Department of Physics and Astronomy

University of Heidelberg

Diploma thesis

in Physics

submitted by

Ioannis Kokkinos

born in Reutlingen

2012

Feasibility Study On Declarative Routing

For Neuromorphic Hardware

This diploma thesis has been carried out by Ioannis Kokkinos

at the

Kirchoff-Institure for Physics

under the supervision of

Prof. Dr. Karlheinz Meier

Machbarkeitsstudie zu deklarativem Routing für neuromorphe Hardware

Eine passende Konfiguration für ein neuromorphes Hardware System zu finden, die einem gegebenen neuronalen Netzwerk entspricht, ist eine komplexe Aufgabe. Dies ist äquivalent zur Bestimmung von Graphenisomorphismen, was zu den NP-harten Problemen zählt. Die Eignung zweier neuronaler Netzwerktopologien für ein beschleunigtes, paralleles Konfigurationsverfahren wurde untersucht. Des Weiteren wurde eine Machbarkeitsstudie zu einem deklarativen Routing Ansatz durchgeführt. Zu diesem Zweck wurden boolesche Bedingungen hergeleitet, die strukturelle Eigenschaften einer neuromorphen Hardware und neuronaler Netzwerke beschreiben. In Folge dessen kann jeder Erfüllbarkeitsproblemlöser verwendet werden um Zuweisungen zu finden, die diese Bedingungen erfüllen. Abschließend wurde ein Simulator für neuromorphe Hardware um ein Leaky Integrate-And-Fire Neuronenmodell mit Adaption erweitert. Die Implementierung wurde durch Vergleich mit einem etablierten Simulator für neuronale Netzwerke verifiziert.

Feasibility Study On Declarative Routing For Neuromorphic Hardware

Finding a suitable configuration for neuromorphic hardware systems, closely resembling a given neural network, is a complex task. It is equivalent to finding graph isomorphisms, which is known to be an NP-hard problem. The suitability of two neural network topologies for an accelerated parallel configuration procedure has been analyzed. Furthermore, a feasibility study has been carried out covering a declarative routing approach. Therefore, boolean clauses have been derived to describe structural features of a neuromorphic hardware and neural networks. In consequence, any satisfiability solver can be used to find an assignment satisfying these clauses. Lastly, a neuromorphic hardware simulator has been extended by a leaky integrate-and-fire neuron model with adaptation. The implementation has been verified against an estabilished neural network simulator.

Contents

1 Introduction

Since the ancient times of the physicians Hippocrates and Galen the brain is said to be the source of the human conciousness and intelligence [Katz and Katz, 1962]. The invention of the microscope enabled studies of its elementary structures and marked the beginning of modern neuroscience [Cajal, 1906]. In an attempt to understand the mechanisms underlying the functioning of the brain, in vitro and in vivo measurements are performed [Logothetis et al., 2007, During and Spencer, 1993]. The huge quantity of neurons, about 10^{11} to 10^{12} in the human brain, and their connections render it impossible to record all activity in the brain simultaneously on a detailed level. The mathematical Hodgkin-Huxley model [Hodgkin and Huxley, 1952] has been developed to describe the behavior of biological neurons. Reduced models like the Adaptive Exponential Integrate-And-Fire model [Brette and Gerstner, 2005] mimic this behavior, but require less computational effort. These are applied in computational neuroscience to simulate neural networks. Still, the level of detail and the scale of simulations in is mainly limited by computational power and energy consumption. The IBM BlueGene supercomputer Dawn consumes about 1.4 MW to simulate a simplified model of the cat brain [Wang et al., 2010, Douglas, 2011]. The human brain uses approximately 20 W.

Neuromorphic hardware [Douglas et al., 1995] physically implements models of neurons and synapses. The Hybrid Multiscale Facilty (HMF) developed within the BrainScaleS project integrates 384 mixed signal High Input Count Analog Neural Network (HICANN) chips [Schemmel et al., 2010] as wafer scale system. Each chip implements up to 512 analog neurons. Multiple HICANNs can be interconnected to form models of vast neural networks.

The executable system specification (ESS) is a software simulation of the HMF hardware system. It allows for analyzing the impact of hardware imperfections and development parameters on to neural networks in more detail. Within this thesis (Section [chapter][2][]2) a numerical neuron model has been implemented. In Section [chapter][3][]3 the partitioning of neural networks is analyzed. Partitioning enables a hardware configuration procedure to operate in parallel. Most importantly, a feasibility study on satisfiability routing on neuromorphic hardware has been carried out in Section [chapter][4][]4.

2 AdEx Neuron Model Implementation

As part of the Executable System Specification (ESS) two neuron models are available for simulations. One model is based on the Adaptive Exponential Integrate-and-Fire (AdEx) neuron model, in the original publication introduced as aEIF [Brette and Gerstner, 2005, Touboul, 2008]. In the framework of this thesis, the second model is derived from this model and implemented.

The AdEx model is presented in Section [section][1][2]2.1 and is implemented in the form described in Section [section][2][2]2.2. A restriction arising with this implementation is explained in Section [section][3][2]2.3, which is solved by the implementation of the model derived in Section [subsection][1][2,3]2.3.1. This model is verified by comparison to a reference simulation in Section [subsection][2][2,3]2.3.2.

2.1 AdEx Model

The AdEx model combines a leaky integrate-and-fire (LIF) neuron with an exponential spiking mechanism and adaptation. In contrast to detailed neuron models such as the Hodgkin-Huxley model [Hodgkin and Huxley, 1952], the AdEx model is strongly reduced. It still is accurate compared to more detailed models, especially for neurons in high-conductance states [Brette and Gerstner, 2005]. The developing of the membrane potential V of the neuron and its adaptation current w depends on parameters, namely membrane capacitance C, leak conductance g_L, leak reversal potential E_L, slope factor Δ_T, spike threshold V_T, reset potential V_R, reset threshold potential V_S, injected current I, adaptation time constant τ_w, sub-threshold adaptation a and the spike triggered adaption constant b. When the membrane potential V reaches the reset threshold V_S, a spike is emitted, it is reset to $V = V_R$ and the adaptation current is set to $w = w + b$. The dynamics of V and w between a reset and the emission of the next spike are described by the following two differential equations:

$$C\frac{dV}{dt} = \underbrace{-g_L(V - E_L)}_{\text{Leakage}} \underbrace{+g_L\Delta_T \exp\left(\frac{V - V_T}{\Delta_T}\right)}_{\text{Exponential}} \underbrace{-w}_{\text{Adaptation}} \underbrace{+I}_{\text{Injected Current}}, \tag{2.1a}$$

$$\tau_w\frac{dw}{dt} = a(V - E_L) - w. \tag{2.1b}$$

The dynamics of a LIF neuron can be obtained by setting $\Delta_T = w = a = b = 0$.

2.2 Dimensionless Model

A dimensionless set of equations was chosen for the implementation of the AdEx model to reduce the number of parameters from 9 down to 4. The parameters left are $\bar{I}, \bar{\tau}_w, \bar{a}, \bar{b}$. Their definitions are listed in Table [table][1][2]2.1. The corresponding differential equations are reduced to

$$\frac{d\bar{V}}{d\bar{t}} = -\bar{V} + \exp(\bar{V}) - \bar{w} + \bar{I}, \tag{2.2a}$$

$$\bar{\tau}_w \frac{d\bar{w}}{d\bar{t}} = \bar{a}\bar{V} - \bar{w}. \tag{2.2b}$$

The dimensionless AdEx model is equivalent to the form shown in **????** [Touboul, 2008]. The reduced form requires less calculations per integration cycle, increasing the overall performance of the simulation. For the simulation, the parameters are transformed into their dimensionless equivalents. With these dimensionless parameters and **????** the simulation is computed by numerical integration. For evaluation, the computed results are transformed back.

2.3 Slope Factor Problem

The slope factor Δ_T is utilized as a scaling factor in the numerator of voltage and current transformations into dimensionless quantities, as can be seen in Table [table][1][2]2.1. Setting the slope factor $\Delta_T = 0$, in order to disable the exponential term of the AdEx model, leads to several undefined parameter transformations. This can be circumvented by choosing a value for $\Delta_T \approx 0$. Since $\lim_{\Delta_T \to 0} \Delta_T \exp(\frac{1}{\Delta_T}) = \infty$, the exponential term is increased, so that spiking occurs instantly at $V = V_T$. A disadvantage of this workaround is, that the now unnecessary exponential term is still evaluated by the numerical integrator. A better solution is to remove the exponential term completely, so its evaluation is skipped.

2.3.1 LIF Neuron Model With Adaptation

The model of the LIF neuron with adaption is very similar to the AdEx model. The exponential term is removed and the slope factor Δ_T becomes dispensable. The new model is now described by the following differential equations:

$$C\frac{dV}{dt} = -g_L(V - E_L) - w + I, \tag{2.3a}$$

$$\tau_w \frac{dw}{dt} = a(V - E_L) - w. \tag{2.3b}$$

The transformations have to be readjusted to get the dimensionless equations of this model as shown in Table [table][1][2]2.1.

By applying the transformations of Table [table][1][2]2.1 on the LIF model the following dimensionless differential equations are obtained:

$$\frac{d\bar{V}}{d\bar{t}} = -\bar{V} - \bar{w} + \bar{I}, \tag{2.4a}$$

$$\bar{\tau}_w \frac{d\bar{w}}{d\bar{t}} = \bar{a}\bar{V} - \bar{w}. \tag{2.4b}$$

2.3.2 Verification Of The Implementation With Reference Simulator

In order to verify the implemented model, it is compared to an AdEx model simulated with the NEURON software [Hines and Carnevale, 1997]. The slope factor parameter of the NEURON instance is set to $\Delta_T = 0$. The plot in Figure [figure][1][2]2.1 shows the expected equal developing of membrane potential of both neuron models in an exemplary simulation. There still is a numerical difference between the two voltage traces. For a time step $t = 0.1\,\mu$s the maximum relative difference observed is < 0.02 for the simulation shown in Figure [figure][1][2]2.1. This is a result of both implementations using discrete time steps, but different binning and different integrator methods. The NEURON simulator uses the Euler method [Butcher, 2003] by default, the model implemented here uses the fourth-order Runge-Kutta method [Press et al., 2007]. The observed numerical discrepancy decreases for smaller time steps. The same error is observed for the synaptic conductances, but because they are of small order they do not affect spike timing or even spiking frequency.

AdEx	LIF With Adaptation
$\bar{V}(\bar{t}) = \frac{V(t)-V_T}{\Delta_T}$	$\bar{V}(\bar{t}) = \frac{V(t)-V_T}{V_T-V_R}$
$\bar{t} = \frac{g_L t}{C}$	
$\bar{w}(\bar{t}) = \frac{w(t)+a(E_L-V_T)}{g_L \Delta_T}$	$\bar{w}(\bar{t}) = \frac{w(t)+a(E_L-V_T)}{g_L(V_T-V_R)}$
$\bar{I} = \frac{I}{g_L \Delta_T} + \left(1 + \frac{a}{g_L}\right) \frac{E_L-V_T}{\Delta_T}$	$\bar{I} = \frac{I}{g_L(V_T-V_R)} + \left(1 + \frac{a}{g_L}\right) \frac{E_L-V_T}{V_T-V_R}$
$\bar{\tau}_w = \frac{g_L \tau_w}{C}$	
$\bar{a} = \frac{a}{g_L}$	
$\bar{b} = \frac{b}{g_L \Delta_T}$	$\bar{b} = \frac{b}{g_L(V_T-V_R)}$
$\bar{V}_R = \frac{V_R-V_T}{\Delta_T}$	$\bar{V}_R = -1$
$\bar{V}_T = \frac{V_T-V_T}{\Delta_T}$	$\bar{V}_T = 0$

Table 2.1: Transformations to dimensionless parameters; The transformations of the left column are defined for the AdEx model in Touboul [2008]. The transformations in the right column are readjusted for the LIF model with adaptation. The terms containing Δ_T in the AdEx model are replaced by terms scaled with $(V_T - V_R)$ for the LIF model. This results in $\bar{V}_R = -1$ and $\bar{V}_T = 0$.

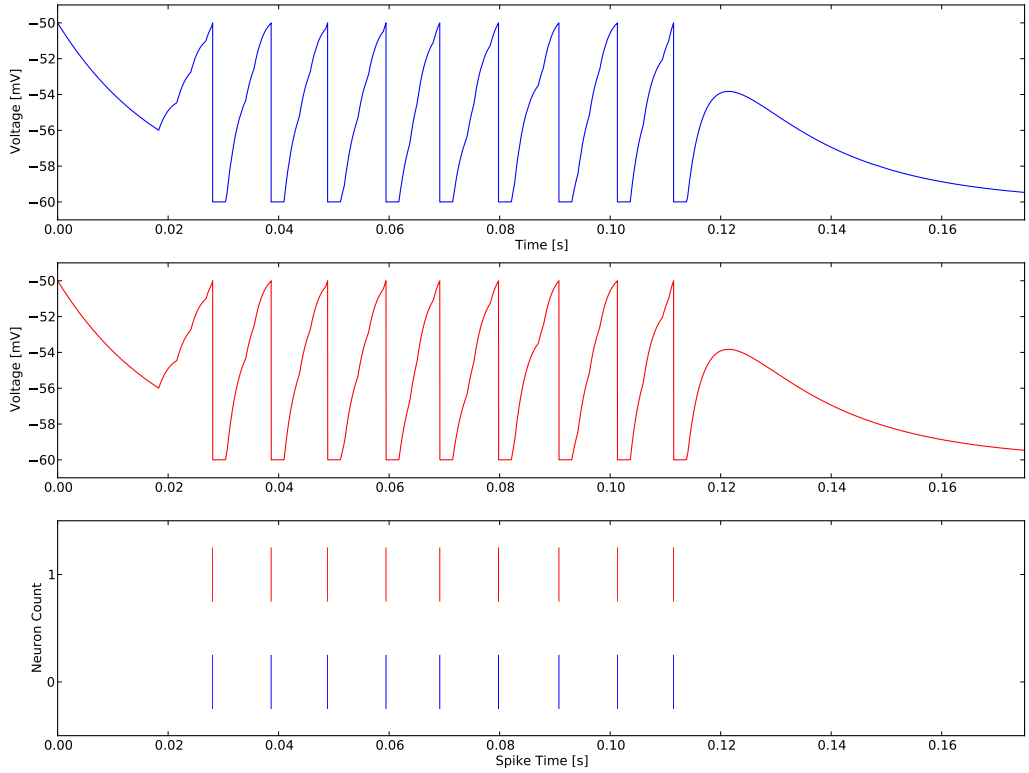

Figure 2.1: Comparison of implemented model with NEURON, simulation of 180 ms; The upper plot shows the voltage trace of the implemented neuron swayed by Poisson distributed input spikes in the first 120 ms. The middle plot shows the voltage trace of a simulation with NEURON for the same parameters and identical stimulation. The lower plot shows the output spikes of the simulated neuron. Neuron 0 shows the simulation of the implemented model, neuron 1 shows the simulation with NEURON. The neuron parameters are set to $C = 0.2\,\mathrm{nF}, g_L = 10\,\mathrm{nS}, E_L = V_R = -60\,\mathrm{mV}, V_T = -50\,\mathrm{mV}, \tau_w = 5\,\mathrm{ms}$.

3 Partitioning Of Neural Networks

In order to carry out any experiments on the HMF system, a configuration needs to be found, resembling the user-provided neural network as closely as possible. This network is presented as PyNN script [Davison et al., 2008], which contains a list of neurons and their synaptic connections. The task of an automated mapping tool is to allocate hardware components representing these neurons, which is called placement. In the next step, the mapping tool has to realize the synaptic connections between hardware neurons according to the input network, which is called routing. For complex hierarchical topologies like the HMF hardware and limited resources, mapping is a nontrivial problem [Liu et al., 2008, Safarpour et al., 2006, Melnik et al., 2002].

Parallel mapping onto disjoint areas of the hardware requires partitioning the network. Optimally, regions of the network with a high connection density are mapped closely together on the hardware to keep required routing resources low. Partitioning also allows to map loosely connected partitions almost independently from other partitions. In this chapter exemplary networks are analyzed regarding their suitability for such a parallel mapping approach.

In Section [section][1][3]3.1 and Section [section][2][3]3.2 two models are outlined in their structures and properties. In Section [section][3][3]3.3 these networks are analyzed for unweighted synaptic connections.

3.1 Layer 2/3 Attractor Memory Network

The *associative attractor memory paradigm* as a functional model for the neocortex has its origins in Hebb's theories of cell assemblies from over 50 years ago. Over time, there has been mounting experimental data [Fuster et al., 1971, Compte et al., 2000, Cossart et al., 2003] which shows clear correlations between persistent cortical activity and working memory, lending strong support to this paradigm. A challenging task lies in formulating a microscopic model on the level of individual neurons and synapses, with parameters constrained by experimental data, which is able to address multiple observed phenomena, such as memory dynamics, population coding and oscillatory behavior at the same time. One such model has been proposed by Lundqvist et al. [2006, 2010], which is of particular interest in the context of this work, as it exhibits a rather high robustness towards hardware-specific distortions

[Brüderle et al., 2011] and has also been successfully implemented, in a reduced form, on the neuromorphic Spikey chip [Brüderle et al., 2011, Pfeil et al., 2012].

3.1.1 Structure

The structure models a neocortical layer 2/3 circuit of columnar architecture. It consists of several modules called hypercolumns. A hypercolumn itself is formed by a number of minicolumns. Hypercolumns as well as minicolumns can be seen as functional units or modules. A minicolumn consists of three different types of cells, namely: basket cells, pyramidal cells and regular spiking non pyramidal (RSNP) cells. The structure of the network is illustrated in Figure [figure][1][3]3.1.

For the case of orthogonal patterns, each pattern is represented by one minicolumn in every hypercolumn. Pyramidal cells in minicolumns representing the same pattern but located in different hypercolumns are linked by excitatory synapses.

3.1.2 Emergent functions and features

The reaction of a minicolumn on excitatory input, whether it is external or from a minicolumn of the same pattern, is not only to excite other minicolumns of the same pattern, but to inhibit neighbouring minicolumns in its hypercolumn. This connectivity allows pattern recognition and pattern completion. The stimulation of a subset of minicolumns can suffice to activate all columns belonging to the same pattern. When becoming active, a minicolumn will switch into an UP state, where the average membrane potential of the pyramidal population is significantly above their resting potential and their firing rates are elevated.

When there is input for different patterns, the network at first behaves like a classical soft winner-takes-all (WTA) network [Kaski and Kohonen, 1994]. In contrast to a simple WTA network however, a single attractor cannot be permanently active. This is prevented by two mechanisms taking effect over time. First is, inter-pyramidal connections are subject to short term plasticity. Second, the excitability of pyramidal cells decreases with the emission of spikes. The cells adapt, so that the overall possibility for mutual excitation decreases over time. This results in the network switching between all stimulated patterns over time.

3.2 Small World Network

Many biological, technical and social networks can be modeled by *small world networks* [Watts and Strogatz, 1998]. They are characterized by the average path length

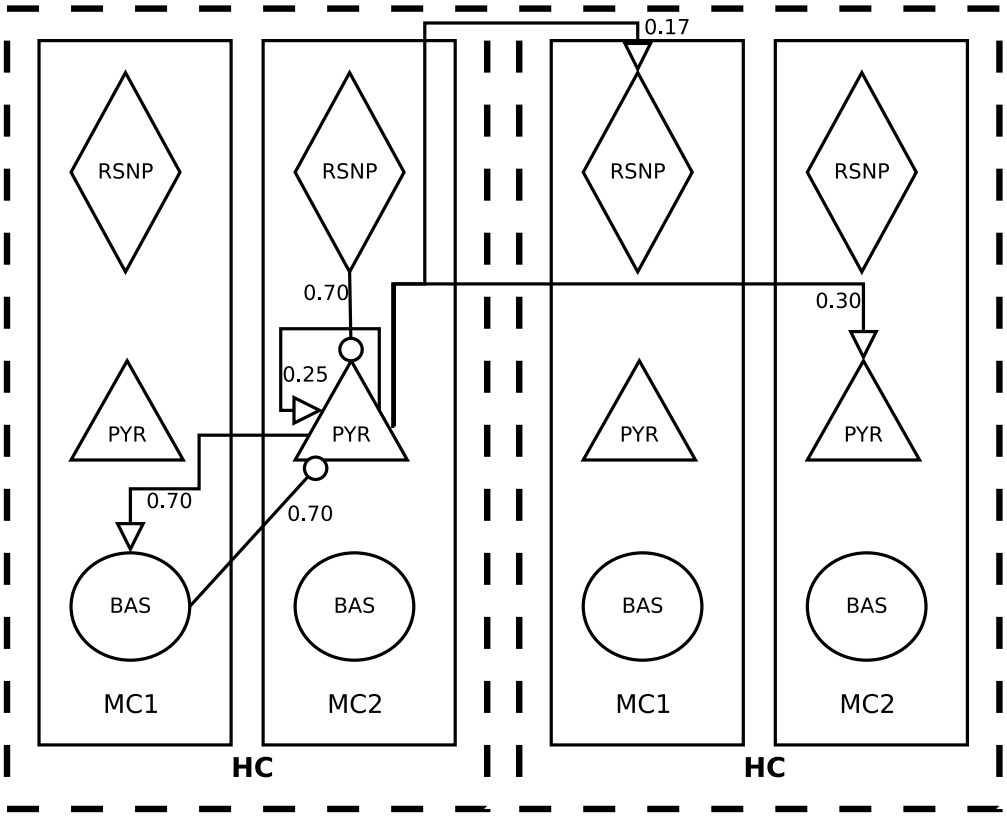

Figure 3.1: Outline of the L2/3 model; Each element represents a population of cells. Excitatory connections are marked by an arrow, inhibitory connections are marked by circles. Note that for every kind of connection only a single representative one is shown for demonstration purposes. The number next to a connection indicates the probability of a connection for two neurons of the linked populations.

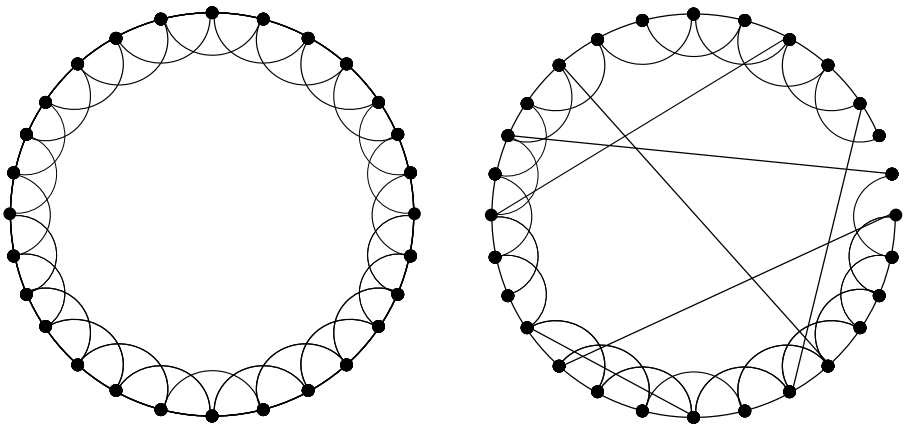

Figure 3.2: Scheme of a regular ring network (left) and a small world network (right); By rewiring the connections of a regular network with the probability p, a small world network can be created. Only a few rewired connections suffice to decrease the average path length l significantly, because they function as shortcuts in the network.

l between the nodes of the networks. A network like the ring network shown in Figure [figure][2][3]3.2, with a regular short ranged connection pattern and therefore a high average path length is called an ordered network [Watts and Strogatz, 1998]. On the other hand, networks with random connectivity are very likely to have a small average path length. A transition between ordered and random networks is created by starting with an ordered network and rewiring every connection with the probability p. The resulting networks are small world networks like the one shown in Figure [figure][2][3]3.2.

Networks of different randomness can be easily produced and analyzed by sweeping the rewiring probability p from ordered $p = 0$ to random $p = 1$, thus enabling observation of the transition between such ordered and random networks. This property is exploited for the partitioning analysis shown in Figure [figure][4][3]3.4 on the right.

3.3 Optimal Set Analysis

In order to find partitions of densely intra-connected neurons in neural networks, *optimal set analysis* can be applied [Hilgetag et al., 2000]. For optimal set analysis, a cost function is defined which decreases for partitions with a high number of connections within partitions and low number of connections between neurons of different partitions. The cost function is minimized by iterating with an evolutionary algorithm. With that technique Hilgetag et al. [2000] showed that macaque and cat cortices are organized in densely intra-connected partitions, suggesting that structure and function are closely linked at this system level.

Instead of using an evolutionary algorithm optimal set analysis can be performed with an edge cut minimizing graph partitioning tool like METIS [Karypis and Kumar, 1998]. The edge cut is the wighted sum over connections with origin and target in different partitions. Since the analysis presented deals with unweighted connections, the edge cut reduces to the number of these connections. METIS allows to sweep over the number of partitions in which the network is split and balances the partition sizes while minimizing edge cut.

3.3.1 Ideal Case

The most simple ideal case for partitioning a network, is a network consisting of densely intra-connected partitions of neurons with no connections between neurons of different partitions at all. This case may not be relevant for application and does not need partitioning, because the partitions can be seen as separate and unrelated networks. On the other hand, this is the best case for demonstrating partitioning analysis with METIS. An exemplary network, consisting of five neuron populations is analysed. There are random connections within each of the five populations, but no connections between neurons of different populations. So it is expected that the analysis shows a minimum of zero edge cut for a partition number of five, which can be seen in Figure [figure][3][3]3.3. Also local minima can be detected for multiples of five, so it should be kept in mind, that the lowest of local minima in the partition analysis can hint to an optimal partition size as seen in Figure [figure][5][3]3.5.

3.3.2 Worst Case

A worst case example for this kind of analysis is a network with random connectivity. In a random network, assuming it being significantly large, the edge cut of equally sized partitions does not depend on the assignment of neurons to the partitions. Instead, the edge cut n_{EdgeCut} depends on the statistical mean of connections per neuron $\frac{n_{\mathrm{Edges}}}{n_{\mathrm{Neurons}}}$. For a higher number of partitions, the overall edge cut will exponentially converge to the number of connections. The maximum edge cut $n_{\mathrm{EdgeCut}max} = n_{\mathrm{Edges}}$ is reached for the number of partitions $n_{\mathrm{Partitions}} = n_{\mathrm{Neurons}}$. For the edge cut per partition $\frac{n_{\mathrm{EdgeCut}}}{n_{\mathrm{Partitions}}}$ it therefor can be assumed that a minimization of the edge cut n_{EdgeCut} on random networks is described by the superposition of two functions. An exponentially decrease is superposed by a linear reciprocal decrease converging to the mean of connections per neuron $\frac{n_{\mathrm{Edges}}}{n_{\mathrm{Neurons}}}$ with the number of partitions $n_{\mathrm{Partitions}}$ approaching the cell count n_{Neurons}:

$$\lim_{n_{\mathrm{Partitions}} \to n_{\mathrm{Neurons}}} \frac{n_{\mathrm{EdgeCut}}(n_{\mathrm{Partitions}})}{n_{\mathrm{Partitions}}} = \frac{n_{\mathrm{EdgeCut}}(n_{\mathrm{Neurons}})}{n_{\mathrm{Neurons}}} = \frac{n_{\mathrm{Edges}}}{n_{\mathrm{Neurons}}}$$

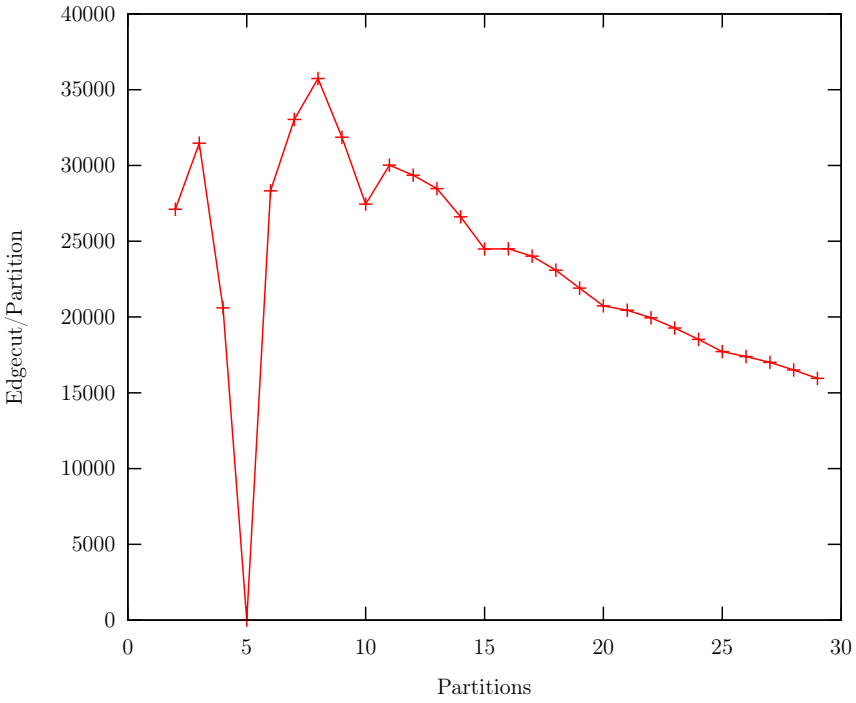

Figure 3.3: Sweep with METIS partitioning tool for a network consisting of five separated partitions; Cell count $n_{\text{Neurons}} = 4,750$, edge count $n_{\text{Edges}} = 563,383$, sweep computation wall clock time $= 15\,\text{s}$

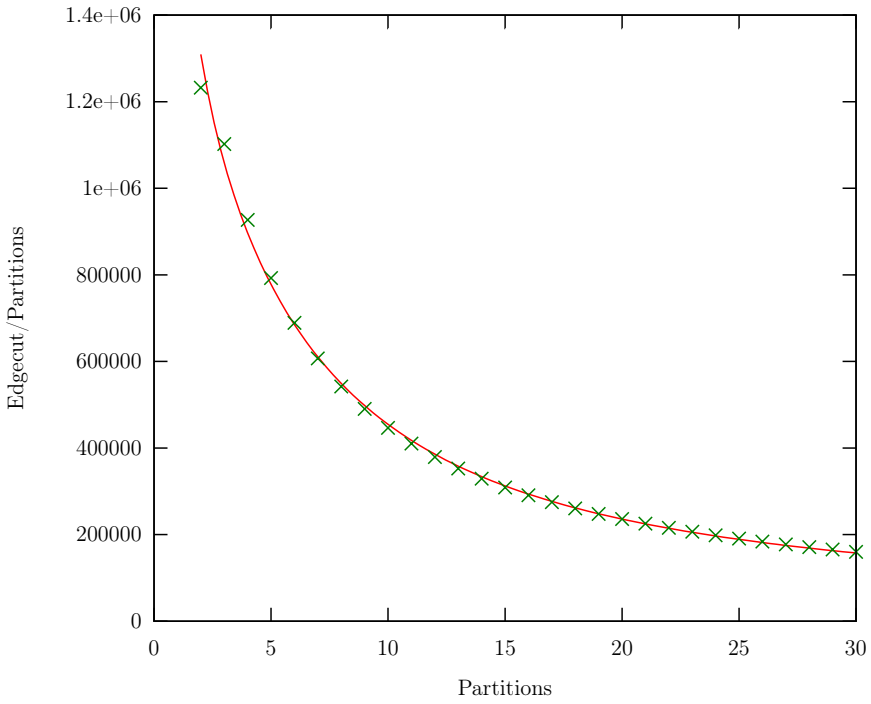

Figure 3.4: Sweep with Metis partitioning tool for a random network with the connection probability $p = 0.1$ for one neuron to another. Cell count $n_{\text{Neurons}} = 10,000$, edge count $n_{\text{Edges}} = 4,999,417$, sweep computation wall time = 9 m 15 s. The function plotted is a fit of $f(x) = \frac{d - a\exp(-bx+c)}{x}$ with the values $a = 19753804$, $b = 0.30741971$, $c = -1.6223542$, $d = 4727134.0$.

This behavior is modeled by the function

$$f(x) = \frac{d - a\exp(-bx + c)}{x},$$

and can be observed in Figure [figure][4][3]3.4. A fit of the function supports the assumption with $f(n_{\text{Neurons}}) = 473 \approx 500 = \frac{n_{\text{Edges}}}{n_{\text{Neurons}}}$.

There is no particular number of partitions providing a significantly low edge cut for a reasonable number of partitions, $n_{\text{Partitions}} \leq 30$. It can be concluded that if a network is similar to this random network, searching for a minimum of edge cut is not of advantage for the successive mapping procedure.

3.3.3 Analyzing Network Models

The structural properties of two types of networks are tested for their suitability for partitioning for the purpose of parallel mapping. The first one is the layer 2/3

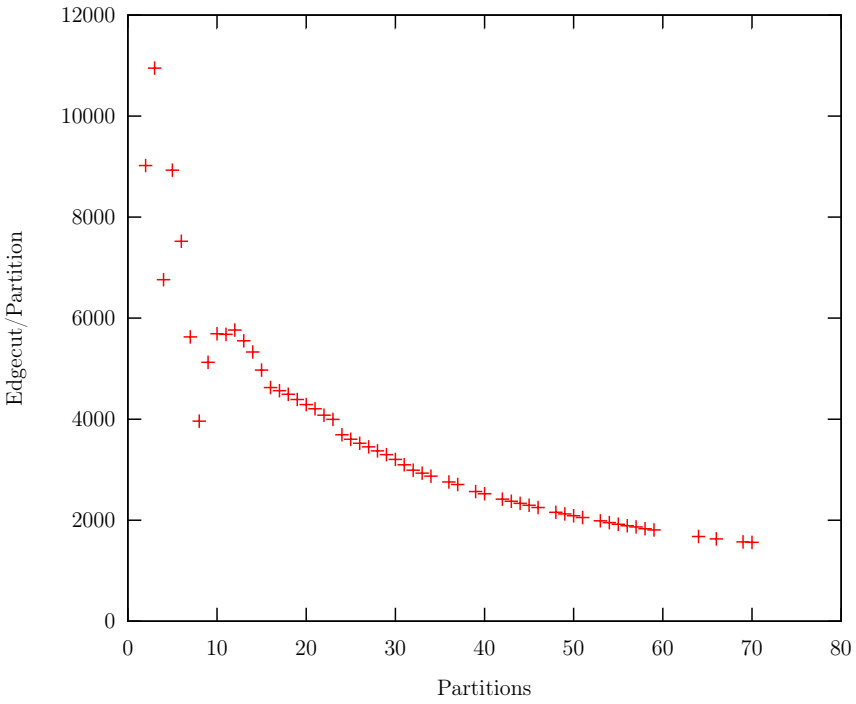

Figure 3.5: Sweep with Metis partitioning tool for the KTH model network, as explained in Section [section][1][3]3.1; Gaps in the data are a result of abnormal termination of Metis for that particular number of partitions. Cell count $n_{\text{Neurons}} = 2,376$, edge count $n_{\text{Edges}} = 120,510$, sweep computation wall time = 31 s

attractor memory network model described in Section [section][1][3]3.1. The second one are small world networks as seen in Section [section][2][3]3.2.

The analyzed implementation of the layer 2/3 network was generated with a Python script connecting neurons with the probabilities specified in Figure [figure][1][3]3.1. The network consists of 9 hypercolumns, each containing 8 minicolumns. A minicolumn consists of 30 pyramidal, 2 RSNP and 1 basket cell. This results in a total count of 2,376 cells.

In the analysis, plotted in Figure [figure][5][3]3.5 local minima can be observed for a partition count of 8 and its multiples. As in Section [subsection][1][3,3]3.3.1 the optimal partition count is the lowest of the local minima, which in this case is 8. The overall edge cut reaches the global minimum for this number of partitions and is at $3,950 \cdot 8 = 31,600$.

The plot in Figure [figure][6][3]3.6 of the analysis of small world networks shows that even for an inherently small rewiring probability $p = 0.2$ the edge cut per partition develops qualitatively towards the random network analyzed in Figure [figure][4][3]3.4. Approximately 10% of the overall connections are shared connec-

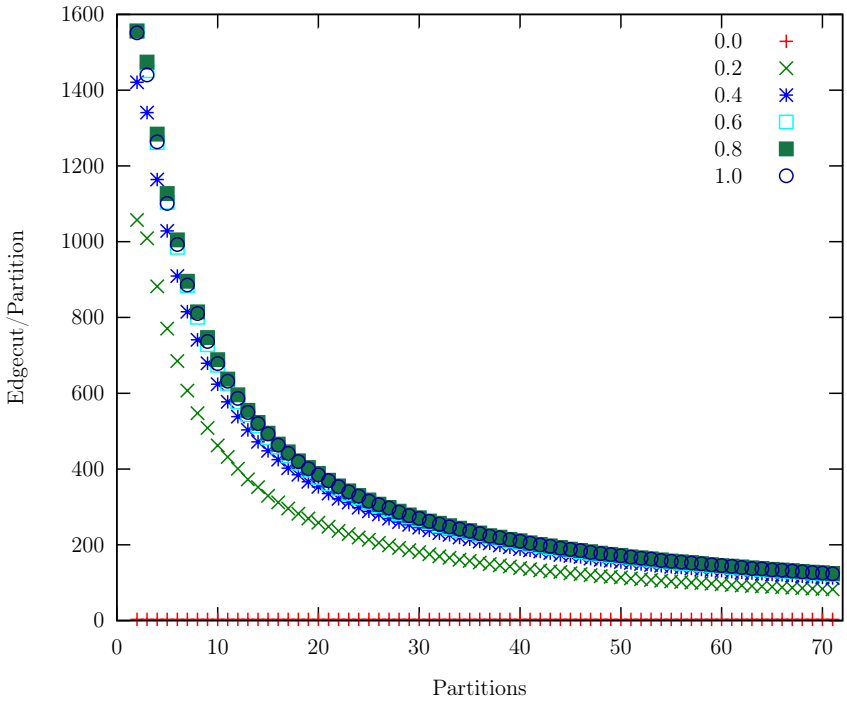

Figure 3.6: Sweeps with METIS for the small world network model; Sweeps for different rewiring probabilities p. Cell count $n_{\text{Neurons}} = 10,000$, edge count $n_{\text{Edges}} = 20,000$, sweep computation wall clock time = 40 s for the whole plot, including five adjustments for $p \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ which are not displayed.

tions for a number of partitions $n_{\text{Partitions}} \leq 4$. This ratio increases exponentially for a higher number of partitions.

3.4 Discussion and Outlook

With a high number of partitions, the edge cut per partition reduces in the analyzed networks. A draw back of a high number of partitions is the increasing total number of dependencies between partitions. In order to achieve feasible parallelization without creating an obstructive overhead of communication between the parallel processes, the number of partitions has to be chosen as a compromise the grade of parallelization and the resulting dependencies.

It becomes apparent that random networks are pathological for the mapping process, but also unsuited for the hardware, because no placement can be found such that routing resources suffice for large networks.

For networks of a distinct structure, like the analyzed layer 2/3, it is shown that preferable numbers of partitions can be found. In future, this can be utilized for a parallelized mapping procedure.

4 SAT Routing

An approach to the mapping problem mentioned in Section [chapter][3][]3 is to transform it into a satisfiability (SAT) problem [Wood and Rutenbar, 1997, Safarpour et al., 2006]. The constraints caused by limited hardware resources are expressed by a set of boolean variables and expressions. A generic SAT solver can be used to map a neural network onto the hardware. A process flow embedding this approach is presented in Section [section][1][4]4.1. After describing the SAT problem and solvers in Section [section][2][4]4.2 and introducing the relevant hardware topology, the concept is demonstrated for an exemplary routing problem in Section [section][5][4]4.5.

4.1 Mapping Flow

A SAT solver based mapping flow consisting of a placement and routing is described in the following.

The task of the placement is to determine the exact location of a neuron on the hardware to provide the assignment of the output of the neurons into the routing net for the next mapping step.

The routing allocates lanes and realizes the required connectivity on a wafer scale level. Depending on the number of required connections and the density of neuron placement, this task can be of high complexity. The highly configurable system allows many possible realizations of a connection between neurons. Utilization of a SAT solver is especially suitable at this point, since it is their purpose to compute assignments while considering a significant number of dependencies.

This thesis provides the rudimentary implementation of a SAT based HICANN routing (Section [section][6][4]4.6). With the HICANN being the major building block of the HMF system, this can also be considered a starting point for a future wafer and multi wafer routing.

4.2 The Satisfiability Problem

The SAT problem is to determine whether there exists an assignment for the variables if a given boolean expression, so that the expression evaluates to `true`.

Every boolean expression can be transformed so that it consists of a collection of clauses which are in conjunctive normal form (CNF). A clause in its conjunctive normal form consists of literals linked by a logical `or`. A literal is the representation of a variable or its negation[1]. A boolean expression in CNF can be formed by a single clause in its CNF or multiple clauses in CNF linked by a logical `and`. So a CNF clause can be written as

$$\sum_i c_i v_i \geq 1, \qquad \text{with} \quad c_i \in \{-1, 1\}, v_i \in \{0, 1\},$$

where a sum represents a logical `or`.

The SAT problem is defined as [Garey and Johnson, 1979]:

Given a set V of boolean variables $v_i \in V$ and a function $t : V \to \{\texttt{true}, \texttt{false}\}$. If $t(v_i) = \texttt{true}$ then v_i is said to be true. If $t(v_i) = \texttt{false}$ then v_i is said to be false. For every variable v_i there are two literals v_i and $\bar{v}_i$. The literal v_i is true under t only if the variable v_i is true under t. The literal $\bar{v}_i$ is true under t only if the variable v_i is false under t.
A clause over V is a set of literals over V. The clause is *satisfied* under t only if at least one of the literals is true under t. A collection of clauses C over V is only satisfiable if there exists an assignment for V so that every clause in C is satisfied simultaneously.
Problem:
Given a set V of variables and a collection C of clauses over V, is there a satisfying assignment for C?

SAT problems are distinguished by their grade k. The grade of a SAT problem is determined by the maximum number k of different variables in one of its clauses. Problems of the grade $k = 2$ are called 2SAT problems, belong to the class of NL complete problems and can be solved in linear time [Aspvall et al., 1982]. SAT problems of the grade $k \geq 3$ belong to the class of NP complete problems along with the *Travelling Salesman* [Garey and Johnson, 1979, ch. 2.1] or *Multiprocessor Scheduling* [Garey and Johnson, 1979, ch. 2.1.3]. In fact, the first problem that was shown to be NP complete is the 3SAT problem [Cook, 1971].

Today SAT solvers are used for industrial applications of the SAT problem, like electronic design automation (EDA) [Moskewicz et al., 2001]. There are several well developed open source SAT solvers available. A good overview of the recent developments and capabilities of SAT solvers can be found within the benchmarking results of the annual SAT competition [Competition, 2012]. Here solvers compete by running benchmarks in multiple categories, such as real-world applications, challenging hard combinatorial problems and parallel solving. The .cnf format has been specified by Center For Discrete Mathematics And Theoretical Computer Science (DIMACS)[DIMACS] and is used as the common input file standard. It is a

[1]The negation either is denoted by a ¬ preceding the literal or a bar over the literal.

text file stating one clause per line. Since the only junction within a clause can be a logical *or*, it is simply not noted. What remains is a listing of numbers representing the variables, which can be negated by a preceding minus sign "-". The end of a clause is always indicated by a `0`. Variable names begin at `1` and are numbered continuously.

For this thesis the following two SAT solvers have been used.

MiniSat

MiniSat 2.2.1 [Eén and Sörensson, 2012] is an open source, minimalistic SAT solver, which won several awards in the SAT competitions. It is compatible to solve problems presented in the .cnf format. With MiniSat+ a SAT solver for problems formulated with pseudo-boolean variables (Section [section][7][4]4.7) is available. With MiniSat+ pseudo-boolean problems can either be solved directly or converted into the .cnf format, which then can be passed on to MiniSat or any other CNF compatible SAT solver.

Sat4j

Sat4j [Le Berre and Parrain, 2010] is an open source Java library for solving SAT problems with a collection of stand-alone Java SAT solvers for a variety of applications. These applications include problems expressed in the DIMACS .cnf format and also a pseudo-boolean SAT solver. Since 2008 the Eclipse Java IDE relies on Sat4j to solve its software dependencies [Le Berre and Parrain, 2012]. The software can be downloaded from `www.sat4j.org`.

4.3 Topology Of The HICANN

The layer 1 communication is the topology on which a HICANN routing tool operates. It consists of 64 horizontal lanes, leading through two crossbar switch matrices and 254 vertical lanes connecting the synapse switch matrices with the crossbar switch matrices [Schemmel et al., 2012]. Spike signals arrive in the repeater blocks at the interface of the HICANN and can be routed along the lanes. Signals either run through the HICANN to another repeater block or target a synaptic driver within the HICANN. Signals to the synaptic drivers have to be lead through a connected synapse switch matrix. As the name suggests, switch matrices are a grid of switches. By activating a switch, the two lanes crossing at the switches location are connected. This is illustrated in Figure [figure][1][4]4.1.

The synapse drivers serve as inputs for the neuron circuits in the ANNCORE (Figure [figure][2][4]4.2). In order to route a signal to the target hardware neuron, the corresponding synapse driver has to be connected. Of the 64 horizontal lanes there

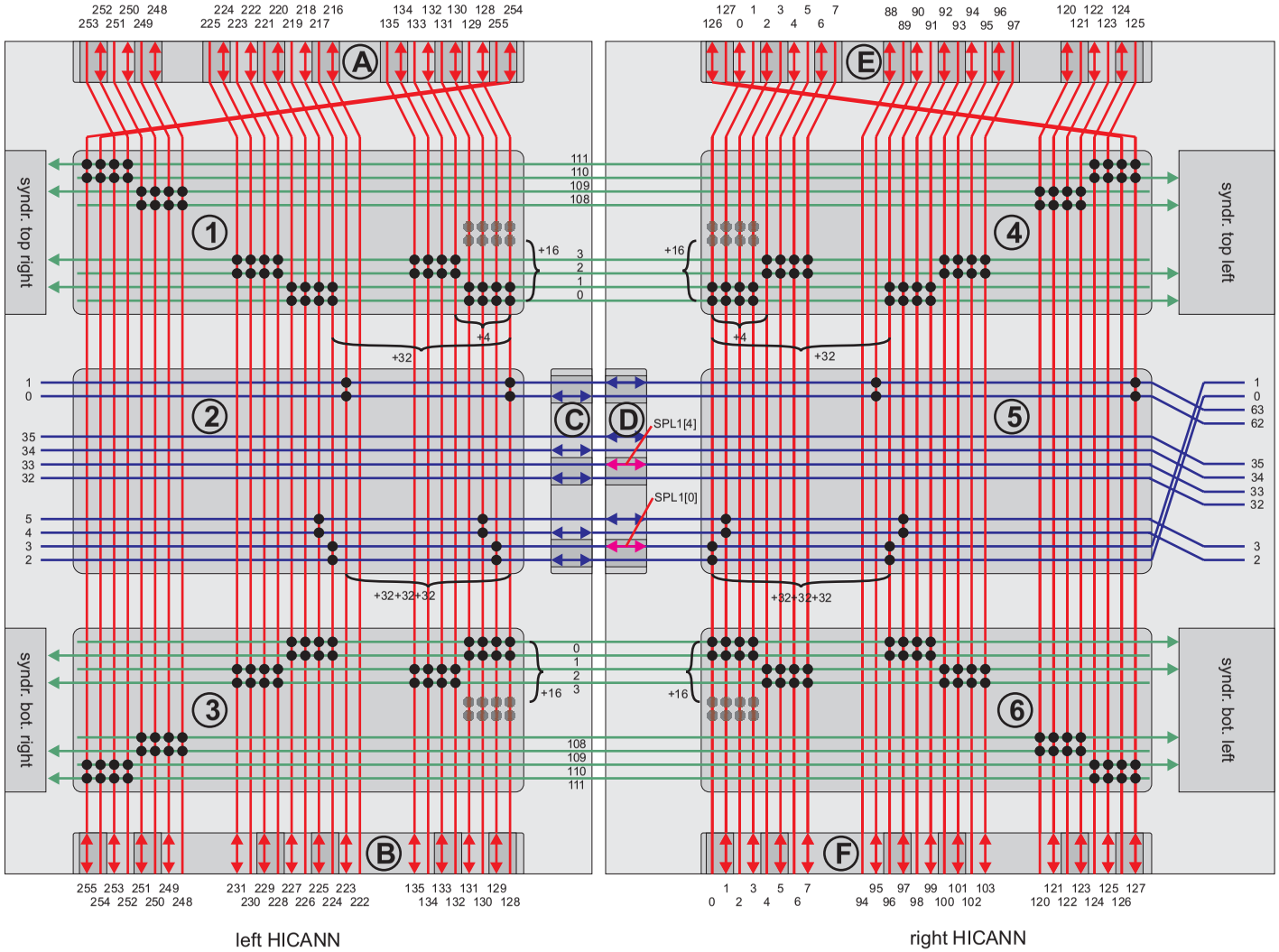

Figure 4.1: Layer 1 topography on HICANN by courtesy of Dr. Andreas Grübl [Schemmel et al., 2012]; The blocks marked 1,3,4 and 6 are called synaptic switch matrices, the blocks marked 2 and 5 are called crossbar switch matrices. The black dots in the switch matrices represent the switches. Blocks marked with letters are repeater blocks and indicate the interface of the HICANN.

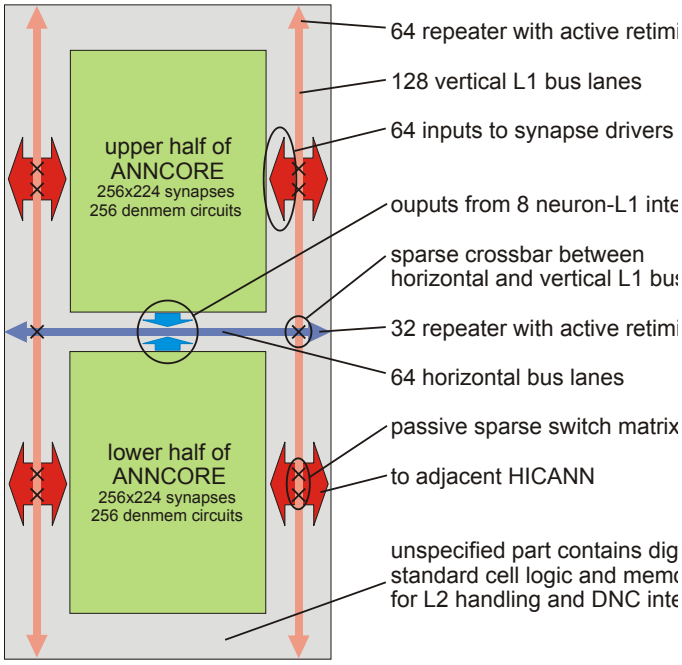

Figure 4.2: HICANN schematic with ANNCOREs by courtesy of Dr. Johannes Schemmel [Schemmel et al., 2012].

are 8 fixed lanes serving as signal outputs for the neurons of the HICANN. The spike signals of the up to 512 neurons are merged onto these 8 lanes, so these lanes are the beginning of routes. Between two HICANNs the lane positions are shifted by two as can be seen in Figure [figure][1][4]4.1 to facilitate a straight horizontal routing and usage of all lanes, but this is not of significance for a routing limited to a single HICANN environment.

Through this layer 1 communcation the HICANNs on the wafer are connectable. A further connectivity can be enabled by the utilization of the layer 2 communication for routes across a wafer, between two wafers or to the systems external interface. Within this feasibility study a HICANN SAT routing for the layer 1 communication is developed, disregarding the layer 2 communication The focus for this is on hardware constraints caused by the limited number of lanes and their connectivity. I.e. due to the sparse switch matrices it is not possible to connect all horizontal lanes to an arbitrary vertical lane and vice versa.

4.4 FPGA Routing

A comparing look at the mapping for a field programmable gate array (FPGA) and mapping of neural networks reveals that the process of mapping logic instructions onto an FPGA bears similarities with configuring the HMF system. An FPGA

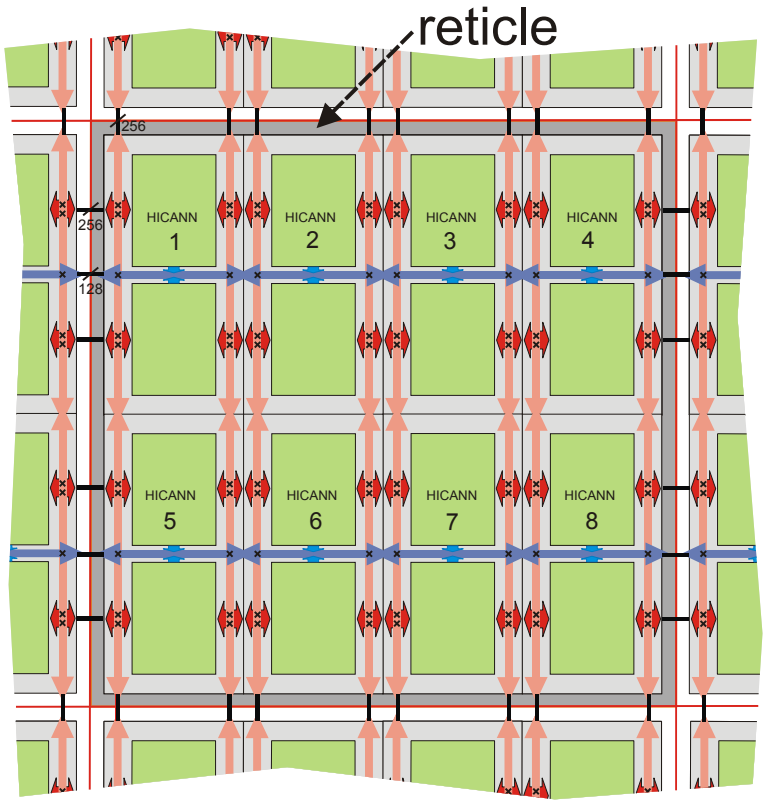

Figure 4.3: Wafer-scale connections of a reticle by courtesy of Dr. Johannes Schemmel [Schemmel et al., 2012]; The wafer is made of repetitions of the illustrated reticle pattern. The density of connections between HICANNs on different reticles are the same as for HICANNs on the same reticle.

HMF System	FPGA
ANNCORE	Logic Block
Crossbar Switch Matrix	Routing Switch
Synapse Switch Matrix	Connection Block

Table 4.1: HMF hardware components for layer 1 communication routing and their comparing equivalents on an FPGA with island style architecture.

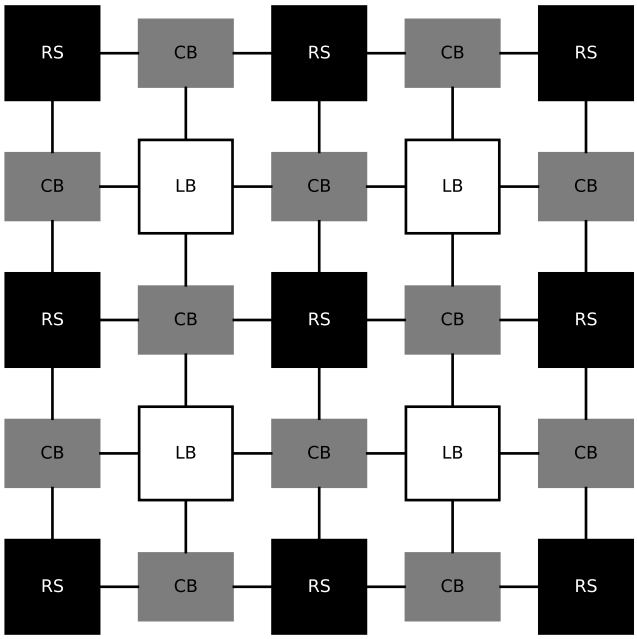

Figure 4.4: Island Style FPGA layout with logical blocks LB connection blocks CB and routing switches RS. The logical blocks form islands in the sea of lanes of the connection blocks and routing switches.

is an array of identical logical units. These units and their wiring are configured in order to provide a specific functionality. The wiring for FPGAs with an island style layout [Betz and Rose, 1999], illustrated in Figure [figure][4][4]4.4, is done by toggling connection block and routing block switches. This is similar to the mapping of neural networks onto the HMF system, where neurons have to be placed and their connections have to be routed. In both cases a software mapping tool has the task to optimally administer hardware resources. The logic blocks are linked by connection blocks like ANNCOREs by synapse driver switches. Routing switches in this layout lead signals similar to the crossbar switches of the HICANN. Therefore strategies similar to the ones for solving the mapping problem for FPGAs might also be applicable for the HMF system. The tools implementing the mapping process for FPGAs are provided by hardware vendors and licenced as proprietary software [Xilinx, 2012], rendering it impossible to find adaptable mapping solutions among these tools.

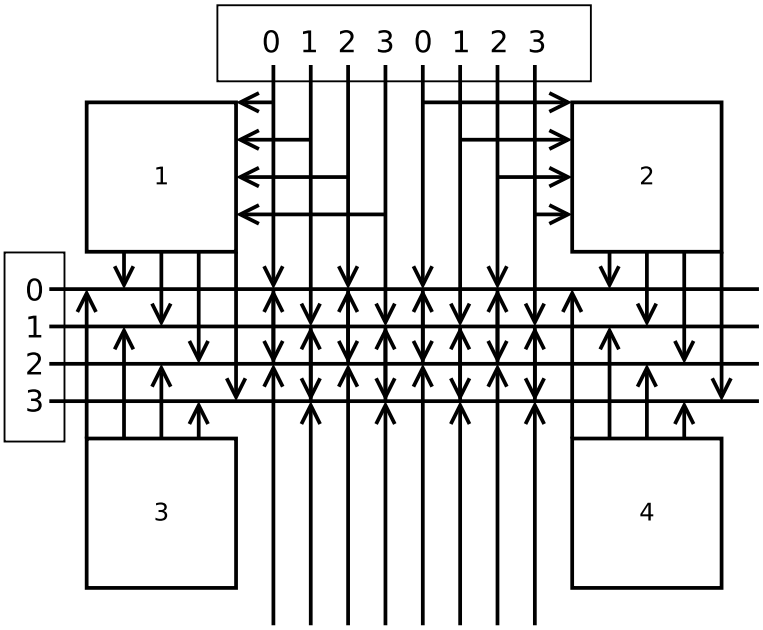

Figure 4.5: Testing environment for SAT routing example; The incoming and outgoing arrows at the numbered boxes represent inputs and outputs accordingly. Other arrows indicate switchable connections between horizontal and vertical lanes. There are one horizontal and two vertical bunches of four lanes available. The outputs of the blocks lead to the horizontal lanes, the inputs from the vertical lanes to the blocks. A solution of the routing task dealt with in the presented example is shown in Figure [figure][6][4]4.6.

It has been shown in Nam et al. [2002] that once routing problems on FPGAs are formulated in CNF they can be solved as a SAT problem. The actual challenge is finding formal descriptions for the defining hardware and software constraints. Hardware constraints, like the ones described in Section [section][3][4]4.3, express the topology of the routing environment, e.g. the number of lanes available and how they are connectable. Software constraints express the connectivity of the neural network to mapped, e.g. between which points connections have to be established.

4.5 SAT Routing Example

To explain the conceptual process of routing with SAT solvers, an exemplified routing task is performed. This is done by expressing the constraints in a CNF stored in the .cnf format and applying a SAT solver. The employed routing test environment is shown in Figure [figure][5][4]4.5.

The task is to establish two routes between specified points. For this example, routing consists of assigning the routes to the available lane resources to satisfy the routing goals. Here the goal is to install a first route from the outputs of block 1 to the inputs of block 2 and a second one from the outputs of block 3 to the inputs of block 2. Routes may follow the available lanes, but they must not use lanes already used by other routes.

Nam et al. [2002] describes a strategy to prepare a routing problem for SAT solvers similar to the following one:

1. Formulate the problem in formal logic expressions
2. Substitute non-boolean variables with boolean ones
3. Convert expressions into CNF

For the given example this strategy is applied in the next sections.

4.5.1 Logic Expressions

Formulating logic expressions is the first step of preparing a routing problem to be solved with a SAT solver. For the given exemplary problem, this can be divided into three subproblems. The first one is to introduce a set of non-boolean variables for both routes. A variable represents the lane allocated for its route in one specific section. The values of the variables range from 0 to 3, since there are four lanes per section. For this setup there are four sections per route, namely the outputs, the horizontal lanes, the vertical lanes and the inputs. To define the routes a set of four variables each is required, resulting in eight variables

$$\text{Route } 1 = \{\text{OUT1}, \text{HL1}, \text{VLR1}, \text{IN1}\},$$
$$\text{Route } 2 = \{\text{OUT2}, \text{HL2}, \text{VLR2}, \text{IN2}\}.$$

The second subproblem is to express the required connectivity of the routes. This is conveyed by setting variables of adjacent sections as connected. Here the $\smile$ sign declares variables as connected. This does not explicitly imply that the values assigned to the variables are equal. It merely states that the lanes of the specified variables have to be connected. This has to kept in mind for the conversion of the equations into CNF in Section [subsection][3][4,5]4.5.3. The connectivity of the routes is expressed by

$$\text{OUT1} \smile \text{HL1} \smile \text{VLR1} \smile \text{IN1}, \tag{4.1}$$
$$\text{OUT2} \smile \text{HL2} \smile \text{VLR2} \smile \text{IN2}. \tag{4.2}$$

The last subproblem is to provide the necessary constraints to prevent routes from occupying the same lanes. At the most, a route is represented by a single variable

per section. So the problem can be solved by declaring variables regarding the same sections on the setup as unequal. The two routes in this example have three sections in common, leading to the three expressions

$$\mathrm{HL1} \neq \mathrm{HL2}, \tag{4.3}$$
$$\mathrm{VLR1} \neq \mathrm{VLR2}, \tag{4.4}$$
$$\mathrm{IN1} \neq \mathrm{IN2}. \tag{4.5}$$

These five lines of expressions can be interpreted as a set of constraints to be satisfied by assigning values to the variables.

4.5.2 Substitution Of Non-Boolean Variables

Translating the problem into a set of equations in CNF requires boolean variables. Therefore, the variables have to be substituted by boolean ones. The values of the variables determine which lanes are used. In this example they can range from 0 to 3, as already mentioned in Section [subsection][1][4,5]4.5.1. These values can be expressed using two boolean variables. The eight introduced variables require sixteen boolean ones, which are defined as

$$\begin{aligned}
\mathrm{OUT1} &= \{\textcircled{01}, \textcircled{02}\} = 2 \cdot \textcircled{01} + \textcircled{02}, \\
\mathrm{OUT2} &= \{\textcircled{03}, \textcircled{04}\} = 2 \cdot \textcircled{03} + \textcircled{04}, \\
\mathrm{HL1} &= \{\textcircled{05}, \textcircled{06}\} = 2 \cdot \textcircled{05} + \textcircled{06}, \\
\mathrm{HL2} &= \{\textcircled{07}, \textcircled{08}\} = 2 \cdot \textcircled{07} + \textcircled{08} \\
\mathrm{VLR1} &= \{\textcircled{09}, \textcircled{10}\} = 2 \cdot \textcircled{09} + \textcircled{10}, \\
\mathrm{VLR2} &= \{\textcircled{11}, \textcircled{12}\} = 2 \cdot \textcircled{11} + \textcircled{12}, \\
\mathrm{IN1} &= \{\textcircled{13}, \textcircled{14}\} = 2 \cdot \textcircled{13} + \textcircled{14}, \\
\mathrm{IN2} &= \{\textcircled{15}, \textcircled{16}\} = 2 \cdot \textcircled{15} + \textcircled{16}, \\
\text{with} \quad & \textcircled{i} \in \{0, 1\}, i \in \{01, \ldots, 16\}.
\end{aligned}$$

The circled numbers represent boolean variables. This representation was chosen because it corresponds to the conventions of the CNF notation [Nam et al., 2002]. With this substitution the hardware of this test environment is described by boolean variables. This suffices to reformulate the problem with expressions in CNF in order to compute a solution with a SAT solver.

Beyond this example, the choice of variables and boolean substitutions has to be adapted for the specific problem. If possible, it is of advantage to directly use natively boolean variables, like hardware switches which can only toggle between

an *on* and *off* state. Using such variables avoids the step of artificial substitution presented in this section before and after applying a SAT solver.

4.5.3 Conversion Into Conjunctive Normal Form

With the variable substitution defined in the last section, the equations set up in Section [subsection][1][4,5]4.5.1 can be written in CNF. The equations in ([equation][1][4]4.1) and ([equation][2][4]4.2) are the ones defining the routes, like

$$\text{OUT1} \smile \text{HL1} \ .$$

Using the boolean variables, the equation is interpreted as

$$2 \cdot \textcircled{01} + \textcircled{02} = 2 \cdot \textcircled{05} + \textcircled{06} \ .$$

This equation has to be to transformed into an expression in CNF evaluating to `true` for values satisfying it. For this it suffices to ensure the equality of the variables 01 and 05 and the variables 02 and 06. This can be expressed by two clauses for each pair of variables. Clauses are connected by a logical `and`,

$$(\neg \textcircled{01} \vee \textcircled{05}) \wedge (\textcircled{01} \vee \neg \textcircled{05}) \wedge (\neg \textcircled{02} \vee \textcircled{06}) \wedge (\textcircled{02} \vee \neg \textcircled{06}) \ . \tag{4.6}$$

There is no general procedure to automatically interpret logical expression and generate CNF clauses. A first clause is generate manually and applied as a template to generate clauses for analogous problems. The correctness of this expression is checked by evaluating a truth table, this is shown in Table [table][2][4]4.2. The remaining equations connecting the sections of the routes can be transformed analogously with the exception of HL1 $\smile$ VLR1, HL2 $\smile$ VLR2. These equations represent the connections of the horizontal with the vertical lanes. For the substitution of the variables with boolean ones, the connectivity at the crossing of horizontal and vertical lanes has to be taken into account. The lanes at the crossing are connected with a sparseness of 2. That means every horizontal lane can be wired to a vertical lane if their numbers are of the same parity. This is realized by two clauses equalizing the least significant boolean variables. For example for HL1 and VLR1 this is

$$(\neg \textcircled{06} \vee \textcircled{10}) \wedge (\textcircled{06} \vee \neg \textcircled{10}) \ .$$

These clauses give an accurate description the connectivity. Adding the purposely left out clauses in analogy to the example of OUT1 and HL1 would lead to an extension of the SAT problem and constrain the possible assignments for a valid solution further. This is unnecessary and obstructive for solving the problem, since possible solutions are lost. So in general there have to be introduced as few clauses as possible and as many as necessary to accurately describe the problem.

(01)	(02)	(05)	(06)	Evaluation
0	0	0	0	`true`
0	0	0	1	`false`
0	0	1	0	`false`
0	0	1	1	`false`
0	1	0	0	`false`
0	1	0	1	`true`
0	1	1	0	`false`
0	1	1	1	`false`
1	0	0	0	`false`
1	0	0	1	`false`
1	0	1	0	`true`
1	0	1	1	`false`
1	1	0	0	`false`
1	1	0	1	`false`
1	1	1	0	`false`
1	1	1	1	`true`

Table 4.2: Truth table evaluating the expression ([equation][6][4]4.6); The expression only evaluates to `true` for four assignments. These stand for the four possibilities of route 1 connecting the output of block 1 with any of the four horizontal lanes of the test setup in Figure [figure][5][4]4.5.

The clauses expressing the required connectivity summed up:

$$(\neg\,\textcircled{01} \vee \textcircled{05}) \wedge (\textcircled{01} \vee \neg\,\textcircled{05}) \wedge (\neg\,\textcircled{02} \vee \textcircled{06}) \wedge (\textcircled{02} \vee \neg\,\textcircled{06}), \quad (\text{OUT1} \smile \text{HL1})$$

$$(\neg\,\textcircled{03} \vee \textcircled{07}) \wedge (\textcircled{03} \vee \neg\,\textcircled{07}) \wedge (\neg\,\textcircled{04} \vee \textcircled{08}) \wedge (\textcircled{04} \vee \neg\,\textcircled{08}), \quad (\text{OUT2} \smile \text{HL2})$$

$$(\neg\,\textcircled{06} \vee \textcircled{10}) \wedge (\textcircled{06} \vee \neg\,\textcircled{10}), \quad (\text{HL1} \smile \text{VLR1})$$

$$(\neg\,\textcircled{08} \vee \textcircled{12}) \wedge (\textcircled{08} \vee \neg\,\textcircled{12}), \quad (\text{HL2} \smile \text{VLR2})$$

$$(\neg\,\textcircled{09} \vee \textcircled{13}) \wedge (\textcircled{09} \vee \neg\,\textcircled{13}) \wedge (\neg\,\textcircled{10} \vee \textcircled{14}) \wedge (\textcircled{10} \vee \neg\,\textcircled{14}), \quad (\text{VLR1} \smile \text{OUT1})$$

$$(\neg\,\textcircled{11} \vee \textcircled{15}) \wedge (\textcircled{11} \vee \neg\,\textcircled{15}) \wedge (\neg\,\textcircled{12} \vee \textcircled{16}) \wedge (\textcircled{12} \vee \neg\,\textcircled{16}). \quad (\text{VLR2} \smile \text{OUT2})$$

The equations ([equation][3][4]4.3), ([equation][4][4]4.4) and ([equation][5][4]4.5) for avoiding the intersection of routes can be transformed in a similar way. Variable assignments of routes allocating the same lane in the same section must be prohibited by the corresponding clauses. For equation ([equation][3][4]4.3), HL1 $\neq$ HL2, this is done by applying the boolean substitution, resulting in

$$2 \cdot \textcircled{05} + \textcircled{06} \neq 2 \cdot \textcircled{07} + \textcircled{08}.$$

The necessary CNF clauses to express this explicitly forbid every possible assignment violating this equation. Since there are four lanes per section, there are four clauses. Each clause excludes the possibility of double occupancy for one lane so that HL1 $\neq$ HL2,

$$\begin{aligned}
&(\neg\,\textcircled{05} \vee \neg\,\textcircled{06} \vee \neg\,\textcircled{07} \vee \neg\,\textcircled{08}) \\
\wedge&(\neg\,\textcircled{05} \vee \quad \textcircled{06} \vee \neg\,\textcircled{07} \vee \quad \textcircled{08}) \\
\wedge\ &(\textcircled{05} \vee \neg\,\textcircled{06} \vee \quad \textcircled{07} \vee \neg\,\textcircled{08}) \\
\wedge\ &(\textcircled{05} \vee \quad \textcircled{06} \vee \quad \textcircled{07} \vee \quad \textcircled{08}).
\end{aligned} \tag{4.7}$$

The clauses for expressing the equations ([equation][4][4]4.4) and ([equation][5][4]4.5) have the same structure as those in ([equation][7][4]4.7). They are formed analogously to them. The logic expressed by these clauses is the negation of the one presented in Table [table][2][4]4.2. This can be seen in Table [table][3][4]4.3.

By looking at the course of the lanes in Figure [figure][5][4]4.5 it becomes apparent that the clauses regarding ([equation][5][4]4.5) can be left out. These clauses prevent double allocation of the inputs of block 2. This case is already covered by the clauses derived of equation ([equation][5][4]4.5), since one input is connected to only one vertical lane. The clauses derived from ([equation][5][4]4.5) are redundant. The remaining eight clauses can be seen in the lines 22-29 of the .cnf file in Listing [lstlisting][1][21474836470]1. They are appended to the clauses expressing the connectivity. Adding these clauses turns the 2SAT problem into a 4SAT problem, because the problem now contains clauses with up to four different variables.

(05)	(06)	(07)	(08)	Evaluation
0	0	0	0	`false`
0	0	0	1	`true`
0	0	1	0	`true`
0	0	1	1	`true`
0	1	0	0	`true`
0	1	0	1	`false`
0	1	1	0	`true`
0	1	1	1	`true`
1	0	0	0	`true`
1	0	0	1	`true`
1	0	1	0	`false`
1	0	1	1	`true`
1	1	0	0	`true`
1	1	0	1	`true`
1	1	1	0	`true`
1	1	1	1	`false`

Table 4.3: Truth table evaluating the expressions of ([equation][7][4]4.7); For each assignment leading to double allocation of lanes one of the clauses in the expression evaluates to `false`.

Limiting Values Of Variables To A Specified Range

In this example, a variable ranges over 4 values, since there are 4 lanes to choose from in each section. This range fits exactly for converting the non-boolean variable into two boolean ones, because the range of values for the two corresponding boolean variables also is 4.

Limiting ranges of variables can be necessary to apply other hardware constraints. For example if there are only a number of n synaptic inputs available, the SAT solver must only be allowed to assign values ranging from 0 to $n-1$ to a variable S representing the target synaptic input of an arbitrary route.

This can be achieved by introducing boolean variables converting n into the binary system. Assuming i is the minimal number of binary digits required to express n in the binary system, S can be expressed by boolean variables s_k with,

$$S = \sum_{k=0}^{i} 2^k s_k \quad , s_k \in \{0, 1\}.$$

The maximum value that could possibly be assigned to the variable S without adding constraining expressions is $S_{max} = 2^i - 1 \geq n - 1$, with $s_k = 1 \ , \forall \ k$.

Picking $n = 5$, results in $i = 3$, three boolean variables s_k for $S = s_0 + 2 \cdot s_1 + 4 \cdot s_2$ and $S_{max} = 7 > n - 1$. The valid set of assignments for S would be $M = \{0, 1, 2, 3, 4\}$, so to ensure $S \in M$, additional expressions have to be introduced, constraining the possible values of S. By adding the expressions for $S \neq 5, S \neq 6$ and $S \neq 7$ to the set of constraints, the range of S is limited to its valid set M. The corresponding CNF clauses,

$$\neg s_0 \vee s_1 \vee \neg s_2, \tag{4.8}$$
$$s_0 \vee \neg s_1 \vee \neg s_2, \tag{4.9}$$
$$\neg s_0 \vee \neg s_1 \vee \neg s_2, \tag{4.10}$$

are evaluated in the truth Table [table][4][4]4.4.

Decimal	s_2	s_1	s_0	Evaluation
0	0	0	0	`true`
1	0	0	1	`true`
2	0	1	0	`true`
3	0	1	1	`true`
4	1	0	0	`true`
5	1	0	1	`false`
6	1	1	0	`false`
7	1	1	1	`false`

Table 4.4: Truth table evaluating the CNF expressions in ([equation][8][4]4.8),([equation][9][4]4.9) and ([equation][10][4]4.10). Only values of the set $M = \{0, 1, 2, 3, 4\}$ evaluate to `true`, confirming the sufficiency of the constraints.

4.5.4 Routing Solution

In order to process the exemplary SAT problem with a SAT solver, a .cnf file is created, containing the CNF clauses generated in the previous section. This file is shown in Listing [lstlisting][1][21474836470]1. By applying MiniSat the solution presented in Table [table][5][4]4.5 is produced.

Variable	Value	Variable	Value
01	0	09	0
02	0	10	0
03	0	11	0
04	1	12	1
05	0	13	0
06	0	14	0
07	0	15	0
08	1	16	1

Table 4.5: The solution of the exemplary routing problem described in the previous section and by the .cnf file in Listing [lstlisting][1][21474836470]1. The assignment is obtained with MiniSat. The assigned values satisfy the CNF clauses of the input file.

The assigned values can be interpreted by reversing the boolean substitution defined in Section [subsection][2][4,5]4.5.2. It becomes apparent that one route is located on the lanes numbered with 0, the other route on the lanes numbered with 1. The routes are visualized in Figure [figure][6][4]4.6.

4.6 HICANN Routing

The concept explicated in Section [section][5][4]4.5 is now extended to the layer 1 communication environment, which leads towards a routing tool on HICANN scale as preliminary implementation of a wafer scale routing as proposed in Section [section][1][4]4.1.

Examples for translating logical constraints into CNF clauses have been shown in Section [subsection][3][4,5]4.5.3. In order to be able tp utilize SAT solvers in an automated mapping flow, the process of generating clauses has to be automated. This can be accomplished by deriving more generalized rules. These rules cover all types of constraints, hardware constraints as well as software constraints. The hardware constraints are the ones needed for the description of the HICANN routing environment specified in Section [section][3][4]4.3. The software constraints are the ones describing the connectivity of the neural network as mentioned in Section [section][4][4]4.4.

A prerequisite of formulating the constraints in CNF is the definition of variables and the assessment of logical constraints as seen in Section [section][5][4]4.5.

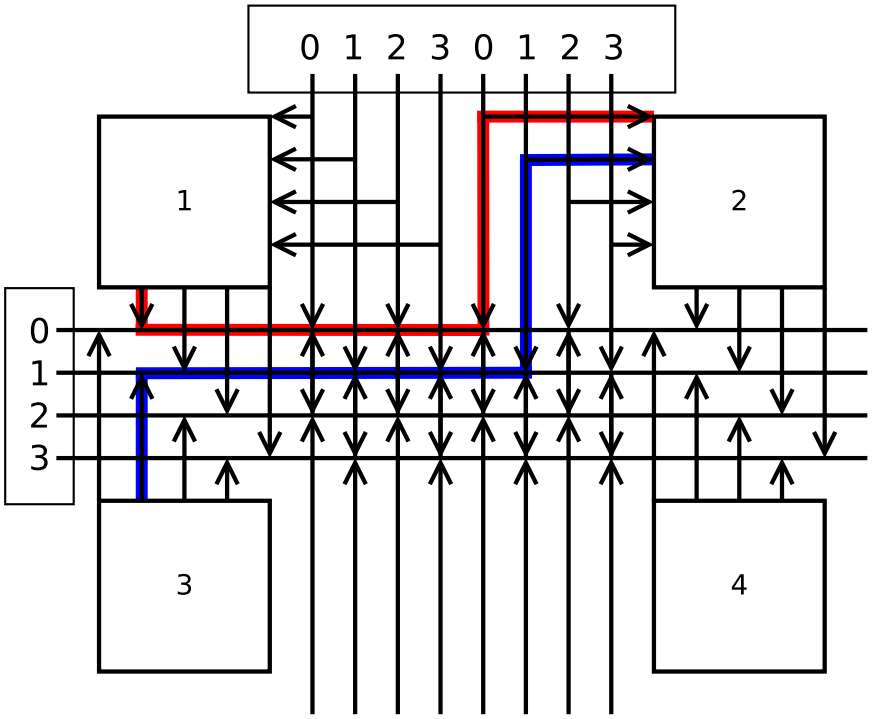

Figure 4.6: Visualization of the testing environment for routing. The task was to find a route connecting block 1 with block 2 and another route connecting block 3 with block 2. The marked routes are the ones computed by Minisat 2.2.1 from the input file in Listing [lstlisting][1][21474836470]1. The routes provide the requested connectivity and do not intersect, so this exemplary routing problem is solved.

Structure	Number of Variables
Crossbar Switch Matrix	128
Synapse Switch Matrix	1,792
Switches HICANN	7,424
Switches Wafer ($\approx 400\times$ HICANN)	2,969,600

Table 4.6: Count of necessary variables for hardware structures of different scale.

4.6.1 Definition And Assessment Of Variables For HICANN Routing

For the routing on the layer 1 communication lanes it is possible to choose a native type of variables. Since the routing is configured by setting the switches is the crossbar matrices and synapse switch matrices, the switches can be interpreted as boolean variables. Routing on a native level has the advantage of reducing the complexity of preparation and post-processing. Instead, the output of the solver can directly be used as hardware configuration for the switch matrices.

Since the pattern of switch matrices is repeating on a HICANN basis, the variables have to be assessed on this level. In one HICANN unit there are two crossbar switch matrices and four synapse switch matrices. As can be seen in Figure [figure][1][4]4.1, the matrices of one type are identical except for horizontal and vertical mirroring. This symmetry can be exploited, only two templates in the form of a matrix with corresponding entries are required to describe the structure of the layer 1 communication topology. One template for the crossbar switch matrices and one for the synapse switch matrices are duplicated and symmetrically transformed to cover the whole hardware space.

Switch Matrix Variables

In order to determine the required number of variables for the topology shown in Figure [figure][1][4]4.1, the number of switches are counted in Table [table][6][4]4.6. The number of variables approximated for one wafer is of the same magnitude as the number of variables of the more complex benchmarks the SAT solvers face in the SAT competition [Competition, 2012]. The variables are represented by entries in matrices of the dimension 128×64 for crossbar switch variables and 128×112 in the case of synapse switch variables. The format of the matrices is explained in Section [subsection][2][4,6]4.6.2.

4.6.2 Assessment Of Constraints

Since the hardware constraints only depend on the hardware used, they can already be generated at compile time, so that they can be recalled during operation for experiments without further computational efforts.

The software constraints describing the network naturally depend on the user input network, which is to be mapped. These constraints have to be generated dynamically during operation.

Sparseness Constraints

The routing limitations given by the sparseness of the switch matrices, are represented in the implemented template matrices mentioned in Section [subsection][1][4,6]4.6.1. The dimensions of a template matrix are chosen to fit an all-to-all connectivity, i.e. every entry represents the connection of the lane of its row with the lane of its column. The connectivity on the HICANN does not allow all-to-all connections, but is sparse. Switches not present on the hardware are marked by the entry -1 in the switch matrix template. Switches which are present on the hardware are marked by an integer entry in the template matrix with the number of the switch. This number also represents the variable name used for this switch. An exemplary 4×4 switch matrix template with 8 variables is written as,

$$\begin{matrix} -1 & 1 & -1 & 2 \\ -1 & 3 & -1 & 4 \\ 5 & -1 & 6 & -1 \\ 7 & -1 & 8 & -1 \end{matrix}$$

When generating constraints to connect lanes, the two switch matrix templates are used to implicitly ensure that lanes are only connected if their regarding switch exists.

The template matrices are only used to generate the hardware constraints and are not related to any kind of hardware user input. Since these constraints change only with the type of hardware used, the template matrices become redundant once the constraints are generated.

Hardware Constraints

As a guideline it is recommended only to activate one switch for the sender and one switch for the signal receiver between two repeaters on a single lane. A consequence of this are clauses representing hardware constraints, prohibiting more than one

signal per lane. This affects synapse switches as well as crossbar switches, because within a switch matrix only one switch per row and per lane can be active.

Formulating hardware constraints means to explicitly prevent illegal configuration states as shown in Section [subsection][3][4,5]4.5.3. If there is a number i of boolean variables, there are 2^i possible states. Assuming these variables represent switches in the same matrix row or column, there are only $i + 1$ legal configuration states (i different states with a single switch turned on and one with all switches turned off). The remaining configuration states are illegal and each of them has to be declared explicitly in its own CNF clause.

By formulating these constraints it appears that the number of CNF expressions for crossbar switch matrices is manageable, being lower than $1,000$ clauses per matrix. Unfortunately this can not be said about the number of CNF expression used to describe the hardware constraints of a synapse switch matrix. In a synapse switch matrix there are up to 14 switches in one row. As a result, $2^{14} - 15 = 16,369$ clauses are generated for a single row. In this case, over $7,000,000$ CNF clauses are needed for one matrix, of which four are part of a single HICANN. This can be avoided by using pseudo boolean clauses as described in the following section.

4.7 Pseudo Boolean SAT Routing

For some applications, problems formulated with pseudo-boolean clauses are shorter than plain boolean clauses in CNF. Pseudo boolean clauses are a generalized form of boolean clauses [Eén and Sörensson, 2006] and can be expressed as

$$\sum_{i=0}^{n} c_i s_i \geq c_{n+1}, \qquad \text{with} \quad c_i \in \mathbb{Z}, s_i \in \{0, 1\}.$$

For $c_i = 1$ the clause reduces to a plain boolean CNF clause. The extension of CNF clauses with linear factors does not alter the formulated SAT problem in its principle, since SAT problems with pseudo boolean clauses can be translated into an equal problem formulated with CNF clauses [Eén and Sörensson, 2006].

Since 2005 there is a additional SAT competition for SAT problems formulated with pseudo boolean clauses [Manquinho et al., 2006]. The input format for the solvers features an extended syntax, called .opb format. The SAT solvers presented in Section [section][2][4]4.2 support this format. In addition to the relational operator $\geq$, the operators $=$ and $\leq$ are valid in this format. A minimization function, which can be provided within this format along with the pseudo boolean clauses, keeps the number of active switches at a minimum, preventing unwanted routes.

The application of pseudo boolean clauses solves the problem stated the end of the previous section. The CNF clauses for the hardware constraints of a single row in an arbitrary switch matrix reduce to a single pseudo boolean clause.

An example for a pseudo-boolean clause confining the switches represented by the variables to a single active switch is

$$1\ s_1 + 1\ s_2 + 1\ s_3 \leq 1.$$

Only one of the variables can be set to `true`/1 to achieve a valid evaluation.

4.7.1 Connecting Switch Matrices

In order to connect the interfaces of switch matrices to form a HICANN structure as shown in Section [section][3][4]4.3, there are functions implemented generating the correspondent hardware constraints. The function for connecting a crossbar switch matrix with a synapse switch matrix produces clauses similar to,

$$1s_1 + 1s_2 - 1s_3 - 1s_4 - \cdots - 1s_{13} - 1s_{14} - 1s_{15} - 1s_{16} = 0.$$

The variables s_1 and s_2 belong to a crossbar switch matrix, the variables from s_3 to s_4 belong to a connected synapse switch matrix. The switches corresponding to the variables belong to the same column on the hardware, i.e. they share a vertical lane on the HICANN environment, shown in Figure [figure][1][4]4.1.

Connecting constraints are generated for switch matrices not separated by an repeater. Repeaters are represented by dedicated variables. These variables are used as input switches by a routing procedure of higher level. So for two repeaters on the same lane s_1, s_n and the row of switches between them s_2 to s_{n-1}, the following constraint is sufficient

$$\sum_{i=1}^{n} s_i \leq 2.$$

If one of the input switches is activated by adding a software constraint $s_1 = 1$ or $s_n = 1$, another switch is activated. This either leads the established route onto another switch matrix on the same HICANN or through the HICANN if the other input switch is explicitly actived by an additional software constraint. If a route is lead onto the next synapse switch matrix and targets into the neighbouring ANNCORE, a connection is automatically established. Here the addressed synapse driver can either be specified by adding the corresponding software constraint or the synapse driver is chosen by the SAT solver. It is to be noted, that the manual specification is not intended to be a feature of an applicable routing tool, but present in the current implementation.

To fully describe the hardware constraints of the layer 1 communication connectivity on a HICANN environment a single clause for every lane is required. This results in $2 \times 112 + 64 = 288$ rows and 256 columns, a total of 544 clauses per HICANN.

4.7.2 Realizing Input To Output Assignment

To implement networks software constraints are used. They are added to the loaded hardware constraints during runtime. To connect a specified input to a specified output, a method is implemented adding the required pseudo boolean expressions. Since the hardware constraints already cover the possible connectivity on the HICANN, the software constraints required as user input are limited to simple pairs of input and output statements for the activation of repeaters and synapse drivers. The software constraints preventing intersections are derived from these statements. The constraints derived until this point are not sufficient to provide an automated input to output assignment. This assignment is a complex problem, since multiplexing of synapse driver has to be considered if the required synapse drivers exceed the number of the physical implementations. This is not part of this feasibility study.

4.7.3 Visual Verification

Currently the routing tool writes the computed assignment into a file and there is no visualization implemented to graphically display the output configuration for the switch matrices. Recently a visualization for the HICANN configuration has been implemented for display in a web browser by Dr. Björn Kindler. The output of the developed routing tool could be linked to this visualization, but this feature is not realized during this work due to temporal limitations.

4.7.4 Handling Of Unsatisfiable Problems

As it is the nature of booleans, a SAT problem either is satisfiable or it is not satisfiable. A major draw back for the practical application of a SAT based routing tool is, that once a SAT solver fails to find a satisfying assignment for a problem, not even an approximated solution is available.

This problem is solved by using the MAXSAT mode of a solver. If there is no solution available satisfying all clauses, the SAT solver tries to find an assignment for the variables satisfying as many clauses of the problem as possible. By specifying weights for the clauses important connections can be preferred. Of course the weights have to be applied by the user and the MAXSAT mode comes with high computational effort compared to standard SAT solving.

4.8 Discussion and Outlook

It has been shown that SAT solvers can be utilized to approach the routing problem on neuromorphic hardware. The current implementation is present in the form of a

feasibility study and is limited to the HICANN environment, but can be extended to a wafer scale routing tool.

4.8.1 Ratio Of The Numbers Of Clauses To Variables

For the 2 and 3SAT problem it has been noted in Clote and Kranakis [2002, p. 208], that for a ratio $r = \frac{m}{n}$ a transition of probability from satisfying to unsatisfying of a random instance can be observed. The ratio is defined by the quotient of the number of clauses m and the number of variables n. It is shown that the point of transition for the 2SAT problem is exactly at $r = 1$ [Clote and Kranakis, 2002]. The point of transition for the 3SAT is approximated numerically and is stated to be at $r \approx 4.2$. At this point, the probability of a random instance of variables to be a satisfying solution to the problem described by the clauses is fifty percent. For a ratio below this point a random instance is more likely to be satisfying solution. For the performance of a SAT solver it is of interest to avoid a balanced ratio. Since the routing problem is not of the grade $k = 2$ or $k = 3$ the rate of transition is unknown, but can be assumed to be above the rate for the 2SAT problem. With the currently implemented constraints with $m < 600$ and $n > 7000$ the ratio of the implemented routing in this thesis is at $r < 0.08$. This is far from a balanced ratio and of benefit for the application of a SAT solver since it is more likely to find a satisfying assignment instantly.

4.8.2 Declarative Programming

In order to solve the routing problem as a SAT problem, constraints have to be generated and a satisfying assignment has to be found. By the utilization of SAT solvers as a back end of the routing, only the task of generating constraints is left to the routing tool. The application of algorithms is restricted to the SAT router. The programming approach used for the routing tool is said to be a declarative programming approach. The routing tool rather controls *what* will be computed, while the applied SAT solver controls the *how*. I.e. the processing of the accumulated clauses is left to the SAT solver.

The advantage of the declarative approach is its easy extendability. Additional hardware constraints as well as software constraints can be added to the list of existing constraints without requiring changes in the algorithmic approach or affecting the functionality of the routing tool. This can be useful for system testing, disabling uncalibrated hardware components or hardware development.

The advantage of utilizing SAT solvers as back end also lies in avoiding software errors in the algorithmic component of the mapping tool. The SAT solvers presented in Section [section][2][4]4.2 have been developed for several years, have proven to be stable and are maintained and developed further on.

To sum up, the current implementation of the HICANN level SAT routing provides template matrices representing the structure of crossbar switch matrices and synapse switch matrices. These are used to produce an image of the HICANN layer 1 routing environment by which the possibilities of connectivity are defined. The tool generates hardware constraints, expressing these possibilities as pseudo boolean clauses. A user is able to require routes with defined inputs and specified outputs and the tool realizes the routes without intersections if the resources are sufficient.

In order to provide a fully applicable routing tool on the basis of this feasibility study, an automated output assignment as explained in Section [subsection][2][4,7]4.7.2 has to be implemented. Another essential requirement is the handling of unsatisfiable problems as described in the previous section.

In future, the routing tool implemented within the framework of this feasibility study can be extended to a wafer scale routing tool. In order to achieve this, multiple HICANN switch matrix templates have to be combined to form a wafer scale image of the layer 1 communication. For a wafer scale routing, the template matrices have to be updated to integrate the layer 2 communication. A parallel step is to integrate the routing tool into the mapping tool framework by means of a proper `C++` interface. Such a tool can provide a solution to the routing problem on the HMF system and is adaptable to neuromorphic hardware to be developed.

Part I

Appendix

```
p cnf 16 28
-1 5 0
1 -5 0
-2 6 0
2 -6 0
-3 7 0
3 -7 0
-4 8 0
4 -8 0
-6 10 0
6 -10 0
-8 12 0
8 -12 0
-9 13 0
9 -13 0
-10 14 0
10 -14 0
-11 15 0
11 -15 0
-12 16 0
12 -16 0
6 5 8 7 0
-6 5 -8 7 0
6 -5 8 -7 0
-6 -5 -8 -7 0
9 10 11 12 0
-9 10 -11 12 0
9 -10 11 -12 0
-9 -10 -11 -12 0
```

Listing 1: A .cnf file describing the problem of and in a for SAT solvers compatible format. The first line is the *problem line* stating the sort of problem that is to be solved, in this case CNF, the number of variables $n = 16$ and clauses $m = 28$.

A Lists

A.1 List of Figures

A.2 List of Tables

B Bibliography

B. Aspvall, M.F. Plass, and R.E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 14 (4):195, 1982.

Vaughn Betz and Jonathan Rose. Fpga routing architecture: segmentation and buffering to optimize speed and density. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, FPGA '99, pages 59–68, New York, NY, USA, 1999. ACM. ISBN 1-58113-088-0. doi: 10.1145/296399.296428. URL http://doi.acm.org/10.1145/296399.296428.

Romain Brette and Wulfram Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of Neurophysiology*, 94(5):3637–3642, 2005. doi: 10.1152/jn.00686.2005. URL http://jn.physiology.org/content/94/5/3637.abstract.

D. Brüderle, M.A. Petrovici, B. Vogginger, M. Ehrlich, T. Pfeil, S. Millner, A. Grübl, K. Wendt, E. Müller, M.O. Schwartz, et al. A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems. *Biological cybernetics*, 104(4):263–296, 2011.

JC Butcher. Numerical methods for ordinary differential equations. 2003.

Santiago Ramón. Cajal. *Studien uber die Hirnrinde des Menschen*. Number Bd. 5 in Studien uber die Hirnrinde des Menschen. Johann Ambrosius Barth, 1906. URL http://books.google.de/books?id=eMmtxDaSFv8C.

P. Clote and E. Kranakis. *Boolean Functions and Computation Models*. Texts in Theoretical Computer Science. Springer, 2002. ISBN 9783540594369. URL http://books.google.de/books?id=3qdnOofDTIIC.

SAT Competition, December 2012. URL http://www.satcompetition.org/.

A. Compte, N. Brunel, P.S. Goldman-Rakic, and X.J. Wang. Synaptic mechanisms and network dynamics underlying spatial working memory in a cortical network model. *Cerebral Cortex*, 10(9):910–923, 2000.

Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM. doi: 10.1145/800157.805047. URL http://doi.acm.org/10.1145/800157.805047.

R. Cossart, D. Aronov, and R. Yuste. Attractor dynamics of network up states in the neocortex. *Nature*, 423(6937):283–288, 2003.

A. P. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger. PyNN: a common interface for neuronal network simulators. *Front. Neuroinform.*, 2(11), 2008.

DIMACS. Dimacs satisfiability suggested format.

R. Douglas, M. Mahowald, and C. Mead. Neuromorphic analogue vlsi. *Annual review of neuroscience*, 18:255–281, 1995.

Rodney Douglas. Lecture on constructive cortical computation, the european future technologies conference and exhibition, May 2011. URL http://videotorium.hu/en/recordings/details/2956,Constructive_cortical_computation.

MJ During and DD Spencer. Extracellular hippocampal glutamate and spontaneous seizure in the conscious human brain. *The lancet*, 341(8861):1607–1610, 1993.

N. Eén and N. Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(3-4):1–25, 2006.

Niklas Eén and Niklas Sörensson. Minisat satrouter. http://www.minisat.se, September 2012.

J.M. Fuster, G.E. Alexander, et al. Neuron activity related to short-term memory. *Science*, 173(997):652–654, 1971.

Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness (A Series of books in the mathematical sciences)*. W.H.Freeman & Co Ltd, 1979. ISBN 0716710447.

CC Hilgetag, GAPC Burns, MA O'Neill, JW Scannell, and MP Young. Anatomical connectivity de®nes the organization of clusters of cortical areas in the macaque monkey and the cat. *Phil Trans R Soc Lond B 355 (2000)*, pages 91–110, 2000.

M.L. Hines and N.T. Carnevale. The neuron simulation environment. *Neural computation*, 9(6):1179–1209, 1997.

A.L. Hodgkin and A.F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.

G. Karypis and V. Kumar. The metis serial graph partitioning tool, 1998.

Samuel Kaski and Teuvo Kohonen. Winner-take-all networks for physiological models of competitive learning. *Neural Networks*, 7(6–7):973 – 984, 1994. ISSN 0893-6080. doi: 10.1016/S0893-6080(05)80154-6. URL http://www.sciencedirect.com/science/article/pii/S0893608005801546. <ce:title>Models of Neurodynamics and Behavior</ce:title>.

A.M. Katz and P.B. Katz. Disease of the heart in the works of hippocrates. *British heart journal*, 24(3):257–264, 1962.

D. Le Berre and A. Parrain. The sat4j library, release 2.2 system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

D. Le Berre and A. Parrain. Sat4j website. http://www.sat4j.org, October 2012.

Weichen Liu, Mingxuan Yuan, Xiuqiang He, Zonghua Gu, and Xue Liu. Efficient sat-based mapping and scheduling of homogeneous synchronous dataflow graphs for throughput optimization. In *Real-Time Systems Symposium, 2008*, pages 492 –504, 30 2008-dec. 3 2008. doi: 10.1109/RTSS.2008.49.

N.K. Logothetis, C. Kayser, and A. Oeltermann. In vivo measurement of cortical impedance spectrum in monkeys: implications for signal propagation. *Neuron*, 55 (5):809–823, 2007.

M. Lundqvist, M. Rehn, M. Djurfeldt, and A. Lansner. Attractor dynamics in a modular network of neocortex. *Network:Computation in Neural Systems*, 17:3: 253–276, 2006.

Mikael Lundqvist, Albert Compte, and Anders Lansner. Bistable, irregular firing and population oscillations in a modular attractor memory network. *PLoS Comput Biol*, 6(6), 06 2010.

V.M. Manquinho, O. Roussel, et al. The first evaluation of pseudo-boolean solvers (pb'05). *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4): 103–143, 2006.

S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 117 –128, 2002. doi: 10.1109/ICDE.2002.994702.

Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, DAC '01, pages 530–535, New York, NY, USA, 2001. ACM. ISBN 1-58113-297-2. doi: 10.1145/378239.379017. URL http://doi.acm.org/10.1145/378239.379017.

Gi-Joon Nam, K.A. Sakallah, and R.A. Rutenbar. A new fpga detailed routing approach via search-based boolean satisfiability. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(6):674 –684, jun 2002. ISSN 0278-0070. doi: 10.1109/TCAD.2002.1004311.

T. Pfeil, A. Grübl, S. Jeltsch, E. Müller, P. Müller, M. Petrovici, M. Schmuker, D. Brüderle, J. Schemmel, and K. Meier. Six networks on a universal neuromorphic computing substrate. *arXiv preprint arXiv:1210.7083*, 2012.

W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge University Press, 2007.

Sean Safarpour, Andreas Veneris, Gregg Baeckler, and Richard Yuan. Efficient sat-based boolean matching for fpga technology mapping. In *Proceedings of the 43rd annual Design Automation Conference*, DAC '06, pages 466–471, New York, NY, USA, 2006. ACM. ISBN 1-59593-381-6. doi: 10.1145/1146909.1147034. URL http://doi.acm.org/10.1145/1146909.1147034.

J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1947–1950, 2010.

Johannes Schemmel, Andreas Grübl, and Sebastian Millner. Specification of the HICANN microchip. FACETS project internal documentation, 2012.

Romain Touboul, Jonathanand Brette. Dynamics and bifurcations of the adaptive exponential integrate-and-fire model. *Biological Cybernetics*, 99(4):319–334, Nov 2008. doi: 10.1007/s00422-008-0267-4. URL http://dx.doi.org/10.1007/s00422-008-0267-4.

Yu Wang, Yong He, Yi Shan, Tianji Wu, Di Wu, and Huazhong Yang. Hardware computing for brain network analysis. In *Quality Electronic Design (ASQED), 2010 2nd Asia Symposium on*, pages 219 –222, aug. 2010. doi: 10.1109/ASQED.2010.5548242.

Duncan J Watts and Steven H Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393:440 – 442, 1998.

R. Glenn Wood and Rob A. Rutenbar. Fpga routing and routability estimation via boolean satisfiability. In *Proceedings of the 1997 ACM fifth international symposium on Field-programmable gate arrays*, FPGA '97, pages 119–125, New York, NY, USA, 1997. ACM. ISBN 0-89791-801-0. doi: 10.1145/258305.258322.

Inc. Xilinx. Spartan 6 product overview. http://www.xilinx.com/products/silicon-devices/fpga/spartan-6/index.htm, June 2012.

Erklärung:

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den (Datum) .....