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

CAS-MS-2014-08

M.Sc. Thesis

Multi-chip dataflow architecture for massive scale biophysically accurate neuron simulation

Jaco A. Hofmann

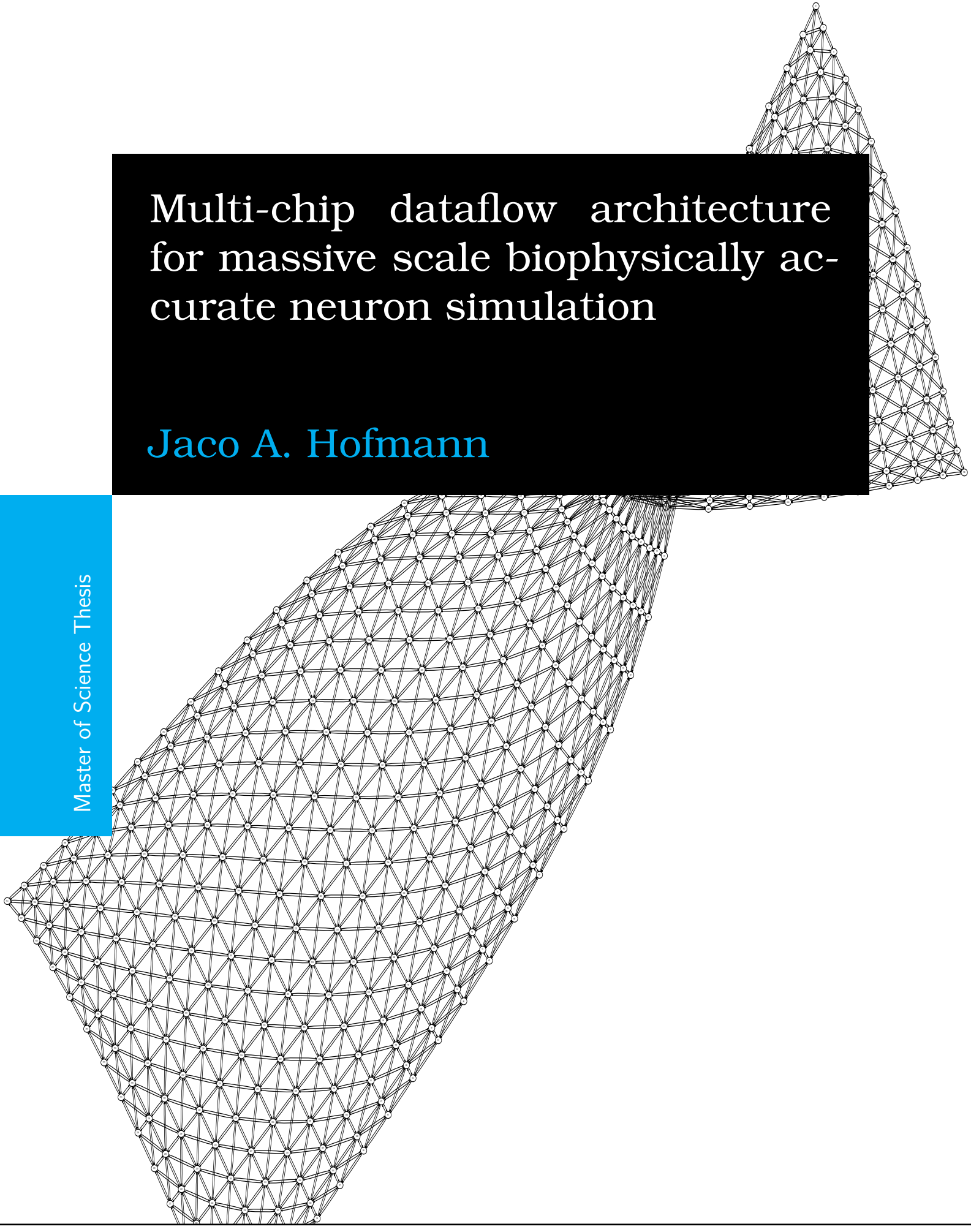
Abstract

The ability to simulate brain neurons in real-time using biophysically-meaningful models is a critical pre-requisite grasping human brain behavior. By simulating neurons' behavior, it is possible, for example, to reduce the need for in-vivo experimentation, to improve artificial intelligence and to replace damaged brain parts in patients. A biophysically accurate but complex neuron model, which can be used for such applications, is the Hodgkin-Huxley (HH) model. State of the art simulators are capable of simulating, in real-time, tens of neurons, at most. The currently most advanced simulator is able to simulate 96 HH neurons in real-time. This simulator is limited by its exponential growth in communication costs. To overcome this problem, in this thesis, we propose a new system architecture, which massively increases the amount of neurons which is possible to simulate. By localizing communications, the communication cost is reduced from an exponential to a linear growth with the number of simulated neurons As a result, the proposed system allows the simulation of over 3000 to 19200 cells (depending on the connectivity scheme). To further increase the number of simulated neurons, the proposed system is designed in such a way that it is possible to implement it over multiple chips. Experimental results have shown that it is possible to use up to 8 chips and still keeping the communication costs linear with the number of simulated neurons. The systems is very flexible and allows to tune, during run-time, various parameters, including the presence of connections between neurons, eliminating (or reducing) resynthesis costs, which turn into much faster experimentation cycles. All parts of the system are generated automatically, based on the neuron connectivity scheme. A powerful simulator that incorporates latencies for on and off chip communication, as well as calculation latencies, can be used to find the right configuration for a particular task. As a result, the resulting highly adaptive and configurable system allows for biophysically-accurate simulation of massive amounts of cells.

Multi-chip dataflow architecture for massive scale biophysically accurate neuron simulation

Jaco A. Hofmann

Master of Science Thesis



TU Delft Delft University of Technology

Circuits and Systems

Multi-chip dataflow architecture for massive scale biophysically accurate neuron simulation

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

Jaco A. Hofmann
born in Hamburg, Germany

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
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 **"Multi-chip dataflow architecture for massive scale biophysically accurate neuron simulation"** by **Jaco A. Hofmann** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 29.08.2014

Chairman:

prof.dr.ir. Alle-Jan van der Veen

Advisor:

dr.ir. René van Leuken

Committee Members:

dr. Carlo Galuzzi

dr.ir. Said Hamdioui

Abstract

The ability to simulate brain neurons in real-time using biophysically-meaningful models is a critical pre-requisite grasping human brain behavior. By simulating neurons' behavior, it is possible, for example, to reduce the need for in-vivo experimentation, to improve artificial intelligence and to replace damaged brain parts in patients. A biophysically accurate but complex neuron model, which can be used for such applications, is the Hodgkin-Huxley (HH) model. State of the art simulators are capable of simulating, in real-time, tens of neurons, at most. The currently most advanced simulator is able to simulate 96 HH neurons in real-time. This simulator is limited by its exponential growth in communication costs. To overcome this problem, in this thesis, we propose a new system architecture, which massively increases the amount of neurons which is possible to simulate. By localizing communications, the communication cost is reduced from an exponential to a linear growth with the number of simulated neurons As a result, the proposed system allows the simulation of over 3000 to 19200 cells (depending on the connectivity scheme). To further increase the number of simulated neurons, the proposed system is designed in such a way that it is possible to implement it over multiple chips. Experimental results have shown that it is possible to use up to 8 chips and still keeping the communication costs linear with the number of simulated neurons. The systems is very flexible and allows to tune, during run-time, various parameters, including the presence of connections between neurons, eliminating (or reducing) resynthesis costs, which turn into much faster experimentation cycles. All parts of the system are generated automatically, based on the neuron connectivity scheme. A powerful simulator that incorporates latencies for on and off chip communication, as well as calculation latencies, can be used to find the right configuration for a particular task. As a result, the resulting highly adaptive and configurable system allows for biophysically-accurate simulation of massive amounts of cells.

Table of Contents

List of Figures

List of Tables

Acronyms

ADC Analog-to-digital converter. 22, 23, 30, 58

ANN Artificial Neural Network. 8, 9, 11

BRAM Block Random Access Memory. 15

DAC Digital-to-analog converter. 22, 23, 30, 31, 58

FIFO First in, First out Buffer. 28, 29, 32, 35–37, 59

FPGA Field Programmable Gate Array. 4–6, 8–10, 13, 15, 16, 21–23, 53, 54, 56–58

HDL Hardware Description Language. 58

HH Hodgkin-Huxley. 59

ION Inferior Olivary Nucleus. 4

JSON JavaScript Object Notation. 25, 32

NOC Network on Chip. 16, 17, 19, 20, 24, 25, 28, 31, 57

PHC Physical Cell. 15, 16, 19, 24, 25, 27, 28, 31, 35–40, 42, 45, 50, 53, 54, 59

RAM Random Access Memory. 15

SNN Spiking Neural Network. 6, 8, 11

SPI Serial Peripheral Interface. 24

UART Universal Asynchronous Receiver Transmitter. 24

VLSI Very Large Scale Integration. 8

Glossary

Cluster Basic building block of the proposed system. Exploits communication locality to achieve linear scalability. 16, 19–21, 25, 27, 28, 30, 33

Fan-out Number of children a router in the system has. 19, 25, 35–37

Graphviz Open source graph visualization software [28]. 25

Half-normal distribution Normal distribution folded at its mean of 0 [25]. 25

Json-CPP Open Source JSON interaction library in C++ [29]. 32

Python Widely used general-purpose programming language that focuses on readability and expressiv power [30]. 25

SystemC C++ library for modelling and simulation of complex electronic systems, supporting both hardware and software simulation. Used in version 2.3 [12]. 2, 6, 7, 12, 25–27, 34, 35, 53, 57, 58

SystemC AMS Extension of SystemC for mixed-signal modeling [14]. 58

Wishbone Open source hardware computer bus used to connect parts in a system on chip [17]. 24

Chapter 1

Introduction

By increasing the computational capacities chips, it would be possible to replicate the behaviour of parts of the brain. Different fields of research are interested in having a device that can simulate parts of the brain in real time. On the lowest level such a system can be used to validate the models in use. More complex scenarios include the use of this system instead of in-vito experiments and to replace (damaged) parts of the brain. The design of such a system requires the work of researchers from many different research areas. Neurobiologists try to understand the brain and generate models of the behaviour of individual neurons with the help of mathematicians. By using these models computer simulations are generated and cross checked with the results of the neurobiologists.

A usable brain simulator for real-life experiments needs to be able to simulate large parts of the brain, which contain thousands, millions or even billions of cells. The highly parallel nature of neuronal networks leads to bad scalability on classical Von-Neumann machines [22]. One possible solution to this problem is the design of highly parallel dedicated hardware. In this thesis we present a hardware architecture capable of dealing with massive numbers of cells needed for large and accurate brain simulators.

After introducing some basic notions about the neurons, the main problem addressed in this thesis is described. After that, we present the main goals and contributions of the work presented in this thesis. Finally, we present a short outline of the thesis.

1-1 Neurons

This thesis uses an abstract model of nerve cells, also known as neurons. Before continuing further, we give some basic notions about neurons. Neurons are cells inside the brain which are used to process and transmit signals. They mainly consist of three parts. Via the *dendrites* inputs from other neurons are received as electrochemical stimuli. These stimuli are transferred to the *soma*, where they are processed and stored as a membrane potential.

If this potential reaches a certain threshold, a spike is transferred via the *axon* to other neurons.

The neurons considered in this thesis are located in the cerebellum and the Inferior Olivary Nucleus (ION). They are important for the coordination of the body's activities [6]. The ION is an especially well chartographed part of the brain [18].

In this thesis, a mathematical model of the neuron is used. In this model the inputs represent electric potentials from other neurons. By using these inputs the output of the neuron is calculated. These results are then forwarded to other neurons. In our work neurons are treated as computing cores that have n inputs and m outputs. These computing cores need a certain amount of time to complete their calculation. Furthermore, they contain parameters that influence their calculations. These parameters represent the concentration levels of different chemicals inside the neuron. Additionally, the neurons are connected following different patterns such as all-to-all connection, connection between neighbouring cells, or more sophisticated connection schemes based on probability or movement directions.

1-2 Problem Description

This thesis is based on a hardware design for an extended Hodgkin-Huxley model of the ION neurons. The simulator in use applies discrete-time steps. In each time step all the neurons are evaluated before proceeding to the next time step. To accurately represent the behaviour of the neurons each time step can not take more $50\,\mu$s. This time is known as brain real-time. The current hardware design[27, 26] is capable of simulating 48 neurons (96 neurons with optimisations) in brain real-time. In order to provide meaningful results, the system should be able to simulate the behaviour of thousands/millions of cells. Existing systems are limited by the time needed for communication by the network connecting all the neurons, as shown in Fig. 1-1. This system is implemented on a Field Programmable Gate Array (FPGA), which is a device that can be used to implement arbitrary sequential or combinational logic. The amount of logic available on an FPGA is limited. As the system supports only one FPGA the design is limited by the time required for communication as well as by the FPGA resources used by the design.

Therefore, to make the system suitable for experiments, the time used for communication has to be independent of the number of cells in the system. By achieving constant communication time, it is possible to simulate arbitrarily large systems only limited by the size of the FPGA. In this thesis a way needs to be found, to overcome the constraints that are posed by the current system. The following section specifies exactly what the requirements of such a system are.

1-3 Goals

Based on the problem description given in the previous section the goal of this thesis is to identify a method to interconnect simulated neuron cells in a way that allows for "massive" numbers of cells compared to the previous approaches. An increase of the number of

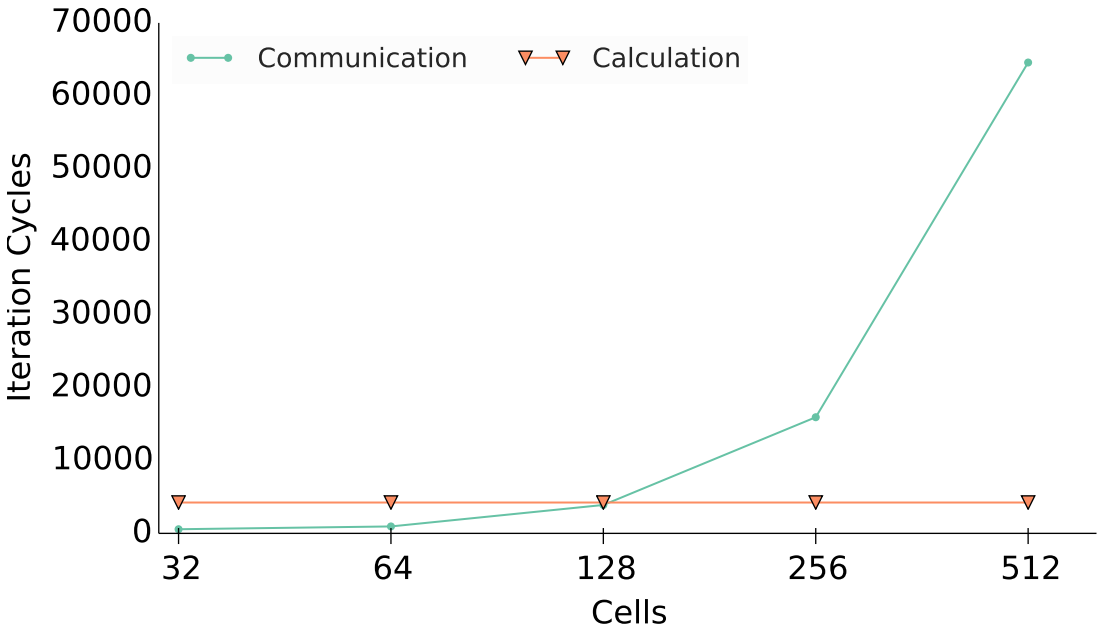


Figure 1-1: Time spent for the communication in each iteration of the extended Hodgkin-Huxley model in the implementation done in [27]. The system is limited by the communication cost which scales exponentially with the number of cells. To meet the brain real-time requirements, each iteration may only take up to 5000 cycles at $100\,\mathrm{MHz}$ clock speed.

cells in the system should not increase the communication time required by the system to handle all the communication between the cells. Furthermore, the system should be expandable to more than one chip as every chip has a certain limit in size. To overcome this limitation, we should be able to connect multiple chips in order to build a larger system.

The connection method itself has to support different types of connection schemes between the cells. Besides the all-to-all connection scheme in which every cell is connected to every other cell, there are more "realistic" connection schemes. The simplest scheme is the neighbour connection scheme in which a cell is connected only to its neighbouring cells in a 2D grid. Additionally, probability based connection schemes have to be supported as well. For this type of connection scheme, two cells are connected based on some distance function, for example geometric distance, and a probability distribution, for example normal distribution.

As synthesizing the system for use on an FPGA takes time, it is beneficial, to be able to change certain parameters of the system at run-time. Using this configuration method, it should be possible to change the parameters of system that do not change its structure. These parameters might be for example, connections between cells or parameters of the calculations that happen inside the cells.

In this thesis we design and evaluate a system which fulfills these requirements. The methods used for the design and the evaluation are presented in the next section.

1-4 Design and Evaluation

The system design is based on the work done in [27] and the improvements applied to that work in [26]. The design is implemented using SystemC, which allows for cycle accurate simulation of the system at a high level compared to classical Hardware Description Languages (HDLs). Based on the features of the brain, an optimal approach is implemented that represents the lower bound for the run-time of the system. While such a system is not implementable in hardware certain features can be extracted to be used in the hardware design. The system design is evaluated by running simulations for different parameters and scenarios and comparing the results with the previous system as well as the optimal system. Furthermore, an estimate for the hardware resource usage of the design is given.

1-5 Contributions

The proposed system provides several key aspects compared to existing approaches:

- **Close to linear growth in the communication cost.** As a result, a massive amount of neurons can be simulated compared to the state-of-the-art design that is limited by the exponential growth in the communication. The linear growth in the communication cost is achieved by *localising* communication. Data localisation and the resulting linear growth in communication cost result in $31\times$ to over $200\times$ more simulated neurons compared to the state-of-the-art design which is capable of simulating 96 cells in brain real-time.
- **Extendability of the system over multiple chips to build more accurate systems.** The system maintains linear growth up to eight FPGA.
- **High run-time configurability** which reduces the need for resynthesizing the system. Additionally, adaption of routing tables and changes to the calculation parameters are also possible. In this way, the system reduces the time required for experiments with biophysically accurate neurons.
- **Powerful simulator**, which can be used to explore different system configurations. As a result, it is possible to find the most suitable system structure for the task at hand. The simulator features configurable on- and off-chip communication latencies as well as neuron calculation latencies.
- **Automatic configuration generation** including structure and connectivity of the neurons. Routing tables are automatically generated based on the connection strategy used.
- **Higher biophysical accuracy in neuron simulations compared to the state-of-the-art.** Possibly, massive amounts of cells can be used in a variety of research areas to increase the understanding of neuron behaviour.
- **Designed for high precision Spiking Neural Network (SNN) simulations** but flexible enough to be used for smaller neural networks.

1-6 Thesis Outline

This thesis is structured as follows:

Chapter 2 presents the work done in [27, 26], which is the base for this thesis. Furthermore, related work is introduced. A comparison of these works is proposed and the advantages of the proposed system are described.

Chapter 3 presents the system design from a high level perspective, sparing out implementation details. The chapter starts by presenting the optimal solutions for the problem. Which are modified to build the basis of the system design. Finally further aspects of the system design are introduced.

Chapter 4 presents the details of the SystemC implementation omitted in Chapter 3.

Chapter 5 contains simulations for different scenarios including network and multi-chip performance in comparison to previous work and the optimal system. The chapter closes with an estimate for the hardware utilisation to show the feasibility of a hardware implementation.

Chapter 6 concludes the thesis and gives final remarks about the work done as well as a summary of the results of the design and evaluation steps. Finally, ideas for improving the proposed system are presented.

Chapter 2

State of the Art

Artificial Neural Networks (ANNs) provide a basic simulation of neuron activity by only modeling the frequency of neuron spikes. Nonetheless, these simple models can already be used in real world application, such as face recognition [15]. In contrast to typical computing models, the algorithms are not processed as sequential instructions. The output of an ANN consists of the connections of artificial neurons and their input and output states, which are evaluated in parallel. Advances in neuroscience and computing allow for more accurate simulations of neuron behaviors in SNNs. Instead of only simulating the frequency of spikes, the shape of these spikes is also taken into consideration [7]. The increase in precision can be used to improve the understanding of the brain [9]. By using accurate models of brain regions, in-vivo experiments that are time consuming can be replaced by simulations. Furthermore, accurate brain simulation allows for better Artificial Intelligence (AI). In patients, very sophisticated SNN could replace damaged parts of the brain in the future.

On the downside, the higher simulation precision of SNNs requires more computational resources compared to ANNs. Simulation on sequential computers is not feasible due to the high parallelism required by SNNs. Large supercomputers, while parallel enough, are too large to be used for use cases other than behaviour studies. Very Large Scale Integration (VLSI) designs provide the necessary parallelism but do not allow for altering the neuron model after manufacturing and, thus, are too expensive for the experimental stage. FPGAs are both fast enough to simulate a SNN in real-time or even hyper-real-time and flexible enough to allow model changes and, additionally, they are low power devices. The following section gives an overview of different SNN models and the corresponding hardware designs.

2-1 Related Work

Nowadays, there exist different hardware simulators for SNN. The main differences between the simulators are in terms of precision and flexibility. A basic model is the Quadratic

Integrate and Fire (IaF) model used in [13]. By using a Virtex-5 FPGA the authors are capable of simulating 161 neurons with 1610 synapses at 4120 times the real-time speed, whereas the real-time speed is given as one simulation step per 1 ms. The model consists of a basic integrator. The neuron sends a spike as soon as the cell membrane potential passes a certain threshold [4]. While this type of model provides improvements over simple ANN, they can not be used to generate biologically-meaningful information about the cell state. The corresponding simulator allows for neuron network adaption at run-time.

The Hodgkin-Huxley (HH) model originating from [1] allows for a more accurate representation of the cell state. The model, which belongs to the group of conductance-based models, incorporates the membrane potential. The model also includes the concentration of various chemicals, inside the neuron, in the calculations to represent its behaviour. In contrast to the simple IaF model, the HH can be used to research the behaviour of neurons accurately enough for biologically-meaningful experiments. In [20] a simplified version of the HH model is used in an FPGA based simulator that is able to simulate 400 physiologically realistic neurons. This design, the NepteronCore, uses a time step of 0.1 ms on a Virtex 4 FPGA. The simulation can be controlled via a graphical user interface.

A different approach is followed by [24]. The authors aim at simulating a large scale neural network using an analog approach. Instead of calculating the cell states using differential equations, the neurogrid system replicates the neuron as an electrical system [2]. The communication between cells is realised in digital hardware. By using 16 so called Neurocores, which are the computation elements of the Neurogrid, the system is able to simulate $1\,048\,576$ neurons with billions of synapses. While providing huge numbers of neurons compared to the FPGA based solutions, the Neurogrid is not as flexible as the FPGA based solutions concerning model changes. Furthermore, it is not possible to observe the neuron behaviour as easily as in the FPGA based solutions.

Large scale experiments are not only done with analog hardware but also with huge numbers of general purpose processors as in the SpiNNaker project [16]. The project uses a system of over one million low power ARM cores connected by a fast mesh based interconnect link. Currently the largest SpiNNaker system is able to simulate over one billion neurons. The huge number of neurons comes at the price of lower biological accuracy. The Izhikevich neuron [8] is more accurate than the IaF models but less than the HH model. The model focuses only on the input and output behaviour of the neurons. Behaviour studies of the inner workings of a neuron itself are impossible. Additionally, the time step used is 1 ms and, thus, slow compared to other models.

Another design based in the Izhikevich neurons is presented in [21]. The authors propose a Izhikevich neuron[10] simulator capable of simulating 64000 cells. The simulator achieves 2.5 times real-time and is 1.4 times faster than a GPU accelerated implementation. Fixed point arithmetic is used and the time step is 1 ms. The design is event driven, which means, that information is only transmitted when the packet contains meaningful information. Compared to the SpiNNaker system, which simulates the same type of neurons, only a single FPGA is used and, as such, the resulting system is comparably energy efficient.

The design of a biophysically accurate representation of the neuron is proposed in [23]. The system simulates all parts of the cell precisely using floating point arithmetic and the Hodgkin-Huxley (HH) model. For each part of the cell specialised processors exist, namely,

the HH somatic, Traub somatic, Traub dendritic [3], and synaptic processors. By combining these specialised processing cores the neural network is built. The price of the biophysical accuracy is low network size. The largest system proposed contains four neurons.

As described in Chapter 1 this thesis focuses on an extended version of the HH model [19]. In [27], an implementation of the extended HH model on a Virtex 7 FPGA is presented. This implementation provides a speed up of $6\times$ over a C implementation and is able to simulate 48 neurons at a real-time rate of $50\,\mu$s. Compared to the other implementations discussed so far, this model is the one with the highest biologically accuracy. [26] presents further improvements to the system. By incorporating different optimisations, 96 neurons can be simulated in real time providing a speedup of $12.5\times$ over the original C implementation. As the system designed in this thesis is based on these implementations, in the following section we provide a short description of these systems.

2-2 Previous Work

The large neuron simulation systems presented in the previous section use simple neuron models. The HH model implemented in this thesis, on the other hand, requires more complex calculations. The basis for this work lies in [27] and [26], as mentioned in Section 2-1. Ensuing from a C implementation of the model, a hardware implementation is devised. The brain real-time of $50\,\mu$s is much larger than typical clock frequencies for FPGA, which is in the range of $100\,$MHz. Thus, it is possible to use a single physical cell to calculate the result of multiple cells in one simulation time step. One Physical Cell (PhC) calculates about eight of the time shared cells. The baseline in [27] can simulate 48 cells in brain real-time which corresponds to six physical cells using a time multiplexing factor of eight. For the 96 cells simulated in [26] the time required to calculate one cell is halved, thus, allowing for more shared cells. The connection between the PhCs is realised as a bus. The bus connects all the cells using a single connection. A bus master decides which physical cell is allowed to use the bus. Due to the shared connection medium, the number of physical cells negatively impacts the communication times. This behaviour is presented in Fig. 2-1. As shown in the graph, the time required for communication between the cells increases exponentially. Thus, increasing the amount of physical cells is not feasible for the system. Furthermore, the system first calculates the outputs of all cells and, then, communicates these results instead of doing both operations in parallel. A scheme of the system is shown in Fig. 2-2.

2-3 Conclusion

As presented in this chapter, there exist different methods for the simulation of the behaviour of the brain, or parts of it. One of the simplest approaches is ANN, which is widely used in all kinds of systems. SNNs represent a group of neural networks that, in contrast, to ANN incorporate more features of the biological neurons in the brain. Depending on the target of the model, the SNN models range from computationally simple to very complex. The very complex models can even be used to do research on the neuron operations in the

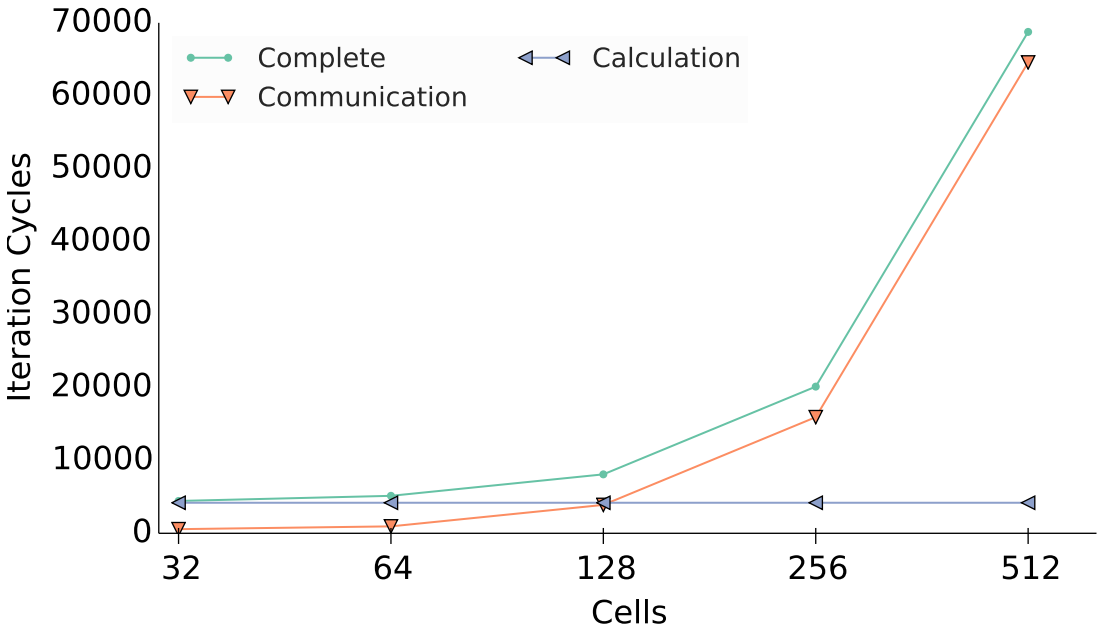


Figure 2-1: Amount of cycles required by the architecture to simulate the HH model as designed in [27]. As shown, the calculation time remains constant while the communication time scales exponentially with the amount of cells in the system.

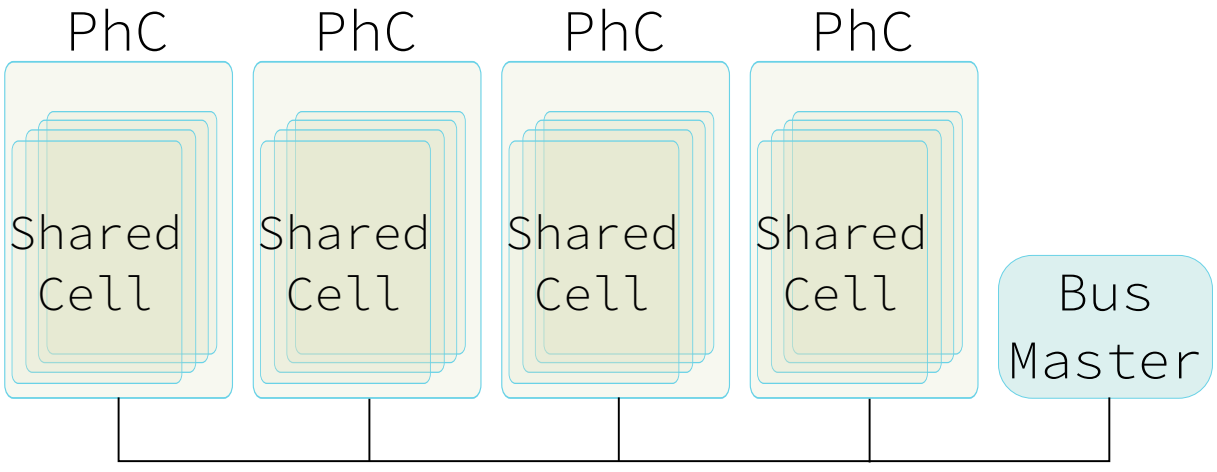


Figure 2-2: HH simulator as designed in [27]. The Physical Cells (PhCs) simulate multiple neurons in a time shared fashion. They are connected by a shared bus, which is controlled by the bus master.

brain, without having to fall back on in-vito experiments. One such model, the Hodgkin-Huxley model, simulates the chemical properties of the cell as well as the properties of the connections between cells. As SNN, require highly parallel operations, hardware implementations are advantageous. One such implementation is presented in [27, 26]. The main bottleneck of this system is the interconnect between cells as it scales exponentially with the number of cells in the system. Currently a maximum of 96 cells can be simulated fast enough to achieve the brain real-time of $50\,\mu s$ per simulation step.

As the current bottleneck is posed by the shared bus connection, the next chapter focuses on designing a better interconnect that achieves linear communication cost growth. The computation blocks (PhC) are incorporated and slightly modified to achieve a better scalability.

Chapter 3

System Design

This chapter presents the features of the designed system. Furthermore important parts of the design space are presented and compared. This chapter is supposed to provide a high level view about the design space whereas details of the SystemC implementation are presented in Chapter 4. The system design follows closely the requirements for the system. Thus, this chapter starts of with presenting the requirements. Afterwards an optimal system implementation for the problem is designed based on the requirements. The optimal design is then transformed into an implementable solution. Finally the different components of the system design are introduced.

3-1 Requirements

When designing a system it is important to properly specify its requirements. This is necessary to understand what area of the design space should be explored. While all requirements should be considered, they are not of the same importance. The system has to be able to achieve brain-real-time of $50\,\mu$s for one calculation round. While the cell calculations themselves are static in execution time for an increasing number of cells as they are executed in parallel, the interconnect used in [27] scales exponentially with increasing number of cells. From this follows the first requirement for the system: *linear scalability*. The network has to be fast enough, even for tens of thousands of cells, to be faster than processing all calculations, thus, making calculations the dominating time consumer.

Modeling learning and other processes dynamic processes in the brain requires *flexibility* of the system. During run-time multiple parameters need to be changeable including generation and destruction of connections between cells. Furthermore, parameters of the calculations themselves need to be changeable. These parameters, among other things, represent the concentration levels of various chemicals in the cell. Being able to change these concentration levels during run-time enables the researcher using the system to explore effects of drugs and other environmental factors.

The properties of the brain model should be closely incorporated in order to achieve a high system performance for the given requirements. The most important property of the system in this case is the connection scheme. As described in Section 2-2 cells that are further apart are connected less likely. The distance in this case depends on the model and might be geographical distance inside a 2D or 3D cell layout or favour a certain direction of movement.

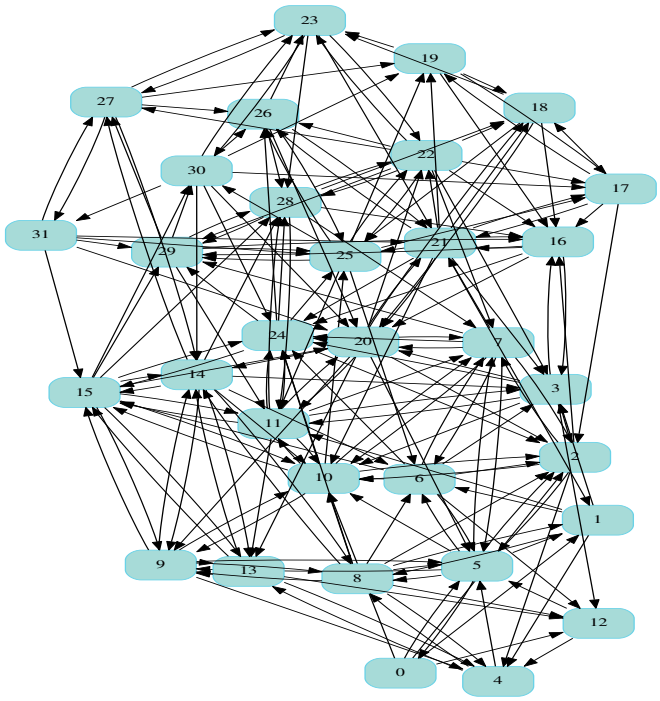
Another requirement to the overall system is imposed by the size of today's FPGA. The brain model profits a lot from massive numbers of cells and it is desirable to be able to extend the system across the boundary that a FPGA imposes. Thus, the system design should incorporate a method of interconnecting multiple FPGAs to form one large system.

Using these requirements for the system an optimal implementation is designed. The implementations proposed in the next section are optimal in respect of run-time but can not be implemented in hardware due to resource constraints. The optimal system design is used to evaluate the absolute limitations of the system as well as deriving a feasible system from it.

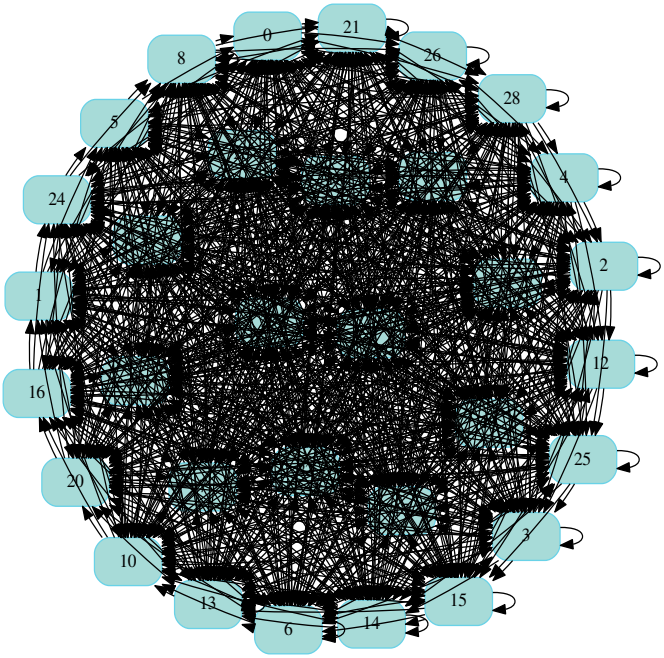
3-2 Zero communication time: The Optimal Approach

One advantage of using simulations in hardware design is the ability to simulate systems that are (not yet) implementable in hardware due to resource or timing constraints. This ability is used to evaluate an optimal approach to inter neuron connections in regards to brain-real-time performance. While the result of this experiment is not implementable in hardware it helps to assess the absolute limits of the system and may provide helpful insights into the neuron model that can be used to generate a synthesizable system.

The first approach might be to model the neuron interconnection following the structure of the brain. The resulting model would require each cell to have physical connections to all neighbouring cells. If a cell is a neighbouring cell depends on the connection scheme in use. It might be that all cells are connected to all other cells or that only direct neighbours in a 2D grid are connected. The connection scheme can be arbitrarily complex. Each of these connections has to be big enough to transfer the complete data of the cell which are 64 bit in this case. Even ignoring any synchronisation lines, which are necessary because of the discrete nature of the system, it is easy to see that such a system is not implementable. When flexibility comes into play the situation is even more grim. As new connections can not be generated during run-time it is necessary to have every possible connection already implemented. Thus, each cell has to be connected to every other cell in the system. Figure 3-1 gives an idea of the interconnect complexity that is already required for only 32 cells. Considering the discrete nature of the system another optimal system can be designed using a shared memory. Instead of connecting each cell to each other cell the communication data is stored in a memory that is accessed by all cells simultaneously. This memory would have to have n read and n write ports for n number of cells as shown in Fig. 3-2 for 4 cells. Again such a memory does not exist for the amount of cells needed in the system. Furthermore the necessary amount and length of wiring makes this approach infeasible. Using IEEE754 doubles with 64 bit as data and an address length of 12 bit (4096



(a) Only necessary connections



(b) All to all connections

Figure 3-1: Connections between 32 cells. While (a) includes enough connections for the system to run, no new connections can be created during runtime. To achieve flexibility all connections would have to be utilised as shown in (b). Both connection schemes are not efficiently implementable in hardware.

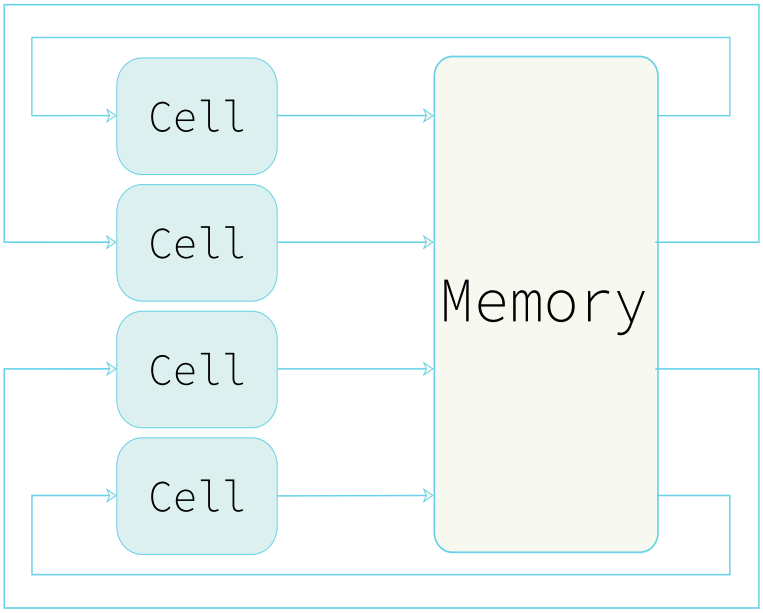


Figure 3-2: Optimal system with 4 cells utilising a shared memory. Each cell can write to and read from the shared memory. Thus, every cell is indirectly connected to any other cell. This system is not implementable in hardware as it requires too many memory ports and the wiring gets too complex.

cells addressable) each cell needs $2 \times (64 + 12)$ bit $= 152$ bit for read and write address and data ports. While corresponding Random Access Memory (RAM) can be modeled on a FPGA using for example time sharing of the Block Random Access Memory (BRAM) it would not be optimal anymore and would still use too much hardware resources to be feasible.

Nonetheless the shared memory optimal system can be used to derive part of a synthesizable system. This process is presented in the following section.

3-3 Localising communication: How to Speed Up the Common Case

While the designs in Section 3-2 deliver the best possible performance they are not implementable in hardware, but they are a good start to design a synthesizable system. A common goal of optimization is to make the common case fast. In this case the common case is communication between cells that are located close together as described in Sections 2-2 and 3-1. For good system performance it is most important, that communication between those cells that are close together is very fast. Communication to cells further away is rare and can be slower.

The main problem with the shared memory optimal approach presented in Section 3-2 is that the shared memory does not have enough ports to handle all cells in parallel. Looking at the calculations of each cell as implemented in [27] indicates, that data from other cells is read seldom in the Physical Cell (PhC). Thus, a single cell does not require memory access for each clock cycle, allowing for a shared memory design with time shared instead of parallel memory access. The main advantage in this approach is, that the common case

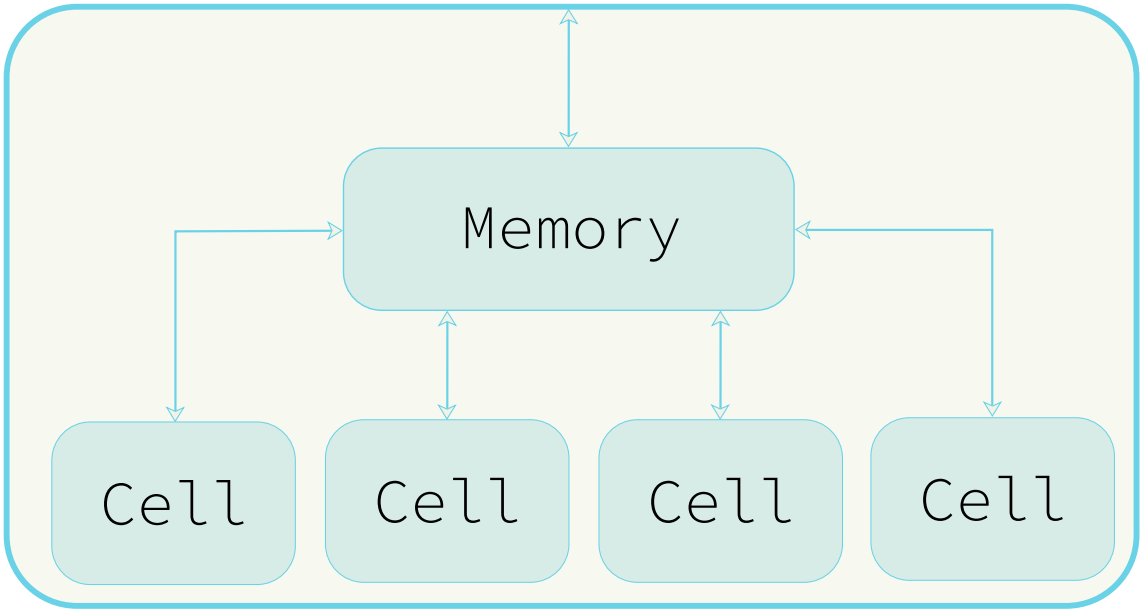


Figure 3-3: Abstract design of a cluster. PhC are grouped around a central shared memory. The cluster is connected to other clusters using an interconnect.

of close communication is still optimal. This claim is only true for a certain number of PhC in a cluster. At some point each PhC has to wait too long for memory access and the performance decreases. Furthermore the number of PhC around one shared memory is limited by placement and wire length constraints of the FPGA technology used. The exact number of PhC around one shared memory has to be determined by simulation. These PhC around a shared memory with corresponding control logic is furthermore called a cluster (Fig. 3-3).

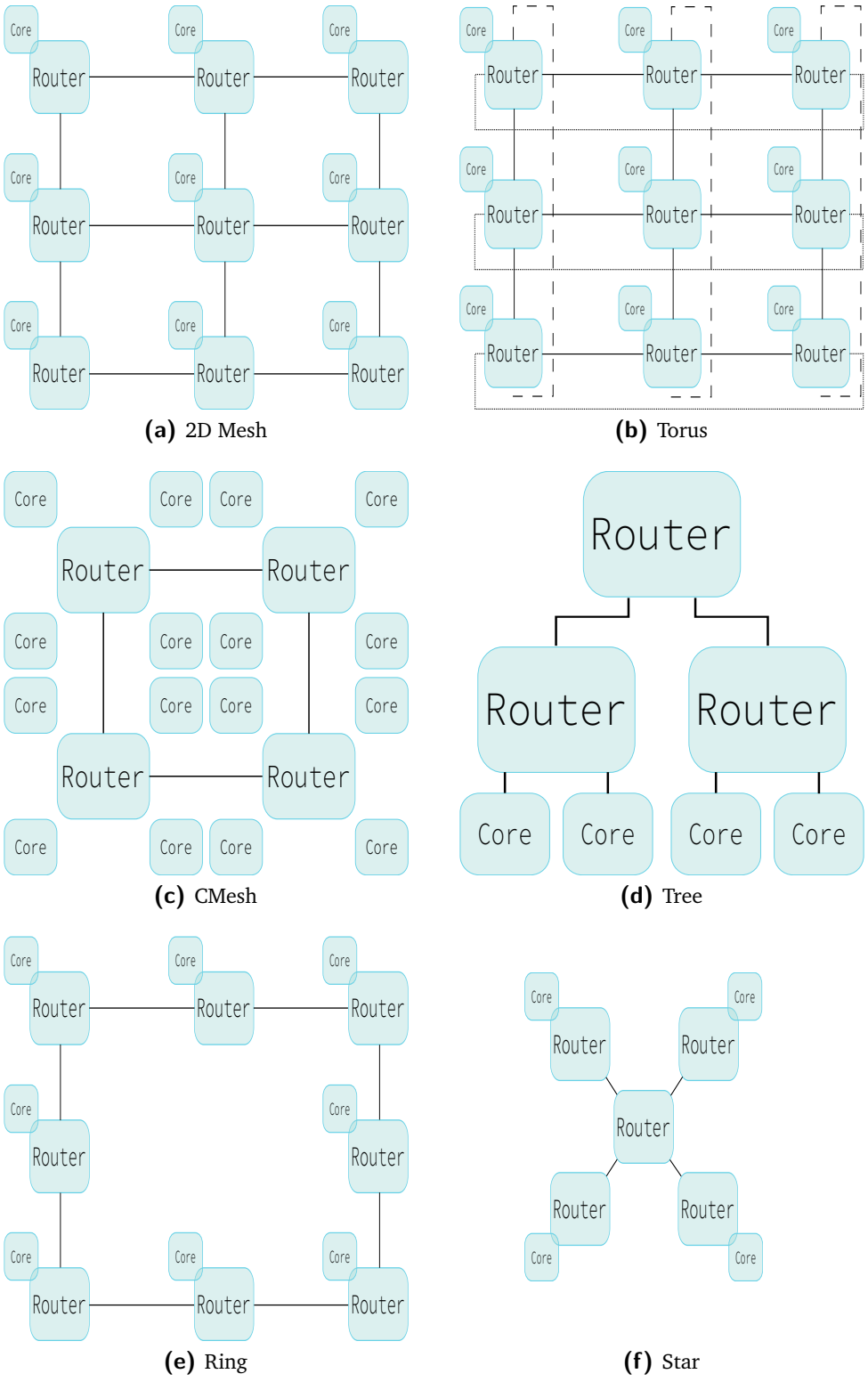
Apart from the common case there is still the uncommon case of communicating with cells outside the own cluster that, while not as important in respect to performance, has to be handled correctly. Many ways to design such an interconnect are possible. The simplest being a bus connection between clusters like [27] used for the connection between PhC. Even though these connections are infrequently used the scalability of the bus imposes a high cost for extending the system with more clusters. An other approach on interconnects becoming increasingly popular in the last years are Network on Chip (NoC). These interconnects promise higher flexibility and performance than classical interconnect approaches. The following section will give a short introduction into NoC and outline how such a NoC can be incorporated into the system.

3-4 Introduction to Network on Chips

There are different methods of connecting components in a system. The simplest and most straight forward is the connecting two components with wires for data and some control lines. While simple this type of connection is also expensive when multiple components have to be connected and furthermore not very efficient as most connections will not be

used all the time. A better and only slightly more complex approach is the use of a bus. Such a communication system uses a single connection between all components and a bus master organizes the access to the single communication medium. As all components can communicate with each other a bus is less expensive than dedicated connections. On the other hand all communications have to time share the one communication line which decreases performance in systems with frequent communications. A newer approach for component connection, that promises higher performance as well as higher efficiency combined with low hardware costs, on a chip are NoCs. Typically a NoC as presented in [5] consist of routers that are connected by wires. Each component of the system is connected to one such router. When a component wants to send data to another component it injects a packet into the network by passing the packet to its router. To support variable size packets a single packet is typically split into flits. These flits contain information about their type to indicate if the flit is the head or tail of a packet or even both at the same time when the packet only contains one flit. Additional information in a flit depends largely on the NoC implementation. Typical additional information contained in a flit besides the data is the size of the data contained in the flit, the route the packet has to take or the destination depending on the routing algorithm in use. Furthermore, information for prioritization of certain data using for example virtual channels can be included in a flit. These flits are routed to a destination according to a routing algorithm and the topology in use. The topology for the NoC is chosen according to the parameters of the system. Typical topologies are presented in Fig. 3-4. These topologies might be enhanced by using express channels to connect further away routers for increase throughput as shown in [11]. The routing algorithm in use depends on the characteristics of the network and the topology. A simple routing algorithm for a 2D Mesh is called XY-Routing which is a deterministic routing algorithm that always results in the same route taken for a flit. First the packet is routed along the X axis then along the Y axis to reach the destination. While simple, this type of routing has the flaw of being agnostic to the traffic in the network. Certain routes might be blocked while others are completely empty. The complexity of routing algorithms can be increased by, for example, considering multiple shortest routes to a target and then selecting one of these paths randomly for each flit resulting in better network load. Even more complexity and efficiency can be added by using heuristics and information about the network load at any given time. Other types of NoC require different routing approaches like static routing tables or source based routing. For source based routing only the source of a flit is included and the network decides which components should receive the packet instead of the sender. Such a routing algorithm can be advantageous when one packet has to arrive at multiple receivers as it avoids resending of the same packet to different destinations.

A general overview of NoC was given in this section. The design in this thesis on the other hand has very specific characteristics which allow for a highly optimised NoC. For example no flits are needed as all packets are of the same size and of the same priority. The next section describes the design of the NoC used in this design.



(a) 2D Mesh **(b)** Torus

(c) CMesh **(d)** Tree

(e) Ring **(f)** Star

Figure 3-4: Typical topologies for a network on chip.

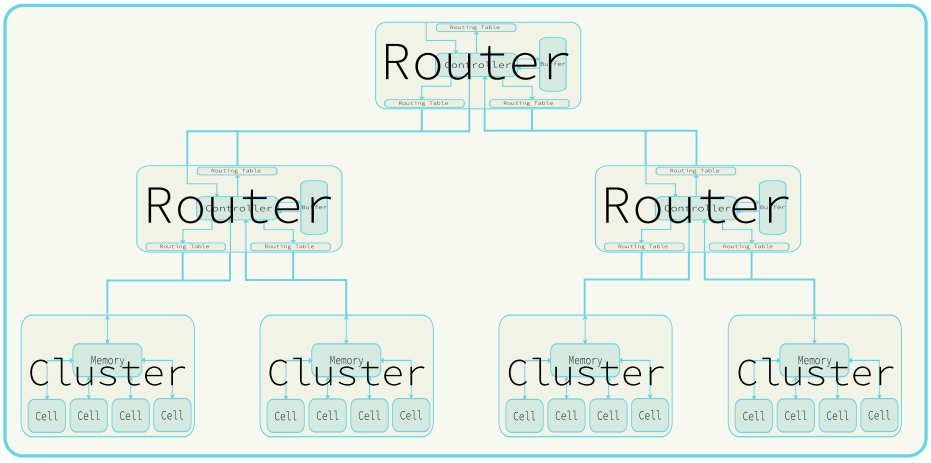


Figure 3-5: The complete network structure as designed in Sections 3-3 and 3-5. The computing elements PhC are grouped inside a cluster to make communication between neighbouring cells fast. These cluster are connected in a tree topology NoC. The router fan-out in this case is 2 and can be changed according to the requirements of the implementation. The same holds true for the number of PhC in any cluster.

3-5 Localise Communication Between Clusters

The last section described different topologies and standard routing schemes. Most of these approaches aim at general NoC, while the NoC to be used in this design has very specific characteristics that allow for a selection of the most fitting NoC. The cluster already localise communication between close by cells. It is preferable for the NoC to also speed up the close by communications. The tree topology described in Section 3-4 is suitable for this type of communication. Most packets between clusters will only incorporate the next cluster. In a tree based structure, as shown in Fig. 3-5, it will rarely happen that a packet has to cross multiple layers in the tree. The most important parameter that has to be determined through simulation is the fan-out of the routers. A larger fan-out results in more cluster any single router has to deal with. On the other hand a lower fan-out will result in higher hardware utilisation by requiring more layers in the tree.

The second important aspect for the NoC is the routing scheme. Any packet might go to any other cell. Sending a single packet to any other cell in the NoC is infeasible as this would result in a lot of duplicated packets. Furthermore, each packet would have to consist of the destination of the packet, the source of the packet and the data, thus requiring a lot of data for each packet. Moreover each router would need to know where the destination of the packet is making the routers more complex. A better approach is to have each packet only include the source of it. The packet is then routed to the destinations based on the source of the packet. Each router has to decide to which of its outputs the packet has to be forwarded to but the router does not need to know where the packet will go in the end, thus ensuring that each router is small and efficient. With the structure of the NoC in place the next step is the design of the routers.

The routers have to fulfill certain requirements. They include n inputs and n outputs. One of these input, output pairs is corresponding to the upstream in the tree and the rest to

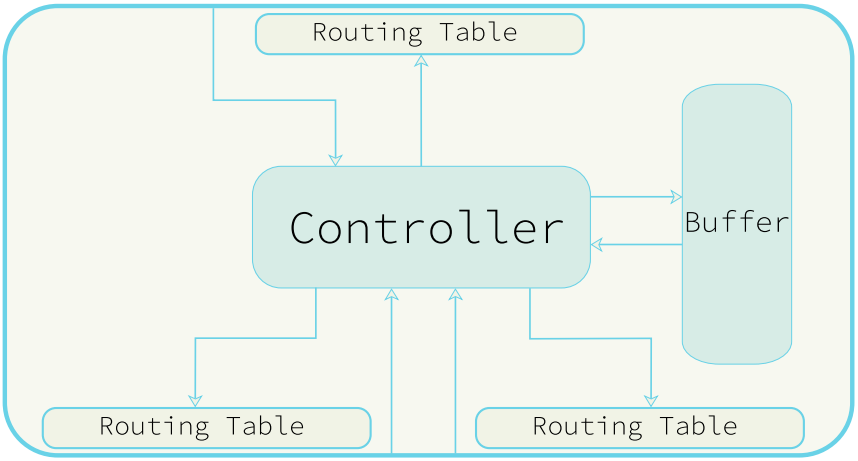


Figure 3-6: The abstract router described in Section 3-5. The controller queries each input and forwards packets according to the routing tables of the outputs. A packet may be forwarded into any direction and therefore each output has its own routing table. If a receiving router can not accept a packet that packet is stored inside the buffer till it can be forwarded.

the downstream targets. The upstream channel is not present in the root router of the tree. From each of the input a packet might go to each of the outputs but it can not be looped back. For each possible source of a packet the router may decide where to route the packet based on static routing tables. Each packet may be routed to all outputs or to no output at all. In addition each router needs to be able to deal with congestion. The packet might need to be routed to a router that signalizes it is full. In such a case the packet has to be send at a later point when the receiving router is able to process the packet. A classical approach would be to drop the packet and inform the source of the packet to resend it at a later time. Considering the properties of the NoC dropping packets has down sides. The router would have to remember that it already got a packet from a certain source and only forward the packet to the blocking router, thus requiring more hardware. Furthermore there needs to be an additional channel to indicate to the cluster that a packet was dropped. A better approach for this scenario is to use memory to store all packets that can not be forwarded right away and try to forward them later at a suitable time. The size of this “delayed” packet buffer can be determined by simulating an all to all cell connection scheme, or more specific cases depending on the use case, calculating the highest fill rate of the buffers. Moreover different optimisations of the routing are possible like preferring upstream packets over downstream packets. The abstract router design is presented in Fig. 3-6.

With the routers in place the fundamental parts of the system are ready as shown in Fig. 3-5. Nonetheless there are more parts that have to be designed. The next section will present solutions of how to synchronize between the clusters.

3-6 Synchronisation between the Clusters

Communications inside one cluster are designed to be much faster than communications between clusters. Hence a cluster that does not require any communication with a different cluster is finished earlier with an iteration than a cluster that requires such communications. Without any synchronisation scheme the faster cluster would pull ahead of all the other clusters which results in faulty calculations. To solve this issue different solutions are possible.

The easiest might be a *ready signal* that is set whenever a cluster is ready for the next iteration. A cluster is ready when it sent out all its data and received all the data it needs from other clusters. This signal can be combined using simple or-gates inside the tree and at the top fed back to the clusters. This approach is very fast but requires two extra wires in the tree. Besides there is no control over the "start a new round" signal. The last point can be solved by adding a controller which fetches the ready signals of the clusters and issues a new round signal at a suitable time.

Another possibility is the use of *a special packet that is routed to the root of the tree* where it is counted. After enough done packets have arrived at the root a start new round packet is issued. This approach has the advantage to utilise the existing infrastructure.

A different approach is to *not use synchronisation* but to incorporate that some clusters are finished earlier than others. The results of the faster clusters can be buffered by the slower clusters and used at the right time. This approach however has a higher complexity and does not have any advantages to the system that is designed to run in real time. For pure simulation it might be interesting when the fast clusters can be reused for a different task after they finished the required amount of iterations. These possibilities are not further evaluated in this work however.

Thus, the following three possibilites are discussed here:

1. Using a ready signal
2. Routing a special packet to the root of the tree
3. Not using synchronisation

As the second approach is reusing the existing infrastructure, and the performance is comparable to the first approach, it is used henceforth. The last approach is more complex and does not have any advantage for the current system. Nonetheless for a task based system where the cluster can calculate a different task after they have finished the last approach might be interesting.

An important aspect for a biologically interesting system is the number of cells. Any single FPGA is limited to a certain number of cells due to hardware constraints. While the system designed in this work promises good scalability to a high number of cells it is of little use when only one FPGA can be in the system, as the massive number of cells required can not be reached inside one FPGA. The following section will characterize the system in respect to multi-FPGA implementations.

3-7 Adjustments to the Network to scale over multiple FPGA

While the system up to this point works very well on one single chip this one chip is limited in size. Even with thorough optimization of the system at some point the chip is full. This issue can be solved by allowing for multiple chips in one large system, which results in new challenges. The connection between the different FPGA is much less parallel than the connections inside the FPGA. The speed largely depends on the techniques used for transmissions between FPGA and can vary from faster than a transmission between routers inside the FPGA for very sophisticated interconnects to many times slower for simple or low power interconnects. For this section it is assumed that the inter-FPGA connection is up to 32 times slower than the communication between two routers inside the FPGA. Furthermore, each FPGA can be connected to only four other FPGAs at the same time. A more detailed description of these parameters can be found in Chapter 5.

Fortunately almost no communication happens between the FPGA as communication that crosses FPGA boundaries is rare, which follows from the cell connection scheme. Thus, the overall system is well suited for scalable multi-FPGA implementations. The tree topology on the other hand is not suited for multi-FPGA systems. While adding another tree layer on top of each FPGA promises easy extendability the limited connection possibilities of each FPGA does not allow for it. Extra FPGA need to be used just for routing between the FPGAs containing the cluster. As a consequence a different topology is advised to connect the different FPGAs. A ring topology promises good performance for this usage scenario. As communication between neighbouring FPGAs in the ring is already rare, communication between FPGAs that are further apart is even less likely. Furthermore, the ring topology generation and administration of the routing tables is less complex compared to a mesh or different topology. Henceforth a ring topology is used to connect the FPGAs.

Nonetheless synchronisation between the clusters is still necessary especially for multi-FPGA implementations. Assuming that one of the FPGAs contains a controller that handles all the synchronization packets. Then the synchronisation packets, described in Section 3-6, need to cross multiple FPGAs to reach that one master. In large systems this results in a large impact on iteration performance as waiting for the synchronization packet becomes the dominant time consumer. Therefore a different synchronization scheme has to be used. One of the FPGA or an extra FPGA is used as master. All FPGAs in the system are connected to this master via two wires. Using this connection a FPGA can signal a finished iteration immediately. This signal does not have to cross multiple stages and the run time is constant for any number of cells. The master FPGA in turn issues the new round signal when adequate. While this approach requires two extra wires as well as a dedicated master FPGA it provides much better scalability and performance compared to the packet based approach for inter-FPGA implementations. The complete system is pictured in Fig. 3-7.

Currently the system is started once and then runs without any interference from the outside world. For a usable system there has to be a way to input data from, for example, an Analog-to-digital converter (ADC) and output data via a Digital-to-analog converter (DAC). The next section will present the design of the interfaces to the outside world.

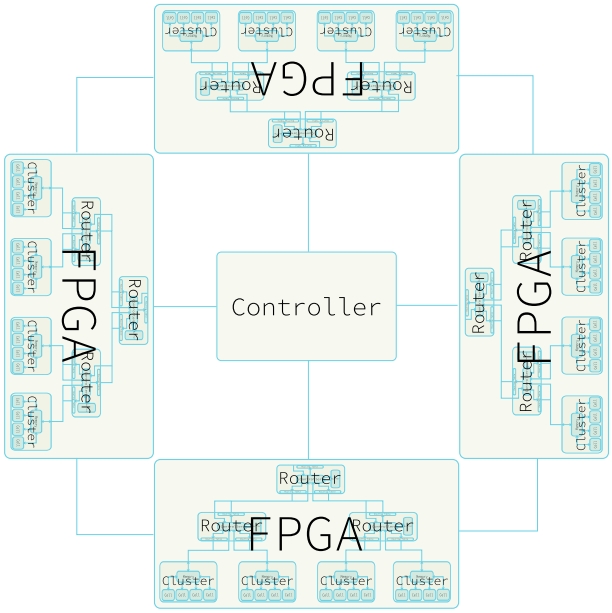


Figure 3-7: This figure shows the multi-FPGA system described in Section 5-3. The single FPGA (Sections 3-3 and 3-5) implementations are connected using a ring topology network. The FPGA are synchronized via a central controller with two wires. One of the wires indicates when the corresponding FPGA is done and the other wire is used by the controller to indicate a new iteration.

3-8 Interfacing the outside world: Inputs and Outputs

The network design is crucial in regard to performance. However, just having a system without any possibility to interact with it, is of little use. This section describes possibilities to design interfaces to the outside world.

In this case the outside world are analog signals that are converted by ADC and DAC. While the system uses 64 bit floating point precision, these converters typically only have 8 bit to 12 bit precision. Thus, the interface has to convert the numbers coming from or going to the converters to the right ranges and input them into the system. The straight forward way of input and output connections are dedicated wires to certain clusters. While this solution is easy to implement and fast it has severe disadvantages when it comes to flexibility. The clusters which receive inputs and outputs have to be predetermined and can not be changed during run-time. A better alternative is using the packet system. The converters can be attached to the top of the tree. Each iteration the converters send a packet to all cells that require the input and receive a packet from all cells that are supposed to output data. This solution is certainly slower than direct connection to the cells but offers superior flexibility. As most of the communication cost of the packet based approach can be hidden behind the cell calculations which results in negligible performance drops compared to the direct wire approach, the packet based approach is used furthermore. The input/output component is pictured in Fig. 3-8.

With the interfacing in place the mandatory systems are designed. The next section describes the design of a system that enables the user to run-time configure the whole system including the creation and destruction of inter cell connections.

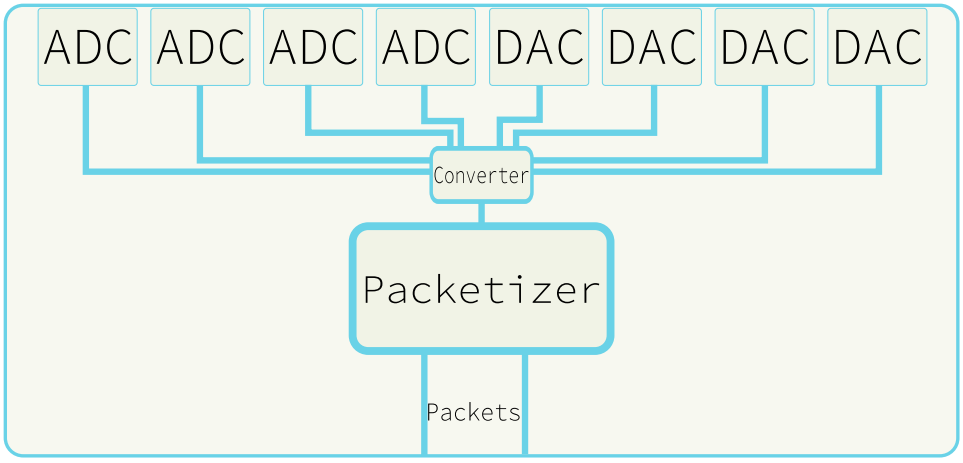


Figure 3-8: Diagram of the interface of the proposed system to the outside world. The system is used to enter analog data into the system and forward the data to specific cells. In return it receives data from the system and transmit the data back to analog signals.

3-9 Adding Flexibility: Run-time Configuration

The system so far is static as no parameters can be changed during run-time. For each change in connection scheme or calculation properties the whole system would have be synthesized with different parameters and reprogrammed again. As synthesizing takes time ([27] mentions 10 minutes for a simpler design), experimenting with different connection schemes and calculation parameters gets tedious. This section describes the design of a configuration bus to allow for run-time change of parameters. Via the configuration bus each element in the system needs to be addressable. Thus, each router, cell, cluster and controller needs an address. Due to the high number of components most commonly used bus systems like Wishbone or Serial Peripheral Interface (SPI) are not applicable. Moreover these buses require a lot of wiring to address every component in the system. On the other hand a well understood bus system is desirable to interface the system with the outer world as it makes interfacing with the system easier. Inside the system it is desirable to follow the tree structure of the NoC. Traversing the tree all components can be addressed, going along the routers down into the clusters and from there into each PhC. Thus, a widely known bus like SPI or a serial interface like Universal Asynchronous Receiver Transmitter (UART) interface the system to the outside world while inside the system a custom tree based bus is used. The easiest way to design the configuration bus is to use the NoC that is already in place. While easy this approach has the disadvantage that the NoC is not designed to address anything but cells. Thus, the address space needs to be increased and the overall performance of the system would decrease. A favorable approach is to use an extra bus that needs to be small. Throughput is of less interest as all configuration should happen while the system is paused and the amount of data transmitted via the bus is low with the largest transmissions being parameter changes of a cell which consists of the address and 64 bit for the address.

The run-time configuration bus is the last system designed. The following section summarises all the configuration parameters that are present in the system to illustrate the

parameters of the design that have to be explored using the SystemC simulation.

3-10 Parameters of the System

The designed system has many parameters. These parameters should be controllable without large changes to the system to enable for simulation of different scenarios. First of all there is the amount of cells. The full amount of cells is determined by the amount of cluster and the amount of PhCs in each cluster as well as the amount of time-shared cells in each PhC. For the NoC the most important parameter is the fan-out of the routers. Furthermore, it has to be determined how the cells in the system are connected logically. All of these parameters determine the final system structure. Additionally there are many more parameters that specify details of the system. To allow for configuration by a human or automated simulation operator without the need to recompile the simulation the system structure should be specified during run-time. The human readable file format used is JavaScript Object Notation (JSON) as it has a large user base and many libraries are available to read and manipulate the format. Specifying the connection between cells by hand is tedious. As all connections between cells are possible a system of 1000 cells has $1000 \times 1000 = 1000000$ possible connections. When considering that the cells are connected using probability patterns it becomes clear that the task to create these adjacency matrices needs to be automated. The following section describes the script that is used to generate system configurations.

3-11 Connectivity and Structure Generation

To ease the generation of configurations for the system a script is used that automatically generates the JSON files. The script has defaults for all parameters so that only the parameters that are of interest for the current experiment need to be specified. The most important feature is the support for different connection schemes. The cells can be aligned as 2D or 3D mesh. The distance calculation that is used to determine if two cells are connected can be changed and ranges from geometrical distance to more complex distance calculations that favor connections that go into a certain direction. Half-normal distribution is used to determine if a certain connection is present or not according to the distance calculated with the previously mentioned distance calculation method. Furthermore, the connections can be visualized as a Graphviz graph. The script is implemented in Python.

3-12 Conclusion

The system consists of clusters that contain multiple PhC around a shared memory. Using the clusters communication between closely connected cells is localised. Network congestion is avoided. The cluster are connected using a tree topology NoC. The tree topology is chosen to reduce the hope count between neighbouring neurons to reduce latency on

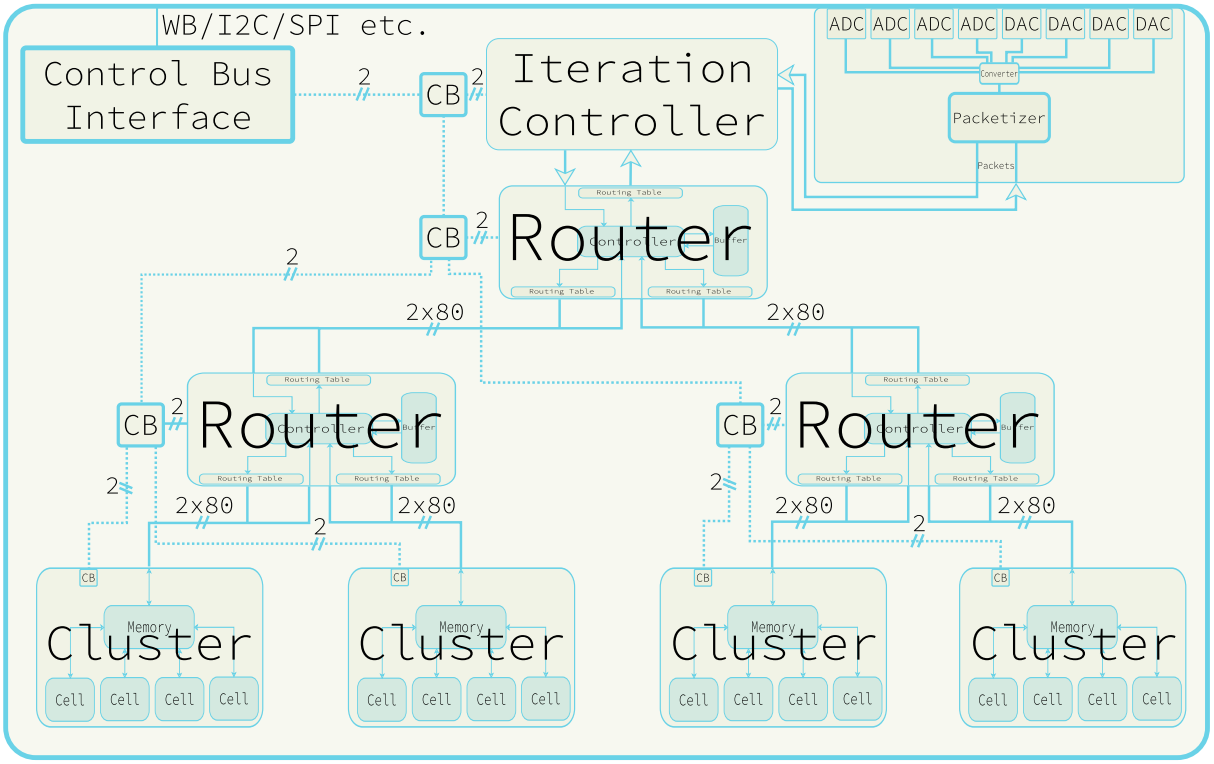


Figure 3-9: Full view of the system as design in Chapters 3 and 4 and as evaluated in Chapter 5.

very frequented parts of the network. This system structure is responsible for the linear scalability of the system by exploiting communication locality.

The system can be configured during run-time using a configuration bus. This bus can be used to change system parameters during run-time. Therefore, increasing the flexibility and adaptability of the system by reducing the need for resynthesizing the system for minor changes to the connectivity or calculation model.

A full view of the system with all components in place is given in Fig. 3-9. While this chapter presents the design decisions that lead to the system design the following chapter characterizes the implementation details of the system in SystemC.

Chapter 4

System Implementation

This chapter presents the details of how the design that is devised in the last chapter is implemented. The implementation is done using cycle accurate SystemC to simulate the hardware while aiming at flexibility to explore the design space and accuracy to allow for meaningful conclusion for a hardware implementation. The chapter is separated into the different parts of the system containing of the clusters, routers, round controller, the real-time control bus and the input/output part. Furthermore, the details of the configuration generation script are presented.

4-1 Exploiting locality: Clusters

The cluster implementation is based on the PhC design implemented in [27] and described in Section 2-2. Each PhC calculates multiple neuron cells time shared as the brain real-time of $50\,\mu$s is slow compared to the clock speeds of modern hardware. The design is changed in a way to better suit the new system structure. In the original system each PhC first completed calculations before proceeding to interchange information with other PhC. For the clusters both types of operations are done in parallel. This is possible as not all the calculations done in a PhC require information from other cells. For implementation details of the PhC please see [27] as this work does not focus on the calculation itself but on the interconnection between cells given the communication pattern.

As shown in Fig. 4-1 the PhCs are grouped around a shared memory. The shared memory is in dual port configuration offering two read and two write ports. Furthermore, the shared memory is double buffered. All writes are done to one memory while all reads are done from another memory whereas after each round the two memories change their task. This is necessary as the algorithm is iteration based and the data for the next iteration needs to be written to the memory while the data for the current iteration is still needed for the calculations. The PhCs time share one port of the dual port memory. Every PhC gains read and write access to the memory using round-robbing. Thus, a PhC has to wait for n clock cycles to gain memory access, where n is the number of PhCs in a cluster.

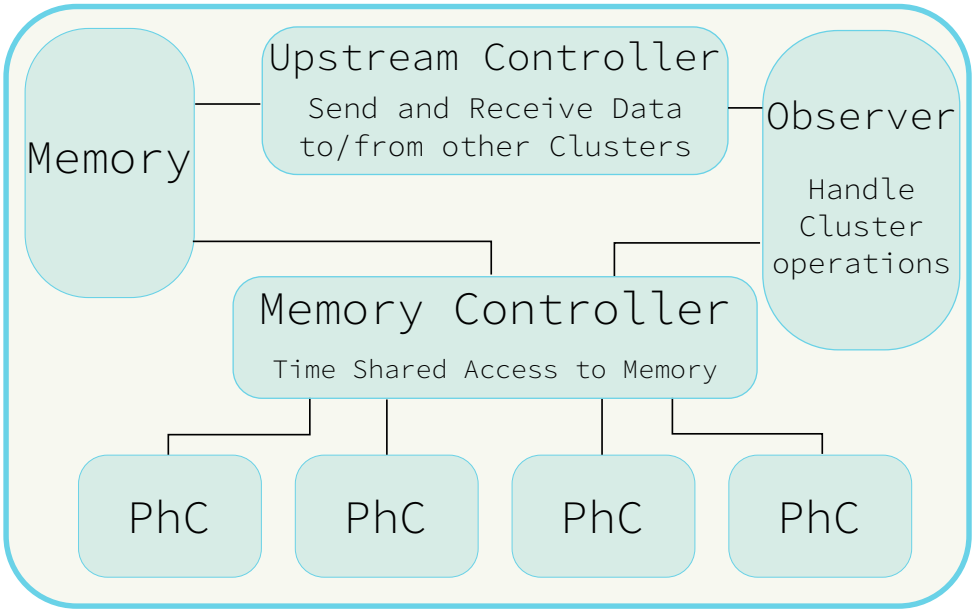


Figure 4-1: A cluster as used in the proposed system. The cluster allows for instant communication between any cells that are located inside the cluster. Each of the PhC time shares the calculation of multiple cells. The data of the cells that might be needed by other clusters is forwarded into a NoC.

The cluster is controlled by three controllers. The first controller handles all the book keeping. This includes monitoring the start signals that are received from the NoC and originate from the round controller. Furthermore, the first controller observes when the PhCs are finished with calculating results. After all data for the next iteration is present in the memory, the book keeping controller issues a done packet to the round controller.

The second controller handles everything related to the NoC. It observes which packets arrive from the NoC and stores them into the memory using the second memory port. Furthermore, the second controller informs the book keeping controller that all needed packets arrived. The last controller handles everything that originates from the PhC. It sends all packets upstream into the NoC and controls the round-robbing memory access. Besides, the third controller stalls the calculation if the upstream router is not able to receive more packets.

The upstream NoC consists of routers that are responsible to forward the packets originating from the clusters to the correct receiving clusters. The following section describes their inner workings.

4-2 Connecting Clusters: Routers

The main goal in designing the routers is fast processing of packets and a low hardware utilisation at the same time. As shown in Fig. 4-2 every input of the router consists of First in, First out Buffers (FIFOs). The depth of these FIFO has to be determined by simulation while keeping the memory utilisation in check. Each of the FIFO is one packet wide. The input FIFO are read in order one after the other in a state machine. After reading a packet

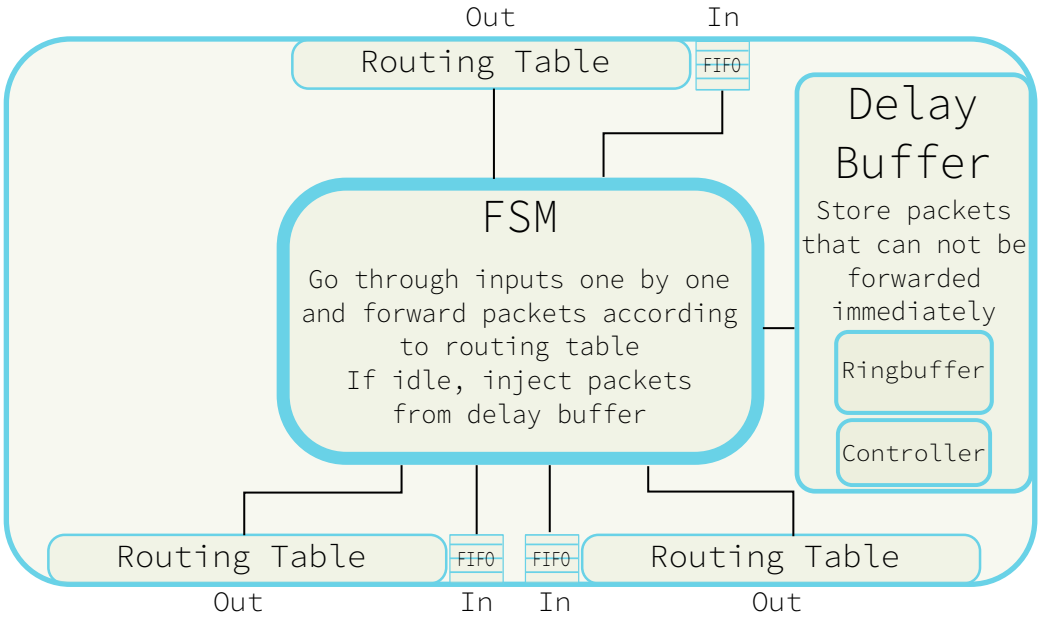


Figure 4-2: Diagram of a router as used by the proposed system. The routers are arranged in a tree topology. Each router has n children and except for the root router one upstream router. The router reads input packets from the input FIFO. Based on routing tables it determines where the packets have to be forwarded. If a receiving FIFO is full the packet is placed in the delayed buffer to be forwarded when the receiving router is free again.

from a input FIFO the source of that packet is checked in the routing tables of the outputs. If the bit corresponding to the packet source is set in a routing table the packet is forwarded in that direction. As no packet can be dropped, packets that can not be forwarded right away, as the receiving FIFO is full, are instead stored for delivery at a later point in time. These delayed packets are stored together with the corresponding output. The width of this delayed buffer is $b_p + \lceil \log_2(n_o) \rceil$ bit, whereas b_p is the amount of bits for a packet and n_o is the amount of outputs of the router. The depth of this delayed buffer needs to be determined by simulation, but as shown in Chapter 5 a buffer size of 240 can be achieved without additional hardware costs. In Chapter 5 it is also shown, that a depth of 240 is suitable even for high usage scenarios with all-to-all connections. New packets on the input FIFO always have precedence over the delayed packets. Only when no new packets are waiting in the input FIFO the delayed packets are forwarded. This is done to avoid cases where the router tries to deliver delayed packets to full routers again and again. Comparisons between an implementation that handles delayed packets after every iteration through all input FIFO and the is presented in Chapter 5. The routing tables are generated during system initialisation from the adjacency matrix that describes the cell connections. The process of routing table generation is described in Section 4-6.

The root router of the tree is connected to the round controller which controls the iterations of the system. The next section will describe the implementation of this controller.

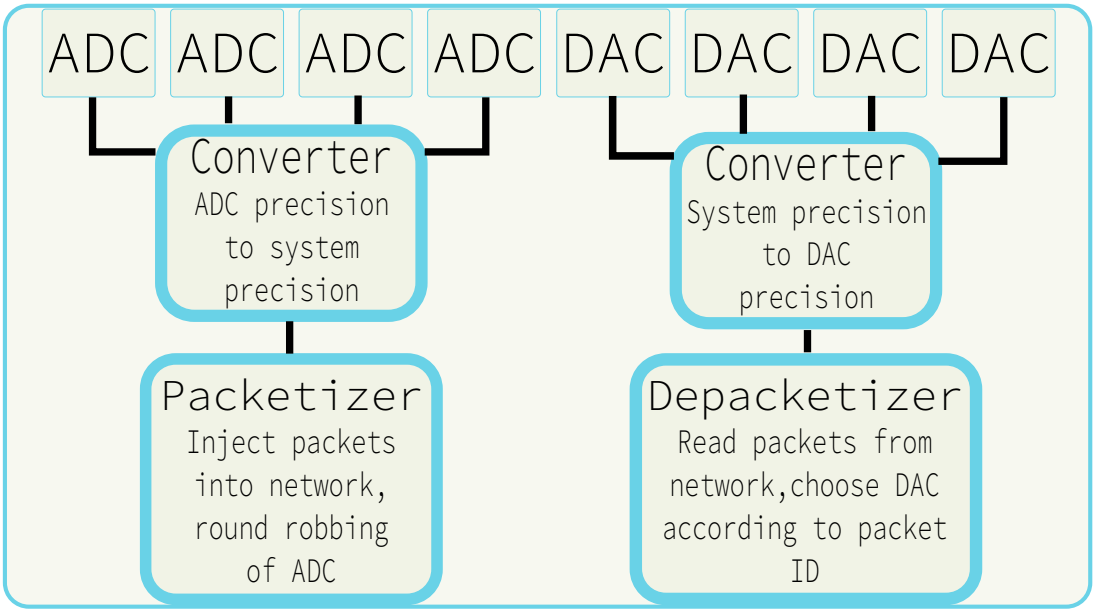


Figure 4-3: Part of the system that interfaces with the outside world. The date from the converters is put into packets and routed down the tree to the correct clusters. In return clusters can send packets up the stream to the output converters.

4-3 Tracking Time: Iteration Controller

The iteration controller is needed to keep all cluster in the system in sync. It is not necessary for the iteration controller to identify that a specific router is done, but only that all clusters are done. After a cluster finishes a round, which includes completing all its calculations and receiving all data for its next round, the cluster sends a done packet. The done packet is sourced as coming from cell a virtual cell that lies after all the cells in the system. Any packet that originates from this source is routed upstream to the iteration controller. The iteration controller counts any packet that originates from this source. After the controller received as many packets from this source as there are cluster in the system it will issue a packet with a source lying one after the done packet. This packet is in return routed to all clusters and interpreted as a round start signal.

Next to the iteration controller, at the root of the tree, the system to receive information from and route information to the the outside world is located. The next section presents the implementation details about that system.

4-4 Inputs and Outputs

The input and output module is used to interface the system with the outside world as shown in Section 3-8. In this case the outside world are ADC and DAC. Where the data of these converters stem from is of less interest to this work. It is assumed that the converters each have a n bit interface and the system can read from, respectively write to these interfaces. The amount of inputs and outputs is not predetermined and depends on the

actual system implementation. For this document it is assumed there are eight inputs and eight outputs. The first step is to convert the n bit data from the inputs to system precision. Each of the inputs receives a unique ID starting from $n_c + 2$ (n_c being the number of cells in the system), thus after the start and stop packets. Each system iteration these packets are injected into the system. In return the outputs receive packets from specific cells which are converted to DAC precision and outputted.

Another interface to the outside world is the control bus which is used to control the system during run-time. The next section describes the implementation of the control bus.

4-5 The Control Bus for run-time Configuration

The control bus can be used to control the system during run-time. The outside world is interfaced using widely known buses like I2C or SPI. Inside the system a custom tree based bus is used to utilise the tree based NoC structure and allow for easy addressing of any component in the system. The bus consists of two wires, one to indicate that the bus is active and one for transmitting the data. Furthermore, the main clock signal is used. A command consists of two parts, the first part is the address and the second part the payload. Correspondingly the bus first opens up a connection to a specific component using the address and then forwards the payload to the component. The payload itself consists of the command that has to be executed and optionally data. Each component in the system, like a router or a cluster, has an attached control bus router. This control bus router can either forward the bus signal to any of its children or forward the bus signal to the attached component.

The first step to send any control signal is to send the complete data, including address and payload, over the outside world bus. After receiving the complete bus data the bits are injected in order onto the control bus router at the top. When a router receives an active bus signal it will read the first bit on the bus. A high signal indicates that the component corresponding to the router is addressed whereas a low bit indicates that the router has to forward the signal to any of its children. If the signal is low the next bits indicate to which child the router should forward the bus data. After the router has been set up for forwarding either to its component or to its child it will proceed to forward bus data until the bus active signal is low again. The working of a router is pictured in Fig. 4-4. Thus, the addressing follows strictly the structure of the system which allows for efficient address parsing.

After setting up all routers the addressed component receives the bus data. This first 8 bit that the component receives are the command that shall be executed. Currently there are 3 commands:

1. Stop simulation - Only valid when round controller is addressed
2. Replace one bit in routing table - Only valid when router is addressed
3. Replace any concentration level - Only valid when cluster or PhC or cell is addressed

Figure 4-4: Control bus router operation in dependance of its two input signals bus_active and bus_data. The control bus is used to control the system and change parameters of any component of the system. It is realised as a tree following the structure of the network.

The commands are kept simple on purpose and more complex commands like creating a new connection between two cells have to be created by combining multiple simple commands. Each command might require additional data. Command 2 requires the routing table that has to be changed, the entry that has to be changed and the new value for that entry. Command 3 requires the concentration that has to be changed and the new value and 1 does not require additional parameters.

In addition to the two wires used for bus activity and data there might be a third wire that can be used to stall the bus. The stall signal is used when more data is expected from the outside world but has not yet arrived. It is preferred, though, to first buffer all the data from the outside world and then start sending to avoid incomplete commands.

The last section in this chapter describes how all the components are integrated together inside the simulation.

4-6 Automatic Structure Generation and Connectivity Generation

To keep the system simulation flexible a configuration file format is used as described in Sections 3-10 and 3-11. The configuration file is used to generate the structure of the system which consists of initialising all components with their respective initialisation data and connecting them. When the simulation starts the configuration file is either read from a file or from the standard input. The JSON data is then parsed by the Json-CPP library. For any value that is not specified by the configuration file a default value is used. The parsed data is processed and stored in a configuration structure for further usage. During simulation initialisation the system structure is built according to the configuration structure. While building the structure the parameters each component needs are generated. Routing tables are generated in this step considering the position of the router in the tree topology and the adjacency matrix. All components are finally connected using FIFO or signals as necessary.

4-7 Conclusion

This chapter presented implementation details of the system designed in Chapter 3. Details for all the components of the system were given. Apart from the main components of the system, the clusters and routers, the interface to the outside world as well as the synchronisation controller were described. Furthermore, an example of how the system structure is automatically generated was given. The following chapter presents evaluations about the performance of the system. Additionally, comparisons and justifications for various design parameters are given.

```
> ./generate_config.py -c 16 -s 2 -d normal
{
  "CELLS_SHARED": 2,
  "CLUSTER_CELLS_PHYS": 2,
  "CONECTIVITY": 0.3583333333333334,
  "NEIGHBOURS":[
  [0,1,0,1,0,0,0,0,1,1,1,0,1,0,0,0],
  [0,0,0,0,1,0,1,0,0,0,0,0,1,0,0,0],
  [0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,1],
  [0,1,0,0,1,0,1,0,0,0,0,0,1,0,0,0],
  [0,0,0,1,0,0,1,0,0,0,0,1,1,0,0,0],
  [0,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0],
  [0,0,0,1,1,0,0,1,0,1,0,0,1,1,1,0],
  [0,1,0,0,1,1,1,0,0,1,0,0,1,1,0,0],
  [1,0,1,0,0,0,0,1,0,0,1,0,1,0,1,0],
  [1,0,1,1,1,1,1,0,1,0,1,1,1,0,1,0],
  [0,0,1,1,0,1,0,0,0,0,0,1,1,0,0,0],
  [0,0,0,0,0,0,0,1,1,1,1,0,1,0,1,0],
  [1,0,0,0,1,0,0,0,1,1,0,0,0,0,1,1],
  [0,0,0,1,0,0,1,1,1,1,0,1,0,0,0,1],
  [0,1,1,0,0,0,1,0,1,0,0,0,1,0,0,0],
  [0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0]]
"ROUTERS": [
    [0,0],
    [0,0]
  ],
  "SIMULATION_STEPS": 5
}
```

Figure 4-5: Configuration file example. "CONNECTIVITY" indicates how many connections of all possible connections are present, whereas one cell can not be connected to itself. "ROUTERS" presents the structure of the system. Arrays represent routers and '0' represents cluster. "NEIGHBOURS" is an adjacency matrix representing the connections between cells in the system. "SIMULATION_STEPS" is the number of iterations that shall be run before the simulation stops. These are common parameters but the configuration format supports additional features.

Chapter 5

Evaluation

This chapter presents different simulations to assess the performance of the proposed system. First various design decisions are justified. Afterwards the system is compared to a baseline version designed in [27] and an optimal system for both a single chip and a multi chip implementations.

All simulations are done using the proposed system implemented using cycle accurate SystemC. The configurations used to run the simulations are automatically generated using the configuration generator as described in Section 3-11. Three different connection schemes are used to simulate the system behaviour; all-to-all connections, normal distributed distance based connections and neighbour based connections. These schemes are further explained in Section 5-2. As the distance based connection scheme is based on normal distribution and not all configurations need to have the same amount of connections the test is run 10 times with different configurations and the average of all the runs is presented here.

For each simulation all outputs of the simulator are stored and later parsed by a script to retrieve the required information. For most of the tests the number of cycles per iteration is used to determine the performance of the system. This number represents the number of clock cycles the system needs to perform one step of the calculation. To achieve brain-real-time the number of iteration cycles has to be below a certain threshold depending on the clock frequency. Assuming a clock frequency of $100\,\text{MHz}$, which corresponds to a cycle time of $10\,\text{ns}$, as used in [27] and a brain real-time limit of $50\,\mu s$ [26] the maximum number of cycles for one iteration step is $\frac{50\,\mu s}{10\,\text{ns}} = 5000$ cycles.

This chapter begins with the justification of design decisions, followed by presenting the performance of one chip and multi chip implementations and closes with the estimation of the hardware utilisation of the system.

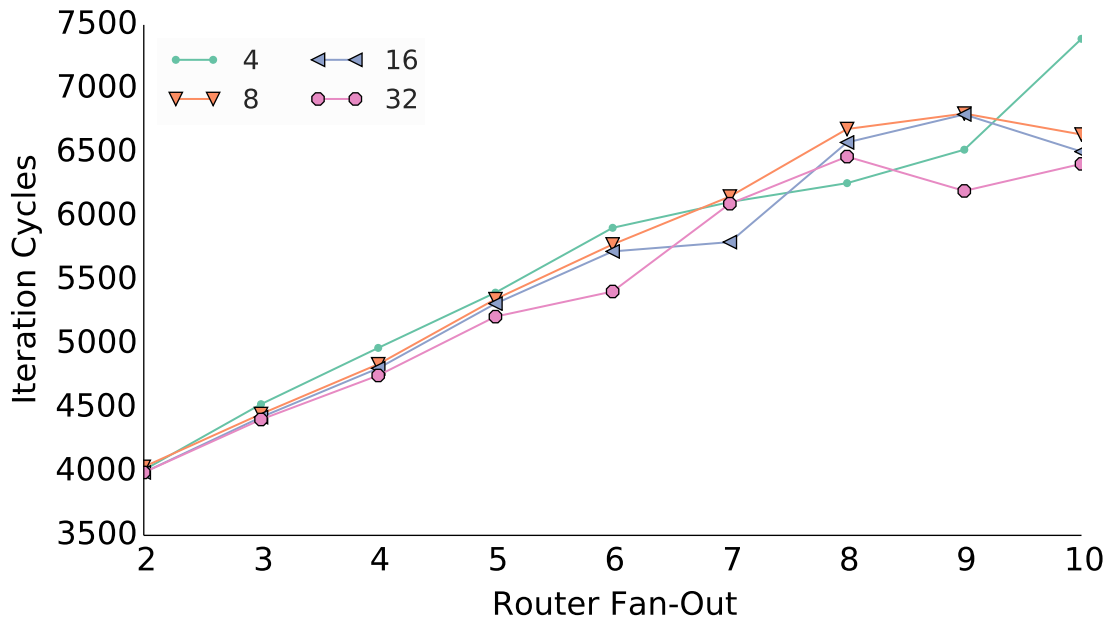


Figure 5-1: Comparison of the system's iteration performance for router input FIFO depths of 4,8,16 and 32 elements. All clusters have 2 PhC and the router fan-out are varied between 2 and 10.

5-1 Design Space Exploration

During system design various assumptions about the performance of certain parts of the design are made. This section justifies those decisions using the SystemC simulation. All times presented in this section do not include calculation times and thus compare network performance.

The first design decision presented here are the two important buffer sizes of the routers. Firstly different FIFO sizes at the inputs are compared and additionally the feasibility of the delayed buffer is shown; the delayed buffer does not grow uncontrollably.

The first simulation run is done using a system with 512 cells and varying router fan-out. All clusters consist of 2 PhC with 8 shared cells per PhC, thus, a total cell count of 16 per cluster. The comparisons use the all-to-all connection scheme to simulate the worst case. Figure 5-1 presents the simulation results for different router input FIFO depths for fan-out from 2 to 10. For router fan-out less than 5 all FIFO depths perform comparably well. Larger router fan-out profit from higher FIFO depths.

The second important aspect of the router design are the buffer depths. These buffers are used to avoid dropping packets when a router is under high use. The size of the delay buffers has to be chosen high enough to be able to deal with any connection scheme. Figure 5-2 presents the required buffer sizes for the all-to-all connection scheme. The amount of cells in the system is varied between 256, 512 and 1024. Furthermore, the fan-outs are varied between 2 and 10 and the FIFO depths are kept at 4. The simulation results suggest that the two factors counteract each other. On the one hand low fan-out results

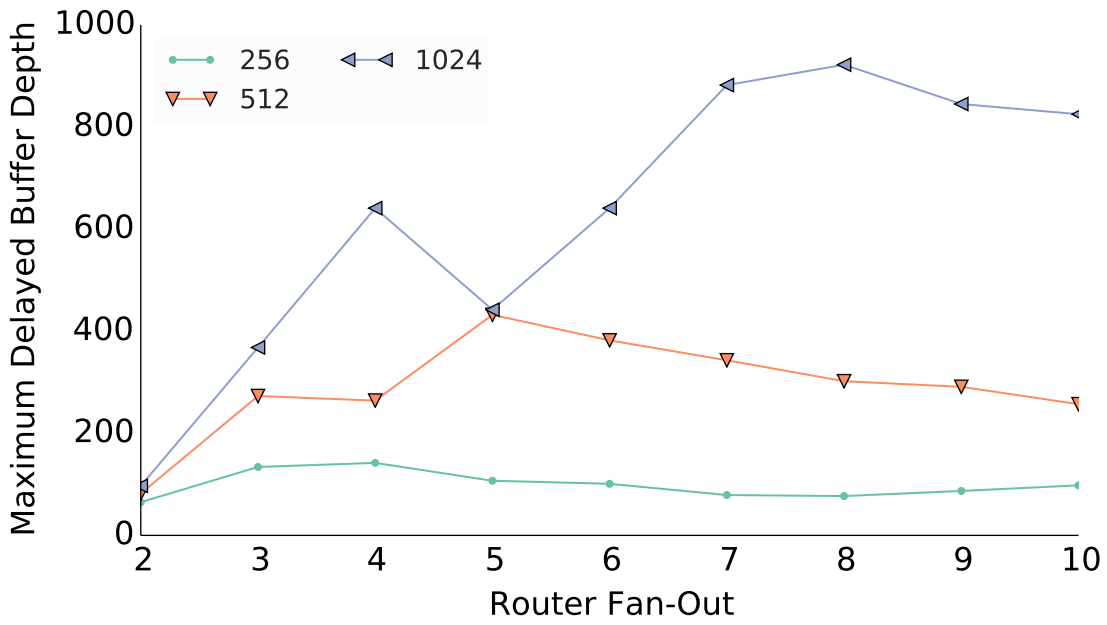


Figure 5-2: Comparison of maximum delayed packet buffer depths for different router configurations. This graph shows that the delayed buffer strategy is feasible without large delayed buffers for lower router fan-outs. Clusters are kept at two PhC with 8 shared cells. The number of cells and thus clusters and routers in the system is varied between 256, 512 and 1024.

in fast routers but more routers are required in the system. On the other hand slower routers with large fan-out result in less routers in the system but each router requires a bigger delayed buffer. A fan-out of 2 results in the lowest size for the delayed buffer. Very high fan-out promise less routers and thus less hardware but at the cost of performance. According to this and the previous simulation results in Fig. 5-1 a fan-out of 2 is favorable and should be used.

For the router performance the time when delayed packets get injected into the network is crucial. To analyze this behaviour, simulations are carried out using the same conditions as for the previous example for different fan-out with 512 cells and two PhC with eight shared cells per cluster. Three injection models are compared. For the first approach packets are injected after each run through all input FIFO, for the second approach packets are only injected if the FIFO have been empty for a complete round and the third approach finally injects packets if there has been no activity on the inputs for 10 rounds. Figure 5-3 presents the results of this simulation. Injection after each iteration is not feasible and results in very long iteration times. The other two approaches perform about four times better while approach two performs best overall. Not only does it provide the best iteration times it also results in lower delay buffer sizes compared to the other two approaches.

While the previous simulations already took the router fan-out into consideration the following comparison highlights the different fan-out in respect of cluster sizes. The simulations are run for 512 cells with 2, 4, 6 and 8 PhC per cluster and 8 shared cells per PhC. Figure 5-4 presents the results of this simulation. While small clusters with small

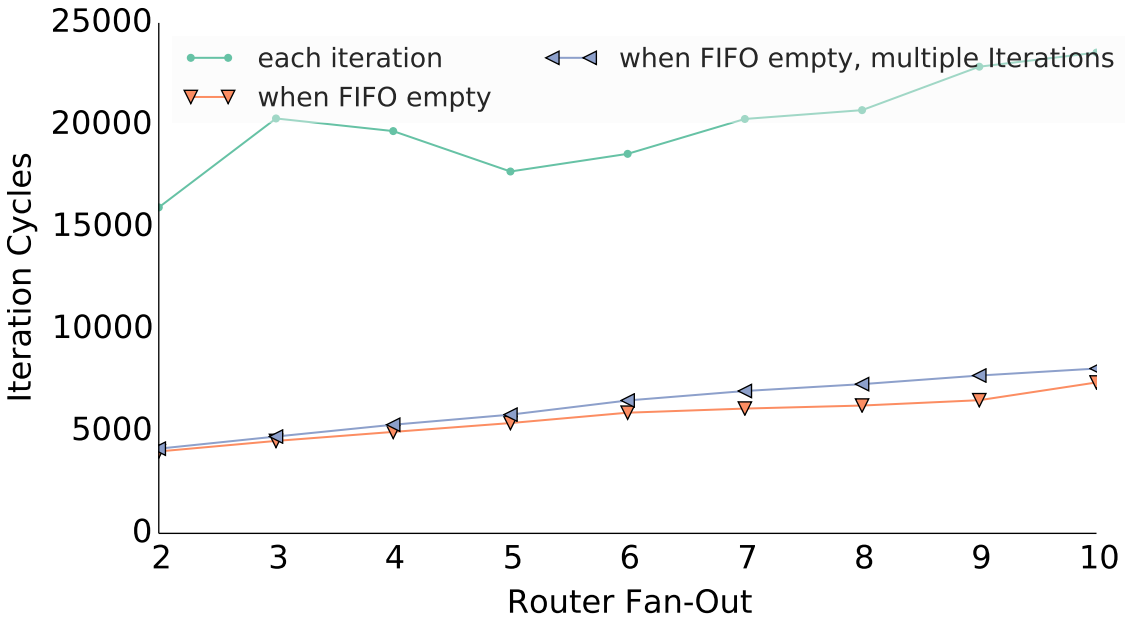


Figure 5-3: Comparison between three delayed packet injection schemes. Injecting the packets after each round through all input FIFO results in the slowest iteration times. Injecting only when all FIFO are empty for one round or for multiple rounds achieves comparable performance but injecting after one round of input inactivity results in the best overall performance.

routers provide the overall best performance, bigger clusters catch up with big fan-out. The overall best performance is achieved by 2 PhC with a router fan-out of 2 with 4012 cycles. The second best result is achieved with 8 PhC and a fan-out of 8, 9 or 10 with 4077 cycles iteration time. Overall the extreme cases with high or low fan-out perform better than the cases in the middle.

The last design aspect to discuss in this section is the choice of the size of a cluster between multiple small clusters or few big clusters. All PhC use 8 shared cells but the amount of PhC in each cluster is varied. All routers are kept at a fan-out of two, which is the optimal fan-out as shown in the previous paragraphs. The simulation is run with cell amounts as multiples of two between 256 and 2048. The results are presented in Fig. 5-5. Due to the fact that all the PhC of a cluster time-share a memory large clusters are expected to be slower than smaller clusters. This is supported by the simulation results which show that 2, 4 and 8 PhC per cluster show no major difference in iteration time. Clusters with more PhC on the other hand perform worse especially at a higher number of cells. The worst performance is achieved by using clusters with 16 or 18 PhC while clusters with 20 PhC perform a bit better. The difference for 2048 cells between 2 PhC cluster and 18 PhC cluster is $30\,\%$ and 20 PhC cluster perform $2.5\,\%$ better than 18 PhC cluster.

This chapter presented simulations about various design decisions. Based on these results the performance comparisons are done for routers with a fan-out of two which showed superior performance for all measurements. Furthermore, the PhC amount per cluster is set to 2, 4 or 8 as bigger clusters resulted in worse performance. Delayed packets will

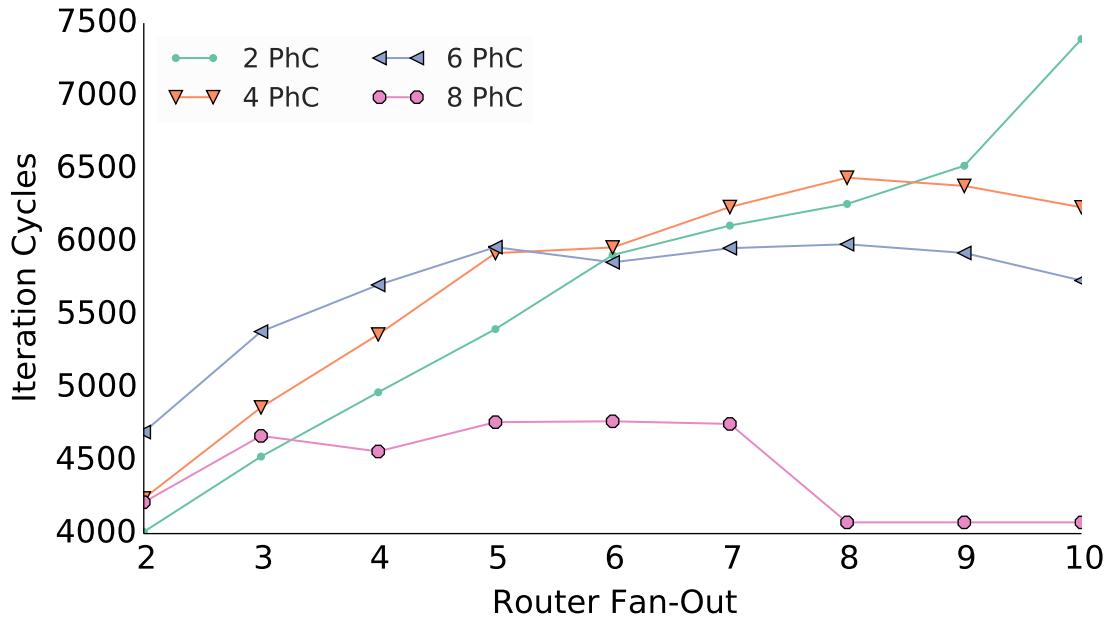


Figure 5-4: Comparison of iteration performance for 512 cells and different cluster sizes. Except for large clusters with eight cells small routers perform better than large routers. The overall best performing configuration consists of small clusters and small routers at two PhC per cluster and a fan-out of two.

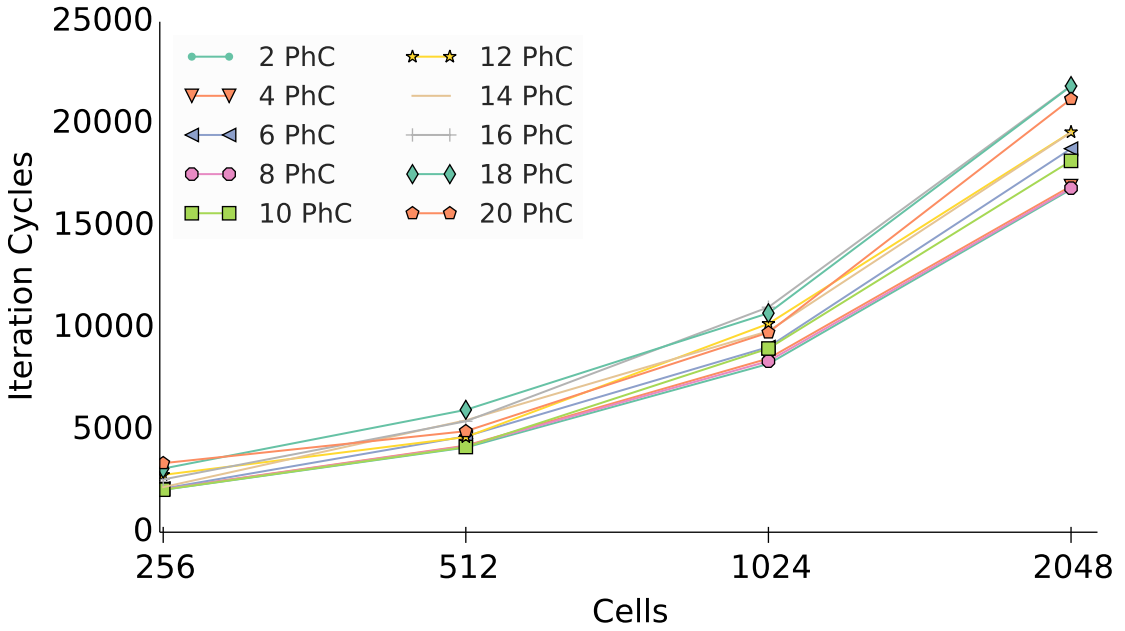
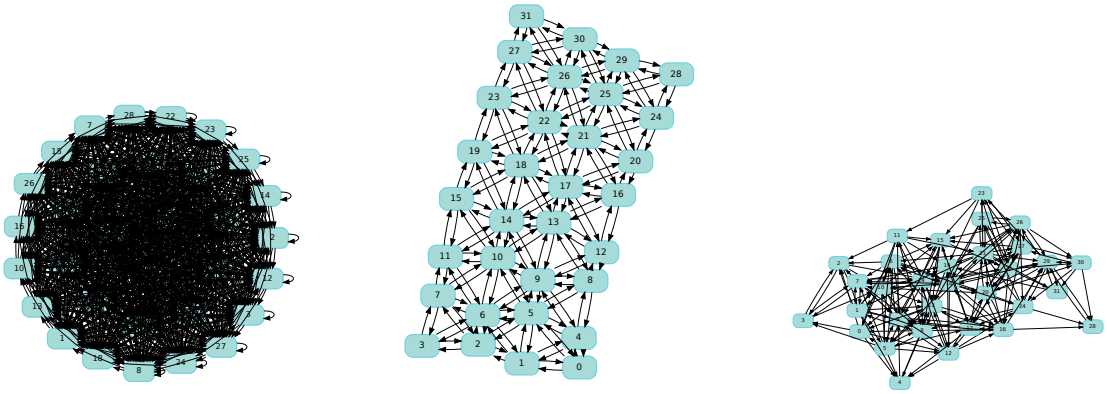


Figure 5-5: Comparison of systems with different cluster sizes. The routers are kept constant at a fan-out of two. Clusters with two, four and eight PhC achieve the best iteration times while larger clusters are generally slower especially at 512 cells.



(a) Complete **(b)** Neighbours **(c)** Normal

Figure 5-6: Examples of the different connection types used to test the iteration performance of the system. While the complete and neighbour calculations are static, the normal configuration is based on the normal distribution and cell distance and is newly generated for each simulation run. The results for the normal distribution are thus averaged over 10 runs to address for faster and slower configurations.

only be forwarded when a router is idle. Using these configurations the following sections focus on the performance of the system in comparison to the system shown in [27]. The following section presents the simulation results for a system on a single chip.

5-2 Performance on one Chip

This section compares the inferior olive simulation system presented in this work to the results presented in [27]. For comparison different scenarios are presented. Initially the optimal performance is determined using an optimal system as shown in Chapter 3. After that the system performance is compared using three connection schemes which are neighbour connection, normal connection and full connection scheme (Fig. 5-6). For the full connection scheme all cells are connected to all other cells. The normal connection scheme uses probability based connections weighted by the distance between cells. Lastly the neighbour connection scheme connects every cell to every neighbouring cell, thus, resulting in 8 connections per cell. The systems are compared showing only network performance as well as full system performance to show how well the network can be hidden behind calculations. For full system performance the cycles needed for all calculations are taken from [27] (528 cycles per cell calculation) as the PhC design has not been changed.

To have an optimal base line, the shared memory optimal system proposed in Chapter 3 is simulated using SystemC. As expected for an optimal system the administration and calculation times stay constant. The administration times amount to 9 cycles per iteration. The simulations shows that including calculation times the optimal system requires 534 cycles per iteration for one shared cell. Increasing the amount of shared cells results in a linear increase in iteration cycles. For 8 shared cells as used in the following comparisons the optimal system requires 4230 cycles.

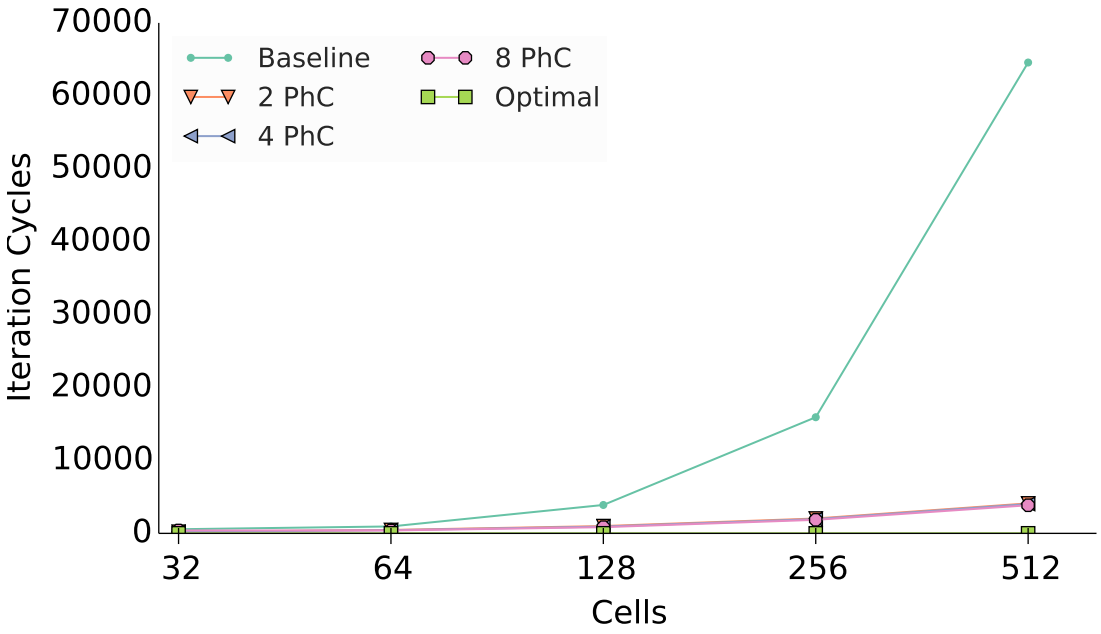


Figure 5-7: Comparison between different cluster sizes for different amounts of cells. The *complete* connection scheme is used and calculation times are *excluded* from the iteration times to show the pure network and administration costs. The systems use a router fan-out of two. The baseline is taken from [27].

The first comparison focuses on iteration performance without calculation times. The system with 2, 4 and 8 PhC per cluster is compared to the optimal approach and the baseline approach from [27]. The results of these simulations are presented in Figs. 5-7 to 5-9. The all-to-all connection scheme already shows the advantages of the newly designed system. While for the baseline approach the time required for each iteration grows exponentionally with the number of cells in the system the proposed system shows linear growth. For twice the amount of cells twice the amount of iteration cycles are needed. The baseline approach requires 567 cycles for 32 cells and 64566 cycles for 512 cells which is an increase of 113.873, the 8 PhC system has an increase in iteration time of 10. The 8 PhC system performs best overall with the 4 PhC and 2 PhC systems lacking behind by $1\,\%$ and $1.1\,\%$ respectively.

Looking at the normal distributed connection scheme shows a clear disadvantage of the baseline approach. Due to the bus based nature of communication inside that system it does not perform any better compared to the all-to-all connection even though less communication occurs. The proposed system performs as expected. While a system with 512 cells has 16 times more cells than a system with 32 cells the proposed system requires only 3.2 times more cycles. For this distribution the system with 2 PhC performs better than the systems with more PhC.

The performance for the neighbour connection scheme is similar. As less communication between cells happen the iteration times decrease slightly. The 4 PhC case performs best and the 32 to 512 cell cycle increase is 1.76.

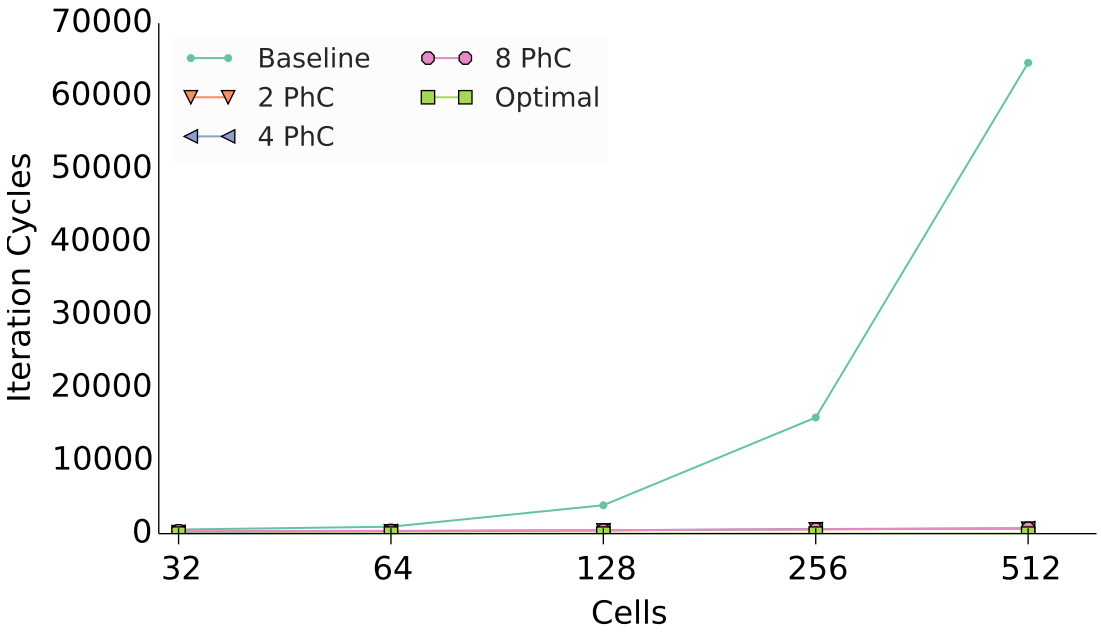


Figure 5-8: Comparison between different cluster sizes for different amounts of cells. The *normal* connection scheme is used and calculation times are *excluded* from the iteration times to show the pure network and administration costs. The systems use a router fan-out of two. The baseline is taken from [27].

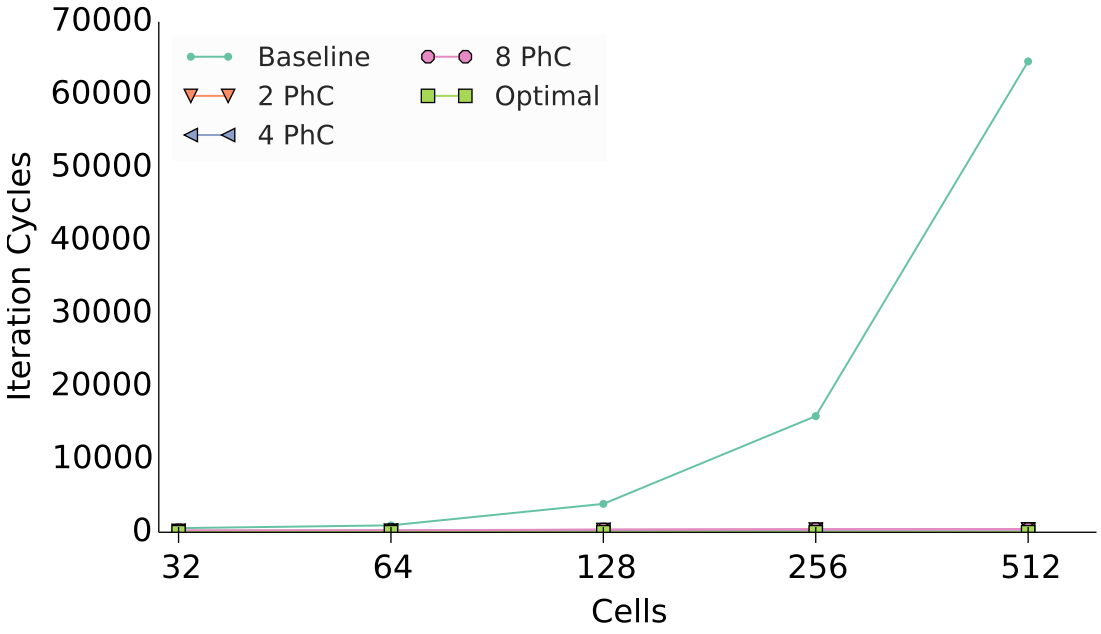


Figure 5-9: Comparison between different cluster sizes for different amounts of cells. The *neighbour* connection scheme is used and calculation times are *excluded* from the iteration times to show the pure network and administration costs. The systems use a router fan-out of two. The baseline is taken from [27].

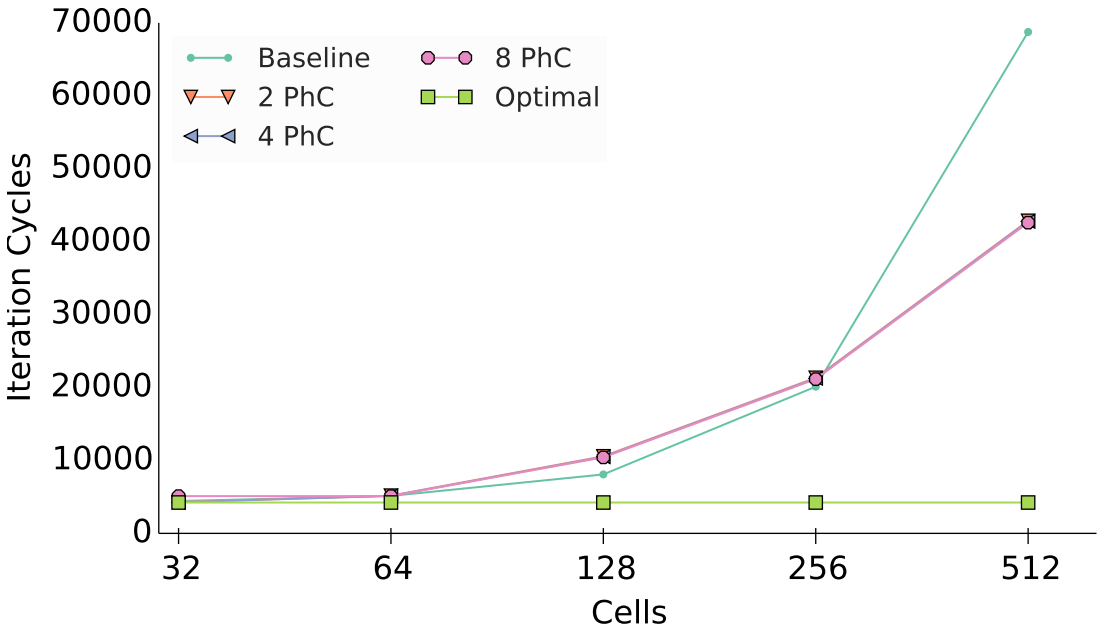


Figure 5-10: Comparison between different cluster sizes for different amounts of cells. The *complete* connection scheme is used and each cell calculation takes *534 cycles*. The systems use a router fan-out of two. The baseline is taken from [27]. For the complete connection scheme the system can not meet the brain real-time. This is to be expected as the structure of the system is not designed for all-to-all connections. Rather it is designed to exploit the communication locality of the other connection modes. Nonetheless, the new system scales better than the baseline.

All of these results show that the proposed network is well suited for the task and performs to expectation. The next paragraphs present the results to simulations with calculation times. These tests show how well the communication can be hidden behind calculations and how well the systems scale in a realistic scenario.

As shown in Fig. 5-10 for small systems with up to 256 cells the baseline system performs better than the proposed system for all-to-all connections. For larger system the exponential growth of the communication bus in the baseline dominates the calculations and the proposed system performs better thanks to the linear growth in communication cost. All three PhC configurations perform similar. For less than 256 cells the proposed system performs up to $30\,\%$ worse than the baseline. For 512 cells the proposed system performs $60\,\%$ better than the baseline.

The picture changes completely when looking at the realistic cases presented in Figs. 5-11 and 5-12 where the proposed system can show its strength. For both scenarios all three PhC configurations perform very close to the optimal configuration. At 512 cells the difference between the optimal approach and the 2 PhC system is $9.8\,\%$ and $5.9\,\%$ for normal and neighbour connection schemes respectively. The absolute difference in execution time for 32 to 512 cells is 354 cycles and 191 cycles in the worst case (2 PhC system).

These simulations show that the system is very feasible for one chip systems. But even with good optimisations the amount of clusters per chip is limited. For large systems

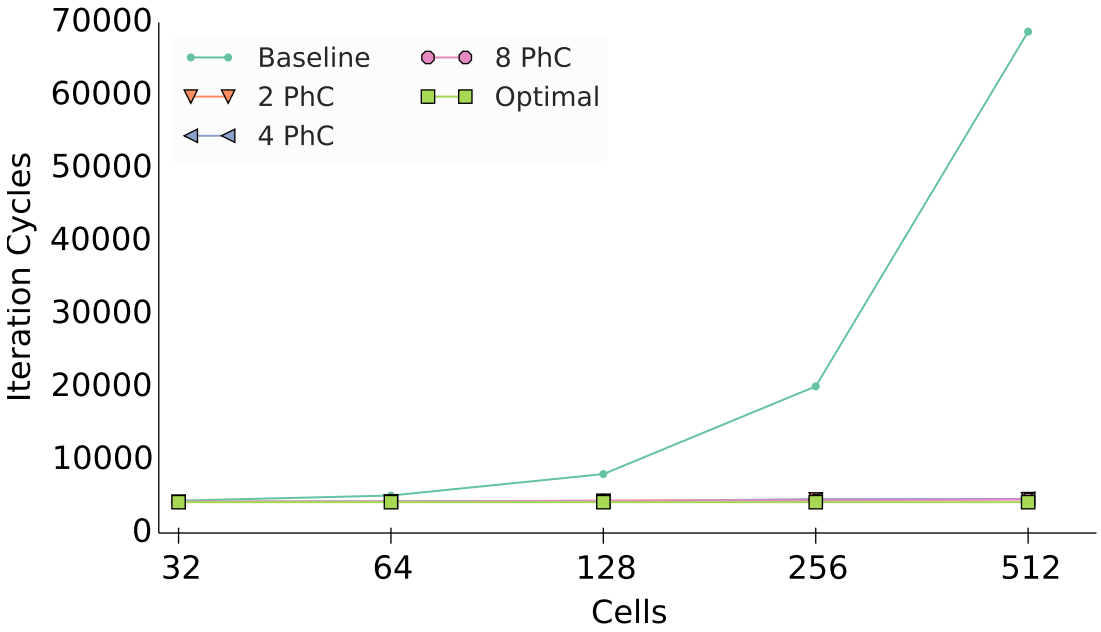


Figure 5-11: Comparison between different cluster sizes for different amounts of cells. The *normal* connection scheme is used and each cell calculation takes *534 cycles*. The systems use a router fan-out of two. The baseline is taken from [27]. All presented configurations of the new system meet the brain real time of $50\,\mathrm{m}$ at $100\,\mathrm{MHz}$. Furthermore, all of them scale linearly with the number of cells. The Baseline on the other hand scales exponentially.

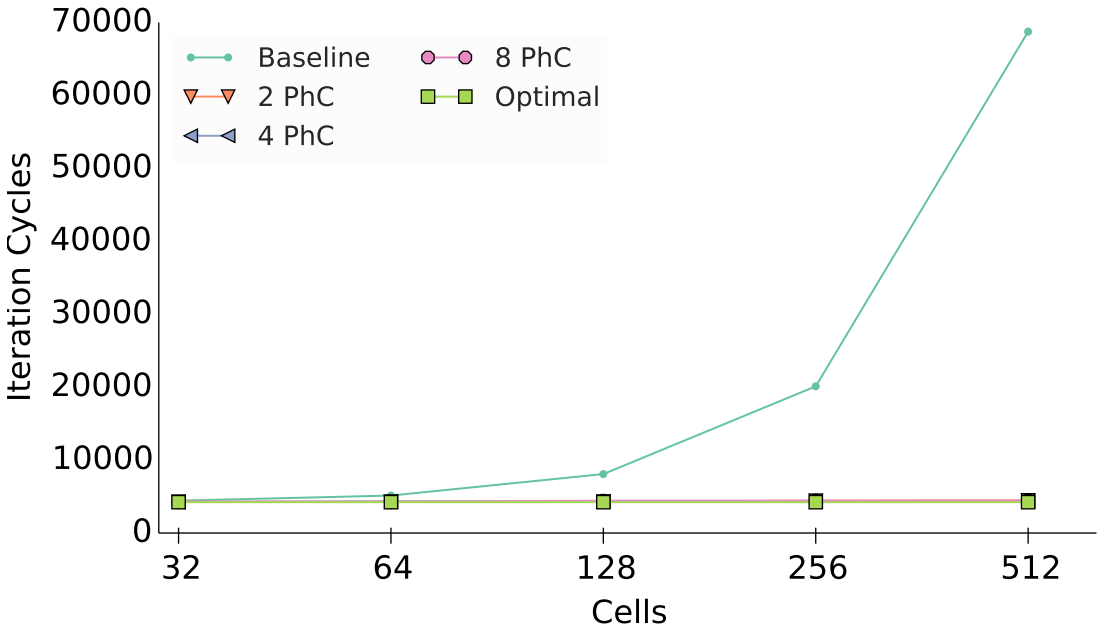


Figure 5-12: Comparison between different cluster sizes for different amounts of cells. The *neighbour* connection scheme is used and each cell calculation takes *534 cycles*. The systems use a router fan-out of two. The baseline is taken from [27]. All presented configurations of the new system meet the brain real time of $50\,\mathrm{m}$ at $100\,\mathrm{MHz}$. Furthermore, all of them scale linearly with the number of cells. The Baseline on the other hand scales exponentially.

multiple chip solutions are mandatory. The next section presents simulation results for environments with multiple chips.

5-3 Scalability to multiple Chips

The previous section showed that the proposed system on one chip performs better by a large margin compared to the baseline in [27]. For the realistic connection scenarios the proposed system performs almost as good as the optimal system. To evaluate the scalability of the proposed system to multi-chip systems, which are needed as a single chip is limited by hardware resources, the comparison with the baseline is left out. Instead the multi-FPGA system is compared to the single chip variant as well as the optimal approach. The comparisons are done using both the dedicated wire as well as the packet based synchronisation methods. In contrast to the previous section only the realistic connection scenarios are presented, the all-to-all connection scheme is not utilised. All simulations are done for the one chip, 2, 4 and 8 chip systems with 2 PhC clusters. Furthermore, all multi-chip simulations are done for three different inter chip connection speeds. A single packet can be transmitted between two chips in 2, 8 or 32 cycles.

The first simulation uses the packet based synchronisation scheme across all chips. As the system controller is located in the last chip the other chips have to send their control packets over multiple intermediate chips. This results in lower performance in systems with many chips or low inter chip connection speed. These findings are supported by the simulations shown in Figs. 5-13 and 5-14. The largest and slowest system with 8 chips and a a very slow inter chip communication performs the worst at 21 times slower iteration times compared to the one chip implementation. When looking at faster inter chip communication on the other hand the 8 chip system requires only 3.4 times as many cycles compared to the one chip system. These results show that the packet based synchronisation scheme can be feasible for smaller systems as less additional hardware is required.

These findings are supported by the simulations with calculation times presented in Figs. 5-15 and 5-16. The communications between the chips required for the packet based synchronisation can not be hidden by calculations. This means that they cannot take place at the same time that calculations take place. Thus, the simulations present the same picture as the simulations without calculation times. The two chip two cycles communication system performs on par with the one chip system. The four chip two cycle system performs $12.8\,\%$ and the eight chip two cycle system performs $29.2\,\%$ worse than the one chip system. Overall the packet based synchronisation is feasible only for fast inter chip communication and small systems. For larger systems or slower communication speeds the communication costs get too high and the iteration performance deteriorates.

The second approach to synchronizing all clusters is using a dedicated controller that is connected with each chip using two wires. In contrast to the packet based approach no packet has to cross multiple chip boundaries. For the normal connection scheme the dedicated controller performs slightly better than the packet based controller as shown in Fig. 5-17. At 8 chips and 2 cycles per communication the second approach performs

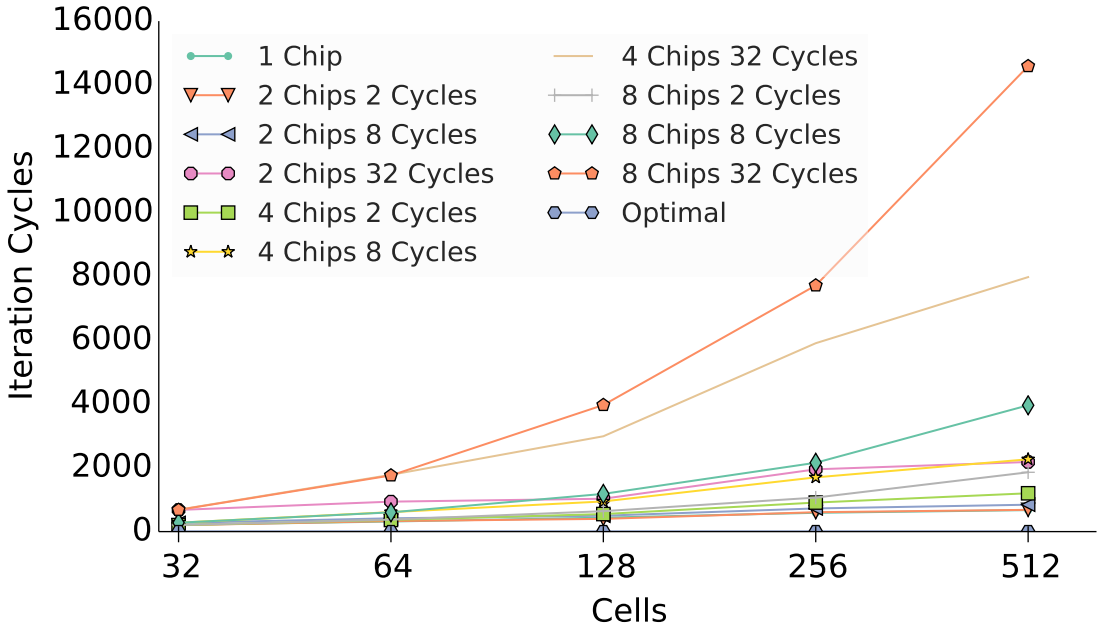


Figure 5-13: Comparison between different system configurations utilising between one and eight chips to simulate cell amounts between 32 and 512. Calculation times are *excluded*. The *normal* connection scheme is used. The multi chip systems utilise the packet based synchronization method.

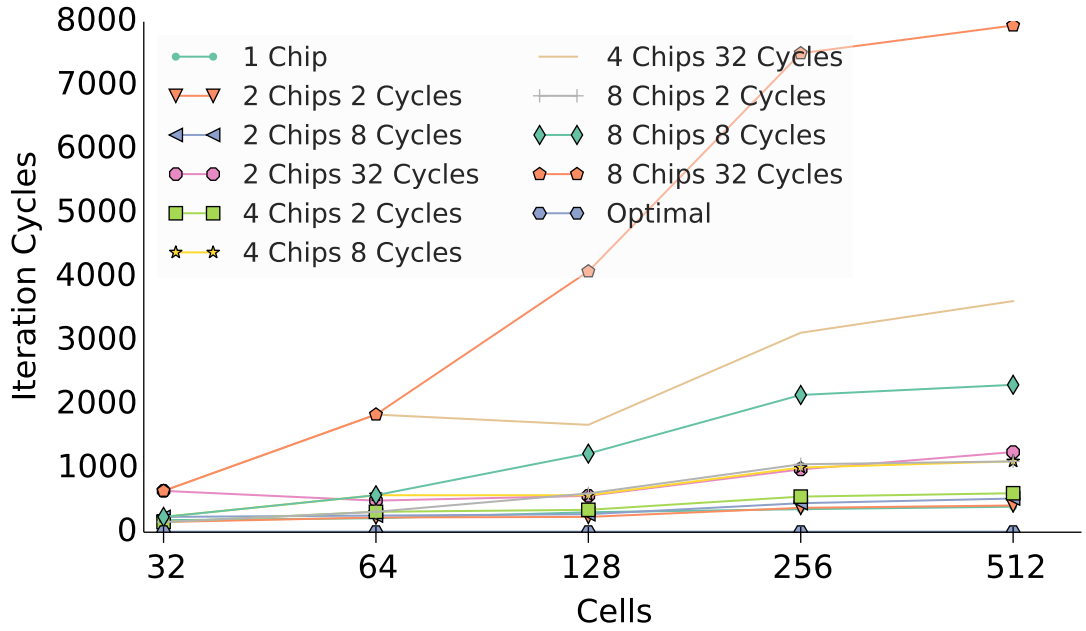


Figure 5-14: Comparison between different system configurations utilising between one and eight chips to simulate cell amounts between 32 and 512. Calculation times are *excluded*. The *neighbour* connection scheme is used. The multi chip systems utilise the packet based synchronization method.

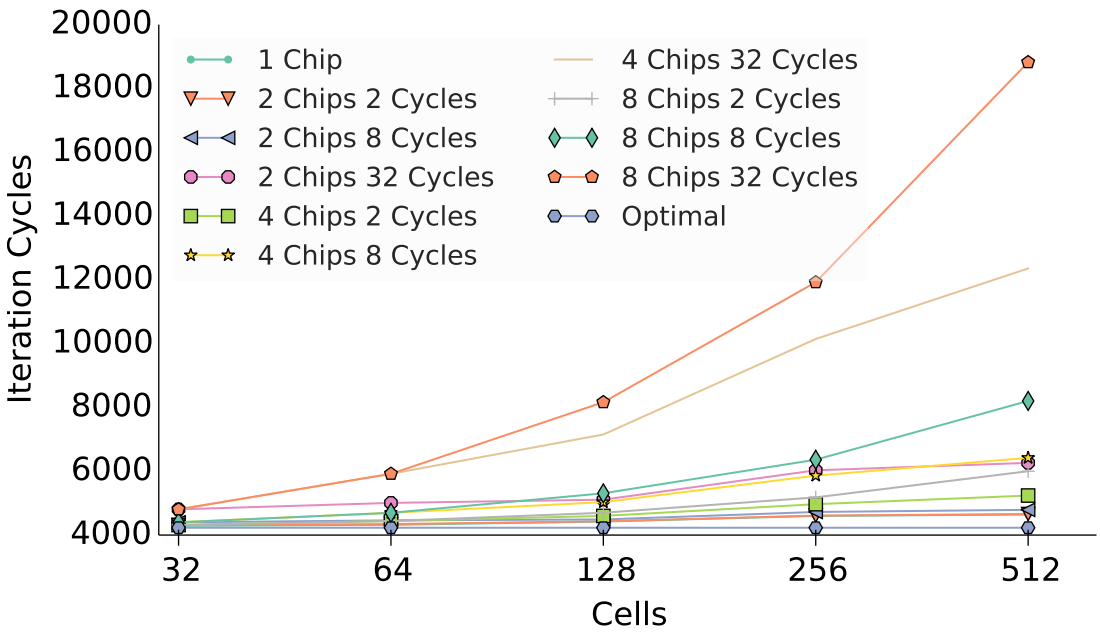


Figure 5-15: Comparison between different system configurations utilising between one and eight chips to simulate cell amounts between 32 and 512. Calculation times are included as *534 cycles* per iteration. The *normal* connection scheme is used. The multi chip systems utilise the packet based synchronization method. It can be seen that most configurations do not meet the brain real-time of $50\,\mathrm{ms}$ at $100\,\mathrm{MHz}$. The packet synchronisation mode is mainly responsible for this bad scalability. This problem is solved by incorporating dedicated wire synchronisation.

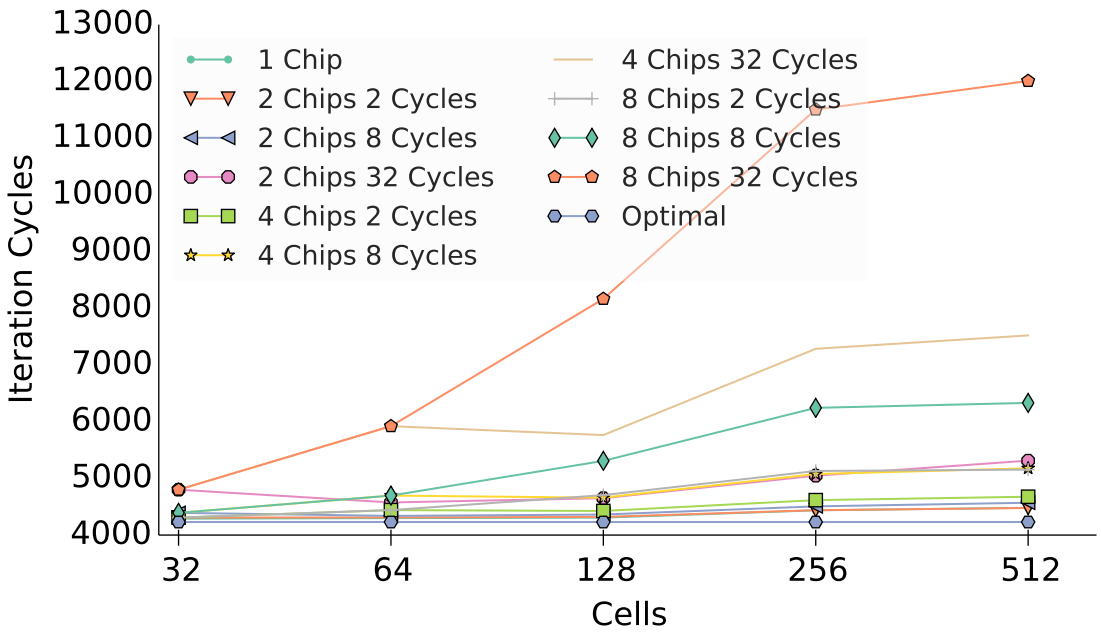


Figure 5-16: Comparison between different system configurations utilising between one and eight chips to simulate cell amounts between 32 and 512. Calculation times are included as *534 cycles* per iteration. The *neighbour* connection scheme is used. The multi chip systems utilise the packet based synchronization method. It can be seen that most configurations do not meet the brain real-time of $50\,\mathrm{ms}$ at $100\,\mathrm{MHz}$. The packet synchronisation mode is mainly responsible for this bad scalability. This problem is solved by incorporating dedicated wire synchronisation.

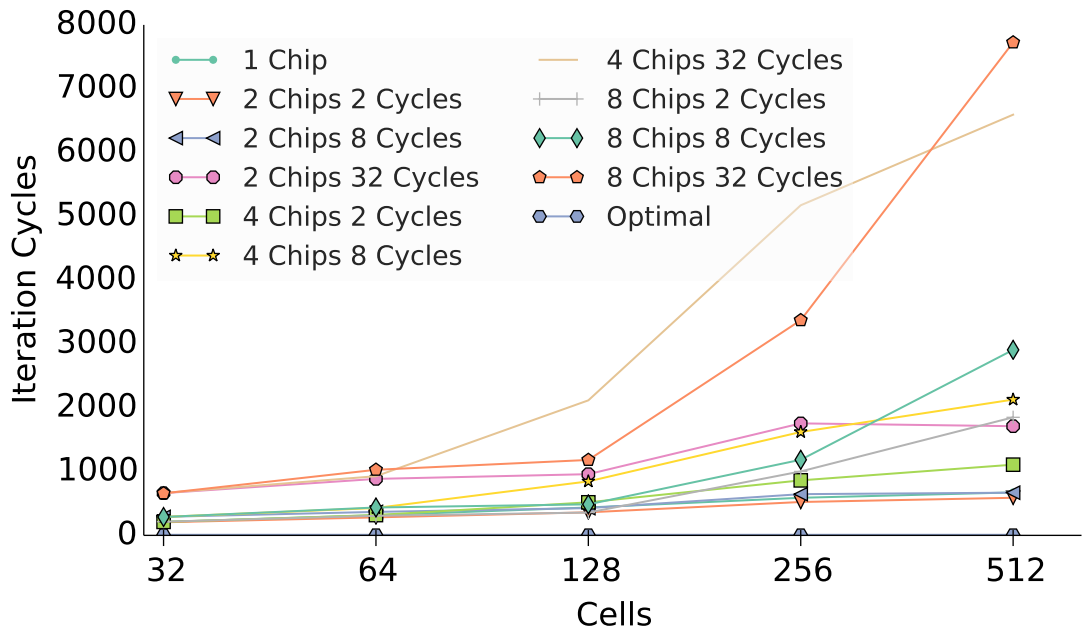


Figure 5-17: Comparison between different system configurations utilising between one and eight chips to simulate cell amounts between 32 and 512. Calculation times are *excluded*. The *normal* connection scheme is used. The multi chip systems utilise the dedicated wire based synchronization method.

only $0.48\,\%$ better than the packet based synchronisation approach. The dedicated controller shows it strength for slow inter chip communication. At 8 chips with 32 cycles per communication the dedicated controller performs $88.4\,\%$ better than the packet based synchronisation method.

The performance difference is even larger when looking at the neighbour connection scheme. For this connection scheme the dedicated controller system is 3.84 times faster than the packet based approach at 8 chips with very slow communication. These results show that especially at slow communication speeds the second approach performs considerably better. For faster connection speeds on the other hand the difference in communication time is in the order of $1\,\%$.

The simulations without calculation already indicate that the system can not meet the brain real-time of $50\,\mu s$ for the normal connection scheme and very slow connection speed. For the middle and high connection speeds on the other hand the communication costs should be low enough to allow for brain real time. The simulation results in Fig. 5-19 support these assessments. A system of 8 chips and a fast inter chip connection is still fast enough to achieve brain real time. Slower communication speeds on the other hand result in a high cost for communication. Compared to the first approach the differences in iteration time remain the same as shown for the simulations without calculation time. The second approach performs $1.1\,\%$ and $57.1\,\%$ worse for the fast and slow communication speed at 8 chips.

For the neighbour connection scheme all configurations are feasible. Even at 8 chips and

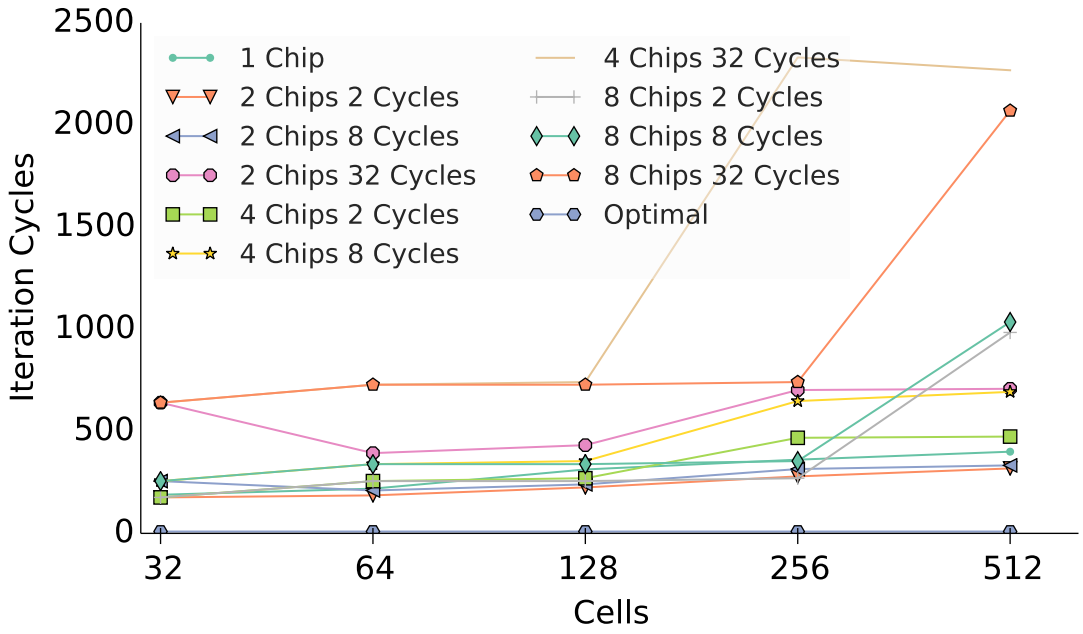


Figure 5-18: Comparison between different system configurations utilising between one and eight chips to simulate cell amounts between 32 and 512. Calculation times are *excluded*. The *neighbour* connection scheme is used. The multi chip systems utilise the dedicated wire based synchronization method.

slow communication speed brain real time can be achieved. The system performs $88.95\,\%$ better at that configuration compared to the packet based synchronization scheme. For fast communication speeds the performance advantage shrinks to $1\,\%$.

The previous sections presented the overall performance of the system in simulations. While the simulations can make statements about the performance of the system they can not give any assertions about the implementability of the system in hardware. The following section gives hardware utilisation estimations to show that the proposed system not only performs to expectation but is also practicably usable.

5-4 Hardware Utilisation Estimations

This section aims at showing that the proposed system is practicably implementable in hardware. The resource usage is estimated by determining the hardware usage of the parts of the system. The most important and biggest parts oft the system are the routers and the clusters. For the clusters the biggest parts are the PhC. As the PhC are based of the work in [27] the hardware utilisation is taken from that work. The size of a cluster is estimated as $n_{PhC} \times u_{PhC} + C$ with n_{PhC} being the number of PhC per cluster and u_{PhC} being the hardware utilisation per cluster. C is a constant overhead for the controllers but is ignored for this estimation as the controllers only consist of small counters and combinational logic which are negligible in size compared to the PhC. The routers are synthesized using

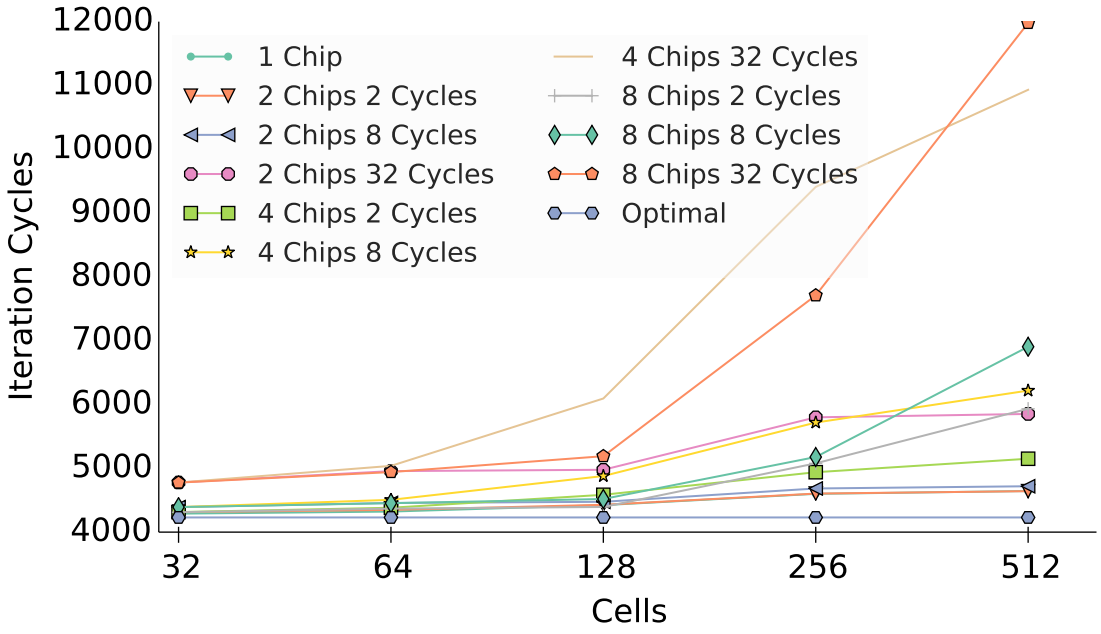


Figure 5-19: Comparison between different system configurations utilising between one and eight chips to simulate cell amounts between 32 and 512. Calculation times are included as *534 cycles* per iteration. The *normal* connection scheme is used. The multi chip systems utilise the dedicated wire based synchronization method. It can be seen that most configurations meet the brain real-time of $50\,\mathrm{ms}$ at $100\,\mathrm{MHz}$. Furthermore, these configurations scale linearly with the number of cells. Nonetheless, some configurations scale much worse. The correct configuration for a specific task has to be chosen by simulation.

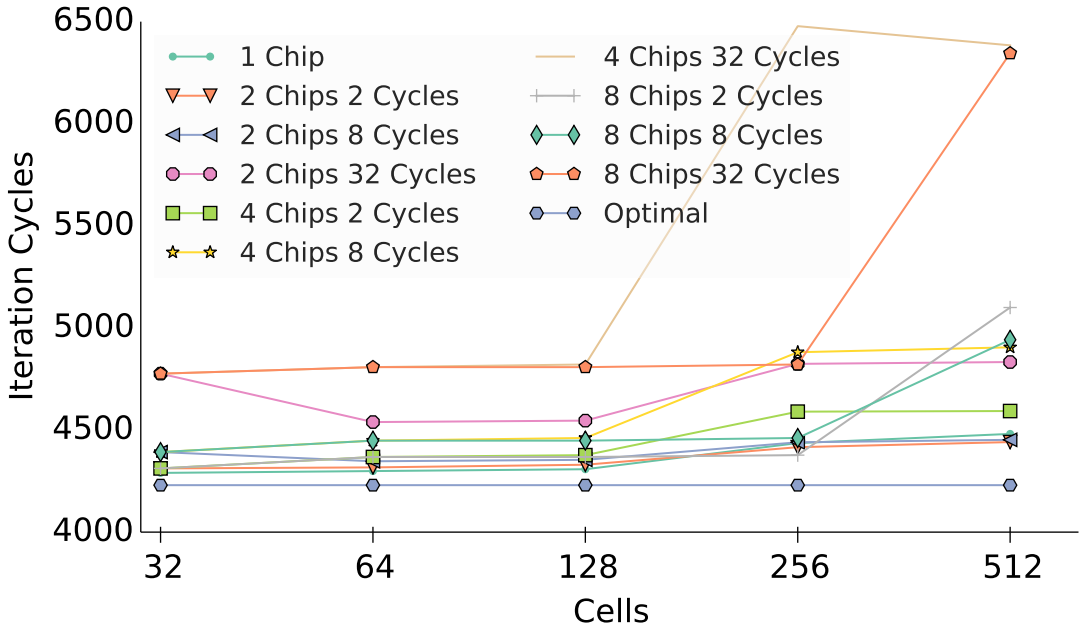


Figure 5-20: Comparison between different system configurations utilising between one and eight chips to simulate cell amounts between 32 and 512. Calculation times are included as *534 cycles* per iteration. The *neighbour* connection scheme is used. The multi chip systems utilise the dedicated wire based synchronization method. It can be seen that most configurations meet the brain real-time of $50\,\mathrm{ms}$ at $100\,\mathrm{MHz}$. Furthermore, these configurations scale linearly with the number of cells. Nonetheless, some configurations scale much worse. The correct configuration for a specific task has to be chosen by simulation.

Component	FF	FF %	LUT	LUT %
Available	692800	100 %	346400	100 %
PhC	20488	2.96 %	30235	8.73 %
Router	132	0.019 05 %	187	0.053 98 %

Table 5-1: Hardware utilisation on a Xilinx Virtex 7 FPGA of the most important components of the system. The PhC numbers are taken from [27] and are used to estimate the cluster sizes of the proposed system. Routers sizes are generated by synthesizing a SystemC model of a router using Vivado HLS 2013.4.

Vivado HLS 2013.4 for the same Xilinx Virtex 7 FPGA as used in [27]. It turns out that assuming a input FIFO depth of 4 and 512 cells with corresponding routing tables the delayed buffer can hold up to 206 delayed packets without requiring more than one BRAM slice of the FPGA. Thus, the routers are also small. Considering that the size of the system is limited mainly by the amount of PhC that fit onto a single chip, the size of the routers is a minor factor. Table 5-1 presents the estimated hardware utilisation numbers for the different components of the system in terms of Flip Flops (FF) and Look Up Tables (LUT) used.

Considering that the LUTs are the most limited resource according to the resource usage in Table 5-1 and that the overhead required to form a cluster out of multiple PhC is estimated at $10\,\%$, an estimate for the maximum number of cells in a system can be given as follows where l is the amount of LUTs in a component and n is the amount of the given component in the system:

$$l_{\text{Cluster}} = 2 \times 30235 \times 1.10 = 66517 \tag{5-1}$$

$$n_{\text{Cluster}} = \frac{346400}{66517} = 5.2076 \tag{5-2}$$

$$n_{\text{PhC}} = \lfloor 5.2076 \rfloor \times 2 = 10 \tag{5-3}$$

$$n_{\text{cells}} = n_{\text{PhC}} \times 8 = 80 \tag{5-4}$$

Thus, the largest possible system on one chip with these estimates is at 10 PhC which corresponds to 5 clusters or 80 cells. For the 5 clusters 10 routers are needed. Thus the routers of this system only require $0.2\,\%$ of the LUTs and $0.5\,\%$ of the FF. These estimates are rough and missing the hardware costs for smaller components like the synchronisation circuits. Thus, the estimates are supposed to show that the network itself is small enough to be feasible and very small compared to the PhC. According to these numbers the proposed system is feasible to implement and is mainly limited by the size of the PhC. As shown in [26] the size of a PhC can be reduced by using different optimisations and the 80 cells is just a lower bound for the number of cells on a chip.

5-5 Conclusion

This chapter presented the performance of the proposed system. Using SystemC simulation techniques it is shown that the system performs considerably faster than the baseline.

Furthermore, it is shown that many configurations scale linearly with the number of neurons. As some configurations scale poorly, the simulation has to be used to find the correct system configuration for a particular problem.

For one chip systems the network alone is 177 times faster than the baseline at 512 cells. Incorporating calculation times the baseline continues to be 15 times faster. More importantly the increase in run time between 32 cells and 512 cells is less than $10\,\%$, for the two realistic connection schemes, compared to the baseline which sits at a $1550.6\,\%$ increase.

For the neighbour connection scheme the increase in run time decreases to less than $3\,\%$. While the new system scales linearly with the number of cells in the system, the baseline scales exponentially. Considering an average increase in run-time of 120 cycles for twice the amount of cells the supported amount of cells on chip can be estimated using the following formula:

$$c_n = c_o + n \times 120 \tag{5-5}$$

$$n = \frac{c_n - c_o}{120}. \tag{5-6}$$

In this formula c_n is the amount of iteration cycles available for one iteration, thus, at $100\,\text{MHz}$ and a brain real-time of $50\,\mu s$ 5000 cycles. c_o stands for the amount of cycles currently used which is 4372 cycles at 512 cells. The value received, n, denotes the amount of times the number of cells in the system can be doubled.

$$n = 5.23$$

Thus, the system supports *more than 19200 cells on one chip* for the neighbour connection scheme. Considering the current maximum of 96 cells on one chip, the system allows for 200 times more cells. For normal connection mode the system requires 4597 cycles at 512 cells. The increase in iteration time for the twice the amount of cells is 155. According to same formula as above the system is able to handle 2.6 doublings, hence, *more than 3000 cells on one chip*.

Using only one chip, the growth of the system is limited by the size of the PhC. Therefore, the system supports multi chip implementations. Based on the current maximum of 96 cells a multi chip system of four FPGA can be built containing *384 cells without optimisation* of the calculations. This multi FPGA system still meets the brain real-time requirements of $50\,\mu s$. Furthermore, a system of eight FPGA is fast enough for *768 cells*, if the connection speed between the FPGA is high.

While the simulations presented in this section incorporate between 32 and 512 cells initial simulations with over 4096 cells suggest that the scalability remains linearly for higher number of neurons.

The potential of the system can be increased by using less shared cells in the PhCs. The increase in resource costs by requiring more clusters for the same amount of cells is counteracted by the increase in free cycle time. The free cycle time can be used to add additional FPGA to the system which in return results in more cells overall.

Additionally, it is shown that the system is implementable in hardware using negligible resources compared to the calculation units.

The following chapter concludes this thesis and gives and outlook on further work.

Chapter 6

Conclusion and Further Work

The goal of the work presented is to design a system capable of simulating spiking neural networks implementing the extended Hodgkin-Huxley model over multiple FPGA. This chapter concludes and summarizes the work done to achieve the goal of simulating large numbers of brain cells and gives an overview of possible extensions to the work presented.

6-1 Conclusion

The thesis starts of by introducing the research area and goals of this thesis in Chapter 1. Current neuron simulators, which are precise enough to simulate neurons in a biophysically-meaningful way, are limited in amount of neurons. The largest system is capable of simulating up to 96 neurons in $50\mu s$ brain real-time simulation time steps. Especially the interconnect between the neurons poses a major limitation to the system growth. The neurons are interconnected by a bus, which scales exponentially with the number of neurons in the system. The main goal of this thesis is to design a neuron interconnect that scales linearly or better with the amount of neurons in the system. Additionally, breaking the barriers of a single chip system by incorporating multiple chips to form a bigger system is desirable. The system has to scale linearly over multiple chips, too. Run-time configurability is another major point in the system design as the resynthesis of the system is a large time factor. All previous points lead to higher biophysical accuracy as more precise neurons can be simulated in shorter periods of time.

The current state of the art does not provide any such system. As shown in Chapter 2 either large or accurate systems exist. Systems that combine both factors have not been developed yet. Very large systems like the SpiNNaker neuron simulator with over one million neurons are not accurate enough to allow research in neuron behaviour. Other approaches like the Neurogrid use analog neurons to achieve huge neuron systems but the analog neurons are neither flexible to model changes nor easily observable. More precise systems like the ones in [27] are limited by the complexity of the calculations as

well as their interconnect performance. Furthermore, the systems do not provide run-time configurability and, as such, they are not adaptive.

In Chapters 3 and 4 a system is designed that is able to bridge the gap between biophysically accuracy and massive numbers of cells. The communication between neurons occur in a limited area. These locality of communication can be exploited by a system to achieve linear communication time growth. While perfect localisation of communication is not possible due to physical constraints, it is shown in Section 3-3 that close by cells can be grouped around a shared memory to allow for instantaneous communication. These groups of neurons are called clusters. Furthermore, each Physical Cell (PhC) calculates the outputs of multiple neurons in a time-shared way. This time sharing is possible as the brain real-time of $50\,\mu s$ is much larger than the time needed to calculate the outputs of a single cell. By using time sharing, each cluster contains even more cells and communications are further localised. Communications between clusters is realised using a tree topology network on the chip. The tree topology is chosen as it localises communications. Clusters that are close by can communicate using only one hop in the network. Clusters that are further away have to communicate less frequently and, as such, the penalty for taking multiple hops is less severe. Another advantage of the tree based topology is that the system can be extended over multiple chips, when connected at the root of the tree. Adding the typically slower connection to another chip does not incur a huge performance penalty. The combination of clusters and a tree topology NoC allows for almost linear scaling of the system. To provide run-time configurability, a tree based communication bus is used as described in Section 3-9. The bus enables the user to configure the connectivity between cells and change the parameters of the calculations. As a result, resynthesising the whole system just to experiment with a different connectivity between cells is not required anymore. Additionally, the creation of the whole system is automated. The user has to enter the amount of neurons in the system as well as the desired connectivity scheme. From this information, all required routing tables and topologies are automatically generated, even for multi-chip systems.

The evaluation in Chapter 5 supports the performance of data localisation. By using SystemC, the whole system can be simulated including all calculation and communication latencies, both on- or off-chip. For evaluation purposes, various configurations of the system for 32 to 512 cells are compared. Depending on the configuration, it is shown that the system scales linearly with the amount of cells. Initial simulations with up to 4096 neurons show that the linear growth continues. The automatic system generation combined with the simulator can be used to easily find the optimal system configuration for the task at hand. The linear scalability of the system leads to massive improvements in the amounts of neurons which is possible to simulate. Compared to the previous systems with 96 neurons the new system is able to support between 31 and 200 times more neurons depending on the connectivity scheme. It is further shown that the ability to almost linearly scale over multiple chips allows for an easy expansion of the 96 neuron system to two, four or even eight additional FPGA. This expansion leads to up to 768 cells in a single system compared to the 96 neuron system. While the system supports many more neurons, it still meets the brain real-time of $50\,\mu s$. Furthermore, the hardware cost to realise the clusters and the network is meager compared to the size of the neuron calculation cores.

The proposed system meets all the goals of the thesis presented in Section 1-3. Linear scalability allows for large increases in the neuron count compared to the state-of-the-art system. Additionally, the system can scale linearly over multiple chips as well. The run-time configurability enables faster experimentation times. All of these results lead to a system that provides higher biophysical accuracy compared to the state of the art.

Summarized the main contributions of the proposed system are:

- **Close to linear growth in communication cost.** Simulations show that the system is able to handle 19200 cells for neighbour connection mode and over 3000 cells in normal connection mode. The constraints used for this result are $50\,\mu$s per simulation step at $100\,$MHz clock frequency (5000 cycles). Furthermore, infinite hardware resources are assumed. The cell calculation latency is 528 cycles. Faster cell calculation latencies improve the capabilities of the system.
- **Extendable system that can cover multiple chips.** The current state of the art system simulates 96 cells. By using the multi-chip capabilities of the proposed system four to eight times the amount of cells are possible. To connect four FPGAs only moderate inter FPGA link requirements need to be fulfilled. For eight FPGAs systems a high link speed is required.
- **Proposed system is limited by the calculation speed of the PhC.** The proposed system scales with improvements to the calculation speed as well as improvements to the FPGA size.
- **Simulation environment to test system configurations.** Before implementing a specific configuration the simulator can be used to determine the performance of the new system. The simulator incorporates all latencies, such as, calculation latencies and on- and off-chip communication latencies.
- **Automatic structure and connectivity table generation.** The system generates all necessary routing tables and structures automatically when provided with the desired specifications, such as, cell amount and desired connectivity scheme.
- **Higher biophysical accuracy compared to the state-of-the-art.** The resulting system with massive amounts of cells can be used in a variety of research areas, such as, artificial intelligence or driving assistance.

This section concludes the work on this thesis. In the next section, we describe possible extensions to the work proposed in this thesis.

6-2 Further Work

This section provides ideas for further research interests following from the work on this thesis. For one these are possibilities to improve the performance of the system and furthermore ways to extend the system to increase the functionality.

Implementation on FPGA: First and foremost, the system should be implemented on an FPGA to prove that the simulation results are correct and hold true for hardware implementations. As the simulation is already done in cycle accurate SystemC, mainly transmission work is necessary to generate synthesizable SystemC. Alternatively the system can be rewritten in any Hardware Description Language (HDL).

Analog Simulation: SystemC AMS can be used to simulate the analog part of the system. Currently the ADC and the DAC are black boxes without functionality. These black boxes can be replaced by SystemC AMS modules to simulate the real life behaviour of the converters and feed the system with actual data.

Calculation Performance: The calculation performance is not part of the work on this thesis. Currently the calculations for all cells are done sequentially without any kind of pipelining and with expensive floating point operations. As the system is not anymore limited by the communication costs, it provides a performance increase to reduce the calculation costs. Currently the communication costs are less than $15\,\%$ of the total iteration costs and these costs are mainly hidden behind the calculations, which run in parallel. Thus, there is a room for improvements without the need to further increase the communication performance. On the contrary, reducing the time required for calculations increases the cluster sizes which results in higher communication localisation and, thus, better interconnect performance, as less communication leaves a cluster.

Router Memory Usage: Depending on the type of FPGA, it can be advantageous to look into ways to reduce the size of the routing tables as well as the various buffers inside the routers to allow for further scaling. Currently on Virtex 7 and on Spartan 6 FPGA, the routers are small enough to support systems with multiple thousand cells across multiple FPGA.

Second Clock Domain: Currently the whole system uses a single clock for all components. As the interconnect can be run at higher clock frequencies compared to the PhC the communication costs can further be lowered by introducing a second faster clock for the interconnect. The clock domains can either be split at the cluster boundaries, which are already implemented as *FIFO* and thus making clock syncing easier or at the PhC domains, which is possible as the shared memory in the clusters is implemented as true dual port memory. The latter approach requires more synchronisation effort and currently would not provide any performance benefit over the former approach.

Multi chip topology: When multiple chips are connected into a larger system a ring topology is used. While this topology has many advantages for the system performance, it is not very suitable for a real life system, where the chips take up space. Thus, it might be feasible to look into different topologies for the multi-chip interconnect that are as fast as the ring topology but also better implementable in real hardware to allow for larger, tightly packet, systems.

Automatic Learning: The system is already controllable by a human controller allowing for changes to cell connections. The process of creating and destroying connections between cells can be automated to simulate learning processes. Such algorithms can either be implemented by using the already implemented control bus or by using hardware as new modules of the system. For the first approach a way to read information from the system is required to receive statistics about the different cell connections that can be used by the algorithms to decide about new connections or to destroy connections. The second approach might be unfeasible due to hardware costs.

Event Based Approach: Currently each communication packet is delivered despite its content (also without information). To increase the efficiency of the system, it could be changed to only react on events instead of always producing new data. Changing the system to an event based approach requires a complete overhaul of almost all parts of the system including the HH-model used.

Optimization of Cell Placements: To better support connection schemes from other sources than the generator, optimization algorithms should be used to optimize the distribution of cells on the clusters. The goal of the optimization is to achieve a cell distribution that reduces communication over cluster of chip boundaries. Currently, the connection schemes that are not generated by the configuration generator are applied to the clusters without respecting the connectivity to other cells.

Priority Packets: In certain situations, it happens that packets that have to travel across chip boundaries are calculated too late in an iteration cycle. Packets that have to travel further should be generated first by the clusters. This would allow for more communication to be hidden behind the calculations.

Bibliography

[1] A. L. HODGKIN and A. F. HUXLEY. "A quantitative description of membrane current and its application to conduction and excitation in nerve." In: *The Journal of physiology* 117.4 (Aug. 28, 1952), pp. 500–544. ISSN: 0022-3751. URL: http://jp.physoc.org/content/117/4/500.abstract.

[2] A Andreou and K Boahen. "Synthetic Neural Circuits Using Current-Domain Signal Representations". In: *Neural Computation* 1.4 (Dec. 1989), pp. 489–501. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.489.

[3] RD Traub et al. "A model of a CA3 hippocampal pyramidal neuron incorporating voltage-clamp data on intrinsic conductances." In: *Journal of neurophysiology* 66.2 (1991), pp. 635–650.

[4] Michael A. Arbib. *The Handbook of Brain Theory and Neural Networks*. 1st. Cambridge, MA, USA: MIT Press, 1995. ISBN: 0262011484.

[5] William J. Dally and Brian Towles. "Route Packets, Not Wires: On-chip Inteconnection Networks". In: *Proceedings of the 38th Annual Design Automation Conference*. DAC '01. Las Vegas, Nevada, USA: ACM, 2001, pp. 684–689. ISBN: 1-58113-297-2. DOI: 10.1145/378239.379048. URL: http://doi.acm.org/10.1145/378239.379048.

[6] Edward J. Fine1, Catalina C. Ionita1, and Linda Lohr. "The History of the Development of the Cerebellar Examination". In: *Seminars in Neurology* (2002), pp. 375–384. DOI: 10.1055/s-2002-36759.

[7] Wulfram Gerstner and Werner Kistler. *Spiking Neuron Models: An Introduction*. New York, NY, USA: Cambridge University Press, 2002. ISBN: 0521890799.

[8] Eugene M. Izhikevich. "Simple model of spiking neurons". In: *IEEE Trans. Neural Networks* (2003), pp. 1569–1572.

[9] Wolfgang Maass. "Noisy Spiking Neurons with Temporal Coding have more Computational Power than Sigmoidal Neurons." In: *NIPS*. Ed. by Michael Mozer, Michael I. Jordan, and Thomas Petsche. MIT Press, May 28, 2003, pp. 211–217. URL: http://dblp.uni-trier.de/db/conf/nips/nipsN1996.html#Maass96.

[10] Eugene M. Izhikevich. “Which Model to Use for Cortical Spiking Neurons”. In: *IEEE Transactions on Neural Networks* 15 (2004), pp. 1063–1070.

[11] James Balfour and William J. Dally. “Design Tradeoffs for Tiled CMP On-chip Networks”. In: *Proceedings of the 20th Annual International Conference on Supercomputing*. ICS ’06. Cairns, Queensland, Australia: ACM, 2006, pp. 187–198. ISBN: 1-59593-282-8. DOI: 10.1145/1183401.1183430. URL: http://doi.acm.org/10.1145/1183401.1183430.

[12] *IEEE Std 1666 - 2005 IEEE Standard SystemC Language Reference Manual*. 2006, pp. –423.

[13] H. Shayani, P.J. Bentley, and AM. Tyrrell. “Hardware Implementation of a Bio-plausible Neuron Model for Evolution and Growth of Spiking Neural Networks on FPGA”. In: *Adaptive Hardware and Systems, 2008. AHS ’08. NASA/ESA Conference on*. June 2008, pp. 236–243. DOI: 10.1109/AHS.2008.13.

[14] Karsten Einwich et al. “Introduction to the SystemC AMS DRAFT standard.” In: *SoCC*. IEEE, 2009, p. 446. ISBN: 978-1-4244-4940-8. URL: http://dblp.uni-trier.de/db/conf/socc/socc2009.html#EinwichGBV09.

[15] Chun He, A Papakonstantinou, and Deming Chen. “A novel SoC architecture on FPGA for ultra fast face detection”. In: *Computer Design, 2009. ICCD 2009. IEEE International Conference on*. Oct. 2009, pp. 412–418. DOI: 10.1109/ICCD.2009.5413122.

[16] Javier Navaridas et al. “Understanding the Interconnection Network of SpiNNaker”. In: *Proceedings of the 23rd International Conference on Supercomputing*. ICS ’09. Yorktown Heights, NY, USA: ACM, 2009, pp. 286–295. ISBN: 978-1-60558-498-0. DOI: 10.1145/1542275.1542317. URL: http://doi.acm.org/10.1145/1542275.1542317.

[17] Richard Herveille. *WISHBONE, Revision B.4 Specification*. Tech. rep. OpenCores, 2010. URL: http://cdn.opencores.org/downloads/wbspec_b4.pdf.

[18] Chris I. De Zeeuw et al. “Spatiotemporal firing patterns in the cerebellum”. In: (2011), pp. 327–344. DOI: 10.1038/nrn3011.

[19] P Bazzigaluppi et al. “Olivary subthreshold oscillations and burst activity revisited”. In: *Frontiers in neural circuits*. 6. Nov. 22, 2012, p. 91.

[20] Marcel Beuler et al. “Real-Time Simulations of Synchronization in a Conductance-Based Neuronal Network with a Digital FPGA Hardware-Core”. English. In: *Artificial Neural Networks and Machine Learning – ICANN 2012*. Ed. by AlessandroE.P. Villa et al. Vol. 7552. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 97–104. ISBN: 978-3-642-33268-5. DOI: 10.1007/978-3-642-33269-2_13. URL: http://dx.doi.org/10.1007/978-3-642-33269-2_13.

[21] Kit Cheung, Simon R. Schultz, and Wayne Luk. “A Large-scale Spiking Neural Network Accelerator for FPGA Systems”. In: *Proceedings of the 22Nd International Conference on Artificial Neural Networks and Machine Learning - Volume Part I*. ICANN’12. Lausanne, Switzerland: Springer-Verlag, 2012, pp. 113–120. ISBN: 978-3-642-33268-5. DOI: 10.1007/978-3-642-33269-2_15. URL: http://dx.doi.org/10.1007/978-3-642-33269-2_15.

[22] H.A. Du Nguyen. “GPU-based simulation of brain neuron models”. MA thesis. Delft, The Netherlands: Delft University of Technology, Aug. 2013.

[23] Yiwei Zhang et al. “Biophysically Accurate Foating Point Neuroprocessors for Reconfigurable Logic”. In: *Computers, IEEE Transactions on* 62.3 (Mar. 2013), pp. 599–608. ISSN: 0018-9340. DOI: 10.1109/TC.2011.257.

[24] B.V. Benjamin et al. “Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations”. In: *Proceedings of the IEEE* 102.5 (May 2014), pp. 699–716. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2313565.

[25] *Normal and Students t Distributions and Their Applications*. Springer, 2014. ISBN: 978-94-6239-061-4.

[26] Georgios Smaragdos et al. “FPGA-based Biophysically-meaningful Modeling of Olivocerebellar Neurons”. In: *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*. FPGA ’14. Monterey, California, USA: ACM, 2014, pp. 89–98. ISBN: 978-1-4503-2671-1. DOI: 10.1145/2554688.2554790. URL: http://doi.acm.org/10.1145/2554688.2554790.

[27] M.F. Van Eijk. “Modeling of Olivocerebellar Neurons using SystemC and High-Level Synthesis”. MA thesis. Delft, The Netherlands: Delft University of Technology, 2014.

[28] ATT and Bell-Labs. *Graphviz - Graph Visualization Software*. URL: https://www.graphviz.org.

[29] Baptiste Lepilleur. *JSONCpp*. URL: https://github.com/open-source-parsers/jsoncpp.

[30] Guido van Rossum. *Python Programming Language*. URL: https://www.python.org/.