TU WIEN Informatics

Modeling Resource Utilization for Spiking Neural Networks in FPGAs

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Hardware- and Software Security

eingereicht von

Markus Müllner, BSc
Matrikelnummer 00728100

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Steininger
Mitwirkung: Univ.Ass. Dott.mag. Alessio Colucci, BSc

Wien, 30. Jänner 2024

Markus Müllner	Andreas Steininger

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Informatics

Modeling Resource Utilization for Spiking Neural Networks in FPGAs

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Hardware- and Software Security

by

Markus Müllner, BSc
Registration Number 00728100

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Steininger
Assistance: Univ.Ass. Dott.mag. Alessio Colucci, BSc

Vienna, January 30, 2024

Markus Müllner	Andreas Steininger

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Markus Müllner, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. Jänner 2024

Markus Müllner

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I want to thank my supervisor, Andreas Steininger, as well as my advisor Alessio Colucci for their support and feedback.

I would also like to thank the Austrian Study Grant Authority (Studienbeihilfenbehörde) for the opportunity to finish this work in the scope of a scholarship (Studienabschlussstipendium), without which finalization would have taken considerably longer.

Finally, I would like to thank my family and friends for giving me time to work on my thesis, as well as providing distraction when I needed to disentangle my thoughts.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Das bereits seit langem bestehende Konzept des maschinellen Lernens als Basis für künstliche Intelligenz war bislang für die meisten Menschen nur ein Science-Fiction-Abenteuer, in dem rebellierende Roboter die Menschheit bedrohten. Erst das Aufkommen und der vermehrte Einsatz neuer Anwendungen wie Sprachmodelle oder Bildsynthese hat den Nutzen von künstlicher Intelligenz einer breiten Öffentlichkeit bewusst gemacht.

Im Bereich des maschinellen Lernens sind künstliche neuronale Netze mathematische Modelle, die von im Gehirn von Tieren beobachteten Strukturen und Prozessen inspiriert sind. Die Untergruppe der Spiking Neural Networks (SNNs) konzentriert sich auf die Nachahmung der pulsbasierten Kommunikation von Nervenzellen, die über zeitlich gesteuerte Reize kommunizieren, anstatt numerische Werte zu übertragen. Obwohl Spiking Neural Networks derzeit in der Praxis nur selten zum Einsatz kommen, sind sie als Ergebnis bahnbrechender Studien zur Modellierung des biologischen Denkens für das Verständnis von intelligentem Leben unerlässlich.

Während es problemlos möglich ist Spiking Neural Networks auf gewöhnlichen Computern zu simulieren, kann der Vorgang ähnlich wie bereits in anderen Bereichen (z. B. Rendering, Kryptographie, ...) üblich, durch speziell entwickelte Hardware signifikant beschleunigt werden. Für die Durchführung spezifischer Aufgaben optimierte Schaltungen verbessern sowohl die Geschwindigkeit als auch die Energieeffizienz und können auch bei der Integration in ansonsten weniger leistungsstarke Systeme unterstützend wirken.

Field-Programmable Gate Arrays (FPGAs) sind integrierte Schaltungen, welche es durch aufspielen einer Konfiguration erlauben, generische Logikblöcke frei anzuordnen und zu verbinden. Mit ihrer Funktionalität können nahezu beliebige digitale Schaltungen nachgebildet werden. Sie eignen sich sehr gut zur Entwicklung und Erprobung von für spezielle Aufgaben optimierte Schaltungen und bieten gleichzeitig deutlich kürzere Entwicklungs- und Produktionszeiten.

Diese Arbeit stellt in einem modularen Ansatz entwickelte Grundbausteine vor, die für den Aufbau von Spiking Neural Networks in FPGAs notwendig sind. Weiters beinhaltet sie ein Hardware-/Software-Framework für die Integration in softwarebasierte Anwendungen. Mithilfe der Implementierung und des Frameworks wird auf die Abschätzung der erforderlichen Hardware-Ressourcen und die damit einhergehenden Simulationsgeschwindigkeit verschieden strukturierter Netze eingegangen. Die Ergebnisse sollen Entwicklern

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

bei der Modellierung von Netzwerken mit bestmöglicher Struktur und bei der Auswahl eines FPGA-Bausteins geeigneter Größe und Geschwindigkeit unterstützen. Darüber hinaus ermöglicht das erarbeitete Modell einen Vergleich mit anderen Technologien, um eine Entscheidung hinsichtlich der Verwendung von FPGAs als geeignete Lösung für den jeweiligen Anwendungsfall zu unterstützen.

Für einen zur Klassifizierung von handschriftlich erstellten Ziffern entwickelten Beispielaufbau werden Messergebnisse in Bezug auf Leistung, Stromverbrauch und Ressourcennutzung präsentiert. Die Ergebnisse stimmen mit dem zuvor abgeleiteten Modell weitgehend überein, einzelne aufgetretene Abweichungen und deren Abhängigkeit von der Netzwerkstruktur werden analysiert und diskutiert.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Abstract

Although the concept of machine learning has been around for a long time, to most people it was nothing more than the stuff of science fiction, where rebelling robots threaten to overthrow humanity. Only recently, the introduction of easily accessible technologies (e.g. large language models, image synthesis tools, ...) made a wide public aware of the advantages of utilizing artificial intelligence.

Artificial neural networks are mathematical models inspired by the structures and processes observed in the brains of animals. Spiking Neural Networks (SNNs) are a subset focusing on the fact that excitable cells communicate via spikes, encoding information in their timing, rather than transmitting values. While being practically employed less often than other network types, they are very interesting for understanding the nature of intelligent life, because they were invented following groundbreaking studies on modeling the biological brain.

The behavior of spiking neural networks can be simulated with general purpose hardware. This is perfectly fine, but as it can be seen in other areas (e.g. rendering, cryptography, ...) systems dedicated to a specific tasks can benefit greatly from hardware accelerators. Purpose-built designs improve performance and energy efficiency, as well as help integrating a specific technology in an otherwise less powerful system.

Field-Programmable Gate Arrays (FPGAs) are a type of integrated circuit, containing an array of logic blocks and interconnects, that can be configured to form virtually any digital circuit. They are perfectly suited for evaluating the optimization potential that comes with a dedicated circuit-level design, while having much shorter development and production times.

This work will introduce a modular FPGA-based implementation, providing the basic building blocks for constructing networks, and a hardware/software framework for embedding SNNs into real-world applications. Employing the implementation and the framework, we focus on measuring the required hardware resources and resulting performance of different networks, depending on their specific structures. The results will aid in modeling the SNN implementation against different parameters, such as hardware resources and power consumption, allowing the designers to properly select the SNN network structure and the FPGA device. Additionally, such model can aid designers in making a conscious decision when determining whether an FPGA is a viable device for their use-case.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Furthermore, we present testing results, in regards to performance, power consumption and resource utilization, based on an example setup, built to classify handwritten digits. The results are in line with our model, with small discrepancies that are further analyzed and discussed, as they are dependent on the occupation and the size of the FPGA device.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Contents

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

In the last five decades, there has been a steady increase in the number and density of transistors packed into integrated circuits [22]. Along with continuous improvements in thermal management and the steadily increasing switching frequencies, we gained the computational power to solve increasingly complex problems. With an abundance of readily available computational resources, the challenges started shifting.

Scientist and programmers have a hard time coming up with algorithms for complex tasks, like interpreting language or recognizing pictures. Many problems lack a clear mathematical definition and therefore don't have an analytical solution. Computers struggle with complex tasks, which can often be summarized as: given some input, "do what a human would do!" Therefore, simulating the processes that lead to human decision making is the most obvious solution.

Machine learning is an active field of research [5], that aims to help computers in finding their own algorithms. Artificial neural networks are specific algorithms that try to imitate, what was achieved in nature through an evolutionary process. They are an abstract interpretation of how the brains of living beings operate. Given a set of samples, neural networks are able to learn how to solve the corresponding task on their own.

Training a network often requires an extensive collection of inputs and can be a lengthy and resource intensive process. It isn't guaranteed to yield the expected results and might not be viable for critical applications that demand 100 % accuracy. Despite these drawbacks, neural networks are getting increasingly popular [3], especially for tasks like: classification, pattern recognition, function approximation and regression analysis.

Spiking neural networks (SNN) are a subset of artificial neural networks, focusing on a more precise simulation of biological processes. They mimic the spiking behavior of excitable cells, like the neurons in human brains. On one hand, the increased biological plausibility of such models could help us understand the nature of life. On the other

hand, more abstract implementations have the potential to improve the abilities and performance of existing technologies.

In this work, we discuss the possible advantages of using FPGAs for the simulation of SNNs. We will introduce a modular implementation, providing the basic building blocks for constructing networks, and a hardware/software framework for embedding SNNs into real-world applications.

We will employ the framework to estimate latency, performance and resource utilization for any given network structure. On one hand, we provide the exact number of clock cycles per simulation interval, depending on the number and arrangement of neurons. On the other hand, we give an estimation of the required resources based on a multitude of measurements, performed in various network configurations.

Our absolute performance metric will allow developers to compare a potential FPGA implementation to other options. Furthermore, will the estimated resource utilization assist in the selection of appropriately sized FPGAs.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Background

This chapter gives a basic introduction to artificial neural networks and spiking neural networks in particular. Later, we take a look at FPGAs, discuss their viability and inspect their structure and components.

2.1 Artificial Neural Networks

Artificial neural networks (ANNs) are mathematical models inspired by the structures and processes observed in the brains of animals. Similar to their biological counterparts, they are compositions of nodes organized in complex networks. Following the conventional nomenclature, nodes are referred to as neurons, whereas the structures interconnecting them are called synapses.

Artificial **neurons** are elementary units, each processing an input signal according to its underlying mathematical model, before propagating the results towards all neurons connected to its output. The non-linear transfer function characterizing a neuron is also known as its activation function.

Neurons are connected through **synapses**, which shape and combine the output signals of multiple neurons to provide input for another neuron. Synapses are based on mathematical models as well. They can emulate complex electrochemical processes, but more often simply compute a weighted sum.

Figure 2.1 shows a generic model of a neuron. The synapses scales the incoming signals x according to the synaptic weights w before combining them. The resulting signal is then transformed by the neuron's activation function f and propagated to its output y.

Figure 2.2 shows a representation of a small network. Its neurons are grouped into **layers**, visually arranged in columns. Each neuron's input is composed from outputs of the previous layer. There are no connections within a layer or across multiple layers. Since all possible connections exist, the layers are considered **fully connected**.

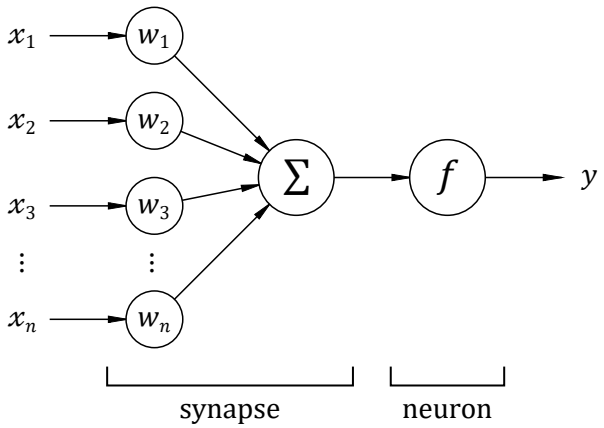

Figure 2.1: Generic model of synapse and neuron

The first (leftmost) layer is the **input layer**. Neurons of the input layer are placeholders, whose output is injected by an external source. The last (rightmost) layer is the **output layer**. Its neurons perform normally, but their output is considered the overall output of the network. All layers in-between are called **hidden layers**, because their signals aren't visible from the outside.

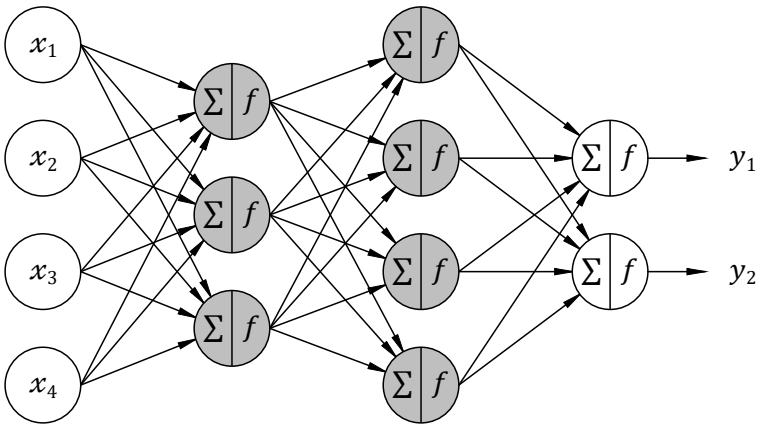

Figure 2.2: Example of a Neural Network

Each neuron and each synapse has a set of adjustable parameters. The available parameters depend on the underlying models. A common occurrence are the connection weights stored within the synapses. Fine-tuning these and other parameters for each entity defines the overall behavior of the network. A well-structured network with a capable model can be configured to solve astoundingly complex tasks. ANNs have

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

been used in many fields, most notably for classification, pattern recognition, function approximation and regression analysis.

Reasonably large networks, suitable for solving complex task, have a huge amount of configurable parameters. Due to the inability to properly adjust them manually, another concept of nature is emulated – the process of learning [20]. A specifically designed algorithm repeatedly challenges the network and subsequently tries to modify parameters to improve the accuracy of its output. Thereby a general structure with randomly initialized parameters can be trained by providing it with examples of a problem and its intended solutions.

Learning oftentimes requires a huge amount of input samples where the desired output is known. Depending on the complexity of the task, it can be a lengthy process that occupies computational resources while consuming significant amounts of energy. In return, a well-trained network can not only handle the data it was trained on, but also provides good approximations for inputs it hasn't experienced before.

2.1.1 Spiking Neural Networks

Spiking neural networks (SNNs) are a type of ANN that focus on a specific aspect of biological brains [19], [15]. In SNNs information is transmitted through spikes, rather than numerical values. Spikes are events triggered by neurons, that propagate to connected neurons, encoding information solely through their distribution over time.

In an SNN, synapses combine the spikes from multiple outputs into a continuous input signal. A simple implementation amplifies the signals according to weights associated with each source and output the combined sum. In more complex models, synapses behave like an RC circuit filtering the signal, and introduce delays, jitter or noise. In general, a synapse models the way spikes propagate and how big of an incentive they are for other neuron to generate spikes of their own.

For neurons, the Hodgkin–Huxley model [12] is well known for closely approximating the behavior of excitable cells. It describes the development of the membrane voltage V_m based on an input current I according to the electric circuit depicted in figure 2.3.

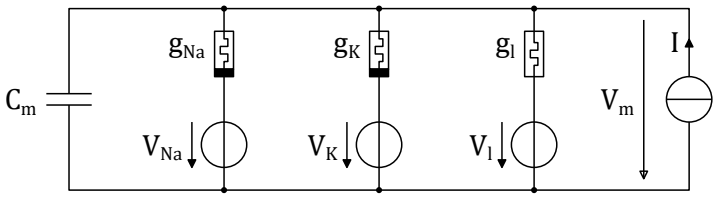

Figure 2.3: Schematic of the Hodgkin–Huxley Model

The behavior of the circuit can be described by a set of differential equations. Alan Hodgkin and Andrew Huxley were able to predict the initiation and propagation of action

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

potentials in the squid giant axon and received the 1963 Nobel Prize in Physiology or Medicine for their work.

$$I = C_m \frac{\mathrm{d}V_m}{\mathrm{d}t} + \bar{g}_{\mathrm{K}} n^4 (V_m - V_{\mathrm{K}}) + \bar{g}_{\mathrm{Na}} m^3 h (V_m - V_{\mathrm{Na}}) + \bar{g}_l (V_m - V_l)$$

$$\begin{aligned}
\frac{\mathrm{d}n}{\mathrm{d}t} &= \alpha_n (V_m) \cdot (1 - n) - \beta_n (V_m) \cdot n \\
\frac{\mathrm{d}m}{\mathrm{d}t} &= \alpha_m (V_m) \cdot (1 - m) - \beta_m (V_m) \cdot m \\
\frac{\mathrm{d}h}{\mathrm{d}t} &= \alpha_h (V_m) \cdot (1 - h) - \beta_h (V_m) \cdot h
\end{aligned}$$

It is not always the goal to accurately simulate nature. Thus we will later introduce another more elementary model, which is better suited for optimization.

2.2 Field-Programmable Gate Arrays

Field-programmable gate arrays (FPGA) are a type of integrated circuit (IC). They fill a niche between microcontrollers and application-specific integrated circuits (ASIC).

Microcontrollers are general-purpose integrated circuit, which performs a task by executing a programmable sequence of predefined operations. There are many different families, focusing on various fields of applications. Some are fast and power-hungry; others are slow but very energy efficient. Models can specialize on signal processing and some even offer accelerators for neural networks.

In scenarios where microcontrollers won't meet the requirements, ASICs become an option. They are purpose-built for a specific application. Unfortunately, designing ASICs on the circuit level is a complex and time-consuming process, followed by a significant investment in having them manufactured.

These options sometimes results in situations, where microcontrollers aren't well suited and ASICs would be too expensive. In these cases, FPGAs can be a viable option. They contain an array of logic blocks and interconnects, that can be configured to form virtually any digital circuit. Hence, they provide design freedom down to the circuit level, combined with a convenient development cycle using programmable off-the-shelf components.

2.2.1 Structure and Components

The structure of most FPGAs follows a very similar basic concept, but the nomenclature varies from manufacture to manufacture. For illustration purpose, let's take a closer look at the Xilinx 7-series of FPGAs. Figure 2.4 shows one of eight regions within an

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

XC7A100T. The majority of the area (blue) is filled with configurable logic blocks. Also strewn in are some other useful components like block RAM (red) or DSP slices (green). On the edge, we see clocking resources (yellow) and interfaces to I/O pads (grey).

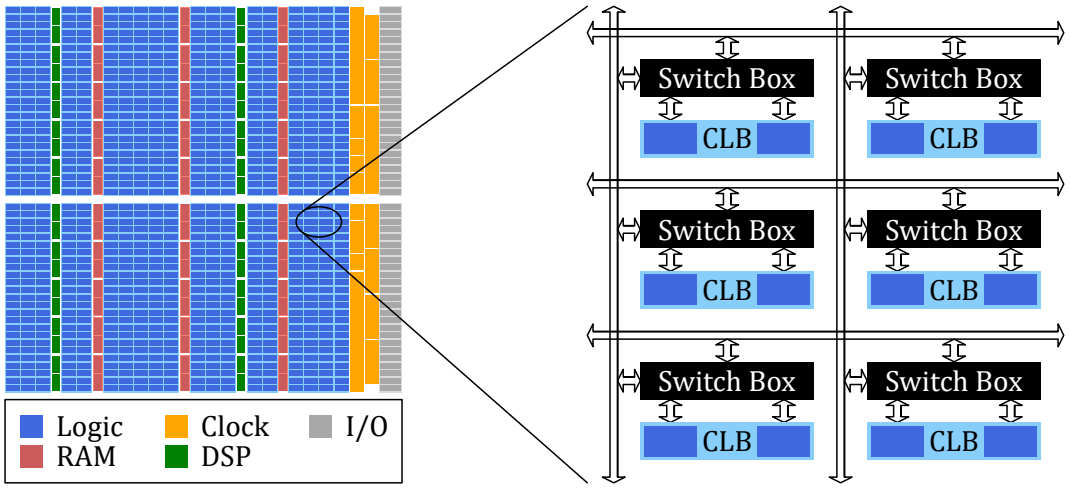

Figure 2.4: Layout of an FPGA Region

All components are arranged on a grid with switch boxes connecting them to the horizontal and vertical busses of wires running in between. Programming an FPGA configures not only the individual components to perform basic logical functions, but also their interconnections through the switch boxes. Using FPGAs ultimately enables us to synthesize complex digital designs without the costly process of manufacturing custom ICs.

After getting familiar with the overall structure, let's take a closer look at the most important components.

Configurable Logic Blocks

Figure 2.5 shows a simplified version of a configurable logic block (CLB), consisting of the following elements:

- A **look-up table** (blue), built from a small memory that stores the truth table of some combinatorial logic. When addressing the memory, it applies the logical function and outputs the result. The memory content is part of the FPGAs configuration and enables the implementation of time-independent logic.
- The **full adder** (green) is an optional component, found in many modern FPGAs. Circuits for adding or subtracting binary numbers are a common occurrence in most

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

digital designs and providing optimized resources oftentimes results in a significant performance boost.

- A **D flip-flop** (red) memory element that stores a value whenever triggered by its clock signal. The signal used as input is determined by the FPGAs configuration. DFFs can be bypassed when building combinatorial logic across multiple CLBs.

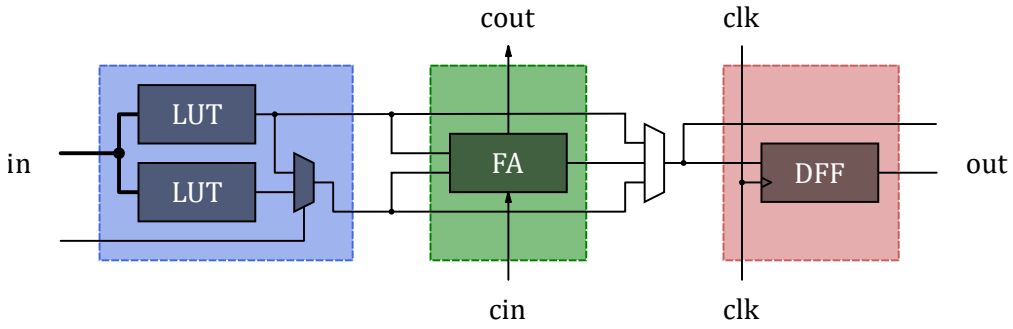

Figure 2.5: Simplified circuit of a Configurable Logic Block

Actual CLBs, like the ones of the Xilinx 7 Series FPGAs, depicted in figure 2.6 [23], are a little bit more complex. A CLB contains a pair of slices, each composed of four 6-input look-up tables, a 4-bit carry chain and eight storage elements.

The 6-input look-up tables have a second output, enabling them to represent two individual 5-input functions that share the same inputs. It is also possible to pair look-up tables for 7-input functions or combine all four of them into an 8-input function.

Utilizing the 4-bit carry chain, a single slice can implement a 4-bit addition or subtraction. The carry-in and -out signals are connected to the adjacent slices through dedicated wires. This structure allows for low-latency carry chains across multiple cascaded slices, that could otherwise bottleneck a design.

The storage elements can be configured as flip-flops or latches. Their inputs either connect to an external signal, one of the look-up table's outputs or the result of the carry-chain. A slice also features several output signals, which can either source from the storage elements or directly from one of their input options.

The XC7A100T features 15,850 logic slices with a total of 63,400 look-up tables and 126,800 flip-flops.

Clocking and Interfaces

Most FPGAs are optimized for synchronous logic, focusing on storage elements, triggered by a global clock signal. Therefore, they provide high speed clock distribution trees with

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

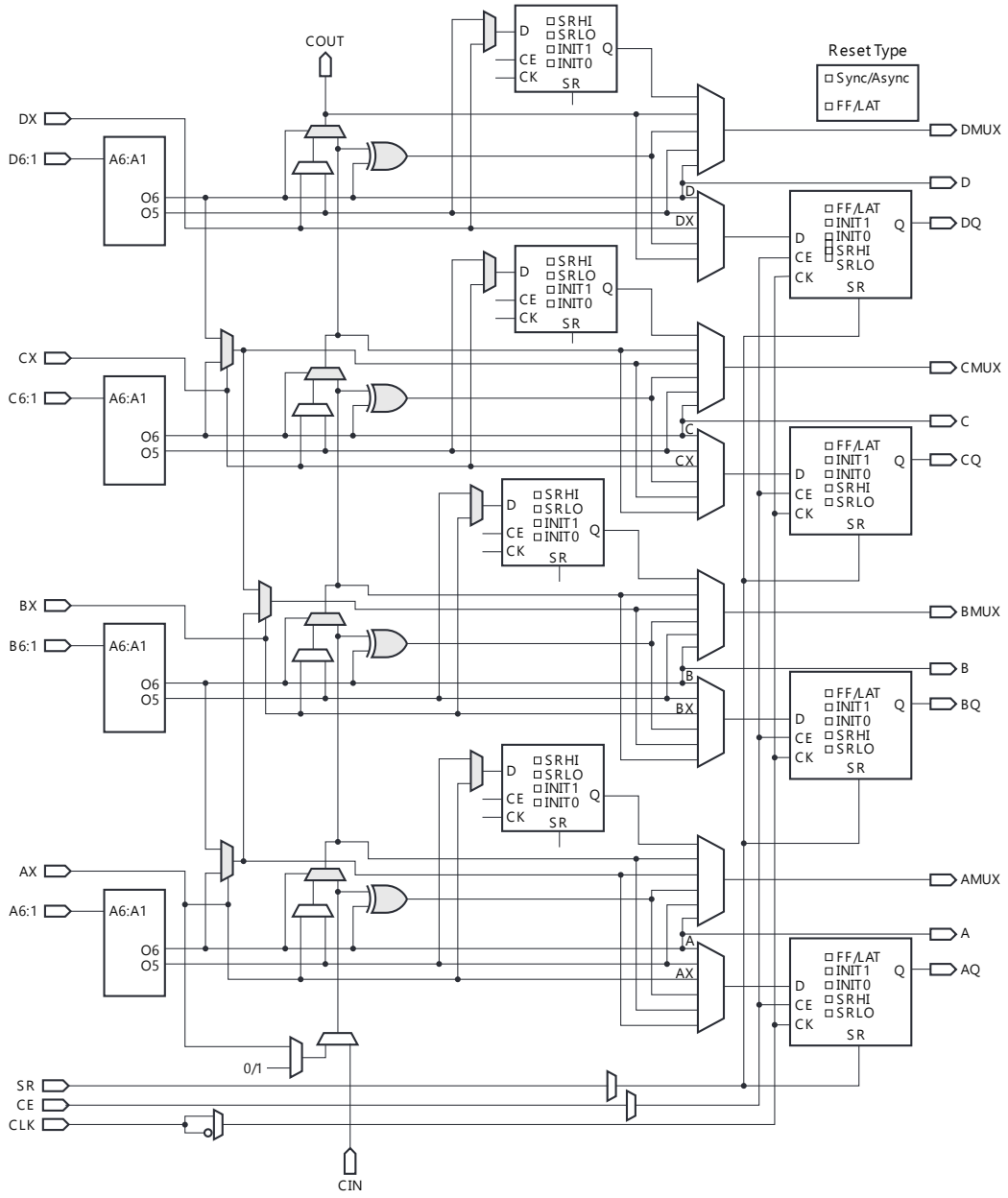

Figure 2.6: Configurable Logic Block of Xilinx 7 Series FPGAs

high fan-out, short propagation delay and minimal skew. In addition, resources like mixed-mode clock managers (MMCM) and phase locked loops (PLL) provide a means to synthesize clock signals with specific frequencies and phase relations.

An IC can only be useful when processing data. Thus FPGAs provide a large amount of general-purpose IO-pins. Sometimes, even analog-to-digital and digital-to-analog

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

converters or dedicated interfaces for accessing external memory, e.g., DDR-RAM, or peripherals, e.g., PCIe, are available.

Memory Resources

A lot of applications require memory, but look-up tables and flip-flops can only provide small amounts. To overcome this limitation, many FPGAs have columns of dedicated memory blocks mixed into their array. They are faster than distributed memory built from CLBs and take up less area of the IC.

The Xilinx 7 Series FPGAs incorporate dedicated block RAM resources. Depending on the specific model, devices offer between 5 and 1,880 RAMB36 primitives, each capable of storing up to 36 Kb. RAMB36 blocks can be used in different configurations: 32K x 1, 16K x 2, 8K x 4, 4K x 9, 2K x 18, 1K x 36, as well as 512 x 72. It is also possible to combine 2 RAMB36 blocks, to form a single 64K x 1 memory without the use of any additional resources. Furthermore RAMB36 primitives can be split into pairs of individual RAMB18 blocks configurable to 16K x 1, 8K x 2 , 4K x 4, 2K x 9, 1K x 18 or 512 x 36.

As discussed earlier, look-up tables are a type of memory that stores the truth table of a logical function. Usually the memory is programmed during the configuration phase of the FPGA, but in Xilinx 7 Series FPGAs a portion of the memory can also be accessed by the digital design. Slices, whose look-up tables can also function as distributed memory are referred to as SLICEM, whereas pure logic slices are called SLICEL. With 32 x 2 bits of memory in every 6-input LUT, the 4 LUTs within a SLICEM can store up to 32 x 8 bits of data.

The XC7A100T offers 135 blocks of dedicated RAM with a total capacity of 4,860 Kb. Furthermore 4,752 of its 15,850 slices can be used as distributed RAM, offering an additional 1,188 Kb of memory.

Signal Processing

Another feature are digital signal processing (DSP) slices, that speed up frequently used arithmetic operations. They are especially useful for multiplications that would take up a lot of CLBs. Using a compact and dedicated circuit reduces resource utilization and increases the maximum achievable clock frequency.

2.2.2 Toolchain

Having tens of thousands up to millions of look-up tables, flip-flops and other configurable resources makes it impossible to handle them without some level of abstraction. Creating an FPGA design works slightly different from writing software. Instead of using a programming language, digital circuits are defined using a hardware description language (HDL). The code mainly states where to use memory cells and what combinatorial

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

logic connects their inputs and outputs. Thus, forming functional entities that can be instantiated and combined to eventually create a design that performs the desired task.

Another step in software development would be testing and debugging. When creating a hardware design, we can simulate from the individual components up to the design as a whole. This has the advantage that any intermediate signal which would be buried deep inside the physical circuitry can be observed. Running a simulation requires harnessing the component using a testbench that stimulates its inputs and compares the output to a reference.

The implementation process is automated and involves the following steps:

- Synthesis: The described hardware is translated into a digital circuit and mapped onto the target technology, using the components available on the selected FPGA.
- Placement: Every component is assigned a location within the array of the FPGA. The tools employ strategies to place highly interconnected parts close together, keeping the signal delays small.
- Routing: The placed components are connected through the available switches and wires. The tools take care to meet the timing constraints and verify functionality at the desired clock frequency.

The whole process is based on heuristics and might take several iterations until the design fits onto the FPGA while also meeting the given timing constraints. The final result is a bit stream which can be used to program the FPGA.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Related Works

Spiking neural networks are based on functional models of the physical processes within neurons. The well known Hodgkin–Huxley model [12] describes an excitable cell using an equivalent electric circuit. There have been a lot of improvements [8] and alternatives [16, 18] to the Hodgkin Huxley model, that are even more accurate in simulating the actual behavior of neurons. On the other end of the spectrum, there are also more loose interpretations, that aim to preserve the general properties, but reduce the complexity [13] of the calculations. Most implementations fall back to some variation of the leaky integrate-and-fire model and a conservative structure of multiple fully connected neuron layers.

It has been shown, that various training methods applicable to classical ANNs (e.g. backpropagation [17]) are also applicable for SNNs. Furthermore an implementation [6] shows the viability of a biologically more plausible approach called spike-timing-dependent plasticity.

There are numerous software frameworks supporting the simulation of SNNs (e.g. Brian2 [21], snnTorch) with various levels of optimization and hardware acceleration. Some projects (e.g. SpiNNaker [9]) aiming to simulate large scale SNNs directly in hardware employ a large amount of computational nodes composed of classical microprocessors to form a purpose built supercomputer. Others (e.g. Neurogrid [2]) attempt to perform parts of the simulation using analog circuitry.

FPGAs can be used to simulate very specific networks more efficiently than generic computer hardware. Recently there have been attempts to implement SNNs in FPGAs [4]. While it is proven that FPGAs can simulate small- to mid-scale networks, each implementation is very specific. Estimating the hardware utilization and performance of SNNs prior to implementation can be a challenging task.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

Methodology

This chapter introduces a digital design for hardware accelerated simulation of spiking neural networks in FPGAs. The implementation is split into two main components – neurons and synapses – that can be instantiated and arranged to form interconnected networks. In addition, there are modules to properly format input and output values using a Poisson encoder and rate decoder, respectively.

The resulting networks are not meant to be trained while running on an FPGA. Rather, training shall be performed offline on a more conventional system, whereafter computed parameters, e.g., connection weights, are transferred into the FPGA design. Pre-trained networks are easier to optimize and thereby require fewer resources while yielding better performance.

The implementation simulates an SNN for a predefined amount of time. It divides the simulation time into small discrete steps - so called clock ticks. Every time the simulation clock advances, the network updates the state of its components. The implemented modules are designed to update their state within a constant amount of time, making the overall run-time independent of the applied input or internal state of the network. It solely depends on the network structure, making the design suitable for real-time applications.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.1 Neurons

The implementation is modular and able to easily adopt a variety of neuron models. For the scope of this work, we focused on the leaky integrate-and-fire (LIF) model [1], according to the schematic in figure 4.1. Although not biologically plausible, LIF incorporates the core concept of neurons in the simplest possible way. It is a staple in most every simulation framework available and has been successfully used in many applications.

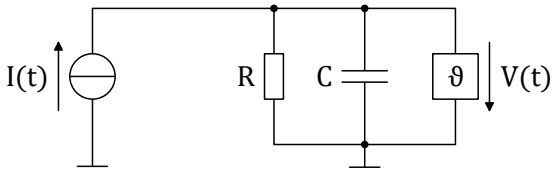

Figure 4.1: Schematic of a neuron based on the Leaky Integrate-and-Fire model

In LIF, neurons are represented by their membrane voltage V. Applying an input current I charges the capacitor C. At the same time, some of its charge leaks through the parallel resistor R. The model's differential equation looks like:

$$I(t) = C\frac{\mathrm{d}V(t)}{\mathrm{d}t} + \frac{V(t)}{R}$$

It describes the development related to the time-dependent input current I until the membrane voltage V reaches a constant threshold ϑ. Once activated, the neuron generates an output spike using the stored charge and subsequently resets to its initial state.

Figure 4.2 shows an example. The neuron processes an input current consisting of differently scaled spikes that cause the output voltage to rise. During periods of low input activity, the voltage drops due to the internal leakage. Once the threshold is reached, the generated output spike causes the voltage to reset to its initial value.

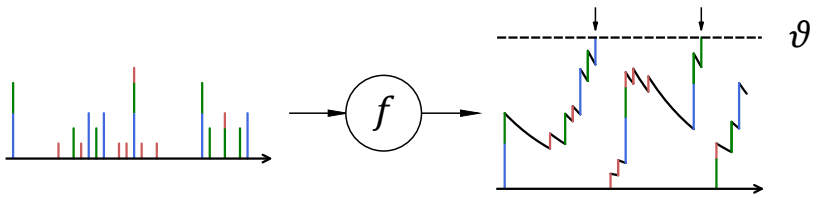

Figure 4.2: Example of a neuron's internal voltage and output spike generation

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

4.1.1 Implementation

Considering clock ticks having a duration Δt and an initial membrane voltage $V(t_0) = V_0$, the value at $t_{n+1} = t_n + \Delta t$ is approximated as:

$$V_{n+1} = \left(1 - \frac{\Delta t}{CR}\right) V_n + \frac{\Delta t}{C} I_n$$

In this equation, $1 - \frac{\Delta t}{CR}$ can be interpreted as a constant factor β describing the leakage, whereas $\frac{\Delta t}{C} I_n$ represents the voltage change caused by the input current.

Choosing Δt is a tradeoff between preserving the accurate behavior of the model and faster, more efficient computation. While a bigger Δt results in the loss of temporal resolution, small values can unnecessarily increases the number of computational steps. Instead of directly specifying Δt, the module is parameterized with a value β and the simulation time is given by the number of clock ticks.

Scaling the input by the constant factor $\frac{\Delta t}{C}$ is offloaded to the preceding synapses, where it can be factored into the connection weights. Thus, updating the membrane voltage requires scaling the previous value by β before adding the preprocessed input.

4.1.2 Optimization

In the digital design, we group multiple neurons into a single module, which has the following advantages:

- In FPGAs multiplications are expensive operations. They tend to be either slow or require an extensive amount of resources. Having multiple neurons share their signal processing circuitry saves resources and significantly increases the number of neurons that can be simulated.
- Sharing an input and output interface reduces the amount of logic needed for synchronization and handshaking. Instead of a single input and output, the module receives a stream of input values and generates a stream of output values.
- Time multiplexing the computational resources also enables the efficient storage of the neuron potentials in dedicated memory blocks. A cyclic counter addresses the state (membrane voltage) of the neuron whose input (electric charge) is currently received through the interface. After processing the neuron, the updated state is written back to the memory and stored until the next iteration.

In the implementation, the initial value equals 0 and the threshold 1. The input is a fixed-point value in the range $[-1, 1]$. Even though negative inputs are permitted, the membrane voltage is reset every time it drops below 0. The computation requires a single multiply-accumulate operation followed by a comparison with the threshold. If the value exceeds 1 it is reset and a spike is generated at the output.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.2 Synapses

The implementation aims to simulate a layered network, where the neurons of consecutive layers are fully connected and each neuron's output is considered an input to all neurons in the following layer. The amount of influence, a specific input has on a neuron's behavior is determined by the connection's weight, stored within the synapses.

Figure 4.3 shows an example of signals propagating through a synapse. The different spike trains are scaled according to their weights before being accumulated into a combined signal.

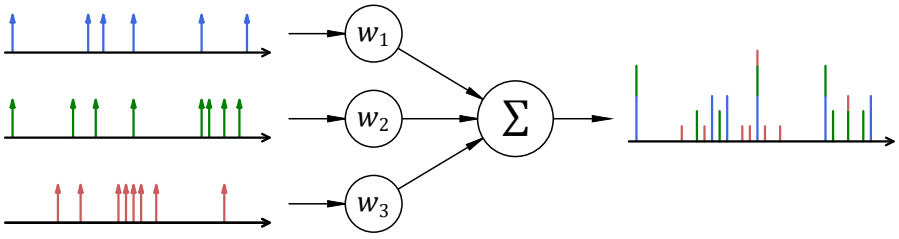

Figure 4.3: Example of synapses combining multiple spike trains

4.2.1 Implementation

Considering an N-wide layer, whose neurons indicate spikes with Boolean values $x_1 \dots x_N \in \{0, 1\}$ and a set of constant weights $w_{j,1} \dots w_{j,N}$ specifying their significance for a neuron j of the following layer, the synapses calculate the neuron's input y_j using:

$$y_j = \sum_{i=1}^{N} w_{j,i} x_i$$

4.2.2 Optimization

The module represents the interconnection between two layers of neurons. Grouping all synapses connecting two adjacent layers has the following advantages:

- The task of the synapses between two layers is computing multiple weighted sums. Since the same calculation is repeated several times, there is potential for sharing hardware resources.
- All computations are based on the same set of inputs coming from the preceding neuron layer, while all results are forwarded to the succeeding neuron layer. Sharing interfaces reduce the complexity of inter-module communication as well as the amount of logic needed for synchronization and handshaking.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- Having more weights and thus more data to store within a single module results in efficient use of memory resources. This addresses the issue of FPGAs having memory blocks of fixed sizes, that can't be split or shared between modules. Having a single module utilizing multiple memory blocks, with only a portion of the last block unused, is preferable to having empty padding in a lot of smaller modules.

The module's input interface accepts a sequence of boolean values, each representing the presence or absence of a spike from one of the neurons within the preceding layer. This approach guarantees a constant amount of input and thus a constant execution time of the module. Internally, processing a single input requires loading a set of weights. Since the neuron, that generated the current input, is known implicitly from the value's position within the input stream, the memory storing the weights can be addressed using a cyclic counter.

The computation of weighted sums is parallelized using an array of accumulators - one for each output. The accumulators add individual weights, if and only if the corresponding input indicates the presence of a spike. The synchronous computation leads to the completion of a full set of outputs after receiving a full set of inputs. To simplify the output interface, values are serialized using a shift register.

To approximate the spiking behavior of a neural network, the described process of computing weighted sums is repeated in each clock tick the simulation advances. The accumulators are automatically reset after a set of computations concludes. The module's state machine handles handshaking at the interfaces and ensures a seamless operation and constant flow of data.

The design is pipelined to achieve maximal throughput while, at the same time, being able to perform at high clock frequencies. The module is optimized to handle one input per clock cycle and generates one output per clock cycle as long as the connected layers have the same number of neurons. When the input layer is wider, the shift register will be empty before a new set of outputs gets available. Thus the design can only provide output after receiving enough input. Conversely, with a wider output layer, the shift register won't be empty before the accumulators finish a new set of outputs. In consequence, the input is pushed back until the output has been transmitted. In summary, the module is designed to always keep up with its interfaces and is only limited by their throughput.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

4.3 Poisson Encoder

The implementation provides a Poisson encoder to assist in rate-encoding input signals. It converts intensity values in the range $[0, 1)$ to spike trains, where spikes have an average rate proportional to the input intensity, while the interval between spikes is Poisson-distributed.

Another popular alternative is latency-coding, where the information is encoded in the precise timing of spikes rather than their average rate. Most commonly a higher intensity causes an earlier spike than a lower intensity value. We opted for rate-encoding because it can generate inputs over an arbitrary timespan and is potentially more robust due to the repetition of the input signal.

The modular nature of the design allows to replace the input layer with a latency-codec or any other kind of more complex coding system, like for example temporal order coding [10].

4.3.1 Implementation

The behavior is approximated by generating a random number in the range $[0, 1)$ and comparing it to the intended rate. If the rate is smaller than the random number, a spike is generated. Repeating the process for every clock tick, results in the desired output. The spike trains depicted in figure 4.3 were generated using this method. The blue, green and red signals have an intensity of 1/8, 1/6 and 1/4. Short trains like these randomly deviate from the requested rate. Only when looking at longer periods, they would approach the desired rate and distribution [11].

The implementation is based on a modified version of Knuth's subtractive random number generator algorithm [14] used in Microsoft's .net framework. Although this might not be the most popular choice for a hardware design, it is easy to implement and generates the exact same sequence as the prototype used for verification of the design. The algorithms downside is its limitation to generating exactly 31 random-bits per cycle. Since most networks will only require a single instance we accept the slight overhead, as a consequence of sticking with the algorithm.

4.3.2 Optimization

The module represents an input layer of multiple neurons that share a common random number generator. At the beginning of a simulation, it requests and buffers an intensity value for each neuron. Afterwards it repeatedly generates random numbers and compares them to the buffered values. The results, stating whether the corresponding neurons generate spikes, are forwarded to the next layer. After cycling through all neurons the simulation advances a tick and the process repeats until the simulation ends.

The design is optimized to match the speed of the connected synapses. It processes one neuron per clock cycle and generates its output for the current tick. The implemented

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

handshaking at the interfaces guarantees that input is accepted as early as possible and a steady stream of outputs is provided as long as the succeeding modules can handle it. Additionally, when using a FIFO, multiple sets of inputs can be queued, enabling seamless execution without any overhead between simulations.

4.4 Rate Decoder

A rate decoder provides a way to interpret the network output. It translates the spike trains of the output neurons into rates. To do so, the module simply counts the number of spikes each output neuron generates throughout the simulation. The computed values are directly proportional to the rates.

The rate decoder is ideally suited for our system, processing rate encoded inputs and running for a predefined duration. Another method would be interpreting the time until the first spike occurs. In that case the first output neuron to spike is considered the winner. This method, although it might be less robust, shortens the simulation time depending on how fast a specific input causes an output spike.

The current decoder is a placeholder that allows efficient testing of networks. If required, the modular design allows for easily replacement with other decoding systems.

4.4.1 Optimization

The module instantiates a counter for each output neuron. At the beginning of a simulation, the counters are reset to zero. Every time a spike registers the corresponding counter increments by one. During the last tick of the simulation the final values are presented at the module's output.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Experimental Setup

This chapter introduces the experimental setup used for testing and benchmarking the previously outlined digital design. It consists of a hardware framework providing programmatic access to an embedded SNN in combination with a software application that utilizes a pre-trained network to classify handwritten digits.

5.1 Hardware Framework

The hardware framework is specifically designed for the Digilent Nexys 4 FPGA board [7] depicted in Figure 5.1. The Nexys 4 is built around a Xilinx Artix-7 FPGA (XC7A100T-1CSG324C) [24], [25], which is considered a mid-range FPGA with a good price-performance ratio. The framework only uses very basic features of the development board, thus, it should be easy to adapt to other Xilinx based platforms.

The framework aims to provide a computational platform with programmatic access to an embedded SNN. The most flexible approach is integrating the SNN into a minimalistic soft-core microcontroller. Software running on the microcontroller can then directly interact with the SNN, providing input data and subsequently retrieving the results. The big advantage of this design is the low-latency high-throughput interface to the SNN, which is ideal for measuring performance. Care must be taken, as the presence of the microcontroller inside the same FPGA might have negative effect on the performance of the SNN.

The framework consists of a block design composed of various components from the IP repository provided by Xilinx with its development environment Vivado. The most essential components and their interconnections are depicted in Figure 5.2.

The Framework is based on the FPGA-optimized soft processor architecture MicroBlaze. The connected Debug Module exposes the MicroBlaze to the JTAG interface of the FPGA. Using JTAG not only allows resetting, halting and resuming execution of the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

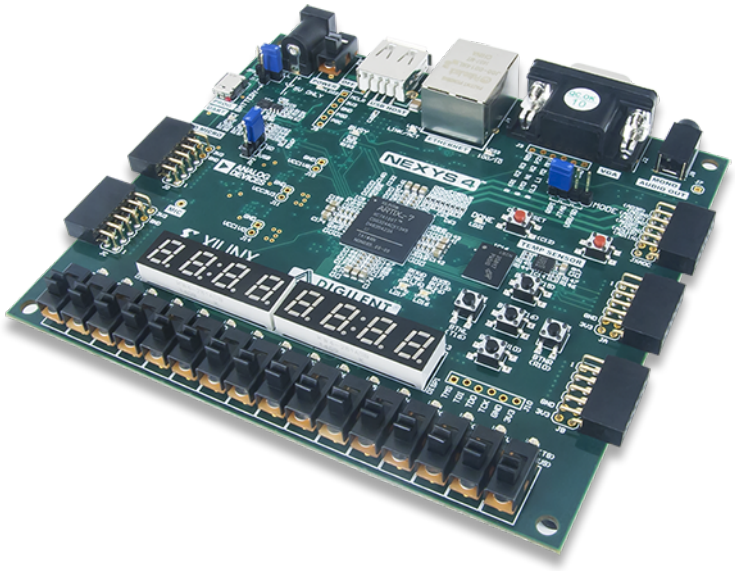

Figure 5.1: Digilent Nexys 4 FPGA Board

Figure 5.2: Block design of the framework

MicroBlaze processor, but also grants full access to its registers and memory regions. The block design also features an UART interface for redirection of standard input and output. The FPGA board conveniently integrates an adapter that exposes JTAG and UART on a single USB connector that also acts as a power supply.

From the framework's perspective, the SNN is encapsulated within a nested block design, having two streaming interfaces for its input and output data respectively. The MicroBlaze is configured with its own set of streaming interfaces, whose connection to the SNN are routed through FIFO structures. The FIFOs buffer data and thereby compensates for different data rates during read and write bursts. The software can access the streaming interfaces through dedicated assembler instructions.

To minimize negative effects due to the framework residing in the same FPGA as the SNN, it has been optimized to use as few hardware resources as possible. On one hand, this enables the use of additional resources to optimize performance of the SNN. On the other hand, unused resources grant the compiler more freedom when placing and routing the design, thus increasing the chances of fulfilling the timing requirements.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The instantiated soft processor and its peripheral components are configured to optimize for minimal area. Furthermore, the framework makes use of the external memory available on the FPGA board by instantiating an external memory controller. Thereby, the instruction and data sections of the processor are relocated into the 16 MiB of external memory provided by the Nexys 4 board, preserving the FPGA's internal memory blocks for the SNN.

The FIFOs not only compensate different data rates during read and write bursts, but also implement a means for clock domain crossing. Being able to run the framework at a considerably lower clock frequency than the SNN reduces the pressure on the synthesis tool. Having two clock domains ensures that the framework does not bottleneck the clock frequency of the SNN.

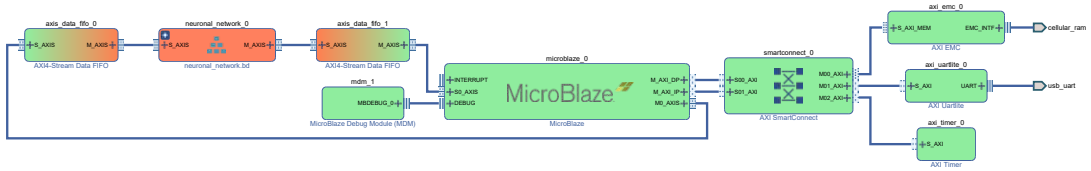

Figure 5.3: Visualization of different clock domains in the framework, with green meaning a more relaxed clock frequency requirement, while blocks in red try to maximize clock frequency

5.2 Software Application

In this section we introduce an example application that utilizes the implemented SNN to classify handwritten digits. The application will be the basis for testing and verifying the digital design as well as benchmarking its performance and resource utilization on FPGAs.

The following sections introduce a widely used database of handwritten digits and its general structure, as well as the input and output interfaces of the spiking neural network designed to classify them.

5.2.1 MNIST Database

The MNIST database is a large collection of handwritten digits. Each entry contains a 28 x 28 pixel grayscale image and is labeled with the digit it represents. The images are preprocessed to appear similarly sized and centered. Furthermore the pixel values are normalized for a uniform contrast and anti-aliased to smooth transitions between foreground and background. Figure 5.4 shows some examples taken from the database.

The database is popular for testing classification algorithms. It is split into 60,000 data points for training and an additional 10,000 data points for testing. By presenting the training set to an algorithm it can first "learn" about handwritten digits. Afterwards the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

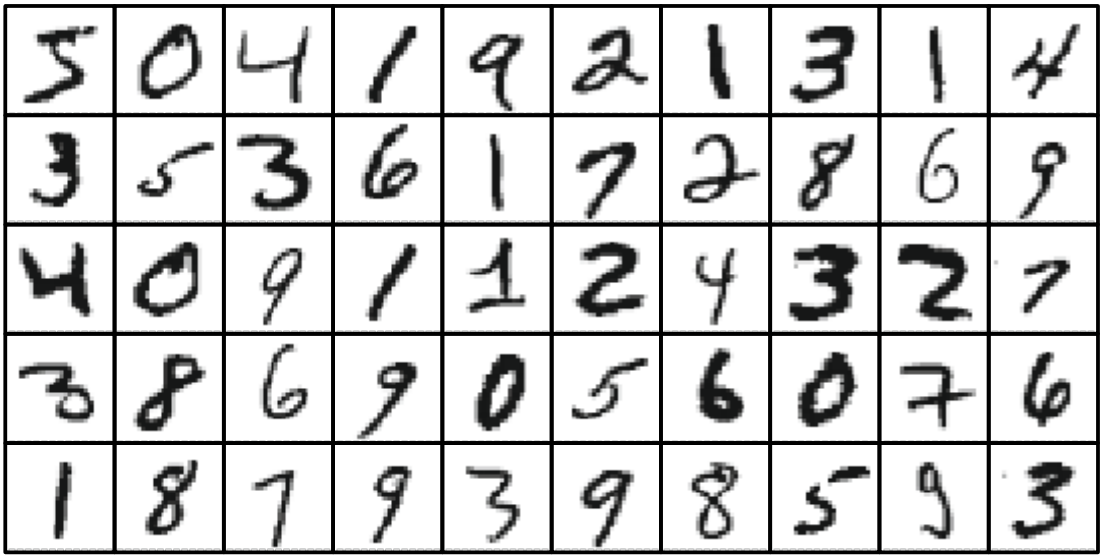

Figure 5.4: Examples of handwritten digits from the MNIST database

algorithm is tasked with classifying each entry of the test set and graded according to its success rate. Most machine learning algorithms reach success rates above 90% with the most optimized ones being correct about 99.5% of the time.

The database can be retrieved from its official website [1]. It comes in a custom file format that is simple, well documented and implemented in most machine learning framework.

5.2.2 Network Structure

The proposed application already outlines the basic structural requirements of the neural network. The input images consists of $28 \cdot 28 = 784$ individual pixels which basically dictates the width of the input layer. It would be possible to down-scale or crop the images to reduce the number of inputs, but for the purpose of evaluating and testing the implemented network, using 784 input neurons is perfectly reasonable. Furthermore, the task of classifying handwritten digits mandates an output layer of exactly 10 neurons that represent the digits from 0 to 9.

There is no restriction on the number of hidden layers or their respective widths. The only limitations arise later, when trying to fit the network onto an actual FPGA. An early estimation of the required hardware resources suggested one or two hidden layers with a combined total between 100 and 400 neurons. To stay on the safe side, the application implements one hidden layer containing 100 neurons.

Besides the network structure there are different methods of encoding the input and interpreting the output signals. The two most widely used methods are temporal coding

[1] http://yann.lecun.com/exdb/mnist/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

and rate coding. While temporal coding may lead to the network making faster decisions, it also comes with the downside of introducing varying simulation times. With temporal coding, the network concludes processing one input as soon as a neuron of the output layer generates a spike. Therefore, the runtime heavily depends on the input data. In contrast, using rate encoding, the network decides on the output neuron generating the most spikes in a constant amount of simulation time. This makes it the preferred choice for the purpose of testing our implementation by generating a constant workload independent of input data and prior training.

For the purpose of testing and verifying the implemented spiking neural network, we decided on the following structure:

- 784 input neurons and 10 output neurons.
- 1 hidden layer containing 100 neurons.
- use of rate encoding for input and output.
- a simulation time of 4096 ticks per input image.

Figure 5.5 depicts the general layout of the example application.

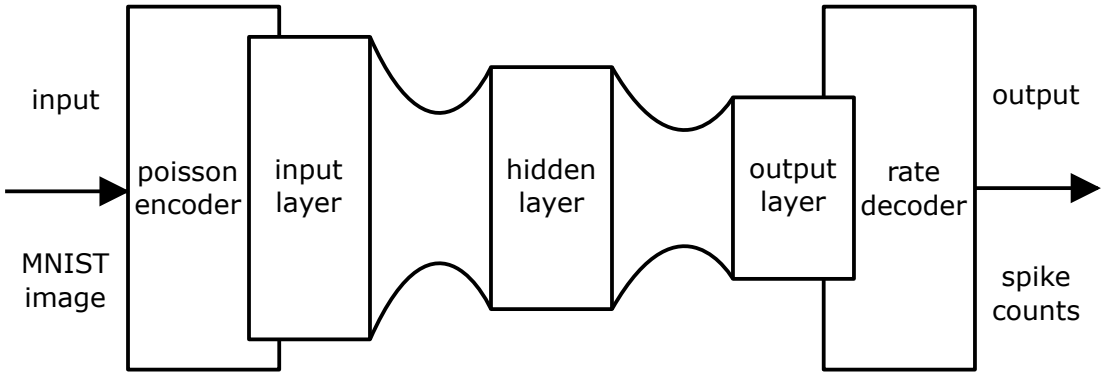

Figure 5.5: General layout of the Spiking Neural Network

Figure 5.6 shows the embedded block design containing the spiking neural network. It instantiates the previously introduced modules and parameterized the neurons layers to the previously agreed on widths.

The block design has been embedded into the previously introduced hardware framework. The overall system was successfully built to run at 250 MHz. Its functionality has been verified by comparing the individual spike trains to a reference implementation. It proved to work both, in simulation and on actual hardware.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

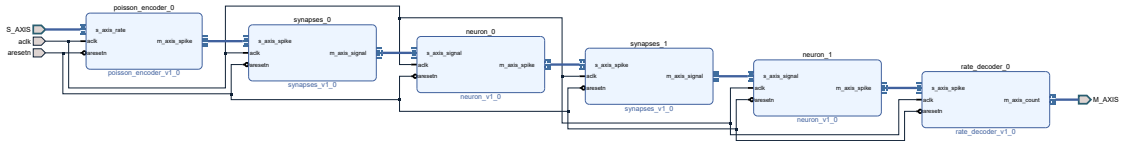

Figure 5.6: Block design of the Spiking Neural Network

5.2.3 Training

In the introduced design, the process of training adjusts the weights defining the interconnections between neurons. Currently, the weighs are the only parameters affected by training, although with the modular design, other parameters (e.g. threshold of LIF neurons) could be added in the future. The training is performed on a conventional software platform and the computed parameters are deployed onto the FPGA.

The design decisions for the current implementation ensure that the performance of a network is independent of its training. Changing the values of individual weights won't change the workload or number of clock cycles required to deliver results. The same holds true for the amount of hardware resources, as long as trivial cases (e.g. all zeros) are avoided. Therefore, measurements on performance and resource utilization can be carried out using untrained and randomly initialized networks.

Although, most measurements are accurate, even without prior training, testing and verifying a network's functionality as well as showcasing its functionality requires a valid set of connecting weights. For that reason, we trained the model on MNIST and transferred the weights on the FPGA memory.

The training was carried out using snnTorch [2], a Python package for performing gradient-based learning with spiking neural networks. The process is based on backpropagation through time using surrogate gradients [17] to overcome the challenges introduced by the non-differentiability of spikes. snnTorch provides a detailed tutorial at: `https://snntorch.readthedocs.io/en/latest/tutorials/tutorial_5.html`.

The sample included with snnTorch trains a network constructed around a hidden layer of 1000 neurons to classify the training set of the MNIST database. After completing the process, the network reaches an accuracy of 93.8 % on the test set. Reducing the hidden layer to 100 neurons causes a slight decrease in accuracy of about 0.5 %. Using the acquired values in our implementation drops the accuracy to 88.5 %.

The drop is due to some differences in the implementation of the neuron model. Our implementation is based on 18-bit fixed-point arithmetic, while snnTorch uses floating-point numbers. To account for the limited value range, the neuron potential is clipped

[2] `https://snntorch.readthedocs.io/en/latest/readme.html`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

to prevent negative numbers and the reset mechanism is slightly different to the default method used in snnTorch.

We expect a similar accuracy, close to 94 %, after backporting our implementation to snnTorch. But since this works focuses on pretrained networks in FPGAs, we will consider optimizing the training method, as well as evaluating other, potentially better methods, a topic for future work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

Results

In this chapter we showcase the models and the actual testing results for the implementation of SNN on FPGA using our framework.

6.1 Performance

One of the main reasons for porting spiking neural networks to FPGAs is the expected increase in performance. Compared to software implementations built upon general-purpose processors, FPGAs are capable of reconnecting basic hardware structures according to their configuration. This enables forming application specific circuits with a high level of parallelization. There are two decisive factors that play a role in the performance of hardware designs: the clock frequency, describing how many computational steps are executed per second, and the number of steps required, to complete the overall task.

The clock frequency depends on the amount of work scheduled for each computational step. The clock period has to be long enough so that all signal processing finishes before the cycle ends. The clock frequency thereby depends on the critical path, which is the signal having the most complex operation, taking the longest time to finish. The synthesis tools, responsible for mapping designs to hardware, aim to optimize the critical path by placing components closer together and thereby reducing the propagation delay of signals. Still, at some point the only method of increasing the clock frequency any further is reducing the complexity of operations.

The concept of pipelining splits complex operations into multiple smaller steps, whose sequential execution yields the final result. When a signal passes through a stage of the pipeline, the next signal can follow immediately. Although each operation takes multiple cycles to finish, the parallel execution of stages still produces one result per cycle. Ultimately, proper pipelining allows for higher clock frequency, while maintaining the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

same throughput per cycle. This comes at the cost of the signal being delayed for multiple clock cycles, as well as some additional hardware resources making up the pipeline stages.

The hardware platform chosen for testing (Digilent Nexys 4) is based on the Xilinx Artix-7 architecture with a speed grade of -1. Checking the AC characteristics in the data sheet (DS181), gives a basic idea of the theoretically achievable clock frequency. While clock buffers support a maximum of 464 MHz, other essential components, like for example block RAM, are limited to 388.20 MHz. In practice, these upper bounds are hardly ever reached, because the limiting factor is most certainly the signal propagation delay of the routed wires. Looking at an example, the reference guide (UG984) of the highly optimized MicroBlaze processor states a maximum frequency of 260 MHz for Artix-7 FPGAs of the highest speed grade.

After carefully fine-tuning and optimizing the pipelined structures, our design was able to pass all timing constraints at up to 250 MHz and slightly above. Considering the lower speed grade and the rather extensive designs, using up to 50-80% of the available hardware resources, further improvements are hardly possible and the additional resources required would be in no relation to the marginal gain in performance.

Let's define the task of the implemented spiking neural network in advancing the simulation one tick. Once the amount of clock cycles required is known, the runtime of any simulation will be predictable. While the problem is ideal for parallel processing, the amount of parallelism is limited by the available hardware resources. We were able to run each layer of the network synchronous, but the neurons within a layer are still processed sequentially. The optimized circuit is capable of processing one neuron per layer in each clock cycle. Thus, the limiting factor is the widest layer, requiring one cycle per neuron to advance a simulation tick.

In our example application the input layer has 784 neurons. As it is the widest layer, the whole network advances one tick every 784 clock cycles. With the rather arbitrarily chosen 4096 ticks to decide on a handwritten digit, the application is capable of analyzing one image every 12.8 ms when operating at 250.88 MHz. This purely demonstrates how to estimate performance for a given network.

6.2 Power Consumption

Measuring the exact power consumption of the introduced digital design is a complicated task. Since we are interested the most in the power consumption under load, the spiking neural network requires the hardware testing framework to provide it with input. As soon as both components are placed inside the same FPGA, it gets impossible to measure their power consumption separately.

Another challenge is our FPGA being situated on a development board, generously filled with other components. Some of them are necessary, i.e., power supply, others purely optional. Measuring the whole board includes all of its components and the bulk of them might very well exceed the power consumption of the FPGA on its own.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Since we neither focused specifically on power consumption, nor prepared an adequate measurement setup, we have to solely rely on the estimations by the development environment. The power consumption report of the hardware testing framework instantiating the spiking neural network of our example application running at 250 MHz is depicted in figure 6.1. We deduct that our design on FPGA uses very little power. The reported 0.83 W are split into 0.104 W of static and 0.726 W dynamic power consumption. Unfortunately the tool's confidence level is only low, due to the widely changing switching factors of the interconnected elements in the design, as they depend on the input pixels.

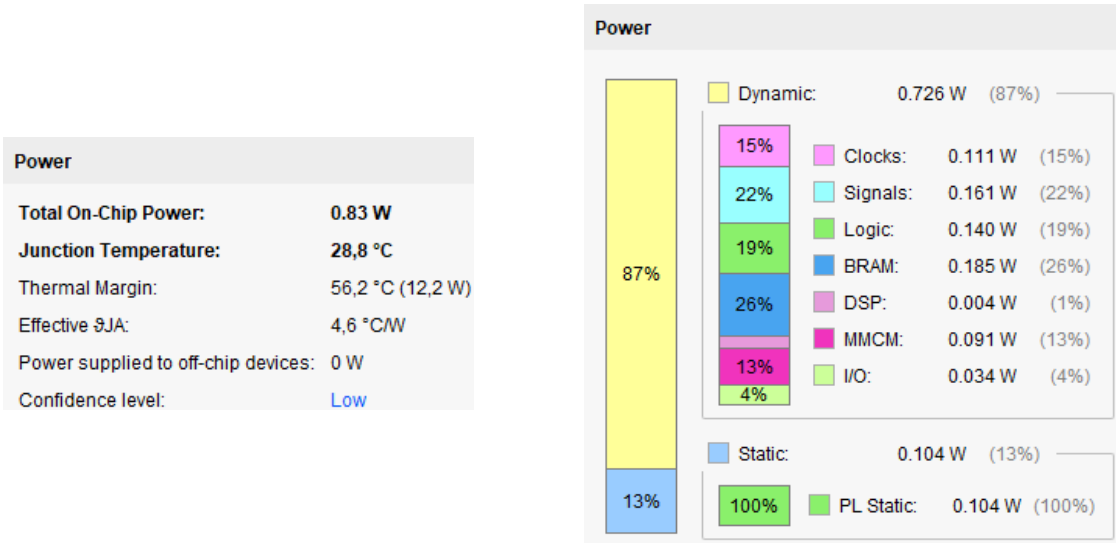

Figure 6.1: Power consumption report

The difficulty in estimating power consumption is that the majority is dynamic consumption, which strongly depends on the application. Transistors require dynamic power whenever they change their state. Thus, a network with a lot of spikes requires more energy than an idle circuit. The tools are hardly able to determine how active a circuit is on average use. Therefore, the values given are only rough estimates, which can be helpful in guiding the design choices, but should also not be considered absolute truth.

Our FPGA requires no cooling at all, giving confidence in a low power consumption. Compared to CPUs or GPUs with their cooling systems capable of dissipating multiple hundred watts of heat, FPGAs are more closely related to smaller, passively cooled microcontrollers. When aiming for a low power consumption, FPGAs and in extension ASICs are the most promising devices.

6.3 Resource Utilization

In this chapter, we analyze the amount of resource the components of the introduced digital design utilize, when synthesized in different configurations. There are three types

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

of resources that are of interest: logic resources like look-up tables, flip-flops and carry chains, dedicated memory blocks and digital signal processing slices. The amount of resources an instance requires depends not only on its parameters, e.g., number of parallel structures, bit-width of operands, but is also affected by the targeted clock frequency and the amount of optimization the synthesis tool applies during the implementation process.

6.3.1 Neurons

The *neurons* module represents a layer of N neurons. It receives a sequence of N weighed sums from the preceding *synapses* and outputs a series of N Boolean values, indicating the generation of spikes, to the succeeding *synapses*. Each input corresponds to a specific neuron and is processed according to the implemented leaky integrate-and-fire model. The module updates each neuron in turn, evaluating their activation conditions for output. Once all neurons are handled, the simulation advances to the next time tick, expecting a new set of inputs.

Each neuron is described by an internal state depending on the implemented model. In our design, the only variable we keep track of is the neuron's current membrane voltage. Since neurons are updated sequentially, all of their states can be stored in dedicated memory resources. Carefully following the basic design pattern, ensuring a single read and write operation in each clock cycle, enables the synthesis tools to automatically infer the proper amount of block RAM for us.

The leaky integrate-and-fire model is based around a multiply-add operation, required to update a neuron's membrane voltage. Unfortunately, multiplications tend to be resource intensive and slow, when implemented using logic primitives only. For this reason, many FPGAs incorporate dedicated hardware, supporting efficient multiplication as well as other commonly used mathematical operations. Our design explicitly requests the instantiation of the digital signal processing slices, available in the targeted FPGA architecture, to efficiently calculate the neurons' state according to their input signals.

Logic Resources

As discussed, our design heavily relies on dedicated memory and signal processing slices, only leaving the module's state machine to be built from logic primitives. The control circuit handles resets and handshaking signals, while also keeping track of the currently processed neuron, to correctly address memory. Altogether this is achieved by a rather small amount of logic that is mostly independent of the number of neurons in the module.

Table 6.1 lists the slice logic from the utilization report of all ($N = 100$) neurons within the first layer of our example application. Compared to the overall available resources and the requirements of other modules, the utilization of 46 slices or 0.29 % is hardly noticeable.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Site Type	Used	Available	Util%
Slice	46	15850	0.29
SLICEL	27		
SLICEM	19		
LUT as Logic	58	63400	0.09
using O5 output only	2		
using O6 output only	29		
using O5 and O6	27		
Slice Registers	185	126800	0.15
Register driven from within the Slice	58		
Register driven from outside the Slice	127		
LUT in front of the register is unused	114		
LUT in front of the register is used	13		
Unique Control Sets	13	15850	0.08

Table 6.1: Logic utilization report of neurons

Memory Resource

The internal state of our neurons consists of a single variable, storing the current membrane voltage. As for the value range, we clip the voltage when dropping below 0 and reset it once reaching the threshold value of 1, generating a spike in the process. There is no discernible advantage in using floating point numbers, because we are really just interested in a very limited range. Thus, we chose an 18-bit fixed-point representation for the range $[0, 1)$, resulting in a resolution of $2^{-18} \approx 3.815 \cdot 10^{-6}$. The number format closely relates to the precision of the weights stored within the synapses.

Block RAM is available in chunks of 18 kbit, configurable as 16K x 1, 8K x 2 , 4K x 4, 2K x 9, 1K x 18 or 512 x 36. Choosing a data width of 18 bits allows for 1024 neurons per BRAM18 primitive. In our example application, the widest layer contains 100 neurons, which uses about 10 % of a single memory block. As long as the layers do not fill a single block, reducing the bit length won't save any resources.

Table 6.2 shows the memory utilization reported for the implementation of the 100 neurons within the first layer of our example application. The output layer, even though having only 10 neurons, uses another RAMB18 primitive, bringing the total to 0.74% of the available block RAM resources.

Site Type	Used	Available	Util%
Block RAM Tile	0.5	135	0.37
RAMB18	1	270	0.37
RAMB18E1 only	1		

Table 6.2: Memory utilization report of neurons

Signal Processing Resources

The membrane voltage is calculated each simulation tick. First, the stored value is multiplied by a constant smaller than 1 (e.g. 99 %), which accounts for the leakage. Then

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the input from the synapses is added. When the new value exceeds the threshold, the module outputs a 1 to signal the generation of a spike. Otherwise no spike is generated and the module outputs 0. In case an underflow occurred and the value dropped below 0, it is reset. Likewise, an overflow and the resulting spike resets the value. Still in the range $[0, 1)$, it is again stored in memory before the computation of the next neuron begins. Once all neurons have been handled, the simulation advances a tick and the process starts over.

The available DSP slices can be configured to perform a multiply-add ($A \cdot B + C$) operation. They feature a 25 x 18 multiplier that is perfectly suited to scale our 18-bit values A with a constant factor (B before adding the current input C. Clipping negative results, as well as performing a comparison to the threshold can easily be done with some supplementary logic.

Table 6.3 shows the DSP utilization reported for the implementation of the first layer from our example application. Within a layer, neurons are updated sequentially. Therefore each layer requires a single DSP slice. Considering the output layer, the overall utilization totals at 2 DSP48E1 primitives or 0.83 % of the available resources.

Site Type	Used	Available	Util%
DSPs	1	240	0.42
DSP48E1 only	1		

Table 6.3: DSP utilization report of neurons

6.3.2 Synapses

The *synapses* module represents the interconnection between two consecutive layers of neurons. It receives a sequence of Boolean values, indicating the generation of spikes, from the preceding N-wide neuron layer and outputs a series of weighted sums to the succeeding M-wide neuron layer. The width L_w of the stored connection weights can be adjusted in the modules parameters, while the width $L_s = L_w + \lceil ld(N) \rceil$ of the weighted sums adapts for the highest possible output value.

Logic Resources

The synapses module instantiates M accumulators with a bit-width of L_s. Figure 6.2 shows the basic structure of an accumulator. During the synthesis process, they map to look-up tables and carry chains performing the addition, as well as a set of flip-flops to store the results. The accumulators are built of $M \cdot L_s$ look-up tables, $M \cdot \lceil L_s/4 \rceil$ 4-bit carry chains and $M \cdot L_s$ flip-flops. On the targeted FPGA architecture, the structure is placed in $M \cdot \lceil L_s/4 \rceil$ logic slices, using all 4 look-up tables, the 4-bit carry chain and 4 of the 8 flip-flops available in each slice.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

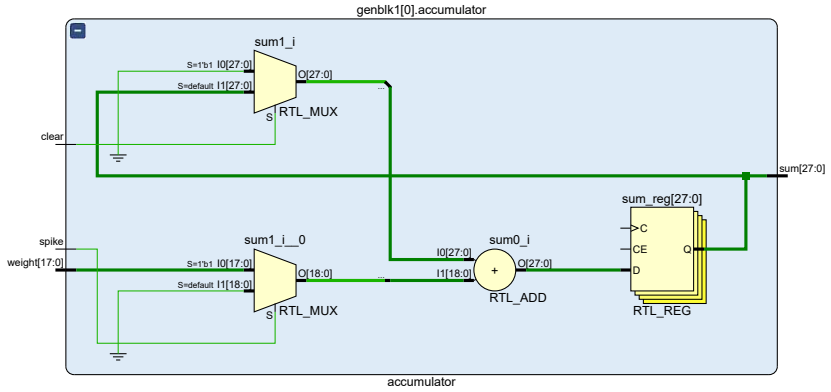

Figure 6.2: Schematic of an accumulator instance

Once the accumulators finish, the weighted sums are loaded into a shift register and serially presented at the module's output. Figure 6.3 shows a partial schematic of the shift register. The shift register stores M output values of length L_s and thus requires $M \cdot L_s$ flip-flops. The 4 look-up tables in a logic slice actually have 2 outputs each and can compute two individual functions as long as they share the same 5 input signals. Therefore, 2 two-input multiplexers with a shared select signal fit into a single look-up table. Fully populating the 4 look-up tables and 8 flip-flops in every slice, results in a total requirement of $\lceil M \cdot L_s / 8 \rceil$ slices.

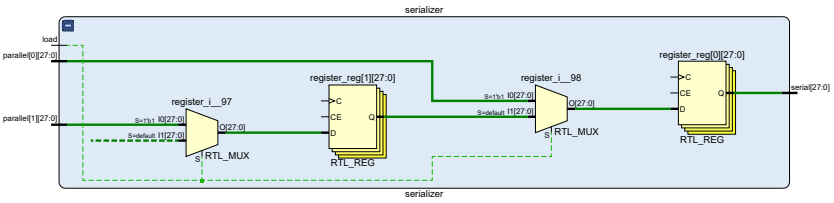

Figure 6.3: Partial schematic of the shift register

Additionally, there is a small amount of control logic handling resets and handshaking. Its size is mostly independent of the parameters, with the exception of two counters for tracking input and output. The module's resource utilization has a linear dependence on the width of the adjacent neuron layers. Thus, having a reasonable large synapses, we can ignore the constant overhead of the control logic, as well as the logarithmic overhead of its counters.

The optimizations applied throughout the synthesis and implementation processes are more relevant, as they depend on the settings and the given timing constraints, inferring additional logic. Oftentimes signals with high fan-out are replicated multiple times and placed close to where they are needed, effectively improving routability. Unfortunately, the exact amount of additional resources is hard to predict.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

In our example the synapses, connecting the input layer to the hidden layer ($N_1 = 784$, $M_1 = 100$, $L_s = 28$), are expected to utilize 4,200 look-up tables and 5,600 flip-flops occupying at least 1,050 slices. In comparison, table 6.4 lists the slice logic distribution from the utilization report. The predicted number of look-up tables (LUT as Logic) and flip-flops (Slice Registers) are within a safety margin of 5%. In contrast, the number of slices differs quite a lot. During placement the resources are assigned locations and spread across the FPGA. A lot of slices are only partially used, resulting in a higher reported number. The estimate is still reasonable, because, once approaching the limit of available resources, the tools will pack the design as dense as possible to still fit onto the FPGA.

Site Type	Used	Available	Util%
Slice	1332	15850	8.40
SLICEL	840		
SLICEM	492		
LUT as Logic	4361	63400	6.88
using O6 output only	1162		
using O5 and O6	3199		
Slice Registers	5726	126800	4.52
Register driven from within the Slice	5669		
Register driven from outside the Slice	57		
LUT in front of the register is unused	44		
LUT in front of the register is used	13		
Unique Control Sets	8	15850	0.05

Table 6.4: Reported slice logic distribution of synapses

Compared to other modules, the synapses are very resource intensive. Therefore, we synthesized and implemented many different synapses with varying parameters and timing constraints to verify our results and implement a proper model. All tests were performed, building the first synapses of our example application, assuming a fixed input width of $N = 784$. First, we stepwise increased the output width M until the FPGA runs out of memory for storing the weights. Then, we decreased the bit-length L_w of the weights from 18 to 9 bits and finally down to 4 bits. All designs have been compiled with the default strategy constrained by a 100 MHz clock, as well as with a strategy for optimal performance and a clock frequency of 200 MHz.

Table 6.5 and the diagrams in figure 6.4, 6.5 and 6.6 compare the estimated look-up tables utilization of different configurations to the values actually reported by the synthesis tool. Similarly, Table 6.6 and figure 6.7, 6.8 and 6.9 show a comparison for the flip-flop utilization. We can clearly see the overhead resulting from the optimization strategies applied by the synthesis and implementation tools.

The results show that utilization increases when optimizing for performance rather than area. Occasionally, when at a very low overall utilization, the standard strategy gave a lower priority to area and used up more resources than the performance optimized version. This happened with look-up tables at ($L_w = 18, M = 50$) in Table 6.5 and is also visible at the first data point of Figure 6.4. The flip-flops in Table 6.6 at ($L_w = 18, M = 50$), ($L_w = 9, M = 100$), ($L_w = 4, M = 200$) show the same behavior. This is also visualized

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

L_w	M	# look-up tables		
		estimated	reported @ 100 MHz	reported @ 200 MHz
18	50	2,100	2,888 (+37.5 %)	2,836 (+35.0 %)
	100	4,200	4,327 (+ 3.0 %)	5,668 (+35.0 %)
	150	6,300	6,456 (+ 2.5 %)	8,483 (+34.7 %)
	200	8,400	8,575 (+ 2.1 %)	11,316 (+34.7 %)
	250	10,500	10,720 (+ 2.1 %)	14,153 (+34.8 %)
9	100	2,850	2,929 (+ 2.8 %)	3,848 (+35.0 %)
	200	5,700	5,845 (+ 2.5 %)	7,667 (+34.5 %)
	300	8,550	8,695 (+ 1.7 %)	11,494 (+34.4 %)
	400	11,400	11,593 (+ 1.7 %)	15,329 (+34.5 %)
	500	14,250	14,475 (+ 1.6 %)	19,155 (+34.4 %)
4	200	4,200	4,283 (+ 2.0 %)	5,657 (+34.7 %)
	400	8,400	8,525 (+ 1.5 %)	11,264 (+34.1 %)
	600	12,600	12,780 (+ 1.4 %)	16,901 (+34.1 %)
	800	16,800	16,977 (+ 1.1 %)	22,535 (+34.1 %)
	1,000	21,000	21,224 (+ 1.1 %)	28,153 (+34.1 %)

Table 6.5: Look-up table utilization of synapses ($N = 784$)

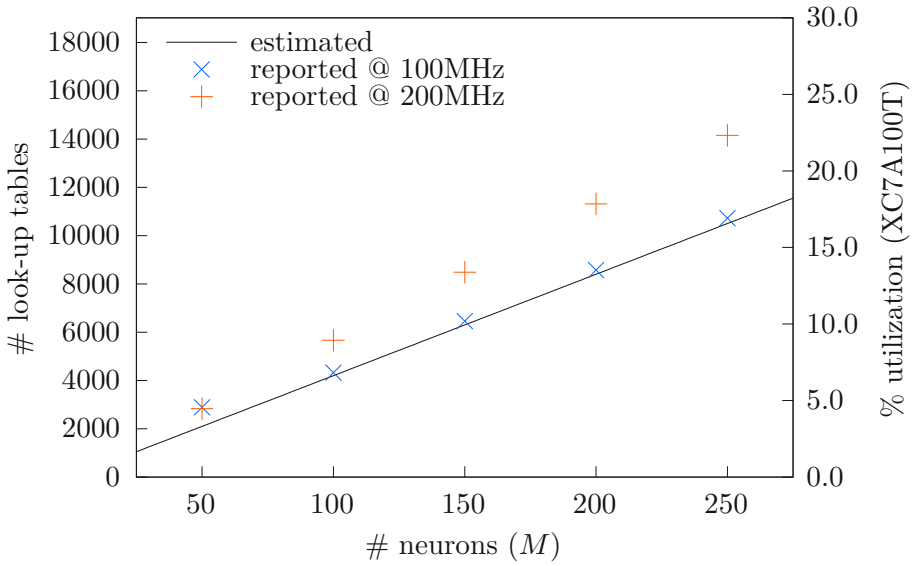

Figure 6.4: Look-up table utilization of synapses ($N = 784$) with 18-bit weights

in the first data point of Figure 6.7, Figure 6.8 and Figure 6.9. Although the results were unexpected, it only ever happened when actual utilization would not affect the successful completion of the synthesis/implementation run. Once the tools were under pressure, the behavior was as expected.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

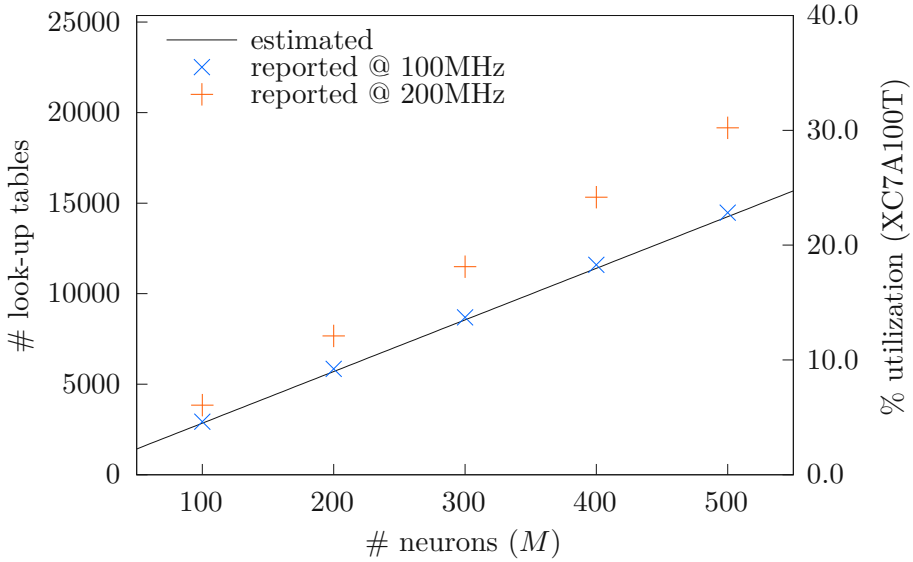

Figure 6.5: Look-up table utilization of synapses ($N = 784$) with 9-bit weights

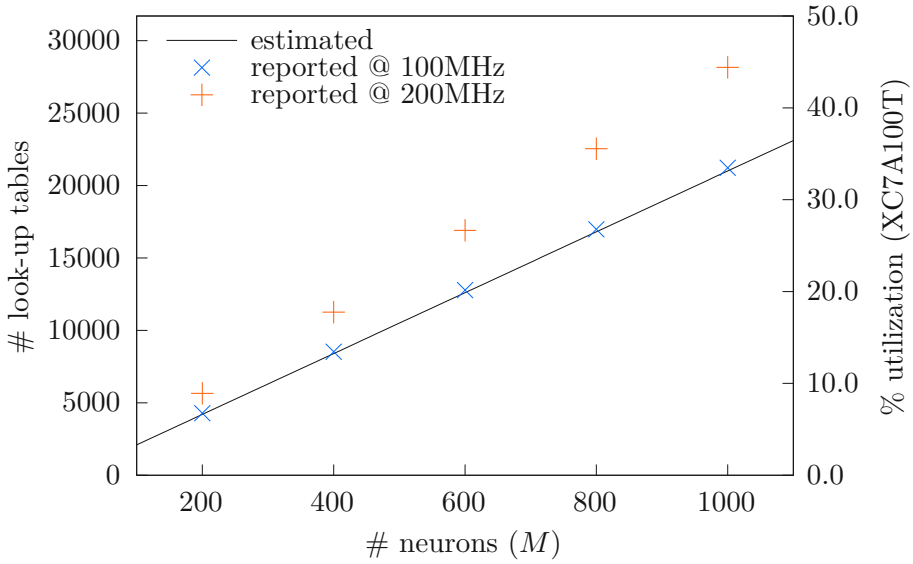

Figure 6.6: Look-up table utilization of synapses ($N = 784$) with 4-bit weights

Another issue arose with the huge discrepancy of look-up tables between the standard and performance optimizing strategy. While flip-flops only increase at a reasonable maximum of about 10 %, look-up tables reached a surplus of more than 33 %. An investigation revealed that this phenomenon results from the synthesis tool not combining look-up tables in order to meet the timing constraints. As discussed earlier, a look-up table can implement 2 two-input multiplexers as long as they share the same select

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

L_w	M	# flip-flops		
		estimated	reported @ 100 MHz	reported @ 200 MHz
18	50	2,800	2,911 (+ 4.0 %)	2,862 (+ 2.2 %)
	100	5,600	5,726 (+ 2.3 %)	6,176 (+10.3 %)
	150	8,400	8,538 (+ 1.6 %)	9,248 (+10.1 %)
	200	11,200	11,334 (+ 1.2 %)	12,312 (+ 9.9 %)
	250	14,000	14,159 (+ 1.2 %)	15,395 (+10.0 %)
9	100	3,800	3,884 (+ 2.2 %)	3,866 (+ 1.7 %)
	200	7,600	7,736 (+ 1.8 %)	8,177 (+ 7.6 %)
	300	11,400	11,510 (+ 1.0 %)	12,252 (+ 7.5 %)
	400	15,200	15,335 (+ 0.9 %)	16,321 (+ 7.4 %)
	500	19,000	19,160 (+ 0.8 %)	20,387 (+ 7.3 %)
4	200	5,600	5,688 (+ 1.6 %)	5,672 (+ 1.3 %)
	400	11,200	11,304 (+ 0.9 %)	11,738 (+ 4.8 %)
	600	16,800	16,940 (+ 0.8 %)	17,575 (+ 4.6 %)
	800	22,400	22,525 (+ 0.6 %)	23,428 (+ 4.6 %)
	1,000	28,000	28,148 (+ 0.5 %)	29,246 (+ 4.5 %)

Table 6.6: Flip-flop utilization of synapses ($N = 784$)

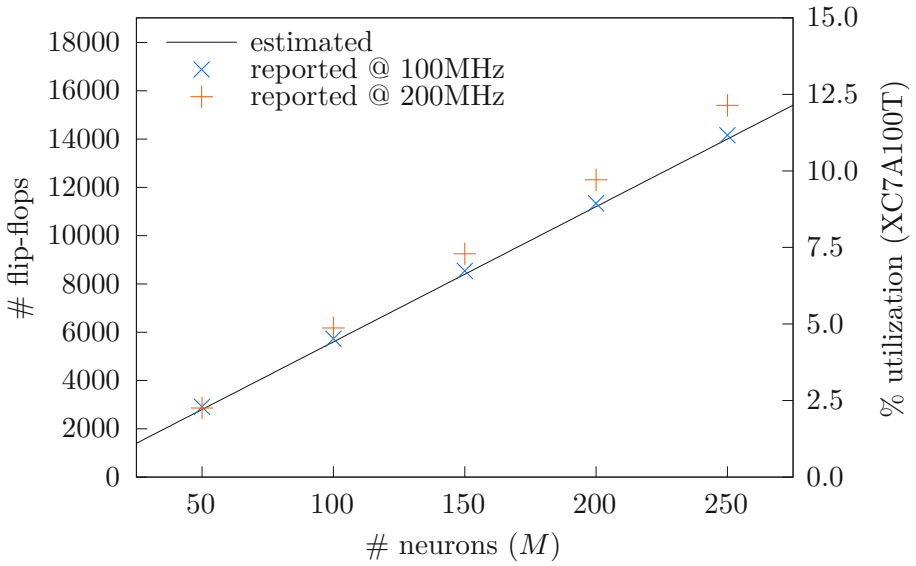

Figure 6.7: Flip-flop utilization of synapses ($N = 784$) with 18-bit weights

signal. Unfortunately, select is a combination of multiple signals that cannot be buffered. Preprocessing the select signal in a separate look-up table adds additional propagation and routing delay that reduces the maximum clock rate. When optimizing for performance, the select signal is oftentimes evaluated in the same look-up table as the multiplexer,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

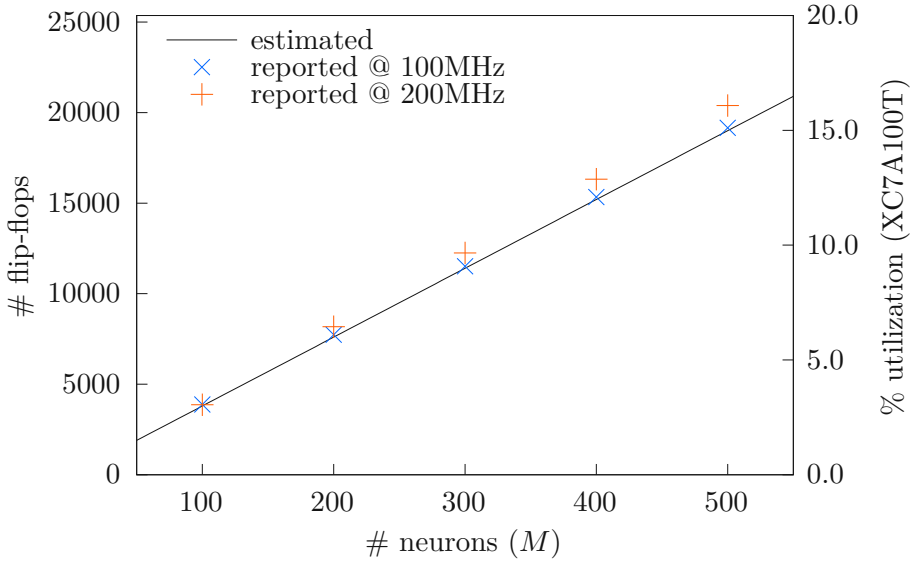

Figure 6.8: Flip-flop utilization of synapses ($N = 784$) with 9-bit weights

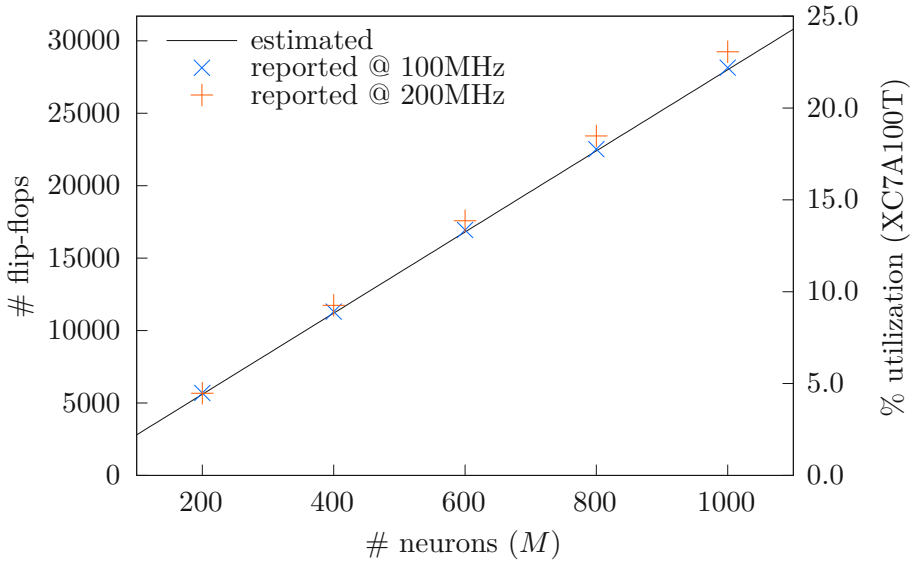

Figure 6.9: Flip-flop utilization of synapses ($N = 784$) with 4-bit weights

which effectively doubles the utilization of the shift register.

One more realization was that the number of occupied flip-flops is actually higher than reported. When looking at the slices of the accumulators, only 4 of 8 available flip-flops are used. Since the remaining 4 flip-flops share the same control set (clock, clock enable and set/reset signals) that is unique to the accumulators and their individual handshaking

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

signals, they are practically unusable anywhere else. When considering this fact, the actual number of occupied flip-flops is twice as high as previously estimated. The same applies to the shift register, when the look-up tables are not combined.

The experiments show that the utilization is most accurately estimated when considering the slices instead of look-up tables or flip-flops. The overhead of control logic and optimization adds up to 10 %. Furthermore, when utilization is an issue, it is necessary to force the synthesis tool to combine the look-up tables, which comes at the cost of a potentially lower clock frequency and needs careful consideration, when planning the network layout.

Memory Resources

A synapse module connecting N neuron outputs to M neuron inputs stores a total of $N{\cdot}M$ weights. The bit-width L_w, as well as the number format, i.e., signed fixed-point, depends on the chosen neuron model. In reasonable large networks the total memory requirements of $N{\cdot}M{\cdot}L_w$ bits warrant the use of dedicated RAM blocks. The module addresses N sets of M weights, perfectly mapping to an N deep memory and $M{\cdot}L_w$ wide.

The required memory can be mapped to the available blocks. First select the widest configuration that fits the required depth and then take as many blocks as necessary to reach the required width. The number B of block RAM resources can be calculated as follows:

$$B = \begin{cases} \lceil ML_w/36 \rceil/2, & \text{if } N \leq 512 \\ \lceil ML_w/18 \rceil/2, & \text{if } N \leq 1024 \\ \lceil ML_w/9 \rceil/2, & \text{if } N \leq 2048 \\ \lceil ML_w/4 \rceil/2, & \text{if } N \leq 4096 \\ \lceil ML_w/2 \rceil/2, & \text{if } N \leq 8192 \\ ML_w/2, & \text{if } N > 8192 \text{ and } N \leq 65536 \end{cases}$$

In our example, the synapses connecting the input layer to the hidden layer ($N_1 = 784$, $M_1 = 100$, $L_w = 18$) require 50 RAMB36E1, whereas the synapses connecting the hidden layer to the output layer ($N_2 = 100$, $M_2 = 10$, $L_w = 18$) require 2.5 RAMB36E1 or rather 2 RAMB36E1 and 1 RAMB18E1. These results are identical to the numbers in the memory utilization report in table 6.7 and table 6.8. Together, the synapses utilize 52.5 out of 135 available block RAMs or approximately 39% of the dedicated memory resources.

The amount of available memory turned out to be the limiting factor, when it comes to implementing bigger networks. With the synapses of our example application already taking up 40 % of the available memory, we are limited to a maximum of about 250 neurons in our hidden layer. Lowering the bit-width of the weights increases the possible

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Site Type	Used	Available	Util%
Block RAM Tile	50	135	37.04
RAMB36	50	135	37.04
RAMB18	0	270	0.00

Table 6.7: Memory utilization report of synapses ($N_1 = 784$, $M_1 = 100$, $L_w = 18$)

Site Type	Used	Available	Util%
Block RAM Tile	2.5	135	1.85
RAMB36	2	135	1.48
RAMB18	1	270	0.37

Table 6.8: Memory utilization report of synapses ($N_2 = 100$, $M_2 = 10$, $L_w = 18$)

number of neurons. Also spreading them over multiple hidden layers decreases the number of connections, effectively reducing the required memory. When looking for a network to solve a specific problem, finding a well working model and structure that is light on the resources is a major challenge.

6.3.3 Poisson Encoder

Given a set of intensity values, the *Poisson encoder* generates randomly distributed spike trains for the network's input neurons. In our example application, each neuron is associated with a pixel of the input image. The bright pixels, making up the background, rarely cause spikes, whereas dark pixels, considered part of the handwritten digit, lead to highly active neurons.

The module is built around a pseudorandom number generator (PSNR). During a simulation, each of the cached input values is compared to a freshly generated random number. The Boolean results, stating whether the associated neurons emitted a spike, are forwarded to the output. Repeating the process for every simulation tick results in Poisson distributed spike trains, whose average rates correspond with the input values.

Logic Resources

The *Poisson encoder* is controlled by a finite state machine that manages the flow of data, addresses memory and triggers the generation of random numbers. It handles resets and handshaking at the interfaces, always keeping track of the input, current simulation tick and output. Table 6.9 list the reported logic utilization of the module in our example application. The amount of resources has a logarithmic dependence on the number of input neurons and the amount of simulation ticks that is usually significantly smaller than the constant part.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Site Type	Used	Available	Util%
Slice	279	15850	1.76
SLICEL	174		
SLICEM	105		
LUT as Logic	242	63400	0.38
using O6 output only	210		
using O5 and O6	32		
LUT as Memory	90	19000	0.47
LUT as Shift Register	90		
using O5 output only	88		
using O6 output only	2		
Slice Registers	1578	126800	1.24
Register driven from within the Slice	331		
Register driven from outside the Slice	1247		
LUT in front of the register is unused	1204		
LUT in front of the register is used	43		
Unique Control Sets	11	15850	0.07

Table 6.9: Logic utilization report of the Poisson encoder

Memory Resources

The *Poisson encoder* operates on a set of intensity values - one for each input neuron. Since they are only provided at the start of a simulation, the values need to be cached throughout the run. The initial write operation, as well as the repeated read operations in each tick of the simulation, happen sequentially and in the same order. This particular usage pattern, is ideally suited for the use of dedicated block RAM, whose addressing can be provided by a pair of cyclic counters.

In our example application, input consists of a 28 by 28 grid of 8-bit values representing a grayscale image of a handwritten digit. The data (784 x 8 bits) conveniently fits into a single RAMB18 primitive, configured to either 1024 x 18 bits or 2048 x 9 bits. Table 6.10 lists the reported memory utilization, showing the expected result.

Site Type	Used	Available	Util%
Block RAM Tile	0.5	135	0.37
RAMB18	1	270	0.37
RAMB18E1 only	1		

Table 6.10: Memory utilization report of the Poisson encoder

Another memory requirement stems from the incorporated PRNG. Its cyclic buffer is constantly updated, based on previously generated values. Since this process requires access to multiple storage locations at once, it can't easily be implemented using block RAM. However, there is still a more efficient method than simply chaining flip-flops.

The PRNG uses distributed memory, configuring look-up tables to act as shift registers. Its memory utilization is reported as "LUT as Shift Register" and table 6.9 list 90 instances or 0.47 % of the total available. The PRNG is currently limited to 31 random bits per clock cycle and its implementation uses a constant amount of resources, independent of the module's parameterization.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Signal Processing Resources

The generation of random numbers within the PRNG, as well as the comparison to the input values require add/subtract operations. The module utilized general logic slices including their low-latency carry chains, rather than DSP slices. The 32-bit addition of the PRNG has been split into multiple pipeline stages not to bottleneck the design at higher clock frequencies. Overall, the signal processing is responsible for a significant portion of the module's logic resources reported in table 6.9. Since the PRNG is not affected by any of the module's parameters, its resource utilization is constant.

6.3.4 Rate Decoder

The *rate decoder* manages a set of counters, keeping track of the number of spikes neurons generate during a simulation run. The spike counts are proportional to the rates, at which the output neurons get activated. The values are provided as overall output of the neural network and for the application to interpret. In our example, each of the ten output neurons represents a digit and the application considers the digit with the highest activity as the most likely one depicted on the handwritten input.

The module receives a sequence of Boolean values from the output layer. Each value corresponds to a specific neuron, stating whether it generated a spike. For each input, the module loads the value of the associated counter, increments it if it signals a spike, and stores it back to memory. The process is repeated for every time tick until the simulation ends. Finally, the counters are presented at the module's output and subsequently reset in preparation for another run.

Logic Resources

The *rate decoder* is based on a finite state machine that keeps track of the input, its associated neuron and counter, as well as the current simulation tick. It also handles resets and handshaking at the input and output interfaces. Table 6.11 lists the reported logic utilization of the module in our example application. The amount of resources has a logarithmic dependence on the number of neurons in the output layer and the amount of simulation ticks. In comparison to other modules, it uses an insignificant amount of logic resources.

Memory Resource

Each counter stores the amount of spikes occurred for each output neuron. Assuming a spike could occur in every tick, a simulation lasting T ticks requires counters of length $\lceil \log_2(T+1) \rceil$ to reflect the full values range $[0, T]$. This number can be reduced by shortening the simulation time or choosing a neuron model that guarantees spike generation at a lower maximum rate than one per tick.

The design supports any number of counters, whose length can be either specified manually or defaults to the worst-case amount of bits required for the given number of simulation

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Site Type	Used	Available	Util%
Slice	24	15850	0.15
SLICEL	19		
SLICEM	5		
LUT as Logic	25	63400	0.04
using O6 output only	20		
using O5 and O6	5		
LUT as Memory	10	19000	0.05
LUT as Distributed RAM	10		
using O6 output only	2		
using O5 and O6	8		
Slice Registers	82	126800	0.06
Register driven from within the Slice	53		
Register driven from outside the Slice	29		
LUT in front of the register is unused	25		
LUT in front of the register is used	4		
Unique Control Sets	5	15850	0.03

Table 6.11: Logic utilization report of the rate decoder

ticks. In our example application, having 10 output neurons and a simulation time of 4096 ticks, the module instantiated 10 counters with a default length of $\lceil \log_2(4096+1) \rceil = 13$ bits.

The module is designed to use block RAM, but at very little capacities, hence the synthesis tool prefers to utilize distributed memory. Instead of wasting 99.3 % of a RAM block, it instantiates 10 look-up tables across 3 different slices. The relevant resources "LUT as memory" are already included in table 6.11. Theoretically, 7 look-up tables yielding 32 x 14 bits in 2 slices should suffice, but, for reasons of optimization, the synthesis tool spreads out resources to connect together short wires and prevent congestion during routing.

Signal Processing Resources

All counters are processed sequentially, requiring only a single adder for conditional incrementing. With the availability of low-latency carry chains, the module prefers general logic resources over DSP slices. The resources utilized by the 13-bit adder of our example application are minuscule and already included in table 6.11.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 7

Conclusion

In this work, we discussed the possible advantages of using FPGAs to simulate spiking neural networks. We presented a modular implementation, providing the network's basic building blocks, and also a hardware and software framework for embedding spiking neural networks into real-world applications. Finally, we provided testing results, in regards to performance, power consumption and resource utilization, based on an example setup, built to classify handwritten digits of the MNIST database.

The presented results elaborate on how to calculate the performance of any given network structure, based on the measurements of our example application. Furthermore they provide means to estimate the required hardware resources and thus aid in the selection of an appropriately sized FPGA. Some of the main takeaways regarding performance and resource utilization of spiking neural networks on FPGAs are:

- The **performance** is limited by the amount of parallel structures that fit into the available hardware resources of an FPGA. Our implementation, running at about 250 MHz on the lowest speed grade XC7A100T, takes one clock cycle per neuron of the widest network layer to advance the simulation for one tick of time. This turned out to be a good balance between performance and resource utilization. A lower parallelization would degrade the performance to the level of microcontrollers, while higher parallelization severely limits the size of the neural networks that can be fit onto an FPGA.
- The **resource utilization** is the limiting factor when it comes to network size. The weights defining the neuron's connections can quickly consume the available memory. Similarly, the many parallel circuits, computing the weighted sums, take up a major portion of the logic resources. Whether a network fits into a specific FPGA, depends on the number of interconnections and the bit-width of their defining weights, rather than the amount of neurons. Using an XC7A100T, we

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

could fit a 200-neuron layer when using 18-bit weights and a 400-neuron layer with 9-bit weights, the limiting factor always being the interconnections to the 784-neuron input layer of our example application.

The results of this work may be extended by exploring alternative concepts. Besides the implementation of additional neuron models, we are especially interested in comparing the current time-driven approach to an event-based system. Furthermore an evaluation of learning algorithms and their eligibility for an on-chip implementations would be a major step towards autonomous, self-improving systems.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[1] L.F Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Research Bulletin*, 50(5):303–304, 1999.

[2] 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(5):699–716, 2014.

[3] Sander M. Bohte and Joost N. Kok. Applications of spiking neural networks. *Information Processing Letters*, 95(6):519–520, 2005. Applications of Spiking Neural Networks.

[4] Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. Spiker: an fpga-optimized hardware accelerator for spiking neural networks. In *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 14–19. IEEE, 2022.

[5] Li Deng and Dong Yu. Deep learning: Methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[6] Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, 9:99, 2015.

[7] Digilent. Nexys 4 reference manual. `https://digilent.com/reference/_media/reference/programmable-logic/nexys-4/nexys4_rm.pdf`. Accessed: 2024-01-01.

[8] Michael D Forrest. Can the thermodynamic hodgkin-huxley model of voltage-dependent conductance extrapolate for temperature? *Computation*, 2(2):47–60, 2014.

[9] Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

[10] Jacques Gautrais and Simon Thorpe. Rate coding versus temporal order coding: a theoretical approach. *Biosystems*, 48(1):57–65, 1998.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[11] David H. Goldberg and Andreas G. Andreou. Distortion of Neural Signals by Spike Coding. *Neural Computation*, 19(10):2797–2839, 10 2007.

[12] 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, 1952.

[13] Eugene M Izhikevich and Richard FitzHugh. Fitzhugh-nagumo model. *Scholarpedia*, 1(9):1349, 2006.

[14] Donald E Knuth. The art of computer programming. vol. 2: Seminumerical algorithms. *Reading*, 1981.

[15] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

[16] Anirban Nandi, Thomas Chartrand, Werner Van Geit, Anatoly Buchin, Zizhen Yao, Soo Yeun Lee, Yina Wei, Brian Kalmbach, Brian Lee, Ed Lein, et al. Single-neuron models linking electrophysiology, morphology, and transcriptomics across cortical cell types. *Cell reports*, 40(6), 2022.

[17] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

[18] Khashayar Pakdaman, Michele Thieullen, and Gilles Wainrib. Fluid limit theorems for stochastic hybrid systems with application to neuron models. *Advances in Applied Probability*, 42(3):761–794, 2010.

[19] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[20] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[21] Marcel Stimberg, Romain Brette, and Dan FM Goodman. Brian 2, an intuitive and efficient neural simulator. *elife*, 8:e47314, 2019.

[22] Yifan Sun, Nicolas Bohm Agostini, Shi Dong, and David R. Kaeli. Summarizing CPU and GPU design trends with product data. *CoRR*, abs/1911.11313, 2019.

[23] Xilinx. 7 series fpgas configurable logic block. `https://docs.xilinx.com/v/u/en-US/ug474_7Series_CLB`. Accessed: 2024-01-01.

[24] Xilinx. 7 series fpgas data sheet: Overview. `https://docs.xilinx.com/v/u/en-US/ds180_7Series_Overview`. Accessed: 2024-01-01.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[25] Xilinx. Artix-7 fpgas data sheet: Dc and ac switching characteristics. https://docs.xilinx.com/v/u/en-US/ds181_Artix_7_Data_Sheet. Accessed: 2024-01-01.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.