Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-2019-4159489

M.Sc. Thesis

A Highly Concurrent, Memory-Efficient AER Architecture for Neuro-Synaptic Spike Routing

J.P. Coenen B.Sc.

Abstract

One of the challenges of neuromorphic computing is efficiently routing spikes from neurons to their connected synapses. The aim of this thesis is to design a spike-routing architecture for flexible connections on single-chip neuromorphic systems. A model for estimating area, power consumption, memory, spike latency and link utilisation for neuromorphic spike-routing architecture is described. This model leads to the proposal for a new spike-routing architecture with a hybrid addressing scheme and a novel synaptic encoding scheme.

The proposed architecture is implemented in a SystemC simulation tool with a supporting tool for encoding arbitrary SNN topologies for the synapse encoding scheme. Running the simulations with synthetic benchmarks and a handwriting recognition SNN shows that the proposed architecture is memory-efficient and provides low latency spike-routing with high synaptic activation concurrency.

A Highly Concurrent, Memory-Efficient AER Architecture for Neuro-Synaptic Spike Routing

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Electrical Engineering

by

J.P. Coenen B.Sc.
born in Vught, The Netherlands

This work was performed in:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Delft University of Technology
Department of
Microelectronics & Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"A Highly Concurrent, Memory-Efficient AER Architecture for Neuro-Synaptic Spike Routing"** by **J.P. Coenen B.Sc.** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 17 April 2019

Advisor:

dr.ir. T.G.R.M. van Leuken

Committee Members:

dr.ir. Z. Al-Ars

dr.ir. S.S. Kumar

dr. A. Zjajo

Abstract

One of the challenges of neuromorphic computing is efficiently routing spikes from neurons to their connected synapses. The aim of this thesis is to design a spike-routing architecture for flexible connections on single-chip neuromorphic systems. A model for estimating area, power consumption, memory, spike latency and link utilisation for neuromorphic spike-routing architecture is described. This model leads to the proposal for a new spike-routing architecture with a hybrid addressing scheme and a novel synaptic encoding scheme.

The proposed architecture is implemented in a SystemC simulation tool with a supporting tool for encoding arbitrary SNN topologies for the synapse encoding scheme. Running the simulations with synthetic benchmarks and a handwriting recognition SNN shows that the proposed architecture is memory-efficient and provides low latency spike-routing with high synaptic activation concurrency.

Acknowledgments

First of all, I would like to thank Rene and Amir for assisting me throughout the overly long process of writing this thesis. I'd also like to thank Johan for the company at HB17.090 and helping me out with some real-world spike data. Sumeet deserves a special mention for not only assisting but always being able to get me motivated again.

Furthermore I'd like to thank all my colleagues at SecretHub for being patient with me during the whole process and also for providing a nice working environment and the occasional distraction.

Last but not least, thanks to my family and friends for their support and patience.

J.P. Coenen B.Sc.
Delft, The Netherlands
17 April 2019

Contents

Abstract **v**

Acknowledgments **vii**

1 Introduction **1**
1.1 Problem Statement . . . 1
1.2 Goals . . . 3
1.3 Contributions . . . 3
1.4 Thesis Outline . . . 3

2 Background **5**
2.1 Brain Inspired Computing . . . 5
2.1.1 Working of the Brain . . . 5
2.1.2 Difference of the brain and traditional pc . . . 6
2.1.3 Neural Networks . . . 7
2.2 Neural Net Topologies . . . 9
2.2.1 Feed-Forward Network . . . 9
2.2.2 Self-Organizing Map . . . 9
2.2.3 Liquid State Machine . . . 10
2.3 Neuromorphic Hardware . . . 11
2.3.1 AER . . . 11
2.4 State of the Art . . . 12
2.4.1 Implementation options . . . 12
2.4.2 State of the Art . . . 13
2.4.3 Research Gap . . . 16

3 Design Space Exploration **17**
3.1 Architecture Overview . . . 17
3.1.1 Neuro-synaptic array . . . 17
3.1.2 Clusters . . . 17
3.2 Design Space . . . 18
3.2.1 Input parameters . . . 18
3.2.2 Model outputs . . . 19
3.2.3 Model Components . . . 20
3.3 Connectivity . . . 20
3.3.1 Connection distribution . . . 20
3.3.2 Notation . . . 21
3.4 Addressing . . . 21
3.4.1 Addressing Schemes . . . 22
3.4.2 Traffic . . . 23
3.4.3 Address Sizes . . . 25
3.4.4 Lookup Tables . . . 25

3.4.5 Latency . . . 27
3.5 Synapse Encoding . . . 29
3.5.1 Banking . . . 29
3.5.2 Column Reuse . . . 31
3.5.3 Row Grouping . . . 31
3.5.4 Column Address Offsetting . . . 32
3.5.5 Combining . . . 33
3.6 NoC . . . 33
3.6.1 Topology . . . 33
3.6.2 Link load . . . 33
3.6.3 Latency . . . 34
3.7 Power and Area estimation . . . 35
3.7.1 Neuro-synaptic array . . . 35
3.7.2 Lookup tables . . . 35
3.7.3 Local interconnect . . . 36
3.7.4 NoC . . . 36
3.8 Evaluation . . . 36
3.8.1 Clustering . . . 37
3.8.2 Addressing . . . 38
3.8.3 Synapse Encoding . . . 39
3.9 Conclusion . . . 39

4 Simulation **41**
4.1 System Overview . . . 41
4.2 Neuro-synaptic array . . . 42
4.3 Simulation Flow . . . 42
4.3.1 Topology Generation . . . 43
4.3.2 Mapping . . . 44
4.3.3 LUT Building . . . 46
4.3.4 Spike Generation . . . 47
4.3.5 Simulator . . . 48
4.3.6 Analysis . . . 49
4.4 Summary . . . 49

5 Performance Evaluation **51**
5.1 Mapping . . . 51
5.1.1 Clustering method . . . 52
5.1.2 Mapping optimisation . . . 52
5.1.3 Banking and Grouping . . . 53
5.1.4 Column Offset . . . 54
5.1.5 Summary . . . 55
5.2 Simulation . . . 55
5.2.1 Saturation point . . . 55
5.2.2 Latency . . . 56
5.2.3 Spike rate . . . 59

5.2.4 Spike bursts 59
5.3 Application 60
5.3.1 Setup 61
5.3.2 Results 61
5.3.3 Power and Area 63

6 Conclusion **65**
6.1 Future work 65

A LUT content configuration file **71**

List of Figures

1.1 Simplified overview of neuro-synaptic array. 1
1.2 Trade-off for storing connections in neuromorphic systems. 2

2.1 Overview of two connected neurons. Adapted from [1] 5
2.2 Leaky integrate and fire model. 6
2.3 Schematic overview of Von Neumann architecture. Adapted from[2] . . 7
2.4 Overview of a 5-layer neural network[3]. 8
2.5 Schematic overview of a Liquid State Machine. Dots represent neurons and the arrows represent synaptic connections. The reservoir neurons form the liquid state. [4] 10
2.6 Conceptual diagram of AER working as a bus for spiking events.[5] . . 11
2.7 Overview of the Neurogrid architecture. Adapted from [6]. 13
2.8 Overview of a SpiNNaker node[7]. 14
2.9 Overview of the TrueNorth core 15
2.10 Overview of the BrainScaleS architecture[8]. 16

3.1 Schematic overview of a modelled cluster. 18
3.2 Schematic overview of Source Addressing 22
3.3 Schematic overview of Destination Addressing 23
3.4 Schematic overview of Hybrid Addressing 24
3.5 Traffic flow from neurons to synapses in a cluster. 24
3.6 Example of a two stage 1 to many lookup 26
3.7 Modelled LUT sizes for the different addressing schemes 27
3.8 Example of difference in latency for addressing schemes 28
3.9 Trend of the change in latency due to parallelisation over multiple clusters for the different addressing schemes 29
3.10 Illustrations of different improvements for synapse encoding 30
3.11 Example of what happens when two synapses in different columns and row are activated simultaneously 31
3.12 Schematic representation of Row Grouping 32
3.13 Memory per synapse for increasing grouping efficiency 32
3.14 Overview of NoC topologies supported by the model. 33
3.15 Modelled area and power for varying cluster sizes. 38
3.16 Utilisation of local and NoC links for different addressing schemes and varying cluster sizes. 38
3.17 LUT size per synapse for the local and uniform connection distribution. 39

4.1 System overview for simulation 41
4.2 NoC router 42
4.3 Implementation of the neuro-synaptic array in the simulation. 43
4.4 Simulation flow 43
4.5 Example of a very simple SNN topology 44
4.6 Numbering of neurons and synapses in neuro-synaptic array 45

4.7 Spikes as generated for a single neuron for different spike generators . . 48

5.1 Mapping performance for varying row group size and banking factors. . 54
5.2 Effect of changing column offset. 54
5.3 Plot maximum synaptic activation rates for varying neuro-synaptic array sizes. 57
5.4 Latency and latency jitter for uniform connection distribution with a poisson spike input of 1 kHz. 57
5.5 Latency and latency jitter for local connection distribution with a poisson spike input of 1 kHz. 58
5.6 Latency and latency jitter for layered connection distribution with a poisson spike input of 1 kHz. 58
5.7 Plot of spike latency jitter for varying neuro-synaptic array configurations. 59
5.8 Plot of spike latency jitter for varying poisson distributed spike inputs. 60
5.9 SNN topology for handwritten digit recognition. 61
5.10 Latency and latency jitter for spikes in handwriting recognition SNN with the optimised configuration. 62
5.11 Latency and latency jitter for spikes in handwriting recognition SNN with the baseline configuration. 62

List of Tables

3.1 Size of the step 1 LUTs for different addressing schemes for cluster i. . 26
3.2 Size of the step 2 LUTs for different addressing schemes for cluster i. . 27
3.3 Properties of the different network topologies supported by the model. N is the number of nodes. 34
3.4 LUT power and area figures from CACTI6.5 for a 45 nm process. . . . 35
3.5 Power and area figures from ORION. 37
3.6 Baseline evaluation parameters. 37

5.1 Effect of clustering on FoM for generated or randomized neuron 52
5.2 Average number of clusters a neuron is connected to. 52
5.3 Effect of mapping optimisation methods on FoM. 53
5.4 Mapping performance for the chosen parameters of the architecture. . . 55
5.5 Default simulation setup 56
5.6 Saturation test setup. 56
5.7 Maximum spike input and output. 56
5.8 Latency test setup 57
5.9 Spike rate test setup 59
5.10 Spike bursts test setup 60
5.11 Latency jitter for varying burstiness of spike input with an average rate of 1 kHz. 60
5.12 Handwriting SNN mapping results 61
5.13 Parameters for power and area estimation. 63
5.14 Area and power estimation for implementation in 45 nm. 64

1 Introduction

One of the currently most visible changes in technology is the increased usage of artificial intelligence. Computers used to be only applicable for algorithmic tasks like calculations and generating 3D images. But computers are more and more used for tasks humans are good at speech recognition, facial recognition and detecting fraud. A driving factor in this development is the use of neural networks. These brain-inspired computation models provide a way to do these tasks by first learning from a set of known training data.

An interesting development in the field of hardware are *neuromorphic systems*. These are hardware implementations of Spiking Neural Networks (SNN) and consist of a large set of hardware neurons that replicate the behaviour of neurons found in brains. Data is encoded as spikes that travel between neurons. By creating many connections between these neurons, a neural network is built. Every connection is called a synapse and has a weight that represents the connection strength.

A neuron can have hundreds to thousands of incoming connections. In this thesis, it is assumed that every connection has its own hardware implementation of a synapse. A logical arrangement for this is the neuro-synaptic array from Figure 1.1.

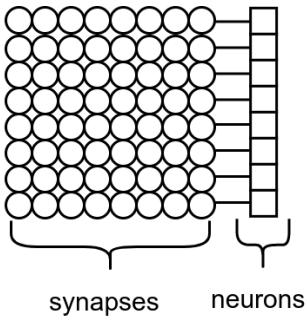


Figure 1.1: Simplified overview of neuro-synaptic array.

When a neuron produces a spike, this spike should be transported to the correct synapses of the neurons it is connected to. Creating direct wire connections between neurons and synapses is often infeasible because of the high number of connections. Therefore an often used approach is using Address Event Representation: encoding spikes as addresses and transferring them on a bus.

1.1 Problem Statement

When designing an AER-based neuro-synaptic spike routing architecture, there are several metrics that are important:

Synapse utilisation: what fraction of the hardware synapses are in use when a neural network is mapped to the array. In most cases, synapses are a significant fraction of the total chip area. Having unused synapses is, therefore, a waste of area.

Connection flexibility: to what degree can a neuro-synaptic array be used for different neural network topologies. More flexibility means the neuromorphic system can be used for a wider variety of applications and the topology can be tweaked after production.

Memory size: how much memory is needed to store the connection configuration. Size of memory can have a significant impact on the total area and power consumption of a neuromorphic system.

Throughput: how many neural spikes can be processed in a time-window. More throughput allows for bigger neural networks or higher spiking rates.

Latency: the time for a neural spike to lead to a synapse activation. Keeping this small and as constant as possible is important because information is encoded in the timing of spikes.

Designing a solution that has the best properties for all these points is impossible. There is always a trade-off to be made. Especially between the first three (Figure 1.2).

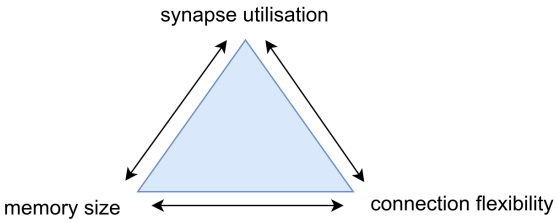


Figure 1.2: Trade-off for storing connections in neuromorphic systems.

For example, by connecting neurons to synapses with wiring, no memory is needed and all hardware synapses can be used, but the neural network topology is fixed at chip design time. Another extreme is having a lookup table with every neuron to synapse connection stored in it. This requires a lot of memory but provides maximum flexibility in connectivity. Finally, a way to optimise for flexibility and memory size is by creating a hardware synapse for every combination of two neurons. Connections are enabled by setting the synaptic weight to a non-zero value. This requires no extra memory but may result in low synapse utilisation.

Latency and throughput are often negatively correlated for neuromorphic systems because of the high fan-out from a single neural spike to many synaptic activations. Therefore the latency is influenced by the rate at which synaptic activations can be processed. If this rate is low, spikes will have to wait for each other to be processed, increasing latency, which is undesirable because it affects the information encoded in the spike timing. By increasing the number of synaptic activations that can be processed concurrently, spike latency can be kept in bounds.

In this thesis, we will research the effect of different design parameters on the aforementioned metrics and use that to design a spike-routing architecture with high connection flexibility while optimising for memory size and synaptic event concurrency.

1.2 Goals

- Identify relevant design parameters for creating a spike routing architectures for neuromorphic systems.
- Analyse and model the effect of design parameters on memory size, area, power, throughput and latency.
- Design a flexible architecture for spike routing in neuromorphic systems optimised for memory size and concurrency.

1.3 Contributions

- Created a model for estimating area, power and throughput for different neuromorphic spike-routing architectures.
- Designed a memory-efficient neuromorphic interconnect architecture and implemented a simulation model in SystemC.
- Designed and implemented software to map neurons and synapses to a grid under the constraints of the designed architecture.

1.4 Thesis Outline

In Chapter 3 a design space exploration of AER-based neuro-synaptic spike routing is made. This leads to a model that can be used to quickly estimate the effects of various design parameters. Next, in Chapter 4 a SystemC simulation of a spike-routing architecture based on the findings in the previous chapter is implemented. Also, various tools assisting in the simulation flow are introduced. In Chapter 5 the simulation toolbox is used to evaluate the performance of the proposed architecture. Finally, Chapter 6 draws conclusions and provides suggestions for future work.

Background 2

2.1 Brain Inspired Computing

In 1990, Mead first proposed the idea of neuromorphic computing[9]. This is the idea of computing inspired by the working of brains. He described the differences between traditional digital computing and the way brains perform computations. Mead identified that brains may be slower at certain types of computations, like arithmetic, but they are fundamentally more energy efficient.

In this section, we will first take a look at how brains function and then take a look at how that is applied for neuromorphic systems.

2.1.1 Working of the Brain

Brains consist of a network of neurons. For human brains, this can be up to 95 billion neurons [10]. A neurons consists of multiple parts, as depicted in Figure 2.1. The cell body of the neuron is called to soma. Connected to the soma are the dendrites, which acts as inputs for the neuron. Also connected to the soma is the axon, with on its end multiple axon terminals. The axon can be many times longer than the size of the soma. Each of these axon terminals is connected to the dendrite of another neuron, thereby creating a network of neurons. Two neurons are not physically connected. Between them is a gap, called the synapse. Any single neuron can be connected to up to 10 thousand other neurons.

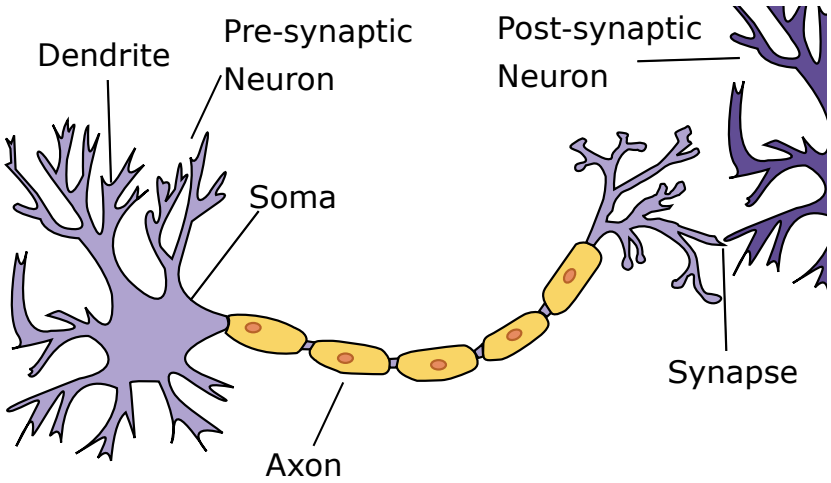


Figure 2.1: Overview of two connected neurons. Adapted from [1]

Communication between neurons is the essence that enables them to process information. This happens by the means of electrical spikes. We will consider the Leaky

Integrate-and-Fire model (Figure 2.2). This is a simplified model of the actual workings, but it allows for a conceptual understanding of the workings of the network of neurons. In this model, every spike at a neuron's dendrite generates a current, dependent on the characteristics of the synapse This current changes the internal potential of the neuron according to the following equation:

$$C\frac{dV(t)}{dt} + \frac{V(t)}{R} = I(t) \tag{2.1}$$

. Where C represents an internal capacitance and R a leakage resistance. $I(t)$ In other words: a positive incoming current increases the potential and a negative current decreases it. The leakage lets the potential gradually drift towards 0. If the internal potential surpasses the threshold voltage, the neuron generates an output spike and the internal potential is reset to 0 and it enters a refractory period. During this period, the neuron ignores any incoming spikes and therefore will not generate any output spikes.

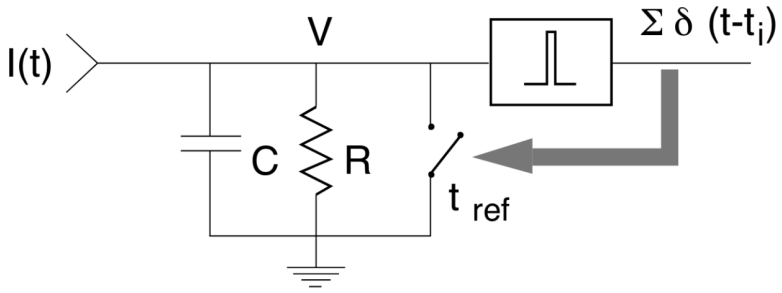


Figure 2.2: Leaky integrate and fire model.

Because of the leakage of the neuron's potential, the timing of the arrival of different input spike becomes important. If multiple spikes arrive in a short period of time, the neuron might generate an output spike. But if the arrival of these spikes is spread of a longer period, the internal potential might have leaked too much it to surpass the threshold.

As stated before, the size and the duration of the current that is generated when a spike traverses a synapse is dependent on the synaptic strength. This strength comes from the biological characteristics of the synapse.

2.1.2 Difference of the brain and traditional pc

When Mead proposed neuromorphic computing, he identified fundamental differences between biological and digital computing which can be found in the difference between analog and digital computation. By being analog in nature, a synapse can have a more complex input to output relation than a digital gate that represents a boolean function. This leads to more power efficient devices because no energy is wasted on rounding outputs to binary values[9].

A second difference that has become more and more prevalent as time progressed, is the absence of a memory bottleneck in biological computations[11]. Because of the division between the computing unit and the memory in the Von Neumann architecture

for general purpose computing, there is an inherent bottleneck on the communication bus between the two. This has driven the need for many layers in memory caches for the computing unit to run at maximum performance. Consequently increasing power consumption and memory footprint. For a brain, on the other hand, data is stored in their synaptic connections and their strength. This means that the memory and computation are intertwined and there is no such thing as a memory bottleneck.

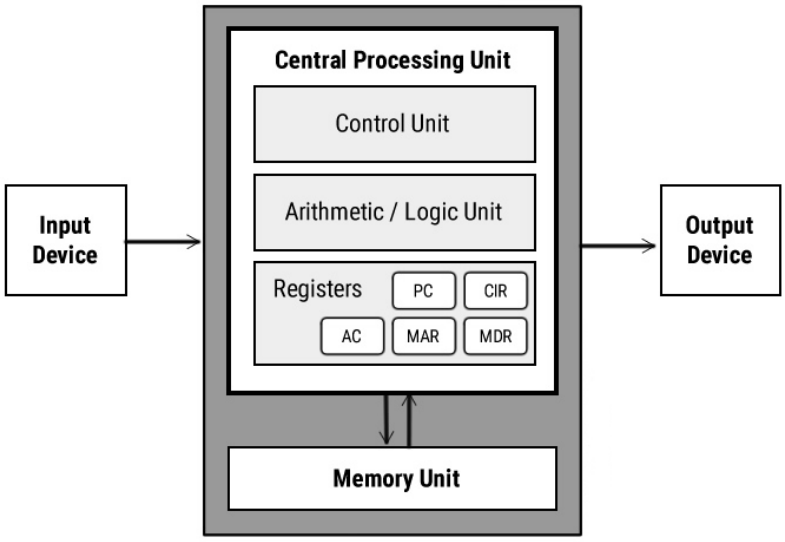


Figure 2.3: Schematic overview of Von Neumann architecture. Adapted from[2]

2.1.3 Neural Networks

Besides brains being more energy and area efficient than man-made computers, they have also been significantly better in performing certain tasks. While computers have long outperformed humans when doing arithmetic, performing tasks like pattern recognition have long been very difficult. Therefore, inspiration has been taken from biology and neural networks were introduced in the field of computer science.

Maass identifies three different generations of neural networks[12]:

1. Digital Neural Networks: neurons have a time-constant binary state value.

2. Analog Neural Networks: neurons have a time-constant analog state value.

3. Spiking Neural Networks: neurons have a time-varying analog state value.

Generation 1 neural networks are very rudimentary and are rarely used in practice. The second generation finds wide application in software usage. Meanwhile, the third generation is interesting for neuromorphic computing applications. The latter of these two will be explained in more detail below.

2.1.3.1 Second Generation

Second generation neural networks consist of multiple neurons that all have numeric state value. These neurons are connected by directional weighted connections. The neuron computes the sum of all neurons that are connected to it weighted by the value of the corresponding weights. The neuron's output is this sum passed through an activation function, e.g. a sigmod function ($\sigma(y) = \frac{1}{1+e^{-y}}$).

Networks from this generation are mostly layer-based. Which means that all neurons are arranged in layers (see Figure 2.4). The first layer is the input layer and consists of N_i neurons. Any data that is inputted to the network is encoded to N_i values and used as state values for the neurons in the input layer. Values of all other neurons are then sequentially calculated. The output of the network can be determined from the state values of neurons in the last layer: the output layer. For example, in the case of digit recognition, there could be one output neuron for every digit and the neuron with the highest value represents the recognised digit.

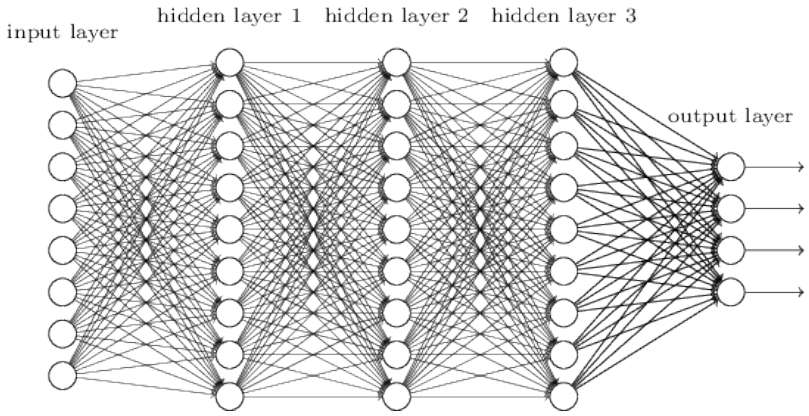


Figure 2.4: Overview of a 5-layer neural network[3].

Essential for the correct functionality of these neural networks is its set of connection weights. The most common way to arrive at a set of working weights, is using a training algorithm in combination with a set of training data. The general working principle of these algorithms is that they iteratively adjust the weights to get the known desired output at the output layer for the inputs of the training data.

These networks have been widely used for tasks like image classification[13], speech recognition[14], face recognition[15] and fraud detection[16]. Implementation specifics, like network structure, vary between applications, but all applications are roughly based on the method described previously. One notable thing is that most applications are for problems that it is difficult to write an algorithm for but that humans are generally considered to be good at.

Second generation neural networks are most often implemented in software. Which means that they run on general purpose hardware. Therefore their main benefit is in the type of applications they can perform, not necessarily power efficiency.

2.1.3.2 Third Generation

Second generation networks have a big difference with biological computation. Whereas brains use the timing of spikes to convey information, in first and second generation networks information is represented by values that are independent of time. To come closer towards the biological functionality, the third generation of neural networks was developed: Spiking Neural Networks (SNN)[12].

Like the previous generation of neural networks, SNNs also consist of many interconnected neurons. Instead of using analog values for encoding information, SNNs are more biological accurate by using spikes. Information is encoded in the relative timing of these spikes compared to both of the same neuron and other neurons.

Most commonly used neuron is the leaky integrate-and-fire neuron as described in 2.1.1. This means that every neuron has an internal state value. This value is modified by the connection strength when a neuron receives an incoming spike on one of its connections. Just as its biological equivalent, neurons in an SNN often have a refractory period in which they do not produce any spikes. This also determines the maximum rate at which a neuron can fire.

Input and output to the network is encoded in the timing of spikes for neurons in their respective layers. Encoding can be either based on the relative spiking frequencies of different neurons or on the relative timing of singles spikes for different neurons.

2.2 Neural Net Topologies

Over the years, SNNs have been used for a variety of applications. The network topologies used vary. But overall there are a few recurring types.

2.2.1 Feed-Forward Network

First of all, the feed-forward topology. As these kinds of networks also find a wide application in generation 1 and 2 neural networks, their application in SNN's is not surprising. A feed-forward network consists of multiple layers of neurons with connections only going to the same or subsequent layers (as described in Section 2.1.3). For example, in both [17] and [18] a 5-layer feed-forward network is used for image recognition.

2.2.2 Self-Organizing Map

A second topology that found its origin in earlier generation neural networks, is the Self-Organizing Map (SOM). These networks consist of two layers of neurons: an input layer and an output layer. Connections between these layers are chosen (semi-)random. As inputs, features of the input signal are encoded to spikes. What these features are, depends on the application. By applying a self-learning algorithm, the network is then trained to give the (mostly) equal outputs for inputs that are similar and give a different output for inputs that are not similar. These networks are therefore very suitable for classification problems. Important to recognize is that the SOM on itself will not tell what the defining characteristic of a class of similar inputs is. For example, in the case

of digit recognition, the SOM will classify two samples of the digit '2' as similar, but it will not output what digit it is. To actually output '2', an extra layer of processing is needed that links the recognized classes into numbers.

In [19] a SOM is used for speech recognition. The inputs features to the network are the relative power of different notes corrected for the human selectivity to certain frequencies (Mel-Frequency Cepstral Coefficients) in a speech sample. This is used to classify these samples into different recognized words. In the network used, the input layer is fully connected to all neurons in the output layer.

A SOM is used for ECG-signal classification in [20]. An input encoder is used to convert the ECG signals into spikes for the neural net. This network consists of 10 input neurons, that all have connections to a random set of circa 25 of the 100 outputs neurons. The neurons in the output layer are all fully connected to each other.

2.2.3 Liquid State Machine

The third commonly used topology for SNN's is a Liquid State Machine (LSM). This topology is based on three layers: an input and output layer with a liquid state in between (Figure 2.5). This liquid state contains a large set of neurons that are quasi-randomly connected to each other. In most implementations, the chance that two neurons are connected is inversely related to their distance in a 2D or 3D grid. The same is true for connections from the input layer to the liquid state and from the liquid state to the output layer.

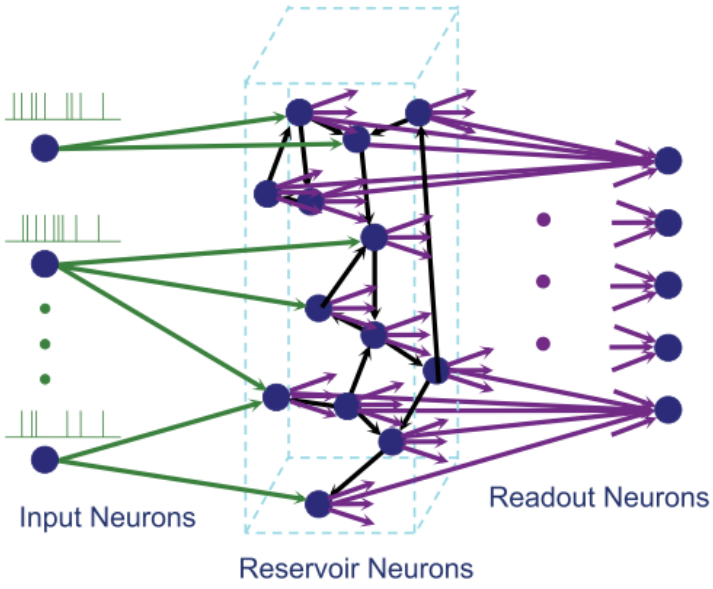


Figure 2.5: Schematic overview of a Liquid State Machine. Dots represent neurons and the arrows represent synaptic connections. The reservoir neurons form the liquid state. [4]

In [4] a LSM is used for word recognition in speech. Just as in [19] (see Section 2.2.2),

Mel-Frequency Cepstral Coefficients are used as the inputs to this network. Any two neurons a and b have their connection probability defined as $P_{conn}(a,b) = C \cdot e^{\frac{-D^2(a,b)}{\lambda^2}}$. With $D(a,b)$ being the euclidean distance between two neurons in a 3D grid and C and λ being constants.

2.3 Neuromorphic Hardware

With the limits of IC scaling getting nearer[21][22], the interest in application specific hardware increases. Using specialised hardware can provide an improved power efficiency, reduced chip area and improved latency and throughput at the cost of reduced application flexibility when compared to general-purpose computing hardware. For these reasons, creating a hardware architecture specifically for SNN's can be very beneficial.

In fact, SNNs are very well suited for implementation in hardware. Biological neurons and synapses have a non-linear response that can be closely mimicked with only a few transistors[23][24]. This means that can be implemented in analog hardware instead of having to simulate their behaviour in digital logic. This means a possible reduction of power consumption by multiple orders of magnitude[9].

2.3.1 AER

One problem that arises with implementing an SNN in hardware is transporting spikes between neurons. The naive method is to create a wire for every connection. However, this does not scale well to a high connection count and does not allow any reconfigurability for the connections.

Therefore [25] proposed the method of Address Event Representation (AER). With AER all spike events are encoded as addresses and sent over a bus (Figure 2.6). At the synapses, these addresses are decoded into the correct synaptic activations. The used addresses can either represent the source neuron of the destination synapse. The time of the event itself does not have to be encoded as long as the latency of the bus is low and does not vary much. This is generally not a problem because neural spike rates are in the range 0-1000 Hz [26] and bus speeds can be orders of magnitudes higher. This also means the events of many neurons can be multiplexed on a single bus.

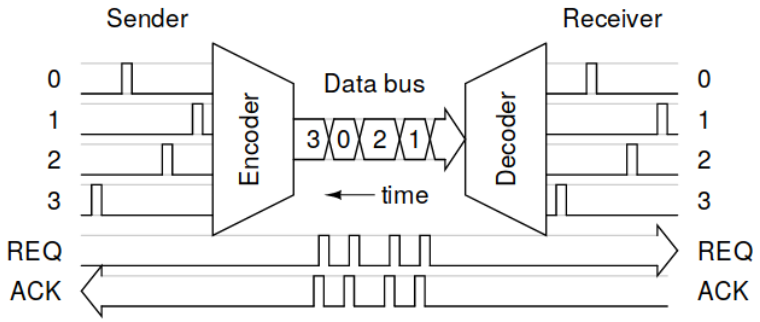


Figure 2.6: Conceptual diagram of AER working as a bus for spiking events.[5]

2.4 State of the Art

2.4.1 Implementation options

2.4.1.1 Array structure

The first difference in the implementation of a neuromorphic system is in how the neuro-synaptic array is structured. In [6] four different types are identified:

Fully dedicated: every neuron has dedicated hardware and every synapse (and its weight) also have dedicated hardware and every connection between neuron and synapses has its own dedicated wire.

Shared axon: neuron and synapses still have dedicated hardware but the connections between neurons and synapses are shared by using a bus.

Shared synapse: every neuron only has a single hardware implementation in hardware. This either means that all synapses have the same weight or that there is some kind of memory for storing the weights of the different synapses.

Shared neuron: in this case multiple neurons share the same hardware. This means that both the synapse weights and the state of the neuron must be stored in a memory.

2.4.1.2 Clustering

Most neuromorphic systems divide the total set of neurons and synapses into clusters to improve scalability. The benefit is that the locality of connections can be utilised. Spikes destined for a synapse in the same cluster, do not have to leave the cluster.

The number of clusters and how interconnection of them is achieved differs.

2.4.1.3 Connectivity storage

If the connections in a neuromorphic system are reconfigurable, some method of storing the current connectivity is needed. The method for encoding connections and the place where the memory is located varies.

2.4.1.4 Supported connectivity

There can be a difference in what kind of connectivity is supported by the implementation. The number of connections a neuron can have is often limited (though this limit might not be reached in any practical applications). There can also be limitations on which neurons the connection can be to. For example, in some implementations there can be limited connections to neurons in other clusters.

2.4.1.5 Simulation speed

For different hardware implementations, it may differ at what speed they run. This can either be similar to biological speeds or faster than that. Biological time has the benefit of being able to handle many real-life signals in real-time, e.g. speech. Keeping as close as possible to biology, so the best way to emulate biological behaviour. Higher than biological speed has the benefit that throughput increases. As long as the input and output are corrected for the fact that the system is running faster. For example, it allows for simulating brains over periods of years in a fraction of the time.

Furthermore, it is also possible to run a system discontinuously. So every event is recorded and provided with a timestamp to be replayed at the desired time. This has the benefit that hardware can be re-used for multiple neurons and synapses, at the cost of having to save spike events in memory.

2.4.2 State of the Art

2.4.2.1 Neurogrid

The Neurogrid system[6] consists of analog neurons and synapses that run at biological speed. Each of the 1 million has 4 separate synapses that can be individually addressed. All synaptic activations are processed sequentially, so only a single synapse can be activated per cluster per clock cycle.

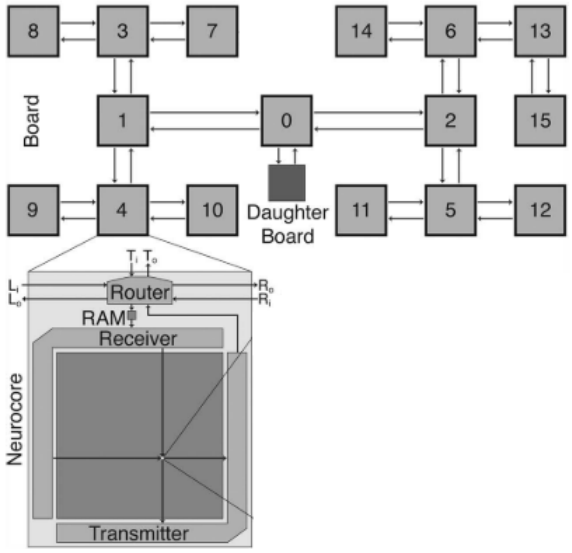


Figure 2.7: Overview of the Neurogrid architecture. Adapted from [6].

Neurons are divided into 16 clusters that are interconnected with a tree network on chip (Figure 2.7). Within a cluster, neurons are arranged in a 256x256 grid. For routing spikes between different clusters, an on-chip memory of 4 Kb per cluster is used. Spikes can only be routed to the synapses of neurons at the same grid location in other clusters with this memory. To route spikes to arbitrary synapses, a daughterboard with 256 Mb RAM is used.

2.4.2.2 SpiNNaker

SpiNNaker[7] is a completely digital neuromorphic architecture to simulate SNN's. A complete system consists of up to 48 SpiNNaker nodes on a single PCB. Each of these nodes is a package containing 18 ARM968 processor cores (Figure 2.8). Every core has its own set of neurons and synapses it processes sequentially, synchronised with a global clock.

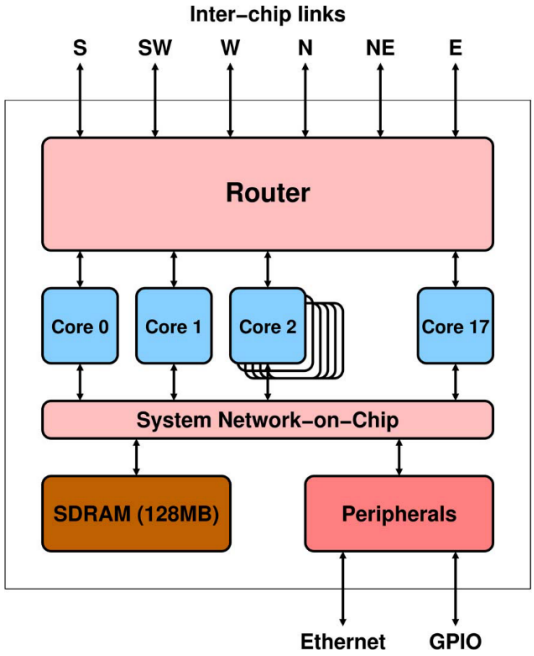


Figure 2.8: Overview of a SpiNNaker node[7].

Within a package, spikes are communicated using a network on chip. To communicate spikes between different nodes, every node is connected to 6 neighbouring nodes with a 32-bit link. Every node has a router with a routing table in a content addressable memory. This table is used to decide to which of its neighbouring nodes a spike should be routed. This routing table is also used to route incoming to any of the node's cores.

2.4.2.3 TrueNorth

This is a completely digital system with its own shared axons-like array architecture. A TrueNorth[27] chip contains 1 million neurons divided over 4096 clusters. Every cluster of 256 neurons has 256 inputs. These inputs are connected to the neurons with a crossbar (Figure 2.9). The activated links on the crossbar represent a synaptic connection. The output spike of a neuron can be routed to exactly one of these inputs and therefore to a maximum of 256 synapses. This input can also be in another cluster. To route spikes between clusters, they are interconnected with a 64x64 mesh grid NoC.

Every cluster contains a routing table with an entry per neuron. This entry deter-

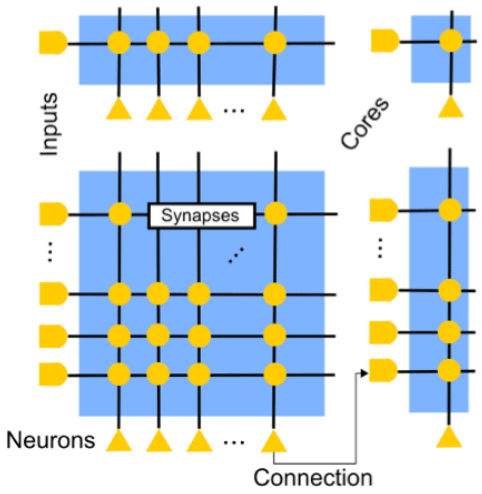


Figure 2.9: Overview of the TrueNorth core. On the left are the inputs and on the bottom are the neurons. They are connected by a switchboard of synapses (the dots). Each of these can individually be turned on or off. The depicted connection is from one core to another. Adapted from [28].

mines to which input in which cluster the spike should be routed. Furthermore, every crossbar has 256x256 bits of memory to store its configuration.

2.4.2.4 ROLLS

ROLLS[23] is a system consisting of 256 neurons with 512 synapses each running. Both have analog implementations and run at biological speed. These neurons are all in a single cluster. Output spikes are directly fed back to the synapses through an AER bus. Connections are not stored explicitly but are implied by the weights of the synapses. If the weight of a synapse between to neurons is non-zero, there is a connection. Therefore it could be reasoned that an equivalent of 256×512 bits of memory is used to store connectivity. This architecture supports any connectivity between neurons is supported, but sparse connectivity leads to a lot of unused synapses.

2.4.2.5 BrainScaleS

BrainScaleS[29] consists of wafers of 180k neurons and 40M synapses. It uses a shared axon architecture with analog synapses and neurons, but weights of the synapses are digitally stored. The system runs at $10^3 - 10^5\times$ biological speed.

Every wafer is divided into 32 clusters (HICANNs). These clusters can route spikes to their neighbours using an on-wafer routing grid. Furthermore, longer-range connections are handled by a 2-layer tree, see Figure 2.10. The storage of the connections takes place across all of these layers. However, what size these memories are, cannot be found. Only that one of the layers uses 120 Mb of memory per wafer. The architecture allows for arbitrary connections between neurons in the same cluster, but connections to other clusters are limited in number. To what degree exactly, is not stated, but it is

assumed that the probability of two neurons being connected drops exponentially over distance.

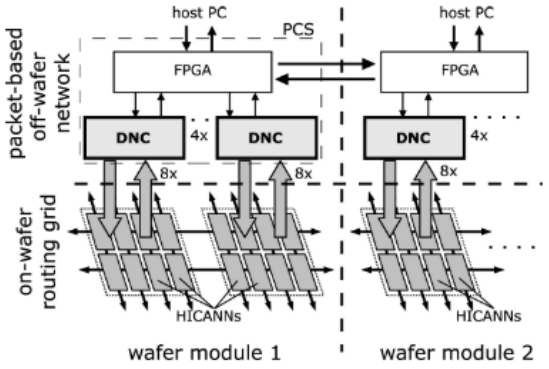


Figure 2.10: Overview of the BrainScaleS architecture[8].

2.4.3 Research Gap

Looking at the current State of the Art, what is missing is a flexible spike-routing architecture usable for single-chip shared axon neuromorphic systems. Neurogrid provides a system for flexible connections but requires a separate daughterboard for anything other than connections to a neuron with the same location in another cluster. The SpiNNaker spike-routing architecture is designed for a shared neuron neuromorphic system. TrueNorth has very limited connectivity by only allowing connections to 256 neurons in a single cluster. ROLLS relies on synaptic weights for defining connections and therefore have a lot of obsolete synapses if connectivity is sparse. Furthermore, the architecture is not expendable beyond a single cluster. Finally, BrainScaleS is fully optimized for a large-scale system. For example, it requires FPGA's for it's routing of spikes between layers.

3 Design Space Exploration

The goal of this chapter is to create a design space exploration for spike-routing in neuromorphic systems. This can be used to quickly estimate the effect of different design parameters on the overall feasibility and limits of the design. First the architecture under consideration is discussed in Section 3.1. Next, different different design parameters are identified in Section 3.2.1. Different interesting output metrics are discussed in Section 3.2.2. After that, the different properties of the system are modelled. Finally, the model is evaluated and the resulting conclusions are stated.

3.1 Architecture Overview

For the model to be further analysed, the system under consideration first has to be described. It is assumed that the system consists of N_{neurons} neurons with on average F_{in} synapses each, totalling to N_{synapses} synapses. Each neuron has an average spike rate R_{avg} at which it produces spikes. Furthermore, each neuron has a refractory period T_{refr}. A neuron will not generate a spike for at least T_{refr} after a previous spike.

3.1.1 Neuro-synaptic array

Neurons and synapses are arranged in a grid called a *neuro-synaptic array*. The array has one output signal for every neuron, which is high if the neuron generates a spike and goes back to low once the spike has been acknowledged by the logic it is connected to. It is assumed that this logic processes simultaneous spikes sequentially.

Synapses are arranged in a two-dimensional grid. The number of rows (N_{rows}) is equal to the number of neurons and the number of columns (N_{cols}) is equal to F_{in}. The grid has a total of $N_{\text{rows}} + N_{\text{cols}}$ inputs, corresponding to each row and column. A synapse is activated by enabling its corresponding row and column input. By activating the input of one column and multiple rows, multiple synapses can be activated simultaneously.

3.1.2 Clusters

To improve the scalability of the system, it can be divided into multiple clusters of neuro-synaptic arrays (see Figure 3.1). The total system consists of N_{clusters} clusters with N_{nc} neurons each. These clusters are connected with a Network on Chip (NoC). See Section 3.6 for a further discussion of the NoC.

Because every neuro-synaptic array has N_{nc} outputs, every cluster has an encoder and lookup table that translate these outputs to one or more addresses. These addresses are then either put on a local link if their destination is the cluster itself or sent to the NoC if they are destined for another cluster. Finally, coming from the local link or the

NoC these addresses are received by another lookup table that converts them to one or more column and row addresses. These addresses are then used to activate the correct column and rows of the synaptic arrays. What the different lookup tables do, depends on the addressing (Section 3.4) and synapse encoding (3.5) schemes used.

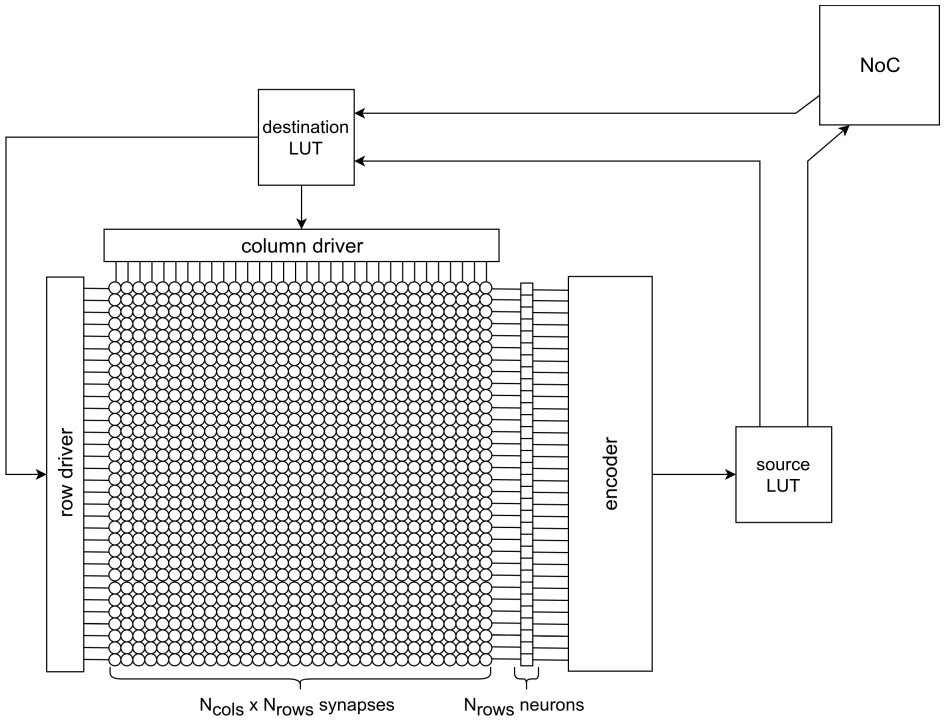


Figure 3.1: Schematic overview of a modelled cluster.

3.2 Design Space

3.2.1 Input parameters

SNN topology parameters

CD is the Connection Distribution and is used to model the characteristics of different SNN topologies (see Section 3.3.1)

$\boldsymbol{N_{\text{neurons}}}$ is the total number of neurons in the topology and therefore dictates the size of the neural net.

$\boldsymbol{F_{\text{in}}}$ is the maximum number of incoming connections a neuron can have. This determines the number of hardware synapses each neuron has.

$\boldsymbol{F_{\text{out}}}$ is the maximum number of outgoing connections a neuron can have. This is mainly used to calculate the sizes of entries in lookup tables (see Section 3.4.4).

Spiking parameters

$\boldsymbol{R_{avg}}$ is the average spike rate of a neuron. It is used to model the spiking behaviour of the neurons.

$\boldsymbol{T_{\text{refr}}}$ is the refractory period of a neuron. From it, the maximum spike rate of a single neuron is calculated.

Architecture parameters

$\boldsymbol{N_{\text{clusters}}}$ is the number of clusters and therefore determines the number of neurons per cluster.

AS is the addressing scheme used (see Section 3.4).

SES is the synapse encoding scheme used (see Section 3.5).

NT is the NoC-topology used (see Section 3.6).

3.2.2 Model outputs

M is the size of different lookup tables in bits. As these lookup tables can be big, they can have a significant impact on the power and area estimations. Furthermore, their size can be used to calculate the memory efficiency of the architecture.

R is the packet rate of different links in the architecture (see 3.4.2). This can be used to estimate the necessary clock frequency.

L is the spike latency, defined as the number of clock cycles between a neural spike and last corresponding synaptic activation. This can be used to check whether spike latency requirements are met.

c is the concurrency of synaptic activations. This gives an insight into how many synapses can be activated in each clock cycle.

A is the estimated area of the spike-routing architecture. It is composed of the areas of different components and can be used to check implementation feasibility on certain die size.

P is the estimated power of the spike-routing architecture. It is composed of the power consumption of different components but does not compose the contain the power consumption of the neuro-synaptic array because that is treated as a black box as much as possible.

For the latter two, a 45 nm process is used for the estimations. The reason for this is that this is the smallest process that is supported by the various power and area models that are used.

3.2.3 Model Components

In the following sections, these points will be under evaluation:

Connectivity: in section 3.3 the connectivity of SNNs is described by simplified connection distributions. These distributions are later used to model the effects of different topologies.

Addressing: section 3.4 describes different Addressing Schemes to use for AER and their effect on the lookup tables, the size and utilisation of local and inter-cluster links and spike latency.

Synapse Encoding: in 3.5 different methods for encoding synaptic activations in the destination lookup table with the goal to increase the synaptic activation concurrency and decrease the size of this LUT.

NoC: section 3.6 describes different topologies and their effect on spike latency and link utilisation.

3.3 Connectivity

An important factor in estimating traffic and latency in the interconnect system is how neurons are connected. This connectivity depends on the topology of the neural network that is used.

3.3.1 Connection distribution

For modelling purposes, the connectivity is approached by a connection probability function. For this, all neurons are considered to be on a numbered line. Let x be the distance between neurons a and b on this line, then the probability of there being a synaptic connection from a to b is given by $f(x)$.

Two different types of distributions will be used: uniform and local. For the uniform distribution, the probability of a connection is independent on the distance. This reflects the connectivity as found in some Self Organizing Maps and Liquid State Machines. The connection probability function for this distribution is:

$$f_{\text{uniform}}(x) = \frac{F_{\text{in}}}{N_{\text{neurons}}} \tag{3.1}$$

For the local distribution the relative position of two neurons influences the chance there is a connection. The closer together, the higher the connection probability. This is mathematically represented by using a scaled exponential distribution:

$$f_{\text{local}}(x) = \lambda e^{\frac{-\lambda x}{C}} \tag{3.2}$$

In equation 3.2, λ is the parameter that describes the locality of the distribution: a higher value means the connections are shorter on average. The parameter C is used

to scale the distribution in such a way that the total average number of connections equals to F_{in}. The value of C can be calculated iteratively using equation 3.3.

$$F_{\text{in}} = \int_0^{N_{\text{neurons}}} \lambda e^{\frac{-\lambda x}{C}} dx = C\left(1 - e^{\frac{-N_{\text{neurons}}\lambda}{C}}\right) \tag{3.3}$$

These probability functions are used later in this chapter to model the different connection distributions.

3.3.2 Notation

In this section, a notation is introduced to count specific connections in the system. This is later used to derive equations for properties of the system.

The construction is always of the following form:

$$\text{Denominator}\,[a \Rightarrow b] \tag{3.4}$$

The part $a \Rightarrow b$ describes all connections that should be select. There are different options for a and b:

$\mathbf{n(i)}$	Neuron i
$\mathbf{c(i)}$	Cluster i
$\mathbf{x}$	Anything

To further clarify, here are some examples:

$\mathbf{n(i)} \Rightarrow \mathbf{c(j)}$	All connections from neuron i to any neuron in cluster j.
$\mathbf{c(i)} \Rightarrow \mathbf{x}$	All connections from any neuron in cluster i
$\mathbf{x} \Rightarrow \mathbf{x}$	All connections.

The first part of the construction is the denominator. It describes what is counted uniquely. It can be one of these options: Synapses, Neurons or Clusters. In the case of Synapses, all synapses in the connection set are counted, for example:

$\textbf{Synapses}\,[\mathbf{x} \Rightarrow \mathbf{c(i)}]$	The number of synapses from any neuron to cluster i.
$\textbf{Synapses}\,[\mathbf{c(i)} \Rightarrow \mathbf{c(j)}]$	The number of synapses from cluster i to cluster j.
$\textbf{Synapses}\,[\mathbf{x} \Rightarrow \mathbf{n(i)}]$	The number of synapses from any neuron to neuron i.

When Neurons or Clusters is used, either a or b should be $\mathbf{x}$. The result is the number of unique neurons or clusters respectively at the place of the $\mathbf{x}$. To illustrate:

3.4 Addressing

The basis of using AER is spike-routing using addresses. What addressing scheme is used, has an impact on the performance and size of the architecture. In this section, first different addressing schemes are discussed and afterwards the impact of these addressing scheme on these properties is modelled.

Neurons $[\mathbf{x} \Rightarrow \mathbf{c(i)}]$	The number of neurons that has any connection to cluster i.
Neurons $[\mathbf{c(i)} \Rightarrow \mathbf{x}]$	The number of neurons in cluster i (with at least one connection).
Clusters $[\mathbf{n(i)} \Rightarrow \mathbf{x}]$	The number of clusters that neuron i has at least one connection to.
Clusters $[\mathbf{x} \Rightarrow \mathbf{n(i)}]$	The number of clusters that neuron i has at least one connection originating from.

3.4.1 Addressing Schemes

In this section three different addressing schemes are discussed: Source Addressing, Destination Addressing and Hybrid Addressing. The first of these two have been under consideration since the early days of AER[30]. The latter is a new proposal for an addressing scheme.

3.4.1.1 Source Addressing

The first addressing scheme is Source Addressing (Figure 3.2). In this scheme, the address of the firing neuron is encoded in the AER packets. When it leaves the cluster, the address of the cluster should be appended to make it globally unique. Because from the source address alone it cannot be told to what the destinations of the spike are, the packet is sent to all clusters. Since the packet is equal for all destinations, it can be broadcast. The destination cluster contains a Lookup Table (LUT). This is used to convert the source address to a sequence of destination synapses.

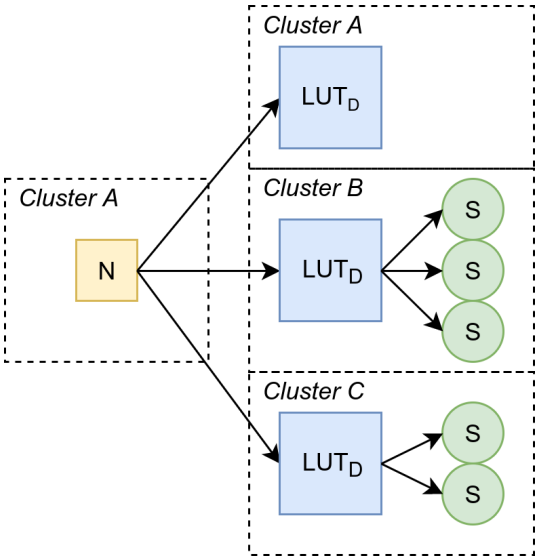


Figure 3.2: Schematic overview of Source Addressing. When a neuron (N) fires, it sends a packet to all clusters (including itself), with the address of the neuron that fired. In these clusters a LUT is used to convert this source address into a list of synapses (S) that should be activated.

3.4.1.2 Destination Addressing

The second addressing scheme is Destination Addressing (Figure 3.3). In this scheme, the Lookup Table is moved to the source cluster. Here the address of the spiking neuron is converted into a list of synapse addresses with their corresponding clusters. These are then routed to the correct cluster by the NoC. In the destination cluster, the addresses are used to activate the correct synapses.

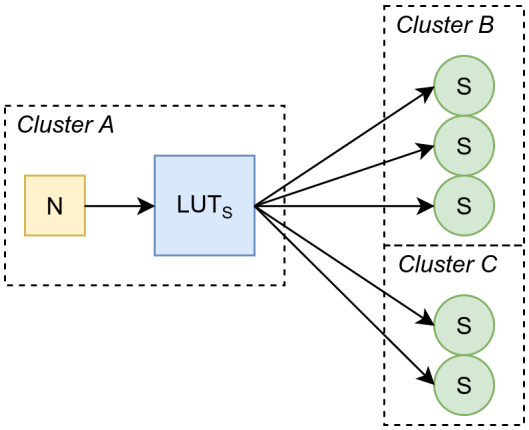


Figure 3.3: Schematic overview of Destination Addressing. When a neuron (N) fires, all destination synapses are looked up in a LUT. These are then sent to the cluster in which the synapse resides, where the synapse (S) is activated.

3.4.1.3 Hybrid Addressing

Finally, the scheme we propose is Hybrid Addressing. This is a combination of the previous schemes. A two-stage lookup is used. In the source cluster, the address of the spiking neuron is converted to a list of intermediate addresses that all correspond to a single cluster with at least one destination synapse in it. These intermediate addresses are then routed to their corresponding clusters. There the second lookup is performed. The intermediate address is translated to a list of synapses that should be activated.

3.4.2 Traffic

For modelling the traffic within and between clusters, the model as shown in Figure 3.5 is used. This describes the flow of spikes from neurons to synapses, expressed in packet rates. The packet rates can be used to determine the clock frequency of the system. The clock frequency should always be higher than the packet rate at any point to avoid congestion.

R_{neurons} is the spike rate of the neurons in a cluster and λ_{out} is the number of packets per spike. The rate of outgoing packets per cluster is given by R_{out}. These packets are distributed over the local link and the NoC by a factor k. The total incoming rate (R_{in}) consist of the traffic from the local link and the incoming traffic from the NoC. Every incoming packet can causes λ_{in} synaptic activations, leading to a synaptic activation rate of $R_{\text{activation}}$.

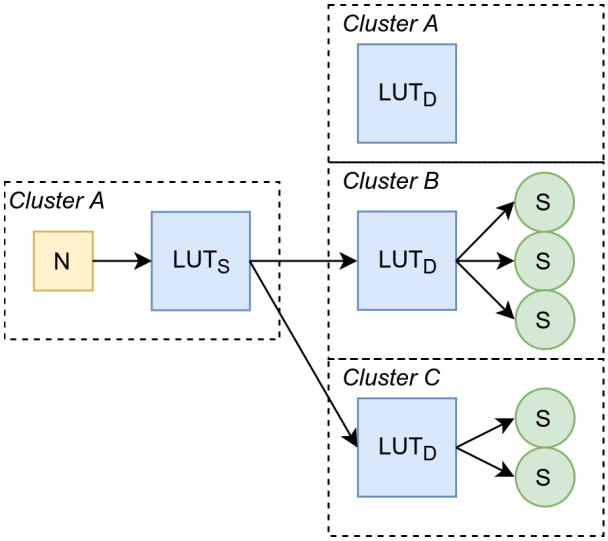


Figure 3.4: Schematic overview of Hybrid Addressing. When a neuron (N) fires, an intermediate address is looked up for all clusters that contain at least one synapse that should be activated. These are then sent to the corresponding cluster, where a LUT is used to translate these intermediate addresses to synapses (S) to activate.

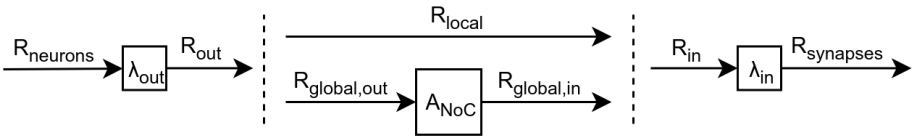


Figure 3.5: Traffic flow from neurons to synapses in a cluster.

These relations can be summarised in the following equations:

$$R_{\text{out}} = \lambda_{\text{out}} R_{\text{neurons}} \tag{3.5}$$
$$R_{\text{local}} = k R_{\text{out}} \tag{3.6}$$
$$R_{\text{global,out}} = (1-k) R_{\text{out}} \tag{3.7}$$
$$R_{\text{out}} = R_{\text{local}} + R_{\text{global,out}} \tag{3.8}$$
$$R_{\text{in}} = R_{\text{local}} + R_{\text{global,in}} \tag{3.9}$$
$$R_{\text{global,in}} = A_{\text{NoC}} R_{\text{global,out}} \tag{3.10}$$
$$R_{\text{activation}} = \lambda_{\text{in}} R_{\text{in}} \tag{3.11}$$

The parameters k, A_{NoC}, λ_{out} and λ_{in} are determined by the used addressing scheme (Section 3.4.1) and the connection distribution (Section 3.3.1). Combing all these equations leads to the following equation for the synaptic activation rate:

$$R_{\text{activation}} = \lambda_{\text{in}} \lambda_{\text{out}} \left[k + A_{\text{NoC}} (1-k) \right] R_{\text{neurons}} \tag{3.12}$$

Because the average number of synaptic activations coming from a single neuron activation is F_{in}, it must be true that:

$$F_{\text{in}} = \lambda_{\text{in}} \lambda_{\text{out}} \left[k + A_{\text{NoC}} (1-k) \right] \tag{3.13}$$

This can then be used to calculate the parameters for different addressing schemes.

Let's first consider Source Addressing. Because the source address is sent twice (once to the NoC and once to the local link), $\lambda_{\text{out}} = 2$ and $k = 0.5$. Because addresses are always broadcasted to all clusters, the incoming traffic is higher: $A_{\text{NoC}} = N_{\text{clusters}} - 1$. From 3.13 then follows that $\lambda_{\text{in}} = \frac{F_{\text{in}}}{N_{\text{clusters}}}$.

Secondly, for Destination Addressing, all address translation is done in the originating cluster, so $\lambda_{\text{out}} = F_{\text{in}}$ and $\lambda_{\text{in}} = 1$. The distribution between local and global traffic depends on the distribution of connections: $k = \frac{\text{Synapses}[c \Rightarrow c]}{\text{Synapses}[c \Rightarrow x]}$. Because all packets going in the NoC have a single destination cluster, $A_{\text{NoC}} = 1$.

Finally, for Hybrid Addressing λ_{in} and λ_{out} are dependent on the connection distribution. λ_{out} is equal to the average number of clusters neurons have synapses in. Since A_{NoC} is also 1 for this scheme, that means that $\lambda_{\text{in}} = \frac{F_{\text{in}}}{\lambda_{\text{out}}}$. Finally, if we assume that every neuron has at least 1 connection in its own cluster, this means that $k = \frac{1}{\lambda_{\text{out}}}$.

If the f_{clock} of the system is known, these different link rates can be converted into link utilisations with the following equation:

$$U = \frac{R}{f_{\text{clock}}} \tag{3.14}$$

3.4.3 Address Sizes

As the addresses are different for all three schemes, so is their size. The size of the address is dependent on what it uniquely has to describe. In the case of Source Addressing, the address is used to identify the neuron that fired. That means that every neuron in the system should have a unique address. With Destination Addressing, the address is used to identify a synapse in a cluster. So every synapse in a cluster should have a unique address. Furthermore, that address should arrive in the correct cluster. Therefore, an address describing the cluster is appended.

The case of Hybrid Addressing is a little more complicated. Here the size of the address is dependent on the connectivity. Every neuron that has at least one synapse in a certain cluster has an entry in its lookup table. That means the size of the address is dependent on the number of neurons that have synapses in a cluster. Furthermore, these intermediate addresses have to be routed to the correct cluster, so they are appended with an address uniquely describing a cluster.

This all leads to the following equations for the sizes of the addresses:

$$S_{SA} = \lceil \log_2(N_{\text{neurons}}) \rceil \tag{3.15}$$

$$S_{DA} = \lceil \log_2(N_{sc}) \rceil + \lceil \log_2(N_{\text{clusters}}) \rceil \tag{3.16}$$

$$S_{HA} = \lceil \log_2(\text{Neurons}\,[x \Rightarrow c(i)]) \rceil + \lceil \log_2(N_{\text{clusters}}) \rceil \tag{3.17}$$

These sizes can then be used as the widths of the links for the NoC.

3.4.4 Lookup Tables

Depending on the used addressing at different places in the system Lookup Tables are used to store connections. These LUTs are used for 1 to many lookups. Where the

number of outputs for a single input is variable. To accommodate for this, the lookups are split in 2 steps (Figure 3.6). In the first step, the number of lookups and an offset for the second step are looked up. In the second step, the corresponding number of lookups is executed.

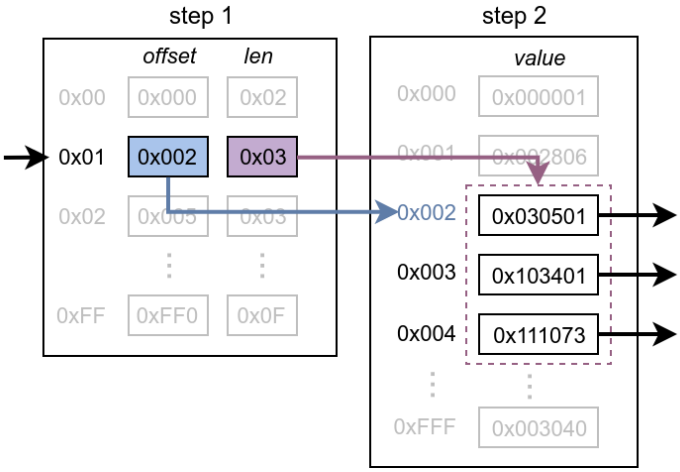


Figure 3.6: Example of a two stage 1 to many lookup. In the example the input `0x01` is converted to three output values.

In the case of Source Addressing and Destination Addressing one 2-step lookup per cluster is needed. In the case of Hybrid Addressing a 2-step lookup is needed in both the source and the destination cluster. In Table 3.1 and Table 3.2 the sizes of the LUTs for different steps can be found.

Table 3.1: Size of the step 1 LUTs for different addressing schemes for cluster i.

			Entry width	
Scheme	*LUT*	*Number of entries*	*Offset*	*Length*
SA	S1	N_{neurons}	$\lceil log_2 (\text{Synapses}\,[x \Rightarrow c(i)]) \rceil$	$\lceil log_2(F_{\text{out}}) \rceil$
DA	D1	N_{nc}	$\lceil log_2 (\text{Synapses}\,[c(i) \Rightarrow x]) \rceil$	$\lceil log_2(F_{\text{out}}) \rceil$
HA	S1	N_{nc}	$\left\lceil log_2 \left(\sum_{n \,\epsilon\, c(i)} \text{Clusters}\,[n \Rightarrow x] \right) \right\rceil$	$\lceil log_2(N_{\text{clusters}}) \rceil$
	D1	$\text{Neurons}\,[x \Rightarrow c(i)]$	$\lceil log_2 (\text{Synapses}\,[x \Rightarrow c(i)]) \rceil$	$\lceil log_2(F_{\text{out}}) \rceil$

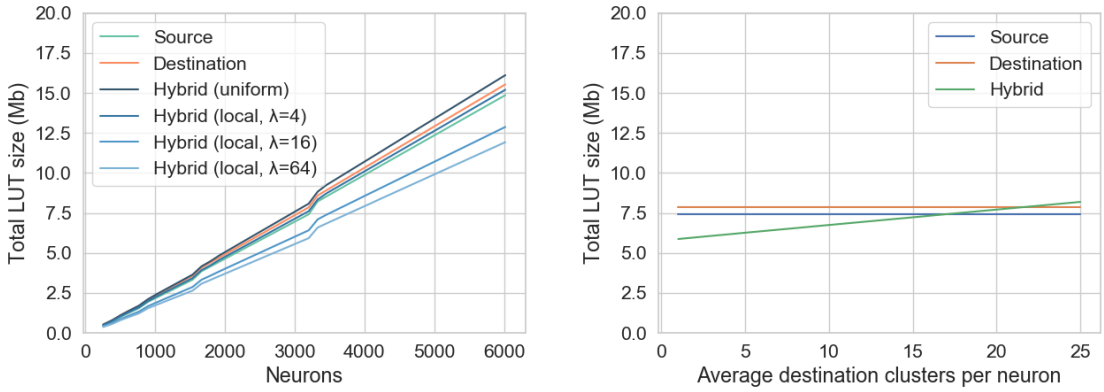
The sizes of the different lookup tables are dependent network topology they have to encode. In the case of Source and Destination Addressing it is only the number of neurons, synapses and clusters that influence the size. For Hybrid Addressing the distribution of connections over clusters is also a factor. The expected size of the LUTs for different neuron counts can be found in Figure 3.7a.

The Hybrid Addressing scheme becomes more memory-efficient for more local connection distributions. This is because Source Addressing has an entry in the step 1 LUT for each neuron in every cluster, where Hybrid Addressing only has an entry in the destination cluster if there is actually a destination synapse. For local connection

Table 3.2: Size of the step 2 LUTs for different addressing schemes for cluster i.

Scheme	*LUT*	*Number of entries*	*Entry width*
SA	S2	Synapses $[x \Rightarrow c(i)]$	$\lceil \log_2(N_{sc}) \rceil$
DA	D2	Synapses $[c(i) \Rightarrow x]$	$\lceil \log_2(N_{sc}) \rceil + \lceil \log_2(N_{\text{clusters}}) \rceil$
HA	S2	$\sum_{n \, \epsilon \, c(i)}$ Clusters $[n \Rightarrow x]$	$\lceil log_2(\text{Neurons}\,[x \Rightarrow c(i)]) \rceil + \lceil \log_2(N_{\text{clusters}}) \rceil$
	D2	Synapses $[x \Rightarrow c(i)]$	$\lceil \log_2(N_{sc}) \rceil$

distributions, the number of clusters with a destination synapse is often smaller than the total number of clusters, giving the Hybrid scheme its benefit in memory efficiency. This effect can also be observed in Figure 3.7b: as the number of destination clusters decreases, so does the expected LUT size for Hybrid Addressing.



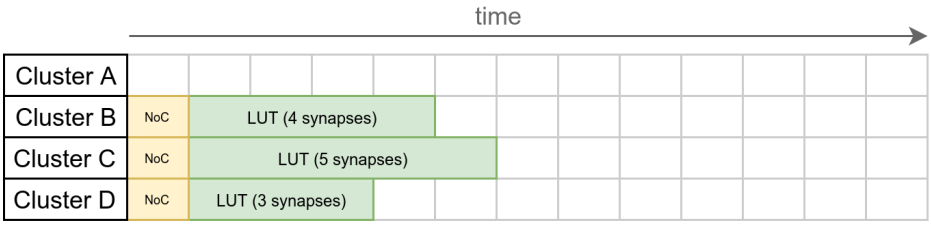
(a) (b)

Figure 3.7: Modelled LUT sizes for the different addressing schemes. The number of clusters is 25 and every neuron has 128 synapses. The first plot (a) shows the sizes for different numbers of neurons. For Destination and Source Addressing there is no difference for different connection distributions. For Hybrid Addressing the LUT sizes for different values of λ is plotted. In the right plot (b) the number of neurons is kept constant at 3200, but the average number of clusters they have synapses in is variable.

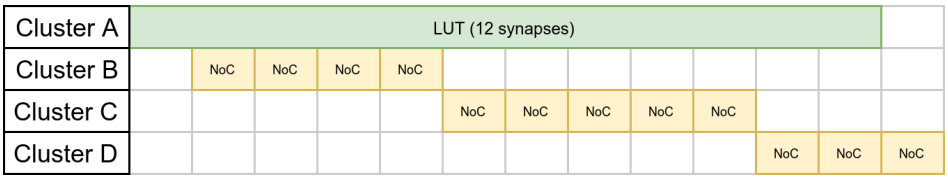
3.4.5 Latency

Because of the difference in lookup structure, all three addressing schemes have differences in expected latencies for transferring spikes. The main reason for this lies in whether lookups can be performed in parallel. As illustrated in Figure 3.8, Source Addressing and Hybrid Addressing can distribute lookups over multiple clusters, which reduces the latency.

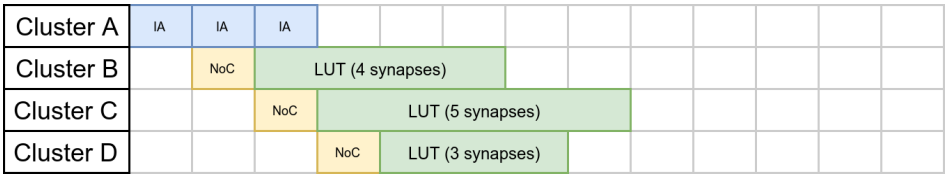
To quantify the difference in latency, we look at the latency assuming destinations are evenly distributed over n clusters. The time differences between a neuronal



(a) Source Addressing



(b) Destination Addressing



(c) Hybrid Addressing

Figure 3.8: Example of the differences in latency for the different addressing schemes. The horizontal axis is the time scale after with the leftmost point representing a spike event of a neuron in cluster A with 12 synapses. The yellow box with NoC represent activity in the NoC and the green boxes with LUT represent the activity of the LUT. For simplicity, it is assumed that all routing and lookup take equal time and are independent of distance. For Source Addressing (a) three different LUTs can work in parallel after the source address has been transmitted. With Destination Addressing (b), all addresses have to be looked up sequentially in cluster A. In the Hybrid Addressing scheme (c) the intermediate addresses have to be looked up sequentially in cluster A, but further lookups can be performed in parallel in the destination clusters.

spike event and the latest corresponding synaptic activation for the different addressing schemes is given by:

$$L_{SA} = L_{\text{NoC}} + L_{\text{LUT}} \cdot \frac{F_{\text{in}}}{n} \tag{3.18}$$

$$L_{DA} = L_{\text{NoC}} + L_{\text{LUT}} \cdot F_{\text{in}} \tag{3.19}$$

$$L_{HA} = L_{\text{NoC}} + L_{\text{LUT}} \cdot \left\lceil \frac{F_{\text{in}}}{n} + n \right\rceil \tag{3.20}$$

With L_{NoC} being the latency for transporting a packet across the diameter of the NoC and L_{LUT} the latency for doing a single lookup in a LUT. This is visualised in Figure 3.9.

For Source Addressing, latency decreases as the number of clusters increases. However, for Hybrid Addressing there is an optimum in the number of clusters because

at a certain point the extra time for looking up intermediate addresses outweighs the benefit of performing lookups in parallel.

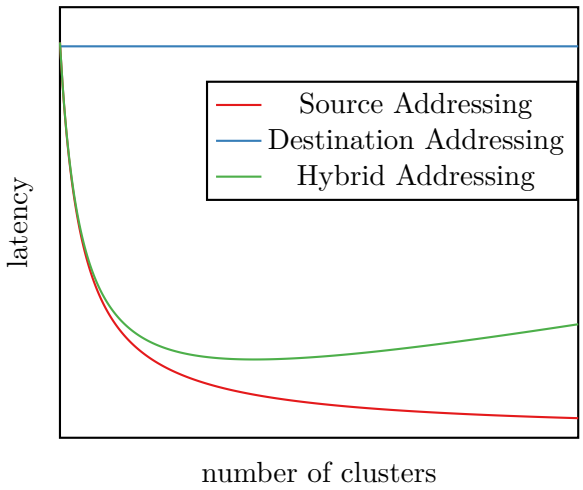


Figure 3.9: Trend of the change in latency due to parallelisation over multiple clusters for the different addressing schemes. An even distribution of neurons over clusters is assumed.

3.5 Synapse Encoding

Until this point, it was assumed that for every synapse that has to be activated, an address is stored in a LUT. However, to increase memory efficiency and concurrency, several optimisations can be made to destination synapses are encoded in the lookup tables. In this section, some proposed measures will be described.

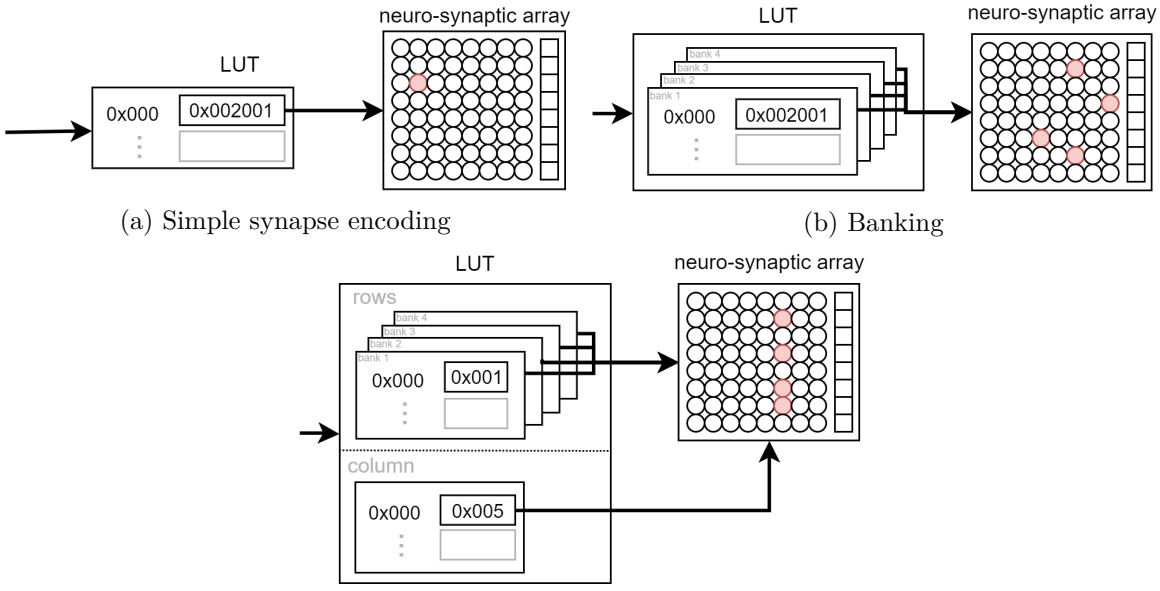
3.5.1 Banking

The first improvement is banking the lookup tables. By splitting the LUT in B banks and sending the same lookup address to all banks, the addresses of B synapses can be looked up in parallel (see Figure 3.10b). In the best case scenario, this improves concurrency by a factor B without a reduction in memory efficiency.

However, in practice this is an upper bound. Because if the number of destination synapses corresponding to the same presynaptic neuron is not an exact multiple of B, an entry in one or more of the banks will be unused. To capture this numerically, the concept of *banking efficiency* (η_{banking}) is introduced. This has a value between $\frac{1}{B}$ and 1 and represents how well the synapses of a presynaptic neuron can be divided over multiple banks. The efficiency for the synapses in cluster c of presynaptic neuron n is defined as:

$$\eta_{\text{banking}}(n, c) = \frac{\text{Synapses}\,[n \Rightarrow c]}{B \left\lceil \frac{\text{Synapses}[n \Rightarrow c]}{B} \right\rceil} \tag{3.21}$$

With B being the number of banks. To make this concept more practical, from now



(a) Simple synapse encoding

(b) Banking

(c) Column reuse

Figure 3.10: Illustrations of different improvements for synapse encoding. Simple synapse encoding (a) is the base situation where the row and column address of every synapse are stored in the LUT and one synapse is activated at a time. With banking (b) multiple synapses are activated simultaneously by splitting the lookups over multiple banks. Finally, with column reuse (c) the same column address is stored only once for multiple synapses.

on the average banking efficiency for a cluster is used. This is defined as:

$$\bar{\eta}_{\text{banking}}(c) = \frac{1}{\text{Synapses}\,[x \Rightarrow c]} \sum_{n \,\epsilon\, c} \eta_{\text{banking}}(n, c)\,\text{Synapses}\,[n \Rightarrow c] \tag{3.22}$$

The banking efficiency can now be used to calculate the effective increase in concurrency for lookups:

$$c = \bar{\eta}_{\text{banking}}\, B \tag{3.23}$$

As stated before, if banking cannot be performed optimally, memory efficiency is lowered. For example, take the case that there are 15 destination synapses and 4 banks. In the first 3 activation rounds, all banks output the address of 1 synapse. However, in the last round, one bank is unused, leaving an unused entry in the LUT. Memory efficiency is reduced by a factor $\frac{1}{\eta_{\text{banking}}}$.

When two synapses (a and b) that are in different rows and columns are activated simultaneously, the row and column select lines of both synapses have to be enabled. This activates not only synapse a and b, but also the synapse that is in the row of synapse a and the column of synapse b and vice versa (see Figure 3.11). To circumvent this problem, only synapses in the same column are activated simultaneously.

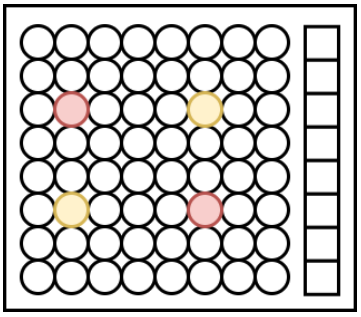


Figure 3.11: Example of what happens when two synapses in different columns and row are activated simultaneously. In this case, the red synapses are meant to be activated and the yellow synapses are also activated as a side effect.

3.5.2 Column Reuse

If banking is applied, an extra improvement that can be made is Column Reuse. This means that column and row addresses are both stored in separate banks of the lookup table: B banks for row addresses and only 1 for a column address. So the same column address is used for B synapses (see Figure 3.10c).

This improves memory efficiency as the number of bits needed for storing column addresses is lowered. The average number of bits in the LUT needed in cluster c when using Column Reuse is given by:

$$\bar{M}_{\text{synapse}}(c) = \frac{\lceil log_2(N_{\text{columns}}) \rceil}{\bar{\eta}_{\text{banking}}(c)\, B} + \lceil log_2(N_{\text{rows}}) \rceil \tag{3.24}$$

Column reuse can have a negative impact on the mapping efficiency of the cluster. Because every synapse of the same activation round has to be placed in the same column. This limits the placement options when mapping synapses to the neuro-synaptic array.

3.5.3 Row Grouping

The third improvement is Row Grouping. This means combining multiple rows into sets of g rows. LUTs for rows now contain two values: the address of a Row Set and g Select bits. The latter encodes which of the rows in the set that should be activated. This way up to g rows can be activated with a single lookup (see Figure 3.12), increasing concurrency.

Row Grouping is especially beneficial when multiple rows in the same set are activated simultaneously. Let η_{grouping} be the grouping-efficiency, defined as the average fraction of activated rows per row set per LUT entry. Then the average memory per synapse is (without any banking):

$$\bar{M}_{\text{synapse}}(c) = \frac{\lceil \log_2(N_{\text{columns}}) \rceil + \lceil \log_2(N_{\text{rows}}) - \log_2(g) \rceil + g}{\bar{\eta}_{\text{grouping}}(c) \cdot g} \tag{3.25}$$

Figure 3.13 shows the effects of different set sizes. Bigger groups lead to higher memory saving potential, but the penalty for a low grouping efficiency also increases.

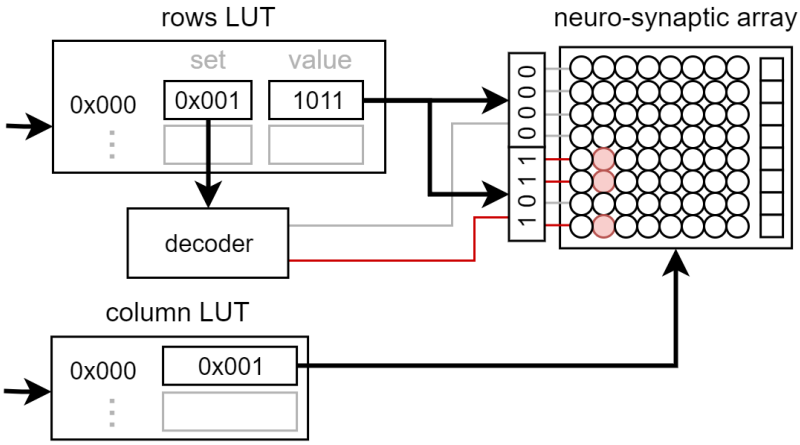


Figure 3.12: Schematic representation of Row Grouping. The LUT for the rows contains the number of a Row Set and a corresponding value. This value represents the rows in the set that should be activated.

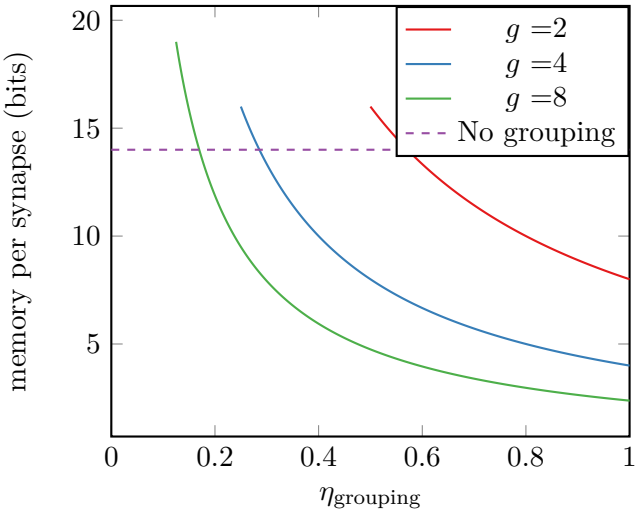


Figure 3.13: Memory per synapse for increasing grouping efficiency. Different set sizes are compared to not using row grouping at all. Calculations are for a cluster with 128 neurons with 128 synapses each. As $\frac{1}{g} \leq \eta_{\text{grouping}} \leq 1$, the lines are only plotted for that domain.

3.5.4 Column Address Offsetting

The final improvement to be discussed is offsetting column addresses. This is done by using the B_{offset} highest significant bits of the LUT lookup address as the highest significant bits for the column address. This decreases the memory required for storing column addresses but also decreases mapping flexibility.

The average memory used per synapse (assuming no other improvements are applied) is:

$$\bar{M}_{\text{synapse}}(c) = \lceil \log_2(N_{\text{columns}}) \rceil - B_{\text{offset}} + \lceil \log_2(N_{\text{rows}}) \rceil \tag{3.26}$$

3.5.5 Combining

The improvements from previous can be combined into a single architecture. If all are applied, that leads to the following equation for the average memory per synaptic connection:

$$\bar{M}_{\text{synapse}}(c) = \frac{\lceil \log_2(N_{\text{columns}}) \rceil - B_{\text{offset}}}{\bar{\eta}_{\text{banking}}(c) \cdot B \cdot \bar{\eta}_{\text{grouping}}(c) \cdot g} + \frac{\lceil \log_2(N_{\text{rows}}) - \log_2(g) \rceil + g}{\bar{\eta}_{\text{grouping}}(c) \cdot g} \tag{3.27}$$

The combined effect on the concurrency from Banking and Row Grouping can simply be multiplied:

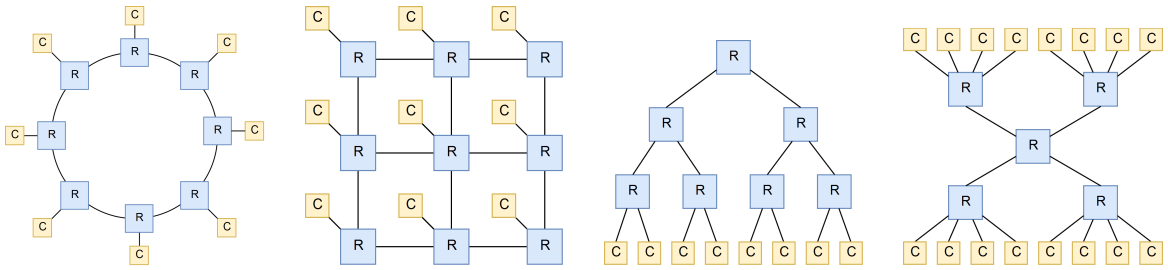
$$c = \eta_{\text{banking}}\, B \cdot \eta_{\text{grouping}}\, g \tag{3.28}$$

The overall effect on the mapping efficiency is more difficult to capture in an equation. Banking, Row Grouping, Column Address Offsetting have a potential negative impact on what fraction of the synapses in the neuro-synaptic array can be used.

3.6 NoC

3.6.1 Topology

Different network topologies can be used to arrange the clusters into a network. The model supports four different options: ring, mesh, binary tree (B-tree) and quaternary tree (Q-tree). A schematic overview of their layout can be found in Figure 3.14. The choice of topology has an effect on the latency, the bandwidth and the area needed for implementation.



(a) Ring (b) Mesh (c) B-tree (d) Q-tree

Figure 3.14: Overview of NoC topologies supported by the model. Yellow boxes represent clusters and the blue boxes represent routers.

3.6.2 Link load

For the analysis of the different topologies, the properties as found in Table 3.3 are used. The number of links is used to calculate the average packet rate per link. If all traffic is evenly divided over all links, this is:

$$R_{\text{link}} = \frac{N_{\text{clusters}} \cdot R_{\text{global,in}} \left(\bar{H} + 1\right)}{N_{\text{links}}} \tag{3.29}$$

Table 3.3: Properties of the different network topologies supported by the model. N is the number of nodes.

Topology	**Diameter** D	**Node degree**	**Routers**	**Total links** N_{links}	**Bisection Width** $W_{\text{bisection}}$
Ring	$\lfloor N/2 \rfloor + 2$	3	N	N	2
Mesh	$2\sqrt{N}$	5	N	$2(N - \sqrt{N})$	$\sqrt{N}$
B-Tree	$2\lceil \log_2(N) \rceil$	3	$N - 1$	$N - 2$	1
Q-Tree	$2\lceil \log_4(N) \rceil$	5	$\frac{N-1}{3}$	$\frac{N-1}{3} - 1$	2

Where $\bar{H}$ is the average hop count for a packet. This is calculated by calculating the hop count for every combination of two clusters and taking the weighted average based on the occurrence of that cluster combination in the connection distribution.

In practice, traffic is not always evenly divided over all links. Especially in the case of tree topologies, the links near the root node can have a relatively higher load when a lot of the package traverse from one side of the tree to the other. For this, the bisection width is used. The bisection width describes the minimum number of links that have to be cut to divide the network into two equal parts. An estimation of the fraction of the packets going through the bisection is $\Pr(H \geq \frac{D}{2})$, the probability that a packet has a hop count at least half the network diameter. This leads to the following equation for the packet rate through a bisection link:

$$R_{\text{bisection-link}} = R_{\text{link}} \frac{\Pr(H \geq \frac{D}{2})}{\frac{N_{\text{links}}}{W_{\text{bisection}}}} \tag{3.30}$$

The higher value of R_{link} and $R_{\text{bisection-link}}$ is used as the effective average link rate. This should always be lower than the clock frequency of the NoC.

3.6.3 Latency

The latency introduced by the NoC depends on the current state of the network. If the load is low, latency is lower than when the NoC congested. However, something can be said about the minimum latency. This merely depends on the number of hops a packet for synapse s must travel and the time spent in each router:

$$L_{\text{NoC}}(s) = H(s) \cdot T_{\text{hop}} \tag{3.31}$$

The number of hops can be calculated based on the used topology and T_{hop} depends on the router architecture.

3.7 Power and Area estimation

3.7.1 Neuro-synaptic array

Design of the neuro-synaptic array itself falls outside the scope of this thesis. However, its area is needed to estimate the length of the different links. Therefore, the following equation is used to estimate the area, assuming the area of individual neurons and synapses is known:

$$A_{\text{array}} = N_{sc} \cdot A_{\text{synapse}} + N_{nc} \cdot A_{\text{neuron}} \tag{3.32}$$

3.7.2 Lookup tables

As the various lookup tables in the clusters can get quite big, their area and power consumption should be taken into account. To get an estimation, the model from CACTI6.5 is used[31]. This tool gives area, power and delay estimations for SRAM caches. By configuring it for a cache with associativity of 1, results for an SRAM lookup table can be obtained.

Table 3.4: LUT power and area figures from CACTI6.5 for a 45 nm process.

Size [bits]	Static power [nW]	Energy per lookup [pJ]	Area [µm^2]	Access time [ns]
512	16	1.1	506	0.77
1024	31	1.3	781	0.90
2048	59	1.6	1303	0.92
4096	113	2.1	2311	0.98
8192	268	2.7	4032	1.0
16384	477	3.6	7522	1.2
32768	922	5.0	13819	1.2
65536	1726	7.3	26219	1.5
131072	3409	11.0	49862	1.5
262144	6456	18.7	94943	2.1

The model outputs as found in Table 3.4 are linearly interpolated to the actual sizes of the LUTs.

Combining the static power dissipation and the energy per lookup leads to the total power consumption for the lookup tables(P_{LUT}):

$$P_{\text{LUT}} = P_{\text{static}} + R_{\text{lookup}} \cdot E_{\text{lookup}} \tag{3.33}$$

R_{lookup} depends on the location of the lookup table. For LUTs in the source cluster (for Destination Addressing and Hybrid Addressing), R_{lookup} is R_{neurons} for step 1 of the lookup and R_{out} for step 2. For the lookup tables in the destination cluster (for Source Addressing and Hybrid Addressing), this rate is R_{in} for step 1 and $R_{\text{activation}}$ for step 2.

3.7.3 Local interconnect

The local interconnect is the combination of all data links within a cluster. To estimate the power and area of these links, the length should be known. The assumption is made that the local interconnect spans 2 sides of the neuro-synaptic array, giving $l_{\text{local-links}} = 2\sqrt{A_{\text{array}}}$.

Wire pitch and switching energy are taken from the CACTI6.5 model: $W_{\text{wire}} = 0.36\mu\text{m}$ and $E_{sw} = 0.4\text{fJ}\,\mu\text{m}^{-1}$. For the power consumption of the wiring, the data rate is also needed. As an estimation, the average of R_{out}, R_{local} and R_{in} is used as the data rate and the switching probability α is set at 0.5.

Combining, this leads to the following equations for the area and power consumption of the local interconnect in a cluster:

$$A_{\text{local-links}} = S \cdot l_{\text{local-links}} \cdot W_{\text{wire}} \tag{3.34}$$

$$P_{\text{local-links}} = \alpha \cdot S \cdot l_{\text{local-links}} \cdot E_{sw} \frac{R_{\text{out}} + R_{\text{local}} + R_{\text{in}}}{3} \tag{3.35}$$

S is taken from Section 3.4.3

3.7.4 NoC

For estimating the power consumption and area of the NoC, ORION is used[32]. This tool gives an estimation for the area and power consumption of NoC routers. It has many configuration options, e.g., setting the depth of different buffers, choosing technology size and setting the number of virtual channels per link.

A simple router design is used to minimise power and area consumption. So no virtual channels and only input buffers of depth 1 are used. The technology size is set to 45 nm. The flit width is based on the address sizes as discussed in Section 3.4.3 and the number of router ports is based on the node degree from Table 3.3. This leads to the power and energy numbers as found in Table 3.5.

The area is directly taken from this table, linearly interpolated if needed. For the power consumption the following equation is used:

$$P_{\text{router}} = P_s + R_{\text{link}} \cdot E_{\text{flit}} \tag{3.36}$$

For calculating the area and power consumption of the NoC links, the model from Section 3.7.3 is used. But now the length of the links is given by:

$$l_{\text{NoC-link}} = \sqrt{A_{\text{cluster}}} \tag{3.37}$$

3.8 Evaluation

The last step of the design space exploration is evaluating the model. To achieve this, most of the described model is implemented in an Excel spreadsheet. By using the built-in *data tables* functionality, the results for iterating over one or more parameters can be easily calculated and plotted. This allows for fast evaluation of the model.

Table 3.5: Power and area figures from ORION.

Degree	Flit width [bits]	A [µm^2]	P_s [µW]	E_{flit} [pJ/flit]
3	16	1006	32.5	272
	32	1409	41.7	385
	64	2215	60.3	611
5	16	2251	67.7	428
	32	3204	86.8	601
	64	5112	125	948

As a baseline for the evaluation the parameters from Table 3.6 are used. The baseline local connection distribution with $\lambda = 2$ has on average 87% of a neuron's connections within the same cluster and the other 13% to synapses in other clusters.

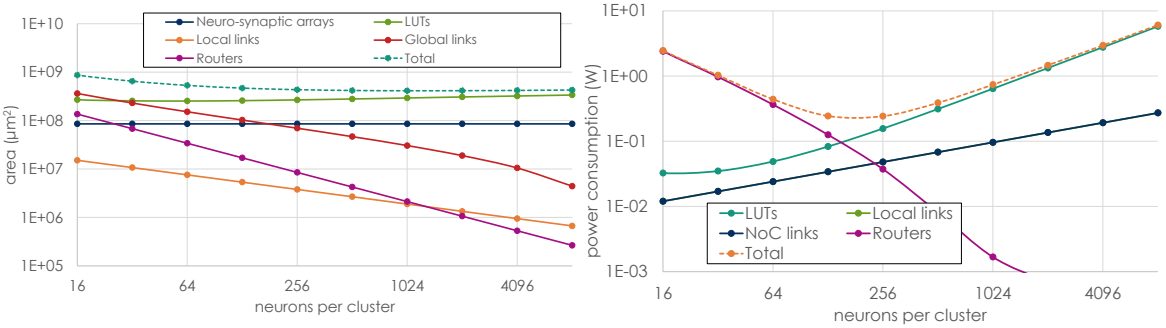
Table 3.6: Baseline evaluation parameters.

CD	Local ($\lambda = 2$)	AS	Hybrid
N_{neurons}	16,384	SES	Simple
F_{in}	512	NT	Mesh
F_{out}	512	A_{neuron}	100µm^2
R_{avg}	1 kHz	A_{synapse}	10µm^2
N_{clusters}	16	f_{clock}	100 MHz
N_{nc}	1024		

3.8.1 Clustering

First the effect of cluster sizes is evaluated. To achieve this, the number of neurons per cluster is varied while keeping the total number of neurons constant. Plots of the modelled area and power numbers can be found in Figure 3.15. First conclusion from this is that the total area of the LUTs and neuro-synaptic are mostly independent of the cluster size. That means that after a certain cluster size, the total area remains almost constant. In this case that is around 256 neurons per cluster.

Secondly, regarding the power, there seems to be an optimal cluster size. At a certain point, the decrease in power from having fewer routers, does not weigh up against the increased lookup table power consumption. This power consumption increases because the average size per LUT increases for bigger clusters. The bigger the memory for a LUT, the higher the energy cost for a single lookup. As the total lookups stays roughly the same, this leads to a higher overall power consumption.

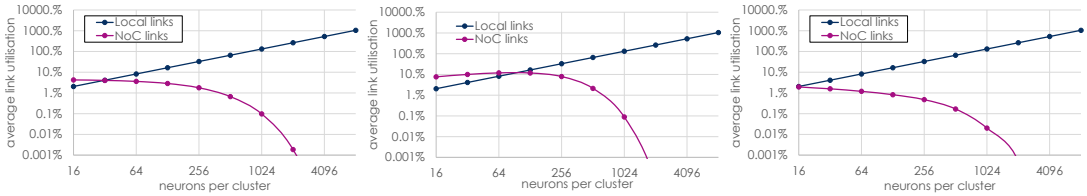


(a) Area (b) Power

Figure 3.15: Modelled area and power for varying cluster sizes.

3.8.2 Addressing

Figure 3.16 shows the effects on the local link utilisation for the different addressing schemes. The first takeaway is that for the smaller cluster sizes, Hybrid Addressing gives the lowest load on the NoC. As the sizes of the clusters increase, this difference diminishes because most connections are local. Furthermore, the local links seem to be the bottleneck for almost all cluster sizes. This stresses the Synapse Encoding methods to improve activation concurrency, as a higher concurrency leads to a fewer packets having to be sent over the local link.



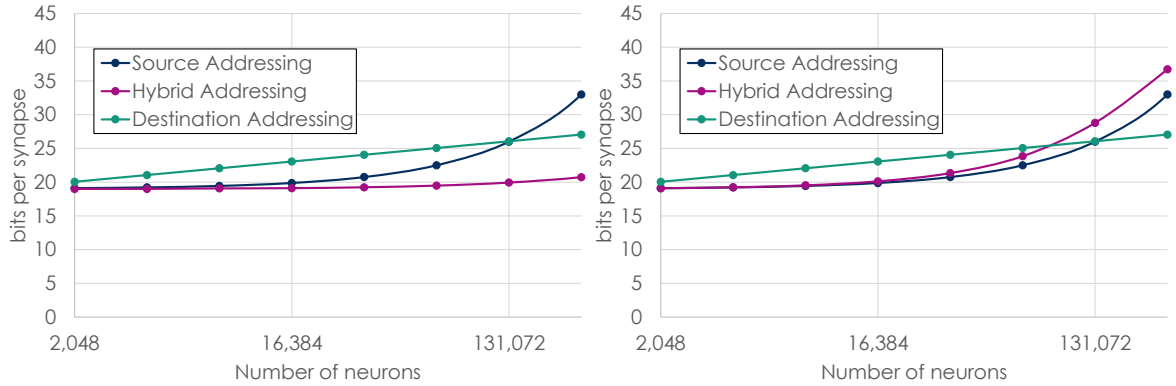
(a) Source Addressing (b) Destination Addressing (c) Hybrid Addressing

Figure 3.16: Utilisation of local and NoC links for different addressing schemes and varying cluster sizes.

Also the effect of the addressing schemes on memory size is investigated. This is done by plotting memory size per synapse for the different addressing for a varying number of neurons (Figure 3.17). The number of neurons per cluster is kept constant. For the local distribution, Hybrid Addressing is definitely the best option. For the uniform distribution, Source Addressing is best for $N_{\text{neurons}} < 131,072$. For bigger networks, Destination Addressing is better memory-wise.

It should be taken into account that the uniform connection distribution is a worst-case connectivity, as it has no structured pattern. Therefore the memory requirements from Figure 3.17b should be seen as a maximum for the number of neurons. In practice, any connectivity that is not completely random will have results closer to Figure 3.17a.

Previous points combined with the latency considerations from Section 3.8 make us consider Hybrid Addressing as an interesting addressing scheme for further evaluation in Chapter 4.



(a) Local connections

(b) Uniform connections

Figure 3.17: LUT size per synapse for the local and uniform connection distribution.

3.8.3 Synapse Encoding

Effects of Synapse encoding are more difficult to evaluate because of the heavy dependence on the η_{banking} and η_{grouping}, which are difficult to estimate. They will be further addressed in Chapter 4. There these values will be obtained by mapping SNNs on neuro-synaptic arrays. However, the previously found link utilisations and memory requirements show that there is a lot of potential for efficient synapse encoding to improve these.

3.9 Conclusion

In this chapter we made a design space exploration of neuro-synaptic spike-routing in neuromorphic systems. This has led to a model that can be used to estimate the consequences of several design parameters on memory size, area, power consumption, latency and link utilisation. Furthermore, the model has been evaluated for a variety of parameters. This has shown the necessity of dividing neurons into multiple clusters. Furthermore, it has shown that Hybrid Addressing is an interesting addressing scheme for further use. Finally, evaluation of Synapse Encoding has been left to the following chapters.

4 Simulation

The model from the previous chapter can only be partially used to estimate the performance of the spike-routing architecture. Mainly because it only takes into account connection distributions and spike rates, not actual connections and individual spikes. To get a better understanding of the performance of the architecture, a SystemC simulation was created.

SystemC is chosen because of its ability to simulate large systems at high speed. And because it is based on C, interfacing with other tools is relatively easy.

4.1 System Overview

The system that's simulated is based on the system that is described in Chapter 3. It uses Hybrid Addressing and all options for synapse encoding are implemented with configurable parameters. An overview of a single cluster is shown in Figure 4.1.

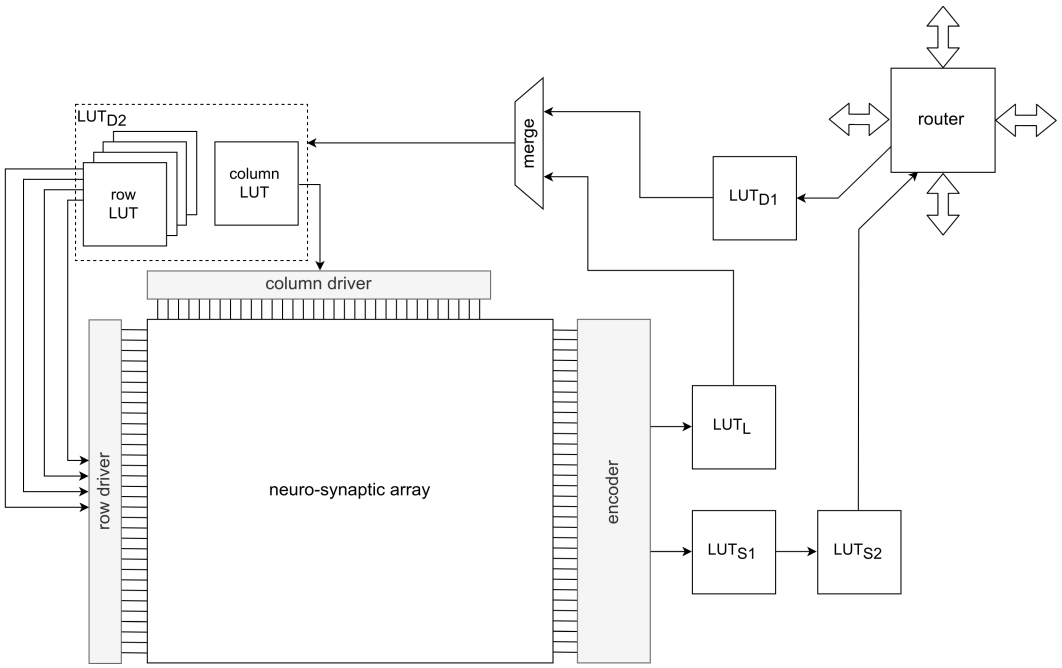


Figure 4.1: System overview for simulation

For the inter-cluster links, 32 bits are used. 8 of which are for the cluster address and the other 24 for the intermediate address. This means that the maximum number of clusters is 256 and every cluster can have 16M incoming axons.

Implementation of the NoC has been kept as simple as possible (Figure 4.2). As

a topology, the mesh grid is chosen. This allows for the use of dimension order routing, which is a simple and deadlock-free routing algorithm. The size of the router input buffers is configurable and set at 1 by default. Wormhole switching is used in combination with round robin arbitration.

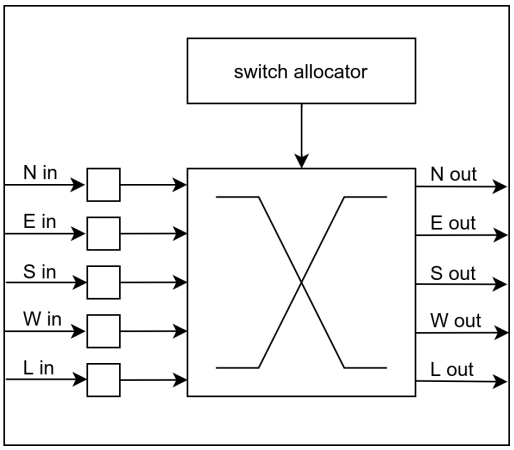


Figure 4.2: NoC router

4.2 Neuro-synaptic array

The neuro-synaptic array is considered a black box in the simulation system. There is no interaction between the synapses and the neurons. Neural spikes are injected from a file source and all synaptic activations are logged to a file. How it is implemented can be seen in Figure 4.3. The input of the neuro-synaptic array consists of $N_{\text{rows}} + N_{\text{cols}}$ wires.

The output of the neuro-synaptic array consists of N_{rows} wires. These correspond to the neuronal activations and serve as input to the simulation. The neuronal activations do no depend on the synaptic activations but are loaded from a file of input spikes (see Section 4.3.4). The Spike Loader loads this file and passes these spikes one by one to the Neuron Driver, which activates the wires corresponding to the neuron that spikes. A fifo in front of the Neuron Driver makes sure only 1 output wire is active simultaneously. Only after a spike has been acknowledged, will the next spike be popped from the fifo.

4.3 Simulation Flow

Running the simulation consists of multiple steps, as shown in Figure 4.4. All those steps will be discussed one by one in this section.

Throughout the flow, multiple files are output. Most of them are in JSON-format. This is not a very size efficient file format, but it has the benefit that is very well readable by the human eye. This makes it a lot easier to inspect the results of different steps in the pipeline. Furthermore, it allows an easy interface between Python and SystemC components, as both have libraries for reading those files. Because JSON-

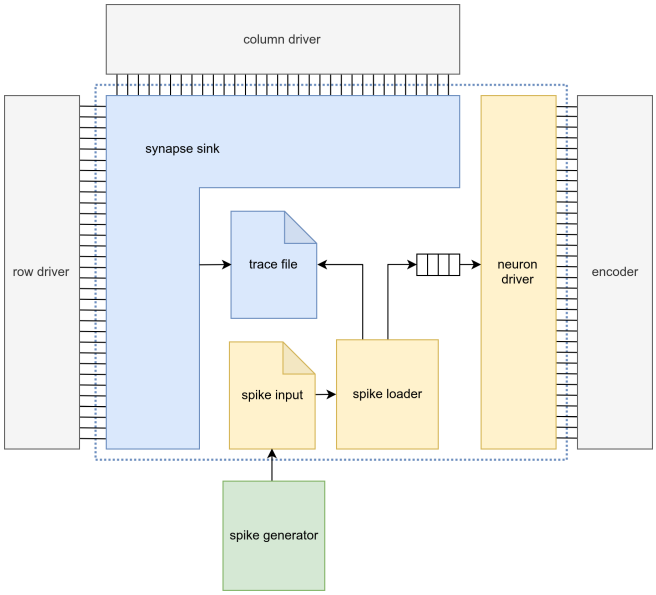


Figure 4.3: Implementation of the neuro-synaptic array in the simulation.

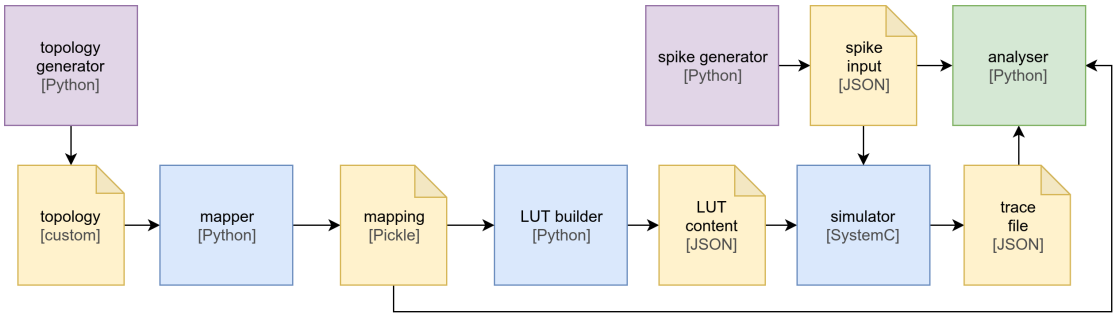


Figure 4.4: Simulation flow

files contain a lot of repeated content, the file size can be reduced by using gzip when necessary.

4.3.1 Topology Generation

To create different SNN for the simulation, different topology generators have been created. These scripts are implemented in Python and generate an SNN topology based on different input parameters. These are the three generators that were implemented:

Uniform($N_{\mathbf{neurons}}$, $F_{\mathbf{in}}$) creates a network of N_{neurons} neurons where each neuron has F_{in} incoming connections coming from a random selection of neurons.

Layered($N_{\mathbf{neurons}}$, $F_{\mathbf{in}}$, n) creates a network of N_{neurons} divided over n layers. Every neuron in layer i has F_{in} incoming connections coming from the layer $i-1$. Neurons in layer 1 have no incoming connections.

Local($F_{\mathbf{in}}$, λ) creates a network of N_{neurons} neurons where each neuron has F_{in} incom-

ing connections. These connections follow the local distribution as described in Section 3.3.1.

All neurons are assigned an incremental number as identifier. The output of the generator is a list with all neurons listed on a separate line. Behind the neuron is a space separated list of all neurons it has a connection to. This would be the output for the topology from Figure 4.5:

Topology generator output sample

```
0 1 2 3
1 3
2 3
3
```

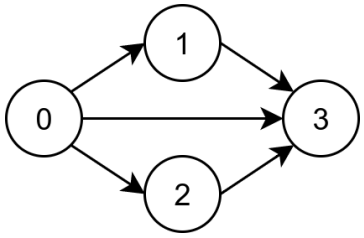


Figure 4.5: Example of a very simple SNN topology

4.3.2 Mapping

This topology file can then be used in the next step to map the topology to the neurosynaptic arrays. This step consists of two steps: assigning neurons to clusters and mapping neurons and synapses to an array.

4.3.2.1 Clustering

For assigning the neurons over N_{clusters} different clusters, two different methods are used.

The first method just takes the neurons in the order as declared in the input file and divides them into clusters of size N_{nc}. This means the first N_{nc} are clustered together and the second N_{nc}, etcetera. Especially for the layered and local topologies, this can work well because neurons with many connections between them are also near each other in the topology file.

The second method makes use of graph partitioning to divide the neurons into clusters. By using the adjacency-matrix of all connections as input to a graph partitioning algorithm the number of inter-cluster connections is minimised. The algorithm objective is set to N_{clusters} partitions. For implementation, the Python wrapper for the METIS algorithm is used[33].

4.3.2.2 Array Mapping

After all neurons have been assigned to a cluster, they have to be assigned to one of the hardware neurons in an array. Furthermore, each connection has to be assigned to a hardware implementation of a synapse. Symbolically this can be seen as mapping a connection $c_{k,i}$ (from neuron k to neuron i) to a synapse $s_{i,j}$ (see Figure 4.6).

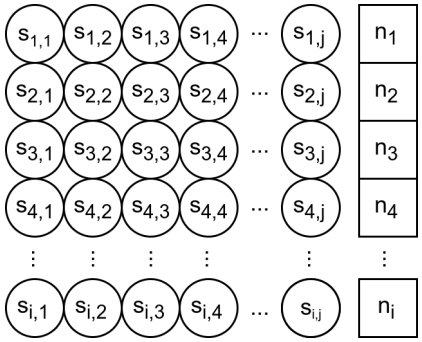


Figure 4.6: Numbering of neurons and synapses in neuro-synaptic array

The importance of this step depends on the synapse encoding scheme that is used. For example, if the simple encoding scheme from Section 3.5 is used, the mapping of the neurons does not matter and for the synapses, it is only important that the synapse is in the row of the correct neuron. However, if a more complex encoding scheme is used, the placement does become of importance as it directly influences η_{banking} and η_{grouping}, which influences the memory efficiency and the number of hardware synapses that are needed (or the inverse: the number of connections that can be mapped to a single neuro-synaptic array).

First, an ordered list of all connections is made. Initial ordering is performed by the id of the pre-synaptic neuron. Then all connections are iterated over one by one. The first connection originating from a pre-synaptic neuron is always mapped to the leftmost free synapse in the correct row. So if $s_{i,1}$ is already used, $c_{k,i}$ will be mapped to $s_{i,2}$. For all consequent connections originating from the same pre-synaptic neuron will first be tried to map to synapses in columns that already used for connections originating from the same pre-synaptic neuron. This is to maximise η_{banking} and η_{grouping}.

Because the mapping is stored in the state of many Python objects, the easiest way to save it to a file is by using the Python Pickle functionality. This allows saving of arbitrary Python objects in a custom binary format.

4.3.2.3 Mapping Optimisation

The above process leads a mapping, but not necessarily to an efficient mapping. In an effort to improve efficiency, an optimisation process is added. This process is a search for the optimal order of neurons in the array and the optimal order in which the connections are iterated. For both, the optimisation is performed by using Simulated Annealing.

Simulated Annealing is a technique for iteratively converging towards an optimum.

It consists of three looped steps: changing the state, evaluating state energy and accepting or reverting the change.

In the first step, a swap is performed. In case the neuron order is optimised, two neurons (a and b) are swapped in position, so neuron b is placed in row a and neuron a is placed in row b. When connection order is optimised, all connections of two pre-synaptic neurons are swapped in the ordered list of connections.

Secondly, the state energy is evaluated. This means that a mapping is performed with the procedure from Section 4.3.2.2. The energy of this mapping is expressed as $E = \eta_{\text{banking}} \cdot \eta_{\text{grouping}}$. So the lower the energy, the more efficient the mapping.

Finally, the difference in the energy between the new and old state is calculated. Based on this difference, it is decided whether the change is accepted or reverted. If the new energy is lower, the change is directly accepted. If it is higher, the chance of the change being accepted is given by:

$$p_{\text{accept}} = e^{-\frac{k \cdot \Delta E}{T}} \tag{4.1}$$

Here T is the current temperature; a value starting at T_0 that is decremented by one every n iterations until it reaches 0 (then the optimisation is over). It is found that a $T_0 = 30$, $n = N_{nc}$ and $k = 10^4$ leads to reliable improvement of the mapping performance.

4.3.3 LUT Building

The next step in the pipeline is building content for the LUTs. This is done from destination to source, so first the content of D2 and D1 and next S1, S2 and L.

First, a list of all row sets in a cluster C that belong to a pre-synaptic neuron is created. A maximum of g of these sets in the same column are then combined into an activation group. Every activation group gets 1 entry in LUT D2. All entries in D2 corresponding to the same pre-synaptic neuron are placed consecutively. Next, for every presynaptic neuron, an entry is added to D1 or L, with an offset referring to the first entry in D2 and the length being the number of entries in D2. It is added to L if the pre-synaptic neuron is in the same cluster. The address in L being the address of the neuron.

If the pre-synaptic neuron is not in the same cluster, the entry is added to D1 and the address in D1 is then used at the Intermediate Address. This Intermediate Address in combination with the coordinates of cluster C is then added to S2 of the cluster of the pre-synaptic neuron. There can be multiple consecutive entries for different destination clusters in S2 for the same pre-synaptic neuron. For all of these together, one entry is added to S1 with the offset corresponding to the first entry in S2 and the length corresponding to the number of entries in S2. The address of this entry in S1 corresponds to the address of the pre-synaptic neuron.

All these LUT entries are combined into a single JSON file. Below is a sample overview of the LUT configuration file. For a full overview see Appendix A.

LUT content configuration file

```
{
  "0": { // cluster number
  "L": [...],  // content for LUT L
  "S1": [...], // content for LUT S1
  "S2": [...], // content for LUT S2
  "D1": [...], // content for LUT D1
  "D2": [...]  // content for LUT D2
},
  "1": {...}
}
```

After the content of all LUTs is generated, the size of all lookup tables is calculated by multiplying the number of entries by the width of their entries in bits.

4.3.4 Spike Generation

As an input for spike data the simulation takes in a JSON-formatted file with the following format:

Spike input file sample

```
{
  "10":[ // time in ns
    {
      "neuron":5,
      "x": 0,
      "y": 1
    },
  ...
  ],
  "50":[ // time in ns
    {
      "neuron": 2,
      "x": 0,
      "y": 0
    },
    ...
  ],
  ...
}
```

The example above triggers a spike in neuron 5 of the cluster at location (0,1) at 10ns and one in neuron 2 of the cluster at location (0,0) at 50ns.

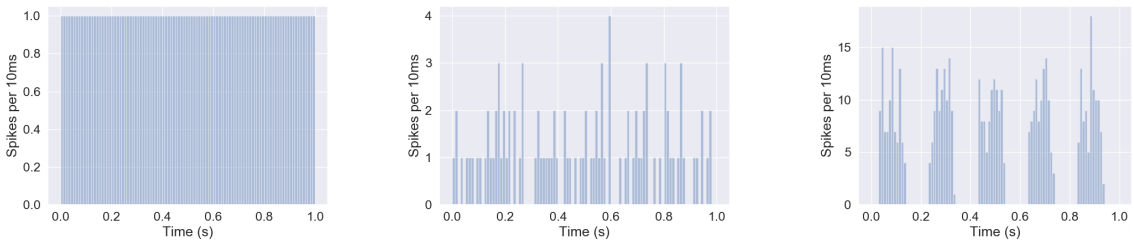
These input files can be obtained in two ways. The first way is by running an existing SNN simulation and extracting all spike events. This can be used to easily test the performance of an existing SNN.

Secondly, one of the spike generators can be used. There are three of these that are implemented in Python:

Constant Rate(R_{avg}) This generator has a R_{avg} as parameter and generates a spike for every neuron every $\frac{1}{R_{avg}}$s with a random offset for the first spike (Figure 4.7a).

Poisson(R_{avg}) This generator has a R_{avg} as parameter and creates a poisson distribution of spikes for every neuron (Figure 4.7b).

Burst(G_1, G_2, α) This generator takes two other generators and an activation fraction (α) as inputs. It convolutes the spikes of G_1 with G_2 for a fraction α of the time. For example, if G_1 is a Constant Rate 5Hz generator, G_2 is a Constant Rate 1000Hz generator and $\alpha = 0.5$, the Burst generator will generate poisson distributed spikes at 1000Hz every 0.2s for 0.1s (Figure 4.7c).



(a) Constant Rate (b) Poisson (c) Burst

Figure 4.7: Spikes as generated for a single neuron for different spike generators. Y-axis represents the number of spikes per 10ms. For a and b R_{avg} =100Hz. For c G_1 is 5Hz Constant Rate generator, G_2 is a 1000Hz Poisson generator and $\alpha = 0.5$.

For all these generators a refractory period (T_{refr}) can be set. A generator does not generate two spikes for the same neuron within the refractory period. If this were to happen, the second spike is moved to the moment when the refractory period has passed.

4.3.5 Simulator

As stated before, the simulator is a tool implemented in SystemC. It takes in a spike input file and the LUT configuration file. It runs for the specified time and outputs a trace file.

This trace file contains a list of all inputted spikes and all synaptic activations. Because all inputted spikes are tagged with a unique identifier and this identifier is copied to all packets that originate from this spike, the input spike can be linked to the synaptic activations it caused. All events are accompanied by the time in nanoseconds when they occurred. This leads to the following format for the trace file:

Trace file sample

```
[
  {
    "src": {
      "cluster_id": 0,
      "neuron": 75,
      "time": 930.0
    },
    "dsts": [
      {
        "cluster_id": 0,
        "neuron": 1,
        "time": 1000.0
      },
      {
        "cluster_id": 5,
        "neuron": 9,
        "time": 1210.0
      },
      ...
    ]
  },
  ...
]
```

4.3.6 Analysis

The last step in the pipeline is analysis. This step, first of all, serves to check whether the trace file is a possible output given a mapping and spike input. It verifies whether all correct synapses were activated.

Furthermore, it also computes the latency for every synaptic activation from the input of the spike to the moment of the synaptic activation. Just as the latency jitter, which is the difference between the latency and the minimum recorded latency for that connection. With the total number of synaptic activations per cluster and the simulation time, the synaptic activation rate is calculated.

4.4 Summary

In this chapter a simulation toolbox for evaluating the performance of the proposed spike-routing architecture was described. Also support tools for the simulation flow were discussed, including a tool for mapping SNN topologies onto neuro-synaptic arrays. In the next chapter these tools will be used to evaluate the performance of the proposed spike-routing architecture.

5 Performance Evaluation

In this chapter, the performance of the architecture will be evaluated. The focus will be on an architecture of 3×3 clusters with 128 neurons with 128 synapses each. The reason for choosing these dimensions is that it provides a good balance between seeing the effects of clustering, having reasonable size neuro-synaptic arrays and being able to actually run simulations on it.

Three different SNN topologies are used throughout the first part of this chapter. All of them consist of 1152 neurons with 128 synapses each. The first one has a uniform connection distribution. The second has a local connection distribution with $\lambda = 2$. The last topology is a layered topology with 5 layers.

The chapter is structured as follows. First, the mapping tool from Chapter 4 will be evaluated and an optimal configuration for the architecture will be derived. Then, the latency and throughput will be tested for the above-mentioned topologies. Finally, the architecture will be evaluated for a custom SNN topology for handwritten digit recognition.

5.1 Mapping

In this section, the performance of the mapping algorithm for varying parameters is evaluated to come to an optimal set of parameters.

To compare the performance of different mappings, the following figure of merit is used:

$$\text{FoM} = \frac{\text{concurrency} \times \text{mapping efficiency}}{\text{bits per connection}} \tag{5.1}$$

With the following definition for the number of bits per connection:

$$\text{bits per connection} = \frac{\text{memory size}}{\text{number of synapses} \times \text{mapping efficiency}} \tag{5.2}$$

The basis of this FoM is that high concurrency and few bits per synapse should improve the value of the FoM. However, if only these two factors are incorporated an optimum will be found at a mapping configuration with a very low resulting mapping efficiency (fraction of the synapses that is actually in use). Consequently, neuro-synaptic arrays have to be a lot bigger, leading to a bigger overall system. To address this, the mapping efficiency is also incorporated in the FoM. This is defined as:

$$\text{mapping efficiency} = \frac{\text{maximum number of connections}}{\text{number of synapses}} \tag{5.3}$$

So if on an array of 16,384 synapses only 14,000 connections can be mapped, the mapping efficiency is 0.85.

In the end, the found configuration will be compared with the baseline configuration. This is the configuration that only uses simple synapse encoding i.e., $B = 1$, $g = 1$ and $B_{\text{offset}} = 0$.

5.1.1 Clustering method

First, the effect of using graph partitioning for clustering neurons is analysed. To achieve this, the difference in FoM for a variety of mapping configurations is tested for three different SNN topologies. As can be seen in Table 5.1, the graph partitioning worsens the FoM if the order from the generated topology is used. An explanation for this can be found in the way SNN topologies are generated. The generation method already results in neurons that have many connections being in the same cluster.

This is further substantiated by running the same test again, but shuffling all neurons. Graph partitioning does improve mapping performance in this case.

Nevertheless, graph partitioning will not be used for any further tests because it does not improve performance for the used topology generators.

Table 5.1: Effect of clustering on FoM for generated or randomized neuron .

Topology	Generated order	Randomized order
Uniform	0%	+1%
Local	-14%	+229%
Layered	-3%	+228%

With the clustering method known, it can also be determined to how many clusters a neuron is connected on average. This can be found in Table 5.2. The first column show what fraction of neurons has a connection to at least one neuron in the same cluster. This means that every neuron in the uniform and local connection distribution has at least one connection to the cluster in the same cluster. For the layered topology, this is only 44%.

Table 5.2: Average number of clusters a neuron is connected to.

Topology	Self	Other clusters
Uniform	100%	8
Local	100%	3.3
Layered	44%	2.5

5.1.2 Mapping optimisation

After running many mapping optimisations for different topologies, it is found that the effectiveness of the optimisation depends on the SNN topology used, as can be seen in

Table 5.3. For all three topologies, optimising the connection order improves the FoM. For the uniform and layered topology, optimising the neuron order also improves the FoM. However, for the latter combining both methods is only marginally better than optimising for the connection order.

Because the optimisation process is a compute-intensive process, only for the uniform topology both optimisation methods will be used. For the other topologies, only connection order optimisation is applied.

Table 5.3: Effect of mapping optimisation methods on FoM.

Topology	Neuron Order	Connection Order	Both
Uniform	+8%	+15%	+20%
Local	0%	+25%	+22%
Layered	+9%	+81%	+83%

5.1.3 Banking and Grouping

Next is investigating the effects of different values of B and g on the mapping performance. This is done by mapping different topologies on a 16-cluster system with 128 neurons per cluster and 128 synapses per neuron. Performance is evaluated by looking at the mapping efficiency, the concurrency factor and the average number of bits needed per connection. For B, a test range of 1-8 has been chosen because higher values lead to more complex hardware implementation. Furthermore, $B \cdot g$ is kept below 128, because higher values are guaranteed less efficient. The column address offset is kept at 0 for now. The results can be found in Figure 5.1.

From the results, it follows that increasing B increases the concurrency for all values of g. This is unsurprising, as the concurrency will increase as long as $\eta_{\text{banking}} > \frac{1}{B}$. Since $\frac{1}{B}$ is also the minimum value for η_{banking}, the concurrency can only increase or remain equal.

As for the number of bits per synapse, this is lowest for $B = 2$ and $B = 4$. The reason for this is that first adding more banks reduces the number of LUT bits because of Column Reuse. However, as the number of banks increases, not all banks are used every cycle, i.e., η_{banking} decreases, increasing the total number of LUT entries.

The relation between g and concurrency is also positive. Here the same holds as for B: concurrency can either increase or remain equal for larger row groups. However, as g increases, this comes at an increased memory cost. This can be explained by looking at the rightmost part of 3.25: $\frac{g}{\eta_{\text{grouping}} \cdot g} = \frac{1}{\eta_{\text{grouping}}}$. This part becomes big when η_{grouping} decreases. Which is exactly what happens for high values of g.

Overall, the best FoM is achieved for $B = 4$ and $g = 8$. Therefore this configuration has been chosen for further evaluation.

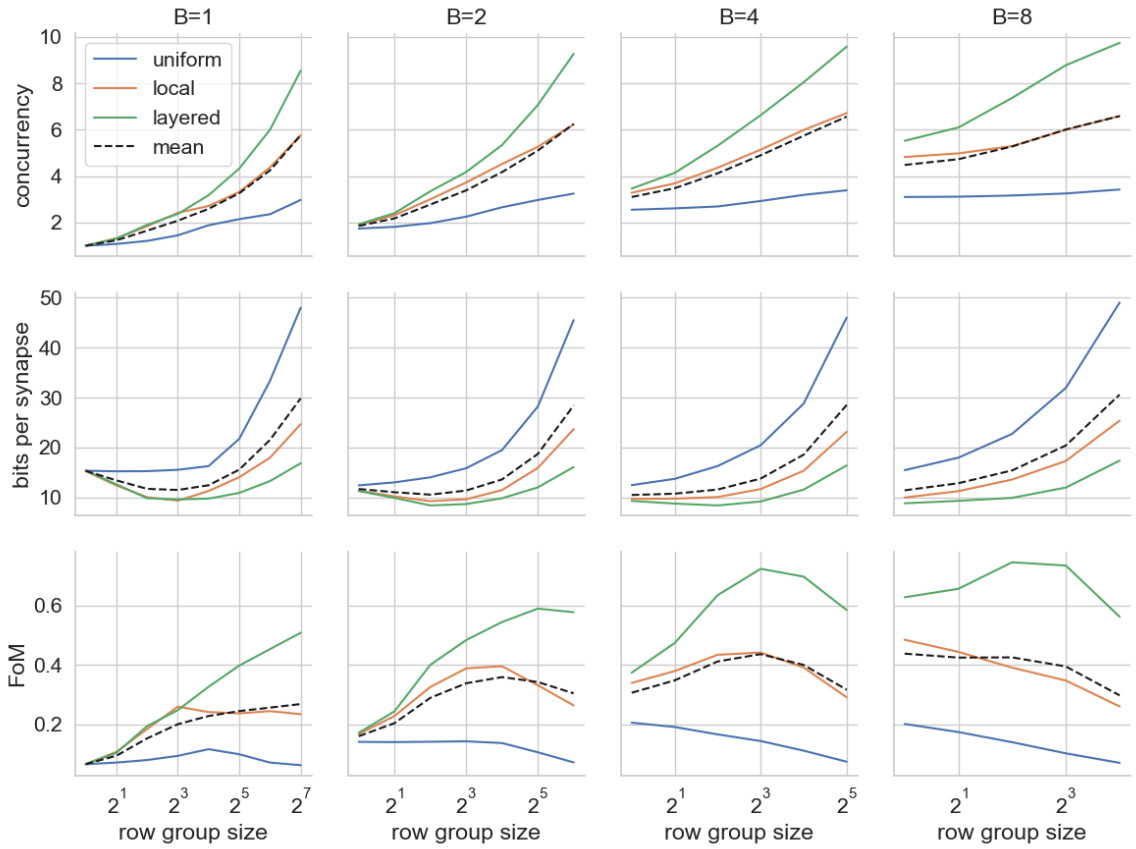


Figure 5.1: Mapping performance for varying row group size and banking factors.

5.1.4 Column Offset

Finally, the effect of the column offset is investigated. The number of column offset bits has no relation to the concurrency. Therefore only the effect on the mapping efficiency and memory consumption is considered. The same test as in Section 5.1.3 is run, but now the column offset is ranged from 0 to 7 bits. The can be found in Figure 5.2.

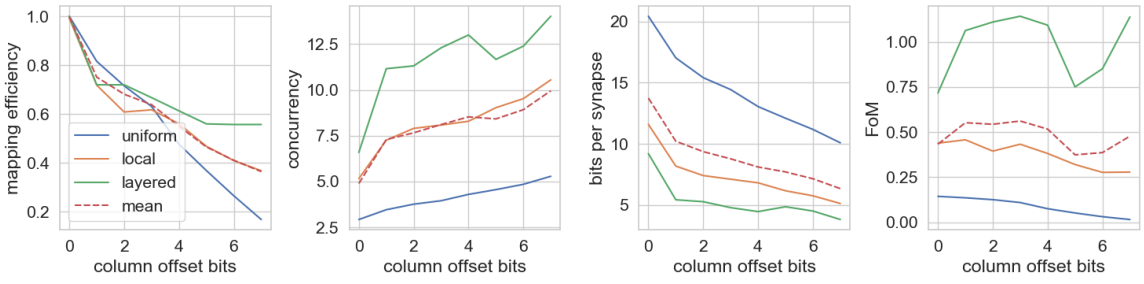


Figure 5.2: Effect of changing column offset.

Increasing the column offset decreases the mapping efficiency and the number of

bits per synapse. However, it also increases concurrency. Reason for this is that a bigger column offset means that there are fewer columns the connections of a presynaptic neuron can be mapped to. This in its turn means that if during mapping a column has a few synapses left, these cannot necessarily be used for the connections of any presynaptic neuron. This leads to some unmapped synapses, hence the decreases mapping efficiency. But it also leads to more connections of the same presynaptic neuron being mapped to the same column, increasing banking and grouping efficiency and thereby the concurrency.

The mean FoM for the different topologies is highest for a column offset of 1 bit. Therefore this value will be used in the following sections.

5.1.5 Summary

Combining these parameters leads to the results for mapping the different topologies onto the architecture as can be found in Table 5.4. The results are compared with the baseline configuration.

Table 5.4: Mapping performance for the chosen parameters of the architecture.

Topology	Mapping efficiency	Concurrency	Bits per synapse	FoM	Improvement
Baseline	100%	1.0	15.2	0.066	-
Uniform	82%	3.5	17.0	0.14	×2.13
Local	72%	7.3	8.2	0.46	×6.99
Layered	72%	11.2	5.4	1.1	×16.72

5.2 Simulation

In this section, the performance of the architecture with the previously chosen parameters will be evaluated by simulating with a variety of spike inputs. The default setup for these simulations can be found in Table 5.5. First, the maximum spike input rate and synaptic activation rates are determined. Next, the effect of varying the input rate and finally the effect of burst traffic is evaluated.

In all cases the system is simulated with a clock cycle time of 10 ns. Unless stated otherwise, the spike input is Poisson distributed and the neuron refractory period is half of the average time between spikes.

5.2.1 Saturation point

To evaluate the maximum throughput of the system, the system is saturated with a spike rate of 300 kHz and no refractory period. The resulting synaptic activations rate is treated as the maximum the system can handle. Divided by the total number of synapses, this gives the maximum spike frequency per neuron. Full test setup parameters can be found in Table 5.6. The results can be found in Table 5.7.

Table 5.5: Default simulation setup

N_{clusters}	9
N_{nc}	128
F_{in}	128
B	4
g	8
B_{offset}	1

Table 5.6: Saturation test setup.

Spike distribution	Poisson(300 kHz)
T_{refr}	0s
f_{clock}	100 MHz

Table 5.7: Maximum spike input and output.

Topology	Spike input	Synaptic activations per cluster
uniform	13.9 kHz	228M act/s
local	27.6 kHz	453M act/s
layered	38.5 kHz	631M act/s

Next, the effect of the number of neurons and the number of synapses per neuron is determined. So the same setup is taken, but N_{nc} and F_{in} are varied. Maximum rates for different topologies are averaged. This leads to the results found in Figure 5.3. For this setup, an increase in the number of synapses leads to a higher saturation point. Furthermore, the saturation point is a little lower for 256 neurons per cluster. An explanation can be found in that the architecture parameters are optimised for 128 and that increasing the number of neurons decreases η_{grouping} and hence the synaptic activation concurrency.

5.2.2 Latency

To get spike latency during regular operation, the system is tested with a Poisson spike input of 1 kHz with a minimum inter-spike distance of 500 µs (Table 5.8). In Figures 5.4, 5.5 and 5.6 the results of these simulations can be found. On the left are violin plots of the distribution of latency for spikes separated by the number of hops used on the NoC. On the right side, the latency jitter for spikes is plotted, which is calculated as the difference between the minimum latency for a spike of a certain connection and the actual latency of a spike.

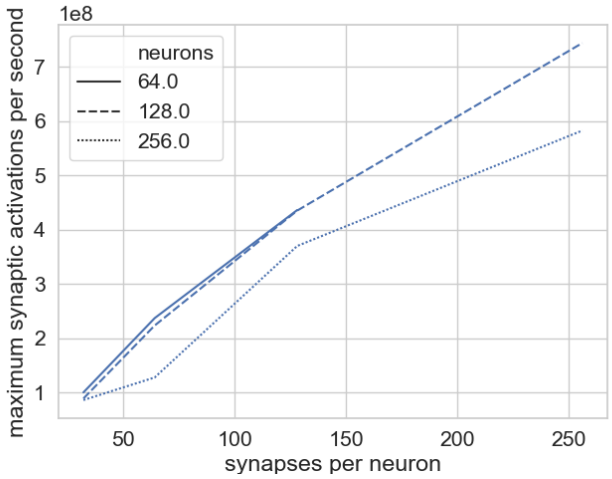
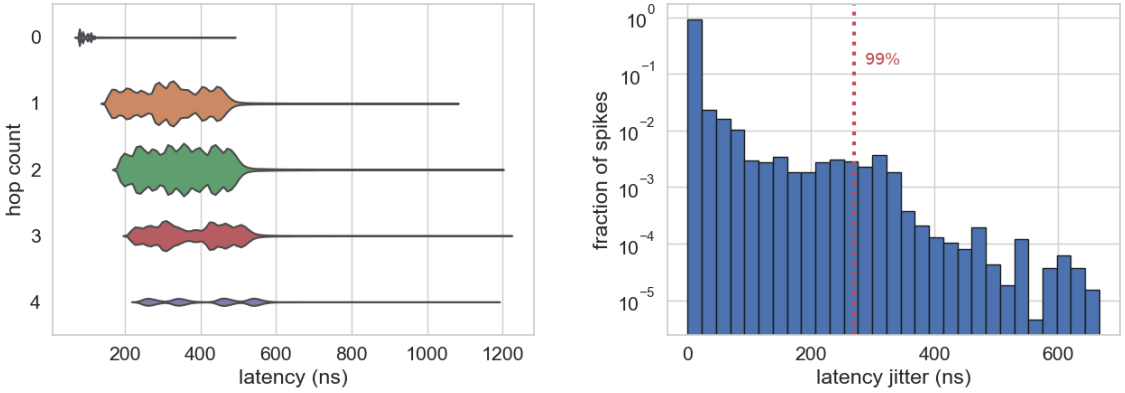


Figure 5.3: Plot maximum synaptic activation rates for varying neuro-synaptic array sizes.

Table 5.8: Latency test setup

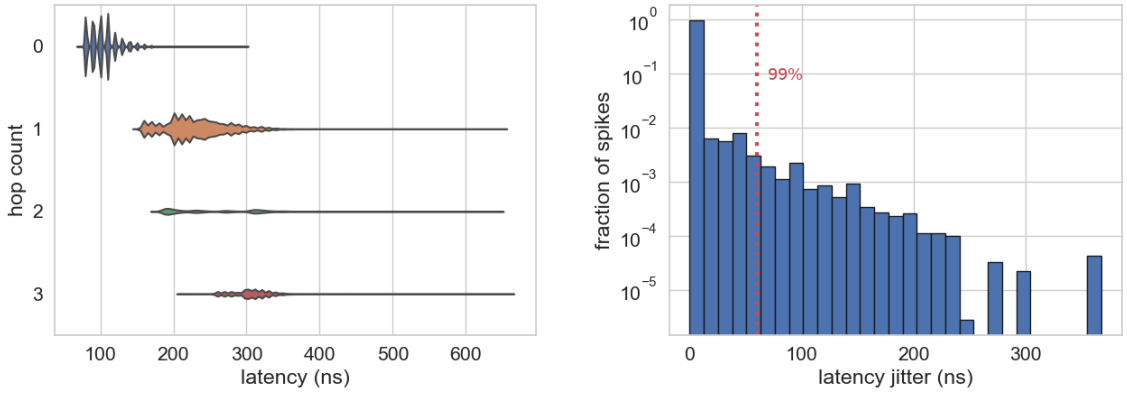
Spike distribution	Poisson(1 kHz)
T_{refr}	500 µs
f_{clock}	100 MHz

Latency and latency jitter are highest for the uniform connection distribution. The first is a direct consequence of the lower concurrency for this topology and the lower synaptic activation rate. This means more cycles are needed to activate all correct synapses, hence the longer latency. The higher latency jitter can be explained by the high number of clusters neurons have connections to (Table 5.2). This means



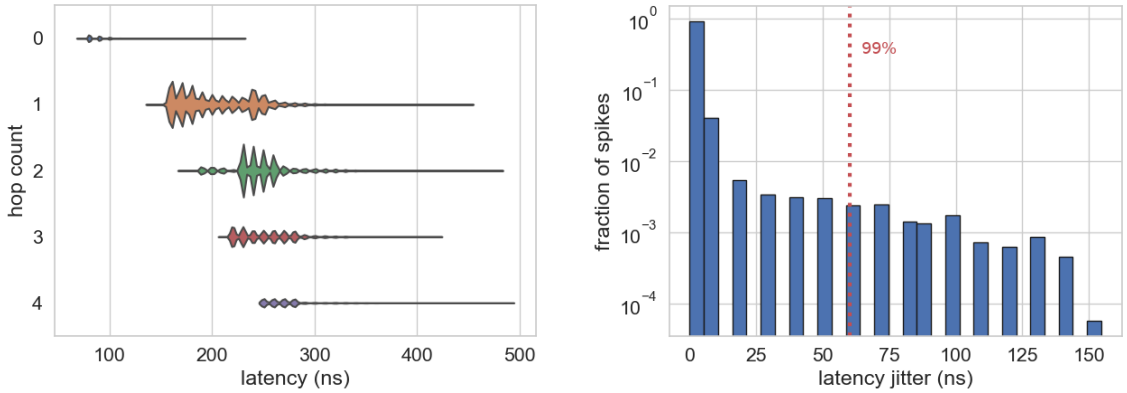
(a) (b)

Figure 5.4: Latency and latency jitter for uniform connection distribution with a poisson spike input of 1 kHz.



(a) (b)

Figure 5.5: Latency and latency jitter for local connection distribution with a poisson spike input of 1 kHz.



(a) (b)

Figure 5.6: Latency and latency jitter for layered connection distribution with a poisson spike input of 1 kHz.

more traffic on the NoC and therefore a higher probability of contention of one on the components on a spike's route which causes additional delay and therefore latency jitter.

For the local and layered connection distribution, the results are comparable. The main difference is a higher maximum latency jitter for the local distribution. An explanation for this can probably be found in the higher number of clusters a neuron has connections to and lower concurrency in comparison to the layered connection distribution. This increases the chance of contention and therefore the latency jitter.

Nevertheless, the latency jitter is low for all cases. Even for the uniform connection distribution, 99% of all spikes arrive within 280ns of the minimum latency, which is orders of magnitude lower than the average time between two spikes.

Also, the effect of varying the neuro-synaptic array configurations on the latency

jitter is tested. The setup from Table 5.8 is again used, but now the number of neurons per clusters and the number of synapses per neuron is varied. This leads to the result of Figure 5.7.

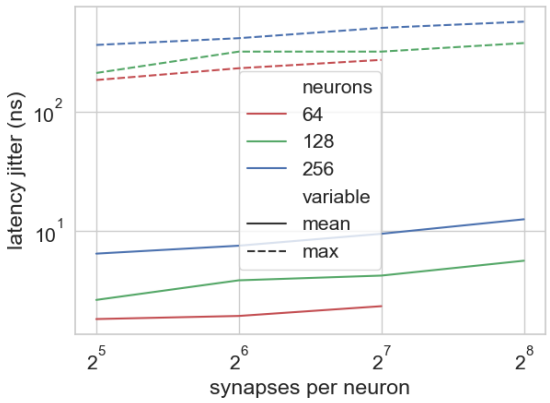


Figure 5.7: Plot of spike latency jitter for varying neuro-synaptic array configurations.

5.2.3 Spike rate

For evaluating the influence of spike input rate, the latency jitter for varying spike rates is plotted in Figure 5.8. See Table 5.9 for the test parameters. Spike rates vary from 100Hz to 38.5 kHz but are always kept below the maximum spike rate from Table 5.7. The refractory period is always half of the average time between two spikes.

Table 5.9: Spike rate test setup

Spike distribution	Poisson(R_{avg})
R_{avg}	100Hz - 38.5kHz
T_{refr}	$\frac{1}{2\,R_{avg}}$
f_{clock}	100 MHz

As the input spike rate increases, both the mean and maximum latency jitter increase. The reason for this is that contention in the system increases as the traffic increases. Therefore the probability that a spike has to wait for another packet somewhere on its path from neuron to synapse increases and so does the latency jitter.

5.2.4 Spike bursts

Next, the effect of the burstiness of the input spikes is evaluated. An average spike rate of 1 kHz is applied to a system with local distributed connections. A burst spike pattern from 4.3.4 is used to model the burstiness of the traffic. By decreasing α, spikes of a neuron are more focussed in a small time window. Test setup details can be found in Table 5.10. Results can be found in Table 5.11.

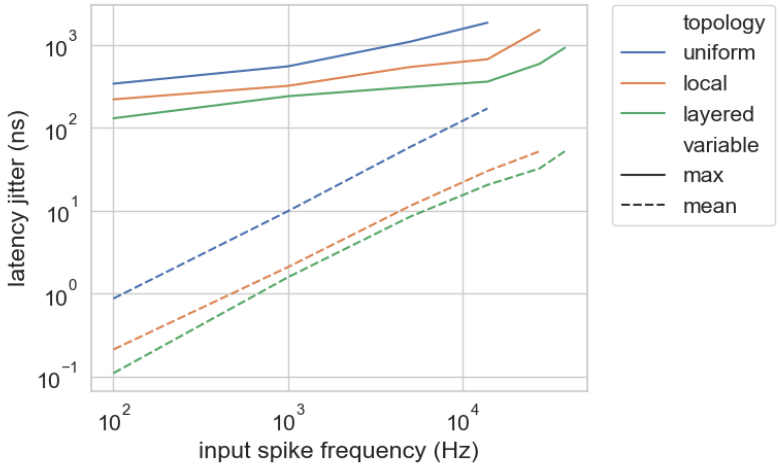


Figure 5.8: Plot of spike latency jitter for varying poisson distributed spike inputs.

Table 5.10: Spike bursts test setup

Spike distribution	Burst(Constant(100Hz), Poisson($\frac{1000Hz}{\alpha}$), α)
Burst fraction (α)	0.01 - 1
T_{refr}	$\frac{500\mu s}{\alpha}$
f_{clock}	100 MHz

Table 5.11: Latency jitter for varying burstiness of spike input with an average rate of 1 kHz.

Burst fraction	Mean latency jitter	Max latency jitter
100%	1.39 ns	260 ns
50%	1.97 ns	300 ns
10%	2.08 ns	330 ns
1%	3.05 ns	340 ns

Latency jitter increases as the burstiness of the spike input increases. This can be explained by the increased spike rate when a neuron is bursting. When two neurons are bursting simultaneously, the chance that they both produce a spike in a small time frame is higher than when the spike rate is constant. Therefore the chance of contention is higher.

5.3 Application

Finally, the performance of the spike-routing architecture is verified by using it for a real-world application. The application under consideration is a handwritten digit

recognition system. It can be used to classify the digits of the MNIST handwritten digit database[34]. This is a set of 70,000 28×28 pixel images of handwritten digits.

5.3.1 Setup

The used SNN is a 4-layer network, see Figure 5.9. The input layer consists of 784 neurons which all correspond to a single input image pixel. Every neuron in the first layer is randomly connected to 10 different neurons in the second layer, which consists of 100 neurons. These neurons are fully connected to the third layer of 50 neurons. This layer is fully connected to the output layer of 10 neurons. These 10 neurons all represent one digit.

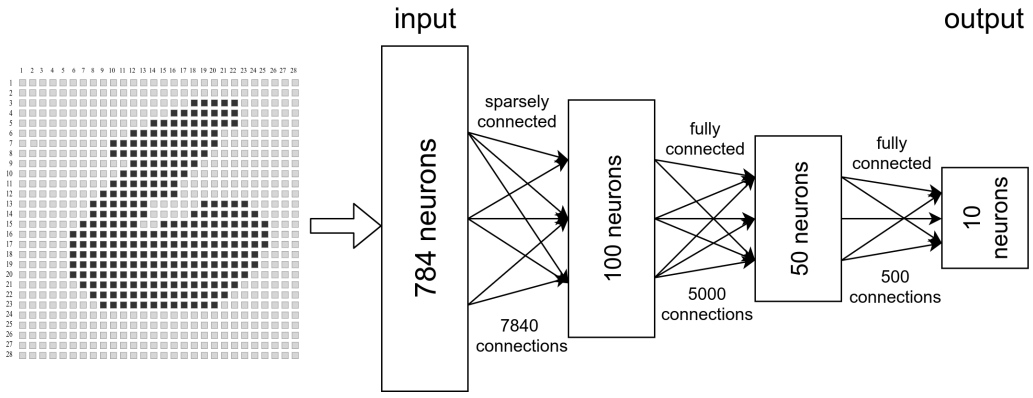


Figure 5.9: SNN topology for handwritten digit recognition.

Because only 160 of the SNN's neurons have incoming connections and therefore need synapses, the topology can fit 7.2 times in the architecture from Section 5.1 with 1152 neurons. So the SNN is replicated 6 times to mimic a fully utilised system. This leads to the following

Table 5.12: Handwriting SNN mapping results

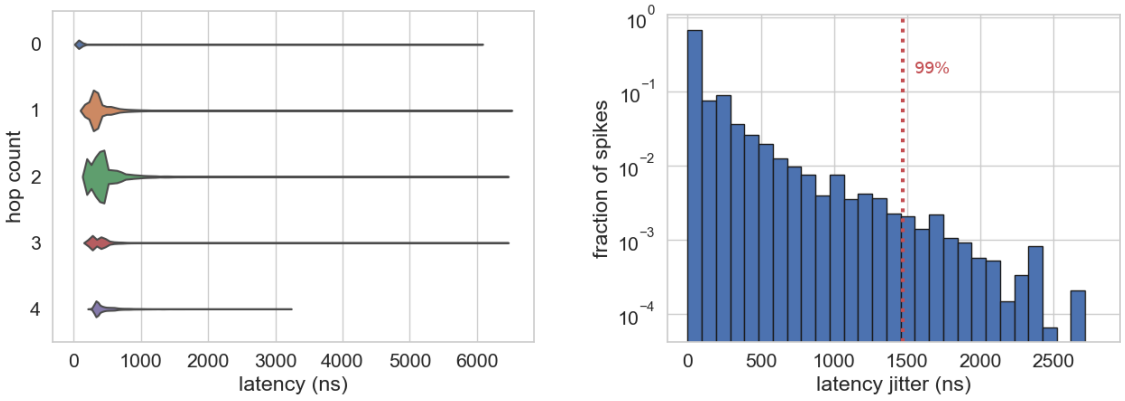
Total LUT size	1.02 Mb
Memory per connection	12.7 bits
Synaptic activation concurrency	7.2

Spike input data is generated by an external SNN simulator that was also used to train the network for digit recognition. It is then fed with a Poisson spike input of 100 Hz. The simulator is run and all resulting spikes in the network are captured. This capture is then used as a spike input for the spike-routing simulator.

5.3.2 Results

The described setup is run for 100ms with a total of 78,522 neural spikes and 1,365,826 synaptic activations. The resulting latency and latency jitter can be found in Fig-

ure 5.10.

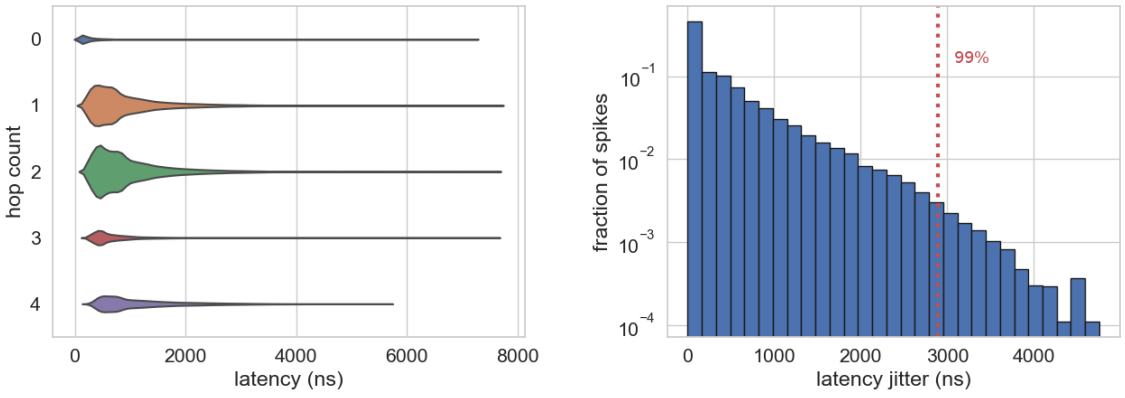


(a) (b)

Figure 5.10: Latency and latency jitter for spikes in handwriting recognition SNN with the optimised configuration.

The latency and latency jitter are significantly higher than for the previous benchmarks. The reason for this that there are more neurons spikes in close succession than in previous tests. A cause for this is that in SNNs neurons can synchronise and fire at (roughly) the same moment. This effect is not present in the synthetic spike inputs. Nevertheless, latency jitter never exceeds 1µs and for 99% of the spikes, it is below 520ns.

The same test is also run for the baseline configuration of $g = 1$, $B = 1$ and $B_{\text{offset}} = 0$. These results can be found in Figure 5.11. This clearly shows the effect of the higher synaptic activation concurrency of the synapse encoding scheme, as both the average latency and the latency jitter are significantly higher for the baseline configuration.



(a) (b)

Figure 5.11: Latency and latency jitter for spikes in handwriting recognition SNN with the baseline configuration.

5.3.3 Power and Area

Using the power and area models from Chapter 3 and the parameters from Table 5.13, the power consumption and area of the spike-routing architecture is estimated.

Table 5.13: Parameters for power and area estimation.

R_{avg}	100Hz
A_{neuron}	100µm^2
A_{synapse}	10µm^2

The main consideration from this estimation is the relatively big part of the power consumption for the Noc routers. So this is the main focus point for reducing overall power consumption. This can be achieved by either increasing the cluster size (and decreasing the number of routers) or by trying to reduce an individual router's static power consumption.

Furthermore, the area of the spike-routing architecture is 22% of the total system including the neuro-synaptic array. This means the area is a significant part, but inclusion on the same die should be feasible.

Table 5.14: Area and power estimation for implementation in 45 nm.

Cluster (9x)	
Area	0.223 mm^2
Power	91.2 µW
Neuro-synaptic array	
Neurons area	12,800 $µm^2$
Synapses area	163,840 $µm^2$
Total area	176,640 $µm^2$
Local interconnect	
Area	4,838 $µm^2$
Power	1.8 µW
LUTs	
Area	39,540 $µm^2$
Static power	2.5 µW
Lookup power	14.1 µW
Router	
Area	2,251 $µm^2$
Static power	67.7 µW
Dynamic power	5.1 µW
NoC links	
Area	15,450 $µm^2$
Power	0.30 µW
Total	
Area (excl. neuro-synaptic arrays)	0.44 mm^2
Area (incl. neuro-synaptic arrays)	2.0 mm^2
Power (excl. neuro-synaptic arrays)	821 µW

6 Conclusion

The aim of this thesis has been to design a memory-efficient spike-routing architecture for neuromorphic systems with high synaptic activation concurrency. In Chapter 3 we made a design space exploration. This led to a simplified model of spike-routing in neuromorphic systems that allows for quick evaluation of design parameters. Furthermore, the design space exploration shows that Hybrid Addressing can be really memory-efficient and can decrease the load on the NoC in comparison to the other evaluated schemes. Finally, it shows that clustering can improve power consumption and the importance of having an efficient synapse encoding scheme.

Chapter 4 described the implementation of a simulation of the spike-routing architecture described in the previous chapter. It also described several tools for supporting the simulation flow. In Chapter 5 this simulation was used to evaluate the mapping process and the spike-routing architecture itself. From these results, it can be concluded that the proposed synapse encoding scheme can help in improving memory-efficiency and synaptic activation concurrency. The results also show that the mapping optimisation process has an overall positive effect on mapping performance. The performance of the spike-routing itself is good. In the tests, spike latency rarely exceeded 1µs and latency jitter typically stayed below 500ns. Finally, the digit recognition test shows that the architecture also works well for a real-world application and it confirms the benefits of using the synapse encoding scheme.

6.1 Future work

Synthesizing for FPGA Current results are only based on modelling and simulation. Synthesizing the SystemC-code for an FPGA could provide a more accurate performance evaluation.

Variable synapse encoding for different clusters Currently, the assumption is that all clusters use the same synapse encoding. However, the architecture allows for changing this on a per cluster basis. Then neurons and synapses can be assigned to the cluster where their mapping efficiency is highest. This could especially be beneficial if an irregular connection distribution is used e.g., a layered network with different layer sizes.

Improve mapping optimisation performance For big SNNs, the mapping optimisation process currently takes a long time. This could be improved by using a heuristic to determine which change to make in the state instead of making a random mutation. By doing this, the algorithm can more quickly converge to an optimal solution because the probability that a mutation positively affects the state energy increases.

Implement neuro-synaptic array Not having an implementation for the neurons and synapses, drives the need for a synthetic spike generator. By actually creating implementations for neurons and synapses in the simulation, simulation results can become more accurate.

I/O-interface Currently, the only way to get spikes in and out of the system is by injecting spikes at neurons and read incoming spikes at the synapses. This does not work for an implementation with an actual neuro-synaptic array. For a practical application, there should be an I/O-interface that allows sourcing and recording of AER packets.

Packet time-to-live In some rare cases, a spike event might have a way long latency than average. By adding a timestamp to the packet, it can be tracked how long the packet has been in the system. If this exceeds the maximum time-to-live (TTL), the packet is dropped. As long as the fraction of dropped packets is kept low, this does not have to negatively impact the performance of the SNN.

Bibliography

[1] US National Cancer Institute's Surveillance Epidemiology and End Results Program, "Anatomy and Physiology."

[2] Computer Science GCSE, "Von Neumann architecture."

[3] M. Nielsen, "Neural Networks and Deep Learning."

[4] Y. Zhang, P. Li, S. Member, Y. Jin, Y. Choe, and S. Member, "A Digital Liquid State Machine With Biologically Inspired Learning and Its Application to Speech Recognition," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 11, pp. 2635–2649, 2015.

[5] M. Mahowald, *An Analog VLSI System for Stereoscopic Vision*. 1994.

[6] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[7] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[8] S. Scholze, S. Schiefer, J. Partzsch, S. Hartmann, C. G. Mayr, S. Höppner, H. Eisenreich, S. Henker, B. Vogginger, and R. Schüffny, "VLSI implementation of a 2.8 Gevent/s packet-based AER interface with routing and event sorting functionality," *Frontiers in Neuroscience*, vol. 5, no. OCT, pp. 1–13, 2011.

[9] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990.

[10] F. A. Azevedo, L. R. Carvalho, L. T. Grinberg, J. M. Farfel, R. E. Ferretti, R. E. Leite, W. J. Filho, R. Lent, and S. Herculano-Houzel, "Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain," *The Journal of Comparative Neurology*, vol. 513, pp. 532–541, apr 2009.

[11] D. Drubach, *The Brain Explained*. 2000.

[12] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, pp. 1659–1671, dec 1997.

[13] D. Cires and U. Meier, "Multi-column Deep Neural Networks for Image Classification," pp. 3642–3649, 2012.

[14] L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," *IEEE Signal Processing Magazine*, vol. 29, no. November, pp. 82–97, 2012.

[15] S. Lawrence, C. L. Giles, S. Member, A. C. Tsoi, S. Member, and A. D. Back, "Face Recognition : A Convolutional Neural-Network Approach," vol. 8, no. 1, pp. 98–113, 1997.

[16] A. Srivastava, A. Kundu, S. Sural, and S. Member, "Credit Card Fraud Detection Using Hidden Markov Model," vol. 5, no. 1, pp. 37–48, 2008.

[17] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, mar 2017.

[18] A. Tavanaei, "Acquisition of Visual Features Through Probabilistic Spike-Timing-Dependent Plasticity," pp. 1–15, 2016.

[19] J. Wu, Y. Chua, M. Zhang, H. Li, and K. C. Tan, "A Spiking Neural Network Framework for Robust Sound Classification," *Frontiers in Neuroscience*, vol. 12, no. November, pp. 1–17, 2018.

[20] J. Mes, E. Stienstra, X. You, S. S. Kumar, A. Zjajo, C. Galuzzi, and R. Van Leuken, "Neuromorphic self-organizing map design for classification of bioelectric-timescale signals," *Proceedings - 2017 17th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2017*, vol. 2018-Janua, pp. 113–120, 2018.

[21] T. Cross, "After Moore's law," 2016.

[22] I. committee, "International Technology Roadmap For Semiconductors," tech. rep., 2015.

[23] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses," *Frontiers in Neuroscience*, vol. 9, p. 141, apr 2015.

[24] X. You, *Full-Custom Multi-Compartment Synaptic Circuits in Neuromorphic Structures*. PhD thesis, 2017.

[25] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Silviotti, and D. Gillespie, "Silicon Auditory Processors as Computer Peripherals," *IEEE Transactions on Neural Networks*, vol. 4, no. 3, pp. 523–528, 1993.

[26] N. Brunel, "Dynamics of Sparsely Conntected Networks of Excitatory and Inhibitory Spiking Neurons," *Journal of Computational Neuroscience*, vol. 8, pp. 183–208, 2000.

[27] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, pp. 668–673, aug 2014.

[28] T. Hwu, J. Isbell, N. Oros, and J. Krichmar, "A Self-Driving Robot Using Deep Convolutional Neural Networks on Neuromorphic Hardware," 2016.

[29] J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 1947–1950, IEEE, 2010.

[30] "Extended Address Event Representation Draft Standard v0.4," *Event (London)*, pp. 1–10, 2002.

[31] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6 . 0 : A Tool to Model Large Caches," *International Symposium on Microarchitecture*, no. HPL-2009-85, pp. 0–24, 2007.

[32] A. B. Kahng, B. Lin, and S. Nath, "ORION3.0: A comprehensive noc router estimation tool," *IEEE Embedded Systems Letters*, vol. 7, no. 2, pp. 41–45, 2015.

[33] G. Karypis and V. Kumar, "Metis-1," vol. 20, no. 1, pp. 359–392, 1998.

[34] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

LUT content configuration file

A

An example of a configuration file for LUT contents:

```
{
  "0": { // cluster number
    "L": [
      {
        "offset": 5,
        "length": 23
      },
      ...
    ],
    "S1": [
      {
        "offset": 7,
        "length": 3
      },
      ...
    ],
    "S2": [
      {
        "cluster_x": 1,
        "cluster_y": 0,
        "intermediate_address": 33
      },
      ...
    ],
    "D1": [
      {
        "offset": 11,
        "length": 55
      },
      ...
    ],
    "D2": [
      {
        "column": 9,
        "rows": [
          {
            "set_address": 3,
            "activate": "1011",
          },
          ...
        ],
      },
      ...
    ],
  },
  ...
}
```