Technische Universität Dresden

Implementation of bioinspired algorithms on the neuromorphic VLSI system SpiNNaker 2

Dipl.-Ing.

Yexin Yan

der Fakultät Elektrotechnik und Informationstechnik der Technische Universität Dresden

zur Erlangung des akademischen Grades

Doktoringenieur

(Dr.-Ing.)

genehmigte Dissertation

Vorsitzender:	Prof. Dr. rer. nat. Stefan Mannsfeld	Tag der Einreichung	24.03.2022
Gutachter:	Prof. Dr.-Ing. habil. Christian Georg Mayr	Tag der Verteidigung	15.12.2022
Gutachter:	Prof. Dr. Emre Neftci		

Acknowledgements

First and foremost I would like to thank my supervisor Prof. Christian Mayr who provided me the opportunity to work on SpiNNaker 2 and guided me throughout the journey of my PhD. His expertise and insightful feedback were invaluable and pushed me to sharpen my thinking and brought my work to a higher level.

This work wouldn't have been possible without my collaborators from the chair of highly parallel VLSI systems and neuromorphic circuits at Technische Universität Dresden, the University of Manchester, Technische Universität Graz, the University of Waterloo and Applied Brain Research. In particular, I would like to thank Dr. Andrew Rowley and Andreas Dixius for explanations of the software of SpiNNaker 1 and the hardware of the first SpiNNaker 2 prototype when I was getting started, and Dr. Sebastian Höppner and Bernhard Vogginger for the collaboration on the implementation of the software demonstrating DVFS with spiking neural networks. In addition, I would like to thank Dr. David Kappel, Prof. Robert Legenstein and Prof. Wolfgang Maass for showing me the world of Bayesian reinforcement learning and for the collaboration on the implementation of reward-based synaptic sampling on the SpiNNaker 2 prototype. Also, I would like to thank Dr. Terrence Stewart, Dr. Xuan Choo and Prof. Chris Eliasmith for introducing me to control theory and for the collaboration on the implementation of keyword spotting and adaptive control and comparison with Loihi. Furthermore, I would like to thank Felix Neumärker, Dr. Johannes Partzsch and Florian Kelber for designing the random number generator, the exponential accelerator and the MAC array which my work was based on.

Last but not the least, I would like to thank my family for their unconditional support throughout the years.

Abstract

It is believed that neuromorphic hardware will accelerate neuroscience research and enable the next generation edge AI. On the other hand, brain-inspired algorithms are supposed to work efficiently on neuromorphic hardware. But both processes don't happen automatically. To efficiently bring together hardware and algorithm, optimizations are necessary based on the understanding of both sides. In this work, software frameworks and optimizations for efficient implementation of neural network-based algorithms on SpiNNaker 2 are proposed, resulting in optimized power consumption, memory footprint and computation time. In particular, first, a software framework including power management strategies is proposed to apply dynamic voltage and frequency scaling (DVFS) to the simulation of spiking neural networks, which is also the first-ever software framework running a neural network on SpiNNaker 2. The result shows the power consumption is reduced by 60.7% in the synfire chain benchmark. Second, numerical optimizations and data structure optimizations lead to an efficient implementation of reward-based synaptic sampling, which is one of the most complex plasticity algorithms ever implemented on neuromorphic hardware. The results show a reduction of computation time by a factor of 2 and energy consumption by 62%. Third, software optimizations are proposed which effectively exploit the efficiency of the multiply-accumulate array and the flexibility of the ARM core, which results in, when compared with Loihi, 3 times faster inference speed and 5 times lower energy consumption in a keyword spotting benchmark, and faster inference speed and lower energy consumption for adaptive control benchmark in high dimensional cases. The results of this work demonstrate the potential of SpiNNaker 2, explore its range of applications and also provide feedback for the design of the next generation neuromorphic hardware.

Kurzfassung

Neuromorphe Hardware trägt die Hoffnung, Neurowissenschaft voranzutreiben und die nächste Generation von Edge KI zu ermöglichen. Zudem wird erwartet, dass sich vom Gehirn inspirierte Algorithmen effizient auf neuromorpher Hardware umsetzen lassen. Aber das Zusammenspiel geschieht nicht automatisch. Um Hardware und Algorithmus effizient zusammenzubringen, sind Optimierungen notwendig, die auf dem Verständnis beider Seiten basieren. In dieser Arbeit werden Software-Frameworks und Optimierungen für die effiziente Implementierung von Algorithmen auf Basis von neuronalen Netzen auf dem neuromorphen System SpiNNaker 2 vorgeschlagen, die Leistungsaufnahme, Speicherbedarf und Rechenzeit optimieren. Insbesondere wird erstens ein Software-Framework inklusive Power Management Strategien vorgeschlagen für die Anwendung von Dynamic Voltage and Frequency Scaling für die Simulation von gepulsten neuronalen Netzen. Dies stellt das erste Software-Framework dar, womit ein neuronales Netzwerk auf SpiNNaker 2 realisiert wurde. Das Ergebnis zeigt eine Reduzierung der Leistungsaufnahme um 60.7%. Zweitens führen numerische- und Datenstruktur-Optimierungen zu einer effizienten Implementierung von Reward-basiertem Synaptic Sampling: einer der kompliziertesten Plastizitäts-Algorithmen, der jeweils auf neuromorpher Hardware implementiert wurde. Die Ergebnisse zeigen eine Reduktion der Rechenzeit um den Faktor 2 und des Energieverbrauchs um 62%. Drittens werden Software-Optimierungen vorgeschlagen, die die Effizienz vom Multiply-Accumulate Array und die Flexibilität vom ARM core von SpiNNaker 2 effektiv ausnutzen können, was im Vergleich zu Intels neuromorphen Chip Loihi zu 3-mal schnellerer Inferenzgeschwindigkeit und 5-mal niedrigerem Energieverbrauch in der Keyword-Spotting Benchmark führt. Ergebnisse dieser Arbeit zeigen das Potenzial von SpiNNaker 2, explorieren dessen Umfang von Anwendungen und stellen Feedback für den Entwurf von der nächsten Generation neuromorpher Hardware zur Verfügung.

Contents

List of Figures

LIST OF FIGURES

LIST OF FIGURES

List of Tables

Glossary

AER address-event representation

AGI Artificial General Intelligence

AI Artificial Intelligence

AMPA α-amino-3-hydroxy-5-methyl-4-isoxazolepropionic acid

BOLD Blood-Oxygen-Level-Dependent

CMOS Complementary metal-oxide-semiconductor

DMA Direct Memory Access

DNN Deep Neural Network

DRAM Dynamic Random-Access Memory

DTCM Data Tightly Coupled Memory

DVFS Dynamic Voltage and Frequency Scaling

EEG electroencephalography

EPSC excitatory postsynaptic current

EU European Union

fMRI functional Magnetic Resonance Imaging

GABA γ-aminobutyric acid

HBP Human Brain Project

IoT Internet of Things

IPSC inhibitory postsynaptic current

LIF Leaky-Integrate-and-Fire

MAC Multiply-Accumulate

MOSFET Metal–Oxide–Semiconductor Field-Effect Transistor

NEF Neural Engineering Framework

NMDA N-methyl-D-aspartate

NoC Network-on-Chip

PE Processing Element

PL Performance Level

PRNG Pseudo Random Number Generator

PSP postsynaptic potential

QPE Quad Processing Element

ReLU Rectified Linear Unit

RMSE root mean squared error

RNG Random Number Generator

SIMD Single Instruction Multiple Data

SNN Spiking Neural Network

SRAM static random access memory

STDP Spike-Timing Dependent Plasticity

TRNG True Random Number Generator

VLSI Very Large-Scale Integration

WTA winner-take-all

1

Introduction

1.1 Motivation and Background

The last decade has witnessed substantial progress in artificial intelligence (AI) [3]. Just like the steam engine, electricity and the internet, AI promises to drastically increase productivity and revolutionize the way we live and work. Numerous AI applications in areas like automobiles, robotics, healthcare and agriculture keep pushing the boundaries of tasks that machines can achieve [4].

If we take a closer look at the history of AI, we would notice its long and intertwined relation with neuroscience. While AI aims to build intelligent machines, neuroscience is concerned with the study of the brain. Although AI is not bound to the constraints of neuroscience, neuroscientific findings often inspire progress in AI. Notable examples include the concept of neural networks and temporal difference learning. Moreover, the brain is the only example we have that can demonstrate cognitive capabilities [5]. On the other hand, the progress of AI also provides inspiration for neuroscience. For example, methodologies in machine learning research can be applied for neuroscience [6], which is the case for meta-learning [7].

While scaling up models in AI and neuroscience requires more and more computational power, Moore's law is slowing down, so that improvements of hardware performance rely more and more on hardware-algorithm codesign. Both AI and neuroscience rely on microelectronics to provide the computational substrate. To efficiently execute AI algorithms, machine learning hardware often includes multiply-accumulate (MAC) arrays to facilitate the MAC operations frequently used in deep neural networks (DNN)

[8].

While the state-of-the-art AI algorithms can perform on the human level in specific areas like the game of Go, the execution of such algorithms requires combining more than one thousand GPUs and consumes power on the kilowatt (kW) level [9]. On the other hand, the human brain with about 10^{10} neurons and 10^{14} synapses [10] consumes only about 20 W of power [11]. As a product of millions of years of evolution, the human brain is arguably optimized for the generation of survival strategies and constrained on energy consumption, which makes it not only an excellent example of cognitive computing for the field of AI but also an excellent example of ultra-low power information processing system for the field of microelectronics.

The suffix '-morphic' means having a specific shape or form [12]. The field of neuromorphic engineering or neuromorphic computing started with Carver Mead, who was the first to attempt to build silicon chips to emulate the structure and function of the nervous system. As shown in Figure 1.1, in a biological neuron, the voltage across the neuron membrane is found to influence the ion flow through the ion channels of the membrane, which creates current flow. The voltage thus controls the height of the energy barrier for charge carriers. Similarly, in a complementary metal-oxide-semiconductor (CMOS) transistor, the gate voltage also controls the energy barrier for charge carriers traveling from the source to drain. Noticing the similar exponential current-voltage dependence in both cases, Carver Mead started building analog circuits which could emulate the behavior of neurons based on this physical property [13]. After three decades of evolution, neuromorphic engineering is still concerned with taking certain aspects from the brain and building them into silicon. Since the start of this area in the late 1980s with only analog VLSI, neuromorphic computing has evolved to also include mixed-signal [14, 15] and digital [16, 17, 18] hardware. Nevertheless, neuromorphic hardware normally makes use of highly parallel Very Large-Scale Integration (VLSI) circuits to emulate or simulate neural dynamics and use address-event representation (AER) [19] for spike communication, analogous to the parallel nature of neurons and the spike communication that occurs in the brain.

As neuroscience and AI continue to interact and merge, various AI-inspired neuroscience models and neuroscience-inspired AI algorithms put new requirements on the computational substrate. An increasing number of neuromorphic hardware platforms

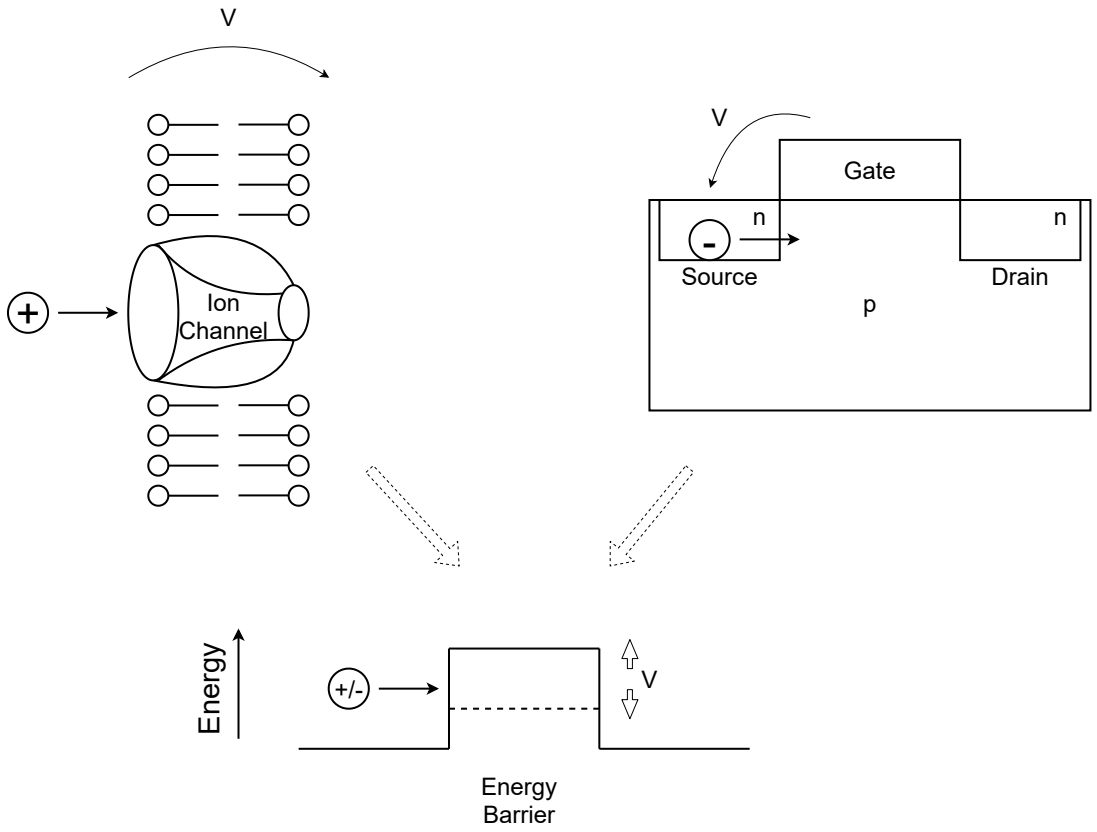

Figure 1.1: Upper left: the ion channels in the neuron membrane control the ions flowing into the neuron, which effectively forms an energy barrier for the ions. This energy barrier is found to depend on various factors including the difference of electrical potential between the extracellular and interior fluid of the neuron cell. Upper right: in a metal–oxide–semiconductor field-effect transistor (MOSFET) (in this case an N-type MOSFET), the gate voltage controls the energy barrier between the source and drain, thus controlling the charge carriers (in this case the electrons) flowing from source to drain. This similarity was first discovered by Carver Mead, who started to build circuits to mimic the biological neural network, which then developed into the field of neuromorphic engineering.

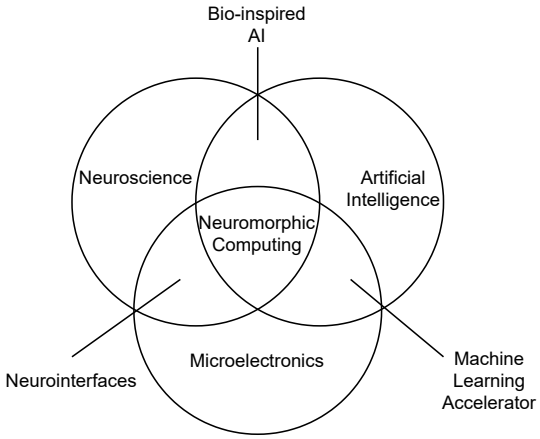

Figure 1.2: Relation between neuromorphic engineering, neuroscience, artificial intelligence and microelectronics. Neuromorphic engineering is an interdisciplinary research area at the intersection between neuroscience, AI and microelectronics.

are starting to support DNNs [2, 20], so that neuromorphic hardware becomes the computational engine for neuroscience, AI and even the mixture of both worlds. This puts neuromorphic computing at the intersection of the multidisciplinary research between neuroscience, AI and microelectronics (Figure 1.2). This dynamically evolving multidisciplinary research is paving the way towards artificial general intelligence (AGI) which will have a huge impact on society.

1.2 SpiNNaker 2 and Contribution of This Work

Among the many initiatives and organizations aiming for progress in neuroscience and AI, the Human Brain Project (HBP) [21] started by the European Union (EU) brings together experts from neuroscience, computer science, electrical engineering, etc. to better understand the brain, develop better treatment for brain diseases and build better brain-inspired computers.

The SpiNNaker 2 project started within the framework of HBP is a joint collaboration of the group of Prof. Christian Mayr at Technische Universität Dresden and the group of Prof. Steve Furber at the University of Manchester. It builds upon the first generation SpiNNaker system (SpiNNaker 1) [16], with the goal of building a supercomputer for large-scale brain simulation and machine learning [22].

In this work, following main contributions are presented:

- a software framework including power management strategies to apply dynamic voltage and frequency scaling (DVFS) to the simulation of spiking neural networks, which is also the first-ever software framework running a neural network on SpiNNaker 2. The result is demonstrated with the synfire chain benchmark and reduces the power consumption by 60.7%.

- numerical optimizations and data structure optimizations for efficient implementation of reward-based synaptic sampling, which is one of the most complex plasticity algorithms ever implemented on neuromorphic hardware. The result shows a reduction of computation time by a factor of 2 and energy consumption by 62%.

- software optimizations which effectively exploit the efficiency of the multiply-accumulate array and the flexibility of the ARM core, resulting in 3 times faster inference speed and 5 times lower energy consumption in the keyword spotting benchmark, and faster inference speed and lower energy consumption for adaptive control benchmark with higher dimensions in comparison to Loihi, the neuromorphic chip developed by Intel.

More generally, from the hardware point of view, the contribution of this work includes the first demonstration of the benefits of the new hardware features of SpiNNaker 2 in various neural network-based algorithms and benchmarks, especially in terms of energy efficiency and computation time. From the application point of view, this work explores the range of applications that SpiNNaker 2 is suitable for, including SNNs, DNNs and hybrid networks like the neural engineering framework (NEF).

The thesis is organized as follows: Chapter 2 introduces the fundamental aspects including the SpiNNaker 2 hardware, neurons and neural networks which form the basis of this thesis. Chapter 3 discusses in details the benefits of power management when applied to SNN simulation. Chapter 4 demonstrates the benefits of the numerical accelerators when simulating the reward-based synaptic sampling algorithm. Chapter 5 highlights the benefits of the MAC array in the keyword spotting and adaptive control benchmarks and shows the comparison with Loihi running the same benchmarks.

While from the hardware perspective, each chapter is focused on a certain feature of the chip which can improve the efficiency, from the application perspective, chapter 3 involves the typical SNN simulation with the Leaky-Integrate-and-Fire (LIF) neuron

model and current based synapses commonly used in SNN simulation. Chapter 4 goes one step further to the more complicated reward-based synaptic sampling model which can be used to explain the stochastic behavior of synapses in recent neurophysiological experiments. Finally, chapter 5 goes beyond SNNs and involves DNNs and hybrid networks like NEF.

1.3 Publications

Much of the work presented in this thesis is based on these previous publications:

1. S. Höppner, **Y. Yan**, B. Vogginger, A. Dixius, J. Partzsch, F. Neumärker, S. Hartmann, S. Schiefer, S. Scholze, G. Ellguth, L. Cederstroem, M. Eberlein, C. Mayr, S. Temple, L. Plana, J. Garside, S. Davison, D. R. Lester, and S. Furber. *"Dynamic voltage and frequency scaling for neuromorphic many-core systems."*, In 2017 IEEE International Symposium on Circuits and Systems (ISCAS), pages 1–4, 2017.

2. S. Höppner, B. Vogginger, **Y. Yan**, A. Dixius, S. Scholze, J. Partzsch, F. Neumärker, S. Hartmann, S. Schiefer, G. Ellguth, L. Cederstroem, L. A. Plana, J. Garside, S. Furber, and C. Mayr. *"Dynamic Power Management for Neuromorphic Many-Core Systems."*, IEEE Transactions on Circuits and Systems I: Regular Papers, 66(8):2973–2986, 2019

3. **Y. Yan**, D. Kappel, F. Neumärker, J. Partzsch, B. Vogginger, S. Höppner, S. Furber, W. Maass, R. Legenstein, and C. Mayr. *"Efficient Reward-Based Structural Plasticity on a SpiNNaker 2 Prototype."*, IEEE Transactions on Biomedical Circuits and Systems, 13(3):579–591, 2019.

4. **Y. Yan**, T. Stewart, X. Choo, B. Vogginger, J. Partzsch, S. Höppner, F. Kelber, C. Eliasmith, S. Furber, and C. Mayr. *"Comparing Loihi with a SpiNNaker 2 Prototype on Low-Latency Keyword Spotting and Adaptive Robotic Control."*, Neuromorphic Computing and Engineering, 2021

In the first two publications my contribution was the implementation and simulation of SNNs on the first and second SpiNNaker 2 prototypes developed by Sebastian Höppner and other coauthors, demonstrating the benefit of DVFS.

In the third publication, my contribution was the implementation and simulation of the synaptic sampling algorithm developed by David Kappel, Wolfgang Maass and Robert Legenstein on the first SpiNNaker 2 prototype, demonstrating the benefit of random number generator and exponential function accelerator.

In the fourth publication my contribution was the implementation and simulation of the keyword spotting and adaptive control algorithms developed by Terrence Stewart, Xuan Choo, and Chris Eliasmith on the second SpiNNaker 2 prototype, and the comparison with the performance of the same algorithms on Loihi, where Terrence Stewart and Xuan Choo contributed to the measurements on Loihi. The results of this publication highlight the benefit of the MAC array.

2

Fundamentals

In this chapter, fundamentals regarding the SpiNNaker 2 system, neuron, synapse and neural network are introduced, providing the basis and background for the following chapters of this work.

2.1 The SpiNNaker 2 System

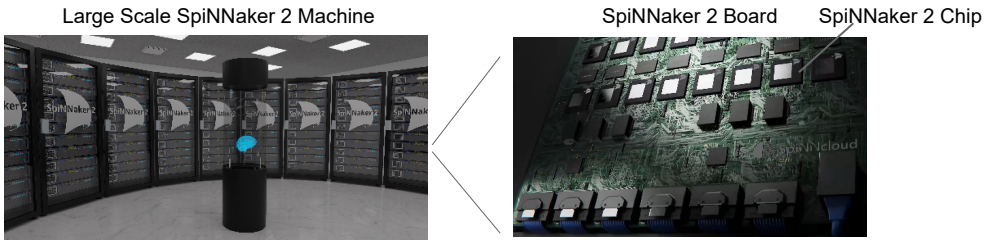

Figure 2.1: Rendering of the final SpiNNaker 2 system consisting of 16 racks. Each rack contains 90 SpiNNaker 2 boards. Each board contains 48 SpiNNaker 2 chips. Each SpiNNaker 2 chip contains 152 cores (Processing Elements). The final SpiNNaker 2 system has 16 racks, 1440 SpiNNaker 2 boards, 69,120 SpiNNaker 2 chips and 10,506,240 cores.

The SpiNNaker 2 system is designed as a supercomputer to simulate large-scale neural networks in real-time. The final machine, which is currently under development, consists of 16 racks. Each rack contains 90 SpiNNaker 2 boards. Each board contains 48 SpiNNaker 2 chips. Each SpiNNaker 2 chip contains 152 cores (Processing Elements). The final SpiNNaker 2 system has 16 racks, 1440 SpiNNaker 2 boards, 69,120 SpiNNaker 2 chips and 10,506,240 cores.(Figure 2.1).

As illustrated in Figure 2.2, the final SpiNNaker 2 chip contains 38 Quad Processing Elements (QPEs), one SpiNNaker Router, IO and periphery [1] [2]. Each QPE contains 4 Processing Elements (PEs). The QPEs are interconnected through the Network-on-Chip (NoC) routers (not shown in the figure). The final SpiNNaker 2 chip is fabricated with the GLOBALFOUNDRIES 22FDX technology [23].

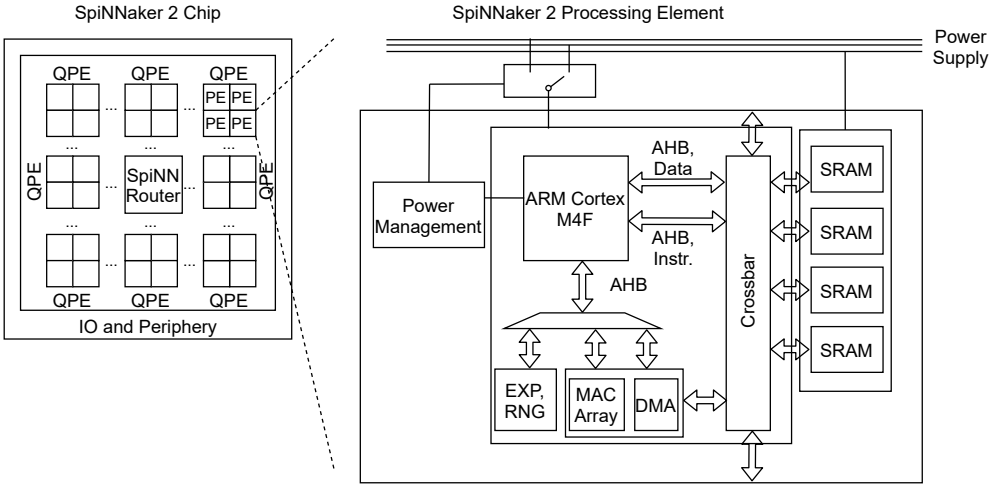

Figure 2.2: Simplified schematic of SpiNNaker 2 chip and SpiNNaker 2 Processing Element. The final SpiNNaker 2 chip contains 38 QPEs where each QPE contains 4 PEs, a SpiNNaker Router and IO and periphery. The PE contains an ARM Cortex M4F core, SRAM, power management module, numerical accelerators like the exponential function accelerator and random number generator, and MAC array.

SpiNNaker 2 consists of a large number of cores interconnected through a dedicated communication fabric. Within each PE, while the ARM core forms the main computational resource of the PE, the additional accelerators can greatly increase the computational and power efficiency for certain applications. To reduce the possibility of contention for static random access memory (SRAM), the SRAM is divided into four addressable banks.

The various computation and memory units of the PE are interconnected through

[1] At the time of writing this thesis, the final SpiNNaker 2 chip is still in development. The results presented in this work are obtained from the first and second SpiNNaker 2 prototype chips, whereas the basic working principles described in this work are the same as the final SpiNNaker 2 chip. In this thesis, when results from both the first and the second prototype chips are available, the results from the second prototype chip are presented since they should be more similar to the final SpiNNaker 2 chip.

the communication units including the AHB bus, DMA and crossbar. As shown in Figure 2.2, the ARM core is the master of three AHB buses, two for access to the memory, one for access to the accelerators and DMA. In addition to the AHB buses for data and instruction for the ARM core of the same PE, the crossbar is also connected to neighboring PEs within the same QPE, which allows low latency memory sharing between PEs in the same QPE. The crossbar is also connected to the DMA for high-speed communication with other PEs or the DRAM, and for the MAC array to read and write data independently of the ARM core.

The numerical accelerators are each connected as an individual slave to an AHB multiplexer. Each accelerator has a specific range in the memory map of the PE that it uses for reading and writing.

When compared to SpiNNaker 1, except for the more advanced technology node of 22 nm, which allows for more ARM cores on a chip, SpiNNaker 2 comes with many improvements regarding the PE architecture. In this work, three important new features of the PE architecture available in SpiNNaker 2 are demonstrated, including

- power management with the Dynamic Voltage and Frequency Scaling (DVFS) [24, 25] for increased energy efficiency in spiking neural network (SNN) simulations,
- numerical accelerators including the random number generator [26] and exponential accelerator [27, 28] for frequently used functions in neural models [29],
- MAC array [2] for the matrix operations commonly used in DNNs [30].

2.2 Neuron and Synapse

The neuron is the basic processing element in the brain. Neurons are connected to each other through synapses. As illustrated in Figure 2.3, a neuron is composed of three parts, the dendrite which receives inputs from other neurons, the soma which is believed to be the main computational part, and the axon which sends out the output to other neurons.

Neurons communicate with each other with action potentials or spikes, which is a voltage pulse generated in the soma that travels along the axon and finally arrives at other neurons through synapses. The most common type of synapse is the chemical synapse. From the perspective of the synapse, the neuron where the spike originates

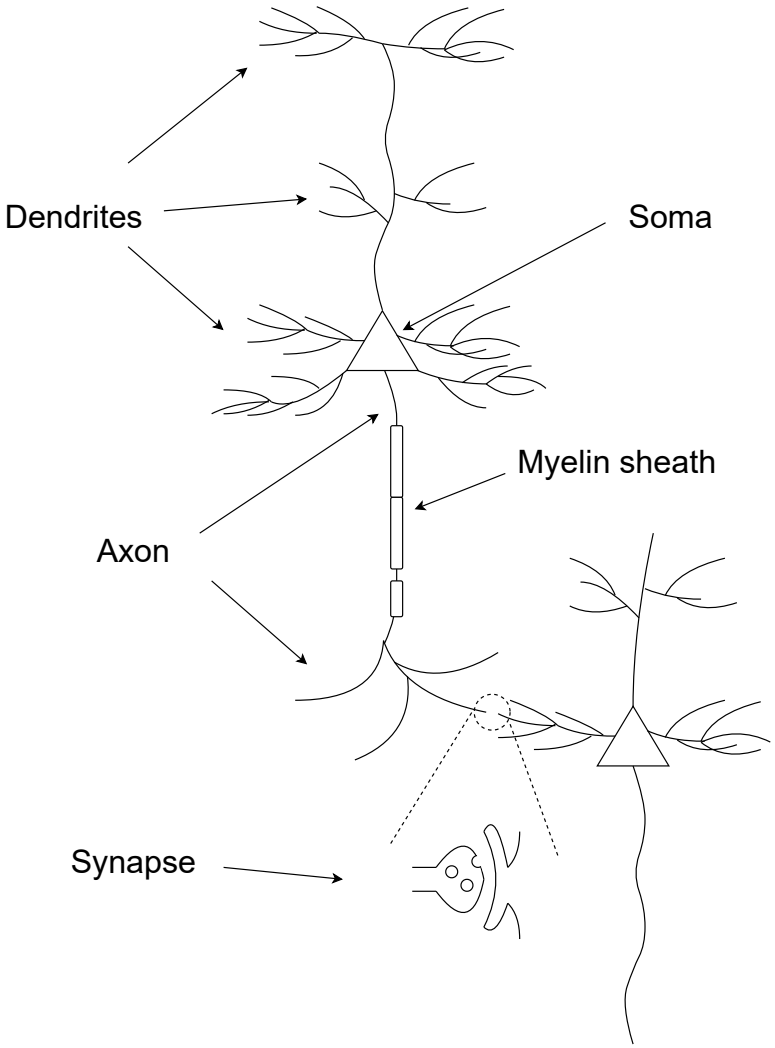

Figure 2.3: Schematic drawing of a pyramidal neuron with dendrites, soma, axon and synapse.

from is the presynaptic neuron, and the neuron receiving the spike is the postsynaptic neuron. Between the presynaptic neuron and the postsynaptic neuron is the synaptic cleft.

The neuron, like other cells, has a membrane which is a bilayer of lipids that separates the internal environment of the cell from the external environment of the cell. Two types of proteins are embedded in the neuron membrane, which are the gates that different ions can travel through. The first type of gate is the ion pump, which transports ions across the membrane actively. The result is that there are more sodium ions (Na^+) outside the neuron and more potassium ions (K^+) inside the neuron. The second type of gate is the ion channel, where ions travel through the membrane depending on the voltage across the membrane. When there is no inputs received by the dendrites and the membrane is "at rest", the ion pumps and ion channels act together to maintain the balance of the ion concentration in the neuron. This ion concentration leads to the membrane potential called resting potential [31].

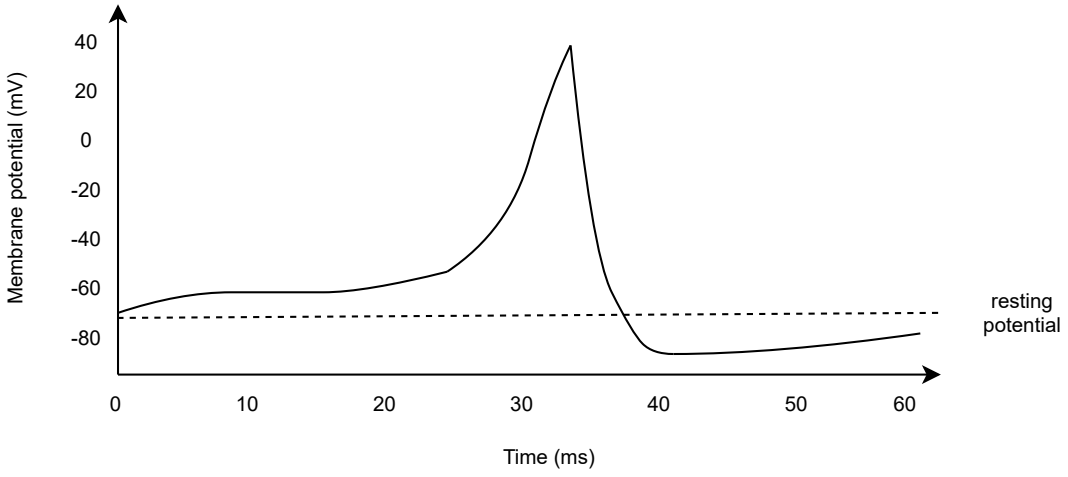

Figure 2.4: Schematic drawing of a spike.

To study the behavior of ion channels, Hodgkin and Huxley performed a series of experiments and published the Hodgkin-Huxley model in 1952 [32], which led to the Nobel Prize for Hodgkin and Huxley in 1963. The model accurately describes the behavior of ions channels depending on voltage and time. One important property of the Hodgkin-Huxley model is that it explains the internal dynamics leading to spike generation (Figure 2.4). At first, the membrane potential is at the resting potential,

where the ions flowing into and out of the cell are at equilibrium. When the input spikes received from the dendrites cause an increase of the membrane potential, the conductance of sodium ion channels increases which allows more sodium ions to flow into the cell to further increase the membrane potential, which leads to the action potential. Then the conductance of sodium channels decreases, and the conductance of potassium channels increases, which leads to potassium ions flowing out of the cell, resulting in the decrease of membrane potential. In fact, there are so many potassium ions flowing out of the cell, that the membrane potential decreases below the resting potential, and then slowly recovers back to the resting potential. Immediately after the spike, the neuron can't spike again, which is also determined by the dynamics explained by the Hodgkin-Huxley model. This period is called the refractory period. In this period, the ion pumps transport ions on both sides of the membrane so that the membrane potential recovers to the resting potential.

The classical Hodgkin-Huxley model is based on the assumption of "point neuron", where the spacial property or the morphology of the neuron membrane is not considered. But in fact, the morphology does have an impact on neural dynamics. To account for the morphology of the membrane, the membrane is discretized into patches and each patch is described by an electrical circuit (Figure 2.5), where the resistance in the direction parallel to the membrane is represented by resistors and the in the direction across the membrane, it is represented by a capacitor and a resistor. This method is called cable theory and it is the basis for the compartmental models where a single neuron is divided into hundreds of compartments and each compartment is represented by some cable equations. When a spike travels down the axon, the propagation speed of the spike can be derived from the cable theory. It turns out that the propagation speed of an unmyelinated axon, i.e. an axon that is not covered by a myelin sheath, is rather slow, which is around 0.25m/s. On the other hand, for myelinated axons, the propagation speed can reach 70 - 80 m/s because the myelin sheath changes the electrical properties of the axon. With the myelin sheath, spikes can travel for long distances very fast [31].

When a spike travels down the axon and arrives at the synapse, it causes the release of neurotransmitters into the synaptic cleft. The neurotransmitters diffuse to the other side of the cleft and bind to the receptors, which causes the opening of certain ion channels. The change in ion concentration leads to the excitatory or inhibitory

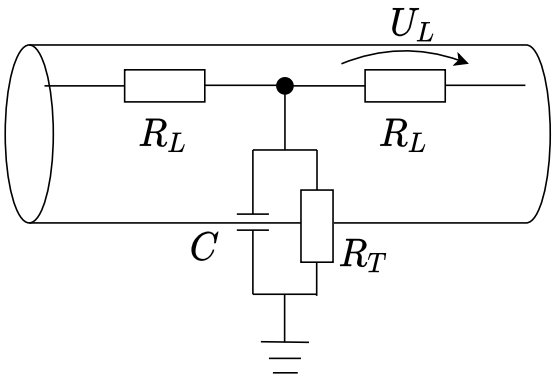

Figure 2.5: Illustration of cable theory.

postsynaptic currents (EPSC or IPSC) . The EPSC makes the neuron more likely to spike and the IPSC makes the neuron less likely to spike.

While biophysics provides the foundation for understanding the neuronal dynamics, in simulations of neural networks where thousands of neurons are involved, detailed biophysical models lead to very long simulation times. Just like in circuit design, where the transistors have complicated internal behaviors, the complicated transistor models based on the understanding of how charge carriers move depending on various physical conditions are normally not used in circuit simulation. Instead, simplified models are necessary, which are still capable of describing the major electrical properties but at a much lower computational complexity.

In neural network simulations, instead of the Hodgkin-Huxley model or the compartmental model, normally the Leaky Integrate-and-Fire (LIF) neuron model or its variations are used, where the shape of the spike, the morphology of the neuron etc. are ignored. The LIF model describes the membrane as a leaky integrator, where the EPSC or IPSC integrate on a capacitor and at the same time "leaks" through a resistor:

$$\tau_m \frac{du}{dt} = -[u(t) - u_{rest}] + RI(t) \tag{2.1}$$

where τ_m is the membrane time constant, u is the membrane potential, u_{rest} is the resting potential, R is the membrane resistance and I is the external current, i.e. EPSC or IPSC. For the LIF model, when the membrane potential u reaches a threshold voltage u_{th} from below, a spike is generated, and the membrane potential is reset to the reset

potential u_{reset} and enters the refractory period, during which no integration of EPSC or IPSC is possible.

Also, to simplify the simulation of synapses, very often the current based synapse model is used:

$$\frac{dI}{dt} = -\frac{I}{\tau_{syn}} + \delta(t - t^j)w \tag{2.2}$$

where I is the synaptic current, τ_{syn} is the synaptic time constant and $\delta(t-t^j)$ accounts for the increase of synaptic current at the spike time t^j.

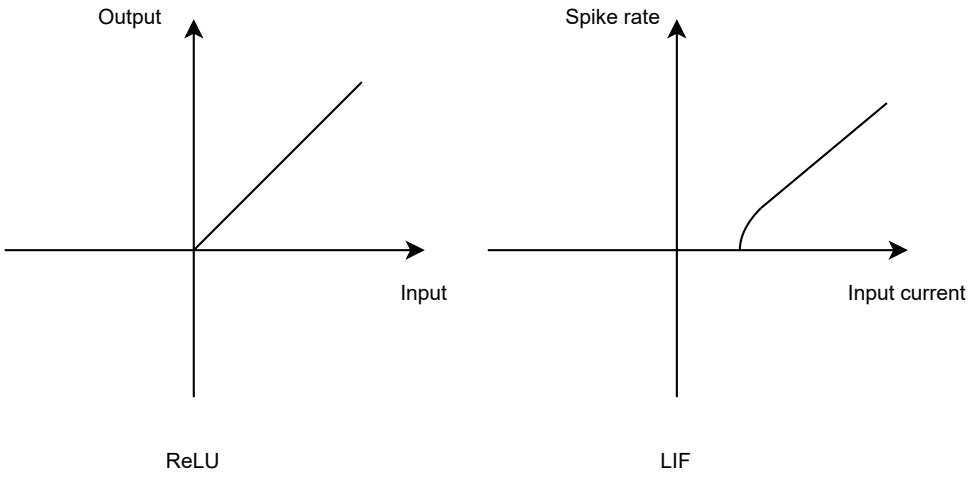

Figure 2.6: Similar input-output relation of ReLU and LIF.

In deep neural networks (DNN) in machine learning, there is also the notion of neuron, which could be viewed as a further simplification of the LIF neuron in neuroscience. For example, the output of Rectified Linear Unit (ReLU) is the same as the input if the input is larger than zero, otherwise the output is zero, which is similar to the relation of input current and output spike rate in the LIF neuron (Figure 2.6).

The LIF neuron model and the current-based synapse model are used in chapter 3. For chapter 4, more advanced models are used, which are described in that chapter. In chapter 5, for the keyword spotting benchmark, the ReLU model is used, and for the adaptive control benchmark, the LIF neuron model is used.

2.3 Neural Network

The human brain mainly consists of three parts: cerebrum, cerebellum (small brain) and brain stem. While the cerebellum and brain stem are believed to be responsible for lower-level tasks, the cerebrum is considered to be responsible for higher-level tasks, such as the processing of sensory information, language and cognition. The outer surface of the cerebrum is the cerebral cortex which is composed of grey matter. The neocortex takes up the largest part of the cerebral cortex and has six layers in the vertical direction. As the name already indicates, the neocortex is the newest part of the brain from an evolutionary point of view. A higher proportion of the neocortex in the brain is often associated with higher cognitive capabilities. In the human brain, 90% of the cerebral cortex is neocortex [33].

In the neocortex, there are different areas responsible for different tasks, like the processing of visual, auditory or somatosensory stimuli or planning and control of movements. Within each area, there is a hierarchy of interconnected regions. For example, neurons in the primary visual cortex respond to simple stimuli like line segments of certain orientations [34], whereas neurons in the inferior temporal cortex respond to more abstract concepts like human faces [35].

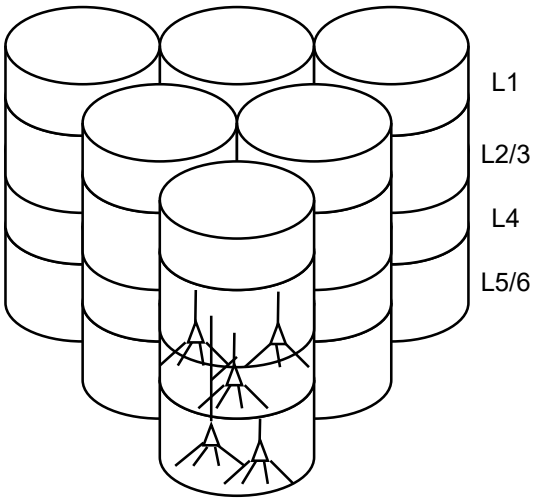

Figure 2.7: The hypothesized columnar structure of the neocortex with six layers in the vertical direction and similar columnar arrangement in the horizontal direction

The limited area where a neuron is sensitive to stimuli is called the neuron's re-

ceptive field. The term is not only used for neurons in the visual cortex but also used for neurons in other cortices like the auditory cortex and somatosensory cortex. In the auditory cortex, a receptive field of a neuron is the neuron's preferred tone frequency, and in the somatosensory cortex, the receptive field of a neuron is the position of the body where the neuron can respond to touch. By stimulating different receptive fields and measuring the response of the neurons, it has been found that neurons with close horizontal positions have different but similar receptive fields, whereas neurons in the different vertical layers have the same receptive fields. This observation gives rise to the hypothesis of the cortical column, where each column extends in the direction vertical to the cortex and consists of the six layers of neurons with the same receptive field, and the cortex is considered to be composed of these cortical columns with similar structure (Figure 2.7) [31].

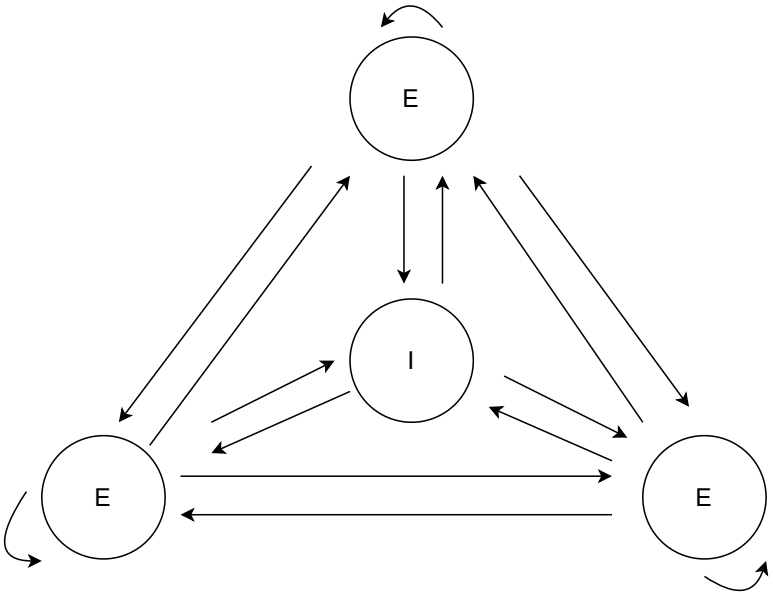

Figure 2.8: A winner-take-all network with three excitatory populations and one inhibitory population.

It is believed that there is a general circuit structure that is repeated in the cortical columns and forms the basis for cortical information processing. The winner-take-all (WTA) network is believed to be one basic building block [36]. In a WTA network, there are several excitatory neuron populations competing through an inhibitory neuron population (Figure 2.8). The excitatory populations are recurrently connected to

themselves, and they are also connected to other excitatory populations and the inhibitory population. At the same time, the inhibitory population is connected to the excitatory populations through inhibitory synapses. When the activity of one excitatory population is higher than the other excitatory populations, i.e. this population produces more spikes than the others, then it sends more spikes to the inhibitory population which raises the membrane potential of the inhibitory neurons, which then sends out spikes to all excitatory populations to suppress their activities. But because of the recurrent connections of the excitatory population, the activity of the excitatory population with higher firing rates can be self sustained.

In [36] it was proposed that the neurons in layer 2/3 form a WTA network which explores possible interpretations of input information, the output of the network is sent to another WTA network formed by the neurons in layer 5 which exploit the interpretations of the previous WTA network.

While the behavior of single neurons has been studied extensively, how neural networks give rise to cognition is still an open question. While the behavior of single neurons can be studied by electrical measurements and the behavior of larger brain areas can be studied with electroencephalography (EEG) or functional magnetic resonance imaging (fMRI), the study of the neural networks on the mesoscopic level is still partially constrained by the number of neurons that can be measured simultaneously. With the progress of neural recording technologies, more data will be available to improve the understanding of the working principles of the brain.

3

Efficient Spiking Neural Network Simulation using Dynamic Voltage and Frequency Scaling

3.1 Introduction

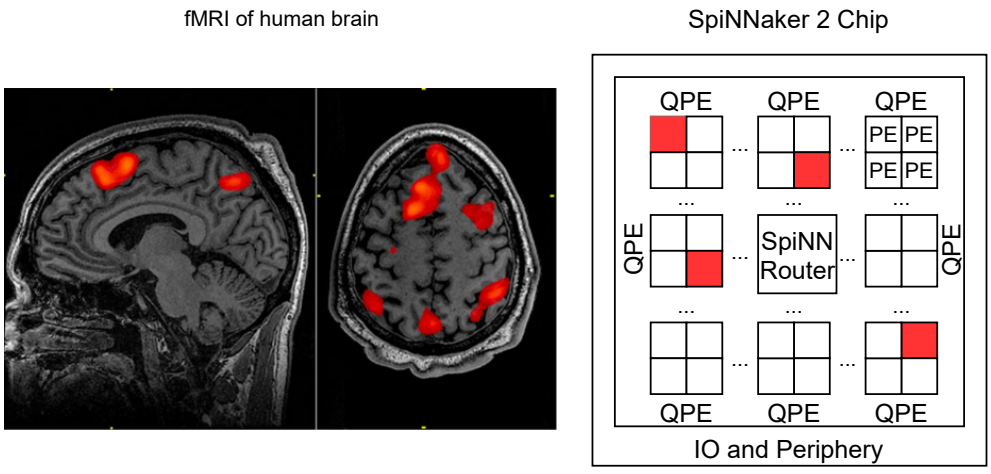

Figure 3.1: Left: Functional magnetic resonance imaging (fMRI) of a human brain undertaking a working memory task. The red color indicates the change in activity level in certain regions during the time when the task is performed [1]. Right: The SpiNNaker 2 chip with DVFS enabling each PE to adjust its Performance Level (PL) individually according to its work load. The red color indicates increased PL of certain PEs at a certain time point.

3. EFFICIENT SPIKING NEURAL NETWORK SIMULATION USING DYNAMIC VOLTAGE AND FREQUENCY SCALING

When an area of the brain becomes active, the neurons in that area consume more energy, which causes increased oxygen consumption. Based on blood-oxygen-level-dependent (BOLD) contrast, functional magnetic resonance imaging (fMRI) reveals the change of the activation level in certain areas in the brain when performing a certain task [37].

Similarly, in silicon chips, there is also the situation where certain parts of a chip consume more power while other parts consume less. When a multi-core neuromorphic chip simulates an SNN and each PE of the chip simulates a part of the SNN, in order to guarantee real-time operation, each PE should have enough computational power for its peak workload. The computational power is typically represented by the clock frequency. Increasing the clock frequency means also increasing the supply voltage and thus the power consumption. If the chip has only one supply voltage, then this should be tailored for the peak workload of all PEs throughout the time, although normally only a fraction of the PEs are at the peak workload.

In this case, it would be desirable to be able to dynamically adjust the supply voltage and computational power for each PE according to its current workload, just like in the brain where more energy is delivered to the areas with higher activity levels (Figure 3.1).

In the semiconductor industry, the Dynamic Voltage and Frequency Scaling (DVFS) technology has been applied to switch the supply voltage and clock frequency of a circuit dynamically during run time, in order to increase power efficiency, e.g. for power critical applications like smart card devices [38]. However, such technology usually requires a central node to predict the workload of each PE. In SNN simulation, due to the distributed nature of the SNN, a centralized control node is not possible.

SpiNNaker 2 comes with a per-core DVFS technology, where each PE decides on its own Performance Level (PL) , i.e. voltage and frequency setting [24, 25]. This allows each PE to dynamically adjust its PL according to its own workload at each simulation time step, which drastically increases power efficiency in SNN simulation compared to an approach without DVFS.

In this chapter, the benefit of DVFS is demonstrated in an SNN simulation example [1] [2]. First, section 3.2 provides more details on the hardware aspect of DVFS. Then,

[1] The results in this chapter are obtained from the second SpiNNaker 2 prototype chip.

section 3.3 describes the SNN model. Third, section 3.4 details the software implementation of the model on the SpiNNaker 2 prototype. Fourth, section 3.5 describes aspects related to power management strategies. The measurement results are then presented in section 3.6. Finally, section 3.7 summarizes this chapter.

3.2 Power Management Hardware

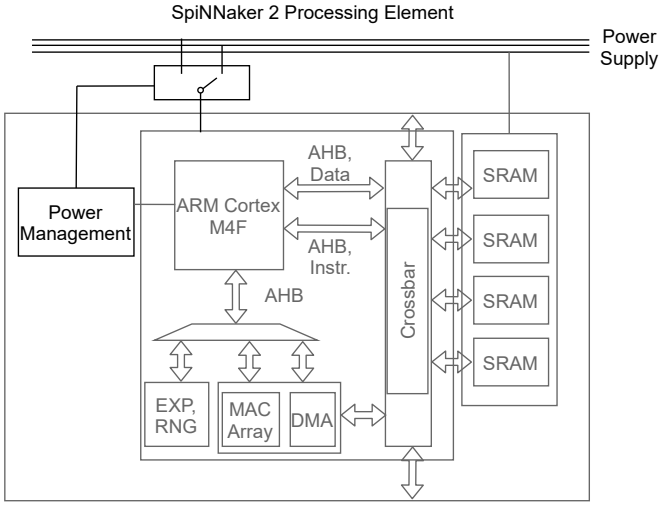

Figure 3.2: Power management in the SpiNNaker 2 PE architecture.

The prototype chip considered in this chapter is the second SpiNNaker 2 prototype [2] which contains 8 PEs and is fabricated with the GLOBALFOUNDRIES 22FDX technology [23], the same as the final SpiNNaker 2 chip. As shown in Figure 3.2, for each SpiNNaker 2 PE, 3 off-chip generated supply voltages are available. One is for the SRAM memory part (0.8 V), two are for the logic and computation part (0.5 V and 0.6 V). The power management module developed in [24, 25] allows the logic and computation part of the PE to switch between the two supply voltages during program run time. The Performance Level (PL) is a voltage and frequency pair. For the measurements in this chapter, three PLs are defined: PL1 (0.5 V, 100 MHz), PL2 (0.5 V, 200 MHz) and PL3 (0.6 V 400 MHz). The 400 MHz of PL3 is chosen for peak computation load. For this frequency, 0.6 V is necessary. The 200 MHz of PL2 is for moderate computation load. And 100 MHz of PL1 is for low computation load. For 200 MHz, 0.5 V is necessary. Since there are only two supply voltages available, for

100 MHz, also 0.5 V is used. From the software point of view, since it is desirable to spread the time point of spike generation across the available 1 ms time step, in order to reduce the pressure on the communication fabric, for low computation load, 100 MHz is used.

The PL is switched by the power management module upon receiving the command from the ARM core. In order to avoid rush current when switching between the supply voltages, a pre-charge scheme is used [39]. Nevertheless, the switching time is fast enough to be considered negligible from the software and application perspective.

3.3 Spiking Neural Network Model

In this section, first, the simple locally connected network for the parameter extraction for the energy per neuron update and energy per synaptic operation is described in section 3.3.1. Then, in section 3.3.2, the synfire chain network model is introduced for the demonstration of the benefit of DVFS when the workload of individual PEs changes over time. The neuron and synapse models used in this chapter are the same as the LIF neuron model and current-based synapse model introduced in chapter 2.

3.3.1 Locally Connected Network

The locally connected network was first used in [40] for the power measurement of SpiNNaker 1. The same network is used in the measurement of SpiNNaker 2.

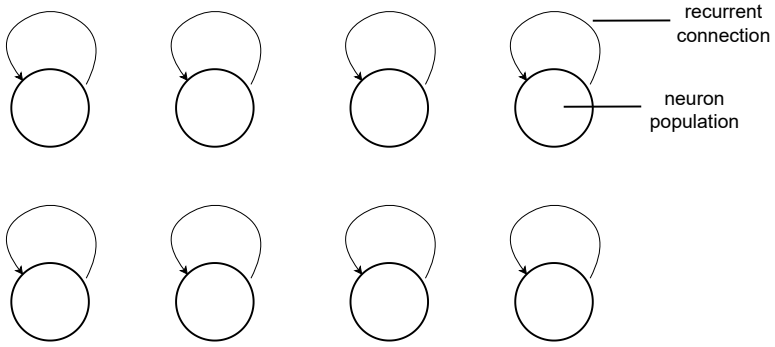

Figure 3.3: The locally connected network for the measurement of the energy per neuron update and energy per synaptic operation.

As shown in Figure 3.3, the network consists of several neuron populations. In each population, each neuron is connected to each other neuron in the same population.

Thus a population with 100 neurons would have 100 x 100 = 10^4 synaptic connections. If each neuron spikes once per 10 ms, then a population generates (1 s / 10 ms) x 100 x 100 = 10^6 synaptic operations per second.

In this network, the neurons spike not depending on the postsynaptic current. Instead, the postsynaptic current is ignored and the spiking behavior is directly controlled by the software. In this way, the number of spikes can be controlled for the purpose of power measurement for the hardware.

3.3.2 Synfire Chain Network

Synfire chain [41, 42] is a network with many layers where the neurons in the previous layer are connected to some neurons in the next layer through excitatory connections, i.e. a spike generated by the presynaptic neuron increases the membrane potential of the postsynaptic neuron, and brings the postsynaptic neuron in the direction of the generation of an action potential, thus 'exciting' the postsynaptic neuron [31]. The structure and dynamics of synfire chain have been studied in computational neuroscience as a cortical network model that can sustain synchronous spiking activity, which is considered important for cortical processing.

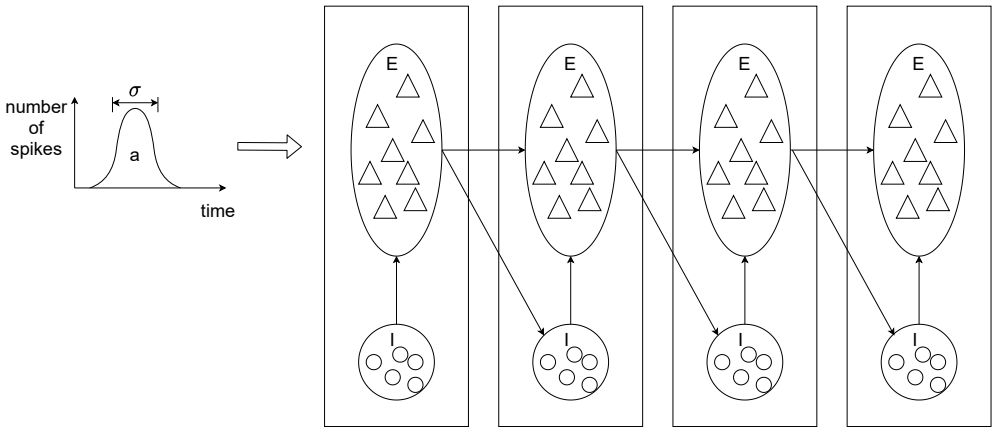

Figure 3.4: Network structure of synfire chain. 'E' stands for excitatory and 'I' stands for inhibitory. A pulse packet is usually used to kick-off the network activity in a simulation.

To kick-off the activity of the network, usually a pulse packet is sent to the first layer of the network. A pulse packet is a number of spikes sent within a time frame, where the number of spikes at each time step follows the normal distribution. The parameters

that characterize the pulse packet is the number of spikes (a) which describes the size of the pulse packet and the standard deviation of the normal distribution (σ) which describes the temporal dispersion, i.e. to what extend are the spikes synchronous. The response of the neuron layer to the pulse packet is another pulse packet. It has been shown that for some parameter sets, the standard deviation of the pulse packet decreases as it travels through the layers, i.e. the spiking activity of the neurons in a layer becomes more synchronous, exhibiting attractor behavior. This happens when the size of the pulse packet is large and the standard deviation is small. On the other hand, when the size of the pulse packet is small and the standard deviation is large, the pulse packet might die out [43].

The synfire chain network architecture considered in this work contains lateral inhibition, i.e. in each layer, there are also some inhibitory neurons sending spikes to the excitatory neurons in the same layer (Figure 3.4), which was proposed in [44] to account for the observation that cortical neurons receive balanced excitatory and inhibitory inputs with inhibition lagging excitation by a few milliseconds. The inhibitory neurons have been proved to increase the stability of the network dynamics, since it prevents the network from falling into synchronous firing due to random background noise. The inhibition also increases the selectivity of the network, so that only pulse packets with higher synchrony, i.e. smaller σ will pass through the synfire chain, which essentially gives the synfire chain a high pass characteristic [44].

3.4 Software Flow

In this section, first, the basic software flow is introduced in section 3.4.1. Based on this, the extended software flow which accounts for the DVFS feature is introduced in section 3.4.2.

3.4.1 Basic Software Flow

The SpiNNaker 2 chip contains many PEs and in each PE the ARM core runs independently. The software running on the ARM core is written in C code. Each PE simulates a number of neurons and synapses. When a neuron spikes, the spike is sent to the SpiNNaker router. The spike packet contains a neuron ID to identify its origin. Depending on the neuron ID, the SpiNNaker router finds the destination PE of the

spike packet based on a predefined routing table. Then the SpiNNaker router performs multicast routing to send the spike to its destination(s).

Typically the simulation runs in real-time and each time step is 1 ms. A timer that is available on each PE is used to trigger a timer interrupt every millisecond. When the timer interrupt comes, the simulation advances to the next time step. The ARM cores are started at the same time so that in subsequent time steps the timer interrupt comes synchronously.

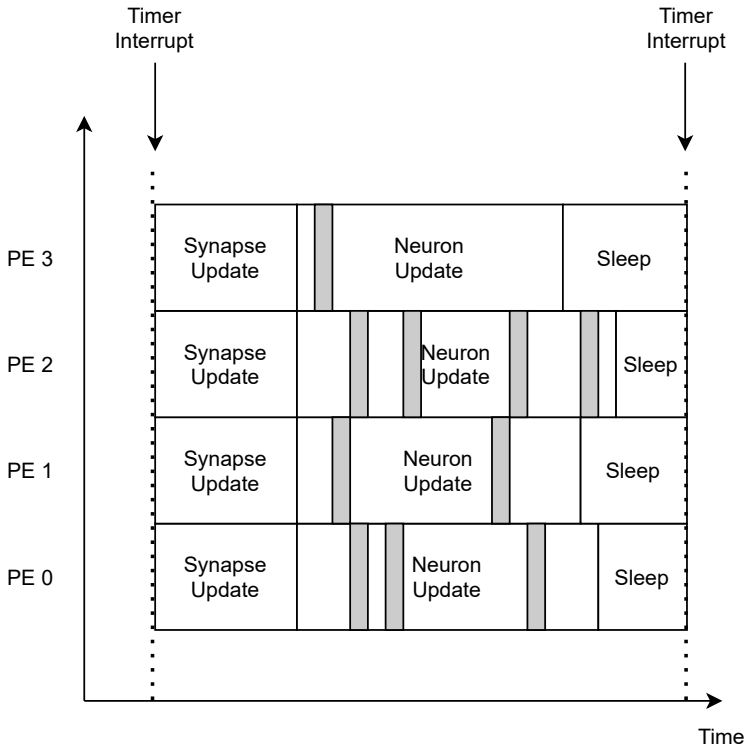

Figure 3.5: Illustration of SpiNNaker 1 software flow. The software runs on the ARM cores independently. In each time step, there are two processing steps: synapse update and neuron update. After these two steps are finished, the ARM core goes to sleep. Note that the processing is interrupted by incoming spike events which need to be processed immediately (shown in grey).

The original SpiNNaker 1 software flow [45] can be illustrated with Figure 3.5. Inside each time step, there are two processing steps: synapse update and neuron update. During the processing, spikes from other PEs interrupt the normal processing flow. After the neuron processing step is done, the ARM core goes into sleep mode,

which also reserves the time buffer until the next timer interrupt.

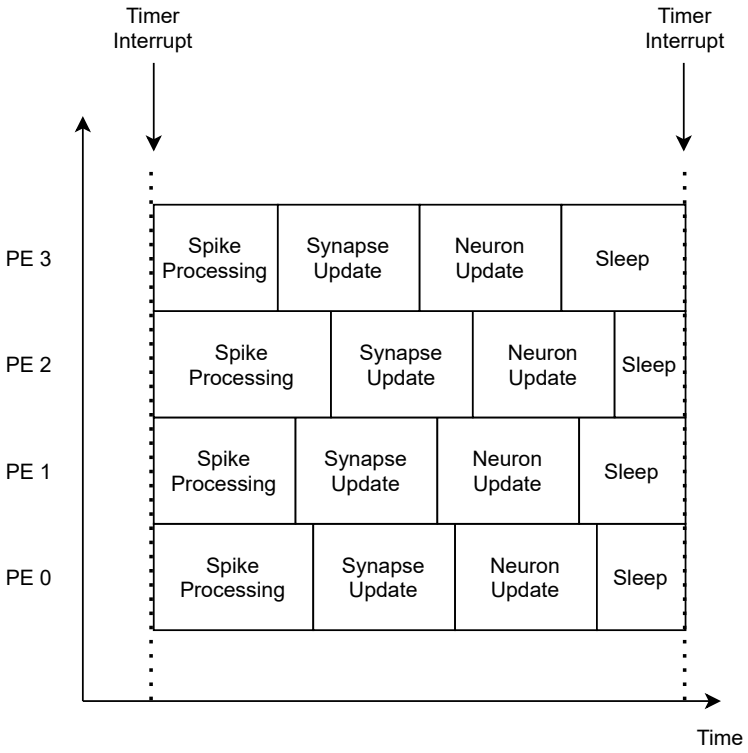

Figure 3.6: The basic software flow running on SpiNNaker 2. In each time step, there are three processing steps: spike processing, synapse update and neuron update. The difference to the SpiNNaker 1 software flow is that the spike processing is not done immediately after receiving the spike. Instead, the spikes are buffered and processed in the next time step, which is the basis for DVFS software flow.

The original SpiNNaker 1 software flow is not suitable for DVFS, because the dynamic adjustment of PLs requires the knowledge of the work load in the future. When the number of spikes to be processed is not known, the application of DVFS is not possible. To overcome this, the basic software flow on SpiNNaker 2 has been adjusted so that spikes are not processed immediately when they are received, but they are buffered and processed in the next time step. In this way, the number of spikes to be processed in the next time step is known, which enables the application of DVFS (Figure 3.6).

Thus, the basic software flow running on SpiNNaker 2 has three processing steps: spike processing, synapse update and neuron update. The three processing steps are described in the following:

First, in the spike processing step, the spikes which have arrived in the last time step are processed. The neuron ID in the spike packet is used to find the postsynaptic neuron of this source neuron. This information is stored in a table called the master population table. It contains the address and length of a memory block for the information of the synapses between each postsynaptic neuron and the source neuron. This information is then retrieved to the ARM core.

Second, the synapse update step accounts for the effects the previous spikes have on neurons in this time step, because the spikes might have an effect on the neuron membrane potential at a later time point due to synaptic delay. In this step, the synapse information, i.e. synaptic weight, synapse type (excitatory or inhibitory) ,etc., is integrated into the appropriate slot in the buffer which accounts for the synaptic delay.

The third step is neuron update, which updates the neuron membrane potential. When the membrane potential reaches the threshold voltage, a spike is emitted. The spike packet is then sent to the destination PE through the SpiNNaker router. When the spike arrives at its destination PE, it will be stored in a buffer and processed in the next time step. After spiking the neuron goes into a refractory period.

3.4.2 DVFS Software Flow

In the spiking neural network simulation, the number of neurons and synapses simulated in a PE is defined, so the computational load for the neuron update and synapse update remains constant, whereas the computational load for spike processing varies from time step to time step depending on the number of spikes arriving at a PE in each time step. This cannot be predetermined before the start of the simulation and depends on the network dynamics.

To guarantee real-time operation, i.e. the time the hardware takes to simulate an amount of time is equal to the amount of time simulated, the number of clock cycles available within a time step defines the computation power available: when the PE runs at 100 MHz, in a time step of 1 ms, 100 000 clock cycles are available, when the PE runs at 200 MHz, 200 000 clock cycles are available, and when the PE runs at 400 MHz, 400 000 clock cycles are available. Thus, in the context of DVFS, switching between the PLs means switching between the computation power, which allows the PE to have more computation power by switching to a higher PL when more spikes

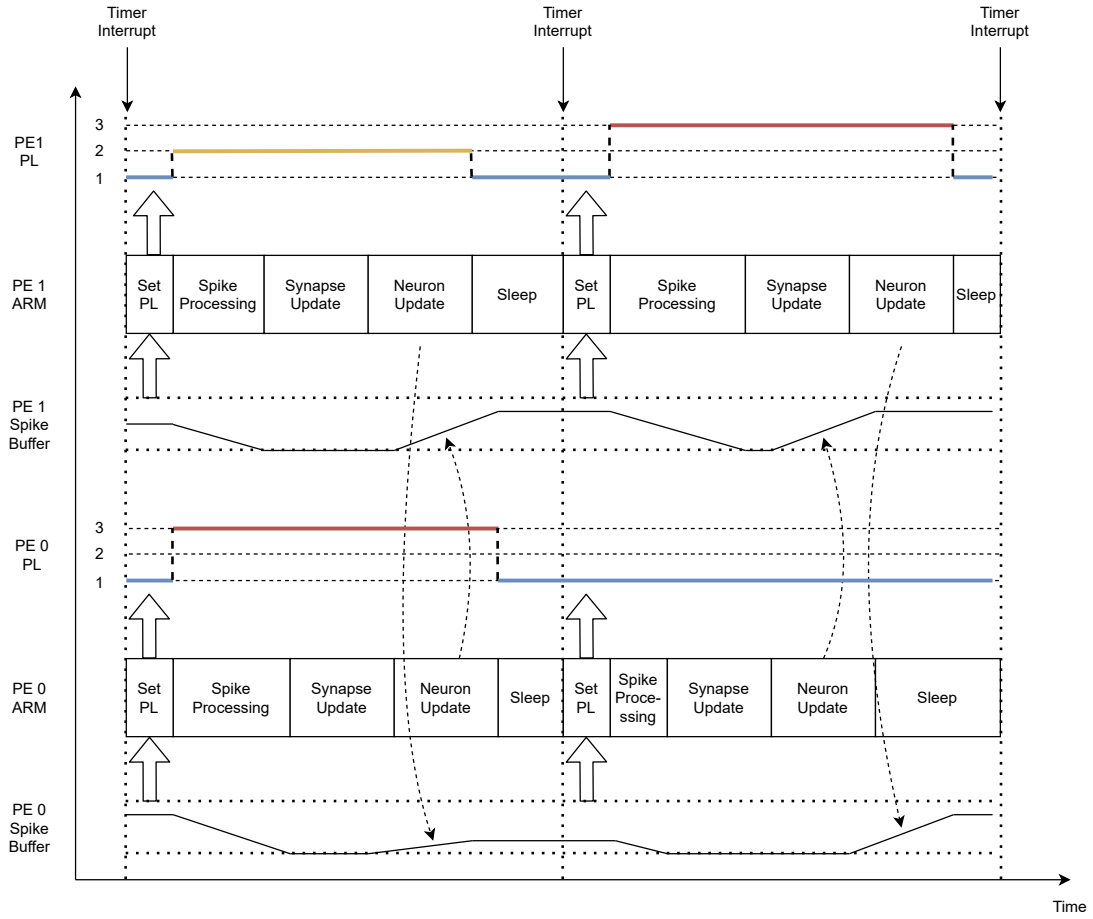

Figure 3.7: DVFS software flow. In this example, PE 0 sends spikes to PE 1 and PE 1 sends spikes to PE 0. At the beginning of each time step, the PEs run at the lowest PL. For PE 0, in the first time step, in the first processing step 'Set PL', the ARM core decides to set PL 3 for this time step, because of the high fill level of the spike buffer. After the neuron update step is finished, the PL is switched back to PL 1. In this time step, during the neuron update step of PE 1, relatively few spikes are sent to PE 0, so that the fill level of the spike buffer of PE 0 stays at a low level. In the next time step, due to this low fill level, PE 0 decides to stay at PL 1 for this time step.

need to be processed and to save power by switching to a lower PL when fewer spikes need to be processed.

The basic software flow described previously has been adapted to work with the DVFS feature in SpiNNaker 2 (Figure 3.7). At the beginning of the time step, the lowest PL, PL 1 is applied. In the 'Set PL' step, the ARM core evaluates the fill level of the spike buffer, i.e. how many spikes have arrived at this PE in the last time step. The spike buffer is a predefined area in the SRAM. When a spike comes, it can be stored into the spike buffer without interrupting the software running on the ARM core. Depending on the fill level of the spike buffer, the PL of this time step is decided and configured accordingly. After the spike processing, synapse update and neuron update steps are finished, the PL is reset to the lowest level and the ARM core goes to sleep.

3.5 Power Management Strategies

In this section, first, detailed analysis of the computational cost and PL selection strategy is described in section 3.5.1. Then an energy model is provided in section 3.5.2. Finally, the differential power measurement strategy is described in section 3.5.3.

3.5.1 Performance Level Selection

For the decision of which PL to choose, a strategy needs to be developed. On the one hand, the PL needs to be as low as possible to save energy. On the other hand, the PL needs to be high enough so that the PE has enough computation power to finish the computation within the time step in order to guarantee real time operation.

Specifically, two workload thresholds, $c_{th,1}$ and $c_{th,2}$ need to be defined. When the workload exceeds $c_{th,1}$, the PL needs to be switched from PL 1 to PL 2. When the workload exceeds $c_{th,2}$, the PL needs to be switched from PL 2 to PL 3:

$$\mathrm{PL}(k) = \begin{cases} \mathrm{PL1}, \text{if } c < c_{\mathrm{th},1} \\ \mathrm{PL2}, \text{if } c_{\mathrm{th},1} \leq c < c_{\mathrm{th},2} \\ \mathrm{PL3}, \text{if } c_{\mathrm{th},2} \leq c \end{cases} \tag{3.1}$$

where c is the computational cost measured in clock cycles. As described previously, the computational cost can be divided into two parts, one part is the fixed computational cost, which includes neuron update and synapse processing, the other part is the variable computational cost, which includes spike processing.

The fixed computational cost is constant because after the network topology is defined, during simulation run time, the computational cost does not change, since it only depends on the number of neurons a PE simulates:

$$c_{\text{fixed}} = n_{\text{neuron}} \cdot c_{\text{neuron}} + c_{\text{other}}, \tag{3.2}$$

where n_{neuron} is the number of neurons simulated on a PE and c_{neuron} is the fixed computational cost per neuron, which includes the synapse update and neuron update cost. c_{other} accounts for some other operations for maintaining the simulation environment.

The variable computational cost depends on the number of spikes that arrive at a PE in each time step. When a spike arrives at a PE, there is an overhead that is related to finding the postsynaptic neurons and fetching the synapse information. Then each postsynaptic connection needs to be processed so that the computational cost is different for each spike depending on the number of postsynaptic connections. The variable computational cost for a certain time step can be formulated as:

$$c_{\text{var}}(t) = \sum_{i=1}^{n_{\text{spikes}}(t)} g(i) \cdot c_{\text{syn}} + n_{\text{spikes}}(t) \cdot c_{\text{spike}}, \tag{3.3}$$

where $n_{\text{spikes}}(t)$ is the number of spikes that arrive at the time step t, $g(i)$ is the number of postsynaptic connections related to the ith spike, c_{syn} is the computational cost for each postsynaptic connection, and c_{spike} is the fixed overhead associated with finding the postsynaptic neurons and fetching the synapse information.

Finally, the total computational cost c of a time step is the sum of the fixed computational cost c_{fixed} and variable computational cost $c_{\text{var}}(t)$:

$$c = c_{\text{fixed}} + c_{\text{var}}(t). \tag{3.4}$$

Based on equations 3.2, 3.3 and 3.4, the PL selection strategy can be derived. Since the fixed computational cost c_{fixed} is known in advance, the PL switching depends

only on the variable computational cost $c_{\text{var}}(t)$. In $c_{\text{var}}(t)$, there are two variables: the number of spikes $n_{\text{spikes}}(t)$ and the number of postsynaptic connections of each spike $g(i)$.

To compute the exact computational cost, it would require iterating through all the spikes that have arrived at the PE in the last time step and finding out the number of postsynaptic connections $g(i)$ for each spike. This is theoretically possible and leads to the most accurate estimation of the computational cost. However, this approach would result in considerable overhead, which again depends on the number of incoming spikes. The overhead is not negligible and reduces the benefit brought by DVFS.

Therefore, approximation strategies are necessary. For example, instead of calculating the total computational cost c, the PE can count the computational cost of each incoming spike until $c_{\text{th},2}$ is reached, then the computational cost is enough to switch to PL 3. For the decision for PL 1 and PL 2 all spikes need to be iterated, but assuming counting all spikes in the case of PL 1 and PL 2 will be less effort than for PL 3, the overhead could be accepted. A more simple approach is to only consider the number of incoming spikes. Since the postsynaptic connections of each PE are known before the simulation start, the thresholds depending on the number of incoming spikes can be calculated based on the worst-case computational cost, i.e. assuming the first spike arriving at the PE has the most postsynaptic connections, the second spike has the second most connections, etc. This simple approach reduces the overhead for the PL decision but increases the chance that the PL setting is over-conservative, e.g. PL 3 is chosen when PL 2 would be enough.

3.5.2 Energy Model

Based on the previous analysis of the computational components, an energy model can be derived. The energy consumption per time step consists of 3 parts: baseline energy, neuron processing energy and spike processing energy.

The baseline power is the power consumed by the PE when no neuron processing and synapse processing is involved. Mainly it consists of the leakage power and the dynamic power for the processing of the timer interrupt itself. The baseline power of PL i is denoted as $P_{\text{BL,i}}$. The baseline energy of a time step is the product of baseline power and time. Since in one time step, two PLs might exist, i.e. the PL chosen for

the neural simulation of this time step and the PL 1 used before and after the neural simulation, the baseline energy of a time step E_{BL} consists of two parts:

$$E_{\mathrm{BL}} = P_{\mathrm{BL,i}} \cdot t_{\mathrm{neural}} + P_{\mathrm{BL,1}} \cdot (t_{\mathrm{ms}} - t_{\mathrm{neural}}), \tag{3.5}$$

where $P_{\mathrm{BL,i}}$ is the baseline power for the PL i chosen for the neural simulation, t_{neural} is the time for the neural simulation of the time step, i.e. the time for spike processing, synapse update and neuron update, $P_{\mathrm{BL,1}}$ is the baseline power for PL 1, t_{ms} is the time of a time step, i.e. 1 ms, and $t_{\mathrm{ms}} - t_{\mathrm{neural}}$ is the time within the time step when the PE is not doing neural processing, i.e. the 'Set PL' phase and sleep phase.

The neuron processing energy corresponds to the energy consumed for the fixed computational cost c_{fixed} in the previous section. Similar to equation 3.2, the neuron processing energy at PL i is modeled as

$$E_{\mathrm{neuron,i}} = E_{\mathrm{neuron,0,i}} + e_{\mathrm{neuron,i}} \cdot n_{\mathrm{neuron}}, \tag{3.6}$$

where $E_{\mathrm{neuron,0,i}}$ is an offset energy at PL i, $e_{\mathrm{neuron,i}}$ is the energy per additional neuron at PL i and n_{neuron} is the number of neurons.

The spike processing energy corresponds to the energy consumed for the variable computational cost $c_{\mathrm{var}}(t)$ in the previous section. Similar to equation 3.3, the spike processing energy at PL i is modeled as

$$E_{\mathrm{spike,i}} = E_{\mathrm{syn,0,i}} + e_{\mathrm{syn,i}} \cdot n_{\mathrm{syn}}, \tag{3.7}$$

where $E_{\mathrm{syn,0,i}}$ is an offset energy at PL i, $e_{\mathrm{syn,i}}$ is the energy per additional synaptic event at PL i and n_{syn} is the number of synaptic events. Note that for the sake of simplicity, only the first term in the r.h.s. of equation 3.3 is considered.

Finally, the total energy of a time step is

$$E_{\mathrm{all}} = E_{\mathrm{BL}} + E_{\mathrm{neuron,i}} + E_{\mathrm{spike,i}}, \tag{3.8}$$

3.5.3 Power Measurement Strategy

To measure the energy components in the previous section when the chip is running a neural simulation, a differential measurement approach is adopted:

- First, the total power P_0 is measured with the neural simulation running on the chip.
- Then spike sending is deactivated and the power P_1 is measured. Since no spike is sent, no spike is received and processed. The difference $P_{\text{spike}} = P_0 - P_1$ is the spike processing power.
- Then all neural processing is deactivated and upon timer interrupt only empty interrupt handlers are called. The power at this stage is measured as P_2. $P_{\text{neuron}} = P_1 - P_2$ is the neuron processing power.
- Then the ARM cores are deactivated and the power is measured as P_3. $P_{\text{BL}} = P_2 - P_3$ is the baseline power.

When DVFS is enabled, the PL is determined by the network activity. Thus, after measuring P_0 with the complete software autonomously switching PLs during simulation, the percentage of simulation time at the 3 PLs are recorded and then applied to the simulations when measuring P_1 and P_2.

3.6 Measurement Results on Test Chip

Based on the power management feature described in section 3.2, the software flow described in section 3.4 and power management strategies described in section 3.5 are applied in neural simulation. In this part, first the parameter extraction results based on the locally connected network (section 3.3.1) are shown in section 3.6.1, then the benefit of DVFS is shown in detail with the synfire chain model (section 3.3.2) in section 3.6.2. The results presented in this section are based on the measurement done on the second SpiNNaker 2 prototype chip [2].

3.6.1 Locally Connected Network

Based on the differential power measurement strategy described in section 3.5.3, the parameters in the energy model described in section 3.5.2 is extracted. For the parameter extraction, the locally connected network described in section 3.3.1 is used. Here no PL selection strategy (section 3.5.1) is applied since for the parameter extraction the PE only needs to run the neural simulation at a predefined PL.

For baseline power P_{BL} extraction the simulation without spike processing, neuron processing and synapse processing has been executed at different PLs.

The neuron processing power has been determined by running the neuron processing and synapse processing for different numbers of neurons per PE in the locally connected network for different PLs. The measurement results are then subtracted by the baseline power to obtain the additional power for neuron processing and synapse processing. The results are shown in Figure 3.8. Based on the results, linear regression is done and the slope of the line is found, which characterizes the additional energy per neuron.

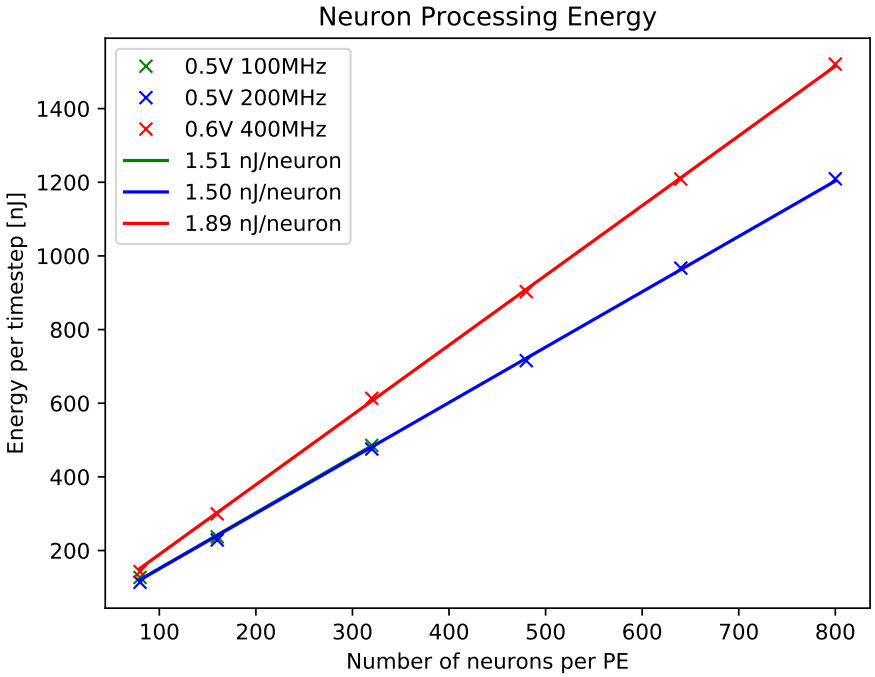

Figure 3.8: Energy per timestep for different number of neurons per PE

Similarly, the spike processing power has been measured by running the spike processing with a varying number of synaptic events. Here, the locally-connected network with 80 neurons per PE is used. The number of synaptic events per time step per PE is varied by varying the number of neurons that are sending out spikes in each time step. The result of the spike processing energy measurement is shown in Figure 3.9. The additional energy per synaptic event is found with linear regression.

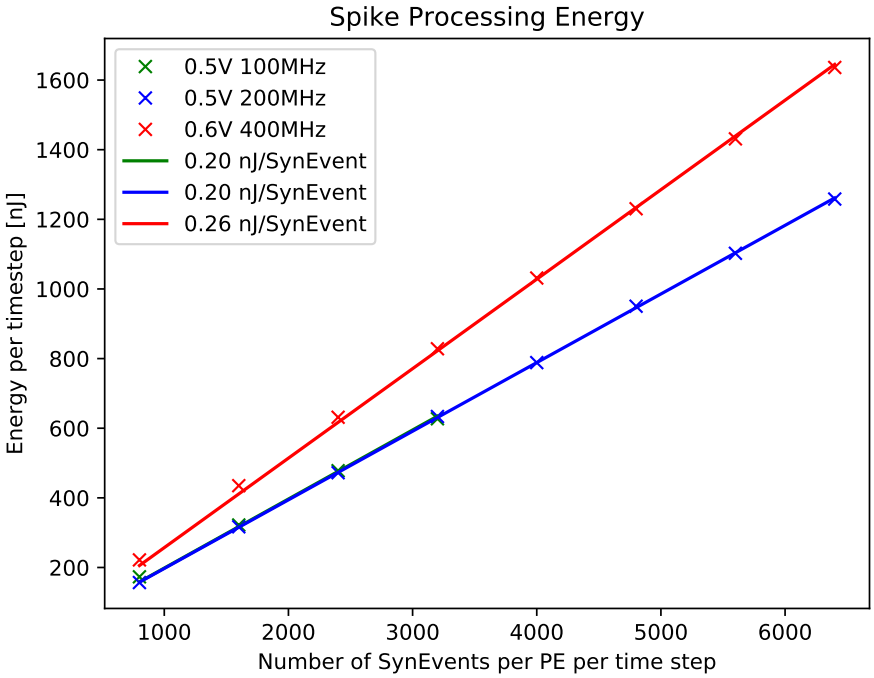

Figure 3.9: Energy per timestep for different number of synaptic events per PE

Finally, the results of the power model parameter extraction are summarized in Table 3.1.

Table 3.1: Measured parameters of energy model

	PL1 (0.5 V 100 MHz)	PL2 (0.5 V 200 MHz)	PL3 (0.6 V 400 MHz)
P_{BL} [mW]	22.38	29.72	66.44
e_{neuron} [nJ]	1.51	1.50	1.89
e_{syn} [nJ]	0.20	0.20	0.26

With these extracted parameters, the energy consumption of a PE can be predicted by inserting these parameters to the energy model described in section 3.5.2. Note that comparing the equations 3.6 and 3.7, the $E_{\text{neuron,0,i}}$ and $E_{\text{syn,0,i}}$ are missing in Table 3.1, because they were found to be 0.

3.6.2 Synfire Chain

Finally, the DVFS hardware feature (section 3.2), DVFS software flow (section 3.4) and power management strategy (section 3.5) described in this section are demonstrated in the synfire chain simulation based on the model described in section 3.3.2.

Specifically, in the synfire chain network model implemented in this work, there are four layers, each consisting of an excitatory and inhibitory population. Each layer is simulated on a PE. the fourth layer is connected back to the first layer to form a loop. In each layer, the excitatory population has 200 neurons, and the inhibitory population has 50 neurons. A normally distributed current is fed into each neuron to simulate the random background noise.

The connection from the inhibitory population to the excitatory population in the same layer has a synaptic delay of 8 ms, and the connection from the excitatory population to both populations in the next layer has a synaptic delay of 10 ms. Each neuron in one layer is connected to 60 excitatory neurons in the previous layer. Within a layer, each excitatory neuron is connected to 25 inhibitory neurons in the same layer.

As described in section 3.3.2, to kick start the network activity, a pulse packet is used at the start of the simulation. For the selection of PL, the simplified worst-case strategy described in section 3.5.1 is adopted, i.e. PL switching is based on the number of incoming spikes, assuming the incoming spikes have most of the postsynaptic connections. The two thresholds $l_{\text{th,1}}$ and $l_{\text{th,2}}$ for PL switching are summarized in Table 3.2, along with other parameters of the network model.

The simulation is done with these settings and the spike trains, the chosen PL for each time step and the number of incoming spikes are shown in Figure 3.10. After PE 3 sends the stimulus pulse packet to PE 0, the number of incoming spikes in PE 0 rises, which causes PE 0 to raise its PL from PL 1 over PL 2 to PL 3 in order to guarantee real-time operation. After a delay of a few milliseconds (i.e. synaptic delay), the neurons in PE 0 start to fire, sending spikes to PE 1. While the spikes from PE 0 are relatively asynchronous, the spike packet becomes more and more synchronous

Table 3.2: Synfire Chain Network Parameters

parameter	value	note
n_{exc}	200	number of excitatory neurons in a layer
n_{inh}	50	number of inhibitory neurons in a layer
$t_{\text{delay,i}}$	8 ms	synaptic delay from the inh. to exc. in the same layer
$t_{\text{delay,e}}$	10 ms	synaptic delay from the exc. to all neurons in the next layer
$n_{\text{pre exc}}$	60	number of presynaptic excitatory neurons from the previous layer
$n_{\text{pre inh}}$	25	number of presynaptic inhibitory neurons from the same layer
$l_{\text{th,1}}$	17	DVFS switching threshold for PL 1 and PL 2
$l_{\text{th,2}}$	59	DVFS switching threshold for PL 2 and PL 3

over time. At the end of the simulation at around 90 ms, the spike packet from PE 3 becomes a very synchronous pulse packet where most of the spikes occur in the same time step.

The synfire chain simulation shows the variability of the network activity in spiking neural network simulations. In hardware without DVFS, PL 3 needs to be used throughout the simulation in order to guarantee enough computational power for real-time operation, which increases power consumption. Whereas in SpiNNaker 2, because of DVFS, the PLs can be switched up and down dynamically to adjust for the network activity and save power consumption. In order to quantify this benefit, power measurement has been carried out.

Following the same incremental power measurement method (section 3.5.3), the baseline power, neuron processing power and spike processing power can be measured. To demonstrate the benefit of DVFS, the power measurement is done for two cases:

- DVFS enabled. DVFS software actively switching PLs based on the power management strategies.
- DVFS disabled. To guarantee real-time operation PL 3 is always used.

The results of the power measurement are compared and shown in Table 3.3.

The comparison of the power consumption for the synfire chain simulation with and without DVFS shows most of the benefit of DVFS in saving baseline power, with a power reduction of 63.4%. The neuron processing power is reduced by 27.3% and the spike processing power is reduced by 18.7%. The total power reduction is 60.7%.

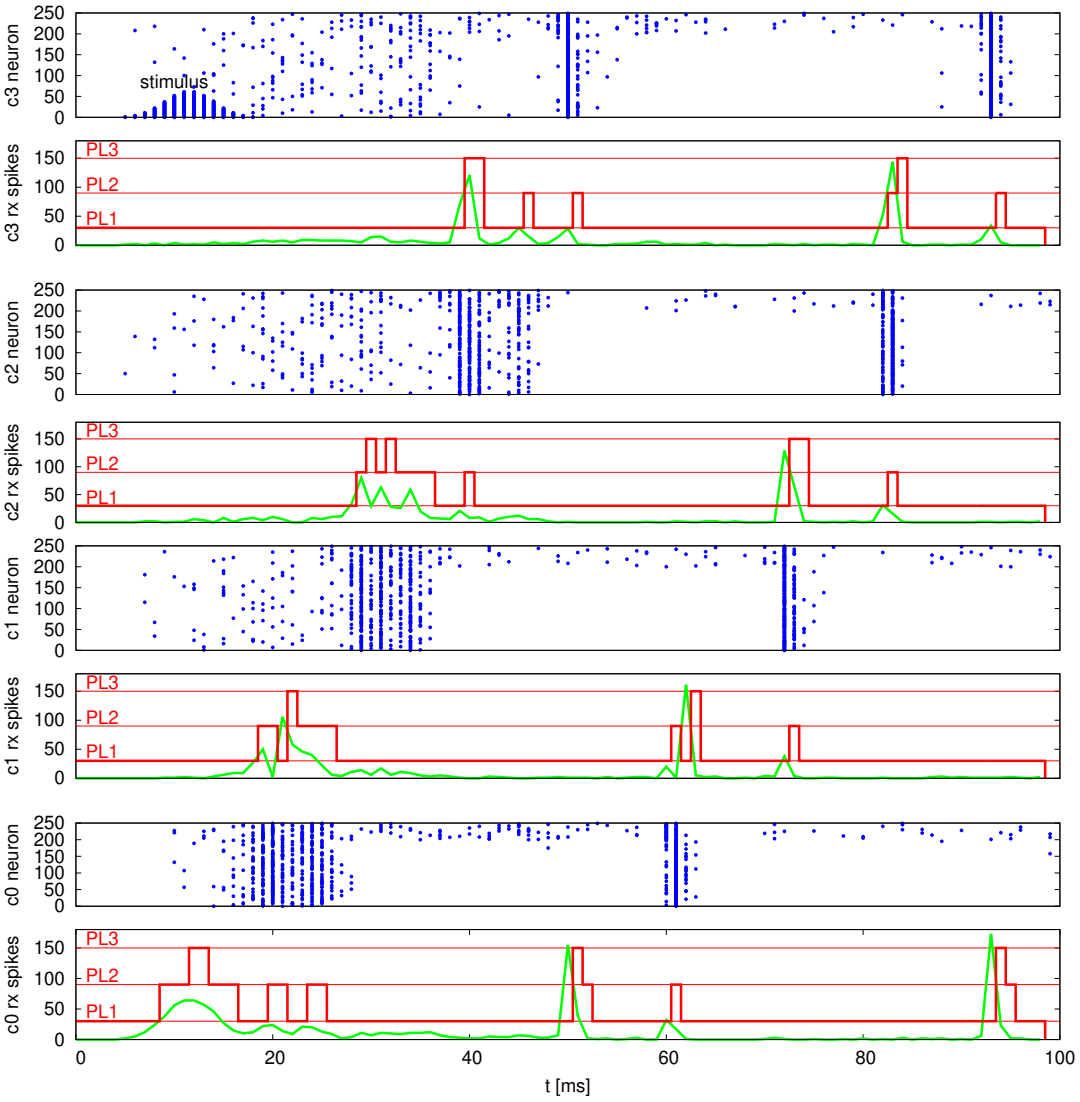

Figure 3.10: Simulation of synfire chain. The X-axis shows time in milliseconds. In the label of the Y-axis, c0 denotes PE 0, c1 denotes PE 1 and so on. For each PE, two panels are shown, one for the spike train (blue), and one for the number of received spikes (green) and the PL chosen (red). At the start of the simulation, a stimulus pulse packet is used to kick start the network activity. These spikes are sent from PE 3 (top panel) to PE 0 (bottom panel), as can be seen from the rise of the number of incoming spikes in PE 0. The rise of the incoming spikes causes PE 0 to raise its PL from PL 1 over PL 2 to PL3. After a short delay, the neurons in PE 0 start to fire, sending spikes to PE 1, and so on. The spike pulse packet from one layer to the next layer becomes more and more synchronous over time.

Table 3.3: Power Measurement Results for Synfire Chain Simulation (mW)

	only PL 3	DVFS	reduction
baseline power	66.4	24.3	63.4%
neuron processing power	3.3	2.6	21.2 %
spike processing power	1.6	1.3	18.7%
total power	71.3	28.2	60.4%

The comparison of the power model and the measurement for total power, baseline power, neuron processing power and spike processing power for the synfire chain simulation is shown in Figure 3.11. The power model shows an overall good prediction for the actual power consumption.

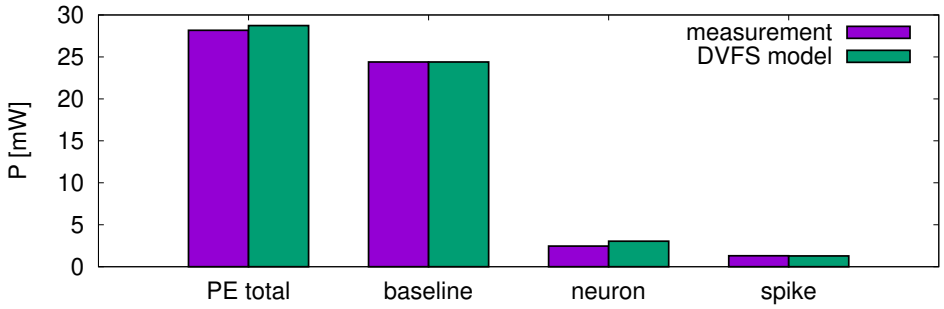

Figure 3.11: DVFS model vs. measurement

Furthermore, the number of simulation time steps for each PL in the simulation is summarized in Figure 3.12. In 88.3% of all the time steps, PL 1 is used, because the network activity is low for most of the time. In 8.3% of the time steps, PL 2 is used. Only in 3.3% of the time steps, PL 3 is used. The DVFS feature allows the PE to run at PL 1 for most of the time while being able to switch to the higher PLs to guarantee real-time operation when the network activity is higher. To better illustrate when all the workload is processed, the time step of 1 ms is further divided into 10 sub-time steps. With the increase of the workload, the finishing time within the time step increases. If the workload would require more than 0.9 ms of a time step, the PL is increased to provide more computational power.

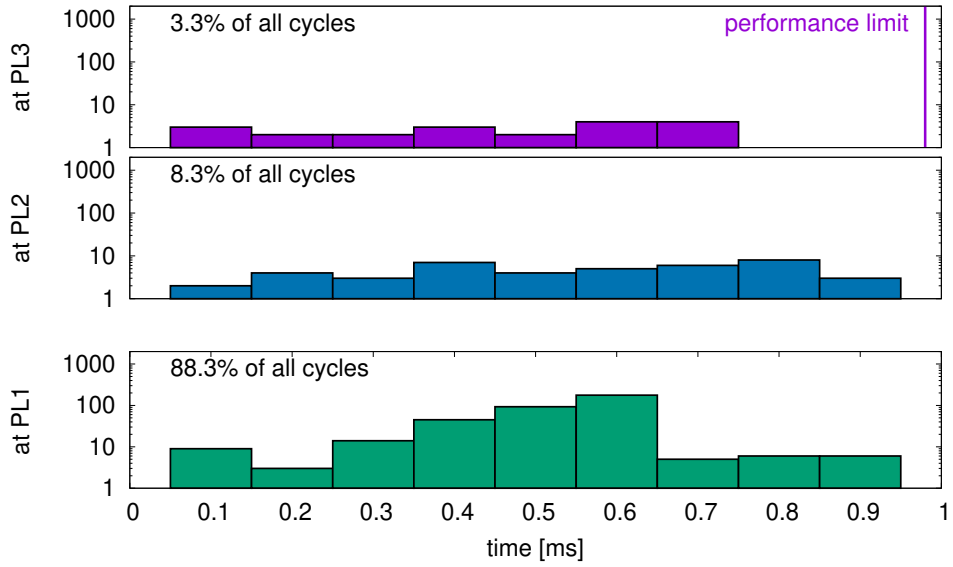

Figure 3.12: Histogram of simulation cycles (1 ms) processed at different PLs versus time

3.7 Conclusion

In this chapter, the benefit of DVFS for spiking neural network simulation is demonstrated. The DVFS technology which is commonly used in the semiconductor industry for dynamic power management is proved to be beneficial for the spiking neural network simulation with SpiNNaker 2 by allowing each PE to dynamically switch its supply voltage and clock frequency according to its own workload, which is implied by the number of incoming spikes.

To effectively exploit the potential of DVFS, the software flow of SpiNNaker 1 has been extended and power management strategies have been developed. The synfire chain network is used to demonstrate the benefit of DVFS, which shows 60.7% overall power reduction compared to the approach without DVFS. At the same time, this is also the first-ever software framework running a neural network model on SpiNNaker 2.

4

Reward-Based Synaptic Sampling Enables Learning in Edge Devices

4.1 Introduction

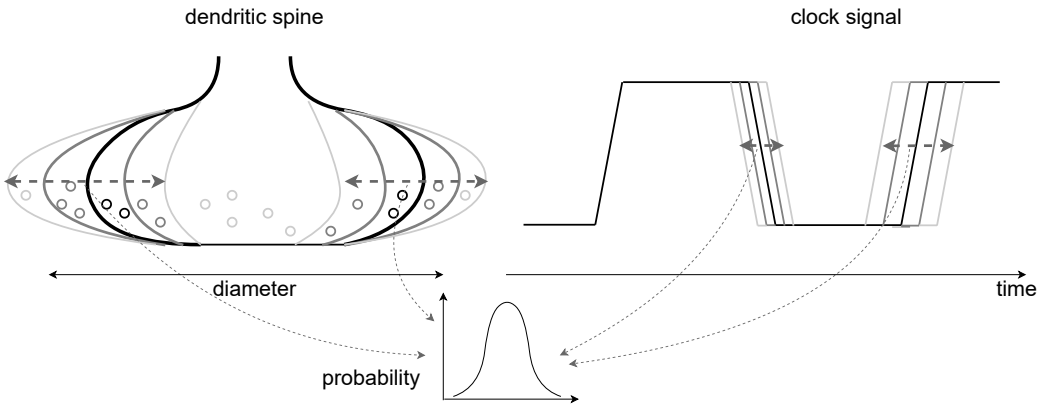

Figure 4.1: Left: as observed in biophysiological experiments, the size of a synapse changes randomly over time, causing random synaptic weight fluctuations. This randomness is believed to be involved in the stochastic computation employed by biology. The different darkness of the lines indicates the probability of the size of a synapse. Right: the rising and falling edges of a clock signal in a digital circuit. The clock signal is generated by the clock-generator. The deviation of the generated clock period from the ideal clock period is the jitter, which could change randomly over time. The different darkness of the lines indicates the probability of the rising and falling edges.

A large number of spiking neural network simulations are based on relatively simple neuron and synapse models like the ones used in the synfire chain network introduced in

the previous chapter. Because of this, neuromorphic hardware platforms that support this kind of neuron and synapse models are sufficient to simulate a variety of SNN models efficiently. However, recent advances in neuroscience indicate that neurons and synapses have much more complicated behavior than could be captured by these simple models, which put new requirements on the next generation neuromorphic hardware.

Observations in experimental neuroscience indicate that the synapses in biological neural networks change stochastically over time: the size and the efficacy of existing synapses fluctuate on the time scale of hours to days [46, 47] (Figure 4.1, left). In addition, new synapses emerge and existing synapses decline and disappear, i.e. the network is constantly rewiring itself. These random fluctuations are found to be caused by noise [48].

In machine learning, training a DNN with noise by adding the noise to gradient descent has been proved to help the network escape from local optima and thus increase robustness [49]. Similarly, computational neuroscientists propose that the brain exploits the noise which inherently exists for the exploration of parameter space, instead of suppressing it [50, 51].

One synapse model which takes noise into account is the synaptic sampling model [52, 53], where the rewiring and fluctuation of synaptic weight are affected by random noise. Although each synapse undergoes dynamic random fluctuations, the synaptic weights across the network can be proved to approach a stationary distribution. In addition, this stationary distribution can be shaped by reward and can be constrained to enforce sparsity. The synaptic sampling model well explains a number of observations in neuroscience [52, 54] and can be adapted to work with backpropagation in machine learning for memory-constrained training [55].

While the brain exploits the noise inherent in the neural substrate for parameter space exploration, in computing systems engineered with a silicon substrate, each building block is supposed to be reliable and deterministic whereas noisy behavior is normally undesired. In fact, random number generation in hardware has been a scientific problem. Over time, mathematicians have come up with algorithms to generate random numbers with the operations that are available in digital circuits, like XOR and shift [56, 57].

In addition, the brain also makes heavy use of the exponential function which is common in the physical world. Many models proposed in computational neuroscience

to capture the behavior of neurons and synapses include the exponential function, for example, various neuron models like the Hodgkin-Huxley model [32] and adaptive exponential integrate-and-fire model [58], and synaptic plasticity rules like Spike-Timing Dependent Plasticity (STDP) [59] and the synaptic sampling model [52, 53].

With the advances in computational neuroscience, more complicated computations such as exponential function and random number generation are required, which are often not supported in neuromorphic hardware or conventional CPUs and GPUs.

Neuromorphic hardware usually has very narrowly configurable plasticity functions unsuitable for this kind of models [15, 60, 61, 62]. Thus, synaptic weights that experience complex plasticity functions are usually precomputed in software and then run statically on mixed-signal [63, 64] or on digital neuromorphic hardware [65]. On the other hand, standard digital compute hardware is in principle flexible enough, but the functions required by the plasticity models are very expensive to compute on standard hardware which significantly narrows down the gain in efficiency. Despite recent efforts to simulate spiking neural networks on GPUs [66], there is no hardware support available for random number generation, especially true random number generation, and exponential function in GPUs. A common workaround on digital hardware is to store a massive amount of random numbers and look-up tables for the exponential function before the simulation starts [67]. This reduces computation time at the cost of increasing the requirements for the already limited memory of embedded applications.

SpiNNaker 2 strives to break the trade-off between computation time and memory by employing dedicated hardware components for these time- (and energy-) consuming operations with hardware accelerators for random numbers [26] and exponential functions [68], which allows for efficient implementation of complex learning algorithms. This potentially offers a new compute substrate especially for efficient learning in edge devices such as neural implants or Internet of Things (IoT) devices that are strictly limited by the power budget, computation speed and memory capacity of the silicon chip.

In this chapter, the benefit of the random number generator and exponential function accelerator is demonstrated with the reward-based synaptic sampling model. First, section 4.2 introduces the random number generator and the exponential function accelerator. Then, section 4.3 provides more details about the reward-based synaptic

sampling model. Third, section 4.4 presents the software implementation and optimizations. Fourth, section 4.5 presents the experimental results. Then, section 4.6 discusses scalability and comparison with other neuromorphic hardware. Finally, section 4.7 summarizes this chapter.

4.2 Hardware Accelerators for RNG and EXP

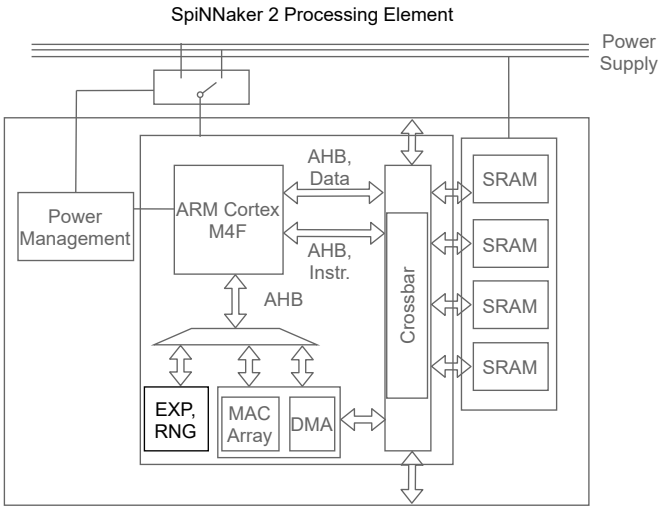

Figure 4.2: Hardware accelerators in the SpiNNaker 2 PE architecture.

The prototype chip (Figure 4.2) considered in this chapter is the first SpiNNaker 2 prototype [24, 25] which contains 4 PEs and is fabricated with the GLOBALFOUNDRIES 28 nm SLP CMOS technology. In addition, the prototype chip is connected with a dynamic random-access memory (DRAM) chip.

4.2.1 Random Number Generator

Two random number generators (RNG) were developed [26] for the SpiNNaker 2 prototype chip: the true random number generator (TRNG) and the pseudo random number generator (PRNG). The PRNG is a hardware implementation of the KISS random number generator [56], where the PRNG always outputs the same deterministic sequence of random numbers for the same initial seed. The output of the PRNG is a sequence of 32-bit integer values with uniform distribution. The hardware PRNG is implemented in a way that every clock cycle a new output can be generated and read out. Accessing

the random output from the ARM core takes 1 clock cycle, whereas simulating the same random number generation algorithm in software takes 35 clock cycles.

Since the output sequence of a PRNG depends only on the seed, the reproducibility is particularly suitable for debugging purposes. However, PRNG is only an approximation of true randomness and can not provide all the effects of randomness. In order to increase the quality of the randomness, the PRNG has the option to take outputs from the TRNG as its seed.

The TRNG makes use of the jitter that inherently exists in the clock-generators as the noise source (Figure 4.1, right) [26]. In this way, the noise source comes with minimum overhead regarding power consumption and chip area, since it is a byproduct of the normal operation of the clock-generators.

4.2.2 Exponential Function Accelerator

The exponential function accelerator developed in [27] calculates an exponential function with the signed fixed-point s16.15 data type. In the implementation, the operand is divided into three parts:

$$y = \exp(x) = \underbrace{\exp(n)}_{f_{\text{int}}(n)} \cdot \underbrace{\exp(p)}_{f_{\text{frac}}(p)} \cdot \underbrace{\exp(q)}_{f_{\text{poly}}(q)} \text{ with } x = n + p + q, \tag{4.1}$$

where n is the integer part, p and q are the upper and lower fractional parts, respectively. $f_{\text{int}}(n)$ and $f_{\text{frac}}(p)$ are calculated with two separate look-up tables (LUTs), and $f_{\text{poly}}(q)$ is a polynomial. The split into two separate LUTs considerably reduces the memory size and thus the silicon area compared to one combined LUT, by taking advantage of the properties of the exponential function. The split of the evaluation of the fractional part into a LUT and a polynomial reduces the computational complexity of the polynomial with minimum memory overhead. The look-up and the polynomial calculation are parallelized, resulting in a latency of four clock cycles for each exponential function. Writing the operand to the accelerator and reading the result from it via the AHB bus adds additional two clock cycles, resulting in 6 clock cycles in total. In pipelined operation the processor writes one operand in one clock cycle and reads the result of a previous exponential function in another clock cycle, resulting in two clock cycles per exponential function [27].

4.3 Reward-based Synaptic Sampling

The reward-based synaptic sampling model was introduced in [52, 53, 54]. The model describes the synaptic plasticity as a guided random walk, where the synaptic weight randomly changes while being guided by a reward signal.

In particular, for all the potential synaptic connections between neurons in a network, only a fraction of the connection is realized. The synapses change their weights continuously. When a synaptic weight goes below a predefined value, the connection is considered disconnected. On the other hand, for the potential connections where no connections exist, new connections could randomly emerge. In this way, the synaptic connections and weights in a network keep constantly changing, but over time, the synaptic weights across the network approximate a target distribution. Furthermore, the shape of the target distribution can be influenced by a reward signal.

In section 4.3.1 the neuron model considered in this algorithm is introduced, which differs from the standard LIF neuron. In section 4.3.2 the synapse model is introduced. Then in section 4.3.3, the reward-based synaptic sampling is introduced. Finally, in section 4.3.4, the random reallocation of synapse memory is introduced, which is developed to reduce the memory footprint of the algorithm on hardware.

4.3.1 Neuron Model

The neuron model considered in this chapter is a stochastic variant of the spike response model [31]. The membrane potential of neuron k at time t is given by

$$u_k(t) = \sum_{i \in \mathrm{SYN}_k} y_{\mathrm{PRE}_i}(t)\, w_i(t) + \vartheta_k(t), \tag{4.2}$$

where $w_i(t)$ is the synaptic strength of synapse i at time t, SYN_k denotes the set of all synapses projecting to neuron k, PRE_i denotes the index of the presynaptic neuron of synapse i, $\vartheta_k(t)$ denotes the bias of neuron k, and $y_{\mathrm{PRE}_i}(t)$ denotes the trace of the postsynaptic potentials (PSPs) in the synapse i caused by neuron PRE_i at time t, defined as the convolution of the spike train of neuron PRE_i $z_{\mathrm{PRE}_i}(t)$ and the PSP kernel $\epsilon(t)$ given by

$$\epsilon(t) = \Theta(t)\, \frac{\tau_r}{\tau_m - \tau_r} \left(e^{-\frac{t}{\tau_m}} - e^{-\frac{t}{\tau_r}} \right) \tag{4.3}$$

with time constants τ_m and τ_r, and the Heaviside step function $\Theta(t)$. Note that $y_{\text{PRE}_i}(t)$ only describes the unweighted trace. The effect of a presynaptic spike from the neuron PRE_i on the postsynaptic neuron over time is then described by $y_{\text{PRE}_i}(t)\, w_i(t)$. The shape of $\epsilon(t)$ with the values of τ_m and τ_r in Table 4.1 is shown in figure 4.3.

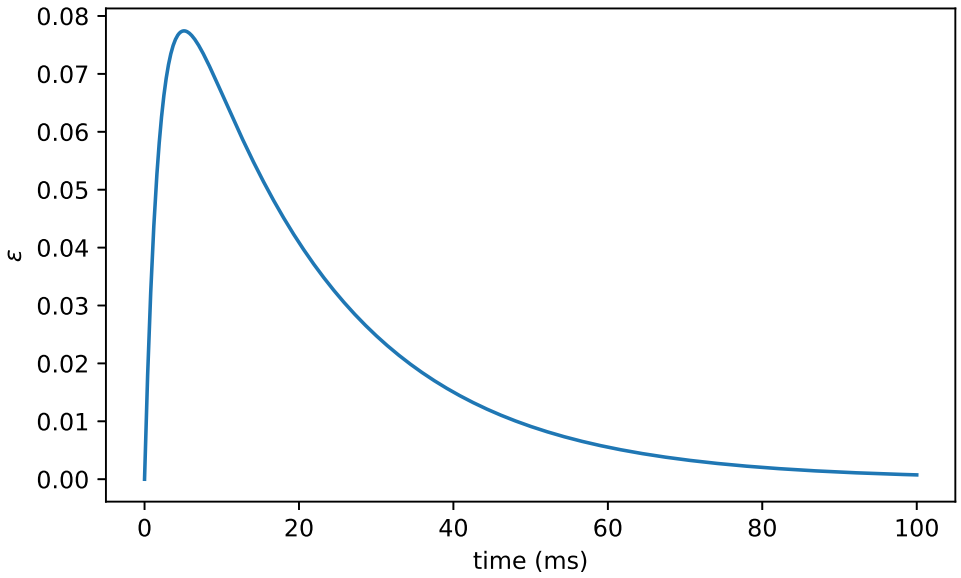

Figure 4.3: Illustration of the shape of $\epsilon(t)$ in equation 4.3.

The bias potential $\vartheta_k(t)$ in equation 4.2 evolves with time:

$$\tau_\vartheta \frac{d\vartheta_k(t)}{dt} = \nu_0 - z_k(t), \tag{4.4}$$

where τ_ϑ is a time constant and ν_0 is the target firing rate of the neuron.

In a normal LIF neuron model, a spike is generated when the membrane potential crosses a threshold. In this model, spikes are generated stochastically. The instantaneous rate of neuron k at time t has an exponential dependence on the membrane potential: $f_k(t) = \exp(u_k)$. Spikes are then generated from a Poisson process with rate $f_k(t)$. Similar to LIF neuron, there is a refractory period of t_{ref}.

4.3.2 Synapse Model

The synaptic strength w_i in equation 4.2 depends exponentially on the synapse parameter θ_i:

$$w_i \;=\; \exp(\theta_i - \theta_0) \tag{4.5}$$

where θ_0 is a positive offset parameter. When θ_i is smaller than 0, the connection is considered disconnected. In this model the plasticity only applies to excitatory connections where w_i is larger than 0.

The idea of synaptic sampling is that the dynamic development of synaptic weights approaches and samples from a target distribution $p^*(\boldsymbol{\theta})$ over synaptic parameters, where $\boldsymbol{\theta}$ is the vector of all synaptic parameters θ_i. It has been proved in [54] that the target distribution can be approached if the synaptic plasticity is defined by the stochastic drift-diffusion process

$$d\theta_i(t) \;=\; \beta \;\frac{\partial}{\partial \theta_i} \log p^*(\boldsymbol{\theta})\bigg|_t \; dt \;+\; \sqrt{2\beta T}\, d\mathcal{W}_i(t) \tag{4.6}$$

where β is the learning rate and $d\mathcal{W}_i$ models the Wiener process, controlled by the temperature parameter T. The target distribution is also scaled by the temperature parameter, such that the final stationary distribution of the synaptic parameters is proportional to $p^*(\boldsymbol{\theta})^{\frac{1}{T}}$.

The idea of reward-based synaptic sampling is that the target distribution $p^*(\boldsymbol{\theta})$ with an initial state or prior of $p_S(\boldsymbol{\theta})$ is shaped by the expected discounted reward $\mathcal{V}(\boldsymbol{\theta})$ such that

$$p^*(\boldsymbol{\theta}) \;\propto\; p_S(\boldsymbol{\theta}) \;\times\; \mathcal{V}(\boldsymbol{\theta}) \tag{4.7}$$

where $\propto$ means "proportional to". Under the assumption of a Gaussian prior with mean μ and variance σ^2, the log distribution of the prior with respect to parameter θ_i is given by

$$\frac{\partial}{\partial \theta_i} \log p_{\mathbb{S}}\left(\boldsymbol{\theta}\right) = \frac{1}{\sigma^2}\left(\mu - \theta_i(t)\right) \tag{4.8}$$

which essentially drives the synaptic parameter θ_i back to the mean μ.

4.3.3 Reward-based Synaptic Sampling

The last part of the model is about the "reward", or how the gradient of the value function $\frac{\partial}{\partial \theta_i} \log \mathcal{V}(\boldsymbol{\theta})$ is estimated by the instantaneous reward $r(t)$ and the network dynamics described by the eligibility trace $e_i(t)$.

The value function $\mathcal{V}(\boldsymbol{\theta})$ is calculated as the expected discounted reward

$$\mathcal{V}(\boldsymbol{\theta}) \;=\; \left\langle \int_0^\infty e^{-\frac{\tau}{\tau_e}}\, r(\tau)\, d\tau \right\rangle \tag{4.9}$$

where τ_e is a time constant and $\langle\,\cdot\,\rangle$ averages over all possible network activities.

To estimate the gradient of the value function $\frac{\partial}{\partial\theta_i}\log\mathcal{V}(\boldsymbol{\theta})$, first, the eligibility trace of a synapse i is considered to capture the neuronal activities:

$$\frac{de_i(t)}{dt} \;=\; -\frac{1}{\tau_e}e_i(t) \;+\; w_i(t)\, y_{\text{PRE}_i}(t)\,(z_{\text{POST}_i}(t) - f_{\text{POST}_i}(t))\;, \tag{4.10}$$

where τ_e is a time constant, PRE_i is the index of the presynaptic neuron and POST_i the index of the postsynaptic neuron for synapse i.

The gradient of the value function is estimated by the variable $g_i(t)$ with

$$\frac{dg_i(t)}{dt} \;=\; -\frac{1}{\tau_g}g_i(t) \;+\; \left(\frac{r(t)}{\hat{r}(t)} + \alpha\right) e_i(t)\;, \tag{4.11}$$

where τ_g is a time constant, α is a constant and $\hat{r}(t)$ is a low-pass filtered version of $r(t)$. The variable $g_i(t)$ acts as an online estimator for $\frac{\partial}{\partial\theta_i}\log\mathcal{V}(\boldsymbol{\theta})$ [54] and essentially describes how the synaptic parameter θ_i is driven by the gradient of the value function, which is described as a temporally averaged interaction between the instantaneous reward $r(t)$ and the neuronal activities described by the eligibility trace $e_i(t)$.

Finally, the update rule for the synaptic parameter θ_i is given by

$$d\theta_i(t) \;=\; \beta\left(\frac{1}{\sigma^2}\,(\mu - \theta_i(t)) \;+\; g_i(t)\right) dt \;+\sqrt{2\beta T}\, d\mathcal{W}_i(t)\;. \tag{4.12}$$

with parameter values summarized in Table 4.1.

To better illustrate the dynamics of the synaptic parameters, the change of the synaptic parameters over time is visualized for a simple network where a single neuron receives spikes from two input neurons (input 1 and input 2), and the reward signal shapes the target distribution so that synapse 1 should be connected and synapse 2 should be disconnected, as shown in figure 4.4. At first, both synapses are disconnected, i.e. θ is smaller than 0 (figure 4.4a). The synaptic parameters explore the parameter space randomly with the feedback of the reward signal. After exploring the area with a low reward for a while, the network finds the gradient which leads to a higher reward (figure 4.4b). After reaching the area with a higher reward, the network continues to

Table 4.1: Parameters of the neuron and synapse model Eqs. (4.2)-(4.12).

symbol	*value*	*description*
τ_r	2 ms	time constant of EPSP kernel (rising edge)
τ_m	20 ms	time constant of EPSP kernel (falling edge)
τ_e	1 s	time constant of eligibility trace
$\tau_\vartheta = \tau_g$	50 s	time constants for equations (4.4) and (4.11)
ν_0	5 Hz	desired output rate
t_{ref}	5 ms	refractory time
T	0.1	temperature
α	0.02	offset to reward signals
β	10^{-5}	learning rate
μ	0	mean of prior
σ	2	std of prior

explore the parameter space, and even tried to reconnect synapse 2 for a short time (figure 4.4c). But the lower reward drives the network to disconnect synapse 2 and continue exploration in other directions (figure 4.4d).

4.3.4 Random Reallocation of Synapse Memory

The rewiring scheme in the original synaptic sampling model requires that a disconnected synapse (i.e. when $\theta_i \leq 0$) continues to be simulated until the connection reappears. This would require all possible synapses to be simulated although only the connected synapses contribute to the network activity, which is a waste of computing resources.

An alternative solution is to randomly connect another synapse whenever one synapse disappears so that the total number of connected synapses remains the same, and only the connected synapses are simulated. Theoretical considerations have proved that this solution also leads to a stationary distribution of the synaptic parameters equivalent to the original approach [55]. In this work, when a synapse disappears, the resource is randomly reallocated to another postsynaptic neuron of the same presynaptic neuron due to the memory structure of the simulation software on SpiNNaker 2, so that the fanout of each neuron stays constant.

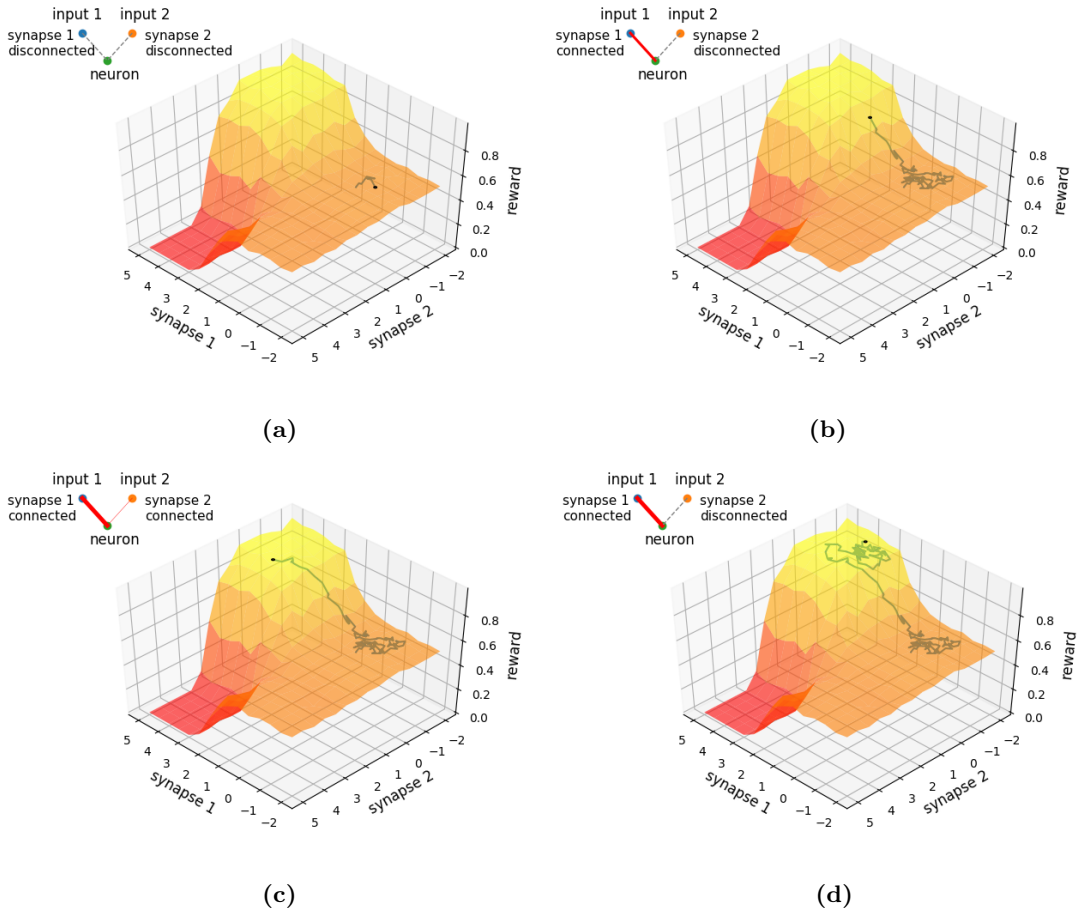

Figure 4.4: Dynamics of synaptic parameters of a simple network with 2 inputs and a single neuron (upper left). The strength of the synaptic connection is illustrated with the width of the red line. When the synapse is disconnected, it is shown as a dashed line. The two horizontal axes are the synaptic parameters for synapse 1 and synapse 2. The vertical axis is the reward. The synaptic parameters explore the parameter space stochastically guided by the reward signal.

4.4 Software Implementation and Optimizations

In order to bridge the gap between state-of-the-art biologically plausible neural models and efficient execution of the model in hardware, the software implementation of this model is optimized regarding computation time, memory, power consumption and scalability. This is explained in more detail in the following.

4.4.1 Reducing computation time with hardware generated uniform random numbers

The synaptic sampling model draws one random number for each synapse in each simulation time step (1 ms). Since thousands of synapses are simulated in each core [1], random number generation could dominate the computation time. As described in section 4.3, the Wiener process requires Gaussian random numbers to be generated. But as described in section 4.2.1, only uniform random numbers can be generated by the accelerator. As shown in Table 4.2, the generation of a pseudo Gaussian random number with Box-Muller transform [69] in software requires 172 clock cycles. One option could be to convert the hardware-generated uniform random number into Gaussian random number with Inverse CDF method [70] and look-up table, which reduces the computation time to 21 clock cycles.

Since the simulation of the Wiener process is often needed in Physics, the computational complexity was also a problem for the Physicians. Indeed, in the 1990s, Physicians in the Jülich research center encountered the same problem and reduced the computational complexity of the simulation by replacing the Gaussian random numbers with uniform random numbers. They also proved theoretically and empirically that the simulation results were not affected [71].

The generation of a uniform random number in software with Marsaglia RNG [56, 57] requires 42 clock cycles, whereas with hardware it takes only 5 clock cycles, including fetching the integer random number from the accelerator and converting it to floating-point type in the range of 0 to 1.

[1] A "core" is equivalent to a "PE"

Table 4.2: Computation time for random number generation and exponential function

Computation time for random number generation

Random number type	#clock cycles
Gaussian (software, Box-Muller Transform)	172
Gaussian (hardware, Inverse CDF, optimized)	21
Uniform (software, Marsaglia)	42
Uniform (Hardware)	5

Computation time for exponential function

Exponential function	#clock cycles
Software (floating point, Newlib)	163
Software (fixed point, hardware emulation)	104
Hardware (fixed point, precision not enough)	6
Hardware (conversion from and to float)	15

4.4.2 Reducing computation time with exponential function accelerator

In the synapse model, the parameter θ of each synapse accumulates small changes in each time step. The exponential function accelerator, which calculates the exponential function within 6 clock cycles (section 4.2.2), uses a fixed-point data type whose precision is not enough for this model because the change of θ would be rounded to zero. Calculating a floating-point exponential function with software libraries like Newlib takes 163 clock cycles. Since high precision is only necessary for storing the small change of θ, but not necessary for calculating intermediate variables like w, θ can be stored as floating-point in memory, and when calculating w with an exponential function, θ can be converted to fixed-point and calculated with the exponential function accelerator. The result is then converted back to a floating-point. Simulations show that the performance of the model is not affected. This reduces the computation time to 15 cycles with 6 cycles required by the hardware accelerator and 9 additional cycles for the conversion of data type. For the sake of comparison, emulation of exponential accelerator in software takes 95 cycles instead of 6 [27]. Thus, with the conversion of data type, this approach would take 104 cycles with software (Table 4.2).

4.4.3 Reducing memory footprint with 16-bit floating point data type

In order to simulate more synapses with limited memory, which is the case when the synapse parameters are stored in SRAM (see section 4.4.4), the single-precision floating point with 32 bits can be converted into half-precision floating point with 16 bits. For each synapse i, three parameters need to be stored in memory: *eligibility trace* e_i, *estimated gradient* g_i and *synaptic parameter* θ_i. Simulations show that converting e_i and g_i to half-precision does not affect the model performance.

4.4.4 Local Computation

By avoiding external DRAM access and instead storing all parameters and state variables of the model locally in SRAM, both energy and computation time can be saved.

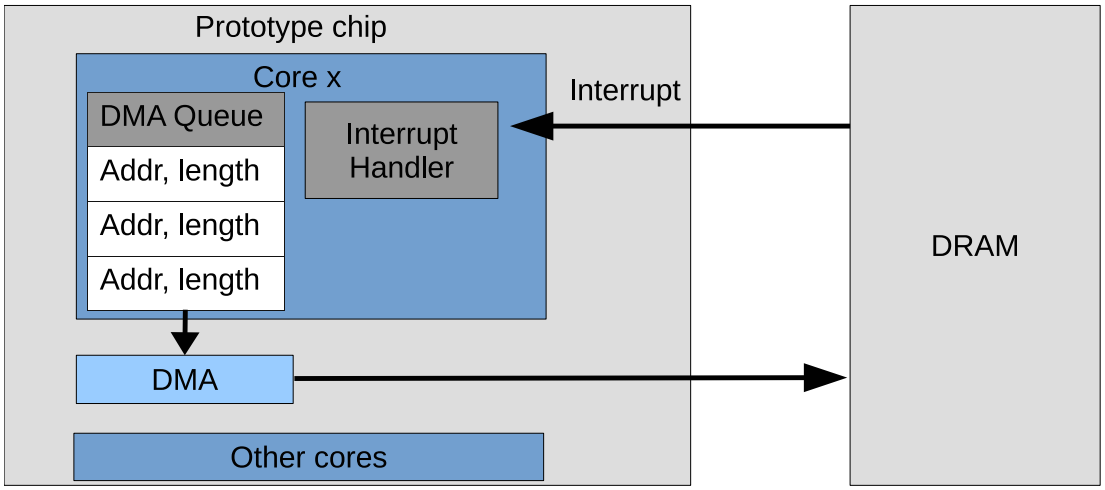

Figure 4.5: The time and energy-consuming interaction between the SpiNNaker 2 prototype chip and the DRAM chip, which could be saved by storing data locally in SRAM.

To read (write) data from (to) the off-chip DRAM, the core sends a read (write) request which is first stored in a Direct Memory Access (DMA) queue in software, then sent to the DMA unit, and at last sent to the DRAM. When the read (write) process is complete, an interrupt is triggered and an interrupt handler is called, which, in case of a read request, processes the data from DRAM. Then the next read/write request in the queue is sent to DMA (Figure 4.5). Since the DRAM access is time-consuming, the software can let DMA run in the background and continue with other tasks. When

the read/write process is complete, the core stops with the current task, handles the interrupt and then resumes the stopped task after the interrupt handler is complete. Although this saves computation time compared to waiting for the read/write process to complete, it still has the following drawbacks:

1. Retrieving all synapse parameters in each time step, which is necessary in this model, could easily saturate DRAM bandwidth especially in the scaled-up case with tens of cores per chip [72, 73].

2. The energy consumption of DRAM access can be two orders of magnitudes higher than SRAM access [74].

3. This only works if the other tasks are independent of the data being fetched.

4. Managing the DMA queue and calling the interrupt handler still consumes computation time, which becomes a problem when memory is frequently accessed.

The drawback when not using external DRAM is the limited memory space available in SRAM. This is not a problem for this model, since on the one hand the required memory is reduced with 16-bit floating-point, and on the other hand due to the complexity of the model, the number of synapses per core is limited by computation as is shown in section 4.5.2.

4.4.5 Memory Model

The memory model (Figure 4.6) is based on the software for the first generation SpiNNaker system [45]. The spike packet contains the ID of the presynaptic neuron. The master population table contains keys which are presynaptic neuron IDs. Each key is 4 bytes long and is stored together with the 4 byte starting address of the synapse parameters for the presynaptic neuron. These synapse parameters are stored in a contiguous memory block called synapse row. Each row is composed of 4-byte words. For each presynaptic neuron, the first word is the length of the plastic synapse region. In the implementation, the plastic synapse region consists of 8-byte blocks with 2 bytes for e_i, 2 bytes for g_i and 4 bytes for θ_i. After the plastic synapse region, there is one word for the length of the fixed synapse region. The next word is the length of the plastic control region which stores special parameters needed by the plasticity rules.

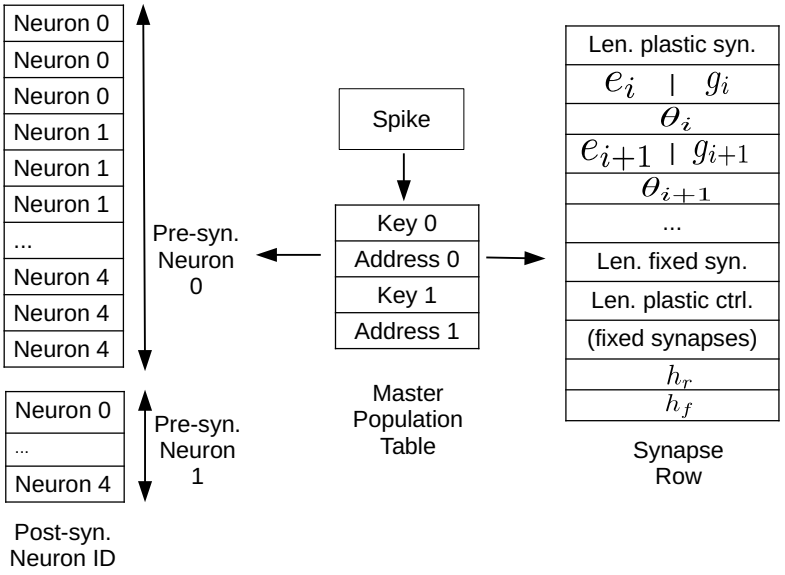

Figure 4.6: Memory model with master population table, synapse rows and postsynaptic neuron ID.

For synaptic sampling this region is used to store the parameters for the PSP kernel of input spike, e.g. h_r and h_f (corresponding to the time constants τ_m and τ_f). Since the PSP kernel of the incoming spike is the same for all synapses of the same presynaptic neuron, the parameters for the PSP kernel are shared in order to reduce memory footprint. After the word for the length of the plastic control region follow the parameters for fixed synapses.

The synapse parameters should also include the index of the postsynaptic neuron. One way to implement this is to add a 4-byte word for each postsynaptic neuron in addition to the 8 bytes for e_i, g_i and θ_i, which is the case in the original SpiNNaker software framework. Alternatively, since in this network all input neurons have the same fanout, the indexes are stored in a 2-d array (Post-syn. Neuron ID in Figure 4.6), where the column index stands for the presynaptic neuron ID and the entries represent the postsynaptic neuron IDs. Each entry represents a synapse and occupies one byte,

supporting maximum of 256 target neurons per core. Since multiple synapses are allowed between a pair of neurons, the ID of a postsynaptic neuron can appear multiple times in each column of the 2-d array. In general, depending on the application, one of the two approaches can be chosen.

The master population table, synapse rows and postsynaptic neuron ID are arrays generated by each core after the network configuration is specified. Each core generates its own data in a distributed way instead of having a centralized host PC generating data for all cores. This, combined with local computation (section 4.4.4), drastically reduces the time for data generation and transmission of data from host PC to chip, which could make up a significant amount of total simulation time especially in the case of large systems [75, 76].

4.4.6 Program Flow and SpiNNaker Software Framework Integration

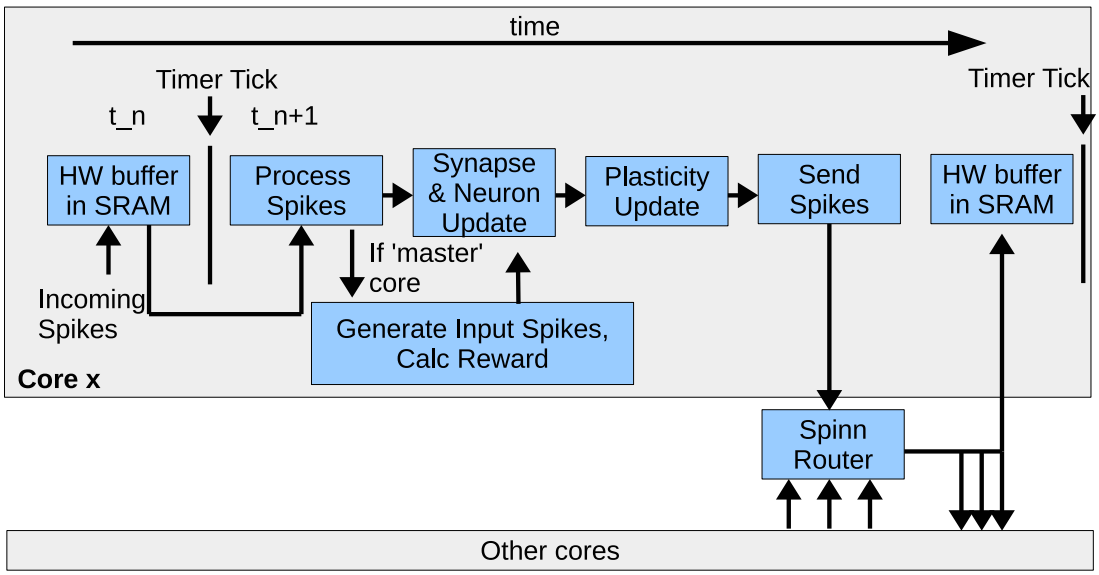

Figure 4.7: SpiNNaker software framework. Each simulation time step t_n is triggered by the timer tick interrupt. At the end of the time step, the spikes are sent to the SpiNNaker router which then multicasts the spikes to other cores.

The software framework for the simulation of synaptic sampling (Figure 4.7) is based on the SpiNNaker 1 software flow [45] and is similar to the DVFS software flow introduced in chapter 3. Since the computation is dominated by the plasticity update which is independent of the incoming events, DVFS is not necessary. Note that the

'Send Spikes' was part of the neuron update in chapter 3. In contrast, in Figure 4.7, it is done after the plasticity update, so it is shown separately. More details about the software flow are described in the following.

The timer tick signal of the ARM core is used to trigger each time step in real-time. The length of a time step can be arbitrarily chosen. For this implementation, one time step is one millisecond. The timer tick signal triggers an interrupt. Then the handler of the interrupt is called and processes the incoming spikes from the last time step, which are stored in a hardware buffer in SRAM. In this step, for each incoming spike, first, the starting memory address of its corresponding synapse parameters is found with binary search in the master population table, then the synaptic weights of the activated synapses in the synapse row are added to the synaptic input buffers of the target neurons.

For the network model implemented in this work (section 4.5.2), one of the cores, the "master core", then simulates the environment that computes the global reward signal. All cores continue with the synapse update and neuron update, which integrate the synaptic weight onto the membrane potential of the postsynaptic neuron. Next, the synaptic plasticity update is performed, as now all required information is available, i.e. incoming spikes, neuron states and global reward.

At last, the spikes of the neurons in each core are sent to the SpiNNaker router, which then multicasts the spikes to the cores containing the corresponding postsynaptic neurons. The SpiNNaker router [77] allows for fast multicast of small packets, which is key to efficient spike communication for many-core neuromorphic systems like SpiNNaker. The distributed computation, synchronization with timer tick and communication with the SpiNNaker router allows for scaling up the model implementation onto large systems consisting of millions of cores.

4.5 Measurement Results on Test Chip

In the following, first, it is shown how the hardware accelerators and numerical optimizations reduce the computation time for one plasticity update of the synaptic sampling model. Then, a network model that performs reward-based synaptic sampling on the SpiNNaker 2 prototype is implemented, for which the power and energy measurements are also provided.

4.5.1 Computation Time of Plasticity Update

Table 4.3: Number of clock cycles for plasticity update

	HW Accelerator	only Software
Random number generation	5	42
Exponential function	15	104
Rest	90	90
Total	110	236
(RNG + EXP) / Total	18%	62%

As shown in section 4.4.1 the generation of a uniformly distributed random number takes 5 clock cycles with hardware accelerator and 42 clock cycles with software. The floating-point exponential function with exponential accelerator and conversion of data type takes 15 clock cycles, whereas the same algorithm in software takes 104 clock cycles. The rest of the plasticity update of a synapse takes 90 clock cycles. In total, the plasticity update takes 110 clock cycles with hardware accelerators and the equivalent implementation with only software takes 236 clock cycles (Table 4.3). For this application, the hardware accelerators result in a speedup of 2 regarding the number of clock cycles. Considering the increase of clock frequency from 200 MHz in SpiNNaker 1 to 500 MHz in the current prototype chip, in total a speedup factor of 5 is achieved. In the plasticity update, the computation time for random number generation and exponential function reduced from 62% to 18%.

4.5.2 Network Description

Figure 4.8 illustrates the network topology and the mapping to the prototype chip. The network consists of 200 input neurons which are all-to-all connected to 20 neurons with plastic synapses. Multiple synapses between each pair of neurons are allowed. In this implementation, 3 synapses between each pair of neurons are initiated, resulting in 200 x 20 x 3 = 12000 plastic synapses. 2 spike patterns are encoded in the spike rate of the input neurons and are sent to the hidden neurons (Figure 4.9). The 20 hidden neurons

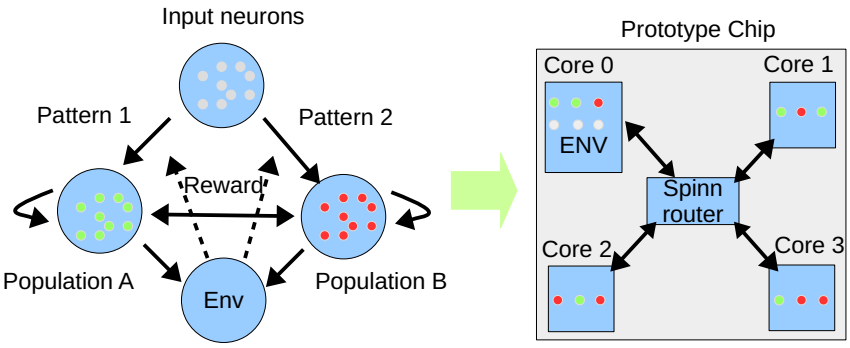

Figure 4.8: Illustration of the network topology (left) and its mapping to the prototype chip (right).

are divided into two populations (A and B). The output spikes of the hidden neurons are sent to the environment (Env), which evaluates the global reward. A high reward is obtained if input pattern 1(2) is present and the mean firing rate of population A(B) is higher than population B(A). The global reward is sent back to the network and shapes the plastic synapses between the input neurons and the two populations. The goal is to let the two populations 'know' which spike pattern they represent and signal this with a high firing rate when their pattern is present. In addition to the feedforward input, hidden neurons receive lateral inhibitory synapses that are initiated to fixed random weights between each pair of hidden neurons.

The network is mapped to the prototype chip with each core simulating 5 neurons from the two populations (see Figure 4.8). The first core ("master core") also generates the input spikes and evaluates the reward. The 200 input neurons lead to $200 \times 5 = 1000$ pairs of neurons in each core.

The profiling results in section 4.5.1 provide the computational aspect when assigning the number of synapses to simulate on each core. The ARM Cortex M4F core used in this prototype chip is configured to run at 500 MHz, which means 500 000 clock cycles are available in each time step (1 ms). The computation for one time step without plasticity update takes ca. 45 000 clock cycles for core 0 and 40 000 clock cycles for the other cores. Since each plasticity update takes 110 cycles with hardware accelerators and 236 cycles without hardware accelerators, the theoretical upper limit for the number of synapses per core is ca. 4100 with hardware accelerators and ca.

1900 without hardware accelerators.

In terms of memory, the prototype chip has 64 kB Data Tightly Coupled Memory (DTCM) per core, for all initialized data, uninitialized data, heap and stack. By checking the binary file size after compilation, the maximum number of synapses is estimated as 4 700. Thus, this model is limited by computation rather than memory (see Table 4.4).

Table 4.4: Maximum Number of Synapses per Core

	Memory Constraint	Real Time Constraint
With Accelerators	4 700	4 100
Without Accelerators	4 700	1 900

In the implementation, 3 000 plastic synapses per core are simulated, in order to ensure the stability of the software. Since 3 000 plastic synapses can be simulated in each core, each pair of neurons has 3 plastic synapses. Multiple synapses are allowed between each pair of neurons, which is a phenomenon observed experimentally in biological neural networks. This phenomenon has been suggested to allow more complex prior distributions [78][54]. Note that this is only the initial configuration. Due to random reallocation of synapse memory, the postsynaptic neuron could change, so that not every single pair of neurons has 3 plastic synapses.

4.5.3 Implementation Results

The usability of the network is demonstrated in a closed-loop reinforcement learning task implemented with 4 cores. The generation of input spikes and evaluation of output spikes are also implemented on the chip.

As shown in Figure 4.9, the 200 input neurons send two spike patterns in random order. Each spike pattern lasts for 500 ms. Resting periods of 500 ms are inserted between two pattern presentations, where the input neurons only send random spikes with a low firing rate representing background noise. The numbers at the top of Figure 4.9 and shaded colored areas indicate which pattern is present. As discussed above,

the 20 neurons are divided into 2 populations (A and B), each representing one of the two patterns. Neuron 1 to neuron 10 belong to population A, neuron 11 to neuron 20 belong to population B. In the second row of Figure 4.9, blue and green curves represent population firing rates of A and B, respectively. The firing rates were obtained with a Gaussian filter ($\sigma = 20$ ms) applied to the raw spike trains. The goal of learning is to let population A fire at a higher rate when pattern 1 is present and let population B fire at a higher rate when pattern 2 is present.

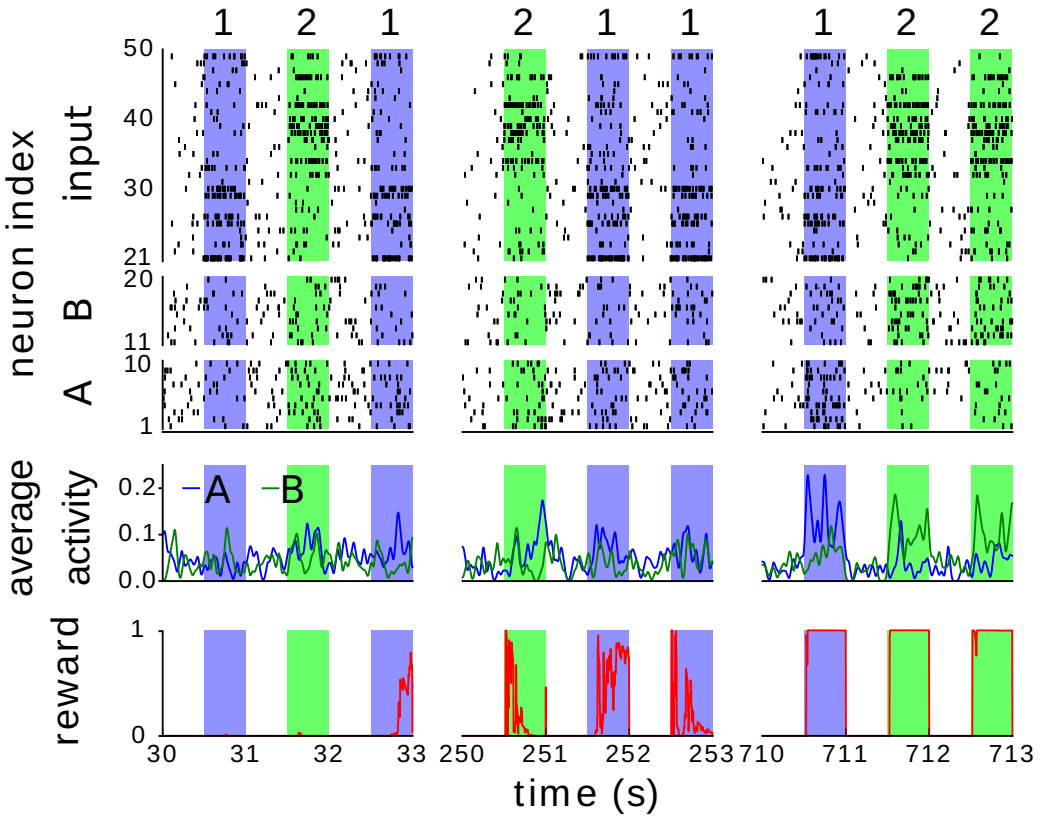

Figure 4.9: Network activity and reward throughout learning. Shaded areas indicate the presented patterns. Spike trains (top) of the two populations and input spikes. 30 neurons were randomly chosen from the 200 inputs.

Figure 4.10 shows the evolution of the mean reward with and without random reallocation of synapse memory (see section 4.3.4). The mean reward in each minute is low-pass filtered with a Gaussian kernel with $\sigma = 2$ min. Averages over 5 independent trial runs using the true random number generator are shown with solid lines, shaded areas indicate standard deviations. The reward is normalized to the theoretically max-

imum reachable reward. At learning onset, the two populations respond randomly to input spike patterns and the reward is low. The synaptic weights explore the parameter space with the random process guided by the global reward as described in section 4.3. Over time, the network learns the desired input/output mapping, and the reward increases. After ca. 10 minutes of training, the two populations learn to respond correctly to the two spike patterns with the firing rate of one population higher than the other when the corresponding spike pattern is present, and reward becomes high, which is also shown in the third row of Figure 4.9.

The results show that the reward increases much faster with reallocation due to the accelerated exploration of the parameter space. After the reward reaches a high value, the network continues exploration and the reward might fluctuate while the network searches for equally good network configurations.

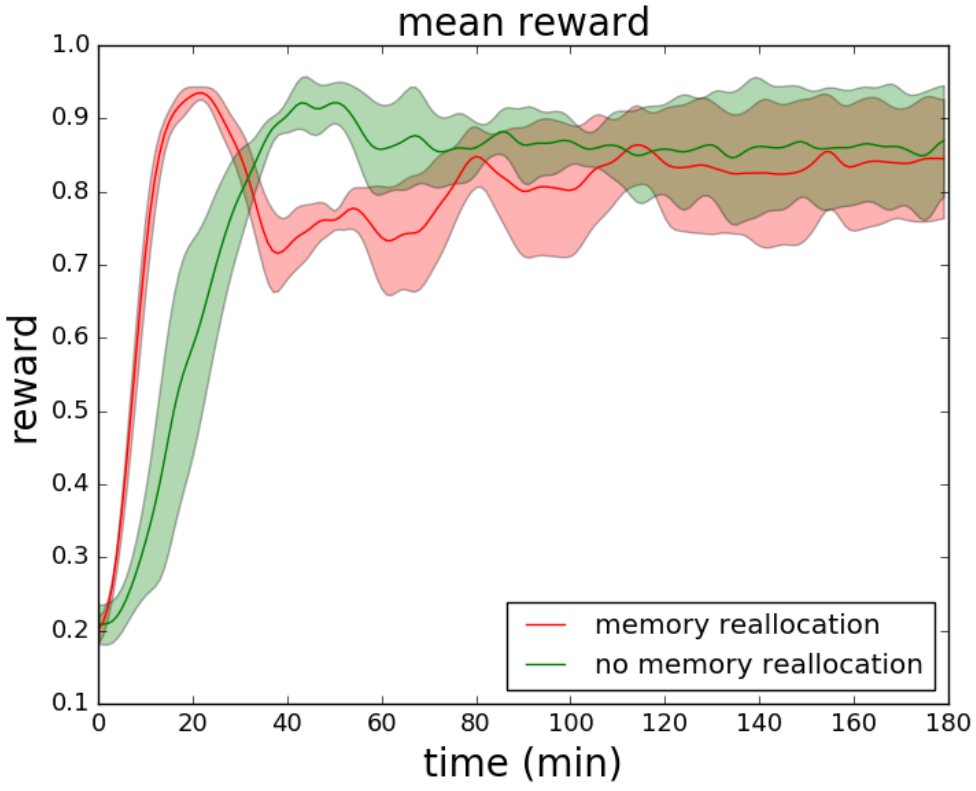

Figure 4.10: Time-averaged reward throughout learning for networks with (red) and without (green) random reallocation of synapse memory.

4.5.4 Power and Energy Measurement Results

Table 4.5: Power and Energy Consumption

Local Computation	No	Yes	Yes
Accelerator	No	No	Yes
Power (mW)	285	225	225
Time (ms)	1.58	1.58	0.76
Energy (μJ)	450.3	355.5	171
Reduction of Energy	0%	21%	62%

The optimizations described in section 4.4 result in considerable reduction of power and energy consumption. To show the benefit of the optimizations, power and energy consumption is measured in three cases. First, the synapse rows are stored in the external DRAM memory, and the exponential function and random number generation are done only with the software running on ARM core. Second, the synapse rows are stored in the local SRAM memory, and the exponential function and random number generation are still only done with the software running on ARM core. At last, the synapse rows are stored in the local SRAM memory, and the exponential function and random number generation are done with the hardware accelerators. For this measurement, the software is run without random reallocation of synapse memory. As summarized in Table 4.5, the power and energy consumption is reduced by local computation without external DRAM and reduction of computation time.

First, the memory footprint is optimized by employing a 16-bit floating-point data type and the compact arrangement of the memory model described in sections 4.4.1 to 4.4.5. The random reallocation described in section 4.3.4 increases the effective number of synapses which is otherwise only achievable with external memory like DRAM. The reduction of memory footprint allows for local computation with SRAM, as described in section 4.4.4. Switching off DRAM allows for a reduction of power consumption by 21%, from 285 mW to 225 mW.

In addition, as summarized in section 4.5.1, the computation time for each plasticity update is reduced by 53.4%. Without the hardware accelerators, simulating the network

with 3 000 plastic synapses per core for one time step (1 ms) takes 1.58 ms, losing the real-time capability. With the hardware accelerators, the simulation of one time step is finished within 0.76 ms. To measure the energy consumption, the length of the time step is chosen to be the minimum required for each time step to finish, i.e. 1.58 ms for without accelerators and 0.76 ms for with accelerators. The reduction of computation time for plasticity update reduces the energy consumption for one time step by 51.9%, from 355.5 μJ to171 μJ .

In total, the energy consumption for the simulation of the network for one time step is reduced by 62%, from 450.3 μJ to 171 μJ, making the system attractive for mobile and embedded applications.

4.6 Discussion

In this section, first the scalability of the implementation of the reward-based synaptic sampling model for larger networks on the final SpiNNaker 2 system is considered. Then, the possibility to realize this network model on SpiNNaker 1 and other neuromorphic platforms with learning capabilities is discussed.

4.6.1 Scalability

Although the synaptic sampling model here is only implemented on the 4 cores that are available on the SpiNNaker 2 prototype chip, the implementation is done with scalability in mind. The scalability is mainly supported by

- the distributed computation concept of SpiNNaker, where each core computes a part of the network
- the efficient multi-cast communication infrastructure
- the scalable software framework

Specifically for this model, the scalability is further guaranteed by the local computation, so that no data transfer between the cores and the DRAM is necessary, which could potentially become a communication bottleneck for larger networks.

In addition, the higher-level software stack, which is not included in this work, provides the necessary functionalities like partitioning and mapping a large network

into smaller parts based on the computational resources of the hardware, and the generation of routing tables for the SpiNNaker router. The higher-level software stack is generically written and could be ported from SpiNNaker 1 to SpiNNaker 2 with minimum effort.

4.6.2 Comparison with SpiNNaker 1

Reward-based learning and structural plasticity have been implemented on the SpiNNaker system before [72] [79]. The reward-based synaptic sampling model implemented in this work is more complex because of the need for random number generation and exponential function for each plastic synapse in each time step. In addition, due to the lack of floating-point arithmetic, this synapse model would be very hard, if possible at all, to be implemented in the first generation SpiNNaker system, since the change of synaptic weight is very small in each time step and can not be captured by the precision of fixed-point format.

4.6.3 Comparison with other neuromorphic platforms

The implementation of the reward-based synaptic sampling model on other neuromorphic platforms would also be almost impossible, due to the complexity of the model and the available flexibility of other neuromorphic platforms.

Fist of all, for neuromorphic hardware with only static synapses, such as TrueNorth [18], NeuroGrid [80], HiAER-IFAT [81], DYNAPs [82] and DeepSouth [83], this model cannot be implemented directly. The workaround would be to simulate the model on the hardware, collect the simulation data, do the synapse update on the host PC, and update the synapses on the hardware.

Secondly, exact implementation of the synaptic sampling model seems infeasible on neuromorphic hardwares with configurable (but not programmable) plasticity, like ROLLS [84], ODIN [85] and TITAN [86] (see [87] and [88] for reviews). However, it might be possible to realize simplified, approximate versions of synaptic sampling on these neuromorphic platforms.

Finally, architectures that do support synaptic plasticity on the chip, such as Loihi[17] and the BrainScales 2 system[14], have so far quite limited weight resolutions (9-bit signed integer on Loihi and 12-bit on BrainScales 2). Since the 32-bit fixed-point format was found to be insufficient for this model, it is questionable, even

with stochastic rounding, whether synaptic sampling can be implemented with such low weight resolution, and at what cost in performance. Also, in the case of Loihi, the size of the microcode that is allowed for computing synaptic updates is quite limited (e.g. 16 32-bit words). Besides, hardware accelerators for complex functions like the exponential function are not available on these two platforms, which makes the implementation more challenging, especially in the case of Brainscales 2, because the high data rate caused by accelerated operation requires fast execution of learning rules. These restrictions put some doubt on whether complex learning mechanisms, as the one considered here, can be implemented exactly.

4.7 Conclusion

In this chapter, a reward-based synaptic sampling model is implemented in the prototype chip of the second-generation SpiNNaker system. This real-time online learning system is demonstrated in a closed-loop online reinforcement learning task.

While the model provides a theoretical framework that nicely explains biophysiological findings, the computational complexity makes it one of the most complex synaptic plasticity models ever implemented on neuromorphic hardware. The difficulties in the implementation due to this complexity is solved by optimizations on the algorithm, software and hardware level, including the random reallocation of synapse memory, local computation, numerical optimizations and the use of the random number generator and the exponential function accelerator, which together lead to optimized computation time, memory and energy consumption.

In particular, in terms of computation time, the number of clock cycles is reduced by a factor of 2 compared to a pure software implementation. In addition, the energy consumption is reduced by 62% compared to implementation without the use of hardware accelerators and with external DRAM.

5

Hybrid SNN DNN operation

5.1 Introduction

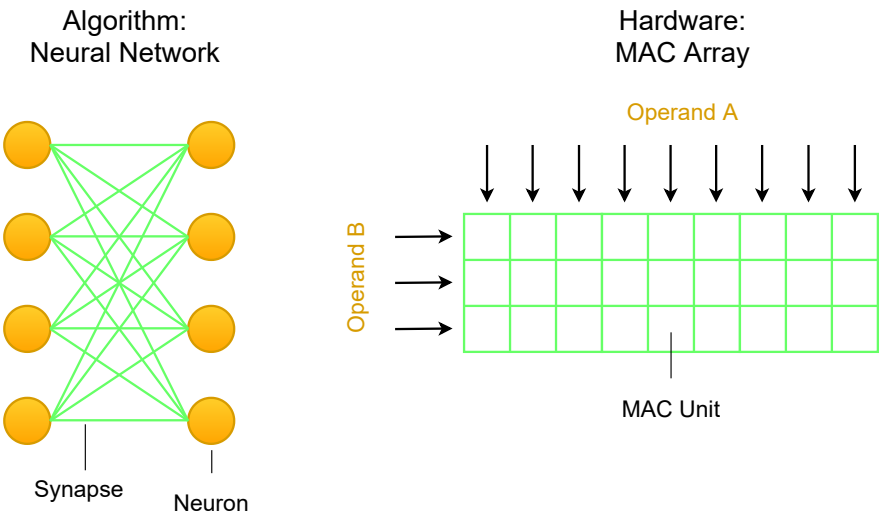

Figure 5.1: Left: connections in a densely connected layer in a DNN, the computation of which requires vector-matrix multiplication, where the vector contains the output values of the neurons of the previous layer, and the matrix contains the weights that the output values need to be multiplied with. Right: MAC array typically found in machine learning accelerators for efficient execution of this kind of vector-matrix multiplications.

In chapter 3 and 4, DVFS, the exponential function accelerator and random number generator of SpiNNaker 2 are demonstrated with the synfire chain network and the reward-based synaptic sampling model. In this chapter, the benefit of the MAC array is demonstrated. While in the previous chapters the applications in neuroscientific models

are shown, in this chapter, the applications are mainly in the DNN and hybrid network (e.g. neural engineering framework) domain, expanding the range of applications that SpiNNaker 2 is suitable for.

With the substantial progress of artificial intelligence (AI) in recent years, neural network-based algorithms are increasingly being deployed in embedded AI applications. Smart speakers which continuously listen for keywords like "Alexa" and robotic applications which employ neural network-based adaptive control algorithms are examples from industry and research. To improve the efficiency regarding power consumption and computation time, various hardware architectures have been proposed.

The neural networks employed in these AI applications are most commonly deep neural networks (DNNs). A substantial amount of computation in DNNs is caused by the multiply-accumulate (MAC) operations. For the efficient computation of DNNs, many machine learning hardware architectures include a MAC unit to facilitate the MAC operations in DNNs [89] (Figure 5.1).

At the same time, neuromorphic hardware supporting spiking neural networks (SNNs) is increasingly gaining attention. And the application of SNNs and hybrid networks has been an active area of research. Two prominent examples of neuromorphic hardware are Loihi from Intel and SpiNNaker 2 from TU Dresden and the University of Manchester. While Loihi has dedicated circuits for synapses and neurons, which increases the efficiency for the implemented models, and a programmable learning engine for more flexibility for various learning rules, SpiNNaker 2 uses general-purpose processors (ARM cores) connected with numerical accelerators. While the processor increases the flexibility of the synapse and neuron models and learning rules, the accelerators increase the efficiency for certain computations like exponential function and random number generation which are often required in neuromorphic applications. Besides the neuromorphic accelerators, SpiNNaker2 also contains MAC arrays for efficient matrix operations and is thus able to merge SNN and DNN operations.

In this chapter, the implementation of the keyword spotting and adaptive control benchmark tasks on the second SpiNNaker 2 prototype [2] is shown. While the keyword spotting network demonstrates SpiNNaker 2's suitability for the implementation of DNN, the adaptive control benchmark is realized with the neural engineering framework (NEF), which has vector-matrix multiply as input processing but can have spiking neurons as the nonlinear activation. Because of this, the NEF can be considered as

a hybrid network combining the perspectives from DNN and SNN. In addition, the computation time and active energy consumption of the benchmark tasks are compared with Loihi, and the results of the comparison highlight the benefit of the MAC array. Specifically, for keyword spotting, because the original DNN version is implemented on the SpiNNaker 2 prototype with the MAC array, and the SNN version is implemented on Loihi because it only supports SNN, the SpiNNaker 2 prototype shows better efficiency regarding computation time and energy consumption. For adaptive control, the spiking neuron model is used as the activation function on both hardware platforms. Loihi shows better efficiency when low dimensional vector-matrix multiplication is involved, and the SpiNNaker 2 prototype shows better efficiency when high dimensional vector-matrix multiplication is involved.

In section 5.2 first the MAC array is introduced. Section 5.3 describes the keyword spotting benchmark, its mapping strategy onto hardware and the software implementation. Section 5.4 describes the adaptive control benchmark, its mapping strategy onto hardware and the software implementation. Finally, the experimental results are presented in section 5.5.

5.2 MAC Array

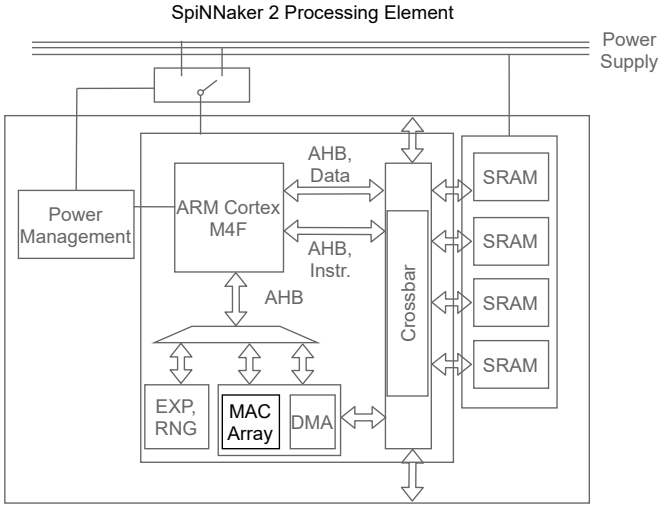

Figure 5.2: MAC array in the SpiNNaker 2 PE architecture.

The prototype chip considered in this chapter is the second SpiNNaker 2 prototype

[2], which is the same as in chapter 3. While in chapter 3 the main focus was the benefit of DVFS, in this chapter, the main focus is the benefit of the MAC array.

The MAC array has communication channels to the ARM core and the SRAM (Figure 5.2) and has 64 MAC units in a 4 x 16 layout. Figure 5.3 illustrates the MAC array. The data of operand A and operand B are arrays of 8-bit integer values. In each clock cycle, 16 values from the array of operand A and 4 values from the array of operand B are fed into the MAC array. Every MAC unit in the same column is fed with the same value from operand A, and every MAC unit in the same row is fed with the same value from operand B. The software running on the ARM core is responsible for arranging the data in the SRAM and notifying the MAC array of the address and length of the data to be processed. After the data is processed, the results are written back to predefined addresses in the memory. The result of each MAC unit is 29-bit.

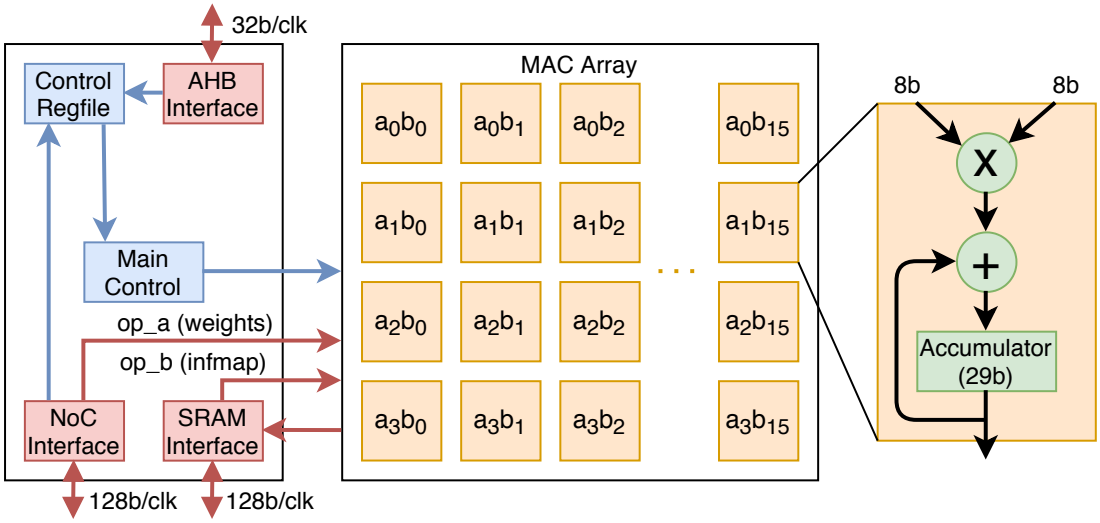

Figure 5.3: Schematic of the MAC array. Each square in the 4 x 16 block represents one MAC unit. The squares around the block represent the data to be executed. In each clock cycle, 4 values from operand B and 16 values from operand A are fed into the MAC array simultaneously. (Figure from [2], reused with permission)

When computing a matrix multiplication, a general-purpose processor like the ARM core needs to:

1. fetch the operand A and operand B into the registers
2. do the multiply-accumulate

3. write the result back
4. check the condition of the loop
5. compute the addresses of the data in the next iteration

While the MAC array essentially does the same, it is more efficient due to the Single Instruction Multiple Data (SIMD) operation. In particular, the efficiency is made possible by:

1. 64 MAC operations can be done in one clock cycle in parallel.
2. 16 x 8 bits of data of operand A and 4 x 8 bits of data of operand B can be fetched in one clock cycle in parallel
3. control logic and data transfer in parallel to MAC operations, hiding the overhead of data transfer for the next iteration.

5.3 Keyword Spotting Model and Implementation

In this section, the keyword spotting model, its mapping strategy onto the hardware and the software implementation are introduced.

5.3.1 Model Description

Keyword spotting is a speech processing problem which deals with identifying keywords in utterances. A practical use case is the identification of wake words for virtual assistants (e.g. "Alexa"). Here, the keyword spotting network that is implemented on the SpiNNaker 2 prototype is the same as in [90], which consists of 1 input layer with 390 input values, 2 dense layers each with 256 neurons and 1 output layer with 29 output values (Figure 5.4). Also, the same as in [90], no training is involved and only inference is considered. The 390-dimensional input to the network is the Mel-frequency cepstral coefficient (MFCC) features of an audio waveform in each time step. The 29-dimensional output of the network basically corresponds to the alphabetical characters, with additional special characters for e.g. silence etc. One 'inference' with this network involves passing 10 time steps of the MFCC features into the network. The outputs are then post-processed to form a result for the inference. The difference

to the implementation on Loihi is that on the SpiNNaker 2 prototype, the network is implemented with normal DNN with ReLU activations, whereas on Loihi, the SNN version was implemented since Loihi only supports SNNs.

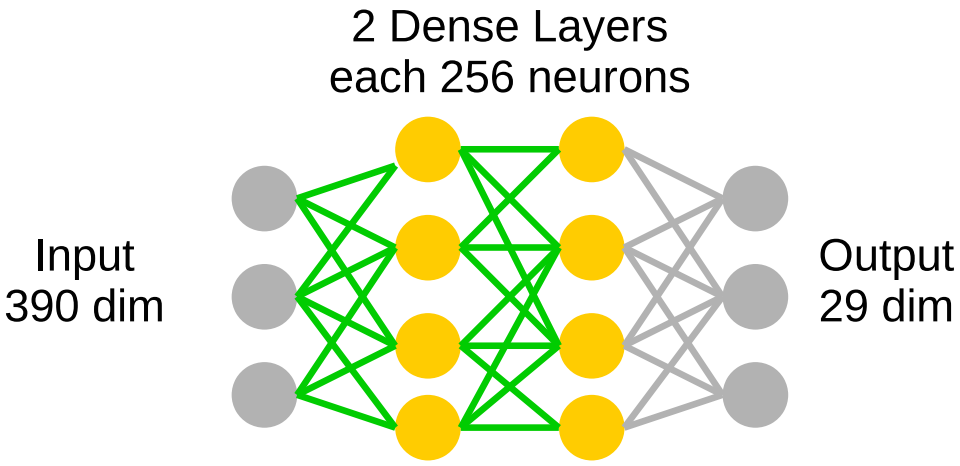

Figure 5.4: Keyword Spotting Network Architecture

5.3.2 Mapping Strategy and Software Implementation

The keyword spotting benchmark is implemented on the SpiNNaker 2 prototype with the MAC array and ARM core responsible for different computational tasks. Since the same benchmark has also been implemented on Loihi [90], this allows side-by-side comparison between both neuromorphic hardware platforms.

The keyword spotting network consists of 2 computational steps: vector-matrix multiplication and ReLU. The vector-matrix multiplication can be efficiently processed with the MAC array. Because of the flexibility of the ARM core, arbitrary neuron models can be implemented, and ReLU update is processed the ARM core.

Because of memory constraints (see Section 5.5.1) layer 1 is split into 2 PEs. The weights in this network are the same as in [90]. The input to the network is a 390-dimensional vector of 8-bit integers. The ReLU activations of each layer are also 8-bit integers. The ReLU activations of layer 2 are directly sent back to the host PC, where the vector-matrix multiplication for the output layer with 29 dimensions is performed, the same as in [90]. Figure 5.5 shows the implementation of the keyword spotting network on the SpiNNaker 2 prototype.

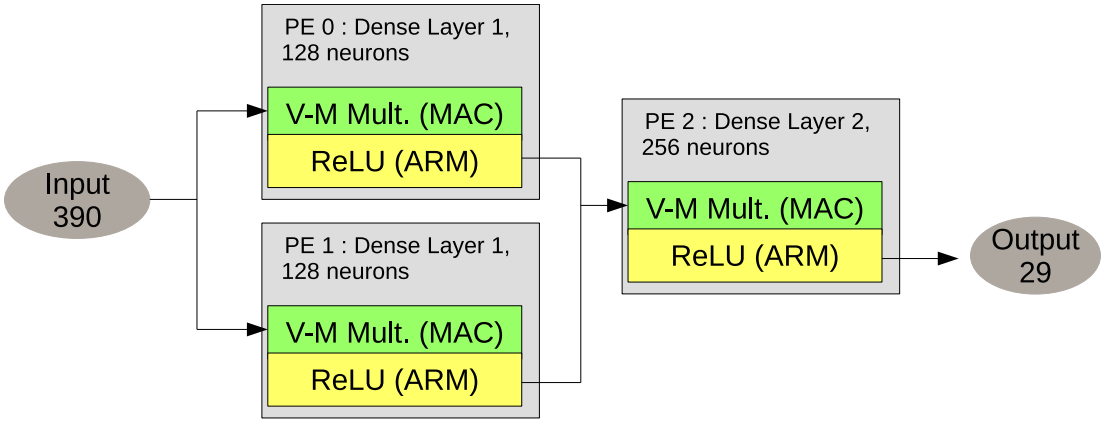

Figure 5.5: Implementation of keyword spotting network on the SpiNNaker 2 prototype

5.4 Adaptive Control Model and Implementation

In this section, the adaptive control model, its mapping strategy onto the hardware and the software implementation are introduced.

5.4.1 Model Description

The adaptive control model considered in this chapter is a combination of the adaptive control algorithm proposed by Jean-Jacques Slotine [91] and the neural engineering framework [92]. The details of the algorithm was described in [93] and used as a benchmark in [94] and [95].

As shown in Figure 5.6, the input to the neural network is the system states of the system that the control algorithm tries to control. The system states could be the position and velocity of a robot arm, in the case that the system to be controlled is a robot arm. But in general, the system to be controlled could also be more abstract. The neural network is created within the software framework of NEF. The input weight matrix is a dense connection and the neuron model can be LIF, ReLU or other models. In this work, LIF is chosen for a direct comparison with Loihi. The output of the neural network is a part of the control signal which is to be combined with the output of a PD controller to build the final control signal for the system to be controlled. The input signal to the PD controller is the desired system states and the actual system states. The output of the PD controller also acts as a feedback error signal for the neural network which takes part in the online weight update of the output weights.

The weight update of the output weights of the neural network is

$$\Delta\omega_{ij} = \alpha a_i E_j \tag{5.1}$$

where α is the learning rate, a_i is the output activity of the ith neuron, and E_j is the jth dimension of the error signal, which is the output from the PD controller. In the case of a robot arm, the first dimension could be the position and the second dimension could be the velocity.

Although this update rule takes a form that is reminiscent of back-propagation that is commonly used in machine learning, it is actually derived from control theory. In fact, it can be rigorously proved that this update rule drives the system to be controlled towards equilibrium [91].

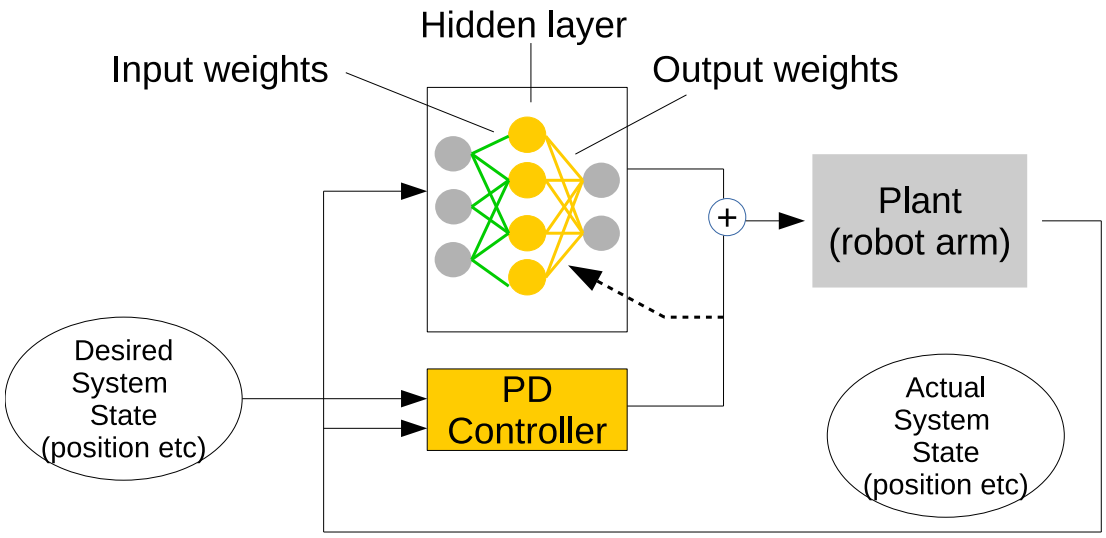

Figure 5.6: Adaptive Control Network Architecture

5.4.2 Mapping Strategy and Software Implementation

The adaptive control benchmark is implemented on the SpiNNaker 2 prototype with the MAC array and ARM core responsible for different computational tasks. Since the same benchmark has also been implemented on Loihi [95], this allows side-by-side comparison between both neuromorphic hardware platforms.

The implementation of adaptive control on the SpiNNaker 2 prototype is based on [96] and [97]. There are mainly 4 computational steps: input processing, neuron update, output processing and weight update. The input processing step is a vector-matrix multiplication, which can be efficiently processed on the MAC array. The rest of the steps are processed on the ARM core. In particular, in the output processing and weight update steps, the flexibility of the ARM core enables event-based processing, i.e. the computation is only done when there is a spike, which is not possible in SIMD operations.

In input processing, the inputs to the network are multiplied with the input weight matrix to produce the input current for each neuron in the hidden layer. The weights are quantized to 8-bit integers with stochastic rounding. The vector-matrix multiplication with only the ARM core and without the MAC array is also implemented and serves as a reference.

The rest of the computation is implemented on the ARM core which allows event-based processing.

In neuron update, the neuron dynamics is updated according to the input current. The Leaky-Integrate-and-Fire (LIF) neuron model is used in the hidden layer to allow for event-based processing of the spikes in the following steps.

In output processing, the outputs of the neurons are multiplied by the output weight matrix. In the case of non-spiking neuron models like ReLU, this process is a vector-matrix multiplication. In the case of spiking neuron models, a connection is only activated when there is a spike, so this output processing step corresponds to adding the weights associated with the neuron which has spiked to the output of the network.

In weight update, the output weight matrix is updated according to the neuron activity and error signal. In order to do weight update in an event-based manner, the low pass filter for the output activity has been removed, similar to [97]. Because of the short time constant of the low pass filter in this application, this modification doesn't affect the performance. Since the learning rate is normally very small, floating-point data type is chosen for the weights in the output weight matrix.

The adaptive control network is implemented on a single PE. The implementation is done with scalability in mind. In the case that the size of a neuron population exceeds the memory limit of a PE, it can be split into many PEs [96]. In addition, the PE also simulates the PD controller. The overhead is negligible.

The computational steps and the hardware component used for each step are summarized in Figure 5.7. The PD controller is not shown since the computation is relatively simple.

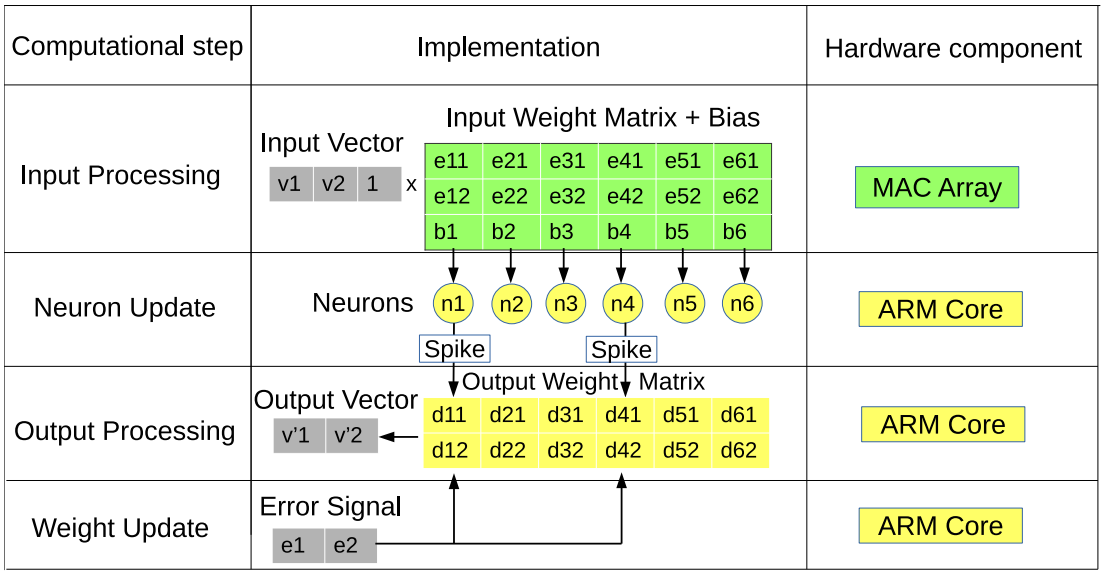

Figure 5.7: Main computational steps and hardware component for each step in adaptive control

5.5 Measurement Results and Comparison with Loihi

In this section, the results of both benchmarks running on the SpiNNaker 2 prototype chip are shown. In particular, results are shown regarding the memory footprint, computation time and energy measurement when the PE is running with 0.5 V and 250 MHz. The results of computation time and energy measurement are compared with Loihi. In addition, for adaptive control, the SpiNNaker 2 prototype chip is connected to a robotic arm to demonstrate real-time control. Since the same models are implemented on the SpiNNaker 2 prototype and on Loihi, the differences between both hardware platforms in terms of classification accuracy in the case of keyword spotting and mean squared error between actual and desired trajectories in the case of adaptive control are negligibly small, so that this will not be further discussed in this section. Since for both benchmarks, there is not much data movement between the PEs, the throughput of the NoC is not a bottleneck.

5.5.1 Keyword Spotting: Memory Footprint

For the keyword spotting benchmark, the required SRAM memory mainly consists of 2 parts: weight memory and neuron input memory.

The weight memory is the memory for storing the weights and biases, which are quantized as 8-bit integers. The required memory in bytes is

$$M_w \;=\; (D+1)N \tag{5.2}$$

where D is the number of input dimensions, N is the number of neurons.

The neuron input memory is the memory for storing the results from the MAC array after the vector-matrix multiplication is complete. Each input is a 32-bit integer. The required memory in bytes is

$$M_i \;=\; 4N \tag{5.3}$$

Since the ReLU unit doesn't need to hold its output value between inferences, which is the case for the LIF neuron model, there is no neuron memory needed.

The total memory for a neural network on a PE is

$$M_{total} \;=\; M_w + M_i \tag{5.4}$$

Based on equations (5.2), (5.3), (5.4) for memory footprint, the first hidden layer of the keyword spotting network would require ca. 100 KBytes of memory. For each PE, in total 128 KBytes of SRAM memory is available, which is used for the program code as well as the program data. In this work, it is assumed that each PE has 90 KBytes of SRAM memory available for the data of the neural network. So the first hidden layer is split into two PEs.

5.5.2 Keyword Spotting: Computation Time and Comparison with Loihi

In the keyword spotting benchmark, the number of clock cycles for the vector-matrix multiplication (C_{mm}) and the ReLU update (C_{relu}) are measured. After the measurement, polynomial models can be fitted by minimizing the mean-squared error. Here simple linear models have been adopted to capture the behavior of the system depending on the model parameters. It is found that these models are accurate enough for

the parameter range considered here. Since the parameters are limited by the hardware constraints such as SRAM memory anyways, adopting more complicated models doesn't seem to be necessary for the benchmarks. The number of clock cycles for the vector-matrix multiplication with the MAC array is found to be

$$\begin{aligned} C_{mm} &= 74.0 + 5.38N \\ &\quad +0.13ND + 24.0D \end{aligned} \tag{5.5}$$

where N is the number of neurons and D is the number of input dimensions. The time for the vector-matrix multiplication is mostly reflected in $0.13ND$. Before the vector-matrix multiplication starts, the inputs to the network need to be prepared for the MAC array. This pre-processing step is mostly reflected in $24.0D$. After the vector-matrix multiplication, a post-processing step is necessary for the resulting neuron input current. The computation time depends on both D and N, and this is reflected in $24.0D$ and $5.38N$. For each of the computational steps, there is a constant overhead, which is reflected in the constant 74.0.

The number of clock cycles for ReLU update with ARM core is found to be

$$C_{relu} = 17.70N + 117.5 \tag{5.6}$$

The total number of clock cycles is

$$C_{total} = C_{mm} + C_{relu} \tag{5.7}$$

Based on equations (5.5), (5.6), (5.7) for computation time, with the keyword spotting network split into 3 PEs (Figure 5.5), the computation of one time step consumes less than 21k clock cycles. With a safety margin of 4k clock cycles, one time step would take less than 25k clock cycles. When the PE is running at 250 MHz, this means the duration of one time step can be reduced to 0.1 ms. Since 10 time steps are combined into 1 time window to form one inference, a time step duration of 0.1 ms would correspond to 1000 inferences per second. In [90], 296 inferences per second has been reported for Loihi (Table 5.1). One reason for the reduced speed of Loihi might be that the inputs to the neural network are coming from an FPGA which could cause some latency, while the SpiNNaker 2 prototype is using inputs generated by one of the PEs of the same chip.

5.5.3 Keyword Spotting: Energy Measurement and Comparison with Loihi

Both QPEs are used for the measurement. In each QPE, 3 PEs are switched on to simulate a keyword spotting network. The measured result is then divided by 2 to obtain the energy per network. The energy is measured incrementally, similar to previous measurements on SpiNNaker 1 [40] and on the first SpiNNaker 2 prototype [25]. When measuring the idle energy, the PLL is started and the software is running on the ARM cores. In each time step, after the timer tick interrupt wakes up the ARM core from the sleep mode, the ARM core only handles the interrupt itself, with no neural processing involved, and then it goes back to sleep mode. The result presented in this section is the active energy which is obtained by subtracting the idle energy from the total energy. The resulting active energy per inference is 7.1 μJ.

The keyword spotting network is implemented as a normal DNN on the SpiNNaker 2 prototype. The MAC array is used for the computation of the connection matrix, and the ARM core is used for the computation of ReLU activation function. Since Loihi only supports SNN, the spiking version of the keyword spotting network is implemented on Loihi. This could be the reason that the SpiNNaker 2 prototype consumes less energy for each inference in the keyword spotting benchmark (Table 5.1). Note that in [90], the reported energy per inference on Loihi was 270 μJ, including a 70 mW overhead presumably caused by the x86 processor on Loihi. In this work the overhead has been removed which results in 37 μJ per inference.

Table 5.1: Comparison of the SpiNNaker 2 prototype (SpiNN) and Loihi for the keyword spotting task

Hardware	inference/sec	energy/inference (μJ)
SpiNN	1000	7.1
Loihi	296	37

5.5.4 Adaptive Control: Memory Footprint

For an adaptive control network simulated on a PE, the required SRAM memory mainly consists of 4 parts: input weight matrix and bias memory, output weight matrix memory, neuron input current memory and neuron memory.

The input weight matrix and bias memory is the memory for storing the input weight matrix and bias, which are quantized as 8-bit integers. The required memory in bytes is

$$M_{ib} \;=\; (D_{in}+1)N \tag{5.8}$$

where D_{in} is the number of input dimensions, N is the number of neurons.

The output weight matrix memory is the memory for storing the output weight matrix, which is 16-bit floating-point numbers. The required memory in bytes is

$$M_o \;=\; 2D_{out}N \tag{5.9}$$

where D_{out} is the number of output dimensions.

The neuron input current memory is the memory for storing the results from the MAC array after the input processing is complete. Each input current is a 32-bit integer. The required memory in bytes is

$$M_{ic} \;=\; 4N \tag{5.10}$$

The neuron memory is the memory to hold the LIF neuron parameters like the membrane potential and refractory time. Each of them has 32 bits. The required memory in bytes is

$$M_n \;=\; 8N \tag{5.11}$$

The total memory for a neural network on a PE is

$$M_{total} \;=\; M_{ib} + M_o + M_{ic} + M_n \tag{5.12}$$

Since it is assumed that each PE has 90 KBytes of SRAM memory available for the data of the neural network, the maximum number of output dimensions given the number of input dimensions and number of neurons in a neural network can be derived with equations (5.8), (5.9), (5.10), (5.11), (5.12). The result is shown Figure 5.8.

5.5.5 Adaptive Control: Computation Time and Comparison with Loihi

For adaptive control, the number of clock cycles for input processing (C_{i_mlacc} / $C_{i\_no\_mlacc}$), neuron update (C_n), output processing (C_o) and weight update (C_w) are measured. After the measurement, polynomial models can be fitted by minimizing the mean-squared

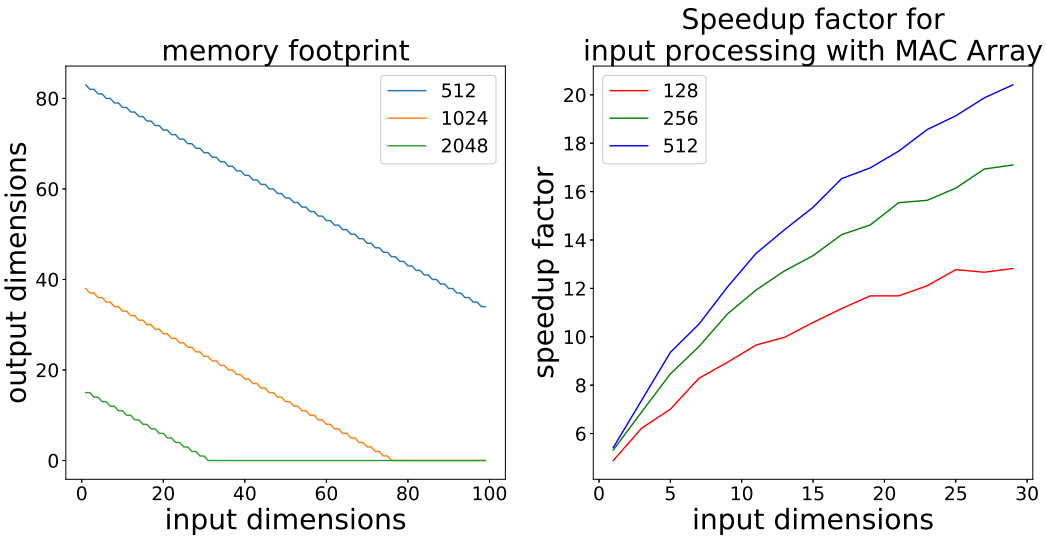

Figure 5.8: left: Maximum number of output dimensions for each input dimension and number of neurons for a neural network simulated on a PE. right: Speedup of input processing time with the MAC array. Numbers in the legend indicate the number of neurons.

error. For input processing with MAC array, the number of clock cycles is

$$\begin{aligned} C_{i\_mac} &= 131.21 + 5.07N \\ &+0.13ND_{in} + 35.79D_{in} \end{aligned} \tag{5.13}$$

where N is the number of neurons, D_{in} is the number of input dimensions. Equation (5.13) is very similar to equation (5.5), because the main computation is in both cases done by the MAC array. The difference is caused by the different data types. In keyword spotting, the inputs are assumed to be 8-bit integers, but in adaptive control, each input is assumed to be floating-point. This is necessary because in general, the same implementation can be used as a building block for NEF implementation on SpiNNaker 2 to construct large-scale cognitive models as mentioned in section 5.4.1, so that the input data type needs to be the same as the output data type. Since the output weights are floating-point, and their values change dynamically due to learning, an extra range check is performed for each input value, and an extra data type conversion is performed. This is reflected in $35.79D_{in}$ and the constant 131.21.

The number of clock cycles without MAC array is

$$\begin{aligned} C_{i\_no\_mac} &= 102.52 + 22.54N \\ &\quad +7.07ND_{in} + 25.54D_{in} \end{aligned} \tag{5.14}$$

The main benefit of MAC array is reflected in the reduction of $7.07ND_{in}$ in equation (5.14) to $0.13ND_{in}$ in equation (5.13), which is made possible by the SIMD operation of the MAC array. The speedup is higher for higher dimensions. Figure 5.8 shows the speedup of the computation time for input processing with the MAC array compared to without the MAC array.

Unlike in keyword spotting, where the ReLU neuron model is used, in adaptive control, the LIF neuron model is used, which is the same as in Loihi. The neuron update time in terms of number of clock cycles is

$$C_n = 28.19N - 26.90NP + 509.18 \tag{5.15}$$

where P is the firing probability. The minus sign in $-26.9NP$ is because, during the refractory period, the computation needed is reduced. Since this is event-based, it depends on P.

The output processing time in terms of number of clock cycles is

$$C_o = 5.8ND_{out}P + 19.31NP \tag{5.16}$$

where D_{out} is the number of output dimensions.

The weight update time in terms of number of clock cycles is

$$C_w = 8.28ND_{out}P + 28.04NP \tag{5.17}$$

The total time in terms of number of clock cycles is

$$C_{total} = C_{i\_mac} + C_n + C_o + C_w \tag{5.18}$$

Since output processing and weight update are event-based, the firing rate of 130 Hz corresponding to a firing probability P of 0.13, which is used for comparing the SpiNNaker 2 prototype with Loihi, would reduce the computation time by 87% compared to a non-event-based implementation.

Typically, the SpiNNaker system runs in real-time with 1 ms time step. When the PE is running at 250 MHz, the available number of clock cycles for each time step is

250 000, which is the computational constraint. According to equation (5.18), for the range of the parameters shown in Figure 5.8, the computation can be done within 1 ms. So the maximum implementable size of a network on a single PE in this benchmark is constrained by memory rather than computation.

For the adaptive control benchmark task with different numbers of input dimensions, output dimensions and neurons, the duration of a time step of SpiNNaker 2 prototype and Loihi is compared and shown in Figure 5.9, with the mean population firing rate kept at around 130 Hz for both hardware platforms. Here the duration of a time step for the SpiNNaker 2 prototype refers to the time for the PE to complete the computation of a time step. From the comparison, it is clear that for small numbers of input dimensions, Loihi is faster than the SpiNNaker 2 prototype, and for large numbers of input dimensions, the SpiNNaker 2 prototype is faster than Loihi. The maximum ratio of the duration of a time step between both hardware platforms is summarized in Table 5.2.

Because of the MAC array, the computation time of the SpiNNaker 2 prototype increases less rapidly with the number of input dimensions, so that the SpiNNaker 2 prototype could catch up with Loihi in terms of computation time for higher input dimensions.

Table 5.2: Maximum ratio of duration of a time step between the SpiNNaker 2 prototype (SpiNN) and Loihi for the adaptive control task

Input Dimensions	1	100
Output Dimensions	1	1
Number of Neurons	1024	512
Duration of a Time Step SpiNN : Loihi	1 : 0.37	0.49 : 1

5.5.6 Adaptive Control: Energy Measurement and Comparison with Loihi

The energy consumption of the SpiNNaker 2 prototype and Loihi is measured with the same parameters as in the computation time comparison. The result is shown in Figure 5.10. Similar to section 5.5.3, only the active energy is shown. For small numbers of input dimensions, Loihi is more energy-efficient than the SpiNNaker 2 prototype, and

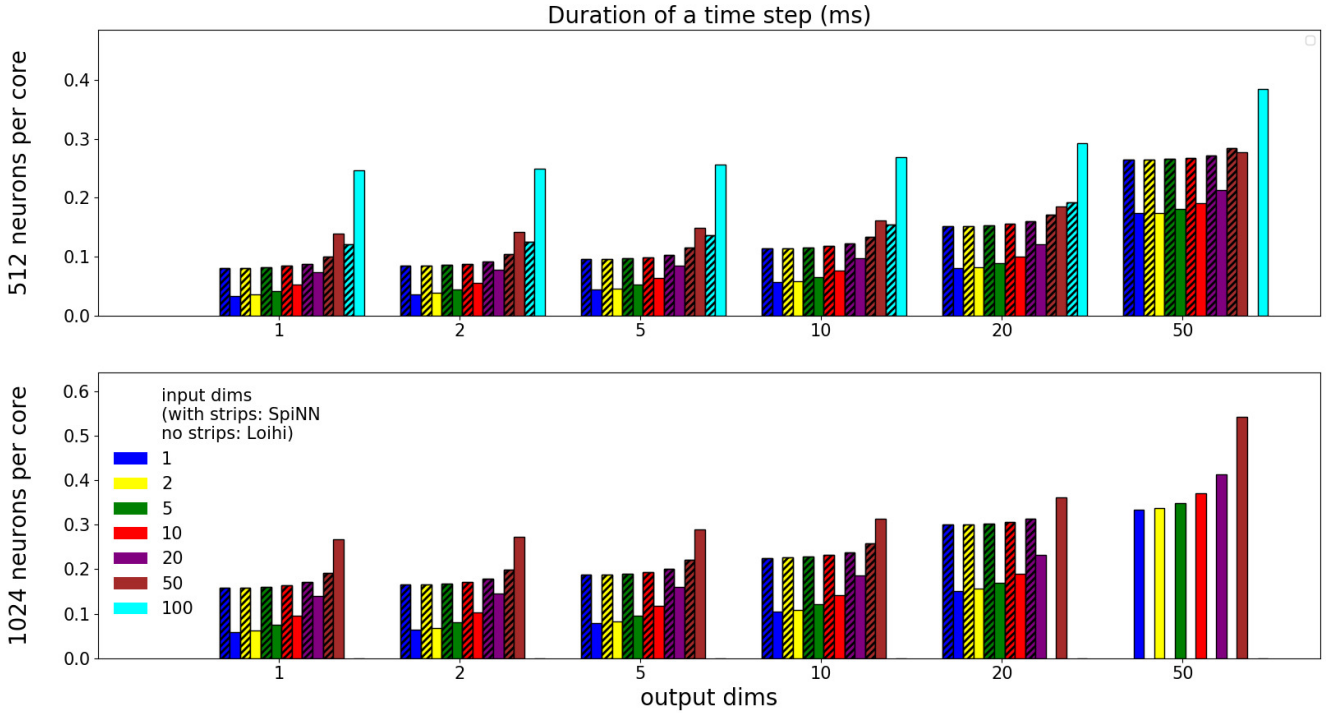

Figure 5.9: Duration of a time step of SpiNNaker 2 prototype (with strips) and Loihi (without strips) for different number of neurons per core, different input and output dimensions for the adaptive control benchmark. No measurement result for the SpiNNaker 2 prototype is shown where the implementation is limited by memory.

Figure 5.10: Active energy of SpiNNaker 2 prototype (with strips) and Loihi (without strips) for different number of neurons per core, different input and output dimensions for the adaptive control benchmark. No measurement result for the SpiNNaker 2 prototype is shown where the implementation is limited by memory.

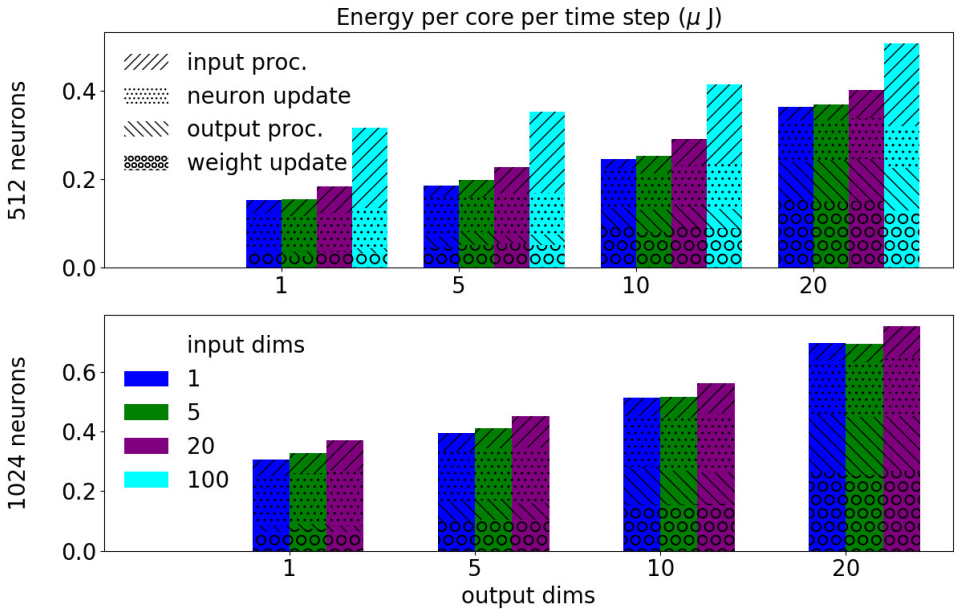

Figure 5.11: Breakdown of energy consumption per core per time step of the SpiNNaker 2 prototype into 4 energy components: input processing, neuron update, output processing and weight update.

for large numbers of input dimensions, the SpiNNaker 2 prototype is more energy-efficient than Loihi. The maximum ratio of active energy consumption between both hardware platforms is summarized in Table 5.3.

Table 5.3: Maximum ratio of active energy consumption between the SpiNNaker 2 prototype (SpiNN) and Loihi for the adaptive control task

Input Dimensions	1	100
Output Dimensions	1	1
Number of Neurons	1024	512
Active Energy SpiNN : Loihi	1 : 0.81	0.36 : 1

Similar to the computation time comparison, the benefit of the MAC array is shown especially for high input dimensions, when the MAC array is more extensively used. This is made more clear in the energy breakdown in Figure 5.11. Here, it is clear how the input processing energy increases with the input dimensions for the same number of neurons and output dimensions, how the neuron update energy increases with the

number of neurons for the same input dimensions and output dimensions, and how the output processing and weight update energy increases with the number of output dimensions for the same input dimensions and number of neurons.

5.5.7 Adaptive Control: Robotic Demo

The SpiNNaker 2 prototype running the adaptive control benchmark is connected to a robotic arm built with Lego Mindstorms EV3 robot kit. The setup is based on [94]. The input to the neural network is the position and velocity of the motor and the output of the neural network is the motor control signal to be combined with the PD controller output, as described in section 5.4.1.

In this demo two situations are considered: the normal case and the simulated aging case (Figure 5.12). In the case of simulated aging an extra weight is added to the robotic arm to resemble the aging effect or unknown disturbance. For each case, the performance of the adaptive controller is compared with a normal PID controller. In the normal case, both controllers perform equally well, but in the simulated aging case, the PID controller cannot adapt itself to the new situation, while the adaptive controller can learn from the error feedback and adapt its parameters to improve the performance (Figure 5.13). The difference between both controllers is made more clear with the root mean squared error (RMSE) (Figure 5.14).

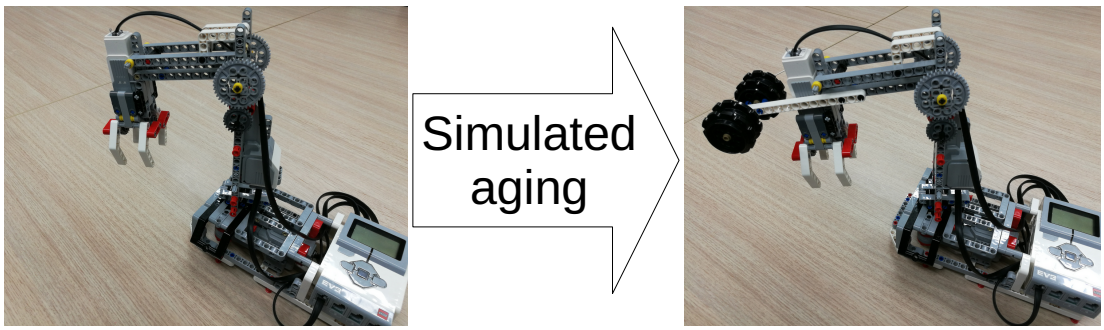

Figure 5.12: Robotic demo: in the normal case (left), there is no extra weight attached to the robotic arm. In the simulated aging case (right), an extra weight is attached to resemble the aging effect.

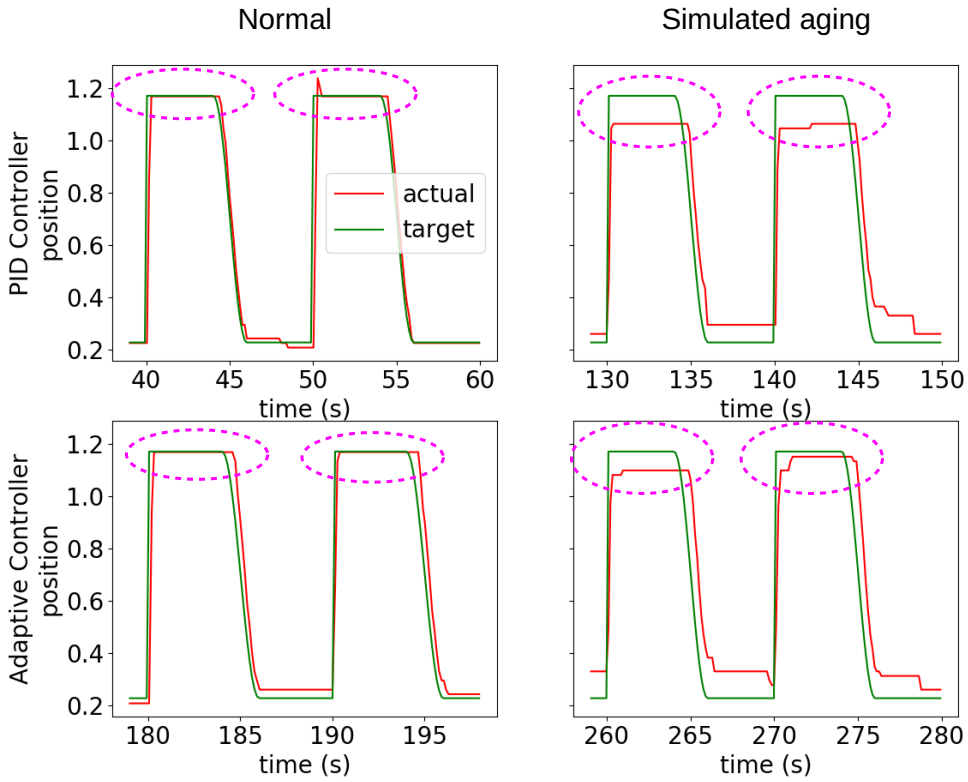

Figure 5.13: Robotic demo: performance of the PID controller and the adaptive controller in both cases. The y-axis is the normalized angle position of the motor. In the normal case, both controllers perform well. But in the simulated aging case, the PID controller cannot adapt to the new situation, while the adaptive controller can improve the performance by adaptation.

5.6 Discussion

In this section, the suitability of other neuromorphic platforms for implementing the benchmarks is considered. Since the comparison between the SpiNNaker 2 prototype and Loihi has already been extensively discussed in previous sections, the summary of this comparison is left to the Conclusion section.

5.6.1 Comparison with SpiNNaker 1

One could assume that the same benchmarks in this work could also be implemented on SpiNNaker 1. However, since in SpiNNaker 1 there is no MAC array, the vector-matrix

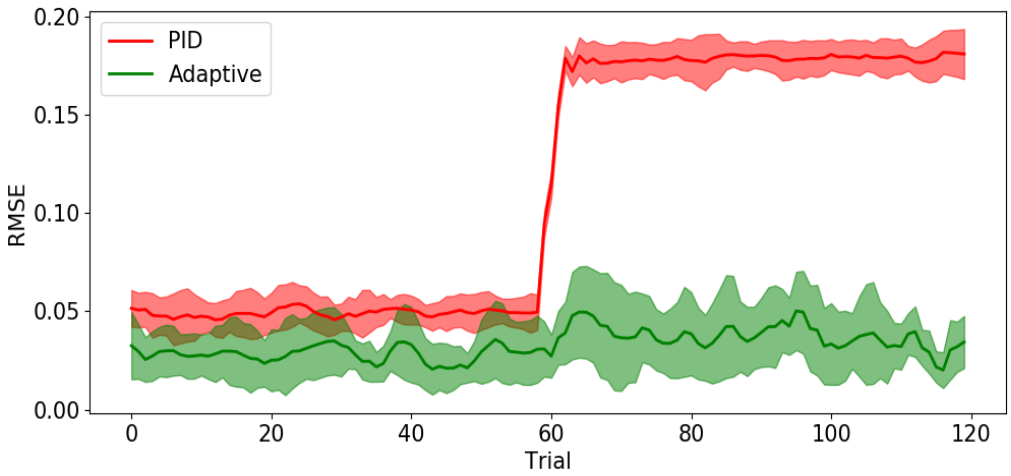

Figure 5.14: Robotic demo: root mean squared error (RMSE) of both controllers. In each trial, the robotic arm attempts to reach either the upper or lower position. The error is measured as the difference between the actual and the target position when the arm has finished transitioning between the upper and lower positions. The mean RMSE and the standard deviation are shown for 10 runs each with 120 trials. The extra weight is added to the arm during the 60th trial. The curve is smoothed with a moving average with a window size of 4.

multiplication would be much slower and therefore consume much more energy than the SpiNNaker 2 prototype. Figure 5.8 indicates the speedup in terms of number of clock cycles for the vector-matrix multiplication in the SpiNNaker 2 prototype compared to what it would be in SpiNNaker 1. The differences in fabrication technology and supply voltage etc. further increases the difference between the SpiNNaker 2 prototype and SpiNNaker 1.

5.6.2 Comparison with other neuromorphic platforms

To ease the discussion, the neuromorphic platforms are grouped into 3 categories:

1. Neuromorphic platforms with static synapses, such as TrueNorth from IBM [18], NeuroGrid [80], Braindrop [98], HiAER-IFAT [81], DYNAPs [82], Tianjic [20], NeuroSoC [99] and DeepSouth [83],

2. Neuromorphic platforms with configurable (but not programmable) plasticity, such as ROLLS [84], ODIN [85] and TITAN [86],

3. Neuromorphic platforms with programmable plasticity, such as (except SpiNNaker 1/2 and Loihi) the BrainScales 1/2 system[14, 100].

All 3 groups of neuromorphic platforms should be able to implement the keyword spotting benchmark. However, DNNs can not be directly implemented on these platforms since they only support SNNs (except Tianjic, which also supports DNNs). Solutions similar to the SNN version implemented on Loihi would be an option.

For adaptive control, since learning is involved, the neuromorphic platforms in group 1 would not be able to support this benchmark. It would still be possible to have an external host PC to reprogram the synaptic weights, but that would not be suitable for embedded applications.

Although the learning rule in adaptive control is relatively simple, it involves multiplying an external error signal with the activity of the presynaptic neuron in every time step, which is quite different from the learning rules normally supported in the neuromorphic community, like Spike-Timing Dependent Plasticity (STDP) [59] or Spike-Driven Synaptic Plasticity (SDSP) [101]. Therefore the neuromorphic platforms in group 2 wouldn't be able to implement the adaptive control benchmark.

The BrainScales 2 system in group 3 comes with programmable plasticity, but since the neural network runs in accelerated time, it is unclear whether the neural activity of each time step can be used for the weight update. Also, it is unclear how to interface robotic applications which require real-time response with a neural network running in accelerated time.

5.7 Conclusion

The PE of the SpiNNaker 2 prototype consists of a general-purpose processor plus highly efficient accelerators, while Loihi employs dedicated circuits for neuron and synapse models plus a flexible learning engine. In this chapter, these two platforms are compared regarding their performance in the same applications, namely keyword spotting and adaptive control. To fully exploit the efficiency of the MAC array and the flexibility of the ARM core, the benchmarks are divided into sub-steps and the vector-matrix multiply is computed with the MAC array and the rest is computed with ARM core. Especially, for adaptive control, the event-based spike generation and learning which is not possible with SIMD is computed with the ARM core.

When comparing SpiNNaker 2 and Loihi, for keyword spotting, because of the MAC array used for vector-matrix multiplication and ARM core used for ReLU activation, the DNN version of keyword spotting network can be directly implemented on the SpiNNaker 2 prototype, while on Loihi the SNN version is implemented for the same task. The result of this is 3 times faster inference and 5 times higher energy efficiency of the SpiNNaker 2 prototype.

For adaptive control, both the SpiNNaker 2 prototype and Loihi are efficient in specific parameter regions. The SpiNNaker 2 prototype is more efficient than Loihi both regarding the computation time and active energy, when the number of input dimensions is high, because that is where the vector-matrix multiplication is more complicated and the MAC array is more dominant. On the other hand, the SpiNNaker 2 prototype is less efficient than Loihi when the number of input dimensions is low, because that is where the vector-matrix multiplication is less complicated and the ARM core is more dominant.

Through the comparison of the SpiNNaker 2 prototype and Loihi in these two benchmarks, more insight into the SpiNNaker 2 system can be gained and the benefit of the MAC array in neuromorphic applications is highlighted. This insight is beneficial for the design of future hybrid networks and also serves as feedback for the design of the next generation SpiNNaker system.

6

Summary and Outlook

6.1 Summary

In this work, the new hardware features in SpiNNaker 2 including DVFS, random number generator, exponential function accelerator and MAC array are demonstrated in various benchmarks including synfire chain, reward-based synaptic sampling, keyword spotting and adaptive control. The measurement results of the implementations of these benchmarks show that the hardware features together with the software optimizations lead to improved energy efficiency, computation time and memory footprint. In particular:

- The DVFS feature is able to increase the energy efficiency for spiking neural network simulation by allowing each PE to dynamically adjust its supply voltage and clock frequency. To make DVFS work for event-based neuromorphic computing, the software flow from SpiNNaker 1 has been adapted and power management strategies have been developed. The benefit of DVFS is demonstrated in the synfire chain network simulation, where a power reduction of 60.7% is achieved.

- Certain computations such as an exponential are very common in neuromorphic or computational neuroscience benchmarks. The random number generator and exponential function generator reduce the computation time for these operations significantly when compared with a software solution on a CPU. These two features are demonstrated with the implementation of a reward-based synaptic sampling algorithm on the SpiNNaker 2 prototype, which is one of the most complex

synaptic plasticity models ever implemented on neuromorphic hardware. The optimizations on the algorithm, software and hardware levels, including the random reallocation of synapse memory, local computation enabled by data structure optimization, numerical optimizations and the use of the random number generator and the exponential function accelerator, together lead to optimized computation time, memory and energy consumption, where the number of clock cycles is reduced by a factor of 2 compared to pure software implementation, and the energy consumption is reduced by 62%.

- The MAC array reduces the computation time and power consumption for vector-matrix multiplications. To efficiently exploit the efficiency of the MAC array and the flexibility of the ARM core, mapping strategies are developed and the benefit is shown in the keyword spotting and adaptive control benchmarks. In addition, the results are compared with Loihi. Due to the parallel processing of vector-matrix multiplication of the MAC array and flexibility of the ARM core which allows the implementation of ReLU activation and event-based processing, the computation of keyword spotting on SpiNNaker 2 is 3 times faster than Loihi and the energy consumption is 5 times lower. The benefit of the MAC array is also shown in the adaptive control task. While for the network parameters where lower-dimensional vector-matrix multiplication is required, Loihi shows better results regarding computation time and power consumption, for the network parameters where higher-dimensional vector-matrix multiplication is required, SpiNNaker 2 shows better results regarding computation time and power consumption.

6.2 Outlook

Neuromorphic computing has been an interdisciplinary research field from the beginning. It started with the analogy between the exponential dependence of current and voltage in the neuron membrane and the transistor in subthreshold operation, which was discovered by Carver Mead [13]. At the same time, machine learning and neuroscience have inspired each other for decades. From that sense, it is natural to envision the future of AI as a combination of machine learning, neuroscience and microelectronics, where machine learning targets the problem of building intelligent machines,

neuroscientific findings provide inspirations and insights from the brain and microelectronics provides the compute engine.

As SpiNNaker 2 has evolved to an efficient supercomputer for brain simulation and machine learning, new questions arise such as how the combination of DNN and SNN could help accelerate neuroscience research, and how the combination of DNN and SNN could help with practical problems which are currently solved only with DNN.

Also, when compared to the design process of machine learning accelerators, where the target neural network architecture type is known from the beginning, e.g. dense layer and convolutional layer, the neuromorphic hardware architectures are still very much "general purpose". As more experience and insights are gained from computational neuroscience and machine learning, the next questions that might need to be studied in the longer term could be e.g. a spiking neural network or a DNN/SNN hybrid network architecture that is particularly efficient for solving a certain kind of task, similar to convolutional neural networks solving image tasks in machine learning, and how this kind of architecture could influence the design of neuromorphic hardware architecture.

Apparently, for these problems, we have barely scratched the surface. And hopefully, the road towards solving these problems would take neuromorphic computing to the next stage.

References

[1] John Graner, Terrence Oakes, Louis French, and Gerard Riedy. **Functional MRI in the Investigation of Blast-Related Traumatic Brain Injury**. *Frontiers in Neurology*, **4**:16, 2013. xii, 21

[2] Sebastian Höppner, Yexin Yan, Andreas Dixius, Stefan Scholze, Johannes Partzsch, Marco Stolba, Florian Kelber, Bernhard Vogginger, Felix Neum/"arker, Georg Ellguth, Stephan Hartmann, Stefan Schiefer, Thomas Hocker, Dennis Walter, Genting Liu, Jim Garside, Steve Furber, and Christian Mayr. **The SpiNNaker 2 Processing Element Architecture for Hybrid Digital Neuromorphic Computing**, 2021. xvi, 4, 10, 11, 22, 23, 35, 72, 74

[3] **Artificial Intelligence**. `https://en.wikipedia.org/wiki/Artificial_intelligence`. Accessed: 2021-05-15. 1

[4] Jacques Bughin, Jeongmin Seong, James Manyika, Michael Chui, and Raoul Joshi. **Notes from the AI frontier: Modeling the impact of AI on the world economy**. discussion paper, McKinsey Global Institute, 2018. 1

[5] Demis Hassabis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick. **Neuroscience-Inspired Artificial Intelligence**. *Neuron*, **95**(2):245–258, 2017. 1

[6] Blake A. Richards, Timothy P. Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, Surya Ganguli, Colleen J. Gillon, Danijar Hafner, Adam Kepecs, Nikolaus Kriegeskorte, Peter Latham, Grace W. Lindsay, Kenneth D. Miller, Richard Naud,

Christopher C. Pack, Panayiota Poirazi, Pieter Roelfsema, João Sacramento, Andrew Saxe, Benjamin Scellier, Anna C. Schapiro, Walter Senn, Greg Wayne, Daniel Yamins, Friedemann Zenke, Joel Zylberberg, Denis Therien, and Konrad P. Kording. **A deep learning framework for neuroscience**. *Nature Neuroscience*, **22**(11):1761–1770, Nov 2019. 1

[7] Jane X. Wang, Zeb Kurth-Nelson, Dharshan Kumaran, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Demis Hassabis, and Matthew Botvinick. **Prefrontal cortex as a meta-reinforcement learning system**. *Nature Neuroscience*, **21**(6):860–868, Jun 2018. 1

[8] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. **Efficient Processing of Deep Neural Networks: A Tutorial and Survey**. *Proceedings of the IEEE*, **105**(12):2295–2329, 2017. 2

[9] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. **Mastering the game of Go with deep neural networks and tree search**. *Nature*, **529**(7587):484–489, Jan 2016. 2

[10] Frederico A.C. Azevedo, Ludmila R.B. Carvalho, Lea T. Grinberg, José Marcelo Farfel, Renata E.L. Ferretti, Renata E.P. Leite, Wilson Jacob Filho, Roberto Lent, and Suzana Herculano-Houzel. **Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain**. *Journal of Comparative Neurology*, **513**(5):532–541, 2009. 2

[11] Louis Sokoloff. **The metabolism of the central nervous system in vivo**. *Handbook of Physiology, section, I, Neurophysiology*, **3**:1843–64, 1960. 2

[12] **-morphic (suffix)**. `https://en.wiktionary.org/wiki/-morphic`. Accessed: 2021-05-15. 2

[13] Carver Mead. *Analog VLSI and Neural Systems*. Addison-Wesley Longman Publishing Co., Inc., USA, 1989. 2, 98

[14] S. Friedmann, J. Schemmel, A. Grübl, A. Hartel, M. Hock, and K. Meier. **Demonstrating Hybrid Learning in a Flexible Neuromorphic Hardware System**. *IEEE Transactions on Biomedical Circuits and Systems*, **11**(1):128–142, Feb 2017. 2, 68, 94

[15] Marko Noack, Johannes Partzsch, Christian G Mayr, Stefan Hänzsche, Stefan Scholze, Sebastian Höppner, Georg Ellguth, and Rene Schüffny. **Switched-capacitor realization of presynaptic short-term-plasticity and stop-learning synapses in 28 nm CMOS**. *Frontiers in neuroscience*, **9**:10, 2015. 2, 45

[16] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana. **The SpiNNaker Project**. *Proceedings of the IEEE*, **102**(5):652–665, May 2014. 2, 4

[17] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang. **Loihi: A Neuromorphic Manycore Processor with On-Chip Learning**. *IEEE Micro*, **38**(1):82–99, January 2018. 2, 68

[18] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. **A million spiking-neuron integrated circuit with a scalable communication network and interface**. *Science*, **345**(6197):668–673, 2014. 2, 68, 93

[19] Misha Mahowald. **VLSI analogs of neuronal visual processing: a synthesis of form and function**. 1992. 2

[20] Jing Pei, Lei Deng, Sen Song, Mingguo Zhao, Youhui Zhang, Shuang Wu, Guanrui Wang, Zhe Zou, Zhenzhi Wu, Wei He, Feng Chen, Ning Deng, Si Wu, Yu Wang, Yujie Wu, Zheyu Yang, Cheng Ma, Guoqi Li, Wentao Han, Huanglong Li, Huaqiang Wu, Rong Zhao, Yuan Xie, and Luping Shi. **Towards artificial general intelligence with hybrid Tianjic chip architecture**. *Nature*, **572**(7767):106–111, Aug 2019. 4, 93

[21] Katrin Amunts, Christoph Ebell, Jeff Muller, Martin Telefont, Alois Knoll, and Thomas Lippert. **The human brain project: creating a European research infrastructure to decode the human brain**. *Neuron*, **92**(3):574–581, 2016. 4

[22] Christian Mayr, Sebastian Hoeppner, and Steve Furber. **SpiNNaker 2: A 10 Million Core Processor System for Brain Simulation and Machine Learning**. *arXiv e-prints*, page arXiv:1911.02385, November 2019. 4

[23] R. Carter, J. Mazurier, L. Pirro, J-U. Sachse, P. Baars, J. Faul, C. Grass, G. Grasshoff, P. Javorka, T. Kammler, A. Preusse, S. Nielsen, T. Heller, J. Schmidt, H. Niebojewski, P-Y. Chou, E. Smith, E. Erben, C. Metze, C. Bao, Y. Andee, I. Aydin, S. Morvan, J. Bernard, E. Bourjot, T. Feudel, D. Harame, R. Nelluri, H.-J. Thees, L. M-Meskamp, J. Kluth, R. Mulfinger, M. Rashed, R. Taylor, C. Weintraub, J. Hoentschel, M. Vinet, J. Schaeffer, and B. Rice. **22nm FDSOI technology for emerging mobile, Internet-of-Things, and RF applications**. In *2016 IEEE International Electron Devices Meeting (IEDM)*, pages 2.2.1–2.2.4, 2016. 10, 23

[24] S. Höppner, Y. Yan, B. Vogginger, A. Dixius, J. Partzsch, F. Neumärker, S. Hartmann, S. Schiefer, S. Scholze, G. Ellguth, L. Cederstroem, M. Eberlein, C. Mayr, S. Temple, L. Plana, J. Garside, S. Davison, D. R. Lester, and S. Furber. **Dynamic voltage and frequency scaling for neuromorphic many-core systems**. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2017. 11, 22, 23, 46

[25] S. Höppner, B. Vogginger, Y. Yan, A. Dixius, S. Scholze, J. Partzsch, F. Neumärker, S. Hartmann, S. Schiefer, G. Ellguth, L. Cederstroem, L. A. Plana, J. Garside, S. Furber, and C. Mayr. **Dynamic Power Management for Neuromorphic Many-Core Systems**. *IEEE Transactions on Circuits and Systems I: Regular Papers*, **66**(8):2973–2986, 2019. 11, 22, 23, 46, 83

[26] F. Neumärker, S. Höppner, A. Dixius, and C. Mayr. **True random number generation from bang-bang ADPLL jitter**. In *2016 IEEE Nordic Circuits and Systems Conference (NORCAS)*, pages 1–5, Nov 2016. 11, 45, 46, 47

[27] J. Partzsch, S. Höppner, M. Eberlein, R. Schüffny, C. Mayr, D. R. Lester, and S. Furber. **A fixed point exponential function accelerator for a neuromorphic many-core system**. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2017. 11, 47, 55

[28] M. Mikaitis, D. R. Lester, D. Shang, S. Furber, G. Liu, J. Garside, S. Scholze, S. Höppner, and A. Dixius. **Approximate Fixed-Point Elementary Function Accelerator for the SpiNNaker-2 Neuromorphic Chip**. In *2018 IEEE 25th Symposium on Computer Arithmetic (ARITH)*, pages 37–44, 2018. 11

[29] Y. Yan, D. Kappel, F. Neumärker, J. Partzsch, B. Vogginger, S. Höppner, S. Furber, W. Maass, R. Legenstein, and C. Mayr. **Efficient Reward-Based Structural Plasticity on a SpiNNaker 2 Prototype**. *IEEE Transactions on Biomedical Circuits and Systems*, **13**(3):579–591, 2019. 11

[30] Yexin Yan, Terrence Stewart, Xuan Choo, Bernhard Vogginger, Johannes Partzsch, Sebastian Hoeppner, Florian Kelber, Chris Eliasmith, Steve Furber, and Christian Mayr. **Comparing Loihi with a SpiNNaker 2 Prototype on Low-Latency Keyword Spotting and Adaptive Robotic Control**. *Neuromorphic Computing and Engineering*, 2021. 11

[31] Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014. 13, 14, 18, 25, 48

[32] A. L. HODGKIN and A. F. HUXLEY. **A quantitative description of membrane current and its application to conduction and excitation in nerve**. *The Journal of physiology*, **117**(4):500–544, Aug 1952. 12991237[pmid]. 13, 45

[33] Norman L. Strominger, Robert J. Demarest, and Lois B. Laemle. *Cerebral Cortex*, pages 429–451. Humana Press, Totowa, NJ, 2012. 17

[34] M. Carandini. **Area V1**. *Scholarpedia*, **7**(7):12105, 2012. revision #137292. 17

[35] C. G. Gross. **Inferior temporal cortex**. *Scholarpedia*, **3**(12):7294, 2008. revision #137535. 17

[36] Rodney J. Douglas and Kevan A.C. Martin. **NEURONAL CIRCUITS OF THE NEOCORTEX**. *Annual Review of Neuroscience*, **27**(1):419–451, 2004. PMID: 15217339. 18, 19

[37] Nikos K. Logothetis, Jon Pauls, Mark Augath, Torsten Trinath, and Axel Oeltermann. **Neurophysiological investigation of the basis of the fMRI signal**. *Nature*, **412**(6843):150–157, Jul 2001. 22

[38] C. Steger, C. Bachmann, A. Genser, R. Weiss, and J. Haid. **Power-aware hardware/software codesign of mobile devices**. *e & i Elektrotechnik und Informationstechnik*, **127**(11):327–334, Nov 2010. 22

[39] Sebastian Höppner, Chenming Shao, Holger Eisenreich, Georg Ellguth, Mario Ander, and René Schüffny. **A power management architecture for fast per-core DVFS in heterogeneous MPSoCs**. In *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 261–264, 2012. 24

[40] Evangelos Stromatias, Francesco Galluppi, Cameron Patterson, and Steve Furber. **Power analysis of large-scale, real-time neural networks on SpiNNaker**. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2013. 24, 83

[41] Moshe Abeles. *Corticonics: Neural circuits of the cerebral cortex*. Cambridge University Press, 1991. 25

[42] M. Abeles. **Synfire chains**. *Scholarpedia*, **4**(7):1441, 2009. revision #150018. 25

[43] Markus Diesmann, Marc-Oliver Gewaltig, and Ad Aertsen. **Stable propagation of synchronous spiking in cortical neural networks**. *Nature*, **402**(6761):529–533, Dec 1999. 26

[44] Jens Kremkow, Laurent U. Perrinet, Guillaume S. Masson, and Ad Aertsen. **Functional consequences of correlated excitatory and inhibitory conductances in cortical networks**. *Journal of Computational Neuroscience*, **28**(3):579–594, Jun 2010. 26

[45] Oliver Rhodes, Petruţ A. Bogdan, Christian Brenninkmeijer, Simon Davidson, Donal Fellows, Andrew Gait, David R. Lester, Mantas Mikaitis, Luis A. Plana, Andrew G. D. Rowley, Alan B. Stokes, and Steve B. Furber. **sPyNNaker: A Software Package for Running PyNN Simulations on SpiNNaker**. *Frontiers in Neuroscience*, **12**:816, 2018. 27, 57, 59

[46] Anthony JGD Holtmaat, Joshua T Trachtenberg, Linda Wilbrecht, Gordon M Shepherd, Xiaoqun Zhang, Graham W Knott, and Karel Svoboda. **Transient and persistent dendritic spines in the neocortex in vivo**. *Neuron*, **45**(2):279–291, 2005. 44

[47] Simon Rumpel and Jochen Triesch. **The dynamic connectome**. *e-Neuroforum*, **7**(3):48–53, 2016. 44

[48] Roman Dvorkin and Noam E Ziv. **Relative contributions of specific activity histories and spontaneous processes to size remodeling of glutamatergic synapses**. *PLoS biology*, **14**(10):e1002572, 2016. 44

[49] Mo Zhou, Tianyi Liu, Yan Li, Dachao Lin, Enlu Zhou, and Tuo Zhao. **Towards Understanding the Importance of Noise in Training Neural Networks**, 2019. 44

[50] Mark D McDonnell and Lawrence M Ward. **The benefits of noise in neural systems: bridging theory and experiment**. *Nature Reviews Neuroscience*, **12**(7):415, 2011. 44

[51] Wolfgang Maass. **Noise as a resource for computation and learning in networks of spiking neurons**. *Proceedings of the IEEE*, **102**(5):860–880, 2014. 44

[52] David Kappel, Stefan Habenschuss, Robert Legenstein, and Wolfgang Maass. **Network plasticity as Bayesian inference**. *PLoS computational biology*, **11**(11):e1004485, 2015. 44, 45, 48

[53] David Kappel, Stefan Habenschuss, Robert Legenstein, and Wolfgang Maass. **Synaptic sampling: a Bayesian approach to neural network plasticity and rewiring**. In *Advances in Neural Information Processing Systems*, pages 370–378, 2015. 44, 45, 48

[54] David Kappel, Robert Legenstein, Stefan Habenschuss, Michael Hsieh, and Wolfgang Maass. **A Dynamic Connectome Supports the Emergence of Stable Computational Function of Neural Circuits through Reward-Based Learning**. *eNeuro*, **5**(2), 2018. 44, 48, 50, 51, 63

[55] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. **Deep Rewiring: Training very sparse deep networks**. *ICLR*, 2018. 44, 52

[56] George Marsaglia. **Xorshift RNGs**. *Journal of Statistical Software, Articles*, **8**(14):1–6, 2003. 44, 46, 54

[57] Michael Hopkins. **random.c (source code)**, 2014. 44, 54

[58] W. Gerstner and R. Brette. **Adaptive exponential integrate-and-fire model**. *Scholarpedia*, **4**(6):8427, 2009. revision #90944. 45

[59] Henry Markram, Joachim Lübke, Michael Frotscher, and Bert Sakmann. **Regulation of Synaptic Efficacy by Coincidence of Postsynaptic APs and EPSPs**. *Science*, **275**(5297):213–215, 1997. 45, 94

[60] Giacomo Indiveri, Federico Corradi, and Ning Qiao. **Neuromorphic architectures for spiking deep neural networks**. In *Electron Devices Meeting (IEDM), 2015 IEEE International*, pages 4–2. IEEE, 2015. 45

[61] Nan Du, Mahdi Kiani, Christian G Mayr, Tiangui You, Danilo Bürger, Ilona Skorupa, Oliver G Schmidt, and Heidemarie Schmidt. **Single pairing spike-timing dependent plasticity in BiFeO3 memristors with a time window of 25 ms to 125 μs**. *Frontiers in neuroscience*, **9**:227, 2015. 45

[62] Timothée Levi, Takuya Nanami, Atsuya Tange, Kazuyuki Aihara, and Takashi Kohno. **Development and Applications of Biomimetic Neuronal Networks Toward BrainMorphic Artificial Intelligence**. *IEEE Transactions on Circuits and Systems II: Express Briefs*, **65**(5):577–581, 2018. 45

[63] Sebastian Schmitt, Johann Klaehn, Guillaume Bellec, Andreas Grübl, Maurice Guettler, Andreas Hartel, Stephan Hartmann, Dan Husmann de Oliveira, Kai Husmann, Vitali Karasenko, Mitja Kleider, Christoph Koke, Christian Mauch, Eric Müller, Paul Mueller, Johannes Partzsch, Mihai A. Petrovici, Stefan Schiefer, Stefan Scholze, Bernhard Vogginger, Robert A. Legenstein, Wolfgang Maass, Christian Mayr, Johannes Schemmel, and Karlheinz Meier. **Neuromorphic Hardware In The Loop: Training a Deep Spiking Network on the BrainScaleS Wafer-Scale System**. *Proceedings of the 2017 IEEE International Joint Conference on Neural Networks*, pages 2227–2234, 2017. 45

[64] Mihai A Petrovici, Sebastian Schmitt, Johann Klähn, David Stöckel, Anna Schroeder, Guillaume Bellec, Johannes Bill, Oliver Breitwieser, Ilja Bytschok, Andreas Grübl, et al. **Pattern representation and recognition with accelerated analog neuromorphic systems**.

In *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*, pages 1–4. IEEE, 2017. 45

[65] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. **A million spiking-neuron integrated circuit with a scalable communication network and interface**. *Science*, **345**(6197):668–673, 2014. 45

[66] James C. Knight and Thomas Nowotny. **GPUs Outperform Current HPC and Neuromorphic Solutions in Terms of Speed and Energy When Simulating a Highly-Connected Cortical Model**. *Frontiers in Neuroscience*, **12**:941, 2018. 45

[67] Bernhard Vogginger, René Schüffny, Anders Lansner, Love Cederström, Johannes Partzsch, and Sebastian Höppner. **Reducing the computational footprint for real-time BCPNN learning**. *Frontiers in Neuroscience*, **9**:2, 2015. 45

[68] J. Partzsch, S. Höppner, M. Eberlein, R. Schüffny, C. Mayr, D. R. Lester, and S. Furber. **A fixed point exponential function accelerator for a neuromorphic many-core system**. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2017. 45

[69] G. E. P. Box and Mervin E. Muller. **A Note on the Generation of Random Normal Deviates**. *Ann. Math. Statist.*, **29**(2):610–611, 06 1958. 54

[70] Wolfgang Hörmann and Josef Leydold. **Continuous Random Variate Generation by Fast Numerical Inversion**. *ACM Trans. Model. Comput. Simul.*, **13**(4):347–362, October 2003. 54

[71] Burkhard Dünweg and Wolfgang Paul. **Brownian Dynamics Simulations Without Gaussian Random Numbers**. *International Journal of Modern Physics C*, **2**(3):817–827, 1991. 54

[72] Mantas Mikaitis, Garibaldi Pineda García, James C. Knight, and Steve B. Furber. **Neuromodulated Synaptic Plasticity on the SpiNNaker Neuromorphic System**. *Frontiers in Neuroscience*, **12**:105, 2018. 57, 68

[73] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber. **SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation**. *IEEE Journal of Solid-State Circuits*, **48**(8):1943–1953, Aug 2013. 57

[74] Song Han, Jeff Pool, John Tran, and William J. Dally. **Learning Both Weights and Connections for Efficient Neural Networks**. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 1135–1143, Cambridge, MA, USA, 2015. MIT Press. 57

[75] T. Sharp and S. Furber. **Correctness and performance of the SpiNNaker architecture**. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, Aug 2013. 59

[76] Sacha J van Albada, Andrew G Rowley, Johanna Senk, Michael Hopkins, Maximilian Schmidt, Alan B Stokes, David R Lester, Markus Diesmann, and Steve B Furber. **Performance comparison of the digital neuromorphic hardware SpiNNaker and the neural network simulation software NEST for a full-scale cortical microcircuit model**. *Frontiers in neuroscience*, **12**, 2018. 59

[77] Javier Navaridas, Mikel Luján, Luis A. Plana, Steve Temple, and Steve B. Furber. **SpiNNaker: Enhanced multicast routing**. *Parallel Computing*, **45**:49 – 66, 2015. Computing Frontiers 2014: Best Papers. 60

[78] Tarec Fares and Armen Stepanyants. **Cooperative synapse formation in the neocortex**. *Proceedings of the National Academy of Sciences*, **106**(38):16463–16468, 2009. 63

[79] Petruț Antoniu Bogdan, Andrew Graham David Rowley, Oliver Rhodes, and Steve B Furber. **Structural plasticity on the SpiNNaker many-core neuromorphic system**. *Frontiers in Neuroscience*, **12**:434, 2018. 68

[80] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R. Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V. Arthur, Paul A. Merolla, and Kwabena Boahen. **Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations**. *Proceedings of the IEEE*, **102**:1–18, 05 2014. 68, 93

[81] Jongkil Park, Theodore Yu, Siddharth Joshi, Christoph Maier, and Gert Cauwenberghs. **Hierarchical address event routing for reconfigurable large-scale neuromorphic systems**. *IEEE transactions on neural networks and learning systems*, **28**(10):2408–2422, 2017. 68, 93

[82] 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*, **12**(1):106–122, Feb 2018. 68, 93

[83] Runchun M. Wang, Chetan S. Thakur, and André van Schaik. **An FPGA-Based Massively Parallel Neuromorphic Cortex Simulator**. *Frontiers in Neuroscience*, **12**:213, 2018. 68, 93

[84] Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. **A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses**. *Frontiers in Neuroscience*, **9**:141, 2015. 68, 93

[85] C. Frenkel, M. Lefebvre, J. Legat, and D. Bol. **A 0.086-mm^2 12.7-pJ/SOP 64k-Synapse 256-Neuron Online-Learning Digital Spiking Neuromorphic Processor in 28-nm CMOS**. *IEEE Transactions on Biomedical Circuits and Systems*, **13**(1):145–158, Feb 2019. 68, 93

[86] C. Mayr, J. Partzsch, M. Noack, S. Hänzsche, S. Scholze, S. Höppner, G. Ellguth, and R. Schüffny. **A Biological-Realtime Neuromorphic System in 28 nm CMOS Using Low-Leakage Switched Capacitor Circuits**. *IEEE Transactions on Biomedical Circuits and Systems*, **10**(1):243–254, Feb 2016. 68, 93

[87] Chetan Singh Thakur, Jamal Lottier Molin, Gert Cauwenberghs, Giacomo Indiveri, Kundan Kumar, Ning Qiao, Johannes Schemmel, Runchun Wang, Elisabetta Chicca, Jennifer Olson Hasler, Jae-sun Seo, Shimeng Yu, Yu Cao, André van Schaik, and Ralph Etienne-Cummings. **Large-Scale Neuromorphic Spiking Array Processors: A Quest to Mimic the Brain**. *Frontiers in Neuroscience*, **12**:891, 2018. 68

[88] M. Rahimi Azghadi, N. Iannella, S. F. Al-Sarawi, G. Indiveri, and D. Abbott. **Spike-Based Synaptic Plasticity in Silicon: Design, Implementation, Application, and Challenges**. *Proceedings of the IEEE*, **102**(5):717–737, May 2014. 68

[89] V. Sze, Y. Chen, T. Yang, and J. S. Emer. **Efficient Processing of Deep Neural Networks: A Tutorial and Survey**. *Proceedings of the IEEE*, **105**(12):2295–2329, 2017. 72

[90] Peter Blouw, Xuan Choo, Eric Hunsberger, and Chris Eliasmith. **Benchmarking Keyword Spotting Efficiency on Neuromorphic Hardware**. In *Proceedings of the 7th Annual Neuro-Inspired Computational Elements Workshop*, NICE'19, New York, NY, USA, 2019. Association for Computing Machinery. 75, 76, 82, 83

[91] J.J. Slotine and W. Li. **On the adaptive control of robot manipulators**. *International Journal of Robotics Research*, pages 49–59, 1987. 77, 78

[92] Chris Eliasmith and Charles H. Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems.* MIT Press, Cambridge, MA, 2003. 77

[93] Travis DeWolf, Terrence C. Stewart, Jean-Jacques Slotine, and Chris Eliasmith. **A spiking neural model of adaptive arm control**. *Proceedings of the Royal Society B: Biological Sciences*, **283**(1843):20162134, 2016. 77

[94] Terrence C. Stewart, Travis DeWolf, Ashley Kleinhans, and Chris Eliasmith. **Closed-Loop Neuromorphic Benchmarks**. *Frontiers in Neuroscience*, **9**:464, 2015. 77, 91

[95] Travis DeWolf, Pawel Jaworski, and Chris Eliasmith. **Nengo and low-power AI hardware for robust, embedded neurorobotics**. *arXiv e-prints*, page arXiv:2007.10227, July 2020. 77, 78

[96] A. Mundy, J. Knight, T. C. Stewart, and S. Furber. **An efficient SpiNNaker implementation of the Neural Engineering Framework**. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2015. 79

[97] J. Knight, A. R. Voelker, A. Mundy, C. Eliasmith, and S. Furber. **Efficient SpiNNaker simulation of a heteroassociative memory using the Neural Engineering Framework**. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 5210–5217, 2016. 79

[98] A. Neckar, S. Fok, B. V. Benjamin, T. C. Stewart, N. N. Oza, A. R. Voelker, C. Eliasmith, R. Manohar, and K. Boahen. **Braindrop: A Mixed-Signal Neuromorphic Architecture With a Dynamical Systems-Based Programming Model**. *Proceedings of the IEEE*, **107**(1):144–164, 2019. 93

[99] Johannes Partzsch, Christian Mayr, Massimiliano Giulioni, Marko Noack, Stefan Hänzsche, Stefan Scholze, Sebastian Höppner, Paolo Del Giudice, and Rene Schüffny. **Mean field approach for configuring population dynamics on a biohybrid neuromorphic system**. *Journal of Signal Processing Systems*, pages 1–19, 2020. 93

[100] Stephan Hartmann, Stefan Schiefer, Stefan Scholze, Johannes Partzsch, Christian Mayr, Stephan Henker, and René Schüffny.

Highly integrated packet-based AER communication infrastructure with 3Gevent/s throughput. In *2010 17th IEEE International Conference on Electronics, Circuits and Systems*, pages 950–953. IEEE, 2010. 94

[101] Joseph M. Brader, Walter Senn, and Stefano Fusi. **Learning Real-World Stimuli in a Neural Network with Spike-Driven Synaptic Dynamics**. *Neural Computation*, **19**(11):2881–2912, 2007. 94