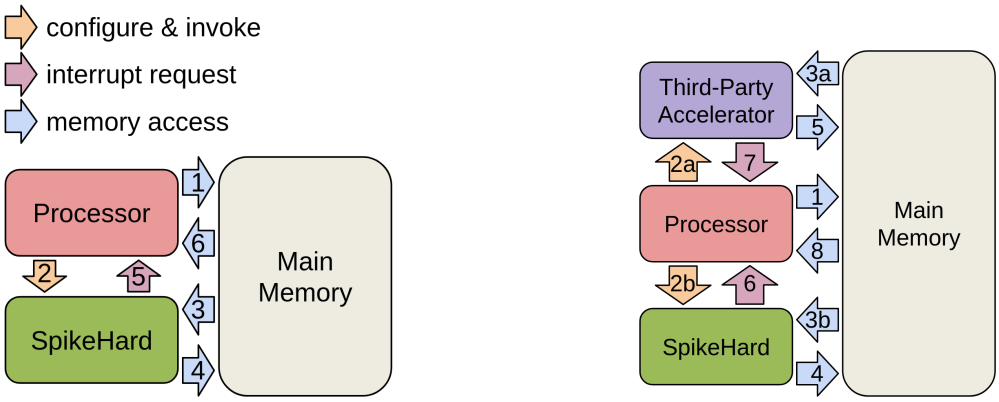
Runtime model loading. The model-loading mechanism in SpikeHard is designed for reusability and efficiency as well. As mentioned previously, hard resetting a model (`rst_model`) involves disabling all the cores, where a core is re-enabled upon receiving a corresponding CD frame. In this way, model data for empty cores is not loaded, which would otherwise be time-consuming. Moreover, this mechanism could be leveraged to reduce the power consumption of empty cores by clock gating the cores or even power gating them. This mechanism is especially beneficial when the accelerator has over-provisioned the number of cores, which happens when the accelerator must be able to run a diverse set of models with different resource requirements. For now, a disabled core is blocked from generating spikes.

Data protocol reusability. the data protocol relies on features used in various neuromorphic processor designs. In particular, the tick control signal and neuron-axon structure is notably used by Loihi [143] and TrueNorth [142]. Hence, we expect that the hardware interface and data protocol could be extended and reused to interface with a variety of neuromorphic processors with little to no modification.

9.3 SoC Integration

9.3.1 SoC Design Platform

To integrate SpikeHard in a heterogeneous SoC and prototype it on FPGA we used ESP [196]. We designed SpikeHard with a standard interface, and by following the accelerator integration flow of ESP [262], we were able to integrate SpikeHard in a tile-based SoC with a multi-plane network-on-chip (NoC) as the main on-chip interconnect, together with a desired mix of provided accelerators, processor tile, memory tile (containing a channel to main memory), and an I/O tile that manages various peripherals. ESP also provided us with a full software stack to load data into main memory and invoke SpikeHard from a Linux user-space application. As the main processor in the SoC, we selected the 64-bit CVA6 RISC-V processor [193].



(a) SpikeHard + processor

(b) SpikeHard + 2nd accelerator

Figure 9.8: Data-flows on the heterogeneous SoC.

9.3.2 SpikeHard and the Processor

Figure 9.8(a) illustrates the sequence of interaction between SpikeHard, the processor and main memory. The processor starts by loading the base addresses into the read and write configuration registers of SpikeHard. Then, all the frames are loaded by the processor into the input-stream region in main memory, including the sequence of commands to execute and the data.

Upon invoking SpikeHard, the accelerator reads the input stream from main memory, executes the sequence of instructions, completes the computation, and writes all the output frames into the output-stream region in main memory. The accelerator then sends an interrupt through the `irq` signal (Figure 9.1(b)) to notify the processor that the computation is done.

The processor then reads the output stream from main memory until it reaches a TERM frame. At that point, the processor holds all the computation results, and can decode the output spikes.

9.3.3 SpikeHard and Other Accelerators

To run complex applications in constrained environments, namely on the edge, we need devices based on heterogeneous many-accelerator SoCs as we require a variety of specialized hardware to efficiently run many different tasks [89, 285].

Neuromorphic hardware has been developed for ASIC-based edge devices [89]. We are also aware of applications that partially use neuromorphic computations [279, 280, 150]. In particular,

Arsalan *et al.* [279] present an algorithm where the input data is pre-processed using a Fast Fourier Transform (FFT) before being fed into a neuromorphic model. Similarly, Chandarana *et al.* [280] describe a method that applies 2D-convolutions (CONV2D) to an input image so as to encode it into a sequence of spikes, which is then processed by a neuromorphic model. However, no work had investigated neuromorphic hardware as a specialized hardware accelerator in a largely non-neuromorphic many-accelerator SoC. For this reason, we integrate SpikeHard in an SoC containing an FFT accelerator, as well as a CONV2D accelerator (Section 9.4.8).

Figure 9.8(b) illustrates a scenario where the application running on the processor takes advantage of the combined computation of SpikeHard and a third-party accelerator. Note that in Figure 9.8(b), arrows labeled with the same number followed by "a" or "b" denote actions that are performed concurrently with no defined order. In Section 9.4.8, we evaluate such a scenario.

9.4 Evaluation

9.4.1 Experimental Setup

We evaluate various implementations of SpikeHard by integrating them into different SoCs, each containing a 64-bit RISC-V CVA6 processor [193], an FFT accelerator, and a CONV2D accelerator. We deploy SoC prototypes on the Xilinx Virtex UltraScale+ VCU128 FPGA with a clock frequency of 75MHz, which is set by the ESP platform. We synthesize these prototypes with Vivado, which reports the power dissipation and usage of FPGA resources: look-up tables (LUTs), flip-flops (FFs), and block RAMs (BRAMs). We do not discuss DSP usage as it remained constant across all implementations. We estimate power dissipation based on the dynamic power reported by Vivado for each of the synthesized designs. We use this power estimate together with the accelerator execution time on FPGA to calculate the energy used by each design. To test SpikeHard on FPGA, we implement Linux applications in C that run on the CVA6 processor and invoke the accelerator via a device driver.

9.4.2 *Tick Frequency Tuning*

As described in Section 9.2.4, maximizing the tick frequency improves performance and energy efficiency. Accordingly, for each model and hardware architecture, we optimize the tick frequency using the following procedure.

Given a tick period and set of input spikes, we run the model on FPGA, or alternatively in a Verilog simulation consisting of the accelerator in isolation and an emulated DMA bus. The period is valid if no debug bits in the output-stream TERM frame (Section 9.2.4) are set. We first try with a small period, then increase it using a gradually larger step size until we obtain a valid period. The largest invalid period is the lower bound on the optimal tick period, and the smallest valid period is the upper bound. We then perform binary search within these bounds to find the optimal tick period. To account for real-world uncertainty, notably due to non-deterministic spike traffic, we add 10% slack to the optimal tick period.

In all the experiments, the most time-consuming operation during each tick period was the neuron state updates. The duration of this operation depends only on the core capacity (Section 9.2.4). Accordingly, we found that for a given core capacity, all evaluated models have the same optimal tick period. This means that communication, such as reading an IN frame from main memory and sending the spikes in its payload to their corresponding cores, is not a performance bottleneck during computation. Thus, the only performance overhead incurred by the runtime configurability of SpikeHard is the model-loading overhead, which can be amortized away (Section 9.4.3). However, communication may become a performance bottleneck for larger models with smaller cores. This is because larger models produce more spike traffic, and smaller cores can run at higher tick frequencies. Furthermore, applications that interact with a mixture of hardware accelerators can create fluctuating congestion on the SoC, which is why dynamic tick frequency tuning is essential.

9.4.3 *SNN Models*

The models loaded on SpikeHard classify images from the MNIST dataset [146] and perform signed VMM using the algorithm described in [147]. For MNIST, we classify $batch\_size = 200$

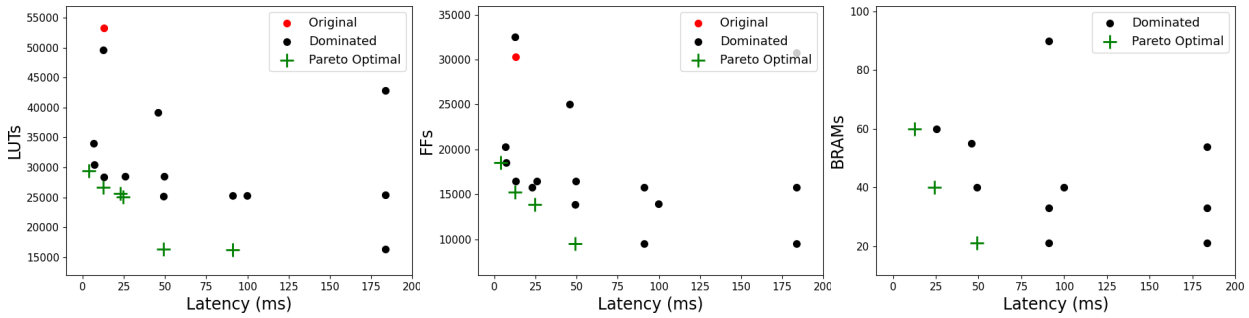
images per run in a pipelined fashion, where a new image is provided as input at every tick. In contrast, VMM uses feedback connections to iteratively converge to an approximate solution. Consequently, only a single vector-matrix product can be computed per run, that is, $batch\_size = 1$. The performance of a model is its throughput, which is defined as the $batch\_size$ divided by the average latency of a run. This includes the model-loading overhead at the start of every run. In practice, if the same model is used across several accelerator invocations, it only needs to be loaded once so that the model-loading overhead becomes negligible. Furthermore, if a variety of models are being frequently used, multiple instances of SpikeHard can be integrated within a single SoC, where each model is loaded to a distinct instance and reused across invocations.

We test models with various combinations of neurons (N) and axons (A) per core, denoted as $A \times N$. We start with two models generated by RANC: 64×64 VMM (VMM-O) and 256×256 MNIST (MNIST-O). We optimize the tick frequency for each model using the method described in Section 9.4.2, resulting in performance and energy-efficiency gains of 1.9× for MNIST-O and 27× for VMM-O over the original RANC models, while preserving functional correctness. *Unless stated otherwise, the results reported in this section are relative to the corresponding RANC-generated model with optimized tick frequency.*

In our evaluation, we use the same bit precision as in TrueNorth, namely signed 9-bit values for weights and neuron potentials, which is also the default used in RANC. In practice, to reduce resource utilization, bit precision would be optimized to the minimum required by the models.

9.4.4 MNIST Performance and Energy-Efficiency

By applying model restructuring on MNIST-O without changing $A \times N$, we decrease the number of cores used from 11 down to 5. Accordingly, FPGA resource usage is reduced by 2×, yielding the same improvement in energy efficiency. Although core capacity is unchanged, performance improves by 0.2% thanks to a 2.4× decrease in model-loading overhead as a result of utilizing fewer cores. However, computation time excluding overheads remain the same since the tick frequency is unaffected.



(a) (b) (c) excludes #BRAMs = 0

Figure 9.9: Design-space exploration of hardware implementations originating from VMM-O.

MNIST-O cannot be restructured to a smaller power-of-two core capacity ($A, N < 256$) as MNIST-O has an MCC with 256 axons and 250 neurons. Restructuring MNIST-O into architectures satisfying $A, N > 256$ yields a negative impact on resource utilization without improving performance. Ultimately, MNIST-O restructured to the same core capacity (256×256) gave the best performance and energy efficiency.

9.4.5 Design-Space Exploration for a Fixed-Size VMM

We restructure VMM-O to all core capacities satisfying $A, N \in \{32, 64, 128, 256, 512\}$. We are able to reduce the number of axons and neurons per core down to 32 since none of the MCCs have more than 32 axons or neurons. The DSE of VMM-O is summarized in Figure 9.9 with Pareto-optimal points for each FPGA resource: LUTs (Figure 9.9(a)), FFs (Figure 9.9(b)), BRAMs (Figure 9.9(c)). These points are also listed in Table 9.3. Several implementations do not use BRAMs, which are replaced by FFs. These implementations are not considered when determining the Pareto-optimal points for BRAM utilization.

Pareto-optimal implementations with smaller core capacity utilize more FFs and LUTs. This is due to the fact that in these implementations, FFs are preferred over BRAMs, and more cores are needed to fit the model. In some cases, the use of BRAMs instead of FFs and LUTs improved power consumption. In fact, implementation F and G, which replace FFs and LUTs with BRAMs, consume between 1.3× and 1.9× less power than Pareto-optimal implementations that do not use BRAMs. Furthermore, implementation G, which has the largest core capacity among the Pareto-

Table 9.3: VMM-O: Original vs. Pareto-optimal Implementations

Name	$A \times N$	#Cores	Pareto Optimal			Resource Usage			Latency (ms)	Tick Freq. (kHz)	Power (mW)
			LUT	FF	BRAM	LUT	FF	BRAM			
Original	64×64	17			N/A*	53314	30296	0	12.69	15.49	211
A	32×32	15	✓	✓	N/A*	29483	18540	0	3.82	60.53	112
B	64×64	8	✓	✓	N/A*	26669	15286	0	12.61	15.49	92
C	32×128	15			✓	28406	16456	60	12.75	15.49	135
D	128×64	5	✓		N/A*	25502	15807	0	23.35	8.05	80
E	64×128	8	✓	✓	✓	25240	13895	40	23.98	8.05	113
F	128×128	3	✓	✓	✓	16334	9539	21	48.93	3.82	61
G	128×256	3	✓			16320	9544	21	91.12	2.04	60

*Implementation does not use any BRAMs.

optimal implementations, has the lowest power consumption. However, cores of larger capacity require a lower tick frequency, resulting in higher latency and thus worse performance.

When stringent power constraints take precedence over overall latency, or when there is a limited amount of LUTs or FFs available, using larger-capacity cores should be considered.

Figure 9.10 shows the performance and energy efficiency of the Pareto-optimal implementations of VMM-O, normalized relative to those of the original implementation. Implementation *A* (smallest core capacity) gives the best results, with 3.3× better performance and 6.3× higher energy efficiency over VMM-O. In total, the improvements in performance and energy efficiency over the original RANC model without optimized tick frequency are 89× and 170×, respectively. Moreover, implementation *B* has the same core capacity as VMM-O restructured into 2.1× fewer cores. As a result, energy efficiency improved by 2.3× mostly thanks to a less power-consuming design. Additionally, the latency of implementation *B* is reduced with respect to the original design as the model-loading overhead was halved. However, the reduction is only by 0.6% as model loading originally constituted only 1.6% of the overall latency, hence, it had no significant impact on energy efficiency.

Altogether, model restructuring to the same core capacity significantly improves energy efficiency, and restructuring into smaller core capacities yields higher gains in energy efficiency and performance.

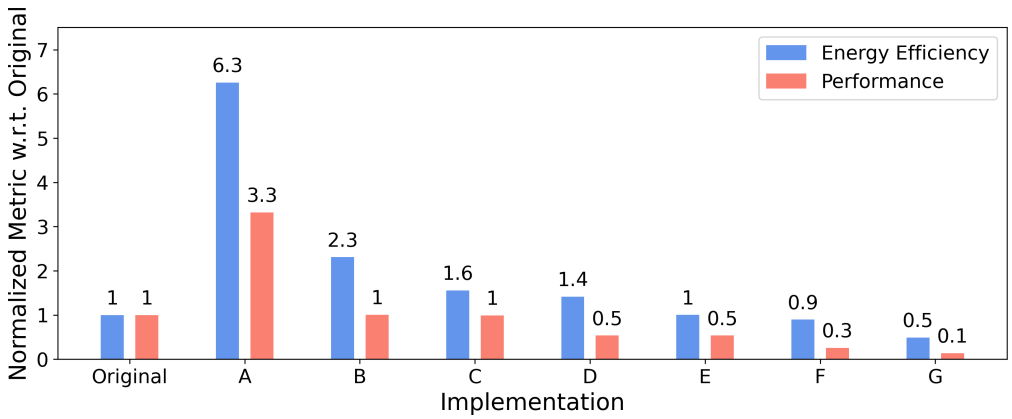


Figure 9.10: Performance and energy efficiency analysis of Pareto-optimal implementations of VMM-O.

9.4.6 Design-Space Exploration for Varying-Size VMM

In SNN models that compute a VMM, the connections between neurons and axons as well as the model size are highly dependent on the matrix shape. To determine whether model restructuring is effective regardless of the model structure, we test our optimizer on models with different matrix shapes. As VMM-O is generated for a matrix with 6 rows and 6 columns, we perform DSE on VMM models trained with larger matrices. We start by generating models for two additional matrix shapes: 15 rows by 15 columns (VMM-15), and 16 rows by 64 columns (VMM-64).

The DSE for VMM-15 is summarized in Figure 9.11. It includes Pareto-optimal points for each FPGA resource in the top three plots, along with an analysis of performance and energy efficiency in the bottom plot. VMM-15 is generated with a core capacity of 128×64. We restructure the model to a variety of power-of-two core capacities, with the smallest one being 64×32 in order to fit the biggest MCC, which has 60 axons $\times$ 28 neurons. We gather all the Pareto-optimal implementations from Figure 9.11(a), Figure 9.11(b) and Figure 9.11(c), and compare their performance and energy efficiency normalized to that of the original VMM-15 implementation. Figure 9.11(d) shows this comparison, where the x-axis denotes the core capacity of the implementation. The implementation with the smallest core capacity (64×32) achieves the best improvements in performance and energy efficiency of 3.2$\times$ and 4.9$\times$, respectively.

The DSE for VMM-64 is summarized in Figure 9.12. It includes Pareto-optimal points for

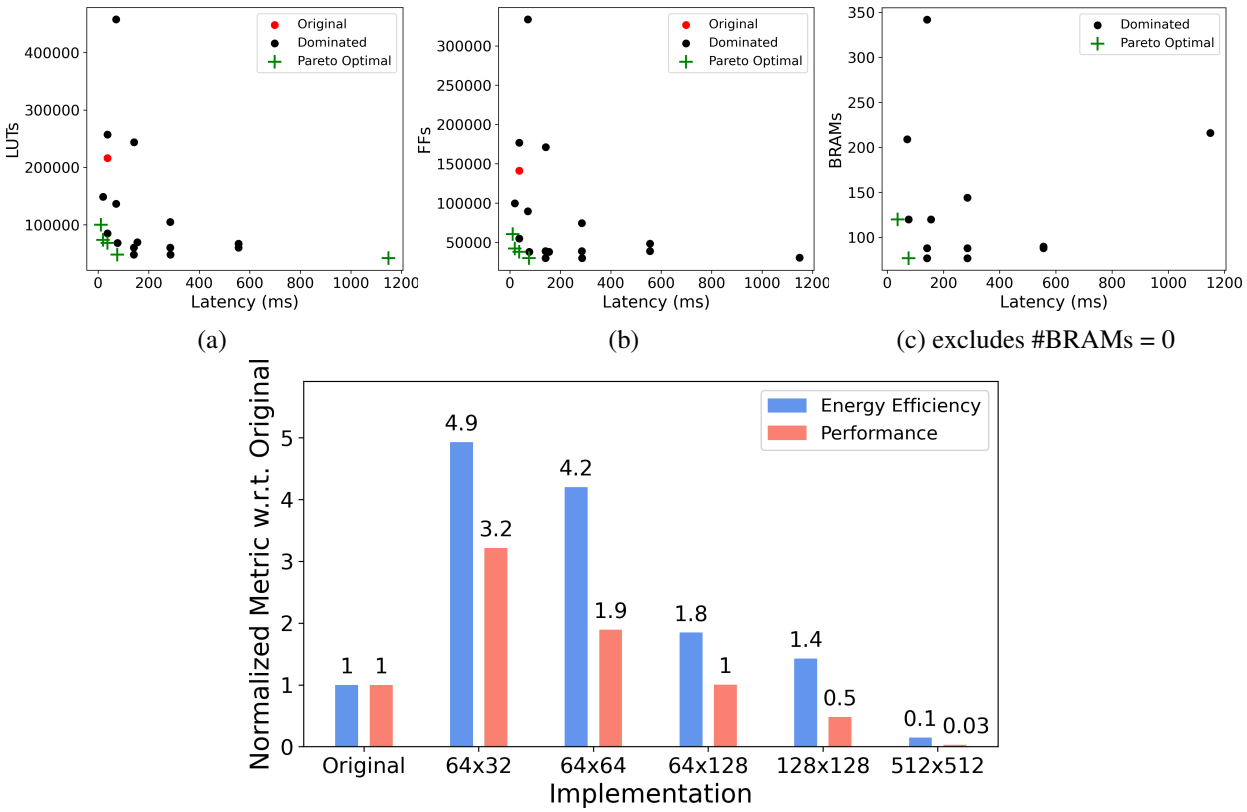
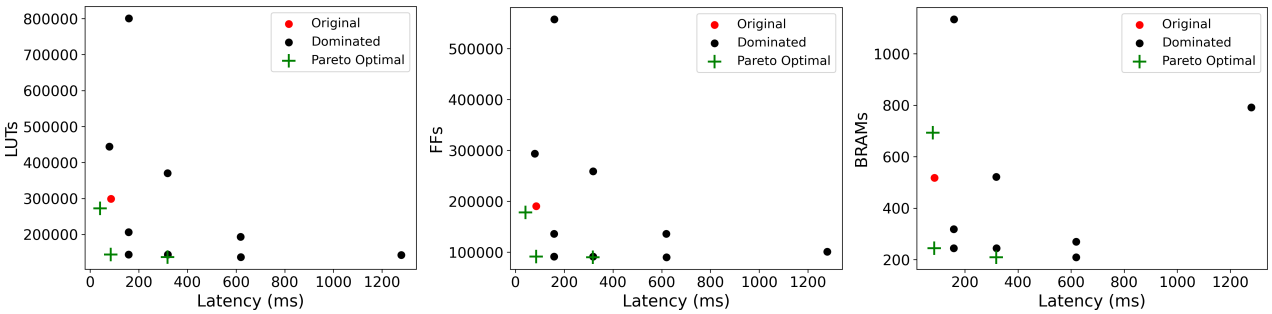


Figure 9.11: Design-space exploration of hardware implementations originating from VMM-15.

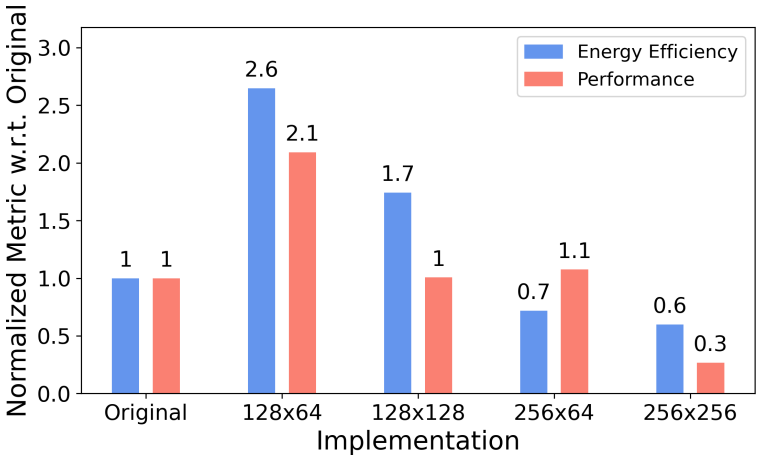
each FPGA resource in the top three plots (Figure 9.12(a), Figure 9.12(b) and Figure 9.12(c)), along with an analysis of performance and energy efficiency in the bottom plot (Figure 9.12(d)). VMM-64 is originally generated with a core capacity of 128×128 and could only be restructured down to 128×64 as the biggest MCC of VMM-64 has 92 axons $\times$ 64 neurons. After model restructuring and DSE, the best implementation for VMM-64 is the one with the smallest core capacity (128×64), which improves performance by 2.1$\times$ and energy efficiency by 2.6$\times$.

For all the SNN models, restructuring to the smallest core capacity possible gives the best performance and energy efficiency. As we increase the size of the matrix in the VMM model, the size of the largest MCC increases as well, which directly affects the minimum core capacity. A bigger matrix also results in a larger model with more MCCs.

Hence, we generate additional VMM models with different matrix shapes and inherently distinct maximum MCC sizes, which range between 264 (22 axons $\times$ 12 neurons) and 10112 (158



(a) (b) (c) excludes #BRAMs = 0



(d) Analysis of performance and energy efficiency

Figure 9.12: Design-space exploration of hardware implementations originating from VMM-64.

axons × 64 neurons). We restructure each model to its corresponding minimum power-of-two core capacity. We compare the performance and energy efficiency of the restructured architectures to the ones generated by RANC.

Figure 9.13 shows the results of performance and energy efficiency for models with different maximum MCC sizes. In particular, restructuring to the smallest core capacity improves performance by 1.90 – 3.73× and energy efficiency by 1.97 – 6.96×. Overall, the greater the maximum MCC size, the lower the performance, which in turn results in lower energy efficiency. The main reason is that bigger MCCs necessitate larger core capacities, which operate at slower tick frequencies. Another reason is that VMM models with bigger matrices typically require more tick periods to provide results, which also worsens performance.

Typically, energy efficiency and performance worsen as MCCs grow in size, but model restructuring helps minimizing this effect for varying-size VMM computations.

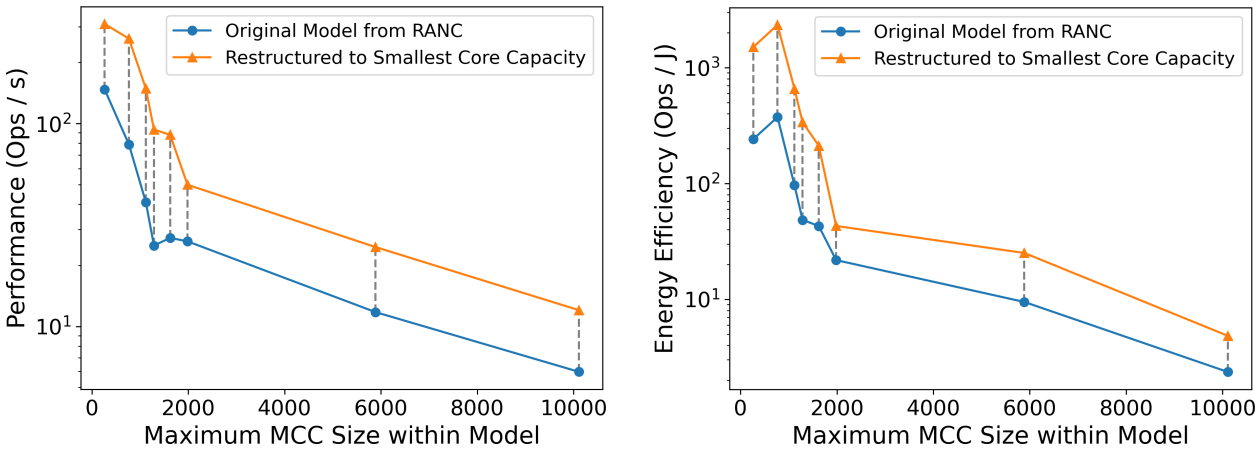


Figure 9.13: Performance and energy-efficiency for VMM models with different matrix shapes.

However, the two models with the smallest MCC sizes exhibit a unique phenomenon in terms of their energy efficiency. Specifically, the model with larger MCCs demonstrates better energy efficiency, which contrasts with the usual trend. These two models correspond to an 8 by 8 matrix (VMM-8) and a 6 by 6 matrix (VMM-O). The largest MCC size for VMM-8 is 264 (22 axons $\times$ 12 neurons) and for VMM-O it is 768 (32 axons $\times$ 24 neurons). Therefore, although VMM-8 has a bigger matrix, it has a smaller maximum MCC size. As a result, VMM-8 is restructured to a core capacity of 32 $\times$ 16, which is smaller than the capacity used by VMM-O, namely 32 $\times$ 32. However, VMM-8 has 1.6$\times$ more axons than VMM-O and requires more resources, making it less energy efficient.

9.4.7 Resource Overhead of Programmability

While SpikeHard is a runtime-programmable neuromorphic accelerator, the architecture of the inner neuromorphic processor is largely the same as in RANC. The controller and DMA handler at the interface of SpikeHard represent the main components that enable programmability and differentiate it from the non-programmable RANC. The SpikeHard interface replaces the basic AXI interface and immutable tick generator used in RANC, and requires additional resources for dynamic model-loading that scale with the core capacity and number of cores.

To better understand the overhead of adding programmability to the neuromorphic processor,

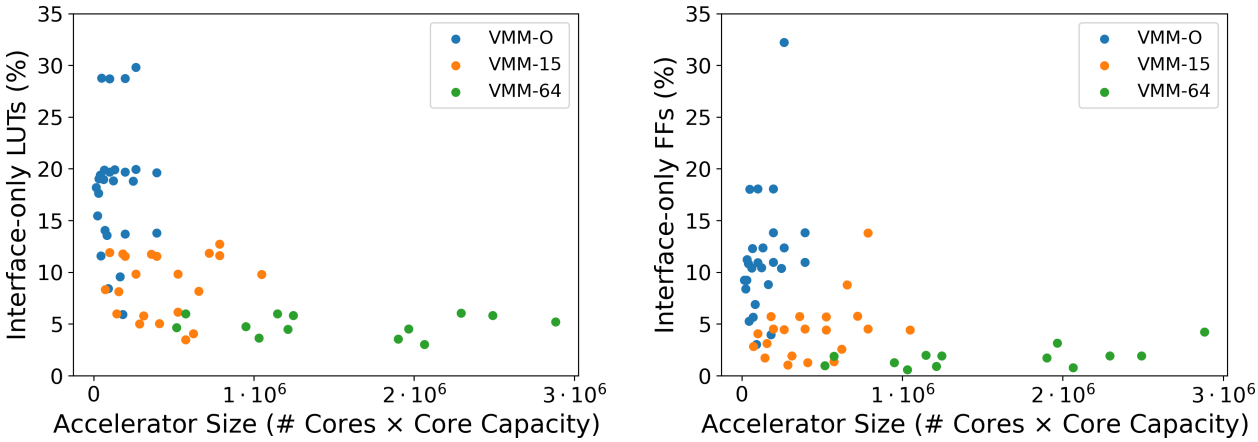


Figure 9.14: Proportion of resources used by the accelerator interface of SpikeHard.

we analyze the resource utilization of the SpikeHard interface in proportion to that of the entire accelerator. For all VMM implementations, namely VMM-O, VMM-15, and VMM-64, the interface does not use any BRAMs and requires only 11 DSPs, which is also the total number of DSPs used by the entire accelerator. The results for LUTs and FFs are summarized in Figure 9.14.

Typically, the larger the cores are and the greater their number, so are the total LUTs and FFs. However, the amount of resources used for the interface scales logarithmically while for the rest of the accelerator, namely the neuromorphic processor, the amount of resources scales linearly. When the accelerator is small, the proportion of resources used by the interface is around 30%, but the larger the accelerator, the smaller this proportion becomes, which means less overhead is incurred by enabling programmability. Overall, for an accelerator of average size, the proportions of LUTs and FFs used for programmability are only ~8% and ~5%, respectively.

9.4.8 Parallel Execution of Accelerators in a Many-Accelerator SoC

We evaluate SpikeHard as an accelerator in a many-accelerator SoC to determine whether SpikeHard has equivalent performance to accelerators that execute non-neuromorphic kernels, namely FFT and CONV2D, which are both used in practice by other ML workloads and alongside neuromorphic kernels [279, 280]. To this end, we create a software application where SpikeHard, FFT, and CONV2D accelerators are invoked in parallel, each taking as input ~32 kB of data.

We first test VMM-O without applying model restructuring as the baseline model for SpikeHard. In this test, SpikeHard, FFT, and CONV2D take an average of 12.9 ms, 5.1 ms, and 9.7 ms, respectively. In other words, SpikeHard with non-restructured VMM-O produces a slowdown of $1.3-2.5\times$ with respect to the other accelerators.

We then test implementation *A* of SpikeHard from our DSE as it shows the best performance and energy efficiency among the Pareto-optimal implementations of VMM-O (Table 9.3). With implementation *A*, SpikeHard performs the same computation in only 5.0 ms, running faster than FFT and CONV2D, which take the same amount of time as in the previous test. Thus, by using model restructuring and DSE, we are able to speed up SpikeHard to the point that it is not a performance bottleneck on the heterogeneous many-accelerator SoC, and can be safely used to execute neuromorphic workloads in synergy with other kinds of workloads.

SpikeHard is a neuromorphic accelerator with equivalent performance to non-neuromorphic accelerators, and can be seamlessly used with other accelerators in a heterogeneous many-accelerator SoC.

9.5 Related Work

A variety of embedded applications handle both neuromorphic and non-neuromorphic tasks [279, 280, 150]. Typically, the input data is pre-processed using a non-neuromorphic operation, such as an FFT [279], a 2D convolution [280], or a Singular Value Decomposition (SVD) [150], with the output of this being fed into a neuromorphic model.

Many algorithms have already been proposed for mapping an SNN to neuromorphic hardware, most of which minimize spike traffic on the neuromorphic core interconnect [142, 286, 287, 288]. For instance, Titirsha *et al.* [287] iteratively try to find a randomly-generated mapping that minimizes the energy consumption of spike communication. Balaji *et al.* [286] run the SNN in simulation and analyze the spike traffic to find a mapping that minimizes spike communication latency and energy consumption. In contrast, our model restructuring method, which can be used for mapping SNNs to hardware, minimizes resource utilization by leveraging a fine-grained partitioning

strategy for SNNs (MCC analysis), and has been shown to improve energy efficiency.

Unlike the TrueNorth [142] and Loihi 2 [145] neuromorphic processors, RANC [144] is a neuromorphic architecture that is highly reconfigurable at design time, available as an open-source project, and deployable on FPGA. To shorten the path of prototyping a heterogeneous SoC with neuromorphic hardware on FPGA, we used RANC as the baseline design for SpikeHard. Unlike RANC, SpikeHard can be reconfigured at runtime and has an interface that allows it to be seamlessly integrated in heterogeneous many-accelerator SoCs.

SoCs equipped with neuromorphic hardware are available, but they rarely consider the integration of non-neuromorphic hardware accelerators alongside the neuromorphic ones. For instance, Akida [277, 278] is an SoC containing an ARM processor that controls the neuromorphic hardware, but does not support the on-chip integration of other hardware accelerators. Similarly, the Loihi SoC [143] consists of a neuromorphic processor and an Intel x86 processor, which merely manages the neuromorphic processor. Loihi is typically deployed as part of a Nahuku expansion board [289, 290], which contains up to 32 interconnected Loihi chips that all interface with the same on-board Arria 10 FPGA. This FPGA is used to interact with the surroundings, notably via sensors and actuators. Additionally, the FPGA is controlled by an off-chip processor (distinct from the aforementioned x86 processor) from which the high-level software application is executed, such as compiling the neuromorphic model and visualizing results. Other neuromorphic chips, such as Caspian [291, 292] and DYNAP-CNN [293, 294], implement pure neuromorphic processing, without the realization of heterogeneous SoCs. Yet, embedded applications that use neuromorphic and non-neuromorphic tasks create a need for heterogeneous SoCs that can accelerate all of these tasks.

By contrast, SpikeHard is integrated as part of a standalone SoC consisting of not only a general-purpose 64-bit CVA6 RISC-V processor [193], but also non-neuromorphic accelerators, where the processor orchestrates the execution of the accelerators via a common standard interface. In this way, the neuromorphic hardware interfaces directly with an on-chip processor that runs the software applications. This results in less off-chip communication, potentially yielding

lower latency and higher efficiency. Moreover, designing an SoC with all the components integrated together on the same chip allows for a reduction in communication overhead.

SpikeHard is a neuromorphic accelerator designed for integration alongside non-neuromorphic accelerators in heterogeneous SoCs, allowing all accelerators to be invoked from the same software application. This work supports the future development and deployment of efficient neuromorphic hardware that can be seamlessly integrated into predominantly non-neuromorphic and highly heterogeneous SoCs.

Epilogue

Discussion and Future Work

In this dissertation, I have divided BCI system-level design into three key focus areas, each corresponding to a distinct time domain. For each domain, I have outlined my contributions. However, this work represents only the groundwork toward the final goal, and much more remains to be done in all three domains to bring real-world, widely accessible, self-contained BCI systems closer to reality.

In the ***Pre-BCI*** domain, I have presented a potential standard for a BCI system, based on an implanted SoC and a wearable SoC, supporting short-range wireless communication through a custom communication protocol, integrating a large-scale neural interface, and enabling reconfiguration capabilities through a custom control flow implemented with specialized microcontrollers for both SoCs. This work led to the development of two implant-based BCI systems, **BISC** [21, 49] and **IBISX** [50], which were successfully proven safe and reliable in in-vivo testing, and hopefully, human testing will follow soon. However, there is still much work ahead:

1. **Miniaturizing the Wearable SoC:** The wearable SoC, or relay station, should be optimized to reduce its size. While the current design based on FPGA technology allows flexibility for research and updates, it also adds unnecessary bulk that is impractical for commercial or mobile applications. Future work should focus on specializing the design of the relay station for BCI applications by transitioning to an ASIC-based system or a hybrid system that incorporates an embedded FPGA (eFPGA) for partial reconfiguration [216]. This would enable future updates to the instruction set architecture (ISA) while maintaining a more compact and efficient design.

2. **Expanding Neural Interface Integration:** Currently, our implant-based BCI systems integrate

a non-penetrating ECoG MEA and an array of SPADs with LEDs. To further enhance system capabilities, future work should explore integrating additional neural interface types that offer greater biocompatibility, longer interface longevity, and improved computational performance. This would ensure the system stays aligned with the latest advancements and standards in neural interfaces [295, 31, 296, 297]. Furthermore, given that we have successfully used a similar control flow and communication protocol across BCI systems that integrate different neural interface types, combining multiple types of interfaces into the same SoC design could provide significant benefits. This approach would enable BCI applications to leverage diverse neural data measurements and accommodate various neuronal stimulation flows.

3. **Improving Wireless Communication:** To accommodate the increased data throughput required by larger neural interfaces, the BCI system will need to integrate low-power wireless communication IPs that can support higher bandwidths [168, 169, 170]. Future work should focus on developing and integrating these advanced communication technologies to handle the growing demands of high-resolution neural data transmission.

4. **Supporting Multi-Tenancy in BCI Systems:** With the increasing scale of neural interfaces on implanted SoCs, future BCI systems should be designed to integrate multiple implanted SoCs, located in different parts of the brain [46], which would allow different BCI applications to run on the wearable SoC simultaneously, each communicating with a separate implanted SoC. This multi-tenancy of BCI applications would require developing communication protocols for seamless interaction between the wearable SoC and multiple implanted SoCs, enabling efficient data management and synchronization across the system. The idea of supporting multi-tenancy of ML-based applications on SoCs has already been discussed in other fields as well [298, 299].

In the ***Intra-BCI*** domain, I have analyzed the integration of computation within the BCI system. For the pre-BCI domain, computation is not yet essential for the core function of the BCI system, which is to provide high-resolution, high-throughput neural data for understanding the brain. However, to transition to a fully self-contained BCI system capable of running complete

real-world applications, computation needs to be integrated into the BCI system itself.

Through **MINDFUL**, I have demonstrated the potential benefits of performing computation on the implanted SoC, particularly in reducing transmission data rates. Despite this, the current state of machine learning models used in BCI applications presents challenges for efficient hardware offloading, due to the power limitations of implanted SoCs. Substantial optimizations are needed to design both the SoC and computational workloads to meet these constraints. On the other hand, **MasterMind** [43] introduces a practical methodology for implementing BCI applications on the wearable SoC, where all neural data can be transmitted in real-time from the implanted SoC. This is an intermediate solution that enables the creation of self-contained BCI systems. Building upon this, **MindCrypt** [44] expands the concept of a self-contained BCI system by utilizing the brain as a resource for high-throughput random bit generation to support various applications on the wearable SoC. However, there is still significant work ahead, with key areas for future development:

1. **Expanding the MINDFUL model and adapting to emerging technologies:** While the current focus of MINDFUL is on DNNs for BCI applications, it should be extended to include other computational methods, such as SNNs and control algorithms, as well as incorporating wetware and neuronal organoids. These emerging computational techniques can be modeled similarly to DNNs, with the emphasis on one-way non-Von Neumann data flow, although different processing units might be necessary to better represent these methods. Additionally, the MINDFUL model should be regularly updated to reflect advancements in neural interfaces, wireless communication technologies, and computational models, ensuring it remains aligned with the state of the art and the evolving capabilities of BCI systems.

2. **Optimizing computation on the wearable SoC:** MasterMind provides a methodology for implementing BCI applications in hardware on the wearable SoC, addressing its constraints. However, the maximum transmission data rates of the implanted SoC may limit the real-time transfer of neural data. To overcome this, MasterMind should be extended to guide the development of hybrid computation on the wearable and implanted SoCs. The goal would be to design self-contained BCI systems where critical computation is executed directly on the implanted SoC,

based on insights gained from MINDFUL.

3. **Supporting multi-tenancy of BCI applications:** With the increasing scale of neural interfaces on implanted SoCs, future BCI systems should be designed to integrate multiple implanted SoCs to support multi-tenancy, as discussed in the Pre-BCI future work (point 4). This would allow different BCI applications to run on the wearable SoC simultaneously, each communicating with a separate implanted SoC. In the Intra-BCI domain, this multi-tenancy should be supported by software running on the wearable SoC, enabling coordination between multiple implanted SoCs and managing the parallel execution of BCI applications. This can be incorporated into the extension of the MasterMind methodology. I have already made contributions in the field of transparent offloading of ML workloads on heterogeneous many-accelerator SoCs, with systems like WOLT [299] and EigenEdge [285], as well as improving the integration flow of hardware accelerators in heterogeneous SoCs [262, 300] and supporting unique features like triple modular redundancy (TMR) [301]. These contributions could be leveraged to optimize multi-tenant execution in BCI systems.

4. **The brain as a computational resource:** The vision behind MindCrypt is to show how the brain itself can be utilized as a resource in the BCI system to relax physical constraints. In the future, the BCI system should be considered a co-processor for the brain [209, 302]. The brain, with its energy-efficient processing capabilities, consuming only 20W of power [274, 303], can be harnessed to execute complex tasks alongside the computation performed by SoCs. As neural interfaces grow larger and our understanding of the brain improves, the BCI system can integrate the brain into the computational flow, allowing us to implement new computational processes that consider the brain as a computational engine in BCI applications.

5. **Enhancing security and privacy in the BCI system:** MindCrypt provides a pathway to enhance security and privacy within BCI systems. By leveraging brain activity to generate random numbers, both the implanted SoC and the wearable SoC can implement cryptographic protocols to protect data transmission between the two SoCs. With the addition of authentication mecha-

nisms based on brain activity [13], and further exploration of the statistical attributes of neural data, future work can focus on developing mechanisms to secure the communication channel between the two SoCs. Additionally, differential privacy techniques [304, 305] can be explored, especially in scenarios where data providers may be driven by commercial interests that could compromise security [236].

In the ***Post-BCI*** domain, I have focused on exploring how to specialize computation models for processing neural data in an energy-efficient and real-time manner. The goal is to address the growing scale of generic ML models, which have become increasingly difficult to integrate into a self-contained BCI system.

Through **KalmMind** [52, 51], I demonstrated how small adjustments to the Kalman filter algorithm can significantly enhance energy efficiency, while also offering the ability to specialize the algorithm for specific types of neural data, potentially tuning computational accuracy for different BCI applications. In addition, **SpikeHard** [53] introduced a programmable hardware accelerator for neuromorphic computing, integrated within a many-accelerator heterogeneous SoC. This work serves as an initial step in enabling systems capable of running both neuromorphic and ML workloads on the same device. This aligns with the vision of creating self-contained BCI systems that interface with biological neurons using neuromorphic hardware, optimizing BCI applications through a complex computational flow combining biological neurons, SNNs, and DNNs. However, several areas of future work remain:

1. **Outsourcing computation for BCI systems:** As discussed in Chapter 7, one way to address the computational bottleneck in BCI systems is by extending the system to outsource computation to external high-performance machines capable of running advanced ML models. However, this approach necessitates the development of highly efficient communication protocols that preserve the real-time execution of BCI applications. Furthermore, there are strong ethical implications concerning the privacy of neural data, which must be addressed before BCI systems can be widely deployed for public use [306, 307].

2. **Specializing more algorithms for BCI applications:** The approach demonstrated in Kalm-

Mind for optimizing the Kalman filter can be extended to other algorithms commonly used in BCI applications. Future work should focus on identifying steps within these algorithms that can be optimized based on the specific characteristics of neural data. For instance, neural data is often sampled over time from the same sensors, with clear correlations between measurements from nearby neurons [308], which can be leveraged for algorithmic optimizations.

3. **Standardizing neuromorphic hardware integration:** There is still much work to be done in standardizing the implementation and integration of neuromorphic hardware, both for BCI systems and other applications [19, 17]. As shown in Figure 7.1, neuromorphic SoCs are already capable of supporting millions of neurons. However, our understanding of how to effectively leverage SNNs for solving complex tasks remains limited. Future research should focus on incorporating SNNs into BCI systems to enable energy-efficient and real-time computations [250, 88]. Additionally, advancements in wetware technology should be explored to improve interactions with biological neurons in the brain [245, 247, 246].

4. **Optimizing ML models:** Advancements in ML models often prioritize increasing the scale of computation to improve results [17, 227]. However, the future of BCI systems should focus on optimizing these models to achieve similar accuracy with significantly less computational power. Techniques such as the ones offered by DeepSeek [309, 310, 311] demonstrate that this is possible, and comparable computational accuracy can be achieved by reevaluating the structure and implementation of large AI models to reduce the computational power while maintaining performance. These optimizations are essential to accommodate the currently slower pace of hardware advancements compared to the increase in computation power needed for modern AI models [17], and to support the integration of applications in constrained embedded systems [188, 189].

Conclusions

This dissertation has provided a comprehensive exploration of system-level design in Brain-Computer Interfaces (BCI), addressing critical challenges and exploring potential pathways for the realization of fully functional, self-contained BCI systems. I have defined the system development process through three key domains—Pre-BCI, Intra-BCI, and Post-BCI—each of which plays a crucial role in overcoming the existing barriers to implementing self-contained BCI systems. Throughout the dissertation, I have emphasized the self-contained BCI system as the target system, providing a framework for advancing towards real-world BCI applications.

I contributed to the standardization of BCI system design by developing a custom communication protocol and microcontrollers for both the implanted and wearable SoCs. These contributions lay the groundwork for self-contained BCI systems, providing the essential infrastructure for neural data acquisition, wireless communication, and system configuration. The two implant-based BCI systems I presented—BISC and IBISX—were successfully tested in-vivo and proven safe, representing significant progress toward the development of practical, implant-based BCI systems.

I focused on integrating computation within the BCI system, particularly addressing the need for real-time execution of BCI applications. Through the development of MINDFUL, I demonstrated the need for optimizing ML models and implanted SoC designs for BCI applications, improving power efficiency and overall feasibility. Building on this, I presented MasterMind, as a methodology for implementing BCI applications on the wearable SoC, enabling the realization of self-contained BCI systems. With MindCrypt, I further expanded this concept, introducing the brain as a resource for high-throughput random number generation to support various BCI appli-

cations, thereby opening the path for including the brain as part of the self-contained BCI system.

I initiated a discussion on the future of BCI systems, focusing on enhancing and specializing computational models to interact with biological neurons. I developed KalmMind to demonstrate how simple modifications to the Kalman Filter algorithm could improve energy efficiency and offer tunable accuracy in BCI-based motion decoding. Meanwhile, I introduced SpikeHard, a programmable neuromorphic hardware accelerator, to promote the integration of neuromorphic and ML workloads on the same heterogeneous SoC. These contributions lay the foundation for developing more energy-efficient, specialized computation for BCI systems and for integrating biological neurons, SNNs, and DNNs into a unified computational flow.

While this dissertation has provided key insights and contributions, much work remains. Future research should focus on specializing, implementing, and integrating additional computational methods into BCI systems to address the unique challenges posed by BCI applications. Additionally, refining the integration of neuromorphic hardware, optimizing DNNs for lower power consumption, supporting security and data privacy, and enabling multi-tenancy in BCI systems hold significant potential. These advancements are essential for ensuring that BCI systems can scale effectively and meet the demands of real-world applications.

Ultimately, this dissertation lays the foundations for future advancements in BCI system design. As computer architects and engineers, it is our responsibility to continue refining these systems and pushing the boundaries of what is possible. The goal is to create safe, efficient, and practical BCI systems that not only revolutionize healthcare and neuromodulation but also enhance human-computer interaction and contribute to the development of emerging brain-inspired computational models. Through continued research and collaboration, we can turn the vision of widely accessible, fully functional, implant-based, self-contained BCI systems into reality, transforming the way we interact with the world and each other.

References

[1] A. Kübler, "The history of BCI: From a Vision for the Future to Real Support for Personhood in People with Locked-in Syndrome," *Neuroethics*, vol. 13, no. 2, pp. 163–180, 2020.

[2] T. Estrin, "On-line Electroencephalographic Digital Computing System," *Electroencephalography and Clinical Neurophysiology*, vol. 19, no. 5, pp. 524–526, 1965.

[3] J. J. Vidal, "Toward Direct Brain-Computer Communication," *Annual review of Biophysics and Bioengineering*, vol. 2, no. 1, pp. 157–180, 1973.

[4] J. d. R. Millán, R. Rupp, G. Mueller-Putz, R. Murray-Smith, C. Giugliemma, M. Tangermann, C. Vidaurre, F. Cincotti, A. Kubler, R. Leeb, *et al.*, "Combining Brain–Computer Interfaces and Assistive Technologies: State-of-the-Art and Challenges," *Frontiers in neuroscience*, p. 161, 2010.

[5] R. Rupp, S. C. Kleih, R. Leeb, J. del R. Millan, A. Kübler, and G. R. Müller-Putz, "Brain–Computer Interfaces and Assistive Technology," *Brain-Computer-Interfaces in their ethical, social and cultural contexts*, pp. 7–38, 2014.

[6] M. Vilela and L. R. Hochberg, "Applications of Brain-Computer Interfaces to the Control of Robotic and Prosthetic Arms," *Handbook of clinical neurology*, vol. 168, pp. 87–99, 2020.

[7] V. Gilja, P. Nuyujukian, C. A. Chestek, J. P. Cunningham, B. M. Yu, J. M. Fan, M. M. Churchland, M. T. Kaufman, J. C. Kao, S. I. Ryu, *et al.*, "A High-Performance Neural Prosthesis Enabled by Control Algorithm Design," *Nature neuroscience*, vol. 15, no. 12, pp. 1752–1757, 2012.

[8] P. Romanelli, M. Piangerelli, D. Ratel, C. Gaude, T. Costecalde, C. Puttilli, M. Picciafuoco, A. Benabid, and N. Torres, "A Novel Neural Prosthesis Providing Long-Term Electrocorticography Recording and Cortical Stimulation for Epilepsy and Brain-Computer Interface," *Journal of Neurosurgery*, vol. 130, no. 4, pp. 1166–1179, 2018.

[9] A. Dillen, E. Lathouwers, A. Miladinović, U. Marusic, F. Ghaffari, O. Romain, R. Meeusen, and K. De Pauw, "A Data-Driven Machine Learning Approach for Brain-Computer Interfaces Targeting Lower Limb Neuroprosthetics," *Frontiers in human neuroscience*, vol. 16, p. 949 224, 2022.

[10] A. Lozano, J. S. Suárez, C. Soto-Sánchez, J. Garrigós, J. J. Martínez-Alvarez, J. M. Ferrández, and E. Fernández, "Neurolight: A Deep Learning Neural Interface for Cortical Visual Prostheses," *International journal of neural systems*, vol. 30, no. 09, p. 2 050 045, 2020.

[11] M. van der Grinten, J. d. R. van Steveninck, A. Lozano, L. Pijnacker, B. Rueckauer, P. Roelfsema, M. van Gerven, R. van Wezel, U. Güçlü, and Y. Güçlütürk, "Towards Biologically Plausible Phosphene Simulation for the Differentiable Optimization of Visual Cortical Prostheses," *Elife*, vol. 13, e85812, 2024.

[12] S. Niketeghad and N. Pouratian, "Brain Machine Interfaces for Vision Restoration: The Current State of Cortical Visual Prosthetics," *Neurotherapeutics*, vol. 16, no. 1, pp. 134–143, 2019.

[13] F. Lin, K. W. Cho, C. Song, W. Xu, and Z. Jin, "Brain Password: A Secure and Truly Cancelable Brain Biometrics for Smart Headwear," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, 2018, pp. 296–309.

[14] J. Szczepanski, E. Wajnryb, J. M. Amigó, M. V. Sanchez-Vives, and M. Slater, "Biometric Random Number Generators," *Computers & Security*, vol. 23, no. 1, pp. 77–84, 2004.

[15] M. W. Mathis, A. P. Rotondo, E. F. Chang, A. S. Tolias, and A. Mathis, "Decoding the Brain: From Neural Representations to Mechanistic Models," *Cell*, vol. 187, no. 21, pp. 5814–5832, 2024.

[16] S. A. Cadena, K. F. Willeke, K. Restivo, G. Denfield, F. H. Sinz, M. Bethge, A. S. Tolias, and A. S. Ecker, "Diverse Task-Driven Modeling of Macaque V4 Reveals Functional Secialization Towards Semantic Tasks," *PLOS Computational Biology*, vol. 20, no. 5, e1012056, 2024.

[17] A. Mehonic and A. J. Kenyon, "Brain-Inspired Computing Needs a Master Plan," *Nature*, vol. 604, no. 7905, pp. 255–260, 2022.

[18] W. Zhang, S. Ma, X. Ji, X. Liu, Y. Cong, and L. Shi, "The Development of General-Purpose Brain-Inspired Computing," *Nature Electronics*, pp. 1–12, 2024.

[19] D. Kudithipudi, C. Schuman, C. M. Vineyard, T. Pandit, C. Merkel, R. Kubendran, J. B. Aimone, G. Orchard, C. Mayr, R. Benosman, *et al.*, "Neuromorphic Computing at Scale," *Nature*, vol. 637, no. 8047, pp. 801–812, 2025.

[20] K. Kumarasinghe, N. Kasabov, and D. Taylor, "Deep Learning and Deep Knowledge Representation in Spiking Neural Networks for Brain-Computer Interfaces," *Neural Networks*, vol. 121, pp. 169–185, 2020.

[21] N. Zeng, T. Jung, M. Sharma, G. Eichler, J. Fabbri, R. J. Cotton, E. Spinazzi, B. Youngerman, L. Carloni, and K. L. Shepard, "A Wireless, Mechanically Flexible, 25μm-Thick, 65,536-Channel Subdural Surface Recording and Stimulating Microelectrode Array with Integrated Antennas," in *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, IEEE, 2023, pp. 1–2.

[22] D.-Y. Yoon, S. Pinto, S. Chung, P. Merolla, T.-W. Koh, and D. Seo, "A 1024-Channel Simultaneous Recording Neural SoC with Stimulation and Real-Time Spike Detection," in *2021 Symposium on VLSI Circuits*, IEEE, 2021, pp. 1–2.

[23] J.-H. Cha, J.-H. Park, Y. Park, H. Shin, K. S. Hwang, I.-J. Cho, and S.-J. Kim, "A Reconfigurable Sub-Array Multiplexing Microelectrode Array System with 24,320 Electrodes and 380 Readout Channels for Investigating Neural Communication," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, IEEE, vol. 65, 2022, pp. 342–344.

[24] D. Kleinfeld, L. Luan, P. P. Mitra, J. T. Robinson, R. Sarpeshkar, K. Shepard, C. Xie, and T. D. Harris, "Can One Concurrently Record Electrical Spikes from Every Neuron in a Mammalian Brain?" *Neuron*, vol. 103, no. 6, pp. 1005–1015, 2019.

[25] A. E. Urai, B. Doiron, A. M. Leifer, and A. K. Churchland, "Large-Scale Neural Recordings Call for New Insights to Link Brain and Behavior," *Nature neuroscience*, vol. 25, no. 1, pp. 11–19, 2022.

[26] I. H. Stevenson and K. P. Kording, "How Advances in Neural Recording Affect Data Analysis," *Nature neuroscience*, vol. 14, no. 2, pp. 139–142, 2011.

[27] R. Abiri, S. Borhani, E. W. Sellers, Y. Jiang, and X. Zhao, "A Comprehensive Review of EEG-based Brain–Bomputer Interface Paradigms," *Journal of neural engineering*, vol. 16, no. 1, p. 011 001, 2019.

[28] K. Värbu, N. Muhammad, and Y. Muhammad, "Past, Present, and Future of EEG-based BCI Applications," *Sensors*, vol. 22, no. 9, p. 3331, 2022.

[29] G. Udovičić, A. Topić, and M. Russo, "Wearable Technologies for Smart Environments: A Review with Emphasis on BCI," in *2016 24th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, IEEE, 2016, pp. 1–9.

[30] A. Y. Dogan, J. Constantin, M. Ruggiero, A. Burg, and D. Atienza, "Multi-Core Architecture Design for Ultra-Low-Power Wearable Health Monitoring Systems," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2012, pp. 988–993.

[31] Y. Wang, X. Yang, X. Zhang, Y. Wang, and W. Pei, "Implantable Intracortical Microelectrodes: Reviewing the Present with a Focus on the Future," *Microsystems & Nanoengineering*, vol. 9, no. 1, p. 7, 2023.

[32] S. Wang, S. K. Garakoui, H. Chun, D. G. Salinas, W. Sijbers, J. Putzeys, E. Martens, J. Craninckx, N. Van Helleputte, and C. M. Lopez, "A Compact Quad-Shank CMOS Neural Probe with 5,120 Addressable Recording Sites and 384 Fully Differential Parallel Channels," *IEEE transactions on biomedical circuits and systems*, vol. 13, no. 6, pp. 1625–1634, 2019.

[33] S. Moazeni, E. H. Pollmann, V. Boominathan, F. A. Cardoso, J. T. Robinson, A. Veeraraghavan, and K. L. Shepard, "A Mechanically Flexible, Implantable Neural Interface for Computational Imaging and Optogenetic Stimulation Over 5.4× 5.4 mm 2 FoV," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 15, no. 6, pp. 1295–1305, 2021.

[34] J. Choi, A. J. Taal, E. H. Pollmann, C. Lee, K. Kim, L. C. Moreaux, M. L. Roukes, and K. L. Shepard, "A 512-Pixel, 51-KHz-Frame-Rate, Dual-Shank, Lens-Less, Filter-Less Single-Photon Avalanche Diode CMOS Neural Imaging Probe," *IEEE journal of solid-state circuits*, vol. 54, no. 11, pp. 2957–2968, 2019.

[35] O. Yizhar, L. E. Fenno, T. J. Davidson, M. Mogri, and K. Deisseroth, "Optogenetics in Neural Systems," *Neuron*, vol. 71, no. 1, pp. 9–34, 2011.

[36] E. H. Pollmann, Y. Gilhotra, H. Yin, and K. L. Shepard, "Fully Implantable 192× 256 SPAD Sensor with Global-Shutter and Micro-LEDs for Bidirectional Subdural Optical Brain-Computer Interfaces," in *ESSCIRC 2022-IEEE 48th European Solid State Circuits Conference (ESSCIRC)*, IEEE, 2022, pp. 205–208.

[37] N. Even-Chen, D. G. Muratore, S. D. Stavisky, L. R. Hochberg, J. M. Henderson, B. Murmann, and K. V. Shenoy, "Power-Saving Design opportunities for Wireless Intracortical Brain–Computer Interfaces," *Nature biomedical engineering*, vol. 4, no. 10, pp. 984–996, 2020.

[38] X. Tang, H. Shen, S. Zhao, N. Li, and J. Liu, "Flexible Brain–Computer Interfaces," *Nature Electronics*, vol. 6, no. 2, pp. 109–118, 2023.

[39] T. J. Oxley, N. L. Opie, S. E. John, G. S. Rind, S. M. Ronayne, T. L. Wheeler, J. W. Judy, A. J. McDonald, A. Dornom, T. J. Lovell, *et al.*, "Minimally Invasive Endovascular Stent-Electrode Array for High-Fidelity, Chronic Recordings of Cortical Neural Activity," *Nature biotechnology*, vol. 34, no. 3, pp. 320–327, 2016.

[40] P. D. Wolf and W. Reichert, "Thermal Considerations for the Design of an Implanted Cortical Brain–Machine Interface (BMI)," *Indwelling Neural Implants: Strategies for Contending with the In Vivo Environment*, pp. 33–38, 2008.

[41] C. Serrano-Amenos, F. Hu, P. T. Wang, S. Kellis, R. A. Andersen, C. Y. Liu, P. Heydari, A. H. Do, and Z. Nenadic, "Thermal Analysis of a Skull Implant in Brain-Computer Interfaces," in *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, IEEE, 2020, pp. 3066–3069.

[42] K. J. Miller, D. Hermes, and N. P. Staff, "The Current State of Electrocorticography-based Brain-Computer Interfaces," *Neurosurgical focus*, vol. 49, no. 1, E2, 2020.

[43] G. Eichler, L. Piccolboni, D. Giri, and L. P. Carloni, "MasterMind: Many-Accelerator SoC Architecture for Real-Time Brain-Computer Interfaces," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, IEEE, 2021, pp. 101–108.

[44] G. Eichler, B. Seyoum, K. L. Chiu, and L. P. Carloni, "MindCrypt: The Brain as a Random Number Generator for SoC-Based Brain-Computer Interfaces," in *2023 IEEE 41th International Conference on Computer Design (ICCD)*, IEEE, 2023.

[45] I. Karageorgos, K. Sriram, J. Veselỳ, M. Wu, M. Powell, D. Borton, R. Manohar, and A. Bhattacharjee, "Hardware-Software Co-Design for Brain-Computer Interfaces," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2020, pp. 391–404.

[46] K. Sriram, R. P. Pothukuchi, M. Gerasimiuk, M. Ugur, O. Ye, R. Manohar, A. Khandelwal, and A. Bhattacharjee, "SCALO: An Accelerator-Rich Distributed System for Scalable Brain-Computer Interfacing," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–20.

[47] N. Rao, *400 Billion Reasons To Believe In Brain-Computer Interfaces*, Forbes Innovation in Healthcare, Accessed: 2025-01-15, 2024.

[48] J. T. Robinson, S. L. Norman, M. R. Angle, T. G. Constandinou, T. Denison, J. P. Donoghue, R. M. Field, A. Forsland, S. Kouider, J. d. R. Millán, *et al.*, "An Application-Based Taxonomy for Brain–Computer Interfaces," *Nature Biomedical Engineering*, pp. 1–3, 2024.

[49] T. Jung, N. Zeng, J. D. Fabbri, G. Eichler, Z. Li, K. Willeke, K. E. Wingel, A. Dubey, R. Huq, M. Sharma, *et al.*, "Stable, Chronic In-Vivo Recordings from a Fully Wireless Subdural-Contained 65,536-Electrode Brain-Computer Interface Device," *bioRxiv*, pp. 2024–05, 2024.

[50] Y. Gilhotra, H. Overhauser, H. Yin, E. Pollmann, G. Eichler, A. Cheng, J. Taesung, N. Zeng, L. P. Carloni, and K. Shepard, "A Wireless Subdural Optical Cortical Interface Device with 768 Co-Packaged Micro-LEDs for Fluorescence Imaging and Optogenetic Stimulation," in *IEEE Custom Integrated Circuits Conference (CICC)*, IEEE, 2024.

[51] G. Eichler, J. Zuckerman, and L. P. Carloni, "An Energy-Efficient Kalman Filter Architecture with Tunable Accuracy for Brain-Computer Interfaces," in *2025 ACM/IEEE 61st Design Automation Conference (DAC)*, IEEE, 2025.

[52] G. Eichler, J. Zuckerman, and L. P. Carloni, "KalmMind: A Configurable Kalman Filter Design Framework for Embedded Brain-Computer Interfaces," in *2025 IEEE Design, Automation and Test in Europe Conference (DATE)*, IEEE, 2025.

[53] J. Clair, G. Eichler, and L. P. Carloni, "SpikeHard: Efficiency-Driven Neuromorphic Hardware for Heterogeneous Systems-on-Chip," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 5s, pp. 1–22, 2023.

[54] R. Ajmeria, M. Mondal, R. Banerjee, T. Halder, P. K. Deb, D. Mishra, P. Nayak, S. Misra, S. K. Pal, and D. Chakravarty, "A Critical Survey of EEG-based BCI Systems for Applications in Industrial Internet of Things," *IEEE Communications Surveys & Tutorials*, 2022.

[55] X. Wang, C. A. Gkogkidis, O. Iljina, L. D. Fiederer, C. Henle, I. Mader, J. Kaminsky, T. Stieglitz, M. Gierthmuehlen, and T. Ball, "Mapping the Fine Structure of Cortical Activity with Different micro-ECoG Electrode Array Geometries," *Journal of neural engineering*, vol. 14, no. 5, p. 056 004, 2017.

[56] M. Hettick, E. Ho, A. J. Poole, M. Monge, D. Papageorgiou, K. Takahashi, M. LaMarca, D. Trietsch, K. Reed, M. Murphy, *et al.*, "The Layer 7 Cortical Interface: A Scalable and Minimally Invasive Brain–Computer Interface Platform," *bioRxiv*, pp. 2022–01, 2022.

[57] M. Jang, W.-H. Yu, C. Lee, M. Hays, P. Wang, N. Vitale, P. Tandon, P. Yan, P.-I. Mak, Y. Chae, *et al.*, "A 1024-Channel 268 nW/pixel 36x36 μm 2/ch Data-Compressive Neural Recording IC for High-Bandwidth Brain-Computer Interfaces," in *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, IEEE, 2023, pp. 1–2.

[58] H. Zaer, A. Deshmukh, D. Orlowski, W. Fan, P.-H. Prouvot, A. N. Glud, M. B. Jensen, E. S. Worm, S. Lukacova, T. W. Mikkelsen, *et al.*, "An Intracortical Implantable Brain-Computer Interface for Telemetric Real-Time Recording and Manipulation of Neuronal Circuits for Closed-Loop Intervention," *Frontiers in Human Neuroscience*, vol. 15, p. 618 626, 2021.

[59] L. C. Moreaux, D. Yatsenko, W. D. Sacher, J. Choi, C. Lee, N. J. Kubat, R. J. Cotton, E. S. Boyden, M. Z. Lin, L. Tian, *et al.*, "Integrated Neurophotonics: Toward Dense Volumetric Interrogation of Brain Circuit Activity—at Depth and in Real Time," *Neuron*, vol. 108, no. 1, pp. 66–92, 2020.

[60] N. A. Steinmetz, C. Aydin, A. Lebedeva, M. Okun, M. Pachitariu, M. Bauza, M. Beau, J. Bhagat, C. Böhm, M. Broux, *et al.*, "Neuropixels 2.0: A Miniaturized High-Density Probe for Stable, Long-Term Brain Recordings," *Science*, vol. 372, no. 6539, eabf4588, 2021.

[61] Y. Jia, U. Guler, Y.-P. Lai, Y. Gong, A. Weber, W. Li, and M. Ghovanloo, "A Trimodal Wireless Implantable Neural Interface System-on-Ship," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 6, pp. 1207–1217, 2020.

[62] J. Jung, S. Zhu, P. Liu, Y.-J. E. Chen, and D. Heo, "22-pJ/bit Energy-Efficient 2.4-GHz Implantable OOK Transmitter for Wireless Biotelemetry Systems: In Vitro Experiments

Using Rat Skin-Mimic," *IEEE transactions on microwave theory and techniques*, vol. 58, no. 12, pp. 4102–4111, 2010.

[63] C. K. Ho, J. H. Cheong, and Y. Gao, "A High Datarate Wideband OOK Transmitter for Wireless Neural Signal Recording," in *2015 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, IEEE, 2015, pp. 178–181.

[64] A. D. Degenhart, W. E. Bishop, E. R. Oby, E. C. Tyler-Kabara, S. M. Chase, A. P. Batista, and B. M. Yu, "Stabilization of a Brain-Computer Interface via the Alignment of Low-Dimensional Spaces of Neural Activity," *Nature biomedical engineering*, vol. 4, no. 7, pp. 672–685, 2020.

[65] J. Lee, M. Dabagia, E. Dyer, and C. Rozell, "Hierarchical Optimal Transport for Multimodal Distribution Alignment," *Advances in neural information processing systems*, vol. 32, 2019.

[66] E. L. Dyer, M. Gheshlaghi Azar, M. G. Perich, H. L. Fernandes, S. Naufel, L. E. Miller, and K. P. Körding, "A Cryptography-based Approach for Movement Decoding," *Nature biomedical engineering*, vol. 1, no. 12, pp. 967–976, 2017.

[67] E. Fernández, A. Alfaro, C. Soto-Sánchez, P. Gonzalez-Lopez, A. M. Lozano, S. Peña, M. D. Grima, A. Rodil, B. Gómez, X. Chen, *et al.*, "Visual Percepts Evoked with an Intracortical 96-Channel Microelectrode Array Inserted in Human Occipital Cortex," *The Journal of clinical investigation*, vol. 131, no. 23, 2021.

[68] Z. Ding, P. G. Fahey, S. Papadopoulos, E. Y. Wang, B. Celii, C. Papadopoulos, A. B. Kunin, A. Chang, J. Fu, Z. Ding, *et al.*, "Functional Connectomics Reveals General Wiring Rule in Mouse Visual Cortex," *bioRxiv*, 2023.

[69] A. M. Zador, "A Critique of Pure Learning and what Artificial Neural Networks can learn from Animal Brains," *Nature communications*, vol. 10, no. 1, p. 3770, 2019.

[70] L. Yao, B. Zhu, and M. Shoaran, "Fast and Accurate Decoding of Finger Movements from ECoG through Riemannian Features and Modern Machine Learning Techniques," *Journal of Neural Engineering*, vol. 19, no. 1, p. 016 037, 2022.

[71] M. Śliwowski, M. Martin, A. Souloumiac, P. Blanchart, and T. Aksenova, "Decoding ECoG Signal into 3D Hand Translation Using Deep Learning," *Journal of neural engineering*, vol. 19, no. 2, p. 026 023, 2022.

[72] M. Angrick, C. Herff, E. Mugler, M. C. Tate, M. W. Slutzky, D. J. Krusienski, and T. Schultz, "Speech Synthesis from ECoG Using Densely Connected 3D Convolutional Neural Networks," *Journal of neural engineering*, vol. 16, no. 3, p. 036 019, 2019.

[73] S. Luo, Q. Rabbani, and N. E. Crone, "Brain-computer interface: Applications to speech decoding and synthesis to augment communication," *Neurotherapeutics*, vol. 19, no. 1, pp. 263–273, 2022.

[74] J. Berezutskaya, Z. V. Freudenburg, M. J. Vansteensel, E. J. Aarnoutse, N. F. Ramsey, and M. A. van Gerven, "Direct Speech Reconstruction from Sensorimotor Brain Activity with Optimized Deep Learning Models," *Journal of Neural Engineering*, vol. 20, no. 5, p. 056 010, 2023.

[75] S. T. Kanth and S. Ray, "Electrocorticogram (ECoG) is Highly Informative in Primate Visual Cortex," *Journal of Neuroscience*, vol. 40, no. 12, pp. 2430–2444, 2020.

[76] J. P. Cunningham, P. Nuyujukian, V. Gilja, C. A. Chestek, S. I. Ryu, and K. V. Shenoy, "A Closed-Loop Human Simulator for Investigating the Role of Feedback Control in Brain-Machine Interfaces," *Journal of neurophysiology*, vol. 105, no. 4, pp. 1932–1949, 2011.

[77] D. Wu, J. Li, Z. Pan, Y. Kim, and J. S. Miguel, "uBrain: A Unary Brain Computer Interface," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 468–481.

[78] A. Zador, S. Escola, B. Richards, B. Ölveczky, Y. Bengio, K. Boahen, M. Botvinick, D. Chklovskii, A. Churchland, C. Clopath, *et al.*, "Catalyzing Next-Generation Artificial Intelligence through Neuroai," *Nature communications*, vol. 14, no. 1, p. 1597, 2023.

[79] M. Schrimpf, J. Kubilius, H. Hong, N. J. Majaj, R. Rajalingham, E. B. Issa, K. Kar, P. Bashivan, J. Prescott-Roy, F. Geiger, *et al.*, "Brain-Score: Which Artificial Neural Network for Object Recognition is Most Brain-Like?" *BioRxiv*, p. 407 007, 2018.

[80] J. Zhu, T. Zhang, Y. Yang, and R. Huang, "A Comprehensive Review on Emerging Artificial Neuromorphic Devices," *Applied Physics Reviews*, vol. 7, no. 1, 2020.

[81] W. Wu, M Black, Y. Gao, M Serruya, A Shaikhouni, J Donoghue, and E. Bienenstock, "Neural Decoding of Cursor Motion Using a Kalman Filter," *Advances in neural information processing systems*, vol. 15, 2002.

[82] P. Sykacek, S. J. Roberts, and M. Stokes, "Adaptive BCI based on Variational Bayesian Kalman Filtering: An Empirical Evaluation," *IEEE Transactions on biomedical engineering*, vol. 51, no. 5, pp. 719–727, 2004.

[83] J. I. Glaser, A. S. Benjamin, R. H. Chowdhury, M. G. Perich, L. E. Miller, and K. P. Kording, "Machine Learning for Neural Decoding," *Eneuro*, vol. 7, no. 4, 2020.

[84] T. Hosman, M. Vilela, D. Milstein, J. N. Kelemen, D. M. Brandman, L. R. Hochberg, and J. D. Simeral, "BCI Decoder Performance Comparison of an LSTM Recurrent Neu-

ral Network and a Kalman Filter in Retrospective Simulation," in *2019 9th International IEEE/EMBS conference on neural engineering (NER)*, IEEE, 2019, pp. 1066–1071.

[85] S. Wojtowytsch and E Weinan, "Can Shallow Neural Networks Beat the Curse of Dimensionality? a Mean Field Training Perspective," *IEEE Transactions on Artificial Intelligence*, vol. 1, no. 2, pp. 121–129, 2020.

[86] F. Y. Kuo and I. H. Sloan, "Lifting the Curse of Dimensionality," *Notices of the AMS*, vol. 52, no. 11, pp. 1320–1328, 2005.

[87] M. Ugur, R. Pothukuchi, and A. Bhattacharjee, "Swapping-Centric Neural Recording Systems," 2024.

[88] P. Hueber, G. Tang, M. Sifalakis, H.-P. Liaw, A. Micheli, N. Tömen, and Y.-H. Liu, "Benchmarking of Hardware-Efficient Real-Time Neural Decoding in Brain-computer Interfaces," *Authorea Preprints*, 2023.

[89] C. Hao, J. Dotzel, J. Xiong, L. Benini, Z. Zhang, and D. Chen, "Enabling Design Methodologies and Future Trends for Edge AI: Specialization and Codesign," *IEEE Design & Test*, vol. 38, no. 4, pp. 7–26, 2021.

[90] S. Naveen and M. R. Kounte, "Key Technologies and Challenges in IoT Edge Computing," in *2019 Third international conference on I-SMAC (IoT in social, mobile, analytics and cloud)(I-SMAC)*, IEEE, 2019, pp. 61–65.

[91] A. Gamatié, G. Devic, G. Sassatelli, S. Bernabovi, P. Naudin, and M. Chapman, "Towards Energy-Efficient Heterogeneous Multicore Architectures for Edge Computing," *IEEE Access*, vol. 7, pp. 49 474–49 491, 2019.

[92] A. Garofalo, Y. Tortorella, M. Perotti, L. Valente, A. Nadalini, L. Benini, D. Rossi, and F. Conti, "DARKSIDE: A Heterogeneous RISC-V Compute Cluster for Extreme-Edge On-Chip DNN Inference and Training," *IEEE Open Journal of the Solid-State Circuits Society*, vol. 2, pp. 231–243, 2022.

[93] N. Verma, A. Shoeb, J. Bohorquez, J. Dawson, J. Guttag, and A. P. Chandrakasan, "A Micro-Power EEG Acquisition SoC with Integrated Feature Extraction Processor for a Chronic Seizure Detection System," *IEEE journal of solid-state circuits*, vol. 45, no. 4, pp. 804–816, 2010.

[94] M. Cuturi, "Sinkhorn Distances: Lightspeed Computation of Optimal Transport," *Advances in neural information processing systems*, vol. 26, 2013.

[95] *Neuralign - All Things Neural Distribution Alignment*, https://nerdslab.github.io/neuralign/, Accessed: 2025-04-14.

[96] P. Kline, *An Easy Guide to Factor Analysis*. Routledge, 2014.

[97] D. Sussillo, S. D. Stavisky, J. C. Kao, S. I. Ryu, and K. V. Shenoy, "Making Brain–Machine Interfaces Robust to Future Neural Variability," *Nature communications*, vol. 7, no. 1, p. 13 749, 2016.

[98] F. Liu, S. Meamardoost, R. Gunawan, T. Komiyama, C. Mewes, Y. Zhang, E. Hwang, and L. Wang, "Deep Learning for Neural Decoding in Motor Cortex," *Journal of Neural Engineering*, vol. 19, no. 5, p. 056 021, 2022.

[99] A. S. Benjamin, H. L. Fernandes, T. Tomlinson, P. Ramkumar, C. VerSteeg, R. H. Chowdhury, L. E. Miller, and K. P. Kording, "Modern Machine Learning as a Benchmark for Fitting Neural Responses," *Frontiers in computational neuroscience*, vol. 12, p. 317 343, 2018.

[100] X. Zhang, Z. Song, and Y. Wang, "Reinforcement Learning-Based Kalman Filter for Adaptive Brain Control in Brain-Machine Interface," in *2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, IEEE, 2021, pp. 6619–6622.

[101] M. Asgharpour, R. Foodeh, and M. R. Daliri, "Regularized Kalman Filter for Brain-Computer Interfaces Using Local Field Potential Signals," *Journal of Neuroscience Methods*, vol. 350, p. 109 022, 2021.

[102] F. R. Willett, D. R. Young, B. A. Murphy, W. D. Memberg, C. H. Blabe, C. Pandarinath, S. D. Stavisky, P. Rezaii, J. Saab, B. L. Walter, *et al.*, "Principled BCI Decoder Design and Parameter Selection Using a Feedback Control Model," *Scientific reports*, vol. 9, no. 1, p. 8881, 2019.

[103] G. Revach, N. Shlezinger, X. Ni, A. L. Escoriza, R. J. Van Sloun, and Y. C. Eldar, "KalmanNet: Neural Network Aided Kalman Filtering for Partially Known Dynamics," *IEEE Transactions on Signal Processing*, vol. 70, pp. 1532–1547, 2022.

[104] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[105] C. Urrea and R. Agramonte, "Kalman Filter: Historical Overview and Review of Its Use in Robotics 60 Years After Its Creation," *Journal of Sensors*, vol. 2021, no. 1, p. 9 674 015, 2021.

[106] P. Babu and E. Parthasarathy, "FPGA Implementation of Multi-Dimensional Kalman Filter for Object Tracking and Motion Detection," *Engineering Science and Technology, an International Journal*, vol. 33, p. 101 084, 2022.

[107] M. Ordubağ and B. Örs, "Model-Based Kalman Filter Design on an FPGA," in *2021 13th International Conference on Electrical and Electronics Engineering (ELECO)*, IEEE, 2021, pp. 157–161.

[108] A. Valade, P. Acco, P. Grabolosa, and J.-Y. Fourniols, "A Study about Kalman Filters Applied to Embedded Sensors," *Sensors*, vol. 17, no. 12, p. 2810, 2017.

[109] K. S. Babu and K. Detroja, "Inverse Free Kalman Filter Using Approximate Inverse of Diagonally Dominant Matrices," *IEEE control systems letters*, vol. 3, no. 1, pp. 120–125, 2018.

[110] Y. Liu, C.-S. Bouganis, and P. Y. Cheung, "Efficient Mapping of a Kalman Filter into an FPGA Using Taylor Expansion," in *2007 International Conference on Field Programmable Logic and Applications*, IEEE, 2007, pp. 345–350.

[111] P. T. L. Pereira, G. Paim, P. Ü. L. da Costa, E. A. C. da Costa, S. J. M. de Almeida, and S. Bampi, "Architectural Exploration for Energy-Efficient Fixed-Point Kalman Filter VLSI Design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 7, pp. 1402–1415, 2021.

[112] Z. Irwin, K. Schroeder, P. Vu, A. Bullard, D. Tat, C. Nu, A Vaskov, S. Nason, D. Thompson, J. Bentley, *et al.*, "Neural Control of Finger Movement via Intracortical Brain–Machine Interface," *Journal of neural engineering*, vol. 14, no. 6, p. 066 004, 2017.

[113] A. K. Vaskov, Z. T. Irwin, S. R. Nason, P. P. Vu, C. S. Nu, A. J. Bullard, M. Hill, N. North, P. G. Patil, and C. A. Chestek, "Cortical Decoding of Individual Finger Group Motions Using ReFIT Kalman Filter," *Frontiers in neuroscience*, vol. 12, p. 751, 2018.

[114] D. Shin, H. Watanabe, H. Kambara, A. Nambu, T. Isa, Y. Nishimura, and Y. Koike, "Prediction of Muscle Activities from Electrocorticograms in Primary Motor Cortex of Primates," *PloS one*, vol. 7, no. 10, e47992, 2012.

[115] N. J. Higham, "Gaussian Elimination," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 3, no. 3, pp. 230–238, 2011.

[116] W. Q. Malik, W. Truccolo, E. N. Brown, and L. R. Hochberg, "Efficient Decoding with Steady-State Kalman Filter in Neural Interface Systems," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 19, no. 1, pp. 25–34, 2010.

[117] A. Ben-Israel, "An Iterative Method for Computing the Generalized Inverse of an Arbitrary Matrix," *Mathematics of Computation*, pp. 452–455, 1965.

[118] J. I. Glaser, M. G. Perich, P. Ramkumar, L. E. Miller, and K. P. Kording, "Population Coding of Conditional Probability Distributions in Dorsal Premotor Cortex," *Nature communications*, vol. 9, no. 1, p. 1788, 2018.

[119] Q. Li, D. Ding, and M. Conti, "Brain-Computer Interface Applications: Security and Privacy Challenges," in *2015 IEEE conference on communications and network security (CNS)*, IEEE, 2015, pp. 663–666.

[120] W. Byun, M. Je, and J.-H. Kim, "Advances in Wearable Brain-Computer Interfaces From an Algorithm-Hardware Co-Design Perspective," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 7, pp. 3071–3077, 2022.

[121] P. Kietzmann, T. C. Schmidt, and M. Wählisch, "A Guideline on Pseudorandom Number Generation (PRNG) in the IoT," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–38, 2021.

[122] X. Zhang and K. K. Parhi, "Implementation Approaches for the Advanced Encryption Standard Algorithm," *IEEE Circuits and systems Magazine*, vol. 2, no. 4, pp. 24–46, 2002.

[123] D. Boneh and M. Franklin, "Efficient Generation of Shared RSA Keys," in *Advances in Cryptology—CRYPTO'97: 17th Annual International Cryptology Conference Santa Barbara, California, USA August 17–21, 1997 Proceedings 17*, Springer, 1997, pp. 425–439.

[124] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, "IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8182–8201, 2019.

[125] L. Bottou, "Large-Scale Machine Learning with Stochastic Gradient Descent," in *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, Springer, 2010, pp. 177–186.

[126] W. Cao, X. Wang, Z. Ming, and J. Gao, "A Review on Neural Networks with Random Weights," *Neurocomputing*, vol. 275, pp. 278–287, 2018.

[127] P. Lacharme, "The Linux Pseudorandom Number Generator Revisited," *IACR ePrint Archive*, vol. 251, 2012.

[128] Z. Gutterman, B. Pinkas, and T. Reinman, "Analysis of the Linux Random Number Generator," in *2006 IEEE Symposium on Security and Privacy (S&P'06)*, IEEE, 2006, 15–pp.

[129] N. Ferguson and B. Schneier, *Practical Cryptography*. Wiley New York, 2003, vol. 141.

[130] M. O. Ojo, S. Giordano, G. Procissi, and I. N. Seitanidis, "A Review of Low-End, Middle-End, and High-End IoT Devices," *IEEE Access*, vol. 6, pp. 70 528–70 554, 2018.

[131] M. Hillerström, I. Ullah, and P. J. Havinga, "Sensor-Based PUF: A Lightweight Random Number Generator for Resource Constrained IoT Devices," in *IFIP International Internet of Things Conference*, Springer, 2022, pp. 89–105.

[132] P. K. Sadhu and V. P. Yanambaka, "MC-PUF: A Robust Lightweight Controlled Physical Unclonable Function for Resource Constrained Environments," in *2022 IEEE computer society annual symposium on VLSI (ISVLSI)*, IEEE, 2022, pp. 452–453.

[133] B. Chatterjee, D. Das, S. Maity, and S. Sen, "RF-PUF: Enhancing IoT Security through Authentication of Wireless Nodes Using In-Situ Machine Learning," *IEEE internet of things journal*, vol. 6, no. 1, pp. 388–398, 2018.

[134] Y. Gao, S. F. Al-Sarawi, and D. Abbott, "Physical Unclonable Functions," *Nature Electronics*, vol. 3, no. 2, pp. 81–91, 2020.

[135] B. Sen, "Puf: A new era in iot security," *CSI Transactions on ICT*, vol. 8, no. 2, pp. 185–191, 2020.

[136] K. Wallace, K. Moran, E. Novak, G. Zhou, and K. Sun, "Toward Sensor-Based Random Number Generation for Mobile and IoT Devices," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1189–1201, 2016.

[137] M. P. Pawlowski, A. Jara, and M. Ogorzalek, "Harvesting Entropy for Random Number Generation for Internet of Things Constrained Devices Using On-Board Sensors," *Sensors*, vol. 15, no. 10, pp. 26 838–26 865, 2015.

[138] H. Korn and P. Faure, "Is there Chaos in the Brain? II. Experimental Evidence and Related Models," *Comptes rendus biologies*, vol. 326, no. 9, pp. 787–840, 2003.

[139] G. Chen, "Are Electroencephalogram (EEG) Signals Pseudo-Random Number Generators?" *Journal of Computational and Applied Mathematics*, vol. 268, pp. 1–4, 2014.

[140] D. Nguyen, D. Tran, W. Ma, and K. Nguyen, "EEG-Based Random Number Generators," in *International Conference on Network and System Security*, Springer, 2017, pp. 248–256.

[141] L. E. Bassham III, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, E. B. Barker, S. D. Leigh, M. Levenson, M. Vangel, D. L. Banks, *et al.*, *Sp 800-22 rev. 1a. a Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. National Institute of Standards & Technology, 2010.

[142] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, *et al.*, "Truenorth: Design and Tool Flow of a 65 mw 1 Million Neuron Programmable Neurosynaptic Chip," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[143] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A

Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[144] J. Mack, R. Purdy, K. Rockowitz, M. Inouye, E. Richter, S. Valancius, N. Kumbhare, M. S. Hassan, K. Fair, J. Mixter, *et al.*, "RANC: Reconfigurable Architecture for Neuromorphic Computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 11, pp. 2265–2278, 2020.

[145] M. Davies *et al.*, "Taking Neuromorphic Computing to the Next Level with Loihi2," *Intel Labs' Loihi*, vol. 2, pp. 1–7, 2021.

[146] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research," *IEEE signal processing magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[147] K. L. Fair, D. R. Mendat, A. G. Andreou, C. J. Rozell, J. Romberg, and D. V. Anderson, "Sparse Coding Using the Locally Competitive Algorithm on the TrueNorth Neurosynaptic System," *Frontiers in neuroscience*, vol. 13, p. 754, 2019.

[148] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman, "Architecture Support for Accelerator-Rich CMPs," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 843–849.

[149] H. Yin, J. B. Lee, X. Kong, T. Hartvigsen, and S. Xie, "Energy-Efficient Models for High-Dimensional Spike Train Classification Using Sparse Spiking Neural Networks," in *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, 2021, pp. 2017–2025.

[150] Z. Yu, "Low-Power Neuromorphic Sensor Fusion for Elderly Care," Ph.D. dissertation, University of Glasgow, 2022.

[151] M. Kang, Y. Lee, and M. Park, "Energy Efficiency of Machine Learning in Embedded Systems Using Neuromorphic Hardware," *Electronics*, vol. 9, no. 7, p. 1069, 2020.

[152] AMD, *PYNQ*™, http://www.pynq.io/, Accessed: 2025-04-14.

[153] E. H. Pollmann, H. Yin, I. Uguz, A. Dubey, K. E. Wingel, J. S. Choi, S. Moazeni, Y. Gilhotra, V. Andino-Pavlovsky, A. Banees, *et al.*, "A Subdural CMOS Optical Device for Bidirectional Neural Interfacing," *Nature Electronics*, vol. 7, no. 9, pp. 829–841, 2024.

[154] E. Musk *et al.*, "An Integrated Brain-Machine Interface Platform with Thousands of Channels," *Journal of medical Internet research*, vol. 21, no. 10, e16194, 2019.

[155] S. Gupta, W. T. Navaraj, L. Lorenzelli, and R. Dahiya, "Ultra-Thin Chips for High-Performance Flexible Electronics," *npj Flexible Electronics*, vol. 2, no. 1, p. 8, 2018.

[156] M. Śliwowski, M. Martin, A. Souloumiac, P. Blanchart, and T. Aksenova, "Impact of Dataset Size and Long-Term ECoG-based BCI Usage on Deep Learning Decoders Performance," *Frontiers in Human Neuroscience*, vol. 17, p. 1 111 645, 2023.

[157] L. Goldstein, M. Dewhirst, M Repacholi, and L Kheifets, "Summary, Conclusions and Recommendations: Adverse Temperature Levels in the Human Body," *International Journal of Hyperthermia*, vol. 19, no. 3, pp. 373–384, 2003.

[158] C. Santos, P. Vivet, J.-P. Colonna, P. Coudrain, and R. Reis, "Thermal Performance of 3D ICs: Analysis and Alternatives," in *2014 International 3D Systems Integration Conference (3DIC)*, IEEE, 2014, pp. 1–7.

[159] Y. Shen, C. Yang, Y. Zhang, W. Wang, Y. Luo, C. Yu, K. Xu, G. Pan, and B. Zhao, "A Battery-Free Neural-Recording Chip Achieving 5.5 cm Fully-Implanted Depth by Galvanically-Switching Passive Body Channel Communication," *IEEE Journal of Solid-State Circuits*, 2024.

[160] R. Muller, H.-P. Le, W. Li, P. Ledochowitsch, S. Gambini, T. Bjorninen, A. Koralek, J. M. Carmena, M. M. Maharbiz, E. Alon, *et al.*, "A Minimally Invasive 64-Channel Wireless μECoG Implant," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 344–359, 2014.

[161] C. Yang, Y. Zhang, Z. Chang, Z. Li, T. Zheng, Y. Luo, S. Zhang, K. Xu, G. Pan, and B. Zhao, "A 0.4 mm 3 Battery-Less Crystal-Less Neural-Recording SoC Achieving 1.6 cm Backscattering Range with 2mm× 2mm On-Chip Antenna," in *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, IEEE, 2022, pp. 164–165.

[162] C. S. Mestais, G. Charvet, F. Sauter-Starace, M. Foerster, D. Ratel, and A. L. Benabid, "WIMAGINE: Wireless 64-Channel ECoG Recording Implant for Long Term Clinical Applications," *IEEE transactions on neural systems and rehabilitation engineering*, vol. 23, no. 1, pp. 10–21, 2014.

[163] K. Sriram, I. Karageorgos, X. Wen, J. Veselỳ, N. Lindsay, M. Wu, L. Khazan, R. P. Pothukuchi, R. Manohar, and A. Bhattacharjee, "HALO: A Hardware–Software Co-Designed Processor for Brain–Computer Interfaces," *Ieee micro*, vol. 43, no. 3, pp. 64–72, 2023.

[164] S. Simmich, A. Bahr, and R. Rieger, "Noise Efficient Integrated Amplifier Designs for Biomedical Applications," *Electronics*, vol. 10, no. 13, p. 1522, 2021.

[165] X. Chen, A. Morales-Gregorio, J. Sprenger, A. Kleinjohann, S. Sridhar, S. J. Van Albada, S. Grün, and P. R. Roelfsema, "1024-Channel Electrophysiological Recordings in Macaque V1 and V4 During Resting State," *Scientific data*, vol. 9, no. 1, p. 77, 2022.

[166] T. S. Rappaport, *Wireless Communications: Principles and Practice*. Cambridge University Press, 2024.

[167] A. Goldsmith, *Wireless Communications*. Cambridge university press, 2005.

[168] J. Rosenthal, A. Sharma, E. Kampianakis, and M. S. Reynolds, "A 25 Mbps, 12.4 pJ/b DQPSK Backscatter Data Uplink for the Neurodisc Brain–Computer Interface," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 5, pp. 858–867, 2019.

[169] X. Liu, M. M. Izad, L. Yao, and C.-H. Heng, "A 13 pJ/bit 900 MHz QPSK/16-QAM Band Shaped Transmitter based on Injection Locking and Digital PA for Biomedical Applications," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 11, pp. 2408–2421, 2014.

[170] J. Rosenthal and M. S. Reynolds, "A 1.0-Mb/s 198-pJ/bit Bluetooth Low-Energy Compatible Single Sideband Backscatter Uplink for the NeuroDisc Brain–Computer Interface," *IEEE Transactions on Microwave Theory and Techniques*, vol. 67, no. 10, pp. 4015–4022, 2019.

[171] J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. M. A. Patwary, Y. Yang, and Y. Zhou, "Deep Learning Scaling is Predictable, Empirically," *arXiv preprint arXiv:1712.00409*, 2017.

[172] D. Justus, J. Brennan, S. Bonner, and A. S. McGough, "Predicting the Computational Cost of Deep Learning Models," in *2018 IEEE international conference on big data (Big Data)*, IEEE, 2018, pp. 3873–3882.

[173] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, *et al.*, "Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2021, pp. 769–774.

[174] C. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Kiraly, P. Montino, D. Kanter, S. Ahmed, D. Pau, *et al.*, "MLPerf Tiny Benchmark," *arXiv preprint arXiv:2106.07597*, 2021.

[175] J. Haris, P. Gibson, J. Cano, N. B. Agostini, and D. Kaeli, "SECDA-TFLite: A Toolkit for Efficient Development of FPGA-based DNN Accelerators for Edge Inference," *Journal of Parallel and Distributed Computing*, vol. 173, pp. 140–151, 2023.

[176] A. Ganguly, R. Muralidhar, and V. Singh, "Towards Energy Efficient Non-Von Neumann Architectures for Deep Learning," in *20th international symposium on quality electronic design (ISQED)*, IEEE, 2019, pp. 335–342.

[177] H. Chhajed, G. Raut, N. Dhakad, S. Vishwakarma, and S. K. Vishvakarma, "Bitmac: Bit-Serial Computation-based Efficient Multiply-Accumulate Unit for DNN Accelerator," *Circuits, Systems, and Signal Processing*, pp. 1–16, 2022.

[178] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2018, pp. 764–775.

[179] E. Cai, D.-C. Juan, D. Stamoulis, and D. Marculescu, "NeuralPower: Predict and Deploy Energy-Efficient Convolutional Neural Networks," in *Asian Conference on Machine Learning*, PMLR, 2017, pp. 622–637.

[180] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and< 0.5 MB Model Size," *arXiv preprint arXiv:1602.07360*, 2016.

[181] C. Alippi, S. Disabato, and M. Roveri, "Moving Convolutional Neural Networks to Embedded Systems: The Alexnet and VGG-16 Case," in *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, IEEE, 2018, pp. 212–223.

[182] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. Van Esesn, A. A. S. Awwal, and V. K. Asari, "The History Began from Alexnet: A Comprehensive Survey on Deep Learning Approaches," *arXiv preprint arXiv:1803.01164*, 2018.

[183] R. Sharma, M. Kim, and A. Gupta, "Motor Imagery Classification in Brain-Machine Interface with Machine Learning Algorithms: Classical Approach to Multi-Layer Perceptron Model," *Biomedical Signal Processing and Control*, vol. 71, p. 103 101, 2022.

[184] N. Pusarla, A. Singh, and S. Tripathi, "Learning DenseNet Features from EEG based Spectrograms for Subject Independent Emotion Recognition," *Biomedical signal processing and control*, vol. 74, p. 103 485, 2022.

[185] M. S. Lewicki, "A Review of Methods for Spike Sorting: The Detection and Classification of Neural Action Potentials," *Network: Computation in Neural Systems*, vol. 9, no. 4, R53, 1998.

[186] A. M. Abdelhadi, E. Sha, C. Bannon, H. Steenland, and A. Moshovos, "Noema: Hardware-Efficient Template Matching for Neural Population Pattern Detection," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 522–534.

[187] S. Borkar, "Design Challenges of Technology Scaling," *IEEE micro*, vol. 19, no. 4, pp. 23–29, 1999.

[188] R. H. Dennard, J. Cai, and A. Kumar, "A Perspective on Today's Scaling Challenges and Possible Future Directions," in *Handbook of Thin Film Deposition*, Elsevier, 2018, pp. 3–18.

[189] P. R. Kinget, "Scaling Analog Circuits into Deep Nanoscale CMOS: Obstacles and Ways to Overcome Them," in *2015 IEEE Custom Integrated Circuits Conference (CICC)*, IEEE, 2015, pp. 1–8.

[190] J. Putzeys, B. C. Raducanu, A. Carton, J. De Ceulaer, B. Karsh, J. H. Siegle, N. Van Helleputte, T. D. Harris, B. Dutta, S. Musa, *et al.*, "Neuropixels Data-Acquisition System: A Scalable Platform for Parallel Recording of 10000+ Electrophysiological Signals," *IEEE transactions on biomedical circuits and systems*, vol. 13, no. 6, pp. 1635–1644, 2019.

[191] J. Saad, A. Evans, I. M. Panades, T. Aksenova, and L. Anghel, "Towards Low-Power Embedded ECoG Decoding," in *GDR SoC2*, 2023.

[192] R. J. Kosinski, "A Literature Review on Reaction Time," *Clemson University*, vol. 10, no. 1, pp. 337–344, 2008.

[193] F. Zaruba and L. Benini, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-bit RISC-V Core in 22-nm FDSOI Technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, 2019.

[194] *CVA6 RISC-V CPU*, https://github.com/openhwgroup/cva6, Accessed: 2025-04-14.

[195] T. Zhang, M. R. Azghadi, C. Lammie, A. Amirsoleimani, and R. Genov, "Spike Sorting Algorithms and their Efficient Hardware Implementation: A Comprehensive Survey," *Journal of Neural Engineering*, vol. 20, no. 2, p. 021 001, 2023.

[196] P. Mantovani, D. Giri, G. Di Guglielmo, L. Piccolboni, J. Zuckerman, E. G. Cota, M. Petracca, C. Pilato, and L. P. Carloni, "Agile SoC Development with Open ESP," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[197] *ESP - Open SoC Platform*, https://esp.cs.columbia.edu/, Accessed: 2025-04-14.

[198] L. P. Carloni, "The Case for Embedded Scalable Platforms," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[199] E. G. Cota, P. Mantovani, G. Di Guglielmo, and L. P. Carloni, "An Analysis of Accelerator Coupling in Heterogeneous Architectures," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1–6.

[200] D. Giri, K.-L. Chiu, G. Di Guglielmo, P. Mantovani, and L. P. Carloni, "ESP4ML: Platform-Based Design of Systems-on-Chip for Embedded Machine Learning," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2020, pp. 1049–1054.

[201] C. Pilato, P. Mantovani, G. Di Guglielmo, and L. P. Carloni, "System-Level Optimization of Accelerator Local Memory for Heterogeneous Systems-on-Chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 3, pp. 435–448, 2016.

[202] L. Piccolboni, P. Mantovani, G. D. Guglielmo, and L. P. Carloni, "COSMOS: Coordination of High-Level Synthesis and Memory Optimization for Hardware Accelerators," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, pp. 1–22, 2017.

[203] *Eigen API*, http://eigen.tuxfamily.org/dox/index.html, Accessed: 2025-04-14.

[204] K. Wang, M. Xu, Y. Wang, S. Zhang, L. Chen, and D. Ming, "Enhance Decoding of Pre-Movement EEG Patterns for Brain-Computer Interfaces," *Journal of neural engineering*, vol. 17, no. 1, p. 016 033, 2020.

[205] R. Mohanty, G. Anirudh, T. Pradhan, B. Kabi, and A. Routray, "Design and Performance Analysis of Fixed-Point Jacobi SVD Algorithm on Reconfigurable System," *IERI Procedia*, vol. 7, pp. 21–27, 2014.

[206] M Aravind and S. S. Babu, "Embedded Implementation of Brain Computer Interface Using FPGA," in *2016 International Conference on Emerging Technological Trends (ICETT)*, IEEE, 2016, pp. 1–5.

[207] R. R. Shrivastwa, V. Pudi, and A. Chattopadhyay, "An FPGA-Based Brain Computer Interfacing Using Compressive Sensing and Machine Learning," in *2018 IEEE computer society annual symposium on VLSI (ISVLSI)*, IEEE, 2018, pp. 726–731.

[208] M.-N. Wahalla, G. P. Vaya, and H. Blume, "CereBridge: An Efficient, FPGA-based Real-Time Processing Platform for True Mobile Brain-Computer Interfaces," in *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, IEEE, 2020, pp. 4046–4050.

[209] R. P. Rao, "Towards Neural Co-Processors for the Brain: Combining Decoding and Encoding in Brain-Computer Interfaces," *Current opinion in neurobiology*, vol. 55, pp. 142–151, 2019.

[210] C. Yang, H. Wu, Z. Li, W. He, N. Wang, and C.-Y. Su, "Mind Control of a Robotic Arm with Visual Fusion Technology," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 9, pp. 3822–3830, 2017.

[211] K. Xia, W. Duch, Y. Sun, K. Xu, W. Fang, H. Luo, Y. Zhang, D. Sang, X. Xu, F.-Y. Wang, *et al.*, "Privacy-Preserving Brain–Computer Interfaces: A Systematic Review," *IEEE Transactions on Computational Social Systems*, 2022.

[212] L. S. Kumari and A. Z. Kouzani, "TinyLFP: A Tiny Local-Field-Potential Sensor," *IEEE Transactions on Medical Robotics and Bionics*, vol. 4, no. 1, pp. 266–273, 2022.

[213] P. Mantovani, G. Di Guglielmo, and L. P. Carloni, "High-Level Synthesis of Accelerators in Embedded Scalable Platforms," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2016, pp. 204–211.

[214] L. Piccolboni, G. Di Guglielmo, S. Sethumadhavan, and L. P. Carloni, "Hardroid: Transparent Integration of Crypto Accelerators in Android," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, IEEE, 2021, pp. 1–8.

[215] S Ishmukhametov and B Mubarakov, "On Practical Aspects of the Miller-Rabin Primality Test," *Lobachevskii Journal of Mathematics*, vol. 34, pp. 304–312, 2013.

[216] B. Seyoum, D. Giri, K.-L. Chiu, B. Natter, and L. Carloni, "PR-ESP: An Open-Source Platform for Design and Programming of Partially Reconfigurable SoCs," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2023, pp. 1–6.

[217] J. Zuckerman, D. Giri, J. Kwon, P. Mantovani, and L. P. Carloni, "Cohmeleon: Learning-Based Orchestration of Accelerator Coherence in Heterogeneous SoCs," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 350–365.

[218] A. Nafkha and Y. Louet, "Accurate Measurement of Power Consumption Overhead During FPGA Dynamic Partial Reconfiguration," in *2016 International Symposium on Wireless Communication Systems (ISWCS)*, IEEE, 2016, pp. 586–591.

[219] H. Taleb, A. Nasser, G. Andrieux, N. Charara, and E. Motta Cruz, "Wireless Technologies, Medical Applications and Future Challenges in WBAN: A Survey," *Wireless Networks*, vol. 27, pp. 5271–5295, 2021.

[220] B. Jarosiewicz, N. Y. Masse, D. Bacher, S. S. Cash, E. Eskandar, G. Friehs, J. P. Donoghue, and L. R. Hochberg, "Advantages of Closed-Loop Calibration in Intracortical Brain–Computer Interfaces for People with Tetraplegia," *Journal of neural engineering*, vol. 10, no. 4, p. 046 012, 2013.

[221] X. Gao, H. Luo, B. Ning, F. Zhao, L. Bao, Y. Gong, Y. Xiao, and J. Jiang, "RL-AKF: An Adaptive Kalman Filter Navigation Algorithm Based on Reinforcement Learning for Ground Vehicles," *Remote Sensing*, vol. 12, no. 11, p. 1704, 2020.

[222] A. Du, S. Yang, W. Liu, and H. Huang, "Decoding ECoG Signal with Deep Learning Model based on LSTM," in *TENCON 2018-2018 IEEE Region 10 Conference*, IEEE, 2018, pp. 0430–0435.

[223] M. Agrawal, S. Vidyashankar, and K. Huang, "On-Chip Implementation of ECoG Signal Data Decoding in Brain-Computer Interface," in *2016 IEEE 21st International Mixed-Signal Testing Workshop (IMSTW)*, IEEE, 2016, pp. 1–6.

[224] D. H. Lee and C. K. Chung, "Enhancing Neural Decoding with Large Language Models: A GPT-Based Approach," in *2024 12th International Winter Conference on Brain-Computer Interface (BCI)*, IEEE, 2024, pp. 1–4.

[225] Y. Yang, Y. Duan, H. Jo, Q. Zhang, R. Xu, O. P. Jones, X. Hu, C.-t. Lin, and H. Xiong, "NeuGPT: Unified Multi-Modal Neural GPT," *arXiv preprint arXiv:2410.20916*, 2024.

[226] OpenAI, *AI and Compute*, https://openai.com/index/ai-and-compute/, Accessed: 2025-01-24.

[227] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The Computational Limits of Deep Learning," *arXiv preprint arXiv:2007.05558*, vol. 10, 2020.

[228] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Compute and Energy Consumption Trends in Deep Learning Inference," *arXiv preprint arXiv:2109.05472*, 2021.

[229] A. Vallentin, K. R. Thórisson, and H. Latapie, "Addressing the Unsustainability of Deep Neural Networks with Next-Gen AI," in *International Conference on Artificial General Intelligence*, Springer, 2023, pp. 296–306.

[230] L. Wei, Z. Ma, C. Yang, and Q. Yao, "Advances in the Neural Network Quantization: A Comprehensive Review," *Applied Sciences*, vol. 14, no. 17, p. 7445, 2024.

[231] Amazon Web Services, *Generative AI Solutions*, https://aws.amazon.com/ai/generative-ai/, Accessed: 2025-01-24.

[232] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid Method for Minimizing Service Delay in Edge Cloud Computing through VM Migration and Transmission Power Control," *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 810–819, 2016.

[233] S. Shukla, M. F. Hassan, D. C. Tran, R. Akbar, I. V. Paputungan, and M. K. Khan, "Improving Latency in Internet-of-Things and Cloud Computing for Real-Time Data ransmission: A Systematic Literature Review (SLR)," *Cluster Computing*, pp. 1–24, 2023.

[234] S. L. Bernal, A. H. Celdrán, G. M. Pérez, M. T. Barros, and S. Balasubramaniam, "Security in Brain-Computer Interfaces: State-of-the-Art, Opportunities, and Future Challenges," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–35, 2021.

[235] P. Chaudhary and R. Agrawal, "Emerging Threats to Security and Privacy in Brain Computer Interface," *International Journal of Advanced Studies of Scientific Research*, vol. 3, no. 12, 2018.

[236] A. Hastings and S. Sethumadhavan, "WaC: A New Doctrine for Hardware Security," in *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security*, 2020, pp. 127–136.

[237] M. Barbosa and P. Farshim, "Delegatable Homomorphic Encryption with Applications to Secure Outsourcing of Computation," in *Topics in Cryptology–CT-RSA 2012: The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27–March 2, 2012. Proceedings*, Springer, 2012, pp. 296–312.

[238] D. Natarajan and W. Dai, "Seal-embedded: A Homomorphic Encryption Library for the Internet of Things," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 756–779, 2021.

[239] X. Salleras and V. Daza, "ZPiE: Zero-Knowledge Proofs in Embedded Systems," *Mathematics*, vol. 9, no. 20, p. 2569, 2021.

[240] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-Propagating Errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[241] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[242] W. Maass, "Networks of Spiking Neurons: The Third Generation of Neural Network Models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[243] S. Kim, S. Park, B. Na, and S. Yoon, "Spiking-YOLO: Spiking Neural Network for Energy-Efficient Object Detection," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, 2020, pp. 11 270–11 277.

[244] W. J. Szczerek and A. Podobas, "A Quarter of a Century of Neuromorphic Architectures on FPGAs–an Overview," *arXiv preprint arXiv:2502.20415*, 2025.

[245] P. L. Gentili, M. P. Zurlo, and P. Stano, "Neuromorphic Engineering in Wetware: The State of the Art and its Perspectives," *Frontiers in Neuroscience*, vol. 18, p. 1 443 121, 2024.

[246] L. Smirnova and T. Hartung, "The Promise and Potential of Brain Organoids," *Advanced Healthcare Materials*, p. 2 302 745, 2024.

[247] F. D. Jordan, M. Kutter, J.-M. Comby, F. Brozzi, and E. Kurtys, "Open and remotely accessible neuroplatform for research in wetware computing," *Frontiers in Artificial Intelligence*, vol. 7, p. 1 376 042, 2024.

[248] N. Hagiwara, T. Asai, K. Ando, and M. Akai-Kasaya, "Fabrication and Training of 3D Conductive Polymer Networks for Neuromorphic Wetware," *Advanced Functional Materials*, vol. 33, no. 42, p. 2 300 903, 2023.

[249] L. Martis, G. Leone, L. Raffo, and P. Meloni, "Low-Power FPGA-based Spiking Neural Networks for Real-Time Decoding of Intracortical Neural Activity," *IEEE Sensors Journal*, 2024.

[250] S. Shaikh, R. So, T. Sibindi, C. Libedinsky, and A. Basu, "Towards Intelligent Intracortical BMI (i^2 BMI): Low-Power Neuromorphic Decoders that Outperform Kalman Filters," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 6, pp. 1615–1624, 2019.

[251] J. Liao, L. Widmer, X. Wang, A. Di Mauro, S. R. Nason-Tomaszewski, C. A. Chestek, L. Benini, and T. Jang, "An Energy-Efficient Spiking Neural Network for Finger Velocity Decoding for Implantable Brain-Machine Interface," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, IEEE, 2022, pp. 134–137.

[252] Y. Zhang and S. M. Chase, "A Stabilized Dual Kalman Filter for Adaptive Tracking of Brain-Computer Interface Decoding Parameters," in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, IEEE, 2013, pp. 7100–7103.

[253] A. Moly, T. Costecalde, F. Martel, M. Martin, C. Larzabal, S. Karakas, A. Verney, G. Charvet, S. Chabardes, A. L. Benabid, *et al.*, "An Adaptive Closed-Loop ECoG decoder for Long-Term and Stable Bimanual Control of an Exoskeleton by a Tetraplegic," *Journal of Neural Engineering*, vol. 19, no. 2, p. 026 021, 2022.

[254] V. Pan and J. Reif, "Efficient Parallel Solution of Linear Systems," in *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, 1985, pp. 143–152.

[255] J. Dongarra, M. Faverge, H. Ltaief, and P. Luszczek, "High Performance Matrix Inversion Based on LU Factorization for Multicore Architectures," in *Proceedings of the 2011 ACM international workshop on Many task computing on grids and supercomputers*, 2011, pp. 33–42.

[256] A. Krishnamoorthy and D. Menon, "Matrix Inversion Using Cholesky Decomposition," in *2013 signal processing: Algorithms, architectures, arrangements, and applications (SPA)*, IEEE, 2013, pp. 70–72.

[257] A. Irturk, S. Mirzaei, and R. Kastner, "An Efficient FPGA Implementation of Scalable Matrix Inversion Core Using QR Decomposition," *eScholarship: Open Access Publications from the University of California*, 2009.

[258] V. V. Vazirani, *Approximation Algorithms*, 2001.

[259] G. Schulz, "Iterative Berechung der Reziproken Matrix," *ZAMM-Journal of Applied Mathematics and Mechanics*, 1933.

[260] F. Toutounian and F. Soleymani, "An Iterative Method for Computing the Approximate Inverse of a Square Matrix and the Moore–Penrose Inverse of a Non-Square Matrix," *Applied Mathematics and Computation*, vol. 224, pp. 671–680, 2013.

[261] K. Mizuseki, A Sirota, E Pastalkova, and G Buzsáki, "Multi-Unit Recordings from the Rat Hippocampus Made During Open Field Foraging," *CRCNS. org*, 2009.

[262] D. Giri, K.-L. Chiu, G. Eichler, P. Mantovani, and L. P. Carloni, "Accelerator Integration for Open-Source SoC Design," *IEEE Micro*, vol. 41, no. 4, pp. 8–14, 2021.

[263] D. Pritsker, "Hybrid Implementation of Extended Kalman Filter on an FPGA," in *2015 IEEE Radar Conference (RadarCon)*, IEEE, 2015, pp. 0077–0082.

[264] B. Xu, L. Bai, K. Chen, and L. Tian, "A Resource Saving FPGA Implementation Approach to Fractional Kalman Filter," *IET Control Theory & Applications*, vol. 16, no. 13, pp. 1352–1363, 2022.

[265] F. Sandhu, H. Selamat, S. Alavi, and V. B. S. Mahalleh, "FPGA-based Implementation of Kalman Filter for Real-Time Estimation of Tire Velocity and Acceleration," *IEEE sensors journal*, vol. 17, no. 17, pp. 5749–5758, 2017.

[266] J. Soh and X. Wu, "An FPGA-Based Unscented Kalman Filter for System-on-Chip Applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 4, pp. 447–451, 2016.

[267] R. Patton, C. Schuman, S. Kulkarni, M. Parsa, J. P. Mitchell, N. Q. Haas, C. Stahl, S. Paulissen, P. Date, T. Potok, *et al.*, "Neuromorphic Computing for Autonomous Racing," in *International conference on neuromorphic systems 2021*, 2021, pp. 1–5.

[268] G. Tang and K. P. Michmizos, "Gridbot: An Autonomous Robot Controlled by a Spiking Neural Network Mimicking the Brain's Navigational System," in *Proceedings of the International Conference on Neuromorphic Systems*, 2018, pp. 1–8.

[269] E. Nichols, L. J. McDaid, and N. Siddique, "Biologically Inspired SNN for Robot Control," *IEEE transactions on cybernetics*, vol. 43, no. 1, pp. 115–128, 2012.

[270] M. H. Baslow, "The Languages of Neurons: An Analysis of Coding Mechanisms by which Neurons Communicate, Learn and Store Information," *Entropy*, vol. 11, no. 4, pp. 782–797, 2009.

[271] W Gerstner, "Spiking Neurons," *Pulsed Neural Networks*, pp. 3–53, 1998.

[272] S. Li, Z. Zhang, R. Mao, J. Xiao, L. Chang, and J. Zhou, "A Fast and Energy-Efficient SNN Processor with Adaptive Clock/Event-Driven Computation Scheme and Online Learning,"

IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 68, no. 4, pp. 1543–1552, 2021.

[273] D. Ivanov, A. Chezhegov, and D. Larionov, "Neuromorphic Artificial Intelligence Systems," *Frontiers in Neuroscience*, vol. 16, p. 959 626, 2022.

[274] B. Sengupta and M. B. Stemmler, "Power Consumption During Neuronal Computation," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 738–750, 2014.

[275] Y. Chen, B. Zheng, Z. Zhang, Q. Wang, C. Shen, and Q. Zhang, "Deep Learning on Mobile and Embedded Devices: State-of-the-Art, Challenges, and Future Directions," *ACM Computing Surveys (CSUR)*, vol. 53, no. 4, pp. 1–37, 2020.

[276] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, and S. R. Risbud, "Advancing Neuromorphic Computing with Loihi: A Survey of Results and Outlook," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021.

[277] BrainChip. "Akida." (2023).

[278] A. Vanarse, A. Osseiran, A. Rassau, and P. van der Made, "A Hardware-Deployable Neuromorphic Solution for Encoding and Classification of Electronic Nose Data," *Sensors*, vol. 19, no. 22, p. 4831, 2019.

[279] M. Arsalan, A. Santra, and V. Issakov, "RadarSNN: A Resource Efficient Gesture Sensing System based on mm-Wave Radar," *IEEE Transactions on Microwave Theory and Techniques*, vol. 70, no. 4, pp. 2451–2461, 2022.

[280] P. Chandarana, J. Ou, and R. Zand, "An Adaptive Sampling and Edge Detection Approach for Encoding Static Images for Spiking Neural Networks," in *2021 12th International Green and Sustainable Computing Conference (IGSC)*, IEEE, 2021, pp. 1–8.

[281] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms, Second Edition," in MIT Press and McGraw-Hill, 2001, Section 22.5.

[282] S. Martello, *Knapsack problems: Algorithms and Computer Implementations*, 1990.

[283] D. S. Johnson, "Fast Algorithms for Bin Packing," *Journal of Computer and System Sciences*, vol. 8, no. 3, pp. 272–314, 1974.

[284] T. Achterberg, "SCIP: Solving Constraint Integer Programs," *Mathematical Programming Computation*, vol. 1, pp. 1–41, 2009.

[285] K.-L. Chiu, G. Eichler, B. Seyoum, and L. Carloni, "EigenEdge: Real-Time Software Execution at the Edge with RISC-V and Hardware Accelerators," in *2023 Cyber-Physical Systems and Internet of Things Week 2023*, 2023, pp. 209–214.

[286] A. Balaji, A. Das, Y. Wu, K. Huynh, F. G. Dell'Anna, G. Indiveri, J. L. Krichmar, N. D. Dutt, S. Schaafsma, and F. Catthoor, "Mapping Spiking Neural Networks to Neuromorphic Hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 76–86, 2019.

[287] T. Titirsha, S. Song, A. Balaji, and A. Das, "On the Role of System Software in Energy Management of Neuromorphic Computing," in *Proceedings of the 18th ACM International Conference on Computing Frontiers*, 2021, pp. 124–132.

[288] S. Song, M. L. Varshika, A. Das, and N. Kandasamy, "A Design Flow for Mapping Spiking Neural Networks to Many-Core Neuromorphic Hardware," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, IEEE, 2021, pp. 1–9.

[289] Intel, *Loihi Deep Dive*, https://niceworkshop.org/wp-content/uploads/2019/04/NICE-2019-Day-4c_Loihi-Overview.pdf, Accessed: 2025-04-14.

[290] Intel, *Loihi*, https://en.wikichip.org/wiki/intel/loihi, Accessed: 2025-04-14.

[291] J. P. Mitchell, C. D. Schuman, R. M. Patton, and T. E. Potok, "Caspian: A Neuromorphic Development Platform," in *Proceedings of the 2020 Annual Neuro-Inspired Computational Elements Workshop*, 2020, pp. 1–6.

[292] J. P. Mitchell, C. D. Schuman, and T. E. Potok, "A Small, Low Cost Event-Driven Architecture for Spiking Neural Networks on FPGAs," in *International Conference on Neuromorphic Systems 2020*, 2020, pp. 1–4.

[293] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A Scalable Multicore Architecture with Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)," *IEEE transactions on biomedical circuits and systems*, vol. 12, no. 1, pp. 106–122, 2017.

[294] SynSense. "DYNAP-CNN." (2023).

[295] J. Wang, T. Wang, H. Liu, K. Wang, K. Moses, Z. Feng, P. Li, and W. Huang, "Flexible Electrodes for Brain–Computer Interface System," *Advanced Materials*, vol. 35, no. 47, p. 2 211 012, 2023.

[296] D. Yang, G. Tian, J. Chen, Y. Liu, E. Fatima, J. Qiu, N. A. N. N. Malek, and D. Qi, "Neural Electrodes for Brain-Computer Interface System: From Rigid to Soft," *BMEMat*, e12130, 2025.

[297] J. J. Levett, L. M. Elkaim, F. Niazi, M. H. Weber, C. Iorio-Morin, M. Bonizzato, and A. G. Weil, "Invasive Brain Computer Interface for Motor Restoration in Spinal Cord Injury: A

Systematic Review," *Neuromodulation: Technology at the Neural Interface*, vol. 27, no. 4, pp. 597–603, 2024.

[298] S. Kim, H. Genc, V. V. Nikiforov, K. Asanović, B. Nikolić, and Y. S. Shao, "MoCA: Memory-Centric, Adaptive Execution for Multi-Tenant Deep Neural Networks," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, 2023, pp. 828–841.

[299] K.-L. Chiu, G. Eichler, C.-T. Lin, G. Di Guglielmo, and L. P. Carloni, "WOLT: Transparent Deployment of ML Workloads on Lightweight Many-Accelerator Architectures," in *2024 IEEE 42nd International Conference on Computer Design (ICCD)*, IEEE, 2024, pp. 637–644.

[300] D. Giri, K.-L. Chiu, G. Eichler, P. Mantovani, N Chandramoorth, and L. P. Carloni, "Ariane+ NVDLA: Seamless Third-Party IP Integration with ESP," in *Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2020.

[301] F. Restuccia, B. Seyoum, A. Redding, Z. Ma, G. Eichler, L. Carloni, and R. Kastner, "OPEN-CFR: Open-source Co-design Framework for Redundancy with DPR in COTS FPGA SoCs," in *2024 IEEE Space Computing Conference (SCC)*, IEEE, 2024, pp. 65–74.

[302] R. P. Rao, "Brain Co-Processors: Using AI to Restore and Augment Brain Function," *Handbook of neuroengineering*, 2020.

[303] V. Balasubramanian, "Brain Power," *Proceedings of the National Academy of Sciences*, vol. 118, no. 32, e2107022118, 2021.

[304] W.-S. Choi, M. Tomei, J. R. S. Vicarte, P. K. Hanumolu, and R. Kumar, "Guaranteeing Local Differential Privacy on Ultra-Low-Power Systems," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2018, pp. 561–574.

[305] M. Maycock and S. Sethumadhavan, "Hardware Enforced Statistical Privacy," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 21–24, 2015.

[306] S. Naufel and E. Klein, "Brain–Computer Interface (BCI) Researcher Perspectives on Neural Data Ownership and Privacy," *Journal of Neural Engineering*, vol. 17, no. 1, p. 016 039, 2020.

[307] R. Yuste, S. Goering, B. A. y. Arcas, G. Bi, J. M. Carmena, A. Carter, J. J. Fins, P. Friesen, J. Gallant, J. E. Huggins, *et al.*, "Four Ethical Priorities for Neurotechnologies and AI," *Nature*, vol. 551, no. 7679, pp. 159–163, 2017.

[308] I. Haalman and E. Vaadia, "Dynamics of Neuronal Interactions: Relation to Behavior, Firing Rates, and Distance between Neurons," *Human brain mapping*, vol. 5, no. 4, pp. 249–253, 1997.

[309] X. Bi, D. Chen, G. Chen, S. Chen, D. Dai, C. Deng, H. Ding, K. Dong, Q. Du, Z. Fu, *et al.*, "Deepseek LLM: Scaling Open-Source Language Models with Longtermism," *arXiv preprint arXiv:2401.02954*, 2024.

[310] A. Picchi and A. Sherter, *What is DeepSeek, and why is it causing Nvidia and other stocks to slump? CBS News*, Accessed: 2025-01-24, 2025.

[311] Z. Wu, X. Chen, Z. Pan, X. Liu, W. Liu, D. Dai, H. Gao, Y. Ma, C. Wu, B. Wang, *et al.*, "DeepSeek-VL2: Mixture-of-Experts Vision-Language Models for Advanced Multimodal Understanding," *arXiv preprint arXiv:2412.10302*, 2024.